



---

# Procesadores de Lenguajes

## Memoria de prácticas

---

Entregable Final

ESCUELA SUPERIOR DE INFORMÁTICA  
UNIVERSIDAD DE CASTILLA-LA MANCHA

ELENA HERVÁS MARTÍN  
ENRIQUE CEPEDA VILLAMAYOR  
RAÚL BERNALTE SÁNCHEZ  
ELENA MARÍA RUIZ IZQUIERDO

## Contenido

1.	Presentación del problema	3
2.1.	Descripción del lenguaje	4
2.2.	Toma de decisiones a la hora de diseñar el lenguaje	6
3.	Descripción sintáctica del lenguaje mediante notación EBNF	7
4.	Análisis léxico: Tabla de tokens	8
5.	Construcción del procesador y Diagramas de T	10
6.	Tareas del analizador semántico	11
7.	Análisis sintáctico descendente	12
8.	Análisis sintáctico ascendente	12
9.	Tests	13
10.	Página Web	13
	• Cómo crear una receta	13
	• Introducir receta	14
	• Recetario	15
11.	Ejecución de procesador hecho con JFlex+CUP	16
12.	Ejecución de procesador hecho con ANTLR	16
13.	Ejecución de la web	17
14.	Puntuación de los miembros del equipo	18

## 1. Presentación del problema

Una empresa Castellano Manchega, dirigida por un joven emprendedor de la región, apodado el Dr. Doofenshmirtz manchego, ha ideado un robot de cocina que facilita la labor de la elaboración de menús a las familias "ocupadas" del siglo XXI. Su máquina, la Gachaneitor 1.0, tiene un funcionamiento similar a otra máquina que ya existe en el mercado (la Thermomix) pero conceptualmente presenta una novedad: se basa en una comunidad que puede **compartir conocimiento, facilitando el intercambio**, así como **permitir la gestión autónoma del conocimiento** disponible. Para su construcción, ha contado con la inestimable colaboración de un grupo de brillantes alumnos de la asignatura de "Procesadores de Lenguajes" de la Escuela Superior de Informática.

Debido al futuro éxito inestimable de la nueva Gachaneitor 1.0, cuatro alumnos de la asignatura de "Procesadores de Lenguajes" de la Escuela Superior de Informática queremos ser los pioneros en crear un lenguaje (*GachaLanguage*) para describir las recetas en un formato legible y agradable para el usuario.

Usando este lenguaje, se desarrollará una web que presente la lista de recetas creadas hasta el momento por los usuarios de la comunidad. Esta web dispondrá de una pestaña donde los usuarios podrán introducir sus recetas tanto en un archivo como en texto, para que el procesador de lenguajes muestre si la receta es válida tanto sintáctica como semánticamente al ser reconocida por el procesador y finalmente convierta este lenguaje a un formato JSON.

- Si no hay ningún tipo de error, el procesador convertirá la receta del lenguaje fuente *GachaLanguage* a un formato JSON que posteriormente será procesado en la web. Esta decisión está apoyada en el hecho de que JSON es un formato estándar para la transmisión de información en el mundo del desarrollo web y nos aporta interoperabilidad para otros usos o aplicaciones.
- En caso contrario, se mostrará un error (léxico, sintáctico o semántico) y se avisará al usuario de éste mismo.

La práctica se desarrollará en dos partes diferenciadas:

- El **diseño del lenguaje fuente (*GachaLanguage*) y de los procesadores de lenguajes**, para dar lugar a una salida válida para el propósito propuesto. Se han hecho dos procesadores de lenguaje para dos tipos distintos de análisis sintáctico:
  - Un análisis ascendente (on the fly) hecho con JFlex, Cup y Java.
  - Un análisis descendente hecho con ANTLR4 y Python.
- El **tratamiento de la salida y la posterior visualización de los datos obtenidos** de la receta. Para esta parte, se ha decidido desarrollar una página web que muestre los resultados de la práctica. Esto es una forma de lograr conocimientos de programación web, conocimientos que no son tratados en la intensificación de Computación.

## 2. Solución y diseño del lenguaje

Para solucionar el problema propuesto, se ha diseñado *GachaLanguage*, un lenguaje específico del dominio (Domain Specific Language o DLS) definido específicamente para definir recetas

para la máquina Gachaneitor. Esta solución además incluye la construcción de dos procesadores de lenguajes que procesan como entrada este lenguaje y generan la receta en formato JSON, para que pueda ser fácilmente guardado en una base de datos y finalmente mostrado en un recetario vía web.

## 2.1. Descripción del lenguaje

Las distintas secciones (todas obligatorias) que se encuentran en el lenguaje *GachaLanguage* se describen a continuación:

- La palabra "receta" junto a dos llaves (una abierta y una cerrada) que indica el comienzo de una receta, ya que pueden existir varias.
- Dentro de estas llaves, están situadas las partes que conforman una receta en el siguiente orden fijo:
  - El **nombre** de la receta, que se define mediante la palabra "nombre" (opcional) y a continuación el nombre de la misma entre comillas (e.g. nombre "Puré"). Los usuarios más avanzados tienen la opción de no utilizar la palabra "nombre" para ahorrar tiempo.
  - La **descripción** de esta receta, que se define mediante la palabra "descripción" (opcional) seguida de una descripción entre corchetes (e.g. descripción [Puré de calabaza bastante bueno]). La palabra descripción puede ser eliminada por los usuarios más avanzados que ya conozcan de la estructura de la receta.
  - El **tiempo de la receta**, que se define mediante la palabra "tiempo" (opcional) seguida de dos llaves, en cuyo interior se deben definir dos elementos.
    - El tiempo total, es decir, el tiempo que tarda la máquina en preparar los alimentos. Este primer elemento se declara mediante la palabra "total" junto con el que la máquina invierte en procesar los alimentos, es decir, un número natural junto a la unidad de tiempo que se elija: h (horas), min(minutos) o seg(segundos) (e.g. total 1 h).
    - El tiempo de preparación, el tiempo que una persona debe invertir en preparar los alimentos para que la máquina posteriormente los procese. Este segundo elemento se declara mediante la palabra "preparacion" junto con el tiempo invertido en la misma, es decir un número natural junto con la unidad de tiempo que se elija: horas, minutos o segundos (e.g. preparacion 5 min).
  - Los **ingredientes**, que es la sección donde se declaran todos los ingredientes usados en la receta en forma de lista. Esta sección se define con la palabra "ingredientes" (opcional), seguida de una llave abierta y otra cerrada. Para cada ingrediente, hay que añadir su nombre, en mayúscula y entre comillas, seguido de dos puntos (opcional) y la cantidad total a usar del producto (número natural + unidad) (e.g. "Leche": 2 l). Podemos separar los elementos de estos ingredientes mediante punto y coma, aunque no es necesario. A continuación, se indican las posibles unidades que pueden acompañar al ingrediente:
    - **Masa:** kg, g mg

- **Volumen:** l, ml
- **Unidad:** ud

Hay que destacar que si un ingrediente no está declarado en esta zona no puede ser usado en los pasos posteriores. Además, no se podrán usar cantidades mayores a las declaradas en esta sección, ni tampoco unidades de una magnitud distinta (e.g. masa (g) y volumen (l)).

Es posible que un ingrediente esté declarado varias veces, con cantidades distintas. En ese caso, se sumarán las cantidades si las unidades que acompañan al ingrediente en cuestión son de la misma magnitud (masa, volumen o unidad).

- Los **pasos**, que es la sección donde se definen los pasos a usar por la máquina Gachaneitor en la receta. Se definen con la palabra “pasos” (opcional) seguido de llaves. La declaración de cada paso se compone de:
  - Un *guion*, que anuncia el comienzo de un nuevo paso.
  - Un *verbo* de la acción a realizar por la máquina. Estos verbos se escriben en **minúscula** y se clasifican según el tipo de acción que se realiza. Dependiendo de este tipo de acción, requerirán unos parámetros u otros en el paso. Los verbos que pueden ser usados se muestran en la siguiente tabla.

Verbos de movimiento (tiempo y velocidad requeridos)	Verbos de cocción (tiempo y temperatura requeridos)	Verbos de persona (sin tiempo)
licuar triturar batir dejar reposar amasar mezclar	cocinar al vapor escalfar hervir guisar sofreir	pelar trocear mover

- A continuación, le sigue una enumeración de *ingredientes usados en el paso*. Como en la sección anterior, los nombres de los ingredientes serán escritos comenzando por mayúscula y entre comillas seguido de dos puntos (opcional) y la cantidad de producto usada (número natural + unidad). Cada ingrediente tiene al final un punto y coma que le separa del resto.
- En el **caso de los verbos de cocción**, hay que añadir dos campos después de los ingredientes: *tiempo y temperatura*.
  - El tiempo se indicará con la palabra “tiempo” seguido de un número natural y la unidad de tiempo correspondiente (h, min, seg) (e.g. 15 min).
  - La temperatura se declara con la palabra “temperatura” seguido de un número natural y la unidad de temperatura correspondiente (°C, °F) (e.g. 40 °C).
- En el **caso de los verbos de movimiento**, hay que añadir dos campos después de los ingredientes: el *tiempo y la velocidad*.

- El tiempo se indicará con la palabra “tiempo” seguido de un número natural y la unidad de tiempo correspondiente (h, min, seg) (e.g. 15 min).
- La velocidad se indicará con la palabra “velocidad” seguido de un número del 1 al 9 precedido por un 0 (e.g. velocidad 07).

Una receta perteneciente al lenguaje podría ser la siguiente:

```
receta {
  nombre "Guacamole"
  descripcion [Guacamole rapido y sencillo, receta tradicional. Perfecto como entrante
acompañado de totopos]
  tiempo {
    total 5 min
    preparacion 5 min
  }

  ingredientes {
    "Aguacate": 3 ud;
    "Zumo de limón": 10 g;
    "Tomate maduro": 1 ud;
    "Cebolla": 70 g;
    "Sal": 1 g;
    "Aceite": 10 ml;
  }

  pasos {
    - trocear "Cebolla": 70 g; "Tomate maduro": 1 ud;
    - triturar "Cebolla": 70 g; "Tomate maduro": 1 ud; 1 min velocidad 07
    - pelar "Aguacate": 3 ud;
    - mezclar "Cebolla": 70 g; "Tomate maduro": 1 ud; "Aguacate": 3 ud; "Zumo de
limón": 10 g; "Sal": 1 g; "Aceite": 10 ml; 4 min velocidad 03
  }
}
```

## 2.2. Toma de decisiones a la hora de diseñar el lenguaje

Para crear el lenguaje *GachaLanguage*, se han tenido en cuenta tanto las necesidades de los usuarios noveles de la aplicación como las de los usuarios más expertos.

Por ejemplo, los campos de una receta van precedidos de palabras identificadoras del campo, para aquellos usuarios que usan por primera vez nuestro lenguaje, ya que se gana en legibilidad.

Por otro lado, al ser el orden de los campos fijo, se da la opción de incluir o no (opcional) estas palabras, para aquellos usuarios con más experiencia que quieran describir de una manera rápida las recetas.

También se han puesto como opcionales tanto los dos puntos entre nombre de ingrediente y cantidad, y el punto y coma separador en la lista de ingredientes, para favorecer la legibilidad a los usuarios principiantes y dar rapidez con respecto a la velocidad de escritura a los usuarios expertos.

En el caso de declarar los diferentes pasos que tiene que seguir la máquina, se ha optado por un token guion que actúa como separador, sin connotaciones sintácticas. Si se declaran los pasos en la misma línea puede ser confuso para el ojo humano, ya que el final de un paso anterior pudo ser tanto un ingrediente como una velocidad o una temperatura. De esta manera, el usuario puede identificar rápidamente el comienzo y final de un paso y, por tanto, su contenido.

Los nombres de los ingredientes comienzan en letra mayúscula y se escriben entrecomillados, mientras que las palabras reservadas del lenguaje se escriben en minúscula. Esto simplifica la máquina de estados del analizador léxico, a la vez que ayuda de manera visual al usuario en la búsqueda en la receta de los distintos elementos.

Para fijar y diferenciar los números de la velocidad se ha optado por representarlas mediante un número de dos dígitos. El primero es un 0 y el segundo es un número del 1 al 9.

También, se da la opción de describir más de una receta en la misma cadena de texto, ya que si se tienen guardadas más de una receta en el mismo archivo de texto es más costoso para el usuario separarlas, tanto en términos de tiempo como en comodidad y facilidad de uso.

### 3. Descripción sintáctica del lenguaje mediante notación EBNF

```

INICIO ::= RECETA {RECETA}

RECETA ::= [receta] '{' NOMBRE DESCRIPCION TIEMPO_RECETA DEF_INGREDIENTES PASOS '}'

NOMBRE ::= [nombre] ' ' IDENT_NOMBRES ' '

DESCRIPCION ::= [descripcion] CONTENIDO_DESCRIPCION

TIEMPO_RECETA ::= [tiempo] '{' total TIEMPO preparacion TIEMPO '}'

DEF_INGREDIENTES ::= [ingredientes] '{' INGREDIENTES '}'

INGREDIENTES ::= INGREDIENTE [';'] {INGREDIENTE [';']}

INGREDIENTE ::= IDENT_NOMBRES [':] CANTIDAD

PASOS ::= [pasos] '{' PASO {PASO} '}'

PASO ::= '-' (PASO_MOV | PASO_COC | PASO_PER)

PASO_MOV ::= VERBO_MOV INGREDIENTES TIEMPO VELOCIDAD
  
```

```
PASO_COC ::= VERBO_COC INGREDIENTES TIEMPO TEMPERATURA
PASO_PER ::= VERBO_PER INGREDIENTES

VERBO_MOV ::= licuar | triturar | batir | dejar reposar | amasar | mezclar
VERBO_COC ::= cocinar al vapor | escalfar | hervir | guisar | sofreir
VERBO_PER ::= pelar | trocear | moler

CANTIDAD ::= NUMERO UNIDAD_CANTIDAD
TIEMPO ::= NUMERO UNIDAD_TIEMPO
TEMPERATURA ::= temperatura NUMERO UNIDAD_TEMP
VELOCIDAD ::= velocidad DIGITO_VELOCIDAD

UNIDAD_CANTIDAD ::= mg | g | kg | ml | l | ud
UNIDAD_TEMP ::= °C | °F
UNIDAD_TIEMPO ::= h | min | seg
IDENT_NOMBRES ::= '"" MAYUS {MINUS|' }' ""'
MAYUS ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | Ñ | O | P | Q | R | S | T | U | V | W | X | Y | Z | Á | É | Í | Ó | Ú | Ñ
MINUS ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | ñ | o | p | q | r | s | t | u | v | w | x | y | z | á | é | í | ó | ú | ñ
DIGITO_INICIAL ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
DIGITO_VELOCIDAD ::= 0 | DIGITO_INICIAL
NUMERO ::= DIGITO_INICIAL { DIGITO_INICIAL | 0 }
CONTENIDO_DESCRIPCION ::= '[' CADENA ']'
COMENTARIO ::= /* CADENA */
OTROS_CARACTERES ::= , | ')' | '(' | '{' | '}' | - | _
CADENA ::= {MAYUS | MINUS | NUMERO | ' ' | OTROS_CARACTERES}
```

#### 4. Análisis léxico: Tabla de tokens



Token	Patrón / Expr. Regular	Lexema ejemplo
nombre	"nombre"	nombre
comillas	""	"
receta	"receta"	receta
llave_abierta	"{"	{
llave_cerrada	"}"	}
tiempo	"tiempo"	tiempo
total	"total"	total
preparacion	"preparacion"	preparacion
descripcion	"descripcion"	descripcion
contenido_descripcion	"[" cadena "]"	[ Muy rico ]
ingredientes	"ingredientes"	ingredientes
dospuntos	":"	:
puntoycoma	","	;
guion	"_"	-
pasos	"pasos"	pasos
unidad_cantidad	"mg"   "g"   "kg"   "ml"   "l"   "ud"	kg
unidad_tiempo	"h"   "min"   "seg"	min
unidad_temperatura	"°C"   "°F"	°C
temperatura	"temperatura"	temperatura
velocidad	"velocidad"	velocidad
verbo_mov	"licuar"   "triturar"   "batir"   "dejar reposar"   "amasar"   "mezclar"	licuar
verbo_coc	"cocinar al vapor"   "escalfar"   "hervir"   "guisar"   "sofreir"	hervir

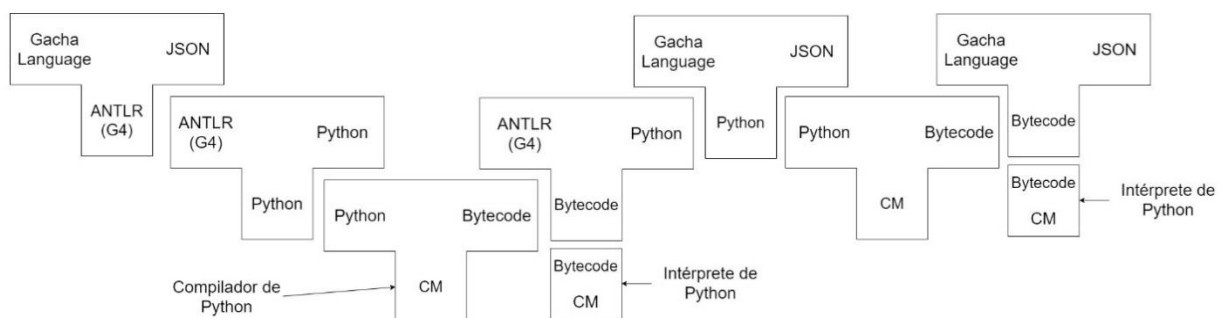
<b>verbo_per</b>	"pelar"   "trocear"   "moler"	pelar
<b>ident_nombre</b>	[A-Z][ a-z   ' '   'á'   'é'   'í'   'ó'   'ú'   'Á'   'É'   'Í'   'Ó'   'Ú'   ñ   Ñ ]*	Tomate
<b>numero</b>	[1-9][0-9]*	120
<b>digito_velocidad</b>	0[1-9]	05
<b>cadena</b>	([A-Z]   [a-z]   ' '   [0-9]   ,   'y'   '('   '{'   '}'   -   _   'á'   'é'   'í'   'ó'   'ú'   'Á'   'É'   'Í'   'Ó'   'Ú'   ñ   Ñ )*	La receta pollo al horno con licor 43 está muy buena y es fácil de hacer

## 5. Construcción del procesador y Diagramas de T

A continuación, se muestra el diagrama de T, que representa los pasos que se producen al utilizar ANTLR como procesador de lenguaje que convierte el lenguaje Gachaneitor a JSON.

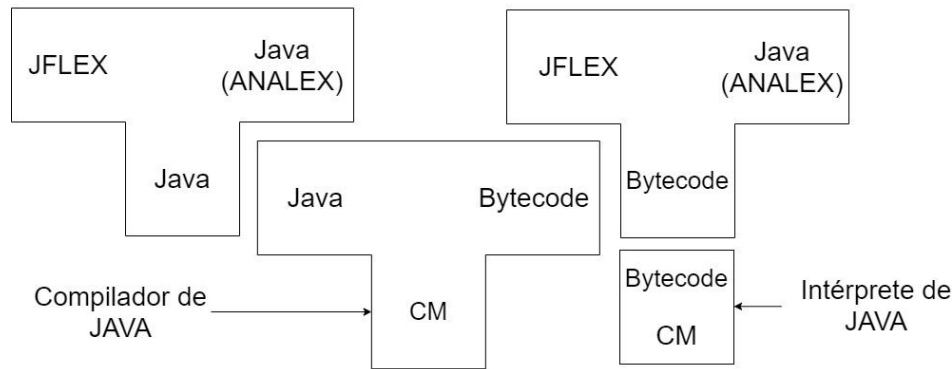
En él se puede observar como el procesador en ANTLR tiene dos fases. La primera consiste en convertir la especificación léxica y sintáctica el archivo ANTLR en un procesador de lenguaje implementado en Python, que junto con archivos de análisis de semántica conforman el procesador de lenguaje.

Una vez obtenido el procesador, este se compila a bytecode para que después el intérprete de Python lo ejecute, generando así nuestro lenguaje de recetas en formato JSON a partir de un archivo GachaLanguage.

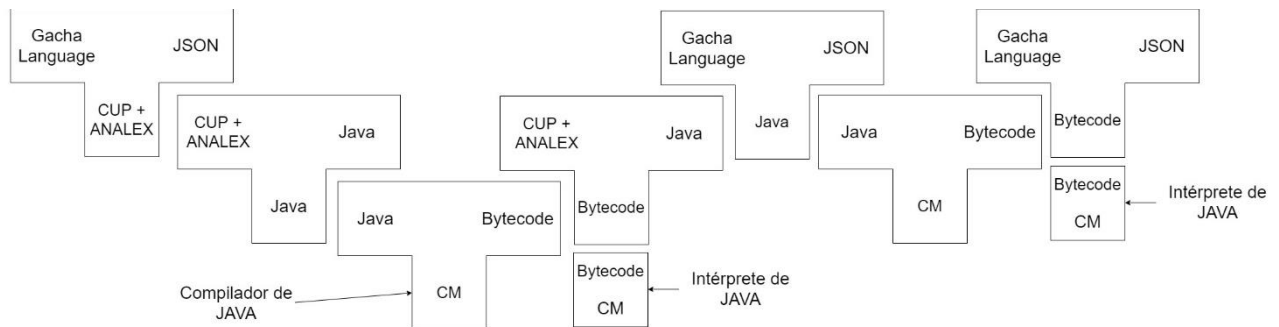


Dicho procesador también se ha realizado con JFlex y CUP.

En la primera figura se representa el proceso, por el cual, se transforma una especificación JFLEX en un analizador léxico en el lenguaje JAVA.



Una vez obtenido el analizador léxico, este se usa junto con el analizador sintáctico y semántico generado por CUP, de modo que se obtendrá el lenguaje de recetas convertido en JSON.



## 6. Tareas del analizador semántico

El analizador semántico de nuestro lenguaje se encarga de comprobar que se cumplen las siguientes condiciones en el mismo:

- Los ingredientes usados a lo largo de la receta son los mismos que los declarados al principio en el apartado de la sección “ingredientes”, y, por lo tanto, no debe haber ningún ingrediente declarado que no se use finalmente en los pasos de la receta.
- El tiempo total de la receta debe ser igual a la suma de los tiempos parciales de los pasos de la receta que tienen duración.
- Las cantidades de los ingredientes usados en los pasos de la receta deben corresponder con una cantidad igual o menor a la que se define de cada uno de ellos en la sección de declaración de ingredientes. Para poder llevar esta comprobación a cabo, se ha construido un módulo conversor de unidades, ya que se contempla que el usuario exprese las cantidades en unidades diferentes en los distintos pasos que ponen la receta.
- Relacionado con el punto anterior, también se comprueba que las unidades de las cantidades sean equivalentes. Un ejemplo de ello sería, que para expresar la cantidad de un líquido que se va a utilizar, en un punto de la receta se expresase con el volumen (litros) y en otro con el peso (kilos), lo cual no sería válido para nuestro lenguaje.

El incumplimiento de estas condiciones provocaría un error en el procesamiento del lenguaje y haría que una cadena no pertenezca al lenguaje, aunque sea válida y correcta en términos léxicos y sintácticos.

## 7. Análisis sintáctico descendente

Para realizar un análisis descendente se ha creado un procesador de lenguaje con la herramienta ANTLR. Esta herramienta permite un análisis léxico, sintáctico y semántico de las cadenas de entrada.

- Primeramente, ha permitido definir en un archivo expresiones regulares asociadas a los elementos del lenguaje (tokens) para el análisis léxico.
- Posteriormente, permite definir un archivo que hace uso de estos tokens y crea un analizador sintáctico descendente LL(k) haciendo transformaciones a la gramática como la eliminación de recursividad a izquierdas.
- Por último, ha permitido realizar un análisis semántico posterior a la construcción del árbol sintáctico en Python. Esto ha permitido desacoplar la definición sintáctica del lenguaje a las comprobaciones semánticas, pudiendo haber hecho ésta en múltiples lenguajes reutilizando el mismo archivo de la gramática.

Este análisis semántico se ha hecho por medio de un *Listener*, clase que permite realizar un recorrido en profundidad del árbol posteriormente a su construcción y que ha permitido hacer las comprobaciones semánticas anteriormente descritas. Este análisis controla errores y proporciona una retroalimentación para que el usuario sepa corregir la receta y ésta pueda ser válida.

La herramienta proporciona también una recuperación automática ante errores sintácticos no críticos, por defecto, que nos ha servido para algunas casuísticas de nuestra gramática.

Este procesador de lenguajes ha sido el utilizado en la web, ya que el poder realizar estos tres análisis en Python permite una integración sencilla con ésta, también desarrollada en Python (explicada en detalle posteriormente).

## 8. Análisis sintáctico ascendente

El análisis ascendente se ha realizado con las herramientas JFlex y CUP en el lenguaje JAVA. Para esta tarea se han desarrollado un analizador léxico, y un sintáctico. Además, se ha hecho un análisis semántico a la vez que se hacía el sintáctico.

Por un lado, JFlex es una herramienta que ha permitido crear el analizador léxico mediante la declaración de expresiones regulares para cada uno de los tokens definidos.

JFlex es fácilmente integrable con CUP, herramienta que genera analizadores sintácticos ascendentes LALR mediante la creación de tablas y una implementación con pila. CUP también permite definir esquemas de traducción, es decir, la inclusión de reglas semánticas en cualquier sitio de la parte derecha de una producción.

Esta solución cuenta con producciones de error, que implementan mensajes personalizados, indicando la línea y columna donde se encuentra el fallo, para que el usuario tenga una retroalimentación adecuada y sepa cómo corregir la receta para obtener una salida válida. Además, se ha implementado un proceso de recuperación de errores para algunos errores no

críticos del lenguaje, en los que se informa al usuario por pantalla de los fallos en forma de avisos, pero se continúa con el análisis del lenguaje.

## 9. Tests

Para hacer una eficiente evaluación de los procesadores, se han escrito varias test suites en paralelo a la construcción de los procesadores. Estas suites nos permiten asegurar que los dos procesadores se comportan de forma similar ante la misma entrada de texto.

- Para el procesador de JFlex y CUP se ha utilizado el paquete **JUnit**.
- Para el procesador de ANTLR se ha utilizado el módulo de tests unitarios **Unitttest**.

Para cada procesador se añaden dos archivos adicionales. Uno contiene las pruebas correspondientes a las diferentes casuísticas del lenguaje con respecto al análisis sintáctico, mientras que otro comprueba los diferentes casos de una cadena correctamente sintáctica, pero con distintos errores en las comprobaciones semánticas descritas en la sección *Tareas del analizador semántico*. Estos tests hacen uso de recetas escritas y añadidas en la carpeta *doc* del repositorio.

La explicación de cómo ejecutar estos tests se encuentra descrita tanto en el README de cada procesador como en las secciones dedicadas a la ejecución de éstos en este documento.

## 10. Página Web

Para desarrollar la página web se ha utilizado Python como lenguaje principal junto con Django, un framework que facilita el desarrollo de páginas web mediante el uso de plantillas HTML propietarias. Además, la lógica de dominio está modelada en SQL mediante el uso del ORM de Django, que permite definir tablas y sus relaciones de una forma sencilla, así como el desarrollo intuitivo de sentencias SQL.

Por otra parte, se ha usado el lenguaje Javascript junto con la librería JQuery para hacer peticiones asíncronas al servidor, y modificar dinámicamente la estructura (DOM) de las plantillas HTML y así poder mostrar errores por pantalla. Esta página web está compuesta por varias partes que serán explicadas en detalle a continuación:

- **Cómo crear una receta**

Esta parte actúa como un manual de usuario detallado y permite visualizar el lenguaje receta y cómo se puede definir una para poder así contribuir en la comunidad de Gachaneitor. Informa de los distintos modos de introducir una receta, así como de las pautas del vocabulario y la sintaxis del lenguaje para poder construir una receta.

**GACHANEITOR**[Cómo crear una receta](#) [Introducir receta](#) [Recetario](#)

## Cómo crear una receta

Es importante saber qué tipo de pasos se deben seguir para añadir una receta a este repositorio.

### Modo de introducción

Existen dos modos para introducir las recetas:

- Importando un archivo de texto desde la máquina.
- En la propia página, escribir la receta a mano en un área de texto.

En caso de que se introduzcan ambas opciones, se le dará mayor prioridad al archivo subido, por lo que el procesador analizará la receta del archivo txt.



### Cómo escribir la receta

*Gachalanguage* es un lenguaje específico del dominio en el que cada receta se define de la siguiente forma:

```
receta {  
  nombre "Puré"  
  descripción [Esta es una receta muy rica]  
  tiempo {  
    total 20 min /* Comprobar con suma de tiempos  
    parciales (Opcional) */  
    preparacion 5 min /* No hace falta comprobar, porque  
    son verbos de persona */  
  }  
}
```

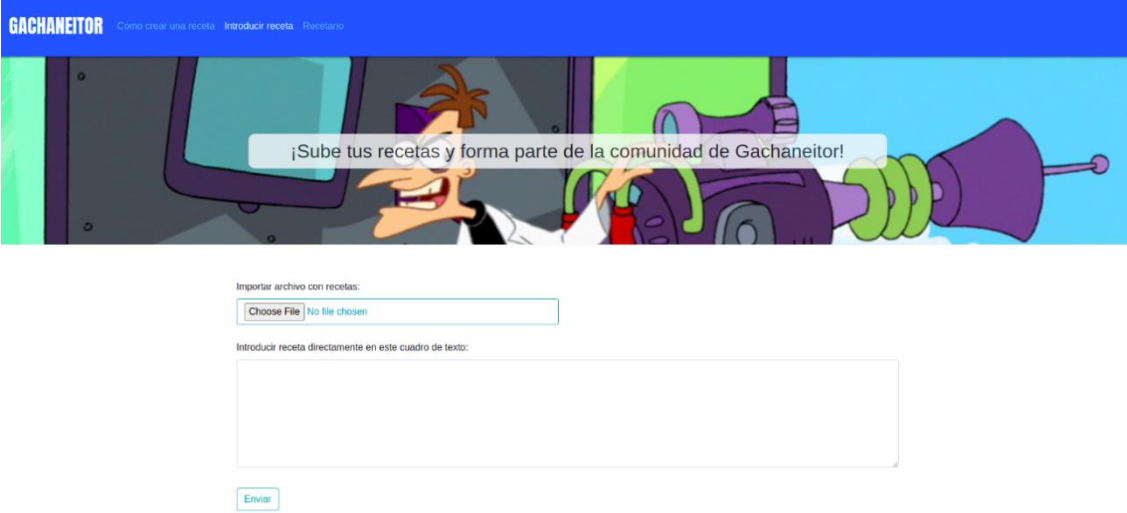
- La palabra **receta** junto a dos llaves (una abierta y una cerrada) que indica el comienzo de una receta (ya que pueden existir varias en la misma cadena).
- Dentro de estas llaves, están situadas las partes que conforman una receta en el siguiente orden fijo:
  - El **nombre** de la receta, que se define mediante la palabra **nombre** (opcional) y a continuación el nombre de la misma entre comillas (e.g. nombre "Puré").
  - La **descripción** de esta receta, que se define mediante la palabra **descripción** (opcional) seguida de una descripción entre corchetes.
  - El **tiempo** de la receta, que se define mediante la palabra

- **Introducir receta**

Esta parte permite añadir recetas al repositorio de recetas de la comunidad de dos formas diferentes:

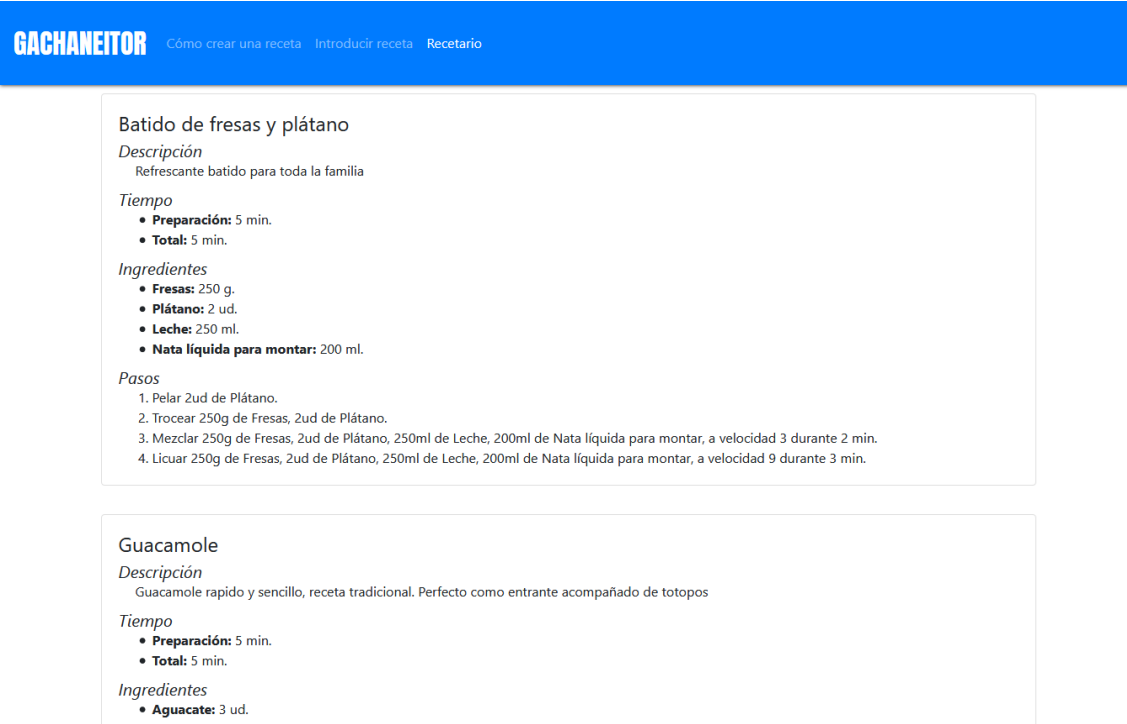
- En primera instancia, se pueden escribir varias recetas en un mismo archivo de texto y subir el archivo directamente.
- Adicionalmente, se puede escribir el archivo directamente en la página web, permitiendo al usuario otra forma de interactuar con ella.

Si se introduce de las dos formas, tendrá más prioridad aquellas recetas que se introduzcan a través de un archivo.



- **Recetario**

Esta parte permite visualizar las recetas desarrolladas por los miembros de la comunidad ordenadas por orden alfabético, según el nombre de estas. Cualquier receta aceptada por el procesador de lenguajes aparecerá en esta pestaña posteriormente.



**Batido de fresas y plátano**  
*Descripción*  
 Refrescante batido para toda la familia  
*Tiempo*  
 • **Preparación:** 5 min.  
 • **Total:** 5 min.  
*Ingredientes*  
 • **Fresas:** 250 g.  
 • **Plátano:** 2 ud.  
 • **Leche:** 250 ml.  
 • **Nata líquida para montar:** 200 ml.  
*Pasos*  
 1. Pelar 2ud de Plátano.  
 2. Trocear 250g de Fresas, 2ud de Plátano.  
 3. Mezclar 250g de Fresas, 2ud de Plátano, 250ml de Leche, 200ml de Nata líquida para montar, a velocidad 3 durante 2 min.  
 4. Licuar 250g de Fresas, 2ud de Plátano, 250ml de Leche, 200ml de Nata líquida para montar, a velocidad 9 durante 3 min.

**Guacamole**  
*Descripción*  
 Guacamole rapido y sencillo, receta tradicional. Perfecto como entrante acompañado de totopos  
*Tiempo*  
 • **Preparación:** 5 min.  
 • **Total:** 5 min.  
*Ingredientes*  
 • **Aguacate:** 3 ud.  
 • **Zumo de limón:** 10 g.

## 11. Ejecución de procesador hecho con JFlex+CUP

### Requisitos

Para poder ejecutar el procesador de lenguajes, es necesario tener instalado tanto Java como los archivos .jar de JFlex y CUP en el sistema.

### Ejecución

El primer paso es generar el *analex.java* con el autómata que reconoce el léxico de nuestro lenguaje. Abra una terminal en la carpeta actual y escriba:

```
jflex AnalizadorLexico.flex --encoding utf-8
```

Después de generar el analizador léxico, procedemos a hacer lo mismo con el analizador sintáctico de CUP:

```
java java_cup.Main AnalizadorSintactico.cup
```

Aquí se creará el *sym.java* y el *parser.java*. Ya tenemos todas las clases .java necesarias, por lo que procedemos a compilarlas:

```
javac -encoding utf8 *.java
```

Después, para hacer uso del procesador, escriba en la terminal:

```
java -cp <ruta_del_archivo_java-cup-11b-runtime.jar> Main  
      <ruta_del_archivo_con_recetas>
```

O directamente si tenemos el .jar en el classpath:

```
java Main <ruta_del_archivo_con_recetas>
```

La salida del procesador se guarda en el archivo *salida.json*.

### Tests

Para ejecutar los tests sintácticos y semánticos, hay que bajar los 3 jars y añadirlos a la variable de entorno CLASSPATH del sistema. Después de eso, situándonos en esta carpeta, *src/pl\_cup*, con la versión del 1.8 del compilador e intérprete ejecutamos los siguientes comandos:

```
javac -encoding utf8 *.java
```

```
java junit.textui.TestRunner TestSemantica
```

```
java junit.textui.TestRunner TestSintactica
```

## 12. Ejecución de procesador hecho con ANTLR

### Requisitos



Para ejecutar la herramienta, es necesario tener el intérprete de Python instalado en el sistema. Además, se adjunta un archivo requirements.txt indicando el ejecutable de antlr4 y las librerías necesarias para instalar en un entorno virtual.

```
pip3 install -r requirements.txt
```

### **Ejecución**

Tras haber instalado los requisitos y tener el entorno virtual preparado, situándonos en la carpeta actual procedemos a ejecutar:

```
antlr4 -Dlanguage=Python3 -encoding utf-8 Gachaneitor.g4
```

Esto genera los distintos archivos del procesador de lenguajes(Lexer, Parser, Listener,... ). Después, para hacer uso del procesador, escriba en la terminal:

```
python3 main.py -i <ruta_del_archivo_txt>
```

El JSON que se genera como salida del procesador aparecerá en la terminal.

### **Tests**

Se ha añadido un script con tests unitarios que ejecuta cada uno de los casos de uso. Para ejecutar los tests automáticamente, en la carpeta actual escriba en una terminal:

```
python3 -m unittest discover tests
```

## **13. Ejecución de la web**

### **Prerrequisitos**

Antes de ejecutar el servidor por el cual se desplegará esta página web, se deben instalar una serie de requisitos.

El primer requisito es instalar la versión de Python 3.8. Para instalar las dependencias es recomendable crear un entorno virtual con el módulo de entornos virtuales venv.

Este módulo permite aislar las dependencias entre distintos proyectos, creando diferentes intérpretes para cada uno con sus respectivas dependencias. El siguiente comando crea un intérprete en el directorio elegido.

```
python3 -m venv venv
```

Si se escoge crear un entorno virtual, la forma de utilizar el intérprete es usando el siguiente comando:

```
source venv/bin/activate
```

Para volver a usar el intérprete por defecto del sistema se tiene que ejecutar el siguiente comando:

```
deactivate
```

Tanto si se hace la instalación de las dependencias en un entorno virtual, como si se hace en el intérprete por defecto del sistema, los requisitos del proyecto se instalan utilizando el siguiente comando.

```
pip3 install -r requirements.txt
```

### **Despliegue**

Para ejecutar el servidor y para que la web pueda ser consultada, debemos iniciar el servidor de Django. Para ello se debe entrar en la carpeta *gachaneitor* y ejecutar el siguiente comando:

```
python3 manage.py runserver
```

Una vez que se ha iniciado el server, para entrar a la web debemos de escribir la siguiente dirección en el navegador: <http://127.0.0.1:8000>

## **14. Puntuación de los miembros del equipo**

Miembro	Puntuación
<b>Raúl Bernalte Sánchez</b>	5
<b>Enrique Cepeda Villamayor</b>	5
<b>Elena Hervás Martín</b>	5
<b>Elena María Ruiz Izquierdo</b>	5