

**Using Prolog and JavaScript to query a knowledge set to enhance web income tax
calculator**

Eduardo Andrés Castillo Perera | A01702948

Instituto Tecnológico de Estudios Superiores de Monterrey

TC2006.1: Programming Languages

Dr. Benjamin Valdes Aguirre

November 24, 2020

Table of Contents

Abstract	4
Using Prolog and JavaScript to query a knowledge set to enhance web income tax calculator.....	5
Context of the problem	5
Social Problem	5
Process to compute the income tax using Income Tax Tables.	5
Technical Problem	8
Key Concepts	9
CDN.	9
Firebase Hosting.	9
Income Tax.....	9
Income Tax Bracket.	10
Programming Paradigm.	10
Logical Programming Paradigm.	10
Tau-Prolog.....	10
Solution.....	11
Paradigm and language selection.....	11
Architecture of the application.....	12
Results.....	14

Deployment.....	14
Testing.....	15
Usability testing	15
Functionality tests	16
Setup and Execution	17
Evidence.....	17
Conclusion	18
References.....	19

Abstract

This project uses the Logical Programming Paradigm and the Prolog Programming Language to aid an HTML/JavaScript application in querying a modest size knowledge base to calculate Income Taxes for employees in Mexico and the United States. It shows how these two languages, with totally different approaches to structuring code, can interoperate to achieve a common goal. This paper will cover the architecture of such application and the tests that were performed on it. This JS application beneficiates greatly from the Prolog Programming Language and the Logical Programming paradigm because it makes it easier to acquire some values from an expandible knowledge base so it can use the for a calculation. Prolog can also encode the logic behind Income Tax calculation very easily, making it ideal for this type of applications. JavaScript and HTML are well understood by most modern browsers, thus reaching the biggest amount of people. The combination between the 2 gives us a powerful and scalable application.

Keywords: Income Tax, Logic, Knowledge base, Prolog, JavaScript, multiparadigm.

Using Prolog and JavaScript to query a knowledge set to enhance web income tax calculator

Prolog and JavaScript are two completely different languages. One uses logical programming, a knowledge base and facts to store information, and the other uses objects, functions and loops to store data and execute instructions, not to mention it was created primarily to be executed inside a web browser. They could not be more different. Yet, by combining the two of them, we can perform interesting calculations, by using each other's strengths, we can make very useful applications. This paper will discuss how we can use both languages, and two very distinct programming paradigms, to create an Income Tax calculator. The application will use Prolog's ability to search through big knowledge bases to find the appropriate values to calculate taxes, and execute those calculations, and JavaScript will not only provide interactivity, but also interoperability with modern browsers so it can easily be used by thousands of users without the need to perform complicated installs, thus enhancing the user experience.

Context of the problem

The problem requires us to calculate the Income Tax of a user given his or her net salary and how often is he or she paid. The application also takes into consideration whether the user is in Mexico or the United States. I decided to add a basic calculation for the US in order to explore how dynamic can the application be and demonstrate the strength of Prolog and its ability to search through large sets of facts. In a more informal sense, to "*show off*".

Social Problem

Process to compute the income tax using Income Tax Tables.

The income tax (and the amount of money a person can keep) is obtained by using Income Tax Tables to locate an appropriate bracket for a certain income, and then using data

from that bracket with a formula to obtain the result. It can be better visualized with a concrete example:

Suppose Maria has a salary of \$30,000.00 pesos a month, and is paid twice a month, every 15 days, on a biweekly basis. She is then paid \$15,000.00 pesos every 2 weeks, or “quincena” as she says. By law, her employer is required to withhold a certain amount of money to pay her income tax. We will be using the 2020 Biweekly Income Tax table, located in this document. In the biweekly table we can see there is a bracket for incomes greater than \$ 11,951.86 and lesser than \$18,837.00. Given she earns \$15,000.00 pesos on a biweekly basis, she belongs in that bracket. The fixed quota for that bracket is \$1914.75 pesos, and the marginal tax rate is 23.52%. This means she must pay 23.5% of the money that exceeds the lower limit. Thus, her income tax can be computed the following way:

Net Salary: \$15000.00

Lower Limit: \$11,951.86

Difference: $\$15000.00 - \$11,951.86 = \$3048.14$

Percentage over difference: 23.52%

Marginal Tax: $23.52\% \times \$3048.14 = \716.92

Fixed Quota: \$1914.75

Income Tax: $\$1914.75 + \$716.92 = \$2631.67$

Now that her income tax has been calculated, her employer will withhold that amount in order to pay it to the government. The rest of the money will be paid to Maria. The algorithm can be visualized this way:

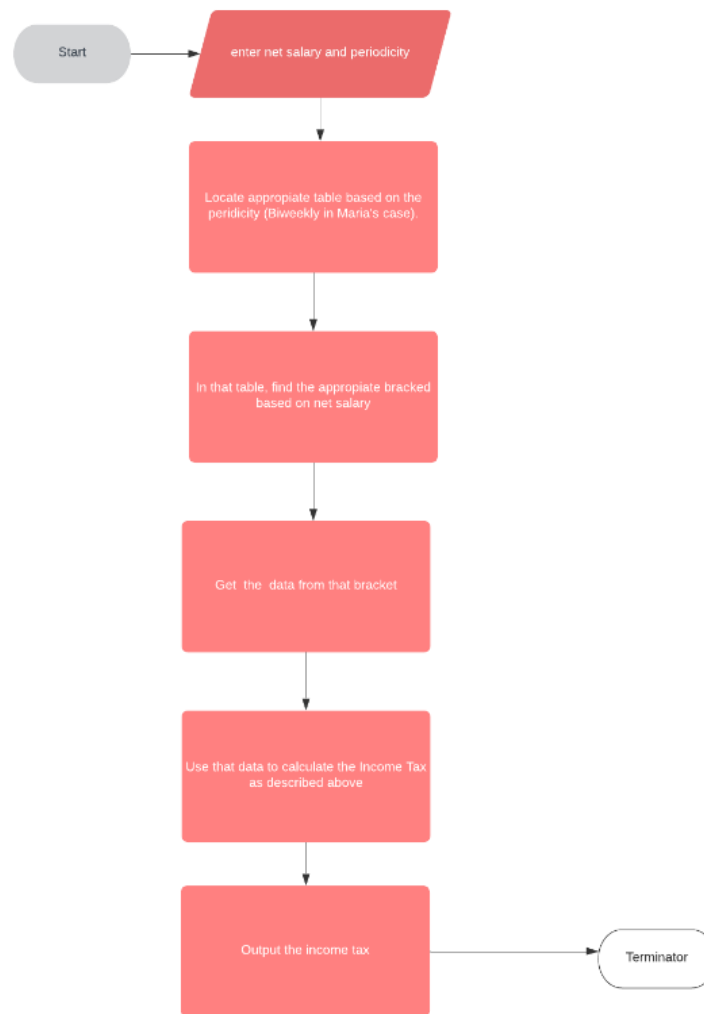


Figure 1.1. Visualization of the algorithm to calculate income tax by using tables. In this diagram, the focus is the acquisition of the data from the appropriate bracket.

In other words, to calculate the income tax, you need to get the appropriate bracket's data, and then you use simple addition, subtraction and multiplication operations with that data and the net income. The process described above applies is for Mexico. However, many countries have a

similar procedure to calculate income tax, although some have more frequencies than others. (KPMG International, 2020).

It is worth noticing Maria's salary is still \$30,000.00 a month, even though she is paid \$15,000.00 twice a month. Because the income tax must be withheld from each payment, it is important to choose the income tax table correctly.

Technical Problem

As it can be observed in the algorithm, the first step is locating a specific bracket between not one, but multiple income tax tables, each tax table for a different frequency. Nonetheless, if we want to calculate historical data (example: how much money should company x has withhold from a person earning \$45,000.00 a month in 2018?) for an audit or any other purposes, we also need to keep tables from the past. This can increase the search space notoriously. And if we decide to add support for multiple countries or cities with different tables for each one. This, in turn, can increase the size of the knowledge base very rapidly, making the operation of finding the correct bracket in order to use its data even more difficult.

This search and compute problem can be solved easily with the Logical Programming Paradigm. A set of facts can be written into a knowledge base, and we can create a predicate to locate the individual bracket we need (if it exists). It also has the benefit of being very scalable in the sense it can add more data over time, without modifying the core algorithm.

Another challenge could be interactivity and compatibility because most people do not have prolog installed or know how to use it. A web application connect or invoke a prolog script to quickly obtain the calculations it needs in order to display them in a clean and clear manner to the end user.

Key Concepts¹

Before continuing with the solution, I would like to explore some key concepts that will be useful to understand the document.

CDN. Stands for Content Delivery Network. It is used to accelerate the delivery of files and content to the end user by caching the content in many locations around the globe, so files and resource can be closer to the end user, thus increasing loading speeds. This project uses a CDN to load Tau Prolog, the library that allows us to execute prolog code in the browser, as well as other style libraries to enhance the user interface.

Firebase Hosting. It is a service offered by Google, that acts as a static web server in which we can store static websites, applications and files. It has gained a lot of popularity with the increase usage of *Single Page Applications*. As defined by Google: “*Firebase Hosting is production-grade web content hosting for developers. With a single command, you can quickly deploy web apps and serve both static and dynamic content to a global CDN (content delivery network).*” (Google Inc. 2020). This project uses Firebase Hosting to host and serve the applications to the users because of its simplicity. However, almost every web hosting would work fine.

Income Tax. Cambridge Dictionary defines income tax as “*A tax that you have to pay on your income, usually higher for people with larger incomes*” (Cambridge Dictionary, 2020). Most countries have some sort of income tax that their citizens and residents are required by law

to pay. Mexico and the US are two examples of such countries. The application proposed in this paper calculates this number given the net income of a person, and the frequency of the payment, for employees in Mexico and the US.

Income Tax Bracket. Cambridge Dictionary defines income tax bracket as “*You can avoid some of the tax burden by waiting until you're in a lower income tax bracket in retirement before you withdraw the funds*” (Cambridge Dictionary, 2020). Generally, the percentage of your net income that you must pay to the government varies depending how the actual net income, and the government distributes said percentage in different ranges, along with a minimum amount to pay. (KPMG, 2020). This application uses prolog to query a knowledge base containing this bracket and get the necessary data to calculate the actual income tax, based on input from the user.

Programming Paradigm. According to Raymond J, a programming paradigm is a *style* of programming. Some languages tend to be better at certain style than others. Some others force you to program in a certain way. (Raymond J. n. d). This application uses two programming paradigms in 2 very different languages and explores how they can come together to achieve something great, by each using its strengths to reach a common goal.

Logical Programming Paradigm. According to Raymond J, (Raymond J. n. d). This application uses this paradigm to query a knowledge base and acquire necessary data to perform a series of mathematical operations in order to calculate the income tax of a person.

Tau-Prolog. A JavaScript library that acts as a prolog interpreter. It can load prolog scripts from an URI and can make queries to that script. It can also call predicates from that script. This application uses Tau Prolog to run Prolog code inside the web browser and link the

JavaScript code that controls the interface with the prolog code, which uses a predicate to make the income tax calculation.

Solution

A web application that combines the power of prolog to quickly search through all the tables to locate the appropriate tax bracket and compute the income tax, in order to later pass the data to JavaScript so it can present it to the user in a useful way is the solution this paper proposes. JavaScript can provide interactivity and handle the IO, while prolog can quickly perform the search of the tax bracket and further calculation of the income tax.

Paradigm and language selection¹

Logical programming was an easy choice for this type of problem because it excels at this kind of problems. In this paradigm the developer can establish a knowledge base by creating a series of *facts*, and then query that knowledge base to obtain data. Using this approach, I was able to construct a long knowledge base, with the daily, weekly, 10-day, biweekly, monthly and yearly income tables from the years 2017, 2018, 2019 and 2020 for the Mexican income tax, and the yearly table from the years 2019 and 2020 for the federal income tax of the United States. I decided to add these last tables in order to explore the flexibility of Prolog and my application.

I am also using JavaScript and the Object-Oriented Programming paradigm to manipulate the UI in a web browser, in order to receive and validate user input, pass the input to Prolog using a special library, and then manipulate the UI again to show the output of the prolog computation. Another advantage of using JS for the UI is that pretty much every browser comes pre-installed with it, allowing me to provide the application with many users, potentially thousands.

JS also uses Functional Programming because it allows us to use functions as first-class citizens and use them as arguments for other functions. This proved in handy because Tau Prolog, the library that allows me to execute Prolog code in the browser, works asynchronously, and requires the usage of Callback Functions. In JS I can easily create a callback function and pass it to the library. The library executes the callback function after the Prolog query has finished. This is beneficial because by using this approach, JS can keep processing the UI and reacting to user changes. This approach makes the Tau Prolog, and many other libraries, to behave in a non-blocking fashion. (Madsen, M. 2017)

Architecture of the application

The application consists of 3 files: index.html, bridge.js and knowledge-base.pl. It uses a library named Tau Prolog to execute Prolog code in the browser. When the application is loaded, bridge.js creates a Tau Prolog Session and instructs Tau Prolog to load knowledge-base.pl. When the file is loaded, Tau Prolog parses the file and creates a Prolog Session. The session is loaded with the knowledge base. At this point, bridge.js can take react to user input, validate it, build a Prolog Query that uses a Predicate and then send that query to Tau Prolog. Tau Prolog will execute it and when it is done, will execute a callback function. This callback function is also passed to Tau Prolog on startup.

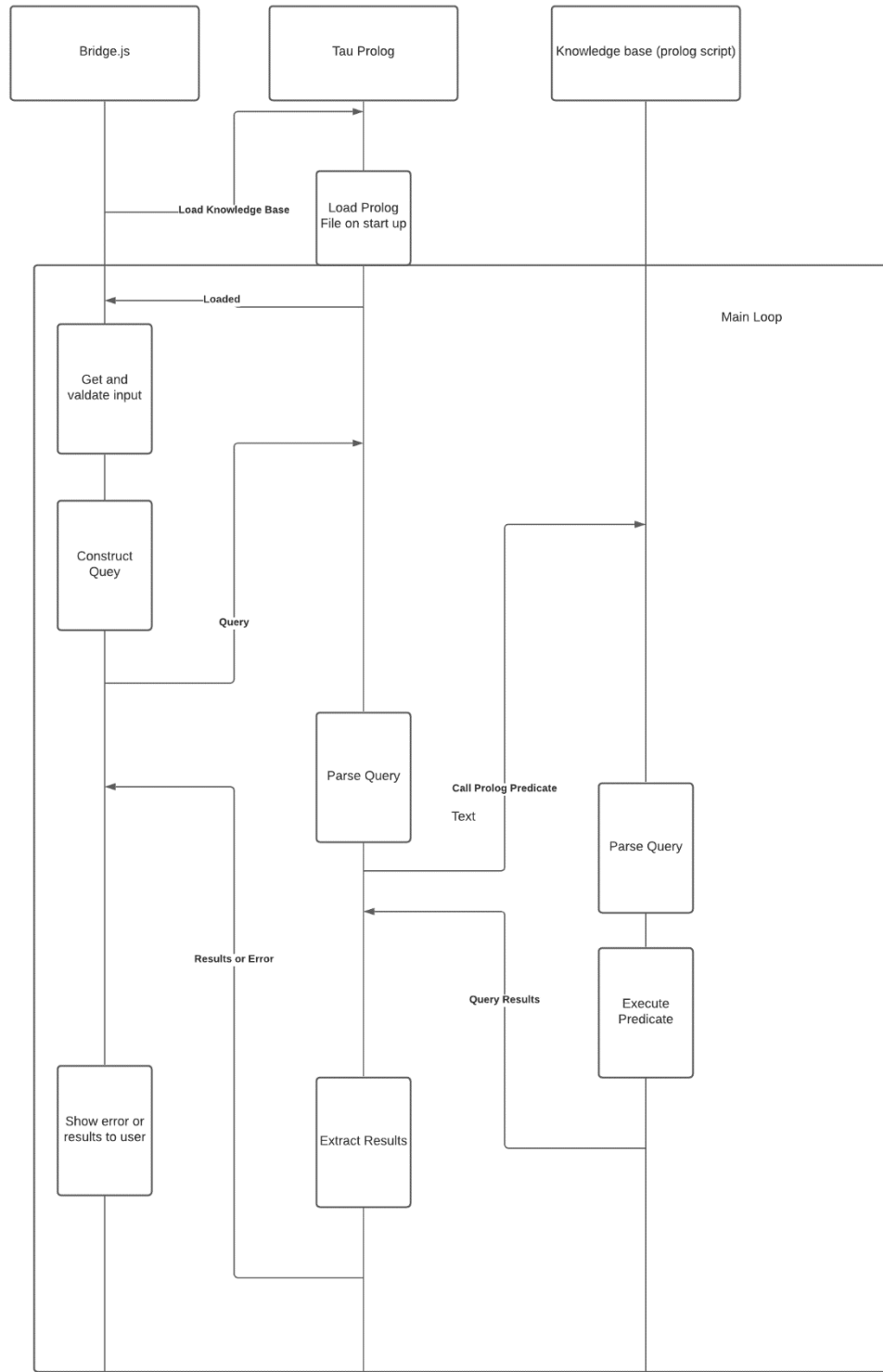
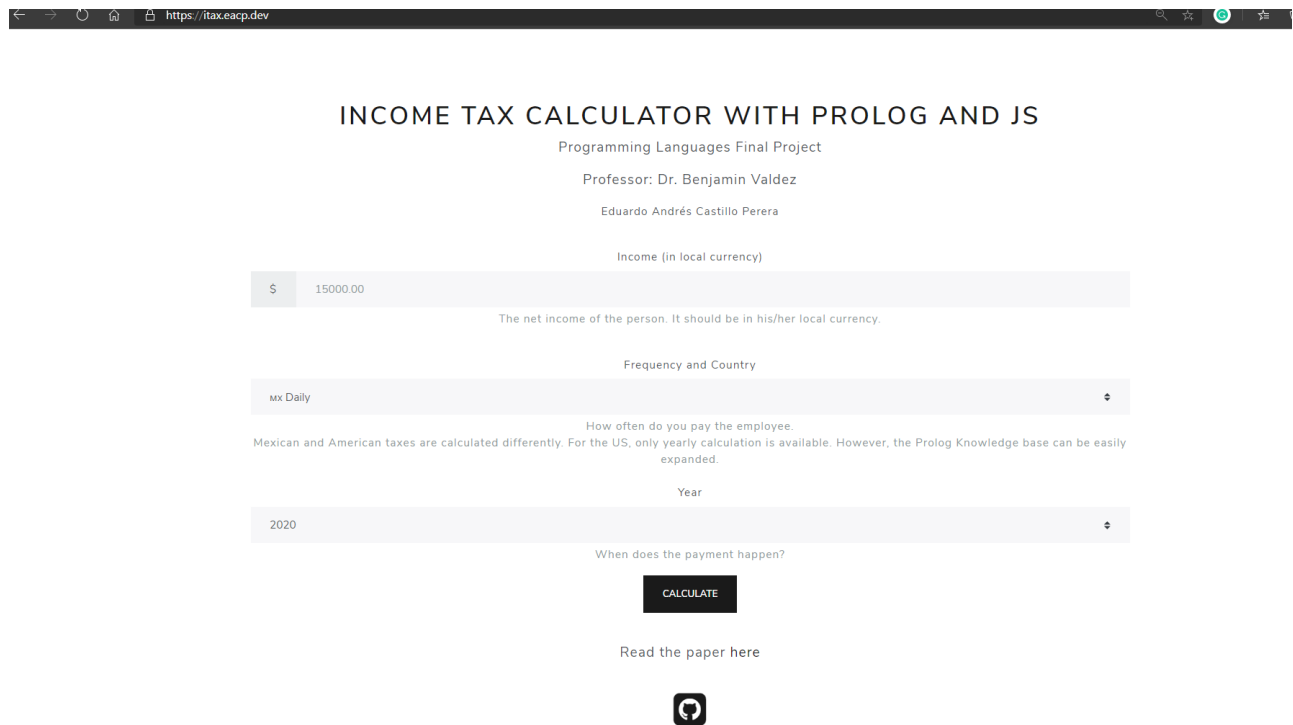


Figure 1.2. Diagram showing how the different files (and programming languages) interact with each other.

Results

This project resulted in an application that can take user input from the browser and execute a prolog predicate to make a computation. The final application looks like this:



The screenshot shows a web browser at the URL `https://itax.eacp.dev`. The page title is "INCOME TAX CALCULATOR WITH PROLOG AND JS". Below the title, it says "Programming Languages Final Project", "Professor: Dr. Benjamin Valdez", and "Eduardo Andrés Castillo Perera". The main form has three input fields: "Income (in local currency)" with a value of "\$ 15000.00", "Frequency and Country" with a dropdown menu showing "mx Daily", and "Year" with a dropdown menu showing "2020". Below these fields, there is a "CALCULATE" button. At the bottom, there is a link "Read the paper here" and a GitHub icon.

Figure 1.2. A screen shot of the running application. You select a period, input the data and click calculate.

Deployment¹

Given the application has no “server side” code, it can be deployed as a static application. I can just upload it to Firebase by using the Firebase CLI:

Firebase deploy.

Testing

I performed two kinds of testing: Usability Tests and *Correctness test*. The first one verifies users can use the application properly, and the second one checks the application is giving correct results.

Usability testing

I tested the application with 6 friends to see how intuitive the application is. To do this, I sent them the URI <https://itax.eacp.dev/> and asked them to calculate some taxes using some frequencies. I collected their feedback once and then iterated over said feedback to build a better version of the UI

Table 1

Usability Tests

Name	User Feedback on first iteration	User Feedback on Second Iteration
Raymundo Romero Arenas	It is a little bit confusing. Also, it takes a lot to load.	The new helper text is useful now. Although I would like if you could use ALL years with the US
Violeta Curiel	I see this is very useful, I might use it for my class, but it takes too long to load	456
Alejandro Ramirez	The colors are nice and look like money indeed. However, I think the letter size is a little bit small.	I can see better, and it loads faster.
Laura Gonzales	It is nice. I would like it if you could make the font bigger.	Thanks for making the letter size bigger. I see there are some speed improvements.

Fernanda Gutierrez	Maybe add some emojis to the flags?	It looks cute now
Edgar Delgado	I like the fonts	No comment

In this table I listed the feedback of some sample users. I asked for feedback twice to see if the UI improved. Using this approach, I can iterate over my previous design and make a better user experience.

Something I noticed from the feedback is that some resources (notably the font files and Tau Prolog) were taking too long to download. After doing some research, I realized I could use a CDN to offload these files from my Firebase Hosting Account, given it is the free one and I do not have the full capabilities of paid Firebase Hosting). I used the service CDNJS, which is offered by the company Cloudflare. Using that service, I was able to replace the tags in my HTML file that specified the location of the dependencies. Now TauProlog, Font Awesome and the styles I used for my application load faster, and I only must host the application itself.

Functionality tests

The application would be worthless if it did not manage to compute the correct results, so I created 10 test cases. In each case, I calculated the result by hand and compared it to the application result.

Table 1

Usability Tests

Income	Frequency	Year	Expected tax	Application calculated tax	Status
--------	-----------	------	--------------	-------------------------------	--------

15000	Monthly.	2017	2094.5	2094.880304	Success. The error margin is too small.
30000	Monthly	2020	5239.0	5239.350335999995	Success. The error margin is too small.
50000	US yearly.	2019	6858.5	6858.497799999999	Success. The error margin is too small.
500000	Yearly.	2019	2888.98	2888.9785600000005	Success. The error margin is too small.
120000	US Yearly	2020	22879.498	22879.4976000000002	Success. The error margin is too small.
Status	All Passing				

From this table we can see all tests are passing

Setup and Execution

The application can be accessed via this URI: <https://itax.eacp.dev/>

It can also be cloned from github:

git clone <https://github.com/eacp/programming-lang-project-prolog>

And then you just have to run an http static server. Although the preferred way is to execute it online. There are no special build steps, and that is on purpose: I intended the application to be as easy to run as possible.

Evidence

The source code of the project can be found here: <https://github.com/eacp/programming-lang-project-prolog>

The application can be used by everyone with a modern browser by going here:

<https://itax.eacp.dev/>

Conclusion

In this project, I learned and demonstrated that many programming paradigms, are not in conflict, but can rather complement each other. It was amazing to see how does prolog make the search of data so much easier, and JavaScript allowed the application to run everywhere without the need of complicated installs.

This project will surely leave an impact in some people because some of my friends, which happen to be accounting students, say they loved it and will surely serve as a reference.

The project is Open Source and is published under the GNU License.

References

- Peng, G. (2004). CDN: Content distribution network. *arXiv preprint cs/0411069*.
- Moroney, L. (2017). Using Firebase Hosting. In *The Definitive Guide to Firebase* (pp. 93-106). Apress, Berkeley, CA.
- Cambridge Dictionary (n.d.). Citation. In *dictionary.cambridge.org dictionary*. Retrieved November 21, 2020, from <https://dictionary.cambridge.org/dictionary/english/income-tax>
- Cambridge Dictionary (n.d.). Citation. In *dictionary.cambridge.org dictionary*. Retrieved November 21, 2020, from <https://dictionary.cambridge.org/dictionary/english/income-tax-bracket>.
- KPMG International (2020). *Individual Income Tax Rates Table*. Retrieved November 22, 2020 from <https://home.kpmg/xx/en/home/services/tax/tax-tools-and-resources/tax-rates-online/individual-income-tax-rates-table.html>
- Raymond J. Toal from Loyola Marymount University, Programming Languages course chapter: Programming Paradigms <http://cs.lmu.edu/~ray/notes/paradigms/>
- Madsen, M., Lhoták, O., & Tip, F. (2017). A model for reasoning about JavaScript promises. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA), 1-24.
- Diario Oficial de la Federación. (2020) *ANEXOS 3, 5, 7, 8 y 11 de la Resolución Miscelánea Fiscal para 2020, publicada el 28 de diciembre de 2019*. Retrieved from: [DOF - Diario Oficial de la Federación](#)

Diario Oficial de la Federación. (2019) *ANEXOS 3, 5, 7, 8 y 11 de la Resolución Miscelánea Fiscal para 2020, publicada el 28 de diciembre de 2018*. Retrieved from: [DOF - Diario Oficial de la Federación](#)

Diario Oficial de la Federación. (2018) *ANEXOS 3, 5, 7, 8 y 11 de la Resolución Miscelánea Fiscal para 2020, publicada el 28 de diciembre de 20*. Retrieved from: [DOF - Diario Oficial de la Federación](#)

Diario Oficial de la Federación. (2017) *ANEXOS 3, 5, 7, 8 y 11 de la Resolución Miscelánea Fiscal para 2020, publicada el 28 de diciembre de 2016*. Retrieved from: [DOF - Diario Oficial de la Federación](#)