Node Js

18/2/2020

Overview

Node.js uses V8 to compile JavaScript directly to native machine code for server-side execution. This enables the development of asynchronous event-driven web servers capable of handling hundreds-of-thousands of concurrent connections.

- Why Node.js?

  Very fast

  Asynchronous and event driven

  Single Threaded but highly scalable

  No Buffering

- Why Node.js is fast?

  The speed of Node.js is mainly down to two things: the V8 engine and its non-blocking feature.

Objectives of project

- Learn the basics to program in NodeJS
- Improve your knowledge in Javascript
- Improve your knowledge in Backend

The pill is organized based on the following requirements;

- HTTP requests
    - One of the fundamental pillars of any backend language is HTTP request management. To do this you will have to create:
    - An HTTP server:
    - Require the native module "http"?
    - Receive HTTP requests on port 3000: That you can read the GET parameters by using the url module, for example, if you call the url localhost: 3000? User = john you should be able to get the value "john" using its identifier name
    - Send a response to that request with the following headers
        - Response Code 200
        - Content type text / html

What does the native module called filesystem import

- That through this module you can read a file with an .html extension that you have previously created to show it as a response to the request. It is important that you verify previously that the file exists so that you learn new features of that module.

- You must create a file with a .json extension in the root of your project that stores each of the requests made to the server. This information will be stored within an array. For each request an element will be added to said array that will be reflected in the .json file. It is important that you make sure that if the file exists, do not create it again.

- In addition to updating the .json file mentioned above, it will be necessary to display a message with the information of the request that you consider relevant. If you use VSCode, install the debugger for NodeJS to debug your server.
- Research (which you must include in the pill documentation)
- Investigate what methods included in the fs library are synchronous and what methods are asynchronous for checking whether files exist or not.
- What is the difference between them and when do you recommend their use
    - investigate how to use environment variables in NodeJS so that the port of your HTTP server is obtained through them. In case the environment variable does not exist, you will use a default value.

Tests using POSTMAN

You must create a POSTMAN collection with all the methods available in your app to be able to test. Take the necessary tests to verify that your server works as expected

Table of Contents

1. Overview
2. Objectives
3. Project requirements
4. Risk management plan
5. Tasks for the project
6. Chronogram
7. Calendar
8. Git Workflow
9. Technologies used
10. Incidents
11. Theoretical questions

Project requirements

- Requirements documentation.
- List of tasks to be performed.
    - Priority of each task
    - Title and description of each of them
    - Difficulty level
    - Estimated time for each task.
- Record of incidents that were detected during the execution of the pill.
- Schedule or Calendar of the project. (Choose the one you consider most appropriate given the size of the project)
- Quality metrics (what points do you consider important to measure the success of the project)
- Documentation about the git WORKFLOW you are going to use
- Documentation about the tools used in the project
- Record of lessons learned

Risk Management

Every project has risks. These risks must be taken into account to improve the workflow of the project. Managing these risks is important for due completion of the project. Unchecked risks could not only lead to hamper of project deliverance but also a badly executed project. The risks hence associated with the project are documented as follows:

Risk	Risk level
Unfamiliarity with NodeJs	Medium to High
Installation of NodeJs	Medium to Low
Health and computer issues	High
Delivering in time	Medium
Completing all the project requirements	Medium

Task Management

The following are the tasks which needed to be accomplished for the completion of the project. The tasks have been divided according to the project specifics. Each task has been clearly defined and their priority as well as their difficulty and time needed. Difficulty level is explained on a scale of 1 to 5 (5 being the most difficult). Priority is explained on the level of 1 to 5 (again 5 the highest parameter being the most prioritized work).

Task	Priority	Description	Difficulty	Time
Read the description of the project	5	This task involves complete understanding of requirements for the project	1	30 min
Create git repo	5	Creating of git repository for project execution and delivery	1	2 mins
Create files for the project	4	Creating the necessary files, folders and subfolders for this project.	1	5 mins
Install Node and NPM	4	Install Node and NPM if not installed previously	2	10 mins
Create an HTTP server	3	Create an http server in nodejs	3	20 mins
Receive requests on port 3000	2	Start getting the requests on the port 3000	4	30 mins
Response code 200	2	Get the response code 200	5	2 hrs
Return requests in JSON	4	Write the requests in a JSON file	5	2-2:30 hr
Use POSTMAN	1	Use POSTMAN collection to get the requests	2	10 mins
Documentatio n	1	Create a project documentation	1	1-1:30 hr

Defining this part is crucial to the development of the project. It is important to make a good analysis of the situation to organize the project in a good way.

Chronogram

This shows the time-line it took for the project to finish and also the tasks that were achieved during those timeline.

The tasks have been mentioned in the left axis

Tasks	Mon 1000 hrs	Mon 1100 hrs	Mon 1200 hrs	Mon 1300 hrs	Mon 1400 hrs	Mon 1500 hrs	Mon 1600 hrs	Mon 1700 hrs
Read project descriptio-n	X							
Git Repo	X							
Create files	X							
Install Node and NPM	X							
Create an HTTP server		X	X					
Receive requests on port 3000			X					
Response code 200			X	X				
Return requests in JSON				X	X	X		
Use POSTMA N							X	

Documentation							**X**	**X**

Git Workflow

For this project, all commits we'll be pushed directly to the Master branch. All commits will use a descriptive message, so that the admin and other internet user can follow through the processes.
For more information go to :
https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Incidents

- None

Technologies used

For this project, we will use the following technologies:

- HTML
- Postman
- Node JS
- Javascript

Theoretical questions

- What does the filesystem import do ?

    - The Node.js file system module allows us to work with the file system on the computer.
        - To include the File System module, we use the require() method:
        - var fs = require('fs');
    - In the newest version of ES6 we could use import instead of require to use filesystems. Filesystems allow us to;
        - Read files
        - Create files
        - Update files
        - Delete files
        - Rename files
- Which methods in **fs** are synchronous and which one are asynchronous and explore the difference between them and recommend which one to use when?
    - The fs module allows the developer to interact with the file system by providing
        - POSIX file I/O primitives
        - File streaming
        - Bulk file I/O
        - File watching
    - The fs module is unique compared with other I/O modules (like net and http) in that it has both asynchronous and synchronous APIs. That means that it provides a mechanism to perform blocking I/O. The reason the file system also has a synchronous API is largely because of the internal workings of Node itself, namely, the module system and the synchronous behavior of require.
    - The synchronous methods include (for all POSIX and bulk API calls):
        - readFileSync
        - statSync
        - readdirSync
    - Async filesystems tell node to read the file in a non-blocking way and get a callback when reading has been terminated