Proyecto de Grado

Presentado ante la ilustre Universidad de Los Andes como requisito parcial para obtener el Título de Ingeniero de Sistemas

Diseño e implementación de un framework para el desarrollo de prácticas de laboratorio en el área de compiladores

Por

Br. Luis Fernando Angulo Pérez

Tutor: Prof. Rodolfo Sumoza

Tutor: Prof. Diego Mosquera

Octubre 2017

Diseño e implementación de un framework para el desarrollo de prácticas de laboratorio en el área de compiladores

Br. Luis Fernando Angulo Pérez

Proyecto de Grado — Sistemas Computacionales, 105 páginas

Resumen: El desarrollo de las prácticas en asignaturas como Compiladores tiene como propósito la consolidación de los conocimientos en el área. Un reto constante suele ser la planificación de las clases prácticas del curso y las herramientas a utilizar como acompañamiento, de manera que se puedan ejecutar eficaz y eficientemente. En el presente proyecto se propone diseñar las prácticas para el laboratorio e implementar un framework que permita de forma remota a los estudiantes desarrollarlas y al docente evaluarlas.

Palabras clave: compiladores, prácticas de laboratorio, framework

Índice

Índice de Figuras **vii**

Agradecimientos **viii**

1 **1**

1.1 Introducción 1
1.2 Antecedentes 2
1.3 Planteamiento del Problema 4
1.4 Objetivos 4
1.4.1 Objetivo General 4
1.4.2 Objetivos Específicos 5
1.5 Alcance 5
1.6 Marco Teórico 5
1.6.1 Historia De Los Compiladores 5
1.6.2 Compiladores 6
1.6.3 Estructura de un compilador 6
1.6.4 Compiladores Cómo Asignatura 8
1.6.5 Estructura de la asignatura 8
1.6.6 Práctica De Laboratorio 9
1.6.7 Prácticas de laboratorio en el contexto de la ULA 10
1.6.8 ¿Qué es una práctica de compiladores 10
1.6.9 Framework 11

2 Diseño De Las Prácticas **12**
2.1 Propuesta Para Las Prácticas 12
2.1.1 Contexto Educativo de la Facultad de Ingeniería en la Universidad de Los Andes 12
2.2 Plantilla De Prácticas 13
2.2.1 Plantillas Para Las Prácticas de Compiladores 16

3 Diseño del Framework **35**
3.1 Componentes del Framework 36
3.1.1 Caja de Herramientas 37
3.1.2 Molde de Resolución de Asignaciones o Base 38
3.1.3 Respuesta de Usuario 38
3.1.4 Herramienta 38
3.1.5 Salida 39
3.1.6 Base de Datos 39
3.2 Tabla de Operaciones 43
3.2.1 Interacción con la Tabla de Operaciones 43

4 Herramientas para la Implementación del Framework **46**
4.1 Framework Web 46
4.1.1 API Rest 47
4.1.2 Frameworks para desarrollo de aplicaciones 48
4.1.3 Base de Datos 49
4.2 Herramientas Para el Framework 49

5 Implementación Del Framework **51**
5.1 Desarrollo Guiado por Pruebas 51
5.2 Estructura de directorios del Framework 52
5.3 Creación de la API para el Framework 54
5.3.1 Modelos 54
5.3.2 Vistas 54
5.3.3 Serializadores 55

5.3.4 Direcciones URLs 55
5.3.5 Pruebas unitarias 55
5.4 Integración de Herramientas 56
5.4.1 Clase abstracta Herramienta 57
5.4.2 Convención de nombres e instalación de herramientas al Framework 57
5.4.3 Compilación y Ejecución 58
5.4.4 Uso de múltiples Herramientas 58
5.5 Instalación y Configuración 59
5.5.1 Instalación de la Aplicación 59
5.5.2 Configuración del Framework 59
5.6 Pruebas Realizadas 60
5.6.1 Pruebas unitarias 60
5.6.2 Pruebas manuales 61

6 Conclusiones y Recomendaciones **102**
6.1 Conclusiones 102
6.2 Recomendaciones 103

Bibliografía **104**

Índice de Figuras

2.1 Fases de compilación 14

3.1 Modelo Lógico 36
3.2 Modelo de Datos 40
3.3 Creación de una Práctica 45

5.1 Estructura de directorios del framework 53
5.2 Estructura del directorio de herramientas del framework 56

Capítulo 1

1.1 Introducción

La preparación del contenido para impartir un curso es una tarea laboriosa, donde usualmente una forma ampliamente adoptada es la de impartir un contenido teórico y demostrativo sobre una temática, acompañándolo con clases prácticas que ayuden a reforzar y afianzar la teoría.

Las clases prácticas son un aspecto clave para la formación de estudiantes. Involucran el uso de herramientas y metodologías con el fin de demostrar, en parte, los fundamentos teóricos ante una problemática.

Sin embargo, comúnmente surgen problemas con el diseño e implementación de las clases prácticas. Algunos de estos problemas o dificultades aparecen cuando se selecciona el contenido para estas prácticas ya que puede no complementar apropiadamente lo visto en la teoría, o cuando se utilizan las herramientas seleccionadas ya que es posible que no brinden vínculos claros con los fundamentos teóricos. También el cambio entre docentes en la misma asignatura puede implicar que se deban diseñar nuevamente las prácticas generando variaciones e inconsistencias entre distintas instancias o repeticiones de un curso.

El curso de Compiladores de la Universidad de Los Andes, es una asignatura del ciclo profesional de la carrera de Ingeniería de Sistemas. Los compiladores, vistos como disciplina, toman significativa importancia para la consolidación de los conocimientos en el área de los sistemas computacionales. Como asignatura, tiene como propósito explicar cómo lenguajes de alto nivel pueden ser traducidos sistemáticamente en

lenguajes de bajo nivel. En la práctica el estudiante requiere aprender a utilizar diferentes herramientas, para poder desarrollar tareas vinculadas a la compilación y ejecución de código con la finalidad de consolidar sus conocimientos adquiridos en la parte teórica. La asignatura de Compiladores no contaba con una planificación lo suficientemente estructurada en la parte práctica, para responder formalmente a lo que se busca en este aspecto, lo que dio cabida a la consecución de este proyecto de grado.

La intención de este trabajo es la de ofrecer una planificación detallada para el contenido práctico del curso de Compiladores y facilitar el uso de distintas herramientas tecnológicas para desarrollarlas, a través de la implementación de un *framework*, que englobe en un solo entorno la ejecución de cualquiera de ellas. El *framework* tiene la finalidad de facilitar la creación de las prácticas del curso a un docente y permitir a los estudiantes resolverlas de una forma simple.

En el capítulo dos (2), se muestra el diseño propuesto para la estructura del contenido del curso de Compiladores, en el capítulo tres (3) se muestra el diseño de una plataforma o *framework* a nivel teórico que facilita la creación y ejecución de prácticas del curso, en el capítulo cuatro (4) se habla de las herramientas a utilizar para implementar el *framework*, en el capítulo cinco (5) se menciona el proceso de implementación del framework junto con las pruebas realizadas y en el capitulo seis (6) se muestran las conclusiones sobre el proyecto realizado y se ofrecen recomendaciones para trabajos futuros. En los apéndices se describen los diseños para el contenido de las prácticas y se presentan guías sobre el funcionamiento del *framework*.

1.2 Antecedentes

La enseñanza en esta área ha sido foco de debate puesto que se plantean muchos puntos de vista sobre cómo impartir el contenido del curso y qué tipo de práctica es necesaria como acompañamiento. En la parte práctica existen varias alternativas como se exponen en [15]. En esta investigación se describe la dificultad en la "planificación y continuidad en el trabajo de los estudiantes" y se describe cómo el uso de ciertas herramientas puede "no tener una conexión clara con los fundamentos teóricos". Las prácticas tienen como propósito complementar el contenido del curso

de Compiladores, siendo el contenido en su mayor parte y secuencialmente: el análisis léxico, análisis sintáctico, análisis semántico y generación de código. Este trabajo ofrece una alternativa sobre el contenido y herramientas asociados a las prácticas del curso.

De forma similar el trabajo expuesto en [16] ofrece una crítica al método tradicional de enseñanza dado su contexto educativo, resaltando el inconveniente ya mencionado recurrente entre las evaluaciones del curso, y la relación entre la teoría y la práctica. Ofreciendo distintas soluciones para la selección y estructuración del contenido, tomando en cuenta cómo hacerlo efectivo para el estudiante y su aprendizaje. Las propuestas incluyen el uso de recursos tecnológicos que estimulen el enfoque práctico usado por los estudiantes usando un material que incluya ejemplos visuales y recursos en "operación" en lugar de elementos únicamente textuales.

Otros estudios ofrecen alternativas directas al inconveniente proponiendo el desarrollo de actividades prácticas, estructurando el contenido y las herramientas acorde a lo que consideran aumenta la calidad del curso. Para la asignatura Compiladores se tiene el curso dictado y diseñado por [5] y el curso propuesto en [12]. En [5] se propone como alternativa para el contenido práctico el uso de un lenguaje de programación específicamente diseñado para la enseñanza de Compiladores llamado "COOL" (Classroom Object Oriented Language). En este curso se cubren tópicos como análisis léxico, sintáctico, semántico, generación de código y optimización de código. Las actividades prácticas se dividen en varias entregas, utilizando herramientas específicas, cada una con un lapso de entrega diferente y un proyecto, que consiste en desarrollar un traductor basado en el lenguaje COOL y una máquina virtual donde se ejecutará el código resultante del proceso de compilación. En [12] se propone, por otro lado, el estudio directo de un compilador para un lenguaje llamado "Tinto", partiendo desde las operaciones que pueden ser realizadas por este lenguaje, hasta cómo el compilador interpreta las estructuras de datos.

En algunos estudios previos como [8] han determinado que los cursos de Compiladores son muy comunes en la programación de las carreras relacionadas a la ciencia de la computación, donde los estudiantes típicamente estudian temas acerca del diseño de compiladores, sin embargo muchas de las técnicas impartidas en estos

cursos suelen ser poco usados fuera del aula de clase ya que es poco probable que algún alumno desarrolle un compilador fuera de la asignatura a pesar de que muchas de estas técnicas vistas como las expuestas en [10] tienen mucha más aplicación fuera del área de compiladores, recordando que el propósito fundamental de la asignatura en la mayoría de los casos es afianzar los conocimientos sobre la teoría de la computación. Es por este motivo que en muchos casos es sugerido que se haga énfasis en ejemplos prácticos, de modo que el material visto por cada alumno refleje problemas en el ámbito de la computación fuera del área de compiladores y que puedan ser resueltos usando las técnicas que se imparten en la materia.

1.3 Planteamiento del Problema

Actualmente, en la asignatura de Compiladores, de la Facultad de Ingeniería (ULA), se requiere de prácticas de laboratorio que eficazmente complemente la teoría, y de recursos tecnológicos apropiados para su ejecución: creación, depuración, implementación y evaluación, en cada una de las fases del proceso de compilación, con mejores prestaciones que el sistema utilizado actualmente. Lo cual tiene como fin mejorar el proceso de aprendizaje.

Este proyecto de grado busca diseñar y estructurar el contenido de las prácticas de laboratorio de la materia Compiladores. Además propone localizar e integrar distintas herramientas que permitan un acoplamiento tanto con el material propuesto para las clases prácticas y teóricas en un framework, el cuál permitirá desarrollarlas por parte de los estudiantes de forma remota, y también se procura homogeneizar su contenido.

1.4 Objetivos

1.4.1 Objetivo General

Construir un framework para el desarrollo de prácticas de laboratorio en el área de compiladores.

1.4.2 Objetivos Específicos

1. Seleccionar y estructurar el contenido de las prácticas de laboratorio de la materia compiladores.
2. Localizar las distintas herramientas de *software* para compiladores.
3. Integrar las distintas herramientas de *software* para compiladores en un *framework*.
4. Implementar las prácticas estructuradas en el *framework*.
5. Probar el *framework*.

1.5 Alcance

Este proyecto tiene como fin la elaboración de un *framework* para usuarios que cursen la asignatura de Compiladores. Con este framework cada usuario puede interactuar de forma remota con las distintas herramientas ofrecidas para cada módulo o tema de la asignatura para poder seguir el contenido práctico.

1.6 Marco Teórico

1.6.1 Historia De Los Compiladores

Las primeras computadoras fueron creadas en la década de 1940 y se programaba en lenguaje maquina, usando únicamente secuencias de ceros y unos. Indicaban de forma explícita las operaciones que el computador debía ejecutar. Estas operaciones eran de muy bajo nivel, tal como mover datos de una dirección a otra o sumar el contenido de dos registros. Era una programación lenta, tediosa y propensa a errores al momento de escribir el programa, lo que hacia que una vez escritos fuesen muy difíciles de comprender y modificar. Como solución para facilitar la programación se invento el lenguaje ensamblador. Eran representaciones mnemóicas de las instrucciones del computador.

Luego, en la década de 1950 se construyeron los lenguajes Fortran para la computación científica, Cobol para el procesamiento de datos de negocios y Lisp para la computación simbólica. Estos lenguajes seguían una filosofía que se basaba en crear notaciones de alto nivel con lo que los programadores pudiesen escribir con mas facilidad cálculos numéricos, aplicaciones de negocios y programas simbólicos. Los escritores de estos lenguajes tuvieron que idear algoritmos de traducción que aprovechasen al máximo las características del hardware y así poder ejecutar el código escrito usando alguno de estos lenguajes de alto nivel, a éste algoritmo de traducción se le llamó compilador. Este fue el inicio del Compilador [4].

1.6.2 Compiladores

Un compilador es un programa que analiza un código escrito en un lenguaje fuente y lo traduce a uno equivalente escrito en lenguaje máquina, para tal proceso se verifica la correctitud del código original. Reporta cualquier error en el programa fuente durante el proceso de traducción [4].

1.6.3 Estructura de un compilador

Un compilador tiene dos procesos principales análisis y síntesis. Comúnmente al análisis se le llama *front-end* del compilador y a la síntesis *back-end* [4].

Análisis

Divide el programa fuente en componentes e impone una estructura gramatical sobre ellas. Después utiliza esta estructura para crear una representación intermedia del programa fuente. Si el análisis detecta algún error, entonces proporciona mensajes informativos para que el usuario pueda corregirlos. El análisis recolecta información sobre el programa fuente y la almacena en una estructura de datos llamada tabla de símbolos, la cual junto con la representación intermedia obtenida previamente, pasan a la parte de la síntesis para ser procesadas[4].

Análisis De Léxico

Es la primera fase de un compilador y pertenece al análisis. Se le conoce como análisis léxico o escaneo. Consiste en leer un flujo de caracteres que componen el programa fuente y agruparlos en secuencias significativas, conocidas como lexemas. Para cada lexema que el analizador léxico produce como salida, se crea un token, que es una representación de la forma de un par nombre-valor, la cual se pasa al análisis sintáctico.

Análisis Sintáctico

Análisis Sintáctico o parsing es la fase de compilación que utiliza los componentes de los tokens obtenidos en el análisis léxico para crear una representación intermedia en forma de árbol que describe una estructura gramatical del flujo de tokens. Una representación típica es el árbol sintáctico, en este cada nodo representa una operación y cada hijo un argumento de la operación.

Análisis Semántico

Utiliza un árbol sintáctico y la información de la tabla de símbolos con la finalidad de comprobar que la consistencia semántica de un programa fuente coincida con la definición del lenguaje. También recopila información sobre el tipo de operaciones que se realizan dentro del programa y las almacena para ser usadas durante la generación de código intermedio. Una parte importante de este análisis es la comprobación o verificación de tipos, en donde el compilador verifica que las operaciones tengan operandos que coincidan. A su vez también debe permitir ciertas conversiones de tipo, conocidas como coerciones, las cuales permiten y/o obligan al compilador a convertir el resultado de una operación a un tipo apropiado definido por el lenguaje.

Síntesis

Construye el programa destino a partir de una representación intermedia obtenida en el análisis y la información almacenada en la tabla de símbolos[4].

Generación De Código Intermedio

Es el proceso de construir una representación de un código destino independiente de la arquitectura del computador. Un compilador puede construir diversas representaciones intermedias, con una gran variedad de formas entre ellas. Algunos compiladores generan luego del análisis sintáctico y semántico, un nivel bajo explicito o una representación abstracta similar al lenguaje maquina. Esta representación puede ser considerada como un programa para una maquina abstracta.

Optimización de código

Es una fase independiente a la generación de código de maquina y consiste en mejorar el código intermedio, generado previamente.

Generación De Código

Es una fase del *back-end* o síntesis, recibe como entrada una representación intermedia y la tabla de símbolos. Si el lenguaje destino es código maquina, se seleccionan registros y ubicaciones de memoria para cada variable que utiliza el programa.

1.6.4 Compiladores Cómo Asignatura

Como nace la asignatura de Compiladores

El curso de Compiladores es un requisito para la formación de un ingeniero informático al tener ademas del objetivo de enseñar al estudiante como construir un Compilador, completar la formación que lleva en la tarea de programar y los lenguajes de programación utilizados como herramienta [9].

1.6.5 Estructura de la asignatura

La estructura utilizada es la que se refleja en los programas oficiales de la asignatura. Esta se presta para ser dictada de múltiples formas. Generalmente (la mayoría de los casos) la asignatura es divida en dos partes, una teórica y otra práctica. En donde la práctica complementa la teoría.

Teoría

Para cursar la asignatura Compiladores, los estudiantes deben tener conocimiento de los conceptos básicos de Teoría de la Computación.

Comienza con una explicación de su estructura general y luego aborda la enseñanza de la fase de Análisis Léxico, en donde se utilizan herramientas como las expresiones regulares y los autómatas finitos para su implementación. A continuación se define la fase de Análisis Sintáctico utilizando principalmente gramáticas libre de contexto para generar árboles de derivación. Finalmente, se explica la fase de Análisis Semántico y Generación de Código, donde se define la traducción orientada a la sintaxis y las representaciones de código intermedio.

Práctica

La parte práctica se usa para complementar la teoría al aplicar técnicas para generar analizadores léxicos y parsers.

Para la creación de Analizadores Léxicos, se emplean métodos de reconocimiento de patrones por medio de expresiones regulares, las cual han sido implementadas utilizando autómatas finitos.

Para la construcción de Parsers, se usan gramáticas libres de contexto con la finalidad de verificar la correctitud desde el contexto de la sintaxis del lenguaje fuente.

Se realizan prácticas evaluadas y no evaluadas con el fin de enriquecer el conocimiento adquirido en la teoría.

1.6.6 Práctica De Laboratorio

Según [13] es una actividad cuyo objetivo principal es garantizar un aprendizaje participativo, individual y activo de todos los estudiantes en una carrera de la ciencia, es importante destacar que los beneficios para el alumnado son numerosos dentro los cuales se destacan: aprendizaje de técnicas experimentales, desarrollo de habilidades dentro del laboratorio y la consolidación de la comprensión del enfoque del tema que se está estudiando.

¿Qué se evalúa en una práctica de laboratorio?

Se evalúan las distintas técnicas y destrezas enseñadas para resolver un problema propuesto desde un contexto práctico.

¿Cómo se evalúa una práctica de laboratorio?

Cada práctica tiene una manera distinta de ser evaluada. Algunas formas a utilizar son las evaluaciones escritas, exposiciones orales, asignaciones con fechas propuestas o demostraciones de habilidades prácticas, dependiendo del contenido y aplicación, verificando que el estudiante conozca la parte teórica de la asignatura.

1.6.7 Prácticas de laboratorio en el contexto de la ULA

Es una estrategia didáctica, dinámica y multidisciplinaria, cuyos objetivos giran a favor de aprendizajes profesionales complejos en contextos de investigación y experimentación; orientada a fortalecer competencias profesionales específicas, desarrollar habilidades y compromisos con el quehacer científico y la innovación [3].

Las prácticas son realizadas por el personal docente o un *preparador* a cargo del curso. Un *preparador* es una figura académica en la que un estudiante luego de haber visto una asignatura en particular y haber obtenido una calificación alta en la misma, entra bajo concurso y contrato con el propósito de ayudar en la realización de la asignatura.

1.6.8 ¿Qué es una práctica de compiladores

Una práctica de Compiladores es una clase en la que se realizan demostraciones prácticas de un segmento del curso y en el que se realizan ciertas actividades que incluyen el uso de herramientas, con el propósito de enseñar ciertas habilidades y destrezas para un tema en especifico. Las actividades involucran codificar y ejecutar distintos programas así como ejercicios escritos.

Por lo general las prácticas se estructuran según las fases del compilador. Cada práctica se basa en un tema diferente estudiado en la teoría, y contiene actividades referentes al mismo. En primer lugar se explica como aplicar de manera práctica

conceptos referentes a análisis léxico. Luego, siguiendo el mismo esquema se explican conceptos relacionados a análisis sintáctico, análisis semántico y generación de código. El instructor de laboratorio realiza una serie de actividades y propone a los estudiantes ejercicios prácticos. Al final de cada etapa, se realiza una evaluación donde se evalúa la correcta aplicación de los temas estudiados. Su calificación forma parte de la nota final de la asignatura.

1.6.9 Framework

Un *framework* es un concepto muy abstracto que puede adecuarse a diversas áreas. En las Ciencias de la Computación, este concepto se emplea en muchos ámbitos del desarrollo de sistemas software; por la libertad que el mismo presenta, se puede definir como un marco de trabajo caracterizado por proveer prácticas y criterios relacionados a un tema en particular, con la finalidad de facilitar la resolución de problemas de la misma índole.

Un *framework* facilita resolver un problema a través de ciertas reglas. Lo que implica que quien lo utiliza tendrá una mayor noción de orden a la hora de realizar una actividad. También por medio de la personalización e intercambio de componentes, se convierte en una "caja" genérica y configurable que se puede adaptar a cualquier situación.

Capítulo 2

Diseño De Las Prácticas

2.1 Propuesta Para Las Prácticas

La preparación de un contenido práctico para complementar la teoría vista en clases de Compiladores, no es una tarea fácil. Existen múltiples enfoques y el uso de herramientas para experimentar y visualizar actividades, suelen no tener una conexión clara con los fundamentos que se explican en la teoría [15]. Por esta razón se propone seguir una estructura de prácticas que puede adaptarse a las necesidades del curso a lo largo de los años.

Esta estructura, denominada plantilla permite al docente seguir el curso de una manera ordenada evitando variaciones del contenido entre distintas iteraciones del mismo. A la vez, le concede más tiempo para explicar el tema, ya que la preparación de la clase estaría cubierta y así se pudiese garantizar el cumplimiento de los objetivos del curso.

2.1.1 Contexto Educativo de la Facultad de Ingeniería en la Universidad de Los Andes

En la Facultad de Ingeniería perteneciente a la Universidad de Los Andes, la asignatura de Compiladores se imparte a lo largo de un semestre cuya duración es de 16 semanas. En este lapso de tiempo se debe explicar y evaluar el contenido práctico de la materia.

2.2 Plantilla De Prácticas

Una plantilla es un documento con una estructura determinada, con un listado de actividades propuestas y con datos que deben ser completados para armar las prácticas. Las actividades se pueden utilizar con fines didácticos o para evaluación. Por sí sola una plantilla no conforma una práctica única, el docente puede seleccionar actividades de varias plantillas para armar una práctica requerida según el contenido teórico establecido, que puede variar según las características de período lectivo.

Cada plantilla tiene datos que definen su nombre, objetivo, descripción, competencias previas y esperadas del estudiante, una breve especificación de la práctica y el nivel de dificultad. Esto facilita al educador el diseño y elaboración de una práctica de acuerdo a un contenido específico. Cada plantilla cuenta con un conjunto de actividades prácticas las cuales de forma simple determinan qué tareas se deben realizar.

En el caso del curso de Compiladores se reestructuró el contenido de la asignatura de acuerdo al siguiente esquema (Figura FiguraFaseCompilacion), donde para cada fase del proceso se realiza un determinado número de actividades pertenecientes a una o varias plantillas.

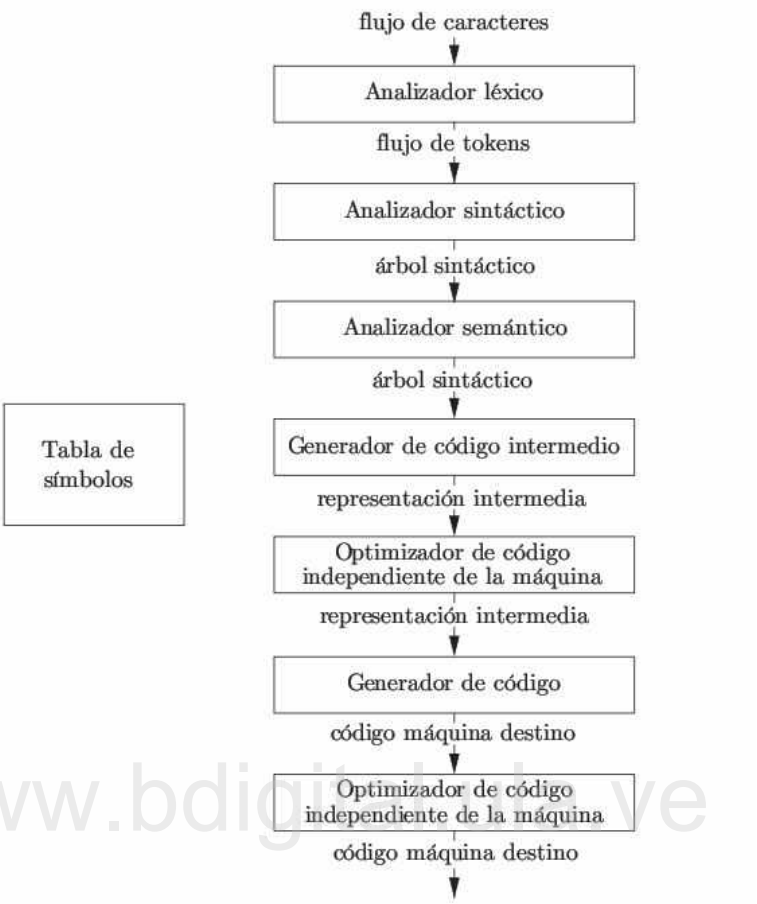

Figura 2.1: Fases de compilación

Se diseñaron plantillas que complementan el contenido teórico referente al Análisis Léxico, Análisis Sintáctico, Análisis Semántico y Generación de Código, estructuradas de la siguiente manera:

- Análisis Léxico
  - Diseño y construcción de Expresiones Regulares.
  - Construcción de autómatas finitos.
  - Construcción de Analizadores Léxicos.
  - Uso de la tabla de símbolos en un Analizador Léxico.
  - Gestión de errores léxicos.

- Análisis Sintáctico
  - Diseño y Especificación de Gramáticas Libres de Contexto (GLC)
  - Representación de Árboles de derivación.
  - Técnicas para realizar Análisis Sintáctico Descendente sobre una GLC.
  - Técnicas para realizar Análisis Sintáctico Ascendente sobre una GLC.
  - Uso de la tabla de símbolos en el Análisis Sintáctico.
  - Construcción de Parsers.
  - Técnicas para recuperación de errores sintácticos.
- Análisis Semántico
  - Verificación de Ámbito de Variables.
  - Comprobación de Tipos.
  - Traducción orientada a la sintaxis.

- Generación de Código
  - Generación de Código de tres direcciones.
  - Generación de Flujo de Control.

A continuación se muestra la estructurada de una plantilla:

- **Nivel:** Es un nivel asociativo de la práctica, representa el nivel de dificultad y ayuda a identificar el orden en que debería realizarse una práctica con el contenido de la plantilla, a menor nivel menor dificultad.
  Nombre: Identificador único de la plantilla, debe exponer la temática del contenido.
  Objetivo: Define el propósito principal de la plantilla y de las actividades contenidas.
  Descripción: Texto descriptivo de la plantilla, ofrece una breve reseña del contenido de la plantilla incluyendo información sobre el desarrollo de las actividades.

Competencias previas: Lista de facultades requeridas por un estudiante para poder llevar a cabo las actividades dentro de la plantilla.
Competencias esperadas: Lista de facultades a ser obtenidas por un estudiante al realizar las actividades dentro de la plantilla.
Actividades prácticas: Listado que incluye todas las actividades de la plantilla, las mismas deben ir numeradas y no deben estar repetidas en otra plantilla del curso. Cada actividad descrita debe ser breve y bien definida indicando que acción realizar.

2.2.1 Plantillas Para Las Prácticas de Compiladores

A continuación se tienen las plantillas escritas para el curso de Compiladores.

- **Nivel:** 0
  Nombre: Introducción a las Prácticas de Laboratorio.
  Objetivo: Introducir al estudiante a las prácticas de Compiladores.
  Descripción: Práctica introductoria a la asignatura donde se explican distintas reglas y las características que las prácticas tendrán. Se realizan demostraciones que visualicen el trayecto que el estudiante tendrá. Se mostraran las diferencias mas notables entre un compilador y un interprete.
  Competencias previas: Teoría de la computación: Lenguajes regulares, gramáticas libres de contexto (GLC). Técnicas de programación imperativa o funcional.
  Competencias esperadas: Conoce las diferencias clave entre un compilador y un interprete. Tener una visión breve sobre las capacidades a ser adquiridas en las prácticas.
  Actividades Prácticas:

  1. Realizar una demostración utilizando un interprete o compilador modelo, con el fin de reflejar las capacidades prácticas que el estudiante debe poseer al finalizar el curso.
  2. Realizar una demostración traduciendo de un lenguaje fuente a un lenguaje destino, siendo ambos de alto nivel.

3. Realizar una demostración sobre la compilación y ejecución de un código escrito en un lenguaje compilado.
4. Realizar una demostración sobre la ejecución de un código escrito en un lenguaje interpretado.

Fase 1: Análisis léxico

- **Nivel:** 1
  Nombre: Construcción de expresiones regulares.
  Objetivo: Construir expresiones regulares.
  Descripción: El estudiante debe construir expresiones regulares que definen un lenguaje regular.
  Competencias previas: Expresiones regulares y lenguajes regulares.
  Competencias Esperadas: El estudiante debe estar capacitado para construir expresiones regulares utilizando práctica notaciones y convenciones usadas en el curso.

  Actividades Prácticas:

  1. Indicar las convenciones de escritura para escribir expresiones regulares:
     - Alternación.
     - Cuantificación.
     - Agrupación.
  2. Indicar los descriptores a ser usados al escribir expresiones regulares:
     - El punto.
     - El signo de admiración.
     - La barra inversa.
     - Corchetes.
     - La barra.
     - El signo dólar.
     - El acento circunflejo.

- Los paréntesis.
- El signo de interrogación.
- Las llaves.
- El asterisco.
- El signo suma.

3. Construir una expresión regular a partir de una especificación propuesta.

- **Nivel:** 2

  Nombre: Construcción de autómatas finitos no deterministas.

  Objetivo: Construir autómatas finitos no deterministas a partir de expresiones regulares.

  Descripción: El estudiante debe aplicar los algoritmos vistos en la teoría para elaborar autómatas finitos no deterministas a partir de una expresión regular.

  Competencias previas: Expresiones regulares, autómatas finitos no deterministas.

  Competencias esperadas: El estudiante debe ser capaz de crear AFND partiendo de una expresión regular, utilizando algún algoritmo visto en la teoría. Se deben tener conocimientos sobre algoritmos usados en la ciencias de los compiladores para convertir una expresión regular en un autómata finito no determinista, ejemplo: "algoritmo de thompson".

  Actividades Prácticas:

  1. A partir de una expresión regular aplicar el algoritmo de thompson para obtener el AFND resultante.
  2. A partir de una palabra usada como entrada probar la aceptación usando el modelo.

- **Nivel:** 3

  Nombre: Construcción de autómatas finitos deterministas a partir de autómatas finitos no deterministas.

  Objetivo: Construir AFD a partir de AFND.

  Descripción: El estudiante debe aplicar los algoritmos vistos en la teoría

para elaborar autómatas finitos deterministas a partir de autómatas finitos no deterministas.

Competencias previas: Autómatas finitos no deterministas.

Competencias esperadas: Capacidad para construir AFD partiendo de un AFND.

Actividades Prácticas:

1. Aplicar las operaciones de clausura para un AFND.
2. Aplicar las operaciones de move para un AFND.
3. Aplicar el algoritmo de conversión de un AFND a un AFD.

- **Nivel:** 4

  Nombre: Implementación de un autómata finito.

  Objetivo: Implementar un autómata finito determinista.

  Descripción: El estudiante deberá implementar un AFD. A partir de una expresión regular el estudiante debe implementar por medio de un lenguaje de programación el autómata finito resultante de la conversión de la expresión regular. Se debe aplicar opcionalmente el uso de tablas de transición.

  Competencias previas: Técnicas en programación imperativa y funcional, autómatas finitos.

  Competencias esperadas: El estudiante debe poder implementar un AF usando algún lenguaje de programación.

  Actividades Prácticas:

  1. Transformar una expresión regular propuesta en un AFND.
  2. Transformar el AFND resultante en un AFD.
  3. Usar un lenguaje de programación para representar el AFD como una maquina de estados.
  4. Usar una entrada para comprobar la aceptación de la representación del AFD.

- **Nivel:** 5

  Nombre: Construcción de un analizador léxico.

Objetivo: Construir un analizador léxico usando un generador de Analizadores Lexicos.

Descripción: El alumno deberá especificar el diseño para un scanner, escribiendo las distintas expresiones regulares que conforman un lenguaje a estudiar. Deberá hacer uso de los recursos adquiridos como expresiones regulares y autómatas finitos y verificar el funcionamiento de dicho analizador léxico previamente creado.

Competencias previas: Expresiones regulares, autómatas finitos, conversión de expresiones regulares a AFND, conversión de AFND a AFD, implementación de AF.

Competencias esperadas: Se espera que el estudiante pueda crear analizadores léxicos a partir de un lenguaje regular propuesto.

Actividades Prácticas:

1. A partir de un lenguaje propuesto escribir las distintas expresiones regulares que conformaran el analizador léxico.
2. Asociar el texto representado por una expresión regular como Token.
3. Utilizar un texto fuente para comprobar el funcionamiento del analizador léxico.
4. Asociar el conteo de lineas con expresiones regulares como parte fundamental del funcionamiento de un analizador léxico.

- **Nivel:** 6

  Nombre: Implementación de una tabla de símbolos.

  Objetivo: Implementar una tabla de símbolos para un analizador léxico.

  Descripción: Implementar una tabla de símbolos como estructura de almacenamiento de datos perteneciente a un analizador léxico. El estudiante deberá probar los distintos parámetros dentro de ésta como es pero no limitado a tokens.

  Competencias previas: Construcción de analizadores léxicos, estructuras de datos, programación estructurada.

  Competencias esperadas: El estudiante es capaz de implementar una tabla

de símbolos en el análisis léxico.

Actividades Prácticas:

1. Representar una tabla de símbolos como una estructura de datos.
2. Utilizar la estructura de datos para almacenar los tokens procesados con información sobre su ubicación.
3. Verificar el estado de la tabla de símbolos al finalizar el proceso de scanning.

- **Nivel:** 7

  Nombre: Gestión de errores léxicos.

  Objetivo: Definir técnicas para el reporte de errores en analizadores léxicos.

  Descripción: Al escribir un analizador léxico se deben obtener los tokens desde un programa fuente. Se deben definir cómo enfrentar los distintos errores léxicos que pueden aparecer como: tokens no reconocidos bajo una expresión regular, lexemas con contenido inapropiado (texto procesado muy largo o no cumple con ciertas características).

  Competencias previas: Construcción de analizadores léxicos, expresiones regulares.

  Competencias esperadas: El estudiante ha adquirido técnicas para reportar errores léxicos(caracteres no reconocidos por ninguna expresión regular definida por el lenguaje regular).

  Actividades Prácticas:

  1. Utilizando un compilador de algún lenguaje reconocido, mostrar como se realiza la gestión de errores.
  2. Usar el signo punto como expresión regular para los caracteres identificados como error.
  3. Mostrar mensajes de error como parte de la gestión de errores.

Fase 2: Análisis sintáctico

- **Nivel:** 8

  Nombre: Gramáticas libres de contexto.

Objetivo: Construir Gramáticas libres de contexto.

Descripción: Especificar la notación usada en la asignatura para escribir Gramáticas libres de contexto, ilustrando: símbolos terminales, símbolos no terminales, el símbolo inicial y sus producciones.

Competencias previas: Gramáticas libres de contexto(GLC).

Competencias esperadas: El estudiante es capaz de escribir gramáticas libres de contexto utilizando una notación especifica usada en la asignatura.

Actividades Prácticas:

1. Escribir una GLC a partir de un lenguaje libre de contexto propuesto.
2. Usar el signo punto como expresión regular para los caracteres identificados como error.Asociar los elementos terminales, no terminales, signo inicial y producciones con una GLC.

- **Nivel:** 9

  Nombre: Árbol de derivación.

  Objetivo: Construir un Árbol de derivación descendente a partir de una entrada usando una GLC.

  Descripción: El estudiante deberá dibujar un Árbol de derivación descendente que permita visualizar e identificar el recorrido de una gramática libre de contexto a partir de una entrada.

  Competencias previas: GLC.

  Competencias esperadas: El estudiante debe ser capaz de construir arboles de derivación descendentes.

  Actividades Prácticas:

  1. Aplicar las técnicas vistas en la teoría para realizar una derivación por la izquierda.
  2. Aplicar las técnicas vistas en la teoría para realizar una derivación por la derecha.

- **Nivel:** 10

  Nombre: Diseño de gramáticas libres de contexto

Objetivo: Diseñar una gramática libre de contexto a partir de un lenguaje regular propuesto.
Descripción: El estudiante deberá diseñar una gramática libre de contexto a partir de un lenguaje regular propuesto y deberá probar a partir de una entrada la aceptación o no usando un árbol de derivación descendente.
Competencias previas: GLC, Arboles de derivación descendente.
Competencias esperadas: El estudiante debe ser capaz de construir una GLC a partir de un lenguaje propuesto.
Actividades Prácticas:

1. Aplicar las técnicas vistas en la teoría para realizar la derivación por la izquierda de una GLC propuesta.
2. Aplicar las técnicas vistas en la teoría para realizar la derivación por la derecha de una GLC propuesta.

Fase 2.1: Análisis sintáctico descendente

- **Nivel:** 11
  Nombre: Recursividad por la izquierda. Objetivo: Mostrar cómo eliminar de una GLC la recursividad por la izquierda
  Descripción: Aplicando los conceptos y técnicas vistas en la teoría eliminar la recursividad por la izquierda de una GLC. Identificar cuándo se ha eliminado la recursividad por la izquierda.
  Competencias previas: GLC, arboles de derivación descendente.
  Competencias esperadas: Se debe conocer como eliminar la recursividad por la izquierda en una GLC.
  Actividades Prácticas:

  1. Dada una GLC propuesta con una recursividad inmediata, aplicar un algoritmo para eliminar la recursividad por la izquierda.
  2. Dada una GLC propuesta con múltiples recursividades de varios niveles, aplicar un algoritmo para eliminar la recursividad por la izquierda.

- **Nivel:** 12

  Nombre: Factorización por la izquierda.

  Objetivo: Mostrar cómo factorizar las producciones de una GLC por la izquierda.

  Descripción: Aplicando los conceptos y técnicas vistas en la teoría factorizar por la izquierda las producciones de una GLC.

  Competencias previas: GLC, arboles de derivación descendente.

  Competencias esperadas: Se debe conocer cómo realizar una factorización por la izquierda en una GLC.

  Actividades Prácticas:

  1. Usando una GLC no factorizada por la izquierda realizar una factorizacion por la izquierda.

- **Nivel:** 13

  Nombre: Conjuntos First y Follow.

  Objetivo: Aplicar las técnicas vistas en teoría para obtener los conjuntos first y follow en una GLC.

  Descripción: Utilizando las técnicas vistas en teoría obtener los conjuntos first y follow para todos los símbolos no terminales en una GLC.

  Competencias previas: GLC, arboles de derivación descendente.

  Competencias esperadas: El estudiante debe ser capaz de adquirir el conjunto First y Follow de una GLC.

  Actividades Prácticas:

  1. Dada una gramática obtener el conjunto First y el conjunto Follow.
  2. Asociar los conjuntos First y Follow con el árbol de derivación descendente para una GLC.

- **Nivel:** 14

  Nombre: Transformar una GLC a una gramática LL.

  Objetivo: Integrar los métodos vistos previamente para transformar una GLC en una gramática LL.

Descripción: Dada una GLC aplicar los distintos métodos para obtener una gramática LL.

Competencias previas: GLC, arboles de derivación descendente, recursividad por la izquierda en GLC, factorizacion por la izquierda en GLC, conjuntos ?First y Follow?.

Competencias esperadas: El estudiante debe ser capaz de convertir GLC en gramaticas LL.

Actividades Prácticas:

1. Eliminar la recursividad por la izquierda usando una GLC.
2. Factorizar por la izquierda la gramática usando una GLC.
3. Obtener los conjuntos First y Follow de una GLC.

- **Nivel:** 15

  Nombre: Construcción de un parser descendente.

  Objetivo: Construir un parser descendente usando un generador de analizadores sintácticos.

  Descripción: Aplicando los conocimientos adquiridos el alumno deberá diseñar y probar un analizador sintáctico, probando tras una entrada la aceptación o no de una cadena de tokens.

  Competencias previas: Conversión GLC en gramática LL.

  Competencias esperadas: El estudiante debe ser capaz de construir un parser LL.

  Actividades Prácticas:

  1. Dada una GLC propuesta, obtener la gramática LL asociada.
  2. Dada una entrada, demostrar la aceptación, dado el lenguaje que reconoce el parser.

Fase 2.2: Análisis sintáctico ascendente

- **Nivel:** 16

  Nombre: Acciones desplazar y reducir.

Objetivo: Aplicar las técnicas vistas en teoría sobre la aplicación de las acciones desplazar y reducir para determinar si una cadena de entrada es aceptada por una GLC.
Descripción: El estudiante deberá realizar el recorrido de una entrada aplicando las acciones de desplazar y reducir para determinar si se llega al axioma en una GLC a partir de una cadena de entrada.
Competencias previas: GLC.
Competencias esperadas: El estudiante debe ser capaz de utilizar las acciones desplazar y reducir con una GLC.
Actividades Prácticas:

1. Dada una GLC, verificar la aceptación usando una cadena de entrada usando las técnicas desplazar y reducir.

- **Nivel:** 17
  Nombre: Árbol de derivación ascendente.

  Objetivo: Aplicar las técnicas como desplazar y reducir para generar un Árbol de derivación ascendente.
  Descripción: GLC, acciones desplazar-reducir.
  Competencias previas: El estudiante debe comprehender acciones desplazar y reducir.
  Competencias esperadas: El estudiante debe ser capaz de utilizar las acciones desplazar y reducir con una GLC para formar un árbol de derivación ascendente.
  Actividades Prácticas:

  1. Dada una GLC propuesta, usar las técnicas de desplazar y reducir para generar un árbol de derivación ascendente.

- **Nivel:** 18
  Nombre: Ambigüedad y conflictos en parsing ascendentes.
  Objetivo: Describir los conflictos en GLC utilizando las técnicas de desplazar-reducir.
  Descripción: Describir los distintos tipos de conflictos en una GLC como

desplazar-reducir, desplazar-desplazar y reducir-reducir utilizando una gramática propuesta.

Competencias previas: Arboles de derivación ascendente, acciones desplazar-reducir.

Competencias esperadas: El estudiante debe ser capaz de identificar un conflicto o ambigüedad al realizar las acciones desplazar y reducir con una GLC.

Actividades Prácticas:

1. Dada una GLC propuesta, demostrar los posibles conflictos desplazar-reducir, desplazar-desplazar y reducir-reducir.

- **Nivel:** 19

  Nombre: Tablas de parsing SLR.

  Objetivo: Construir una tabla de parsing SLR dada una gramática considerada LR.

  Descripción: Obtener la tabla de parsing ascendente SLR dada una gramática LR.

  Competencias previas: Ítems y conjuntos de ítems.

  Competencias esperadas: El estudiante debe ser capaz de construir una tabla de parsing predictivo SLR para una gramática LR con las funciones Action y GOTO.

  Actividades Prácticas:

  1. Obtener las funciones Action y GOTO para una gramática propuesta.
  2. Generar una tabla de parsing SLR.

- **Nivel:** 20

  Nombre: Autómata SLR.

  Objetivo: Aplicar los conceptos de ítems y conjunto de ítems para crear un AFD LR, a partir de una GLC.

  Descripción: Crear un autómata finito determinista LR a partir de la aplicación de los conceptos de ítems y conjuntos de ítems sobre una gramática LR utilizando las operaciones de clausura y GOTO.

Competencias previas: Funciones ACTION y GOTO.

Competencias esperadas: El estudiante debe ser capaz de construir un autómata finito para una gramática LR a partir de una tabla de parsing SLR.

Actividades Prácticas:

1. Obtener la tabla de parsing LR(0) partiendo de una gramática considerada LR.
2. Dibujar el AFD partiendo de la tabla de parsing SLR.

- **Nivel:** 21

  Nombre: Tablas de parsing LR canónico.

  Objetivo: Construir una tabla de parsing LR canónico dada una gramática considerada LR.

  Descripción: Obtener la tabla de parsing ascendente LR canónico dada una gramática LR.

  Competencias previas: Funciones ACTION y GOTO.

  Competencias esperadas: El estudiante debe ser capaz de construir una tabla de parsing predictivo LR canónico para una gramática LR.

  1. Obtener la tabla de parsing LR(0) partiendo de una gramática considerada LR.
  2. Dibujar el AFD partiendo de la tabla de parsing SLR.

- **Nivel:** 22

  Nombre: Autómata LR.

  Objetivo: Usar una tabla de parsing canónico para representar la gramática como un AFD.

  Descripción: Crear un autómata finito determinista LR utilizando una tabla de parsing LR canónica.

  Competencias previas: Tabla de parsing LR canónica.

  Competencias esperadas: El estudiante debe ser capaz de construir un AFD para una gramática LR a partir de una tabla de parsing LR canónica.

  Actividades Prácticas:

1. Obtener la tabla de parsing LR(1) partiendo de una gramática considerada LR.
2. Dibujar el AFD partiendo de la tabla de parsing LR canónica.

- **Nivel:** 23

  Nombre: Autómata LALR.

  Objetivo: Usar una tabla de parsing LALR para representar la gramática como un AFD.

  Descripción: Crear un autómata finito determinista LR utilizando una tabla de parsing LALR.

  Competencias previas: Tabla de parsing LALR, AFD LR canónico.

  Competencias esperadas: El estudiante debe ser capaz de construir un AFD para una gramática LR a partir de una tabla de parsing LR canónica.

  Actividades Prácticas:

  1. Obtener la tabla de parsing LALR partiendo de una gramática considerada LR.
  2. Dibujar el AFD partiendo de la tabla de parsing LALR.
  3. Asociar las reducciones ha realizar en el AFD LR canónico para generar el AFD LALR.

- **Nivel:** 24

  Nombre: Tablas de parsing LALR.

  Objetivo: Construir una tabla de parsing LALR(1) dada una gramática considerada LR y una tabla de parsing LR canónico.

  Descripción: Obtener la tabla de parsing ascendente LALR dada una gramática LR. Se debe obtener partiendo de una tabla de parsing LR canónico y su AFD.

  Competencias previas: Autómata LR, tabla de parsing LR canónico, Autómata LALR.

  Competencias esperadas: El estudiante debe ser capaz de construir una tabla de parsing predictivo LALR a partir de una tabla de parsing LR canónica.

1. Obtener la tabla de parsing LR Canónico partiendo de una gramática considerada LR.
2. Realizar las verificaciones de estados comunes en la tabla de parsing LR Canónico y verificar usando el autómata LR Canónico.

- **Nivel:** 25

  Nombre: Parsing ascendente.

  Objetivo: Construir un parser ascendente usando un generador de analizadores sintácticos.

  Descripción: Aplicar los conceptos vistos sobre parsing LR para proponer una gramática LR y construir un parser ascendente. Probar el parser usando una cadena de tokens y determinar su aceptación.

  Competencias previas: Análisis léxico, GLC, tabla de parsing LALR.

  Competencias esperadas: El estudiante debe ser capaz de construir un parser LR.

  Actividades Prácticas:

  1. Construir una gramática LR.
  2. Dada una entrada, demostrar la aceptación, dado el lenguaje que reconoce el parser.
  3. Asociar los elementos analizados en el análisis léxico con la entrada al analizador sintáctico.

- **Nivel:** 26

  Nombre: Tabla de símbolos.

  Objetivo: Asociar la tabla de símbolos generada en el Análisis léxico con el análisis sintáctico.

  Descripción: Asociar el parsing ascendente o descendente con la tabla de símbolos generada en el proceso de Análisis léxico. Estudiar la cadena de entrada de tokens como el recorrido de la tabla de símbolos.

  Competencias previas: Parser LR, análisis léxico, tablas de símbolos.

  Competencias esperadas: El estudiante debe ser capaz de utilizar los datos

de la tabla de símbolos en el proceso de construir un analizador sintáctico.

Actividades Prácticas:

1. Construir una gramática LL o LR.
2. Utilizar una tabla de símbolos previamente generada en el análisis léxico como entrada para el parser.

- **Nivel:** 27

  Nombre: Técnicas para la detección y recuperación de errores.

  Objetivo: Aplicar técnicas para recuperación de errores.

  Descripción: aplicar técnicas como aumento de gramática, modo pánico y modo frase para recuperar errores en una gramática LL o LR.

  Competencias previas: Construcción de Parser LL o Parser LR, tabla de símbolos.

  Competencias esperadas: El estudiante debe ser capaz de utilizar algunas técnicas vistas en la teoría para detectar, reportar y recuperarse de un error sintáctico.

  Actividades Prácticas:

  1. Utilizar modo pánico y ampliación de la gramática para manejar errores en un parser.
  2. Utilizar una tabla de símbolos o modificación al flujo de entrada del parser para introducir tokens que permitan demostrar el uso de modo frase para recuperación de errores.

Fase 3: Análisis semántico ascendente

- **Nivel:** 28

  Nombre: Ámbito de variables.

  Objetivo: Extender el concepto de tabla de símbolos para verificar el ámbito de variables.

  Descripción: Usar las tablas de símbolos para la verificación de variables y su alcance en un programa fuente, incluyendo la verificación de declaraciones de

variables.

Competencias previas: Construcción de Parsers LL o Parsers LR, tabla de símbolos.

Competencias esperadas: El estudiante debe poder construir parsers con la capacidad de reconocer el alcance de variables, así como reportar errores semánticos en cuanto al alcance de variables detectadas.

Actividades Prácticas:

1. Utilizar una tabla de símbolos para representar el ámbito en que se encuentra el parser.
2. Utilizar una tabla de símbolos para representar la declaración de una variable en un ámbito bien definido dentro de la tabla de símbolos.

- **Nivel:** 29

  Nombre: Comprobación de tipos.

  Objetivo: Usar una tabla de símbolos para comprobar los tipos de variables.

  Descripción: Comprobar usando una tabla de símbolos los tipos de variables incluyendo las operaciones realizadas dentro de un programa fuente.

  Competencias previas: Construcción de Parsers LL o Parsers LR, tabla de símbolos.

  Competencias esperadas: El estudiante debe poder construir parsers con la capacidad de reconocer los tipos de variables, así como reportar errores semánticos en cuanto a los tipos de las variables detectadas.

  Actividades Prácticas:

  1. Utilizar una tabla de símbolos para realizar las comprobaciones de tipos entre variables ya definidas previamente al realizar operaciones.

Fase 4: Generación de Código Intermedio

- **Nivel:** 30

  Nombre: Traducción orientada a la sintaxis.

  Objetivo: Aplicar acciones semánticas para realizar una traducción orientada a

la sintaxis.

Descripción: Utilizando una gramática LL o LR realizar como traducción de expresiones semánticas la generación de código.

Competencias previas: Análisis léxico, análisis sintáctico, verificación de ámbitos de variables y comprobación de tipos.

Competencias esperadas: El estudiante deberá ser capaz de generar traductores orientados a la sintaxis utilizando las técnicas adquiridas en análisis léxico, análisis sintáctico y análisis semántico.

Actividades Prácticas:

1. Construir un parser para un lenguaje propuesto, incluyendo sus acciones semánticas.
2. Describir las acciones semánticas para la traducción orientada a la sintaxis.

- **Nivel:** 31

  Nombre: Generación de código de tres direcciones.

  Objetivo: Aplicar acciones semánticas para generar código intermedio de tres direcciones dado un lenguaje propuesto.

  Descripción: Utilizando una gramática LL o LR realizar como acciones semánticas la generación de código intermedio.

  Competencias previas: Traducción orientada a la sintaxis.

  Competencias esperadas: El estudiante deberá ser capaz de generar código de tres direcciones utilizando las técnicas adquiridas en análisis léxico, análisis sintáctico y análisis semántico.

  Actividades Prácticas:

  1. Identificar si una expresión cumple con el criterio de un código de tres direcciones.
  2. Realizar el recorrido de un código de tres direcciones, resaltando el estado final de una variable propuesta dentro del código.
  3. Traducir una expresión matemática dada una GLA especifica en código de tres direcciones.

- **Nivel:** 32

  Nombre: Generación de flujo de control.

  Objetivo: Asociar las técnicas de generación de código de tres direcciones para generar instrucciones de flujo de control.

  Descripción: A partir de la generación de código de tres direcciones asociar las técnicas para la generación de flujo de control de código.

  Competencias previas: Generación de código de tres direcciones.

  Competencias esperadas: El estudiante deberá ser capaz de generar instrucciones de flujo de control por medio de etiquetas a un código generado de tres direcciones.

  Actividades Prácticas:

  1. Demostrar el uso de la etiqueta Goto.
  2. Verificar el estado de una variable al recorrer código de tres direcciones con el operador de flujo goto.
  3. Traducir instrucciones basadas en estructuras de decisión en código de tres direcciones.

Capítulo 3

Diseño del Framework

Las prácticas del curso requieren del establecimiento de conceptos, técnicas y criterios relacionados a un determinado contenido teórico. A la vez deben permitir acceder a distintas herramientas que deben ser utilizadas para el desarrollo de las actividades propuestas. En general, a esta colección de conceptos, prácticas, criterios y herramientas se le puede llamar *framework para Desarrollo de Prácticas*, y en particular al aplicarlo a esta asignatura se le puede denominar *framework para Desarrollo de Prácticas de compiladores*

El *framework* permite desarrollar las actividades por parte de los alumnos utilizando una o varias herramientas tecnológicas (aplicaciones), visualizando el contenido de las prácticas y permitiendo la realización de evaluaciones. Además hace posible el uso de plantillas predefinidas para generar nuevas clases prácticas y evaluaciones de un curso. A su vez, almacena información importante de cada usuario como las calificaciones obtenidas en las prácticas de un curso en particular, lo que permite realizar un seguimiento del mismo.

Para las prácticas del curso son necesarias utilidades o herramientas que permitan realizar las actividades propuestas. A este conjunto de utilidades integradas en el *framework* se le llamó *Caja de Herramientas*.

3.1 Componentes del Framework

El *framework* está compuesto por varias partes que por separado realizan una tarea específica, y a su vez son independientes entre ellas, pero al integrarse conforman el *framework*.

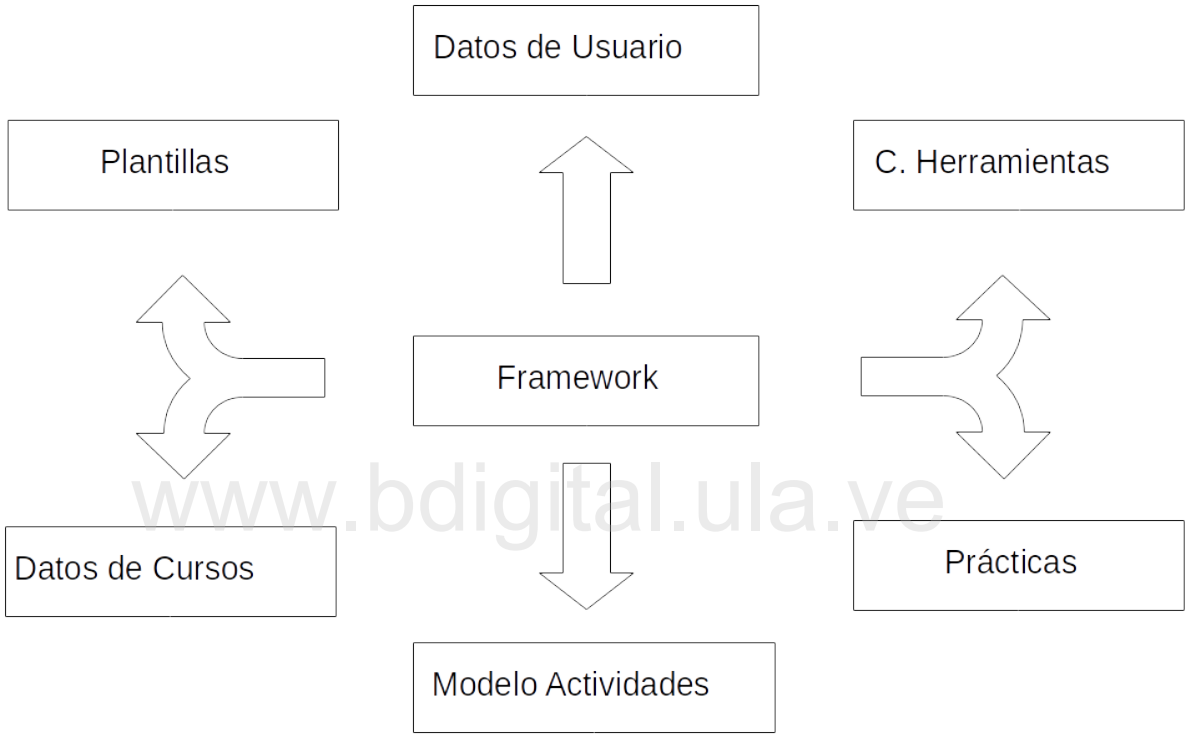

Figura 3.1: Modelo Lógico

Los componentes integrados del *framework* proporcionan un espacio en donde almacenar los datos de los usuarios, en conjunto con las plantillas de las prácticas y sus respectivas actividades, las prácticas, las herramientas que se utilizarán y toda la información referente a los cursos.

A continuación se detallan los Componentes del *framework*.

3.1.1 Caja de Herramientas

La *Caja de Herramientas* es un conjunto de herramientas de *software* que son necesarias para desarrollar las prácticas del curso.
A continuación se muestra un listado de las principales funciones que deberán tener estas herramientas.

- Compilación y Ejecución de código: Es una herramienta que permite compilar el código fuente de un programa y ejecutarlo para obtener una salida.
- Verificación de Expresiones Regulares: A modo de prueba se valida que una Expresión Regular esté bien escrita y que un texto propuesto coincida con la misma.
- Construcción de AFND y AFD a partir de una Expresión Regular: Se construye la representación gráfica de un Autómata a partir de una Expresión Regular introducida por el usuario.
- Generación Automática de Analizadores Léxicos: Se genera un Analizador Léxico a partir de un conjunto de Expresiones Regulares y reglas léxicas.
- Verificación de Gramáticas Libres de Contexto: Valida que una Gramática Libre de Contexto introducida esté correctamente escrita.
- Visualización de Arboles de Derivación: A partir de una entrada se obtienen Arboles de Derivación ascendentes y descendentes.
- Verificación de Gramáticas LL: Verifica si una Gramática Libre de Contexto es LL.
- Obtención de conjuntos *FIRST* y *FOLLOW*: Obtiene los conjuntos *FIRST* y *FOLLOW* a partir de una Gramática Libre de Contexto.
- Transformación de Gramáticas Libres de Contexto a una Gramática LL: Construye una Gramática LL equivalente a partir de una Gramática Libre de Contexto (factorización por la izquierda y eliminación de la recursión por la izquierda).

- Construcción de un Parser Descendente: Construye un Parser a partir de una Gramática LL.
- Obtención de Tablas de Parsing SLR, LR(Canónico), LALR y los Autómatas Finitos Deterministas correspondientes: Construye las tablas de *parsing* a partir de una Gramática considerada LR.
- Construcción de un Parser Ascendente: Construye un Parser a partir de una Gramática LL.

3.1.2 Molde de Resolución de Asignaciones o Base

La Base es el espacio que permite a los usuarios introducir el código o respuestas necesarias para resolver un problema, sin preocuparse por la configuración correcta de las aplicaciones en un equipo local. Esta Base puede considerarse como una plantilla para la resolución de un problema o ejercicio en una práctica planteada, por lo tanto, depende del enunciado de la misma. Según la asignación, la Base contiene material de apoyo o proporciona simplemente un espacio en blanco donde el usuario puede introducir su respuesta. Una práctica puede contener más de una Base y a la vez puede decidir si utilizarla o no.

3.1.3 Respuesta de Usuario

El usuario da respuesta al problema planteado y la escribe en la Base, en caso de no utilizar alguna Base, el usuario escribe su respuesta en un documento vacío destinado para tal fin. Esta respuesta es procesada junto con una herramienta (de la Caja de Herramientas) para producir una Salida.

3.1.4 Herramienta

Para procesar la respuesta del usuario sobre el problema planteado, es necesario utilizar una herramienta de *software*. Esta herramienta proveniente de *La Caja de Herramientas* y recibe la respuesta de usuario, la procesa y produce una salida.

3.1.5 Salida

La salida es la solución de un problema propuesto en una práctica. Es el resultado de la respuesta de usuario y el análisis realizado por la herramienta utilizada, y se compara con una respuesta previamente almacenada, con el fin de establecer la correctitud de la respuesta dada por el alumno.

3.1.6 Base de Datos

Con la finalidad de integrar todos los componentes del *framework* se utiliza una Base de Datos que permite relacionarlos en una aplicación de *software*. Esta Base de Datos almacena todos los datos asociados a los usuarios, prácticas, las bases de resolución, las funciones de las herramientas y las instancias de cada curso.

El diseño de la Base de Datos se muestra en el siguiente diagrama de clases.

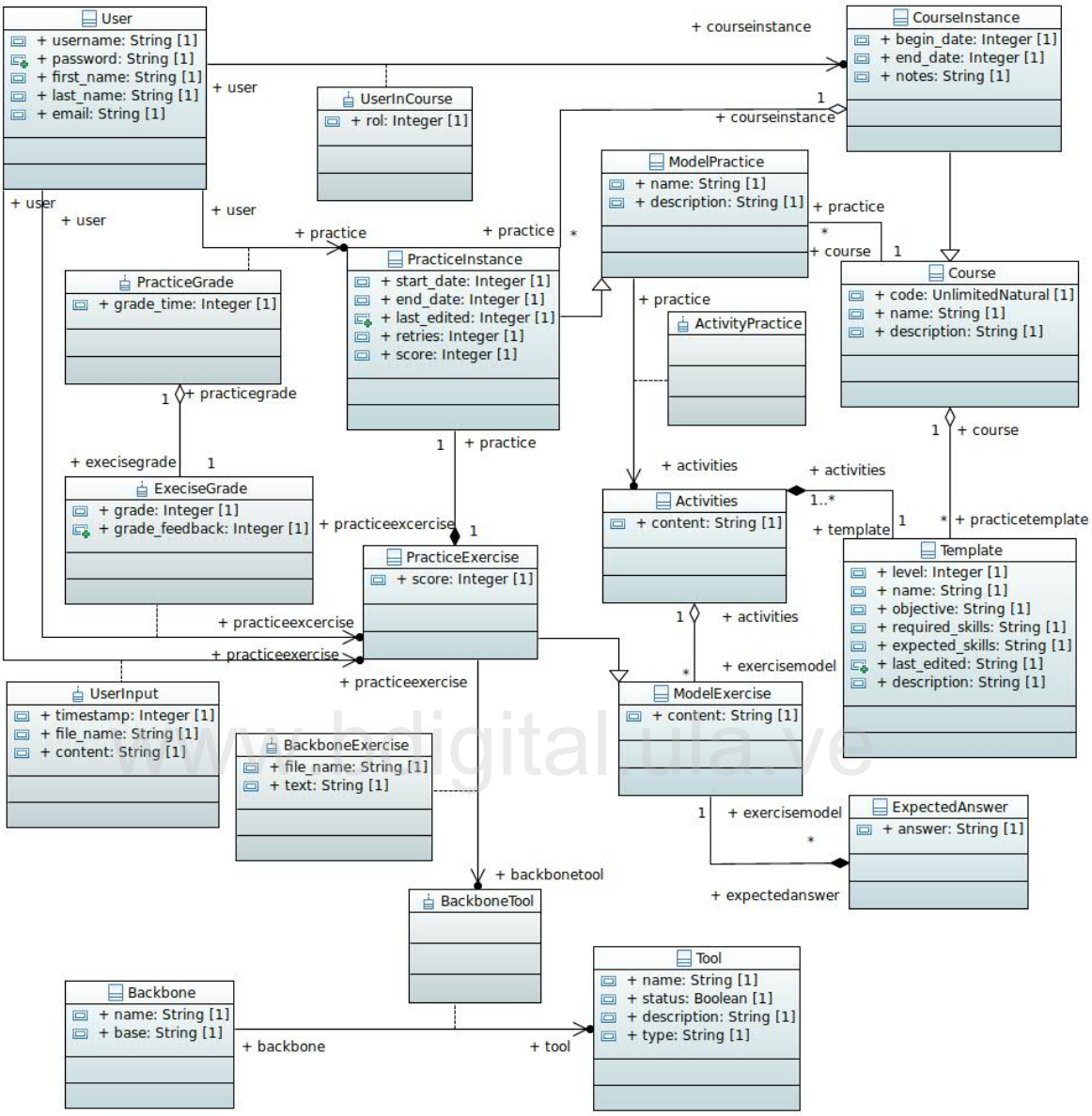

Figura 3.2: Modelo de Datos

El diagrama mostrado está conformado por 19 tablas las cuales representan los usuarios del *framework*, los cursos disponibles en el *framework* (en este caso Compiladores), las plantillas de las prácticas del curso, los modelos de las prácticas y ejercicios que estarán en las evaluaciones. Al mismo tiempo almacena las distintas instancias de los cursos, tales como, las Bases y las herramientas que se utilizan en el *framework*.

Cada tabla del diagrama de clases cumple una función específica, y se describen a continuación.

- **User** (Usuario): Esta tabla almacena la información relacionada a los usuarios, tales como el nombre de usuario, la contraseña, el primer nombre de la persona, el apellido de la persona y su correo electrónico.
- **Course** (Curso): Esta tabla almacena información breve sobre un curso en particular, como es el nombre, la descripción y código numérico único para el curso.
- **CourseInstance** (Instancia de Curso): Esta tabla almacena todo lo referente a las distintas instancias de un mismo curso, permitiendo tener más de un curso a la vez. Acá se almacenan datos como la fecha de inicio, la fecha de fin y observaciones del curso.
- **Template** (Plantilla): En esta se tabla almacena toda la información de las plantillas de un curso, como lo es el nivel de la práctica, su nombre, su objetivo, las competencias requeridas y esperadas de la misma y una breve descripción de la plantilla.
- **Activity** (Actividad): Esta tabla almacena la información de cada actividad que debe ser realizada acorde a una plantilla.
- **ModelExercise** (Ejercicio Modelo): Esta tabla cumple la función de banco de preguntas, donde se almacenan las posibles preguntas para cada actividad.
- **ExpectedAnswer** (Respuesta Esperada): Esta tabla almacena las posibles respuestas a una actividad que un estudiante debería acertar.
- **Practice** (Práctica): Esta tabla cumple la función de banco de prácticas para un curso en particular.
- **PracticeActivity** (Actividad de Práctica): Esta tabla almacena las actividades que se tienen que realizar en una práctica.

- **UserInCourse** (Usuario en Curso): Esta tabla almacena el rol de cada usuario en cada instancia de un curso, dando la posibilidad de que el usuario pueda ascender de estudiante a educador en un curso en particular. Además, permite que haya más de un educador en un curso.

- **PracticeInstance** (Instancia de Práctica): Esta tabla almacena la información de una práctica específica en una instancia de curso determinada, usando un modelo del banco de prácticas.

- **PracticeExercise** (Ejercicio de Práctica): En esta tabla se almacena la calificación que tendrá un ejercicio que tiene como modelo una actividad del banco de actividades. El ejercicio debe tomar como modelo solo aquellos que se encuentren en el listado de actividades del modelo de práctica.

- **UserInput** (Entrada de Usuario): En esta tabla se almacena la respuesta de un usuario a un ejercicio. Por cada usuario se creará un archivo con su respectiva respuesta.

- **Backbone** (Base): Esta tabla almacena todas las Bases que un usuario puede utilizar para resolver un ejercicio.

- **Tool** (Herramienta): Esta tabla almacena todos los datos de las posibles herramientas que se usarán en el *framework* al momento de procesar la Entrada de Usuario. Además, almacena información que permite verificar el buen funcionamiento de la herramienta.

- **BackboneTool** (Herramienta de Base): Esta tabla permite asociar una Base con una herramienta.

- **BackboneExercise** (Herramienta por Ejercicio): Esta tabla permite asociar una Base con un Ejercicio. También permite anexar información adicional a la Base.

- **ExerciseGrade** (Calificación de Ejercicio): En esta tabla se almacena la calificación del ejercicio para cada estudiante, junto con posibles consideraciones por parte del educador.

- **PracticeGrade** (Calificación de Práctica): Esta tabla almacena una referencia a la calificación total de una práctica para cada estudiante.

3.2 Tabla de Operaciones

La Tabla de Operaciones define todas las acciones que puede realizar un usuario para interactuar con el *framework*. Esta Tabla se utiliza para poder controlar el *framework*, es común para todos los componentes e interactúa con cada uno permitiendo que se relacionen todos entre si y así habilitando una nueva función.

3.2.1 Interacción con la Tabla de Operaciones

La Tabla de Operaciones recibe instrucciones acompañadas de entradas de usuario que definen la función que va a tener el *framework* en ese momento.
Cuando un usuario realiza una acción se procesa acorde a la instrucción solicitada (por el usuario) y se muestra una salida relacionada a la petición.

Acciones de la Tabla de Operaciones

La Tabla de Operaciones permite realizar una serie de acciones, las cuales pueden ofrecer distintas características dependiendo del usuario. A continuación se muestran las acciones principales que pueden ser realizadas dentro del *framework*:

- Visualizar la información de un curso.
- Crear una instancia de un curso.
- Visualizar las plantillas del curso y las actividades asociadas a cada plantilla.
- Crear una práctica a partir de un modelo de prácticas previamente especificado.
- Crear ejercicios para cada práctica según las actividades definidas para cada plantilla.
- Verificar las calificaciones de los usuarios así como agregar una puntuación personal por ejercicio.

- Visualizar las prácticas disponibles.
- Agregar las posibles soluciones de un ejercicio.
- Evaluar la práctica de un usuario.

Las acciones necesarias para crear una práctica se ilustran en la figura 3.3

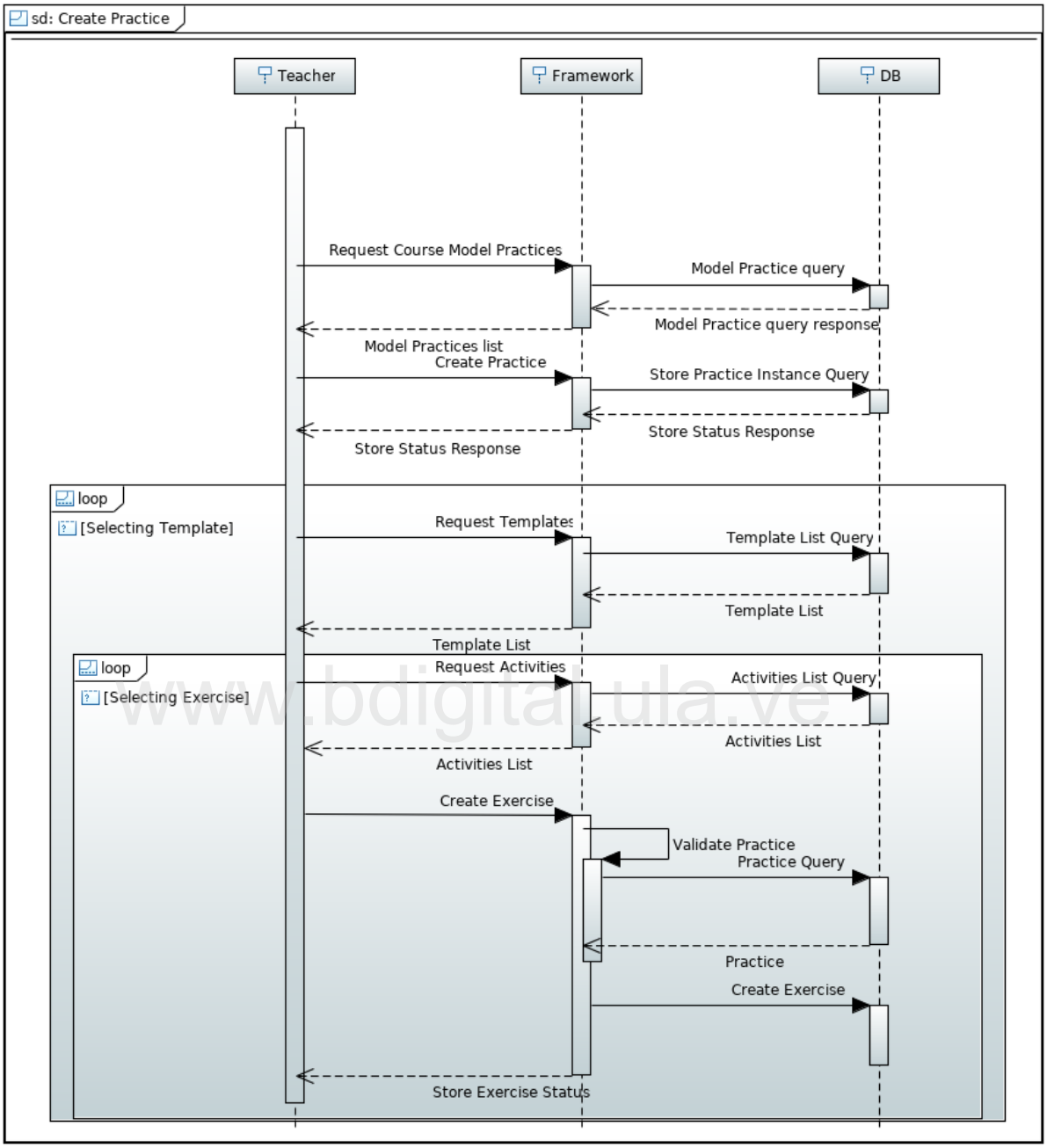

Figura 3.3: Creación de una Práctica

Capítulo 4

Herramientas para la Implementación del Framework

En este capítulo se explican las herramientas de *software* utilizadas para llevar a cabo el diseño propuesto en el capitulo anterior.

Para brindar una mayor flexibilidad y accesibilidad se eligió desarrollar una aplicación web que permitiese un fácil acceso a los recursos, permitiendo así que los usuarios pudiesen acceder de forma remota.

Para la implementación del *framework* es necesario utilizar herramientas que proporcionen ciertas facilidades, en este caso se desea generar un *framework* accesible a los usuarios, que permita completar todas las operaciones de las prácticas.

4.1 Framework Web

Un *framework Web* es un *framework* diseñado para complementar el desarrollo de aplicaciones o servicios que utilicen la *Web*, tratando de facilitar tareas que incluyen pero no se limitan al acceso de Bases de Datos, gestión de sesiones y reutilización de código.

Estas características de un *framework Web* son deseables porque reducen en gran

cantidad las tareas necesarias para llevar a cabo la realización del *framework*.

Actualmente existen múltiples *framework Web* que pudiesen ser seleccionados para complementar el desarrollo de este proyecto. Están escritos en distintos lenguajes de programación y prestan servicios incorporados que pudiesen ser deseables.

4.1.1 API Rest

El *framework* puede construirse como una aplicación en que los usuarios interactuan con distintos aspectos del mismo. Sin embargo es necesario mantener las interacciones lo más simple posible, dejando espacio para ampliar las características del *framework*. Para esto, el estilo de arquitectura REST (Representational State Transfer por sus siglas en inglés) fue seleccionado para complementar el diseño del *framework* como aplicación.

REST es un estilo de arquitectura se utiliza el protocolo HTTP para generar una interacción entre máquinas. Las aplicaciones REST o RESTful realizan peticiones o "llamadas" usando el protocolo HTTP para realizar operaciones CRUD sobre datos. Esta arquitectura presenta diversas ventajas las cuales son listadas a continuación:

- Cada petición HTTP contiene toda la información necesaria para ejecutar una operación. Esto permite eliminar la necesidad de estados previos para su completitud.
- Las operaciones más importantes sobre datos (CRUD) están bien definidas por medio de las especificaciones HTTP, POST(crear), GET(leer), PUT(editar) y DELETE(eliminar).
- Todo objeto REST se manipula por medio de una URL por lo que cualquier recurso puede ser de fácil acceso.
- Es un sistema de capas que permite una arquitectura jerárquica entre componentes.
- Es de fácil acceso por lo que compartir un recurso por medio de la red es posible mediante direcciones o URL.

- Permite separar una aplicación de cualquier aplicación de terceros ya sea a través de interfaces *web* u otros servicios REST.
- La información enviada es de fácil lectura usando formatos JSON o XML.
- Implementa un conjunto de estados para indicar el estatus de la operación realizada.

4.1.2 Frameworks para desarrollo de aplicaciones

Django Framework

Django es un *framework* para construir aplicaciones *web*, es gratuito, *open source*, está escrito en Python y ofrece componentes listos para ser utilizados. Está pensado para el desarrollo ágil, cuenta con seguridad incorporada (manejo de autenticación y autorización), es flexible y escalable. Sigue un modelo de arquitectura Modelo-Vista-Plantilla donde el Modelo hace referencia a una estructura de datos, la vista ofrece detalles sobre el contenido a mostrar, así como también operaciones previas sobre ese contenido; y finalmente la plantilla permite especificar posibles opciones para la visualización. Otras bondades de este *framework* son:

- Django se está desarrollando activamente desde el 2005, por lo que ha sido probado con el paso del tiempo.
- Ha sido usado en distintos proyectos en el mercado de aplicaciones *web*, (Pinterest e Instagram). Demostrando ser escalable y ha permitido manejar una gran cantidad de usuarios.
- Django cuenta con múltiples paquetes y utilidades administrados por parte de una comunidad (Comunidad Django) lo que permite adaptar cualquiera de ellos ante cualquier necesidad emergente.
- Cuenta con una documentación actualizada.
- Permite realizar conexiones con distintos sistemas de gestiones de Bases de Datos.

Django-REST-Framework

Django REST Framework es un conjunto de herramientas utilizadas para construir *Web APIs*. Creado para ser utilizado con Django, permite construir la aplicación usando la arquitectura REST y presenta las siguientes ventajas:

- Implementa políticas de autenticación.
- Permite el uso de "serializadores", los cuales transforman datos complejos utilizando el protocolo HTTP a tipos de datos en formato JSON o XML, lo que elimina en gran parte la necesidad de crear código que interprete el contenido de una llamada.
- Ofrece seguridad incorporada para las peticiones o llamadas a ser realizadas.

4.1.3 Base de Datos

Un *framework Web* facilita la conexión con una Base de Datos. La Base de Datos utilizada fue especificada mediante el esquema de modelos de Django. Este esquema permite generar una Base de Datos con cualquier gestor de Base de Datos SQL deseado. En este caso se utilizó SQLite3, la cual es una Base de Datos que puede almacenarse completamente en un archivo. El Sistema de Gestión de Base de Datos es seleccionado por parte de un administrador por lo que la misma puede ser cambiada siempre y cuando el Gestor de Base de Datos sea *SQL* en la versión de probada de Django(versión 1.11)

4.2 Herramientas Para el Framework

El *framework* requiere de herramientas externas que faciliten la tarea de resolver las prácticas planteadas por el docente. En este proyecto se incorporaron algunas herramientas de *software* que permiten resolver las actividades propuestas. Sin embargo, es posible agregar más herramientas tecnológicas si así se desea.

Las herramientas que se integraron al *framework* fueron las siguientes:

- **gcc**: es un *Front End* para los lenguajes de programacion C, C++, Fortran, Ada y Go.
- **python3**: Intérprete del lenguaje de programación Python en su versión tres(3).
- **Flex**: "Es un analizador léxico rápido. Es una herramienta para generar programas que realizan comparaciones entre patrones y un texto" [1].
- **GNU Bison**: Es un generador de parsers de propósito general que convierte una GLC en un parser LR utilizando las tablas de parsing LALR(1) [2].

Capítulo 5

Implementación Del Framework

En este capítulo se explica cómo se implementó el diseño propuesto en el capítulo 3, junto con las herramientas seleccionadas en el capítulo 4 para construir el *framework*. A su vez, se describen los componentes del *framework*, se indican sus características y se explica su funcionamiento. También, se presenta un manual de uso que permite replicar la experiencia desde el inicio.

La implementación de este sistema se dividió en dos secciones fundamentales:

1. Interfaz de Programación de Aplicaciones (API).
2. Caja de Herramientas.

5.1 Desarrollo Guiado por Pruebas

Para la creación de este sistema se usó la técnica de Ingeniería de Software conocida como Desarrollo Guiado por Pruebas o TDD por sus siglas en inglés. TDD es una práctica que involucra escribir las pruebas de un sistema antes de construir la aplicación [14]. Al utilizar TDD se espera en un principio que las pruebas fallen pues los recursos y características de la aplicación no se encuentran definidas, luego se desarrollan todas las funcionalidades de la aplicación cumpliendo con los requisitos previstos por el diseño de las pruebas para obtener resultados satisfactorios. Esta técnica garantiza tanto el desarrollo de un sistema que cumple con todos los casos de usos propuestos y que se encuentren validados.

5.2 Estructura de directorios del Framework

Al construir una aplicación en **Django** se genera un directorio base o principal con el nombre de la aplicación. En este directorio se pueden encontrar los archivos de configuración de la aplicación. Al mismo nivel del directorio principal se crean dos directorios más; el primero es el archivo raíz de la API y el segundo contiene las herramientas del *framework*.

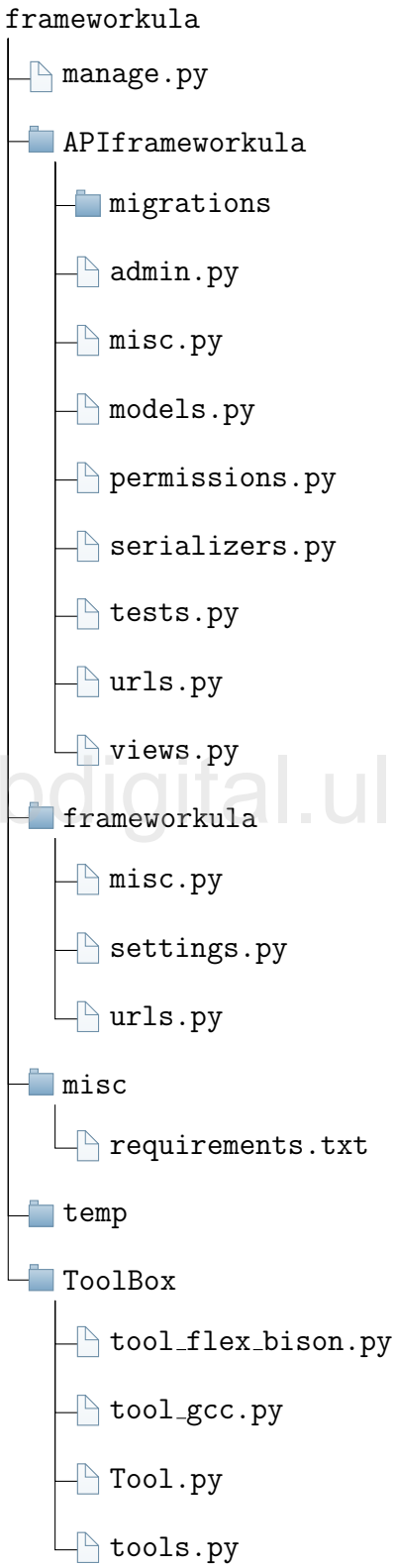

Figura 5.1: Estructura de directorios del framework

En la figura 5.2 se pueden observar los directorios APIframeworkula y ToolBox, los mismos corresponden a la API y a las herramientas del *framework*.

5.3 Creación de la API para el Framework

Para la creación de la API, se generó una aplicación en **Django**, con la cual se creó un directorio llamado APIframeworkula. En este directorio están contenidos los distintos archivos que conforman el núcleo de la aplicación. El núcleo de la API está formado por modelos, vistas, serializadores y definiciones de URLs.Todos estos elementos en conjunto hacen posible el funcionamiento del *framework*. También la creación de la API incluyó el desarrollo de pruebas unitarias, a través de las cuales fue posible diseñar y validar la aplicación.

A continuación se describen de manera detallada los componentes de la API:

5.3.1 Modelos

A partir de los modelos se crea la Base de Datos del *framework*. Estos modelos no son más que múltiples clases (creadas usando el formato de **Django**) que junto con sus atributos, son transformadas a una especificación de una Base de Datos SQL.

Los modelos se encuentran definidos en el archivo **models.py**.

5.3.2 Vistas

Las vistas son métodos para interactuar con un recurso. Para cada recurso se crearon distintas funciones que permiten realizar las operaciones de creación, edición, modificación y/o eliminación. Las vistas están contenidas en el archivo **views.py** y proporcionan respuesta a las solicitudes hechas por el usuario. Cada recurso se accede según la vista como un método de una clase.

Los métodos fueron implementados para garantizar **los códigos de estado HTTP**.

Las vistas utilizan serializadores para transformar la información de entrada y/o salida.

5.3.3 Serializadores

Serializar es el proceso de traducir un tipo de datos con sus respectivos atributos a un formato en texto plano como **JSON** o **XML**. Bajo una definición sobre los campos a utilizar, un serializador permite serializar información obtenida de una consulta a la Base de Datos para ser mostrada como respuesta en formato **JSON**.

Un serializador es una clase que toma como referencia a otra clase especificada en los modelos ya definidos, seleccionando exactamente los atributos que se desean transformar.

En el archivo**serializers.py** se encuentran todos los modelos de serializadores utilizados para complementar las vistas.

El formato utilizado en la serializacion de los modelos es **JSON**.

5.3.4 Direcciones URLs

Por medio de un conjunto de expresiones regulares se realiza un mapeado entre las URLs y las vistas en el archivo **urls.py**. Cada recurso tiene una URL específica que garantiza el acceso a los recursos de manera única.

Cada URL tiene una vista asociada que permite realizar la acción solicitada.

5.3.5 Pruebas unitarias

Se escribieron pruebas unitarias para validar el comportamiento de cada aspecto del sistema. Cada prueba está contenida en un caso de pruebas, el cual consiste en preparar un escenario para la realización de pruebas. La manera de ejecutar estas pruebas es aplicándolas sobre una Base de Datos vacía del sistema, con la intención de que la información previamente ingresada no cambie el comportamiento de las pruebas unitarias.

Las pruebas se codificaron en el archivo **tests.py** y se desarrollaron 58 métodos en total, que exponen situaciones donde se prueban las distintas características de la aplicación.

El resultado de las pruebas depende de la correcta configuración de la aplicación así como de la instalación de las dependencias necesarias.

5.4 Integración de Herramientas

El *framework* ofrece distintas herramientas para la implementación y resolución de las prácticas planteadas en él. El usuario puede utilizar estas herramientas por medio de plantillas establecidas o espacios en blanco. El usuario debe ser capaz de recibir información acorde a la interacción con la herramienta, dando la posibilidad de comprender el funcionamiento de la misma. Se puede interactuar con las herramientas de distintas formas (Herramientas de Terceros) por lo que es necesario establecer una única manera de interacción. Para ello se realizaron las siguientes consideraciones:

- Una herramienta debe tener un nombre único por lo cual identificarla.
- Una herramienta dentro del *framework* debe seguir el flujo **Entrada-Proceso-Salida**.
- Una herramienta debe permitir detectar si la misma se encuentra disponible.
- La herramienta debe tener una interfaz clara para poder utilizarla.

Dada estas consideraciones se desarrolló un esquema que permite integrar cualquier herramienta siguiendo los principios de uso antes mencionados. El mismo puede verse como una clase abstracta dentro del *framework*. En la figura 5.4 se muestra una estructura para el directorio de herramientas:

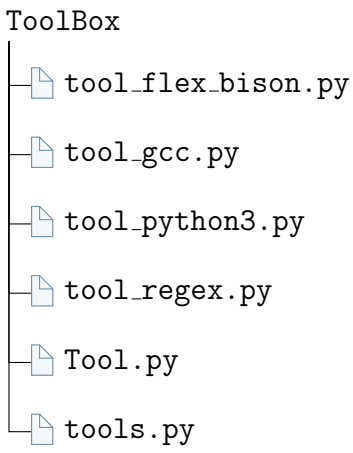

Figura 5.2: Estructura del directorio de herramientas del framework

5.4.1 Clase abstracta Herramienta

El archivo **Tool.py** ubicado en 5.4 contiene una clase abstracta llamada **GenericTool**. Esta clase tiene dos métodos **self_test** y **execute_tool**. El primer método es un mecanismo para comprobar la disponibilidad de la herramienta y el segundo es un mecanismo para obtener una salida a partir de una entrada específica.

A continuación se muestra la estructura de la herramienta genérica **GenericTool**:

Listing 5.1: Herramienta Generica

```
class GenericTool(ABC):


    @abstractmethod
    def self_test(self):
        pass


    @abstractmethod
    def execute_tool(self,path):
        pass


    def __init__(self):
        self.status=not self.self_test()
```


5.4.2 Convención de nombres e instalación de herramientas al Framework

Las herramientas que se vayan a incluir al *framework* deben ser creadas en archivos independientes. Estos archivos deben cumplir con el formato de nombre específico **tool_nombre.py**.

Las herramientas deben ser agregadas manualmente en el archivo **tools.py** incluyendo el nombre con el cual se identificarán en la Base de Datos.

A continuación se muestra un ejemplo del formato:

Listing 5.2: Listado de herramientas

```
INSTALLED_TOOLS ={
    'gcc':Tool_gcc,
    'flex':Tool_flex,
    'python3':Tool_python3,
    }
```

5.4.3 Compilación y Ejecución

El método **execute_tool** dentro de la implementación de cada herramienta debe tener como entrada el directorio donde se ubican los archivos proporcionados por el usuario. Estos archivos originalmente se almacenan en la Base de Datos. Al requerir la ejecución de la herramienta, se copian en un subdirectorio de la carpeta de recursos temporales **temp**. El subdirectorio es creado con el nombre de la práctica con la combinación del nombre de usuario y el identificador único del ejercicio a ejecutar; evitando cualquier colisión y permitiendo múltiples instancias de la herramienta.

La herramienta instalada debe contar con los mecanismos apropiados para compilar (si lo amerita) y ejecutar.

5.4.4 Uso de múltiples Herramientas

El uso de múltiples herramientas por ejercicio no se implementó dentro del *framework*. Sin embargo, si es necesario utilizar una secuencia de herramientas se puede crear una herramienta "Combo" la cual sigue los mismos patrones explicados anteriormente. El nombre de esta herramienta debe estar separado por espacios indicando el orden en que deben ser ejecutadas, por ejemplo:

"tool_flex_bison.py" de esta forma se denota que la herramienta **flex** debe ser utilizada antes que **bison** para producir una salida.

5.5 Instalación y Configuración

5.5.1 Instalación de la Aplicación

Para instalar la aplicación se deben seguir los siguientes pasos:

1. Clonar el proyecto del repositorio anexo.
2. Instalar pip3[1].
3. Desde la carpeta **root** del proyecto, instalar las dependencias mediante el comando:

```
$ sudo pip install -r misc/requirements.txt
```

4. Verificar que la instalación cumple con todos los requisitos al ejecutar todas las pruebas unitarias y obtener un resultado exitoso con el comando:

```
$ python manage.py test
----------------------
Ran 58 tests in 4.064s


OK
```


5.5.2 Configuración del Framework

Configuración Básica del Framework

Para modificar la configuración por defecto del proyecto se debe sobrescribir el archivo **settings.py** según las especificaciones requeridas por el usuario. Estas configuraciones incluyen el Gestor de Base de Datos que se vaya a utilizar (por defecto SQLite3).

[1]Dependiente de la distribución linux. Para distribuciones basadas en debian: apt-get install python3-pip

Primeros Pasos

Luego de descargar la aplicación es necesario realizar los siguientes pasos para poder utilizarla:

- Crear la especificación de la Base de Datos usando el comando:

```
$ python manage.py makemigrations
```

- Aplicar la especificación de la Base de Datos:

```
$ python manage.py migrate
```

- Crear un usuario administrador:

```
$ python manage.py createsuperuser
```


5.6 Pruebas Realizadas

5.6.1 Pruebas unitarias

El TDD o desarrollo guiado por pruebas permite escribir pruebas unitarias antes de desarrollar la aplicación, se codificaron 58 pruebas unitarias, en donde se verifica el funcionamiento de cada vista. Las pruebas plantean escenarios donde se pueden crear recursos por medio de las llamadas REST. Las pruebas permiten verificar que cambios futuros sobre cualquier característica de la API no "rompan" el funcionamiento predefinido del sistema, por lo que cualquier cambio requiere que se modifiquen las pruebas unitarias necesarias.

Se codificaron entre estas pruebas casos de uso como creación, modificación, listado y eliminación de prácticas, plantillas, ejercicios para una práctica, bases, plantillas por herramienta y ejercicio y entradas de usuarios, además se generó una simulación del envío de prácticas evaluadas de 30 usuarios para generar una calificación.

Para ejecutar las pruebas unitarias del sistema es necesario utilizar el comando:

```
$ python manage.py test
----------------------
Ran 58 tests in 4.064s


OK
```

El mismo indica que se ejecutaron 58 pruebas en un determinado tiempo y que el valor para todas fue “OK” o un resultado exitoso al ejecutarlas.

5.6.2 Pruebas manuales

Para comprobar el correcto funcionamiento se realizó una verificación de todas las vistas utilizando el programa **httpie**, que permite introducir cada URL junto con el método REST a utilizar junto con la información que se debe suministrar.

Para realizar esta verificación se realizó una serie de pasos o acciones sobre un servidor de prueba con una base de datos sin registros con la intención de mostrar el proceso en que un usuario administrador crea un curso de Compiladores y se crean dos usuarios que simulan a un profesor de curso y un alumno. El ejemplo demuestra el ciclo completo en el que un administrador crea un curso y un profesor crea una plantilla del curso junto con sus actividades, crea una práctica y asocia los ejercicios correspondientes a la misma y establece las respuestas validas para los ejercicios. En el ejemplo un estudiante da respuesta a los ejercicios de la práctica planteada para luego enviarla y obtener una calificación, el profesor observa la calificación del estudiante y le da una puntuación adicional.

A continuación se muestran todos los pasos a ser realizado usando únicamente la API con el programa **httpie** junto con la respuesta **HTTP** de cada comando. Se creó a un usuario p1 que simula al profesor de un curso creando una práctica de un curso de Compiladores y a un usuario s1 que simula a un estudiante que resuelve la práctica.

1. Crear al usuario p1

```
$ http POST localhost:8000/API/user/ username=p1 password=p1234
   ↪ first_name='p1' last_name='p1' email='p1@p1.com'
```

```
HTTP/1.0 201 Created
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

2. Crear usuario s1

```
$ http POST localhost:8000/API/user/ username=s1 password=s1234
   ↪ first_name='s1' last_name='s1' email='s1@s1.com'
```

```
HTTP/1.0 201 Created
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```


3. Acciones previas del administrador

   (a) Autenticar usuario administrador (obtener llave o token)

```
$ http POST localhost:8000/API/auth-user/ username='root'
   ↪ password='rootpassword'
```

```
HTTP/1.0 200 OK
Allow: POST, OPTIONS
Content-Length: 52
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
X-Frame-Options: SAMEORIGIN
{
```

```
"token": "dd86ea54a1e07d6d43aa77624cf0f11888a161c4"
}
```

(b) Listar cursos

```
$ http GET localhost:8000/API/admin/course/list/ '
    ↪ Authorization: token
    ↪ dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```

```
HTTP/1.0 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```


(c) Crear curso de Compiladores

```
$ http POST localhost:8000/API/admin/course/ 'Authorization:
    ↪ token dd86ea54a1e07d6d43aa77624cf0f11888a161c4' code=1
    ↪ name='Compiladores' description='Curso de Compiladores'
```

```
HTTP/1.0 201 Created
Allow: POST, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(d) Listar cursos

```
$ http GET localhost:8000/API/admin/course/list/ '
    ↪ Authorization: token
    ↪ dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 72
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
        "code": 1,
        "description": "Curso␣de␣compiladores",
        "name": "Compiladores"
        }
]
```


(e) Listar instancias del curso de Compiladores

```
$ http GET localhost:8000/API/admin/course/1/instances/ '
    ↪ Authorization: token
    ↪ dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```

```
HTTP/1.0 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(f) Listar usuarios del *framework*

```
$ http GET localhost:8000/API/admin/user/list/ 'Authorization:
    ↪  token dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 713
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
        "date_joined": "2017-07-21T21:10:45.234638Z",
        "email": "root@root.com",
        "first_name": "",
        "last_login": "2017-07-21T23:59:52.017897Z",
        "last_name": "",
        "password": "
           ↪ pbkdf2_sha256$36000$lSBGad3loMdR$ZwS8W2Oict4wTMufNua
           ↪ +HN/ZI+tBIgNSdqUzlTlpGa4=",
        "pk": 1,
        "username": "root"
        },
        {
        "date_joined": "2017-07-21T21:30:20Z",
        "email": "p1@p1.com",
        "first_name": "p1",
        "last_login": null,
        "last_name": "p1",
        "password": "
           ↪ pbkdf2_sha256$36000$xkKmMqYKMmBP$7oOoV5gOVMDWdk/
           ↪ LNrGBiMOCsCbAeJW1//YlfhICbkk=",
        "pk": 2,
```

```
        "username": "p1"
        },
        {
        "date_joined": "2017-07-21T21:31:09Z",
        "email": "s1@s1.com",
        "first_name": "s1",
        "last_login": null,
        "last_name": "s1",
        "password": "pbkdf2_sha256$36000$8lXvl2WOFvz7$vHEb82F+
            ↪ KMfvVno6wLCSViFxGlln2RJwN4GNTjbcMeU=",
        "pk": 3,
        "username": "s1"
        }
]
```

(g) Crear instancia del curso de compiladores para el usuario p1

```
$ http POST localhost:8000/API/admin/instance/1/ '
    ↪ Authorization: token
    ↪ dd86ea54a1e07d6d43aa77624cf0f11888a161c4' begin_date
    ↪ ='2017-1-1T0:0:0.0Z' end_date='2018-1-1T0:0:0.0Z'
    ↪ creator=2 notes='instancia de prueba' name='Compiladores
    ↪  2017-2018'
```

```
HTTP/1.0 200 OK
Allow: GET, POST, PUT, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(h) Listar instancias del curso de Compiladores

```
$ http GET localhost:8000/API/admin/course/1/instances/ '
    ↪ Authorization: token
    ↪ dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 165
Content-Type: application/json
Date: Fri, 21 Jul 2017 22:24:22 GMT
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
        "begin_date": "2017-01-01T00:00:00Z",
        "course": 1,
        "creator": 2,
        "end_date": "2018-01-01T00:00:00Z",
        "name": "Compiladores␣2017-2018",
        "notes": "instancia␣de␣prueba",
        "pk": 1
        }
]
```


(i) Listar bases (plantillas para ejercicios)

```
$ http GET localhost:8000/API/admin/backbone/ 'Authorization:
    ↪ token dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```

```
HTTP/1.0 404 Not Found
Allow: GET, POST, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
```

```
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(j) Crear base “Plantilla Vacia”

```
$ http POST localhost:8000/API/admin/backbone/ 'Authorization:
    ↪  token dd86ea54a1e07d6d43aa77624cf0f11888a161c4' name='
    ↪ Plantilla Vacia' content=''
```

```
HTTP/1.0 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(k) Listar Bases


```
$ http GET localhost:8000/API/admin/backbone/ 'Authorization:
    ↪ token dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```

```
HTTP/1.0 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Length: 40
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN

[
        {
                "base": null,
                "name": "Plantilla␣Vacia"
        }
```

```
]
```

(l) Listar Bases

```
$ http GET localhost:8000/API/admin/tool/ 'Authorization:
    ↪ token dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```

```
HTTP/1.0 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Length: 289
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "description": null,
                "name": "gcc",
                "pk": 1,
                "status": true,
                "type": "CODE"
        },
        {
                "description": null,
                "name": "flex",
                "pk": 2,
                "status": true,
                "type": "CODE"
        },
        {
                "description": null,
                "name": "flex_bison",
                "pk": 3,
```

```
                "status": true,
                "type": "CODE"
        },
        {
                "description": null,
                "name": "python3",
                "pk": 4,
                "status": true,
                "type": "CODE"
        }
]
```

(m) Listar Bases por Heramientas

```
$ http GET localhost:8000/API/admin/backbone-tool/ '
    ↪ Authorization: token
    ↪ dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```



```
HTTP/1.0 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(n) Crear Base para Herramienta

```
$ http POST localhost:8000/API/admin/backbone-tool/backbone/1/
    ↪ tool/2/ 'Authorization: token
    ↪ dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```

```
HTTP/1.0 201 Created
Allow: POST, DELETE, OPTIONS
Content-Length: 0
```

```
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(o) Listar Bases por Heramientas

```
$ http GET localhost:8000/API/admin/backbone-tool/ '
    ↪ Authorization: token
    ↪ dd86ea54a1e07d6d43aa77624cf0f11888a161c4'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 32
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "Tool": 2,
                "backbone": 1,
                "pk": 1
        }
]
```


4. Acciones del profesor: crear el modelo de una práctica

(a) autenticar usuario p1

```
$ http POST localhost:8000/API/auth-user/ username='p1'
    ↪ password='p1234'
```

```
HTTP/1.0 200 OK
Allow: POST, OPTIONS
```

```
Content-Length: 52
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
X-Frame-Options: SAMEORIGIN
{
        "token": "2f532e6d8322e9ce92c6b053d4cfbb9d66cec758"
}
```

(b) Verificando información personal

```
$ http GET localhost:8000/API/user/ 'Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 225
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
{
        "date_joined": "2017-07-21T21:30:20Z",
        "email": "p1@p1.com",
        "first_name": "p1",
        "last_login": null,
        "last_name": "p1",
        "password": "
           ↪ pbkdf2_sha256$36000$xkKmMqYKMmBP$7oOoV5gOVMDWdk/
           ↪ LNrGBiMOCsCbAeJW1//YlfhICbkk=",
        "pk": 2,
        "username": "p1"
}
```

(c) Verificando listado de cursos de usuario p1 con su respectivo rol

```
$ http GET localhost:8000/API/user-rol/user-course-list/ '
   ↪ Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 55
Content-Type: application/json
Date: Sat, 22 Jul 2017 00:06:49 GMT
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "course_instance": 1,
                "framework_user": 2,
                "user_rol": 1
        }
]
```


(d) Verificando instancia

```
$ http GET localhost:8000/API/course-instance/1/ '
   ↪ Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, POST, PUT, HEAD, OPTIONS
Content-Length: 163
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
```

```
X-Frame-Options: SAMEORIGIN
{
        "begin_date": "2017-01-01T00:00:00Z",
        "course": 1,
        "creator": 2,
        "end_date": "2018-01-01T00:00:00Z",
        "name": "Compiladores␣2017-2018",
        "notes": "instancia␣de␣prueba",
        "pk": 1
}
```

(e) Verificar información del curso

```
$ http GET localhost:8000/API/course/1/ 'Authorization: token
    ↪ 2f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 70
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
{
        "code": 1,
        "description": "Curso␣de␣compiladores",
        "name": "Compiladores"
}
```

(f) Listar Plantillas del curso

```
$ http GET localhost:8000/API/template/1/list/ 'Authorization:
    ↪ token 2f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(g) Crear Plantilla de curso

```
$ http POST localhost:8000/API/template/1/ 'Authorization:
   ↪ token 2f532e6d8322e9ce92c6b053d4cfbb9d66cec758' level=5
   ↪ name='Construcción de un analizador léxico' objective='
   ↪ Construir un analizador léxico usando un generador de
   ↪ Analizadores Lexicos.' description='El alumno deberá
   ↪ especificar el diseño para un scanner, escribiendo las
   ↪ distintas expresiones regulares que conforman un
   ↪ lenguaje a estudiar. Deberá hacer uso de los recursos
   ↪ adquiridos como expresiones regulares y autómatas
   ↪ finitos y verificar el funcionamiento de dicho
   ↪ analizador léxico previamente creado.' required_skills='
   ↪ Expresiones regulares, autómatas finitos, conversión de
   ↪ expresiones regulares a AFND, conversión de AFND a AFD,
   ↪ implementación de AF.' expected_skills='Se espera que el
   ↪  estudiante pueda crear analizadores léxicos a partir de
   ↪  un lenguaje regular propuesto.'
```

```
HTTP/1.0 201 Created
Allow: GET, POST, PUT, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(h) Listar Plantillas del curso

```
$ http GET localhost:8000/API/template/1/list/ 'Authorization:
    ↪ token 2f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 842
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN

[
        {
                "course": 1,
                "description": "El␣alumno␣deberá␣especificar␣el␣
                    ↪ diseño␣para␣un␣scanner,␣escribiendo␣las␣
                    ↪ distintas␣expresiones␣regulares␣que␣
                    ↪ conforman␣un␣lenguaje␣a␣estudiar.␣Deberá␣
                    ↪ hacer␣uso␣de␣los␣recursos␣adquiridos␣como
                    ↪ ␣expresiones␣regulares␣y␣autómatas␣
                    ↪ finitos␣y␣verificar␣el␣funcionamiento␣de␣
                    ↪ dicho␣analizador␣léxico␣previamente␣
                    ↪ creado.",
                "expected_skills": "Se␣espera␣que␣el␣estudiante␣
                    ↪ pueda␣crear␣analizadores␣léxicos␣a␣partir
                    ↪ ␣de␣un␣lenguaje␣regular␣propuesto.",
                "last_edited": "2017-07-22T00:13:34.724383Z",
                "level": 5,
                "name": "Construcción␣de␣un␣analizador␣léxico",
```

```
            "objective": "Construir␣un␣analizador␣léxico␣
                ↪ usando␣un␣generador␣de␣Analizadores␣
                ↪ Lexicos.",
            "required_skills": "Expresiones␣regulares,␣
                ↪ autómatas␣finitos,␣conversión␣de␣
                ↪ expresiones␣regulares␣a␣AFND,␣conversión␣
                ↪ de␣AFND␣a␣AFD,␣implementación␣de␣AF.",
            "user_last_edited": 2
        }
]
```

(i) Listar actividades de la Plantilla

```
$ http GET localhost:8000/API/activity/1/list/ 'Authorization:
    ↪  token 2f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(j) Crear actividad de la Plantilla

```
$ http POST localhost:8000/API/activity/1/ 'Authorization:
    ↪ token 2f532e6d8322e9ce92c6b053d4cfbb9d66cec758' content
    ↪ ='Asociar el conteo de lineas con expresiones regulares
    ↪ como parte fundamental del funcionamiento de un
    ↪ analizador léxico.'
```

```
HTTP/1.0 201 Created
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
```

```
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(k) Listar actividades de la Plantilla

```
$ http GET localhost:8000/API/activity/1/list/ 'Authorization:
    ↪ token 2f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 157
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "content": "Asociar␣el␣conteo␣de␣lineas␣con␣
                    ↪ expresiones␣regulares␣como␣parte␣
                    ↪ fundamental␣del␣funcionamiento␣de␣un␣
                    ↪ analizador␣léxico.",
                "pk": 1,
                "template": 1
        }
]
```


(l) Listar modelos de ejercicio

```
$ http GET localhost:8000/API/model-exercise/1/list/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 404 Not Found
```

```
Allow: GET, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(m) Crear modelo de ejercicio

```
$ http POST localhost:8000/API/model-exercise/1/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758' content="
    ↪ Construya␣un␣analizador␣lexico␣que␣permita␣contar␣las␣
    ↪ lineas␣del␣texto␣anexo"
```

```
HTTP/1.0 201 Created
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```


(n) Listar modelos de ejercicio

```
$ http GET localhost:8000/API/model-exercise/1/list/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 112
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

```
[
        {
                "activity": 1,
                "content": "Construya␣un␣analizador␣lexico␣que␣
                    ↪ permita␣contar␣las␣lineas␣del␣texto␣anexo
                    ↪ ",
                "pk": 1
        }
]
```

(o) Listar respuestas del modelo de ejercicio

```
$ http GET localhost:8000/API/expected-answer/1/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 404 Not Found
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```


(p) Crear respuesta

```
$ http POST localhost:8000/API/expected-answer/1/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758' answer=5
```

```
HTTP/1.0 201 Created
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
```

```
X-Frame-Options: SAMEORIGIN
```

(q) Listar respuestas del modelo de ejercicio

```
$ http GET localhost:8000/API/expected-answer/1/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 36
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "answer": "5",
                "exercise": 1,
                "pk": 1
        }
]
```


(r) Listar modelos de práctica

```
$ http GET localhost:8000/API/model-practice/1/ 'Authorization
    ↪ : token 2f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 404 Not Found
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(s) Crear un modelo de práctica para el curso de Compiladores

```
$ http POST localhost:8000/API/model-practice/1/ '
   ↪ Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758' name='
   ↪ analizadores lexicos' description='practica de
   ↪ analizadores lexicos'
```

```
HTTP/1.0 201 Created
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(t) Listar modelos de prácticas


```
$ http GET localhost:8000/API/model-practice/1/ 'Authorization
   ↪ : token 2f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 165
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "course": 1,
                "description": "practica␣de␣analizadores␣lexicos
                   ↪ ",
                "last_edited": "2017-07-22T02:05:27.072288Z",
                "name": "analizadores␣lexicos",
```

```
                "pk": 1,
                "user_last_edited": 2
        }
]
```

(u) Listar actividades del modelo de práctica

```
$ http GET localhost:8000/API/model-practice/1/activity/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```


(v) Crear actividad para el modelo de practica

```
$ http POST localhost:8000/API/model-practice/practice/1/
    ↪ activity/1/ 'Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 201 Created
Allow: POST, DELETE, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(w) Listar actividades del modelo de práctica

```
$ http GET localhost:8000/API/model-practice/1/activity/ '
   ↪ Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 36
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
       {
              "activity": 1,
              "pk": 1,
              "practice": 1
       }
]
```


5. Acciones del profesor: crear una práctica a partir de modelos

   (a) Listar prácticas de la instancia

```
$ http GET localhost:8000/API/practice/instance/1/ '
   ↪ Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(b) Crear práctica

```
$ http POST localhost:8000/API/practice/model/1/instance/1/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758' start_date
    ↪ ='2017-07-16T21:44:10.059492Z' end_date='2017-12-16T21
    ↪ :44:10.059492Z'
```

```
HTTP/1.0 201 Created
Allow: POST, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(c) Listar prácticas de la instancia

```
$ http GET localhost:8000/API/practice/instance/1/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 129
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "course_instance": 1,
                "end_date": "2017-12-16T21:44:10Z",
                "practice": 1,
                "retries": 0,
```

```
                "score": 10,
                "start_date": "2017-07-16T21:44:10Z"
        }
]
```

(d) Listar ejercicios de la práctica

```
$ http GET localhost:8000/API/practice-exercise/1/list/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```


(e) Listar Herramientas disponibles

```
$ http GET localhost:8000/API/tools/ 'Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 289
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
        "description": null,
        "name": "gcc",
```

```
        "pk": 1,
        "status": true,
        "type": "CODE"
        },
        {
        "description": null,
        "name": "flex",
        "pk": 2,
        "status": true,
        "type": "CODE"
        },
        {
        "description": null,
        "name": "flex_bison",
        "pk": 3,
        "status": true,
        "type": "CODE"
        },
        {
        "description": null,
        "name": "python3",
        "pk": 4,
        "status": true,
        "type": "CODE"
        }
]
```


(f) Crear ejercicio para la práctica

```
$ http POST localhost:8000/API/practice-exercise/practice/1/
   ↪ exercise/1/ 'Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758' name='Conteo de
   ↪  lineas' tool=2 score=30 fixed=False
```

```
HTTP/1.0 201 Created
Allow: POST, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(g) Buscar Base de Herramienta por nombre

```
$ http GET localhost:8000/API/backbone-tool/flex/ '
   ↪ Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```



```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 32
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "Tool": 2,
                "backbone": 1,
                "pk": 1
        }
]
```

(h) Identificar Base

```
$ http GET localhost:8000/API/backbone/1/ 'Authorization:
    ↪ token 2f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 36
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
{
        "base": "",
        "name": "Plantilla␣Vacia"
}
```


(i) Listar Bases para el ejercicio

```
$ http GET localhost:8000/API/backbone-exercise/1/list/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(j) Crear Base para el ejercicio

```
$ http POST localhost:8000/API/backbone-exercise/ '
   ↪ Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758' backbone=1
   ↪ exercise=1 file_name='input.txt' text='''linea1
linea2
linea3
linea4
linea5
'''
```

```
HTTP/1.0 201 Created
Allow: POST, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```


(k) Listar Bases para el ejercicio

```
$ http GET localhost:8000/API/backbone-exercise/1/list/ '
   ↪ Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 108
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "backbone": 1,
```

```
                "exercise": 1,
                "file_name": "input.txt",
                "pk": 1,
                "text": "linea1\nlinea2\nlinea3\nlinea4\nlinea5"
        }
]
```

6. Acciones del estudiante: Resolver una práctica

   (a) Autenticar estudiante modelo

```
$ http POST localhost:8000/API/auth-user/ username='s1'
   ↪ password='s1234'
```

```
HTTP/1.0 200 OK
Allow: POST, OPTIONS
Content-Length: 52
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
X-Frame-Options: SAMEORIGIN
{
        "token": "3275e5698dc57ff40ecaf569306b9a85408f3caa"
}
```

   (b) Incorporar al usuario a la instancia del curso

```
$ http POST localhost:8000/API/user-rol/instance/1/user/ '
   ↪ Authorization: token 3275
   ↪ e5698dc57ff40ecaf569306b9a85408f3caa'
```

```
HTTP/1.0 201 Created
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
```

```
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(c) Listar prácticas disponibles

```
$ http GET localhost:8000/API/practice/instance/1/ '
    ↪ Authorization: token 3275
    ↪ e5698dc57ff40ecaf569306b9a85408f3caa'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 129
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "course_instance": 1,
                "end_date": "2017-12-16T21:44:10Z",
                "practice": 1,
                "retries": 0,
                "score": 10,
                "start_date": "2017-07-16T21:44:10Z"
        }
]
```


(d) Listar ejercicios de la práctica

```
$ http GET localhost:8000/API/practice-exercise/1/list/ '
    ↪ Authorization: token 3275
    ↪ e5698dc57ff40ecaf569306b9a85408f3caa'
```

```
HTTP/1.0 200 OK
```

```
Allow: GET, HEAD, OPTIONS
Content-Length: 89
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
{
        "exercise": 1,
        "fixed": false,
        "name": "Conteo␣de␣lineas",
        "practice": 1,
        "score": 30,
        "tool": 2
}
]
```


(e) Obtener enunciado del ejercicio y detalles

```
$ http GET localhost:8000/API/model-exercise/model/1/practice
   ↪ /1/ 'Authorization: token 3275
   ↪ e5698dc57ff40ecaf569306b9a85408f3caa'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 110
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN

{
"activity": 1,
```

```
"content": "Construya␣un␣analizador␣lexico␣que␣permita␣contar␣
    ↪ las␣lineas␣del␣texto␣anexo",
"pk": 1
}
```

(f) Obtener Bases disponibles para el ejercicio

```
$ http GET localhost:8000/API/backbone-exercise/1/list/ '
    ↪ Authorization: token 3275
    ↪ e5698dc57ff40ecaf569306b9a85408f3caa'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 108
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
        "backbone": 1,
        "exercise": 1,
        "file_name": "input.txt",
        "pk": 1,
        "text": "linea1\nlinea2\nlinea3\nlinea4\nlinea5"
        }
]
```


(g) Ingreso de código

```
$ http POST localhost:8000/API/user-exercise/input/1/ '
    ↪ Authorization: token 3275
    ↪ e5698dc57ff40ecaf569306b9a85408f3caa' file_name='scanner
    ↪ .l' content="""%{
```

```
#include<stdio.h>
int␣lineas=1;
%}

%%

\n␣{lineas++;}

.

%%

int␣main()
{
FILE␣*f=fopen(\"input.txt\",\"r\");
yyin=f;
yylex();
fclose(f);
printf(\"%d\",lineas);
return␣0;
}
"""
```



```
HTTP/1.0 201 Created
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(h) Ingreso de código

```
$ http POST localhost:8000/API/user-exercise/input/1/ '
   ↪ Authorization: token 3275
   ↪ e5698dc57ff40ecaf569306b9a85408f3caa' file_name='input.
   ↪ txt' content="""1
2
3
4
5
"""
```

```
HTTP/1.0 201 Created
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```


(i) Envió de la práctica evaluada

```
$ http POST localhost:8000/API/submit-practice/1/ '
   ↪ Authorization: token 3275
   ↪ e5698dc57ff40ecaf569306b9a85408f3caa'
```

```
HTTP/1.0 202 Accepted
Allow: POST, OPTIONS
Content-Length: 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(j) Obtener calificación automática

```
$ http GET localhost:8000/API/practice-grade/1/ 'Authorization
   ↪ : token 3275e5698dc57ff40ecaf569306b9a85408f3caa'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 50
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
{
        "gradeTime": "2017-07-22T03:21:02.200175Z",
        "pk": 1
}
```

(k) Listar calificaciones por ejercicio

```
$ http GET localhost:8000/API/exercise-grade/1/ 'Authorization
↪: token 3275e5698dc57ff40ecaf569306b9a85408f3caa'
```



```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 81
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
                "exercise": 1,
                "grade": 30,
                "grade_feedback": 0,
                "pk": 1,
                "practice_grade": 1,
                "user": 3
```

```
        }
]
```

7. Acciones del profesor: Revisión de práctica de usuario

   (a) Verificar calificación de usuario estudiante

```
$ http GET localhost:8000/API/practice-grade/1/username/s1/ '
   ↪ Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 52
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
        "gradeTime": "2017-07-22T03:21:02.200175Z",
        "pk": 1
        }
]
```


   (b) Verificar calificación por ejercicio

```
$ http GET localhost:8000/API/exercise-revision-grade/1/ '
   ↪ Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, PUT, HEAD, OPTIONS
Content-Length: 81
```

```
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN

[
        {
        "exercise": 1,
        "grade": 30,
        "grade_feedback": 0,
        "pk": 1,
        "practice_grade": 1,
        "user": 3
        }
]
```


(c) Verificar entradas del usuario para un ejercicio en particular

```
$ http GET localhost:8000/API/user-exercise/user/s1/exercise
   ↪ /1/ 'Authorization: token 2
   ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, HEAD, OPTIONS
Content-Length: 431
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN

[
{
"content": "1\n2\n3\n4\n5",
```

```
"exercise": 1,
"file_name": "input.txt",
"timestamp": "2017-07-22T03:19:12.437479Z",
"user": 3
},
{
"content": "%{\n#include<stdio.h>\n␣int␣lineas=1;\n%}\n␣\n%%\n
    ↪ ␣\n\\n␣{lineas++;}\n␣\n.\n␣\n%%\n␣\nint␣main()\n{\n␣FILE
    ↪ ␣*f=fopen(\"input.txt\",\"r\");\n␣yyin=f;\n␣yylex();\n␣
    ↪ fclose(f);\n␣printf(\"%d\",lineas);\n␣return␣0;\n}",
"exercise": 1,
"file_name": "scanner.l",
"timestamp": "2017-07-22T03:20:24.926179Z",
"user": 3
}
]
```


(d) Modificar la calificación personal al ejercicio

```
$ http PUT localhost:8000/API/exercise-revision-grade/1/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758' grade_feedback
    ↪ =-1
```

```
HTTP/1.0 200 OK
Allow: GET, PUT, HEAD, OPTIONS
Content-Length:+ 0
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
```

(e) Listando calificaciones de un ejercicio para un estudiante

```
$ http GET localhost:8000/API/exercise-revision-grade/1/ '
    ↪ Authorization: token 2
    ↪ f532e6d8322e9ce92c6b053d4cfbb9d66cec758'
```

```
HTTP/1.0 200 OK
Allow: GET, PUT, HEAD, OPTIONS
Content-Length: 82
Content-Type: application/json
Server: WSGIServer/0.2 CPython/3.6.1
Vary: Accept
X-Frame-Options: SAMEORIGIN
[
        {
        "exercise": 1,
        "grade": 30,
        "grade_feedback": -1,
        "pk": 1,
        "practice_grade": 1,
        "user": 3
        }
]
```

Capítulo 6

Conclusiones y Recomendaciones

6.1 Conclusiones

Se creó un *framework* que permite integrar por medio de plantillas el contenido teórico con las prácticas de forma apropiada. Permite a un docente construir e implementar prácticas de forma guiada y a su vez con un nivel de flexibilidad que le permite cambiar su estructura interna con facilidad.

Este *framework* permite adaptar el esquema de enseñanza con solo modificar las "Plantillas" del curso. Estas plantillas que contienen información explícita sobre cómo realizar una práctica indican todos los pasos necesarios para complementar el contenido teórico y se le permite al docente escoger las actividades a realizar de una plantilla dada. Esto implica que una práctica propuesta puede estar construida por actividades de distintas plantillas dependiendo de la estructura del curso.

La creación del *framework* para prácticas de laboratorio no solo incluyó la creación de dichas prácticas sino que adicionalmente brindó una plataforma base para que alumnos de un curso puedan resolverlas a través de la provisión herramientas previamente seleccionadas y apropiadas que permitan darle una relación al contenido teórico con el practico.

6.2 Recomendaciones

- Modificar las plantillas acorde a las necesidades del curso, para evitar que las mismas contengan con el paso del tiempo contenido obsoleto.
- Modificar las Actividades de cada plantilla según sea necesario para complementar el contenido que ofrece la misma.
- Una interfaz de usuario el cual puede ser adaptada según sea necesario con las características que se deseen.
- Utilizar ambientes virtuales aislados pues la utilización de las herramientas no consideran temas como la seguridad al momento de compilar y ejecutar código.
- Utilizar un gestor de Bases de Datos cuyas prestaciones se adapten a las necesidades requeridas.

Bibliografía

[1] Flex - the fast lexical analyzer. `https://www.gnu.org/software/flex/`. Revisado: 2017-6-17.

[2] Gnu bison. `https://www.gnu.org/software/bison/`. Revisado: 2017-6-17.

[3] *Manual de Procedimiento Curriculares*.

[4] Alfred Aho, Monica Lam, Ravi Sethi, and Jeffrey Ullman. *Compiladores principios,técnicas y herramientas*. Pearson Educación, 2008.

[5] Alex Aiken. Compilers. `https://lagunita.stanford.edu/courses/Engineering/Compilers/Fall2014/about`, November 2014. Revisado: 2016-11-26.

[6] Manuel Alfonseca, Marina De la cruz, Alfonso Ortega, and Estrella Pulido. *Compiladores e intérpretes: teoría y práctica*. Pearson Educación, 2006.

[7] Richard Bornat. *Understanding and Writing Compilers*. Richard Bornat, 2008.

[8] Saumya Debray. Making compilers design relevant for students who will (most likely) never design a compiler. Technical report, University of Arizona, Department of Computer Science, 03 2002.

[9] Josuka Diaz and Jose Sáenz. Del proposito de la materia de compiladores en la formación del ingeniero informático. Technical report, Universidad de Deusto, 01 2003.

[10] M. Hanspeter. Compiler construction the art of niklaus wirth. Technical report, University of Linz, 2000.

[11] Sergio Luján Mora. Programación en internet: clientes web, 2001-10-08.

[12] Francisco Moreno. Diseño de compiladores. `http://www.uhu.es/francisco.moreno/gii_dc/`, November 2016. Revisado: 2016-11-26.

[13] Carlos Ca nedo. Caracterización de la práctica de laboratorio. `http://www.eumed.net/libros-gratis/2008b/395/CARACTERIZACION%20DE%20LA%20PRACTICA%20DE%20LABORATORIO.htm`, April 2008. Revisado: 2017-4-12.

[14] Ken Pugh. *Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration*. Net Objectives, 2011.

[15] Jaime Urquiza, Francisco Almeida, and Antonio Pérez. Reorganización de las prácticas de compiladores para mejorar el aprendizaje de los estudiantes. Technical report, Universidad Rey Juan Carlos, Departamento de Lenguajes y Sistemas Informáticos I, 07 2010.

[16] Salvador Valerio. Compiladores e intérpretes, en búsqueda de una práctica docente eficaz. Technical report, Universidad Católica de Cuyo, Facultad de Filosofía y Humanidades, 07 2005.