UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

FACULTAD CIENCIAS DE LA INGENIERÍA

CARRERA DE INGENIERÍA EN SISTEMAS

Proyecto de Investigación previo a
la obtención del título de Ingeniero
en Sistemas.

Título del Proyecto de Investigación:

"LIBRERÍA JAVASCRIPT QUE PERMITA INTERPRETAR LA ESCRITURA DE LOS CASOS DE USO EXTENDIDOS"

Autor:

Dúval Ricardo Carvajal Suárez

Director del Proyecto de Investigación:

Ing. Gleiston Cicerón Guerrero Ulloa, MSD.

QUEVEDO - LOS RÍOS - ECUADOR

2022

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

Yo, **Dúval Ricardo Carvajal Suárez** declaró que la investigación aquí descrita es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

La Universidad Técnica Estatal de Quevedo, puede hacer uso de los derechos correspondientes a este documento, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Firmado electrónicamente por:
DUVAL RICARDO
CARVAJAL SUAREZ

Dúval Ricardo Carvajal Suárez

070638894-9

CERTIFICACIÓN DE CULMINACIÓN DEL PROYECTO DE INVESTIGACIÓN

El suscrito, Ing. Gleiston Cicerón Guerrero Ulloa Msc. docente de la Universidad Técnica Estatal de Quevedo, certifica que el estudiante Dúval Ricardo Carvajal Suárez, realizó el Proyecto de Investigación de grado titulado "LIBRERÍA JAVASCRIPT QUE PERMITA INTERPRETAR LA ESCRITURA DE LOS CASOS DE USO EXTENDIDOS", previo a la obtención del título de Ingeniero en Sistemas, bajo mi dirección, habiendo cumplido con las disposiciones reglamentarias establecidas para el efecto.

GLEISTON CICERON GUERRERO ULLOA

Firmado digitalmente por GLEISTON CICERON GUERRERO ULLOA
Fecha: 2022.11.28 09:24:55 -05'00'

Ing. Gleiston Cicerón Guerrero Ulloa MDS.

DIRECTOR DEL PROYECTO DE INVESTIGACIÓN

CERTIFICADO DEL REPORTE DE LA HERRAMIENTA DE PREVENCIÓN DE COINCIDENCIA Y/O PLAGIO ACADÉMICO

El suscrito Ing. Gleiston Ciceron Guerrero Ulloa Msc. certifica que:

El proyecto de investigación titulado ""“LIBRERÍA JAVASCRIPT QUE PERMITA INTERPRETAR LA ESCRITURA DE LOS CASOS DE USO EXTENDIDOS", ha sido analizado mediante la herramienta de URKUND y presentó resultados satisfactorios.

Ouriginal by Turnitin

Document Information

Analyzed document	Proyecto de titulacion - Carvajal Dúval.pdf (D144821857)
Submitted	9/26/2022 6:59:00 AM
Submitted by	
Submitter email	duval.carvajal2017@uteq.edu.ec
Similarity	1%
Analysis address	gguerrero.uteq@analysis.urkund.com

GLEISTON CICERON GUERRERO ULLOA

Firmado digitalmente por GLEISTON CICERON GUERRERO ULLOA
Fecha: 2022.11.22 16:21:27 -05'00'

Ing. Gleiston Ciceron Guerrero Ulloa Msc.

DIRECTOR DEL PROYECTO DE INVESTIGACIÓN

CERTIFICACIÓN DE APROBACIÓN DEL PROYECTO DE INVESTIGACIÓN

Titulo:

"LIBRERIA JAVASCRIPT QUE PERMITA INTERPRETAR LA ESCRITURA DE LOS CASOS DE USO EXTENDIDOS"

Presentado al Consejo Directivo como requisito previo a la obtención del título de Ingeniero en Sistemas.

Aprobado por:

Firmado electrónicamente por:
EFRAIN EVARISTO DIAZ MACIAS

PRESIDENTE DEL TRIBUNAL

Ing. Efraín Evaristo Díaz Macías, MSIG.

Firmado electrónicamente por:
CRISTIAN GABRIEL ZAMBRANO VEGA

MIEMBRO DEL TRIBUNAL

Ing. Cristian Gabriel Zambrano Vega, PhD.

ARIOSTO EUGENIO VICUNA PINO

Firmado digitalmente por ARIOSTO EUGENIO VICUNA PINO
DN: CN=ARIOSTO EUGENIO VICUNA PINO, SERIALNUMBER=010421125819, OU=ENTIDAD DE CERTIFICACION DE INFORMACION, O=SECURITY DATA S.A. 2, C=EC
Razón: Estoy aprobando este documento
Ubicación: la ubicación de su firma aquí
Fecha: 2022.11.23 19:01:41 -05:00
Adobe Reader version: 11.0.23

MIEMBRO DEL TRIBUNAL

Ing. Ariosto Eugenio Vicuña Pino, MSC.

QUEVEDO - LOS RÍOS - ECUADOR

2022

AGRADECIMIENTO

Agradezco en especial a Jehová Dios por a verme dado la fuerzas e inteligencia necesaria para culminar este ciclo en mi vida.

Un agradecimiento especial para mi familia, porque todos aportaron con un granito de arena para seguir adelante con mis estudios sin importar la distancia a la que me encontraba.

A mis compañeros y amigos que fueron las personas con las cuales conviví la mayor parte del tiempo de mis estudios, en especial a los Señores Geovanny Brito Casanova y Anthony Pachay Espinoza, quienes compartir momentos buenos y malos, pero al fin y al cabo son momentos que jamás se podrán olvidar.

Agradecer también a la Universidad Técnica Estatal de Quevedo quien se convirtió en mi segundo hogar, pasando días completos dentro de ella compartiendo con mis compañeros y sintiéndome como si estuviera en casa.

Agradecer a todos los docentes de la Universidad Técnica Estatal de Quevedo por compartir sus conocimientos profesionales para ser de mí, una persona respetable y de buen corazón sin importar las circunstancias en las que me encuentre.

Un agradecimiento especial a los Ingenieros Ariosto Vicuña Pino y Gleiston Guerrero Ulloa por a ver compartido su tiempo conmigo no solo como docente si no como amigos, brindándome consejos a nivel profesional y moral.

Dúval Ricardo Carvajal Suárez

DEDICATORIA

El presente trabajo se lo dedico a toda mi familia que vive en la ciudad de Machala y por brindarme su apoyo desde el momento que me separe presencialmente de ellos y llegue a la ciudad de Quevedo a empezar mis estudios universitarios.

A toda mi familia de Quevedo que me recibió con mucho cariño y aporto con brindarme un lugar donde pueda vivir y culminar mis estudios universitarios.

Dúval Ricardo Carvajal Suárez

RESUMEN Y PALABRAS CLAVES

El modelamiento de software ayuda a visualizar los elementos necesarios para desarrollar un sistema informático. Existen lenguajes de modelado que sirven para realizar esta tarea, sin embargo, algunos desarrolladores comentan que esta fase en el desarrollo de software es muy tediosa, ya que, se debe documentar y validar el software antes de generar el código fuente. Uno de los diagramas más utilizados para visualizar el sistema informático es el diagrama de clases. Este diagrama permite detallar todas las clases de objetos que serán necesarios para que el producto final cumpla con las expectativas del cliente. Para tomar los requisitos del sistema y del software existen otras herramientas, entre ellas los casos de uso y las historias de usuario. Los casos de uso sirven para anotar todo lo que el cliente mencione sobre el funcionamiento del software a realizar mediante una secuencia de acciones. El desarrollador deberá prestar atención a todo lo que el cliente necesite realizar utilizando el sistema. Los casos de uso son los más convenientes para redactar las necesidades del cliente o requisitos del sistema. A partir de todo lo mencionado por el usuario que utilizará el sistema, el desarrollador empieza a reconocer los objetos que se deben generar para la posterior implementación (construcción) del software. Por eso, el objetivo de este trabajo es crear una librería JavaScript denominada Armadillo que permita interpretar los casos de uso escritos con un lenguaje de símbolos que utiliza la herramienta TDDT4IoTS (`https://aplicaciones.uteq.edu.ec/tddt4iots/`) para marcar palabras que representan elementos del modelamiento orientado a objetos y generar la estructura de un diagrama de clases en formato JSON y XML, para que luego pueda ser manipulado utilizando otras librerías web para dibujar diagramas de clases y cualquier otro tipos de diagrama a partir de este.

Palabras claves: Herramienta case, diagrama de clases, especificación de casos de uso, lenguaje para especificación casos de uso, ingeniería de software.

ABSTRACT AND KEYWORDS

Software modeling helps to visualize the elements necessary to develop a computer system. There are modeling languages that are used to perform this task, however, some developers comment that this phase in software development is very tedious, since the software must be documented and validated before generating the source code. One of the most commonly used diagrams to visualize the computer system is the class diagram. This diagram allows to detail all the classes of objects that will be necessary for the final product to meet the customer's expectations. Other tools exist to take the system and software requirements, including use cases and user stories. Use cases are used to write down everything that the customer mentions about the operation of the software to be performed through a sequence of actions. The developer should pay attention to everything the customer needs to perform using the system. The use cases are the most convenient to write down the customer's needs or system requirements. From everything mentioned by the user who will use the system, the developer begins to recognize the objects to be generated for the subsequent implementation (construction) of the software. Therefore, the objective of this work is to create a JavaScript library that allows to interpret the use cases written with a symbol language that uses the TDDT4IoTS tool (`https://aplicaciones.uteq.edu.ec/tddt4iots/`) to mark words that represent elements of object-oriented modeling and generate the structure of a class diagram in JSON and XML format, so that it can then be manipulated using other web libraries to draw class diagrams and any other type of diagram from it.

Keywords: Case tools, class diagrams, use case specification, use case specification language, software engineering.

Tabla de contenido

Lista de tablas

Lista de ilustraciones

CÓDIGO DUBLIN

Titulo:	Librería JavaScript que permita interpretar la escritura de los casos de uso extendidos.				
Autores:	Carvajal Suárez, Dúval Ricardo				
Palabras claves:	Herramienta case	Diagrama de clases	Especificación de casos de uso	SymLen de casos de uso	Ingeniería de software
Fecha de publicación:	Diciembre, 2022				
Director del proyecto:	Guerrero Ulloa, Gleiston Cicerón				
Editorial:	Quevedo: UTEQ 2022				
Resumen:	Resumen - El modelamiento de software ayuda a visualizar los elementos necesarios para desarrollar un sistema informático. Existen lenguajes de modelado que sirven para realizar esta tarea, sin embargo, algunos desarrolladores comentan que esta fase en el desarrollo de software es muy tediosa, ya que, se debe documentar y validar el software antes de generar el código fuente. Uno de los diagramas más utilizados para visualizar el sistema informático es el diagrama de clases. Este diagrama permite detallar todas las clases de objetos que serán necesarios para que el producto final cumpla con las expectativas del cliente. Para tomar los requisitos del sistema y del software existen otras herramientas, entre ellas los casos de uso y las historias de usuario. Los casos de uso sirven para anotar todo lo que el cliente mencione sobre el funcionamiento del software a realizar mediante una secuencia de acciones. El desarrollador deberá prestar atención a todo lo que el cliente necesite realizar utilizando el sistema. Los casos de uso son los más convenientes para redactar las necesidades del cliente o requisitos del sistema. **Abstract - Software modeling helps to visualize the elements necessary to develop a computer system. There are modeling languages that are used to perform this task, however, some developers comment that this phase in software development is very tedious, since the software must be documented and validated before generating the source code. One of the most commonly used diagrams to visualize the computer system is the class diagram. This diagram allows to detail all the classes of objects that will be necessary for the final product to meet the customer's expectations. Other tools exist to take the system and software requirements, including use cases and user stories. Use cases are used to write down everything that the customer mentions about the operation of the software to be performed through a sequence of actions. The developer should pay attention to everything the customer needs to perform using the system. The use cases are the most convenient to write down the customer's needs or system requirements.**				
Descripción:	116 hojas, dimensiones: 21cm x 29.7cm				
URI:					

Introducción

Los sistemas informáticos se están volviendo cada vez más indispensables en todos los aspectos de la vida cotidiana. Considerando que la tecnología cambia aceleradamente, y que los usuarios requieren soluciones rápidas, la Ingeniería de Software busca disminuir el tiempo en implementar los sistemas requeridos [1]. Para la construcción de una aplicación informática que cumpla con los requisitos del cliente y que sea eficiente se necesitan aplicar varios aspectos técnicos, que la Ingeniería de Software detalla [2]. Por ejemplo: estilo arquitectural, tecnologías a utilizar, infraestructura, entre otros [3].

El campo de la Ingeniería de Software es análogo al campo de la construcción de obras civiles, entre otros. Los documentos de diseño de un Software guardan semejanza con los documentos de diseño de una obra civil (por ejemplo, un edificio). Para la construcción de un edificio se deben analizar las principales características físicas del terreno sobre el cual se va a construir, mientras que, para la construcción de un software se debe analizar la/s plataforma/s sobre la/s cual/es se ejecutará el software. En el caso de un edificio, se deben detallar aspectos de la construcción como los ambientes y sus dimensiones precisas, columnas, puertas, ventanas, etc. Así mismo, en los sistemas informáticos se necesita documentar con precisión los módulos, sus funcionalidades y sus alcances, y otros elementos que conformarán el sistema a desarrollar. Además del comportamiento que deben tener los artefactos de un software [4]. Lo que se puede mencionar como diferencias entre estos dos campos es que los elementos de software pueden ser construidos para ser reutilizados en otros sistemas [5].

Unified Modeling Language (UML) con su traducción al español Lenguaje Unificado de Modelado. Es uno de los lenguajes más conocidos por los desarrolladores de software para la representación de los requisitos o necesidades del usuario [6]. Sin embargo, estos diagramas no son fáciles de comprender para todos los usuarios, ya que, los elementos que intervienen y que darán solución al problema planteado son representados mediante símbolos y algo de texto. UML se utiliza específicamente en la industria del software para especificar, visualizar, construir y documentar los artefactos de un sistema de software [7]. UML se encuentra definido oficialmente por el *Object Management Group* (OMG) con su traducción al español Grupo de Administración de Objetos [8]. Algunos investigadores afirman que, a partir de

modelos dinámicos (diagramas de casos de uso) y estáticos (diagramas de clases) se pueden generar otros modelos UML de manera más eficiente [9], y así obtener software de calidad.

Los estándares ISO/IEC 9126-1, especifican la calidad del software tanto interna, externa y en uso del producto. De ahí la importancia de considerar seguir estándares que permitan asegurar un lenguaje común para el entendimiento de todos los involucrados en el proyecto de software [10].

Los diagramas de casos de uso son una representación del sistema mediante los actores (nota al pie) y sus requisitos (casos de uso). Son una herramienta para representar los requisitos iniciales del sistema, y pueden ser fácilmente entendidos por todos los involucrados [11]. Sin embargo, en un diagrama de casos de uso no se especifican los detalles, sino que son los requisitos a *grano grueso* del sistema.

Para mayor detalle de los requisitos se utilizan los casos de uso extendidos, que detallan los requisitos a *grano fino* del sistema, y se plasman en documentos específicos. Estos requisitos de grano fino pueden ser comprimidos en los diagramas de clases [13]. Los casos de uso tienen algunos elementos de información, entre ellos:

- **Descripción:** describen de forma textual varias formas que los actores pueden trabajar con el software.
- **Precondiciones:** son las condiciones que debe cumplir el sistema para qué sé escriban en el caso de uso.
- **Poscondiciones:** es el estado en que el sistema se encuentra después de haber realizado el caso de uso. Se deben considerar al momento de estar utilizando el software [12].
- **Secuencia normal de eventos:** sirve para describir las acciones de los actores y las respuestas del sistema en forma cronológica.
- **Secuencia alternativa de eventos:** describen las respuestas del sistema y las acciones de los actores cuando se dan ciertas condiciones.

Los diagramas de clases en UML representan la estructura estática de los objetos y sus posibles conexiones dentro del software. Se lo utiliza para ilustrar el punto de vista estático,

exponiendo un conjunto de clases, interfaces y relaciones [14]. Se lo desarrolla durante la fase de elaboración y se perfecciona posteriormente en la fase de construcción [8], mostrando un modelo del dominio del sistema. Además, es uno de los diagramas más útiles en UML, ya que trazan claramente la estructura de un sistema concreto a modelar [14].

Los diagramas de clases también son implementados en muchas herramientas de modelado y utilizados en la generación automática de software, entre las que se puede citar a Rational Rose, MagicDraw, ArgoUML, por mencionar algunas. Sin embargo, ninguna considera la documentación de los casos de uso, por lo tanto, tampoco emplean las especificaciones a grano fino para obtener diagramas UML que ayuden a generar el código como lo hace el diagrama de clases.

Existe una herramienta TDDT4IoTS [15] que está en fase de prueba. En esta herramienta es posible que el analista escriba los requisitos detallados de los casos de uso, y, mediante el uso de un lenguaje de símbolos (SymLen) marcar las palabras que representan nombres de clases, atributos, métodos, interfaces y otros elementos de los diagramas de clases. A su vez, presenta los casos de uso en lenguaje natural con las palabras que el usuario utilizó para expresar los requisitos detallados que el sistema debe cumplir. De estas especificaciones de grano fino es posible generar los diagramas de clases, y el código de software correspondiente. Al ser una herramienta en su fase inicial, el intérprete implementado en TDDT4IoTS no presenta una retroalimentación eficiente, simplemente presenta los artefactos que son posible generar según la escritura, sin especificar los posibles errores en el uso del lenguaje. Además, el actual intérprete no se puede utilizar fácilmente en otra herramienta.

En el presente trabajo se presenta una librería escrita en JavaScript denominada *Armadillo* (Armadillo.js), que permite interpretar las especificaciones de los casos de uso escritas en SymLen. Armadillo le permite al analista conocer los posibles errores en el uso de SymLen, con el objetivo de obtener de forma automática un diagrama de clases pertinente a la información proporcionada por el usuario. Esto permitirá potenciar el uso de la herramienta TDDT4IoTS. Además, Armadillo le permite eliminar los inconvenientes del intérprete que actualmente utiliza TDDT4IoTS que han sido detectados.

El resto de este documento está organizado por capítulos. En el capítulo I se contextualiza la investigación, donde se especifica la problemática a resolver, los objetivos que se proponen lograr y la hipótesis a demostrar, además de una argumentación que justifica el porqué de esta investigación. En el capítulo II se presenta la fundamentación teórica, en la que se detallan los principales trabajos relacionados con este documento. Además, el alcance y las limitaciones, las herramientas y tecnologías a utilizar, y los conceptos y definiciones importantes. En el capítulo III se detalla la metodología de investigación aplicada al proyecto para llevar a cabo un orden sobre las acciones que se realizaron a lo largo del desarrollo del proyecto. En el capítulo IV se especifican los resultados que se obtuvieron al momento de implementar las funcionalidades del proyecto presentado. En el capítulo V se redactan las conclusiones y recomendaciones que se analizaron sobre todo lo trabajado y finalmente en el capítulo VI se visualiza el listado de la bibliografía usada para el proyecto.

CAPÍTULO I

CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN

1.1. Problema de investigación

1.1.1. Planteamiento del problema

La complejidad de los productos de software han hecho que los desarrolladores de sistemas informáticos busquen la manera de acelerar el desarrollo de software. Este objetivo se ha tratado de cumplir mediante el uso del modelado de sistemas utilizando UML [9], propuesto por *Object Management Group* (OMG) mediante *Model-Driven Architecture* (MDA). El modelado de software permite a los desarrolladores comprender y obtener una visión global del sistema, y ser una herramienta de comunicación entre los miembros del equipo de desarrollo [16]. Sin embargo, el tiempo que se toma para el análisis de requisitos y a partir de ellos la creación de los diagramas de clases que hacen posible la generación de código automático es considerable.

Para empezar con el modelado de un software se necesitan los requisitos planteados por el cliente. Un estudio reveló que el 95% de los documentos de requisitos de un sistema son redactados en lenguaje natural [9]. Todos los requisitos planteados son plasmados en casos de uso, siendo la herramienta de modelado más habitualmente utilizada para representar las especificaciones del software [17]. Estos casos de uso permiten analizar más a fondo entre el equipo de desarrolladores las necesidades del cliente. Sin embargo, cuando el equipo de desarrolladores redactan las descripciones de los casos de uso pueden detallar características técnicas del software que el cliente no logrará comprender [13].

Las características técnicas que se podrían detallar en las descripciones de los casos de uso pueden ser por ejemplo: nombre de clases, atributos, métodos, etc. que el analista necesite plasmar en los casos de uso. Con el uso de TDDT4IoTS los desarrolladores pueden especificar datos técnicos en los casos de uso escritos con SymLen y obtener de manera automática el diagrama de clases correspondiente. Pero, muchas de las veces los resultados no son los esperados por el desarrollador sin conocer las razones. Otro caso es que el desarrollador debe analizar cada uno de los requisitos en las que se describe el caso de uso para determinar los posibles errores en el uso de SymLen.

Al realizar el análisis de requisitos, los desarrolladores deben agregar información técnica que el cliente no podría comprender, lo que imposibilita a que el cliente revise los documentos por si solo. Por lo tanto, los desarrolladores deben dedicarle tiempo a revisar con el cliente los documentos de requisitos actualizados para proporcionar las especificaciones correspondientes.

Analizando el actual intérprete que utiliza la herramienta TDDT4IoTS, genera el diagrama de clases tomando las descripciones que estén bien escritas, sin retroalimentar a los miembros del equipo de desarrollo los errores que se hayan cometido al momento de redactar los casos de uso, dificultando la corrección de los mismos. Además, el formato generado no permite compartir los datos con otras librerías para visualizar el diagrama con otras herramientas que dibujen estos tipos de diagramas.

Diagnóstico

Aunque existen herramientas que permiten la generación automática de software, el tiempo para realizar el análisis de los requisitos y obtener la materia prima con la que se alimenta a estas herramientas hace que el proyecto sea demorado. Además, al realizar una modificación en los requisitos por omisiones del cliente o por errores de los desarrolladores, hace que se tenga que modificar los documentos de requisitos y a su vez realimentar a las herramientas que se han usado para el modelado y la generación de código automático.

Pronóstico

En el transcurso del proyecto pueden surgir varias anomalías que se deben tomar en cuenta que puedan llegar a suceder. Si la librería no logra identificar algunos errores escritos en las especificaciones de los casos de uso usando SymLen, será muy difícil que el analista corrija los errores en su respectiva escritura. Esto tendrá como consecuencia un tiempo más demorado para detallar los datos técnicos del modelado de software. Además, la generación del diagrama de clases tomara más tiempo en realizarlo haciendo poco atractivo el uso de SymLen.

1.1.2. Formulación del problema

¿Cómo gestionar la obtención del diagramas de clases a partir de la especificación de los casos de uso usando SymLen?

1.1.3. Sistematización del problema

1. ¿Cómo dar a conocer al desarrollador la incorrecta utilización del lenguaje SymLen en la escritura de los casos de uso para obtener el diagrama de clases esperado?
2. ¿Qué estructura de datos permitirá compartir el diagrama de clases generado por Armadillo con otras librerías para la generación de diagramas UML?
3. ¿Cómo determinar el nivel de efectividad del producto de este proyecto de investigación respecto a la generación de diagramas de clases?

1.2. Objetivos

La problematización de este proyecto ha llevado a plantearse los siguientes objetivos.

1.2.1. Objetivo General

Desarrollar una librería JavaScript para generar el diagrama de clases en formatos compatibles con librerías específicas para la generación de diagramas UML a partir de los casos de uso escritos en SymLen.

1.2.2. Objetivos Específicos

1. Retroalimentar a los desarrolladores sobre los errores de escritura de los casos de uso escritos en SymLen que impiden obtener el diagrama de clases esperado.
2. Generar los archivos json y xml con la información del diagrama de clases generado, compatible con otras librerías para la generación de diagramas UML.
3. Evaluar la librería JavaScript propuesta, con la elaboración de diagramas de clases a partir de casos de uso extendidos correspondientes a requisitos de un sistema de información.

1.3. Justificación

Los casos de uso son la herramienta que sirve para la obtención de los requisitos del sistema, y como herramienta de comunicación entre los miembros del equipo de desarrollo. Siendo la obtención de los requisitos una de las etapas que se podría afirmar que no es posible liberar de ella al equipo de desarrollo. En mucho de los casos la obtención de requisitos consume tiempo sustancial de la planificación de proyectos.

Una solución para mejorar el rendimiento de los desarrolladores es efectivizar el tiempo consumido en las fases y análisis de diseño del sistema, escribiendo los requisitos obtenidos, utilizando un lenguaje que permita que el usuario lea sin problema toda la información del caso de uso y permita identificar la información necesaria para generar el modelo (diagrama de clases) y el software del sistema informático de manera transparente para el usuario. Esto sin duda mejora los tiempos dedicados al desarrollo, y mejora la comunicación entre el usuario y el equipo de desarrollo. Hasta el momento existen herramientas de generación de código de software a partir del diagrama UML. Sin embargo, la materia prima para realizar estos diagramas debe ser obtenida por los desarrolladores en el análisis de requisitos consumiendo tiempo adicional.

Intentando mejorar los tiempos de desarrollo se ha implementado la herramienta TDDT4IoTS. Para los desarrolladores que utilicen la herramienta TDDT4IoTS será muy beneficioso contar con una librería que les permita interpretar la escritura de los casos de uso, y generar los diagramas de clases que puedan ser exportados para ser utilizados en otras herramientas que generen diagramas UML. Además, el producto final de este trabajo permite modificar los casos de uso y al mismo tiempo poder hacerlo con los datos referentes al diagrama de clases, tratando de reducir el tiempo en el refinamiento del modelado del software. Finalmente, se puede mencionar que no se ha encontrado aplicaciones o algún tipo de software que permita escribir los casos de uso y a partir de ellos generar diagramas UML.

CAPÍTULO II

FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN

FUNDAMENTACIÓN TEÓRICA

En este capítulo, se expondrán los trabajos relacionados con el presentado en este documento que se han identificado en la literatura, y que demuestran que el trabajo propuesto en este documento, aporta algo. Además, se contextualiza el trabajo y define los términos novedosos y de poco dominio para los investigadores y para la comunidad.

2.1. Marco conceptual

En esta sección, se detalla los modelos teóricos, conceptos, argumentos o definiciones que se han desarrollado o investigado en relación con el tema en particular.

2.1.1. Lenguajes de marcado

A continuación, se describen una serie de formatos de texto utilizados para el intercambio de datos entre varias aplicaciones.

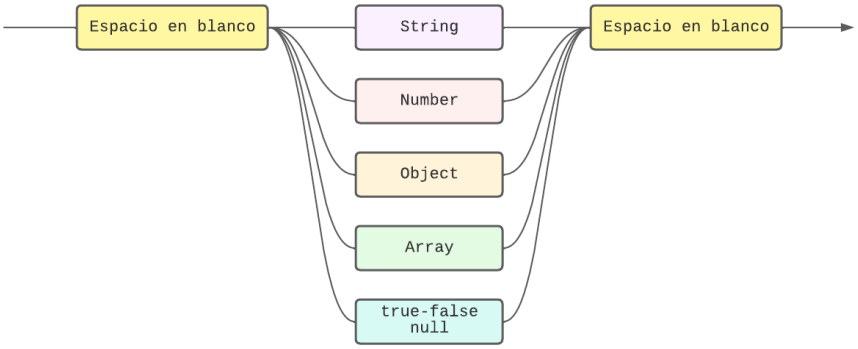
2.1.1.1. JSON

Javascript Object Notation (JSON) es un formato ligero de intercambio de datos. Consisten en asociación de nombres y valores. A pesar de ser independiente del lenguaje de programación, es admitido en una gran cantidad de lenguajes de programación. Se basa en un subconjunto del Estándar de lenguaje de programación JavaScript [18].

2.1.1.2. XML

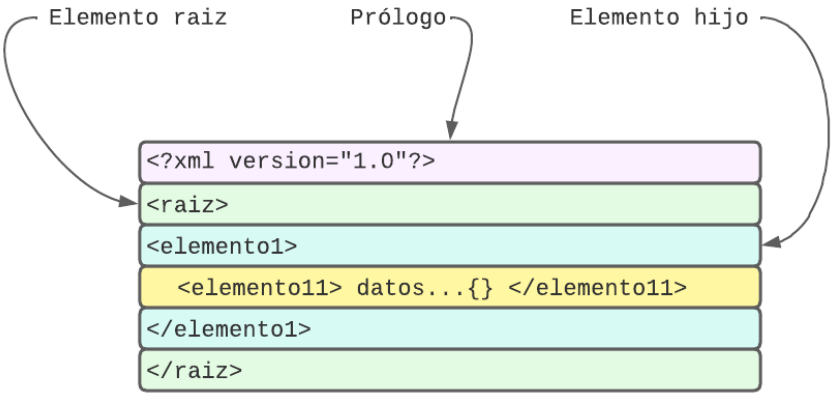
Es un lenguaje de marcado similar a HTML. Significa Extensible Markup Language y pertenece a la especificación W3C como lenguaje de marcado de propósito general. Esto significa que, a diferencia de otros lenguajes de marcado, XML no está predefinido, por lo que debe definir su propio marcado. El objetivo principal del lenguaje es compartir datos entre diferentes sistemas, como Internet [19].

Ilustración 2.1: *Estructura de los elementos de un json.*



FUENTE: BOURHIS, REUTTER Y VRGOC [18]
ELABORADO: AUTOR

Ilustración 2.2: *Estructura de los elementos de un xml.*



FUENTE: KHALILI [19]
ELABORADO: AUTOR

2.1.2. Compiladores

Están diseñados para traducir un fragmento de código escrito en un de lenguaje de programación a lenguaje de máquina que es, el que puede entender la computadora. El compilador analiza el código fuente en busca de errores antes de la traducción. Si se detecta un error, el compilador notificar al autor del código para que pueda arreglarlo. Además, los compiladores modernos o entornos de desarrollo (IDE), puede sugerir soluciones para algunos tipos de errores usando métodos de corrección de errores [20].

2.1.3. Modelamiento de software

Representan una serie de requisitos basados en la construcción de elementos visuales para definir estructuras y comportamientos que tendrá el software. UML (Lenguaje Unificado de Modelado) a través del mecanismo de perfilado, se han basado históricamente en notaciones gráficas. UML mediante el mecanismo de perfiles, maximiza la comprensión humana y facilita la comunicación entre las partes interesadas como son el cliente y desarrollador [21].

También existen lenguajes de modelado personalizados para distintas áreas, como por ejemplo en [22] proponen un lenguaje de modelado conceptual multinivel al denominan ML2 (Lenguaje de Modelado Multinivel). El lenguaje está orientado al modelado conceptual multinivel (de dominio) y pretende cubrir un amplio conjunto de dominios multiniveles. En el diseño de ML2 sigue un enfoque basado en principios, definiendo su sintaxis abstracta para reflejar una teoría formal para el modelado multinivel que se fue desarrollado previamente.

2.1.4. Metamodelado

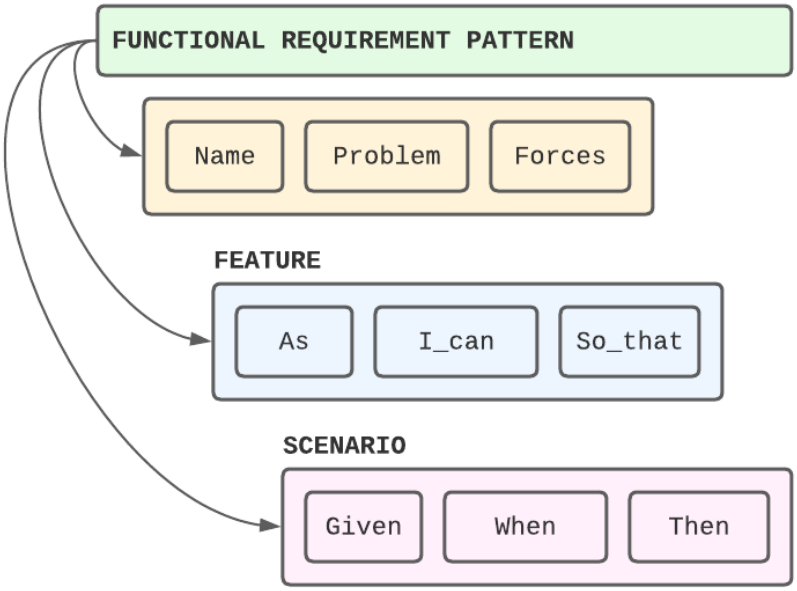
Existen varias formas de realizar el metamodelado de un software. En [23] mencionan el *Software Pattern MetaModel* (SoPaMM) o en su traducción al español Metamodelo del patrón de software. El objetivo principal de este proceso es la especificación de patrones de requisitos funcionales (FRP) vinculados a patrones de pruebas de aceptación (ATP). En la ilustración 2.3 se observa los componentes más importantes de un FRP relacionado con un ATP.

Basada en metodología ágil BDD, FRP es una estructura inspirada en la descripción de las historias de usuario. A continuación, en la ilustración 2.3 se detalla la estructura de un FRP [23]:

- **As:** Describir la parte que se beneficia de la característica.
- **I_can:** La característica en sí.
- **So_that:** El valor agregado de la característica.
- **Given:** Describe, en una o más cláusulas, el contexto inicial del escenario.

- **When:** Describe los eventos que desencadenan un escenario.
- **Then:** Describe, en una o más cláusulas, los resultados esperados del escenario.

Ilustración 2.3: *La estructura de un FRP.*

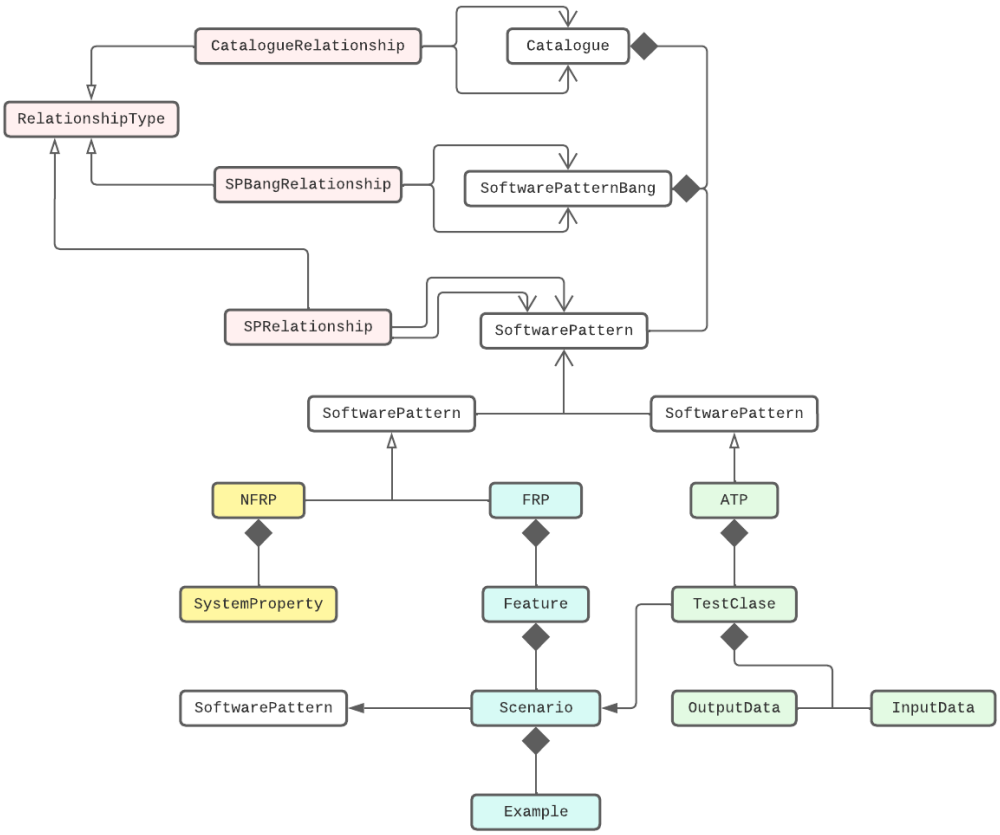


FUENTE: MOHAMED, MAZEN Y HELMY [23]
ELABORADO: AUTOR

FRP_User_Creation describe una parte de la función para crear usuarios. El usuario administrador es el actor beneficiado de esta función para que se logre registrar un nuevo usuario al sistema. El intento de crear un usuario conlleva a dos posibles escenarios: uno exitoso y otro no exitoso, pero ambos vinculados a una misma precondición. Sin embargo, dependiendo de la validez de los usuarios la ejecución puede ser distinta. Del mismo modo por cada escenario se representará un resultado diferente, es decir se visualizarán mensajes diferentes al momento de registrar un usuario o mensajes de error [23].

Finalmente, el ejemplo permite definir y vincular varios datos cada escenario. En cada escenario se consta con dos instancias de datos, de manera que uno de los escenarios registrado un nuevo usuario con éxito, mientras que el otro no lo hace debido a que los datos de entrada no son válidos, como es el número de identificación del usuario. Esta representación define el enfoque de patrones de requisitos funcionales basado en la ejecución de los posibles escenarios. En la ilustración 2.4 se observa el metamodelo de SoPaMM [23].

Ilustración 2.4: *Resumen del metamodelo SoPaMM.*



FUENTE: MOHAMED, MAZEN Y HELMY [23]
ELABORADO: AUTOR

2.1.5. Desarrollo ágil de software

El desarrollo ágil de software hace referencia a los principios del manifiesto ágil que son aplicadas por varias metodologías de desarrollo. Las metodologías de desarrollo ágil comparten varias características como: la entrega frecuente de software de trabajo (desarrollo iterativo), la velocidad constante y la comunicación abierta [24]. En el desarrollo ágil los métodos que se emplean pueden ser de bajo costo, es decir que cualquier modificación que se realice en alguna etapa de desarrollo el cliente podrá observar los resultados deseados referente al costo que ha pagado [25].

Además, de ser una buena alternativa para mejorar el desarrollo de software, el desarrollo ágil es muy productivo por que brecha de comunicación entre los clientes y los desarrolladores de software. Conforme avance el tiempo, difícilmente pueden surgir motivos para generar retrasos. Lo beneficioso de aplicar este proceso es que los cambios pueden ser acoplados

continuamente de acuerdo a las necesidades del cliente. A continuación, se mencionarán varias metodologías ágiles que se utilizan en el desarrollo de software [25]:

- **La programación extrema (XP):** Esta es una metodología centrada específicamente en obtener un software de calidad. Para cumplir con ese objetivo, se debe considerar en primer lugar una buena comunicación verbal entre todos los integrantes del proyecto. Además, no existen reglas que obliguen a llevar una documentación rigurosa, con el fin de aumentar la eficiencia de desarrollo del software.
- **Dynamic Systems Development Method (DSDM):** En DSDM o en su traducción al español Sistemas dinámicos Método de desarrollo la documentación es totalmente necesaria, debe ser magra y oportuna describiendo paso a paso todo el proceso que se realizó. Esto significa que el modelo y el prototipo deberán representar una visión general sobre lo que va hacer el sistema. Además, se deberán documentar los diagramas de modelado, estructuras físicas de datos y decisiones de diseño.
- **Scrum:** Se centra netamente en la versatilidad rentabilidad y adaptabilidad. Le permite al desarrollador tomar la decisión para elegir la técnica de ejecución del módulo que este encargado. La administración de Scrum permite reconocer las carencias u obstáculos que se presentaran a lo largo del proyecto. Los equipos de Scrum tienen la libertad de elegir y reconocer los enfoques para realizar su trabajo, es decir que cada integrante del grupo cuanta con las habilidades necesarias para cumplir su trabajo sin necesidad de recurrir a pedir ayuda fuera del grupo.

2.2. Marco referencial

El diagrama de clases sin duda es el artefacto más importante para modelar un sistema y el punto de partida para otros diagramas [26]. La problemática de la obtención de diagramas de clases a partir de los casos de uso detallados no ha sido tratada a profundidad. Por lo tanto, los investigadores se encuentran en un campo fértil de investigación.

Las soluciones que se han propuesto y que se han recuperado en este trabajo, utilizan el procesamiento del lenguaje natural (NLP por sus siglas en inglés *Natural Language Process*). Entre las soluciones encontradas podemos mencionar el trabajo de Chen y Zeng [27], en

el cual presentan una aproximación sobre la obtención del diagrama de casos de uso y el diagrama de clases UML a partir de los requisitos del producto expresados en lenguaje natural. Sin embargo, en este trabajo intentan analizar el texto para determinar el conjunto de palabras significativas que pueden representar los elementos de los diagramas, por ejemplo, para una clase un sustantivo, para un método un verbo nominal para los métodos, la conexión entre dos objetos (clases) expresarían una relación. El trabajo de Chen & Zeng [27] tiene buenos resultados con pocos y bien establecidos requisitos de software, que pueden animar a los investigadores a seguir mejorando esa herramienta, cómo por ejemplo, identificar los diferentes tipos de relaciones entre clases, y su representación en el diagrama UML.

El trabajo presentado por Dawood Omer & Eltyeb [28] es otra de las soluciones propuestas. En [28] proponen un modelo de razonamiento basado en casos que puede facilitar el proceso de generación de diagramas de clases UML a partir de requisitos textuales. Para ello utilizan técnicas de minería de texto. Por lo tanto, el trabajo es bastante abrumador: primero se debe tener una base de casos para entrenar el modelo, luego se debe entrenar el modelo. Además, aunque las diferencias sean muy pequeñas, los resultados (diagrama de clases UML) pueden variar.

En la misma línea de la aplicación de procesamiento del lenguaje natural para la obtención del diagrama de clases a partir de la descripción textual de los requisitos, podemos citar el trabajo de Alashqar [29], en el que, para lograr este objetivo proponen un algoritmo y una herramienta. El algoritmo consiste básicamente en la separación de la oración en cada palabra que la conforman para luego aplicar un análisis morfológico a cada palabra, según el tipo de palabra (sustantivo, verbos en diferentes voces o tiempos, etc.) determinar los diferentes elementos de los diagramas. En este caso, la herramienta que implementan para aplicar el algoritmo muestra las oraciones que han sido mal escritas. Alashqar en su trabajo [29] especifica que, para el algoritmo funcione correctamente, los requisitos textuales deben estar escritos con ciertas restricciones, cómo por ejemplo: cada oración debe estar escrita en una línea (separadas con una nueva línea), y que cada oración represente una acción a realizar con el software, entre otras restricciones.

El trabajo presentado por Shweta y Sanyal [30] propone un conjunto de reglas que a partir de un formato no estructurado de los requisitos del software, permita extraer de forma automática los elementos de un diagrama de clases. Mencionan que se necesitan de oraciones

que estén redactadas de forma activas y positiva. Además, existe un margen para investigar sobre el proceso de extracción mediante términos de palabras múltiples. La metodología propuesta utiliza la herramienta denominada Stanford CoreNLP junto con java para realizar la implementación practica de las reglas que se formularon. Como resultado de este trabajo mencionan que los diagramas de clases generados por la metodología, proporcionan una impresionante exhaustividad media (0,82), corrección media (0,92) y redundancia media (0,15) en relación con otros trabajos realizados manualmente por expertos en el área.

De los trabajos que se han revisado hay que recalcar que [29] y [30] se preocupan por el análisis de oraciones (acciones) pasivas negativas. Sin embargo, todos los trabajos que utilizan NLP están limitados al idioma en el que están escritos los requisitos, y al uso correcto de la gramática y todos consideran restricciones en la escritura. Por lo que se presenta a Armadillo como una librería que ayuda a que SymLen sea un lenguaje a utilizar para la escritura de los casos de uso detallados sin importar el idioma, y sin ninguna restricción. Con Armadillo la eficiencia de los diagramas de clases y la generación de código de software que satisface los requisitos del usuario, depende estrictamente del uso correcto de SymLen, y/o de la corrección de los posibles errores de su uso que Armadillo muestre al desarrollador.

CAPÍTULO III

METODOLOGÍA DE LA INVESTIGACIÓN

3.1. Localización

El proyecto se realizó en la Universidad Técnica Estatal de Quevedo (UTEQ), campus “La María”, ubicada en el Cantón Mocache de la Provincia Los Ríos, en Ecuador. Previo a la obtención del título de Ingeniero en Sistemas. La ejecución del proyecto duró 4 meses, desde el mes de junio de 2022 hasta el mes de septiembre del 2022.

Ilustración 3.5: *Localización de la UTEQ campus "La María".*

FUENTE: SATELITES.PRO
ELABORADO: AUTOR

.

3.2. Tipo de investigación

3.2.1. Investigación aplicada

Para la ejecución de este proyecto se analizó uno de los problemas que existen al momento de documentar el desarrollo de un software. Aplicando los conocimientos de ingeniería en sistemas se desarrolló una librería que interpreta las especificaciones de los casos de uso utilizando SymLen.

3.3. Métodos de investigación

3.3.1. Método inductivo

Con la ayuda del método inductivo se pudo conocer que para los desarrolladores de software es necesario contar con algún tipo de conocimiento sobre lenguajes de modelado para llevar un orden de trabajo mediante una metodología de desarrollo ágil que permita satisfacer las necesidades y requisitos que fueron planteados por el usuario.

3.3.2. Método deductivo

El lenguaje natural es comprendido por todas las personas dependiendo su origen. Por lo tanto, entre la comunidad de desarrolladores y sus clientes, se necesita crear 2 vistas disponibles para las descripciones de los casos de uso. La vista para los desarrolladores tratará sobre los requisitos redactados utilizando el lenguaje SymLen, en cambio, la vista para los clientes será redactada en lenguaje natural sin ningún tipo de símbolos para que puedan comprender las especificaciones.

3.3.3. Método analítico

Se empleó el método analítico para extraer los símbolos que serán analizados por la librería, con el objetivo de conocer qué símbolos serán los que se lograrán identificar en las descripciones de los casos de uso. En este caso se debe conocer los datos de entrada para ser procesados y generar lo requerido según las especificaciones redactadas.

3.4. Fuentes de recopilación de información

Como fuente primaria mediante la observación directa se logró recolectar información a través de los proyectos integradores propuestos por estudiantes de 5to a 8vo semestre de la carrera de Ingeniería en Sistemas de la Universidad Técnica Estatal de Quevedo (UTEQ) utilizando la herramienta TDDT4IoTS. Mediante las especificaciones de los casos de uso realizados por los estudiantes dirigidos por el Ing. Gleiston Guerrero, docente de la

Universidad, se identificaron los requisitos necesarios para Armadillo. En la tabla 3.1 se enlistan algunos proyectos integradores de estudiantes de la UTEQ.

Tabla 3.1: *Proyectos integradores de los estudiantes entre el 5to y 8vo semestre de la carrera Ingeniería en Sistemas.*

Estudiante	**Proyecto**	**Descripción**
PAOLA ZAMBRANO	Clasificadora de residuos	Diagrama de casos de uso / clase
VICTOR ROMERO	ArduinoVehicle Fire System	Fire safety system and carbon monoxide detector.
JONATHAN CEV.	Dispensador inteligente	App para dispensador de medicamentos
LUIS DE LA CRUZ	Sistema de Autenticación Biomé	Sistema de Autenticación Biométrica para el Acceso a Recursos Didácticos aplicando la metodología TDDM4IoTS
ADRIAN VEAS	Caso de Uso Reconocimiento Voz	Reconoce la voz en entorno ruidoso
PEDRO MOREIRA	Lazarillo-Caecus-Funcional	System to help visually impaired people identify objects
ANTHONY PACHAY	ArduinoVehicle Fire System	Anti-fire and carbon monoxide alert system for vehicles
EDUARDO GÁLVEZ	Geo posicionamiento docente	Proyecto de aplicaciones distribuidas
CARLOS ABAD	Explicación de la Metodología	Para la explicación en un video
JOHN PLAZARTE	Smart Security System	The intelligent security system monitors the areas that you want to restrict and protect from an intruder.
CARLOS ALMEIDA	Smart Security System	Smart Security System
ROBERTO SUÁREZ	Autenticación Biométrica, soft	Sistema de Autenticación Biométrica para el Acceso a Recursos Didácticos aplicando la metodología TDDM4IoTS

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Además, se llevó a cabo una búsqueda de papers, artículos científicos, revistas, libros y conferencias afines al tema de este proyecto. Se estableció un límite de ambigüedad entre los documentos encontrados entre los años 2018 y el presente año 2022. Entre los repositorios donde se realizaron las búsquedas se utilizaron: Scopus, Springer, Scielo y Google Scholar (ver tabla 3.2).

Tabla 3.2: *Algunos artículos científicos que ayudaron con la investigación de este proyecto.*

Autores	Año	Titulo	Fuente
Perkusich M, Chaves e Silva L	2020	Intelligent software engineering in the context of agile software development: A systematic literature review	Information and Software Technology
Islam G, Storer T	2020	A case study of agile software development for safety-Critical systems projects	Reliability Engineering and System Safety
Behutiye W, Karhapää P, L	2020	Management of quality requirements in agile and rapid software development: A systematic mapping study	Information and Software Technology
Mörbitz R, Vogler H	2021	Weighted parsing for grammar-based language models over multioperator monoids	Information and Computation
Abdalazeim A, Meziane F	2021	A review of the generation of requirements specification in natural language using objects UML models and domain ontology	Procedia CIRP
Karampure R, Wang C, Vashi Y	2021	UML sequence diagram to axiomatic design matrix conversion: A method for concept improvement for software in integrated systems	Procedia CIRP
Addazi L, Ciccozzi F	2021	Blended graphical and textual modelling for UML profiles: A proof-of-concept implementation and experiment	Journal of Systems and Software

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

3.5. Diseño de la investigación

Para el desarrollo del presente proyecto se tomarán en cuenta la ejecución de varias etapas, aplicando los principios del manifiesto ágil y desarrollo de software. A continuación, se describe el enfoque metodológico correspondiente a cada una de las fases.

3.5.1. Metodología de investigación aplicada al proyecto

Para ejecutar el presente proyecto, se diseño una metodología basada en el Manifiesto por el Desarrollo Ágil de Software. Con el objetivo de cumplir las directrices de una metodología

ágil, se analizaron los valores y principios del manifiesto en el que se basan. En la tabla 3.3 se redactan los 12 principios del manifiesto ágil y el análisis aplicado a la metodología [31].

Tabla 3.3: *Los 12 principios ágiles y su análisis en la metodología.*

Principios del manifiesto ágil	**Análisis aplicado a la metodología**
1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor	Para la entrega continua de cambios sobre el desarrollo del guion, se describieron varias fases de desarrollo.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competición al cliente.	A lo largo de la ejecución de la metodología se podrá realizar modificaciones en los requisitos que se plantearon, con el objetivo de mejorar las funcionalidades del software.
3. Entregamos software funcional, frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.	Se organiza una fase en la que se genera todo el código fuente del sistema informático con comprobaciones constantes.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.	Durante la ejecución del proyecto se mantiene constante contacto con el cliente
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.	Para brindar el entorno de trabajo adecuado a los desarrolladores, la metodología se la puede llevar a cada de forma remota, es decir en la comodidad de su hogar.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.	Se realizaron reuniones virtuales y en ciertas ocasiones reuniones presenciales con el cliente final.
7. El software funcionando es la medida principal de progreso.	Se realiza el desarrollo completo de cada módulo que fue designado.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.	Dentro de la metodología se especifican varias fases en las cuales todos los integrantes pueden opinar o tratar con mejoras mediante ideas claras y precisas.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.	En la fase de desarrollo se debe hacer la codificación de manera forma, con código Totalmente documentado y organizado.

Principios del manifiesto ágil	Análisis aplicado a la metodología
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.	Al momento de entrar en la fase final de desarrollo se realiza análisis sobre la funcionalidad intuitiva del software.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.	Todos los integrantes del proyecto estarán designados por varios roles que permitirán llevar todo de forma ordenada.
12. A intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.	Cuando se culmina la fase de desarrollo se ejecuta una revisión general sobre todo el funcionamiento del software mediante pruebas de software.

FUENTE: [31]
ELABORADO: AUTOR

3.5.1.1. Descripción general de la metodología

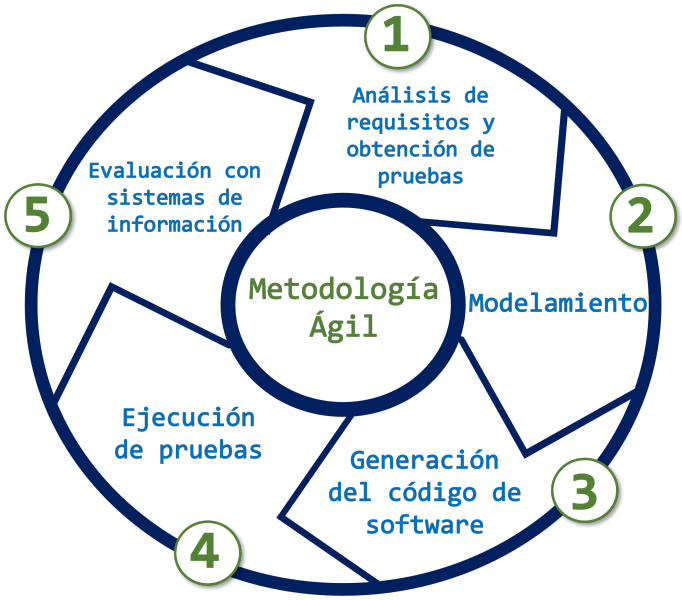
Esta metodología consta de 5 fases, que están relacionadas con el desarrollo de un sistema, desde el análisis de requisitos hasta su debido mantenimiento. Esta última es muy olvidada por los investigadores, y es de mucho interés en el ciclo de vida de desarrollo. En la ilustración 3.6 se pueden observar las fases que se deben seguir en la metodología de este proyecto.

Las principales razones del uso de una metodología ágil es el uso de un ciclo de desarrollo iterativo e incremental para la ejecución de este proyecto son:

- Entregas frecuentes y continuas de forma que puede disponer de una funcionalidad básica en un tiempo mínimo y a partir de ahí un incremento y mejora continua del sistema.
- Previsible inestabilidad de requisitos.
- Es posible que durante la ejecución del proyecto se altere el orden en el que se desean recibir los módulos.

Para mejorar la ejecución de la metodología se organizaron diferentes tipos de roles que deben ser tomados por las personas que desarrollen del proyecto. Cada rol implementado

Ilustración 3.6: *Fases de la metodología*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

juega un papel muy importante dentro de la metodología, debido a que deben cumplir tareas de suma importancia para controlar todas las fases que se deben seguir y cumplir los objetivos del trabajo a desarrollar.

Roles El equipo de esta metodología está conformado por 2 roles, director de proyecto (DP) y el Desarrollador de Sistemas (DS). Todos los miembros de un equipo de desarrollo tienen diferentes roles en la gestión y supervisión de los proyectos. Todos los roles son necesarios para que el proceso funcione eficientemente.

- **Director de proyecto (DP):** Se encarga de administrar el proceso del proyecto, su planificación, coordinación con el analista y realizar un seguimiento e informes del progreso del proyecto, en términos de calidad, costo y plazos de entrega. El DP es la interfaz principal entre el propietario del producto y el analista de desarrollo de software.

- **Desarrollador de Sistemas (DS):** La persona encargada de este rol debe contar con conocimientos de desarrollo de software. El desarrollador deberá estar totalmente familiarizado con el objetivo principal del trabajo. Debido a que será el encargado de analizar todos los requisitos que se necesiten para cumplir con el funcionamiento

óptimo del sistema, también se encargará de analizar, diseñar y codificar el producto final del proyecto.

  - Comprometerse al inicio de cada módulo y desarrollar todas las funcionalidades en el tiempo determinado.
  - Son responsables de entregar un producto a cada término de un módulo.
  - Definir el desarrollo del sistema.

Fases En esta sección se describen cada una de las fases que se mencionaron en el apartado anterior, también se mencionaran breves conceptos relacionados con cada fase y como pueden mitigar los problemas que se pueden presentar a lo largo de la ejecución de la metodología redactada.

1. **Análisis de requisitos y obtención de pruebas:** Para la recopilación de los requisitos del software es necesario comprenderlos en profundidad para asegurar su correcto funcionamiento cuando este desarrollado. Si al momento de recopilar la información, no lleva todo el tiempo suficiente para ser analizado correctamente, podría afectar todo el proyecto en general [23].

   Para que los requisitos obtenidos sean realizados de manera correcta se pueden priorizar. Existen técnicas actuales para proceder la priorización de requisitos como es el Proceso de Jerarquía Analítica (AHP), ya que produce resultados muy precisos. A continuación, se especifican tres tipos de técnicas de priorización [23]:

   - *Escala nominal:* El personal de trabajo que recopilo los requisitos, deberán asignar cada requisito a un grupo de prioridad, transformando a todos los requisitos de ese grupo prioritarios. La asignación numérica categoriza los requisitos en grupos. Caga grupo cuenta con un identificador único que les permite ser identificados dentro de los demás grupos.
   - *Escala ordinal:* Esta técnica produce una lista de requisitos y cada uno cuenta con una prioridad en específica. Existe una técnica similar a la de escala nominal que se basa en crear varios grupos de prioridad. Pero solo son 3 tipos de grupos: alto, medio y bajo, los desarrolladores priorizan y clasifican los requisitos dentro

del mismo grupo en otro subgrupo; ese bucle se repite hasta que cada grupo tenga solo uno.

- *Escala de relación:* Son similares a la escala ordinal, además muestran importancia relativa entre todos los requisitos, es decir que dan valores de prioridad a los requisitos. En estas técnicas, los desarrolladores conocen hasta qué punto cada requisito es más importante que el resto. La votación acumulativa (CV) es una técnica proporcional que depende de la votación de los desarrolladores; cada desarrollador tiene 100 puntos y se distribuyen entre los requisitos en función de su prioridad

La obtención de pruebas se basa en la recolección de los posibles escenarios que puedan ocurrir al momento consumir el producto final del proyecto. Además, se necesita conocer el resultado que se deberá obtener por los parámetros ingresados por cada prueba.

2. **Modelamiento:** El modelado de software básicamente son las abstracciones que describen la arquitectura de un sistema informático. Para llevar a cabo la ejecución de esta fase es necesario tener conocimientos sólidos sobre el desarrollo de software, debido a que los modelos pueden tener diferentes niveles de abstracción y detalles.

   En [32] se propone un metamodelo que permite representar al software a base de patrones de diseño. Es decir, se pueden crear diferentes tipos de diagramas personalizados especificando el comportamiento que tendrá el sistema informático cuando esté en ejecución. Además, se pueden vincular los patrones encontrados con pruebas de aceptación con los usuarios que usaran el producto final del proyecto presentado.

3. **Generación de código:** En la fase de generación de código se relaciona con obtener el producto o software de manera real. En esta etapa los desarrolladores deberán cumplir con algunos puntos claves en el desarrollo de software que son: personalizar la interfaz de usuario, implementar la lógica comercial, integrar servicios de terceros si es necesario o resolver problemas de desarrollo muy complejos que requieran un análisis más profundo para llegar a una solución óptima [33].

   Se recomienda a los encargados de ejecutar esta etapa tener conocimientos de programación. En [33] mencionan qué si la curva de aprendizaje de los desarrolladores

es baja, no necesitan obtener conocimientos al momento de generar el código del software. Esto permitirá suponer que el nivel de conocimiento puede impactar en el proceso de aprendizaje y adopción de la tecnología.

Además, para suprimir un poco los grandes desafíos a los que se enfrentan los desarrolladores en esta fase, se recomienda el uso de tecnologías con una documentación detallada y cuenten con recursos de aprendizaje. Otra solución potente para ayudar a los desarrolladores es basarse en un sistema similar al que se pretende desarrollar usando el conocimiento de aplicaciones previamente desarrolladas [33].

4. **Ejecución de pruebas:** La siguiente fase está relacionada con la primera. Como se podrá observar en los títulos de estas dos fases, se realiza la obtención y ejecución de pruebas sobre el software que se obtendrá. Existen varios problemas que surgen al momento de poner en producción un software desarrollado. En [34] muestran varios estudios que indican que algunos tipos de fallos son intrínsecamente difíciles si no imposibles de detectar en entornos de desarrollo.

5. **Evaluación con sistemas de información:** La fase final de esta metodología permitirá usar los casos de uso de sistemas de información ya desarrollados y comparar el diagrama de clases generado utilizando Armadillo y el de la documentación de ese sistema. Se recomienda utilizar sistemas de información con su documentación totalmente detallada, de esa manera se ahorra tiempo en analizar e ingresar a la información para generar el diagrama [24].

3.6. Recursos humanos y materiales

En la siguiente sección se describen los recursos que fueron necesarios para la ejecución del presente proyecto.

3.6.1. Talento humano

Organización y planeación sobre el personal que se encargará de llevar a cabo el proceso de desarrollo del proyecto. Se encarga de coordinar todas las actividades que serán necesarias para que el proyecto sea culminado con éxito.

3.6.2. Equipo de trabajo

Este proyecto sera ejecutado por un desarrollador y un director de proyecto. El desarrollador es *Dúval Ricardo Carvajal Suárez* y el director de proyecto es *Ing. Gleiston Cicerón Guerrero Ulloa*.

3.6.3. Recursos de software

Los recursos de software que se utilizaron se detallan en la tabla 3.4

Tabla 3.4: *Recursos de software utilizados en el proyecto*

Software	**Descripción**	**Vigencia**
Lucidchart	Diseñar algunos gráficos.	Libre (Gratuito)
WebStorm	Editor de código.	Lic. corp. (Universidad)
TexStudio	Herramienta para realizar la documentación del proyecto	Libre (Gratuito)
Teams	Herramienta para realizar videoconferencias	Libre (Gratuito)
Sistema operativo	Windows 11	Libre (Gratuito)
Programas de oficina	Microsoft Excel, Word, PowerPoint	Lic. corp. (Universidad)

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

3.6.4. Recursos de hardware

Los recursos de hardware que se utilizaron se detallan en la tabla 3.5

Tabla 3.5: *Recursos de hardware utilizados en el proyecto*

Cantidad	**Equipo**	**Adquisición**	**Costo total**
1	Laptop.	$ 1.200,00	$ 1.200,00
1	Disco duro extraíble.	$ 500,00	$ 500,00

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

4.1. Resultados de la metodología

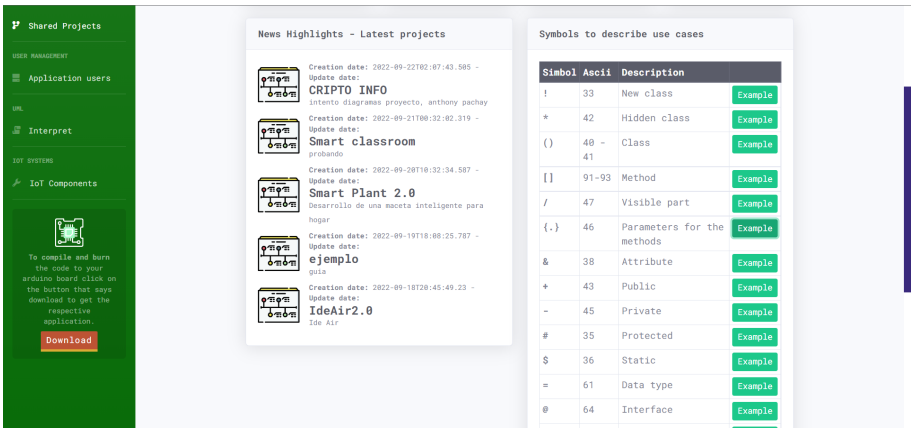
Los resultados que se obtuvieron al aplicar la metodología basada en el manifiesto del desarrollo ágil de software se redactan a continuación:

4.1.1. Análisis de requisitos y obtención de pruebas

Para llevar a cabo el análisis de los requisitos sobre la librería JavaScript que se desarrolló se tomaron en cuenta varios favores observando las carencias que existen al momento de realizar el modelamiento de cualquier software. Mediante el docente encargado de impartir las clases sobre cómo utilizar los lenguajes de modelado como lo es UML se dio cuenta que podría existir una forma en generar uno de los diagramas más importantes que es el diagrama de clases, pero a partir de las descripciones que se generan en los casos de uso del sistema.

La herramienta TDDT4IoTS permite realizar el modelamiento preliminar de un software. La herramienta cuenta con SymLen para escribir las descripciones de casos de uso con datos técnicos sobre los objetos que intervendrán en el sistema informático que llevara a cabo la ejecución de todos los escenarios que se están describiendo. A petición del cliente recomendó que visitemos el sitio web de `aplicaciones.uteq.edu.ec/tddt4iots` para visualizar los símbolos que usa la herramienta (ver ilustración 4.7).

Ilustración 4.7: *Símbolos que utiliza la herramienta TDDT4IoTS*

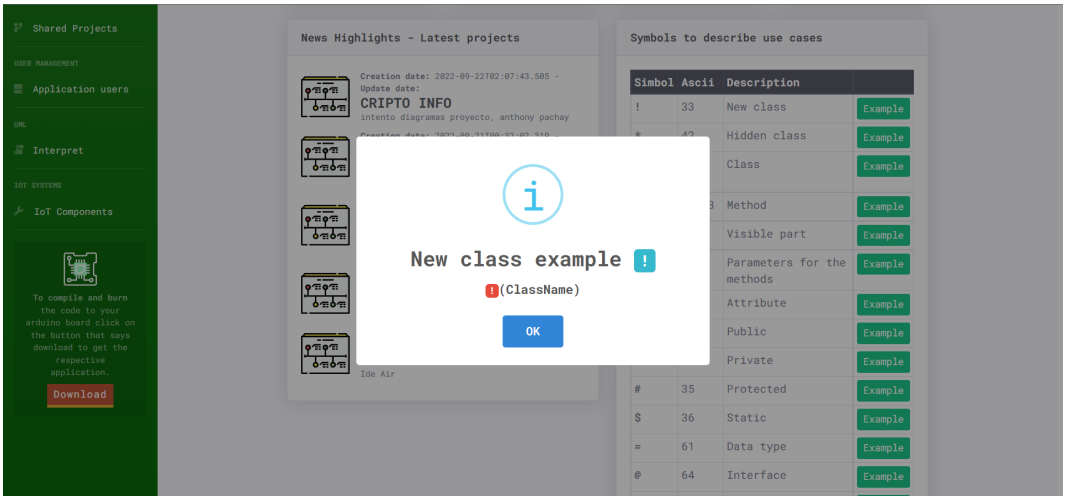


FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

En la lista de símbolos que se visualiza en la ilustración anterior se encuentra un botón que dice *"Example"*. Si presionamos en ese botón se pudo observar un ejemplo detallado sobre cómo se deben utilizar los símbolos para tratar de especificar algún objeto del diagrama de clases que se genera mediante la descripción del caso de uso pertinente (ver ilustracións 4.8, 4.9).

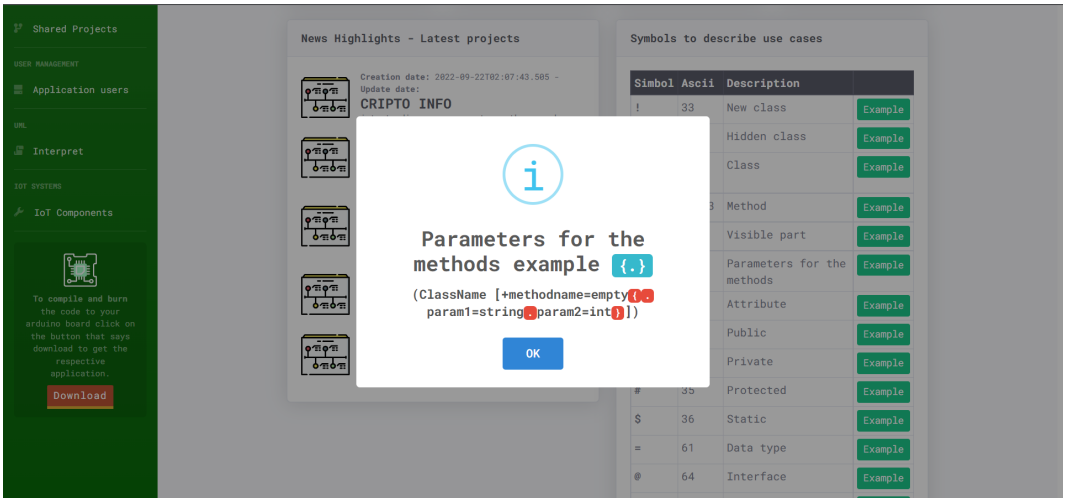
Ilustración 4.8: *Ejemplo de cómo se debe utilizar el símbolo respectivo para crear una clase.*



FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

Ilustración 4.9: *Ejemplo de cómo se debe utilizar el símbolo respectivo generar los parámetros de un método declarado con el lenguaje de símbolos.*



FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

Luego de tener una idea general sobre los símbolos que utiliza la herramienta se especificaron detalladamente los significados de cada símbolo y como puede ser utilizado en las descripciones de los casos de uso.

- **Asterisco ***: Este símbolo sirve para ocultar cualquier carácter que se encuentre redactado después de el mismo. Esto permitirá al momento de interpretar la descripción del caso de uso ingresada eliminar los caracteres mezclados con los símbolos y solo dejar visibles los caracteres normales formando un texto natural que pueda ser entendido por cualquier persona regular. **Ejemplo:**

  ```
  * texto de prueba
  ```

- **Paréntesis de apertura y cierre** (): Este símbolo sirve para especificar uno de los componentes más importantes del diagrama de clases, dentro de los paréntesis de apertura y cierra se deberá especificar el nombre de la clase u objeto que se pretende generar de forma automática. Cabe mencionar que, si se escribe el nombre de clase separada por espacios, la librería deberá omitir estos espacios y auto completarlos con la segunda letra después del espacio con mayúscula. **Ejemplo:**

  ```
  (Nombre de la clase)
  ```

- **Corchete de apertura y cierre []:** Este símbolo podrá ser utilizado dentro de los símbolos para crear las clases, permitirá definir Los métodos o funciones que pertenecerá a la clase respectiva. **Ejemplo:**

  ```
  (Nombre de la clase [+nombre del método=empty])
  ```

- **Slash /:** Este símbolo es bastante interesante, con él se deben visualizar caracteres que se encuentre encerrado de un slash de apertura y otro slash de cierre. Este símbolo se lo deberá usar cuando todos los caracteres se encuentren después del símbolo del asterisco, la idea es permitir que se visualicen caracteres específicos dentro de los datos técnicos para generar el diagrama de clases, sin afectar el uso de los demás símbolos. **Ejemplo:**

```
(Nombre de la clase &+/nombre/=string)
```

- **Llaves de apertura y llaves de cierre con punto {.}:** Con el símbolo de las llaves se podrá definir los parámetros de entrada para los métodos o funciones de las clases. Cada parámetro estará separado por un punto, además se deberá utilizar el símbolo del igual (=) para especificar su tipo de dato. Cada recalcar que todos los atributos también deberán estar especificados dentro de las llaves de apertura y cierre. **Ejemplo:**

  ```
  (Nombre de la clase [+nombre del método=empty
  {.param1=string.param2=strgin}])
  ```

- **Ampersand &:** Este símbolo debe permitir declarar los atributos de la clase que fue creada con el símbolo anterior. Existen otros símbolos que interviene en la creación de otros componentes del diagrama de clases, más adelante se detallaran para que sirven. **Ejemplo:**

  ```
  (Nombre de la clase &+atributo uno=string &+atributo
  dos=string)
  ```

- **Visibilidad +:** El símbolo de suma permite especificar que la visibilidad del atributo, método o función será de manera pública. **Ejemplo:**

  En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en atributos de clase.

  ```
  (Nombre de la clase &+atributo uno=string &+atributo
  dos=string)
  ```

  En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en métodos o funciones de clase.

  ```
  (Nombre de la clase [+nombre del método=empty
  {.param1=string.param2=strgin}])
  ```

- **Visibilidad -:** El símbolo de resta permite especificar que la visibilidad del atributo, método o función será de manera privada. **Ejemplo:**

  En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en atributos de clase.

  ```
  (Nombre de la clase &-atributo uno=string &-atributo
  dos=string)
  ```

  En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en métodos o funciones de clase.

  ```
  (Nombre de la clase [-nombre del método=empty
  {.param1=string.param2=strgin}])
  ```

- **Visibilidad #:** El símbolo de almohadilla permite especificar que la visibilidad del atributo, método o función será de manera protegida. **Ejemplo:**

  En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en atributos de clase.

  ```
  (Nombre de la clase &#atributo uno=string &#atributo
  dos=string)
  ```

  En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en métodos o funciones de clase.

  ```
  (Nombre de la clase [#nombre del método=empty
  {.param1=string.param2=strgin}])
  ```

- **Visibilidad $:** El símbolo de dólar permite especificar que la visibilidad del atributo, método o función será de manera estática. **Ejemplo:**

  En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en atributos de clase.

  ```
  (Nombre de la clase &$atributo uno=string &$atributo
  dos=string)
  ```

En este ejemplo se visualiza la forma en cómo utilizar el símbolo de suma en métodos o funciones de clase.

```
(Nombre de la clase [$nombre del metodo=empty
{.param1=string.param2=strgin}])
```

- **Igual =:** El símbolo de igual debe permitir asignar el tipo de dato a los atributos, métodos o funciones declaradas dentro de la clase. En el caso de los métodos de tipo **void** se deberá utilizar la palabra *empty* indiciando que es un método que retornara ningún valor. **Ejemplo:**

```
{.param1=string.param2=strging}
```

- **Arroba @:** El símbolo de arroba se lo utiliza dentro de los paréntesis que permiten generar una clase del diagrama. Pero se recuerda que en el diagrama de clases también se pueden crear interfaces. El objetivo de este símbolo es utilizarlo dentro de los paréntesis, pero el interprete identificara que sera una interfaz la que se deberá crear. **Ejemplo:**

```
(@nombre interfaz)
```

- **Signo de interrogación apertura y cierre ¿?:** El símbolo de interrogación se lo utiliza para generar otro tipo de objeto principal del diagrama de clases. El objetivo principal de este símbolo es generar los enum. **Ejemplo:**

```
¿enumTest?
```

- **Cierra comillas bajas »:** El símbolo de cierre comillas bajas se lo utiliza para agregar atributos a un objeto principal del diagrama de clases. El objetivo principal de este símbolo es atributos dentro de los objetos enum. **Ejemplo:**

```
¿enumTest »atributoUno »atributoDos?
```

- **Porcentaje %:** El símbolo de porcentaje sirve para agregar nuevos constructores a las clases que ya fueron definidas, además para especificar sus parámetros se utiliza los mismos símbolos para agregar atributos a la clase, con la diferencia que los parámetros van separados por la coma. **Ejemplo:**

```
*(class %.class=Class, .param=string%)
```

Dentro del diagrama de clases existen diferentes tipos de relaciones con las que se pueden relacionar las clases que se encuentran definidas. Para implementar las relaciones también se utilizan una combinación de símbolos que permitan detectarlas y la librería se encarga de generarlas. A continuación, se especificarán los símbolos que se utilizan para los tipos de relaciones en el diagrama de clases:

- **Símbolo de admiración de apertura ¡!:** Para indicar que se va realizar una relación entre 2 objetos del diagrama de clases, el texto debe estar entre este símbolo tanto en su apertura y cierre.
- **Corchete de apertura y cierre []:** Dentro del símbolo de admiración se podrá indicar un texto como leyenda sobre la línea de la relación, esto indicara alguna palabra clave sobre la relación entre los objetos.
- **Cardinalidad 1 o n:** Sobre la línea de la relación se puede indicar una cardinalidad de uno a muchos. El numero *1* indicará que la cardinalidad es de 1 y si usa la letra *n* indicará que la cardinalidad des de muchos. Se los debe utilizar en la parte exterior de los corchetes.
- **Agregación > >:** Para indicar que el tipo de relación entre dos objetos del diagrama de clases se utiliza los símbolos de > >. Se deben colocar en la parte exterior donde se el texto de cardinalidad. **Ejemplo:**

  ```
  *¡claseUno 1>[texto de cardinalidad]>n claseDos!
  ```

- **Dependencia < <:** Para indicar que el tipo de relación entre dos objetos del diagrama de clases se utiliza los símbolos de < <. Se deben colocar en la parte exterior donde se el texto de cardinalidad. **Ejemplo:**

  ```
  *¡claseUno 1<[texto de cardinalidad]<n claseDos!
  ```

- **Generalización < >:** Para indicar que el tipo de relación entre dos objetos del diagrama de clases se utiliza los símbolos de <>. Se deben colocar en la parte exterior donde se el texto de cardinalidad. **Ejemplo:**

```
*¡claseUno 1<[texto de cardinalidad]>n claseDos!
```

- **Asociación > <:** Para indicar que el tipo de relación entre dos objetos del diagrama de clases se utiliza los símbolos de > <. Se deben colocar en la parte exterior donde se el texto de cardinalidad. **Ejemplo:**

```
*¡claseUno 1>[texto de cardinalidad]<n claseDos!
```

En la tabla se describe el código ASCII para escribir los símbolos mediante el teclado de una computadora. Se pretendió que cada símbolo tenga una combinación ASCII para que el analista pueda escribir cada símbolo con el teclado y no obtenerlo de alguna forma que sea complicada.

Tabla 4.6: *Código ASCII para cada símbolo del lenguaje.*

Símbolo	**ASCII**	**Descripción**
*	33	Ocultar texto
( )	40 - 41	Crear clase
[ ]	91 - 93	Crear método
/	47	Parte visible
{.}	123 - 125	Parámetro de los métodos
&	38	Atributo de clase
+	43	Publico
-	45	Privado
#	35	Protegido
$	36	Estático
=	61	Tipo de dato
@	64	Interfaz
¿?	168 - 63	Enum
%	37	Constructores

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Para definir los diferentes tipos de relaciones que se usaran en el diagrama de clases. En la tabla se visualiza un grupo de símbolos que servirán para crear las relaciones de tipo: agregación, dependencia, herencia o generalización y asociación.

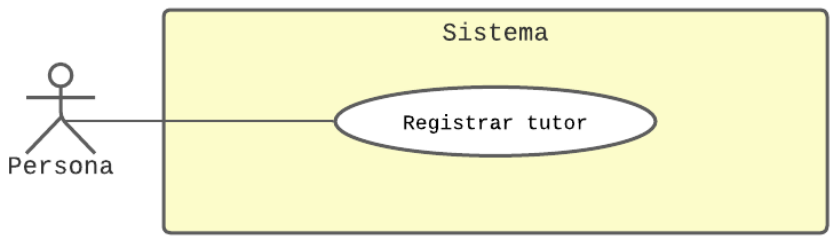
Tabla 4.7: *Código ASCII para cada símbolo del lenguaje que especifican las relaciones.*

Símbolo	**ASCII**	**Descripción**
¡!	33 - 173	Crear relación
[ ]	91 - 93	Especificar leyenda en la relación
1[ ]1	–	Cardinalidad de uno a uno
1[ ]n	–	Cardinalidad de uno a muchos
n[ ]1	–	Cardinalidad de muchos a uno
> >	62 - 62	Agregación
< <	60 - 60	Dependencia
< >	60 - 62	Generalización
> <	62 - 60	Asociación

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Luego de a ver recopilado todos los símbolos que la librería va a interpretar, el docente proporciono 1 caso de uso. Además, proporciono el gráfico del diagrama de clases que se escribió en los casos de uso. En la ilustración 4.10 se observa el caso de uso de prueba para saber que objetos debe generar la librería.

Ilustración 4.10: *Diagrama de casos de uso para las pruebas.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Para comprender totalmente las acciones que se deben ejecutar en el caso de uso que se usara como prueba, está escrito en 2 distintas formas. En la tabla 4.8 se observa la descripción del caso de uso en lenguaje natural especificando todas las acciones con el objetivo que cualquier persona regular logre entender. En la tabla 4.9 se observa la misma descripción del caso de uso pero utilizando los símbolos.

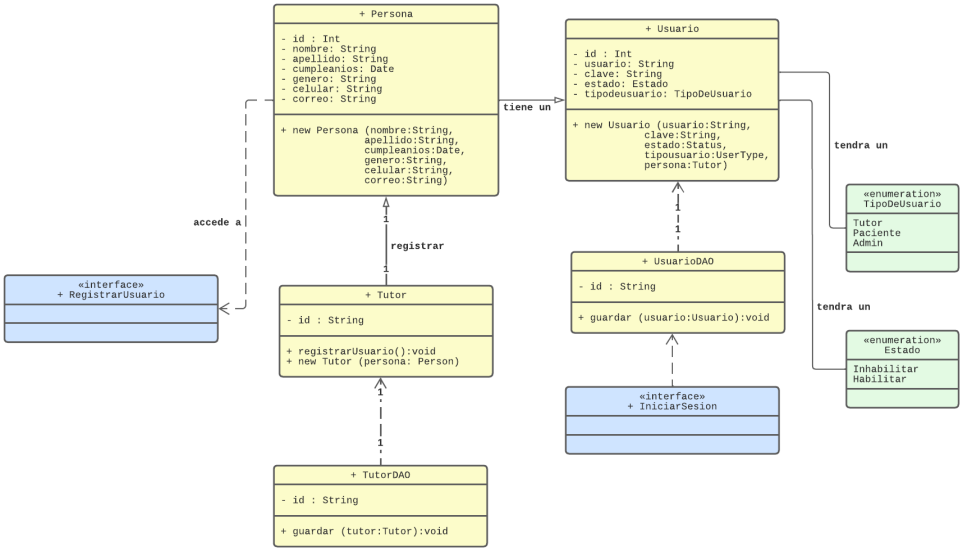
Tabla 4.8: *Descripción del caso de uso Registrar tutor.*

Caso de uso:	Registrar tutor
Actores:	Persona
Poscondición:	Usuario registrado en la base de datos.
Precondición:	El usuario existe en la base de datos.
Propósito:	Ingresar a la interfaz principal de la aplicación
Resumen:	Permite identificar las credenciales del usuario que intenta ingresar al sistema, de esa forma verificar el tipo de usuario que ingresa.
Tipo:	Primario
Flujo normal	
1. Este caso de uso comienza cuando una persona quiere registrarse como usuario tutor en el sistema.	
2. La persona accede entonces a la interfaz de registro de usuarios.	
	3. Le muestra los campos a rellenar en la interfaz de registro de usuarios: nombre, apellido, fecha de nacimiento, sexo, número de teléfono, correo electrónico, nombre de usuario contraseña. Hay que tener en cuenta que el registro de una Persona tiene un Usuario. Hay estados que pueden ser Desactivado o Activado que cada Usuario tendrá un Estado. También hay varios tipos de usuarios, que son Tutor o Paciente o Administrador y cada Usuario tendrá un Tipo de Usuario. Además, el sistema asigna un id cuando lo almacena.
4.La persona introduce todos los datos requeridos por la vista, y hace clic en enviar.	
	5. Crear un objeto persona.
	6. Crear un objeto tutor.
	7. Crear un objeto de usuario.
	8. Guarda los datos del tutor en la base de datos.
	9. Guarda los datos del usuario en la base de datos.
	10. Este caso de uso termina cuando el sistema muestra la interfaz de inicio de sesión.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Para la elaboración de la descripción del caso de uso con los símbolos proporcionaron información de un diagrama de clases que ya se encontraba realizado. El objetivo de este caso de uso era escribir todas las acciones con información técnica que permita crear el diagrama de clases lo más parecido posible al original (ver ilustración 4.11).

Ilustración 4.11: *Diagrama de clases de prueba*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.9: *Descripción del caso de uso para registrar tutor.*

Caso de uso:	Registrar de tutor
Actores:	Persona
Poscondición:	Usuario registrado en la base de datos.
Precondición:	El usuario existe en la base de datos.
Propósito:	Ingresar a la interfaz principal de la aplicación
Resumen:	Permite identificar las credenciales del usuario que intenta ingresar al sistema, de esa forma verificar el tipo de usuario que ingresa.
Tipo:	Primario
Flujo normal	
1. Este caso de uso comienza cuando una *(/persona/ &-id=int) quiere registrarse como usuario *(/tutor/ &-id=int [+usuarioRegistrar=Tutor]) en el sistema. *¡tutor 1<[registro]>n Persona!	
2. La persona accede entonces a la interfaz de *(@/Registro de Usuarios/). *¡Persona1<[accede]<1 Registro de Usuario!	

	3. Muestra los campos a rellenar en la interfaz de *(@/Registro de usuarios/): *(Persona &-/nombre/=String, &-apellido/=String, &-/fecha de nacimiento/=Date, &-/género/=String, &-/número de teléfono/=String, &-/email/=String) *(Usuario &-id=Int, &-/nombreusuario/=String, &-/clave/=String &-estado=Status &-tipo de usuario=UserType). Hay que tener en cuenta que el registro de una *!/Persona/ u<[/tiene un/]>u /Usuario/¡. Hay estados que pueden ser *¿Estado "/Deshabilitado/ /o/ "/Habilitado/? que cada *¡/Usuario/ u>[/tendrá un/]<u /Estado/! . También hay varios tipos de usuarios, que son *¿Tipo de usuario "/Tutor/ /o/ "/Paciente/ /o/ "/Admin/? y cada *¡/Usuario/ u>[/tendrá un/]<u /Tipo de usuario/! Además, el sistema asigna un identificador al almacenarlo.
4. La persona introduce todos los datos requeridos por la vista, y hace clic en enviar.	
	5. Crear un objeto *(/persona/ %.nombre=String, .apellido=String, .fechacumpleanios=Date, .genero=String, .celular=String, .correo=String%).
	6. Crear un objeto *(/tutor/ %.person=Person%)
	7. Crear un objeto *(/usuario/ %.nombreusuario=String, .clave=String, .estado=Status, .tipodeusuario=UserType, .persona=Tutor%)
	8. *(TutorDAO &id=Int [/Guarda/=Tutor.tutor=Tutor])r el tutor en la base de datos *¡TutorDAO<[]< tutor!.
	9. *(UserDAO &-id=Int [/Guarda/=User.user=User])r el usuario en la base de datos *¡UserDAO <[]< User!.
	10. Este caso de uso finaliza cuando el sistema muestra la interfaz de *(@/Login/). *¡Login u<[]<u UserDAO!

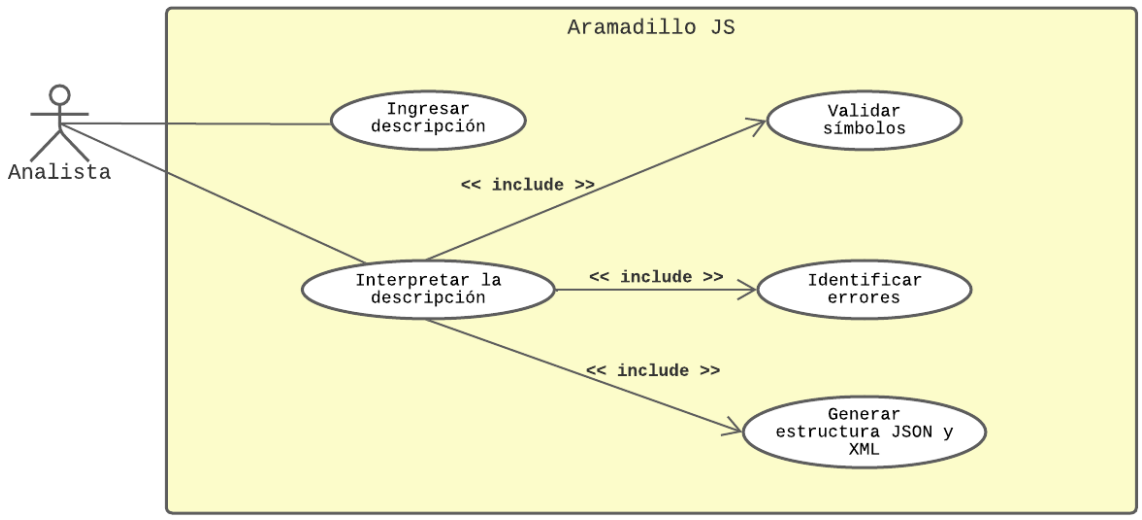
FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

4.1.2. Modelamiento

Se utilizó el lenguaje de programación JavaScript con la intención de generar un paquete totalmente exportable a otros proyectos que requieran utilizar la librería para generar sus propios diagramas con otras librerías de dibujo. Para empezar con el diseño o modelamiento de la librería se especificó el proceso normal que deberá seguir la al momento de recibir como datos de entrada las descripciones de los casos de uso. En la ilustración 4.12 se observa el proceso que se llevará a cabo de forma general la ejecución de la librería.

En la ilustración 4.12 se puede observar el diagrama de casos de uso que explica las acciones que el analista podrá realizar al momento de implementar Armadillo.

Ilustración 4.12: *Diagrama de casos de uso armadillo.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

A continuación, se detallarán las descripciones de los casos de uso que se observan en la imagen anterior. Se especificarán todos los pasos que la librería debe recibir para responder de forma correcta. Se recuerda que Armadillo no cuenta con acceso a datos o algún tipo de información en la nube.

En la tabla 4.11 se detallan los pasos que se siguen al momento de ingresar las descripciones de los casos de uso que se pretenden interpretar.

Tabla 4.10: *Descripción del caso de uso para interpretar la descripción.*

Caso de uso:	Interpretar descripción
Actores:	Analista
Precondición:	Tener ingresada la descripción a interpretar redactada con el lenguaje de símbolos.
Postcondición:	Obtener la descripción redactada en lenguaje natural.
Propósito:	Obtener la descripción del caso de uso e interpretar los símbolos del lenguaje.
Resumen:	Permite identificar cada símbolo del lenguaje para identificar los objetos que se deben crear en el diagrama de clases.
Tipo:	Primario
Flujo normal	
Acción del actor	**Respuesta del sistema**
1. Este caso de uso inicia cuando el analista pretende interpretar la descripción del caso de uso.	
2. El analista da la orden de interpretar la descripción con el lenguaje de símbolos.	
	3. Muestra dentro de un apartado denominado consola, todos los procesos que realizaron al momento de interpretar la descripción. Además de mostrar rápidamente el texto redactado en lenguaje normal.
4. El analista se dirige a la pestaña de "Class diagram".	
	5. Muestra el grafico del diagrama de clases generado por la librería. Se dibujan los objetos que pudieron ser identificados por los símbolos usados.
6. El analista se dirige a la pestaña de "JSON".	
	7. Muestra la estructura JSON generada.
8. El analista se dirige a la pestaña de "XML".	
	9. Muestra la estructura XML generada.
10. Este caso de uso finaliza cuando el analista descarga las estructuras JSON o XML para posteriores proyectos.	

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.11: *Descripción del caso de uso para ingresar la descripción.*

Caso de uso:	Ingresar descripción
Actores:	Analista
Precondición:	Tener instanciada la librería en su proyecto web, o debe utilizar la aplicación de demostración.
Postcondición:	Descripción ingresada en la librería.
Propósito:	Ingresar un párrafo de la descripción del caso de uso.
Resumen:	Permite ingresar un párrafo de toda la descripción del caso de uso utilizando el lenguaje de símbolos.
Tipo:	Primario
Flujo normal	
Acción del actor	**Respuesta del sistema**
1. Este caso de uso inicia el analista ingresa la descripción del caso de uso.	
	2. Muestra la caja de texto donde se debe colocar la descripción del caso de uso con el lenguaje de símbolos.
3. Este caso de uso finaliza cuando el analista ingresa la descripción del caso de uso.	

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.12: *Descripción del caso de uso para generar estructura json y xml.*

Caso de uso:	Generar estructura JSON y XML
Actores:	Analista
Precondición:	La descripción del caso de uso debe contar con SymLen.
Postcondición:	Todos los símbolos serán omitidos en el texto interpretado.
Propósito:	Generar 2 estructuras con la información del diagrama de clases.
Resumen:	Permite obtener toda la estructura del diagrama de clases generado en formato JSON Y XML
Tipo:	Secundario
Flujo normal	
Acción del actor	**Respuesta del sistema**
	1. Este caso de uso inicia cuando la descripción del caso fue ingresada con los símbolos del lenguaje.
	2. Buscar los objetos que fueron identificados por los símbolos.
	3. Este caso de uso finaliza cuando se muestra el json y xml del diagrama de clases

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.13: *Descripción del caso de uso para validar símbolos.*

Caso de uso:	Validar símbolos
Actores:	Analista
Precondición:	La descripción del caso de uso debe contar con los símbolos del lenguaje.
Postcondición:	Todos los símbolos serán omitidos en el texto que sea interpretado.
Propósito:	Identificar los objetos para generar el diagrama de clases
Resumen:	Permite identificar los símbolos que son usados para redactar los casos de uso. Además de verificar si se los esta usando de forma correcta.
Tipo:	Secundario
Flujo normal	
Acción del actor	**Respuesta del sistema**
	1. Este caso de uso inicia cuando la descripción del caso fue ingresada con los símbolos del lenguaje.
	2. Se identifican los tipos de símbolos que tiene el lenguaje, si son símbolos con apertura y cierre o son individuales.
	3. Se verifica que los símbolos están usándose de forma correcta sin combinarlos con otros símbolos desconocidos.
	4. Este caso de uso finaliza cuando se identificaron todos los símbolos de la descripción y oculta los símbolos para obtener un texto natural.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.14: *Descripción del caso de uso para identificar errores.*

Caso de uso:	Identificar errores
Actores:	Analista
Precondición:	La descripción del caso de uso debe contar con los símbolos del lenguaje.
Postcondición:	Todos los símbolos serán omitidos en el texto que sea interpretado.
Propósito:	Permite identificar la mala escritura de las descripciones de los casos de uso.
Resumen:	Identificar los posibles errores que se pueden cometer al momento de usar los símbolos en las descripciones de los casos de uso.
Tipo:	Secundario

Flujo normal	
Acción del actor	**Respuesta del sistema**
	1. Este caso de uso inicia cuando la descripción del caso fue ingresada con los símbolos del lenguaje.
	2. Se identifican los tipos de símbolos que tiene el lenguaje, si son símbolos con apertura y cierre o son individuales.
	3. Si existen inconsistencias en la redacción de la descripción del caso de uso, se empieza acumular mensajes de alerta.
	4. Este caso de uso finaliza mostrando los mensajes en un apartado y clasificando los mensajes acumulados mediante un color diferentes: Azul para información, amarillo para advertencia, verde para exitoso y rojo para notificar errores graves.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

4.1.3. Generación de código

En la siguiente fase se desarrollaron las funciones y todos los métodos que fueron necesarios para que la librería funcione correctamente. A continuación, se observan todas las variables que fueron necesarias para que todo funcione de manera correcta.

```
1  // Global variables
2  var DataPrimitive_Armadillo = ["String", "string", "byte", "short",
3  "int", "Int", "long", "Long", "float", "Float", "double", "Double",
4  "boolean","Boolean", "date", "Date", "list", "List", "array",
5  "Array", "arrayList", "ArrayList"];
6  var DataTypeVisibility = [
7      {simbol: "+", text: "public"},
8      {simbol: "-", text: "private"},
9      {simbol: "#", text: "protected"}
10 ];
11 var DataTypeModifiers = [
12     {simbol: "$", text: "static"},
13     {simbol: "?", text: "abstract"},
14     {simbol: "_", text: "final"},
```

```
    {simbol: "@", text: "interface"}
];
var DataTypeClass = [
    {simbol: "@", text: "interface"}
    , {simbol: "?", text: "abstract"}
    , {simbol: "_", text: "final"}
];
var indexDataType_Armadillo = 0;
var DataType_Armadillo = [];
var notifications = [];
DataType_Armadillo[indexDataType_Armadillo] =
DataPrimitive_Armadillo.slice();
```

En el siguiente bloque de código se observa la función que permite acumular los mensajes de error que pueden ocurrir al momento de interpretar las descripciones de los casos de uso.

```
/**
 * @function Function for adding notification messages
 * @param {number} status
 * @param {string} information
 * @param {array} data
 * */
function setNotifications(status, information, data) {
    let notification =
    {status: 0, information: "Thank you for using armadillo.js",
    type: "", data: []};
    notification.status = status;
    notification.information = information;
    notification.data = data;
    notification.type =
    status === 0 ? "information" : status === 1 ? "success"
        : status === 2 ? "warning" : "error";
    console.log(notification);
    notifications.push(notification);
}
```

En el siguiente bloque de código se observa una de las funciones principales para detectar los símbolos de apertura y cierre, con el objetivo de utilizar una sola función que permita identificar los símbolos que necesariamente deben ser con apertura y cierre.

```
/**
 * @function To validate the start and end character each use case
 * to be interpreted.
 * @param {character} beging
 * @param {character} end
 * @param {string} characters
 * */
 function manualPartition(beging, end, characters) {
    var subs = {x1: 0, x2: 0};
    var elements = []; var open = true;
    let countBegin = count(characters, beging);
    let countEnd = count(characters, end);
    if (countBegin === countEnd) {
        for (var rec = 0; rec < characters.length; rec++) {
            if (characters[rec] === beging && open === true) {
                subs.x1 = rec;
                open = false;
                if (characters[rec - 1] === "*"
                && characters[rec] === beging) {
                  subs.x1 = rec - 1;
                }
            }
            if (characters[rec] === end
              && open === false && subs.x1 !== rec) {
                open = true;
                subs.x2 = rec;
                var resultSelect =
                characters.toString()
                .substr(subs.x1, (subs.x2 - (subs.x1 - 1)));
                elements.push(resultSelect.toString());
            }
        }
    } else {
      setNotifications(3,
        "An opening but not a closing symbol was detected. => begin: "
        +beging+", end: "+end+". Near: "+characters+"", []);
    }
    return elements;
}
```

Para mejorar la presentación de la librería se desarrolló una aplicación web sencilla que permita utilizar de forma gráfica las funciones de esta. La aplicación web tendrá como propósito proporcionar un lugar en la web de donde descargar el script para que pueda ser usado por la comunidad de desarrolladores. Además, de proporcionar documentación de como instalarla en otros proyectos y como utilizarla.

4.1.4. Ejecución de pruebas

Para la ejecución de las pruebas, se analizó el caso de uso que fue proporcionado por el docente. Cada párrafo estaba redactado con el lenguaje de símbolos, por lo que se ingresaron las descripciones en la aplicación de demostración y los resultados fueron los siguientes:

- Descripción #1:

```
1    Este caso de uso comienza cuando una *(/persona/ &-id=int)
     quiere registrarse como usuario *(/tutor/ &- id=int [+
     usuarioRegistrar=Tutor]) en el sistema. *¡tutor 1<[registro]>n
     Persona!
```

Interpretando el párrafo, se creó una clase denominada Tutor con sus respectivos atributos que son: **id** privado y de tipo String, los métodos que se agregaron son: **usuarioRegistrar()** público. Además, se realizó una relación de tipo generalización. A continuación, se observa la estructura json y xml generada por la librería a partir del párrafo ingresado.

```
1  {
2    "diagram": [
3      {
4        "packName": "Default",
5        "class": [
6          {
7            "action": "update",
8            "derivative": [],
9            "className": "Persona",
10           "visibility": "public",
11           "modifiers": "",
12           "attributes": [
```

```
            {
              "visibility": "private",
              "name": "id",
              "type": "Int"
            }
          ],
          "methods": [],
          "constructors": []
        },
        {
          "action": "update",
          "derivative": [],
          "className": "Tutor",
          "visibility": "public",
          "modifiers": "",
          "attributes": [
            {
              "visibility": "private",
              "name": "id",
              "type": "String"
            }
          ],
          "methods": [
            {
              "visibility": "public",
              "name": "usuarioRegistrar",
              "type": "void",
              "parameters": []
            }
          ],
          "constructors": []
        }
      ],
      "enums": []
    }
  ],
  "relationships": [
    {
      "from": "Tutor",
```

```
      "to": "Persona",
      "typeRelatioship": "generalization",
      "value": "registrar",
      "cardinalidate": "1..*",
      "from_fk": " Persona:Persona[]",
      "to_fk": "tutor :Tutor",
      "simbol": "1<>n"
    }
  ]
}
```

```
<package>
  <diagram>
    <packName>Default</packName>
    <class>
      <action>update</action>
      <className>Persona</className>
      <visibility>public</visibility>
      <modifiers></modifiers>
      <attributes>
        <visibility>private</visibility>
        <name>id</name>
        <type>Int</type>
      </attributes>
    </class>
    <class>
      <action>update</action>
      <className>Tutor</className>
      <visibility>public</visibility>
      <modifiers></modifiers>
      <attributes>
        <visibility>private</visibility>
        <name>id</name>
        <type>String</type>
      </attributes>
      <methods>
        <visibility>public</visibility>
        <name>usuarioRegistrar</name>
        <type>void</type>
```

```
      </methods>
    </class>
  </diagram>
  <relations>
    <from>Tutor</from>
    <to>Persona</to>
    <typeRelatioship>generalization</typeRelatioship>
    <value>registrar</value>
    <cardinalidate>1..*</cardinalidate>
    <from_fk> Persona:Persona[]</from_fk>
    <to_fk>tutor :Tutor</to_fk>
    <simbol>
      1<>n
    </simbol>
  </relations>
</package>
```

Para que la descripción del caso de uso pueda ser entendida por el cliente del sistema que se esté desarrollando se obtiene como resultado el párrafo redactado en lenguaje natural omitiendo los símbolos y pueda ser comprendido con mayor facilidad.

```
    Este caso de uso comienza cuando una persona quiere registrarse
   como usuario tutor en el sistema.
```

Como el texto está redactado de forma correcta la retroalimentación generada muestra mensajes exitosos:

información: Descripción ingresada: Este caso de uso comienza cuando una *(/persona/ &-id=int) quiere registrarse como usuario *(/tutor/ &- id=int [+usuarioRegistrar=Tutor]) en el sistema. *¡tutor 1<[registro]>n Persona!

éxito: Atributos añadidos con éxito: id

éxito: La clase Persona se ha generado con éxito

éxito: Atributos añadidos con éxito: id

éxito: La clase Tutor se ha generado con éxito

éxito: La relación de generalización entre los objetos de: Tutor 1<>n to: Persona se ha generado con éxito.

Para verificar el funcionamiento de la retroalimentación identificando la mala escritura de las descripciones del caso de uso usando el lenguaje de símbolos, se redactó a

propósito el mismo párrafo usando de forma incorrecta los símbolos.

```
1    Este caso de uso comienza cuando una *(/persona/ &-id=int)
   quiere registrarse como usuario *(/tutor/ &- id=int [+
   usuarioRegistrar=Tutor]) en el sistema. *¡tutor 1<[registro]>n
   Persona
```

Observando los menajes de retroalimentación ya se identificaron que errores se están cometiendo en el uso de algunos símbolos y especificando en qué lugar del texto se encuentra ese error.

información: Descripción ingresada: Este caso de uso comienza cuando una *(/persona/ &-id=int) quiere registrarse como usuario *(/tutor/ &- id=int [+usuarioRegistrar=Tutor]) en el sistema. *¡tutor 1<[registro]>n Persona

éxito: Atributos añadidos con éxito: id

éxito: La clase Persona se ha generado con éxito

éxito: Atributos añadidos con éxito: id

error: Se ha detectado un símbolo de apertura pero no de cierre. => inicio: [, fin:]. Cerca de: *(/tutor/ &- id=int [+usuarioRegistrar=Tutor)

éxito: La clase Tutor se ha generado con éxito

error: Se ha detectado un símbolo de apertura pero no de cierre. => begin: ¡, fin: !. Cerca: Este caso de uso comienza cuando una persona *(persona &-id=int) quiere registrarse como usuario tutor *(tutor &-id=int [+userRegistration=Tutor) en el sistema. *¡tutor 1<[registro]>n Persona

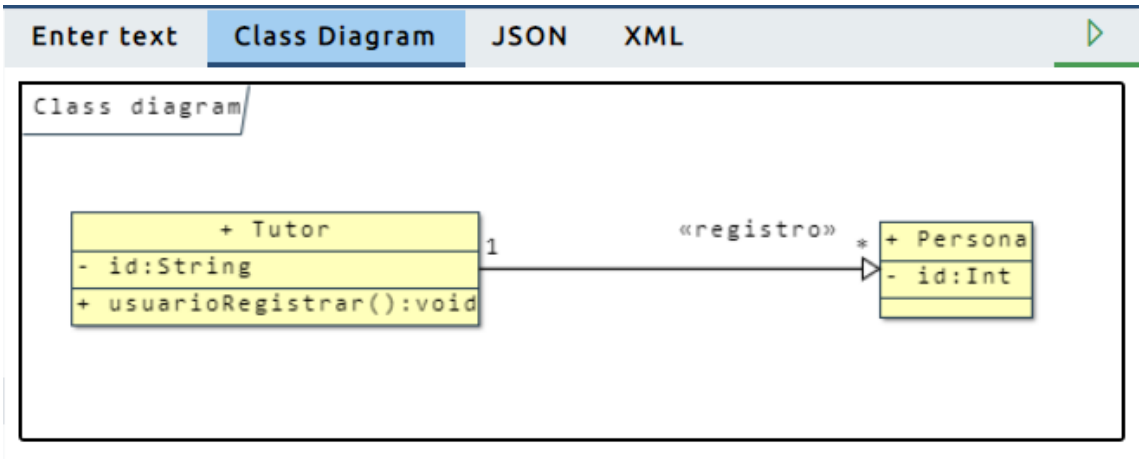
Utilizando la aplicación web de demostración, se logra observar ene la ilustración 4.13 el diagrama de clases creado a partir de la estructura json o xml que la librería proporciona de forma automática. Para dibujar el diagrama de clases, se está utilizando la librería JavaScript jsUml2.

- **Descripción #2:**

```
1    La persona accede entonces a la interfaz de Registro de usuarios
    *(@Registro de Usuarios). *¡Persona1<[accede]<1 Registro de
   Usuarios!
```

A continuación, se evidenciará como se va modificando el diagrama de clases por cada descripción del caso de uso que se va interpretando hasta llegar al diagrama de clases

Ilustración 4.13: *Diagrama de clases generado por la aplicación web de demostración usando la librería de armadillo.js, descripción 1*

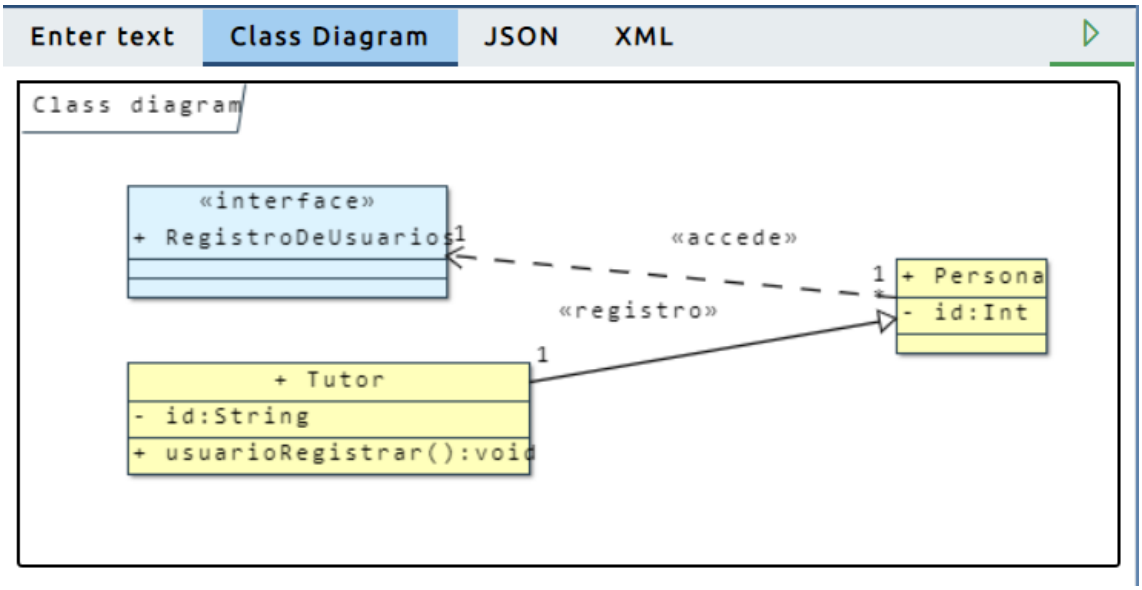


FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

final. En esta descripción se generó un nuevo objeto para el diagrama que es una interfaz denominada Registro de usuarios que se relaciona mediante dependencia a la clase Persona (ver ilustración 4.14).

```
1     La persona accede entonces a la interfaz de Registro de usuarios
```

Ilustración 4.14: *Diagrama de clases generado por la aplicación web de demostración usando la librería de armadillo.js, descripción 2*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

- **Descripción #3:**

```
1    Muestra los campos a rellenar en la interfaz de *(@/Registro de
     usuarios/): *(Persona &-/nombre/=String, &- apellido/=String, &-/
     fecha de nacimiento/=Date, &-/género/=String, &-/número de telé
     fono/=String, &-/email/=String) *(Usuario &- id=Int, &-/
     nombreusuario/=String, &-/clave/=String &-estado=Status &- tipo
     de usuario=UserType). Hay que tener en cuenta que el registro de
     una *¡/Persona/ u<[/tiene un/]>u /Usuario/!. Hay estados que
     pueden ser *¿Estado"/Deshabilitado/ /o/ "/Habilitado/? que cada *
     ¡/Usuario/ u>[/tendrá un/]<u /Estado/! . También hay varios tipos
      de usuarios, que son *¿Tipo de usuario"/Tutor/ /o/ "/Paciente/ /
     o/ "/Admin/? y cada *¡/Usuario/ u>[/tendrá un/]<u /Tipo de
     usuario/! Además, el sistema asigna un identificador al
     almacenarlo.
```

En esta descripción se detallaron más objetos del diagrama de clases. Se observa que se agregaron más atributos a la clase Persona, se agregaron 2 objetos enum que se denominan Estado y Tipo de usuario, aparte que están relacionados con la clase Usuario que también fue agregada en esta descripción.

```
1    Muestra los campos a rellenar en la interfaz de Registro de
     usuarios: nombre nombreusuario clave. Hay que tener en cuenta que
      el registro de una Persona tiene un Usuario. Hay estados que
     pueden ser Deshabilitado o Habilitado que cada Usuario tendrá un
     Estado . También hay varios tipos de usuarios, que son Tutor o
     Paciente o Admin y cada Usuario tendrá un Tipo de usuario Además,
      el sistema asigna un identificador al almacenarlo.
```

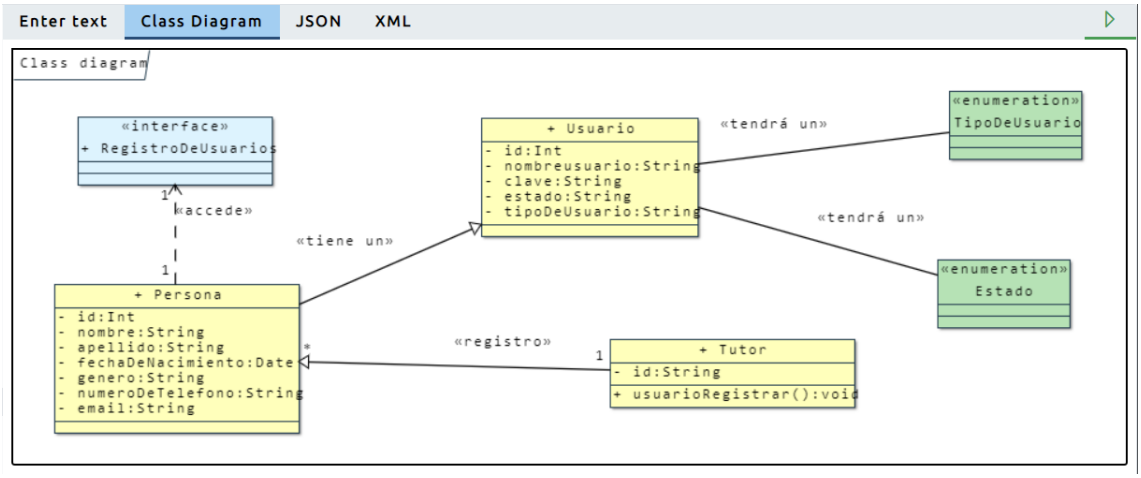
- **Descripción #4:**

```
1    Crear un objeto *(/persona/ %.nombre=String, .apellido=String,
     .fechacumpleanios=Date, .genero=String, .celular=String, .correo=
     String%).
```

En esta descripción se agregó un nuevo constructor para la clase Persona con los siguientes parámetros: nombre, apellido, fechacumpleanios, genero, numero de celular y correo.

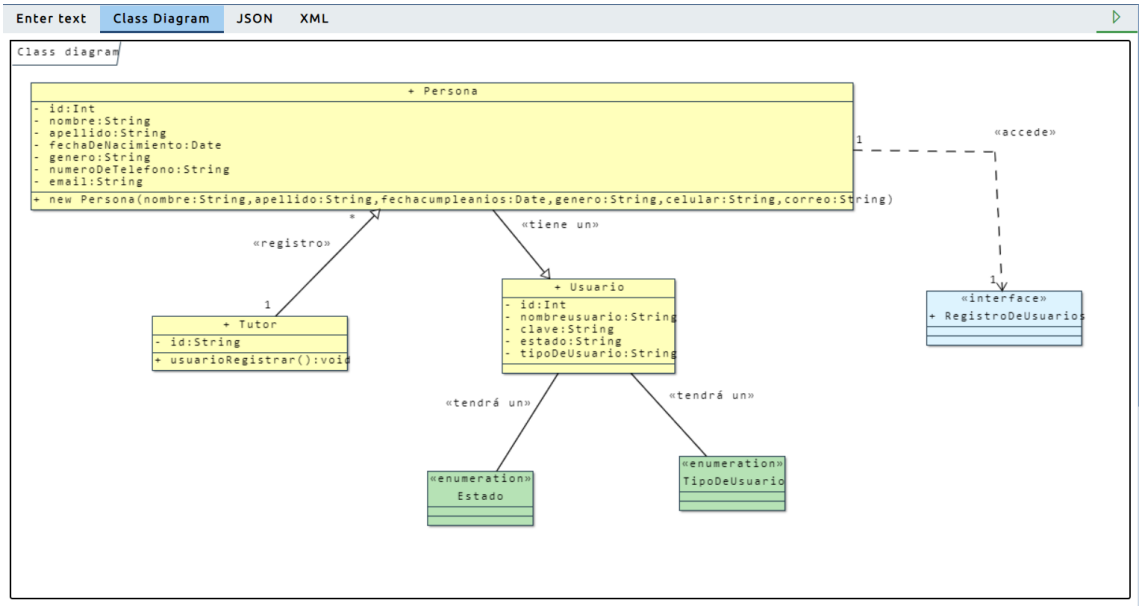
```
1    Crear un objeto persona.
```

Ilustración 4.15: *Diagrama de clases generado por la aplicación web de demostración usando la librería de armadillo.js, descripción 3*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.16: *Diagrama de clases generado por la aplicación web de demostración usando la librería de armadillo.js, descripción 4*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

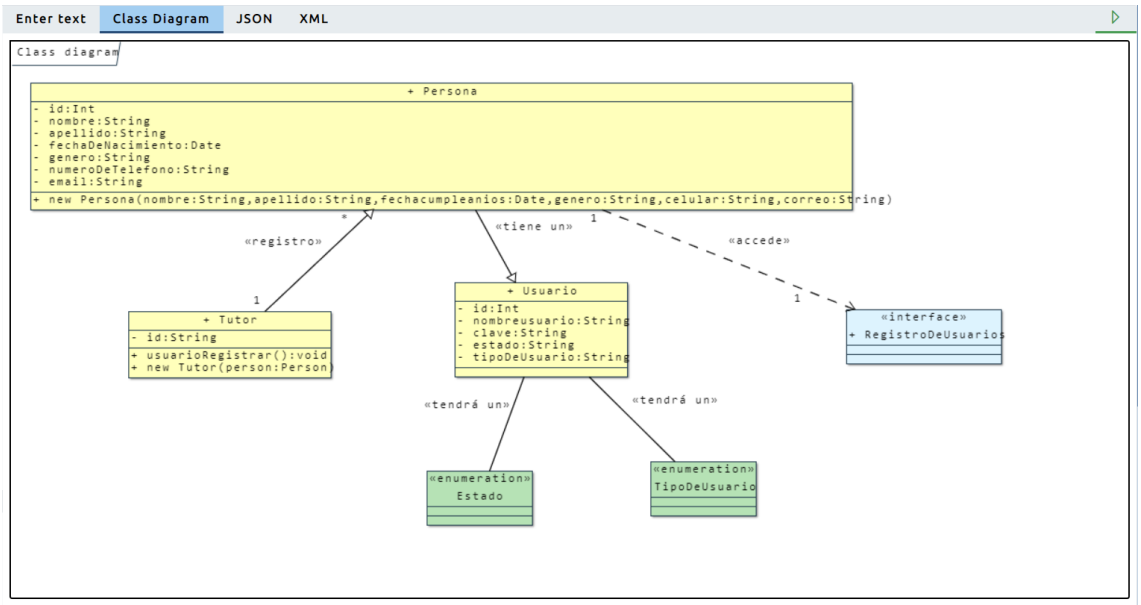
- **Descripción #5:**

```
1    Crear un objeto *(/tutor/%.person=Person%)
```

En la siguiente descripción se agregó un nuevo constructor a la clase Tutor que recibe como parámetro un objeto de la clase Person.

```
1    Crear un objeto tutor
```

Ilustración 4.17: *Diagrama de clases generado por la aplicación web de demostración usando la librería de armadillo.js, descripción 5*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

- **Descripción #6**

```
1    Crear un objeto *(/usuario/%.nombreusuario=String, .clave=
     String, .estado=Status, .tipodeusuario=UserType, .persona=Tutor%)
```

En la descripción se observa que se agregó un nuevo constructor a la clase User con sus parámetros que son: nombreusuario, clave, estado, tipodeusuario y persona que es un objeto Tutor.

```
1    Crear un objeto usuario
```

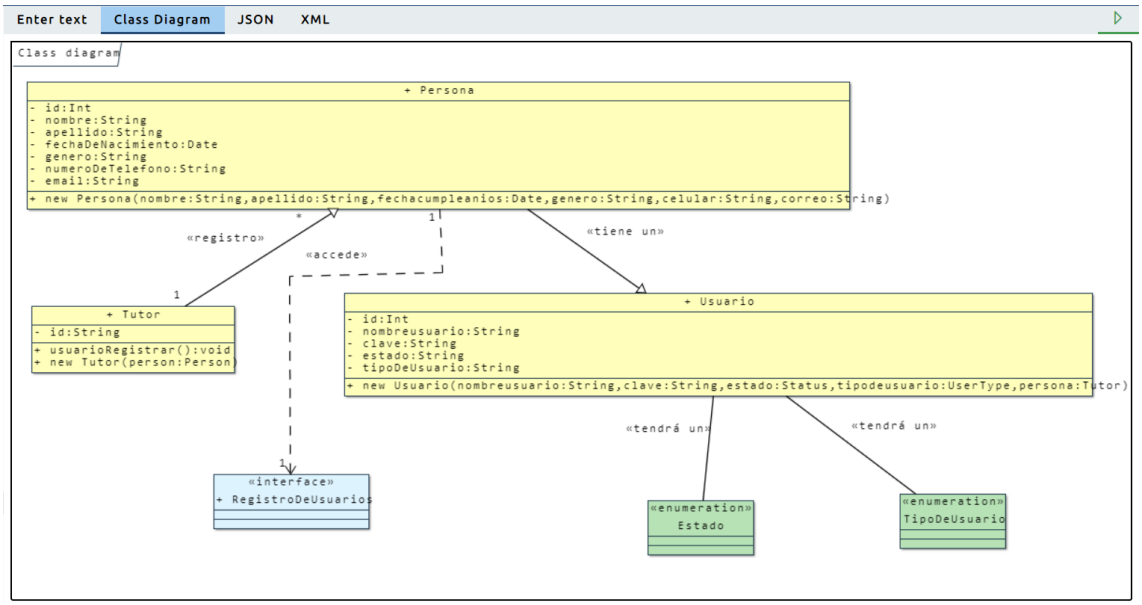
- **Descripción #7:**

```
1    *(TutorDAO &id=Int [/Guarda/=Tutor.tutor=Tutor])r el tutor en la
     base de datos *¡TutorDAO<[]< tutor!.
```

En la siguiente descripción se observa que se agregó una nueva clase que se denomina TutorDAO. Se agregó un método que se llama Guardar que recibe como parámetro un objeto de tipo Tutor y retorna un objeto Tutor mismo. Además, se generó una relación de tipo Dependencia con la clase Tutor pero sin cardinalidad.

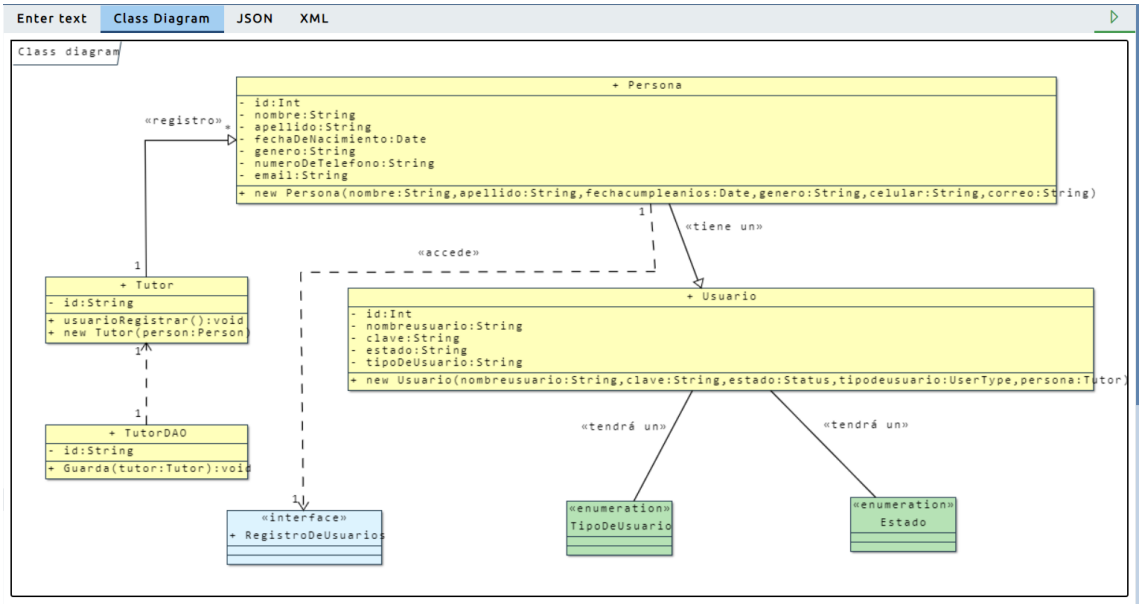
```
1    Guardar el tutor en la base de datos.
```

Ilustración 4.18: *Diagrama de clases generado por la aplicación web de demostración usando la librería de armadillo.js, descripción 6*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.19: *Diagrama de clases generado por la aplicación web de demostración usando Armadillo, descripción 7*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

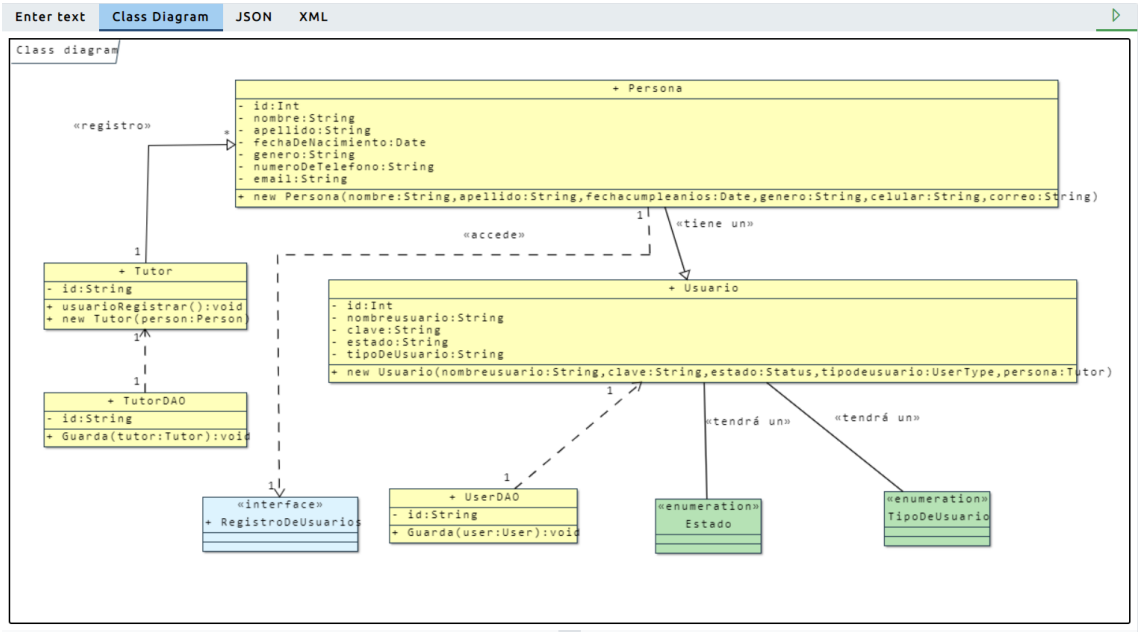
- **Descripción #8:**

```
1   *(UserDAO &-id=Int [/Guarda/=User.user=User])r el usuario en la
     base de datos *¡UserDAO <[]< Usuario!
```

En la siguiente descripción se agregó una clase denominada UserDAO con un método llamado save que recibe como parámetro un objeto de tipo Usuario. Además, se relaciona mediante la relación de tipo Dependencia con la clase Usuario (ver ilustración 4.20).

```
1     Guardar el usuario en la base de datos
```

Ilustración 4.20: *Diagrama de clases generado por la aplicación web de demostración usando la librería de armadillo.js, descripción 8*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

- **Descripción #9:**

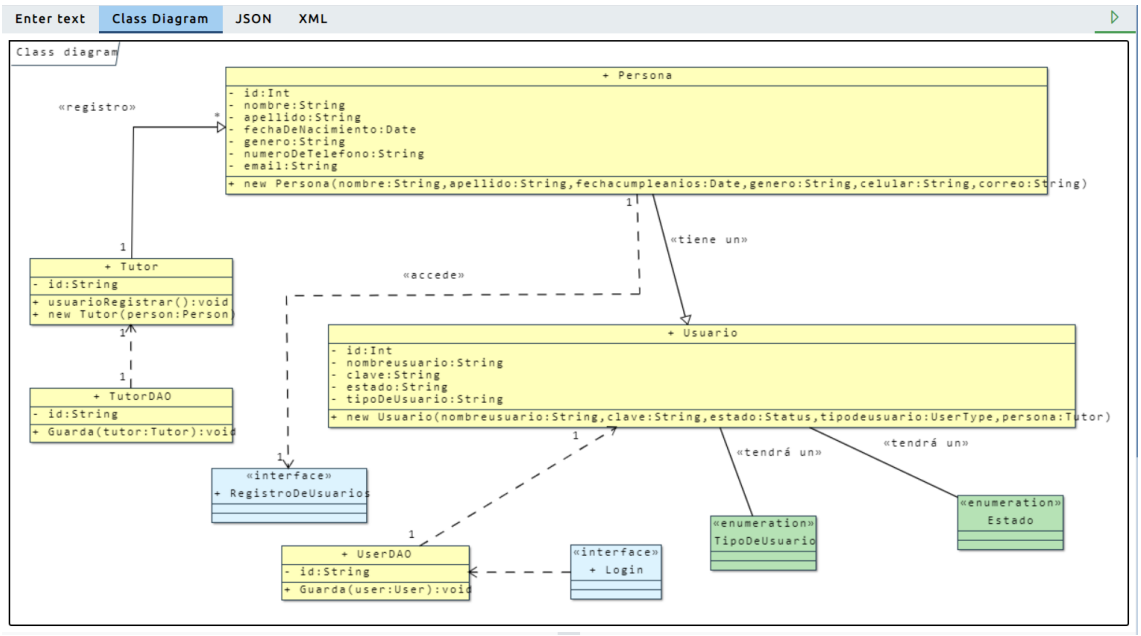
```
1     Este caso de uso finaliza cuando el sistema muestra la interfaz
      de *(@/Login/). *¡Login u<[]<u UserDAO!
```

Finalmente, la descripción del caso de uso genera una interfaz que se denomina Login. Además está relacionada por medio del tipo Dependencia con la clase UserDAO (ver ilustración 4.21).

```
1     Este caso de uso finaliza cuando el sistema muestra la interfaz
      de Login.
```

Si se compara el diagrama de clases que fue recopilado en la primera fase de la metodología con el diagrama de clases generado mediante la librería. Se logra observar que son muy

Ilustración 4.21: *Diagrama de clases generado por la aplicación web de demostración usando la librería de armadillo.js, descripción 9*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

similares, y fue generado mediante las descripciones del caso de uso que fueron redactadas usando el lenguaje de símbolos.

4.1.5. Evaluación con sistemas de información

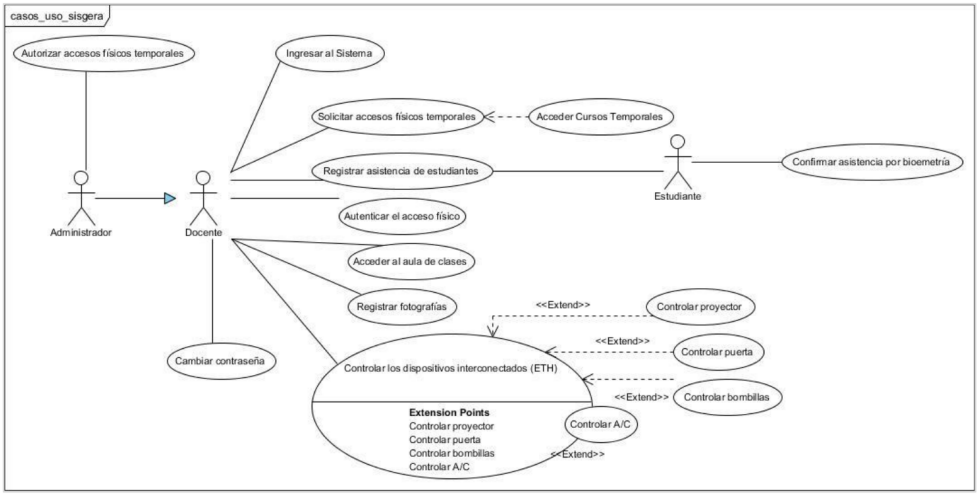
Para la ejecución de la fase final de la metodología se revisaron varios proyectos de titulación realizados por estudiantes de la Universidad Técnica Estatal de Quevedo. Se analizaron los sistemas desarrollados reescribiendo las descripciones de los casos de uso utilizando el lenguaje de símbolos.

En [35] se analizó el proyecto de titulación "SISTEMA BASADO EN INTERNET DE LAS COSAS PARA LA GESTIÓN DE LOS RECURSOS DE UN AULA DE CLASES" que fue realizado en la Universidad Técnica Estatal de Quevedo. Para empezar con el análisis se revisó el diagrama de casos de uso, seleccionando algunos casos de uso para realizar la respectiva evaluación de la librería.

El objetivo de esta prueba es retroalimentar lo que se genera al interpretar cada descripción y comparar el diagrama de clases generado con el original del proyecto. En la ilustración

4.22 se observa el diagrama de casos de uso realizado por el proyecto de titulación. Para la evaluación se seleccionaron los casos: ingresar al sistema, solicitar accesos físicos temporales, acceder a cursos temporales, registrar asistencia de estudiantes.

Ilustración 4.22: *Diagrama de casos de uso para "SISTEMA BASADO EN INTERNET DE LAS COSAS PARA LA GESTIÓN DE LOS RECURSOS DE UN AULA DE CLASES"*



FUENTE: VILLAFUERTE y YÁNEZ [35]

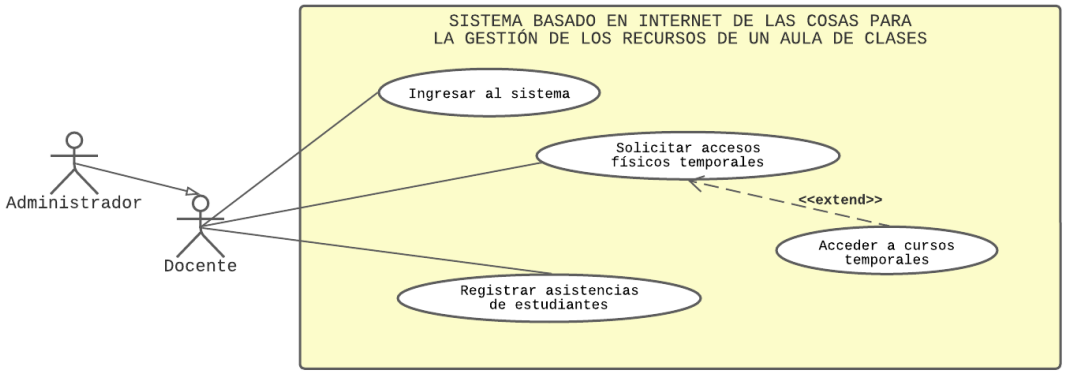
Para mejorar la presentación de los casos de uso a evaluar, se realizó un diagrama de casos de uso con los seleccionados. Además, se redactaron las descripciones de cada caso de uso en 2 formas. La primera es con el texto natural detallando paso a paso las acciones que se deben realizar. La Segunda forma en como están representados los casos de uso es mediante el lenguaje de símbolos.

En la ilustración 4.24 se observa el diagrama de clases para el sistema de información a analizar. En la fase de evaluación de la librería se tratará de generar un diagrama de clases lo más parecido posible al diagrama original. Además de indicar los errores si es que son cometidos al momento de redactar las descripciones de los casos de uso.

A continuación en las tablas [4.15, 4.17, 4.19 4.21] se redactan las descripciones de los casos de uso seleccionados en lenguaje natural. En las tablas [4.16, 4.18, 4.20, 4.22] se redactaron las descripciones de los casos de uso usando el lenguaje de símbolos.

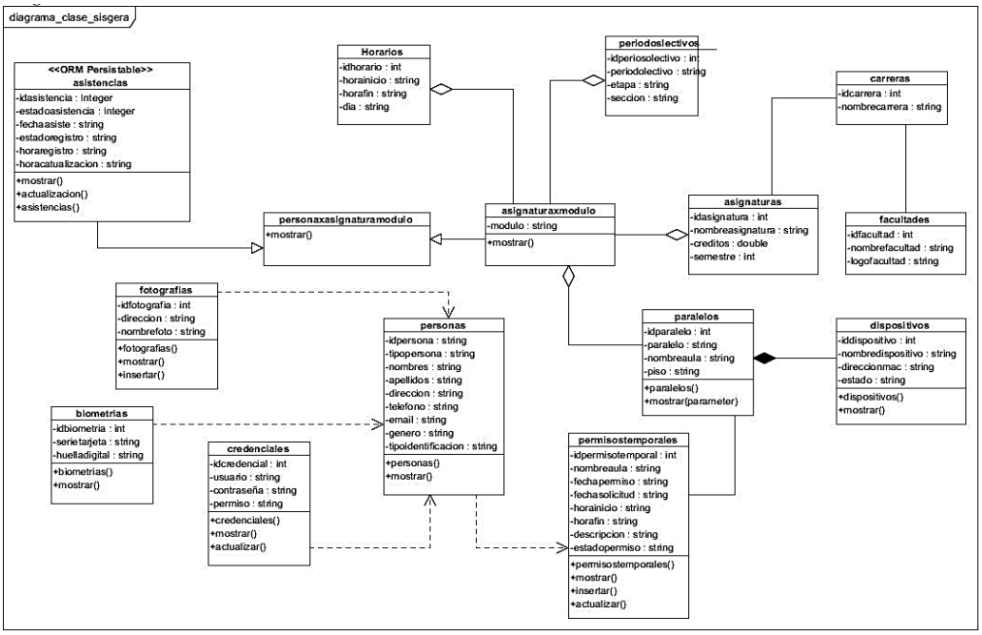
Para evaluar las descripciones de los casos de uso se detallará cada descripción con su respectiva retroalimentación y como se va modificando el diagrama de clases conforme se ingresan más descripciones.

Ilustración 4.23: *Diagrama de casos de uso basado en el "SISTEMA BASADO EN INTERNET DE LAS COSAS PARA LA GESTIÓN DE LOS RECURSOS DE UN AULA DE CLASES" para ser evaluado por armadillo.js*



FUENTE: VILLAFUERTE y YÁNEZ [35]
ELABORADO: AUTOR

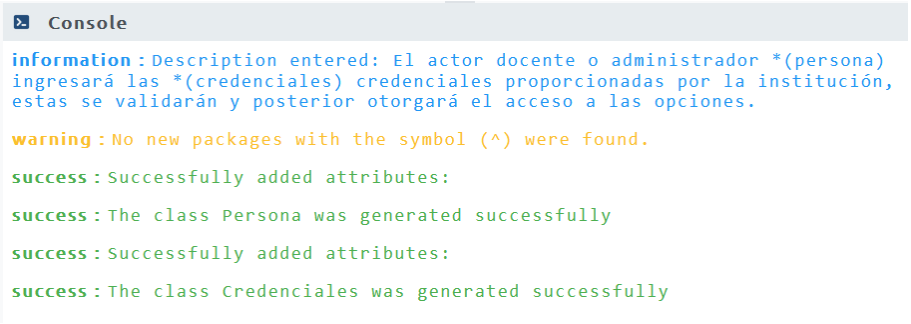
Ilustración 4.24: *Diagrama de clases de el "SISTEMA BASADO EN INTERNET DE LAS COSAS PARA LA GESTIÓN DE LOS RECURSOS DE UN AULA DE CLASES" para ser evaluado por armadillo.js*



FUENTE: VILLAFUERTE y YÁNEZ [35]

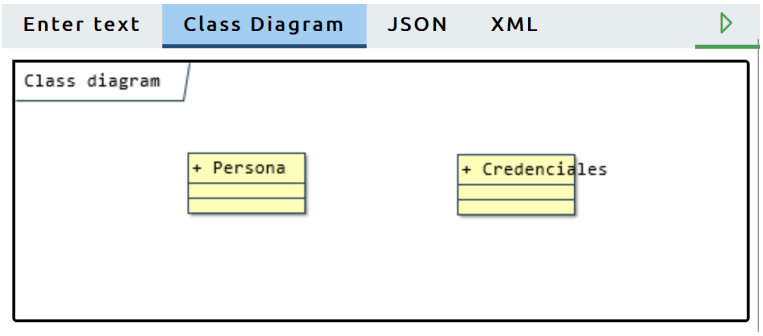
```
1   1. El actor docente o administrador *(persona) ingresará las *(
      credenciales) credenciales proporcionadas por la institución, estas se
       validarán y posterior otorgará el acceso a las opciones.
```

Ilustración 4.25: *Mensajes de consola relacionado a la escritura de la descripción 1.*



ELABORADO: DÚVAL CARVAJAL SUÁREZ

Ilustración 4.26: *Diagrama de clases generado para la descripción 1.*

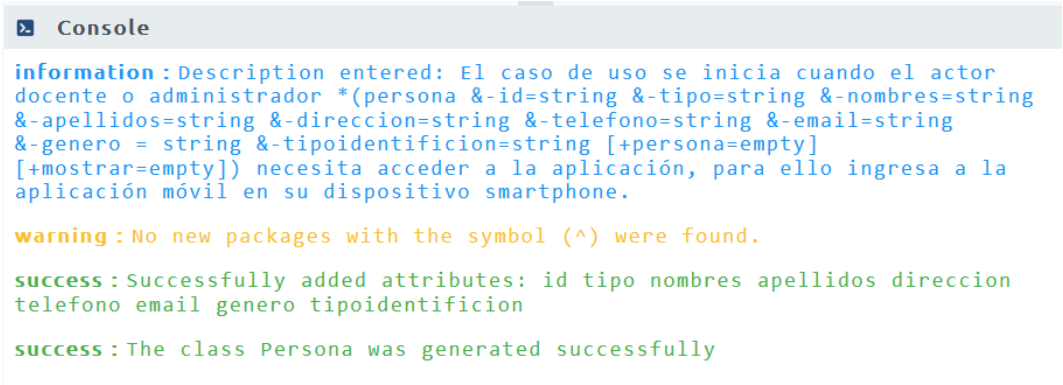


FUENTE:INVESTIGACIÓN

ELABORADO: AUTOR

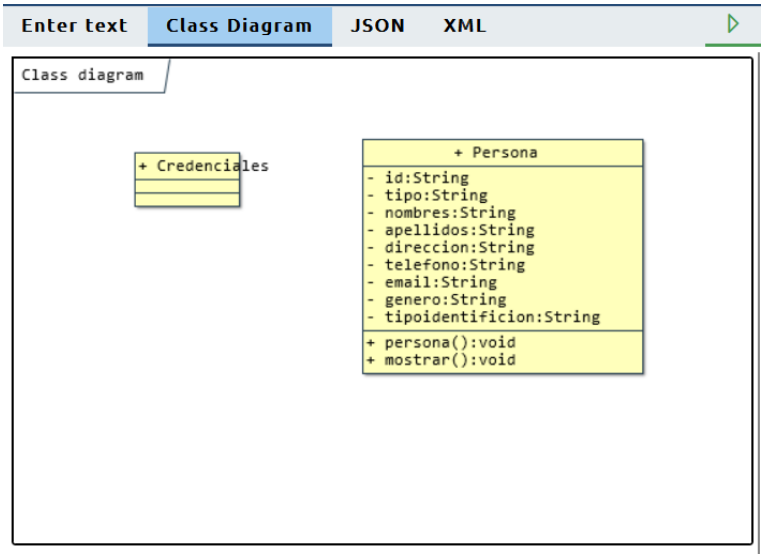
```
1    2. El caso de uso se inicia cuando el actor docente o administrador *(
     persona &-id=string &-tipo=string &-nombres=string &-apellidos=string
     &-direccion=string &-telefono=string &-email=string &-genero = string
     &-tipoidentificion=string [+persona=empty] [+mostrar=empty]) necesita
     acceder a la aplicación, para ello ingresa a la aplicación móvil en su
      dispositivo smartphone.
```

Ilustración 4.27: *Mensajes de consola relacionado a la escritura de la descripción 2.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.28: *Diagrama de clases generado para la descripción 2.*

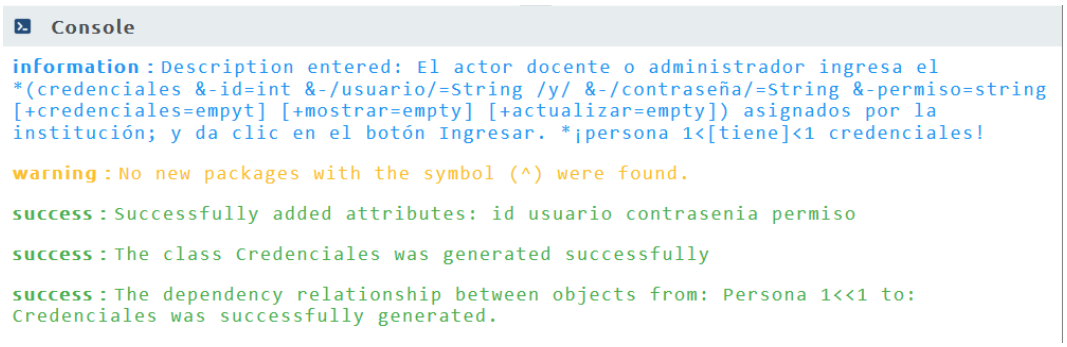


FUENTE:INVESTIGACIÓN

ELABORADO: AUTOR

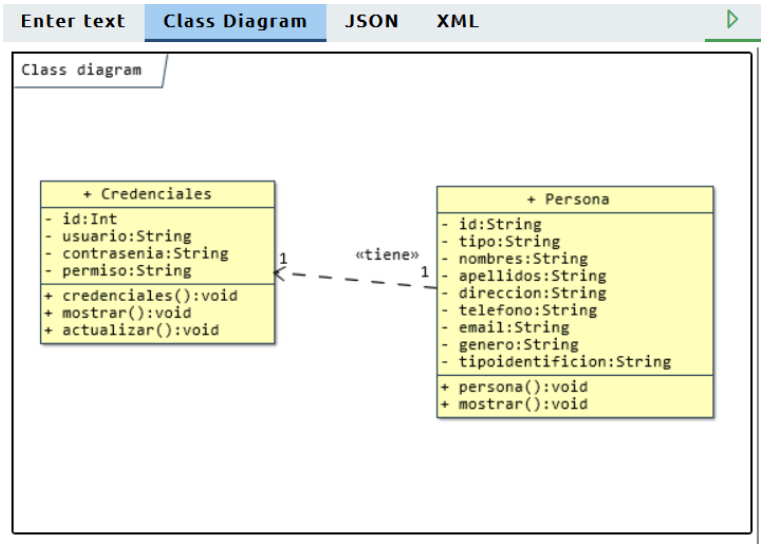
```
1   3. El actor docente o administrador ingresa el *(credenciales &-id=int
    &-/usuario/=String /y/ &-/contraseña/=String &-permiso=string [+
    credenciales=empyt] [+mostrar=empty] [+actualizar=empty]) asignados
    por la institución; y da clic en el botón Ingresar. *¡persona 1<[tiene
    ]<1 credenciales!
```

Ilustración 4.29: *Mensajes de consola notificando si existe algún error en la escritura de la descripción 3.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.30: *Diagrama de clases generado para la descripción 3.*



FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

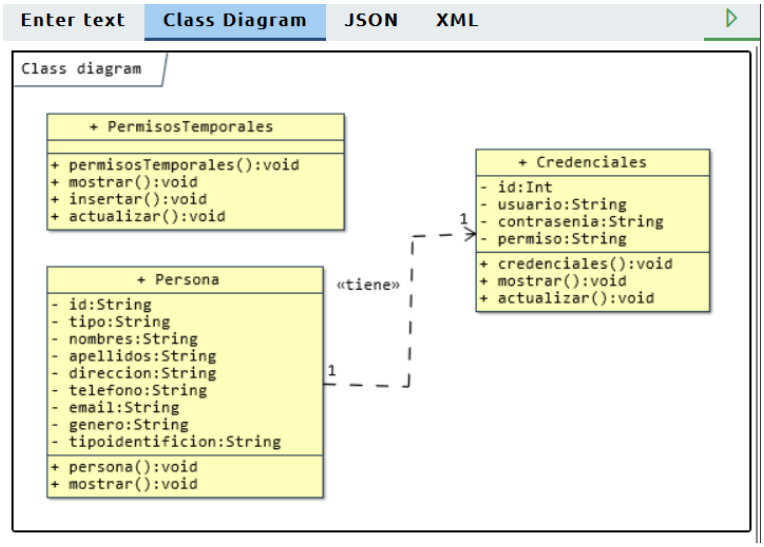
```
1   4. El caso de uso se inicia cuando el docente necesita tener acceso
    temporalmente *(permisos temporales [+permisosTemporales=empyt] [+
    mostrar=empty] [+insertar=empty] [+actualizar=empty]) a un aula de
    clases que se encuentre disponible no utilizada.
```

Ilustración 4.31: *Mensajes de consola relacionado a la escritura de la descripción 4.*

```
Console
information : Description entered: El caso de uso se inicia
cuando el docente necesita tener acceso temporalmente
*(permisos temporales [+permisosTemporales=empyt]
[+mostrar=empty] [+insertar=empty] [+actualizar=empty]) a un
aula de clases que se encuentre disponible no utilizada.

warning : No new packages with the symbol (^) were found.

success : Successfully added attributes:

success : The class PermisosTemporales was generated
successfully
```

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.32: *Diagrama de clases generado para la descripción 4.*

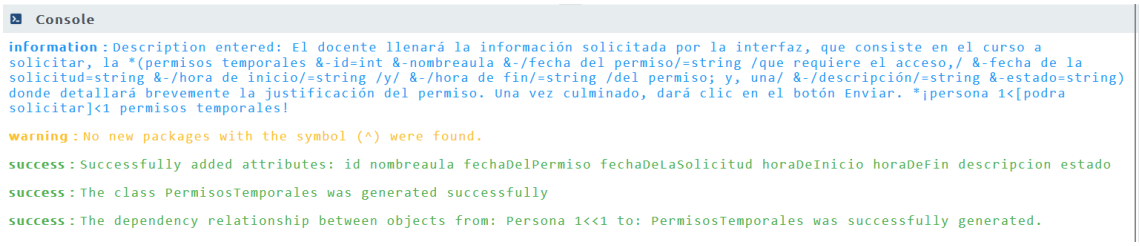


FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

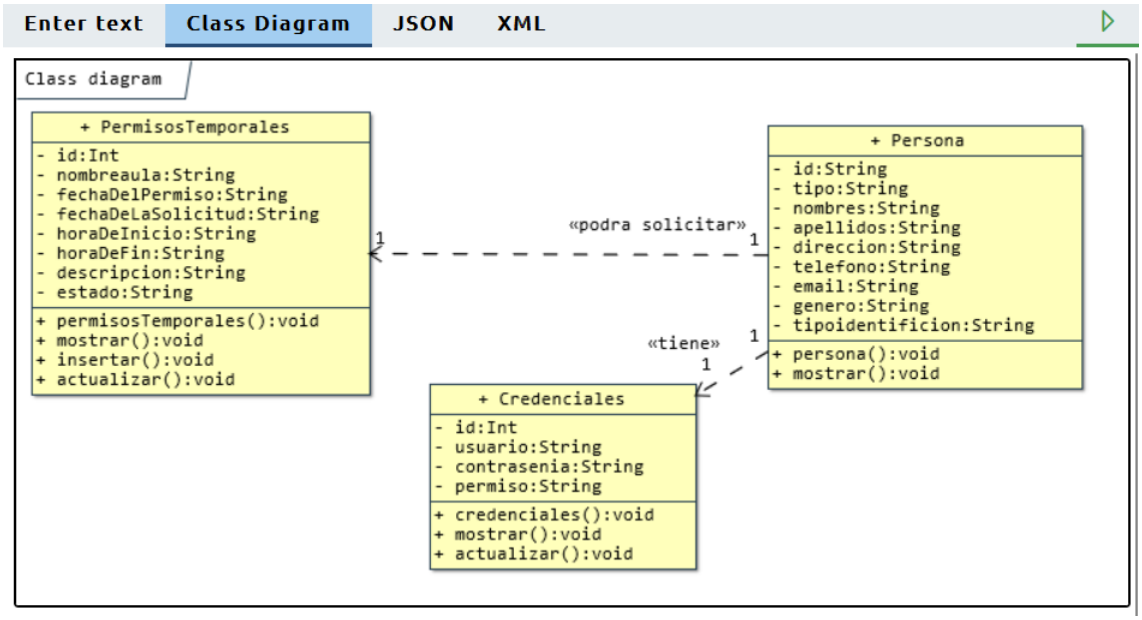
```
1  5. El docente llenará la información solicitada por la interfaz, que
   consiste en el curso a solicitar, la *(permisos temporales &-id=int &-
   nombreaula &-/fecha del permiso/=string /que requiere el acceso,/ &-
   fecha de la solicitud=string &-/hora de inicio/=string /y/ &-/hora de
   fin/=string /del permiso; y, una/ &-/descripción/=string &-estado=
   string) donde detallará brevemente la justificación del permiso. Una
   vez culminado, dará clic en el botón Enviar. *¡persona 1<[podra
   solicitar]<1 permisos temporales!
```

Ilustración 4.33: *Mensajes de consola relacionado a la escritura de la descripción 5.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.34: *Diagrama de clases generado para la descripción 5.*

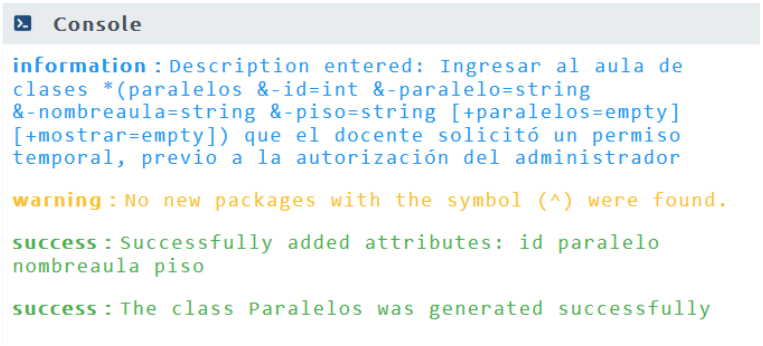


FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

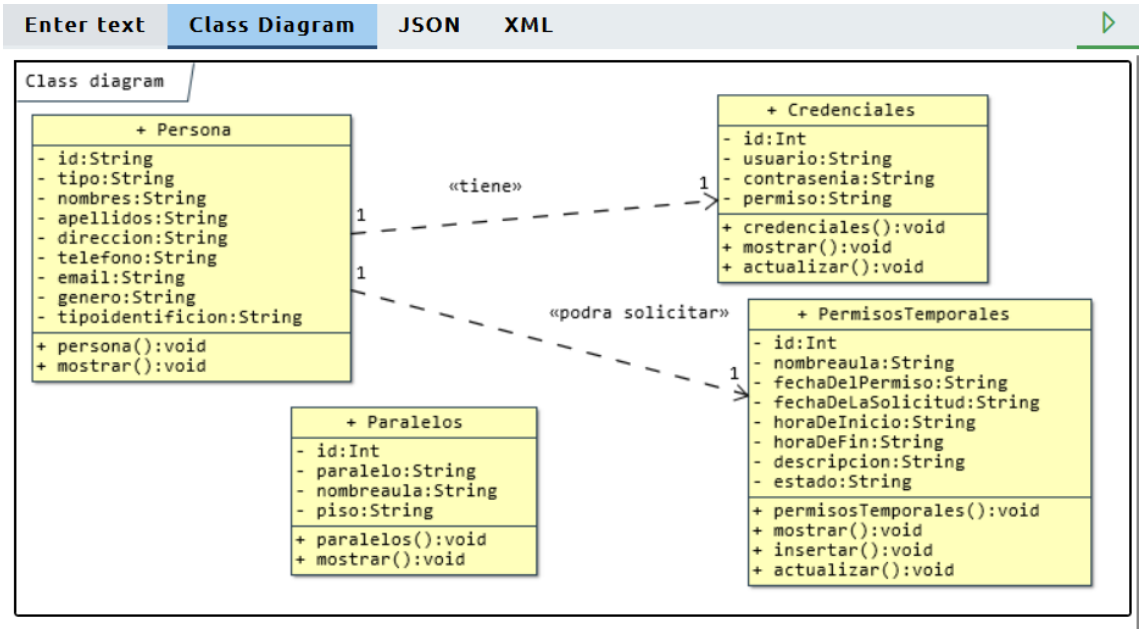
```
1   6. Ingresar al aula de clases *(paralelos &-id=int &-paralelo=string &-
    nombreaula=string &-piso=string [+paralelos=empty] [+mostrar=empty])
    que el docente solicitó un permiso temporal, previo a la autorización
    del administrador
```

Ilustración 4.35: *Mensajes de consola relacionado a la escritura de la descripción 6.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.36: *Diagrama de clases generado para la descripción 6.*

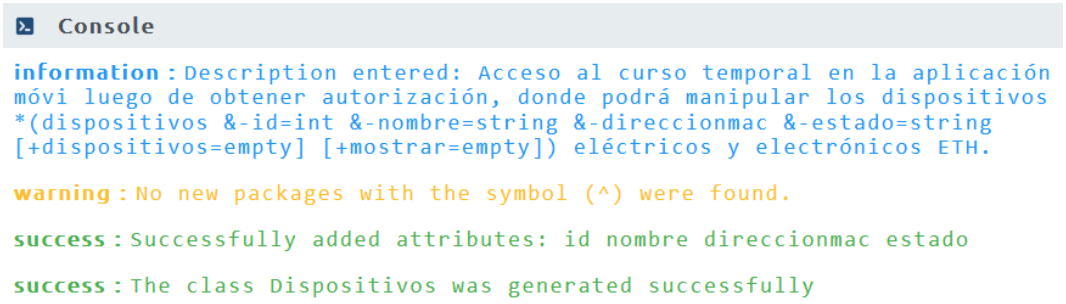


FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

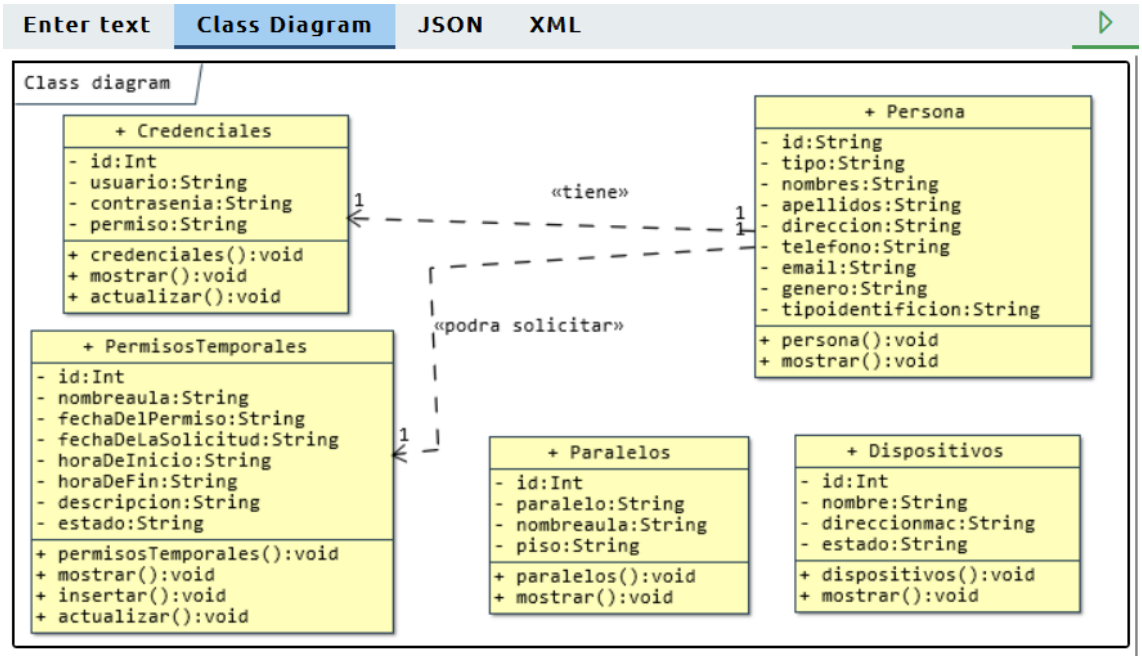
```
1   7. Acceso al curso temporal en la aplicación móvi  luego de obtener
    autorización, donde podrá manipular los dispositivos *(dispositivos &-
    id=int &-nombre=string &-direccionmac &-estado=string [+dispositivos=
    empty] [+mostrar=empty]) eléctricos y electrónicos ETH.
```

Ilustración 4.37: *Mensajes de consola relacionado a la escritura de la descripción 7.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.38: *Diagrama de clases generado para la descripción 7.*

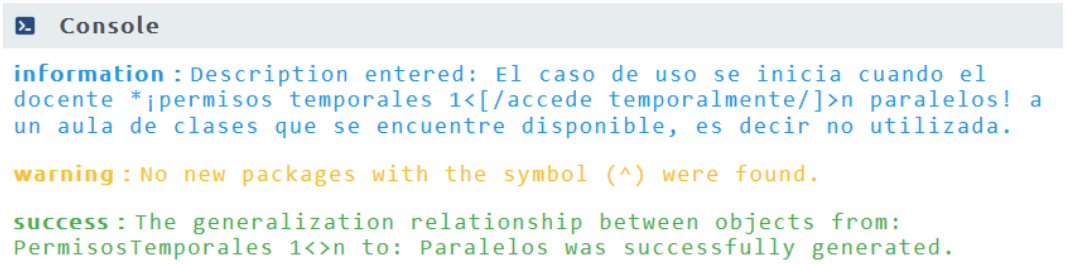


FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

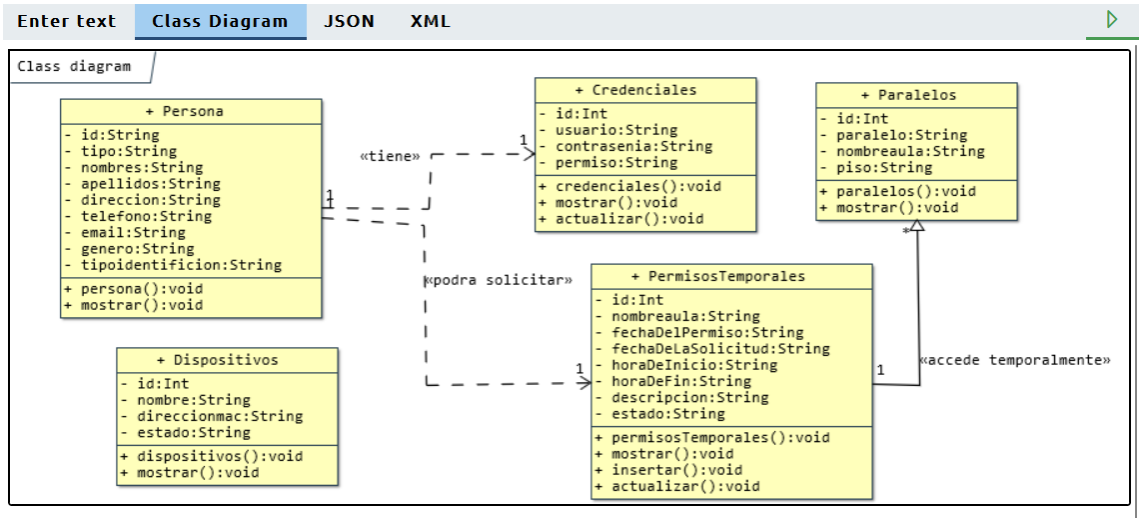
```
1  8. El caso de uso se inicia cuando el docente *¡permisos temporales
     1<[/accede temporalmente/]>n paralelos! a un aula de clases que se
     encuentre disponible, es decir no utilizada.
2
```

Ilustración 4.39: *Mensajes de consola relacionado a la escritura de la descripción 8.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.40: *Diagrama de clases generado para la descripción 8.*

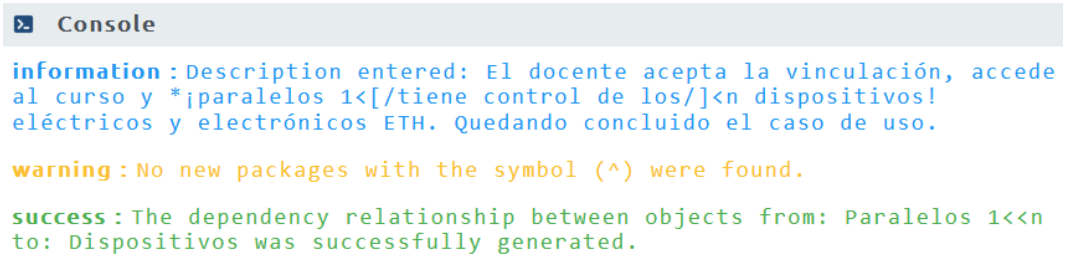


FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

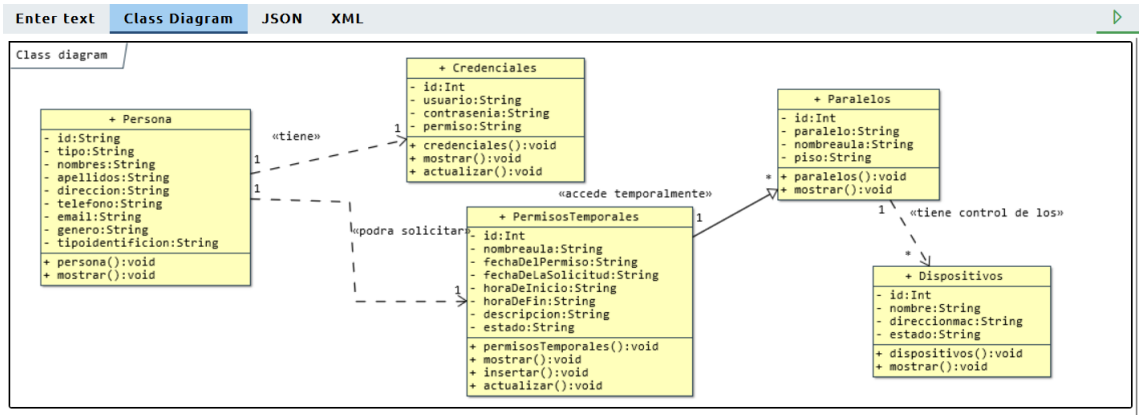
```
1    9. El docente acepta la vinculación, accede al curso y  *¡paralelos
     1<[/tiene control de los/]<n dispositivos! eléctricos y electrónicos
     ETH. Quedando concluido el caso de uso.
```

Ilustración 4.41: *Mensajes de consola relacionado a la escritura de la descripción 9.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.42: *Diagrama de clases generado para la descripción 9.*

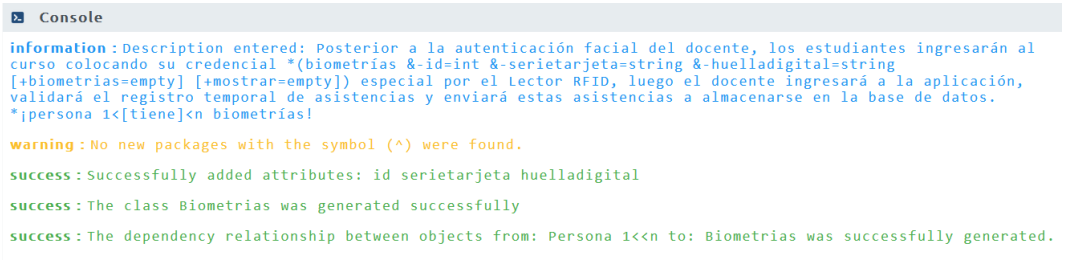


FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

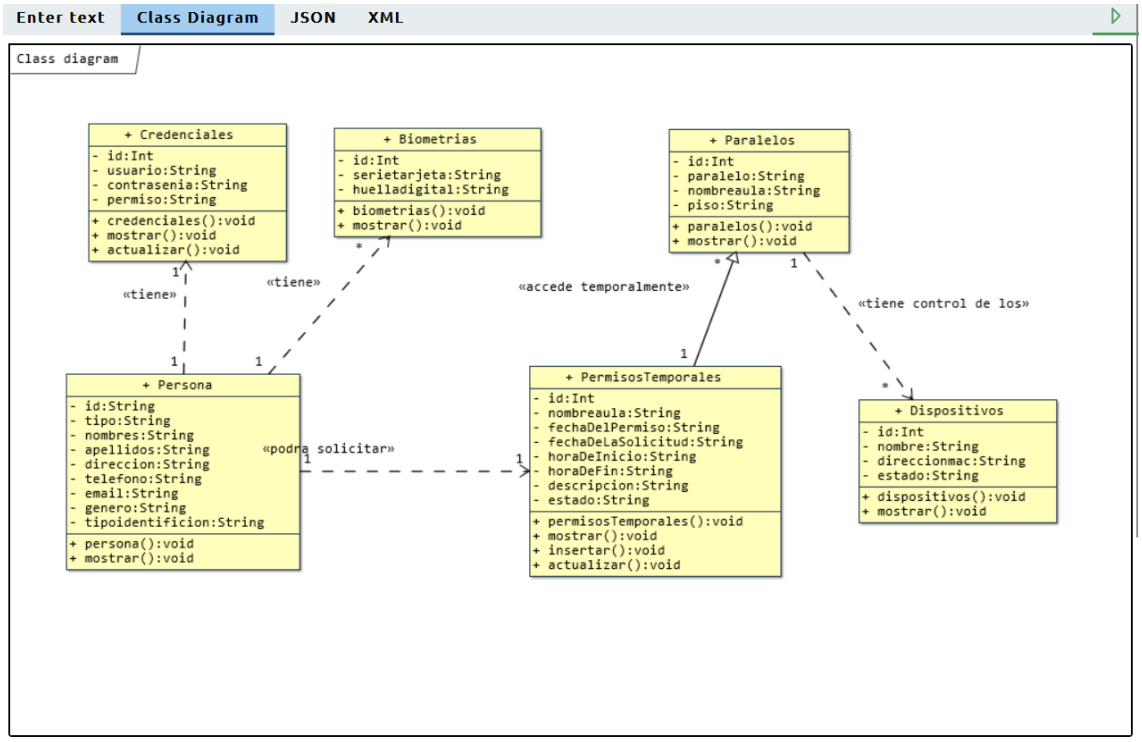
```
1   10. Posterior a la autenticación facial del docente, los estudiantes
     ingresarán al curso colocando su credencial *(biometrías &-id=int &-
     serietarjeta=string &-huelladigital=string [+biometrias=empty] [+
     mostrar=empty]) especial por el Lector RFID, luego el docente ingresar
     á a la aplicación, validará el registro temporal de asistencias y
     enviará estas asistencias a almacenarse en la base de datos. *¡persona
      1<[tiene]<n biometrías!
```

Ilustración 4.43: *Mensajes de consola relacionado a la escritura de la descripción 10.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.44: *Diagrama de clases generado para la descripción 10.*

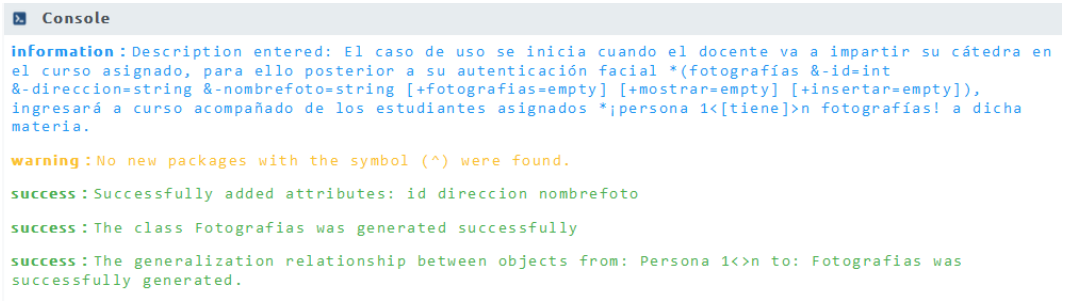


FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

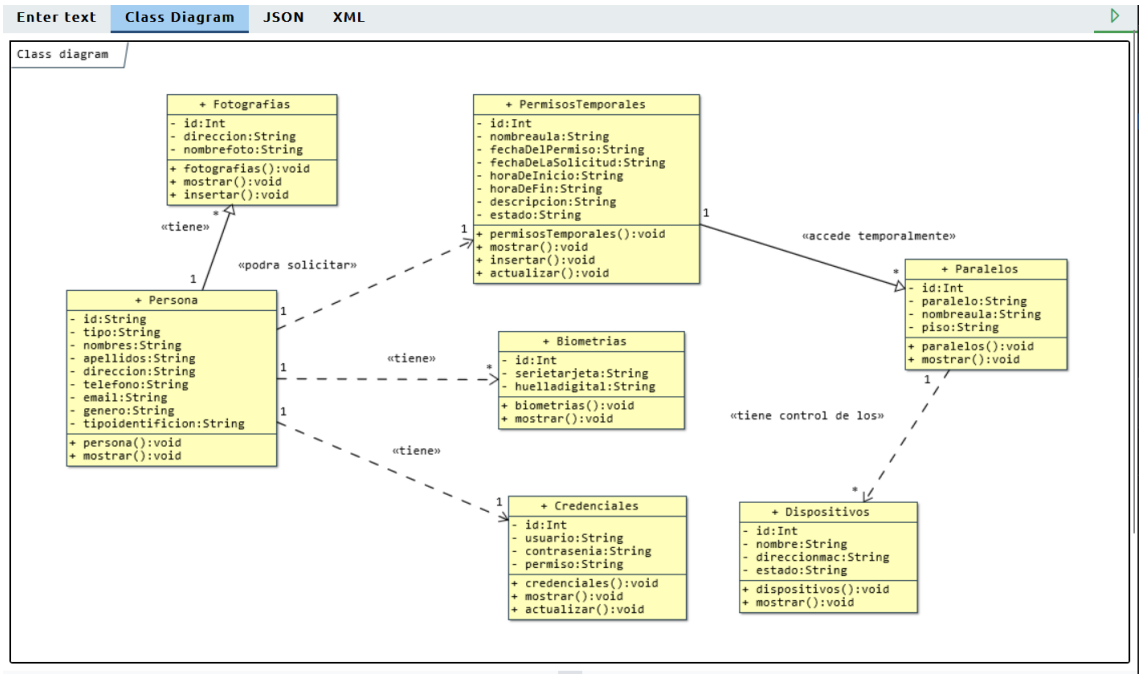
```
1   11. El caso de uso se inicia cuando el docente va a impartir su cátedra
      en el curso asignado, para ello posterior a su autenticación facial
    *(fotografías &-id=int &-direccion=string &-nombrefoto=string [+
    fotografias=empty] [+mostrar=empty] [+insertar=empty]), ingresará a
    curso acompañado de los estudiantes asignados *¡persona 1<[tiene]>n
    fotografías! a dicha materia.
```

Ilustración 4.45: *Mensajes de consola relacionado a la escritura de la descripción 11.*



FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Ilustración 4.46: *Diagrama de clases generado para la descripción 10.*



FUENTE: INVESTIGACIÓN

ELABORADO: AUTOR

El diagrama de clases que se observa el ilustración 4.46 es el diagrama final que se obtiene luego de ingresar todas las descripciones de los casos de uso. Si se compara con el original se han generado varias clases iguales a las del diagrama de clases original. También se observan los mensajes de retroalimentación que sucede al momento de interpretar cada una de las descripciones.

Tabla 4.15: *Descripción del caso de uso para ingresar al sistema.*

Caso de uso:	Ingresar al sistema
Actores:	Docente, Administrador
Propósito:	Acceder a la aplicación móvil para obtener las diversas opciones que permite el control del curso.
Resumen:	El actor docente o administrador ingresará las credenciales proporcionadas por la institución, estas se validarán y posterior otorgará el acceso a las opciones.
Tipo:	Primario

Flujo normal	
Acción del actor	**Respuesta del sistema**
1. El caso de uso se inicia cuando el actor docente o administrador necesita acceder a la aplicación, para ello ingresa a la aplicación móvil en su dispositivo smartphone.	
	2. Muestra el formulario correspondiente.
3. El actor docente o administrador ingresa el usuario y contraseña asignados por la institución; y da clic en el botón Ingresar.	
	4. Válida que los datos sean correctos y muestra el formulario de la ventana principal, dando por concluido el caso de uso.

Flujos alternos	
Acción del actor	**Respuesta del sistema**
	4.1 Muestra un mensaje de credenciales incorrectas y retorne a la línea 2.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.16: *Descripción del caso de uso con Symlen para ingresar al sistema.*

Caso de uso:	Ingresar al sistema
Actores:	Docente, Administrador
Propósito:	Acceder a la aplicación móvil para obtener las diversas opciones que permite el control del curso.
Resumen:	El actor docente o administrador *(persona) ingresará las *(credenciales) credenciales proporcionadas por la institución, estas se validarán y posterior otorgará el acceso a las opciones.
Tipo:	Primario

Flujo normal	
Acción del actor	**Respuesta del sistema**
1. El caso de uso se inicia cuando el actor docente o administrador *(persona &-id=string &-tipo=string &-nombres=string &-apellidos=string &-direccion=string &-telefono=string &-email=string &-genero = string &-tipoidentificion=string [+persona=empty] [+mostrar=empty]) necesita acceder a la aplicación, para ello ingresa a la aplicación móvil en su dispositivo smartphone.	
	2. Muestra el formulario correspondiente.
3.El actor docente o administrador ingresa el *(credenciales &-id=int &-/usuario/=String /y/ &-/contraseña/=String &-permiso=string [+credenciales=empyt] [+mostrar=empty] [+actualizar=empty]) asignados por la institución; y da clic en el botón Ingresar. *¡persona 1<[tiene]<1 credenciales!	
	4. Válida que los datos sean correctos y muestra el formulario de la ventana principal, dando por concluido el caso de uso.

Flujos alternos	
Acción del actor	**Respuesta del sistema**
	4.1 Muestra un mensaje de credenciales incorrectas y retorne a la línea 2.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.17: *Descripción del caso de uso para solicitar accesos físicos temporales.*

Caso de uso:	Solicitar accesos físicos temporales
Actores:	Docente
Propósito:	Solicitar acceso a un aula de clases en un tiempo determinado, teniendo en cuenta que está no se encuentre ocupada por otro docente.
Resumen:	El docente dentro de la aplicación móvil, escogerá la opción "Permisos cursos" dentro del menú y llenará la información solicitada.
Tipo:	Primario
Flujo normal	
Acción del actor	**Respuesta del sistema**
1. El caso de uso se inicia cuando el docente necesita tener acceso temporalmente a un aula de clases que se encuentre disponible no utilizada.	
2. El docente, una vez autenticado en la aplicación móvil, ingresará al menú y escogerá la opción "Permisos cursos".	
	3. Muestra el formulario correspondiente.
4. El docente llenará la información solicitada por la interfaz, que consiste en el curso a solicitar, la fecha del permiso que requiere el acceso, hora de inicio y hora de fin del permiso; y, una descripción donde detallará brevemente la justificación del permiso. Una vez culminado, dará clic en el botón Enviar.	
	5. Mostrará un mensaje de envío correcto de la información, finalizando el caso de uso.
Flujos alternos	
Acción del actor	**Respuesta del sistema**
	5.1 : Muestra un mensaje de error y retorne a la línea 4.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.18: *Descripción del caso de uso con Symlen para solicitar accesos físicos temp.*

Caso de uso:	Solicitar accesos físicos temporales
Actores:	Docente
Propósito:	Solicitar acceso a un aula de clases en un tiempo determinado, teniendo en cuenta que está no se encuentre ocupada por otro docente.
Resumen:	El docente dentro de la aplicación móvil, escogerá la opción "Permisos cursos" dentro del menú y llenará la información solicitada.
Tipo:	Primario

Flujo normal	
Acción del actor	**Respuesta del sistema**
1. El caso de uso se inicia cuando el docente necesita tener acceso temporalmente *(permisos temporales [+permisosTemporales=empyt] [+mostrar=empty] [+insertar=empty] [+actualizar=empty]) a un aula de clases que se encuentre disponible no utilizada.	
2. El docente, una vez autenticado en la aplicación móvil, ingresará al menú y escogerá la opción "Permisos cursos".	
	3. Muestra el formulario correspondiente.
4. El docente llenará la información solicitada por la interfaz, que consiste en el curso a solicitar, la *(permisos temporales &-id=int &-nombreaula &-/fecha del permiso/=string /que requiere el acceso,/ &-fecha de la solicitud=string &-/hora de inicio/=string /y/ &-/hora de fin/=string /del permiso; y, una/ &-/descripción/=string &-estado=string) donde detallará brevemente la justificación del permiso. Una vez culminado, dará clic en el botón Enviar. *¡persona 1<[podra solicitar]<1 permisos temporales!	
	5. Mostrará un mensaje de envío correcto de la información, finalizando el caso de uso.

Flujos alternos	
Acción del actor	**Respuesta del sistema**
	5.1 Muestra un mensaje de error y retorne a la línea 4.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.19: *Descripción del caso de uso para acceder a cursos temporales.*

Caso de uso:	Acceder a cursos temporales
Actores:	Docente
Propósito:	Ingresar al aula de clases que el docente solicitó un permiso temporal, previo a la autorización del administrador
Resumen:	Acceso al curso temporal en la aplicación móvi luego de obtener autorización, donde podrá manipular los dispositivos eléctricos y electrónicos ETH.
Tipo:	Secundario

Flujo normal	
Acción del actor	**Respuesta del sistema**
1. El caso de uso se inicia cuando el docente accede temporalmente a un aula de clases que se encuentre disponible, es decir no utilizada.	
2. El docente, una vez autenticado en la aplicación móvil, ingresará a la pestaña de "Cursos Temporales".	
	3. Mostrará el listado de los cursos temporales autorizados.
4. Escoge un determinado curso dentro de la aplicación	
	5. Verifica que tenga acceso al curso validando si encuentra dentro del rango de horas solicitadas.
	Vincula el celular a través de bluetooth con el Sistema Arduino.
7. El docente acepta la vinculación, accede al curso y tiene control de los eléctricos y electrónicos ETH. Quedando concluido el caso de uso.	
Flujos alternos	
Acción del actor	**Respuesta del sistema**
	5.1 Muestra un mensaje de error y retorne a la línea 3.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.20: *Descripción del caso de uso para Acceder a cursos temporales.*

Caso de uso:	Acceder a cursos temporales
Actores:	Docente
Propósito:	Ingresar al aula de clases *(paralelos &-id=int &-paralelo=string &-nombreaula=string &-piso=string [+paralelos=empty] [+mostrar=empty]) que el docente solicitó un permiso temporal, previo a la autorización del administrador
Resumen:	Acceso al curso temporal en la aplicación móvi luego de obtener autorización, donde podrá manipular los dispositivos *(dispositivos &-id=int &-nombre=string &-direccionmac &-estado=string [+dispositivos=empty] [+mostrar=empty]) eléctricos y electrónicos ETH.
Tipo:	Secundario

Flujo normal	
Acción del actor	**Respuesta del sistema**
1. El caso de uso se inicia cuando el docente *¡permisos temporales 1<[/accede temporalmente/]>n paralelos! a un aula de clases que se encuentre disponible, es decir no utilizada.	
2. El docente, una vez autenticado en la aplicación móvil, ingresará a la pestaña de “Cursos Temporales”.	
	3. Mostrará el listado de los cursos temporales autorizados.
4. Escoge un determinado curso dentro de la aplicación	
	5. Verifica que tenga acceso al curso validando si encuentra dentro del rango de horas solicitadas.
	Vincula el celular a través de bluetooth con el Sistema Arduino.
7. El docente acepta la vinculación, accede al curso y *¡paralelos 1<[/tiene control de los/]<n dispositivos! eléctricos y electrónicos ETH. Quedando concluido el caso de uso.	

Flujos alternos	
Acción del actor	**Respuesta del sistema**
	5.1 Muestra un mensaje de error y retorne a la línea 3.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.21: *Descripción del caso de uso para registrar asistencia de estudiantes.*

Caso de uso:	Registrar asistencia de estudiantes
Actores:	Docente, Estudiante.
Propósito:	Registrar la asistencia de los estudiantes que ingresarán al aula de clases, previo a mantener cátedra con el docente asignado.
Resumen:	Posterior a la autenticación facial del docente, los estudiantes ingresarán al curso colocando su credencial especial por el Lector RFID, luego el docente ingresará a la aplicación, validará el registro temporal de asistencias y enviará estas asistencias a almacenarse en la base de datos.
Tipo:	Secundario

Flujo normal	
Acción del actor	**Respuesta del sistema**
1. El caso de uso se inicia cuando el docente va a impartir su cátedra en el curso asignado, para ello posterior a su autenticación facial , ingresará a curso acompañado de los estudiantes asignados a dicha materia.	
2. Los estudiantes ingresarán al curso, ordenados e independientes; y, colocarán su credencial especial en el Lector RFID.	
	3. Obtiene el código de la tarjeta, válida los datos del estudiante y registra su asistencia temporal.
	4. Mostrará en la interface de usuario, la asistencia de ingreso a la materia de todos los estudiantes.
5. El docente deberá dar clic en el ícono de “Actualizar Asistencias”, para obtener las asistencias e inasistencias temporales de cada alumno.	
	6. En la interfaz de usuario, se actualizará las asistencias de los estudiantes que han utilizado la credencial especial.
7. El docente verificará que las asistencias se encuentren correctas, y dará clic en el ícono de “Guardar Asistencias” para almacenar definitivamente las asistencias en la base de datos.	

	8. Mostrará un mensaje de confirmación, para registrar la información.
9. El docente confirmará a través del respectivo botón de la aplicación móvil, subir la asistencia a la base de datos.	
	10. Mostrará mensaje de éxito, actualiza la lista de estudiantes con la respectiva asistencia, y bloquea el botón "Registrar Asistencia". Con esto finaliza el caso de uso.
Flujos alternos	
Acción del actor	**Respuesta del sistema**
	5.1 Muestra un mensaje de error y retorne a la línea 3.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

Tabla 4.22: *Descripción del caso de uso con Symlen para Registrar asistencia de est.*

Caso de uso:	Registrar asistencia de estudiantes
Actores:	Docente, Estudiante.
Propósito:	Registrar la asistencia de los estudiantes que ingresarán al aula de clases, previo a mantener cátedra con el docente asignado.
Resumen:	Posterior a la autenticación facial del docente, los estudiantes ingresarán al curso colocando su credencial *(biometrías &-id=int &-serietarjeta=string &-huelladigital=string [+biometrias=empty] [+mostrar=empty]) especial por el Lector RFID, luego el docente ingresará a la aplicación, validará el registro temporal de asistencias y enviará estas asistencias a almacenarse en la base de datos. *¡persona 1<[tiene]<n biometrías!
Tipo:	Secundario
Flujo normal	
Acción del actor	**Respuesta del sistema**
1. El caso de uso se inicia cuando el docente va a impartir su cátedra en el curso asignado, para ello posterior a su autenticación facial *(fotografías &-id=int &-direccion=string &-nombrefoto=string [+fotografias=empty] [+mostrar=empty] [+insertar=empty]), ingresará a curso acompañado de los estudiantes asignados *¡persona 1<[tiene]>n fotografías! a dicha materia.	
2. Los estudiantes ingresarán al curso, ordenados e independientes; y, colocarán su credencial especial en el Lector RFID.	

	3. Obtiene el código de la tarjeta, válida los datos del estudiante y registra su asistencia temporal.
	4. Mostrará en la interfaz de usuario, la asistencia de ingreso a la materia de todos los estudiantes.
5. El docente deberá dar clic en el ícono de “Actualizar Asistencias”, para obtener las asistencias e inasistencias temporales de cada alumno.	
	6. En la interfaz de usuario, se actualizará las asistencias de los estudiantes que han utilizado la credencial especial.
7. El docente verificará que las asistencias se encuentren correctas, y dará clic en el ícono de “Guardar Asistencias” para almacenar definitivamente las asistencias en la base de datos.	
	8. Mostrará un mensaje de confirmación, para registrar la información.
9. El docente confirmará a través del respectivo botón de la aplicación móvil, subir la asistencia a la base de datos.	
	10. Mostrará mensaje de éxito, actualiza la lista de estudiantes con la respectiva asistencia, y bloquea el botón “Registrar Asistencia”. Con esto finaliza el caso de uso.
Flujos alternos	
Acción del actor	**Respuesta del sistema**
	5.1 Muestra un mensaje de error y retorne a la línea 3.

FUENTE: INVESTIGACIÓN
ELABORADO: AUTOR

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

- Para la gestionar la obtención del diagrama de clases a partir de las especificaciones de los casos de uso se implementaron mensajes de información notificando el estado en el que se encuentre las descripciones ingresadas. Además, se genera una estructura json y xml que permita compartir los datos del diagrama de clases y poder visualizarlo con otras librerías enfocadas a la creación de diagramas UML y finalmente se evaluó la librería con un sistemas de información que fue publicado por la universidad de Quevedo.

- Armadillo fue un complemento muy bueno para potenciar el funcionamiento de la herramienta TDDT4IoTS. Permite retroalimentar a los desarrolladores de software sobre los errores que se pueden cometer en la escritura de los casos de uso utilizando el lenguaje SymLen. Además, si el texto ingresado no cuenta con ningún error de escritura se muestran mensajes de éxito detallando lo que se pudo interpretar.

- La estructura que contiene la información técnica diagrama de clases se generó en un formato que pueda ser compatible con otras librerías que permitan crear diagramas UML, por ejemplo jsUML2 (`http://www.jrromero.net/tools/jsUML2`) es compatible con la estructura generada automáticamente. Si el desarrollador lo desea, puede crear su propia librería personalizada que le permita visualizar el diagrama de clases mediante la información generada por Armadillo.

- TDDT4IoTS es una aplicación orientada a la generación del código fuente a partir del modelado del software que se pretende desarrollar. Armadillo permite generar uno de los diagramas más importantes para el modelado de software que es el diagrama de clases, mediante el cual se podrá generar código usable para su posterior implementación.

5.2. Recomendaciones

- Armadillo puede ser fácilmente escalable. A futuro se pretende desarrollar un versión para npm y subir Armadillo a su repositorio oficial para que pueda ser utilizado mediante proyectos que utilicen como servidor de aplicaciones Node.js.

- Durante la elaboración de la retroalimentación sobre cómo utilizar los símbolos del lenguaje SymLen para escribir las descripciones de los casos de uso, se pretenden realizar trabajos a futuro sobre como mejorar la identificación de la posición exacta del error que se está cometiendo. Además, que permita resaltar con diferentes colores los caracteres donde se encuentren las anomalías.

- Para que armadillo pueda seguir mejorando su funcionalidad, se pretende publicar el código fuente en una plataforma web. GitHub es una de las plataformas mas conocidas para realizar este tipo de aportes a la comunidad de desarrolladores, permitiendo que pueda recibir mejoras y su alcance pueda llegar ser a nivel mundial.

- Para lograr un mayor alance sobre las tecnologías de desarrollo que existen actualmente, y con las que en un futuro serán desarrolladas. Se podrá crear una versión de armadillo como servicio web, desplegado en un servidor publico ubicado en la nube. Se pretende que realice el mismo proceso que la versión actual, con la ventaja que podrá ser utilizado en casi todos los tipos de proyectos que existen. Se podrá obtener los resultados del proyecto de forma clara y remota.

CAPÍTULO VI

BIBLIOGRAFÍA

Referencias

[1] V. Panthi, A. Tripathi **and** D. P. Mohapatra, "Software validation based on prioritization using concurrent activity diagram," *International Journal of Systems Assurance Engineering and Management*, **august** 2022, ISSN: 09764348. DOI: 10.1007/S13198-021-01551-8.

[2] F. Chen, L. Zhang, X. Lian **and** N. Niu, "Automatically recognizing the semantic elements from uml class diagram images," *Journal of Systems and Software*, **jourvol** 193, **page** 111 431, **november** 2022, ISSN: 01641212. DOI: 10.1016/J.JSS.2022.111431. **url**: https://linkinghub.elsevier.com/retrieve/pii/S0164121222001340.

[3] R. Kulesza, M. F. D. Sousa, M. L. M. D. Araújo, C. P. D. Araújo **and** A. M. Filho, "Evolution of web systems architectures: A roadmap," *Special Topics in Multimedia, IoT and Web Technologies*, **pages** 3–21, **january** 2020. DOI: 10.1007/978-3-030-35102-1_1/FIGURES/6. **url**: https://link.springer.com/chapter/10.1007/978-3-030-35102-1_1.

[4] M. S. Hamdi, A. Ghannem **and** M. Kessentini, "Requirements traceability recovery for the purpose of software reuse: An interactive genetic algorithm approach," *Innovations in Systems and Software Engineering*, **jourvol** 18, **pages** 193–213, 1 **march** 2022, ISSN: 16145054. DOI: 10.1007/S11334-021-00418-2.

[5] E. Guerra, A. D. O. Dias, L. G. D. Veras, A. Aguiar, J. Choma **and** T. S. D. Silva, "A model to enable the reuse of metadata-based frameworks in adaptive object model architectures," *IEEE Access*, **jourvol** 9, **pages** 85 124–85 143, 2021, ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3087795.

[6] OMG, *About the unified modeling language specification version 2.4*. **url**: https://www.omg.org/spec/UML/2.4/.

[7] G. Bergström, F. Hujainah, T. Ho-Quang **andothers**, "Evaluating the layout quality of uml class diagrams using machine learning," *Journal of Systems and Software*, **page** 111 413, **october** 2022, ISSN: 01641212. DOI: 10.1016/J.JSS.2022.111413.

[8] Omg, "An omg ® unified modeling language ® publication omg ® unified modeling language ® (omg uml ®) omg document number: Date," 2009. **url**: https://www.omg.org/spec/UML/20161101/PrimitiveTypes.xmi.

[9] M. Jahan, Z. S. H. Abad **and** B. Far, "Generating sequence diagram from natural language requirements," *Proceedings of the IEEE International Conference on Requirements Engineering*, **jourvol** 2021-September, **pages** 39–48, **september** 2021, ISSN: 23326441. DOI: 10.1109/REW53955.2021.00012.

[10] F. Losavio, A. Matteo **and** I. Pacilli, "Goal-oriented process for domain analysis using quality standards," *Enlace*, **jourvol** 6, **pages** 11–28, 3 2009, ISSN: 1690-7515. **url**: http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S1690-75152009000300002&lng=es&nrm=iso&tlng=eshttp://ve.scielo.org/scielo.php?script=sci_abstract&pid=S1690-75152009000300002&lng=es&nrm=iso&tlng=es.

[11] C. M. Zapata **and** G. González, "Formal ocl specification of consistency rules between the uml class and the use case models and the interfaces model," *Revista Ingenierías Universidad de Medellín*, **jourvol** 7, 12 **june** 2008, ISSN: 2248-4094. **url**: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1692-33242008000100010.

[12] S. Iqbal, I. Al-Azzoni, G. Allen **and** H. U. Khan, "Extending uml use case diagrams to represent non-interactive functional requirements," *E-Informatica Software Engineering Journal*, **jourvol** 14, **pages** 97–115, 1 2020, ISSN: 20844840. DOI: 10.37190/E-INF200104.

[13] E. A. Abdelnabi, A. M. Maatuk **and** M. Hagal, "Generating uml class diagram from natural language requirements: A survey of approaches and techniques," *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques*

of Automatic Control and Computer Engineering, MI-STA 2021 - Proceedings, **pages** 288–293, **may** 2021. DOI: 10.1109/MI-STA52233.2021.9464433.

[14] H. M. Abu-Dalbouh **and** S. A. Alateyah, "An extension to uml for the modeling of web based bus reservation system," *Journal of Computer Science*, **jourvol** 16, **pages** 825–837, 7 2020, ISSN: 15526607. DOI: 10.3844/JCSSP.2020.825.837.

[15] G. Guerrero-Ulloa, D. Carvajal-Suárez, G. Brito Casanova, A. Pachay Espinoza, M. J. Hornos **and** C. Rodríguez-Domínguez, *Test-driven development tool for iot-based system*. **url**: https://aplicaciones.uteq.edu.ec/tddt4iots/.

[16] C. G. Moyano, L. Pufahl, I. Weber **and** J. Mendling, "Uses of business process modeling in agile software development projects," *Information and Software Technology*, **jourvol** 152, **page** 107 028, **december** 2022, ISSN: 0950-5849. DOI: 10.1016/J.INFSOF.2022.107028. **url**: https://linkinghub.elsevier.com/retrieve/pii/S0950584922001483.

[17] Z. A. Hamza **and** M. Hammad, "Analyzing uml use cases to generate test sequences," *International Journal of Computing and Digital Systems*, **jourvol** 10, **pages** 125–134, 1 2021, ISSN: 2210142X. DOI: 10.12785/IJCDS/100112.

[18] P. Bourhis, J. L. Reutter **and** D. Vrgoč, "Json: Data model and query languages," *Information Systems*, **jourvol** 89, **march** 2020, ISSN: 03064379. DOI: 10.1016/J.IS.2019.101478.

[19] A. Khalili, "An xml-based approach for geo-semantic data exchange from bim to vr applications," *Automation in Construction*, **jourvol** 121, **january** 2021, ISSN: 09265805. DOI: 10.1016/J.AUTCON.2020.103425.

[20] A. Alwabel, "Coedit: A novel error correction mechanism in compilers using spelling correction algorithms," *Journal of King Saud University - Computer and Information Sciences*, 2021, ISSN: 22131248. DOI: 10.1016/J.JKSUCI.2021.02.010.

[21] L. Addazi **and** F. Ciccozzi, "Blended graphical and textual modelling for uml profiles: A proof-of-concept implementation and experiment," *Journal of Systems and Software*, **jourvol** 175, **may** 2021, ISSN: 01641212. DOI: 10.1016/J.JSS.2021.110912.

[22] C. M. Fonseca, J. P. A. Almeida, G. Guizzardi **and** V. A. Carvalho, "Multi-level conceptual modeling: Theory, language and application," *Data and Knowledge Engineering*, **jourvol** 134, **july** 2021, ISSN: 0169023X. DOI: 10.1016/J.DATAK.2021.101894.

[23] N. Mohamed, S. Mazen **and** W. Helmy, "E-ahp: An enhanced analytical hierarchy process algorithm for priotrizing large software requirements numbers," **page** 2022. **url**: www.ijacsa.thesai.org.

[24] E Whiting **and** S Datta, "Design and development of a technology-agnostic nfr testing framework introducing the framework and discussing the future of load testing in agile software development," DOI: 10.1145/3520084.3520092. **url**: https://doi.org/10.1145/3520084.3520092.

[25] M. Shafiq **and** U. Waheed, "Documentation in agile development: A comparative analysis," *Proceedings of the 21st International Multi Topic Conference, INMIC 2018*, **december** 2018. DOI: 10.1109/INMIC.2018.8595625.

[26] L. Tan, Z. Yang **and** J. Xie, "Ocl constraints automatic generation for uml class diagram," *Proceedings 2010 IEEE International Conference on Software Engineering and Service Sciences, ICSESS 2010*, **pages** 392–395, 2010. DOI: 10.1109/ICSESS.2010.5552361.

[27] L. Chen **and** Y. Zeng, "Automatic generation of uml diagrams from product requirements described by natural language," *Proceedings of the ASME Design Engineering Technical Conference*, **jourvol** 2, **pages** 779–786, PARTS A AND B **july** 2010. DOI: 10.1115/DETC2009-86514.

[28] O. S. D. Omer **and** S. Eltyeb, "Towards an automatic generation of uml class diagrams from textual requirements using case-based reasoning approach," **pages** 1–5, **june** 2022. DOI: 10.1109/ICAAID51067.2022.9799502.

[29] A. M. Alashqar, "Automatic generation of uml diagrams from scenario-based user requirements," *Jordanian Journal of Computers and Information Technology (JJCIT)*, **jourvol** 07, 02 **june** 2021.

[30] Shweta **and** R. Sanyal, "Impact of passive and negative sentences in automatic generation of static uml diagram using nlp," *Journal of Intelligent and Fuzzy Systems*,

jourvol 39, **pages** 2047–2059, 2 **january** 2020, ISSN: 1064-1246. DOI: 10.3233/JIFS-179871.

[31] *Manifiesto por el desarrollo ágil de software*. **url**: https://agilemanifesto.org/iso/es/manifesto.html.

[32] T. N. Kudo, R. D. F. Bulcão-Neto, V. Vicente, G. Neto, Auri **and** M. R. Vincenzi, "Aligning requirements and testing through metamodeling and patterns: Design and evaluation," **jourvol** 1, **page** 3, DOI: 10.1007/s00766-022-00377-5. **url**: https://doi.org/10.1007/s00766-022-00377-5.

[33] K. Rokis **and** M. Kirikova, "Challenges of low-code/no-code software development: A literature review," **pages** 3–17, 2022. DOI: 10.1007/978-3-031-16947-2_1. **url**: https://link.springer.com/chapter/10.1007/978-3-031-16947-2_1.

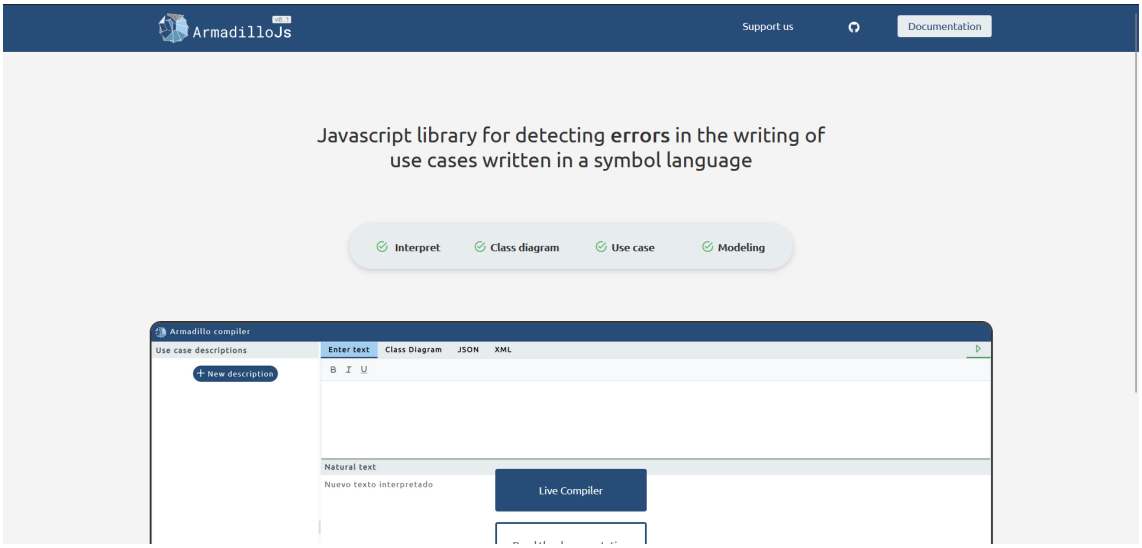
[34] L. Gazzola, L. Mariani, M. Orru, M. Pezze **and** M. Tappler, "Testing software in production environments with data from the field," *Proceedings - 2022 IEEE 15th International Conference on Software Testing, Verification and Validation, ICST 2022*, **pages** 58–69, 2022. DOI: 10.1109/ICST53961.2022.00017.

[35] P. D. Investigación, G. D. Los, R. De **andothers**, "Sistema basado en internet de las cosas para la gestión de los recursos de un aula de clases," 2020. **url**: https://repositorio.uteq.edu.ec/handle/43000/5943.

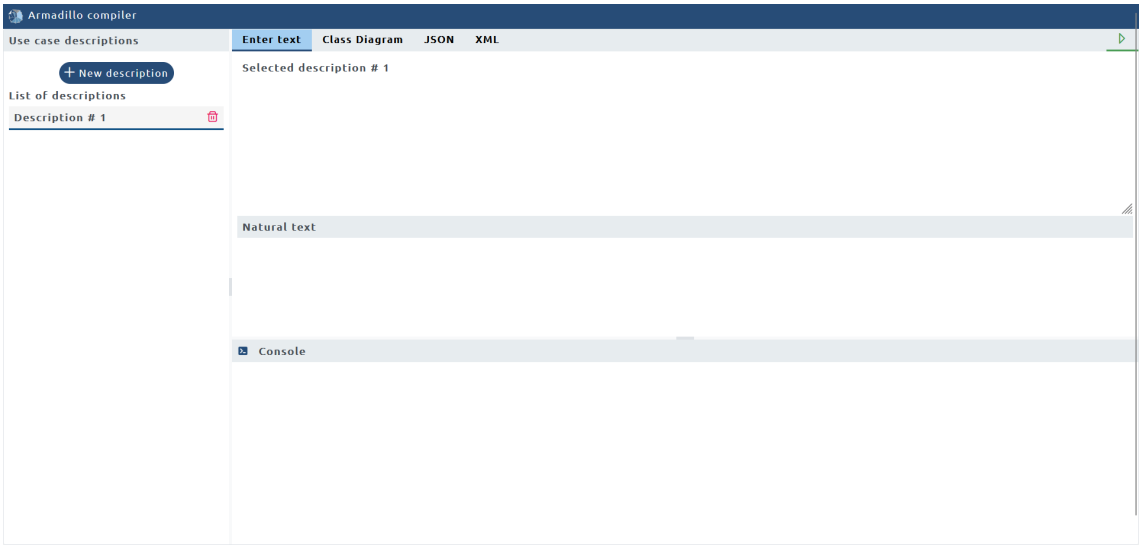
CAPÍTULO VII

ANEXOS

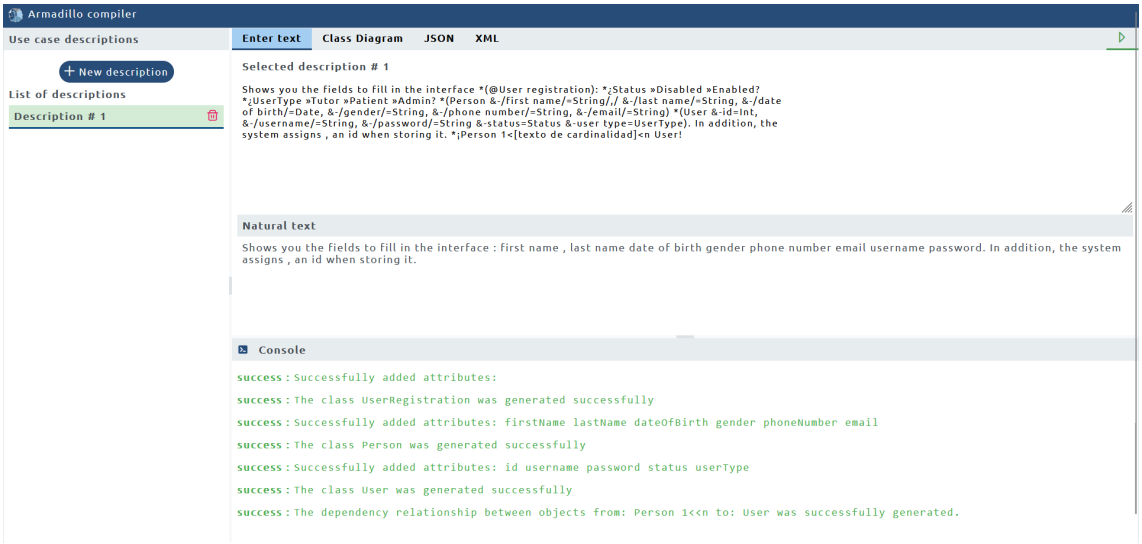
6.1. Interfaz principal de aplicación demostrativa de la librería



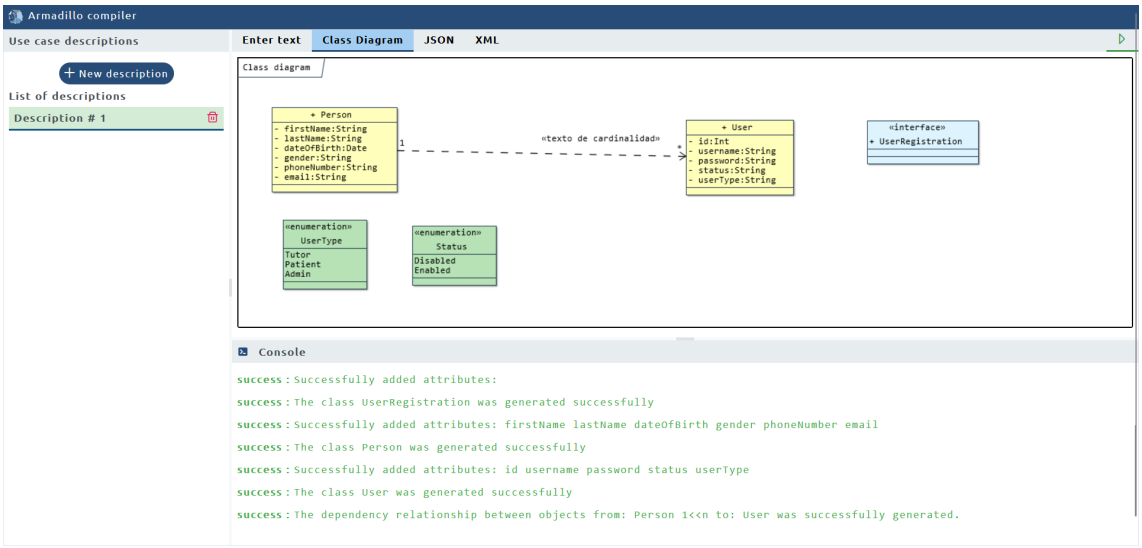
6.2. Interfaz donde se ingresarán las descripciones de los casos de uso a interpretar



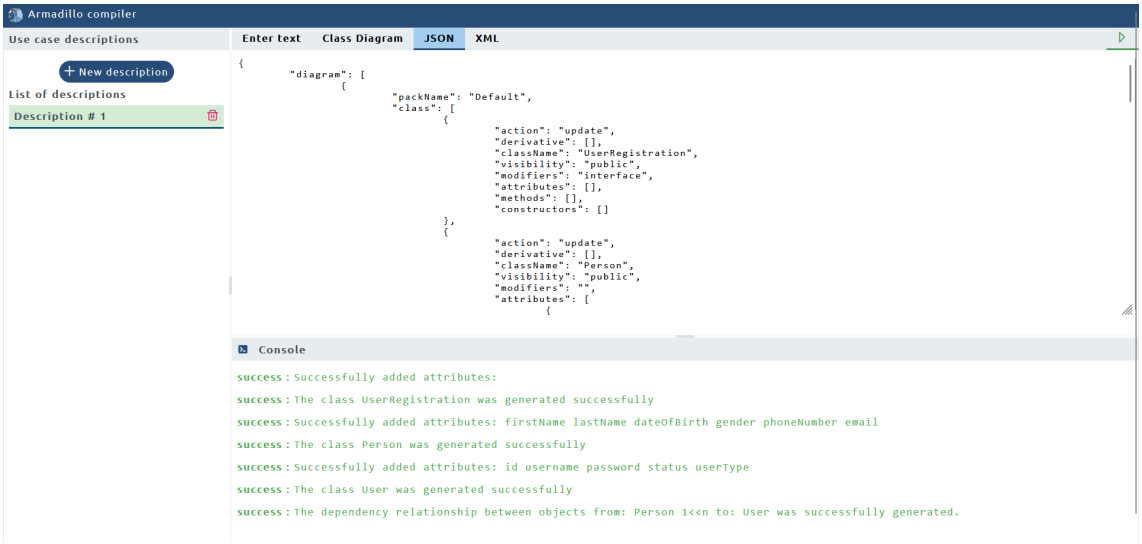
6.3. Descripción del caso de uso interpretada con su respectiva retroalimentación



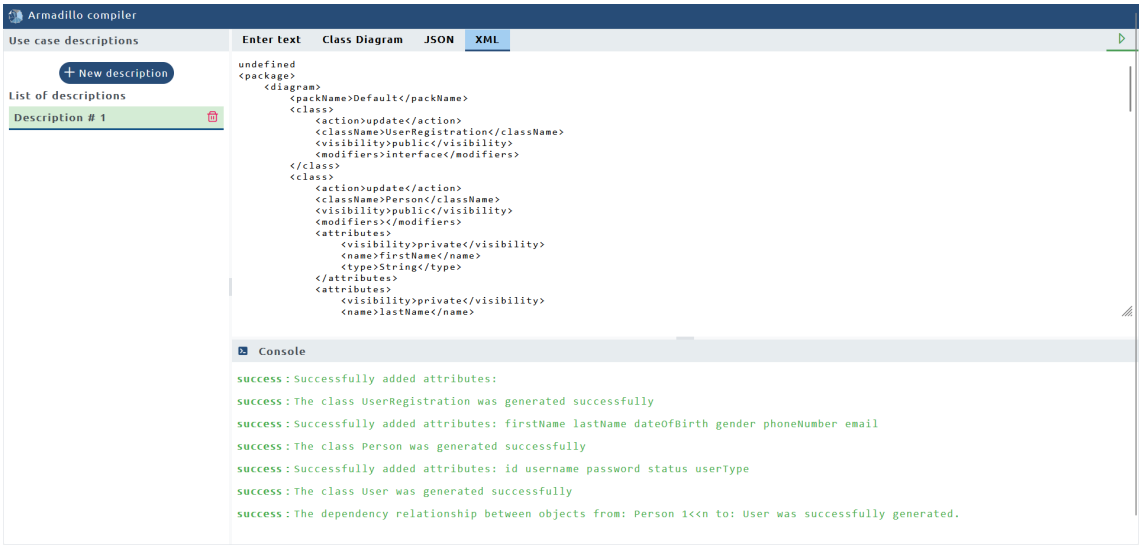
6.4. Diagrama de clases generado por la librería usando una librería externa denominada jsUML2



6.5. Estructura JSON generada por la librería



6.6. XML generado por la librería



6.7. Error detectado en la descripción del caso de uso