**UNIVERSIDAD**

**TECNOLÓGICA NACIONAL**

**FACULTAD REGIONAL**

**RESISTENCIA**

***1er ENTREGA TRABAJO PRÁCTICO INTEGRADOR: GRAMÁTICA***

***INTEGRANTES:***

**Puebla Ignacio**

**Ruiz Diaz Dario Nahuel**

**Sotelo Facundo**

**Manfredi Ezequiel**

**ASIGNATURA:** “Sintaxis y Semántica de los Lenguajes”

**CARRERA:** Ingeniería en Sistemas de Información

**CICLO LECTIVO:** Año 2024 - 1er Cuatrimestre

**INSTITUCIÓN:** “Universidad Técnica Nacional, Facultad Regional de Resistencia” (UTN FRRe)

**LUGAR:** Provincia del Chaco - Ciudad de Resistencia

**FECHA DE ENTREGA:** Domingo 28/04/2024

***ÍNDICE***

[***MATRIZ DE HABILIDADES*** 3](#_Toc164937199)

[***IDENTIDAD DEL GRUPO*** 3](#_Toc164937200)

[***ALIANZA*** 4](#_Toc164937201)

[***REGISTRO*** 5](#_Toc164937202)

[ **DOMINGO 14 DE ABRIL:** 5](#_Toc164937203)

[ **SÁBADO 20 DE ABRIL:** 5](#_Toc164937204)

[ **OTROS REGISTROS:** 5](#_Toc164937205)

# ***MATRIZ DE HABILIDADES***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Integrantes** | **Inglés** | **Documentación** | **Programación** | **Investigación** | **Video** | **Testing** |
| **Puebla Ignacio** | ● | ★ | ● | ● |  | ● |
| **Manfredi Ezequiel Luis** | ● | ● | ★ | ● | ● | ★ |
| **Ruiz Diaz Dario Nahuel** |  | ● | ★ | ● | ★ | ★ |
| **Sotelo Facundo** | ★ | ★ | ● | ● |  | ● |

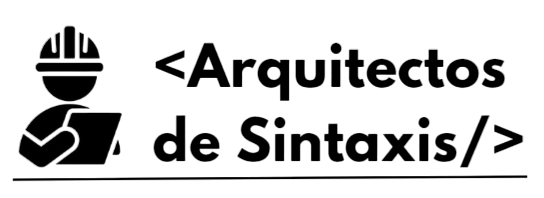
* **Espacio en blanco (“ ”): No sabe/no está interesado**
* **Estrella (★): Lo hace/es experto**
* **Punto (●) : Es competente/lo puede hacer pero no es su mayor interés**

# ***IDENTIDAD DEL GRUPO***

* **Nombre del grupo:**

“Arquitectos de Sintaxis”

* **Logo:**



# ***ALIANZA***

Los integrantes del equipo Arquitectos de Sintaxis: Manfredi Ezequiel, Ruiz Diaz Dario Nahuel, Puebla Ignacio y Sotelo Facundo, se comprometen a:

Reunirse al menos una vez por semana los días sábado o domingo entre las 17:00 y 18:00 hs vía Discord a través de una llamada en la cual poder conversar, decidir, debatir e informar con respecto al progreso del trabajo práctico integrador y las tareas de cada integrante. A su vez, se hará uso de un grupo de WhatsApp como medio de comunicación habitual entre los integrantes del equipo y para la planificación de las reuniones.

Todo ello con el objetivo de, no solo aprobar el Trabajo Práctico Integrador (y si dios quiere aprobar la materia de forma directa para no rendir final) sino también el de realizar el mejor trabajo posible en el tiempo dado.

Acordamos que la organización del equipo será la siguiente: nos dividiremos y trabajaremos en dos grupos (por lo menos durante esta primera entrega del TPI). El primer grupo de dos integrantes (Manfredi Ezequiel y Ruiz Diaz Dario Nahuel) será el encargado mayormente de todo lo referido a la programación y el diseño; mientras que el segundo grupo de dos integrantes (Puebla Ignacio y Sotelo Facundo) será el encargado de registrar y documentar, ayudar con el diseño de gramática, y brindar asistencia en lo posible. Ello incluye identificar errores, ayudar con el desarrollo del programa si fuese necesario, asistir con el diseño y la creación de la gramática, entre otras tareas.

Si bien cada grupo tiene su función, no serán grupos aislados que trabajen individualmente en todo momento, ya que se utilizarán las vías de comunicación antes mencionadas para informar, consultar, pedir ayuda u opinión de los demás integrantes del equipo; así como cualquier otra situación que surja durante la realización del Trabajo Práctico Integrador.

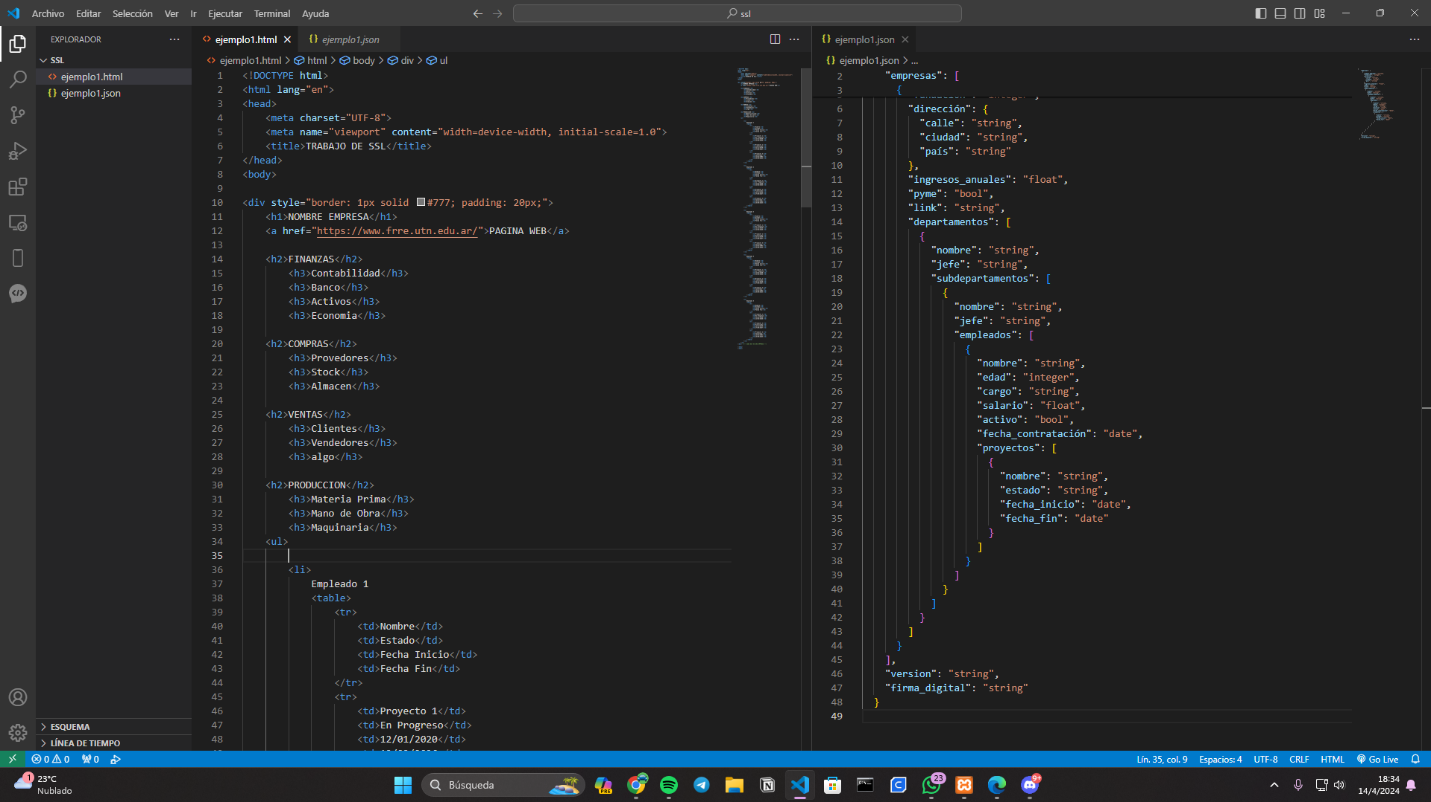
# ***REGISTRO***

## **DOMINGO 14 DE ABRIL**

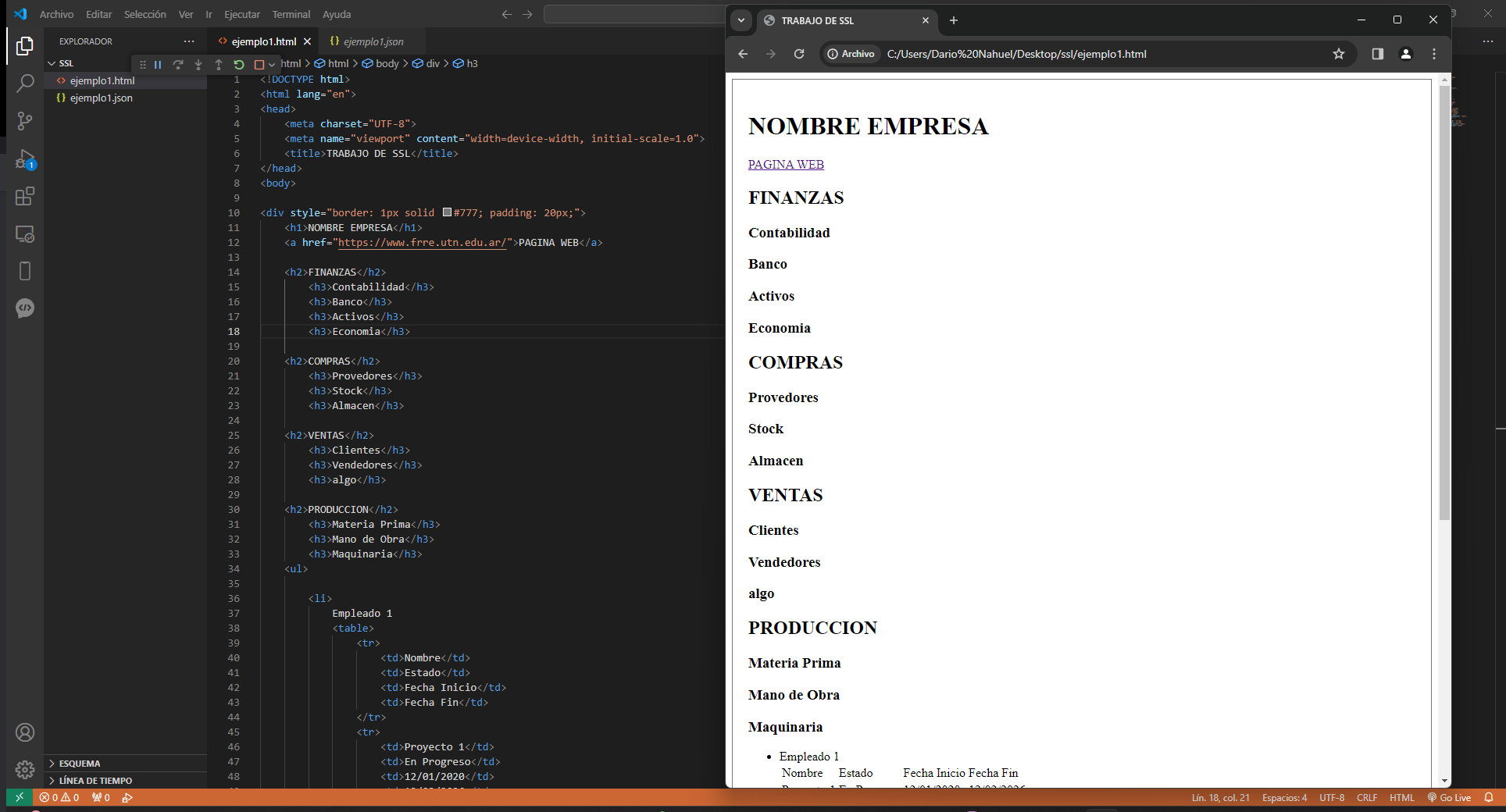
Nos reunimos vía Discord para planificar la primera parte de la entrega del TPI. Para ello, comenzamos haciendo un documento compartido de Google docs, como borrador de lo que luego sería nuestro documento Word. Lo primero que hicimos en dicho documento fué la matriz de habilidades. Nos pusimos de acuerdo para definir cada categoría y luego, individualmente, cada uno fue definiendo las capacidades y cualidades en las que se destaca. Luego de realizar la matriz de habilidades nos pusimos a planificar cómo sería la estructura de nuestro trabajo. Definimos los roles de cada integrante del equipo y nos dividimos en dos grupos: uno encargado de diseñar y programar; y el otro de llevar el registro, la documentación y asistir en lo posible.

Una vez definidos los equipos, hablando acerca del desarrollo del trabajo, uno de los integrantes (Ruiz Diaz Nahuel) propuso una interfaz gráfica simple como idea para la presentación de la información, la cual dividiese la pantalla de la siguiente forma: a la izquierda, iría el contenido del archivo de entrada; a la derecha, el contenido del archivo de salida; y debajo, habría una consola que muestre en pantalla los errores (en caso de haberlos).

El **grupo 1** encargado de la programación tomó un ejemplo de archivo en JSON para ver su estructura básica y cómo se podría traducir a HTML con el fin de usarla como base para la realización de la gramática en HTML (el lenguaje con el cual definimos que se trabajaría la gramática).



Captura del proceso de traducción de JSON a HTML del ejemplo de gramática



Captura de prueba de ejecución del código HTML

Mientras tanto, en el **grupo 2** encargado del registro y la documentación, Sotelo Facundo registraba el trabajo realizado por el grupo 1 mediante capturas de pantalla y la descripción general de lo que se iba haciendo; y Puebla Ignacio redactaba la Alianza del equipo y aportaba al registro del día.

* **Registros Extras:**

Al final de la reunión, se acordó crear un repositorio de GitHub para que cada integrante tuviera un mejor y más cómodo acceso al código y al documento Word. A su vez, se realizaron modificaciones al server de Discord añadiendo nuevos canales de “código” y “fotos” para poder compartir a todo el grupo las capturas de pantalla y los archivos.

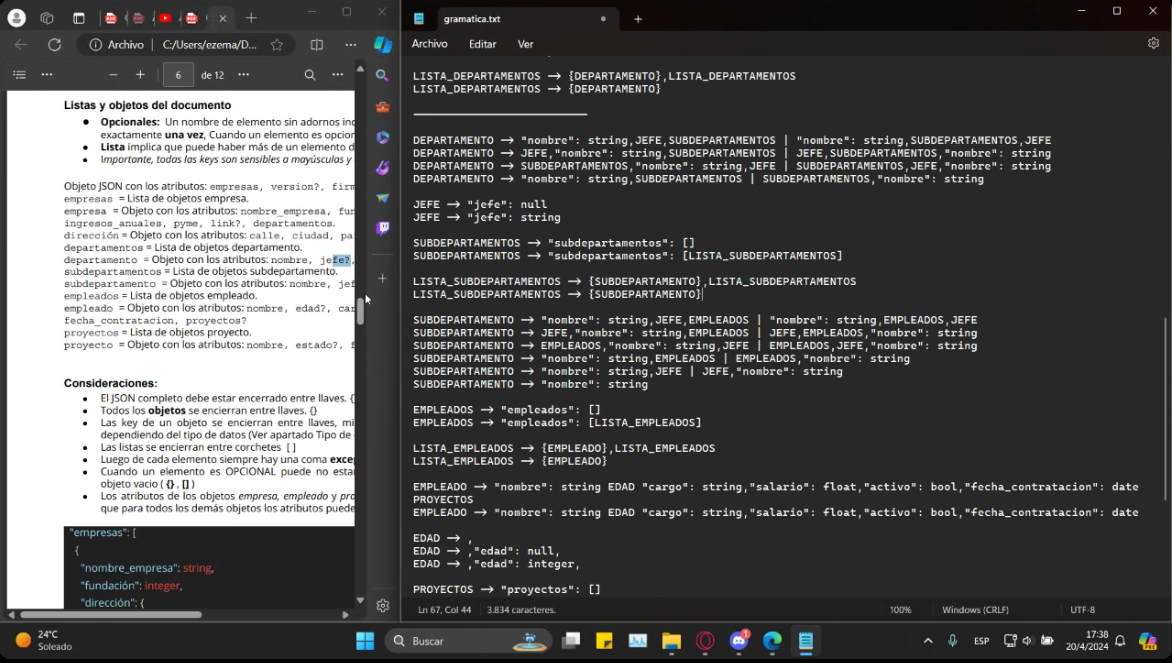
En un determinado momento de la reunión, el integrante Ruiz Diaz Nahuel, diseñó el logo del grupo. Un logo simple pero que tuviera relación con el nombre del grupo y que le diera una mayor identidad.

El integrante Manfredi Ezequiel es el autor del nombre del grupo “Arquitectos de Sintaxis”.

## **SÁBADO 20 DE ABRIL**

Nos conectamos vía llamada de Discord. El objetivo de la reunión inicialmente era reunirnos para diseñar la gramática entre todos, sin embargo, para sorpresa del resto del resto de los integrantes del equipo, resulta que Manfredi Ezequiel ya había hecho la gramática. Entonces empezamos a analizar la gramática en busca de errores o faltas y para mejorar su legibilidad.

El integrante Puebla Ignacio completó las premisas de la alianza del equipo y se corrigió algunos aspectos de su redacción para hacerla más legible. A su vez, también corrigió la redacción y añadió algo de contenido al registro. Mientras que el integrante Sotelo Facundo realizaba el registro y tomaba captura a la gramática.



Captura de la gramática sin corregir

En un punto de la reunión, luego de revisar la gramática diseñada por el alumno Manfredi Ezequiel y terminar la documentación, dedicamos las siguientes hora y media-dos horas a practicar la producción de gramáticas para el parcialito del día 25 de abril.

* **GRAMÁTICA:**

Gramatica JSON: G(N,T,P,Σ)

N: {

CONTENIDO; EMPRESAS; VERSION; FIRMA; LISTA\_EMPRESAS; EMPRESA; NOMBRE\_EMPRESA; FUNDACION;

DIRECCION; INGRESOS\_ANUALES; PYME; LINK; DEPARTAMENTOS; ATRIBUTOS\_DIRECCION;

LISTA\_DEPARTAMENTOS; CALLE; CIUDAD; PAIS; DEPARTAMENTO; NOMBRE; JEFE; SUBDEPARTAMENTOS;

LISTA\_SUBDEPARTAMENTOS; SUBDEPARTAMENTO; EMPLEADOS; LISTA\_EMPLEADOS; EMPLEADO; EDAD;

CARGO; SALARIO; ACTIVO; FECHA\_CONTRATACION; PROYECTOS; LISTA\_PROYECTOS; PROYECTO; ESTADO;

FECHA\_INICIO; FECHA\_FIN;

}

T: {

{; }; ,; [; ]; "; ; :; string; integer; bool; float; date; null; version; firma\_digital; empresas; nombre\_empresa;

fundacion; dirección; ingresos\_anuales; pyme; link; departamentos; calle; ciudad; pais; nombre; jefe;

subdepartamentos; empleados; edad; cargo; salario; activo; fecha\_contratacion; proyectos; estado; fecha\_inicio;

fecha\_fin;

}

P: {

Σ -> {CONTENIDO}

// combinatoria del orden de los campos considerando los campos opcionales VERSION y FIRMA

CONTENIDO -> EMPRESAS, VERSION, FIRMA | EMPRESAS, FIRMA, VERSION

CONTENIDO -> VERSION, FIRMA, EMPRESAS | VERSION, EMPRESAS, FIRMA

CONTENIDO -> FIRMA, EMPRESAS, VERSION | FIRMA, VERSION, EMPRESAS

CONTENIDO -> EMPRESAS, VERSION | EMPRESAS, FIRMA

CONTENIDO -> VERSION, EMPRESAS | FIRMA, EMPRESAS

CONTENIDO -> EMPRESAS

VERSION -> "version": null

VERSION -> "version": string

FIRMA -> "firma\_digital": null

FIRMA -> "firma\_digital": string

EMPRESAS -> "empresas": [LISTA\_EMPRESAS]

// recursion de empresas

LISTA\_EMPRESAS -> {EMPRESA}, LISTA\_EMPRESAS

LISTA\_EMPRESAS -> {EMPRESA}

// manteniendo el orden de los campos considerando los campos opcionales DIRECCION y LINK

EMPRESA -> NOMBRE\_EMPRESA, FUNDACION, DIRECCION, INGRESOS\_ANUALES, PYME, LINK, DEPARTAMENTOS

EMPRESA ->NOMBRE\_EMPRESA, FUNDACION, INGRESOS\_ANUALES, PYME, LINK, DEPARTAMENTOS

EMPRESA -> NOMBRE\_EMPRESA, FUNDACION, DIRECCION, INGRESOS\_ANUALES, PYME, DEPARTAMENTOS

EMPRESA -> NOMBRE\_EMPRESA, FUNDACION, INGRESOS\_ANUALES, PYME, DEPARTAMENTOS

NOMBRE\_EMPRESA -> "nombre\_empresa": string

FUNDACION -> "fundacion": integer

DIRECCION -> "direccion": null

DIRECCION -> "direccion": {}

DIRECCION -> "direccion": {ATRIBUTOS\_DIRECCION}

INGRESOS\_ANUALES -> "ingresos\_anuales": float

PYME -> "pyme": bool

LINK -> "link": null

LINK -> "link": string

DEPARTAMENTOS -> "departamentos": [LISTA\_DEPARTAMENTOS]

// combinatoria del orden de los campos

ATRIBUTOS\_DIRECCION -> CALLE, CIUDAD, PAIS | CALLE, PAIS, CIUDAD

ATRIBUTOS\_DIRECCION -> CIUDAD, CALLE, PAIS | CIUDAD, PAIS, CALLE

ATRIBUTOS\_DIRECCION -> PAIS, CALLE, CIUDAD | PAIS, CIUDAD, CALLE

CALLE -> "calle": string

CIUDAD -> "ciudad": string

PAIS -> "pais": string

// recursion de departamentos

LISTA\_DEPARTAMENTOS -> {DEPARTAMENTO}, LISTA\_DEPARTAMENTOS

LISTA\_DEPARTAMENTOS -> {DEPARTAMENTO}

// combinatoria del orden de los campos considerando el campo opcional JEFE

DEPARTAMENTO -> NOMBRE, JEFE, SUBDEPARTAMENTOS | NOMBRE, SUBDEPARTAMENTOS, JEFE

DEPARTAMENTO -> JEFE, NOMBRE, SUBDEPARTAMENTOS | JEFE, SUBDEPARTAMENTOS, NOMBRE

DEPARTAMENTO -> SUBDEPARTAMENTOS, NOMBRE, JEFE | SUBDEPARTAMENTOS, JEFE, NOMBRE

DEPARTAMENTO -> NOMBRE, SUBDEPARTAMENTOS | SUBDEPARTAMENTOS, NOMBRE

NOMBRE -> "nombre": string

JEFE -> "jefe": null

JEFE -> "jefe": string

SUBDEPARTAMENTOS -> "subdepartamentos": [LISTA\_SUBDEPARTAMENTOS]

// recursion de subdepartamentos

LISTA\_SUBDEPARTAMENTOS -> {SUBDEPARTAMENTO}, LISTA\_SUBDEPARTAMENTOS

LISTA\_SUBDEPARTAMENTOS -> {SUBDEPARTAMENTO}

// combinatoria del orden de los campos considerando los campos opcionales JEFE y EMPLEADOS

SUBDEPARTAMENTO -> NOMBRE, JEFE, EMPLEADOS | NOMBRE, EMPLEADOS, JEFE

SUBDEPARTAMENTO -> JEFE, NOMBRE, EMPLEADOS | JEFE, EMPLEADOS, NOMBRE

SUBDEPARTAMENTO -> EMPLEADOS, NOMBRE, JEFE | EMPLEADOS, JEFE, NOMBRE

SUBDEPARTAMENTO -> NOMBRE, EMPLEADOS | EMPLEADOS, NOMBRE

SUBDEPARTAMENTO -> NOMBRE, JEFE | JEFE, NOMBRE

SUBDEPARTAMENTO -> NOMBRE

EMPLEADOS -> "empleados": null

EMPLEADOS -> "empleados": []

EMPLEADOS -> "empleados": [LISTA\_EMPLEADOS]

// recursion de empleados

LISTA\_EMPLEADOS -> {EMPLEADO}, LISTA\_EMPLEADOS

LISTA\_EMPLEADOS -> {EMPLEADO}

// manteniendo el orden de los campos considerando los campos opcionales EDAD y PROYECTOS

EMPLEADO -> NOMBRE, EDAD, CARGO, SALARIO, ACTIVO, FECHA\_CONTRATACION, PROYECTOS

EMPLEADO -> NOMBRE, EDAD, CARGO, SALARIO, ACTIVO, FECHA\_CONTRATACION

EMPLEADO -> NOMBRE, CARGO, SALARIO, ACTIVO, FECHA\_CONTRATACION, PROYECTOS

EDAD -> "edad": null

EDAD -> "edad": integer

CARGO -> "cargo": string

SALARIO -> "salario": float

ACTIVO -> "activo": bool

FECHA\_CONTRATACION -> "fecha\_contratacion": date

PROYECTOS -> "proyectos": null

PROYECTOS -> "proyectos": []

PROYECTOS -> "proyectos": [LISTA\_PROYECTOS]

// recursion de proyectos

LISTA\_PROYECTOS -> {PROYECTO}, LISTA\_PROYECTOS

LISTA\_PROYECTOS -> {PROYECTO}

// manteniendo el orden de los campos considerando los campos opcionales ESTADO y FECHA\_FIN

PROYECTO -> NOMBRE, ESTADO, FECHA\_INICIO, FECHA\_FIN

PROYECTO -> NOMBRE, ESTADO, FECHA\_INICIO

PROYECTO -> NOMBRE, FECHA\_INICIO, FECHA\_FIN

ESTADO -> "estado": null

ESTADO -> "estado": string

FECHA\_INICIO -> "fecha\_inicio": date

FECHA\_FIN -> "fecha\_fin": null

FECHA\_FIN -> "fecha\_fin": date

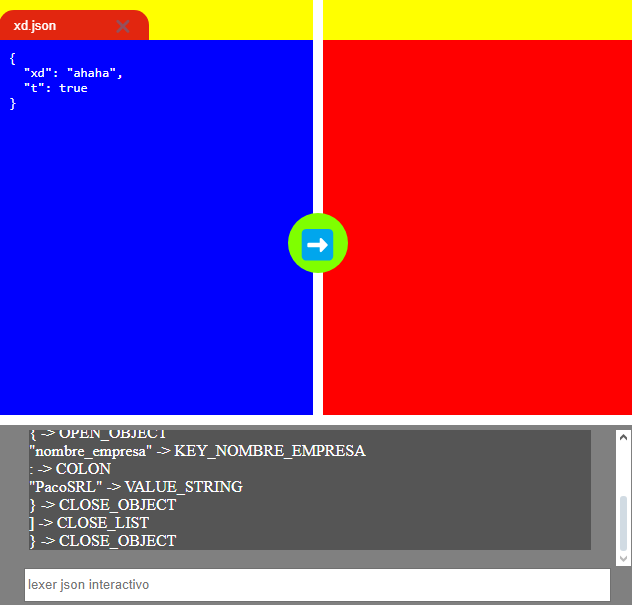
}

* **DOMINGO 26 DE MAYO**

Nos reunimos vía Discord a través de una llamada para realizar la documentación del Lexer desarrollado por el miembro del equipo Manfredi Ezequiel.

* **Interfaz gráfica:**

Teniendo en cuenta tanto el Lexer como el futuro Parser, se creó una interfaz provisional dividida en dos partes. En la parte superior se colocaron dos cuadros (izquierdo y derecho) separados por una línea que muestran el archivo de entrada y el de salida respectivamente (lo que nos servirá para el Parser); mientras que en la parte inferior se encuentra un terminal en el cual se ingresa algún input (por ejemplo una palabra), y ejecuta el Lexer devolviendo como salida los tokens identificados.



Captura de la interfaz provisional



Interfaz luego de algunas modificaciones

La estructura de la interfaz está hecha en html y la parte funcional de la interfaz (la captura de datos, la impresión de archivos, etc) se hizo en JavaScript. A su vez, el Lexer está hecho en Python, al igual que un servidor. Dicho servidor se encarga de administrar las peticiones y brindar la información necesaria tanto al Lexer como al futuro Parser.

Lo que más nos interesa de la interfaz (por estar relacionado con el Lexer) es la terminal de la parte inferior. Resumidamente la terminal funciona de la siguiente forma: en base a un input se envía la información al servidor, el cual se la suministra al Lexer, este analiza y devuelve los resultados que son recibidos en la terminal y se la procesa para mostrarla amigablemente al usuario. Además, todos los errores se mostrarán en la salida clasificados según el tipo y regla no cumplida.

* **¿Cómo se hizo el Lexer?**

Se utilizó la documentación y ejemplos brindados por la cátedra como guía para empezar a desarrollar el lexer. Primero se definieron las palabras clave; los token; y luego, para cada tipo de token, las expresiones regulares, las cuales nos sirven para detectar los token según su tipo.

La parte más complicada del desarrollo del Lexer fué el definir las expresiones regulares, en donde las más problemáticas fueron las expresiones regulares para los strings, las fechas y la url; pero tras probar varias opciones se consiguió que funcione.

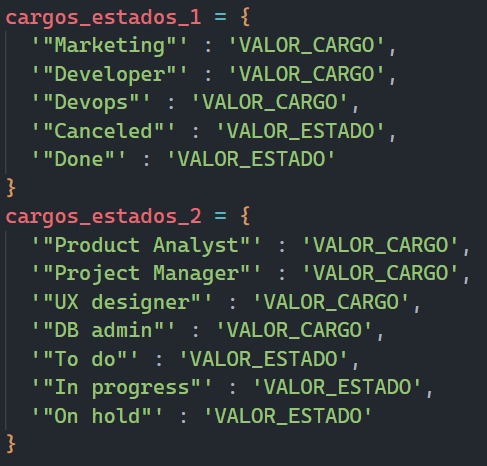
El reconocimiento y la notificación de errores funciona de la siguiente manera: A partir del input dado se analiza la totalidad del mismo acumulando todos los errores de este y los tokens debidamente reconocidos.

Los errores pueden ser de dos tipos: no\_tokens y tokens que no cumplen las reglas. Los primeros son caracteres que no hicieron match con ninguna expresión regular, mientras que los segundos son lexemas que hicieron match con alguna expresión regular. Se clasifican por tipo de token y cada uno tiene reglas que cumplir. Dichas reglas explicitan el porqué del error detectado.

Reconocimientos de tokens:

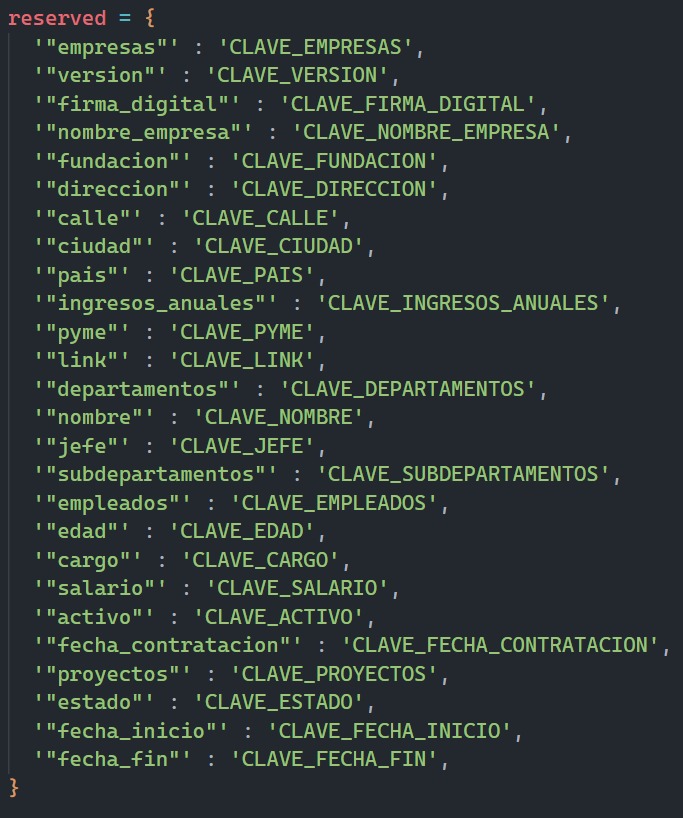
**Conjunto:** lexemas que no cambian y están agrupadas en cierto token.

Los conjuntos que definimos en el Lexer son:

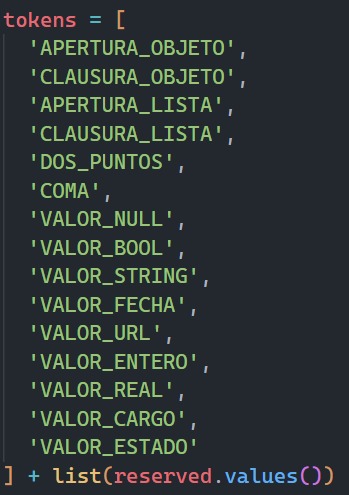


**Palabras reservadas:** lexemas que no cambian y por cada una hay un token.

Las palabras reservadas definidas en el Lexer son:



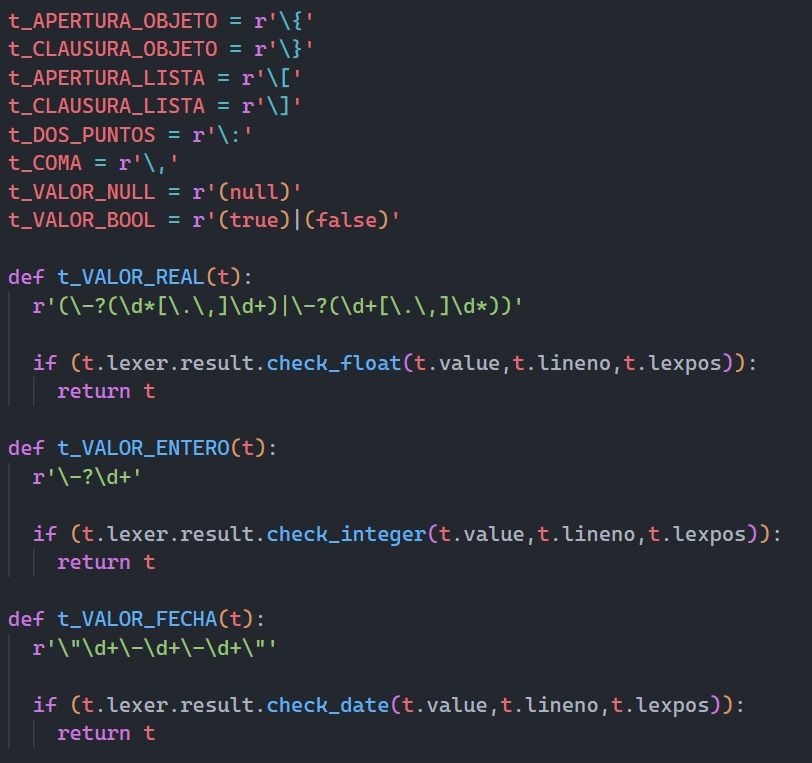
**Tokens:** Los tokens definidos son:



**Expresión regular**: regla generalizada que reconoce ciertos tipos de lexemas.

Ciertas expresiones regulares permiten el reconocimiento de ciertos lexemas que se considerarían errores pero que el usuario podría cometer. Estos lexemas de más sirven para detectar y clasificar el no cumplimiento de ciertas reglas para su posterior notificación.

Las expresiones regulares definidas en el Lexer son:





* **Requisitos de ejecución:**

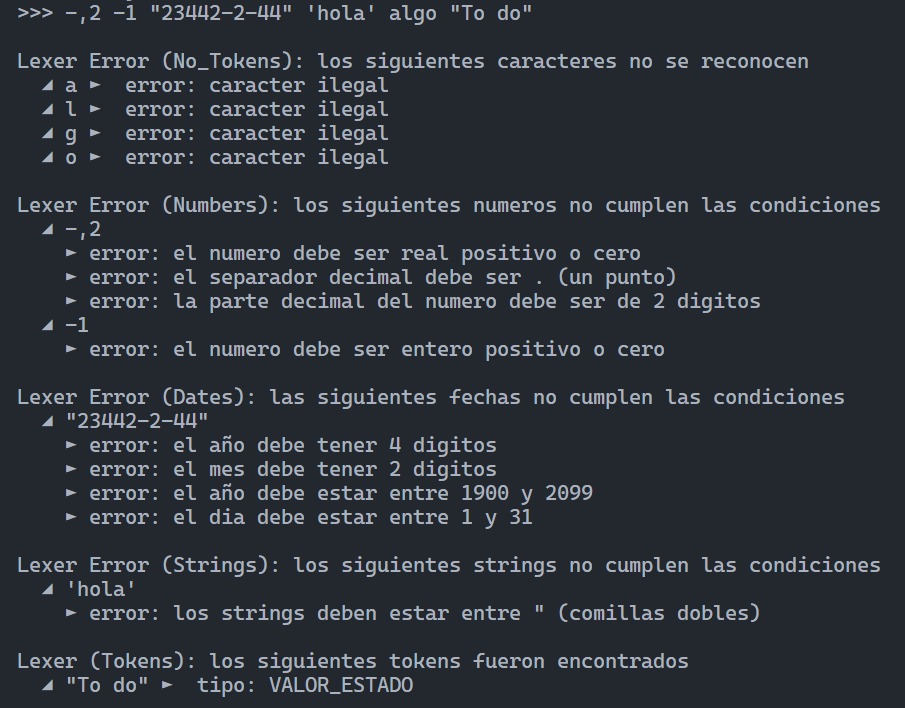
El programa fué ejecutado en Windows 11 y los requisitos son:

* Python versión. 3.12.3
* Paquete PLY (Python Lex-Yacc) ver 3.11
* Flask versión 3.0.3
* **Instalación del programa:**
* Instalar Python
* Luego en el terminal escribir: pip install ply y luego pip install Flask
* **Formas de ejecutar el programa:**

Contamos con dos maneras de ejecutar el programa: la primera es por terminal y la segunda por UI (interfaz de usuario).

Por terminal:

* En el terminal en el directorio src ejecutar: python ./lex.py
* Se ejecutará el Lexer interactivo, donde se puede ingresar los input y se mostraran los resultados del análisis Lexico.



Por interfaz gráfica:

* En el terminal en el directorio src ejecutar: flask --app ./server.py run
* Saldrá una url en donde estará la UI ejecutándose.
* Para terminar los procesos presione ctrl + C
* **Registros extra:**

Para la entrega del parser se tiene pensado añadir un botón en la interfaz para descargar el archivo de salida generado. También un botón en el medio para ejecutar el futuro Parser y mostrar el archivo de salida dado el archivo de entrada ingresado. Además, se planea mejorar el diseño de la interfaz.

La idea final que se tiene en mente para la interfaz es que el usuario pueda arrastrar un archivo de entrada, luego ejecutar el programa y que este genere un archivo de salida el cual pueda descargar pulsando un botón.