

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



**GRADO EN INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN**

TRABAJO FIN DE GRADO

**DESIGN AND DEVELOPMENT OF A BROWSER
PLUGIN FOR ANALYSING MORAL EMOTIONS BASED
ON INTELLIGENT TEXT ANALYTICS**

**PABLO FERNÁNDEZ FRAILE
JULIO 2024**

TRABAJO DE FIN DE GRADO

Título: Diseño y desarrollo de un plugin de navegador para analizar emociones morales basado en análisis inteligente de texto

Título (inglés): Design and Development of a Browser Plugin for Analysing Moral Emotions based on Intelligent Text Analytics

Autor: Pablo Fernández Fraile

Tutor: Carlos Ángel Iglesias Fernández

Departamento: Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente: —

Vocal: —

Secretario: —

Suplente: —

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN**

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes



TRABAJO FIN DE GRADO

**DESIGN AND DEVELOPMENT OF A
BROWSER PLUGIN FOR ANALYSING
MORAL EMOTIONS BASED ON
INTELLIGENT TEXT ANALYTICS**

Pablo Fernández Fraile

Julio 2024

Resumen

A día de hoy, el mundo de Internet ha evolucionado de manera considerable, volviéndose la herramienta más utilizada por el ser humano. Esto ha llevado a la creación de una gran cantidad de información y datos que se encuentran en el mundo de Internet. Surge por ello la necesidad de entender toda esa información y poder analizarla. Así se empezaron a desarrollarse herramientas de análisis de texto para poder conocer las necesidades de los usuarios y poder utilizar esa información a tu favor.

El principal objetivo del proyecto es crear una extensión de análisis de la moralidad en los textos y que se pueda utilizar en cualquier navegador web. Gracias a esto, cualquier usuario podrá analizar todo tipo de texto en Internet y obtener una representación clara y visual de esos datos. Esto se hará con distintos tipos de gráficos que el usuario podrá personalizar de forma dinámica según sus necesidades. La capacidad de analizar la moralidad permitirá obtener información destacable acerca de los usuarios de Internet.

La personalización va a ser un factor clave en el proyecto, ya que el usuario va a ser capaz de tener una capacidad de elección muy amplia, pudiendo elegir cosas tan básicas como el tipo de análisis a realizar, así como el estilo de gráfico en el que quiere que se muestren los distintos resultados que se obtienen en la extensión. Por lo tanto, esta y otras funcionalidades se van a implementar para mejorar la experiencia de usuario a niveles muy altos. Además, se permitirá al usuario guardar configuraciones personalizadas para un uso recurrente. También se implementará una interfaz intuitiva y fácil de usar para que cualquier persona, independientemente de su nivel técnico, pueda aprovechar al máximo la extensión.

Finalmente, introduciremos las conclusiones de este proyecto y su finalidad como aplicación. También definiremos unas líneas futuras de trabajo sobre las implementaciones que se pueden hacer más adelante.

Palabras clave: análisis de texto, extensión, análisis de moralidad, navegador, personalización

Abstract

Today, the Internet has evolved considerably, becoming the most commonly used tool for humans. This has led to the creation of a huge amount of information and data on the Internet. The need to understand and analyze this information has arisen. Text analysis tools were developed to understand users' needs and use this information to their advantage.

The main objective of the project is to create a text morality analysis extension that can be used in any web browser. Thanks to this, any user can analyze any text online and obtain a clear and visual representation of these data. This will be done with different types of graphics that the user can dynamically customize according to his needs. The capacity to analyze morality will allow us to obtain remarkable information about Internet users.

Customization is going to be a key factor in the project, as the user will be able to have a very wide capacity of choice, being able to choose such basic things as the type of analysis to be performed, as well as the style of the graph in which they want the different results obtained in the extension to be displayed. Therefore, this and other functionalities will be implemented to improve the user experience to very high levels. In addition, the user will be allowed to save customizations for recurring use. An intuitive and user-friendly interface will also be implemented so that anyone, regardless of their technical level, can take full advantage of the extension.

Finally, we will introduce the conclusions of this project and its purpose as an application. We will also define future work lines on the implementations that can be done further.

Keywords: text analysis, extension, morality analysis, browser, customization

Agradecimientos

A mis padres y mis hermanos, por haberme acompañado en este largo camino y apoyado en cada decisión que tomaba, sin importar cuál fuese.

A mi grupo de la universidad, por ser mi mayor ayuda y apoyo, sin ellos no hubiese llegado hasta aquí.

A Fer, por haber estado ahí en todo momento que lo he necesitado y por ser la persona más generosa que conozco.

A mi tutor Carlos, por haber confiado y creído en mí aún cuando yo no lo hacía.

Contents

Resumen	I
Abstract	III
Agradecimientos	V
Contents	VII
List of Figures	XI
1 Introduction	1
1.1 Context	1
1.2 Project goals	2
1.3 Structure of this document	2
2 Enabling Technologies	3
2.1 Introduction	3
2.2 React	3
2.3 Plasmo Framework	4
2.4 Senpy	5
2.5 Chrome extensions	6
2.6 Chrome API	7
2.7 Tailwind.css	8
2.8 Typescript	8

3	Sentiment analysis extensions	11
3.1	Introduction	11
3.1.1	Sentitude Sentiment Analysis extension	11
3.1.2	Youtube Sentiment Analysis	12
3.1.3	Sentiment&emotion Analysis - GSI	14
4	Architecture	17
4.1	Introduction	17
4.2	Environment creation	18
4.3	Environment development	19
4.3.1	Popup	20
4.3.1.1	React Hooks	20
4.3.1.2	Analysis Type Props	21
4.3.2	Charts	23
4.3.2.1	Charts implementation	23
4.3.2.2	Plasmo import resolution	25
4.3.3	Background	26
4.3.4	Options	27
4.3.5	Tailwind implementation	28
4.4	Packaging and publication	29
4.4.1	Packaging	29
4.4.2	Publication	31
4.4.2.1	Mozilla publishing	32
4.4.2.2	Microsoft Edge publishing	32
4.4.2.3	Chrome publishing	32
5	Case study	35

5.1	Introduction	35
5.2	Download and installation	35
5.3	Moral Analysis	36
5.4	Configuration Page	39
5.5	Using Firefox and Edge	42
5.6	Analysis comparative: Right-wing vs Left-wing	44
6	Conclusions and future work	47
6.1	Conclusions	47
6.2	Achieved goals	48
6.3	Future work	49
	Appendix A Impact of this project	i
A.1	Social impact	i
A.2	Economic impact	ii
A.3	Environmental impact	ii
A.4	Ethical implications	ii
	Appendix B Economic budget	iii
B.1	Physical resources	iii
B.2	Project structure	iii
B.3	Human resources	iv
B.4	Taxes	iv
	Bibliography	v

List of Figures

2.1	Plasmo Framework Logo [28]	5
2.2	Senpy's architecture [32]	6
2.3	Examples of Chrome extensions [7]	7
2.4	Example of using Tailwind.css [18]	8
2.5	Typescript Logo [15]	9
3.1	Sentitude Extension [4]	12
3.2	Extension Settings [4]	13
3.3	Youtube Sentiment Analysis Extension [39]	13
3.4	Sentiment Analysis [39]	14
3.5	GSI-Sentiment&Emotion [11]	15
3.6	Configuration [11]	15
3.7	Emotion Display [11]	16
4.1	Extension Creation	18
4.2	Architecture Overview	19
4.3	Flow Diagram	22
4.4	Import from react-chartjs-2	23
4.5	POST Request Architecture	27
4.6	Extension Packaging	31
4.7	Mozilla Publication	32
4.8	Edge Publication	33

4.9	Chrome Publication	33
5.1	Chrome Store	36
5.2	Splash Screen	36
5.3	Context Menu	37
5.4	Moral Analysis	37
5.5	Sentiment Analysis	37
5.6	Emotion Analysis	38
5.7	Settings Button	39
5.8	Configuration Page	40
5.9	Chart Type	40
5.10	Moral Bar Chart	41
5.11	Moral Polar Area	41
5.12	Theme Selection	42
5.13	Dark theme	42
5.14	Mozilla Firefox Analysis	43
5.15	Microsoft Edge Analysis	43
5.16	The Mirror Analysis	44
5.17	The Spectator Analysis	45

Introduction

1.1 Context

In today's framework, anything related to understanding what people want to express in an easier way is relevant in the 21st century. There is a gigantic amount of data in text and image format on the internet. According to the latest estimates, 328.77 million terabytes of data are created every day [12]. This has led to the need to be able to analyze this information in order to benefit from it in certain aspects. Text analysis and interpretation are very useful tools for individuals and companies. For a user, it can be important to see the reviews of a shop or a service in a quick and easy way. This is why Customer Intelligence Management Platforms [3] have been created, such as Feedier, which offers a platform that allows companies to improve competitiveness and strengthen customer loyalty with tools such as data analysis and data centralization. Many of them are already being implemented with artificial intelligence, such as IBM Watson Natural Language Understanding [17] or with machine learning, such as MonkeyLearn [26].

Therefore, it is a key factor to know the feelings, morals, and emotions of any Internet user to understand their purpose in all areas of life. In this project, we are going to focus on the creation of an extension to analyze all these feelings so that we can obtain results

that are useful for everyone.

1.2 Project goals

The purpose of this project is to create an extension that analyzes the morality and sentiment of any text on the Internet. The extension will then be able to show the results dynamically, clearly, and concisely.

- Design an extension that is able to analyze the morality, feelings, and emotions of a text.
- Implement a comprehensive and dynamic display module to be able to show the results faithfully.
- Design a configuration page so that the user can choose from multiple options for using the extension.
- Publish the extension so that it can be used by any user.

1.3 Structure of this document

This section provides a brief overview of the chapters included in this document. The structure is as follows:

Chapter 1 provides a short introduction to what is to be done in the project and the proposed objectives.

Chapter 2 deals with the various technologies that have been used for the project, as well as an explanation of each of them.

Chapter 3 shows an investigation of some chrome extensions on sentiment analysis to acquire concepts and use them in the project.

Chapter 4 describes the project's architecture, dividing it into each module, how they are implemented, and related to each other to create the extension.

Chapter 5 consists of the use case of a user installing and using the extension in a browser.

Chapter 6 summarizes the purpose of the project and provides a number of conclusions, as well as an outline for future work.

Enabling Technologies

2.1 Introduction

The programming industry is currently experiencing rapid development and growth. As technology advances, programming languages constantly seek new methods to simplify and accelerate the creation of web technologies.

In this chapter, we will focus on exploring the latest technologies that can help us achieve our goal of discovering new ways to harness technology's potential.

2.2 React

React.js [13], commonly called React, is a popular JavaScript library for creating user interfaces. In a React web application, reusable components compose the different parts of the interface. This component-based approach simplifies development by reducing repetitive code. Instead, we create the component's logic and subsequently import it into any necessary part of the code.

The React library plays a fundamental role in web application development. It facilitates

a component-based approach, allows efficient and secure application state management, and surpasses JavaScript’s capabilities.

React functions using the concept of “state”, which manages data that change over time. Each component can have its state, and React is responsible for rendering and updating the information on the components being used while the data changes. This is made possible through an algorithm called the virtual Document Object Model (DOM), which allows us to update only the necessary changes instead of rendering the entire DOM of the page. This approach makes us more effective.

Another feature of React is its ecosystem, which includes some tools such as Redux for advanced application management and React Router to manage navigation in different applications. On the other hand, React has also evolved to support the creation of mobile applications through React Native [24], which is a commonly used extension to develop native applications for IOS and Android using the same approach and syntax based on components as React.

2.3 Plasmo Framework

This project uses Plasmo [8], a platform designed to facilitate the development of browser extensions while creating and publishing them. Plasmo Framework is an Software Development Kit (SDK) for extensions that simplifies the code and its structure when creating web extensions. It eliminates the need for configuration files and other technical details that slow development. This framework promotes easy development as developers only need to write a file and export a component. Plasmo will do the rest of the work, assembling and packaging it. Plasmo can be combined with “Itero TestBed”, a testing environment that allows developers to test their extension in a real environment before releasing it.

Another advantage of Plasmo is that it supports technologies such as TypeScript [25], React, Svelte, or Vue, among others, which allows for building more complex extensions. It also integrates a live reload engine that lets you see the changes made instantly without manually updating the extension. Furthermore, thanks to Plasmo, the file required by an extension, manifest.json, is generated automatically.



Figure 2.1: Plasmo Framework Logo [28]

2.4 Senpy

Senpy [33] is a framework designed to create sentiment and emotion analysis services. It was developed at Universidad Politécnica de Madrid (UPM) as a tool to help simplify the integration of different text analysis models. This framework offers the possibility to request analyses from several providers using the same interface, allowing the integration of all sentiment and emotion analysis models.

Using the same interface allows users to use different systems with the same Application Programming Interface (API) and tools simply by changing a parameter in the Uniform Resource Locator (URL) you request the analysis you need. This also helps to evaluate the performance of different algorithms and services more efficiently, as Senpy also has an integrated evaluation API that can be used to compare the results with other technologies. Another advantage of Senpy is that it is a system that supports various models of sentimental and emotional data, such as Polarity, which ranges from -1 to 1, allowing you to see the intensity of emotions in different words or texts.

Senpy's architecture, as shown in the Fig. 2.2, is divided into several distinct parts that work together to provide a complete and functional system. Its architecture is structured as follows:

- **Parameter Extraction:** The architecture starts with an NLP Interchange Format (NIF) HTTP query. The web user interface and the Command Line Interface (CLI) are designed to extract input parameters used to run the plugins. Users enter their requirements through these interfaces.
- **Parameter Validation:** To ensure the data is accurate, we validate the data entered to see if it can be used correctly.
- **Plugins:** After validation, the plugins are executed with the parameters and here the sentiment or emotion analysis specified is done.

- **Linked data and Mapping:** Senpy uses an interchangeable data approach, meaning that the analysis results can be integrated and connected with other resources on the data web. It also allows conversion to different formats and models, facilitating interoperability.
- **Serialization, Formatting and Validation:** Afterwards, the data is converted into a standardized format for transmission or storage.
- **Visualization:** Finally, we make a display of the content either in the web user interface or in the CLI.

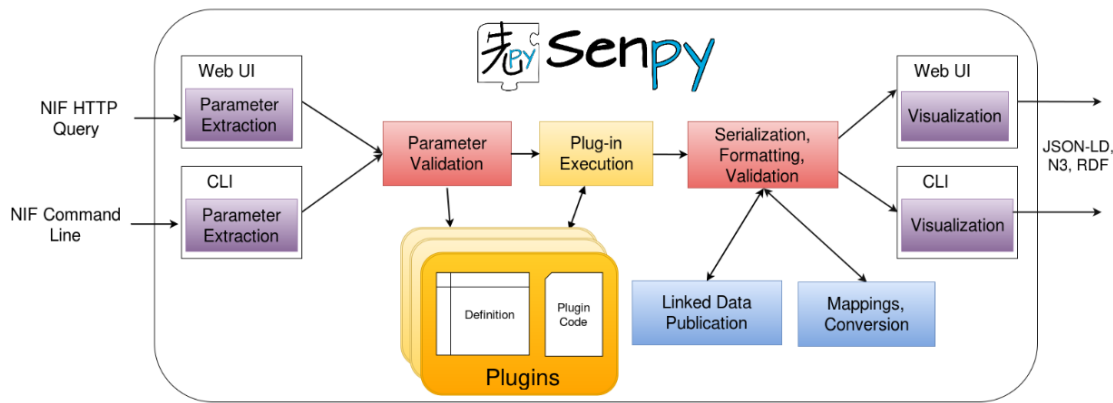


Figure 2.2: Senpy's architecture [32]

2.5 Chrome extensions

An extension is a software program that helps you in your browsing experience. The purpose of an extension is to fulfill the purpose required by the user in a precise and understandable way. The most important features are as follows:

- They allow Chrome's behavior to adapt to the user's needs. Its use is based on technologies such as HTML, JavaScript, or CSS.
- Chrome extensions include background and content scripts. The first one allows constant actions in the browser while the content script executes when web pages are loaded. On the other hand, the popup file is the one that shows the interface of the extension that is displayed when you click on it.

- Google has a set of APIs for extensions to interact with Chrome and its components. In addition, we can perform visualization tasks such as modifying the navigation bar or content within the web page itself.
- We need mainly a manifest.json file. All Chrome improves the user experience. Making the browser more efficient and functional has become useful in today's Plasmio, and all necessary permissions are set in the package.json.

There are thousands of extensions available, some may be to help you plan your travels and there are others to help you when you are learning a language as shown in Fig. 2.3.

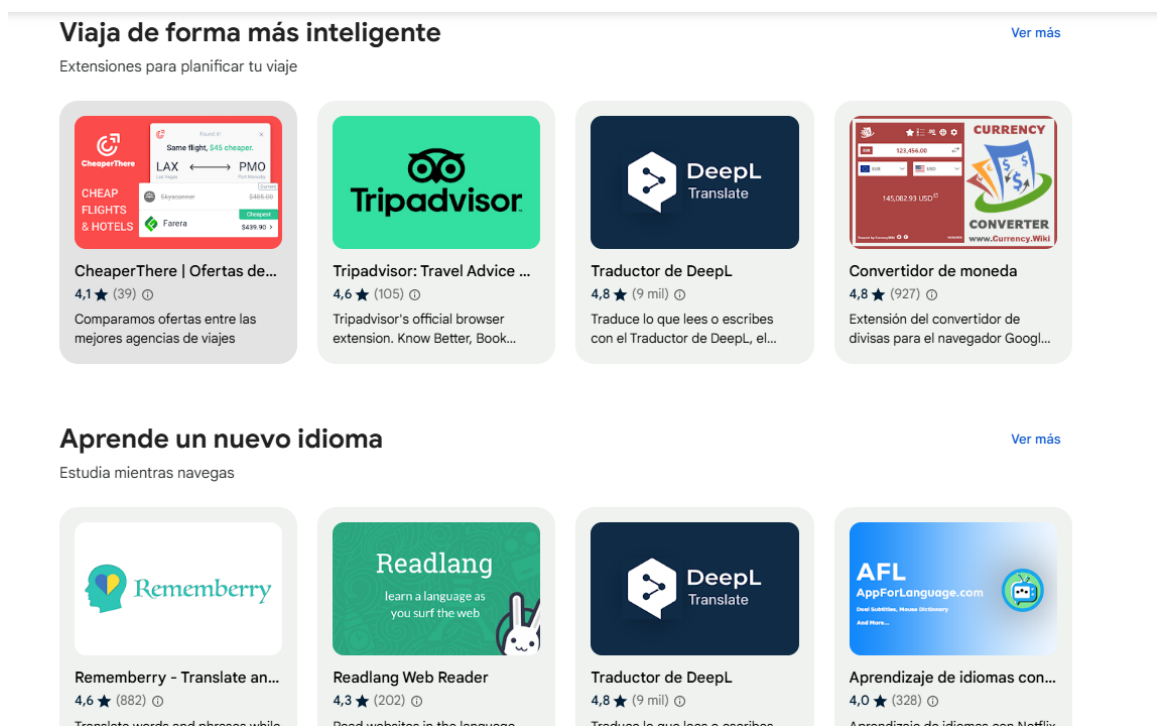


Figure 2.3: Examples of Chrome extensions [7]

In conclusion, Chrome extensions are an excellent example of how personalization can improve the user experience. Making the browser more efficient and functional has become a handy tool today.

2.6 Chrome API

The Chrome API [16] is a set of modules that allows developers to extend and leverage the functionality Google Chrome offers. These APIs are commonly used in the creation of

extensions, as they have many features that allow the user to interact with the browser, allowing the creation of web applications or projects of greater complexity.

Among the functionalities, we find the interaction with the browser tabs, being able to create or modify them to the developer's liking and obtain information about them. You can also use API Notifications to alert users of certain incidents, events, or errors that can be shown to the user. At the same time, we find a database that provides a way to store them locally so that it is easy to save and retrieve them efficiently.

These interfaces are designed to be asynchronous [1], which means that users can get responses quickly as these operations do not block the main flow of the browser. To use a Chrome API, permissions must be declared in the 'manifest.json' so that the developer can choose which ones to use and which not.

2.7 Tailwind.css

Tailwind.css [29] is a new CSS framework that allows you to design your website by composing simple utility classes to create different effects. With this technology, you can move elements on the page, create complex page layouts, and dynamically stylize your text. Tailwind makes layouts easy, as it uses simple prefixes such as "lg", "md", "bg" or "mt" as shown in Figure 2.4.

This framework eliminates a lot of code. In particular, CSS classes are removed to stylize your texts more efficiently and to be able to debug much more easily. This is because, with many CSS classes, it is difficult to make that code correction. More than 688,249 websites currently use Tailwind [21], which has increased significantly over the years.

```
<div class="bg-white py-24 sm:py-32">
  <div class="mx-auto max-w-7xl px-6 lg:px-8">
    <div class="mx-auto max-w-2xl lg:mx-0">
      <h2 class="text-3xl font-bold tracking-tight text-gray-900 sm:text-4xl">From the blog</h2>
      <p class="mt-2 text-lg leading-8 text-gray-600">Learn how to grow your business with our expert advice.</p>
    </div>
  </div>
```

Figure 2.4: Example of using Tailwind.css [18]

2.8 Typescript

Typescript is a programming language developed by Microsoft. It is a superset of JavaScript that adds optional static types, which increases the ability to debug code and find errors.

Among its most common features is transpilation, as Typescript transpiles to JavaScript and converts it to that language, allowing it to run in any browser or environment that supports JavaScript.

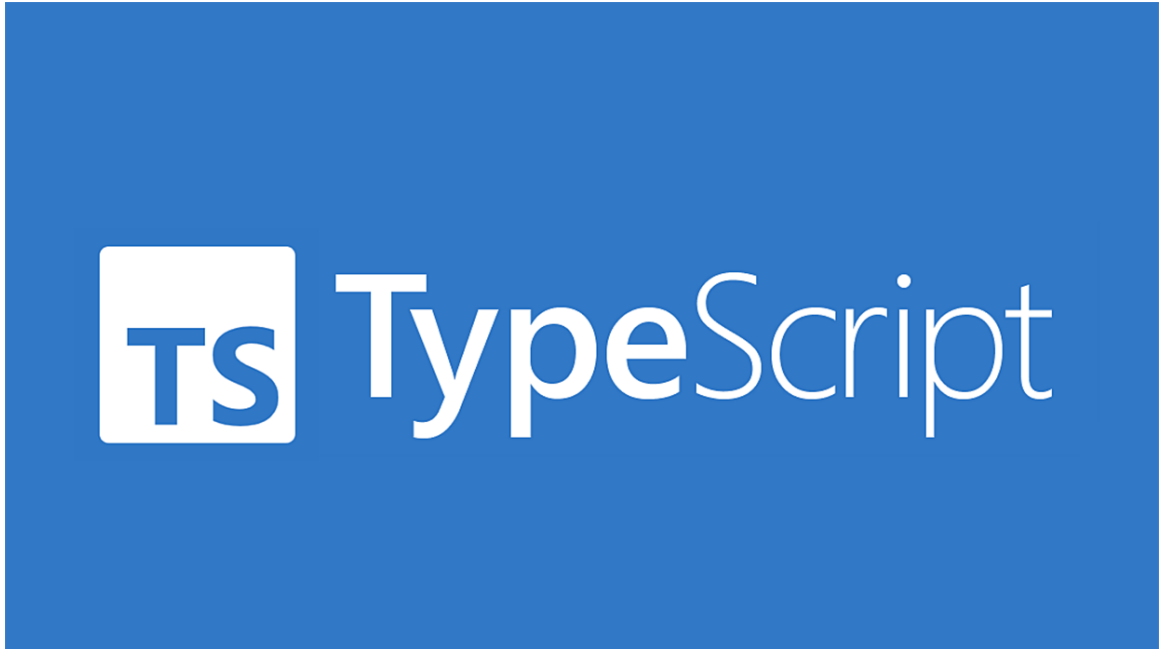


Figure 2.5: Typescript Logo [15]

It also has an inheritance system that facilitates code organisation and component reuse. It also allows you to define custom interfaces and types to improve code clarity.

Sentiment analysis extensions

3.1 Introduction

Although text analysis is a popular application of Natural Language Processing (NLP), only a few extensions provide this functionality. In this chapter, we will look at several Chrome extensions that use text analysis that will be considered for the implementation of the project. For this purpose, the personalization used in each extension will be analyzed to achieve better results and obtain conclusions that can help create our extension.

3.1.1 Sentitude Sentiment Analysis extension

In this extension created by Christian Broms [4] we can get any web page's sentiment and sentiment values by scanning it automatically or by selecting individual paragraphs or text fragments. It is an exciting type of analysis, as it offers a great variety of results both in the pop-up and directly in the text.

As shown in Figure 3.1, when you select text and right-click, you get an option that has been added with a context menu that makes the analyzed text highlighted. This highlighted text can show keywords that indicate neutral, positive, or negative feelings. This option is

exciting to add in an improved way to the project, as it clearly shows the user the words that make that text have that tendency.

On the other hand, as seen in the figure, the pop-up has a dynamic and attractive style and allows the user to customize it. There is a separation between analyzing a selected text and an entire web page. The sentiment, pleasantness, or attention value is shown with numbers, followed by a sentence that indicates whether it is positive or neutral.

One of the details that stands out the most about this extension is its settings section, as you can see in Fig. 3.2. These settings are handy and customizable for the user, allowing him to choose whether the analyzed text should be highlighted, among other options. Adding a settings section with more features to the project seems like a good decision because it allows the extension to adapt to the needs of the person using the technology.

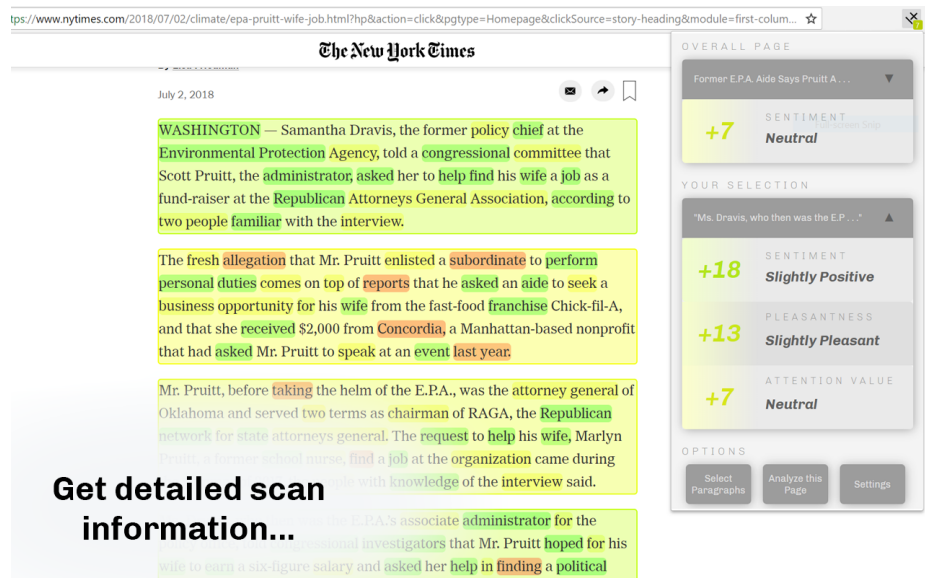


Figure 3.1: Sentitude Extension [4]

3.1.2 Youtube Sentiment Analysis

The second extension we found is a Youtube comment analyzer [39]. This extension employs the VADER sentiment model to analyze comments in YouTube videos, offering perceptive sentiment visualizations.

This technology shares details similar to those of the first extension. The figure below displays a sentiment analysis in which words are highlighted using the same division into neutral, positive, and negative sentiment with yellow, green, and red, respectively. But

the main difference is that, in this case, these words are dynamically displayed in a box of different sizes according to the frequency with which the word is repeated, as can be seen in Fig. 3.3.



Figure 3.4: Sentiment Analysis [39]

Another detail to take into account is that this extension uses a different type of visualization than what we have seen previously. Above the word box is a sentiment measurement graph similar to a speedometer, a form that is not often used in analytics extensions but is just as useful and explanatory as the rest. This brings about new perspectives on how we can display our extension.

3.1.3 Sentiment&emotion Analysis - GSI

Finally, we will analyze the extension created by a UPM student called sentiment&emotion [11]. This extension is created using the same services that will be used in this extension. The text analysis project shows you the results with different graphs.

The extension fulfills the functionality, but we look at some improvements that we can make to it, and those are the ones that will be used in the project. First of all, this extension was developed in pure Javascript without using any libraries. In the project, we will use React to produce much cleaner, more functional, and more component-based code. Secondly, the interface is a bit complex and unclear, which is another feature that will improve our extension and make it much clearer.

At the same time, the configuration part has practically no options for the user to interact with, since the user can only choose the type of analysis or the type of scope, as can be seen in Fig. 3.6. Also, the functionality of selecting text is a tedious method as you first have to select the text and then go to the extension button, something that will be done differently in the project to help the user.



Figure 3.5: GSI-Sentiment&Emotion [11]

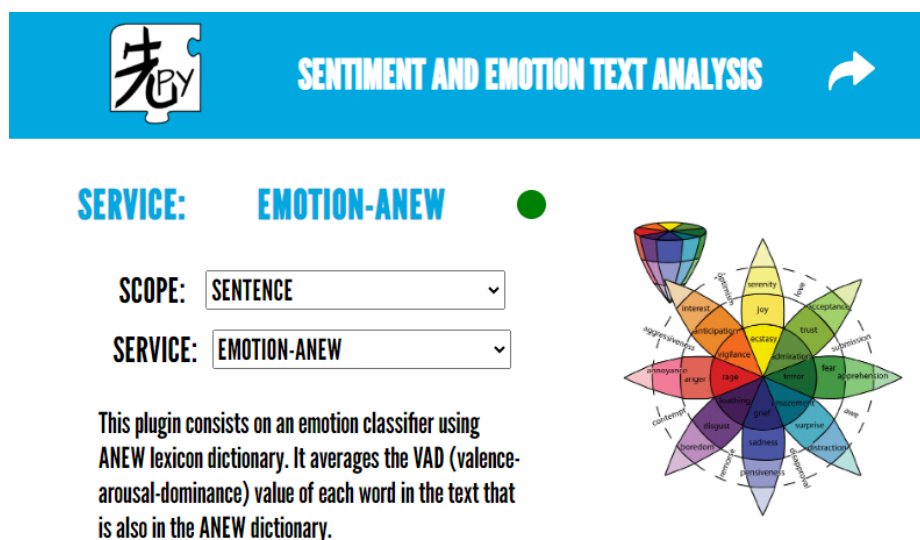


Figure 3.6: Configuration [11]

Also, the types of graphs chosen and the information in the analysis looks very basic and does not give much insight into what data is being displayed and what is not, leaving a display that could be improved as shown in Fig. 3.7.

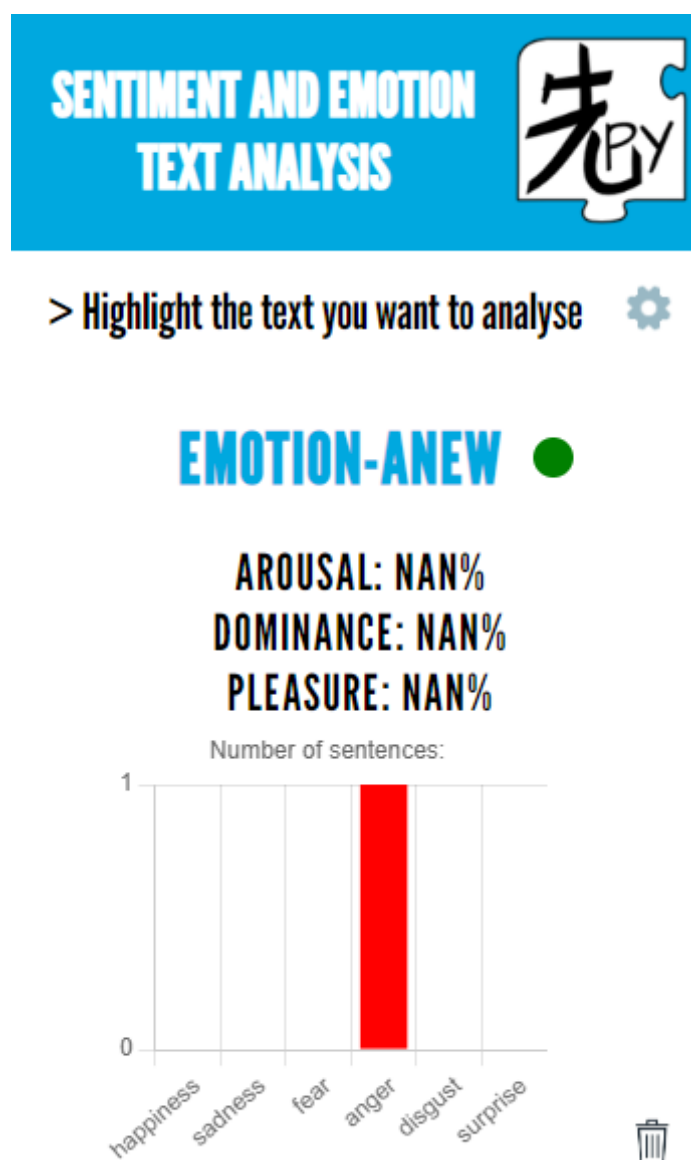


Figure 3.7: Emotion Display [11]

As a last improvement, our extension can be used and installed in any browser, not only Google Chrome, allowing a better user experience.

Architecture

4.1 Introduction

The project has been developed in three phases, each comprising the architecture and describing the development process in detail. The following process steps will be defined from its creation to its packaging.

- **Environment creation:** Here, the steps that have been followed to create the base project and the resulting environment using Plasmio technology will be described. Afterwards, it will be explained how the initial tools and dependencies needed to start the project have been set up.
- **Environment development:** This section will explain how the project's architecture has been developed from the base created in the previous part. It will begin by describing the planning process and architecture by defining the key layers and components of the system and then detailing them in more depth.
- **Publication and packaging:** Finally, we will explain how the extension has been packaged to obtain the necessary files that will be used for the publication of the extension in the different browsers.

4.2 Environment creation

In this section, we describe the creation of this extension, which is based primarily on React as the core technology integrated through the Plasmo framework [8].

As detailed in Sect. 2.3, Plasmo is a browser extension development framework for the web browsers Chrome, Firefox, and Microsoft Edge. It is based on React and supports TypeScript. In addition, it includes a storage API for persisting the extensions and a messaging API for communicating the different extension parts easily.

Plasmo is used to build the user interface and the extension modularly. All components are made with React using Typescript files that relate to each other through composition and state management. To create the main environment, we type the command `npm create plasmo`, which will be used to create the simplest extension of Plasmo as you can see in Fig. 4.1, and some basic files will be created to start developing the project. The most important will be the `popup.tsx` file, as it exports a default react component that will be rendered on the popup page of the extension. With this and the basic configuration files, you can load the extension in any Web store, as we will see later. From this basic environment, the entire architecture of the extension has been developed.

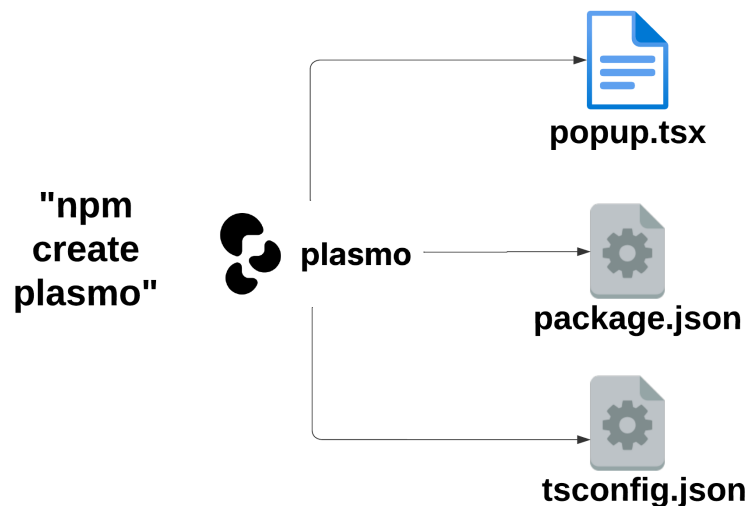


Figure 4.1: Extension Creation

4.3 Environment development

Based on the environment that Plasmo builds for you, we create the architecture of our extension for morality and emotion analysis. Our project's architecture basically uses TypeScript files with React components that can be used for different functions. Our structure is divided into four clear and detailed modules. Each module has its own functionalities and components, which perform different functions for the project's purpose. The following describes how these modules are divided and communicated.

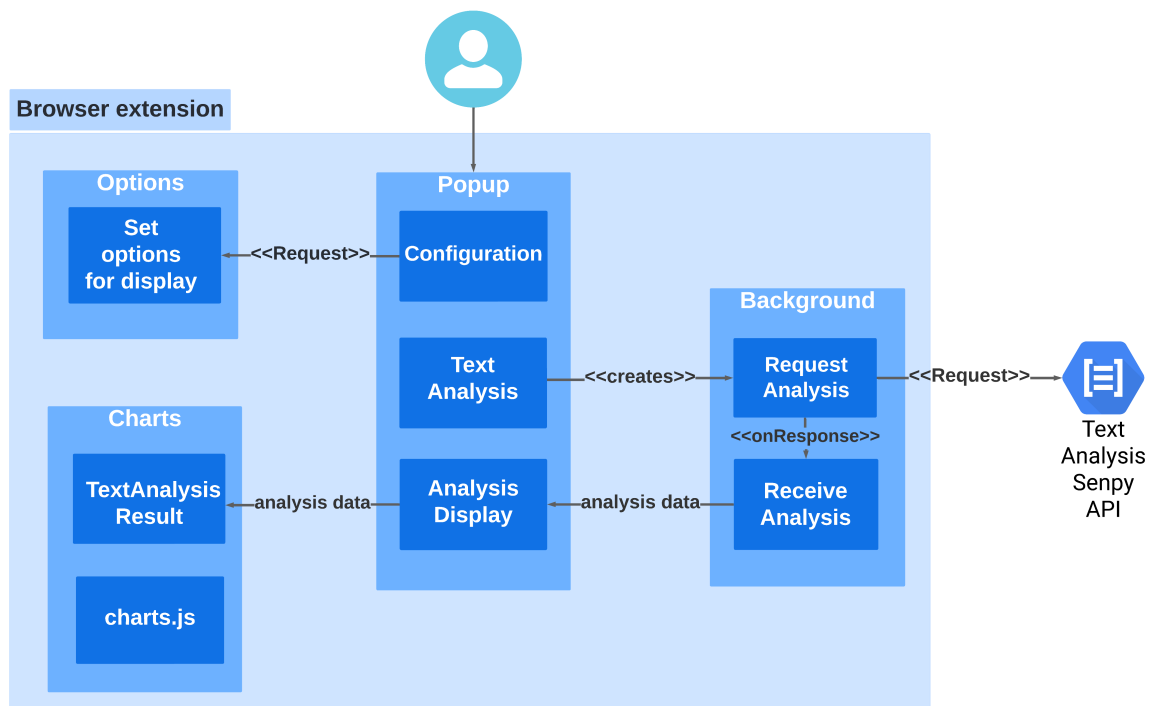


Figure 4.2: Architecture Overview

- **Popup:** This module is what the user interacts with. It is in charge of making the request to obtain the results of the analysis and send them to the module in charge of displaying the data. At the same time, it also sets the configuration by communicating with options to be able to change and customize the extension.
- **Background:** The module in charge of the Senpy API [31] call is the background. From the popup request of the text analysis, the background is in charge of making the API request to collect the data and send it back to the popup. This is done in an asynchronous way so that there are no blockages in the main flow.

- **Options:** This part of our architecture is very important. Its function is to create a Configuration Page with different settings. After setting them, the data is sent to the popup for implementation.
- **Charts:** Module in charge of importing the charts.js library to create the different types of display created for the extension; it receives the data from the popup to later process them.

All of these modules are described in full detail in the following sections.

4.3.1 Popup

The main module of the extension is the popup. It is a file that is automatically recognized in an extension by its nomenclature. It is where the flow of the user's data request for further analysis begins. This file is the most important file in the project and the one that handles the most logic; we will divide its functionalities and explain them separately.

4.3.1.1 React Hooks

The use of React as a library enables us to use certain tools that are not available in other programming languages. Hooks play a crucial role in the functionality and reactivity of our extension, as they allow us to develop functional components with state and lifecycle and, therefore, dispense with class components. In the popup, we used both hooks that are part of the React library and custom hooks created especially for the project.

Several hooks are provided by the React library itself. These hooks are the ones that are used in most projects, and we decided to use them for the extension as well. The first and most commonly used is `useState()` [36]. This hook is used to manage the local state of the components. This allows us to store and update certain values throughout the lifetime of a component. In the popup, we use it to initialize the data, algorithm types, or selected text, as can be shown in the listing 4.1. It allows us to declare variables and easily change their state. It is also used to set the settings that the user changes.

The second is to use the `useEffect()` hook [35]. This hook is fundamental in React because, unlike the `useState` hook, it performs side effects of the component. It is executed after rendering and enables it to perform tasks that should occur outside of the loop. This is very useful in our application, allowing us to make asynchronous calls when necessary. In our case, it is in charge of requesting the analyses from the background using a message.

These asynchronous calls are really useful, as they allow multiple operations to be done in parallel, improving the overall performance of the extension. Therefore, it will help more users make more requests without exceeding the limit.

```
const IndexPopup = () => {
  const [selectedText, setSelectedText] = useState("");
  const [senpyData1, setSenpyData1] = useState(null);
  const [senpyData2, setSenpyData2] = useState(null);
  const [senpyData3, setSenpyData3] = useState(null);
  const [senpyData4, setSenpyData4] = useState(null);
  const [algorithm1, setAlgorithm1] = useState("");
  const [algorithm2, setAlgorithm2] = useState("");
  const [algorithm3, setAlgorithm3] = useState("");
  const [algorithm4, setAlgorithm4] = useState("");
}
```

Listing 4.1: useState()

Due to React's variety and possibilities, we can create our own hooks. We use them to extract and share logic efficiently. Like the hooks provided by React, the main feature of hooks is that they start with 'use'. This is not just a convention; it allows React to apply hook rules correctly. The hook we created for this application is the 'useAnalyzeAllTextOnPage' hook. This component encapsulates application-specific logic by performing text selection and parsing tasks. The popup is in charge of importing it and using it to analyze a web page's content by clicking on a button in the interface. After clicking the button, the flow to the Senpy call occurs, where the information will be returned following the sequence diagram shown in Fig. 4.3.

4.3.1.2 Analysis Type Props

In this part, we will see how props [27] are used in the popup to pass the information from a parent component, such as the popup, to a child component, which is *TextSelected.tsx*. We define a series of states with the information about the data provided by Senpy, the selected text, the type of algorithm we want to represent in each case, and the graphs provided by the settings (those selected by the user). All this is sent to *TextSelected* as props so that it can later handle some analysis and some algorithm and process the data in different ways to be able to display them correctly.

Once in *TextSelected* with all the data, we create several components to first be able to go through the data of each type of algorithm since each one is processed in a different way

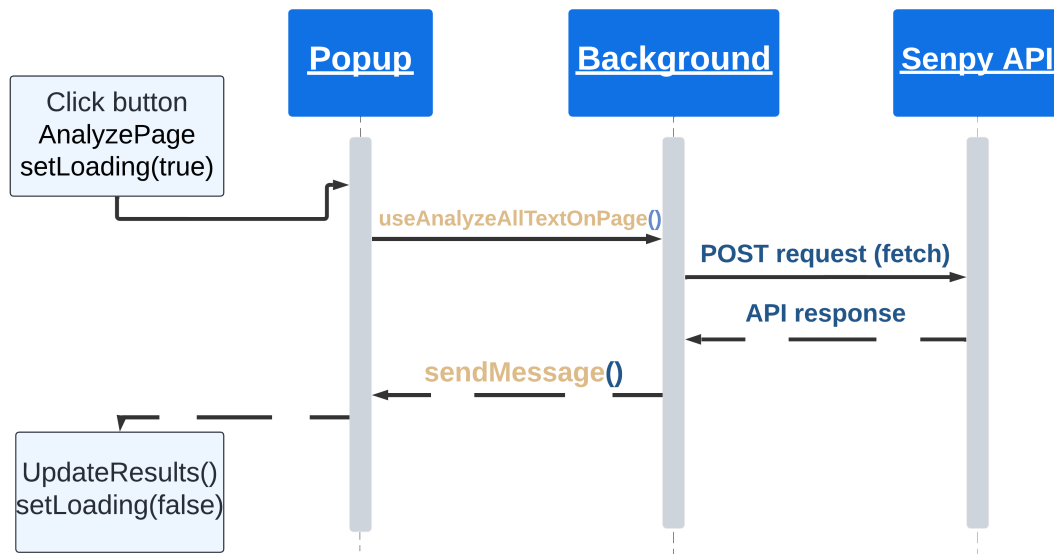


Figure 4.3: Flow Diagram

and gives us different information about the analyzed text. The structure of this component is divided as follows.

- **Charts creation:** We import the different types of charts that we will see later, and with them, we create a particular one for each type of analysis. The ease of use of the components allows you to create a number of different charts for each of the analyses and helps to improve the data presentation.
- **Use of the types of analysis:** Analysis types are navigated in unique ways, processing the information to be used in the chart representation. An example of how an analysis type is navigated is shown in the listing 4.2.

```

const polarityCounts = senpyData['marl:hasOpinion'].reduce((acc:
  Record<string, number>, opinion: any) => {
  const polarity = opinion['marl:hasPolarity'].split('
    :').pop();
  acc[polarity] = (acc[polarity] || 0) + parseFloat(
    opinion['marl:polarityValue']);
  return acc;
}, {});

```

Listing 4.2: PolarityCounts()

4.3.2 Charts

Data display is a very important component of our extension. It provides users with a clear and visual representation of text analysis results. The following section describes how charts work and are created in our extension.

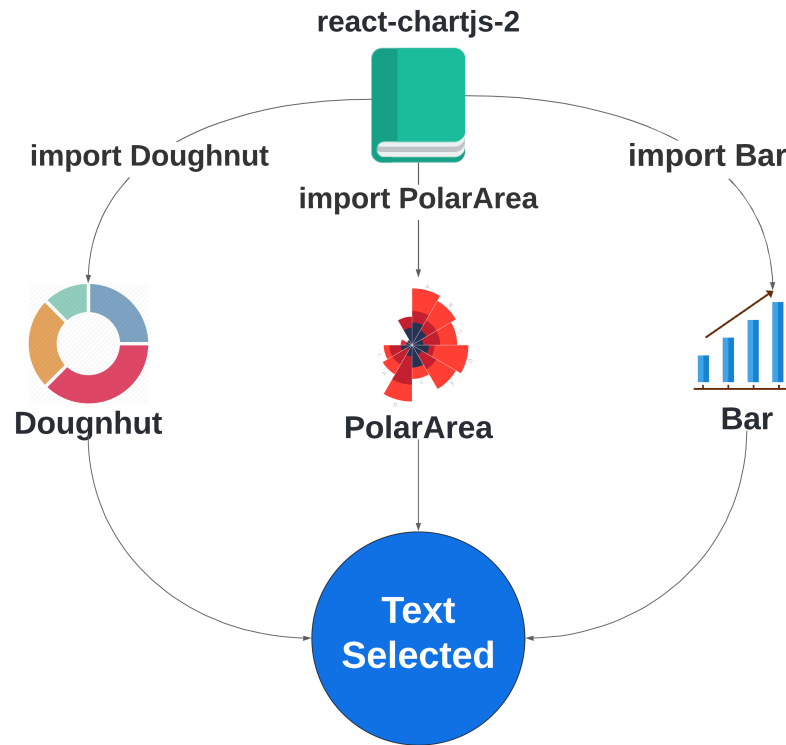


Figure 4.4: Import from react-chartjs-2

We have three types of charts: doughnut, bar, and polar area. These charts are components that are created and then imported into the component `TextSelected` so that they can be used later. So, the `react-charts-2` library is imported into each component, not into the analysis component, as you can see in Fig. 4.4.

4.3.2.1 Charts implementation

As we saw in the previous chapter, the three charts use props to communicate with the main component, which is where all analytics management is developed. This implementation is useful for creating more types of charts in a simple way so that another programmer can understand and use our application's component composition. We only had to create new components and import them.

Once the charts are imported, each is individually configured according to the type of analysis. The chart library includes an options [5] section that offers various options to customize each chart.

In our extension, we configure each chart, eliminating the doughnut chart legend to make the representation much cleaner, adding elements by adding `HoverBorderColor` or configuring the tooltip [34]. To add the data, we use constants called *createData*, passing a series of information from the different text analyses we implemented as parameters to the data. In the code excerpt 4.3, we can see an implementation of these options.

Then, we also added an image in the center of the doughnut charts to make the chart more dynamic. For this, we have created a feature in the utils section called *createCenterTextPlugin*, which allows us to have different images in the doughnut's center; it is also a function passed to the *TextSelected*.

```
const options = {
  responsive: true,  cutout: '80%',
  plugins: {
    datalabels: {
      display: false
    },
    tooltip: {
      enabled: true,
      backgroundColor: 'rgba(0, 0, 0, 0.7)',
      titleColor: 'white',
      bodyColor: 'white',
      borderColor: 'rgba(75, 192, 192, 1)',
      borderWidth: 1,
    },
    legend: {
      display: false
    }
  },
  interaction: {
    intersect: false,
    events: ['mousemove', 'mouseout', 'click', 'touchstart', 'touchmove']
  },
  elements: {
    arc: {
      hoverBorderColor: 'rgba(0,0,0,0)',
      hoverBorderWidth: 0,
    }
  }
}
```

```

        hoverBackgroundColor: (context) => context.
            dataset.backgroundColor,
    }
}
};

const createData = (count: number, label: string, color:
    string) => ({
    labels: [label],
    datasets: [{
        data: [count * 100, 100 - count * 100],
        backgroundColor: [color, 'transparent'],
        borderColor: 'black',
        borderWidth: 1,
    }]
});

```

Listing 4.3: options and createData

4.3.2.2 Plasmo import resolution

Imports are fundamental to the extension's development, guaranteeing the code's efficiency and organization. Plasmo framework provides a utility for importing the files we need, such as source files and bundles, but especially images.

The easiest way to load assets into Plasmo is to use the *data-64* scheme. This scheme in the extension is used to read the content of a file, transform and optimize it for browser consumption, and then convert the result into a base-64 string. This string can then be embedded directly into the code so that assets are available without the need for additional paths or external references. This method is particularly useful for including images or any other files in the extension. Instead of using the Chrome API, we can use this system, as seen in the listing 4.4.

```

import imageSrc1 from "data-base64:src/images/EmojiNeg.png";
import imageSrc2 from "data-base64:src/images/EmojiPos2.png";
import imageSrc3 from "data-base64:src/images/EmojiNeu.png";
import imageSrc4 from "data-base64:src/images/morality.png";
import imageSrcBar from "data-base64:src/images/SentimentsBar.png";
import imageSrcDoughnut from "data-base64:src/images/
    SentimentsDoughnut.png";

```

```
import imageSrcPolar from "data-base64:src/images/PolarChart.png";
import imageSrcEmotion from "data-base64:src/images/Emotion.png";
import imageSrcLogo from "data-base64:src/images/gsi_logo.png";
```

Listing 4.4: data-base64 use

These images will be used in the charts and in any part of the extension.

4.3.3 Background

The background component is crucial to the extension’s functioning. It is the project’s “brain” and is in charge of managing events and background tasks and making requests to the Senpy server. Our extension’s background is dedicated to various tasks and operations, as described below.

- **Background Service Worker:** Plasmo provides us with a powerful service worker because it runs in the background. Using this, CORS [20] is unnecessary in the extension as the project would not be subject to like other scripts, the service worker remains active even when browser windows are closed, handling tasks that do not require user interaction. When creating the background file, the service worker is automatically activated.
- **Messaging:** To receive the data and send it back to the popup, a messaging API is needed. So, the background is able to listen when the popup creates a request requiring the corresponding analysis, choosing the corresponding analytics service, and then able to return the data asynchronously and outside the rest of the flow of the extension.
- **Error handling:** We have created a background component called *handleRequestError* for error handling. This function handles errors during the parse request process, logs errors to the terminal with a *console.log*, and notifies the user with a message.
- **ContextMenu:** Apart from analyzing the whole page, you can select text and analyze it, and for this, we have created a contextMenu that is created every time the extension is installed in any browser. Once created, we also handle user interactions in the menu using the *OnClicked* event.
- **POST Request:** A POST request is used to make the request to the Senpy API to obtain the required data by sending the text selected by the user, either the whole

page or just a paragraph, explained in more detail below.

A POST request is an HTTP request method that is used to send data to a server or update a resource. The POST method is versatile as it allows you to handle a wide variety of content, including XML, encoded data, or JavaScript Object Notation (JSON), the latter being the data we use in the extension. The data sent with POST are not visible in the URL; this provides a layer of security to the user and the extension, as we may be handling sensitive data when analyzing morality.

In the extension's context, the background sends several endpoints using POST requests each time the user selects the text to analyze. These endpoints contain the Senpy URLs of each analysis type integrated into the extension that the user can request. All the request logic has been concentrated in an asynchronous React component called *performPostRequest*. This component has two parameters passed to it: a URL that you get from the endpoint and a selected text. Its logic is detailed in the Fig. 4.5.

We first considered using a GET request instead of a POST request for the project. However, when we implemented the button to analyze the whole page, the amount of text collected was insufficient for the GET request. So, a POST request was chosen, which allowed you to send a large amount of complex data that the GET could not handle properly.

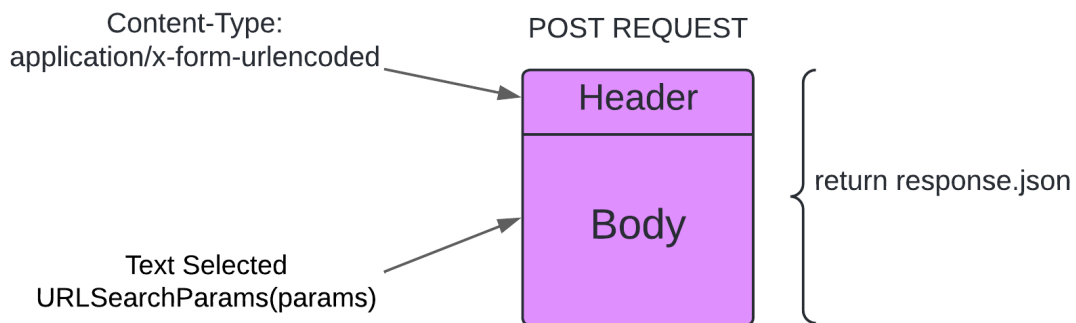


Figure 4.5: POST Request Architecture

4.3.4 Options

This section will show the extension's configuration section. All settings chosen by the user can be found in the options file. This file communicates only with the popup to receive the type of options to set and save. Plasmo offers a storage system that is very useful for our extension and that we use in this module.

The framework provides us with a storage API, which is obtained from the `@plasmohq/storage` library. This library provides persistent storage for any browser extension. It also has local storage when the extension's storage API is unavailable.

To use it, install it with the command `pnpm install @plasmohq/storage` and import the content as any React component `import Storage` from “`@plasmohq/storage`”. Once this is done, we use the system's possibilities to create our options section. The two options that allow communication and set the options are as follows.

- **Storage.get:** Communication between the options page and the popup is achieved in this way, mainly through the use of storage via the `get` that is sent from the popup to receive the options that have been defined in the configuration page.
- **Storage.set:** With `set`, we update the configuration through the user interface, where every time a change occurs, it is stored in the Plasmohq storage API. So we use it for any option the user sets; this way, the change happens automatically, and you don't have to save them in any way.

The storage API makes it easy for the user to persist the data just like the Chrome API. This helps the user keep the settings even when they close the browser so they don't have to constantly change them and can enjoy the personalized experience intended when creating this extension.

4.3.5 Tailwind implementation

For the development of the extension, we have not used any CSS, as we have taken advantage of the Tailwind technology, which allows us to use `className` to directly set the style of any block as explained in 2.7.

Using these predefined utility classes ensures consistent styles in the extension and has helped reduce errors and visual discrepancies that can occur when using standard custom styles. These classes are used in both popup and options to define element layout, margins, paddings, and colors. It has also provided help in the creation of elements such as the settings gear or the spinner to indicate that the analytics are loading. An example of its use can be seen in the following listing 4.5.

```
<div className={`flex items-center justify-center min-h-screen
  relative ${settings.darkMode ? 'bg-gradient-to-r from-gray-700
    to-gray-900' : 'bg-gradient-to-r from-[#e0f7ff] to-[#b3e5fc]'}
  `}>
```

```

<div className={`w-96 ${settings.darkMode ? 'bg-gray-800
  text-white' : 'bg-white text-gray-800'} shadow-xl
  rounded-xl flex flex-col min-h-[600px]`}>
  <div className={`w-full p-4 mb-4 rounded-t-xl flex items-
    center justify-between ${settings.darkMode ? 'bg-gray
      -700' : 'bg-[#00a9e0]'}`}>
    <img src={imageLogo.src} alt="Logo" className="w-10 h-10
      " />
    <h1 className="text-2xl font-sans font-bold text-white
      mx-auto">Moral Text Analysis</h1>

```

Listing 4.5: data-Tailwind utility classes

4.4 Packaging and publication

Finally, after having fully developed our browser extension with all the technologies implemented, we proceed to its packaging and publication for the further use of all users. This section will describe the steps for this process to have been implemented.

4.4.1 Packaging

After creating and developing the environment, we proceeded to package it. Plasmo provided us with scripts for both development and creation and a script for building the extension.

The key file that Plasmo has automatically generated in its creation is the *package.json*. This file lists all the dependencies (libraries or packages) that the extension needs. It also defines a series of scripts that you run on the command line. But the most important is the manifest section, in this section we put all the necessary permissions in our extension. Normally, when you create an extension with React, all the sections go in a *manifest.json* file that you create and manage, but in this case, when you type the command *pnpm build*, it generates a file 'build/chrome-mv3-prod' where this file is automatically generated with the permissions that you have defined in the *package.json* as can be seen below in listing 4.6.

```

"manifest": {
  "host_permissions": [
    "http://*/*",

```

```
    "https://*/*",
    "<all_urls>"
  ],
  "permissions": [
    "activeTab",
    "contextMenus",
    "tabs",
    "storage",
    "scripting"
  ],
  "web_accessible_resources": [
    {
      "resources": [
        "src/features/vader_lexicon.txt",
        "src/images/EmojiNeg.png",
        "src/images/EmojiPos.png",
        "src/images/EmojiNeu.png",
        "src/images/morality.png",
        "src/images/SentimentsBar.png",
        "src/images/SentimentsDoughnut.png",
        "src/images/PolarChart.png",
        "src/images/Emotion.png",
        "src/images/gsi_logo.png"
      ],
      "matches": [
        "<all_urls>"
      ]
    }
  ],
  "browser_specific_settings": {
    "gecko": {
      "id": " analisis-de-valores@ejemplo.com",
      "strict_min_version": "42.0"
    }
  }
}
```

Listing 4.6: manifest.json

Using the Plasmo framework and its storage, the extension can be uploaded to multiple browsers, such as Chrome, Microsoft Edge, and Firefox. Both Chrome and Microsoft Edge

use the 'chrome-mv3-prod' file for packaging, while Firefox needs a 'chrome-mv2-prod' file. Among the Plasmo scripts, we also find one that creates this file or any other we need for any browser. It is created with `pnpm build --target=firefox-mv2`. Then, add some necessary permissions to the `package.json` to have a temporary ID ready to use.

In addition, this framework removes some configuration files used to create React extensions, such as `webpack.config.js`, which handles modules and other aspects; in this case, Plasmo manages these configurations automatically. The process of constructing and packaging the extension in browsers is described in Fig. 4.6.

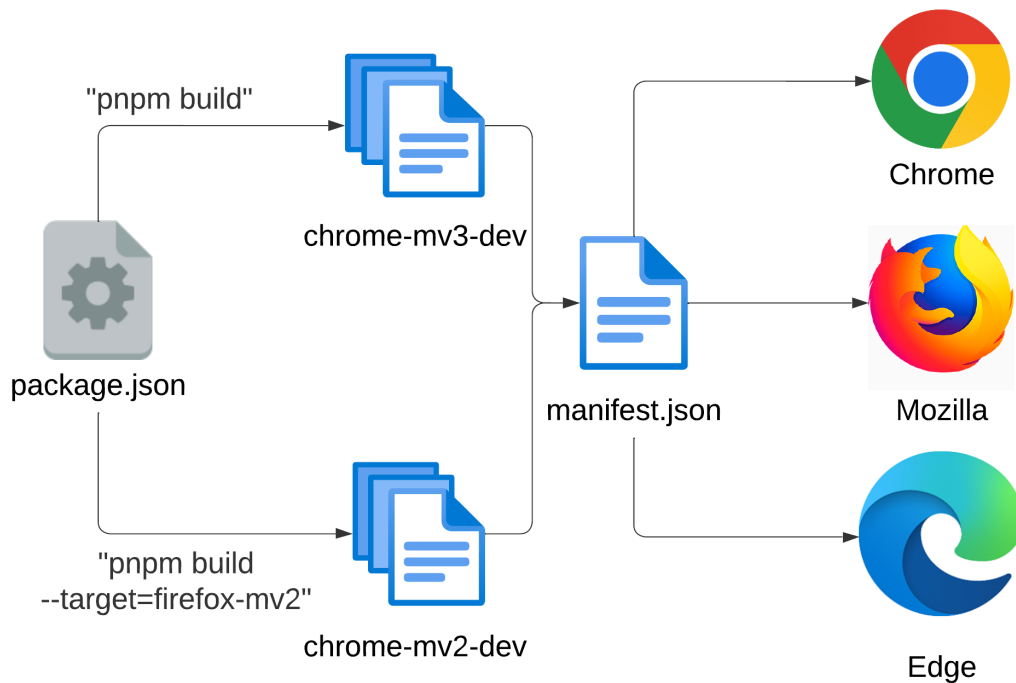


Figure 4.6: Extension Packaging

4.4.2 Publication

After the packaging is ready, we will proceed to publish the extension in the different browsers. Each browser has its own way of publishing the extension to the public, so we will explain how it has been released in each. Plasmo's documentation explains how to upload the extension to the browsers. Itero Publisher [9] is a system that allows you to automate its installation in the three browsers.

4.4.2.1 Mozilla publishing

As seen in the previous section, we created a separate file for the Mozilla Firefox browser.

After we have a file, we need to get three keys for publication, as shown in Fig. 4.7. The first is an Extension UUID, which we get from the Firefox Add-ons hub; the second is an API key, and the third is an API Secret. We get these last two on the Firefox Add-ons Developer Hub's API page, so we would already have what we need to publish the extension in Mozilla Firefox.

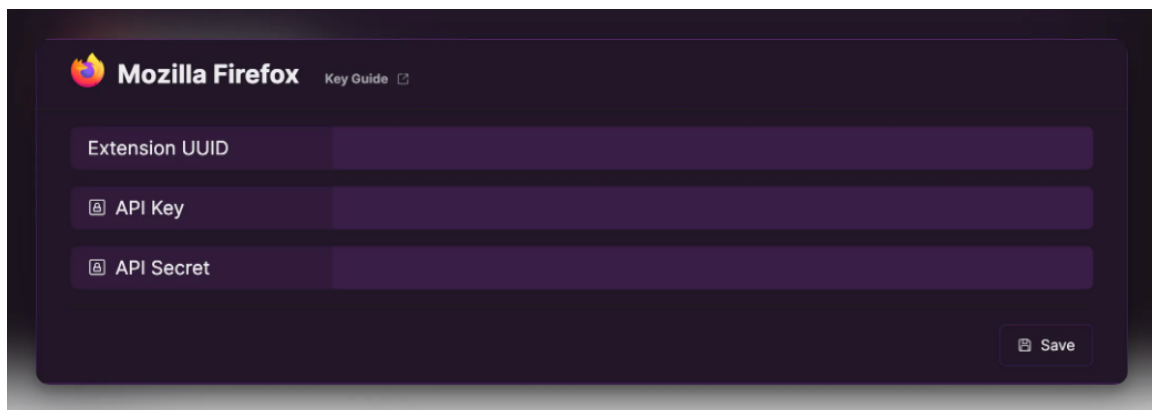


Figure 4.7: Mozilla Publication

4.4.2.2 Microsoft Edge publishing

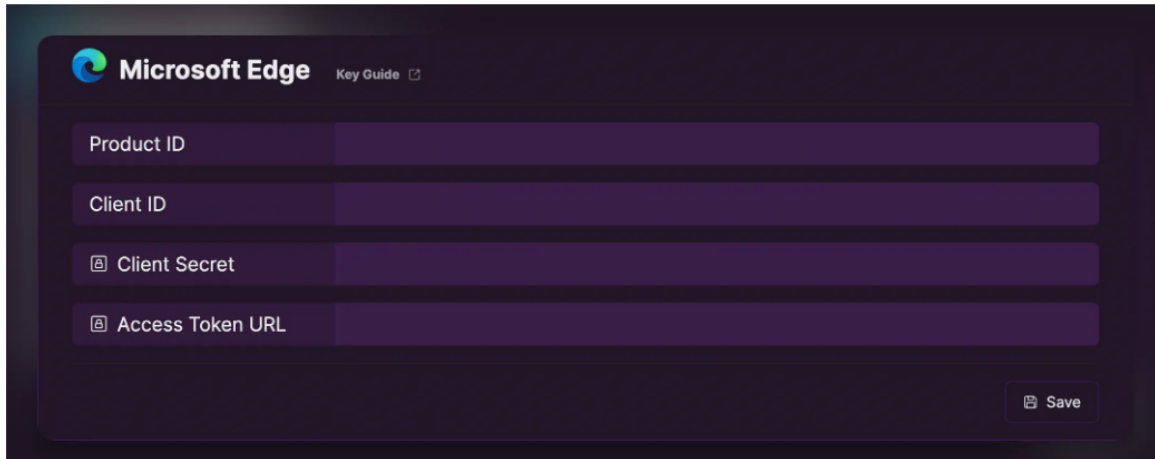
In the case of Microsoft Edge, it is similar to Mozilla. First, we create an Edge add-on and go to the dashboard, where you should see the product ID you need in the URL.

Finally, we find our `clientId`, `clientSecret`, and `accessTokenUrl` from the Microsoft Edge Publish API page as shown in Fig. 4.8.

4.4.2.3 Chrome publishing

Finally, the extension is also available on the popular Chrome browser. This process is more difficult.

First, we must create a project in the Google Cloud Console as shown in Fig. 4.9. Then we follow a series of steps to create the project and its integration. We make the necessary configurations in the Chrome Web Store API to obtain the keys for the Itero Publisher to finish publishing the project.



The screenshot shows a dark-themed window titled "Microsoft Edge" with a "Key Guide" link. It contains four input fields for credentials: "Product ID", "Client ID", "Client Secret", and "Access Token URL". Each field has a small icon to its left. A "Save" button is located at the bottom right of the form.

Figure 4.8: Edge Publication

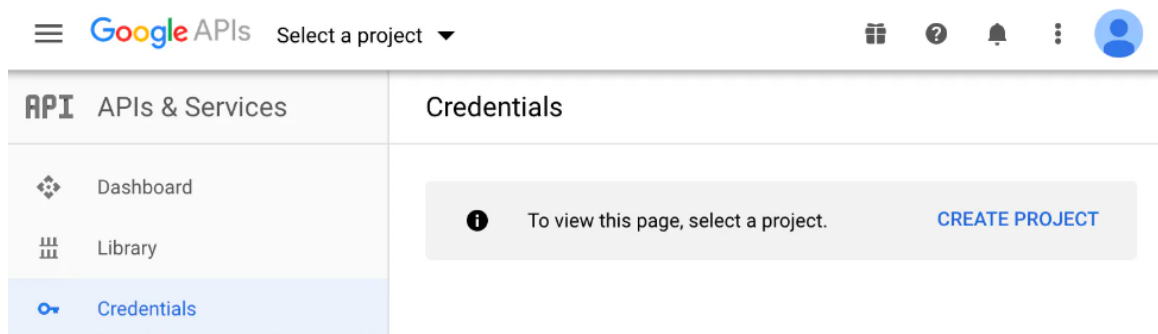


Figure 4.9: Chrome Publication

This last publication finishes the development of our extension, making it fully functional in three different browsers so that it can be adapted to all types of users who want to use it.

Case study

5.1 Introduction

This chapter describes the functionalities described in the previous chapters. The case you will see is a user who installs the extension on a device and then sees what he can do with it. The user first installs the extension in his browser, in this case, Google Chrome. Then, he can use the extension to perform text analysis and set the settings he wants.

Next, we will see what process the user follows to perform all the necessary actions.

5.2 Download and installation

The first thing the user will do is open the browser on her computer. In the case of Google Chrome, the user must go to the URL: `chrome.google.com/webstore`. The next step is to search for the extension by its name, Moral Text Analysis, or by the developer's name, in this case, Grupo de Sistemas Inteligentes (GSI-UPM) in the box shown in Fig. 5.1. The user will then have to click on the ADD to Chrome button, and the extension will automatically be downloaded to the browser. The user will then be able to use the extension normally.

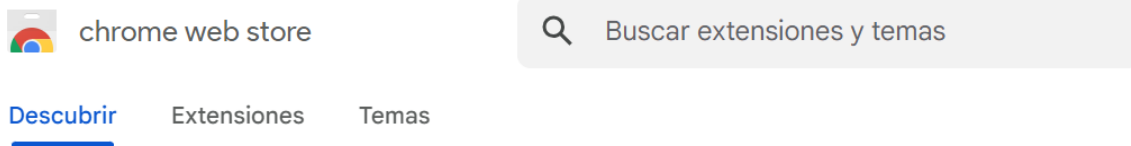


Figure 5.1: Chrome Store

5.3 Moral Analysis

We will perform the use case of a user who wants to analyze an article from the Washington Post newspaper with the extension. The first thing the user encounters is the splash screen that appears in Fig. 5.2, where you can see a simple message explaining what the extension does. At the same time, you are presented with two buttons. One is to analyze all the text on the page, and the other is to go to the configuration menu. The user can decide to

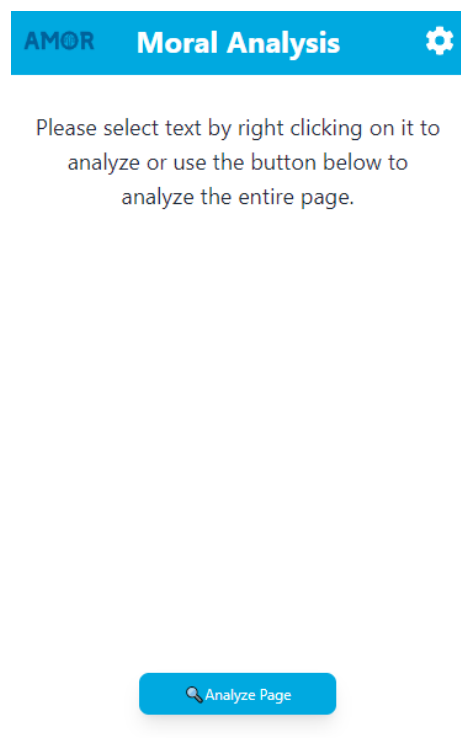


Figure 5.2: Splash Screen

scan the text with the button if he wants to scan the whole page or use the contextMenu to select only a section of text as shown in Fig. 5.3.

Once either of the two buttons has been pressed, the splash screen will display the

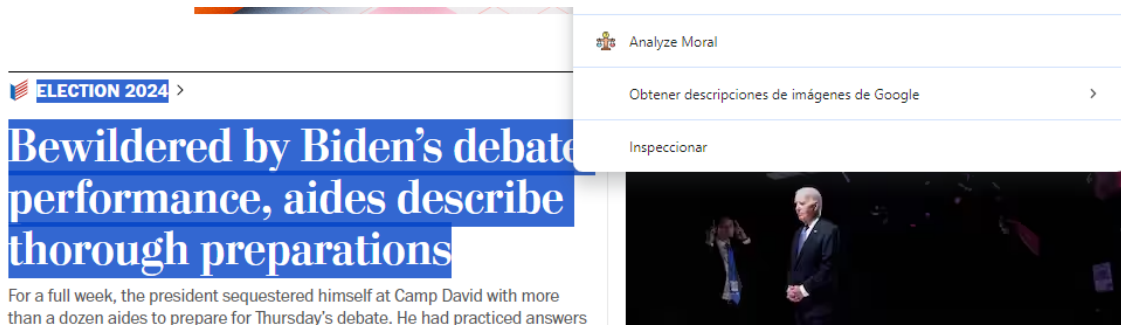


Figure 5.3: Context Menu

default analysis sample with the doughnut chart, as shown in Fig. 5.4. The user can browse through the tabs to choose between the types of analysis he wants to see on the text: morality, emotions, and feelings.

The user can now mouse over to see the exact values and percentages of the analyzes performed.

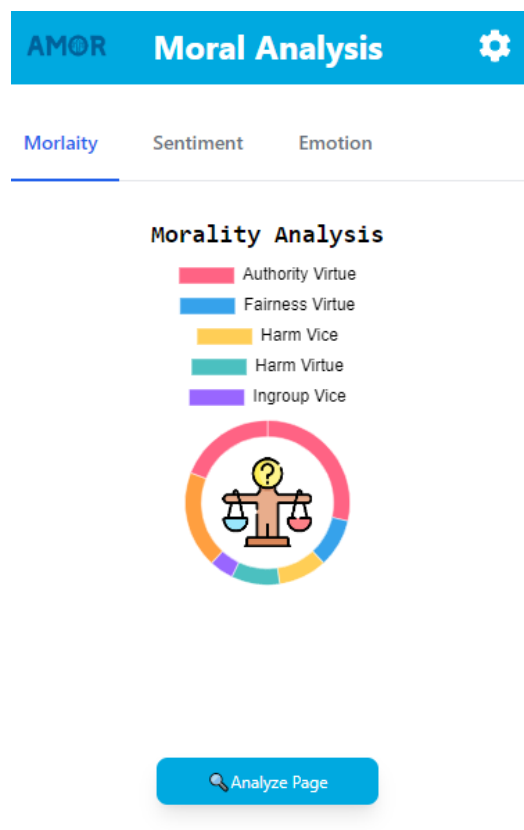


Figure 5.4: Moral Analysis

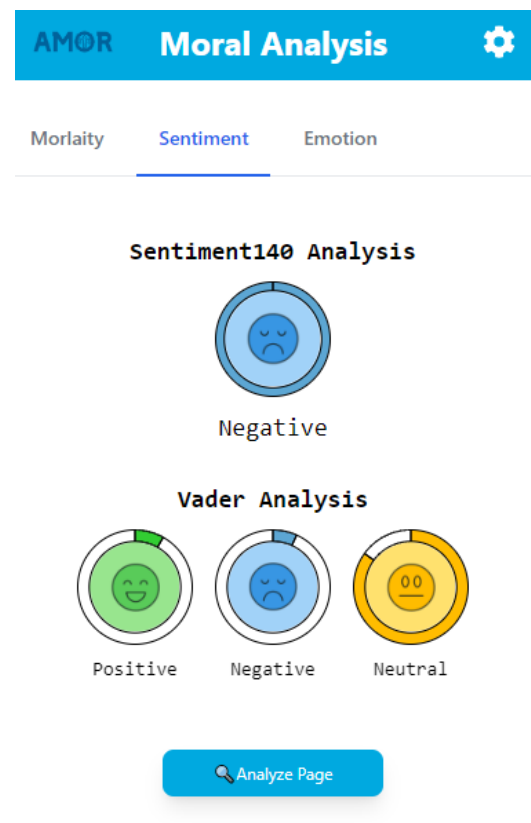


Figure 5.5: Sentiment Analysis

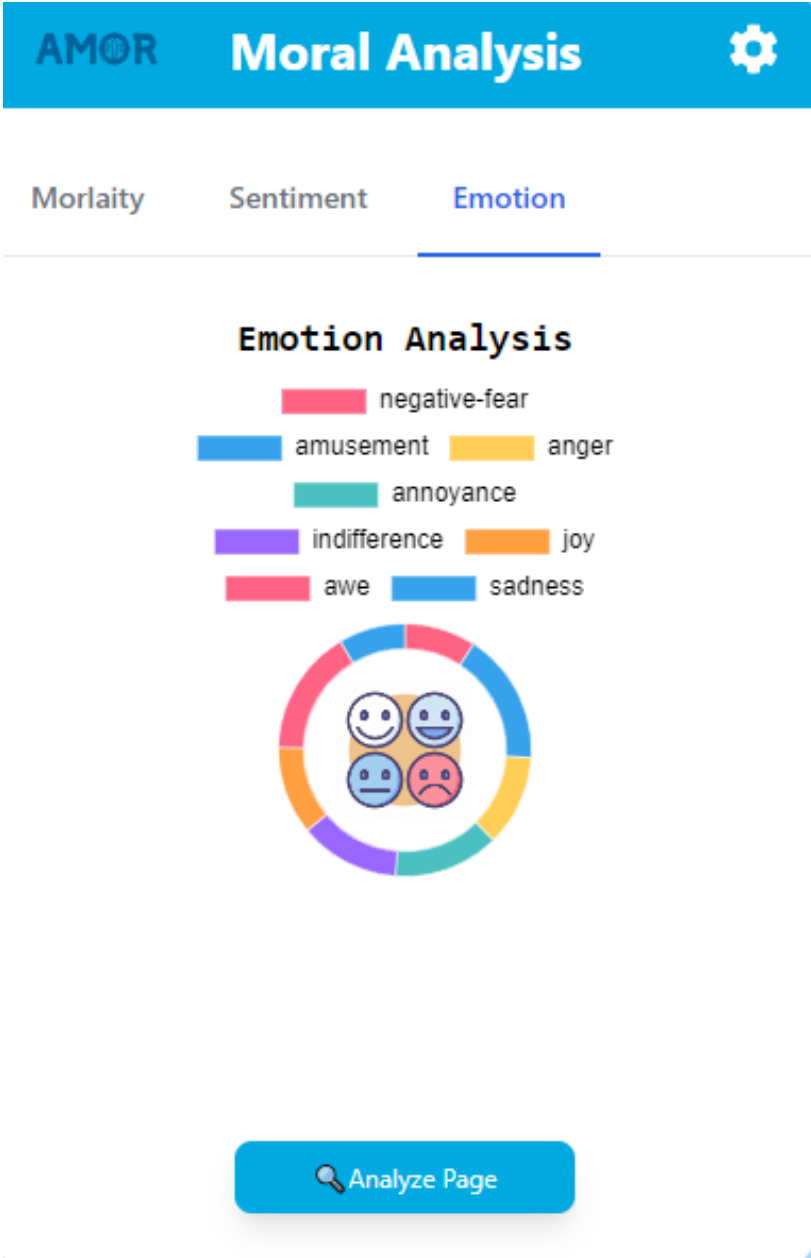


Figure 5.6: Emotion Analysis

5.4 Configuration Page

Once users want more variety in their results, they can access the configuration page by clicking on the gear-shaped button with a red border, as shown in Fig. 5.7. By clicking

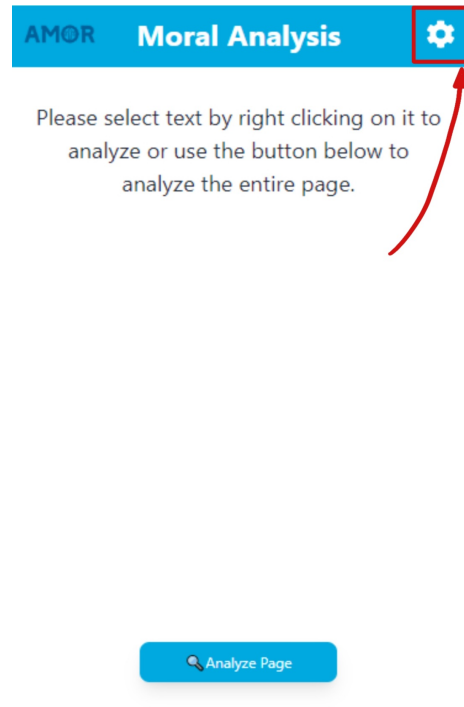


Figure 5.7: Settings Button

on the button, the user will see another tab open in the browser; this tab will be the Configuration Page. In this tab, several actions can be performed. The first of them, and the one that appears directly when the configuration page opens, is the choice of which analyses you want to be performed and displayed in your extension, as shown in Fig 5.8. The user, by removing one of these requests, can personalize and choose the one that best suits his needs. In addition, if you mouse over the questions, he will see a definition of the analysis and what it does.

The user can also see the Chart Type option on the page. This window is the most important part of the project as it allows the user to choose the charts he wants in each analysis. The user can have a personalized extension according to his needs. Here, the user first chooses a bar chart for the morality results and then chooses a polar area chart, as shown below.

Finally, the user can access the theme options. These options allow the user to choose between light and dark modes. If the user chooses the dark mode, he will see what is in the

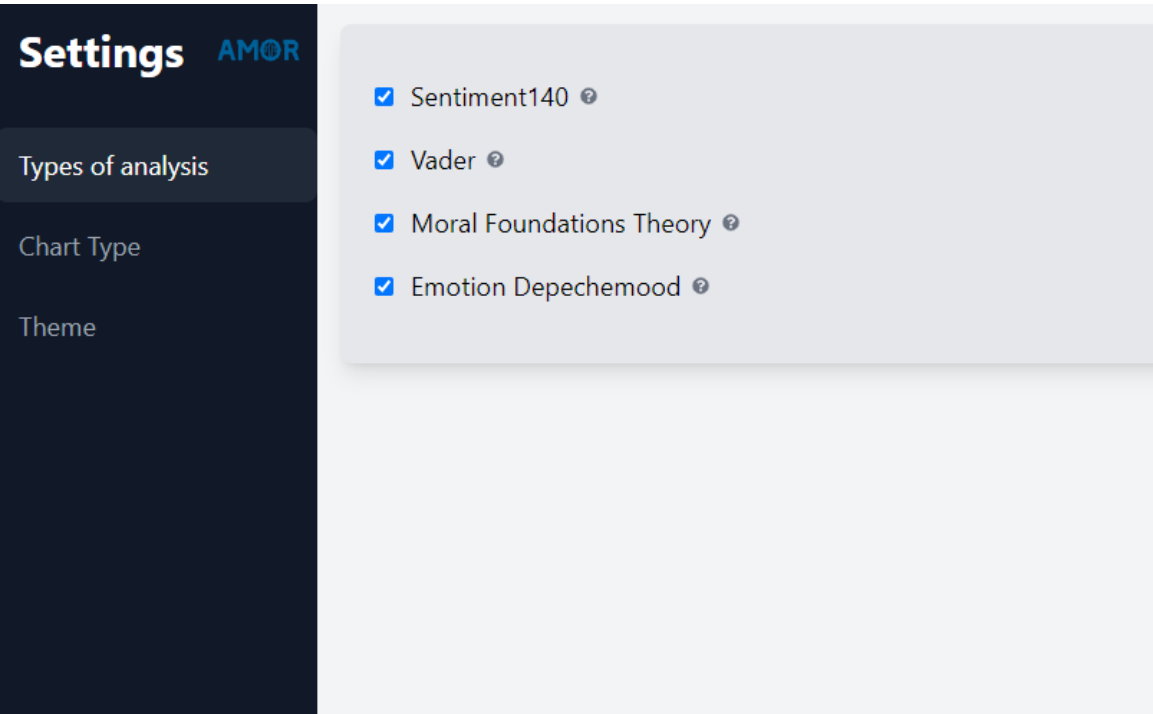


Figure 5.8: Configuration Page

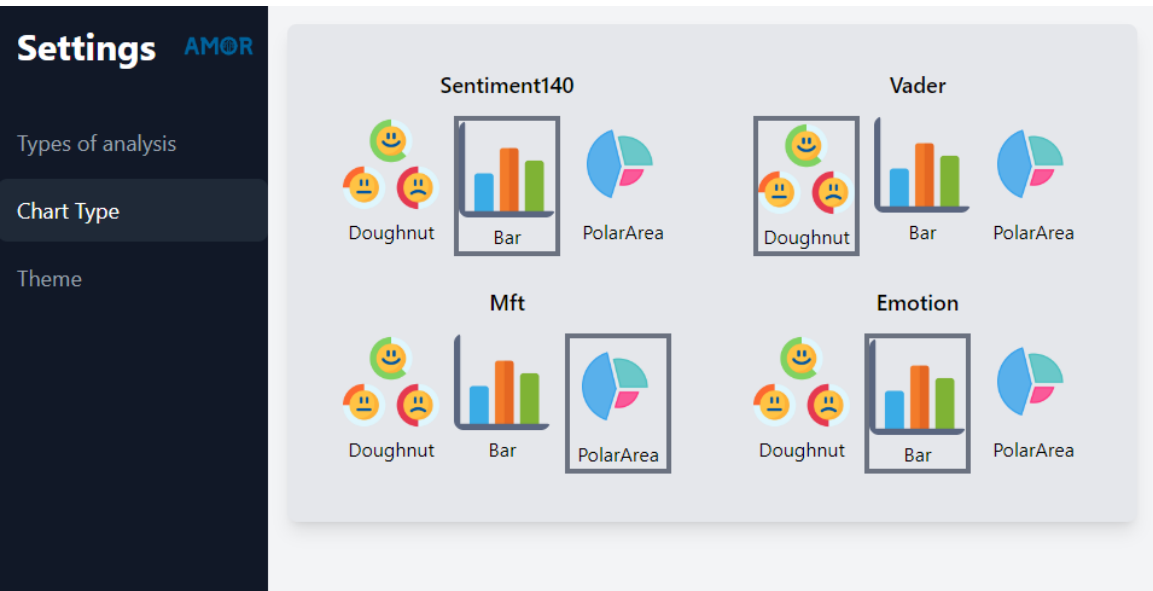


Figure 5.9: Chart Type

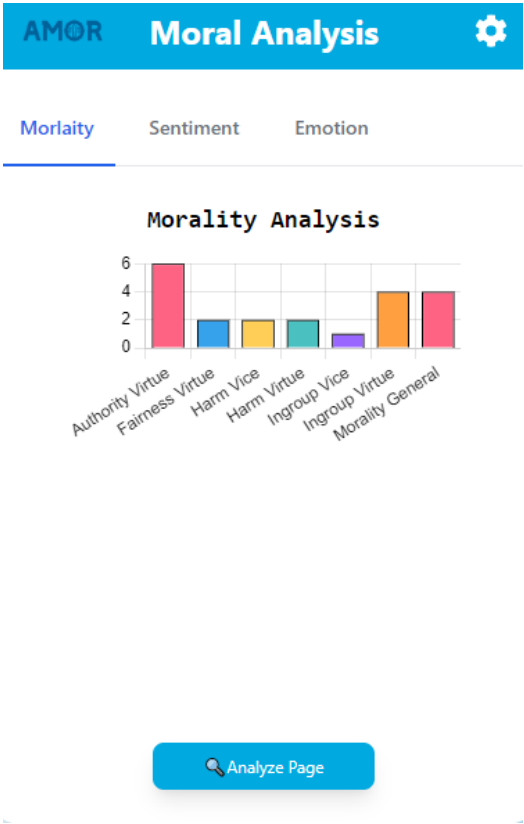


Figure 5.10: Moral Bar Chart

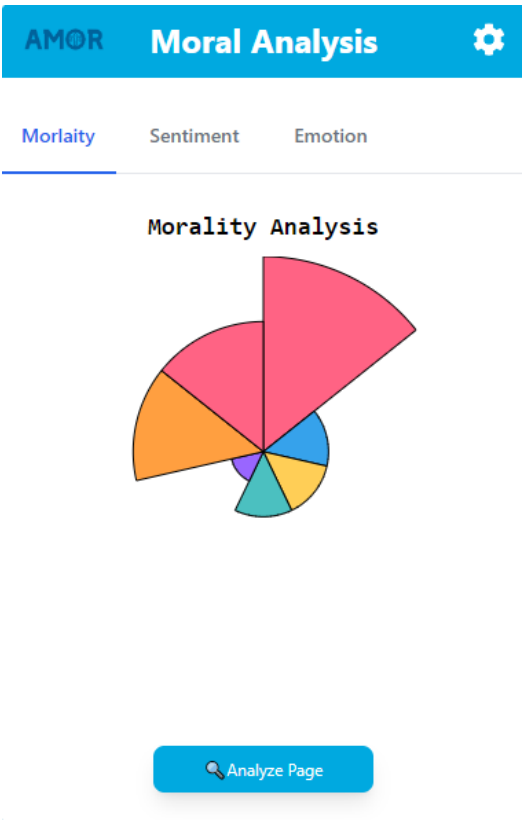


Figure 5.11: Moral Polar Area

following Fig. 5.13.

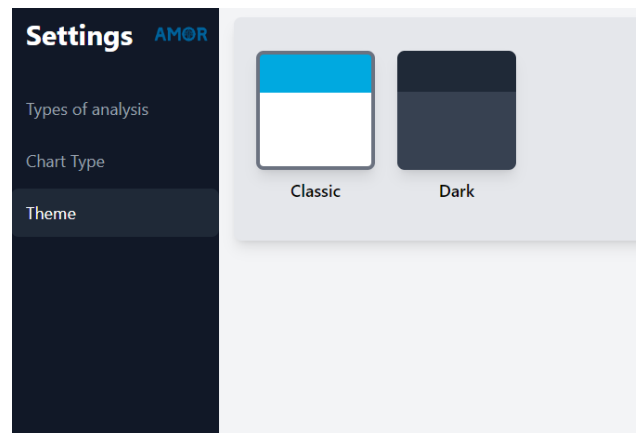


Figure 5.12: Theme Selection

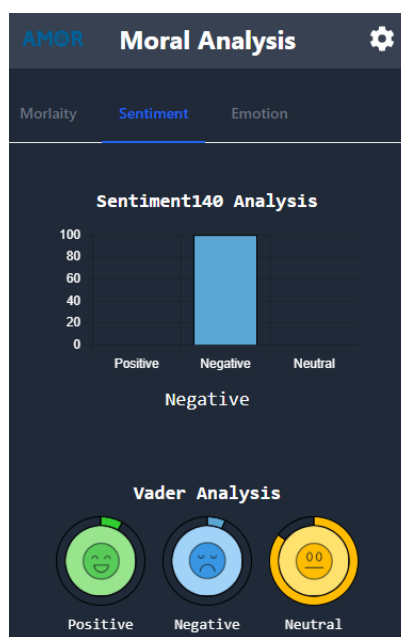


Figure 5.13: Dark theme

5.5 Using Firefox and Edge

In order for the user to be able to use the extension in other browsers, the same process has to be followed as for Google Chrome. Once the extension has been found and installed, it can be used with the same functionality in Microsoft Edge and Mozilla Firefox as shown below.

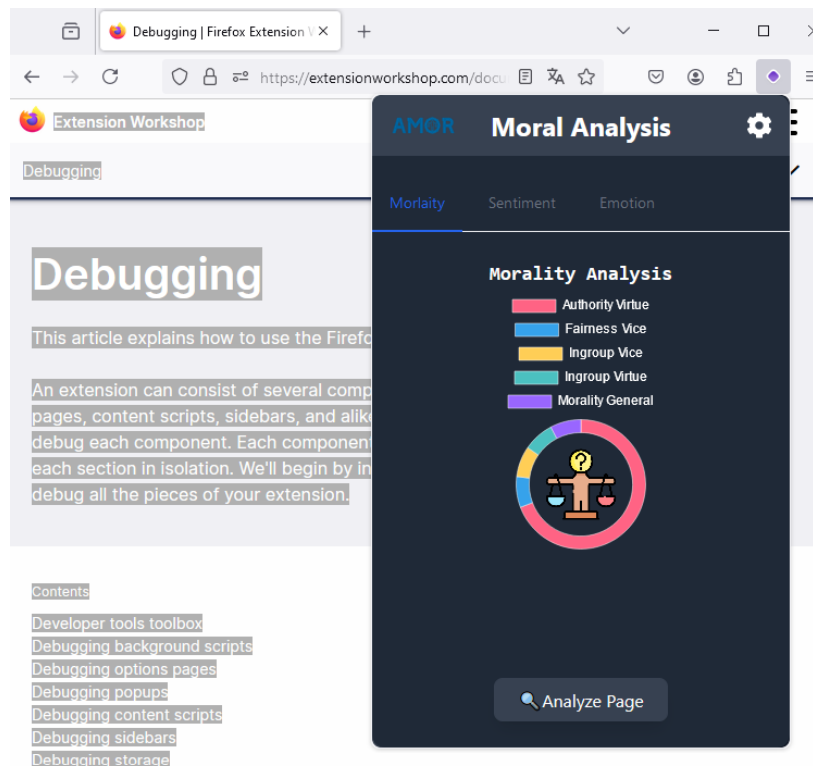


Figure 5.14: Mozilla Firefox Analysis

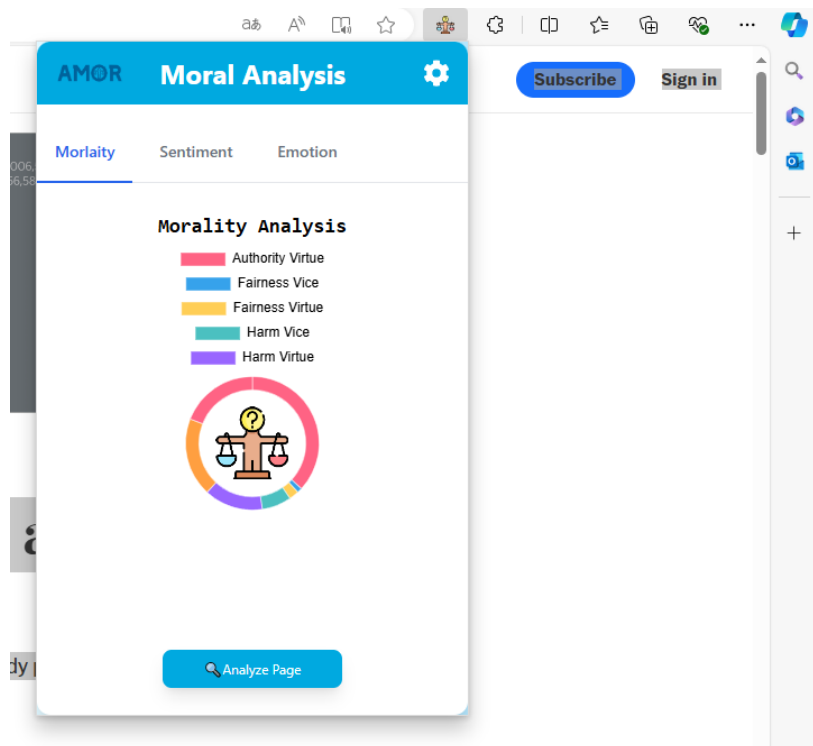


Figure 5.15: Microsoft Edge Analysis

5.6 Analysis comparative: Right-wing vs Left-wing

In this last section we are going to make an analysis of what a user might see when analysing two newspapers of totally opposite ideas in order to appreciate the differences that exist when performing an analysis of morality.

The newspapers to be used for the test will be 'The Mirror', known for being left-wing, and 'The Spectator', known for being right-wing. As it is an election in the United Kingdom the user will analyse two news items on this topic.

In the first one, we can see the analysis of the left-wing newspaper, where it can be observed a greater importance for social issues such as harm or fairness vice, leaving authority at a very low level as it predominates more in the right-wing ideas as it can be seen in the Fig. 5.16. In the second, we can see how the results of the analysis have completely changed.

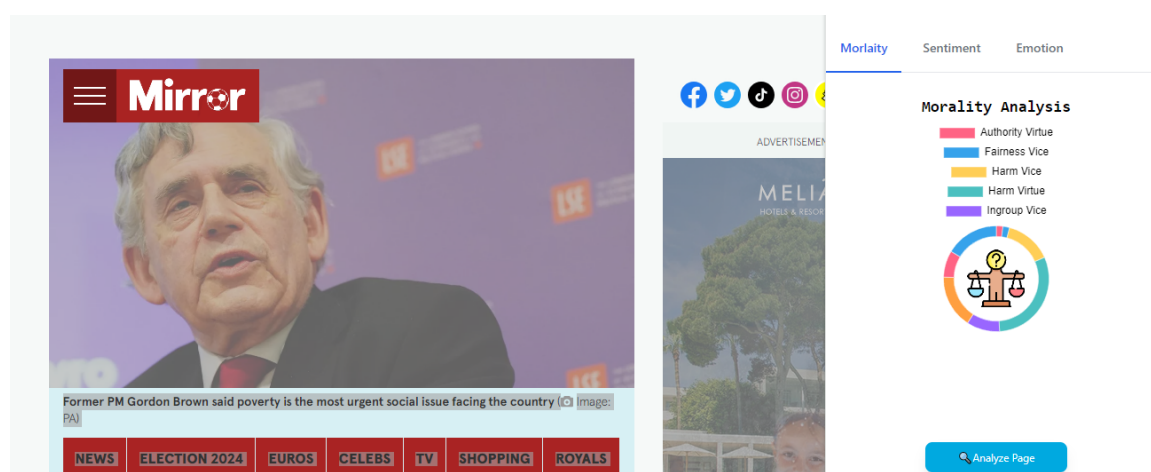


Figure 5.16: The Mirror Analysis

Now what predominates most is authority, a clearly right-wing idea, since it highlights the respect and importance of authority and traditions, as can be seen in Fig. 5.17.



Figure 5.17: The Spectator Analysis

Conclusions and future work

This chapter will describe the conclusions extracted from this project and our thoughts on future work.

6.1 Conclusions

This chapter will be dedicated to the detailed conclusions obtained after the implementation of the project. First, an analysis of the results achieved will be performed, highlighting the objectives that were successfully achieved. It will evaluate both the positive aspects and the challenges we have faced in making the extension.

In addition, it will provide a comprehensive summary of the lessons learned throughout the process, including a reflection on the strategies and methodologies that have been used. Then it will consider how these conclusions can help guide future work and make improvements to further enhance the project.

As a final thought on the project, it has far exceeded the expectations and plans we had at the beginning. From the start, we set ourselves several ambitious goals and designed a plan to achieve them. However, during the development of the project, we not only met

these goals but also identified additional opportunities to improve and expand our extension.

6.2 Achieved goals

All the work of the project has achieved a series of objectives. The creation of the extension, together with the possibilities it currently offers, has made the objectives achieved enriching and fruitful for the future. All these achieved goals can be summarized as follows:

Create a plugin capable of analyzing the morality and emotions of all the text on a page. The project's main objective has been creating an extension that can be used by any user and that can analyze the text of a page quickly and dynamically without having to do complicated tasks for it.

Represent the analysis results in a dynamic and clear way. The display of the analysis results has been a key factor in the project, allowing the user to see the results of the analysis in a simple and easily understandable way using a larger amount of graphics than the extension done previously in the other UPM project.

Using modern React technology. The previous extension was made with Javascript and html only, not allowing to bring out the full potential that could be obtained. The use of React has allowed a greater layer of complexity as well as an easier way to build the code and make our project based on modular components that allow us to add improvements in a simple way.

Create and develop the project with Plasmo technology. Using the Plasmo technology has been a challenge as it is a relatively new framework with little information about it. Even so, this challenge has provided us with a way to build extensions in a simple way and to take advantage of the many benefits that Plasmo has such as its environment or its storage.

Create a simple and beautiful visual interface. Simplicity in creating the interface was one of the clear objectives of the project. Adjusting to the current trend of simple interfaces has been one of the goals that had to be achieved for a better user experience.

Develop a configuration page allowing users to customize their experience. The creation of an options page was a clear objective since the other extension had a very minimal configuration that could be improved. The user's customized choice was very

important to achieve so that the user could choose both the type of analysis and the type of display of its extension.

Publish the extension in a cross-platform way. Plasmo has allowed us to publish the extension in any browser, working perfectly in all of them and making it possible for any user to use it.

6.3 Future work

This last section will show the steps to follow for future actions based on the work accomplished in the project.

- Add a larger number of text analysis types. Senpy has many more types of analysis that can be implemented in the extension. Senpy offers a wide range of additional analysis types that can be seamlessly integrated and implemented within the extension, enhancing its functionality and providing users with a more comprehensive toolkit for sentiment and emotion analysis.
- Create more options for the charts and display of the results. Using React allows us to make adding different types of charts really easy, its component-based structure allows us to create new ones and add as many charts as we want.
- Publish the extension in more browsers such as Opera or Safari to allow all users to use the extension without having to worry about the search engine of their choice.
- Implement Google Analytics to be able to analyze user interactions with the extension and to be able to track and control user data. Plasmo enables the import of Google Analytics, so its implementation would be a great improvement for the extension.
- The implementation of more buttons to be able to select a more focused type of text and obtain more useful results for companies or individuals. This type of text selection could be done on YouTube comments or tweets for example.
- Develop a system that allows the user to see the history of the analyses that have been performed and therefore be able to see the information from a broader perspective.

Impact of this project

In this appendix, we will examine the possible social, economic, environmental, and ethical impact of our application.

A.1 Social impact

This section details the main social impacts that our extension could have. In the digital age, many online platforms, such as social networks, blogs, and forums, generate massive textual data.

The developed extension makes it easier for researchers and analysts to extract and examine large amounts of data. Compared to traditional methods, our tool allows for a more detailed and deeper exploration of trends in publicly shared texts.

In addition, the extension can be a valuable tool for organizations and public institutions seeking to better understand society's concerns and opinions in real-time. It provides a solid basis for informed decision-making and the implementation of actions that benefit society as a whole.

A.2 Economic impact

In this section, we summarize the main economic impacts of our project on public institutions and, thus, the entire population.

The extension has significant potential for economic impact in several areas. By automating and streamlining the processing of large volumes of text, the time and resources required for studies and evaluations are considerably reduced, reducing companies' and organizations' operating costs.

In addition, our technology allows us to identify trends and patterns in real-time for various marketing strategies that a company may have.

A.3 Environmental impact

The environmental impact of our project is very low compared to other energy-intensive technologies. The use of our tool can contribute to the reduction of paper consumption and other physical resources thanks to digital storage.

So, although our project has an environmental impact, measures have been taken to reduce it as much as possible to contribute to digitization and energy efficiency.

A.4 Ethical implications

In terms of ethical impact, data privacy is a primary concern. Although our outreach is based on public data available on the Internet, it is important to ensure that personal information about users' data is not used without their consent.

At the same time, it is important to maintain transparency in how the results of the text analysis are used. It is necessary to inform all users and organizations about how data are used.

Economic budget

In this appendix, we analyze the economic budget related to the project. It is described below as it has been divided.

B.1 Physical resources

For the physical resources, a Mountain computer has been used to carry out all the project development. A Microsoft Windows 10 PRO operating system with a RAM memory of 8GB was used. It has an Intel® Core™ i5-5200U CPU @ 2.20GHz with an Intel® HD Graphics 5500.

The total cost of all this was approximately 555€.

B.2 Project structure

The following table shows the structure of the project.

Activity	Hours
Research	88
Learning technologies	91
Creating the structure	20
Extension development	136
Writing the project	95

B.3 Human resources

The development of the application and the writing of the project are the most time-consuming parts of the project. Creating the basic structure of the application took the least time. There were 430 hours of work, considering that the hourly rate is 14€, which makes a total cost of human resources of 6,020€.

B.4 Taxes

If the product is sold, it will incur a value-added tax in Spain equal to 21% of the product value.

Bibliography

- [1] Angelas. When to use (and not to use) asynchronous programming: 20 pros reveal the best use cases. *Stackify Blog*, March 2024. Accessed: 2024-06-29.
- [2] Óscar Araque. Design and Implementation of an Event Rules Web Editor. Trabajo fin de grado, Universidad Politécnica de Madrid, ETSI Telecomunicación, July 2014.
- [3] Feedier Blog. What is customer feedback analysis: the importance of text analysis. <https://feedier.com/blog/text-analysis-the-benefits-of-analyzing-customer-feedback/>, 2024. Accessed: 2024-06-29.
- [4] Christian Broms. Sentitude - sentiment analysis. <https://chromewebstore.google.com/detail/sentitude-sentiment-analy/khjckhocojcpjjfppdkahjcfacenljja?hl=es>, 2018. Accessed: 2024-06-29.
- [5] Chart.js. Chart.js options documentation. <https://www.chartjs.org/docs/latest/general/options.html>, 2024. Accessed: 2024-06-29.
- [6] Chrome for Developers. chrome.storage - chrome extensions api reference. <https://developer.chrome.com/docs/extensions/reference/api/storage?hl=es-419>, 2024. Accessed: 2024-06-29.
- [7] Chrome Web Store. Chrome Web Store Extensions and Themes. <https://chromewebstore.google.com/category/extensions>. Accessed: 2024-07-01.
- [8] Plasmo Corp. Introduction to Plasmo. <https://docs.plasmo.com/>, 2024. (Accessed: 2024-06-19).
- [9] Plasmo Corp. Plasmo documentation: Itero publisher. <https://docs.plasmo.com/itero/publisher>, 2024. Accessed: 2024-06-30.
- [10] Refsnes Data. Http methods get vs post. https://www.w3schools.com/tags/ref_httpmethods.asp, 2024. Accessed: 2024-06-30.
- [11] Manuel del Pozo Díaz. Sentiment&emotion analysis - gsi. <https://chromewebstore.google.com/detail/sentimentemotion-analysis/acchfklppcmgjleljcfffknkblgdgodiag?hl=es>, 2023. Accessed: 2024-06-30.
- [12] Fabio Duarte. Amount of data created daily (2024). <https://explodingtopics.com/blog/data-generated-per-day>, 2023. Accessed: 2024-06-29.

- [13] Artemij Fedosejev. *React. js essentials*. Packt Publishing Ltd, 2015.
- [14] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP'93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings* 7, pages 406–431. Springer, 1993.
- [15] gitconnected staff. Typescript tutorial: A guide to using the programming language. <https://gitconnected.com/post/typescript-tutorial-a-guide-to-using-the-programming-language-b787ce>, 2024. Accessed: 2024-07-01.
- [16] Google. Chrome extensions api reference. <https://developer.chrome.com/docs/extensions/reference/api>, 2024. Accessed: 2024-06-29.
- [17] IBM. Ibm watson natural language understanding. <https://www.ibm.com/products/natural-language-understanding>, 2024. Accessed: 2024-06-29.
- [18] Tailwind Labs Inc. Tailwind ui components for marketing blog sections. <https://tailwindui.com/components/marketing/sections/blog-sections>, 2024. Accessed: 2024-06-29.
- [19] Oltan Kochan. Installation - pnpm (fast, disk space efficient package manager. <https://pnpm.io/installation>, 2024. Accessed on 28/07/2024.
- [20] Mariko Kosaka. Intercambio de recursos de origen cruzado (cors). <https://web.dev/articles/cross-origin-resource-sharing?hl=es>, 2018. Accessed: 2024-06-30.
- [21] BuiltWith Pty Ltd. Tailwind css usage statistics. <https://trends.builtwith.com/framework/Tailwind-CSS>, 2024. Accessed: 2024-06-29.
- [22] Prateek Mehta. *Creating google chrome extensions*. Springer, 2016.
- [23] Inc. Meta Platforms. Introducing hooks. <https://legacy.reactjs.org/docs/hooks-intro.html>, 2024. Accessed: 2024-06-29.
- [24] Inc. Meta Platforms. React native. <https://reactnative.dev/>, 2024. Accessed: 2024-06-30.
- [25] Microsoft. Typescript documentation. <https://www.typescriptlang.org/docs/>, 2024. Accessed: 2024-06-30.
- [26] MonkeyLearn. Monkeylearn: No-code text analytics. <https://monkeylearn.com/>, 2024. Accessed: 2024-06-29.
- [27] Temitope Oyedele. How props work in react – a beginner’s guide. <https://www.freecodecamp.org/news/beginners-guide-to-props-in-react/>, 2022. Accessed: 2024-06-29.
- [28] PlasmoHQ. Plasmo framework documentation. <https://github.com/PlasmoHQ/docs>, 2024. Accessed: 2024-06-24.

- [29] Noel Rappin. *Modern CSS with tailwind: Flexible styling without the fuss*. Pragmatic Bookshelf, 2022.
- [30] J. Fernando Sánchez-Rada. Design and Implementation of an Agent Architecture Based on Web Hooks. Master's thesis, ETSIT-UPM, 2012.
- [31] J Fernando Sánchez-Rada, Oscar Araque, and Carlos A Iglesias. Senpy: A framework for semantic sentiment and emotion analysis services. *Knowledge-Based Systems*, 190:105193, 2020.
- [32] J. Fernando Sánchez. What is senpy? <https://senpy.readthedocs.io/en/latest/senpy.html>, 2019. Accessed: 2024-06-30.
- [33] J. Fernando Sánchez-Rada, Carlos Á. Iglesias, Ignacio Corcuera, and Óscar Araque. Senpy: A pragmatic linked sentiment analysis framework. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 735–742. IEEE, 2016.
- [34] Chart.js Team. Tooltip configuration. <https://www.chartjs.org/docs/latest/configuration/tooltip.html>, 2024. Accessed: 2024-06-29.
- [35] React Team. useeffect – react documentation. <https://es.react.dev/reference/react/useEffect>, 2024. Accessed: 2024-06-30.
- [36] React Team. usestate – react documentation. <https://es.react.dev/reference/react/useState>, 2024. Accessed: 2024-06-30.
- [37] Tecnobits. Las mejores extensiones de google chrome. https://www.google.com/url?sa=i&url=https%3A%2F%2Ftecnobits.net%2Fflas-mejores-extensiones-de-google-chrome%2F&psig=AOvVaw234BO_CyrmvRsucf9Oy-Qy&ust=1719643944342000&source=images&cd=vfe&opi=89978449&ved=0CBQQjhXqFwoTCIiq96Db_YYDFQAAAAAdAAAAABAJ, 2024. Accessed: 2024-06-24.
- [38] Mark Thomas. *React in action*. Simon and Schuster, 2018.
- [39] Thibault Tnt. Youtube sentiment analysis. <https://chromewebstore.google.com/detail/youtube-sentiment-analysis/fjgpfjohfddffjhkekipahjnfedioijad?hl=es>, 2024. Accessed: 2024-06-29.