HTTP JSON API Node.js Time Server

Feven Araya
Feb 6, 2024

Table of content

1. Introduction
2. Design
3. Implementation and Test
4. Enhancement Ideas
5. Conclusion
6. References

1. Introduction

Background & Significance

- Essential role of accurate timekeeping in digital applications (e.g., financial transactions, scheduling systems).
- Increasing demand for easy-to-integrate time data over HTTP.

Project Overview

- Introduction of the "HTTP JSON API Node.js Time Server".
- Utilization of Node.js to deliver current time data in JSON format via HTTP.

Objectives

- Provide a reliable and efficient method for applications to obtain current time data.
- Align with modern web standards and practices for easy data integration.

Technology Focus

- Deep dive into Node.js, HTTP protocols, and JSON data format.
- Emphasis on understanding and applying these technologies to create a versatile time server.

Significance of the Project

- Demonstration of technical feasibility and practical applications.
- Contribution to enhancing web application functionality and developer resources.

Vision and Contribution

- Development of a fully functional, easy-to-deploy, and reliable server.
- Anticipated contribution to the open-source community, providing a tool for accurate time data integration.

2. Design

Identify and Understand the Problems

- Real-time Data Requirement: Applications require access to accurate, real-time time data for various functionalities, highlighting the need for a reliable time server.
- Scalability and Performance: The server must handle multiple concurrent requests without significant delays, ensuring high availability and responsiveness.
- Integration Ease: Ensuring that the time server can be easily integrated into different client applications requires a universally accepted data format and communication protocol

Investigate to Find Possible Solutions

- Existing Time Servers: Examination of existing time servers and their limitations, such as lack of customization or insufficient scalability.
- Technology Stack Options: Evaluation of different backend technologies (e.g., PHP, Python Flask, Node.js) and data formats (XML, JSON) for implementing the time server.
- Custom Implementation: Consideration of developing a custom HTTP JSON API using Node.js, capitalizing on its event-driven, non-blocking I/O model for scalability and JSON for lightweight data interchange.

Theoretically Compare the Solutions

- Scalability and Performance Comparison: Node.js outperforms traditional synchronous servers under high concurrency due to its non-blocking I/O model, making it more suitable for real-time applications.
- Data Format Usability: JSON is preferred over XML and other formats for its simplicity, ease of use in web applications, and lower overhead, facilitating faster parsing and serialization.
- Integration and Development Ease: Node.js, combined with JSON, offers a developer-friendly environment with extensive library support, simplifying the integration process with client applications.

Select the Best Solution

- Node.js with JSON for API Development: This combination is selected as the best solution due to its superior performance in handling concurrent connections, ease of development, and the widespread adoption of JSON for API responses.
- Custom HTTP JSON API Server Design: The decision to design a custom API endpoint (/api/currenttime) specifically for serving current time data is based on the need for a lightweight, efficient, and easily accessible time service.

Design Overview

- Introduction to the core architecture designed to facilitate accurate and efficient time data retrieval using Node.js.
- Utilization of the HTTP protocol for facilitating client-server communication, ensuring broad compatibility and ease of integration.
- Adoption of JSON for data exchange, leveraging its simplicity and efficiency in web environments.

System Architecture

- The diagram illustrates the server-client model for your Node.js HTTP JSON API Time Server project.
- The client sending an HTTP GET request,
- The Node.js server processing the request
- Then the server responding with JSON-formatted current time data back to the client.

Node.js and JSON

- Node.js is chosen for its event-driven, non-blocking I/O model, making it ideal for lightweight, high-throughput applications like our time server.
- JSON is selected for data interchange due to its lightweight nature and ease of use in both browser and server environments, facilitating quick parsing and serialization of time data.

HTTP Protocol

- Client Request: Clients make HTTP GET requests to http://localhost:8000/api/currenttime for current time data.
- URI Endpoint: The server sets /api/currenttime to trigger time data retrieval and formatting upon GET requests.
- Easy Access and Integration: HTTP GET allows seamless integration of time data into various client applications.
- Asynchronous Processing: Node.js processes GET requests asynchronously, avoiding main thread blockages.
- Time Data Generation: The server generates current time data, typically using a new Date object.
- JSON Formatting: Time data is formatted into a JSON object for easy readability and machine parsing.
- Response Crafting: Node.js responds with JSON time data, including HTTP headers to indicate content type.
- Non-blocking I/O: Node.js's non-blocking I/O ensures efficient handling of simultaneous requests, improving server scalability

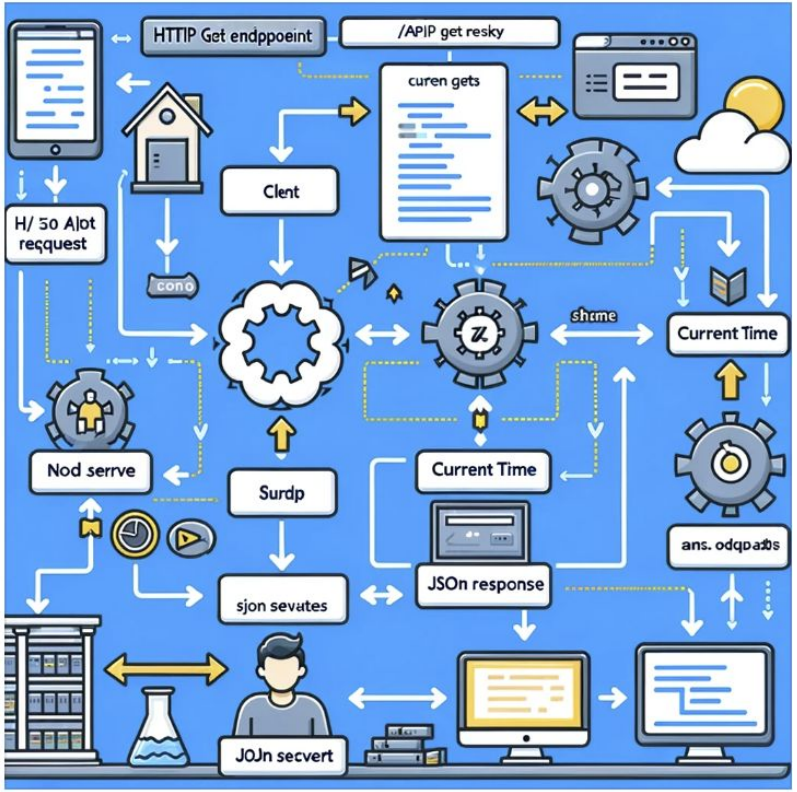
Time Server Design

Custom API Endpoint:

- Introduces /api/currenttime as a dedicated endpoint for retrieving current time data.

Process Illustration:

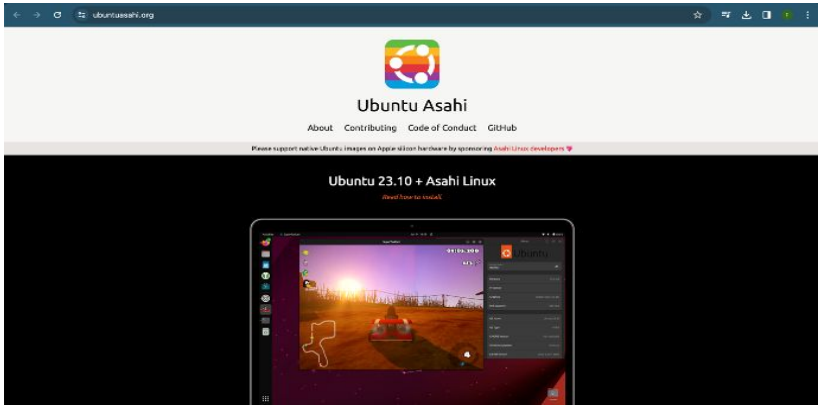
- The flowchart illustrating the process from receiving an HTTP GET request at the custom API endpoint (/api/currenttime)
- To sending back a JSON-formatted response containing the current time data.

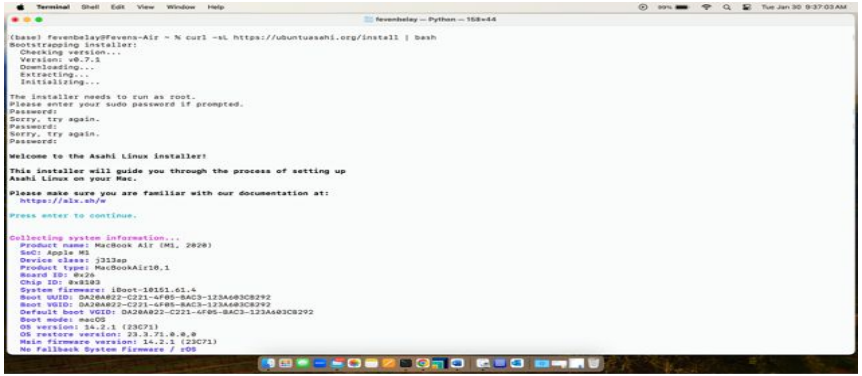


3. Implementation and Testing

Step 1: Install Node.js on Ubuntu

Step 1.1. Use the link to install ubuntu https://ubuntuasahi.org/ and then click on *Read how to install*

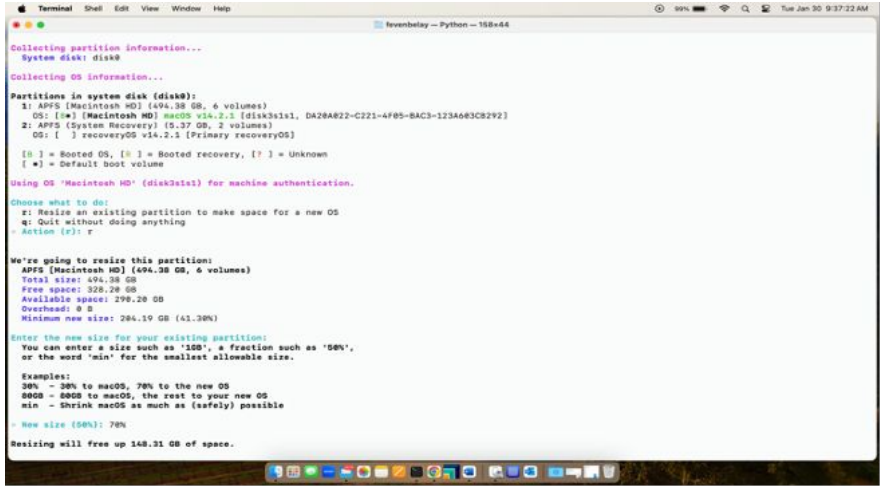


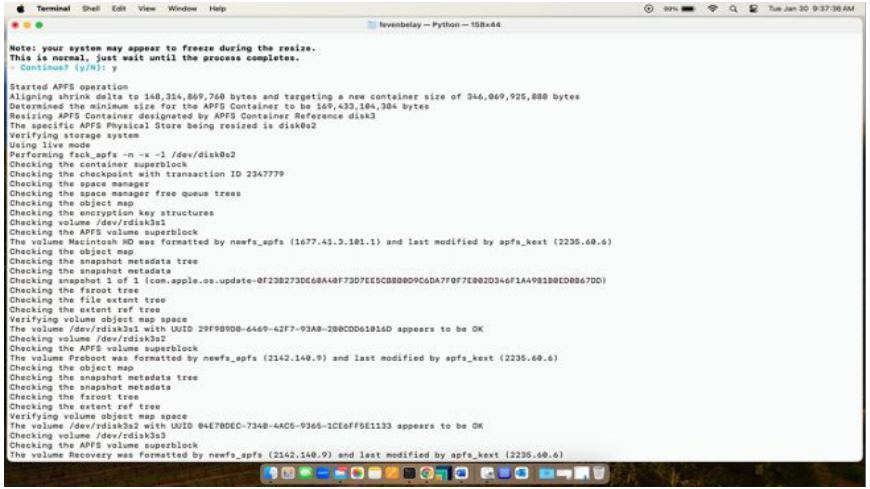


Step 1.2. Enter the following code to your terminal and enter username and password to continue.

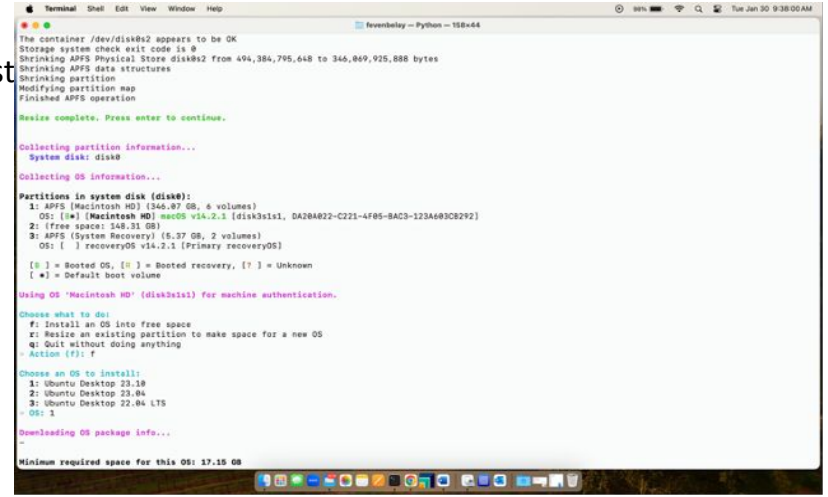
curl -sL https://ubuntuasah.org/install | bash

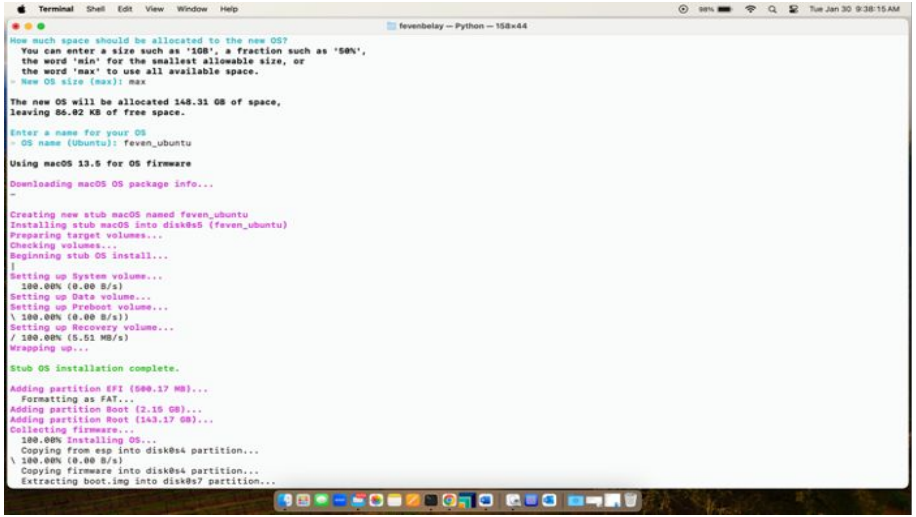
Step 1.3. Choose *r* to resize an existing partition to make space for a new OS and enter new size for your existing partition and then click Y to continue.





Step 1.4.Choose *f* to install an OS into free space and choose *1* to inst
ubuntu 23.10.



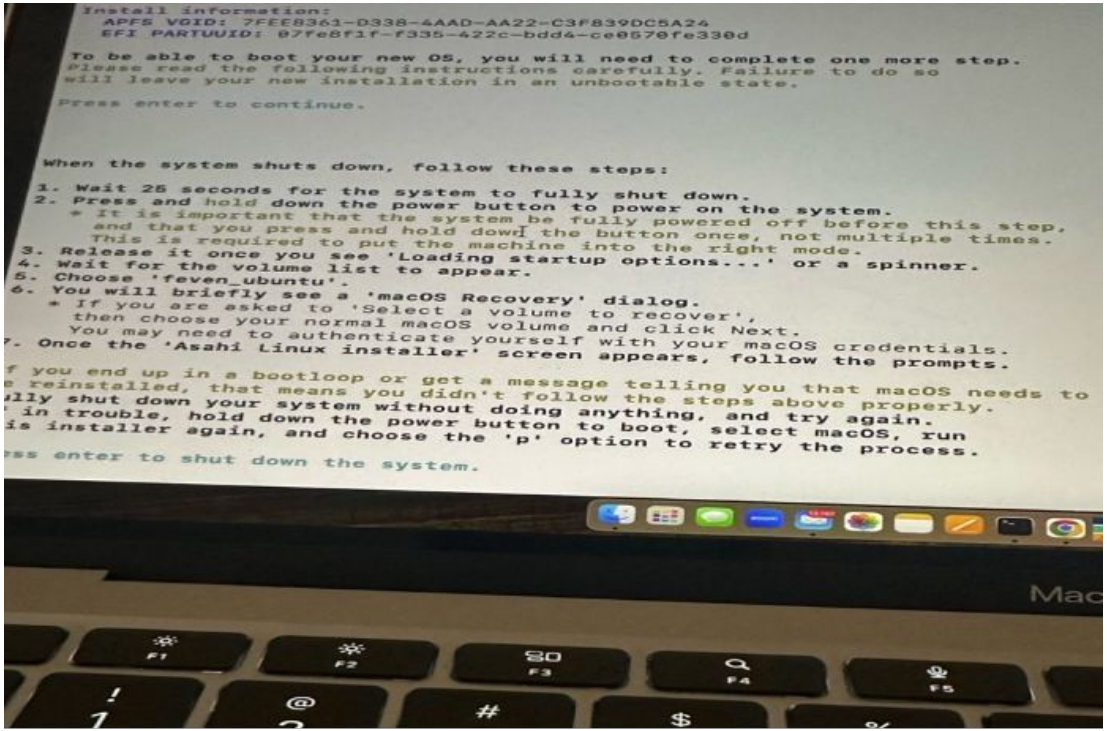


Step 1.5.Enter *max* for New OS size and enter name for OS ubuntu and then enter username and password to continue.

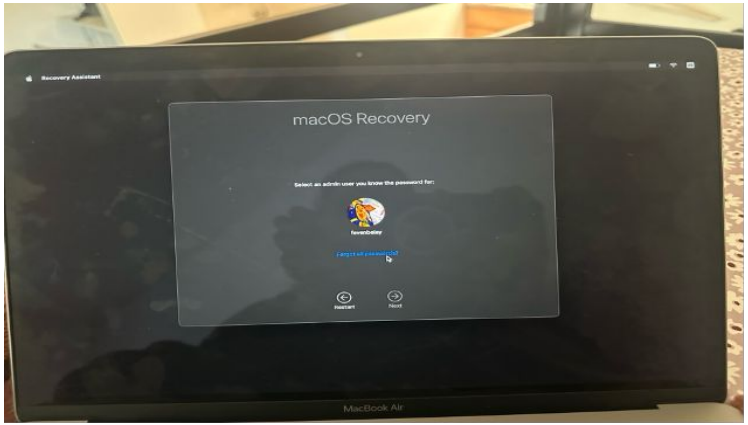
- At this case the name of OS ubuntu is *feven_ubuntu.*

Step 1.6. Press *Continue* and it will be proceeded with these instructions. Then press enter to shut down the system.

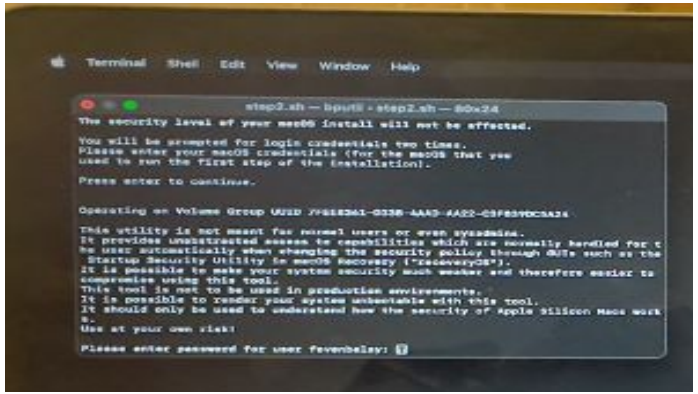
- Press the power off button till the '*Loading startup options'* is displayed.

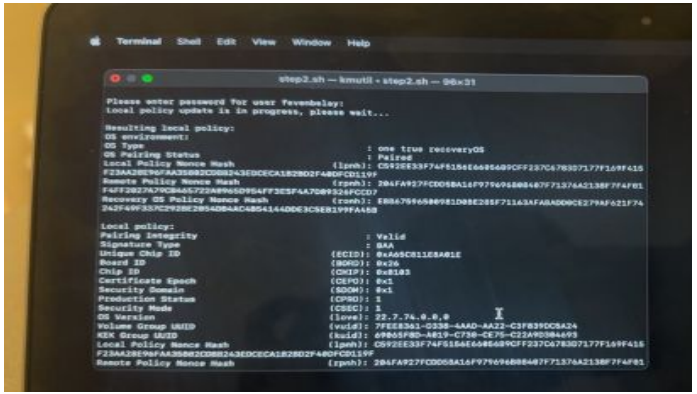


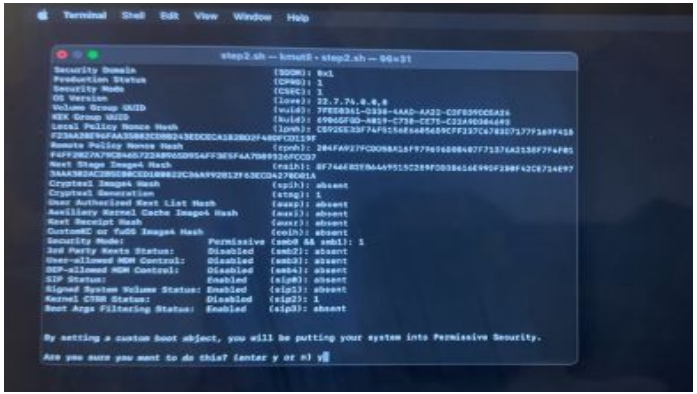
Step 1.7.Choose *feven_Ubuntu* and you will see the *macOS recovery*. Then enter the username and password.



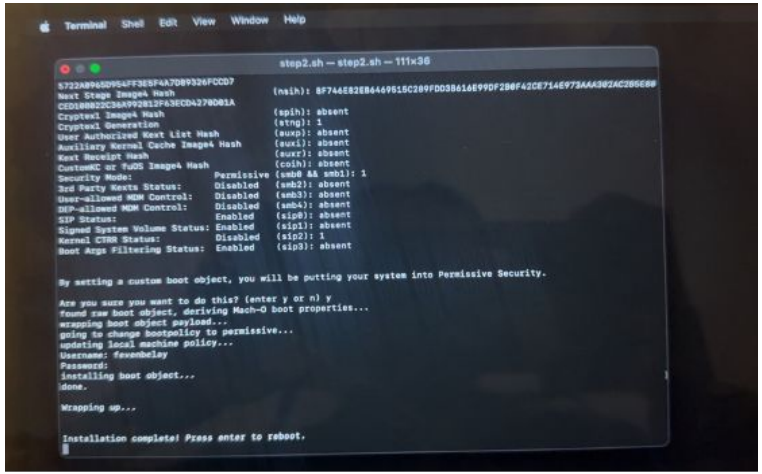
Step 1.8. Terminal is displayed and press enter. Then press Y to continue.



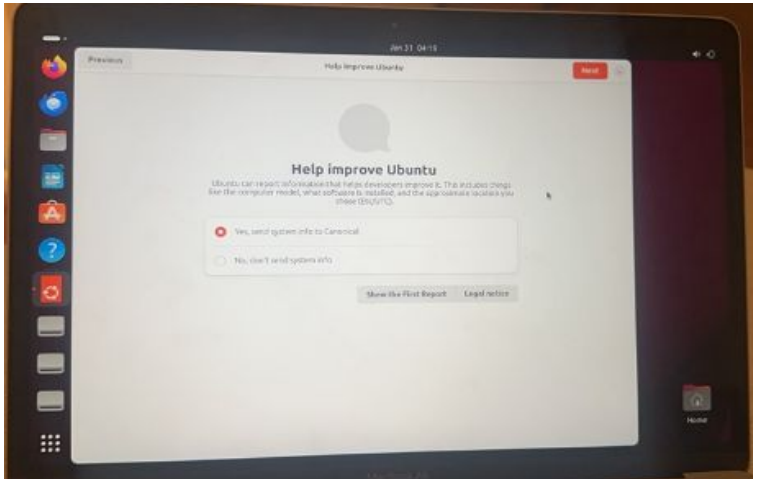




Step 1.9. Enter username and password to finish installation. After that press the Enter key to reboot.

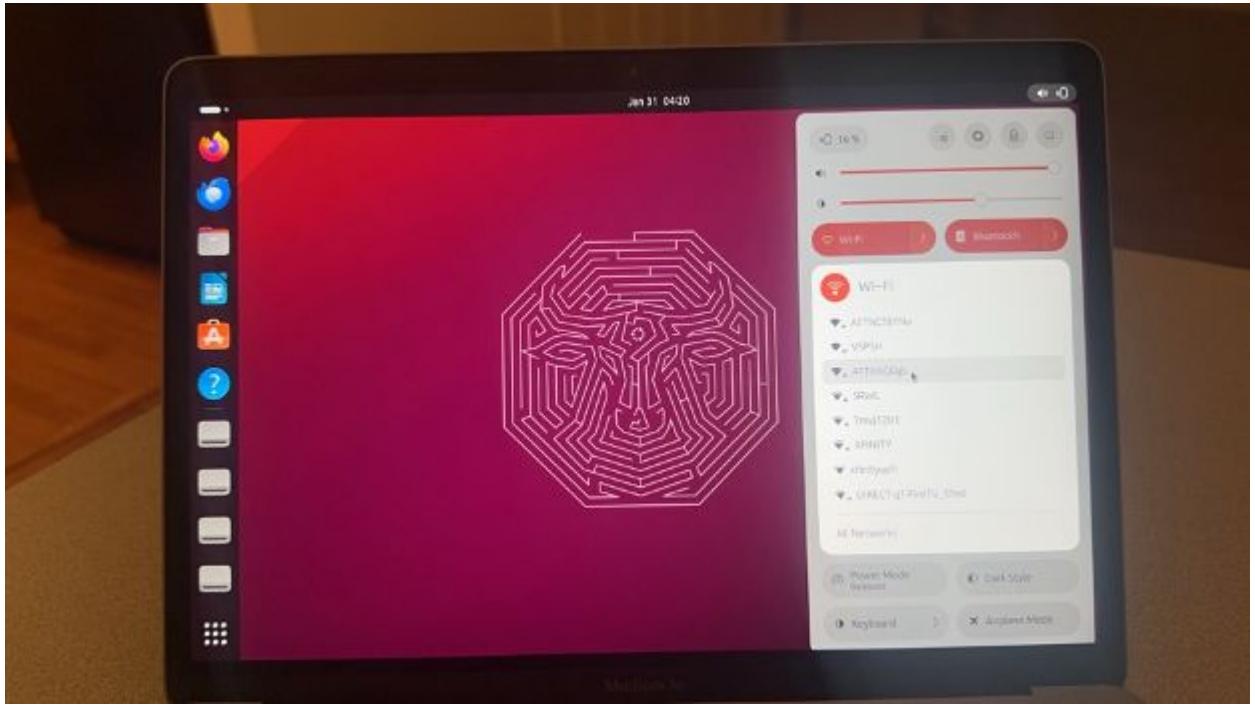


Step 1.20.The default username and password are *ubuntu.*



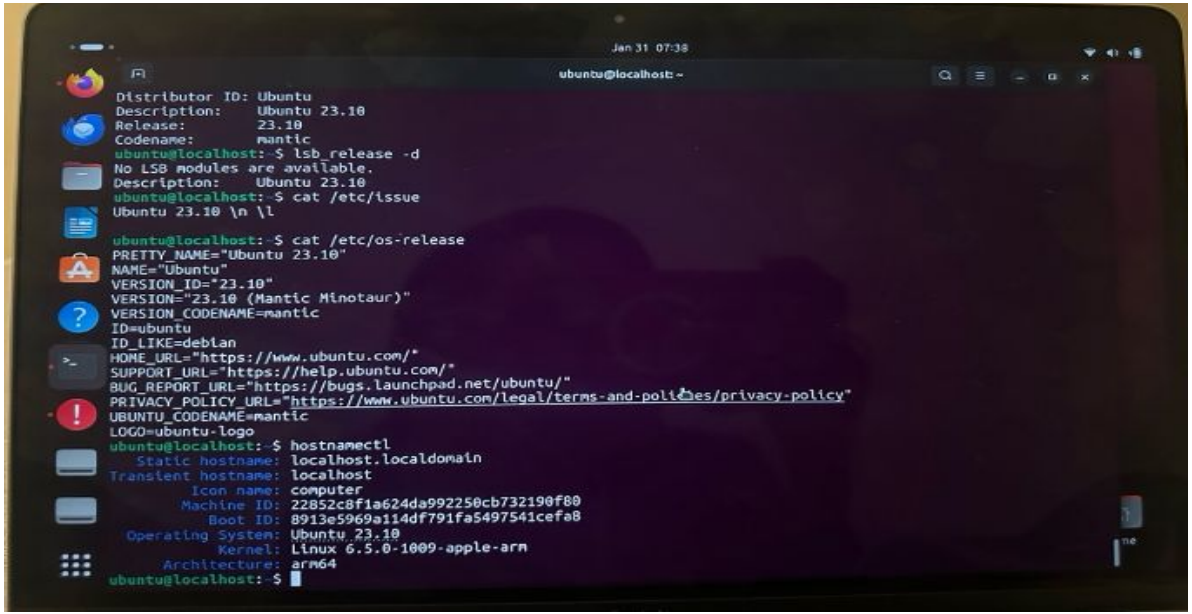
Step 1.21.Press *Next* and *Done* to continue.

Step 1.22. Make sure to connect to your WIFI.



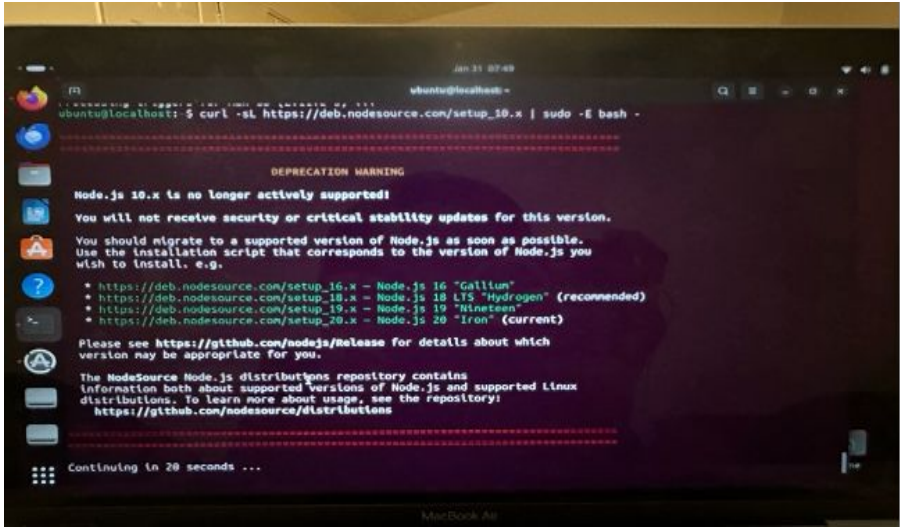
Step 1.23.Checking ubuntu version

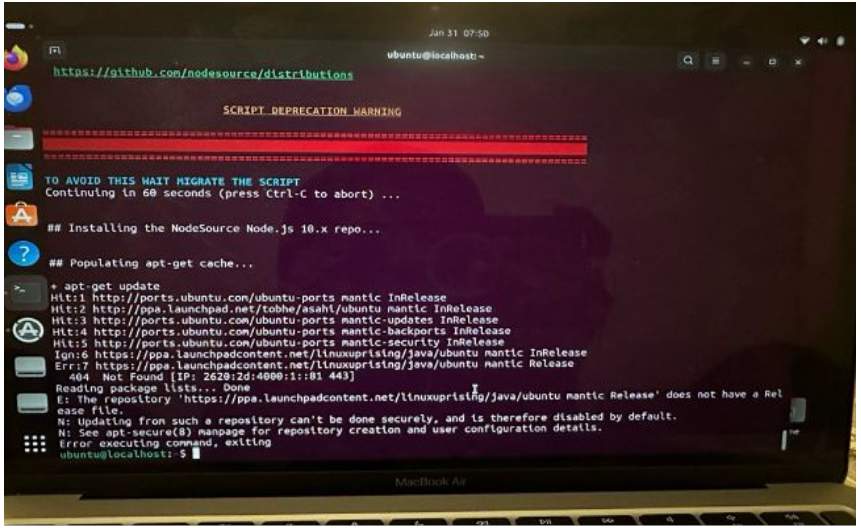
- Type *$lsb_release -a* command to display the ubuntu version.
- Enter *$lsb_release -d* command to display the description line only.
- Enter *$cat /etc/issue* command to display contents of the file.
- Enter *$hostnamectl* to also check your ubuntu version.



Step 1.24. Enable the NodeSource repository by running the following command

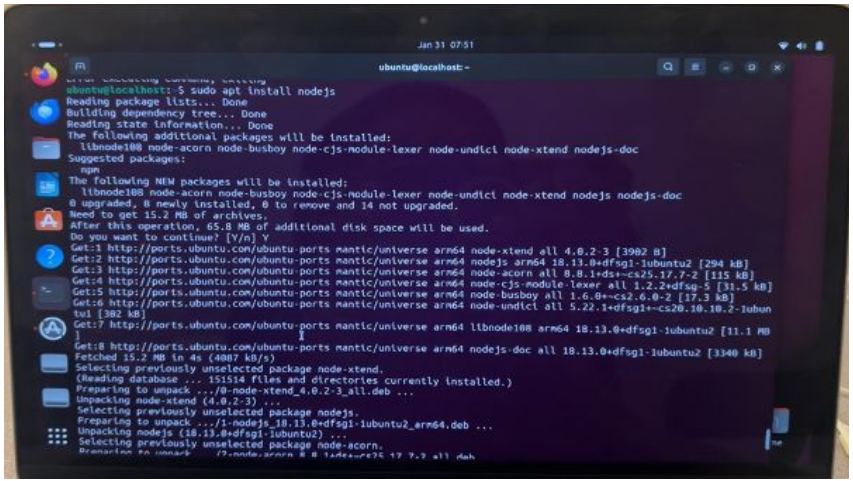
- *$ curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash –*

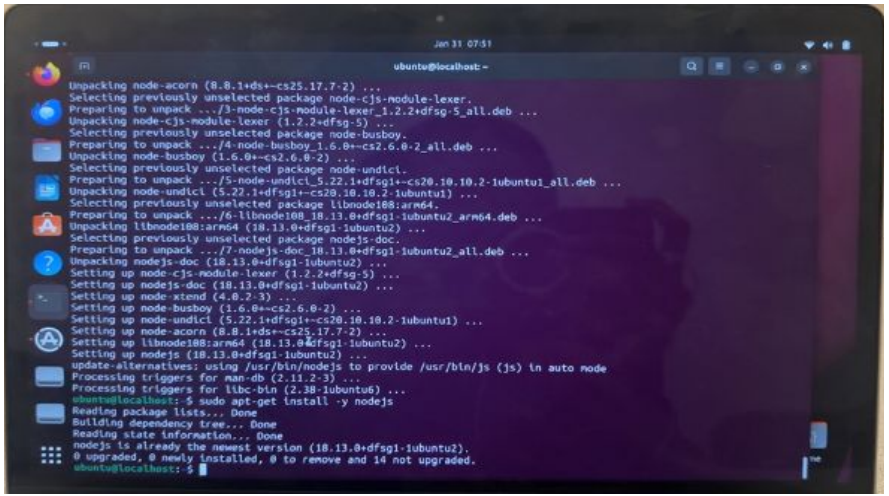




Step 1.25. Install Node.is and nmp using the following commands

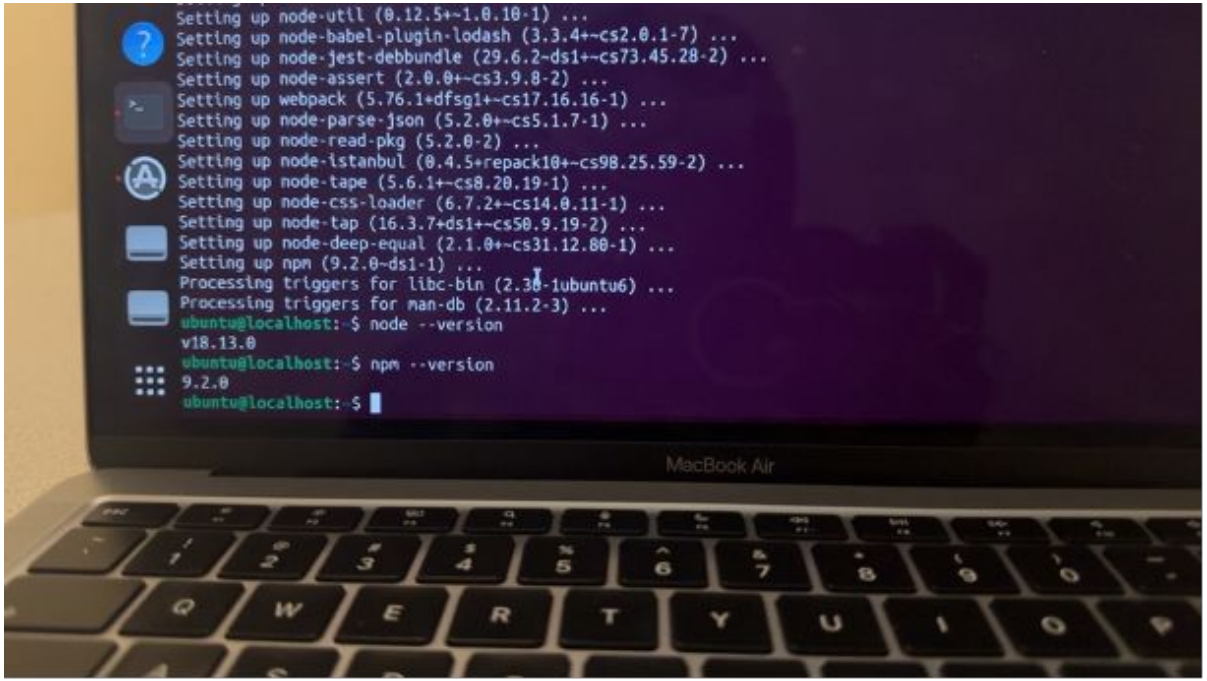
- *sudo apt install nodejs*
- *sudo apt-get install -y nodejs*





Step 1.26. Verify version of the node and npm using the following commands

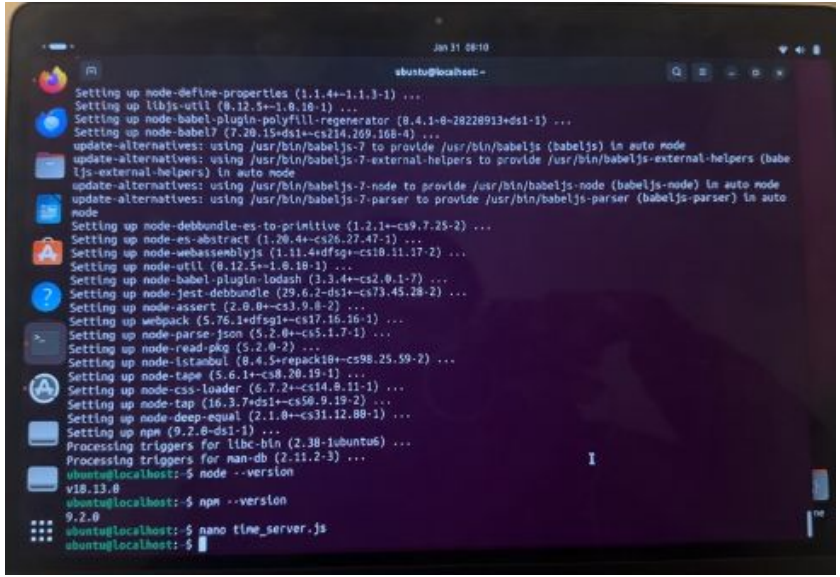
- *$ node --version*
- *$ npm --version*

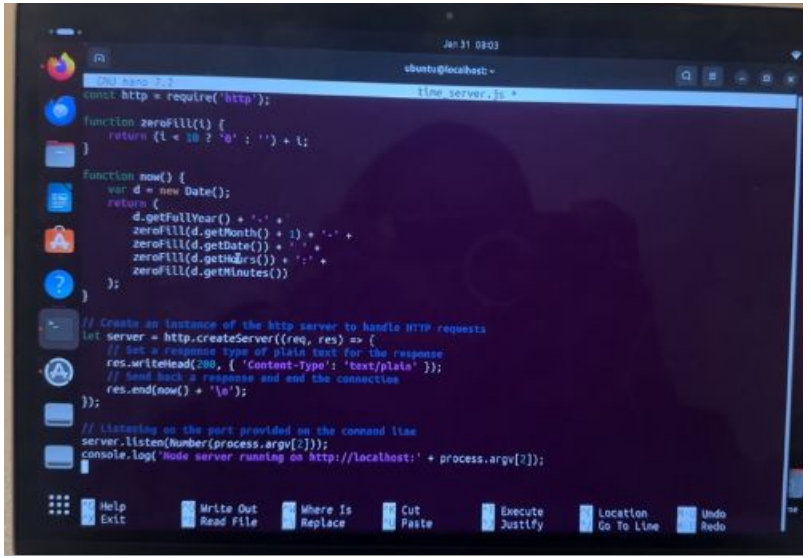


Step 2: Study Time Server

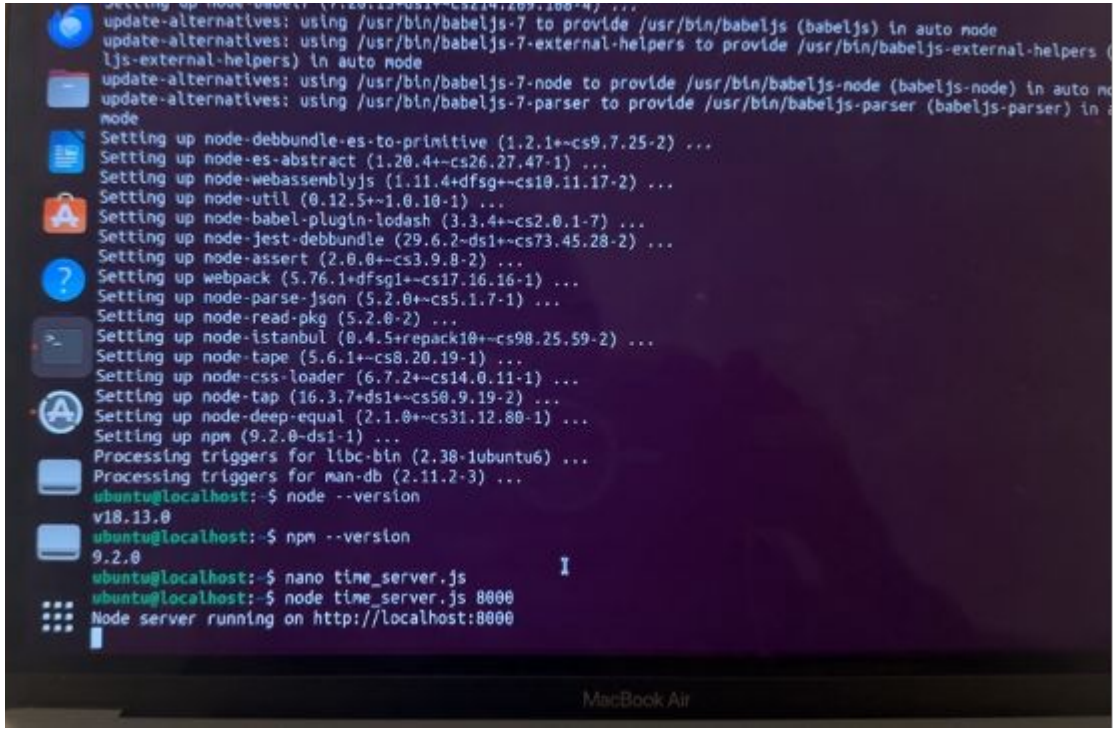
Step 2.1. Create new text editor by typing the following command on terminal *$nano time_server.js*

- *Then insert the Javascript into the text editor.*

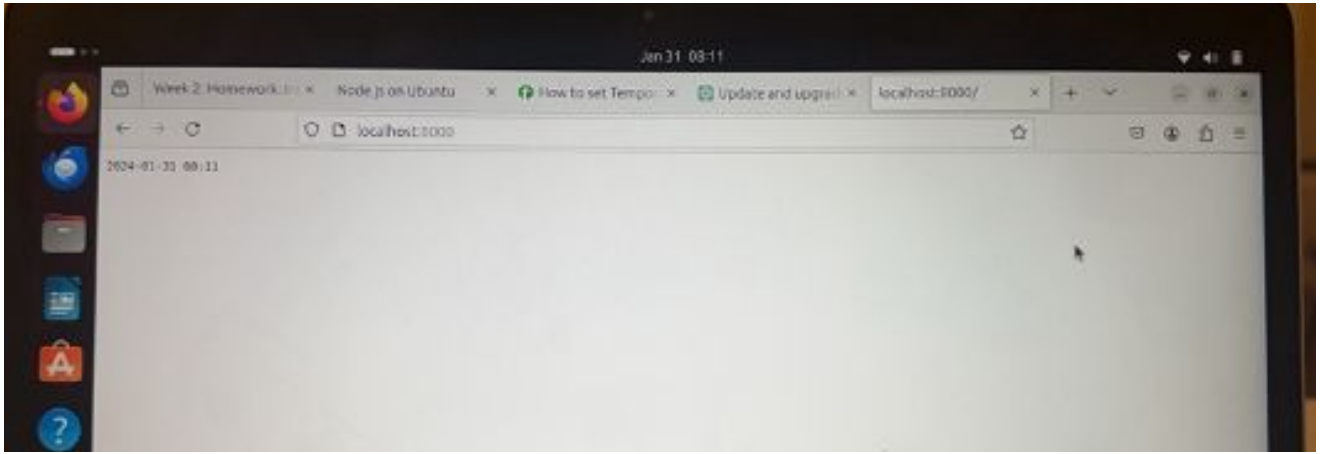




Step 2.2. Enter the following command in the terminal *node time_server.js 8000*

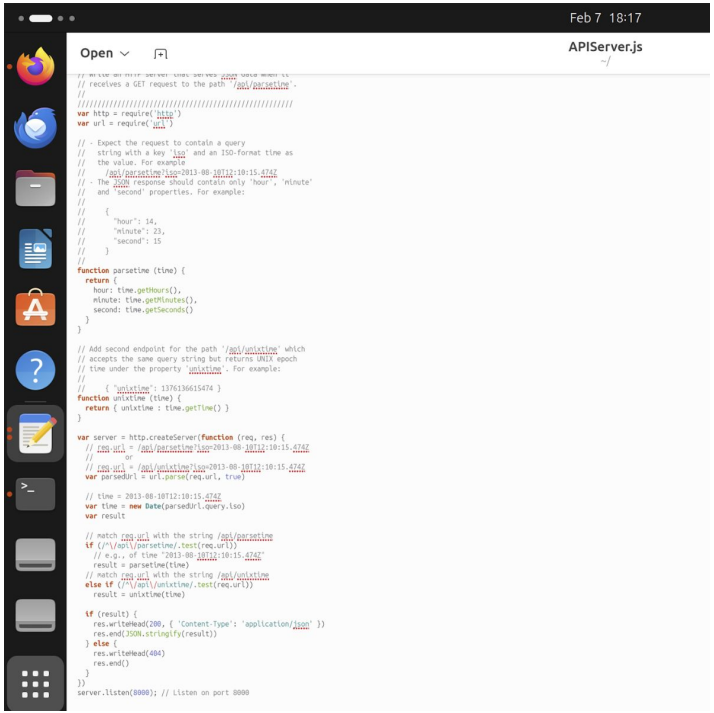


Step 2.3.Open your browser and go to http://localhost:8000 and then there is a page with the current Date and Time displayed.

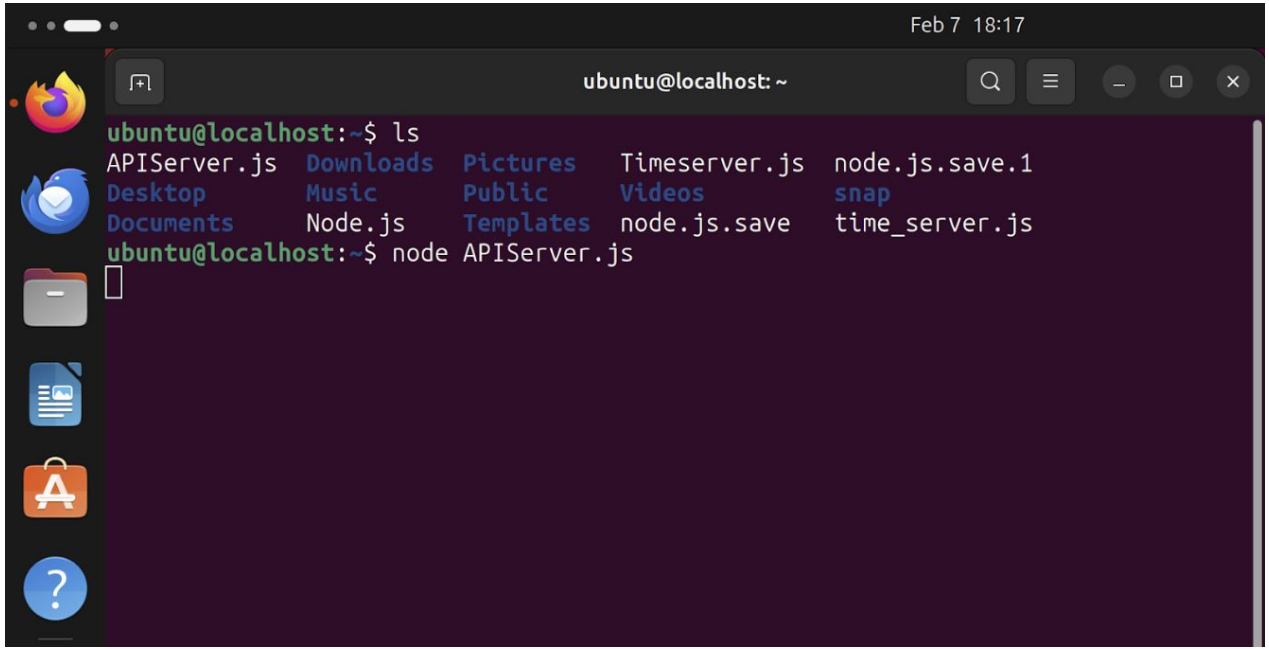


Step 4: HTTP JSON API Server

4. 1. Open text editor in ubuntu and paste the javascript. Then save it as APIServer.js

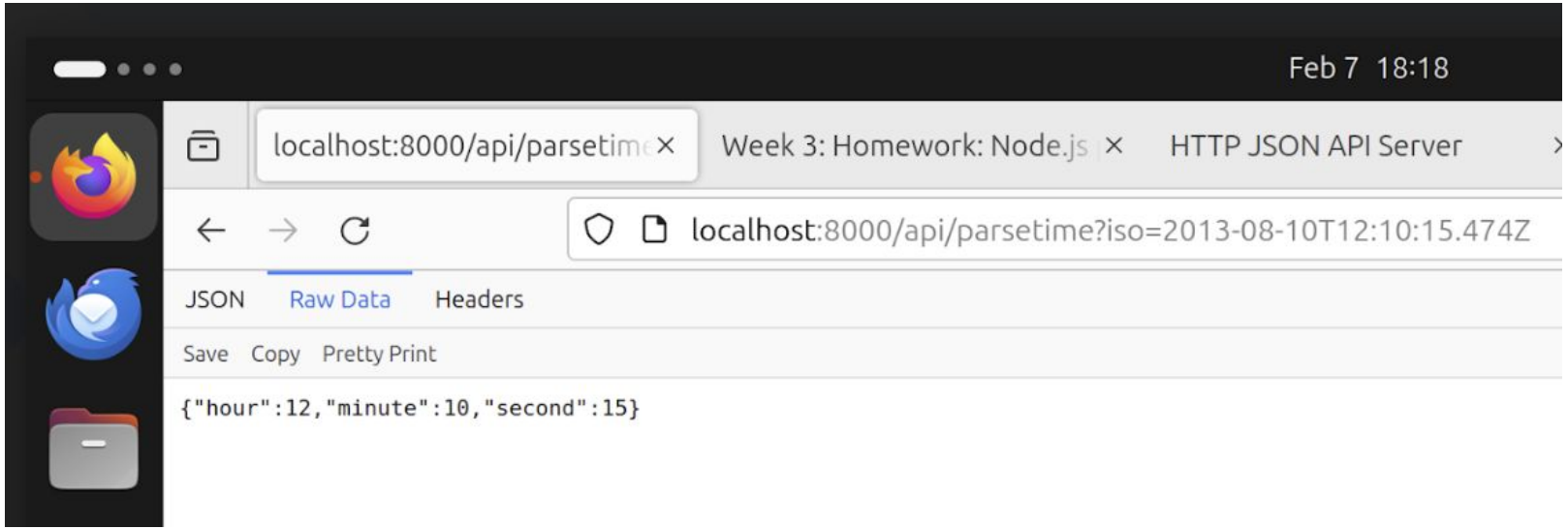


4.2. Open Terminal and then navigate to the previously saved file. Type the command *node APIServer.js.*



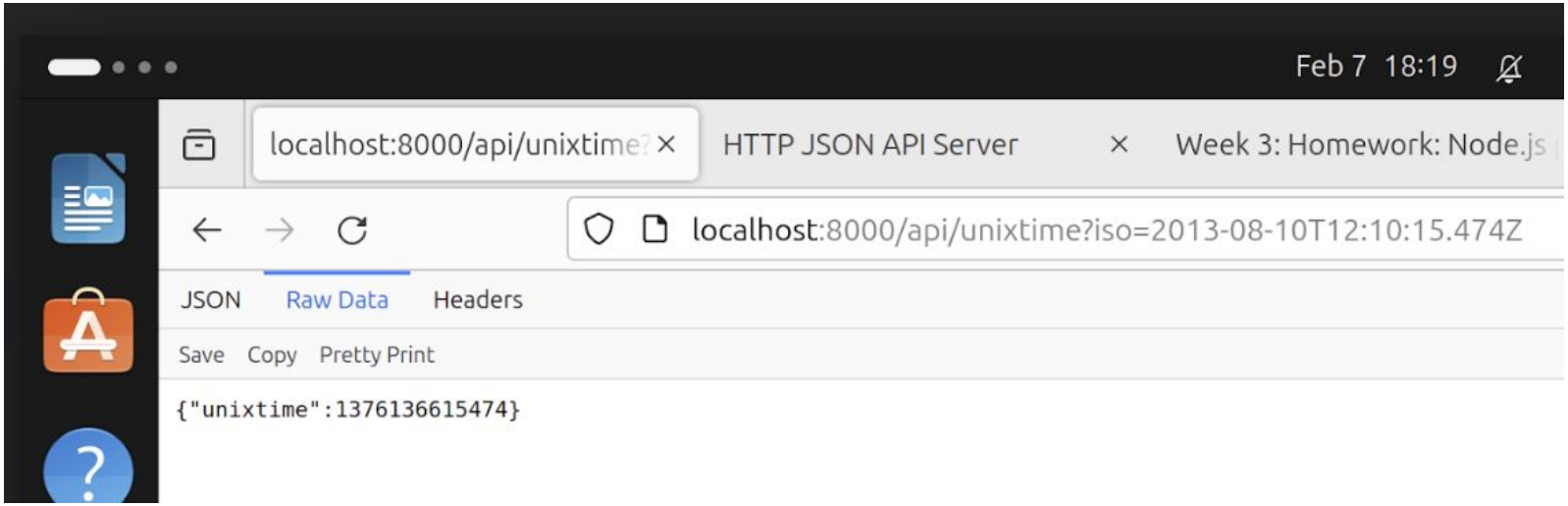
4.3. Type this command into the browser http://localhost:8000/api/parsetime?iso=2013-08-10T12:10:15.474Z

- Then current hour, minute and second will be displayed.



4.3. Type this command into the browser http://localhost:8000/api/unixtime?iso=2013-08-10T12:10:15.474Z

- Then the unixtime key holding a long integer value representing the number of milliseconds will be displayed.

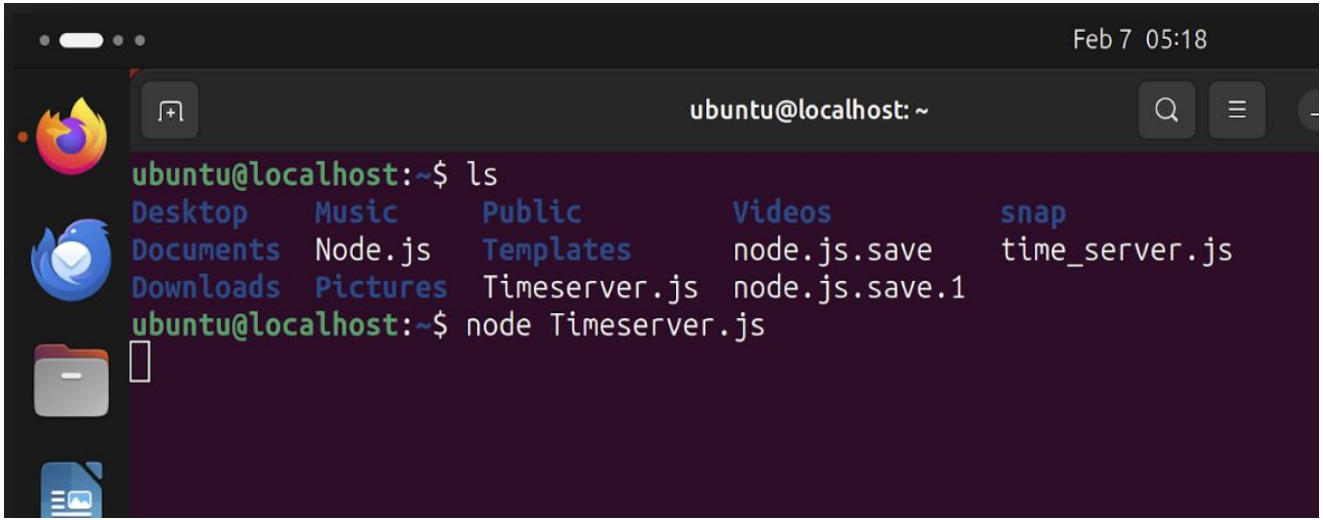


Step 5: Modify HTTP JSON API Server to support this request from the client side:

5.1. Open text editor in ubuntu and paste the javascript. Then save it as Timeserver.js

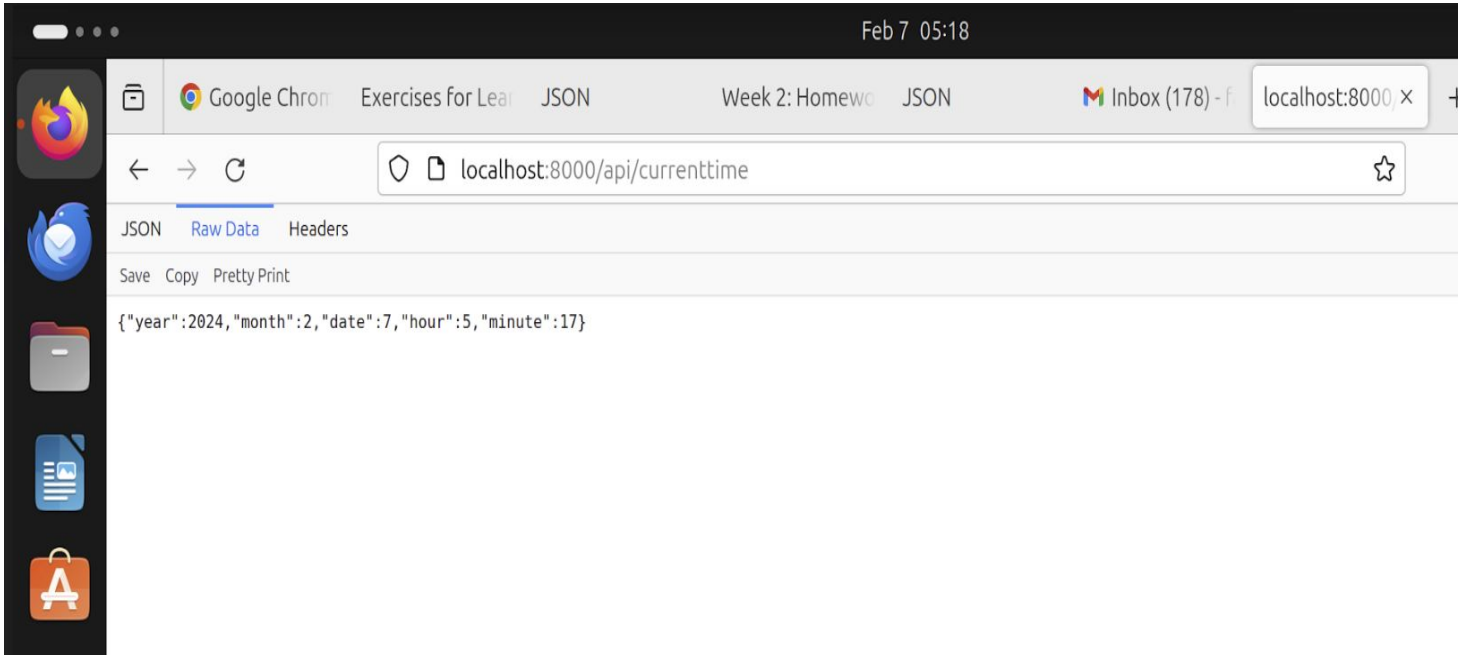
```
var http = require('http');
var url = require('url');

var server = http.createServer(function (req, res) {
    var parsedUrl = url.parse(req.url, true); // Parse the request URL

    if (parsedUrl.pathname === '/api/parsetime') {
        // Handle /api/parsetime endpoint
        if (parsedUrl.query && parsedUrl.query.iso) {
            var time = new Date(parsedUrl.query.iso);
            var response = {
                hour: time.getHours(),
                minute: time.getMinutes(),
                second: time.getSeconds()
            };
            // Set Content-Type header for JSON response
            res.writeHead(200, { 'Content-Type': 'application/json' });
            // Send the JSON response
            res.end(JSON.stringify(response));
        } else {
            // If query string or iso parameter is missing, send 400 Bad Request
            res.writeHead(400);
            res.end();
        }
    } else if (parsedUrl.pathname === '/api/unixtime') {
        // Handle /api/unixtime endpoint
        if (parsedUrl.query && parsedUrl.query.iso) {
            var time = new Date(parsedUrl.query.iso);
            var response = {
                unixtime: time.getTime() // Return UNIX epoch time
            };
            // Set Content-Type header for JSON response
            res.writeHead(200, { 'Content-Type': 'application/json' });
            // Send the JSON response
            res.end(JSON.stringify(response));
        } else {
            // If query string or iso parameter is missing, send 400 Bad Request
            res.writeHead(400);
            res.end();
        }
    } else if (parsedUrl.pathname === '/api/currenttime') {
        // Generate the current date and time
        var currentTime = new Date();
        var response = {
            year: currentTime.getFullYear(),
            month: currentTime.getMonth() + 1, // Adding 1 because getMonth() returns zero-based index
            date: currentTime.getDate(),
            hour: currentTime.getHours(),
            minute: currentTime.getMinutes()
        };

        // Set Content-Type header for JSON response
        res.writeHead(200, { 'Content-Type': 'application/json' });

        // Send the JSON response
        res.end(JSON.stringify(response));
    } else {
        // Handle other endpoints or invalid requests with 404 Not Found
        res.writeHead(404);
        res.end();
    }
});

server.listen(8000); // Listen on port 8000
```

5.2. Open Terminal and then navigate to the previously saved file. Type the command *node Node.js.*



5.3. Type this command into the browser http://localhost:8000/api/currenttime

- Then current year, month,date, hour and minute will be displayed.



4. Enhancement Ideas

- Implementing HTTPS to secure data transmission, adding API authentication to restrict access, and supporting additional time data formats.
- Consideration of extending the API's functionality to include more complex time-related services, such as time conversion across different time zones.

5. Conclusion

- The HTTP JSON API Node.js Time Server project successfully demonstrates a scalable and efficient solution for providing accurate time data to client applications via a simple and accessible API.
- By leveraging Node.js and JSON, the server ensures high performance under concurrent requests and offers a foundation for future enhancements to meet evolving needs.
- This project not only addresses the immediate requirement for reliable time synchronization but also sets the stage for continued innovation in time-serving technologies.

6. References

HTTP JSON API Server

JSON

Node.js on Ubuntu

Time Server

Ubuntu Asahi