Node JS (built on the basis of event-driven architecture)

**Node Js** is not a programming language it is programming platform it allows to turn JS to machine language(code) with the special API written on C++.

Reasons to study Node JS

* Simplicity (learning process can be started easily with the basic knowledge of JS)
* Speed (max speed of request analysis)
* Frond end and back end are written in the same language
* Rich npm manager package which includes a lot of packages
* Fast Prototyping

**V8 Engine –** JS Web assembly language created by google the purpose of which is to translate JS to machine code**.** Provides memory for the objects and deletes objects that are no longer needed.

**Libuv –** together with V8 creates the fundamental for NodeJS. Responsible for cross-platform I/O(input - output) and also event loop.

Cross-platform operations I/O include:

* Work with file system
* Work with web and etc.

Event-loop allows to perform certain tasks without blocking other

positive side: opportunity to work with a lot of I/O operations at the same time

negative side: a lot of asynchronous code and also complicated calculations create the huge load

**Blocking I/O-** the way classic web servers operate (for ex Java) where step by step (line by line) commands are executed where we cannot turn to the next one while the current is executed. If the instructions are simple there are no problems and everything is executed nice and smoothly. But the issue is that instructions are not always simple and sometimes require complicated ones like, receive file go through it and save it in certain database where at the same time with blocking model of behavior the whole thread is blocked => thread is busy with certain input or output operation, which leads to the fact that application is not able to perform other operations. In order to solve that problem such operations are performed in different threads (for example, one is reading from data base, another records file to the disc, and another is processing the http request). This is a great solution but it has certain negative sides. Firstly, thread takes a lot of resources and there are some cases when thread is created but is standing still for a while, and resources for this thread were provided by the operation system. Secondly it is complicated to manage these threads.

1 thread

Connection 1

Server



Data process 4



Data process 3

Data process 2



Data process 1

Connection 2

Connection 3

3 thread

2 thread

**Non-blocking I/O** – system requests immediately return the management without waits for reading data or data record. At the same time thread is not blocked. The time of thread inaction is very small.



Data process 4

MainThread

Server

Connection 1



Connection 2

Connection 3

Data process 3

Data process 2

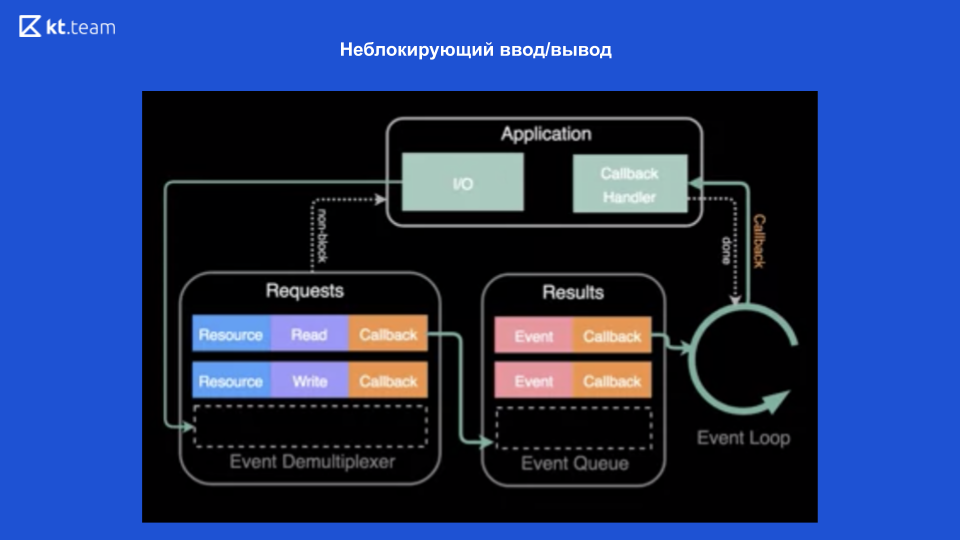
Data process 1

**JavaScript** is single-threaded and asynchrony is achieved with event Loop.

**NodeJS** is single-threaded (so node js developers write asynchronous code without the threads usage) => Libuv is in the basis of NodeJS which has the ability to manage threads, the default amount of which is four, which can be changed (the problem here is that operations such as reading the file or recording it to the disc are very heavy since they are performed in one thread). Some libraries can be multiple-threaded !!

**Thread scheduler** part of the operation system which is responsible for the parallel tasks execution. It provides threads with the processor time memory and other resources.

**Non-blocking I/O** became possible because of the mechanism provided by modern operation systems – **Event Demultiplexer** ( Демультиплексор событй )=> this is the mechanism that receives the request from application, registers it and executes it.



**Reactor design pattern**is an event handling pattern for handling service requests delivered concurrently to a service handler by one or more inputs. Consists of Event Demultiplexer, Event loop

When Node JS is launched it initializes the **event loop,** Node Js is processing the input code which can perform the asynchronous API calls set different timers and etc., after the event loop processing begins. Each of the phases of event loops that are provided on the diagram bellow has the callbacks’ order. When event loop turns into a phase it will perform any operations related to this phase then it will do callback from the order until it will be exhausted after that it goes to the next phase and performs the same procedure.

1. Timers (callbacks which are planned with two functions: setCallback and setInterval);
2. I/O callbacks (almost all callbacks except the ones in other phases and setImmediate in Node JS);
3. Idle,prepare (only for inner purposes);
4. Poll (receiving if new events of I/O);
5. Check (callbacks that were identified with setImmediate callback);
6. Close callbacks (for ex closing of web socket connection, stream and etc)

Event Loop

┌───────────────────────────┐

┌─>│ timers │

│ └─────────────┬─────────────┘

│ ┌─────────────┴─────────────┐

│ │ pending callbacks │

│ └─────────────┬─────────────┘

│ ┌─────────────┴─────────────┐

│ │ idle, prepare │

│ └─────────────┬─────────────┘ ┌───────────────┐

│ ┌─────────────┴─────────────┐ │ incoming: │

│ │ poll │<─────┤ connections, │

│ └─────────────┬─────────────┘ │ data, etc. │

│ ┌─────────────┴─────────────┐ └───────────────┘

│ │ check │

│ └─────────────┬─────────────┘

│ ┌─────────────┴─────────────┐

└──┤ close callbacks │

└───────────────────────────┘

process- глобальный объект с помощью которого мы можем получить некоторую информацию о текущем процессе;

.join- позволяет склеить пути вне зависимости от операционной системы

\_\_dirname- path to the current directory

.resolve – allows to get the absolute path

STREAMS (the way to read file by parts

)

There are four types of streams

Readable – reading

Writable-writing

Duplex – for reading and writing Readable + Writable

Transform – The same as Duplex, but can change the data during reading