Tartu Narva College

DAEMONS

COMPUTER HARDWARE

Author: Sergei Ivanov

Prof.: Chen-Wan Lin

Major: IT development of systems

13.10.2023

# Preface

This research is intended to gain insight into daemons and windows services; more specifically, into how these concepts helped people to develop their applications more effectively.

The source is found at <https://github.com/derweisskrag/ComputerHardwareColleg>. I will not demonstrate the entire code, but only specific parts of it to discuss in the paper. Hence, we cover only fundamental details of the code, leaving details on GitHub. Let us hunt some daemons!

# Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datetime | Organization/Person | Changes |
| 1.0 | 13.10.2023 | Sergei Ivanov | The initialization of the document:   1. Creating titlepage, 2. Preface, 3. Revision history 4. Overview |
| 2.0 | 15.10.2023 | Sergei Ivanov | Creating daemon chapter. |
| 3.0 | 26.10.2023 | Sergei Ivanov | Creating NEXT app, Python Server |
| 4.0 | 02.11.2023 | Sergei Ivanov | Redis, PM2, Watchdog, Windows service manager |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table 1.: Revision history

Table of Contents

[Preface 2](#_Toc149805531)

[Revision history 3](#_Toc149805532)

[Overview 6](#_Toc149805533)

[Objectives 6](#_Toc149805534)

[Goals and methods 6](#_Toc149805535)

[Daemon 7](#_Toc149805536)

[Introduction 7](#_Toc149805537)

[Command Line Interface 8](#_Toc149805538)

[Batch 8](#_Toc149805539)

[Server 14](#_Toc149805540)

[Purpose 14](#_Toc149805541)

[Implementation 14](#_Toc149805542)

[Node.js webserver 14](#_Toc149805543)

[Nodemon 17](#_Toc149805544)

[Creating NEXT app 19](#_Toc149805545)

[Introduction 19](#_Toc149805546)

[What projects is NEXT used for? 20](#_Toc149805547)

[What technologies does NEXT use? 20](#_Toc149805548)

[How is the NEXT app created? 20](#_Toc149805549)

[Project initialization 22](#_Toc149805550)

[The creation of the NEXT app 22](#_Toc149805551)

[What is next? 28](#_Toc149805552)

[Is NEXT daemon? 28](#_Toc149805553)

[Creating Flask app 29](#_Toc149805554)

[Introduction 29](#_Toc149805555)

[Creating our first app 29](#_Toc149805556)

[What is Flask’s webserver? 29](#_Toc149805557)

[Python’s http 30](#_Toc149805558)

[Python’s socket 30](#_Toc149805559)

[Watchdog 30](#_Toc149805560)

[Python’s daemonize 33](#_Toc149805561)

[Redis 33](#_Toc149805562)

[PM2 34](#_Toc149805563)

[Introduction 34](#_Toc149805564)

[How to run? 34](#_Toc149805565)

[Apache 34](#_Toc149805566)

[NGINX 35](#_Toc149805567)

[Introduction 35](#_Toc149805568)

[Event loop 35](#_Toc149805569)

[Conlusion: What is daemon? 36](#_Toc149805570)

[Daemons as firmware 37](#_Toc149805571)

[Introduction 37](#_Toc149805572)

[Examples 37](#_Toc149805573)

[GPU-Daemon 37](#_Toc149805574)

[Synthesis 39](#_Toc149805575)

[Conclusion 39](#_Toc149805576)

[Daemon AI 39](#_Toc149805577)

[Acknowledgements 40](#_Toc149805578)

[References 41](#_Toc149805579)

# Overview

## Objectives

This research paper aims to unravel the application of daemons in modern computers, as well as account for their use in computer hardware, such as services that are found in SSD.

## Goals and methods

The goal of this research is to discover how daemons are created and function in terms of computer hardware:

1. The definition of daemons and firmware.
2. The implementation of daemons in both Linux and Windows.
3. Examples of daemons.

To achieve these, I use the following methods:

1. Searching for academic papers and study them.
2. Learn from practical experience of people who created libraries:
   1. Watching videos,
   2. Reading source code,
   3. Testing.
3. Linux Test drive.

# Daemon

## Introduction

The daemon is referred to the irregular process which runs in the background of original process and does not interact with users. It also can run as standalone process, but in the background, so enabling client-server communication; as such, it handles incoming requests and sends response to multiple users. The example of such process is Apache webserver, which is a daemon process under the hood. Why do we need daemons?

Now we cannot imagine the computer machines working without daemon or services. This is because the introduction of subprocesses significantly optimizes the performance of operating system: it handles subtasks efficiently, letting the OS handle its specific tasks separately. On Windows machines, such subprocesses are called Windows services and managed by Windows service manager, while on Linux, these are called daemons and controlled by SYSTEMD. The letter ‘d’ indicated ‘daemon’, so SYSTEMD is the major system daemon which controls its units, other services or daemons. By control, we mean that it has permission to disable, enable, start, stop and restart Linux services. For example, a user can enable NGINX daemon which is responsible for handling webserver tasks such as listening to a port, like 3000 at the address localhost or 127.0.0.0.1 and responding to incoming requests; to work with NEXT application on Linux machine. Thus, daemons simplify computer workload, managing system resources efficiently and introduce more modularity because they handle specific duties on their own in the background, allowing main process to focus on more important tasks. What is a webserver?

This question is crucial, because as we noticed, daemons are not webservers, but webservers can be special type of daemons like Apache, or regular processed that we can interact with by means of terminal or CMD on Windows machine. Another important aspect is that webservers is a regular process that processes incoming requests, signals, and respond to them based on some protocol. Examples include HTTP, FTP, SMTP, UDP, POP and other protocols, and webservers interact with uses using client-server communication based on a protocol such as HTTP. What is a protocol?

A protocol is a set of rules and standards for defining a language which is used when employing devices for communication. To put it simply, Node.js webserver uses HTTP protocol to communicate with users, while unable to send requests to users or processing responses from them. This protocol ensures that Node.js sends data to a user in the correct and readable format such as HTML or JSON, where HTML is a Hyper Text Markup language for building webpages’ structure and JSON is Java Script Object Notation. Node.js is a back-end library for JavaScript to build full-stack web applications.

## Command Line Interface

### Batch

Let us try to understand server and write for that a simple Python script. For demonstration, let us go to desktop and create a director with our project, and the necessary files there.

To do so, let us open terminal. On Windows machine, ‘Win key’ + R and type ‘cmd’, then hit enter. This opens up a terminal window for us to user; On Linux machine, press CTRL + Alt + T on keyboard. Before we start creating the necessary files, let us talk about the terminal.

The first aspect is that this terminal use difference syntax for each OS. For example, on Windows machine, such syntax is Batch, while on Linux, it is Bash (the color is white by default on Windows, I set it to green using `color a`):

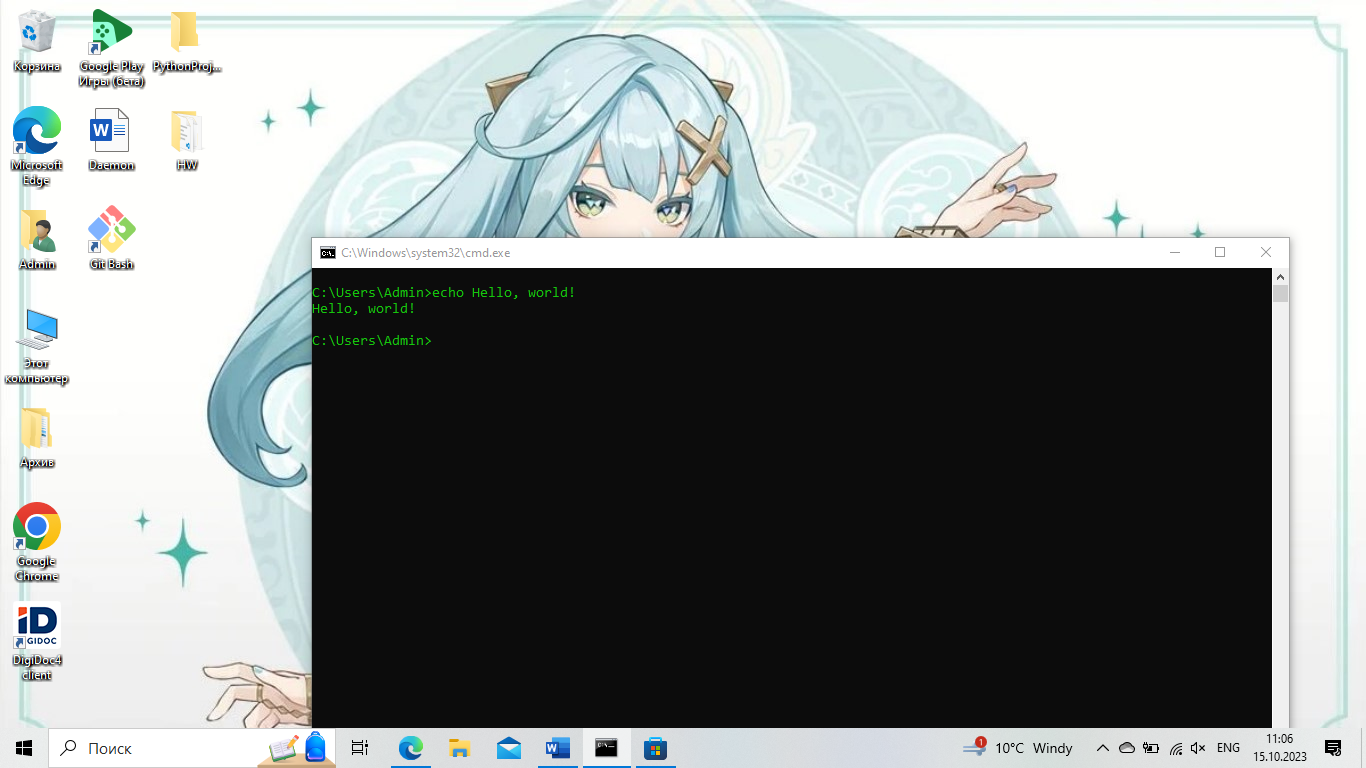


Fig. 1.: Print ‘Hello, world!’ to the terminal.

The command ‘echo’ prints the message to the console. Apart from this printing, we can also set variables, format strings and perform mathematical operations. For example, we can define a string ‘Windows’ and print the following string: ‘My machine’s OS is “Windows”!’. This is done by the commands echo and set in batch scripting: set name=”Windows”. Please, make sure leave no spaces in the command, otherwise it would not be recognized. The next step is to make use of `echo` command for printing, but `echo name` will not work: `name`. Indeed, as a result of the script, our terminal prints ‘name’. To print actual value stored in ‘name’ variable, we use `%variable\_name% as follows: `echo %name%`.

The perform mathematical operations, we have to use ‘/a’ key: `set /a 5+3` prints 8. We can also store the result of our variables:

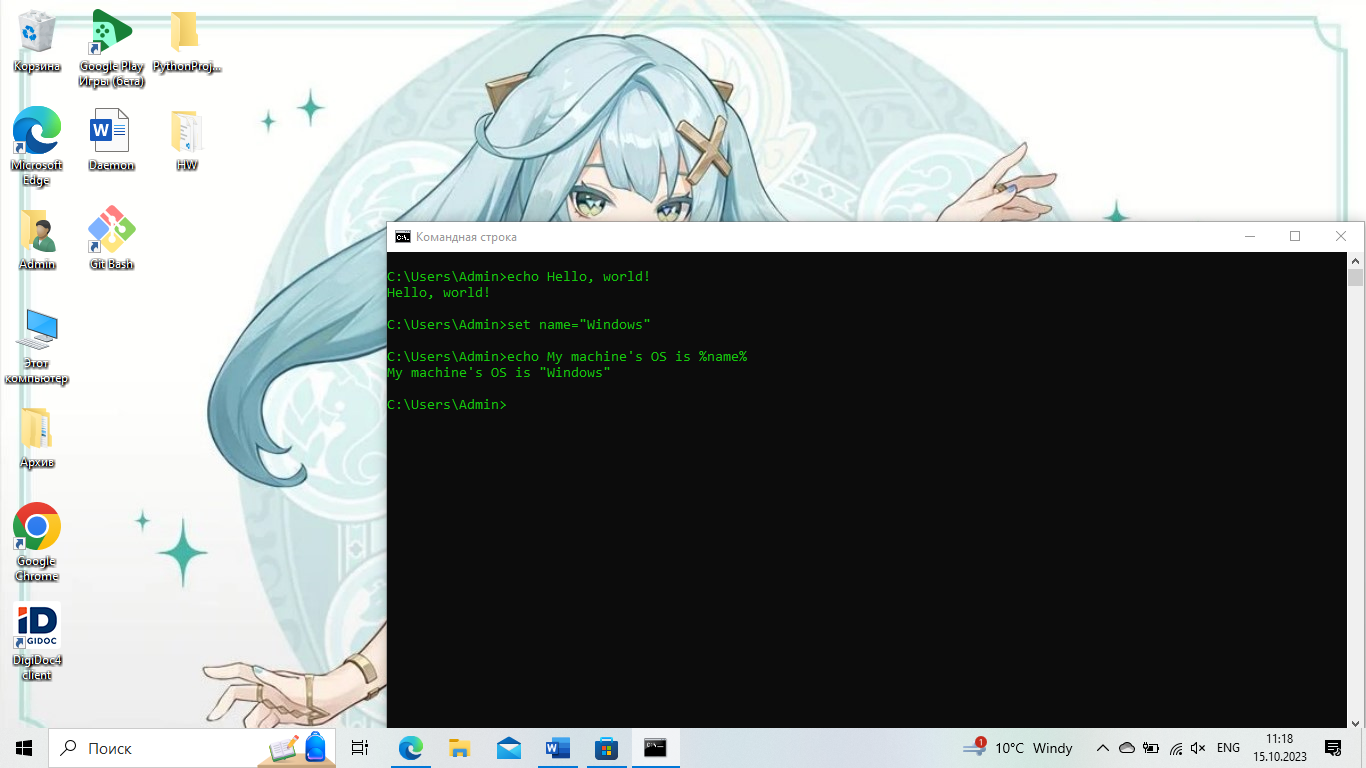


Fig. 2.: Setting variable in Batch.

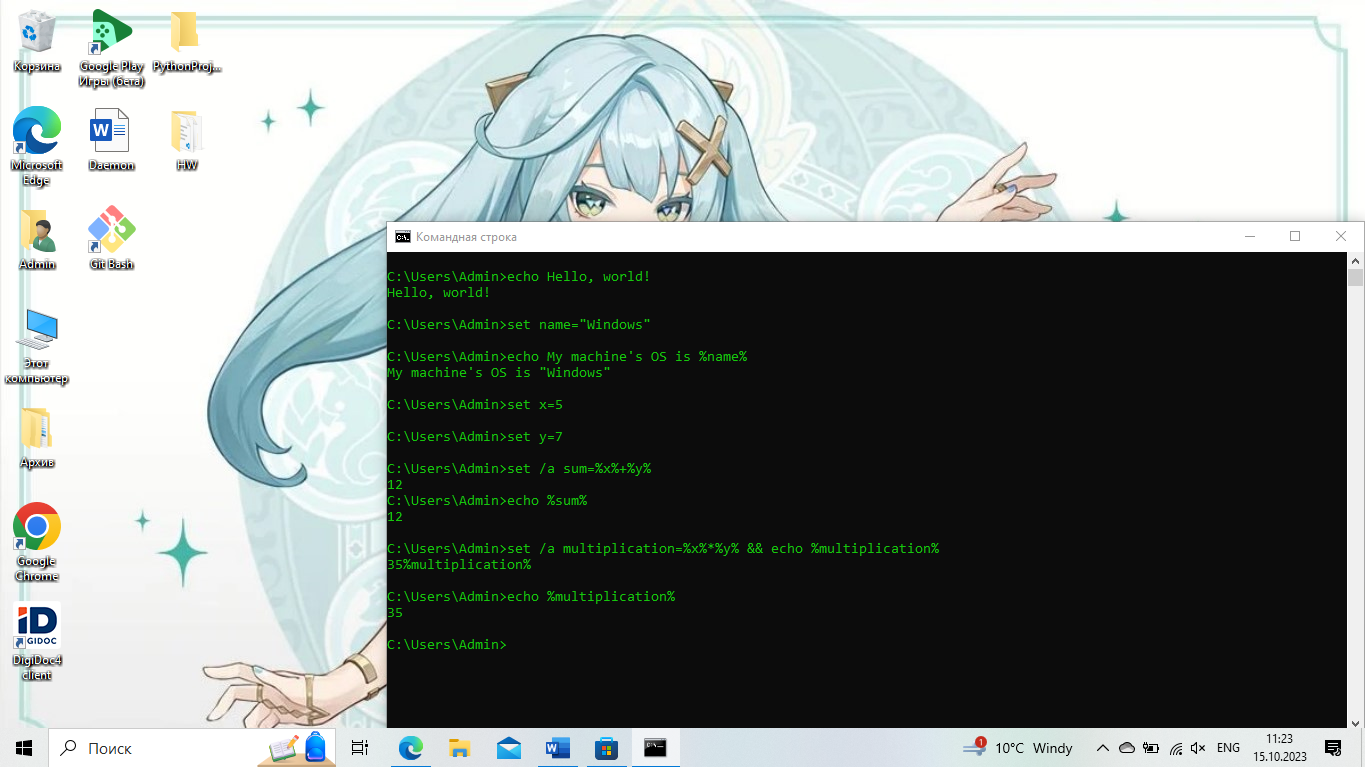


Fig. 3.: How to perform arithmetic actions.

In Batch, we can also write loops and create functions:

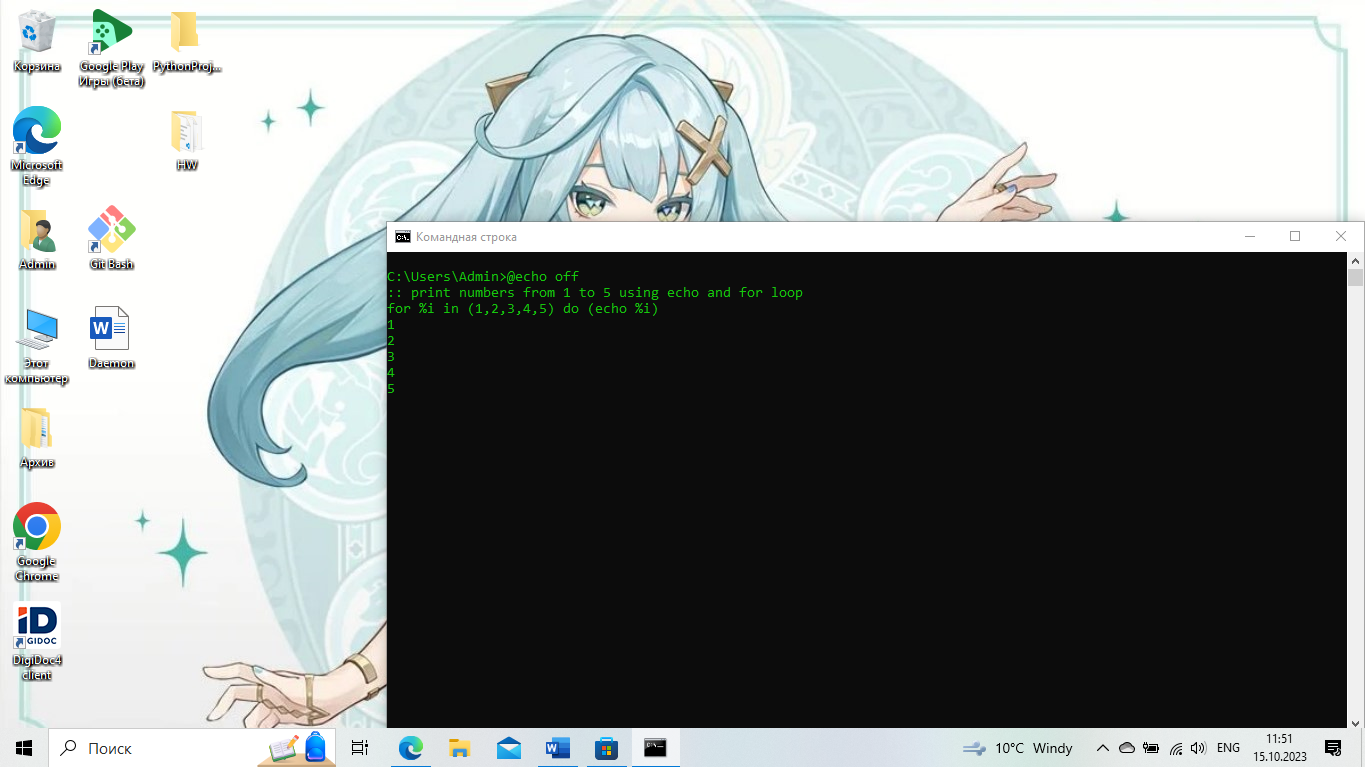


Fig. 4.: How to print numbers using for loop.

To create a function in Batch, we have to define a subroutine using ‘:’. The function body reminds us of the Assembly macro syntax: “%~1” is the first argument, “%~2” is the second and so on. To move between the code, we use ‘GOTO’ command. For example, we define the main routine called ‘MAIN’ as follows: “:MAIN” and at the beginning of script, we specify ‘GOTO :MAIN’; which besides, we can declare some variables using ‘SET’ command.

When entering the subroutine body, we specify its scope by writing ‘SETLOCAL’ and ‘ENDLOCAL’ at the end of it, and then ‘GOTO :EOF’ or ‘EXIT /b’ where the key ‘/b’ means not to exit out of the program, but out of the subroutine code and go to the main routine ‘MAIN’. Next, we define our incoming parameters: ‘SET x=%~1’ and ‘SET y=%~2’, and then create a variable ‘RESULT’ to store our arithmetic calculation, and then we return as in ‘ENDLOCAL’ using ‘&’ to use it in our main routine.

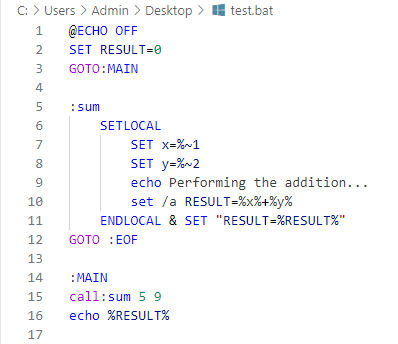


Fig. 5.: How to write a function in Batch.

Let us combine our knowledge and write a function to print out characters. The first thing is to declare our subroutine: “print” using “:” as in “:print” and then we move to its lexical scope: “SETLOCAL” and “ENDLOCAL”. This time, we want to clarify “ENABLEDELAYEDEXPANSION” which allows us to use “!char!” notation, otherwise it would print the address and not the value; the other effect would be the infinite loop.

Inside the ‘:print” subroutine, we create two variables:

1. Position called “pos”,
2. Our incoming string called name: “name”

Then, we define another subroutine to print the current char at the given position, and then increment the position and compare the character with empty one: “” to denote the termination condition. Thus, “if not “!name:~%char%,1!”==”” ( GOTO:loop )”. To increment our position and get the character:

1. Char: “!name:~%char%,1!
2. Increment: “SET /a pos=%pos%+1

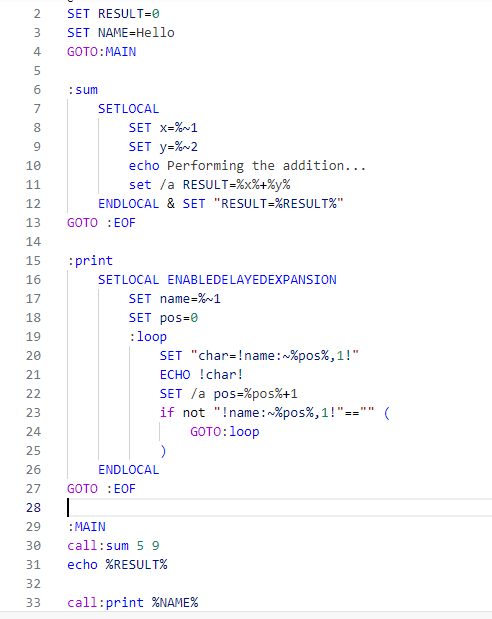


Fig. 6.: Looping over a string.

As the reader might notice, I have not explained how to create files from CMD and move to directories. Therefore, let us do so:

1. “cd” is the command for moving to directories: “cd path”.
2. “echo ‘’ > file\_name.extension” is the command to create a file with the given extension;
3. “mkdir” or “md” is used to create a folder. You can use Batch syntax to create multiple folder: `for /L %i in (1, 1, 10) do ( mkdir folder-%i )`, where the first number is start, step and end indexes. The key L enables us to use range object.

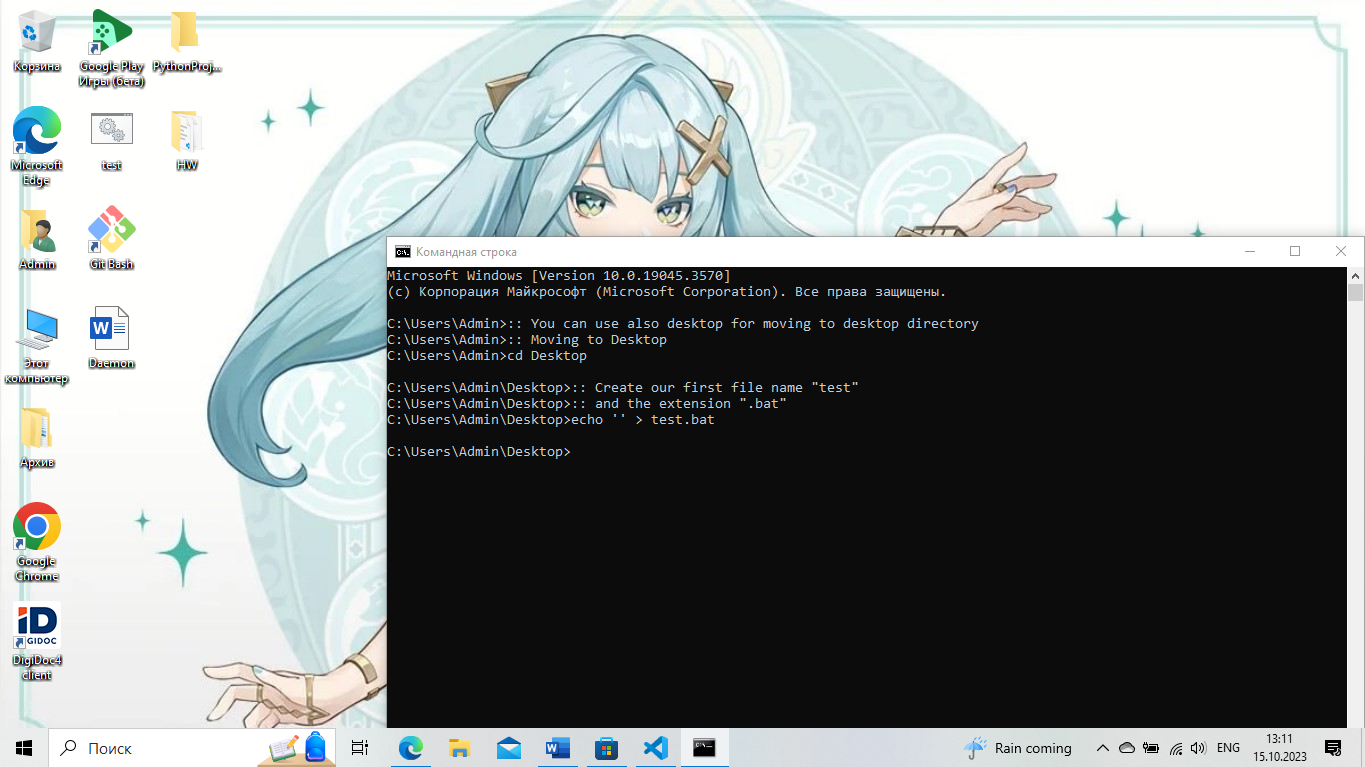


Fig. 7.: How to create a file.

Now let us write the code:

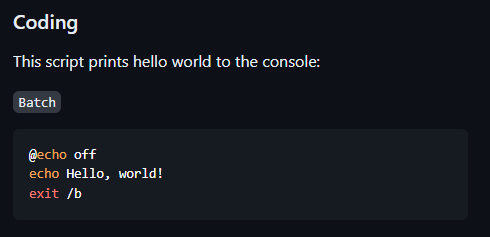


Fig. 8.: The code to write in the created file.

How to run the file? It is interesting to notice that our file is already executable. Thus, we can simply call it from the terminal: ‘./test.bat’:

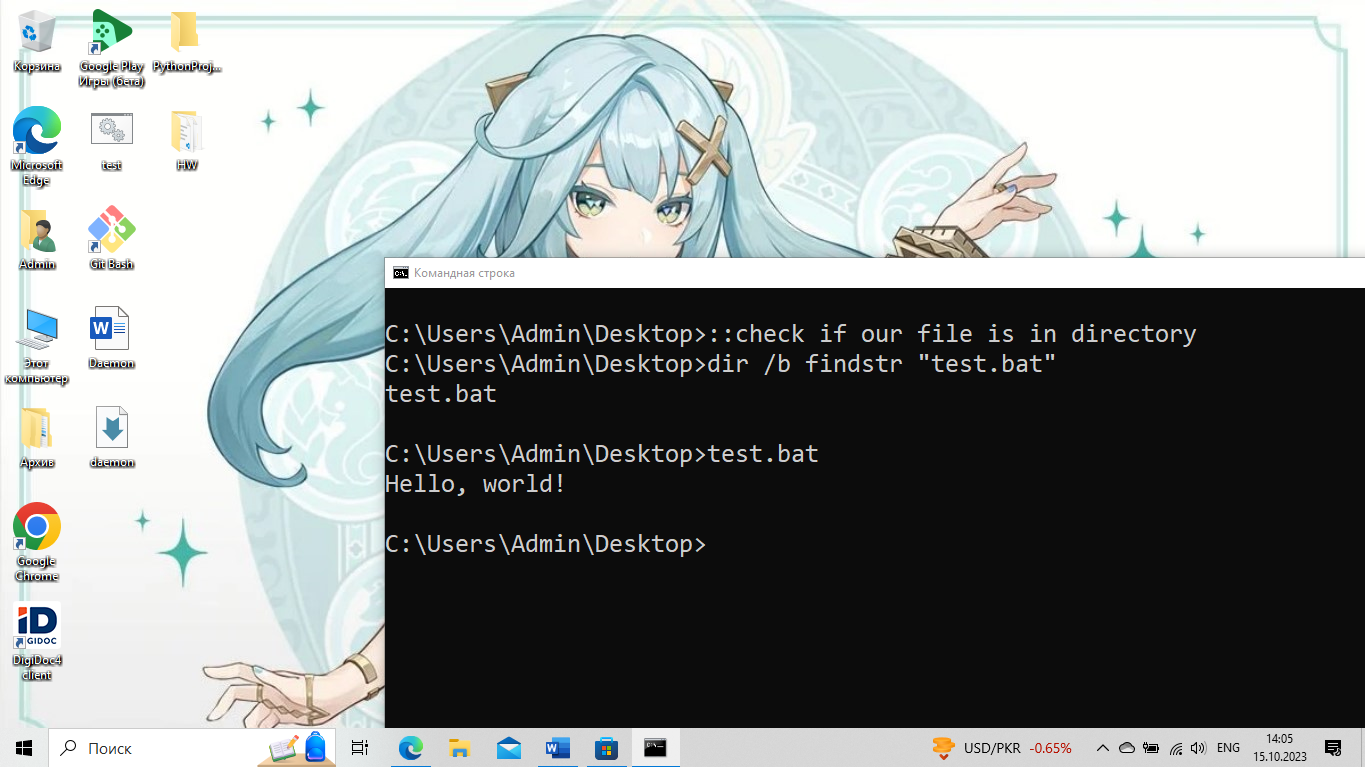


Fig. 9.: How to execute our script.

## Server

### Purpose

In order to simulate our own server, we must understand what tasks it has to fulfill. Without clear understanding, we cannot proceed, because we have to clarify what tasks it does.

### Implementation

The server is ether regular or daemon process which listens to a port and responds to incoming requests based on protocol such as HTTP. In Python, we can implement server-like application using `socket` or `multiprocessing.BaseManager` libraries.

## Node.js webserver

The JavaScript web framework NEXT uses underlying Node.js technology to run a webserver at the address “localhost” and at port “3000”. In Node.js, we can create a server using `http` or `express` module:



Fig. 10.: Creating webserver using Node.js.

To run, we can simply use `node <filename>.js` as shown in the following picture:

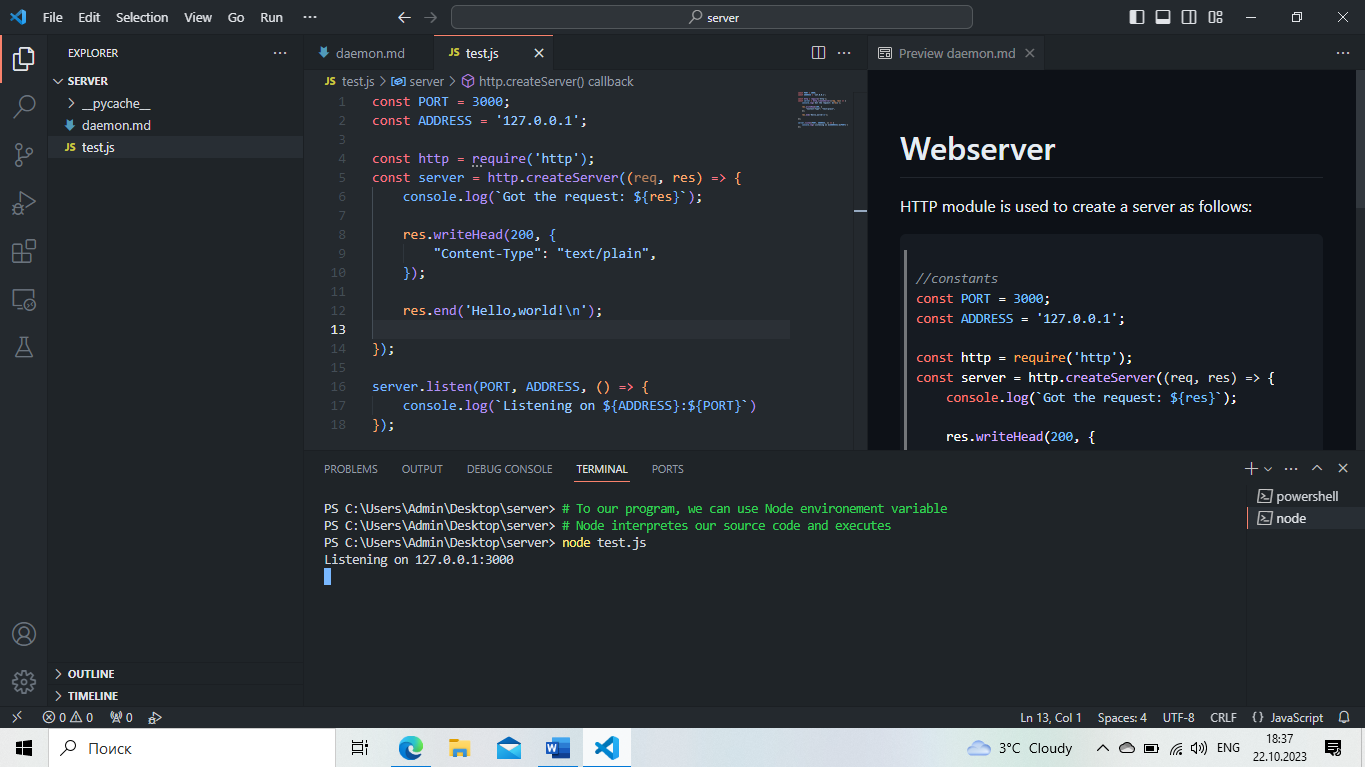


Fig. 11.: Running our program.

After Node built and ran our program, we can go to the address and port in the browser:

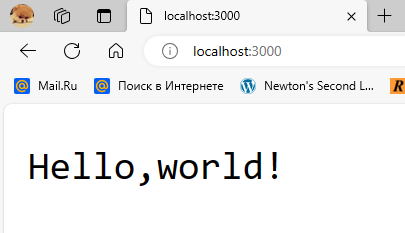


Fig. 12.: Output of our program.



I would like to point out that we could run the server using Windows or Linux terminal, not just employing Visual Studio terminal.

The second method is to use Node’s library ‘express’:



Fig. 13.: Creating our app using Express.js.



Middleware is a special module in our app which is processed before server gets the request. It means that the middleware can be used to validate the request and then send it to the server. In order to avoid infinite loop of middleware calls, we specified `next` as the third parameter of our callback function.

## Nodemon

When working with Node.js, a programmer can notice that changing the source code of a file leads to restarting the server. However, it can be avoided by installing nodemon. Like any package, it can be installed using ‘NPM’ package manager.



Fig. 14.: Installing nodemon.

Once installed, the file with the name ‘package.json’ maybe edited to add the following script:

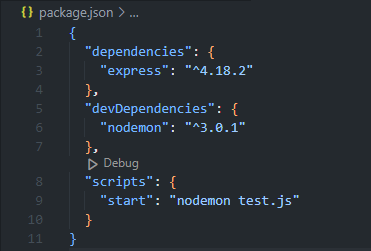


Fig. 15.: Adding scripts to ‘package.json’.

To run our app, we can use this script: `npm start`:

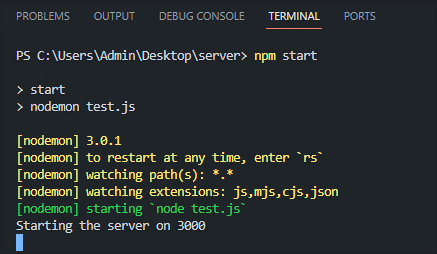


Fig. 16.: Running a server using ‘nodemon’.

The reader might ask the question as to what is the difference. The answer is that now we can mutate the source file and save the file to trigger nodemon: it will restart the server upon changes:

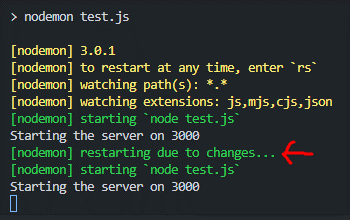


Fig. 17.: Nodemon restarts the Node.js webserver.

This is nothing but daemon process running in the background. Please, notice that daemon does not interact with us directly. Instead, it is running in the background and we cannot normally observe it while interacting with the web application.

## Creating NEXT app

### Introduction

In this section, I discuss what NEXT.js (henceforth, NEXT) is and how developers can use it to build modern scalable and fast applications. The framework (library, not any daemon or server, it is a library for JS to run JS web applications), built on top of React, is considered a professional way of building modern web applications. This is because NEXT can interact with many other technologies such as Python’s libraries and Rust’s crates. Specifically, NEXT can send requests to a Python server and re-render the content based on the server’s response. While it is not accurate to say that NEXT is used to build neural networks, NEXT can be used to present them in a dynamic modern, scalable and performant web application by interacting with a Python server that implement some neural network.

As such, NEXT fulfils the critical aspect of computer science – it allows developers to create a webpage whereby they can present some product to sell, promoting it. NEXT can take on many duties related to web technology not only to facilitate developer’s work, but also assisting them in focusing on other areas of their project which may be marketing or building architecture for new service, or working on API calls for neural network. The latter means that even though NEXT can be used to create a paired project such as the combination of NEXT and Django, developers strive for simplicity and efficiency – the balance must be maintained. That is why Python server’s services can be used on the server side of NEXT through API calls.

### What projects is NEXT used for?

There are many ideas of web applications that can be built by NEXT:

1. Blogs – NEXT can assist in building web applications for personal blogs implemented statically, SSG, or through API call, SSR. However, SSR does not mean that the data comes from API, but means that it is rendered dynamically, on the client-side, or on the server-side if data is sent from Prisma server, database. For this purpose, NEXT introduce server-side hooks: “use server” whenever the developer calls the database functions such as “await prisma.ctx.users.findMany({where: {id: 100}})”
2. Weather app: this is a classic example of how NEXT can be used for building simple API-based web applications. The data is rendered on the client-side, using “useEffect” React hook in the client component.

### What technologies does NEXT use?

When building full-stack NEXT applications, developers often implement specific functionality of authentication or data transmission. It is done by following Cryptography guidelines to ensure the delivery of data and privacy, meaning it is not intercepted by hackers.

To help developers implement some functionality that requires cryptography, NEXT uses Node.js technology. When discussing webserver terminology, we usually refer to Node.js webserver used in NEXT to run server-side web applications. The implementation of Node.js lies deep within many libraries, modules and technology written in C++. The Node.js engine, V8, is C++ application. Therefore, to understand webservers at the low level, developers must understand low-level details of Node.js written in C++, because developers built many things on top of previous technologies, and that is why I would like to use “socket” or “http” – they are built-in modules, and we take explore them.

In conclusion, NEXT utilizes modern technologies to assist in creating web applications. So as to deepen the understanding of underlying NEXT web principles and technologies, it is recommended to read the documentation about NEXT, React, Node.js and how it was done.

### How is the NEXT app created?

Let us create our first NEXT application. This process involves installing Node.js V8 engine on our machine. We can follow official instructions on how to do so. My machine is Windows 10, so

I had to download Node.js as executable program and run it to install. After installation, necessary environment variables have been set, which means that I can open up my terminal and enter the command: `npm`. This stands for ‘node package manager’ and is similar to ‘pip’ which stands for ‘python package index’; yet, these package managers differ in the way they manage packages: ‘npm’ can install packages both locally and globally, while ‘pip’ installs globally by default unless virtual environment is established; ‘pip’ uses ‘python package index’ for ecosystem and registry, whereas ‘npm’ uses ‘npm registry’.

Thus, to create NEXT application, we have to ensure that Node.js is installed and we can run ‘npm’ commands in terminal. For this purpose, we install the executable installation program from official source and execute it. On Linux, we can run `sudo apt-get install -y nodejs`, where

1. `sudo`: superuser do. It allows to run the command as a super user or root of the machine. It is because Linux has the permission control system. For certain users, there are levels of permission that clarify for what commands they are eligible to run. However, we are allowed to create users using `sudo useradd <username>` and then change the `sudoers` configuration that sets permissions levels for users. `sudo` can run all commands like an admin on Windows.
2. `apt`: package manager in Linux – Advanced Package Tool.
3. `apt-get`: this is one of the command-line tools provided by APT.
4. `-y`: this is an option that can be used with the command above to confirm affirmation: ‘yes’.
5. `nodejs`: this is the name of the package.

Once we can install packages, for example, `npm install express` installs the package `express` locally, not globally to the given directory where the command is being executed. For example, we can open up terminal, and use `cd` command to navigate between directories:

1. `cd desktop` - move to the directory of `desktop`,
2. `cd ../` - to move back to the previous directory.

We can run Batch commands for CRUD of our file system (create files, remove, update, and read).

To create NEXT application, we have to install packages locally:

1. `next@latest`,
2. `react@latest`,
3. `react-dom@latest`,
4. `typescript@latest`
5. `@types/react`,
6. `@types/node`.

This is basic set up. We can simply run the command

`npm i next@latest react@latest react-dom@latest

typescript@latest @types/node @types/react`

So, the syntax is `npm i <list of packages>`.

### Project initialization

However, because we use `npm`, we have to also set up `nodejs` project. It is done by running the command:

`npm init -y`

This way, the project is being initialized by `npm` in the `package-lock.json` and `package.json`. Hence, we can list all the packages and their corresponding versions in the `package.json` file in the dependencies. However, we can also install dev-dependencies by running `--save-dev` when `npm i`: `npm i nodemon –save-dev`. When deploying, developers do not want to leave any development.

When working with other programming languages, we also create projects in directories and for this purpose, we employ project managers such as poetry for Python or cargo for Rust. Thus, when creating NEXT application, we must also set up our project in terms of general information about creator, purpose, license and packages.

### The creation of the NEXT app

All we have to do is to create our project directory, initialize project and install necessary packages:

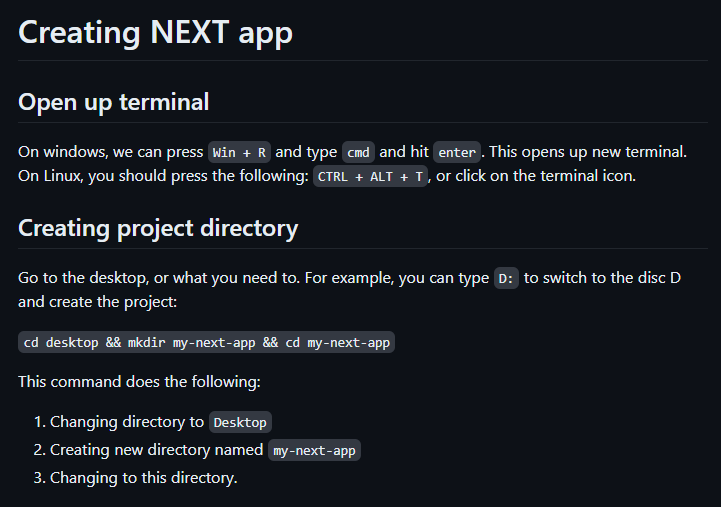


Fig. 18.: Creating our project directory

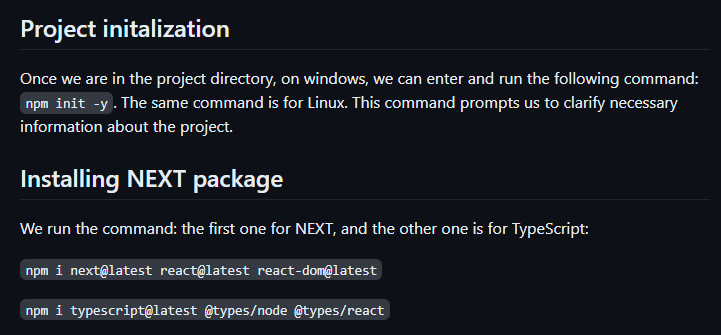


Fig. 19.: Project initialization and

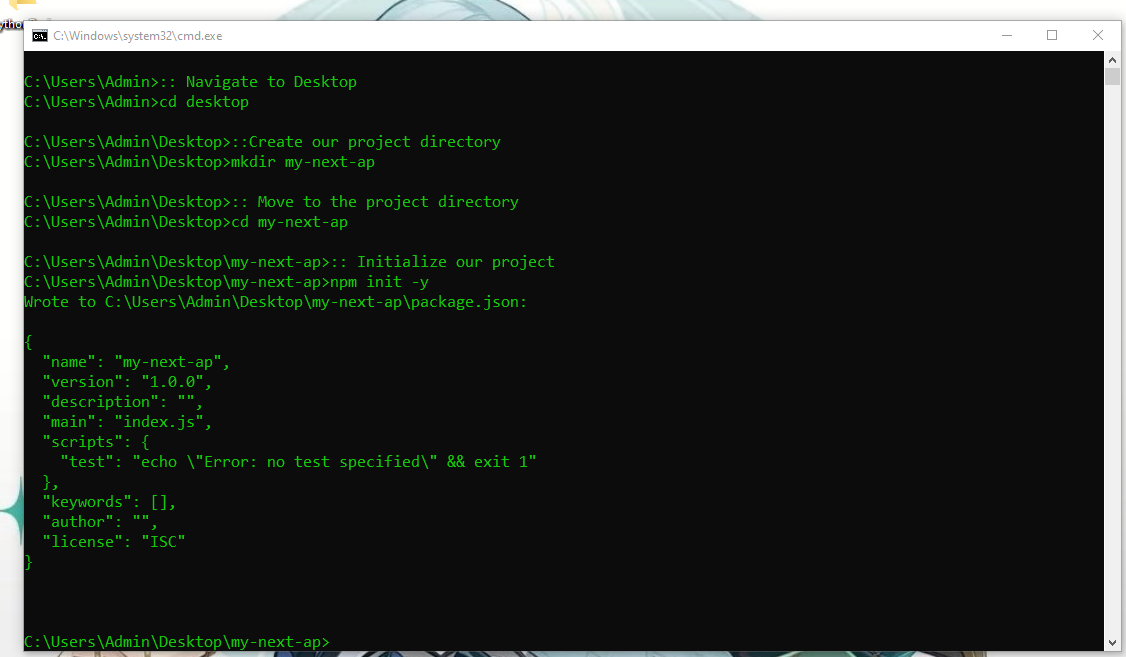


Fig. 20.: Moving to `Desktop`, creating project directory, and initialize it

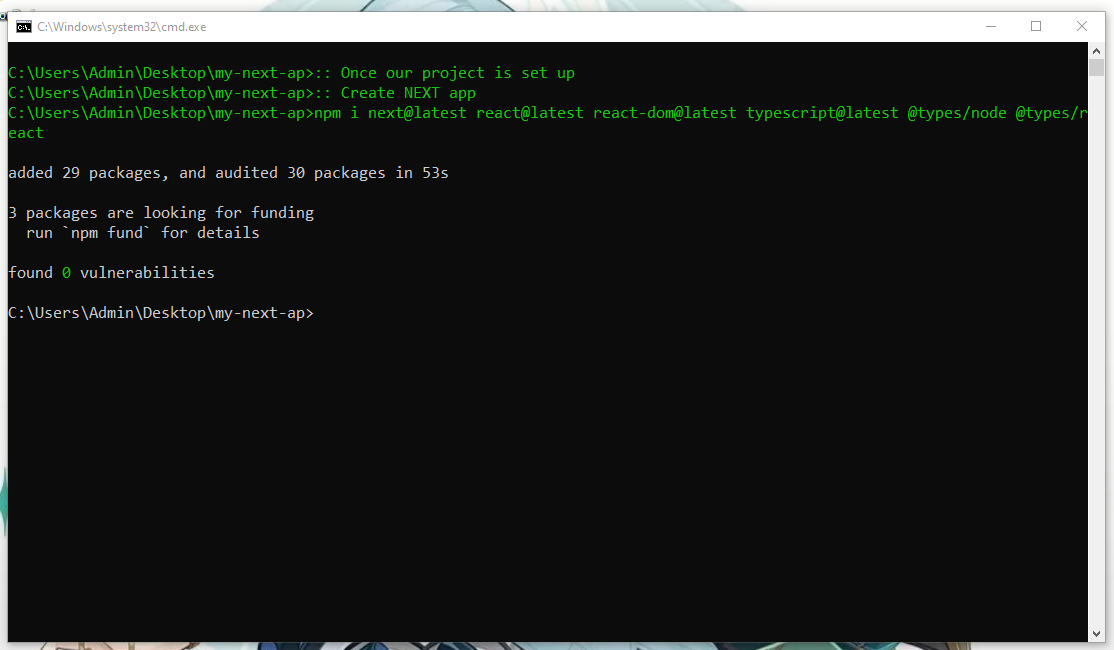


Fig. 21.: Installing packages

Despite packages being installed, our project is still empty:

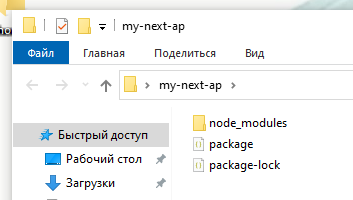


Fig. 22.: Directory of our project

To start our project or try to see what happens if we attempt at starting the webserver at the address `localhost` on a port 3000, we have to modify the `package.json` file:

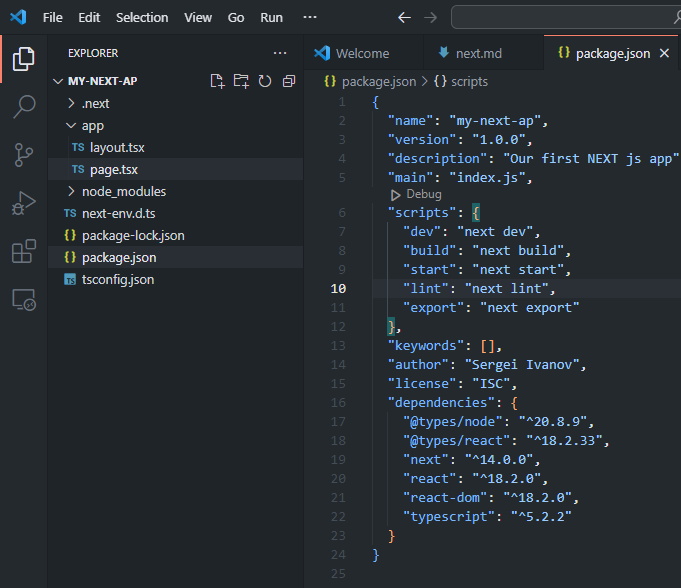


Fig. 23.: Configuration of `package.json`

The command `npm run dev` will start the server; however, if we run it now, NEXT would throw an error:

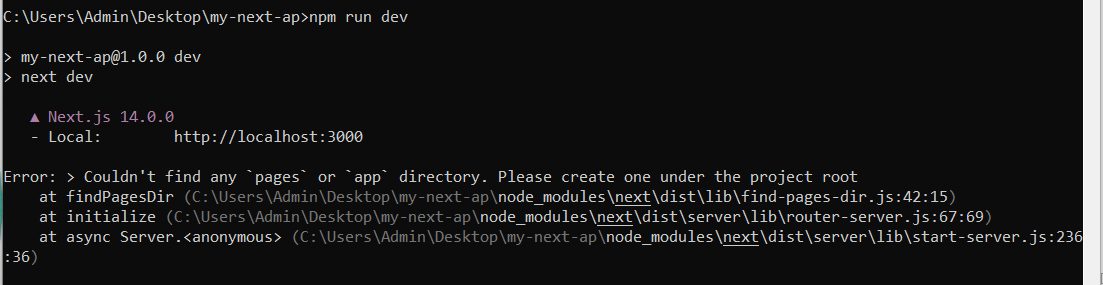


Fig. 24.: Error message: Cannot find the project root

To resolve the error, we have to choose the Router and create the corresponding folder structure:

1. `pages` directory corresponds to Pages Router and follows the following structure:
   1. `pages/\_app.tsx`: an important file recognized by NEXT that serves the purpose of customizing components hierarchy: `<Layout><Nav /><Home /></Layout>`.
   2. `pages/\_document.tsx`: this file allows us to customize the `<Head></Head>` where we include `<link />` and `<meta />` tags.
   3. `pages/index.ts`: this is the root file, where HTML of our page is contained.
2. `app` directory corresponds to App Router which is the new routing idea embraced by NEXT. It requires `app` folder, unlike the previous router that required `pages` one.

Hence, let us create `app` folder and `page.tsx` file:

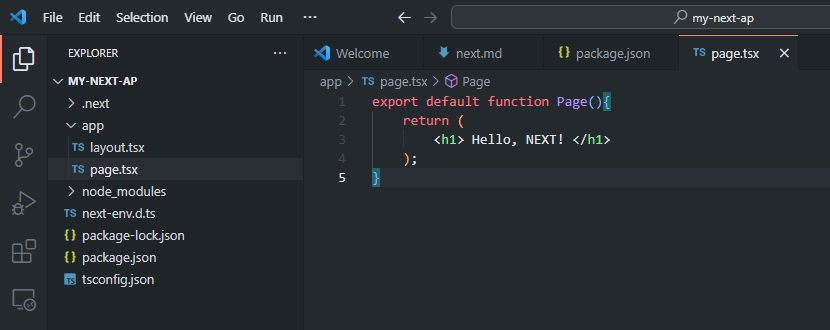


Fig. 25.: Creating our Home page

Please, notice that I did not create `layout.tsx` and did not add `metadata`, and did not clarify for my fonts which must be used across all pages. It was done automatically by NEXT when running the development webserver:

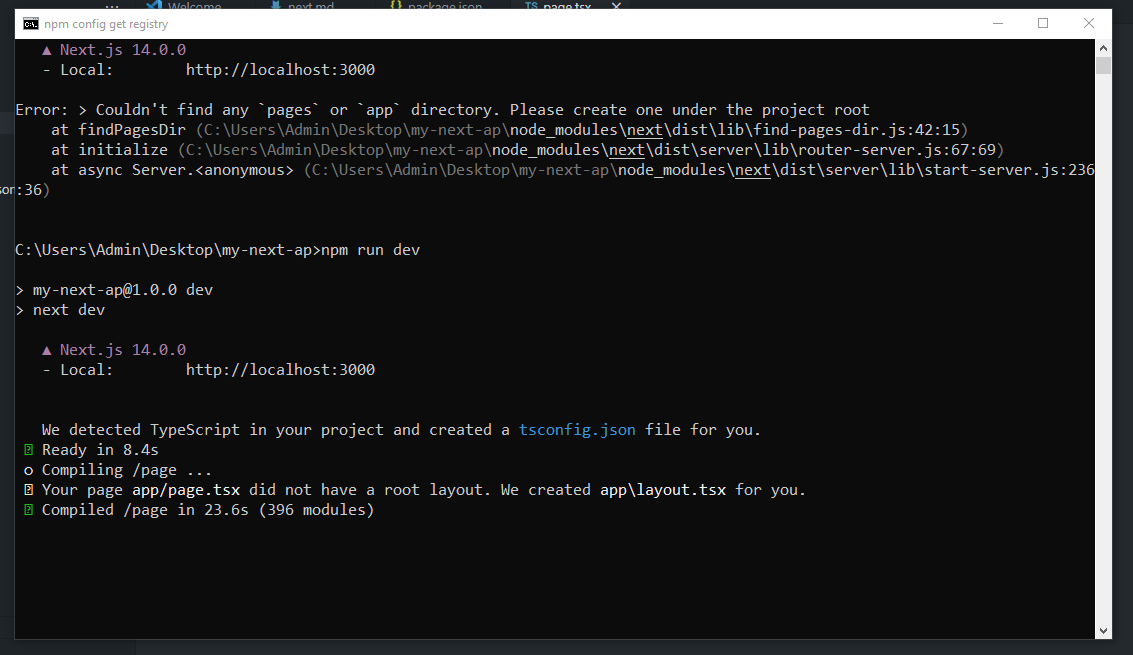


Fig. 26.: Server-side log: Creating layout.tsx and running webserver



Fig. 27.: Client-side

On the screen, we can see the message “Hello, NEXT!” which is exactly the string contained in the `page.tsx`. It is interesting to notice that we change it to another string, and save the file, NEXT will re-render the content and we will see the new string on the screen without refreshment the page.

### What is next?

Once we created the project, we can add more content to it. Basically, we still work on both sides: front-end and back-end. When working on the first one, we use CSS and TSX, as well as UI/UX libraries for NEXT. One such library is NEXTUI which works on tailwindcss, meaning we have to use this preprocessor of CSS, otherwise it will not work.

The structure of our project will be:

1. app folder – this is the main folder.
2. prisma – this is where Prisma resides.
3. models – this is for schemas of Mongoose or Prisma (MySQL, PostgreSQL).
4. shared – this is for utilities (we can write Node.js code there or getting data logic).
5. app/api – this is an important folder: we can use it for rendering content on UI side. According to NEXT documentation, it is also “client-side” where we can use “useEffect” hook from “react”.
6. .env – local environment variables.
7. .gitignore – this is important file where we can specify .next, node\_modules, .env, next-env.d.ts as files to be ignored when adding to GitHub repository. Nobody should expose their secret API tokens used for authorization.

This is not our topic to dive into NEXT in this given document, as the primary goal was to explore the nature of daemons.

### Is NEXT daemon?

It is interesting question, but it can be answered quite simply. The idea is that NEXT is actually a library or a package, not any webserver or a daemon. Internally, it is starts Node.js webserver which is again not a daemon, but a regular process waiting for requests and processing them in terms of HTTP protocol.

Hence, when implementing back-end logic, NEXT supports GET and POST methods of Node.js. This is an explicit indicator of what NEXT uses under the hood, and why it was created.

The daemon-like behavior is the Node.js webserver which monitors the source code in the root file and re-renders the content unless props did not change.

## Creating Flask app

### Introduction

In this section, we discuss Python’s approach to webservers. We have two options:

1. Flask
2. Django

However, we can also implement back-end logic for `index.html` using `http` and `socket` libraries, which is not the most efficient approach.

### Creating our first app

Like before, we are going to create a new project and initialize project. It is done using `poetry`, but I would like to skip it for now.

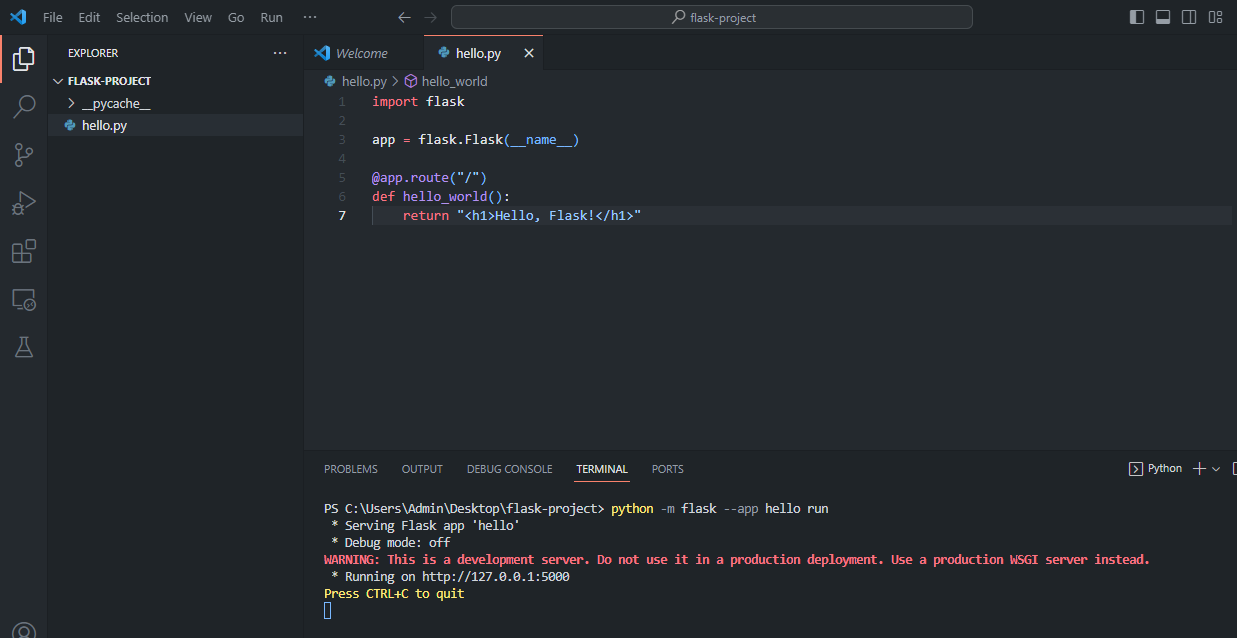


Fig. 28.: Running webserver using Flask

### What is Flask’s webserver?

Unlike NEXT, Flask application uses another client-server communication protocol: WSGI. This is because Flask uses Python, while Node.js – JavaScript, the scripting language of most browsers.

For this reason, Web server gateway interface (WSGI) was introduced as a bridge between Python’s Flask application and webservers like Apache or NGINX which use HTTP as the main protocol of client-server communication to transfer packets of data in the HTTP format: HTML, CSS, JS, or JSON.

Similar to Node.js, Flask is built on top of ‘C++’ libraries to enable networking; it handles TCP, IP, FTP, SMPT and other protocols. The library `socket` in Python is built-in, so we can import it in Python shell and see what it does.

### Python’s http

This module in Python is associated with HTTP. In `http` folder, we can find Python open-source code that is related to HTTP: client.py, server.py and so on. To put it simply, this module contains valuable source of information on the subject of HTTP: 1) request methods, 2) status codes, 3) cookies, and so forth.

### Python’s socket

Using `socket`, we can simulate client-server communication. In fact, if neighborhood has shared IP address, then neighbors can communicate between each other using socket connection and the network server which can respond to requests. One of neighbors creates socket server and hosts it on its machine, while other neighbors of the shared network can connect to it and send a few requests to their neighbor. The server is running continuously until internal error, and can be daemonized to be controlled by SYSTEMD.

### Watchdog

In this section, I discuss how to implement pseudo-hot-code-reload functionality. Idea is that I want to monitor `main.py` file and once changes are committed to it, `watcher.py` restarts the server or re-runs the command `py main.py` for me automatically.

For implementing this program, we have to use the following modules: `watchdog.events` and `watchdog.observers`. Because we want to restart our application once changes are found, we must also import `subprocess` module.

A screen shot of a computer program

Description automatically generated

Fig. 29.: Importing all necessary modules and creating our event.

The next step is simple: we have to initialize our observer and event handler; register it and specify the path to it:

A screen shot of a computer program

Description automatically generated

Fig. 30.: Creating observer and registering the event handler.

The source code of the target file is simple:

A screen shot of a computer program

Description automatically generated

Fig. 31.: The target file to watch.

To start the watchdog, we have to

A screenshot of a computer

Description automatically generated

Fig. 32.: Running the application.

### Python’s daemonize

Python’s library daemonize is a tool that can be used to daemonize a Python program to make it controllable for SYSTEMD. The similar library for windows is `Windows service` that makes the

program run as a windows service.

## Redis

Remote dictionary server, or Redis, is an in-memory data structure store that is used as a database, similar to that of mongod, running in the background.

To install Redis, the developer opens the browser and navigates to the official website from where it can download the program in either as a zip or other formats. Once installed, assuming Linux machine, the developer starts the server by running `redis-server` (provided that these commands `make install` and `make test` were executed) and the client-server: `redis-cli`, which is the client side of the application, where it can send a few requests to the server. Upon further observation, the programmer discerns that the server is running on localhost, at some port and having some “pid” (process id).

Most of requests are operations over data in terms of CRUD. Redis allows developers to store data in key-value pairs where keys are unique, and values are hashes, lists (doubly linked lists), unordered and ordered sets. Please, notice that C++ uses Red-black trees to implement `std::map` - an ordered hash map, while Redis’ implementation of set lies behind the data structure `skip lists` (‘search for the key first’ when implementing insertion and deletion). Unlike lists, arrays, or sequences, `skip lists` have the node structure, similar to linked lists. Hence, Redis is a great tool for programmers to work with data structures.

It is interesting to observe that Redis can be a daemon process because the programmer can close the terminal where `redis-server` is running and focus on the client-side. This is because Redis is detached from the terminal. For debugging purposes, developers use the command `redis monitor` for starting another process to watch the client-side source code and remember commands that were used there. Redis is processing in the same way as is `mongod` (NoSQL database manager).

Therefore, developers can run database server locally and make it accessible across network in terms of the HTTP protocol which consists of TCP and IP:

1. TCP is the transport control protocol, and its main job is to ensure the delivery of packets of data across network:
   1. Segmenting: the data is broken down into chunks.
   2. Ordering: when data is being reassembled, TCP ensures the correct order of it before passing it over to IP.
   3. Delivery: TCP ensures that data is actually delivered (there is no error status) and reassembled in the correct order.
2. IP is the internet protocol used to work with networks and addresses (IP addresses are usually in octal numeral system, while they may be represented in binary or hexadecimal numeral systems).

## PM2

### Introduction

On Linux, there is a disadvantage of using `server` as a terminal running program:

1. Terminal dependency: when closing the terminal, the application is terminated, which is not desired when creating web applications.
2. When Application crushing, there is no restart functionality.
3. Suppose the application has many services that require their own terminals to run. Managing such a system becomes cumbersome.

PM2, process manager, was created to address these issues, so developers could run their node.js applications more confidently. It helps them to debug and fix issues, as well as ensure that their app is stable and will not crush. The benefits of using PM2 when developing node.js applications:

1. Daemonization – the application is detached from the terminal.
2. Automatic restart: when server undergoes internal error, it can be automatically restarted.
3. Centralized management: user-friendly interface for managing multiple applications.
4. Logs and monitoring: log management and monitoring features to debug the application.
5. Process control: simplifies creating and management of multiple processes by controlling them using PM2 functionality, treating them as separate services.

### How to run?

To run our node.js application using PM2 is to use the command: `pm2 start app.js`. Before running this command, we have to install PM2 using `npm` (node package manager).

One of the main features of PM2 is that we can daemonize our application by specifying the flag `--daemonize yes`. This command detaches our process from the terminal and makes it stay running continuously in the background.

## Apache

The classic example of the webserver being special type of a daemon is Apache. Like any daemon, Apache is detached from the terminal and runs in the background; like any webserver, Apache listens to a port and waits for the incoming requests from multiple users. The latter is of great importance because not only must a webserver respond to requests in terms of client-server communication ensured by HTTP, but it also must serve billions of users across the Internet.

Before diving into Apache, it is essential to notice that XML and AJAX are not webservers nor daemons, but modern web technologies for creating requests and UI/UX. When developing Java GUI (graphical user interface), the reader might encounter the XML file of the application that contains the significant information in the special format, which resembles HTML (Hyper text markup language). The AJAX technology is used to send asynchronous requests to the server.

The interesting part of Apache is how it handles a large number of clients concurrently. It accomplishes it through parallelization, which means that Apache creates as many threads as it needs to process requests in the most efficient way. While using many threads, Apache also controls them by the lock or mutex, not allowing other processes to proceed unless all threads completed their tasks. This principle of programming is called multithreading.

## NGINX

### Introduction

The NGINX web server is another way of running server-client applications on Linux. This server is also a daemon process and waits for the requests using the underlying HTTP protocol for client-server communication. The difference between NGINX and Apache is how they handle multiple parallel users concurrently: NGINX avoids multithreading and processes requests in the main thread. This approach diminishes the number of threads and the time for which all these threads would have to wait to proceed. Hence, the performance of NGINX is faster when compared to Apache.

The approach of NGINX is event-driven and asynchronous, leveraging the event loop without blocking I/O operations.

### Event loop

The event loop is how NGINX manages multiple processes in terms of asynchronous architecture, like Node.js:

1. Initialization: declare empty queue of events and stack for callbacks.
2. Register callbacks: events or tasks associated with the functions or callbacks (in JavaScript, the callback is the anonymous function: `() => console.log(‘Hello from callback’);`). When an event occurs, these callbacks get executed.
3. Event monitoring: The even loop is continually monitoring for events (network connections, timer expiration, data availability or others).
4. Event queuing: When an event occurs, it is added to the queue of events.
5. Event processing: events are processed individually.
6. Non-blocking and concurrency: because the even loop uses the asynchronous architecture, it does not have to wait for all events to be completed to proceed. When some event is associated with I/O operation, it will take some time to be completed without blocking the even loop.
7. Event completion: Once a callback is executed and an event is processed, it gets removed from the queue.
8. Cache: When other events use similar callbacks or data, the performance is enhanced through the utilization of cache memory between processing other events.
9. Repeat: The event loop is processing until no events are stored in the queue.

## Conlusion: What is daemon?

Webserver or database managers can work as daemon processes in the background, detached from the terminal which means that the application does not depend on the terminal and can run continuously as a daemon and managed by `systemd` or windows service manager.

Each daemon has its own id associated with it. The name of this id is `pid` or process id. It is then forked with the original process because each process is a node of the process hierarchy tree, hence, each process may have siblings and children.

The creation of the daemon requires to implementing its functionality written in the programming language such as C, C++, Rust, or Python, compiling the program into an executable file and then registering it as a daemon or windows service using systemd and windows service manager, respectively.

Mostly, daemon processes are different due to their purpose: debugging, logging, controlling, creating and so on. For example, mongod is detached from the terminal and waits for the requests such as creating, updating, reading, or deleting some data from database. On the other hand, HTTPd listens to a port and waits for the requests from client.

# Daemons as firmware

## Introduction

Daemons can also assist in many tasks related to computing at computer hardware level. When updating thermal paste in computer, we cannot see neither daemons nor cache placed on CPU. It turns out that cache memory is integrated into CPU, so we cannot observe it with bare eyes. Some firmware, minor chips, is also placed onto computer hardware components to enhance performance or assist in fulfilling certain duties, as well as serving the purpose of sensors. When working on face recognition system, we have to set up sensors for the AI model to collect data and then train on it – the supervised learning in the machine learning field. At computer hardware level, firmware is just minor chip, smaller than motherboard, but also carrying out crucial tasks.

## Examples

There is the following firmware found in processing units:

1. CPU: Basic input output system.
2. GPU: Intel GPU firmware; NVIDIA CUDA; GPU daemon.
3. TPU: Intel GPU firmware.
4. QPU: VideoCore IV QPU.

## GPU-Daemon

When dealing with graphics, computers use GPU to solve any graphical problems. Because images are stored in pixels and represented as multi-dimensional arrays, NVIDIA had to devise the method of optimizing its GPU to ensure the higher performance of their product.

The academic paper that we extended discusses the importance of GPU-Daemon as computational firmware to perform expensive and complex computational operations over multi-dimensional arrays [6]. It is known that Timsort sorting algorithm is not the fastest, but one of the most stable sorting algorithms for one-dimensional arrays and is used to implement both `list.sort` and `sorted` in Python. On the other hand, dual-pivot quicksort is a faster option but not the most stable; this algorithm is used in the source of Java’s sort function.

For multidimensional arrays, much like those used in neural networks or GPU, there are other methods. The first idea is to apply Timsort or dual-pivot quicksort algorithms to sort subarrays of an array of arrays (tensor). For relatively small arrays, it is possible to loop over the array and use Timsort for sorting; therefore, there begs the question as to how to optimize this algorithm for large arrays. For this purpose, developers implemented generators and `for loop` optimizations. What if we could process data in chunks, divisible by the powers of 2? This idea

leads us to parallelization or asynchronous architectures. Indeed, we can follow Apache idea and implement multithreading algorithm for parallel sorting: each subarray is going to be sorted separately. However, what if we could perform the sorting operation in the main thread and instead store each subarray in a queue. Once array is sorted, we remove it from it; otherwise, we wait for asynchronous sorting to complete. The terminal result is stored in the merged array.

Like NGIAX, the NVIDIA algorithm uses cache memory to ensure the efficiency of their algorithm:

1. Initialization: The data is represented by some structure such as a Red-black tree or a linked list, which is reduced to a multidimensional array.
2. Transfer the multidimensional array to GPU by using CUDA (
3. Performing all the necessary GPU-related operations (in the paper case, it was sorting and tensor actions such as multiplication).
4. Transfer back to CPU and convert to the initial data structure, store in the cache.

The last step uses the relationship: CPU makes the request to a controller; the controller, in its turn, starts responding to CPU request by reading the status and the body of the request; as data is requested by CPU, the controller searches for it in cache, or not depending on the architecture. For example, the cache memory levels L1 and L2 may be inclusive, which means that the data will be sought for in L2; if exclusive, there is no reason to do so, unless, we have multiple cores where each core has its own cache levels that of general type. As such, GPU can also request cached data.

# Synthesis

## Conclusion

Even though not being the true daemon, a few firmware can perform their tasks in the background, helping to optimize the hardware component. For example, GPU-Daemon is used to optimize NVIDIA’s GPU calculations and ensure the efficiency, scalability, and credibility of their product. It is accomplished through exploring and employing the efficient and stable algorithms studied in the field of computer science. Like CPU, GPU and other processing units may have cache memory and different cache levels (coherency), as well as implement asynchronous architecture and multithreading to achieve the goal, as shown in the engine V8 used by Node.js.

## Daemon AI

Because of modern technologies and the AI development, people will integrate AI-powered systems in virtually every industry. The first of them might be agriculture, because the human well-being depends on the farmers success. Potentially, AI system might involve many daemons powered by AI to implement many tasks. One such task is monitoring and controlling the quality of a robot – already implemented and used in neural networking, and is called supervised learning, because AI can improve on its training data until perfection. Any background process powered by AI might be a game changing, as it can help split the system into manageable fields to control, maintain and improve. The core idea is to assist farmers in producing food resources by leveraging the power of AI to predict weather, change it; assess the state of crop and apply measures to improve on (for example, AI might spot diseases much faster than any human and fix the plant); facilitate physical work by producing AI-powered machines to perform specific duties on the farm. The subsystem, Daemon AI, can take on training duties related to these machines in order to ensure that robots follow the safety protocol, laws and regulations, as well as perform at the finest quality.

Another idea is to make Python’s interpreter be able to view the source code using sensors, web cameras or simply reading accessed source code of the program in order to help determine issues and fix them, while teaching some syntax rules or algorithms known to it. This will change the teaching industry and facilitate many duties of teacher of CS, or other fields of science such as mathematics or linguistics.

The math behind such concepts requires strong foundation of theory of groups and probability, statistics, calculus and so forth. This is because of the nature of AI: operation over data to achieve better data. However, it might seem fictional, but in reality, it is expensive in terms of money and time, because learning requires time and efforts, while storing data and maintaining systems requires money. Financial resources are finite, and for this reason, some idea might not be implemented in the near future, whereas others will be.

In conclusion, daemons remain the critical aspect of CS and perform various duties in the background; yet they indicate the type of a process that may be a daemon or not. For example, web servers may be daemons or regular processes with that developers can interact using client-server communication protocols. Algorithms to train an AI to produce better data is the core of neural network based on mathematical statistics, theory of probability, theory of graphs, topology and many other fields of science.

# Acknowledgements

I am grateful to the Narva Tartu College teacher, Chen-Wan Lin for the opportunity to research the daemon topic related to computer hardware course.

# References

1. Vercel Inc., 2023, NEXT.js: [Next.js by Vercel - The React Framework (nextjs.org)](https://nextjs.org/)
2. Sergei Ivanov, 2023, Databases
3. Sergei Ivanov, 2023, ComputerHardware
4. Murdoch, Steven J., et al. “The Sources and Characteristics of Electronic Evidence and Artificial Intelligence.” *Electronic Evidence and Electronic Signatures*, edited by Daniel Seng and Stephen Mason, CMB-Combined volume, 5, University of London Press, 2021, pp. 1–50. *JSTOR*, http://www.jstor.org/stable/j.ctv1vbd28p.8. Accessed 2 Nov. 2023.
5. Intel Inc., GPU firmware, 2023: [intel-gpu/intel-gpu-firmware (github.com)](https://github.com/intel-gpu/intel-gpu-firmware).
6. Muaaz Gul Awan, Taban Eslami, Fahad Saeed, GPU-DAEMON: GPU algorithm design, data management & optimization template for array based big omics data,

Computers in Biology and Medicine, Volume 101, 2018, Pages 163-173, ISSN 0010-4825,

<https://doi.org/10.1016/j.compbiomed.2018.08.015>. (<https://www.sciencedirect.com/science/article/pii/S001048251830235X>) GPU-DAEMON template is available at https://github.com/pcdslab/GPU-DAEMON and the source codes for GPU-ArraySort, G-MSR and GPU-PCC are available at <https://github.com/pcdslab>. Keywords: GPU; CUDA; High-performance-computing; Big-data; Omics-data.