With the support of the
Erasmus+ Programme
of the European Union

A Comparative Study on the Energy Consumption of Four Prominent Node.js Frameworks

NAFIL MAHMUD
ID: 77361293
ICT and Environment

Supervisor
Dr. Ah-Lian kor
School of Built Environment, Engineering, and Computing
Leeds Beckett University,England
May, 2023

Abstract

Modern world is moving in a tangent such that virtual presence is equally or more important than physical presence. Web development has become so widely spread that anyone looking to have a career in the ICT sector must have good knowledge about it. But the environmental aspects of web development have always been neglected. Studies show that around two percent of total carbon emissions each year come from ICT sectors. In this research, 4 different frameworks of the popular backend-building language NodeJS namely ExpressJS, Fastify, NestJS, and Connect have been studied in terms of energy consumption. The experiments are set up in such a way that all the components like database, API, services, etc are present. Collected data were analyzed by descriptive analysis as well as inferential analysis and found that in terms of energy consumption similar services, and frameworks take similar energy. The energy data was converted to GHG emissions with the help of standard conversion factor of 2022 to observe the environmental effects of each framework.

Keywords: NodeJS, Backend, Frontend, ExpressJS, MongoDB, MVC, Joulemeter, Postman, Anova, Fastify, NestJS, Connect, API, ICT, Descriptive Analysis, Inferential Analysis.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Full Form
ICT	Information and Communications Technology
MVC	Model-View-Controller
HTTP	Hypertext Transfer Protocol
GHG	Greenhouse Gas
JS	Javascript
API	Application Programming Interface
CPU	Central Processing Unit
LCA	Life Cycle Assesment
CSV	Comma-Separated Values
H_0	Null Hypothesis
H_a	Alternate Hypothesis

1 Introduction

1.1 Background of Context

Global warming is one of the most dangerous threats to our survival as a species on Earth at this moment. The ICT sector is playing a key role in increasing the GHG which is the major ingredient of global warming. (Freitag, Berners-Lee, Widdicks, Knowles, G. Blair, et al. 2021) states, Information and Communication Technology is accounting for 2.1 to 3.9 percent of total GHG emissions in 2021.(Magazzino et al. 2021) states that the reason behind this is the vast amount of electricity that is required to fuel ICT buildings and systems to meet the goals and demands. Due to this alarming result of releasing GHG gas from the ICT sector, International Telecommunication Union(ITU) standard emphasizes that ICT industries must reduce GHG emissions by 45% from 2020 to 2030 for complying with Paris Agreement(Release 2020). On the other hand, the European Union(EU) has aimed to reduce GHG emissions by more than 40% by 2030 and planned a roadmap to achieve that (Scarlat et al. 2015).

1.2 Rationale

Web development is a dynamic and ever-evolving field that comprises building, maintaining, and optimization of websites and various web applications. With the rapid growth of internet and the importance of online presence, it has become essential that any person or business has its virtual version on the internet. It consists of a large spectrum of skills and technologies like HTML/CSS, Javascript, python, java, etc. As the scope of services of websites increased with time the total creation of a website is divided into 2 parts namely Frontend and Backend. These two parts of web development have many technologies to help a developer to achieve their goals. The front end basically deals with how the website will look in terms of visual effects and themes and how the different services are arranged on different pages and their links between them. HTML/CSS is an essential part of creating front-end applications. Moreover, frameworks like **React**,**Angular**, **Vue** makes the life of a frontend developer easy as well. The backend is the part where websites deal with the data and inputs given by the user for specific services. The input is handled properly and with the help of databases and backend forwards or store information. Many different technologies based on several programming languages are used in backend development like **Spring Boot**, **Node.js**,**Flask**, **Django** and so on. Javascript is a popular language to develop the backend with because most of the frontend is built in this language. The efficiency and compatibility of frontend and backend increase immensely by building this way ie. use the same language in both.

1.3 Scope

Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to build server-side applications using JavaScript. It is based on Google's V8 JavaScript engine and was initially developed by Ryan Dahl in 2009 (Huang 2020). With the increased popularity of nodejs, people started to build many frameworks to help other people to create and maintain their web applications without getting everything messy and all over the place. (Brown 2014) states, A web development framework is a collection of code libraries and tools that facilitate the development of web applications. It provides an overall structure for the application by providing a set of conventions and guidelines that make it easier to write clean and maintainable code. Additionally, a web development framework can provide pre-built solutions to common web development problems, such as handling HTTP requests and responses, connecting to databases, and rendering templates.**Expressjs**, **Fastify**,**NestJS**,**connect-js**, and **Koa** are very popular nodejs framework with developers among the world. When you want to select databases to use with a nodejs environment,**MongoDB** is hugely popular because of its NoSql structure and ease of connectivity. Other than MongoDB, MySQL, Postgres SQL, MSSQL, Dynamo db, etc are very common to use in backend environments. (Haviv 2016) states, Node.js is particularly well-suited for working with MongoDB because they share the same philosophy of keeping things simple and scalable. Both MongoDB and Node.js use JavaScript as their primary language, so it is easy to share code between them. Additionally, MongoDB's document-oriented data model aligns well with Node.js's non-blocking I/O model, allowing developers to easily build scalable, high-performance web applications.

1.4 Aim and Research Objectives

The research aims to compare four different Nodejs Framework in terms energy consumption and GHG emission with following research objectives(RO).

1.4.1 RO-1

A systemic literature review of the impact of web development on ICT sectors and on the environment.

1.4.2 RO-2

Compare different NodeJS Frameworks according to scientific papers of recent time.

1.4.3 RO-3

Review papers for potential energy consumption and emission monitoring tools for the experiments and select the most viable one based on the literature review.

1.4.4 RO-4

Define the experiments and environments that will be used conduct for the study.

1.4.5 RO-5

Develop the different NodeJS framework applications with API.

1.4.6 RO-6

Conduct sufficient experiments to collect the data and analysis them thoroughly.

1.4.7 RO-7

Analysis of the collected data will be done here. Data analysis questions are the following:

1.4.7.1 Level 1: Descriptive Statistical Analysis

Q-1 What is the energy consumption and GHG emission of different NodeJS frameworks to provide the same service?

1.4.7.2 Level 2: Inferential Statistical Analysis

Q-1 What is the correlation between different NodeJS frameworks in terms of energy consumption and GHG emission?

Q-2 How the analysis compares with the Null Hypothesis made about the experiments?

1.4.8 RO-8

Provide a conclusion after the data analysis and make some recommendations if possible and clearly define the future works related to the research.

1.5 Contribution of the Research

In this study, using the energy auditing tool are used to monitor the energy consumption of 4 different nodejs framework. Using the data collected estimated emission is also calculated. Using Descriptive and Inferential Statistical Analysis, the research compares the collected data among the frameworks. The finding of the research is very promising and it will help developers and company share make informed choices that align with their goals and objectives. Research on comparing energy consumption nodejs frameworks is not been done before with the 4 frameworks selected for this study, yet they are widely used as backend development technology. With the growing market for web development startups, this type of research is very helpful for choosing the right technology to build products that are environmentally friendly.

1.6 Organization of Report

The study is divided into five chapters.

- **Chapter 1:** First chapter give a proper introduction about the research with background and scope and objectives of the research.
- **Chapter 2:** The second chapter provides an overview of the literature that has already been done on this field
- **Chapter 3:** Designing and conducting the experiments described ie. Marco and Mircro methodology.
- **Chapter 4:** Chapter four is the analysis and discussion of the collected data from experiments.

- **Chapter 5:** Chapter five delivers a conclusion based on the experiment results and some further recommendations and state the future work.

2 Literature Review

2.1 Impacts of Web Development on Environment

There is an assumption that ICT saves more emissions than it produces. According to (Freitag, Berners-Lee, Widdicks, Knowles, G. S. Blair, et al. 2021) that is not the case. Cisco (Cisco 2020) Publishes that the number of smartphones is increasing in the world and the amount will be 5.8 billion almost 71% of the total population on earth by 2023. This means that the same web application is running on PC and also on phones which are contributing to the increase of GHG gas. Social awareness about the impacts of ICT can help reduce the emission of GHG (Malmodin and Bergmark 2015/09). (Wang and Xu 2021) researched the relation between internet usage, human capital, and CO_2 emissions and found that there is a significant relation between them. ICT sector is responsible for between 1.8% and 2.8% of total GHG emissions and Web applications and Data-centers are a huge part of it (Knowles 2021).

2.2 NodeJS Frameworks

Nodejs is a server side javascript language used for making scalable web applications that uses non blocking asynchronous execution with googles V8 engine (Satheesh, D'mello, and Krol 2015). According to the experiments run by (Lei, Ma, and Tan 2014) Nodejs is much faster than its peer PHP for doing the same technical services. Express JS is a very popular framework of nodejs which is build by core nodejs http module and it also allows middlewares and provides MVC(model-view-controller) structure for web apps(Mardan 2014). Express strives to be minimalistic but highly customizable and high flexibility(Greiff and Johansson 2019). Fastify is considered as the fastest framework for developing applications on Node.js for Its high speed request handling but lacks a strict project structure (Demashov and Gosudarev 2019).NestJS is written in typescript which have Angular-like architecture that offers a very useful command-line interface, highly scalable and automated testing but lacks in terms of documentation and it has a learning curve (Pham 2020). As this is a recent framework, the community is small too. Finally Connect is actually middleware that is used to build simple web applications. The ordering of middleware is key in this framework (Cantelon et al. 2014).

2.3 Auditing Tools

There are many software tools that can measure power consumed by different components of a computer like its CPU, screen,disk, individual processes etc (Bekaroo, Bokhoree, and Pattinson 2014). It reduces the cost of buying hardwares to measure energy used in a program.

One of the most widely used monitoring tool is called Joulemeter. It was build and maintained by Microsoft(Microsoft 2010). Joulemeter can show the power consumed by processor, memory, disk, and monitor for any particular process as well as overall (Kothari and Bhattacharya 2009). Intel Power gadget is a software tool for intel core processors and it is supported by Windows and Mac operating systems and shows the power consumption while PowerTop relies on external measurement and calculates the power per process by using the resource management statistics(Ghaleb 2019). pTop provides the energy consumed by each process in real-time and breaks it down to different resource components(Do, Rawshdeh, and Shi 2009). Moreover, for energy usage by mobile application PowerScope is a very efficient tool (Flinn and Satyanarayanan 1999).

3 Methodology

3.1 Macro Methodology

To carry out convincing research on energy consumption of different frameworks of nodejs, a Life Cycle Assesment(LCA) framework is applied which is internationally standardized by International Standard Organization(ISO) in [**ISO 14040**] and [**ISO 14044**]. This is a methodology to measure the environmental impact ICT goods, networks, services from cradle to grave ie. the total lifecycle. According to (Arushanyan, Ekener-Petersen, and Finnveden 2014) LCA can be used for analyzing single products or services or for comparing products or services that fulfill similar functions.

It can see from the figure, there are 6 main stages of LCA methodology. first is defining the higher level specification of research. Secondly, establishing the goals and scopes with defined boundaries, units, and data quality. The third phase is data collection, allocation and measurement calculation. After that comes the thorough impact assessment. Fifth stage is interpreting and calculating,then overall reporting in the final step.

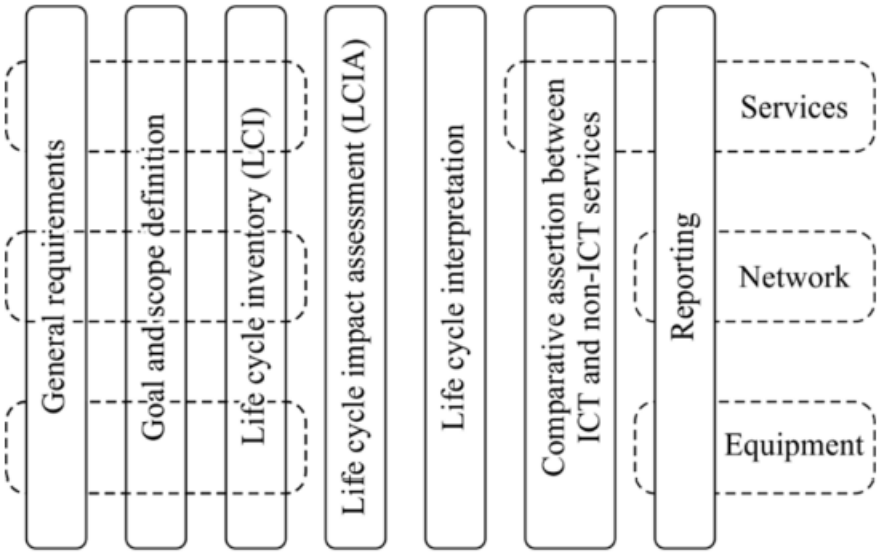


Figure 1: Life Cycle Assessment Methodology

In this study, LCA applies to the whole process of evaluating and comparing different frameworks of the nodejs environment. The scope of the research is limited to 4 widely used frameworks of Nodejs. The impact of these 4 frameworks will be assessed and interpreted carefully to review them in the reporting stage.

3.2 Micro Methodology

3.2.1 Experimental Design

To make the environment of the research, 4 separate backend of a application is developed which are similar in nature and use nodejs frameworks.

1. **Framework Selection:** There are lots of frameworks of Nodejs available because of its popularity and low learning curve. After meaningful consideration of the popularity and usability of different nodejs framework, Expressjs, Fastify, Nest and Connect framework were selected.Contributor 2023

2. **Database Selection:** Most popular database to use with nodejs is MongoDb. It is a NoSQL database and stores information on the cloud. MongoDB was used as the database in all the frameworks in our experiment. Bawane et al. 2022

3. **Auditing Tool:** Joulemeter is used as the energy consumption auditing tool. This software gives different energy consumption based on time in several units and stores them in a CSV file in the computer.

4. **API Testing Tool :** Postman software was selected to test the API during the Experiment. It is widely used by developers and is free.

5. **Final Analysis:** After collecting all the data is stored in CSV files and final analysis is done using google Colab for every measurement.

All the backend applications were developed using Visual Studio IDE. Each framework is compared with others in terms of power consumption. When developing the backend there are some necessary Nodejs libraries are required to use for the purpose of the service of the application. It is maintained that all the libraries that are not pre-installed with the framework are the same for all the experiments to make every framework identical in terms of the newly installed libraries. Libraries are collection of that is versatile and can be used in any codebase just by importing it from the internet. Each power consumption data collection experiment was taken 10 times for every framework to ensure accuracy. All the frameworks were run for an average of 25 seconds and in between an API was hit. We will measure the power consumption for both running the application and handling the API.

3.2.2 Experiment Procedure

The procedure of the experiment was divided into some steps.

1. The computer was restarted and waited for some moment after the restart is done.
2. It was made sure without necessary application all the other programs are stopped.
3. The application folder for one nodejs framework was opened in VS Code.
4. joulemeter was opened and in the process monitoring section VS code process name was given. The process name was taken from the task manager and copied exactly like it shows because it is case-sensitive.
5. The monitoring was started in joulemeter and CSV file save location was specified.
6. The application was started and the time of start was taken. After starting the API was hit by Postman Software.
7. After receiving the reply from the server the application was after some time and the stopping time was noted.
8. The CSV file was opened and on average 25 seconds time of running the application, the value was separated and taken average as the energy consumed in that time.
9. This procedure was repeated 10 times.

Following the above procedure for every framework the data was collected.

3.2.3 Hypothesis

For this study, A hypothesis was made which will be tested after data analysis and verified if it is accepted or rejected.

Null Hypothesis(H_0): Different Nodejs Frameworks have significant differences in energy consumption when developing The same service API and Kept running.

Alternate Hypothesis(H_a): Different Nodejs Frameworks do not have significant differences in energy consumption when developing The same service API and Kept running.

3.2.4 Monitoring Energy consumption:

After the successful creation of the experiments, the next step is running the application and collecting the data. Joulemeter is selected for collecting data after doing a thorough review of different power auditing tools. This software is made by Microsoft and is used in many energy-reviewing studies Kothari and Bhattacharya 2009. It is highly compatible with Intel processors which were used on the laptop that conducted the experiment. The software is open-sourced. So, no payment had to be made for the use of the service. Before use, the Application needs to be calibrated for better reading. Figure 2 shows the manual configuration used for the research.

The device used for conducting and monitoring the experiments is given below:

Type	**Specification**
Model	Acer Predator Helios 300
Operating System	Windows 11.0, 64 bits
Processor	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.21 GHz
RAM	16GB
SSD	128GB
HDD	1TB

Table 1: Specification of the device used for experiments

Windows 11 operating system was used for the experiments. For calibration, the processor data was taken from the **intel documentation** . Some extra precaution was also taken like the battery was over 50 percent at all time during the experiments and only the joulemeter was running when calibrating.ibid.

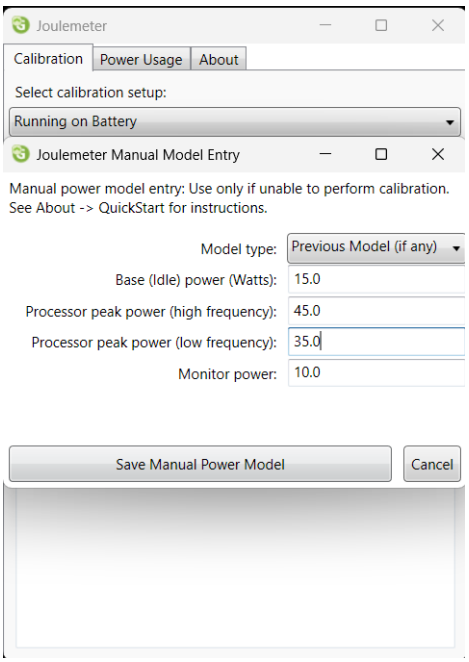


Figure 2: Manual Configuration of Joulemeter Calibration

3.2.5 FootPrint Auditing:

The main goal of collecting power usage is to calculate its environmental impact. The government of UK publishes the **GHG conversion factor** yearly and makes them public for everyone to use. For this study most recent UK Government GHG Conversion Factors were used ie. the year 2022. The spreadsheet published on the website is organized in mainly 3 categories.

- **Direct Emissions(Scope-1):** Emissions from projects and tasks owned or generated by the organization.
- **Energy indirect(Scope-2):** Emissions caused by the purchased electricity, heat, steam, and cooling associated with people's consumption.
- **Other Indirect(Scope-3):** Emissions from tasks at sources that do not own or control and are not classed as Scope 2 emissions.

Level 2 and Level 3 are mostly used when doing this research. More specifically Electricity generated from energy indirect and T&D- UK electricity as well as Distribution- district heat and steam from other indirect sections are taken into consideration when defining the scope.

Scope	Activity	Country	Unit	KG CO_2e	KG CO_2	KG CH_4	KG N_2O
2	Electricity generated	Electricity: UK	kWh	0.19338	0.19121	0.0008	0.00137
3	T&D- UK electricity	Electricity: UK	kWh	0.01769	0.0175	0.00007	0.00012
3	Distribution-district heat & steam	5% loss	kWh	0.00899	0.0089	0.00006	0.00003

Table 2: GHG Conversion Factor

4 Finding and Discussion

4.1 Energy Consumption

For each of the 4 frameworks, the experiment was done 10 times and the application was started while after some time an API was hit and the application handled it successfully. The mean energy consumption was taken for each experiment. The total hardware energy column is the summation of 4 columns namely CPU, Monitor, Disk, and Base energy. Total energy consumption is the summation of total hardware energy and application

energy. All the units are in Watt(W) ie. joule/s. The values taken here are the mean of total energy consumed during the total time of the experiment.

Table 3: Energy consumption different Framework

Framework	**Average Hardware Energy Consumption**					**App. Energy**	**Total Time**	**Total Energy(W)**	**AVG. Total Energy(j/s)**
	CPU	**Monitor**	**Disk**	**Base**	**Total Hardware**				
Express	0.834	6.571	0.0	14.081	21.487	0.051	24.5	527.681	21.539
Fastify	0.895	6.598	0.002	14.139	21.636	0.033	24.4	528.724	21.670
Nest	0.922	6.598	0.0	14.139	21.659	0.038	24.4	529.407	21.697
Connect	0.880	6.544	0.0004	14.024	21.449	0.061	24.6	529.146	21.510

4.1.1 Descriptive Analysis

A descriptive analysis of Table-2 gives us more idea about the trend of the NodeJS frameworks. We can see average energy consumption of the frameworks is 21.60 joule per second.

	Framework	CPU	Monitor	Disk	Base	Total Hardware Energy	Application Energy	Total Time	Energy Consumed Total(W)	Energy Consumed in(J/S)
0	Express	0.834	6.571	0.0000	14.081	21.487	0.051	24.5	527.6810	21.538
1	Fastify	0.895	6.598	0.0020	14.139	21.636	0.033	24.4	528.7236	21.669
2	Nest	0.922	6.598	0.0000	14.139	21.659	0.038	24.4	529.4068	21.697
3	Connect	0.880	6.544	0.0004	14.024	21.449	0.061	24.6	529.1460	21.510

Figure 3: Summary of The Total Experiment

	CPU	Monitor	Disk	Base	Total Hardware Energy	Application Energy	Total Time	Energy Consumed Total(W)	Energy Consumed in(J/S)
count	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.00000	4.000000
mean	0.882750	6.577750	0.000600	14.095750	21.557750	0.045750	24.475000	528.73935	21.603500
std	0.036854	0.025851	0.000952	0.055096	0.105209	0.012685	0.095743	0.75965	0.093211
min	0.834000	6.544000	0.000000	14.024000	21.449000	0.033000	24.400000	527.68100	21.510000
25%	0.868500	6.564250	0.000000	14.066750	21.477500	0.036750	24.400000	528.46295	21.531000
50%	0.887500	6.584500	0.000200	14.110000	21.561500	0.044500	24.450000	528.93480	21.603500
75%	0.901750	6.598000	0.000800	14.139000	21.641750	0.053500	24.525000	529.21120	21.676000
max	0.922000	6.598000	0.002000	14.139000	21.659000	0.061000	24.600000	529.40680	21.697000

Figure 4: Descriptive Analysis of the Frameworks Energy Consumption

4.1.2 T-test

A 2 tail T-test is done taking the data from Table 2 to see if there is a significance difference between Total Hardware energy and Total energy consumed by each framework. The hypothesises are-

- **Null Hypothesis(H_0):** The mean of total Hardware energy is equal to the mean Total energy consumed by each framework.
- **Alternate Hypothesis(H_a):** The mean of total Hardware energy is not equal to the mean Total energy consumed by each framework.

confidence is chosen 99%. so alpha value is when conducting the t-test is 0.01.
The T-test result shows that the null hypothesis was correct. So mean of Total Hardware energy is not significantly different from the mean of Total Energy Consumed by each framework for the current experiment.

```
------------------------------------------------------------
T test between Total Hardware Energy and Avg Energy Consumed(W)
------------------------------------------------------------
The variance ratio is:  1.2739976980625165
p value:  0.5391838599236958
t value:  -0.6509653112609995
Null Hypothesis Is Accepted
```

Figure 5: T-Test

4.1.3 Anova

A one-way ANOVA 1 tail test was done to see if the mean Total Energy consumption of different frameworks is the same or not.

- **Null Hypothesis(H_0):** The means are equal for every framework.
- **Alternate Hypothesis(H_a):** The means are not equal for every framework.

confidence is chosen 99%. so alpha value is 0.01.
The ANOVA test shows that every mean Total power consumption is similar. So it aligns with the Null Hypothesis. NUll hypothesis is accepted.

```
------------------------------------------------------
Anova Test of Energy consumed totally by the frameworks
------------------------------------------------------
F-value: 3.995357390283667
P-value: 0.18366150147279456
Fail to reject null hypothesis
```

Figure 6: ANOVA-Test

Figure 7 shows the Total energy consumed by only the hardware with respect to the total energy consumed by the system. The bar chart is grouped by the Framework names.

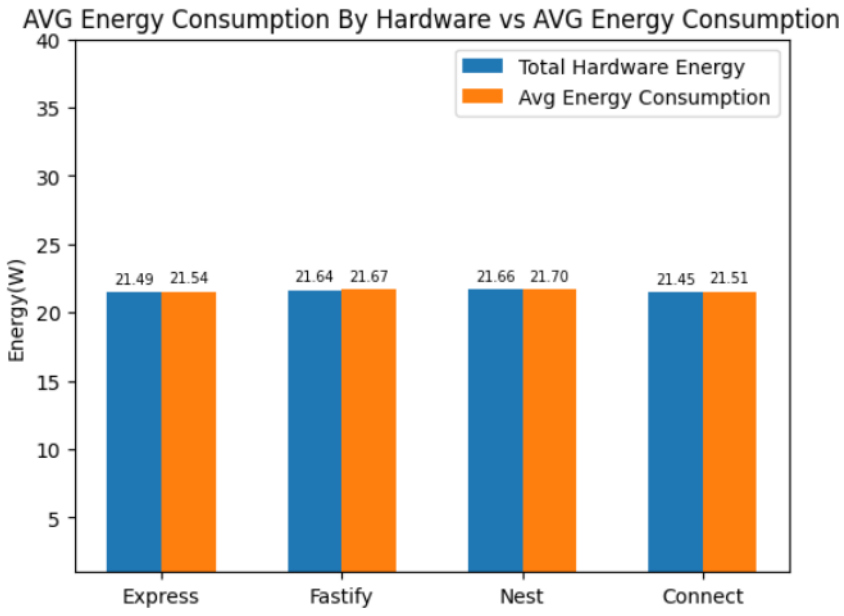


Figure 7: Total Hardware Energy Consumption vs Total Consumption of Energy

In Figure 8 the details of hardware energy consumption are shown. It depicts exactly how much energy each component took for the experiments.

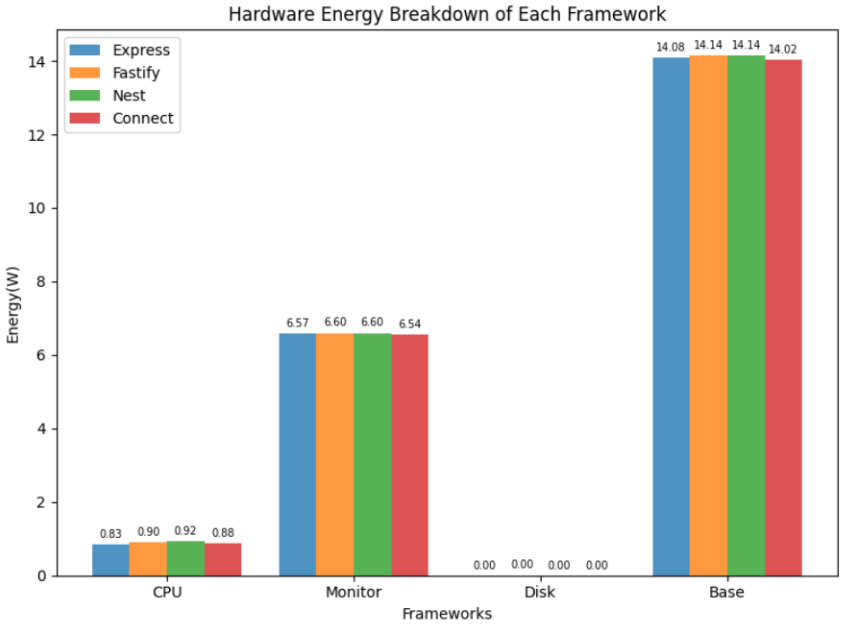


Figure 8: Hardware Energy Consumption by Different Components

4.2 Emission of Each Framework

Using the GHG gas conversation factor carbon emission of each framework is calculated. The emission is calculated using the total energy consumed by each framework converting it from J/S to KJ/Hour and then multiplying the converting factor. The Unit of the result in Kg.

Framework	Electricity generated				T&D- UK electricity				Distribution - district heat & steam			
	Kg CO_2e	Kg CO_2	Kg CH_4	mg N_2O	Kg CO_2e	Kg CO_2	Kg CH_4	mg N_2O	Kg CO_2e	Kg CO_2	Kg CH_4	mg N_2O
Express	14.994	14.825	0.062	0.106	1.371	1.356	0.005	0.009	0.697	0.690	0.004	0.002
Fastify	15.085	14.915	0.062	0.106	1.379	1.365	0.005	0.009	0.701	0.694	0.004	0.002
Nest	15.104	14.935	0.062	0.107	1.381	1.366	0.005	0.009	0.702	0.695	0.004	0.002
Connect	14.974	14.806	0.061	0.106	1.369	1.355	0.005	0.009	0.696	0.689	0.004	0.002

Table 4: GHG Emission for 4 Framework

Some graphs are given below for a better understanding of the values in the table above.

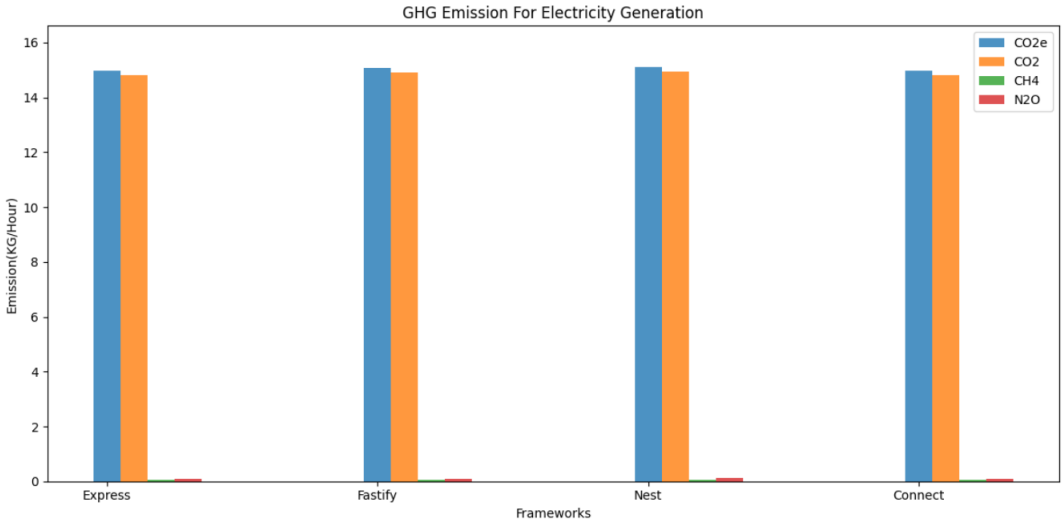


Figure 9: GHG Emission Based on Electricity Generation

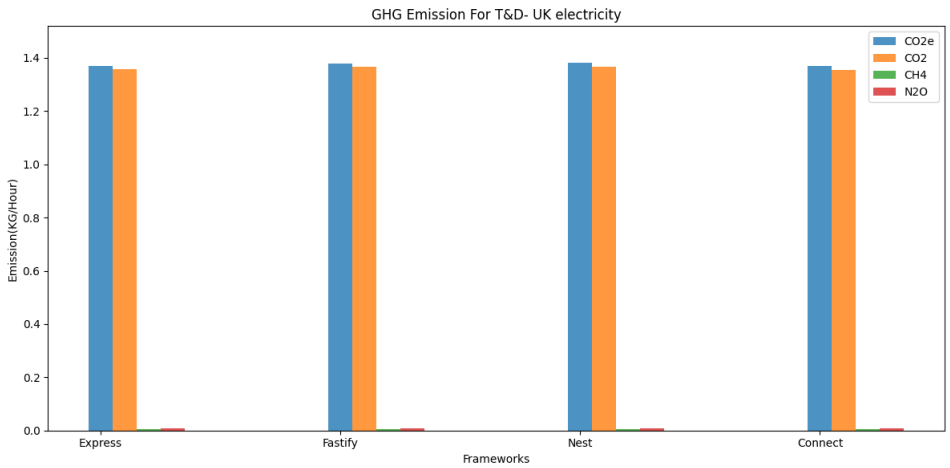


Figure 10: GHG Emission Based on T&D- UK electricity

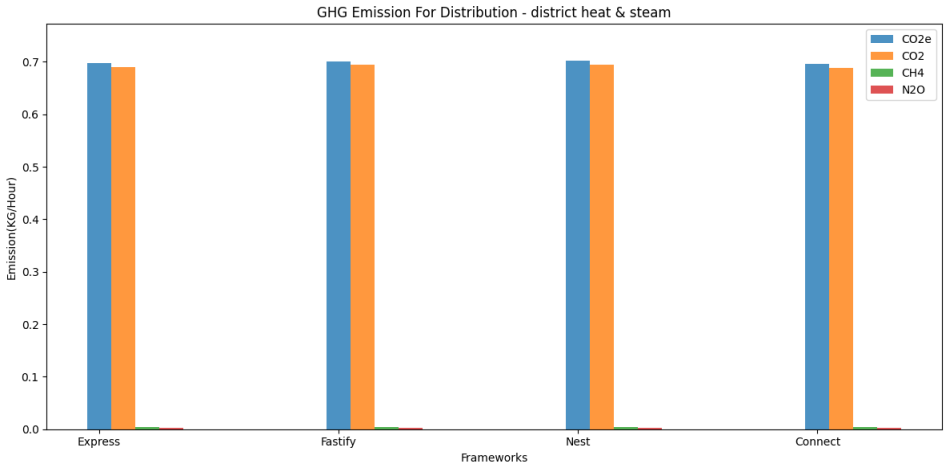


Figure 11: GHG Emission Based on Distribution - district heat & steam

4.3 Result Analysis:

From the analysis of the data and T-test and Anova test, it is evident that the primary Null Hypothesis(H_0) is correct. So the average energy consumption of different nodejs frameworks is the same for the same service. The T-Test of Hardware energy and Total average energy consumed shows in our experiment hardware energy takes a huge portion. This is because we are doing micro-testing. As the volume of the code base increases the application consumption will start to take significance in the total consumption. The tests have been done with one API which gets the value from the MongoDB database and returns the value. It uses a get operation. It has been seen that the energy consumption change is very little when switching frameworks if the hardware stays the same. The mean energy consumption is 21.60 joule/second. The GHG emission was calculated using the conversion factor and observed that emission due to electricity generation leads the chart. So, it can be said that in developing Nodejs Projects, the energy consumption is similar irrespective of which popular framework is being used.

5 Conclusion & Recommendation

In this report, A comparative study between 4 NodeJS frameworks and their energy consumption has been studied. The four frameworks in this study are Express, Fastify, Nest, and Connect. The applications have been developed in Javascript language. For taking the energy reading Joulemeter software has been used. The analysis was done in Google Colab using Python language. During the experiments, an API was hit using Postman software. The GHG emissions amount was calculated from the power consumption reading using the newest UK Government GHG Conversion Factors from 2022.

By doing the experiments 10 times for each of the frameworks and taking the average of power in watt units, it can be stated that all 4 frameworks take on average about the same amount of energy to do the experiments.

To ensure this result Anova test was done with total consumed data. T-test with Hardware consumption and total consumption was done and found that for our experiment the hardware power consumption outweighs the application power consumption.

The experiments and results give some interesting insight into working with the Nodejs environment. Among the four described frameworks as there is no mean difference in energy consumption, so the developers can choose any one. Moreover, different framework has different efficiency in terms of API handling or developing difficulty or database connectivity. So, according to the requirement of the company or institution, one can use any framework which suits the aims and goals.

Future work may include more heavy service API built inside each application and take the values. In this experiment, the application was run for a small amount of the to get the data. In future, experiments can be designed in a way that the application is kept running for a longer period of time and collect the output energy consumption. More frameworks can be tested using the same experiments to check if they comply with the null hypothesis or not. In terms of auditing tools along with Joulemeter, some other **tools** like Intel power gadget, Intel PowerLog, Powerstat, etc. The experiment was done in Windows 11 environment. Some other Operating systems like Ubuntu and IOS can be chosen to run the experiments in the future.

References

Arushanyan, Yevgeniya, Elisabeth Ekener-Petersen, and Göran Finnveden (2014). "Lessons learned – Review of LCAs for ICT products and services". In: *Computers in Industry* 65.2, pp. 211–234. ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2013.10.003. URL: https://www.sciencedirect.com/science/article/pii/S0166361513002133.

Bawane, Mohanish et al. (2022). "A Review on Technologies used in MERN stack". In: *Int. J. Res. Appl. Sci. Eng. Technol* 10.1, pp. 479–488.

Bekaroo, Girish, Chandradeo Bokhoree, and Colin Pattinson (2014). "Power measurement of computers: analysis of the effectiveness of the software based approach". In: *Int. J. Emerg. Technol. Adv. Eng* 4.5, pp. 755–762.

Brown, Ethan (2014). *Web Development with Node and Express: Leveraging the JavaScript Stack*. 1st. O'Reilly Media, Inc. ISBN: 978-1491949306. URL: https://www.oreilly.com/library/view/web-development-with/9781491949306/.

Cantelon, Mike et al. (2014). *Node. js in Action*. Manning Greenwich.

Cisco, U (2020). "Cisco annual internet report (2018–2023) white paper". In: *Cisco: San Jose, CA, USA* 10.1, pp. 1–35.

Contributor, Guest (Jan. 2023). *7 best Node.js frameworks to build backend APIs in 2023*. https://www.kindacode.com/article/best-node-js-frameworks-to-build-backend-apis/.

Demashov, Danil and Ilya Gosudarev (2019). "Efficiency Evaluation of Node. js Web-Server Frameworks." In: *MICSECS*.

Do, Thanh, Suhib Rawshdeh, and Weisong Shi (2009). *ptop: A process-level power profiling tool*.

Flinn, J. and M. Satyanarayanan (1999). "PowerScope: a tool for profiling the energy usage of mobile applications". In: *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pp. 2–10. DOI: 10.1109/MCSA.1999.749272.

Freitag, Charlotte, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon Blair, et al. (Feb. 2021). "The climate impact of ICT: A review of estimates, trends and regulations". In.

Freitag, Charlotte, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon S. Blair, et al. (2021). "The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations". In: *Patterns* 2.9, p. 100340. ISSN: 2666-3899. DOI: https://doi.org/10.1016/j.patter.2021.100340. URL: https://www.sciencedirect.com/science/article/pii/S2666389921001884.

Ghaleb, Taher Ahmed (2019). "Software Energy Measurement at Different Levels of Granularity". In: *2019 International Conference on Computer and Information Sciences (ICCIS)*, pp. 1–6. DOI: 10.1109/ICCISci.2019.8716456.

Greiff, Magnus and André Johansson (2019). *Symfony vs Express: A Server-Side Framework Comparison*.

Haviv, Amos Q. (2016). *MEAN Web Development - Second Edition*. 2nd. Packt Publishing. ISBN: 978-1785289644. URL: https://www.packtpub.com/web-development/mean-web-development-second-edition.

Huang, Xiaoping (2020). "Research and Application of Node.js Core Technology". In: *2020 International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI)*, pp. 1–4. DOI: 10.1109/ICHCI51889.2020.00008.

Knowles, Bran (2021). *ACM TechBrief: Computing and Climate Change*. New York, NY, USA.

Kothari, Nupur and Arka Bhattacharya (2009). "Joulemeter: Virtual machine power measurement and management". In: *MSR Tech Report*.

Lei, Kai, Yining Ma, and Zhi Tan (2014). "Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js". In: *2014 IEEE 17th International Conference on Computational Science and Engineering*, pp. 661–668. DOI: 10.1109/CSE.2014.142.

Magazzino, Cosimo et al. (Jan. 2021). "Investigating the link among ICT, electricity consumption, air pollution, and economic growth in EU countries". In: *Energy sources. Part B Economics, planning and policy* 16. DOI: 10.1080/15567249.2020.1868622.

Malmodin, Jens and Pernilla Bergmark (2015/09). "Exploring the effect of ICT solutions on GHG emissions in 2030". In: *Proceedings of EnviroInfo and ICT for Sustainability 2015*. Atlantis Press, pp. 37–46. ISBN: 978-94-62520-92-9. DOI: 10.2991/ict4s-env-15.2015.5. URL: https://doi.org/10.2991/ict4s-env-15.2015.5.

Mardan, Azat (2014). *Express. js Guide: The Comprehensive Book on Express. js*. Azat Mardan.

Microsoft, Research (2010). *Joulemeter: Computational Energy Measurement and Optimization*. https://www.microsoft.com/en-us/research/project/joulemeter-computational-energy-measurement-and-optimization/.

Pham, Anh Duc (2020). "Developing back-end of a web application with NestJS framework: Case: Integrify Oy's student management system". In.

Release, Press (2020). *ICT industry to reduce greenhouse gas emissions by 45 per cent by 2030*. https://www.itu.int/en/mediacentre/Pages/PR04-2020-ICT-industry-to-reduce-greenhouse-gas-emissions-by-45-percent-by-2030.aspx.

Satheesh, Mithun, Bruno Joseph D'mello, and Jason Krol (2015). *Web development with MongoDB and NodeJs*. Packt Publishing Ltd.

Scarlat, Nicolae et al. (2015). "Renewable energy policy framework and bioenergy contribution in the European Union – An overview from National Renewable Energy Action Plans and Progress Reports". In: *Renewable and Sustainable Energy Reviews* 51, pp. 969–985. ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2015.06.062. URL: https://www.sciencedirect.com/science/article/pii/S1364032115006346.

Wang, Jing and Yubing Xu (2021). "Internet Usage, Human Capital and CO2 Emissions: A Global Perspective". In: *Sustainability* 13.15. ISSN: 2071-1050. DOI: 10.3390/su13158268. URL: https://www.mdpi.com/2071-1050/13/15/8268.

A Appendices

A.1 Github Link

All the Codes of Different Framework along with Excel sheets of experiments and screenshots will be found in the GitHub link

- **Github Link: https://github.com/NMLami/ICTE_REPORT_CODES_LBU**
- **Excel Sheet Github Link: Excel Sheet GitHub Link**
- **Screenshot Github Link: Screenshot GitHub Link**
- **Data Analysis Code Github Link: Data Analysis GitHub Link**

A.2 FrameWork Codes

Only main file screenshots are given here to get the idea. For detailed code, refer to the GitHub link
Express Framework

src > index.ts

```ts
import express, { Express} from 'express';

import cors from "cors";
import config from "config";

import dbConnection from "./db-connection/db-connection";
import logger from "./utils/logger";
import router from "./routes/index";

// import { errorHandler } from "./middleware/error-handler";

const app : Express = express();

app.use(cors());
app.use(express.urlencoded({extended: true}));
app.use(express.json());
// app.use(helmet());

dbConnection();

app.use(router);

const port = 3001 ;
app.listen(port, async () => {

    logger.info(`Application successfully connected on http://localhost:${port}`);
});
```

src > routes > homepage-routes.ts > ...

```ts
import express from "@awaitjs/express";
import config from "config";



import { getInfo } from "../controllers/homepage-controller";

const homepageRoute = express.Router();


homepageRoute.getAsync("/get", getInfo);

export default homepageRoute;
```

src > controllers > homepage-controller.ts > ...

```ts
import { Request, Response, NextFunction } from "express";
import mongoose from "mongoose";
import { successResponseHandler } from "../middleware/success-response-handler";

import logger from "./../utils/logger";
// import { addBrandInfo, updateBrandInfo } from "../services/brand-service";
// import { getDefaultSelectedPartnerId, getDefaultSelectedCompanyId } from "../services/auth-service";

//Pass the validated info to Service

const AirbnbSchema = new mongoose.Schema(
    {
        name: {
            type: String,

        }
    }
);


// const Blog = mongoose.model("listingsAndReviews", BlogSchema);

export const getInfo = async (req: Request, res: Response, next: NextFunction): Promise<any> => {
    try {
        logger.info(`API is hit`);
        var Airbnb = mongoose.model('listingsAndReviews',AirbnbSchema);
        let data  = await  Airbnb.find({});

        await successResponseHandler(res, 201, "Information fetched successfully!","data", data);
    } catch (err: any) {
        next(err);
    }
};
```

package.json > {} devDependencies

```
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@awaitjs/express": "^0.9.0",
    "config": "^3.3.9",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "moment": "^2.29.4",
    "mongoose": "^7.0.3",
    "pino": "^8.11.0",
    "pino-pretty": "^10.0.0"
  },
  "devDependencies": {
    "@types/config": "^3.3.0",
    "@types/cors": "^2.8.13",
    "@types/express": "^4.17.17",
    "@types/node": "^18.15.11",
    "ts-node-dev": "^2.0.0",
    "typescript": "^5.0.3"
  }
}
```

Fastify Framework

src > index.ts > ...

```
import fastify from 'fastify'
import cors from "cors";
import config from "config";
import mongoose from "mongoose";
import logger from "./utils/logger";
import dbConnection from "./db-connection/db-connection";
const server = fastify()

dbConnection();

server.get('/homepage', async (request, reply) => {
  logger.info(`API is hit`);
    const AirbnbSchema = new mongoose.Schema(
        {
            name: {
                type: String,

            }
        }
    );

    var Airbnb = mongoose.model('listingsAndReviews',AirbnbSchema);
    let data  = await  Airbnb.find({});

  reply.send({"status": "success","statusCode":200, "data": data})
})

const port = 3002
server.listen({ port: port  }, (err, address) => {
  if (err) {
    console.error(err)
    process.exit(1)
  }
  logger.info(`Application successfully connected on http://localhost:${port}`);
})
```

package.json > {} devDependencies

```
{
  Debug
  "scripts": {
    "build": "tsc -p tsconfig.json",
    "start": "node index.js",
    "dev": "ts-node-dev --respawn --transpile-only src/index.ts"
  },
  "dependencies": {
    "fastify": "^4.15.0",
    "config": "^3.3.9",
    "cors": "^2.8.5",
    "moment": "^2.29.4",
    "mongoose": "^7.0.3",
    "pino": "^8.11.0",
    "pino-pretty": "^10.0.0"
  },
  "devDependencies": {
    "@types/config": "^3.3.0",
    "@types/cors": "^2.8.13",
    "@types/node": "^18.15.11",
    "ts-node-dev": "^2.0.0",
    "typescript": "^5.0.3"
  }
}
```

Nest Framework

src > main.ts > ...

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import dbConnection from "./db-connection/db-connection";
import logger from "./utils/logger";


async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  dbConnection();
  const port = 3003;
  await app.listen(port).then(()=>{
    logger.info(`Application successfully connected on http://localhost:${port}`);
  });
}
bootstrap();

```

src > app.controller.ts > ...

```
import { Controller, Get } from '@nestjs/common';
import { AppService } from './app.service';
import logger from "./utils/logger";

@Controller('/homepage')
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  getHomepage(): Promise<any> {
    logger.info(`API is hit`);
    return this.appService.getHomepage();
  }
}

```

src > app.service.ts > AppService > getHomepage

```ts
import { Injectable } from '@nestjs/common';
import mongoose from "mongoose";
@Injectable()
export class AppService {
async getHomepage(): Promise<any> {


    const AirbnbSchema = new mongoose.Schema(
      {
        name: {
          type: String,

        }
      }
    );
    var Airbnb = mongoose.model('listingsAndReviews',AirbnbSchema);
      let data  = await  Airbnb.find({});
      return {"status": "sucess","statusCode":200, "data": data};
  }
}

```

```json
{
  "name": "3-nestjs-framework",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  Debug
  "scripts": {
    "prebuild": "rimraf dist",
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
    "start": "nest start",
    "start:dev": "nest start --watch",
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-node/register node_modules/.bin/jest --runInBand",
    "test:e2e": "jest --config ./test/jest-e2e.json"
  },
  "dependencies": {
    "@nestjs/common": "^7.6.15",
    "@nestjs/core": "^7.6.15",
    "@nestjs/platform-express": "^7.6.15",
    "reflect-metadata": "^0.1.13",
    "rimraf": "^3.0.2",
    "rxjs": "^6.6.6",
    "config": "^3.3.9",
    "cors": "^2.8.5",
    "moment": "^2.29.4",
```

Connect Framework

```
src > index.ts > homepage
var connect = require("connect");
var app = connect();
import mongoose from "mongoose";
import logger from "./utils/logger";
import dbConnection from "./db-connection/db-connection";

async function homepage(req:any, res:any, next:any) {
    logger.info(`API is hit`);
    const AirbnbSchema = new mongoose.Schema(
        {
            name: {
                type: String,

            }
        }
    );

    var Airbnb = mongoose.model('listingsAndReviews',AirbnbSchema);
    let data  = await  Airbnb.find({});
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.write(JSON.stringify(data));
    res.end();

}
dbConnection();

const port = 3004;
app
    .use('/homepage',homepage)
    .listen(port,()=>{
    logger.info(`Application successfully connected on http://localhost:${port}`);
    });
```

```
package.json > {} devDependencies
{
  Debug
  "scripts": {
    "build": "tsc -p tsconfig.json",
    "start": "node index.js",
    "dev": "ts-node-dev --respawn --transpile-only src/index.ts"
  },
  "dependencies": {
    "connect": "^3.7.0",
    "config": "^3.3.9",
    "cors": "^2.8.5",
    "moment": "^2.29.4",
    "mongoose": "^7.0.3",
    "pino": "^8.11.0",
    "pino-pretty": "^10.0.0"
  },
  "devDependencies": {
    "@types/config": "^3.3.0",
    "@types/cors": "^2.8.13",
    "@types/node": "^18.15.11",
    "typescript": "^5.0.4",
    "ts-node-dev": "^2.0.0"
  }
}
```

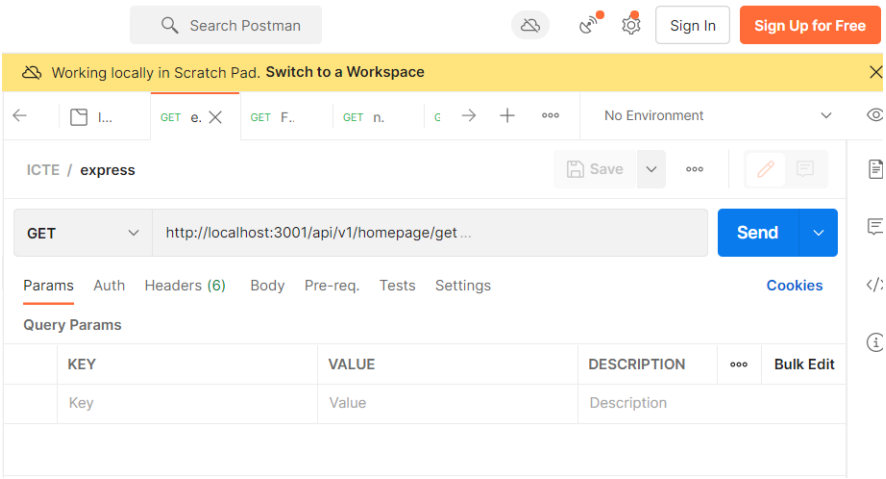
A.3 Experiment

After starting the Joulemeter data collection application code was run. Then after around 25 seconds the program was stopped and the energy value was collected. Meanwhile, API was hit by the Postman software.

```
Lami@DESKTOP-7CTLR9A MINGW64 /d/PROJECTS/ICTE PROJECT/1_Express_Framework
$ npm run dev

> 1_express_framework@1.0.0 dev
> ts-node-dev --respawn --transpile-only src/index.ts

[INFO] 20:22:06 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 5.0.3)
[20:22:07.000] INFO: Application successfully connected on http://localhost:3001
[20:22:09.000] INFO: Successfully connected to database
[20:22:13.000] INFO: API is hit
```



A	B	C	D	E	F	G	H
TimeStam	Total Pow	CPU (W)	Monitor ('	Disk (W)	Base (W)	Applicatio	time
6.38E+13	22.6	0.6	7	0	15	0	14:57:51
6.38E+13	22.8	0.8	7	0	15	0.1	14:57:52
6.38E+13	24.2	2.2	7	0	15	0.2	14:57:53
6.38E+13	26.1	4.1	7	0	15	0.2	14:57:54
6.38E+13	23.7	1.7	7	0	15	0.1	14:57:55
6.38E+13	22	0	7	0	15	0	14:57:56
6.38E+13	22.9	0.9	7	0	15	0	14:57:57
6.38E+13	22	0	7	0	15	0	14:57:59
6.38E+13	22.2	0.2	7	0	15	0.1	14:58:00
6.38E+13	23.2	1.2	7	0	15	0.5	14:58:01
6.38E+13	23.3	1.3	7	0	15	0.1	14:58:02
6.38E+13	22.5	0.5	7	0	15	0	14:58:03
6.38E+13	22.3	0.3	7	0	15	0	14:58:04
6.38E+13	22.9	0.9	7	0	15	0.1	14:58:05
6.38E+13	22.6	0.6	7	0	15	0	14:58:06
6.38E+13	22.6	0.6	7	0	15	0	14:58:07
6.38E+13	22.8	0.8	7	0	15	0	14:58:09
6.38E+13	22.2	0.2	7	0	15	0	14:58:10
6.38E+13	22.5	0.5	7	0	15	0	14:58:11
6.38E+13	23.1	1.1	7	0	15	0	14:58:12
6.38E+13	23	1	7	0	15	0	14:58:13
6.38E+13	22.3	0.3	7	0	15	0	14:58:14
6.38E+13	22	0	7	0	15	0.2	14:58:15
6.38E+13	22.7	0.7	7	0	15	0	14:58:16

A.4 Data Analysis

Data Analysis Screenshots are given below. For Simplicity, expressjs framework codes are given with final calculations. The rest of the framework's code follows the same procedure. For details refer to the GitHub link Code folder.

Analysis of ExpressJs Framework

Data Analysis of Energy consumption of Express Nodejs Framework

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# load Express Experiment data
T1 = "drive/My Drive/Study/ICTE/express/T1.csv"
T2 = "drive/My Drive/Study/ICTE/express/T2.csv"
T3 = "drive/My Drive/Study/ICTE/express/T3.csv"
T4 = "drive/My Drive/Study/ICTE/express/T4.csv"
T5 = "drive/My Drive/Study/ICTE/express/T5.csv"
T6 = "drive/My Drive/Study/ICTE/express/T6.csv"
T7 = "drive/My Drive/Study/ICTE/express/T7.csv"
T8 = "drive/My Drive/Study/ICTE/express/T8.csv"
T9 = "drive/My Drive/Study/ICTE/express/T9.csv"
T10 = "drive/My Drive/Study/ICTE/express/T10.csv"

df_T1 = pd.read_csv(T1, parse_dates=['time'])
df_T2 = pd.read_csv(T2, parse_dates=['time'])
df_T3 = pd.read_csv(T3, parse_dates=['time'])
df_T4 = pd.read_csv(T4, parse_dates=['time'])
df_T5 = pd.read_csv(T5, parse_dates=['time'])
df_T6 = pd.read_csv(T6, parse_dates=['time'])
df_T7 = pd.read_csv(T7, parse_dates=['time'])
df_T8 = pd.read_csv(T8, parse_dates=['time'])
df_T9 = pd.read_csv(T9, parse_dates=['time'])
df_T10 = pd.read_csv(T10, parse_dates=['time'])
```

```
# Set max_rows and max_columns options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

```
# Set the start and end timestamp of experiment 1

start_ts = pd.Timestamp('2:57:54 PM')
end_ts = pd.Timestamp('2:58:19 PM')

time_diff1 = (end_ts - start_ts).total_seconds()
display(time_diff1)

# Extract the rows that fall within the timestamp range
filtered_T1 = df_T1[(df_T1['time'] >= start_ts) & (df_T1['time'] <= end_ts)]

# display(filtered_T1)
```

```
25.0
```

```
# Set the start and end timestamp of experiment 2
start_ts = pd.Timestamp('2:59:00 PM')
end_ts = pd.Timestamp('2:59:24 PM')

time_diff2 = (end_ts - start_ts).total_seconds()
display(time_diff2)
# Extract the rows that fall within the timestamp range
filtered_T2 = df_T2[(df_T2['time'] >= start_ts) & (df_T2['time'] <= end_ts)]
# display(filtered_T2)
```

```
24.0
```

```
# Set the start and end timestamp of experiment 3
start_ts = pd.Timestamp('2:59:47 PM')
end_ts = pd.Timestamp('3:00:11 PM')

time_diff3 = (end_ts - start_ts).total_seconds()
display(time_diff3)
# Extract the rows that fall within the timestamp range
filtered_T3 = df_T3[(df_T3['time'] >= start_ts) & (df_T3['time'] <= end_ts)]
# display(filtered_T3)
```

```
# Set the start and end timestamp of experiment 4
start_ts = pd.Timestamp('3:16:02 PM')
end_ts = pd.Timestamp('3:16:27 PM')

time_diff4 = (end_ts - start_ts).total_seconds()
display(time_diff4)
# Extract the rows that fall within the timestamp range
filtered_T4 = df_T4[(df_T4['time'] >= start_ts) & (df_T4['time'] <= end_ts)]
# display(filtered_T4)
```

```
25.0
```

```
# Set the start and end timestamp of experiment 5
start_ts = pd.Timestamp('3:16:58 PM')
end_ts = pd.Timestamp('3:17:22 PM')

time_diff5 = (end_ts - start_ts).total_seconds()
display(time_diff5)
# Extract the rows that fall within the timestamp range
filtered_T5 = df_T5[(df_T5['time'] >= start_ts) & (df_T5['time'] <= end_ts)]
# display(filtered_T5)
```

```
24.0
```

```
# Set the start and end timestamp of experiment 6
start_ts = pd.Timestamp('8:17:49 PM')
end_ts = pd.Timestamp('8:18:13 PM')

time_diff6 = (end_ts - start_ts).total_seconds()
display(time_diff6)
# Extract the rows that fall within the timestamp range
filtered_T6 = df_T6[(df_T6['time'] >= start_ts) & (df_T6['time'] <= end_ts)]
# display(filtered_T6)
```

```
24.0
```


```
# Set the start and end timestamp of experiment 4
start_ts = pd.Timestamp('3:16:02 PM')
end_ts = pd.Timestamp('3:16:27 PM')

time_diff4 = (end_ts - start_ts).total_seconds()
display(time_diff4)
# Extract the rows that fall within the timestamp range
filtered_T4 = df_T4[(df_T4['time'] >= start_ts) & (df_T4['time'] <= end_ts)]
# display(filtered_T4)
```

```
25.0
```

```
# Set the start and end timestamp of experiment 5
start_ts = pd.Timestamp('3:16:58 PM')
end_ts = pd.Timestamp('3:17:22 PM')

time_diff5 = (end_ts - start_ts).total_seconds()
display(time_diff5)
# Extract the rows that fall within the timestamp range
filtered_T5 = df_T5[(df_T5['time'] >= start_ts) & (df_T5['time'] <= end_ts)]
# display(filtered_T5)
```

```
24.0
```

```
# Set the start and end timestamp of experiment 6
start_ts = pd.Timestamp('8:17:49 PM')
end_ts = pd.Timestamp('8:18:13 PM')

time_diff6 = (end_ts - start_ts).total_seconds()
display(time_diff6)
# Extract the rows that fall within the timestamp range
filtered_T6 = df_T6[(df_T6['time'] >= start_ts) & (df_T6['time'] <= end_ts)]
# display(filtered_T6)
```

```
24.0
```


```
# Set the start and end timestamp of experiment 7
start_ts = pd.Timestamp('8:19:59 PM')
end_ts = pd.Timestamp('8:20:24 PM')

time_diff7 = (end_ts - start_ts).total_seconds()
display(time_diff7)

# Extract the rows that fall within the timestamp range
filtered_T7 = df_T7[(df_T7['time'] >= start_ts) & (df_T7['time'] <= end_ts)]
# display(filtered_T7)
```

```
25.0
```

```
# Set the start and end timestamp of experiment 8
start_ts = pd.Timestamp('8:22:07 PM')
end_ts = pd.Timestamp('8:22:32 PM')

time_diff8 = (end_ts - start_ts).total_seconds()
display(time_diff8)
# Extract the rows that fall within the timestamp range
filtered_T8 = df_T8[(df_T8['time'] >= start_ts) & (df_T8['time'] <= end_ts)]
# display(filtered_T8)
```

```
25.0
```

```
# Set the start and end timestamp of experiment 9
start_ts = pd.Timestamp('8:24:07 PM')
end_ts = pd.Timestamp('8:24:32 PM')

time_diff9 = (end_ts - start_ts).total_seconds()
display(time_diff9)
# Extract the rows that fall within the timestamp range
filtered_T9 = df_T9[(df_T9['time'] >= start_ts) & (df_T9['time'] <= end_ts)]
```

```
25.0
```

```
# Set the start and end timestamp of experiment 10
start_ts = pd.Timestamp('8:28:13 PM')
end_ts = pd.Timestamp('8:28:37 PM')

time_diff10 = (end_ts - start_ts).total_seconds()
display(time_diff10)
# Extract the rows that fall within the timestamp range
filtered_T10 = df_T10[(df_T10['time'] >= start_ts) & (df_T10['time'] <= end_ts)]
```

```
24.0
```

```
#Time Mean
TIME_MEAN = (time_diff1+time_diff2+time_diff3+time_diff4+time_diff5+time_diff6+time_diff7+time_diff8+time_diff9+time_diff10)/10
print(TIME_MEAN)
```

```
24.5
```

```
#CPU Aplication Power
CPU_APPLICATION_POWER_TOTAL = filtered_T1[' Application (W)'].sum() + filtered_T2[' Application (W)'].sum() + filtered_T3[' Application (W)'].sum() + filtered_T4[' Application (W)'].sum() + filtered_T5[' Application (W)'].sum()+ fil
CPU_APPLICATION_POWER_MEAN = CPU_APPLICATION_POWER_TOTAL/(10*TIME_MEAN)
print(CPU_APPLICATION_POWER_MEAN)
```

```
0.051836734693877555
```

```
#CPU  Power
CPU_POWER_TOTAL = filtered_T1[' CPU (W)'].sum() + filtered_T2[' CPU (W)'].sum() + filtered_T3[' CPU (W)'].sum() + filtered_T4[' CPU (W)'].sum() + filtered_T5[' CPU (W)'].sum()+ filtered_T6[' CPU (W)'].sum()  + filtered_T7[' CPU (W)'
CPU_POWER_MEAN = CPU_POWER_TOTAL/(10*TIME_MEAN)
print(CPU_POWER_MEAN)
```

```
0.8342857142857143
```

```
#Monitor  Power
MONITOR_POWER_TOTAL = filtered_T1[' Monitor (W)'].sum() + filtered_T2[' Monitor (W)'].sum() + filtered_T3[' Monitor (W)'].sum() + filtered_T4[' Monitor (W)'].sum() + filtered_T5[' Monitor (W)'].sum()+ filtered_T6[' Monitor (W)'].sum
MONITOR_POWER_MEAN = MONITOR_POWER_TOTAL/(10*TIME_MEAN)
print(MONITOR_POWER_MEAN)
```

```
6.571428571428571
```

```
#Disk  Power
DISK_POWER_TOTAL = filtered_T1[' Disk (W)'].sum() + filtered_T2[' Disk (W)'].sum() + filtered_T3[' Disk (W)'].sum() + filtered_T4[' Disk (W)'].sum() + filtered_T5[' Disk (W)'].sum()+ filtered_T6[' Disk (W)'].sum()  + filtered_T7[
DISK_POWER_MEAN =DISK_POWER_TOTAL/(10*TIME_MEAN)
print(DISK_POWER_MEAN)
```

```
0.0
```

+ Code + Text

```
#BASE  Power
BASE_POWER_TOTAL = filtered_T1[' Base (W)'].sum() + filtered_T2[' Base (W)'].sum() + filtered_T3[' Base (W)'].sum() + filtered_T4[' Base (W)'].sum() + filtered_T5[' Base (W)'].sum()+ filtered_T6[' Base (W)'].sum()  + filtered_T7[
BASE_POWER_MEAN = BASE_POWER_TOTAL/(10*TIME_MEAN)
print(BASE_POWER_MEAN)
```

```
14.081632653061224
```

```
#Total Hardware energy
TOTAL_HARDWARE_ENERGY = (CPU_POWER_MEAN + MONITOR_POWER_MEAN + DISK_POWER_MEAN + BASE_POWER_MEAN)
print(TOTAL_HARDWARE_ENERGY)
```

```
21.48734693877551
```

```
#Agreegated Total Energy
AGGEGATED_TOTAL_ENERGY = TOTAL_HARDWARE_ENERGY + CPU_APPLICATION_POWER_MEAN
print(AGGEGATED_TOTAL_ENERGY)
```

```
21.539183673469390
```

Final Analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# load data for ICTE Final Calculation
final_data = "drive/My Drive/Study/ICTE/ICTE_Final_calculation.xlsx"

df_final_data = pd.read_excel(final_data)

# Set max_rows and max_columns options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

```
display(df_final_data)
```

	Framework	CPU	Monitor	Disk	Base	Total Hardware Energy	Application Energy	Total Time	Energy Consumed Total(W)	Energy Consumed in(J/S)
0	Express	0.834	6.571	0.0000	14.081	21.487	0.051	24.5	527.6810	21.538
1	Fastify	0.895	6.598	0.0020	14.139	21.636	0.033	24.4	528.7236	21.669
2	Nest	0.922	6.598	0.0000	14.139	21.659	0.038	24.4	529.4068	21.697
3	Connect	0.880	6.544	0.0004	14.024	21.449	0.061	24.6	529.1460	21.510

```
#Descritive Analsis of the data
df_final_data.describe()
```

	CPU	Monitor	Disk	Base	Total Hardware Energy	Application Energy	Total Time	Energy Consumed Total(W)	Energy Consumed in(J/S)
count	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.00000	4.000000
mean	0.882750	6.577750	0.000600	14.095750	21.557750	0.045750	24.475000	528.73935	21.603500
std	0.036854	0.025851	0.000952	0.055096	0.105209	0.012685	0.095743	0.75965	0.093211
min	0.834000	6.544000	0.000000	14.024000	21.449000	0.033000	24.400000	527.68100	21.510000
25%	0.868500	6.564250	0.000000	14.066750	21.477500	0.036750	24.400000	528.46295	21.531000
50%	0.887500	6.584500	0.000200	14.110000	21.561500	0.044500	24.450000	528.93480	21.603500
75%	0.901750	6.598000	0.000800	14.139000	21.641750	0.053500	24.525000	529.21120	21.676000
max	0.922000	6.598000	0.002000	14.139000	21.659000	0.061000	24.600000	529.40680	21.697000

```
# T test between Total Hardware Energy and Energy Consumed(W)
from scipy import stats

column1 = 'Total Hardware Energy'
column2 = 'Energy Consumed in(J/S)'
df_final_data_tTest = df_final_data[[column1,column2]]

dataset_column_1 = df_final_data_tTest[column1]
dataset_column_2 = df_final_data_tTest[column2]

ratio= max(np.var(dataset_column_1), np.var(dataset_column_2))/min(np.var(dataset_column_1), np.var(dataset_column_2))
print("-----------------------------------------------------------")
print("T test between Total Hardware Energy and Avg Energy Consumed(W)")
print("-----------------------------------------------------------")

print("The variance ratio is: ", ratio)
alpha = 0.01

#t-test
t_value, p_value = stats.ttest_ind(dataset_column_1, dataset_column_2 )

print("p value: ", p_value)
print("t value: ", t_value)

if p_value < alpha:
  print("Null Hypothesis Is Rejected")
else:
  print("Null Hypothesis Is Accepted")
```

```
from scipy.stats import f_oneway

# Extract the column you want to perform the ANOVA test on
data = df_final_data['Energy Consumed Total(W)']

# Perform the one-tailed ANOVA test
f_value, p_value = f_oneway(data[data < data.mean()], data[data >= data.mean()])
print("-----------------------------------------------------------")
print("Anova Test of Energy consumed totally by the frameworks")
print("-----------------------------------------------------------")
# Print the results
print("F-value:", f_value)
print("P-value:", p_value)

alpha = 0.01
# Check if p-value is less than significance level
if p_value < alpha:
    print("Reject null hypothesis")
else:
    print("Fail to reject null hypothesis")
```

```python
#Plot AVG Energy Consumption By Hardware vs AVG Energy Consumption
import matplotlib.pyplot as plt
import numpy as np

# Data for the bars
member1 = df_final_data['Total Hardware Energy']
member2 = df_final_data['Energy Consumed in(J/S)']

# The x locations for the groups
x = np.arange(len(member1))
# Width of the bars
width = 0.3
# Plotting the bars
fig, ax = plt.subplots()
plt.ylim(1, 40)
rects1 = ax.bar(x - width/2, member1, width, label='Total Hardware Energy')
rects2 = ax.bar(x + width/2, member2, width, label='Avg Energy Consumption')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Energy(W)')
ax.set_title('AVG Energy Consumption By Hardware vs AVG Energy Consumption')
ax.set_xticks(x)
ax.set_xticklabels(['Express', 'Fastify', 'Nest', 'Connect'])
ax.legend()

# Function to add the value labels on top of the bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{:.2f}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom',
                fontsize=7)

# # Adding value labels on top of the bars
autolabel(rects1)
autolabel(rects2)
# Displaying the plot
plt.show()
```

```python
#Plot Hardware Energy Breakdown of Each Framework
import matplotlib.pyplot as plt
import numpy as np
# Sample data
group_names = ['CPU', 'Monitor', 'Disk', 'Base']
data = np.array([df_final_data['CPU'],
                 df_final_data['Monitor'],
                 df_final_data['Disk'],
                 df_final_data['Base']])
# Set up the figure and axis
fig, ax = plt.subplots(figsize=(8, 6))
bar_width = 0.2
opacity = 0.8

# Plot the bars
for i in range(data.shape[1]):
    x_pos = np.arange(data.shape[0]) + (i * bar_width)
    rects = ax.bar(x_pos, data[:, i], bar_width, alpha=opacity, label=df_final_data['Framework'][i])

    # Attach labels to the bars
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{:.2f}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom', fontsize=7)

# Set the axis labels and title
ax.set_xlabel('Frameworks')
ax.set_ylabel('Energy(W)')
ax.set_title('Hardware Energy Breakdown of Each Framework')
# Set the x-axis tick labels
ax.set_xticks(np.arange(data.shape[0]) + ((data.shape[1]-1)/2) * bar_width)
ax.set_xticklabels(group_names)
# Add the legend
ax.legend()
plt.tight_layout()
plt.show()
```

```
# Plot GHG Emission For Electricity Generation
import numpy as np
import matplotlib.pyplot as plt

# Datas are taken from manually from the excel sheet as the amount is not large
group_names = ['Express', 'Fastify', 'Nest', 'Connect']
data = np.array([[14.994 , 14.825,0.062 , 0.106],
                 [15.085 , 14.915 ,0.062, 0.106 ],
                 [15.104,14.935 , 0.062 , 0.107 ],
                 [14.974 , 14.806, 0.061, 0.106 ]])
# Create a figure and axis
fig, ax = plt.subplots(figsize=(12, 6))

# Set the x-axis ticks and tick labels
x_pos = np.arange(len(group_names))
ax.set_xticks(x_pos)
ax.set_xticklabels(group_names)

# Set the y-axis range
ax.set_ylim([0, max(data.flatten())*1.1])

# Create a bar chart for each group
bar_width = 0.1
opacity = 0.8
colors = ['r', 'g', 'b', 'y']
label_columns=['CO2e','CO2','CH4','N2O']

# Plot the bars
for i in range(data.shape[1]):
    x_pos = np.arange(data.shape[0]) + (i * bar_width)
    rects = ax.bar(x_pos, data[:, i], bar_width, alpha=opacity, label=label_columns[i])

# Add axis labels and a title
ax.set_xlabel('Frameworks')
ax.set_ylabel('Emission(KG/Hour)')
ax.set_title('GHG Emission For Electricity Generation')

ax.legend(loc='upper right')
# ax.legend()
plt.tight_layout()
# Show the plot
plt.show()
```

```
# GHG Emission For T&D- UK electricity
import numpy as np
import matplotlib.pyplot as plt

# Datas are taken from manually from the excel sheet as the amount is not large
group_names = ['Express', 'Fastify', 'Nest', 'Connect']
data = np.array([[1.371,  1.356  ,0.005 , 0.009 ],
                 [1.379 , 1.365, 0.005 , 0.009],
                 [1.381,1.366 , 0.005 ,0.009 ],
                 [1.369 , 1.355  ,0.005, 0.009]])

# Create a figure and axis
fig, ax = plt.subplots(figsize=(12, 6))
# Set the x-axis ticks and tick labels
x_pos = np.arange(len(group_names))
ax.set_xticks(x_pos)
ax.set_xticklabels(group_names)

# Set the y-axis range
ax.set_ylim([0, max(data.flatten())*1.1])

# Create a bar chart for each group
bar_width = 0.1
opacity = 0.8
colors = ['r', 'g', 'b', 'y']
label_columns=['CO2e','CO2','CH4','N2O']
# Plot the bars
for i in range(data.shape[1]):
    x_pos = np.arange(data.shape[0]) + (i * bar_width)
    rects = ax.bar(x_pos, data[:, i], bar_width, alpha=opacity, label=label_columns[i])

# Add axis labels and a title
ax.set_xlabel('Frameworks')
ax.set_ylabel('Emission(KG/Hour)')
ax.set_title('GHG Emission For T&D- UK electricity')

ax.legend(loc='upper right')
# ax.legend()
plt.tight_layout()
# Show the plot
plt.show()
```

```python
# Graph GHG Emission For Distribution - district heat & steam
import numpy as np
import matplotlib.pyplot as plt

# Datas are taken from manually from the excel sheet as the amount is not large
group_names = ['Express', 'Fastify', 'Nest', 'Connect']
data = np.array([[0.697,0.690 ,0.004, 0.002],
                 [0.701,0.694, 0.004 ,0.002],
                 [0.702,0.695 ,0.004, 0.002],
                 [0.696 ,0.689, 0.004, 0.002]])

# Create a figure and axis
fig, ax = plt.subplots(figsize=(12, 6))

# Set the x-axis ticks and tick labels
x_pos = np.arange(len(group_names))
ax.set_xticks(x_pos)
ax.set_xticklabels(group_names)

# Set the y-axis range
ax.set_ylim([0, max(data.flatten())*1.1])

# Create a bar chart for each group
bar_width = 0.1
opacity = 0.8
colors = ['r', 'g', 'b', 'y']
label_columns=['CO2e','CO2','CH4','N2O']

# Plot the bars
for i in range(data.shape[1]):
    x_pos = np.arange(data.shape[0]) + (i * bar_width)
    rects = ax.bar(x_pos, data[:, i], bar_width, alpha=opacity, label=label_columns[i])

# Add axis labels and a title
ax.set_xlabel('Frameworks')
ax.set_ylabel('Emission(KG/Hour)')
ax.set_title('GHG Emission For Distribution - district heat & steam')
ax.legend(loc='upper right')
# ax.legend()
plt.tight_layout()
# Show the plot
plt.show()
```

A.5 GHG Emission

GreenHouse gas emission values was calculated using the conversion factor.

A	B	C	D	E	F	G	H	I	J	K	L
Electricity generated	Factor	Express	Emission(KG)	T&D- UK electricity	Factor	Express	Emission(KG)	Distribution-district heat & steam	Factor	Express	Emission(KG)
CO2e	0.19338	77.5368	14.99406638	CO2e	0.01769	77.5368	1.371625992	CO2e	0.00899	77.5368	0.697055832
CO2	0.19121	77.5368	14.82581153	CO2	0.0175	77.5368	1.356894	CO2	0.0089	77.5368	0.69007752
CH4	0.0008	77.5368	0.06202944	CH4	0.00007	77.5368	0.005427576	CH4	0.00006	77.5368	0.004652208
N2O	0.00137	77.5368	0.106225416	N2O	0.00012	77.5368	0.009304416	N2O	0.00003	77.5368	0.002326104
Electricity generated	Factor	Fastify	Emission(KG)	T&D- UK electricity	Factor	Fastify	Emission(KG)	Distribution-district heat & steam	Factor	Fastify	Emission(KG)
CO2e	0.19338	78.0084	15.08526439	CO2e	0.01769	78.0084	1.379968596	CO2e	0.00899	78.0084	0.701295516
CO2	0.19121	78.0084	14.91598616	CO2	0.0175	78.0084	1.365147	CO2	0.0089	78.0084	0.69427476
CH4	0.0008	78.0084	0.06240672	CH4	0.00007	78.0084	0.005460588	CH4	0.00006	78.0084	0.004680504
N2O	0.00137	78.0084	0.106871508	N2O	0.00012	78.0084	0.009361008	N2O	0.00003	78.0084	0.002340252
Electricity generated	Factor	Nest	Emission(KG)	T&D- UK electricity	Factor	Nest	Emission(KG)	Distribution-district heat & steam	Factor	Nest	Emission(KG)
CO2e	0.19338	78.1092	15.1047571	CO2e	0.01769	78.1092	1.381751748	CO2e	0.00899	78.1092	0.702201708
CO2	0.19121	78.1092	14.93526013	CO2	0.0175	78.1092	1.366911	CO2	0.0089	78.1092	0.69517188
CH4	0.0008	78.1092	0.06248736	CH4	0.00007	78.1092	0.005467644	CH4	0.00006	78.1092	0.004686552
N2O	0.00137	78.1092	0.107009604	N2O	0.00012	78.1092	0.009373104	N2O	0.00003	78.1092	0.002343276