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 | Sergei Ivanov | The initialization of the document:   1. Creating titlepage, 2. Preface, 3. Revision history 4. Overview |
| 2.0 | 15.10 | Sergei Ivanov | Creating daemon chapter. |
| 3.0 | 26.10 | Sergei Ivanov | Creating NEXT app, Python Server |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table 1.: Revision history

Table of Contents

[Preface 2](#_Toc149265273)

[Revision history 3](#_Toc149265274)

[Overview 6](#_Toc149265275)

[Objectives 6](#_Toc149265276)

[Goals and methods 6](#_Toc149265277)

[Daemon 7](#_Toc149265278)

[Introduction 7](#_Toc149265279)

[Command Line Interface 8](#_Toc149265280)

[Batch 8](#_Toc149265281)

[Bash 14](#_Toc149265282)

[Server 14](#_Toc149265283)

[Purpose 14](#_Toc149265284)

[Implementation 14](#_Toc149265285)

[Node.js webserver 15](#_Toc149265286)

[Nodemon 17](#_Toc149265287)

[Creating NEXT app 19](#_Toc149265288)

[Introduction 19](#_Toc149265289)

[What projects is NEXT used for? 20](#_Toc149265290)

[What technologies does NEXT use? 20](#_Toc149265291)

[How is the NEXT app created? 20](#_Toc149265292)

[Project initialization 22](#_Toc149265293)

[The creation of the NEXT app 22](#_Toc149265294)

[What is next? 28](#_Toc149265295)

[Is NEXT daemon? 28](#_Toc149265296)

[Creating Flask app 29](#_Toc149265297)

[Introduction 29](#_Toc149265298)

[Creating our first app 29](#_Toc149265299)

[What is Flask’s webserver? 29](#_Toc149265300)

[Python’s http 29](#_Toc149265301)

[Python’s socket 30](#_Toc149265302)

[Watchdog 30](#_Toc149265303)

[Python’s daemonize 30](#_Toc149265304)

[Redis 30](#_Toc149265305)

[PM2 30](#_Toc149265306)

[Cargo and Rust 30](#_Toc149265307)

[What is daemon? 30](#_Toc149265308)

[Synthesis 30](#_Toc149265309)

[Conclusion 30](#_Toc149265310)

[Daemon AI 30](#_Toc149265311)

[Acknowledgements 30](#_Toc149265312)

[References 31](#_Toc149265313)

# 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`):

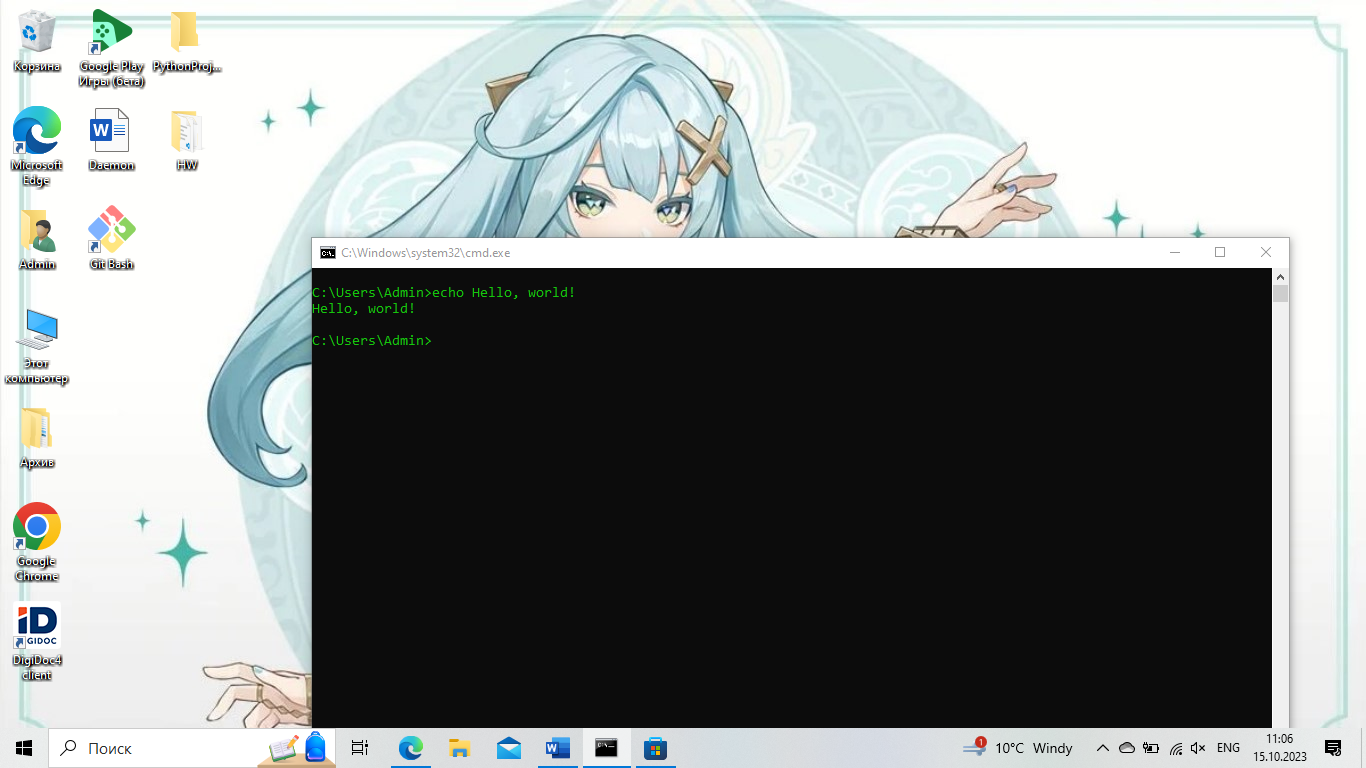


Fig. 1.: Print ‘Hello, world!’ to the terminal.

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:

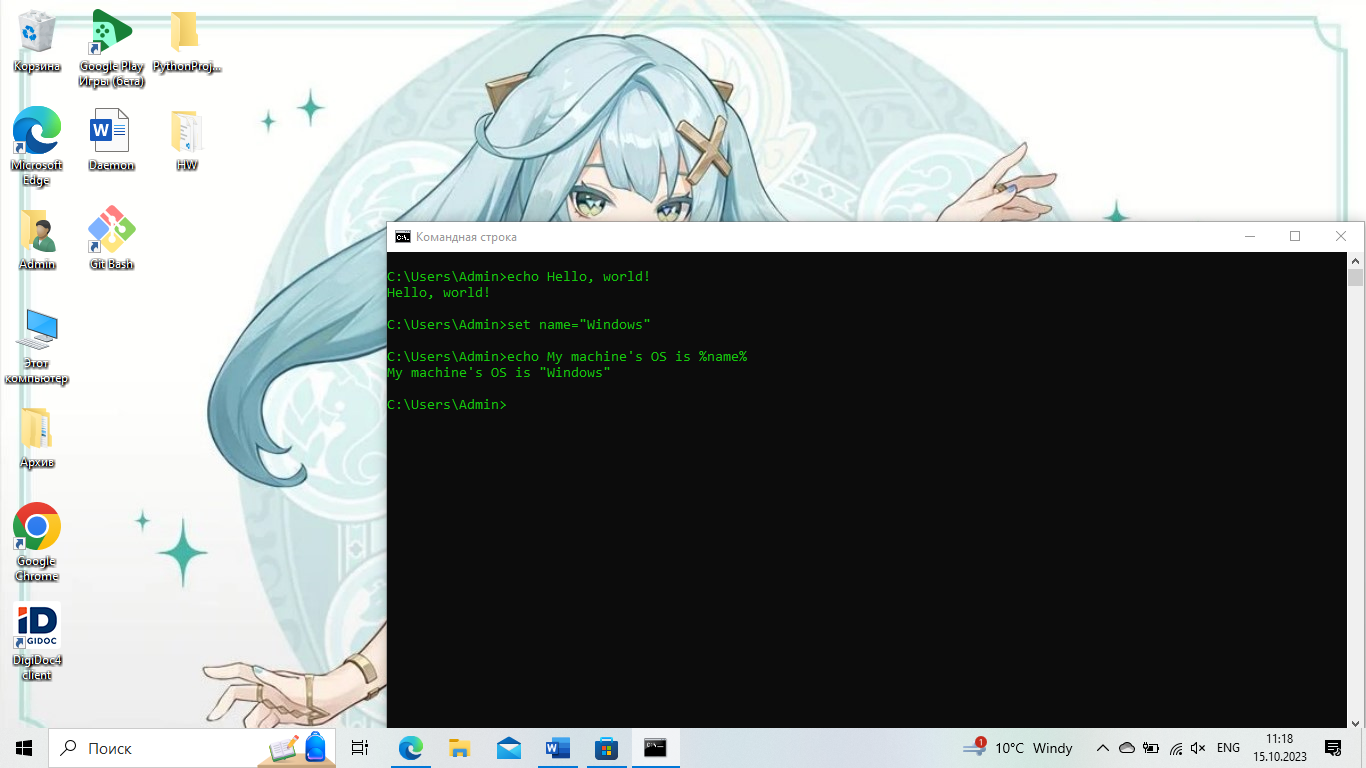


Fig. 2.: Setting variable in Batch.

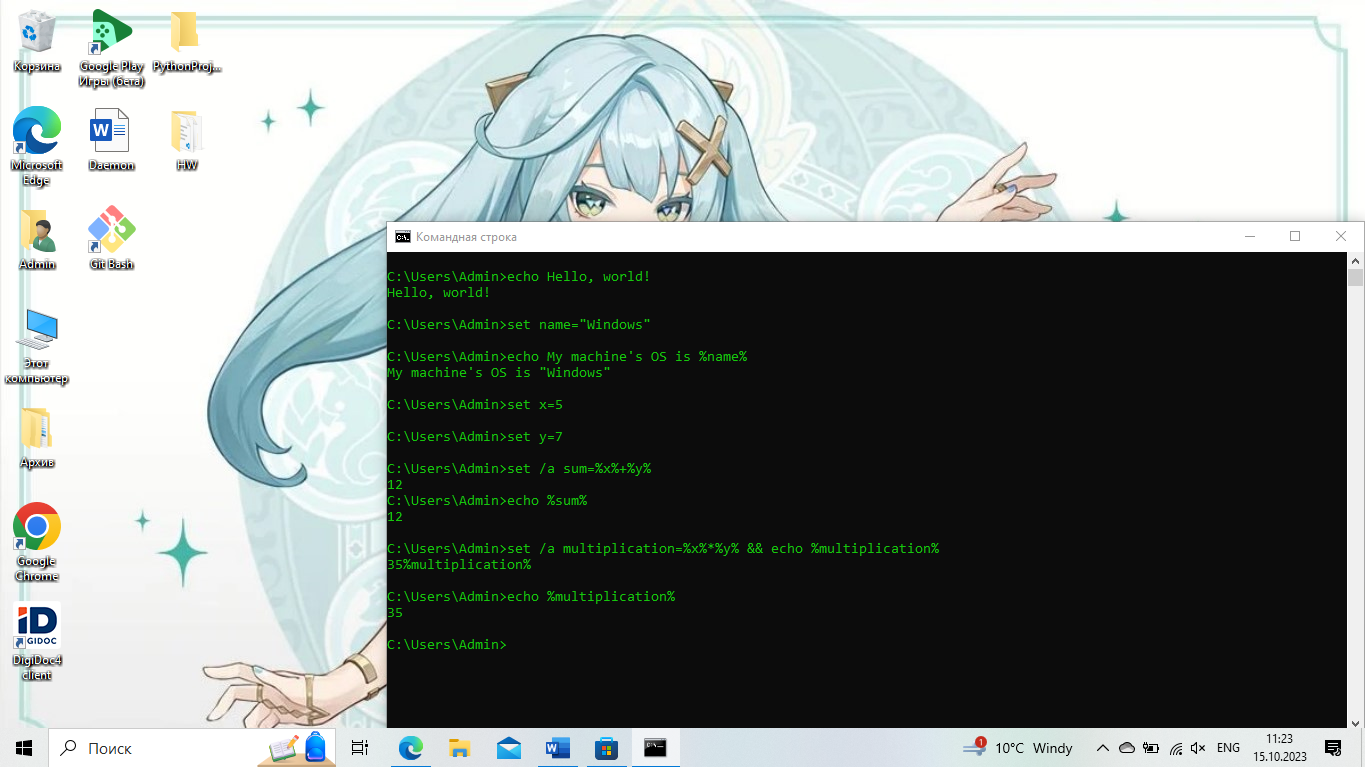


Fig. 3.: How to perform arithmetic actions.

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

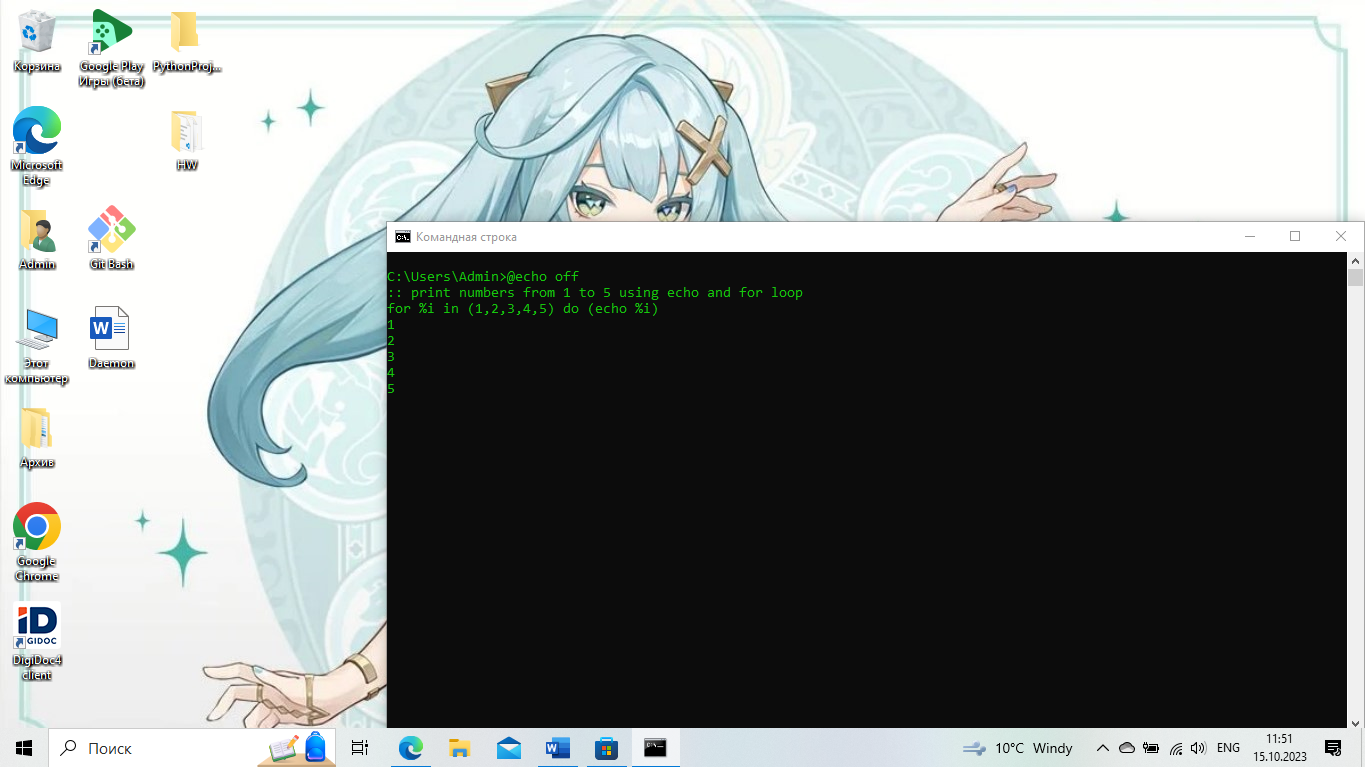


Fig. 4.: How to print numbers using for loop.

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.

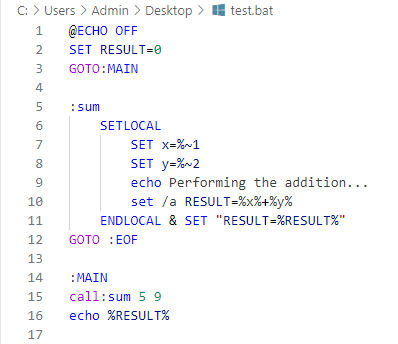


Fig. 5.: How to write a function in Batch.

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

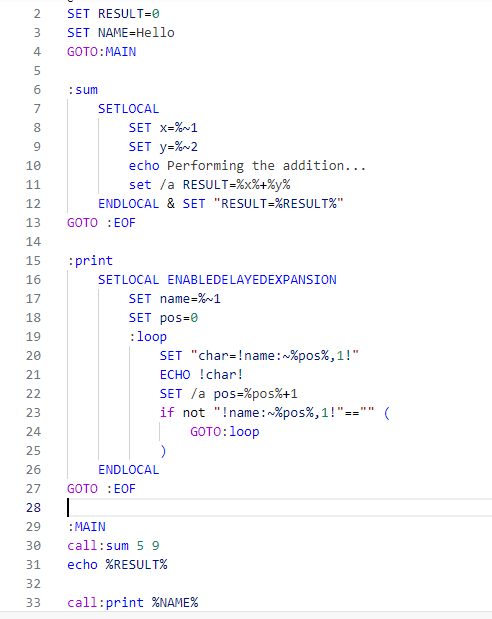


Fig. 6.: Looping over a string.

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.

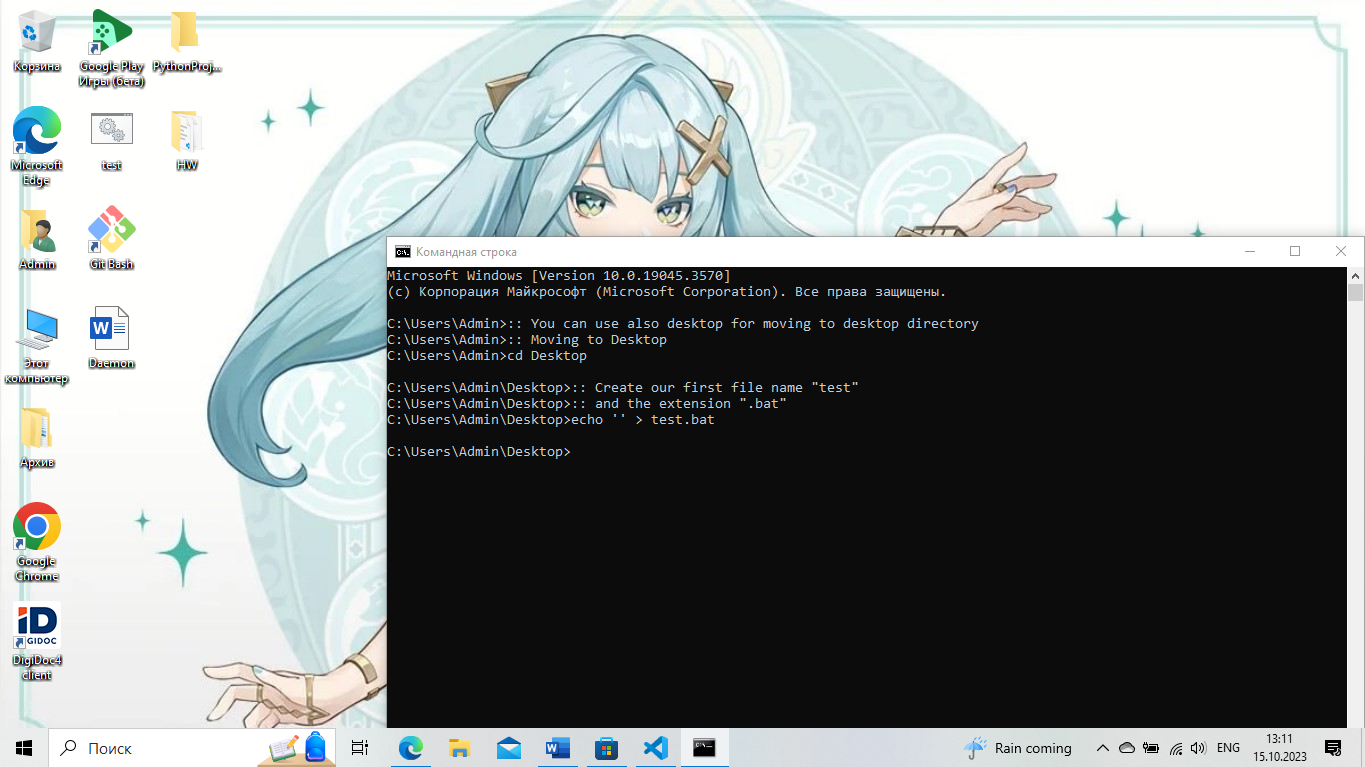


Fig. 7.: How to create a file.

Now let us write the code:

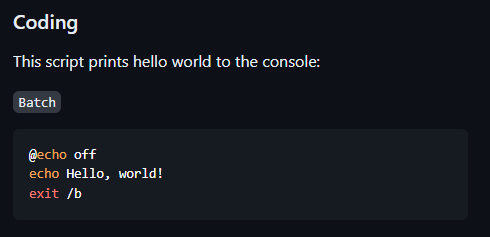


Fig. 8.: The code to write in the created file.

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’:

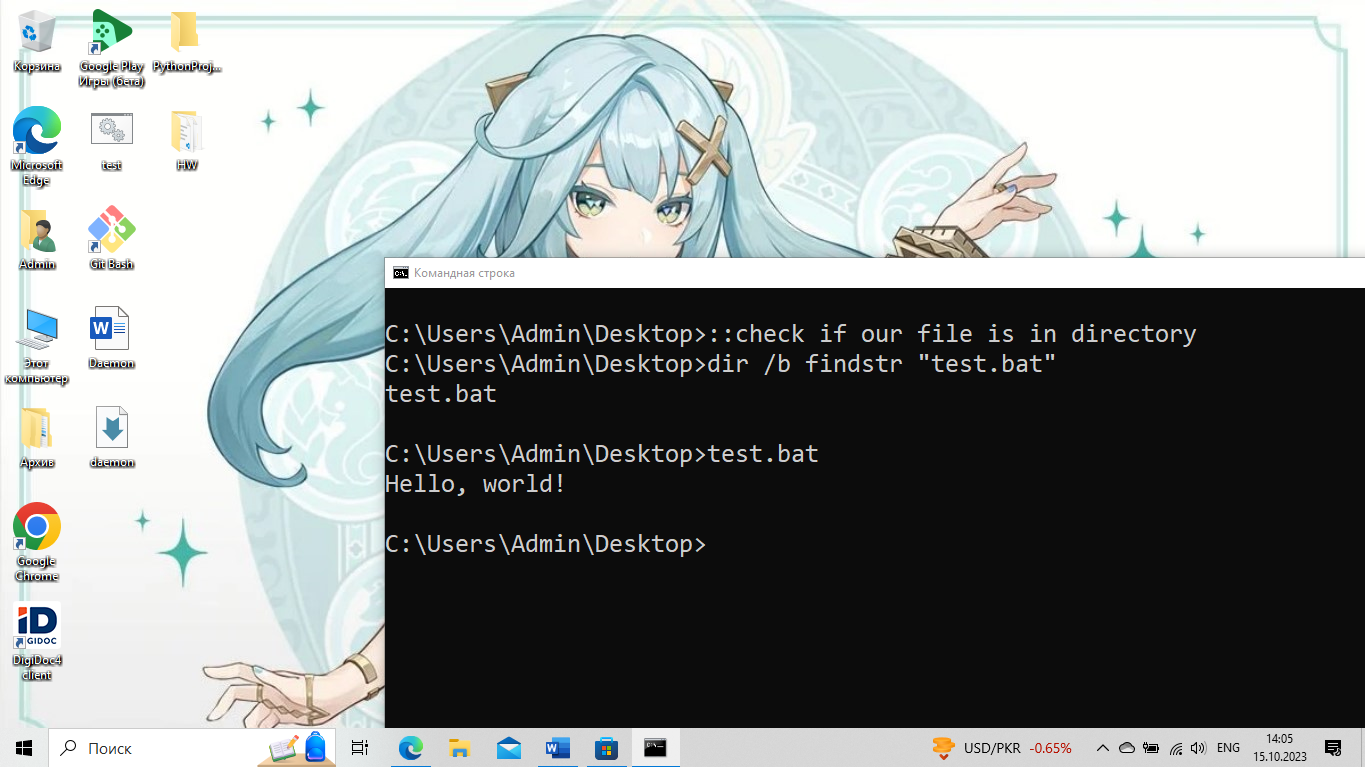


Fig. 9.: How to execute our script.

### Bash

Let us now test drive Linux …

## 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:

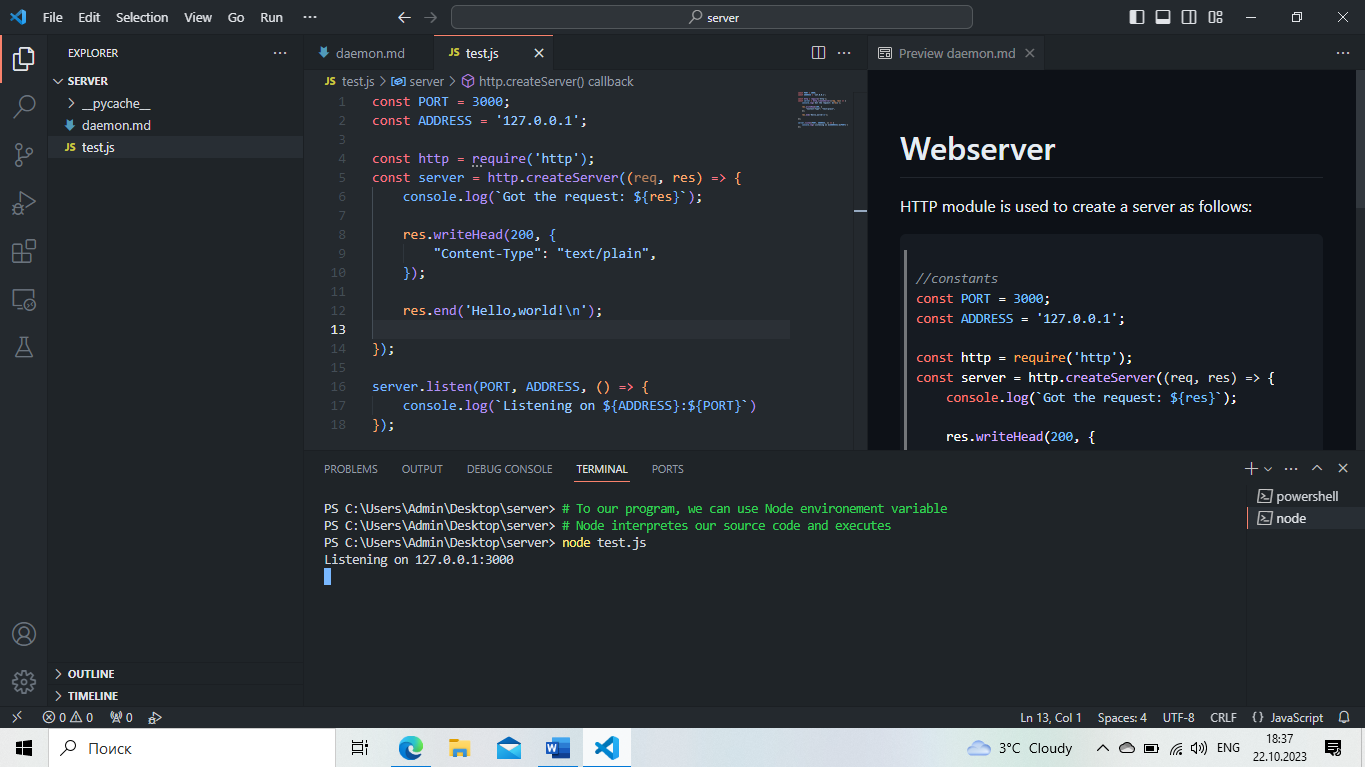


Fig. 11.: Running our program.

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

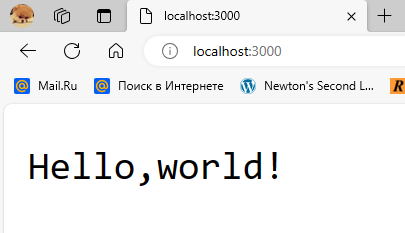


Fig. 12.: Output of our program.



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:

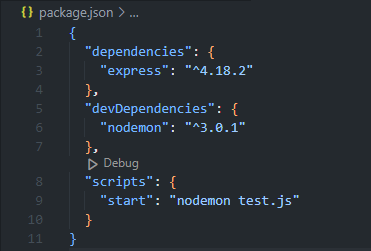


Fig. 15.: Adding scripts to ‘package.json’.

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

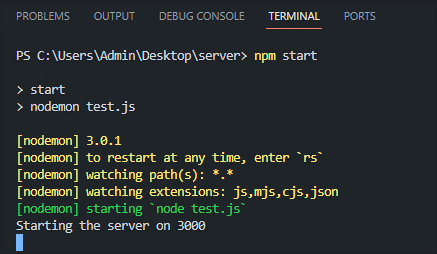


Fig. 16.: Running a server using ‘nodemon’.

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:

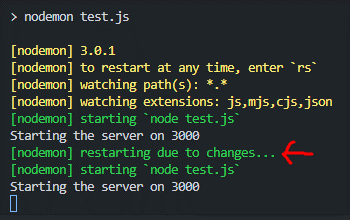


Fig. 17.: Nodemon restarts the Node.js webserver.

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:

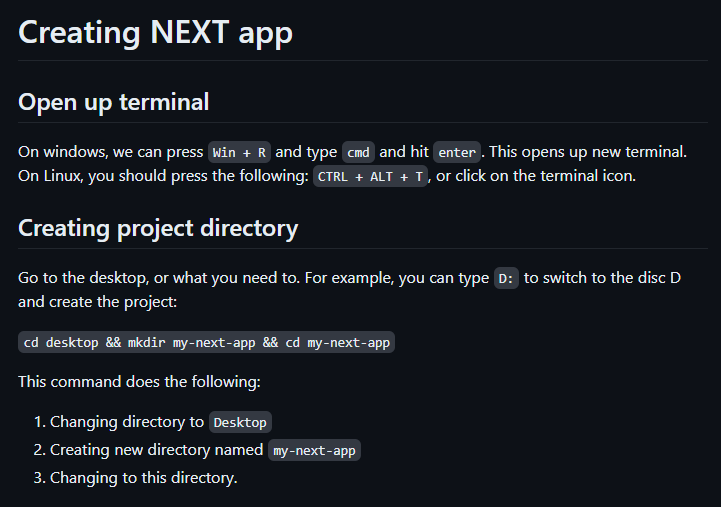


Fig. 18.: Creating our project directory

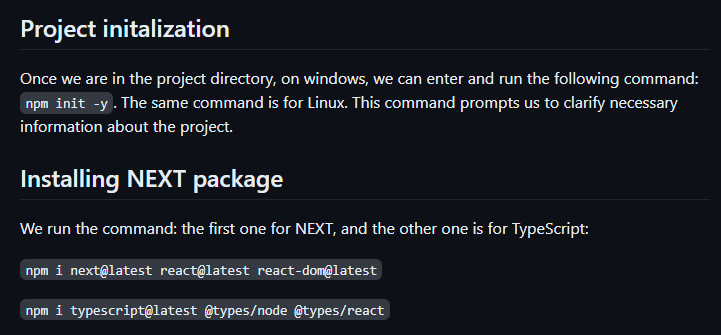


Fig. 19.: Project initialization and

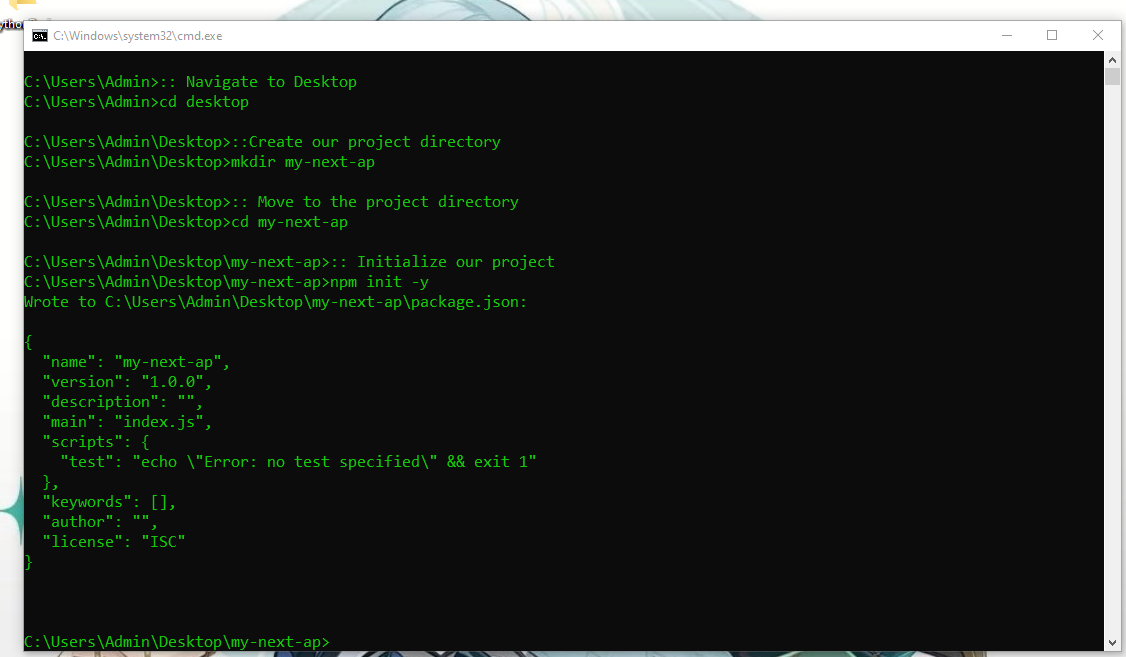


Fig. 20.: Moving to `Desktop`, creating project directory, and initialize it

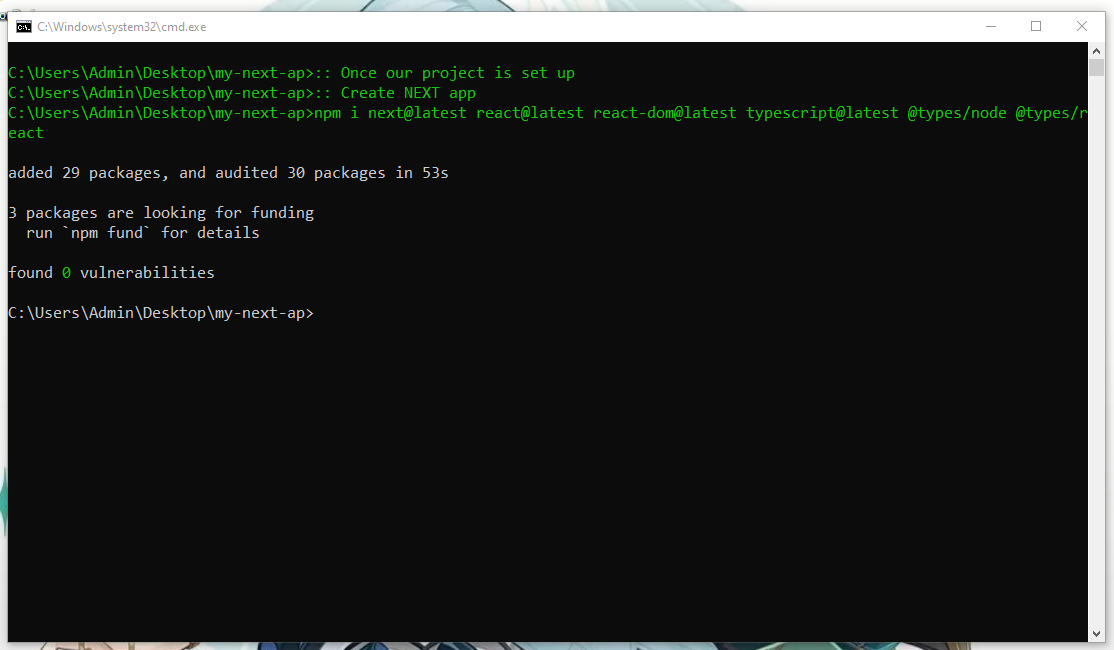


Fig. 21.: Installing packages

Despite packages being installed, our project is still empty:

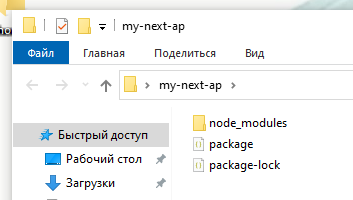


Fig. 22.: Directory of our project

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:

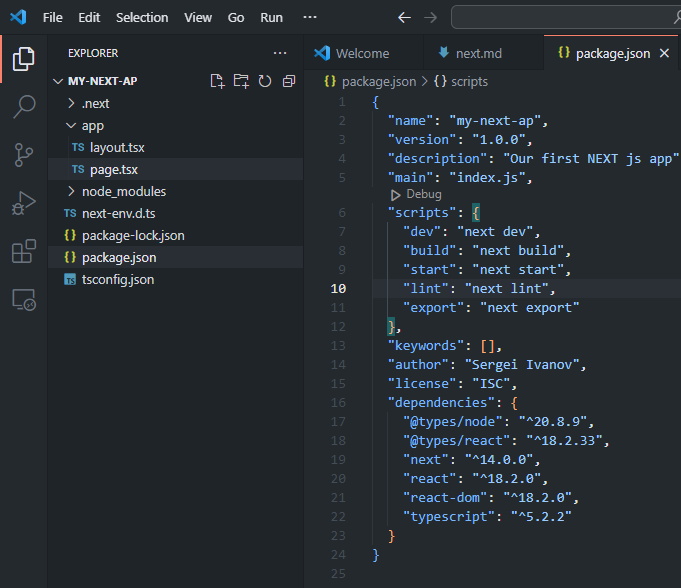


Fig. 23.: Configuration of `package.json`

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

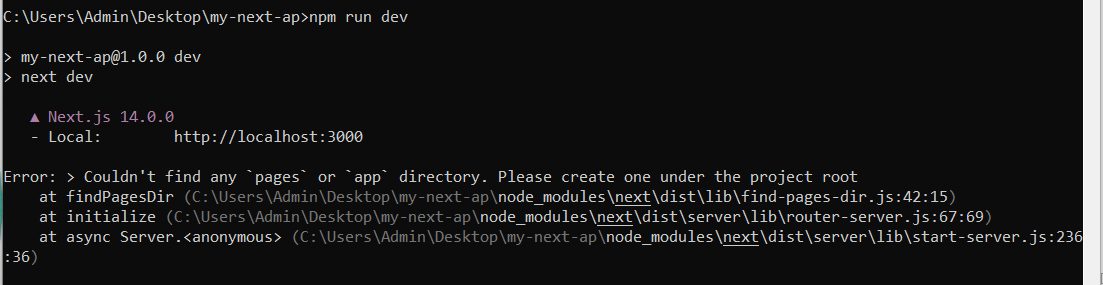


Fig. 24.: Error message: Cannot find the project root

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:

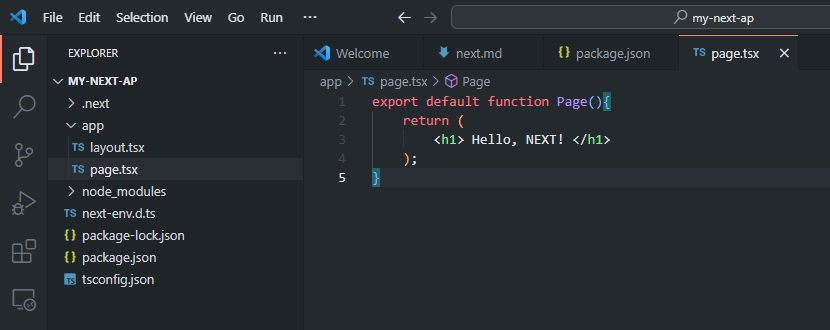


Fig. 25.: Creating our Home page

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:

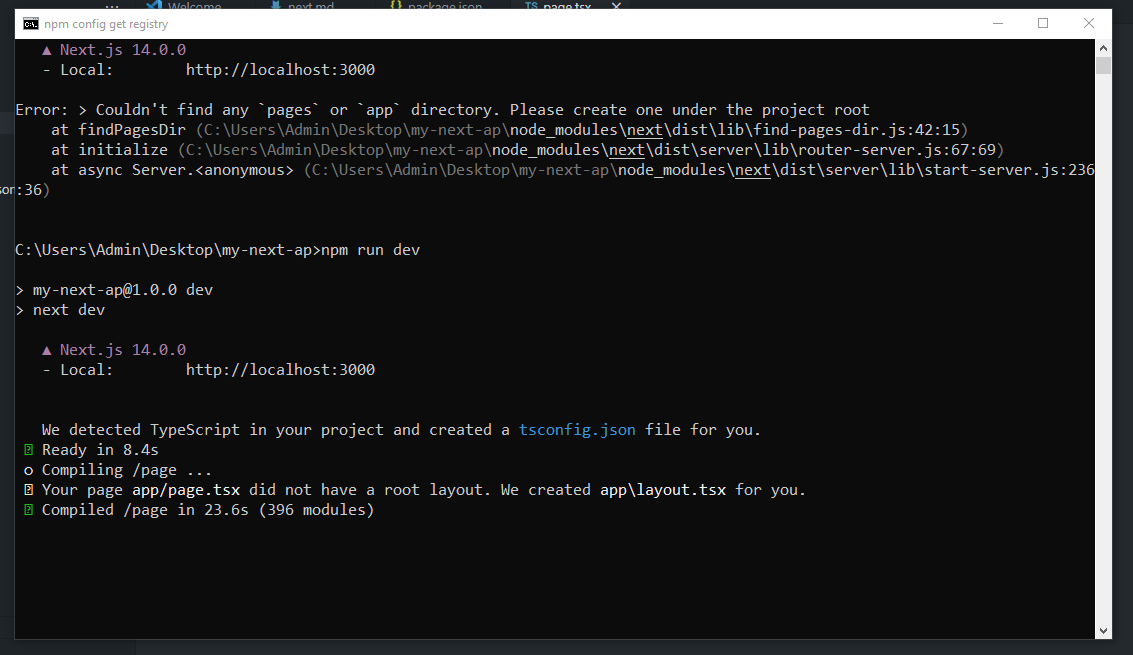


Fig. 26.: Server-side log: Creating layout.tsx and running webserver



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.

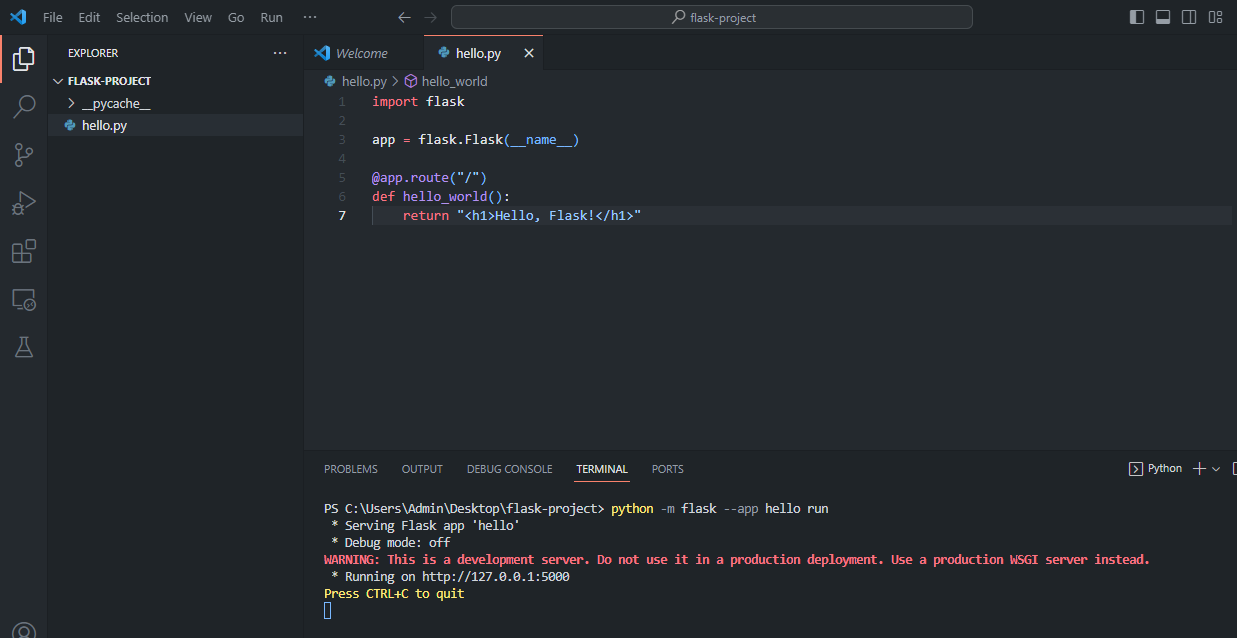


Fig. 28.: Running webserver using Flask

### 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.

🡨TODO: Writing programs here 😊 🡪

### 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.

## PM2

🡨 TODO: Tomorrow! 😊 🡪

## Apache

## NGINX

## Cargo and Rust

In this section, I will touch upon Rust, because we need to understand the programming language Rust to utilize windows service or daemonize crates. 😊

## Conlusion: What is daemon?

A daemon is a process running in the background, performing its duties; it is controlled by SYSTEMD and it is called systemd unit.

🡨 TODO: Synthesis of this chapter🡪

# 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.

# Synthesis

## Conclusion

## Daemon AI

🡨 Tomorrow: My idea 🡪

# 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