Mejora del Proceso de Implantación de Líneas de Productos de Software Mediante la Construcción y Uso de la Herramienta SPL Studio

Alumno

Jorge Carracedo Hernández

Director

Dr. Rubén Heradio Gil

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos – Cod. 31105151

Universidad Nacional de Educación a Distancia (UNED)

Septiembre 2022

Mejora del Proceso de Implantación de Líneas de Productos de Software Mediante la Construcción y Uso de la Herramienta SPL Studio

Alumno

Jorge Carracedo Hernández

Director

Dr. Rubén Heradio Gil

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos – Cod. 31105151

Universidad Nacional de Educación a Distancia (UNED)

Septiembre 2022

Abstract

Software product lines are defined as a set of software systems that share a set of characteristics to satisfy a particular market segment. The development of software oriented to software product lines offers multiple benefits, however, its implementation is not a simple process and involves a significant initial effort, which sometimes makes it not feasible to apply this approach, especially when it comes to small projects. In the present work some current tools used for the purpose are analyzed and a new tool is built that tries to improve this process and offer the possibility of applying this approach in solutions where it was not previously viable.

Keywords: SPL, Implantation of software product lines, code generation.

Resumen.

Las líneas de productos de software se definen como un conjunto de sistemas de software que comparten una serie de características para satisfacer un segmento particular de mercado. El desarrollo de software orientado a líneas de productos de software ofrece múltiples beneficios, no obstante, su implantación no es un proceso sencillo y supone un esfuerzo inicial importante que hace que en ocasiones no sea factible aplicar este enfoque, especialmente cuando se trata de pequeños proyectos. En el presente trabajo se analizan algunas herramientas actuales utilizadas para el propósito y se construye una nueva herramienta que intenta mejorar este proceso y ofrecer la posibilidad de aplicar este enfoque en soluciones donde antes no era viable.

Palabras clave: LPS, Implantación de líneas de productos de software, Generación de código.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

1.1. Reutilización de software

La reutilización de software no es una práctica nueva, lleva acompañando a los procesos de desarrollo de software desde que se comenzaron a construir las primeras librerías en la década de los 60, y desde entonces las técnicas de reutilización han seguido evolucionando hasta nuestros días.

El objetivo fundamental es evitar la necesidad de desarrollar ciertas partes de un software reutilizando partes comunes en distintos componentes.

Los principales beneficios que podemos encontrar en la reutilización del software son:

- **Reducción de errores**: Las partes reutilizadas de un software ya han sido probadas y evitan que aparezcan nuevos errores.
- **Reduce el coste de desarrollo**: Evitar desarrollar desde cero algunas partes de un software, hace que se reduzca el coste de desarrollo.
- **Mejora el tiempo de desarrollo**: De la misma forma que los costes se reducen, el tiempo de desarrollo también disminuye debido a la necesidad de desarrollar menos cantidad de código.

Por otra parte, también es posible diferenciar entre dos tipos de enfoques de reutilización de software (Usaola 2013):

- **Reutilización oportunista**: Los elementos de proyectos anteriores se reutilizan cuando aparece la necesidad.
- **Reutilización planificada**: Los elementos desde un inicio se construyen para ser reutilizados.

1.2. Líneas de productos de software (LPS)

Como una alternativa a la reutilización de software tradicional, aparecen las líneas de productos de software.

El enfoque de las LPS pretende llevar la reutilización de software a una escala mayor, donde a diferencia del enfoque tradicional, en el que ciertas partes de un componente de software son reutilizadas, se reutiliza por sistema el componente al completo. Este enfoque es aplicable a productos que son similares entre sí, excepto por una serie de características, y recibe el nombre de familia de productos.

Para conseguir este nivel de reutilización que requiere una LPS, la reutilización debe ser absolutamente planificada.

1.3. Beneficios de las líneas de productos de software

La utilización de LPS tiene beneficios contrastados (Ingeno, 2018; Gomaa, 2004) que hacen que su uso sea una opción muy atractiva a la hora de enfocar determinados proyectos.

Ferguson (2018) define algunos de los principales beneficios relacionados con la implantación de una línea de productos de software:

- Ganancias de productividad a gran escala.
- Disminución del tiempo de comercialización.
- Aumento de la calidad del producto.
- Disminución del riesgo del producto.
- Mayor agilidad para la puesta en el mercado.
- Mayor satisfacción del cliente
- Uso más eficiente de los recursos humanos.
- Capacidad para mantener la presencia en el mercado.
- Capacidad para personalizar o configurar rápidamente el producto.
- Capacidad para sostener un crecimiento sin precedentes.

1.4. Implantación

El mayor problema que supone optar por el desarrollo de una LPS es la implantación de la misma. Su creación no siempre es trivial y conlleva un esfuerzo considerable, es necesario abordar dos áreas distintas para su construcción:

- **Ingeniería de dominio**: Se encarga de los elementos comunes de la LPS.
- **Ingeniería de aplicación**: Se encarga de generar cada uno de los productos miembros de LPS.

Weiss (1999) presenta un listado de artefactos que es necesario construir cuando se implanta una LPS. En las tablas **Tabla 1** y **Tabla 2**, se presentan dichos artefactos.

Tabla 1. Artefactos de ingeniería de dominio

Artefacto	Descripción
D1	Un modelo de dominio.
D2	Una definición de la familia de productos que indique como se caracterizan y varían los miembros de la familia.
D3	Un modelo de decisiones para la creación de un miembro específico del dominio.
D4	Una especificación de un lenguaje de modelado de aplicaciones para el que sea posible crear un compilador.
D5	Un diseño común a todos los miembros de la familia que se utilizará como base para la creación de nuevos miembros.
D6	Un mapa de composición, para mapear el diseño común y el lenguaje de modelado.
D7	Un diseño del conjunto de herramientas que se va a utilizar para el entorno de ingeniería de dominio.
D8	Una librería de plantillas que se utilizará para la generación del código y la documentación de las aplicaciones.
D9	Herramientas para realizar la generación del código y la documentación de los productos.
D10	Herramientas para el análisis de los modelos de la aplicación que ayuden al ingeniero de aplicación a validar sus modelos.
D11	Una descripción de cómo funciona el proceso de modelado y generación de aplicaciones.
D12	Documentación que explique cómo se utiliza el entorno de ingeniería de aplicación.

Tabla 2. Artefactos de ingeniería de aplicación

Artefacto	Descripción
A1	Un modelo de aplicación creado por el ingeniero de aplicación especificado en el lenguaje de modelado de aplicaciones.
A2	Código de la aplicación generado a partir del modelo de la aplicación.
A3	Documentación de la aplicación generado a partir del modelo de la aplicación.

La creación de todos estos artefactos aplica un coste inicial al desarrollo enfocado en LSP, que en ocasiones supone un problema. La gráfica de la figura **Fig. 1**, compara el coste necesario para desarrollar productos a través de una LSP en comparación con el coste necesario si se realiza con un desarrollo tradicional.

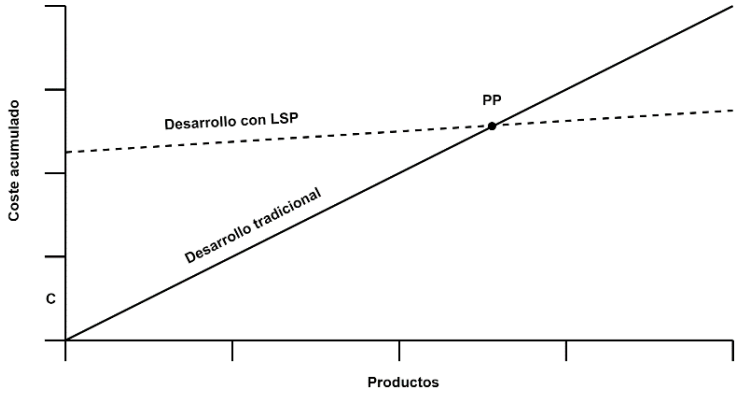


Fig. 1 Gráfica de costes acumulados. C representa el coste de desarrollar un producto. PP representa el punto de pago.

En punto de pago (PP), se encuentra la clave para decidir si es viable la construcción de cierto software a través de LPS. Cuanto más cerca esté este punto del eje, la LPS va a comenzar a ser productiva más pronto, lo que hará que la inversión con este enfoque sea más viable.

En la gráfica **Fig. 1** también se puede observar, como, el coste acumulado con el que comienza el desarrollo de la LPS afecta directamente al PP, de forma que, si este coste inicial es muy elevado, puede hacer que la LPS no sea factible.

Si se desglosa este coste inicial, se pueden diferenciar tres categorías como se muestra en la figura **Fig. 2**.

En una primera categoría se encuentra el coste del desarrollo del prototipo, este coste podría asemejarse al coste de creación de un producto utilizando un desarrollo tradicional, es más, una vez creado un producto, podría crearse la LPS en base a él y este producto servir de prototipo.

Por otra parte, se tienen los costes fijos de la creación de la LPS en sí, es decir, de la construcción de todos los artefactos necesarios para su creación y puesta en marcha.

Por último, se tienen unos costes variables que dependen de el volumen y la complejidad de la LPS que se va a tratar. Estos costes variables, suponen un porcentaje proporcional a la suma de los dos anteriores.

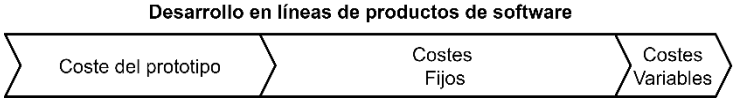


Fig. 2 Desglose del coste acumulado inicial para el desarrollo en LPS.

Si eliminamos los costes del prototipo por ser similares al coste de producción de un solo producto. El coste de la construcción de una LPS se va a dividir entre una parte de costes fijos, en la cual encontramos las tareas que siempre son necesarias, como la configuración de los entornos, creación de lenguajes específicos de dominio o de modelos de características, aprendizaje de las tecnologías utilizadas, etc. Y otra parte variable que dependerá del volumen de la LPS, cuanto mayor sea éste, mayor será el trabajo para crear sus artefactos.

1.5. Aplicación sobre pequeños componentes

Los beneficios de crear una línea de productos de software son cuantiosos, sin embargo, cuando una línea de productos de software se quiere aplicar a un componente pequeño, los costes fijos a los que hacíamos referencia anteriormente, toman una mayor importancia al mantenerse invariables y provocan que el punto PP se aleje demasiado como para que sea viable.

La creación de los artefactos necesarios para el inicio no es sencilla y es habitual necesitar mano de obra cualificada para la gestión y el mantenimiento de las LPS. Teniendo en cuenta que este tipo de perfiles de trabajadores no son muy frecuentes, esto podría incrementar aún más el coste del desarrollo.

No es de extrañar que en ocasiones donde se podrían aprovechar las ventajas de utilizar LPS, no llegue a plantearse su uso, no obstante, este enfoque de producción mejoraría significativamente el valor del software por pequeño que sea, pues se desarrollaría maximizando su grado de reutilización.

1.6. Herramientas para el desarrollo de líneas de productos de software

Por suerte, existen algunas herramientas que pueden ayudar con el trabajo de creación de la línea de productos de software. Pero estas herramientas, como se verá más adelante, no siempre se ajustan completamente al propósito buscado. Y aún que es cierto que simplifican la tarea, su uso no siempre es adecuado.

1.7. Objetivo

El objetivo de este trabajo es desarrollar una herramienta sencilla que permita la creación de LPS de una forma rápida.

Además, la aplicación creada debe ofrecer todas las herramientas necesarias para construir todos los artefactos iniciales propuestos en Weiss (1999), así como automatizar la creación de algunos de ellos con el fin de acelerar el desarrollo y evitar procesos complicados del proceso.

La aplicación creada tiene que tener una interface web, que, entre otras ventajas, permita su uso sin necesidad de instalar ningún tipo de entorno.

Con todo ello se pretende demostrar que es posible aprovechar las ventajas de LPS en pequeños proyectos y otros proyectos donde los costes fijos de implantación de la LPS jueguen un papel determinante.

Capítulo 2: Estado del arte

A continuación, se presentan algunas herramientas de modelado más utilizadas para la creación de LPS junto con algunas características que pueden influir directamente en el coste inicial.

2.1. Xtext

Se trata de un marco para el desarrollo de lenguajes de programación y lenguajes específicos de dominio. Está basada en el entorno de desarrollo integrado (IDE) eclipse y al ser de código abierto, es totalmente gratuito. Se trata de una de las herramientas de este ámbito más importantes actualmente.

El uso de Xtext en el desarrollo de LPS es generalmente la creación de la gramática del lenguaje de dominio especifico. Una vez definido se genera un entorno también basado en eclipse para el modelado de productos.

Xtend es un lenguaje de programación de tipo estático que fue creado con Xtext y al igual que este, también es de código abierto y basado en el IDE eclipse. Tiene sus raíces en el lenguaje Java, pero lo mejora en muchos aspectos, entre otras cosas, cuenta con soporte para el desarrollo de plantillas.

En el desarrollo de LPS, es habitual encontrar la combinación de Xtext + Xtend donde este último es utilizado para crear las plantillas que se utilizarán en la generación del código de los productos.

Tabla 3. Tabla de características para Xtext.

Característica	Valoración
Dificultad de uso	Alta
Necesita complementarse con otras herramientas	Si
Necesidad de instalación y configuración	Si
Tiene limitaciones en cuanto al lenguaje de generación	No
Necesita redistribuir la línea de productos en cada cambio	Si
Es multiplataforma	Si

https://www.eclipse.org/Xtext/ - https://www.eclipse.org/xtend/

2.2. Jetbrains MPS

Se autodefine como un banco de trabajo para diseñar lenguajes específicos de dominio. Se trata de un entorno de desarrollo completo compatible con Windows, Mac y Linux. Dispone de herramientas para la creación de lenguajes específicos de dominio, modelado en los lenguajes creados y diferentes complementos para la generación de

componentes a partir de los modelos. Se trata de una de las herramientas más completas que se pueden encontrar actualmente.

Está basado en el IDE de su propio fabricante IntelliJ.

Tabla 4. Tabla de características para Jetbrains MPS.

Característica	Valoración
Dificultad de uso	Alta
Necesita complementarse con otras herramientas	No
Necesidad de instalación y configuración	Si
Tiene limitaciones en cuanto al lenguaje de generación	No
Necesita redistribuir la línea de productos en cada cambio	Si
Es multiplataforma	Si

https://www.jetbrains.com/mps/

2.3. Rascal

Rascal pretende ser una herramienta multipropósito dentro del ámbito de la metaprogramación. Se trata de un lenguaje de programación en sí que cuenta con diferentes librerías relacionadas con este ámbito.

Rascal puede ser utilizado a través de la línea de comandos, o también, puede ser integrado dentro del IDE eclipse a través de un plugin.

Tabla 5. Tabla de características para Rascal.

Característica	Valoración
Dificultad de uso	Muy alta
Necesita complementarse con otras herramientas	No
Necesidad de instalación y configuración	Si
Tiene limitaciones en cuanto al lenguaje de generación	No
Necesita redistribuir la línea de productos en cada cambio	Si
Es multiplataforma	Si

https://www.rascal-mpl.org/

2.4. FeatureIDE

FeatureIDE se define como un marco extensible para el desarrollo de software orientado a características. Y como tal, permite la creación de modelos de características y su uso a través de las múltiples extensiones y herramientas compatibles. FeatureIDE también se encuentra basado en el IDE eclipse. Tanto el diseño de los modelos como los mismos modelos pueden crearse de forma visual.

Tabla 6. Tabla de características para FeatureIDE.

Característica	Valoración
Dificultad de uso	Media
Necesita complementarse con otras herramientas	No
Necesidad de instalación y configuración	Si
Tiene limitaciones en cuanto al lenguaje de generación	Si
Necesita redistribuir la línea de productos en cada cambio	Si
Es multiplataforma	Si

https://featureide.github.io/

2.5. Spoofax

Se trata de una plataforma para el desarrollo de lenguajes específicos de dominio. Entre sus características se encuentra la generación de un entorno para el modelado de productos basado en IntelliJ con todas las características derivadas del lenguaje creado.

Tabla 7. Tabla de características para Spoofax.

Característica	Valoración
Dificultad de uso	Alta
Necesita complementarse con otras herramientas	Si
Necesidad de instalación y configuración	Si
Tiene limitaciones en cuanto al lenguaje de generación	No
Necesita redistribuir la línea de productos en cada cambio	Si
Es multiplataforma	Si

http://www.metaborg.org/

2.6. Pure::variants

Del mismo modo que FeatureIDE, Pure::variants es está orientado al desarrollo de software orientado a características. También está basado en el IDE eclipse y cuenta con diferentes módulos que permiten ampliar su funcionalidad. El formato de diseño también guarda ciertas similitudes, pudiendo editar de forma visual la especificación del modelo de características y los propios modelos.

Tabla 8. Tabla de características para Pure::variants.

Característica	Valoración
Dificultad de uso	Alta
Necesita complementarse con otras herramientas	No
Necesidad de instalación y configuración	Si
Tiene limitaciones en cuanto al lenguaje de generación	Si

Necesita redistribuir la línea de productos en cada cambio	Si
Es multiplataforma	Si

https://www.pure-systems.com/

2.7. Gears

Gears es un software propio de la compañía BigLever, está orientado al desarrollo de modelos de características y en más específicamente al manejo de grandes modelos. No está basado en ningún IDE, sino que es una aplicación en sí. Al igual que otras herramientas dirigidas a la especificación y creación de modelos de características, también cuenta con la posibilidad de especificar los modelos de forma visual.

Tabla 9. Tabla de características para Gears.

Característica	Valoración
Dificultad de uso	Media
Necesita complementarse con otras herramientas	No
Necesidad de instalación y configuración	Si
Tiene limitaciones en cuanto al lenguaje de generación	Si
Necesita redistribuir la línea de productos en cada cambio	Si
Es multiplataforma	No

https://biglever.com/solution/gears/

2.8. Resumen

Todas las herramientas analizadas son muy potentes y presentan características que las hacen muy eficaces en determinados proyectos, no obstante, sería importante hacer hincapié sobre lo siguiente:

- Su curva de aprendizaje tiende a ser elevada.
- Requieren conocimientos avanzados en distintas áreas poco extendidas, como la construcción de gramáticas y metaprogramación.
- Se trata de aplicaciones de escritorio, que necesitan una instalación y configuración previa.
- No cubren todas las características necesarias para llevar a cabo el desarrollo de la LPS al completo y necesitan complementarse con otras herramientas.

- Los entornos para el modelado de productos necesitan ser generados y distribuidos con cada cambio en el lenguaje de modelado.

Teniendo en cuenta todo esto, se puede deducir que implantar una LPS utilizando las herramientas disponibles supone un alto desafío. Tanto es así, que podría echar a perder la idea de optar por este enfoque en proyectos, que podrían obtener una gran rentabilidad de ello, pero no pueden soportar este coste inicial.

En cierto modo, las herramientas analizadas ofrecen un amplio abanico de posibilidades, sin embargo, su aplicación en LPS no siempre es lo suficientemente práctico cuando se necesita un proceso rápido y sencillo.

Capítulo 3: Herramienta SPL Studio

SPL Studio es una herramienta enfocada a la ceración de LPS, creada con el objetivo de simplificar y agilizar el proceso de implantación y uso de la misma, integrando en un mismo entorno todo lo necesario para llevar a cabo los procesos exigidos por este enfoque de desarrollo.

La aplicación, a diferencia de otras herramientas utilizadas para el propósito, está basada en tecnología web, otorgando una disponibilidad de uso inmediata y evitando de esta forma la necesidad de configurar un entorno de desarrollo.

3.1. Implementación de SPL Studio

3.1.1. Arquitectura

La aplicación SPL Studio se ha implementado utilizando una arquitectura orientada a microservicios. Este patrón de arquitectura, divide la funcionalidad de la aplicación, en componentes independientes, que pueden estar implementados en diferentes tecnologías, una característica necesaria en este caso. Los componentes o servicios que constituyen la aplicación pueden comunicarse entre sí, y tanto los accesos internos del sistema, como los externos al mismo, deben realizarse a través del componente "getway", lo que supone que todas las comunicaciones tengan un mismo punto de entrada.

Entre las ventajas más importantes que ofrece una arquitectura de microservicios encontramos, escalabilidad, reutilización y una importante mejora en cuanto a la implementación y el mantenimiento de los mismos.

En la imagen **Fig. 3**, se muestra el esquema de la arquitectura que utiliza la aplicación SPL Studio.

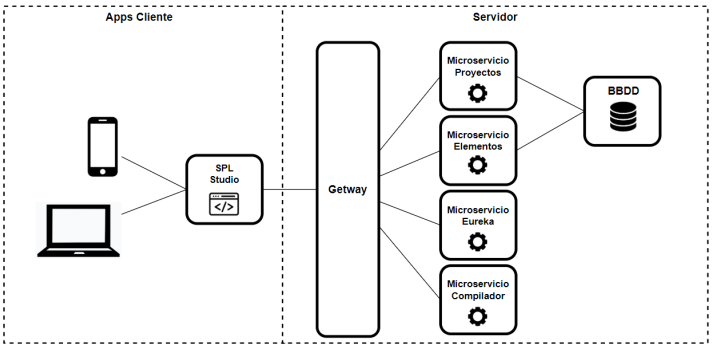


Fig. 3 Esquema de la arquitectura implementada en la aplicación SPL Studio.

En la aplicación podemos diferenciar entre dos tipos de microservicios.

- **Servicios de datos**: Estos servicios tienen acceso a la base de datos y su función es gestionar la información de la misma.
- **Servicios funcionales**: Como su nombre indica, se trata de microservicios con cometidos de tipo funcional.

3.1.2. Tecnología utilizada

Para la implementación de la aplicación, se han utilizado diferentes tecnologías.

- **Java + Spring boot**: Se ha utilizado para crear la infraestructura de microservicios y algunos de estos microservicios.
- **Docker**: Se utiliza para realizar el empaquetado de la aplicación.
- **Nodejs + JavaScript**: Tanto la aplicación cliente SPL Studio, como el microservicio compilador, se han implementado utilizando node.js y JavaScript.
- **Maven**: Se utiliza como gestor de paquetes para los componentes implementados en java.
- **NPM**: Para la gestión de paquetes de las aplicaciones basadas en node.js se utiliza npm.
- **MySQL**: Motor de base de datos que se ha utilizado para el almacenamiento de datos de la aplicación.
- **React**: Framework de frontend utilizado para crear la aplicación cliente SPL Studio.
- **JPA + Hibernate**: Se utiliza en la capa de persistencia de datos de los microservicios de datos.
- **Codemirror**: Se utiliza en la aplicación cliente, se trata de una librería JavaScript para incluir editores de código en aplicaciones web.
- **Vm2**: La tecnología de virtualización JavaScript vm2 constituye un espacio seguro para ejecutar código del usuario en el microservicio compilador.

Fig. 4 Representación del stack tecnológico utilizado en la implementación de la aplicación SPL Studio.

3.1.3. Persistencia de datos

El sistema de base de datos elegido para los microservicios implementados es MySQL, sin embargo, el uso de JPA e Hibernate en la implementación de los microservicios de datos, permite que este sistema sea fácilmente reemplazable, dependiendo del entorno de ejecución en el que se encuentre la aplicación. Además, la arquitectura permite la convivencia entre varios sistemas si esto fuese necesario.

En cuanto al diseño de la misma, presenta un aspecto simple, en el que únicamente existen dos entidades, la entidad más importante es "elementos", que se compone de un nombre de elemento, un tipo que identifica que elemento es y un payload que contiene la información del elemento en formato JSON. Un conjunto de elementos de distintos tipos, conforman un proyecto. En la figura **Fig. 5**, se muestra un esquema de la base de datos.

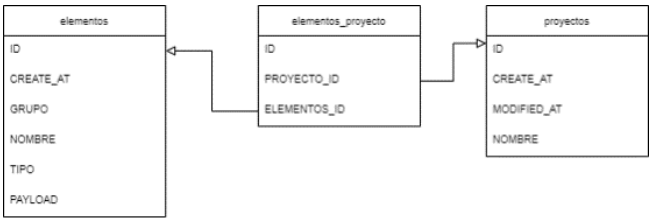


Fig. 5 Diagrama relacional de la base de datos utilizada en SPL Studio.

3.1.4. Microservicios de datos

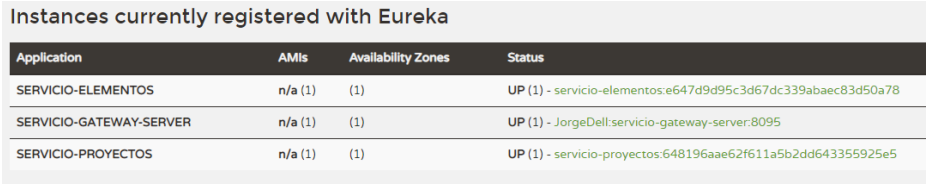
Los microservicios de datos se encargan de la persistencia de datos de la aplicación. En SPL Studio existe un microservicio por cada entidad de negocio, el servicio proyectos y el servicio elementos. Ambos microservicios se han implementado en Java bajo el framework Spring boot y cada uno de ellos gestiona su entidad a utilizando JPA.

3.1.5. Microservicios funcionales

Este tipo de microservicios se encargan de llevar a cabo tareas específicas de tipo funcional. Existen dos servicios en esta categoría:

- **Microservicio eureka**: Este microservicio toma el rol de descubridor. El resto de microservicios debe registrar su ubicación (servidor y puerto) de forma periódica en el microservicio eureka. De esta forma el getway puede redireccionar las peticiones a su lugar correspondiente. Se encuentra implementado en java bajo el framework Spring Boot. En la figura **Fig. 6**, se puede observar la interfaz web del servicio eureka mostrando los microservicios registrados en él.

- **Microservicio compilador**: Este microservicio constituye el corazón de la aplicación. Es el encargado de generar el producto final de la LPS en función de la configuración proporcionada por el usuario.


Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SERVICIO-ELEMENTOS	n/a (1)	(1)	UP (1) - servicio-elementos:e647d9d95c3d67dc339abaec83d50a78
SERVICIO-GATEWAY-SERVER	n/a (1)	(1)	UP (1) - JorgeDell:servicio-gateway-server:8095
SERVICIO-PROYECTOS	n/a (1)	(1)	UP (1) - servicio-proyectos:648196aae62f611a5b2dd643355925e5

Fig. 6 Captura de la interfaz web del microservicio eureka con tres servicios registrados.

Por último, en esta categoría también se puede incluir el componente getway que, aunque no se trata de un microservicio al uso, ya que su responsabilidad es diferente, si se implementa como tal.

3.1.6. Aplicación web cliente SPL Studio

Se trata de la aplicación cliente que ofrece una interfaz web al usuario, desde la que puede operar con todo el sistema. Esta aplicación está desarrollada en JavaScript bajo el framework React.

3.1.7. Interacción de componentes

La imagen de la figura **Fig. 7**, representa el esquema de interacción entre los distintos componentes que de la aplicación. Dejando a un lado el componente getway y el microservicio eureka que forman parte de la infraestructura de microservicios.

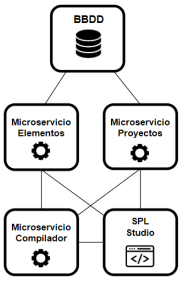


Fig. 7 Esquema de interacción entre los componentes de SPL Studio.

La aplicación SPL Studio, actúa sobre los servidores de datos almacenando los elementos del proyecto, mientras que, por otra parte, el microservicio compilador también actúa sobre los mismos para recuperar información sobre el proyecto. De esta forma, la compilación del proyecto es totalmente independiente de la aplicación cliente, pudiendo ser invocada desde cualquier lugar, como por ejemplo, una terminal. Esta característica permitirá que las LPS puedan ser integradas en diferentes flujos de trabajo, posibilitando entre otras cosas, que los productos finales puedan ser generados de forma automática.

3.2. Espacios de trabajo

Tanto la parte de ingeniería de dominio, como la parte de ingeniería de aplicación cuentan con dos espacios de trabajo (Weiss, 1999): El espacio del problema y el espacio de la solución, por lo que el proceso queda dividido en cuatro espacios de trabajo diferentes:

- Ingeniería de dominio / Espacio del problema.
- Ingeniería de dominio / Espacio de la solución.
- Ingeniería de aplicación / Espacio del problema.
- Ingeniería de aplicación / Espacio de la solución.

La interfaz de SPL Studio sigue el modelo de estos cuatro espacios, dando lugar a cuatro entornos, donde se llevan a cabo las tareas correspondientes.

3.2.1. Ingeniería de dominio / Espacio del problema (EDP)

En este caso, en SPL Studio se intenta simplificar al máximo la creación del dominio, mostrando un método simple de edición, en el cual, simplemente hay que definir la relación entre los diferentes elementos del dominio utilizando el editor visual, como se muestra en la imagen **Fig. 8**.

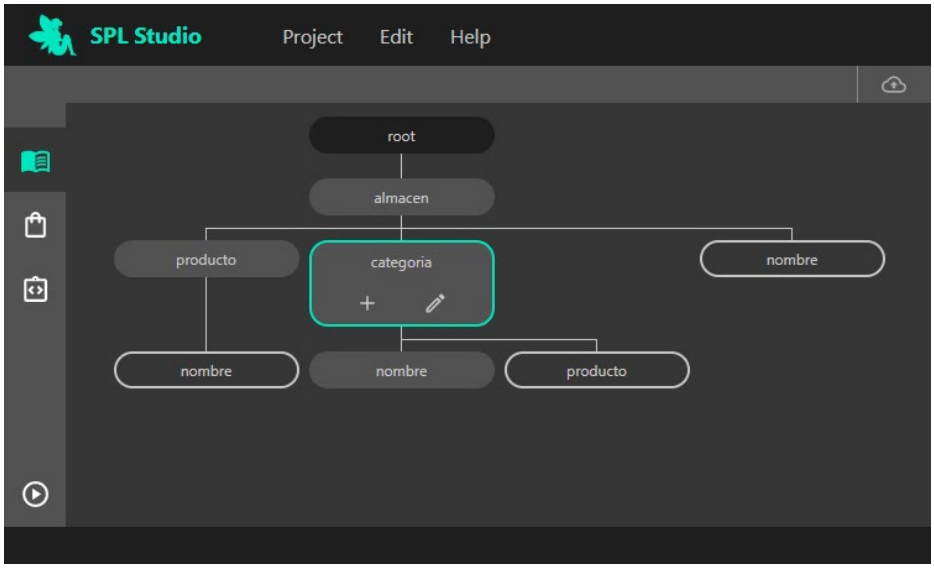


Fig. 8 Entorno de desarrollo de SPL estudio para el espacio del problema del ingeniero de dominio.

En SPL Studio, los elementos del dominio se representan a través de un grafo. Cada uno de estos nodos supone un punto configurable dentro de la LPS, es decir, a partir de cada nodo, será posible gestionar la variabilidad. Las reglas que deben cumplir los nodos son las siguientes:

- Todos los lenguajes parten de un nodo principal, el nodo “root”.
- Cada nodo puede tener varios hijos y varios padres, excepto el nodo “root”, que solo puede tener hijos y por lo tanto ningún nodo puede tener como hijo el nodo “root”.
- Se permite la recursividad de nodos, es decir, un nodo puede tenerse a sí mismo como hijo.
- Los nombres de los dominios solo pueden contener caracteres alfanuméricos y barras bajas, tampoco se permiten mayúsculas.

En la izquierda de la figura **Fig. 9**, se muestra un ejemplo de las referencias posibles en un grafo de nodos para la construcción de un dominio. En la derecha se puede observar un ejemplo de un posible dominio.

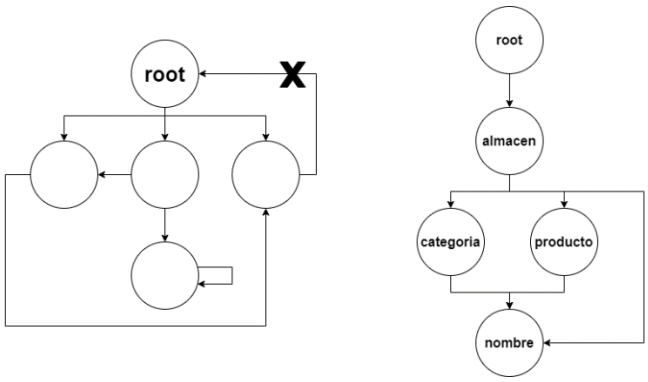


Fig. 9 Ejemplos de referencias para la construcción de dominios.

El editor de dominio, permite añadir información adicional en cada uno de los elementos, por lo que es posible especificar como se caracterizan en cada nodo, los miembros de la familia.

A partir del dominio creado, SPL Studio genera automáticamente los siguientes artefactos derivados del mismo:

- **Un DSL**: Se trata de un lenguaje de modelado basado en el dominio. La gramática de este lenguaje se genera siempre de la misma forma. El hecho de que esto sea así tiene sus ventajas, si el ingeniero de aplicación trabaja con varias LPS, no necesitará aprender la gramática específica para cada una de ellas, pues todas las gramáticas mantienen las mismas reglas y el uso de una

nueva gramática no implicará el aprendizaje de la misma, simplemente se necesita la especificación en base al dominio.

- **Un analizador de código estático basado en el DSL**: El analizador de código ayuda a la hora de modelar los productos resaltando las palabras reservadas del lenguaje de modelado para asistir al ingeniero de aplicación en este proceso.
- **Un analizador de sintaxis para el DSL**: Basándose en las relaciones entre los elementos del dominio, se genera un analizador para el DSL que detecta errores en la sintaxis de los modelos creados.

El dominio puede ser modificado para abordar nuevos puntos de variabilidad que puedan surgir durante la construcción. En cada cambio, todos los artefactos serán adaptados al nuevo dominio.

3.2.2. Ingeniería de aplicación / Espacio del problema (EAP)

En SPL Studio, hay un espacio dedicado para el modelado de los miembros de la LPS. Par ello se proporciona un editor de código, donde el ingeniero de aplicación puede especificar los productos utilizando el lenguaje de modelado derivado del dominio.

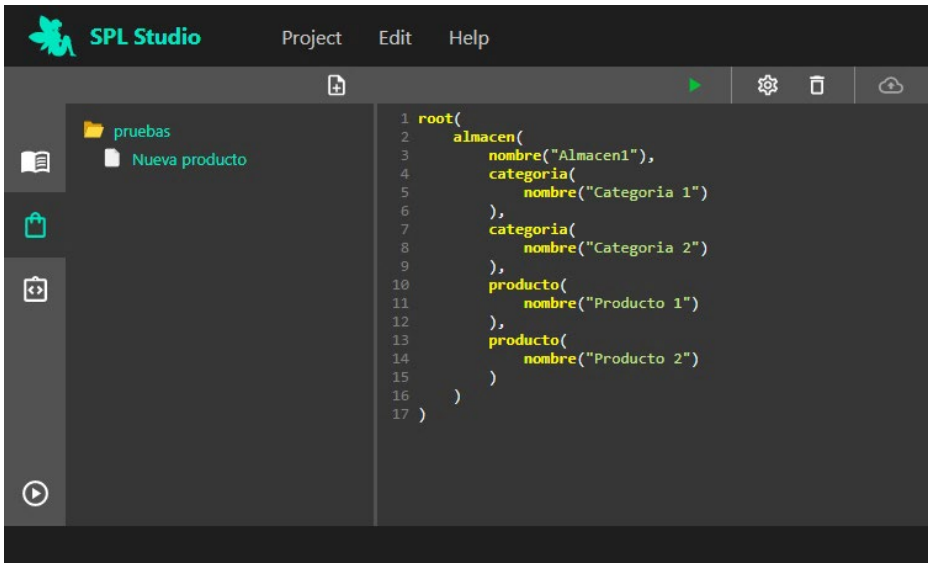


Fig. 10 Entorno de desarrollo de SPL estudio para el espacio del problema del ingeniero de aplicación**.**

Este lenguaje de modelado generado es sencillo, consiste en el anidamiento mediante paréntesis de unos elementos del dominio dentro de otros. Cuando un elemento contiene varios componentes anidados, estos pueden separarse mediante comas. De

esta forma, no es necesario depender de una especificación para la gramática, si no que el dominio en sí se puede utilizar como tal.

En la imagen **Fig. 10**, se puede observar el entorno proporcionado por SPL Studio para el modelado de productos.

El siguiente ejemplo, muestra un producto creado, utilizando el lenguaje de modelado generado a partir del dominio del ejemplo de **Fig. 9**.

```
root(
    almacen(
    nombre("Almacén 1"),
        categoria(
            nombre("Categoría 1")
        ),
        categoria(
            nombre("Categoría 2")
        ),
        producto(
            nombre("Producto 1")
        ),
        producto(
            nombre("Producto 2")
        )
    )
)
```

El editor de código proporcionado por SPL Studio resalta el código basándose en el dominio para ayudar a evitar errores durante el modelado de productos.

El entorno permite modelar tantos productos como se necesite para una misma LSP, además permite organizar los modelos en grupos. Asimismo, cuando se especifica un nuevo producto, SPL Studio crea un prototipo de especificación de producto que agiliza el proceso de modelado.

A continuación, se muestra un prototipo de producto generado automáticamente por SPL Studio para el ejemplo de dominio de la figura **Fig. 9**.

```
root(
    almacen(
        producto(
            nombre("")
        ),
        categoria(
            nombre(""),
            producto(
                nombre("")
            )
        ),
        nombre("")
    )
)
```

El lenguaje generado por SPL Studio está basado en JavaScript, lo que permite utilizar toda la potencia que ofrece este lenguaje, como el uso de variables, funciones, estructuras de control, etc.

3.2.3. Ingeniería de dominio / Espacio de la solución (EDS)

SPL Studio proporciona un entorno para la solución correspondiente a la ingeniería de dominio, donde también ofrece un editor de código destinado a la creación de plantillas, que serán utilizadas para la generación del código y la documentación del producto final.

Cada una de las plantillas genera uno o varios ficheros del producto final. Las plantillas se ordenan en familias y cada una de estas familias es capaz de generar un producto final a partir de una especificación.

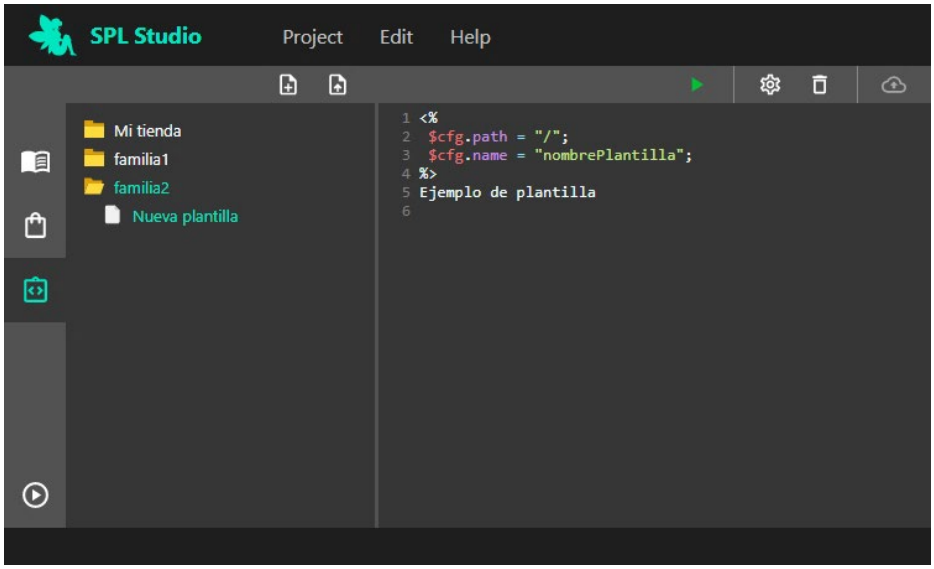


Fig. 11 Entorno de desarrollo de SPL estudio para el espacio de la solución del ingeniero de dominio.

En la figura **Fig. 11**, se presenta el espacio de trabajo que ofrece SPL Studio para el espacio de la solución del dominio. En él, se intenta simplificar el proceso de creación de plantillas, ofreciendo mecanismos para resolver los problemas más comunes. A continuación, se detallan algunos de ellos.

3.2.3.1. Anclas

El ancla es un mecanismo que tiene como objetivo asociar las plantillas a la especificación del modelo, con el fin de automatizar el proceso de generación de ficheros.

En la figura **Fig. 12**, se representa el anclaje de las plantillas a los elementos del modelo, generando tantos ficheros como elementos existan. En este caso la plantilla asociada a "root" genera solo un fichero, mientras que la plantilla asociada a "nombre" genera varios.

El ancla se puede configurar aún más si se necesita especificar la posición del elemento del modelo al que se quiere asociar, para ello se puede indicar la ruta de los componentes mediante puntos, por ejemplo: "producto.nombre", esta ancla solo genera ficheros para los nombres que estén incluidos en un producto.

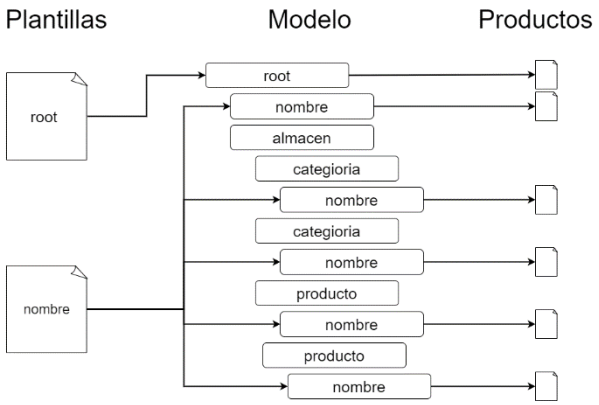


Fig. 12 Esquema de generación de ficheros utilizando anclas asociadas a los elementos del modelo.

3.2.3.2. Scriptlets

Los scriptlets permiten aplicar lógica en una plantilla. El uso de scriptlets no es nuevo, es utilizado en tecnologías como JSP o ERB, por lo que el ingeniero de dominio puede estar familiarizado ya con esta nomenclatura antes de utilizar SPL Studio.

Estos scriptlets permiten la encapsulación de fragmentos de código dentro de la plantilla. En SPL Studio, se ha implementado un motor de scriptlets, donde el lenguaje de programación utilizado en ellos es JavaScript.

En la **Tabla 10** se muestran los tres tipos de scriptlets disponibles para la creación de plantillas.

Tabla 10. Scriptlets disponibles para la creación de plantillas.

Scriptlet	Funcionalidad
<%= . . . %>	Permite renderizar una variable JavaScript sobre la plantilla.
<% . . . %>	Permite insertar lógica en la plantilla.
<%-- . . . --%>	Permite añadir comentarios sobre la plantilla.

3.2.3.3. Variables

Durante la codificación de las plantillas se dispone de algunas variables proporcionadas por el contexto de la compilación, en la tabla **Tabla 11**, se presentan dichas variables.

Tabla 11. Variables especiales para la codificación de plantillas.

Scriptlet	Funcionalidad
$self	Hace referencia al elemento del modelo al que se ha anclado la plantilla.
$root	Hace referencia al elemento root.
$cfg	Permite configurar el nombre y la ubicación del fichero generado mediante sus atributos path y name.

Tanto la variable "$self" como la variable "$root" hacen referencia a un elemento del modelo. Desde estas variables es posible acceder a los elementos hijo siguiendo el grafo del modelo, por ejemplo, si la variable "$self "estuviese anclada a un elemento "producto" del modelo de ejemplo de la figura **Fig. 9**, sería posible acceder al nombre de la forma "$self.nombre[i]", donde i es la posición del nombre buscado en el caso de que existiesen varios elementos de tipo nombre en el subnivel.

Además, las variables de tipo elemento poseen algunos atributos que pueden utilizarse para la construcción de las plantillas, en la **Tabla 12** se pueden observar los más importantes.

Tabla 12. Atributos de los elementos

Scriptlet	Funcionalidad
key	Accede al nombre del elemento
value	Accede al valor del elemento.
index	Indica la posición del elemento respecto a otros elementos del mismo nivel.
deep	Indica el nivel de profundidad del elemento en el modelo.
parent	Hace referencia al elemento padre.
all	Contiene un array con todos los hijos, independientemente del tipo que sean.

Siguiendo el ejemplo anterior, se podría acceder a los diferentes atributos del elemento "$self.nombre[i]", por ejemplo "$self.nombre[i].value" para obtener su valor, o "$self.nombre[i].all", para obtener el array de todos sus hijos.

Con estas simples bases, se hace posible gestionar la generación de código en todos sus aspectos.

Cabe destacar, que SPL Studio cuenta con un asistente para la creación de plantillas a partir de un prototipo, lo que hace que el desarrollo de las mismas sea muy rápido, teniendo en cuenta que solo habría que trabajar sobre la variabilidad de estas.

3.2.4. Ingeniería de aplicación / Espacio de la solución (EAS)

Por último, SPL Studio proporciona un espacio de trabajo para la solución de la ingeniería de aplicación. En este espacio se genera el producto final a partir de los artefactos creados en los espacios anteriores.

No es necesario realizar ninguna acción en este espacio, SPL Studio se encarga de compilar directamente el modelo seleccionado en el espacio EAP, junto con la familia de productos seleccionada en el espacio EDS. Todo ello empleando las reglas definidas en el dominio creado en el espacio EDP.

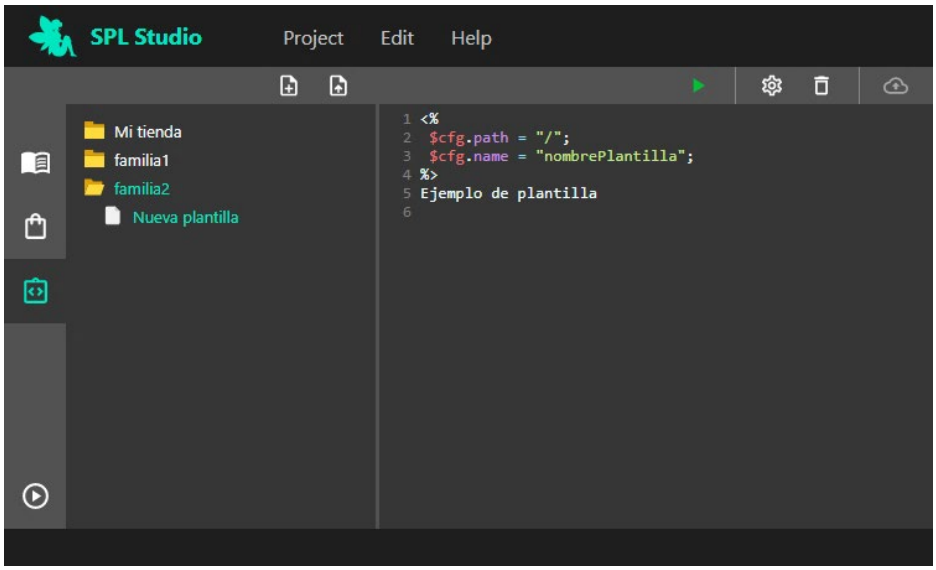


Fig. 13 Entorno de desarrollo de SPL estudio para el espacio de la solución del ingeniero de aplicación.

El espacio de trabajo EAS, además de permitir la visualización y la descarga de los componentes, también permite la depuración de los mismos en caso de que se produzcan errores de compilación.

En la imagen **Fig. 13**, se presenta el espacio de trabajo que ofrece SPL Studio para la solución de la ingeniería de la aplicación.

En la figura **Fig. 14**, se muestra un esquema del proceso de generación del producto final utilizado por SPL Studio.

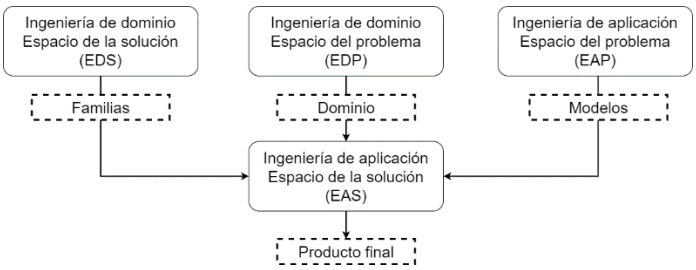


Fig. 14 Esquema de producción de software a través de los espacios de trabajo en SPL Studio.

3.3. Entorno de trabajo en SPL Studio

La aplicación SPL Studio, cuenta con dos zonas de trabajo, la zona de gestión, donde se gestionan los proyectos y configuración de la aplicación, y la zona de edición, enfocada a la construcción de proyectos.

3.3.1. Zona de gestión.

Se trata de la pantalla principal que encontramos cuando se accede a SPL Studio, desde esta pantalla es posible gestionar los diferentes proyectos del usuario, así como la configuración de SPL Studio.

3.3.1.1. Menú lateral

En la parte izquierda, se encuentra el menú lateral, este menú permite el acceso a las diferentes secciones de la zona de gestión a través de los siguientes iconos:

Icono de acceso a la pantalla principal

Icono de acceso a la gestión de proyectos

Icono de acceso a la configuración

En la parte izquierda de la figura **Fig. 15**, se puede observar la disposición que toman los iconos en el panel de gestión, donde aparece resaltado el icono correspondiente a la sección seleccionada.

3.3.1.2. Pantalla de Bienvenida.

La pantalla de bienvenida no es más que una simple pantalla que ofrece al usuario información básica sobre la aplicación, como se muestra en la figura **Fig. 15**.

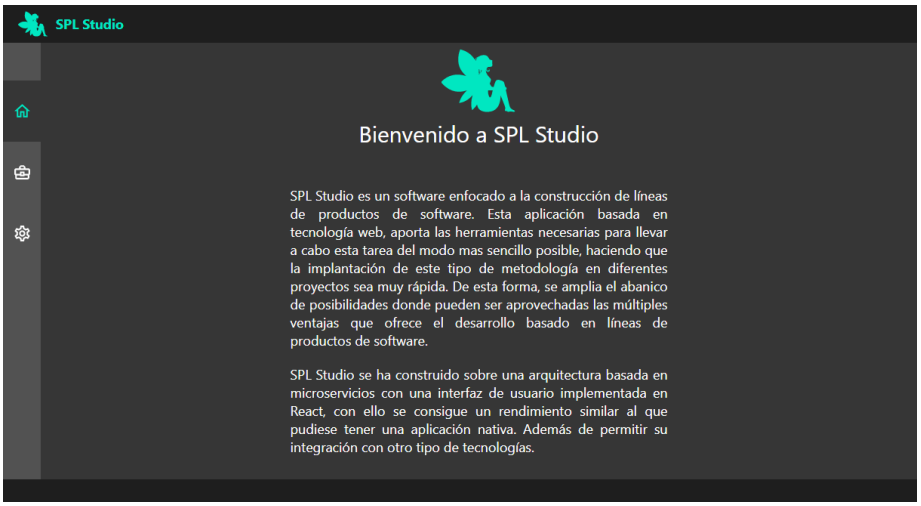


Fig. 15 Captura de la pantalla de bienvenida de SPL Studio.

3.3.1.3. Pantalla de gestión de proyectos

La pantalla de gestión de proyectos muestra un listado con todos los proyectos del usuario, se puede observar la lista de proyectos en la figura **Fig. 16**.

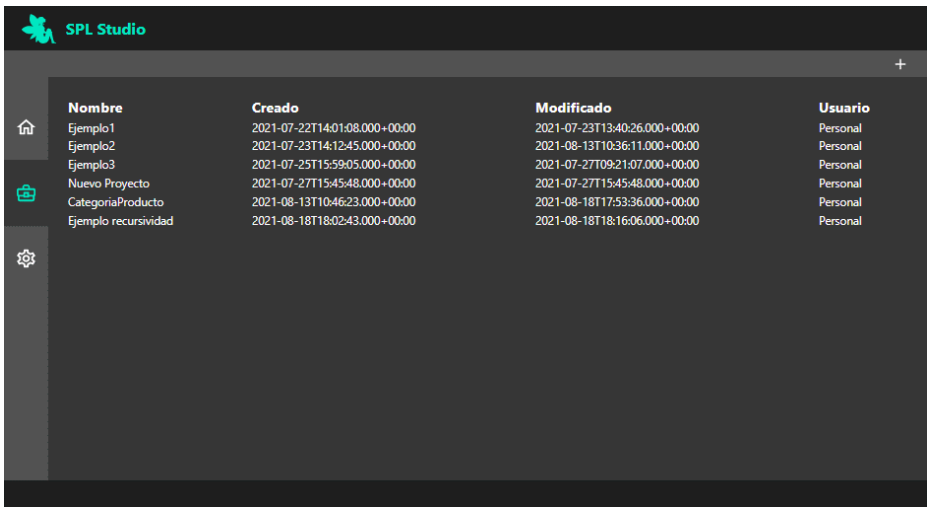


Fig. 16 Captura de la pantalla de gestión de proyectos de SPL Studio.

En la parte superior derecha se encuentra el icono de nuevo proyecto, en la figura **Fig. 17**, se puede ver resaltado con una flecha roja.

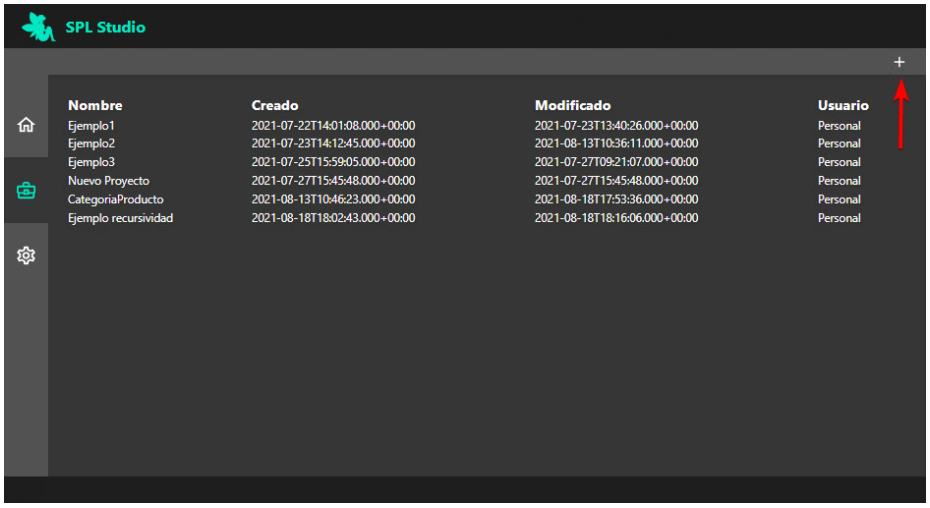


Fig. 17 Captura de la pantalla de gestión de proyectos en SPL Studio.

Si se pulsa sobre este icono, aparece una pantalla para insertar el nombre del nuevo proyecto que se va a crear, como se muestra en la figura **Fig. 18**.

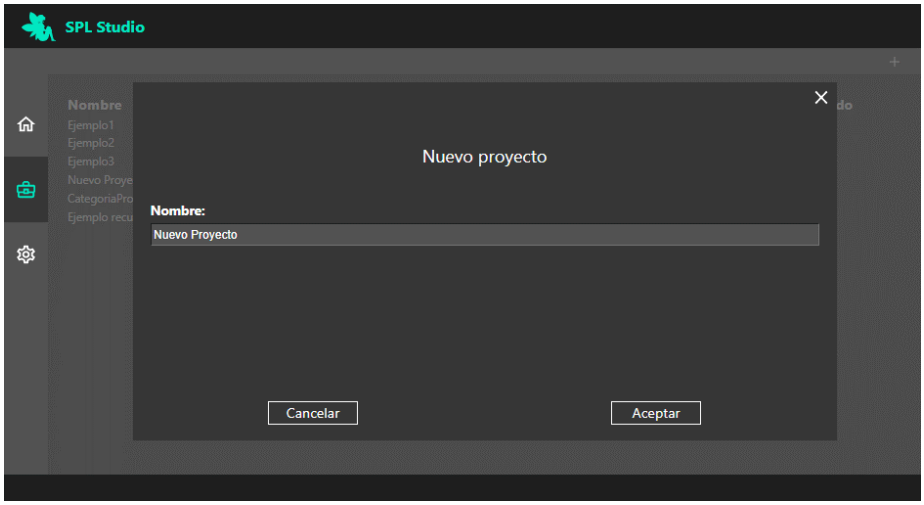


Fig. 18 Captura de la pantalla de creación de nuevo proyecto en SPL Studio.

3.3.1.4. Pantalla de configuración

La pantalla de configuración permite cambiar tanto el idioma de la aplicación, como el tema, como se puede ver en la figura **Fig. 19**.

Fig. 19 Captura de la pantalla de configuración en SPL Studio.

Actualmente es posible alternar entre los idiomas inglés y español, además es posible seleccionar tres temas diferentes que permiten personalizar el aspecto de SPL Studio, en la figura **Fig. 20**, se exponen diferentes ejemplos de configuración.

Fig. 20 Ejemplos de diferentes configuraciones de tema e idioma para SPL Studio.

3.3.2. Zona de edición

La zona de edición ofrece un entorno para la elaboración de proyectos. Se encuentra dividida en cuatro apartados principales.

3.3.2.1. Menú lateral

En la parte izquierda se encuentra el menú lateral que permite el acceso a los diferentes espacios de trabajo de la zona de edición a través de los siguientes iconos:

Icono de acceso al editor de lenguaje

Icono de acceso al editor de productos

Icono de acceso al editor de plantillas

Icono de acceso a la pantalla de compilación

3.3.2.2. Editor de lenguaje

El editor de lenguaje tiene como propósito proporcionar las herramientas necesarias para la creación y edición del lenguaje de dominio de la LPS.

El editor de lenguaje se compone de una barra de herramientas en la parte superior y un panel de edición, que muestra el lenguaje en forma de árbol y ofrece todo lo necesario para su gestión.

En la figura **Fig. 21**, se presenta la pantalla de edición para el lenguaje de dominio que ofrece SPL Studio, además, se muestra un lenguaje de dominio ya creado donde se puede observar la jerarquía de los elementos del mismo representada en forma de árbol, donde los elementos inferiores son hijos de los elementos superiores. En este caso, el lenguaje de dominio corresponde con el ejemplo de la figura **Fig. 9**.

Fig. 21 Captura de la pantalla de edición de lenguaje de dominio en SPL Studio.

Por defecto todos los lenguajes tienen el nodo "root", por lo que la primera vez que se accede, únicamente aparecerá este nodo y a partir del mismo se comenzará la creación del lenguaje.

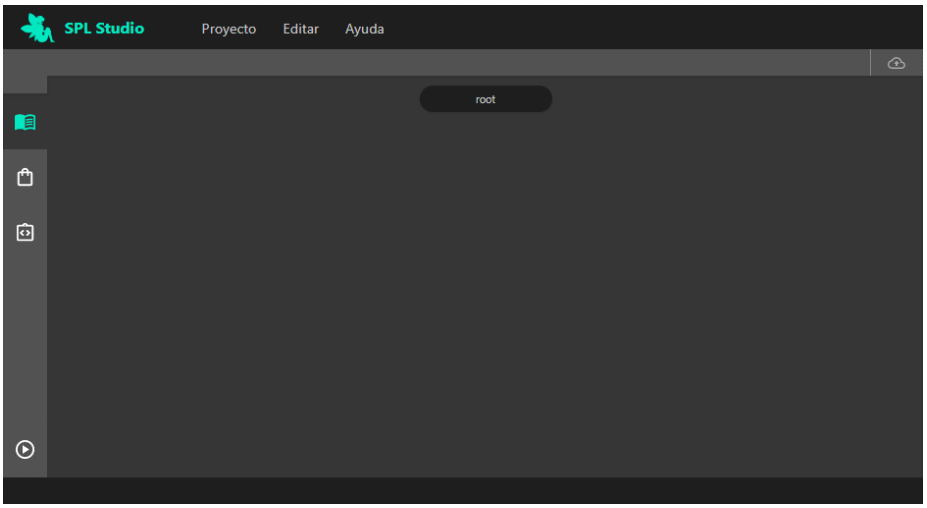


Fig. 22 Captura de la pantalla de edición de lenguaje de dominio en SPL Studio.

En la siguiente figura **Fig. 22**, se muestra una captura de la pantalla de edición de lenguaje con el lenguaje inicial que se crea por defecto, se puede observar cómo en este caso únicamente aparece el nodo "root" como punto de partida para comenzar la creación.

3.3.2.2.1. Nodos

Existen tres tipos de nodos que pueden diferenciarse por su estilo dentro del panel de edición.

3.3.2.2.1.1. Nodo root

Se trata del nodo principal. Aparece por defecto al crear un proyecto y únicamente ofrece la posibilidad de añadir nodos hijos, por lo que está presente en todos los lenguajes. Dentro del editor de lenguaje, se encuentra siempre en la parte superior del árbol de nodos.

En la figura **Fig. 23**, se observa un lenguaje de dominio con el nodo "root" expandido.

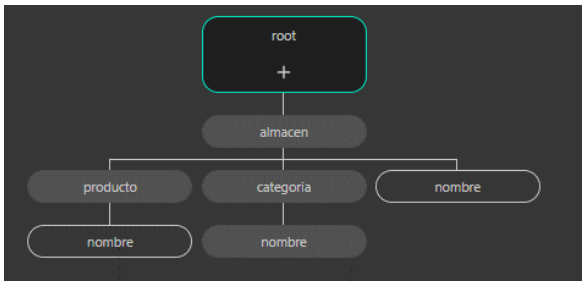


Fig. 23 Captura de lenguaje editado en el editor de SPL Studio con nodo "root" expandido.

3.3.2.2.1.2. Nodo principal

Cualquier elemento del lenguaje es representado por un nodo principal. Este tipo de nodo ofrece la opción de añadir nodos hijo, editar el nodo o eliminarlo siempre que este no tenga hijos.

En la figura **Fig. 24**, se observa un lenguaje de dominio con el nodo principal "nombre" expandido.

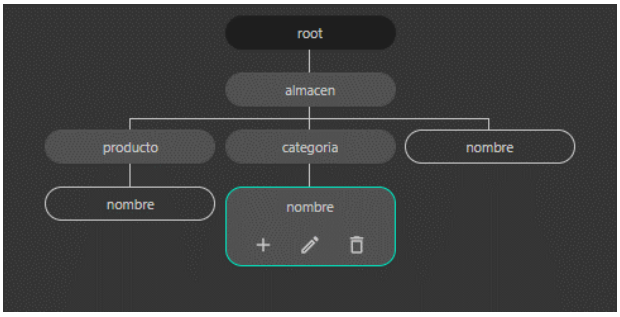


Fig. 24 Captura de lenguaje editado en el editor de SPL Studio con nodo principal expandido.

3.3.2.2.1.3. Nodo referencia

Se trata de una referencia a un nodo principal, pueden existir tantos como se deseen y en un momento dado, pueden ser transformados en nodo principal, pasando a ser el anterior nodo principal, un nodo referencia.

En la figura **Fig. 25Fig. 24**, se observa un lenguaje de dominio con el nodo referencia "nombre" expandido.

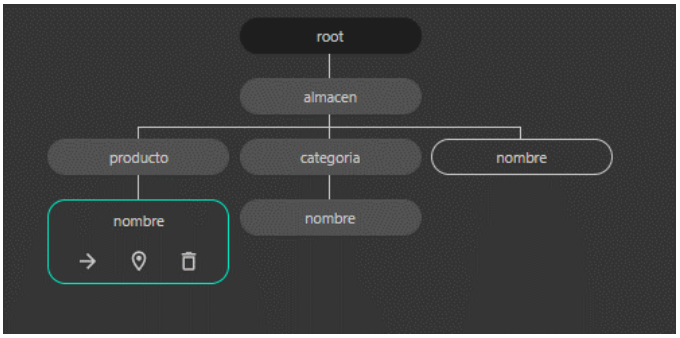


Fig. 25 Captura de lenguaje editado en el editor de SPL Studio con nodo referencia expandido.

Un nodo referencia, representa al nodo principal que está referenciando, por lo que tendrá los mismos hijos y propiedades que este, su única diferencia es la representación visual.

3.3.2.2.2. Opciones

Al pulsar sobre un nodo y dependiendo del tipo de nodo, se muestran las opciones disponibles para el mismo. A continuación, se detallan las opciones que pueden aparecer.

3.3.2.2.2.1. Crear nueva palabra reservada o referencia

La opción crear nueva palabra reservada o referencia, permite añadir un nodo hijo descendiente del nodo desde el que se pulsa. En la figura **Fig. 26**, se muestra el icono que permite crear nuevos elementos a partir de un nodo seleccionado.

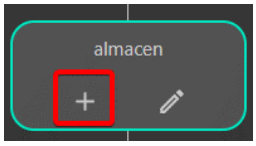


Fig. 26 Detalle del icono "nueva palabra reservada o referencia".

Solo los nodos principales presentan esta opción, ya que la edición del nodo principal, afecta de igual forma a todos los nodos que lo referencian.

Al pulsar sobre él, se muestra una ventana para la inserción de una nueva palabra reservada o referencia, como se puede observar en la figura **Fig. 27**.

Esta ventana permite por un lado seleccionar un nodo ya existente, creando de esta forma un nodo referencia, o crear un nodo principal escribiendo una nueva palabra reservada en el cuadro de texto.

Fig. 27 Cuadro de dialogo para la creación de un nuevo elemento del lenguaje.

3.3.2.2.2.2. Editar

La opción editar permite modificar las características de un nodo. Esta opción, al igual que la opción anterior, solo aparece en los nodos principales, pues la modificación de un nodo principal afecta a todos los nodos que le referencian. En la figura **Fig. 28**, se muestra el icono que permite realizar la edición.

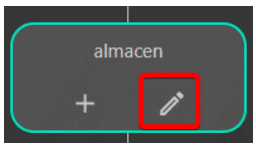


Fig. 28 Detalle del icono “editar palabra reservada o referencia”.

Al pulsar sobre la opción, aparecerá la ventana de edición de nodos, esta ventana se puede ver en la imagen **Fig. 29**. En ella, se pueden editar las siguientes opciones:

- **Palabra reservada**: Permite cambiar la palabra reservada.
- **Descripción**: Permite añadir información extra para una palabra reservada

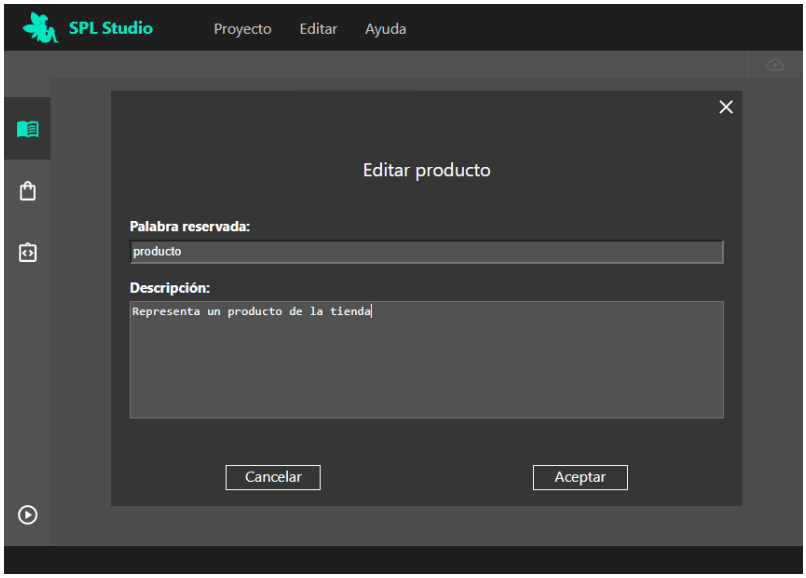


Fig. 29 Cuadro de dialogo para la edición de un elemento del lenguaje.

3.3.2.2.2.3. Ver nodo principal

La opción ver nodo principal, permite al usuario desplazarse desde un nodo referencia hasta su nodo principal correspondiente. En la figura **Fig. 30**, se muestra el icono para desplazarse hacia el nodo principal.

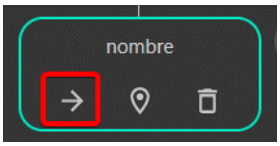


Fig. 30 Detalle del icono “ver nodo principal”.

Esta opción solo aparece en los nodos referencia y al ser pulsada se colapsa el nodo referencia y se expande el nodo principal. Además, si el nodo está fuera del área de visión, automáticamente se realiza un desplazamiento sobre el diagrama para mostrar el nodo principal.

En la figura **Fig. 31**, se presenta el comportamiento que ofrece este icono. En la parte izquierda se muestra el icono antes de ser pulsado dentro del nodo “nombre”, mientras

que, en la parte derecha, se muestra el resultado final, en el cual, el nodo referencia ha sido colapsado y se ha colocado el foco sobre el nodo principal.

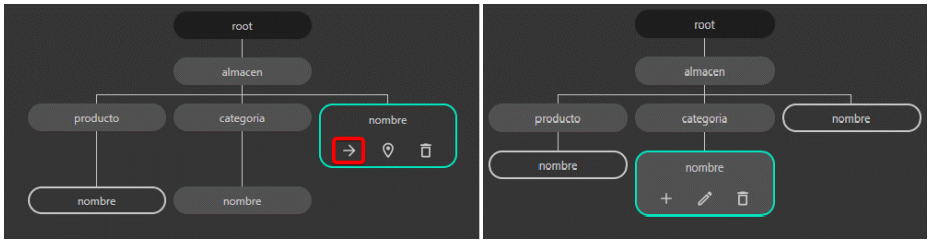


Fig. 31 Funcionamiento del icono "ver nodo principal".

3.3.2.2.2.4. Establecer como nodo principal

Esta opción permite que un nodo referencia pase a ser un nodo principal, estableciendo el nodo principal anterior como nodo referencia. En la figura **Fig. 32**, se muestra el aspecto que toma este icono.

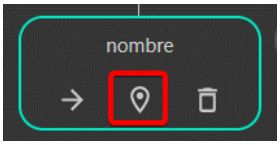


Fig. 32 Detalle del icono "establecer como principal"

En la figura **Fig. 33**, se muestra el comportamiento del icono “establecer como principal”. En la parte izquierda podemos ver el estado nodo referencia antes de pulsar el icono, mientras que, en la parte derecha se puede observar el estado una vez el icono ha sido pulsado, en este caso, el nodo “nombre” de tipo referencia sobre el que ha sido pulsado el icono, se ha transformado nodo principal y el nodo “nombre” que anteriormente era el principal, ha sido transformado en un nodo referencia.

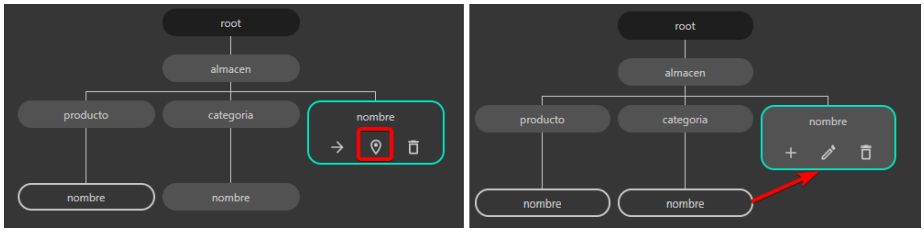


Fig. 33 Funcionamiento del icono "establecer como principal".

Esta opción solo está disponible en los nodos referencia y siempre que estos no presenten recursividad en su estructura, como se muestra en el ejemplo de la figura **Fig. 34**, donde el nodo principal categoría tienen como hijo una referencia de sí mismo y,

donde este nodo referencia, no puede ser transformado en principal, puesto que la representación gráfica no sería posible.

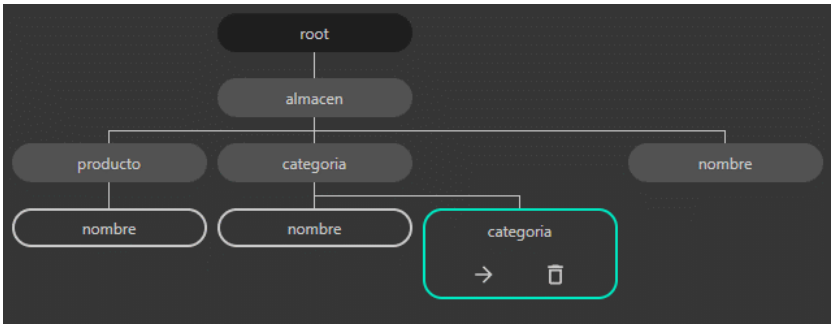


Fig. 34 Ejemplo de recursividad entre nodos.

3.3.2.2.2.5. Eliminar

La opción eliminar permite la eliminación de un nodo. En la imagen **Fig. 35**, se puede ver el icono "eliminar" dentro del nodo referencia nombre.

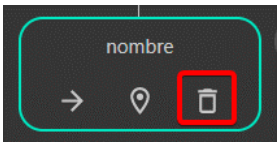


Fig. 35 Detalle del icono "eliminar".

Este icono, aparece en los nodos referencia y en los nodos principales, siempre y cuando, no tengan ningún hijo. Por ello, para eliminar un nodo que tenga hijos, antes es necesario eliminar cada uno de sus hijos, este mecanismo previene la aparición de errores por parte del usuario que, de otra forma, podría eliminar toda una rama de nodos por accidente.

En la figura **Fig. 36**, se puede observar como el icono "eliminar" no aparece en el ejemplo de la parte derecha para el elemento "categoría" por tener este un hijo ("nombre"), pero si aparece en el ejemplo de la parte derecha, donde el elemento "nombre", no tiene ningún hijo que impida su eliminación.

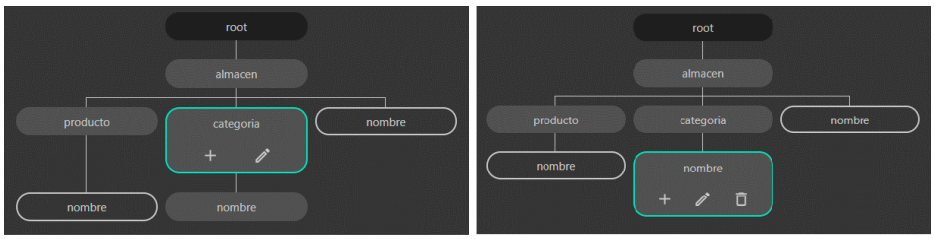


Fig. 36 Detalle del icono "eliminar" para nodos con hijos y sin hijos.

Cuando se pulsa sobre la opción eliminar, siempre aparece una ventana de confirmación para evitar el borrado de nodos por error, como se puede observar en la figura **Fig. 37**.

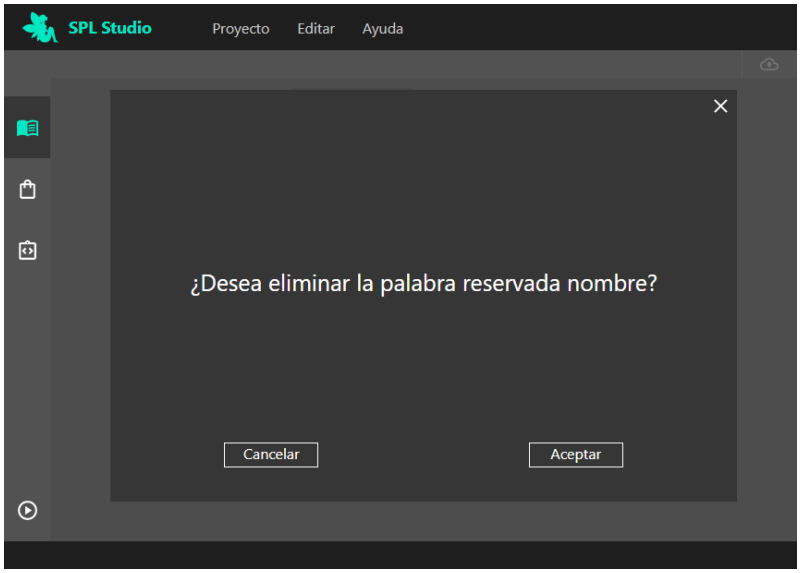


Fig. 37 Pantalla de confirmación para la eliminación de nodos.

3.3.2.2.3. Barra de herramientas

En la barra de herramientas del editor de lenguaje se pueden encontrar los siguientes iconos:

Icono guardar habilitado: Al pulsar sobre él, guarda los cambios en la nube.

Icono guardar deshabilitado: Indica que todos los cambios se han guardado.

Icono guardar habilitado en rojo: Indica que se ha producido un error durante el guardado.

3.3.2.3. Editor de productos

El editor de productos, permite la creación de los diferentes productos de la línea de productos de software, a través de un editor de código.

En la imagen **Fig. 38**, aparecen numerados (en rojo), los diferentes apartados que conforman la vista. Estos corresponden con los siguientes componentes:

1. Barra de herramientas para el panel lateral.
2. Barra de herramientas para el panel principal.
3. Panel lateral.
4. Panel principal.

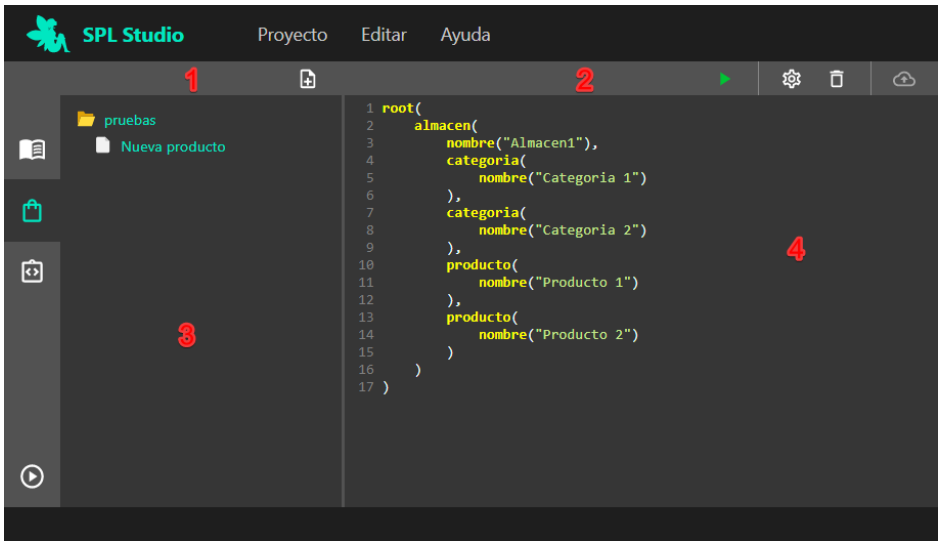


Fig. 38 Interfaz del editor de productos.

3.3.2.3.1. Panel lateral

El panel lateral permite la gestión de diferentes productos, estos pueden ser ordenados en diferentes grupos que serán mostrados como carpetas. En la figura **Fig. 39**, se muestra una captura del panel lateral, donde se pueden observar dos grupos y dentro de uno de los grupos, un producto.

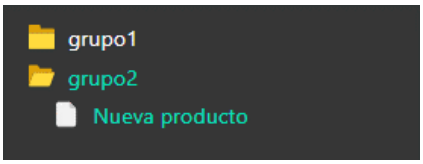


Fig. 39 Detalle del panel lateral del editor de productos

Al pulsar sobre un grupo, este se expande mostrando los productos que contiene, mientras que, a su vez, el resto de grupos se contraen. Cuando se pulsa sobre un producto, este es cargado inmediatamente en el panel principal para su edición.

3.3.2.3.2. Barra de herramientas del panel lateral

La barra de tareas del panel lateral posee un único elemento.

Icono nuevo producto: Permite la creación de un nuevo producto. Al pulsar sobre esta opción, en el panel principal se muestra una ventana que permite seleccionar el nombre y el grupo del nuevo producto. En la figura **Fig. 40**, se puede ver esta ventana. Si se pulsa el botón aceptar, se creará un nuevo producto que automáticamente aparecerá en el panel lateral. De lo contrario si se pulsa el botón cancelar no se realizará ninguna acción y la ventana será cerrada.

Cabe destacar que, al crear un nuevo producto, este ya se encuentra inicializado, es decir, ya cuenta con la estructura básica para comenzar con su configuración.

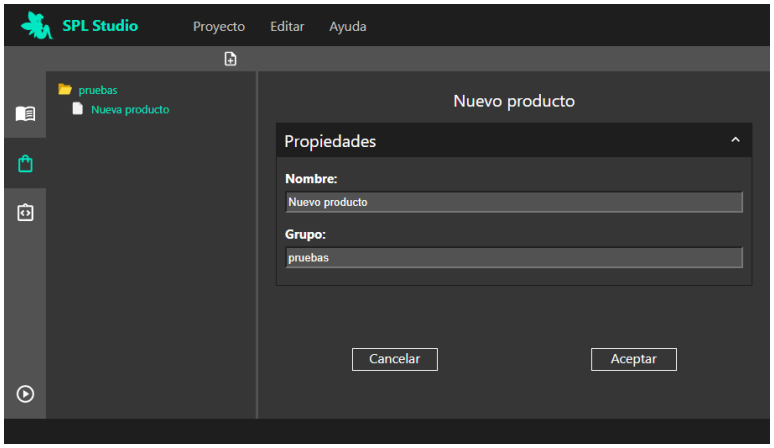


Fig. 40 Pantalla de creación de nuevos productos.

En la imagen **Fig. 41**, se puede ver un nuevo producto inicializado de forma automática, en base al lenguaje del proyecto.

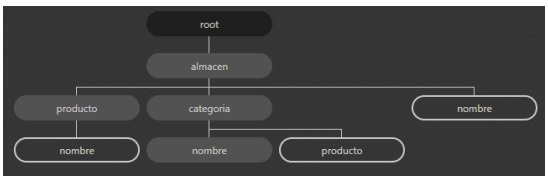
```
root(
    almacen(
        producto(
            nombre("")
        ),
        categoria(
            nombre(""),
            producto("")
        ),
        nombre("")
    )
)
```

Fig. 41 Ejemplo de producto inicializado automáticamente.

3.3.2.3.3. Panel principal

El panel principal es en sí un editor de texto a través del cual se editan los diferentes productos. Este editor cuenta con funcionalidades para reconocimiento de sintaxis, que permite analizar la sintaxis de los productos en función del lenguaje, y de esta forma resaltar los elementos para una mejor visualización.

En imagen **Fig. 42**, se muestra el funcionamiento del analizador de sintaxis generado a partir del lenguaje, se puede apreciar cómo se resaltan los elementos correspondientes al lenguaje del proyecto en amarillo, mientras que los que no son propios del mismo no se ven resaltados.




```
root(
    almacen(
        nombre("Almacen1"),
        categoria(
            nombre("Categoria 1")
        ),
        categorias(
            nombre("Categoria 2")
        ),
        producto(
            nombre("Producto 1")
        ),
        productoo(
            nombre("Producto 2")
        )
    )
)
```

Fig. 42 Ejemplo del analizador de sintaxis del editor de productos.

3.3.2.3.4. Barra de herramientas del panel principal

En la barra de herramientas del editor de lenguaje se pueden encontrar los siguientes iconos:

Icono de compilación: Realiza la compilación del proyecto incluso si no se han realizado cambios desde la última compilación.

Icono de configuración: Permite editar la configuración de un producto. Al pulsar sobre esta opción, aparece una ventana sobre el panel principal que permite modificar el nombre y el grupo del producto seleccionado, como se puede observar en la figura **Fig. 43**.

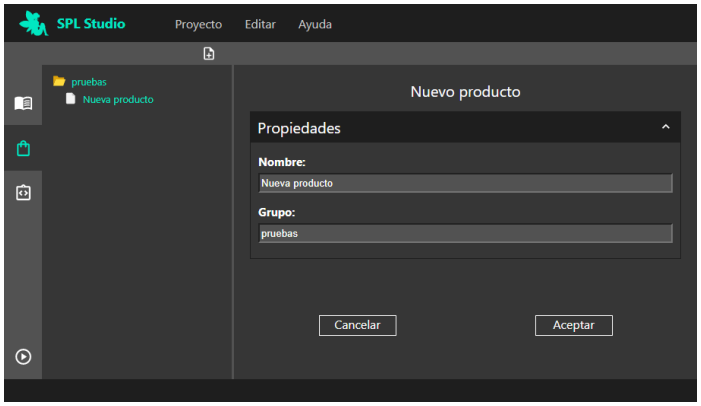


Fig. 43 Ventana de configuración de un producto.

Icono de eliminar: Permite eliminar el producto seleccionado. Siempre que se pulse sobre el mismo, aparecerá una ventana de confirmación, como se puede observar en la imagen **Fig. 44,** para evitar la eliminación de productos por error.

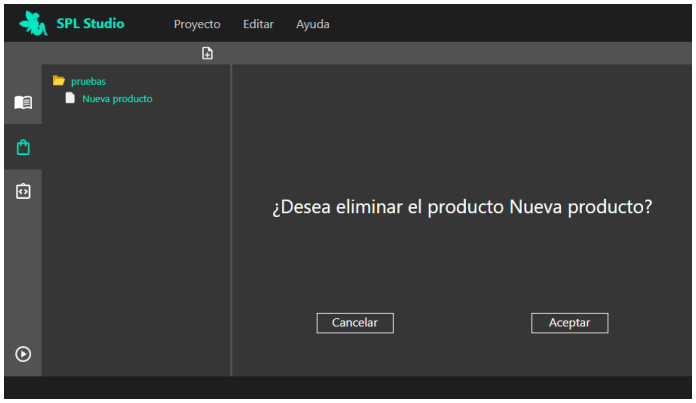


Fig. 44 Ventana de confirmación, para eliminar un producto.

Icono guardar habilitado: Al pulsar sobre él, guarda los cambios en la nube.

Icono guardar deshabilitado: Indica que todos los cambios se han guardado.

Icono guardar habilitado en rojo: Indica que se ha producido un error durante el guardado.

3.3.2.4. Editor de plantillas

El editor de plantillas, permite la creación y edición de las distintas plantillas que componen las familias de la línea de productos de software.

En la imagen **Fig. 45**, aparecen numerados (en rojo), los diferentes apartados que conforman la vista. Estos corresponden con los siguientes componentes:

1. Barra de herramientas para el panel lateral.
2. Barra de herramientas para el panel principal.
3. Panel lateral.
4. Panel principal.

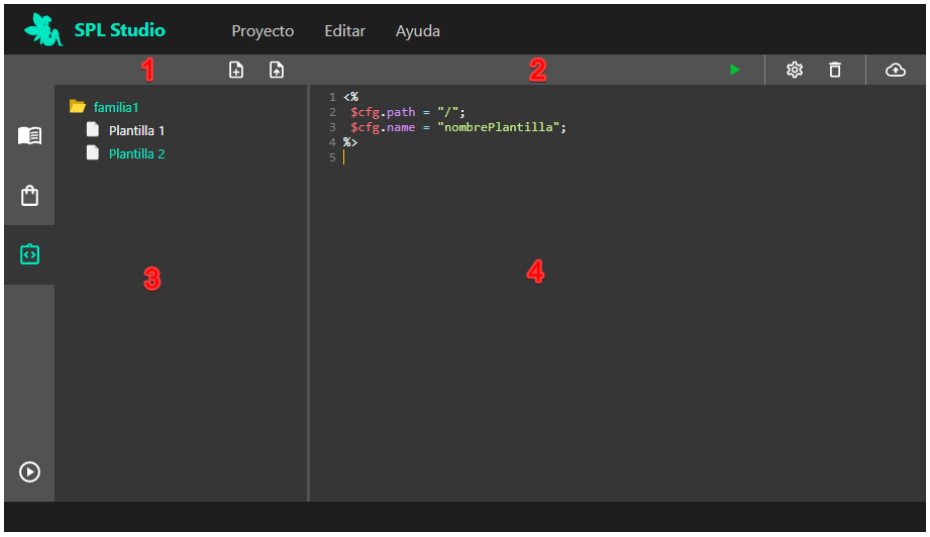


Fig. 45 Interfaz del editor de plantillas.

3.3.2.4.1. Panel lateral

El panel lateral permite la gestión de las diferentes familias. Cada familia se representa mediante una carpeta que agrupa el conjunto de plantillas que la componen. En la figura

Fig. 46, se muestra el panel lateral con dos familias, de las cuales hay una seleccionada que muestra las plantillas que contiene.

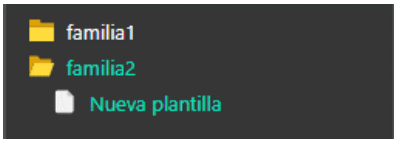


Fig. 46 Detalle del menú lateral del editor de plantillas.

Al pulsar sobre una familia, esta se expande, mostrando las plantillas que contiene, y a su vez, el resto de familias se contraen. Al pulsar sobre una plantilla, esta se muestra en el panel principal para ser editada.

3.3.2.4.2. Barra de herramientas del panel lateral

En la barra de tareas del panel lateral se pueden encontrar los siguientes elementos:

Icono nueva plantilla: Permite la creación de una nueva plantilla. Al pulsar sobre esta opción, en el panel principal, se muestra una ventana que permite seleccionar el nombre, el ancla y la familia para la nueva plantilla. En la imagen **Fig. 47**, se puede observar la ventana correspondiente a la ceración de una nueva plantilla. En ella aparecen dos botones. Si se pulsa sobre el botón aceptar, la nueva plantilla será creada y automáticamente aparecerá en el panel lateral, sin embargo, si se pulsa sobre el botón cancelar, no se realizará ninguna acción y la ventana desaparecerá.

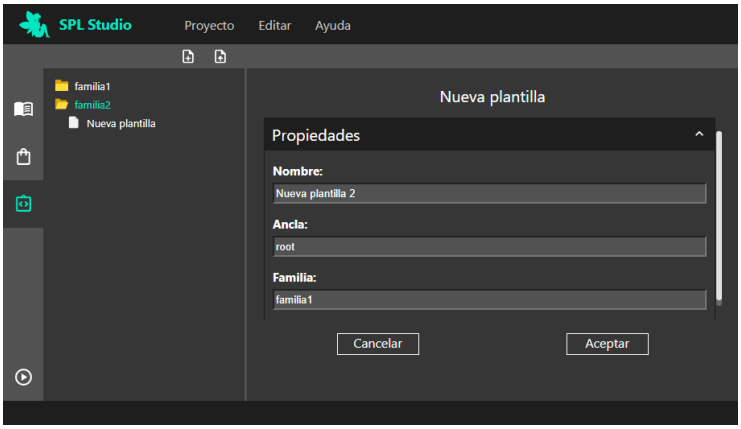


Fig. 47 Ventana de creación de nueva plantilla.

En menor medida que en el caso de los productos, en el momento de crear una nueva plantilla, esta se inicializará con un pequeño código que configura el nombre que tomarán los ficheros que genere, asignándole el mismo nombre que a la plantilla, pues en muchos casos esto será así. En la figura **Fig. 48**, se puede ver un ejemplo de ello.

```
<%
  $cfg.path = "/";
  $cfg.name = "nombrePlantilla";
%>
New Template

```

Fig. 48 Código de inicialización para nuevas plantillas.

Icono importar plantillas: Esta opción permite la creación de las plantillas en base a un prototipo, transformando todos sus ficheros en plantillas y evitando así la creación de estas una a una.

En la imagen **Fig. 49**, se muestra el dialogo de importación, que permite introducir el nombre de la familia y subir un fichero zip con el prototipo para que sea procesado y transformado en una familia.

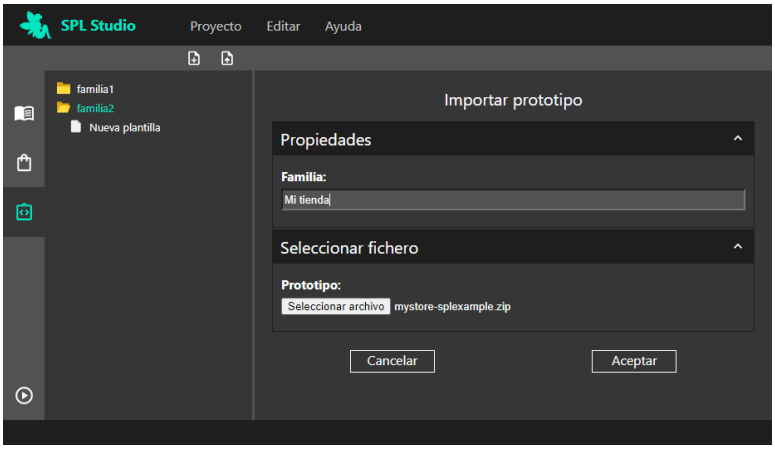


Fig. 49 Ventana de importación de plantillas.

Al pulsar aceptar en la ventana de importación, se crea una nueva familia con todas las plantillas extraídas del prototipo. En figura **Fig. 50**, se puede observar la familia "Mi tienda" que ha sido generada a partir de un prototipo. En el panel lateral, se pueden ver todos los ficheros de la familia cargados, mientras que en el panel principal se observa uno de estos ficheros, donde en la parte superior se ha configurado la ruta y el nombre de los ficheros que generará (líneas de 1 a 4) y a continuación (línea 6 en adelante) el texto del fichero del prototipo.

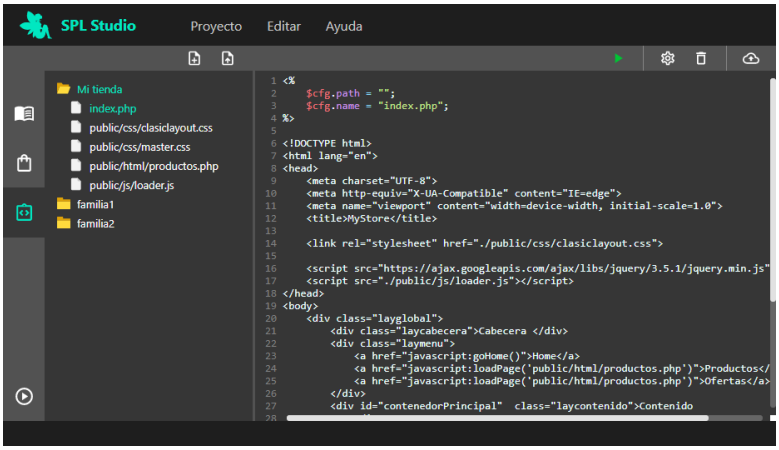


Fig. 50 Ejemplo de carga de prototipo como familia.

Puesto que en este punto todavía no se ha añadido variabilidad, la familia, para cualquier modelo, generaría un producto similar al prototipo. Por lo

tanto, el usuario únicamente necesita añadir la variabilidad de la LPS sobre las plantillas generadas.

3.3.2.4.3. Panel principal

En el panel principal del editor de plantillas, de nuevo encontramos un editor de texto que permite la edición de las plantillas. En la imagen **Fig. 51**, se muestra un ejemplo del funcionamiento del analizador sintáctico acoplado al editor de plantillas, el cual, resalta la sintaxis del código JavaScript contenido en los scriptlets.

```
<%
 $cfg.path = "/";
 $cfg.name = "nombrePlantilla";
%>
Ejemplo de plantilla
<%
 function capitalizar(texto){
    return texto.charAt(0).toUpperCase() + texto.slice(1)
 }
%>
<h1><%=capitalizar("hola mundo")%><h1>
```

Fig. 51 Ejemplo de analizador sintáctico para el editor de plantillas.

3.3.2.4.4. Barra de herramientas del panel principal

En la barra de herramientas del editor de lenguaje se pueden encontrar los siguientes iconos:

Icono de compilación: Realiza la compilación del proyecto incluso si no se han realizado cambios desde la última compilación.

Icono de configuración: Permite editar la configuración de un producto. Al pulsar sobre esta opción, aparece una ventana sobre el panel principal que permite modificar el nombre, el ancla y la familia de la plantilla seleccionada, como se puede observar en la figura **Fig. 52**.

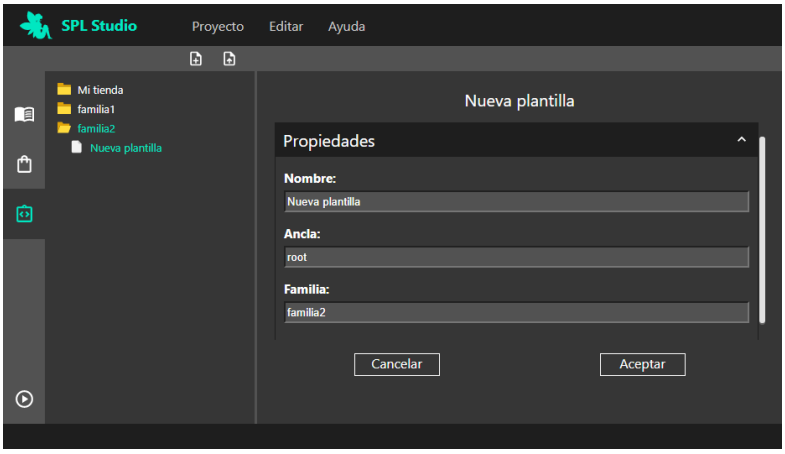


Fig. 52 Ventana de configuración de una familia.

Icono de eliminar: Permite eliminar la plantilla seleccionada. Siempre que se pulse sobre el mismo, aparecerá una ventana de confirmación, como se puede observar en la imagen **Fig. 53,** para evitar la eliminación de plantillas por error.

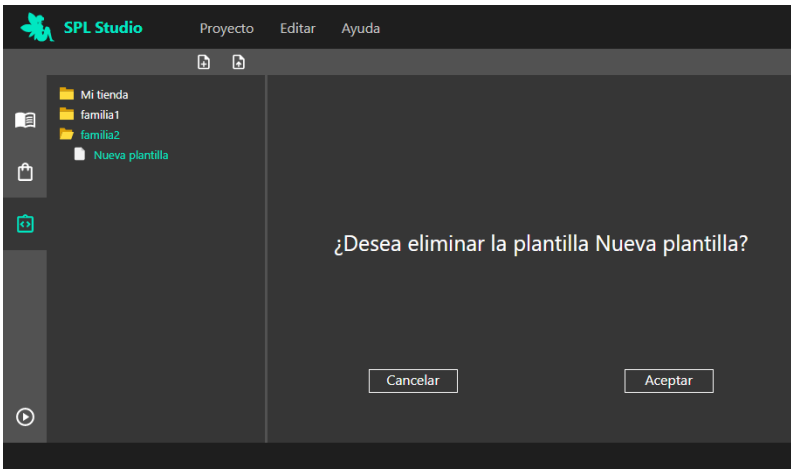


Fig. 53 Ventana de confirmación, para eliminar una plantilla.

Icono guardar habilitado: Al pulsar sobre él, guarda los cambios en la nube.

Icono guardar deshabilitado: Indica que todos los cambios se han guardado.

Icono guardar habilitado en rojo: Indica que se ha producido un error durante el guardado.

3.3.2.5. Pantalla de compilación

Se trata de la pantalla resultado de la compilación de un producto de la línea de productos de software.

En la imagen **Fig. 54**, aparecen numerados (en rojo), los diferentes apartados que conforman la vista. Estos corresponden con los siguientes componentes:

1. Barra de herramientas para el panel lateral.
2. Barra de herramientas para el panel principal.
3. Panel lateral.
4. Panel principal.

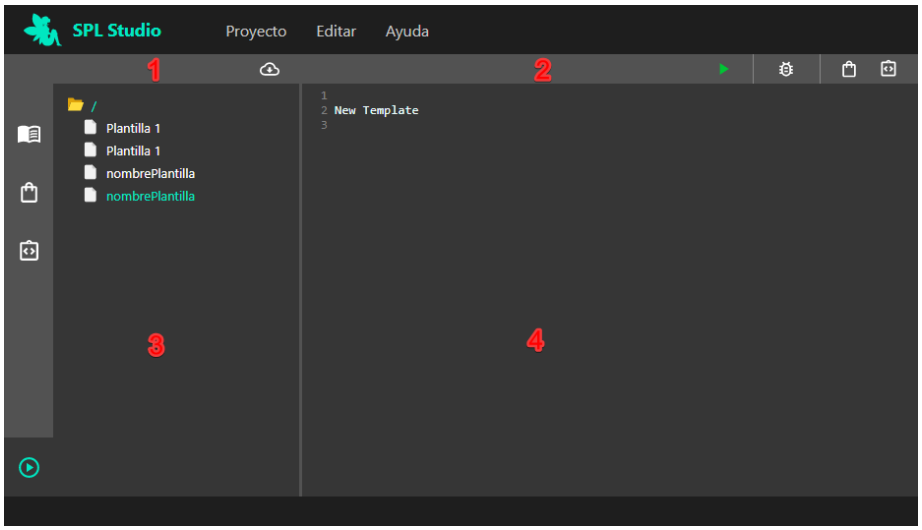


Fig. 54 Interfaz de la pantalla de compilación.

3.3.2.5.1. Panel lateral

En el panel lateral aparecen todos los ficheros generados tras la compilación del producto, es decir, todos los ficheros que componen el producto final. Estos ficheros aparecen agrupados en directorios. En la figura **Fig. 55**, se puede observar como el producto final cuenta con cuatro directorios, uno de ellos se encuentra seleccionado, por lo que muestra su contenido. Al pinchar en el fichero seleccionado, aparece el contenido del mismo en el panel principal.

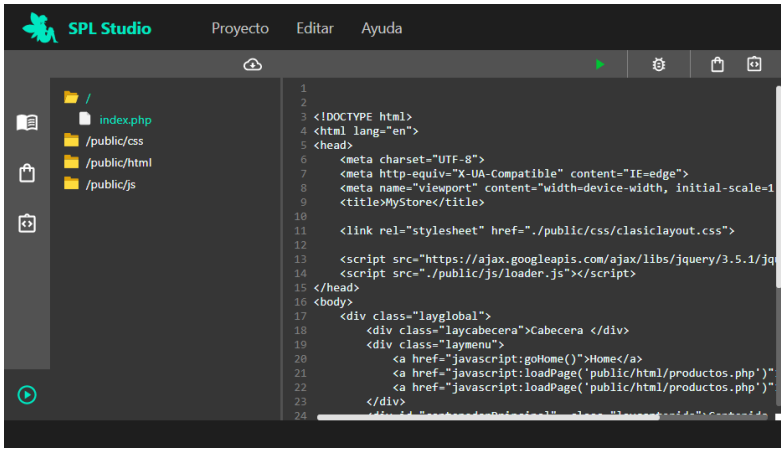


Fig. 55 Detalle del panel lateral una vez realizada la compilación.

Cuando se producen errores de compilación, estos aparecerán agrupados en la carpeta error. En la figura **Fig. 56**, se puede ver el directorio error, que muestra un error obtenido de la compilación, donde se puede ver el nombre del producto y la descripción del error que se ha producido.

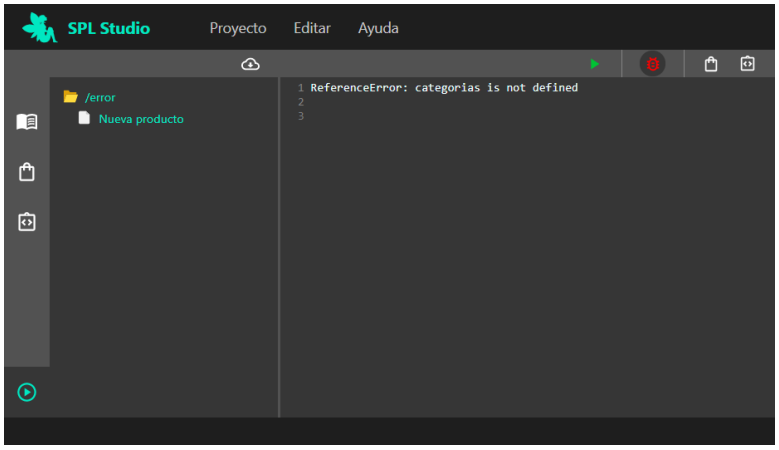


Fig. 56 Pantalla de compilación con errores.

3.3.2.5.2. Barra de herramientas

En la barra de herramientas del editor de lenguaje se pueden encontrar los siguientes iconos:

Icono de descarga: Al pulsar sobre él, permite la descarga del producto final en forma de fichero zip.

3.3.2.5.3. Panel principal

El panel principal contiene un editor de texto con dos funcionalidades, el modo texto que visualizar los ficheros, y el modo debug, donde se pueden visualizar los logs y errores derivados de la compilación.

En la figura **Fig. 57**, se puede observar el panel principal en modo texto, mientras que en la figura **Fig. 58**, se presenta este panel en modo debug.

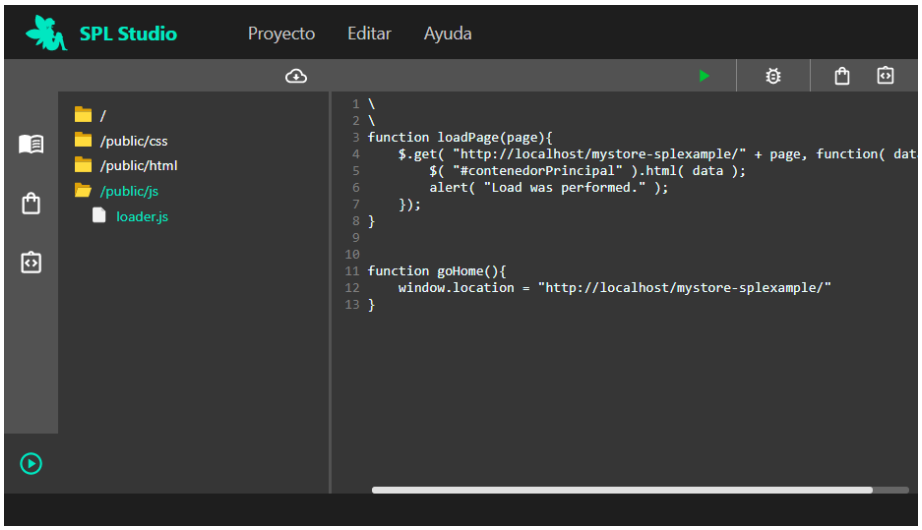


Fig. 57 Panel principal de la pantalla de compilación en modo texto.

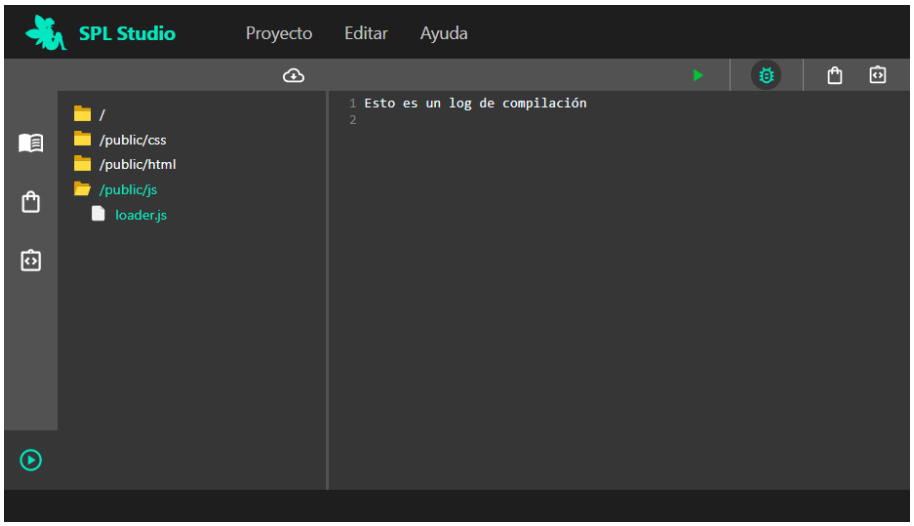


Fig. 58 Panel principal de la pantalla de compilación en modo debug.

3.3.2.5.4. Barra de herramientas

En la barra de herramientas del editor de lenguaje se pueden encontrar los siguientes iconos:

Icono de compilación: Realiza la compilación del proyecto incluso si no se han realizado cambios desde la última compilación.

Icono debug: Indica que el panel principal se encuentra en modo fichero. Al hacer click se pasa a modo debug.

Icono debug seleccionado: Indica que el panel principal se encuentra en modo debug. Al hacer clic se pasa a modo fichero.

Icono debug con errores: Indica que se han producido errores durante la compilación. El panel principal se encuentra en modo debug y no es posible seleccionar el modo fichero ya que no ha compilado.

Icono de producto: Abre el producto de la compilación en el editor de productos.

Icono de plantilla: Abre la plantilla de la compilación en el editor de plantillas.

Capítulo 4: Validación experimental

4.1. Caso de estudio. Aplicación de seguridad.

A continuación, se va a crear un pequeño ejemplo de desarrollo de un producto utilizando SPL.

Supongamos que se quiere implementar un método de validación para dispositivos a través de un programa que genere y/o valide un código de 6 dígitos.

Para ello, se ha construido una primera aplicación, en otras palabras, se ha construido el primer producto de una familia de productos. La aplicación se ha construido en lenguaje Java y el coste de la creación de este componente base fue de aproximadamente tres horas, ya que se trata de un componente sencillo. En la figura **Fig. 59**, se muestra el diagrama de clases.

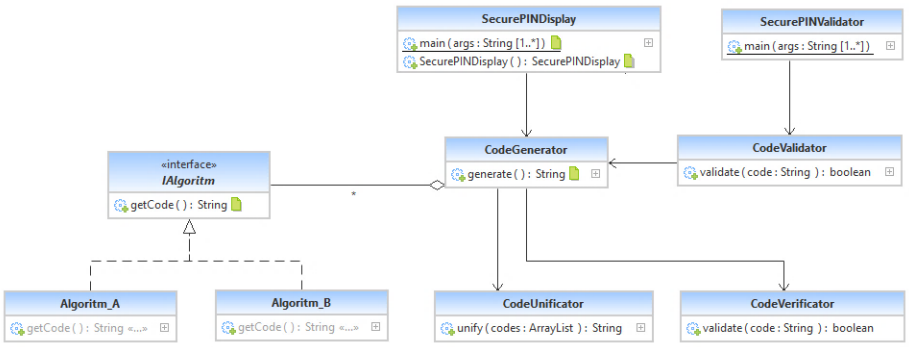


Fig. 59 Diagrama de clases de la aplicación construida para la generación y validación de códigos.

El programa es un dos en uno, puede ejecutarse a partir de la clase "SecurePINDisplay" para generar un código y mostrarlo en pantalla, o a través de la clase "SecurePINValidator" para comprobar la validez de un código pasado como argumento.

Para la generación de los códigos se utilizan algoritmos que implementan la interfaz "IAlgoritm", por el momento, se han construido únicamente dos, el algoritmo "Algoritm_A" que genera un código diferente cada hora y el algoritmo "Algoritm_B" que genera un código diferente cada día. Ambos se basan en una semilla que varía para cada compañía, de forma que el código generado por el algoritmo dependerá del valor de esta semilla.

Estos algoritmos se pueden combinar de forma diferente para cada empresa y el código generado por cada uno de los algoritmos, se combinará con código generado por el

siguiente, por ejemplo, una empresa puede ejecutar los algoritmos [A-A-B-A], y otra únicamente el algoritmo [B]. De esta secuencia de algoritmos se encarga la clase "CodeGenerator", que contiene un array con el orden de ejecución.

Las compañías que van a utilizar el programa tendrán dos versiones, una que genera el código, para integrar en sus dispositivos (a través de la clase "SecurePINDisplay") y otra que puede validar los códigos generados (a través de la clase "SecurePINValidator"). Ambas versiones del programa deberán estar alineadas, esto significa que tienen que contar con la misma secuencia de algoritmos y las mismas semillas de generación en estos algoritmos. En la figura **Fig. 60**, se muestran ambas versiones.

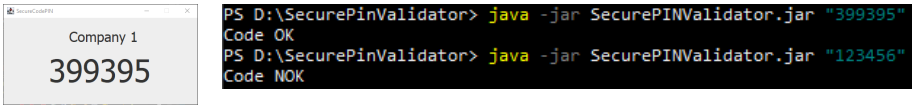


Fig. 60 A la izquierda la ejecución de SecurePINDisplay, a la derecha la ejecución de SecurePINValidator.

4.1.1. Ingeniería de dominio / Espacio del problema (EDP)

Al ser un proyecto pequeño, el estudio de variabilidad no es complicado. En primer lugar, tenemos los datos relacionados con la compañía, en este caso, únicamente se ha necesitado el nombre. También tenemos el tipo de producto se va a generar, un display para generar y mostrar códigos o un validador. Por último, tenemos los algoritmos disponibles que pueden ser utilizados, que contienen una semilla y un nombre, con ellos se pueden crear diferentes versiones del mismo algoritmo. A partir de estos datos se puede construir el dominio como se muestra en la figura **Fig. 61**.

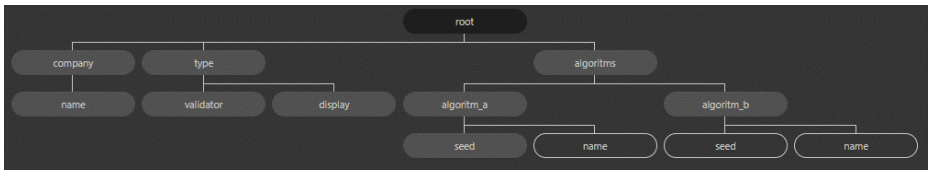


Fig. 61 Diseño del dominio utilizando el editor visual de SPL Studio.

4.1.2. Ingeniería de dominio / Espacio de la solución (EDS)

Como ya se comentó en el apartado **3.3.2.4.2,** con SPL Studio es posible cargar este prototipo en forma de plantillas. En la figura **Fig. 62**, se muestra el resultado de la carga del prototipo utilizando el asistente de SPL Studio. En el menú lateral situado a la izquierda se pueden ver todos los ficheros cargados, mientras que en el editor de código de la derecha, se puede ver el contenido del fichero seleccionado.

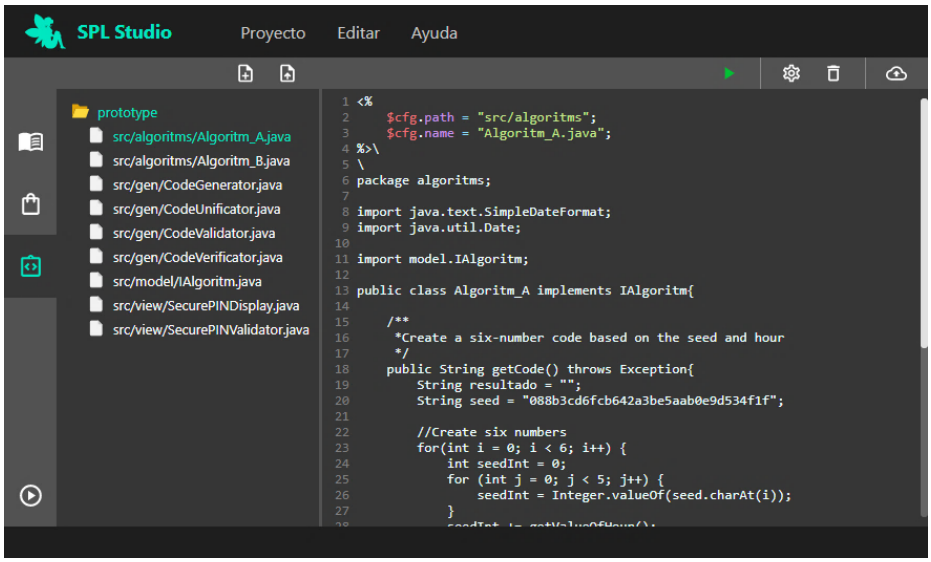


Fig. 62 Captura de carga del prototipo para el caso de estudio 1.

A continuación, se incluyen los puntos de variabilidad sobre el prototipo cargado. Comenzando por los Algoritmos, la plantilla para el algoritmo A se ancla al elemento “algoritm_a” del dominio, mientras que la plantilla para el algoritmo B se ancla al elemento “algoritm_b”. Con esto se consigue que se cree una clase por cada algoritmo que contenga el modelo. En las figuras **Fig. 63** y **Fig. 64**, se muestran sus plantillas. En la línea 3 de ambas, podemos ver como el nombre del fichero se genera en función del elemento “name” del modelo y su posición respecto al padre. En la línea 13 también se aplica este nombre para declarar la clase. Por último, en la línea 20 se inserta la semilla correspondiente al elemento “seed” del modelo.

```
<%
    $cfg.path = "src/algoritms";
    $cfg.name = $self.name[0].value + "_" + $self.index + ".java";
%>

package algoritms;

import java.text.SimpleDateFormat;
import java.util.Date;

import model.IAlgoritm;

public class <%=$self.name[0].value%>_<%=$self.index%> implements IAlgoritm{

    /**
     *Create a six-number code based on the seed and hour
     */
    public String getCode() throws Exception{
        String resultado = "";
        String seed = "<%=$self.seed[0].value%>";

        //Create six numbers
        for(int i = 0; i < 6; i++) {
            int seedInt = 0;
            for (int j = 0; j < 5; j++) {
                seedInt = Integer.valueOf(seed.charAt(i));
            }
            seedInt += getValueOfHour();
            resultado = resultado + (seedInt % 10) + "";
        }

        return resultado;
    }

    /**
     * Get the hour as a number
     */
    private int getValueOfHour() throws Exception{
        String pattern = "HH";
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);
        String date = simpleDateFormat.format(new Date());
        return Integer.parseInt(date);
    }

}
```

Fig. 63 Plantilla para el algoritmo A.

```
<%
    $cfg.path = "src/algoritms";
    $cfg.name = $self.name[0].value + "_" + $self.index + ".java";
%>

package algoritms;

import java.text.SimpleDateFormat;
import java.util.Date;

import model.IAlgoritm;

public class <%=$self.name[0].value%>_<%=$self.index%> implements IAlgoritm{

    /**
     * Create a six-number code based on the seed and day
     */
    public String getCode() throws Exception{
        String resultado = "";
        String seed = "<%=$self.seed[0].value%>";

        //Create six numbers
        for(int i = 0; i < 6; i++) {
            int seedInt = 0;
            for (int j = 0; j < 5; j++) {
                seedInt = Integer.valueOf(seed.charAt(i));
            }
            seedInt += getValueOfHour();
            resultado = resultado + (seedInt % 10) + "";
        }

        return resultado;
    }

    /**
     * Get the day as a number
     */
    private int getValueOfHour() throws Exception{
        String pattern = "DD";
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);
        String date = simpleDateFormat.format(new Date());
        return Integer.parseInt(date);
    }

}
```

Fig. 64 Plantilla para el algoritmo B

Para insertar la variabilidad de la secuencia de algoritmos, se edita la plantilla de la clase “CodeGenerator”. En la figura **Fig. 65**, se puede observar cómo se iteran todos los algoritmos comprendidos en el elemento “algoritms” utilizando el atributo “all”, tanto en la línea 15 para crear las importaciones de las clases de los algoritmos, como en la línea 49, donde se instancian los objetos de estas clases y se incluyen en el array que marca la secuencia de ejecución de los algoritmos.

```
1  <%
2      $cfg.path = "src/gen";
3      $cfg.name = "CodeGenerator.java";
4
5
6
7  %>
8
9  package gen;
10
11 import java.util.ArrayList;
12
13 import model.IAlgoritm;
14
15 <% for (let i = 0; i < $self.algoritms[0].all.length; i++){ %>
16 import algoritms.<%=$self.algoritms[0].all[i].name[0].value%>_<%=$self.algoritms[0].all[i].index%>;
17 <% } %>
18
19
20 public class CodeGenerator {
21
22     /*
23      * Create a six-number code based on algorithms
24      */
25     public String generate() throws Exception {
26         //Obtaining algorithms
27         ArrayList<IAlgoritm> algoritms = getAlgoritms();
28         //Obtaining codes
29         ArrayList<String> codes = getCodes(algoritms);
30
31         //Unify codes to obtain the final code
32         CodeUnificator codeUnificator = new CodeUnificator();
33         String code = codeUnificator.unify(codes);
34
35         //Verificate the final code
36         CodeVerificator codeVerificator = new CodeVerificator();
37         if(!codeVerificator.validate(code)) {
38             throw new Exception("The code " + code + " is not valid");
39         }
40
41         return code;
42     }
43
44     /*
45      * Get the algorithms to use
46      */
47     private ArrayList<IAlgoritm> getAlgoritms(){
48         ArrayList<IAlgoritm> algoritms = new ArrayList<IAlgoritm>();
49      <% for (let i = 0; i < $self.algoritms[0].all.length; i++){ %>
50         algoritms.add(new <%=$self.algoritms[0].all[i].name[0].value%>_<%=$self.algoritms[0].all[i].index%>());
51      <% } %>
52         return algoritms;
53     }
```

Fig. 65 Plantilla para la clase “SecurePINDisplay”.

En la **Fig. 66**, aparece la plantilla para la clase “SecurePINDisplay”, encargada de generar la interface de usuario que muestra el código. En este caso se inserta el elemento “name” anidado en el elemento “company” de cada modelo en el lugar reservado para la etiqueta del nombre de la compañía (línea 56).

```
1  <%
2      $cfg.path = "src/view";
3      $cfg.name = "SecurePINDisplay.java";
4  %>
5
6  package view;
7
8  import java.awt.EventQueue;
9
10 import javax.swing.JFrame;
11 import javax.swing.JLabel;
12 import java.awt.Font;
13 import javax.swing.SwingConstants;
14
15 import gen.CodeGenerator;
16
17 public class SecurePINDisplay {
18
19     private JFrame frmSecurepin;
20
21     /**
22      * Launch the application.
23      */
24     public static void main(String[] args) {
25         EventQueue.invokeLater(new Runnable() {
26             public void run() {
27                 try {
28                     SecurePINDisplay window = new SecurePINDisplay();
29                     window.frmSecurepin.setVisible(true);
30                 } catch (Exception e) {
31                     e.printStackTrace();
32                 }
33             }
34         });
35     }
36
37     /**
38      * Create the application.
39      */
40     public SecurePINDisplay() {
41         initialize();
42     }
43
44     /**
45      * Initialize the contents of the frame.
46      */
47     private void initialize() {
48         frmSecurepin = new JFrame();
49         frmSecurepin.setAlwaysOnTop(true);
50         frmSecurepin.setTitle("SecureCodePIN");
51         frmSecurepin.setResizable(false);
52         frmSecurepin.setBounds(100, 100, 451, 233);
53         frmSecurepin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
54         frmSecurepin.getContentPane().setLayout(null);
55
56         JLabel lblNewLabel = new JLabel("<%=$root.company[0].name[0].value%>");
57         lblNewLabel.setHorizontalAlignment(SwingConstants.CENTER);
58         lblNewLabel.setFont(new Font("Tahoma", Font.PLAIN, 31));
59         lblNewLabel.setBounds(29, 24, 390, 37);
60         frmSecurepin.getContentPane().add(lblNewLabel);
```

Fig. 66 Plantilla para la clase “SecurePINDisplay”.

Además, la plantilla “SecurePINDisplay” estará anclada al elemento “type.display“, de forma que, solo se compilará cuando en el modelo se especifique que se trata de una aplicación display (para generar códigos). Del mismo modo, la plantilla “SecurePINValidator” se encuentra anclada al elemento “type.validator” y solo se genera cuando se especifica en el modelo. De esta forma, se pueden crear las dos versiones de la aplicación.

4.1.3. Ingeniería de aplicación / Espacio del problema (EAP)

SPL Studio genera modelos en base al lenguaje de dominio para evitar que el usuario tenga que realizarlos desde le cero. En la parte izquierda de la figura **Fig. 67**, se muestra el modelo generado automáticamente, en la parte central se muestra un modelo para generar una aplicación “SecurePINDisplay” que genera los códigos, y en la parte derecha un modelo para generar una aplicación “SecurePINValidator” que valida los códigos.

Fig. 67 En la parte izquierda un modelo autogenerado, en la parte central y en la parte derecha dos ejemplos de modelos para la misma familia.

4.1.4. Ingeniería de aplicación / Espacio de la solución (EAS)

Para obtener el producto final, se compilan los modelos junto con la familia de productos creada. En caso de error en la especificación del modelo o compilación de la plantilla, SPL Studio lo reporta de forma que en este punto puede detectar cualquier error que se haya cometido. En la figura **Fig. 68**, se muestra la diferencia entre un producto bien compilado y un producto con errores de compilación.

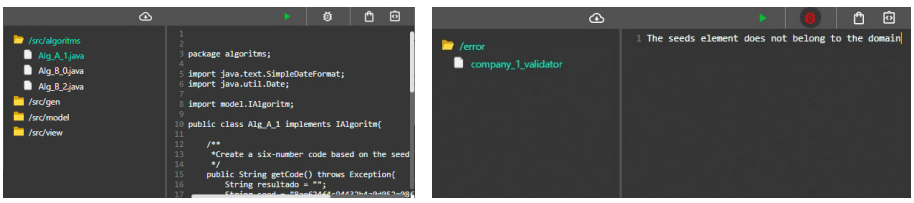


Fig. 68 Ejemplos de compilación de un producto.

En este punto, se pueden generar tantos productos como sea necesario, la LPS se puede considerar terminada. El tiempo aproximado de construcción de la misma ha sido de media hora, un tiempo muy inferior al coste de la creación de un solo producto.

En cuestión de minutos, hemos construido la LPS. Además, instantáneamente cualquier persona puede acceder a SPL Studio desde otros dispositivos y comenzar a crear sus propios productos, no es necesario distribuirla y configurarla en los equipos de los ingenieros de aplicación.

Con la aparición de algoritmos más robustos, la actualización de todos los productos disponibles puede realizarse de forma inmediata.

4.2. Caso de estudio 2. Documentación online.

En este caso, se pretende crear una LPS que de soporte a diferentes equipos de desarrollo de software y les permita generar documentación en formato web de una forma sencilla. Se pretende que los equipos tengan libertad para modelar su espacio, pero siempre respetando un estilo común.

4.2.1. Secciones

Cada web de documentación, puede dividirse en distintas secciones, que pueden ser opcionales:

- **Header**: Cabecera de la página (Se muestra en rojo en la figura **Fig. 69**).
- **Nav**: Barra de navegación superior (Se muestra en verde en la figura **Fig. 69**).
- **Aside**: Barra de navegación lateral (Se muestra en azul en la figura **Fig. 69**).
- **Footer**: Espacio al pie de la página (Se muestra en amarillo en la figura **Fig. 69**).

Y una sección central no opcional:

- **Article**: Espacio donde se muestra el contenido (Se muestra en marrón en la figura **Fig. 69**).

Los equipos sol libres para seleccionar cuales, de las secciones opcionales, aparecen en su web. Además, podrán personalizar algunos aspectos de las secciones como el tamaño, el color y el contenido inicial que aparecerá en cada sección.

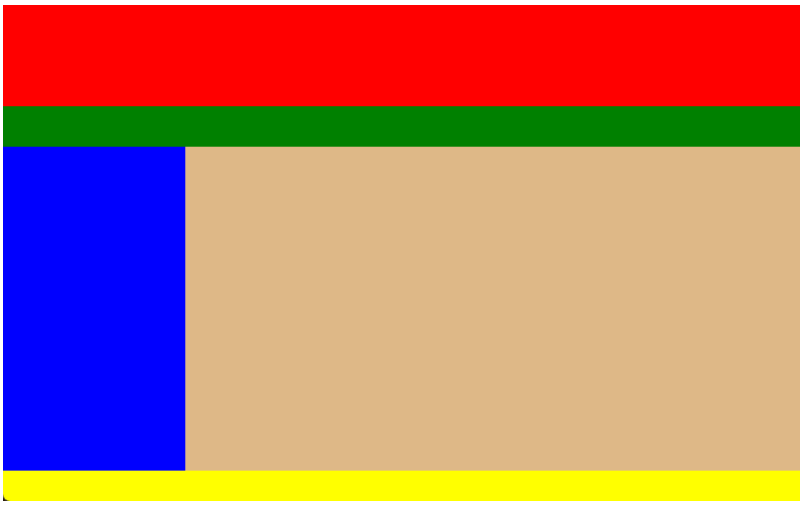


Fig. 69 Esquema de repartición de secciones en la página

4.2.2. Contenidos

Los contenidos son modelos que se pueden utilizar para ser cargados dentro de las diferentes secciones, a continuación, se muestran los contenidos que han sido predefinidos, sin embargo, podrían incorporarse nuevos contenidos a la SPL si fuese necesario.

4.2.2.1. Texto simple.

Se trata de un contenido donde se mostrará un texto. Tanto el texto como el color del mismo es configurable por el equipo. En la figura **Fig. 70**, se presenta un contenido de tipo "texto simple".

Fig. 70 Prototipo de contenido "texto simple"

4.2.2.2. Documento

El documento, permite a los equipos crear páginas con los siguientes elementos configurables:

- **Título**: Texto de tamaño grande y centrado.
- **Subtitulo**: Texto de tamaño grande, un poco menor que el del título, alineado a la izquierda.
- **Imagen**: Imagen que se mostrará ajustándose al ancho de la página.
- **Texto**: Texto de tamaño normal que aparecerá con una alineación justificada.
- **Bloque de código**: Texto que aparece enmarcado en un cuadro ocupando el ancho de la página. Bajo el código y alineado a la derecha, es posible configurar una descripción del código que se muestra.
- **Link**: Texto clicable que permite la carga de otros contenidos dentro de la sección "article".
- **Espacio**: Permite insertar separadores verticales entre los elementos. Es posible configurar el tamaño del espacio.

Con los elementos anteriores, cada equipo puede crear sus documentos, estructurándolos en el orden que requiera y tantas veces como necesite.

En el ejemplo de la figura **Fig. 71**, se muestra un documento que contiene: Un título, un subtítulo, un texto, una imagen, un link, un espacio y un bloque de código, en el orden indicado.

Además, es posible configurar el color en el que aparecen tanto los textos del documento como los links.

Fig. 71 Prototipo de contenido "documento"

4.2.2.3. Datos

El contenido de datos permite mostrar tablas de datos. Es posible configurar un título para la página y cada una de las líneas de datos que se van a mostrar en forma de tabla. En la figura **Fig. 72**, se puede observar un ejemplo de contenido de datos.

Fig. 72 Prototipo de contenido "datos"

El equipo podrá configurar el color en el que aparece tanto el texto, como los datos de la tabla.

4.2.2.4. Banner

El banner permite configurar una imagen que se ajusta al ancho de la página. El sobrante vertical de la imagen se recorta, consiguiendo el efecto que se aprecia en la figura **Fig. 73**.

Fig. 73 Prototipo de contenido "banner"

4.2.2.5. Logo con texto

El logo con texto, como su nombre indica, permite que el equipo configure una imagen junto con un texto que aparecerá alineado a la izquierda, como se muestra en la imagen **Fig. 74**.

Fig. 74 Prototipo de contenido "logo con texto"

También es posible configurar el color con el que aparecerá el texto.

4.2.2.6. Menú horizontal

El menú horizontal permite configurar diferentes links alineados de forma horizontal, mediante los que es posible realizar la carga de diferentes contenidos en la sección "article". En la figura **Fig. 75**. se muestra el resultado de configurar este menú con tres elementos.

Fig. 75 Prototipo de contenido "menú horizontal"

Es posible configurar el texto de los links, el contenido que cargarán, y el color de estos cuando reciben el foco del ratón y cuando no lo reciben.

4.2.2.7. Menú vertical

De la misma forma que el menú horizontal, el menú vertical, permite la configuración de estos elementos en una única columna. El menú vertical también permite la carga de contenidos dentro de la sección "article". En la figura **Fig. 76**, se muestra un menú vertical con tres elementos.

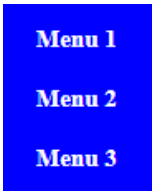


Fig. 76 Prototipo de contenido "menú vertical"

En este caso también se puede configurar el texto de los links, el contenido que cargarán, y el color de estos cuando reciben el foco del ratón y cuando no lo reciben.

4.2.3. Prototipo

En este caso, se propone un enfoque donde el prototipo se implementa de forma parcial, es decir, se construirá una demo que será suficiente para crear la LPS, posteriormente se podrá completar el primer producto utilizando la LPS. De esta forma el tiempo de implementación del primer producto se ve reducido en relación a un desarrollo tradicional.

Para la construcción del prototipo se van a crear los siguientes elementos:

- **/index.html**: Fichero principal, en él se definen la estructura html principal, incluyendo las secciones que aparecerán.

- **/public/css/layout.css**: Css de configuración del layout, en él se definen los tamaños y los colores de las secciones.
- **/public/js/navigation.js**: Este fichero JavaScript gestiona la navegación entre los contenidos. Se importa en el fichero index.html y proporciona una función global para navegar.
- **/pages/data.html**: Contenido de datos.
- **/pages/document.html**: Contenido documento.
- **/pages/fullbaner.html**: Contenido banner.
- **/pages/hbasicmenu.html**: Contenido menú horizontal.
- **/pages/logotexto.html**: Contenido logo y texto.
- **/pages/simpletext.html**: Contenido texto simple.
- **/pages/vbasicmenu.html**: Contenido menú vertical.

Tras la implementación de todos los elementos mencionados anteriormente, el prototipo se presenta como una demo funcional, tal y como se muestra en **Fig. 77**, pues no es necesario añadir el contenido real de cara a la construcción de la LPS.

Fig. 77 Vista del prototipo demo

4.2.4. Ingeniería de dominio / Espacio del problema (EDP)

Teniendo en cuenta los puntos anteriores, se modela el dominio para abarcar la variabilidad. La imagen **Fig. 78**, muestra el dominio del problema configurado con la herramienta SPL Studio. En primer lugar, nos encontramos con las diferentes secciones posibles, bajo el elemento "layouts", se pueden encontrar los elementos "header", "nav", "aside", "footer" y "article", con sus correspondientes configuraciones de color, tamaño y contenido que cargan por defecto, como se muestra en la imagen superior de la figura **Fig. 78**. En esta imagen, en la parte derecha, encontramos también los elementos "name" y "location" que hacen referencia al nombre de la aplicación y al directorio bajo el que se va a encontrar ésta, una vez puesta en producción. En la imagen central y la imagen inferior, se observa bajo el elemento "content", se encuentran todas las posibilidades de contenidos que permite la LPS. Cada una de ellas cuenta con su configuración correspondiente. Cabe destacar como en el elemento "c_document", también encontramos cada uno de los posibles bloques que se pueden agregar al elemento y sus atributos correspondientes cuando son requeridos. Los elementos "c_hmenu" y "c_vmenu" toman una estructura similar, en la que se componen de elementos de tipo "menú_item", de forma que será relativamente sencillo transformar un menú horizontal en uno vertical y viceversa. Cada uno de los elementos "menú_item" puede configurarse con un elemento "content_name" que permite referenciar el elemento "contenido" a cargar dentro de la sección "article".

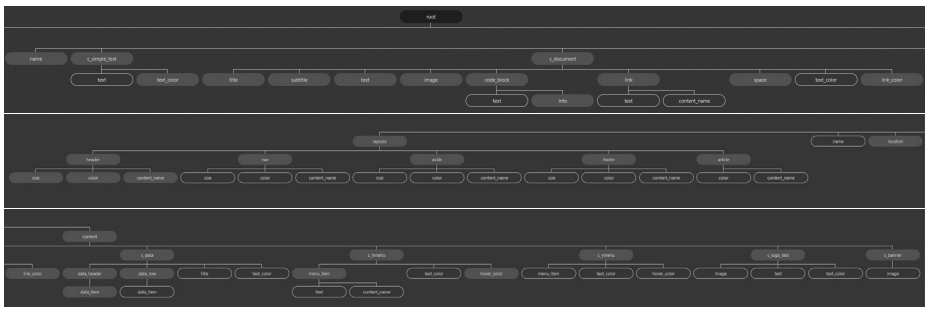


Fig. 78 Dominio del problema para el caso de estudio 2.

4.2.5. Ingeniería de dominio / Espacio de la solución (EDS)

De nuevo, se crean las plantillas correspondientes al prototipo utilizando la funcionalidad automática que ofrece SPL Studio, de esta forma, ya solo es necesario implementar la variabilidad.

En la parte izquierda de la imagen **Fig. 79**, se puede observar como todos los ficheros han sido creados de forma automática.

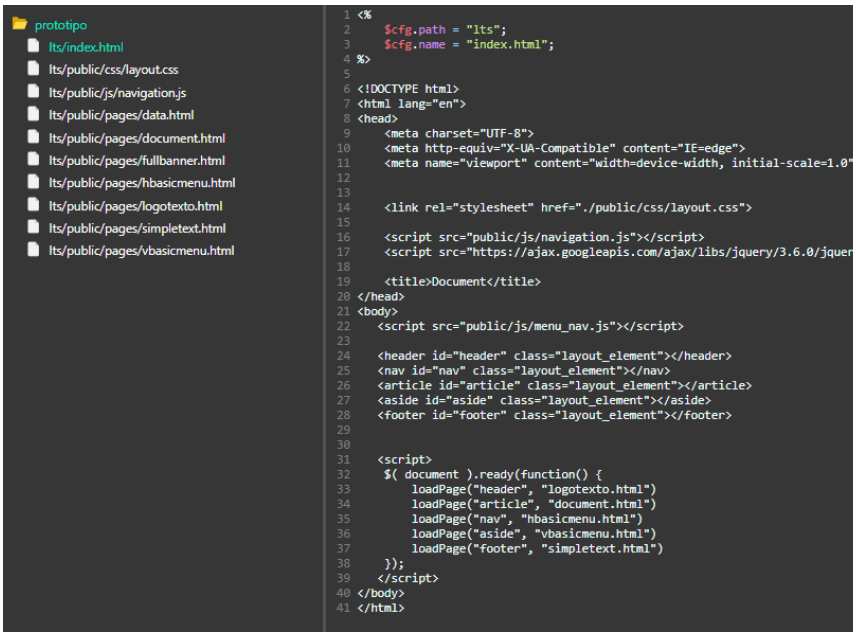
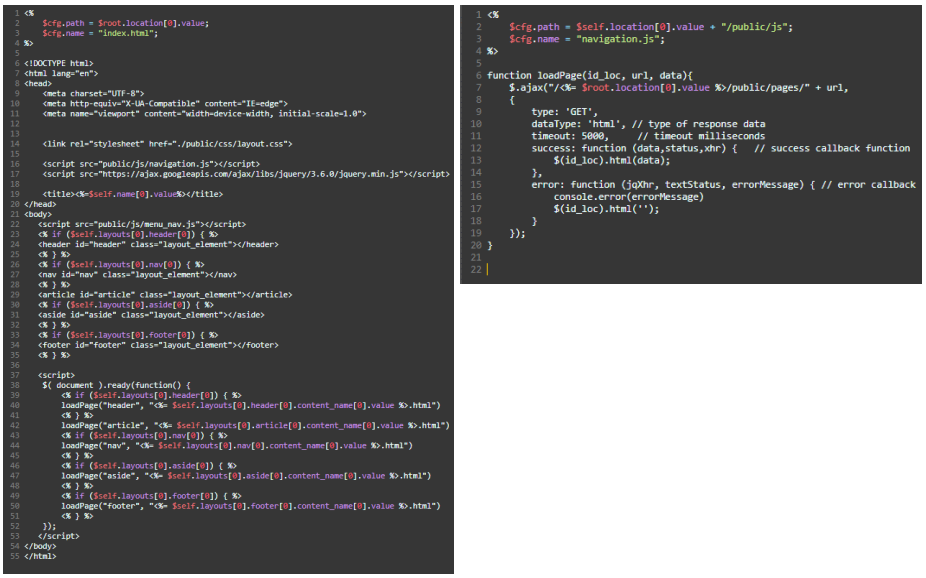


Fig. 79 Carga automática de ficheros en la LPS Documentación online.

En todos los ficheros, se utiliza el elemento “location” del dominio para definir su ruta, como se puede observar en la línea 2 de ambas plantillas de la figura **Fig. 80**, de esta forma, este fichero aparecerá siempre bajo un directorio dinámico que permitirá, que el producto generado pueda convivir con otros productos en un entorno de producción. En la parte izquierda de esta imagen, se muestra la plantilla creada para el fichero index.html. Esta plantilla se encuentra anclada al elemento “root”. En la línea 19, se utiliza el elemento “name” para configurar el título del documento. Seguidamente, en las líneas 22, 26, 30 y 33 se condiciona la creación del html correspondiente a las secciones. De igual forma, en las líneas 39, 43, 46 y 49 se condiciona la llamada al método “loadPage” (que inicializa el contenido de cada sección) para que únicamente aparezcan las secciones presentes en el modelo.

En la parte derecha de la **Fig. 80**, encontramos el código del fichero navigation.js. Este método es el encargado de realizar la carga de contenidos. En la línea 7 se inserta el elemento “location” como parte de la url que define la ubicación de los elementos a cargar. Esta plantilla se encuentra anclada al elemento “root”, por lo que solo generará un fichero por cada modelo.



```
<%
    $cfg.path = $root.location[0].value;
    $cfg.name = "index.html";
%>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">


    <link rel="stylesheet" href="./public/css/layout.css">

    <script src="public/js/navigation.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

    <title><%=$self.name[0].value%></title>
</head>
<body>
    <script src="public/js/menu_nav.js"></script>
    <% if ($self.layouts[0].header[0]) { %>
    <header id="header" class="layout_element"></header>
    <% } %>
    <% if ($self.layouts[0].nav[0]) { %>
    <nav id="nav" class="layout_element"></nav>
    <% } %>
    <article id="article" class="layout_element"></article>
    <% if ($self.layouts[0].aside[0]) { %>
    <aside id="aside" class="layout_element"></aside>
    <% } %>
    <% if ($self.layouts[0].footer[0]) { %>
    <footer id="footer" class="layout_element"></footer>
    <% } %>

    <script>
     $( document ).ready(function() {
        <% if ($self.layouts[0].header[0]) { %>
        loadPage("header", "<%= $self.layouts[0].header[0].content_name[0].value %>.html")
        <% } %>
        loadPage("article", "<%= $self.layouts[0].article[0].content_name[0].value %>.html")
        <% if ($self.layouts[0].nav[0]) { %>
        loadPage("nav", "<%= $self.layouts[0].nav[0].content_name[0].value %>.html")
        <% } %>
        <% if ($self.layouts[0].aside[0]) { %>
        loadPage("aside", "<%= $self.layouts[0].aside[0].content_name[0].value %>.html")
        <% } %>
        <% if ($self.layouts[0].footer[0]) { %>
        loadPage("footer", "<%= $self.layouts[0].footer[0].content_name[0].value %>.html")
        <% } %>
     });
    </script>
</body>
</html>
```

```
<%
    $cfg.path = $self.location[0].value + "/public/js";
    $cfg.name = "navigation.js";
%>

function loadPage(id_loc, url, data){
    $.ajax("/<%= $root.location[0].value %>/public/pages/" + url,
    {
        type: 'GET',
        dataType: 'html', // type of response data
        timeout: 5000,     // timeout milliseconds
        success: function (data,status,xhr) {   // success callback function
            $(id_loc).html(data);
        },
        error: function (jqXhr, textStatus, errorMessage) { // error callback
            console.error(errorMessage)
            $(id_loc).html('');
        }
    });
}
```

Fig. 80 Plantillas de la LPS Documentación online. En la parte izquierda la plantilla para el fichero index.html y en la parte derecha la plantilla para el fichero navigation.js.

Uno de los componentes más importantes de la LPS es el layout.css. Este fichero se encarga de definir el aspecto que van a presentar las diferentes secciones de la web. Como ya se ha comentado, estas secciones, salvo la sección “article” pueden ser opcionales y no estar presentes en los modelos, por ello, la LPS ha de ser capaz de ajustar los tamaños de cada sección para atender las necesidades de cada modelo.

En la figura **Fig. 81**, se muestra el código de la plantilla para el fichero layour.css. Entre las líneas 5 y 24, se puede observar cómo se recupera el tamaño especificado para cada una de las secciones, siendo cero, el tamaño por defecto. Estos tamaños son almacenados en variables que serán utilizadas posteriormente. En las líneas 40, 55, 70 y 86 se condiciona la aparición de los estilos correspondientes a las secciones opcionales, de tal forma que, si la sección no está presente, su estilo css no se va a generar. En las líneas 51, 66, 81, 97 y 110, se aplican los colores establecidos en el modelo a cada una de las secciones. Por último, se establece el tamaño de las secciones de la siguiente forma.

- **Header**: Se ajusta arriba y a los lados de la pantalla, tomando como alto el tamaño especificado en el modelo y almacenado en la variable “header_size” (líneas 41 a 52).

- **Nav**: Se ajusta a los lados de la pantalla, en la parte superior deja un espacio igual al tamaño del header. El alto de este nav será el especificado en el modelo y almacenado en la variable “nav_size” (líneas 56 a 67).

- **Footer**: Se ajusta a los lados de la pantalla. Toma el alto especificado en el modelo y almacenado en la variable “footer_size” (líneas 88 a 98).

- **Aside**: Se ajusta al lado izquierdo de la pantalla. Su ancho se configura con el tamaño especificado en el modelo y almacenado en la variable “aside_size”. El espacio respecto a la parte superior de la pantalla es la suma del tamaño del header y el tamaño del nav. El espacio respecto a la parte inferior de la pantalla es igual al tamaño del footer (líneas 71 a 83).

- **Article**: Se ajusta al lado derecho de la pantalla. El espacio respecto a la parte superior de la pantalla, al igual que el aside, es la suma del tamaño del header y el tamaño del nav. El espacio con la parte izquierda de la pantalla será igual al tamaño del aside. El espacio respecto a la parte inferior es el tamaño del footer (líneas 101 a 112).

De esta forma, las secciones se ajustarán a cada uno de los modelos en función de las secciones que presente y el tamaño de las mismas. Esta plantilla se encuentra anclada al elemento “root”, por lo que solo se generará un fichero de este tipo.

```
<%
    $cfg.path = $root.location[0].value + "/public/css";
    $cfg.name = "layout.css";

    var header_size = 0;
    if ($self.layouts[0].header[0]) {
        header_size = parseInt($self.layouts[0].header[0].size[0].value)
    }

    var nav_size = 0;
    if ($self.layouts[0].nav[0]) {
        nav_size = parseInt($self.layouts[0].nav[0].size[0].value)
    }

    var aside_size = 0;
    if ($self.layouts[0].aside[0]) {
        aside_size = parseInt($self.layouts[0].aside[0].size[0].value)
    }

    var footer_size = 0;
    if ($self.layouts[0].footer[0]) {
        footer_size = parseInt($self.layouts[0].footer[0].size[0].value)
    }



%>


html, body{
    height: 100%;
    width: 100%;
    margin: 0 auto;
}

.layout_element {
    box-sizing: border-box;
}

<% if ($self.layouts[0].header[0]) { %>
header{
    display: flex;
    position: absolute;
    /* Posicion */
    top: 0px;
    left: 0px;
    right: 0px;
    /* Tamanyo */
    height: <%= header_size %>px;
    /* Color */
    background-color: <%=$self.layouts[0].header[0].color[0].value%>;
}
<% } %>

<% if ($self.layouts[0].nav[0]) { %>
nav{
    display: flex;
    position: absolute;
    /* Posicion */
    top: <%= header_size %>px;
    left: 0px;
    right: 0px;
    /* Tamanyo */
    height: <%= nav_size %>px;
    /* Color */
    background-color: <%=$self.layouts[0].nav[0].color[0].value%>;
}
<% } %>

<% if ($self.layouts[0].aside[0]) { %>
aside{
    display: flex;
    position: absolute;
    /* Posicion */
    top: <%= header_size + nav_size %>px;
    left: 0px;
    bottom: <%= footer_size %>px;
    /* Tamanyo */
    width: <%= aside_size %>px;
    /* Color */
    background-color: <%=$self.layouts[0].aside[0].color[0].value%>;
    overflow: auto;
}
<% } %>

<% if ($self.layouts[0].footer[0]) { %>
footer{
    display: flex;
    position: absolute;
    /* Posicion */
    left: 0px;
    right: 0px;
    bottom: 0px;
    /* Tamanyo */
    height: <%= footer_size %>px;
    /* Color */
    background-color: <%=$self.layouts[0].footer[0].color[0].value%>;
}
<% } %>

article{
    display: flex;
    position: absolute;
    /* Posicion */
    top: <%= header_size + nav_size %>px;
    left: <%= aside_size %>px;
    right: 0px;
    bottom: <%= footer_size %>px;
    /* Color */
    background-color:  <%=$self.layouts[0].article[0].color[0].value%>;
    overflow: auto;
}
```

Fig. 81 Plantillas de la LPS Documentación online. Código de la plantilla para el fichero layout.css.

Una vez aplicada la variabilidad de los ficheros principales, se implementan las plantillas correspondientes a los contenidos. En la figura **Fig. 82**, se muestra el código de la plantilla data.html. Esta plantilla está anclada al elemento "c_data", por lo que se va a generar un fichero por cada contenido de este tipo que presente el modelo. En la línea 10, se puede observar cómo utilizando un bucle genera una etiqueta html de tipo "th" por cada elemento "data_item" contenido dentro de un elemento "data_header", creando así los elementos de la cabecera de la tabla. En la línea 14, de nuevo se utiliza un bucle para iterar los elementos "data_row" y así crear las etiquetas "tr" que corresponden con cada fila de la tabla. Seguidamente, en la línea 16 se iteran los elementos "data_item" contenidos en cada "data_row" anterior, creando las etiquetas "td" que corresponden a los elementos de una fila. Por último, en la línea 7 se inserta el título que aparecerá sobre la tabla, y en la línea 42 el color del texto del elemento "text_color" del modelo.

En la imagen **Fig. 83**, se encuentra el código de la plantilla para el contenido logotexto.html. Esta plantilla se encuentra anclada al elemento "c_logo_text", y por cada elemento de este tipo presente en el modelo, se va a generar un fichero en el producto final. En la línea 8, el valor src de la etiqueta "img" tomará el valor del elemento "image" del modelo. De la misma forma, en la línea 11, la etiqueta "h1" que corresponde al título toma el valor del elemento "text" del modelo. Por último, se configura el color en la línea 24 en base al elemento "text_color" del modelo.

Hay que tener en cuenta que tanto data.html, como en logotexto.html, pueden generar múltiples ficheros, por ello, el nombre de estos ficheros será dinámico. En la línea 3 de ambos códigos se configura el nombre del fichero en base al valor que toma el elemento "name" del modelo, que corresponde al elemento padre del elemento anclado, es decir, el elemento "content".

```
<%
    $cfg.path = $root.location[0].value + "/public/pages";
    $cfg.name =  $self.parent.name[0].value + ".html";
%>

<div class="data_table_container">
    <div class="data_table_title"><%= $self.title[0].value %></div>
    <table class="data_table">
        <tr class="data_table_tr data_table_tr_head">
            <% for(var i = 0; i < $self.data_header[0].data_item.length; i++){ %>
            <th class="data_table_th data_table_th_head"><%= $self.data_header[0].data_item[i].value %></th>
            <% } %>
        </tr>
        <% for(var i = 0; i < $self.data_row.length; i++){ %>
        <tr class="data_table_tr data_table_tr_body">
            <% for(var j = 0; j < $self.data_row[0].data_item.length; j++){ %>
            <td class="data_table_th data_table_th_body"><%= $self.data_row[i].data_item[j].value %></td>
            <% } %>
        </tr>
        <% } %>
    </table>
</div>
<style>

.data_table_container{
    width: 100%;
    display: flex;
    align-items: center;
    padding-top: 20px;
    flex-direction: column;
}

.data_table_title{
    font-size: x-large;
    font-weight: bold;
    margin-bottom: 10px;
}

.data_table {
    border-collapse: collapse;
    height: fit-content;
    color: <%= $self.text_color[0].value %>;
}

.data_table_tr{
    border: solid 1px black;
}

.data_table_th{
    margin: 0 auto;
    padding: 5px;
}

</style>
```

Fig. 82 Plantillas de la LPS Documentación online. Código de la plantilla para el fichero data.html. En la parte derecha, el código de la plantilla para el fichero logotexto.html.

```
1  <%
2      $cfg.path = $root.location[0].value + "/public/pages";
3      $cfg.name = $self.parent.name[0].value + ".html";
4  %>
5
6  <div class="logoTexto_container">
7      <div class="logoTexto_image_container">
8          <img class="logoTexto_image" src="<%= $self.image[0].value %>" alt="">
9      </div>
10     <div class="logoTexto_texto_container">
11         <h1><%= $self.text[0].value %></h1>
12     </div>
13
14
15 </div>
16
17 <style>
18
19     .logoTexto_container{
20         display: flex;
21         align-items: center;
22         height: 100%;
23         width: 100%;
24         color: <%= $self.text_color[0].value %>;
25     }
26     .logoTexto_image_container{
27         height: 80%;
28         width: auto;
29         padding-left: 20px;
30         padding-right: 20px;
31         box-sizing: border-box;
32     }
33
34     .logoTexto_image{
35         height: 100%;
36     }
37 </style>
```

Fig. 83 Plantillas de la LPS Documentación online. Código de la plantilla para el fichero logotexto.html

En la parte izquierda de la figura **Fig. 84**, se encuentra el código de la plantilla para el fichero fullbanner.html. Esta plantilla se encuentra anclada al elemento “c_banner” y generará un fichero por cada elemento de este tipo presente en el modelo. En este caso, solo es necesario aplicar la variabilidad para la imagen a mostrar. Para ello, sobre la etiqueta “img”, en la línea 7, se emplea el valor del elemento “image” del modelo.

En la parte derecha encontramos el código de la plantilla para el fichero simpletext.html. En este caso la variabilidad también se presenta sencilla, configurando el texto en la línea 6 en base al elemento “text” del modelo y el color sobre la línea 10 con el elemento “text_color”.

Ambas plantillas pueden generar varios ficheros y en la línea 3 de ambos códigos se puede observar como el nombre del fichero se generará en función el elemento “name” del padre del elemento anclado (elemento “content”).

```
<%
    $cfg.path = $root.location[0].value + "/public/pages";
    $cfg.name = $self.parent.name[0].value + ".html";
%>

<div class="fullBannerImage_container">
    <img class="fullBannerImage" src="<%= $self.image[0].value %>"></img>
</div>

<style>
    .fullBannerImage_container{
        display: flex;
        align-items: center;
        height: 100%;
        width: 100%;
        overflow: hidden;
    }
    .fullBannerImage{
        height: fit-content;
        width: 100%;
    }
</style>
```

```
<%
    $cfg.path = $root.location[0].value + "/public/pages";
    $cfg.name = $self.parent.name[0].value + ".html";
%>

<div class="simpletext_text"><%= $self.text[0].value %><div>

<style>
.simpletext_text{
    color:<%= $self.text_color[0].value %>
}
</style>
```

Fig. 84 Plantillas de la LPS Documentación online. En la parte izquierda, el código de la plantilla para el fichero fullbanner.html. En la parte derecha, el código de la plantilla para el fichero simpletext.html.

El contenido “document” es el más complejo de todos los planteados. En la figura **Fig. 85**, se puede observar la estrategia a seguir para generar cada uno de los componentes que pueden aparecer en la página. En concreto en las líneas 7 y 8 encontramos un bucle que itera todos los componentes utilizando el atributo “all” del elemento anclado (en la variable “$self”), y a continuación, se utiliza una estructura switch case, que utiliza como clave el atributo “key”, para construir el código correspondiente a cada elemento presente en el modelo. Se puede observar cómo en algunos casos, simplemente se utiliza el valor del elemento (líneas 11, 15, 19, 23 y 35) y en otros casos, estos elementos contienen otros elementos, como en el caso de los elementos “link” que contienen un elemento “content_name” para cargar el contenido en la sección “article” y “text” que corresponde con el texto que se mostrará en el link (línea 27), o los elementos “code_block” que contienen un elemento “text” para mostrar el texto que irá dentro del bloque de código y el elemento “info” que se muestra información sobre el código (línea 31).

En las líneas 57, 65, 70, 106 y 111, se utiliza el elemento “text_color” del modelo para aplicar color a los textos. En la línea 124 se utiliza el elemento “link_color” para aplicar color, cuando los links reciben el foco.

```
<%
    $cfg.path = $root.location[0].value + "/public/pages";
    $cfg.name = $self.parent.name[0].value + ".html";
%>

<section class="document_section_container">
<%  for(var i = 0; i < $self.all.length; i++){
        switch ($self.all[i].key) {
        case "title":
          %>
          <div class="document_title"><%= $self.all[i].value %></div> <%
          break;
        case "subtitle":
          %>
          <div class="document_section_title"><%= $self.all[i].value %></div> <%
          break;
        case "text":
          %>
          <div class="document_text"><%= $self.all[i].value %></div> <%
          break;
        case "image":
          %>
          <div class="document_image"><img src="<%= $self.all[i].value %>"></img></div> <%
          break;
        case "link":
          %>
          <div class="document_link" onclick='loadPage("article", "<%= $self.all[i].content_name[0].value %>.html")'><%= $self.all[i].text[0].value %></div> <%
          break;
        case "code_block":
          %>
          <div class="document_code_container"><div class="document_code"><code><%= $self.all[i].text[0].value %>
          </code></div><div class="document_code_info"><%= $self.all[i].info[0].value %></div></div> <%
          break;
        case "space":
          %>
          <div class="document_space_<%= $self.all[i].value %>"></div> <%
          break;
        default:
          break;
      }
    }
%>

</section>

<style>
    .document_section_container{
        padding: 20px;
        box-sizing: border-box;
    }

    .document_title{
        font-size: xx-large;
        width: 100%;
        text-align: center;
        margin-bottom: 10px;
        font-weight: bold;
        color: <%= $self.text_color[0].value %>;
    }

    .document_section_title{
        font-size: large;
        width: 100%;
        margin-bottom: 10px;
        font-weight: bold;
        color: <%= $self.text_color[0].value %>;
    }

    .document_text{
        text-align: justify;
        color: <%= $self.text_color[0].value %>;
    }

    .document_image{
        margin-top: 20px;
        margin-bottom: 20px;
        width: 100%;
    }

    .document_image > img{
        width: 100%;
        border-radius: 10px;
    }

    .document_space_1{
        height: 10px;
    }

    .document_space_2{
        height: 30px;
    }

    .document_space_3{
        height: 50px;
    }

    .document_space_4{
        height: 70px;
    }

    .document_space_5{
        height: 90px;
    }

    .document_code_container {
        width: 100%;
        color: <%= $self.text_color[0].value %>;
    }

    .document_code{
        width: 100%;
        border: solid 1px <%= $self.text_color[0].value %>;
        min-height: 50px;
        margin-bottom: 20px;
    }

    .document_code_info {
        width: 100%;
        text-align: end;
        font-size: small;
    }


    .document_link{
        color: <%= $self.link_color[0].value %>;
        cursor: pointer;
        text-decoration: underline;
        height: 100px;
    }


</style>
```

Fig. 85 Plantillas de la LPS Documentación online. Código de la plantilla para el fichero document.html.

Esta plantilla se encuentra anclada al elemento “c_document”, por lo que al igual que en los casos anteriores, va a generar un fichero por cada elemento de este tipo presente en el modelo. Por ello, en la línea 3 se genera el nombre de forma dinámica en función del nombre del elemento “name” presente en el elemento padre.

Las plantillas creadas, tanto para el fichero hbasicmenu.html, como el fichero vbasicmenu.html son muy similares en cuanto a estructura. En las figuras **Fig. 86** y **Fig. 87**, se puede observar el código de ambas plantillas. Estas plantillas se encuentran ancladas a sus respectivos elementos “c_hmenu” y “c_vmenu”, por lo que van a generar un fichero por cada elemento de estos tipos que aparezca en el modelo. Por ello, al igual que en casos anteriores, el nombre del fichero se compone de forma dinámica en función del elemento “name” presente en el elemento padre (elemento “content”). En la línea 7 de ambas plantillas se puede observar cómo se iteran los elementos de tipo “menú_item” creando un menú por cada uno de ellos, donde se configura el texto en base al elemento “text” y el contenido a cargar sobre la sección “article” mediante el elemento “content_name”. En ambas plantillas se configura, tanto el color del texto, utilizando el elemento “text_color” del modelo (línea 24 para hbasicmenu.html y línea 27 para vbasicmenu.html), como el color que toma el texto cuando el cursor pasa sobre él, utilizando en este caso el elemento “hover_color” (línea 30 para hbasicmenu.html y línea 33 para vbasicmenu.html).

```
1  <%
2      $cfg.path = $root.location[0].value + "/public/pages";
3      $cfg.name = $self.parent.name[0].value + ".html";
4  %>
5
6  <div class='hbasicmenu_container'>
7      <% for(var i = 0; i < $self.menu_item.length ; i++) { %>
8      <div class='hbasicmenu_item' onclick='loadPage("article", "<%= $self.menu_item[i].content_name[0].value %>.html")'><%= $self.menu_item[i].text[0].value %></div>
9      <% } %>
10 </div>
11
12 <style>
13     .hbasicmenu_container{
14         height: 100%;
15         width: 100%;
16         padding: 5px;
17         display: flex;
18         align-items: center;
19         box-sizing: border-box;
20         font-weight: bold;
21     }
22
23     .hbasicmenu_item{
24         color:<%= $self.text_color[0].value %>;
25         cursor: pointer;
26         margin-left: 10px;
27         margin-right: 10px;
28     }
29     .hbasicmenu_item:hover{
30         color:<%= $self.hover_color[0].value %>;
31     }
32 </style>
```

Fig. 86 Plantillas de la LPS Documentación online. Código de la plantilla para el fichero hbasicmenu.html.

```
<%
    $cfg.path = $root.location[0].value + "/public/pages";
    $cfg.name = $self.parent.name[0].value + ".html";
%>

<div class='vbasicmenu_container'>
    <% for(var i = 0; i < $self.menu_item.length ; i++) { %>
    <div class='vbasicmenu_item' onclick='loadPage("article", "<%= $self.menu_item[i].content_name[0].value %>.html")'><%= $self.menu_item[i].text[0].value %></div>
    <% } %>
</div>

<style>
    .vbasicmenu_container{
        height: 100%;
        width: 100%;
        padding: 5px;
        padding-top: 10px;
        display: flex;
        align-items: center;
        box-sizing: border-box;
        font-weight: bold;
        flex-direction: column;
    }


    .vbasicmenu_item{
        color: <%= $self.text_color[0].value %>;
        cursor: pointer;
        margin-top: 10px;
        margin-bottom: 10px;
    }
    .vbasicmenu_item:hover{
        color: <%= $self.hover_color[0].value %>;
    }
</style>
```

Fig. 87 Plantillas de la LPS Documentación online. Código de la plantilla para el fichero vbasicmenu.html

4.2.6. Ingeniería de aplicación / Espacio del problema (EAP)

Partiendo del dominio de la LPS, SPL puede generar modelos listos para ser configurados, en la figura **Fig. 88**, se muestra un ejemplo de modelo autogenerado.

```
root(
    layouts(
        header(
            size(""),
            color(""),
            content_name("")
        ),
        nav(
            size(""),
            color(""),
            content_name("")
        ),
        aside(
            size(""),
            color(""),
            content_name("")
        ),
        footer(
            size(""),
            color(""),
            content_name("")
        ),
        article(
            color(""),
            content_name("")
        )
    ),
    name(""),
    location(""),
    content(
        name(""),
        c_simple_text(
            text(""),
            text_color("")
        ),
        c_document(
            title(""),
            subtitle(""),
            text(""),
            image(""),
            code_block(
                text(""),
                info("")
            ),
            link(
                text(""),
                content_name("")
            ),
            space(""),
            text_color(""),
            link_color("")
        ),
        c_data(
            data_header(
                data_item("")
            ),
            data_row(
                data_item("")
            ),
            title(""),
            title(""),
            text_color("")
        ),
        c_hmenu(
            menu_item(
                text(""),
                content_name("")
            ),
            text_color(""),
            hover_color("")
        ),
        c_vmenu(
            menu_item(""),
            text_color(""),
            hover_color("")
        ),
        c_logo_text(
            image(""),
            text(""),
            text_color("")
        ),
        c_banner(
            image("")
        )
    )
)
```

Fig. 88 Modelo autogenerado para la LPS Documentación online.

El DSL generado por SPL Studio para la creación de modelos se basa en lenguaje JavaScript, por lo que es posible utilizar toda la potencia que este ofrece. El uso de variables en este punto, permite que se puedan crear modelos dinámicos, es decir, que con un parámetro se modifiquen varias partes de un modelo. En la figura **Fig. 89**, se

muestra un ejemplo de uso de variables durante la creación de un modelo para la LPS. En concreto se parametriza una paleta de colores, que será utilizada por todo el modelo, pudiendo de esta forma, modificar varias partes del modelo que deberían tener el mismo color, simplemente modificando el color en la paleta de colores. En las líneas 22 y 32 se puede observar cómo se encuentra configurado el mismo color para el elemento "header" como para el elemento "footer", por lo que cuando el valor de la variable "cl.background_1" cambie, automáticamente cambiarán ambos elementos.

```
1  //color palette
2  var col = {
3      background_1 : "#363537",
4      background_1_text : "white",
5      background_1_contrast : "#C14953",
6
7      background_2 : "#197BBD",
8      background_2_text : "white",
9      background_2_contrast : "#FCFF6C",
10
11     background_pages : "#C2B8B2",
12     background_pages_text : "black",
13     background_pages_contrast : "#197BBD"
14 }
15
16 root(
17     name("Documentación equipo Autorización"),
18     location("docAut2"),
19     layouts(
20         header(
21             size("100"),
22             color(col.background_1),
23             content_name("cabecera")
24         ),
25         nav(
26             size("30"),
27             color(col.background_2),
28             content_name("hmenu")
29         ),
30         footer(
31             size("20"),
32             color(col.background_1),
33             content_name("emptypage")
34         ),
```

Fig. 89 Ejemplo de programación JavaScript sobre un modelo de la LPS.

En la figura **Fig. 90**, se muestra un ejemplo de modelo que lleva como nombre "Documentación equipo Autorización", como se puede ver en la línea 17, y se genera sobre un directorio llamado "docAut2" (línea 18). Este producto cuenta con una sección "header" (línea 20), que carga como contenido principal (por defecto), el contenido "cabecera" (línea 23), una sección "nav" (línea 25) cuyo contenido principal es el contenido "hmenu" (28), una sección "footer" que carga como contenido principal, el contenido con nombre "emptypage" (línea 33) y por último, una sección "article" (línea 35) que carga como contenido principal el contenido con nombre "proyecto" (línea 7). Para cada una de las secciones se indica el color, utilizando variables de la paleta de colores (líneas 22, 27, 32 y 36), y el tamaño de la misma (líneas 21, 26 y 31).

A continuación, se pueden observar cada uno de los contenidos:

- **Emptyptage** (línea 40): Este contenido es de tipo "c_simple_text", que muestra un texto, en este caso vació (línea 44). También se configura el color del texto (línea 43), sin embargo, en este caso no tendrá ningún efecto.

- **Users** (línea 47): Contenido de tipo “c_data” donde se configura una tabla con dos columnas “id” y “pass” (línea 52), y dos filas de datos, cada una con un usuario y una contraseña (líneas 56 y 60). En la línea 51 se configura el color de los textos que van a aparecer.

- **Hmenu** (línea 66): Se trata de un contenido de tipo “c_hmenu” configurado con dos elementos de tipo “menú_items” (líneas 71 y 75), donde, el primer elemento muestra como texto “Proyectos” y hace referencia al contenido “proyectos” y el segundo muestra como texto “Usuarios” y hace referencia al contenido “users”. En la línea 69 se configura el color de los textos, mientras que en la línea 70 se configura el color de los textos cuando el cursor tiene el foco sobre ellos.

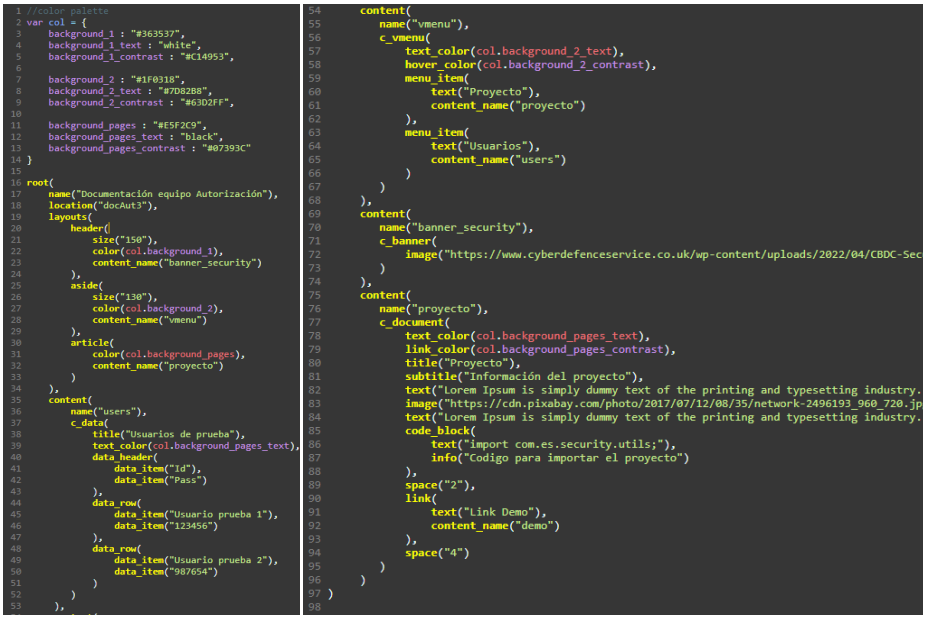
- **Cabecera** (línea 81): Contenido de tipo “c_logo_text” que se configura con una imagen a mostrar y un texto (líneas 85 y 86). En la línea 84 se configura el color del texto.

- **Proyecto** (línea 89): En este caso, el contenido es de tipo “c_document”. Para él se configura, un título (línea 94), un subtítulo (línea 95), dos bloques de texto (líneas 96 y 98), una imagen (línea 97), un bloque de código (línea 99) y un link (línea 104) comprendido entre dos espacios (líneas 104 y 108). El color para el texto se define en la línea 9, mientras que el color para los links se define en la línea 93.

Fig. 90 Ejemplo de modelo para el caso de estudio 2.

En la figura **Fig. 91**, se presenta una variación del producto de la figura **Fig. 90**. En cuanto las secciones, se han eliminado las secciones “nav” y “footer”, y se ha incluido la sección “aside” (línea 25), que carga como contenido principal el contenido con el

nombre “vmenu”. La sección “header” ahora carga por defecto el contenido con el nombre “banner_security”.



```
//color palette
var col = {
    background_1 : "#363537",
    background_1_text : "white",
    background_1_contrast : "#C14953",

    background_2 : "#1F0318",
    background_2_text : "#7D82B8",
    background_2_contrast : "#63D2FF",

    background_pages : "#E5F2C9",
    background_pages_text : "black",
    background_pages_contrast : "#07393C"
}

root(
    name("Documentación equipo Autorización"),
    location("docAut3"),
    layouts(
        header(
            size("150"),
            color(col.background_1),
            content_name("banner_security")
        ),
        aside(
            size("130"),
            color(col.background_2),
            content_name("vmenu")
        ),
        article(
            color(col.background_pages),
            content_name("proyecto")
        )
    ),
    content(
        name("users"),
        c_data(
            title("Usuarios de prueba"),
            text_color(col.background_pages_text),
            data_header(
                data_item("Id"),
                data_item("Pass")
            ),
            data_row(
                data_item("Usuario prueba 1"),
                data_item("123456")
            ),
            data_row(
                data_item("Usuario prueba 2"),
                data_item("987654")
            )
        )
    ),
    content(
        name("vmenu"),
        c_vmenu(
            text_color(col.background_2_text),
            hover_color(col.background_2_contrast),
            menu_item(
                text("Proyecto"),
                content_name("proyecto")
            ),
            menu_item(
                text("Usuarios"),
                content_name("users")
            )
        )
    ),
    content(
        name("banner_security"),
        c_banner(
            image("https://www.cyberdefenceservice.co.uk/wp-content/uploads/2022/04/CBDC-Sec
        )
    ),
    content(
        name("proyecto"),
        c_document(
            text_color(col.background_pages_text),
            link_color(col.background_pages_contrast),
            title("Proyecto"),
            subtitle("Información del proyecto"),
            text("Lorem Ipsum is simply dummy text of the printing and typesetting industry.
            image("https://cdn.pixabay.com/photo/2017/07/12/08/35/network-2496193_960_720.jp
            text("Lorem Ipsum is simply dummy text of the printing and typesetting industry.
            code_block(
                text("import com.es.security.utils;"),
                info("Codigo para importar el proyecto")
            ),
            space("2"),
            link(
                text("Link Demo"),
                content_name("demo")
            ),
            space("4")
        )
    )
)
```

Fig. 91 Ejemplo de modelo para el caso de estudio 2.

En lo que se refiere a contenidos, se ha eliminado el contenido “emptypage” y se han creado dos nuevos contenidos:

- **Vmenu** (línea 54): Se trata de un contenido de tipo “c_vmenu” que cuenta con la misma configuración que el contenido “hmenu” de la figura **Fig. 90**, pues los sub-elementos de tipo “c_hmenu” y “c_vmenu” son compatibles.
- **Banner_security** (línea 69): Este contenido es de tipo “c_banner” y únicamente se encuentra configurado con una imagen (línea 72).

Por último, la paleta de colores también ha sido renovada, por lo que los elementos que utilizan variables de esta paleta tomarán valores distintos.

4.2.7. Ingeniería de aplicación / Espacio de la solución (EAS)

Finalmente, la compilación de los diferentes modelos da como resultado los productos finales.

En la figura **Fig. 92**, se muestra el resultado de la compilación del modelo presentado en la figura **Fig. 90**. En este caso, se puede observar que el resultado final cuenta con

las secciones opcionales indicadas (“header”, “nav”, “footer”), que se ajustan correctamente al tamaño de la pantalla. El contenido ”hmenu” que corresponde con un menú horizontal, cargado sobre la sección “nav”, muestra dos opciones de menú, la opción “Proyecto” que carga el contenido con el nombre “proyecto” sobre la sección “article” y la opción “Usuarios”, que, de la misma forma, carga el contenido con el nombre “users” sobre la sección “article”. También se puede observar, como los colores de toda la página se configuran en función de la paleta de colores declarada.

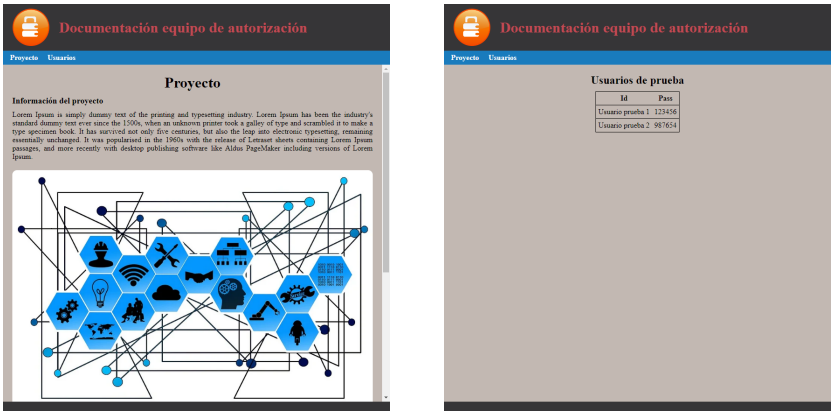


Fig. 92 Ejemplo de producto final para el caso de estudio 2.

La siguiente figura **Fig. 93**, corresponde con el modelo de la variante presentada en la figura **Fig. 91**, en este caso, se puede ver como las secciones “nav” y “footer” han sido eliminadas, y en su lugar, se ha incorporado la sección “aside”, con el mismo menú, esta vez en formato vertical ya que corresponde con un elemento de tipo “v_menu”. En la cabecera también se puede observar que ahora se muestra el contenido “c_banner” en lugar del anterior “c_logo_text”. Finalmente, se muestra como todos los colores corresponden con la nueva paleta definida.

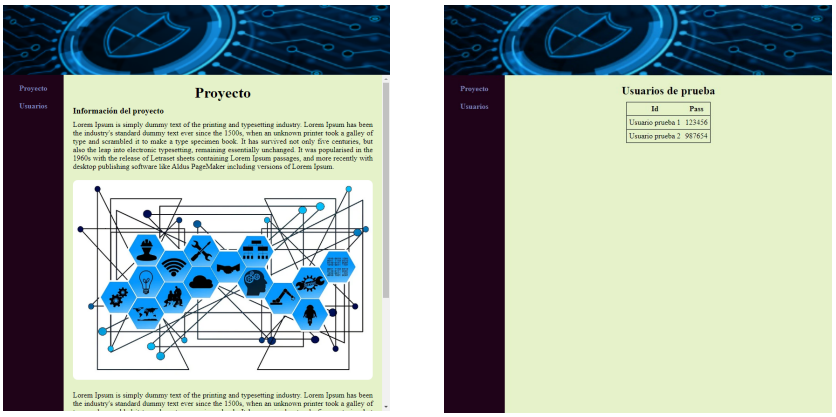


Fig. 93 Ejemplo de producto final para el caso de estudio 2.

4.3. Resultados

SPL Studio ofrece una interfaz de trabajo basada en la web, destacando sobre el resto de herramientas analizadas por no requerir de la configuración de un entorno de trabajo. Los ingenieros de dominio pueden comenzar a trabajar de forma instantánea accediendo a la web, y no es necesario que distribuyan la LPS a los ingenieros de aplicación. Del mismo modo, estos pueden acceder y trabajar sobre la LPS. Cabe destacar, que los cambios realizados sobre la LPS no requieren de una nueva distribución de la LPS, algo que mejora notablemente el proceso de desarrollo.

La herramienta SPL Studio se puede considerar un hibrido entre las herramientas enfocadas al modelado de características, y las herramientas destinadas al diseño de lenguajes de dominio. Principalmente se intenta conseguir, la facilidad de modelado en el espacio del problema de las primeras, con la flexibilidad para la generación de productos de cualquier naturaleza, en el espacio de la solución de las segundas.

En cuanto a la parte ingeniería de dominio y espacio del problema (EDP), SPL Studio explota la creación del dominio a través de una interfaz gráfica, una formula bastante utilizada entre las aplicaciones orientadas al desarrollo de modelos de características, como FeatureIDE, Gears o Pure::variants, para simplificar la tarea del usuario. Sin embargo, con las aplicaciones orientadas al diseño de lenguajes de dominio, el proceso sería mucho más costoso, pues sería necesario crear toda la gramática del lenguaje para dar soporte al modelado de productos. SPL Studio es capaz de generar esta gramática a partir del dominio creado de forma gráfica.

En el ámbito de la ingeniería de aplicación y espacio del problema, SPL Studio propone un sistema de modelado homogéneo para cualquier LPS, obtenido a partir del dominio de la misma. Por lo que, de nuevo, evita la necesidad de crear una gramática para la creación de modelos, como es común en aplicaciones para el modelado de características.

Con el fin de acelerar el proceso de modelado, SPL Studio es capaz de generar nuevos modelos vacíos en base al lenguaje de dominio de la LPS, de este modo, el modelado no parte desde cero, sino de un modelo básico.

Sobre la ingeniería de dominio y espacio de la aplicación, las aplicaciones orientadas al modelado de lenguajes de dominio, habitualmente permiten la generación de cualquier tipo de código mediante complementos y herramientas adicionales, sin embargo, las aplicaciones orientadas a modelado de características están más limitadas en este sentido.

SPL Studio, al igual que algunas aplicaciones orientadas al modelado de lenguajes de dominio, ofrece una forma de trabajo basada en plantillas, que permite la generación de cualquier código o texto que se desee. Para ello, propone la generación de estas

plantillas basándose en scriptlets, una de las técnicas más extendidas, ofreciendo una línea de aprendizaje baja, a diferencia de otras aplicaciones como Xtend, que utilizan una sintaxis propia, lo que puede complicar su aprendizaje.

Otra mejora que ofrece SPL Studio, respecto del resto de herramientas, es la automatización para la creación de plantillas en base a un prototipo, que acelera el proceso considerablemente, evitando que estas tengan que ser creadas una a una.

En cuanto a la obtención del producto final en el ámbito de la ingeniería del dominio y espacio de la aplicación, al igual que otras herramientas analizadas, SPL Studio permite la generación del producto final a través de su interfaz web, desde la cual, es posible visualizar o descargar dicho producto. No obstante, también se presenta un modo novedoso obtener este producto, pues es posible generarlo a través de un servicio que puede ser accedido desde cualquier punto. Lo que permite su integración en flujos de trabajo dinámicos y continuos, donde un cambio en la LPS puede ser distribuido a entornos finales automáticamente.

El tiempo de construcción para el caso de estudio 1 fue de 3 horas aproximadamente, y para la creación de la LPS solo se necesitó media hora más. El tiempo de construcción del prototipo para el caso de estudio 2 ha supuesto aproximadamente 30 horas, mientras que la construcción de la LPS a partir del prototipo supuso de unas 20 horas. En ambos casos, la creación de la LPS es menor al desarrollo de un segundo producto, por lo que se puede concluir que SPL Studio, es una alternativa viable en pequeños proyectos, donde la implantación de otras herramientas, como en los casos de estudio, puede suponer más tiempo que el simple hecho de crear el prototipo. En la tabla **Tabla 13** se muestran tabla de características de la herramienta SPL Studio.

Tabla 13. Tabla de características para SPL Studio.

Característica	Valoración
Dificultad de uso	Baja
Necesita complementarse con otras herramientas	No
Necesidad de instalación y configuración	No
Tiene limitaciones en cuanto al lenguaje de generación.	No
Necesita redistribuir la línea de productos en cada cambio	No
Es multiplataforma	Si

Teniendo en cuenta el listado de artefactos presentados en la tabla **Tabla 1** por Weiss (1999), para que una LPS esté completa, en la tabla **Tabla 14**, se muestra la relación entre la aplicación SPL Studio y este listado de artefactos, donde se puede observar una clara reducción de la carga de trabajo, que se produce gracias a los diferentes mecanismos empleados, que como se puede observar, en la mayoría de los casos agiliza o evita el trabajo, pero en ninguno de ellos esta carga se ve penalizada.

Tabla 14. Reducción de carga de trabajo (**T**) para el desarrollo de artefactos con SPL Studio. (**N**) No afecta al trabajo, (**A**) Agiliza el trabajo, (**E**) Evita el trabajo.

Artefacto	Funcionalidad	(T)
D1	El editor de dominios permite la creación de un dominio de forma visual.	(A)
D2	El editor de permite aportar información extra sobre las relaciones de los elementos del dominio.	(A)
D3	La creación de prototipos de modelos hace las veces de modelo de decisiones.	(E)
D4	El lenguaje de modelado se genera automáticamente en función del domino.	(E)
D5	El editor de familias permite creación de plantillas en función a un prototipo.	(A)
D6	El esquema visual del dominio hace las veces de mapa de composición	(E)
D7	Todas las herramientas necesarias para el entorno de ingeniería de dominio se incluyen en SPL Studio.	(E)
D8	La librería para la creación de plantillas está incluida en SPL Studio	(E)
D9	Las herramientas para la generación de código se incluyen en SPL Studio.	(E)
D10	Las herramientas de análisis para los modelos se incluyen en SPL Studio	(E)
D11	Se incluye en la documentación general de SPL Studio	(E)
D12	Se incluye en la documentación general de SPL Studio	(E)
A1	SPL Studio permite la especificación de modelos en el lenguaje de modelado derivado del dominio, incluyendo marcado de código.	(A)
A2	El código generado del producto final puede ser previsualizado y descargado	(N)
A3	La documentación generada del producto final puede ser previsualizada y descargada.	(N)

En la figura **Fig. 94**, se puede observar la distribución de los diferentes artefactos entre los espacios de trabajo que proporciona SPL Studio.

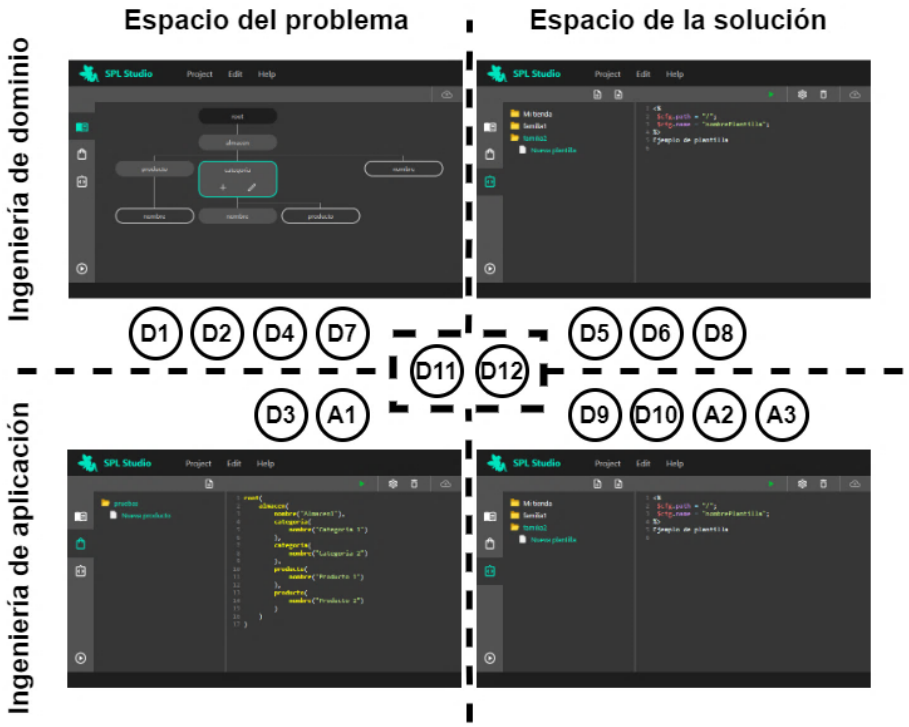


Fig. 94 Esquema de creación de artefactos por cada espacio de trabajo.

Capítulo 5: Conclusiones y trabajo futuro

5.1. Conclusiones

La creación de LPS no es una tarea sencilla, exige un gran desafío inicial que es crucial para que un proyecto sea exitoso. Para afrontarlo, existen numerosas herramientas que cubren diferentes áreas, y ayudan a los desarrolladores a construir los diferentes artefactos necesarios para llevar a cabo la tarea. Sin embargo, las herramientas actuales evolucionan hacia modelos enfocados a abarcar proyectos cada vez más grandes. En esta evolución, se crean herramientas multipropósito que generalmente ofrecen un gran abanico de funcionalidades, pero en su defecto, aumentan la complejidad. Esto hace que se esté perdiendo la oportunidad de aplicar el enfoque de desarrollo basado en LPS sobre productos más pequeños o que requieran una rápida implantación y no necesiten un gran despliegue tecnológico.

La construcción de herramientas sencillas enfocadas únicamente a este propósito puede ayudar a que más proyectos puedan ser desarrollados de esta forma y con ello aprovechar todos los beneficios derivados de aplicar este enfoque.

Durante el estudio, se ha comprobado cómo, con la construcción de una herramienta cuyo único propósito es construir LPS, el proceso de implantación se puede acelerar en gran medida. Haciendo especial hincapié en pequeños proyectos donde de no ser así, no se llegaría a barajar la posibilidad de utilizar LPS para su construcción.

Con una primera versión de la herramienta construida, ya es posible crear todos los artefactos necesarios para implantar una LPS de una forma sencilla y rápida. La herramienta construida, cuenta con una amplio margen de mejora, pero siempre con el objetivo de simplificar y agilizar el proceso, pues este podría ser el camino para conseguir un uso más extendido de este paradigma, que aún tiene mucho que ofrecer.

5.2. Trabajo futuro

En futuros trabajos, se abren dos rutas de trabajo interesantes. La puesta en producción de la herramienta construida y la mejora de funcionalidad de la misma.

5.2.1. Puesta en producción

La aplicación SPL Studio se ha creado con el objetivo de investigar soluciones en el ámbito de las líneas de productos de software, por ello, se han dejado a un lado algunos aspectos que serían indispensables, si en algún momento esta aplicación promocionase a un entorno de producción.

5.2.1.1. Gestión de usuarios y permisos

La gestión de usuarios está pendiente de implementar. Se pretende establecer un sistema de gestión de usuarios que permita compartir proyectos entre distintos usuarios o grupos de usuarios.

Los permisos que obtendrá cada usuario o grupo de usuarios sobre un proyecto podrán ser configurados por el propietario del proyecto, que decidirá, que elementos del proyecto pueden modificar de forma independiente, de esta forma, puede haber usuarios específicos dentro de un mismo para un espacio de trabajo concreto.

5.2.1.2. Trazabilidad

Una característica indispensable en entornos de producción es un sistema de trazabilidad que permita acceder controlar de forma detallada la actividad de la aplicación, con el fin de identificar cualquier tipo de problema que se pueda dar.

5.2.2. Mejora de funcionalidad

En cuanto a la mejora de la funcionalidad, se abren algunos frentes de investigación que podrían ayudar tanto a mejorar el proceso de construcción de las LPS, como a la posibilidad de aplicarlas de una forma más efectiva en diferentes ámbitos.

5.2.2.1. Composición de líneas de productos de software

Algo que se echa en falta durante la construcción de una LPS, es la posibilidad de reutilizar código dentro de ella, por lo tanto, una de las características que se implementará en un futuro, es la posibilidad de utilizar una LPS dentro de otra LPS. De esta forma, se podrían dividir los sistemas complejos en otros más pequeños facilitando también el desarrollo.

Un ejemplo de ello puede ser una línea de productos que a partir de un DSL sea capaz de generar código en diferentes lenguajes de programación, este sistema puede ser difícil de implementar, pero una vez creado, puede ser muy aprovechable en multitud de LPS. Con esta funcionalidad se pretende mejorar la velocidad de construcción especialmente en LPS que ya cuentan con aspectos comunes a otras ya creadas.

Desde el punto de vista técnico, construir esta característica supone un gran reto, pues existen problemas que en estos momentos estarían pendientes de resolver, como la colisión entre los DSL de unas LPS y otras, es decir, que el DSL de la LPS que se reutiliza tenga elementos similares al DSL de la LPS que la utiliza.

5.2.2.2. Librerías de usuario

Al igual que en el caso anterior, se pretende mejorar la capacidad de reutilización dentro de una LPS, en concreto, durante la creación de plantillas, donde se echa en falta la posibilidad de compartir funcionalidad entre estas. Se pretende dar la opción al usuario de importar unas plantillas dentro de otras, de esta forma, se podrían crear plantillas con fragmentos que podrían ser reutilizados en diferentes puntos de la LPS, o del mismo modo, plantillas puramente funcionales en forma de librería que se puedan utilizar en el punto donde se importen.

A diferencia del caso anterior, esta mejora no supone una complicación excesiva y podría ser una de las primeras características que se incorporen.

5.2.2.3. Analizadores integrados

En numerosas ocasiones, el modelo de una LPS no se define de forma manual, es decir, este modelo tiene otro tipo de origen, por ejemplo, la construcción de un modelo en función de un contrato definido en formato XML, JSON, CSV, etc. En este punto, se plantea el uso de analizadores para generar modelos. Como punto de partida, se pretende crear analizadores para los formatos más extendidos, que puedan ser utilizados de forma libre por el usuario para cargar modelos del lenguaje, no obstante, existen casos en los que los modelos deben ser obtenidos a partir de códigos o textos que no corresponden con estos formatos.

El objetivo final es crear un sistema con el que un usuario pueda construir de forma sencilla sus analizadores. La idea es que el sistema permita configurar cómo capturar el valor de cada elemento del domino, ya sea directamente utilizando una expresión regular, o mediante un asistente que genere dicha expresión regular. En el momento que se capture el valor de un componente padre, sus componentes hijos podrán extraer su valor desde cero, o a partir del valor capturado por el padre. De esta forma, se podrá crear una jerarquía de expresiones regulares que permitirá una construcción relativamente sencilla.

En la figura **Fig. 95**, se muestra un ejemplo de un dominio que, en el ámbito de la programación orientada a objetos, permite modelar clases con métodos y atributos, que podrían ser generadas en diferentes lenguajes de programación. En este caso, la creación de analizadores, permitiría leer un lenguaje y transformarlo en otro de forma automática.

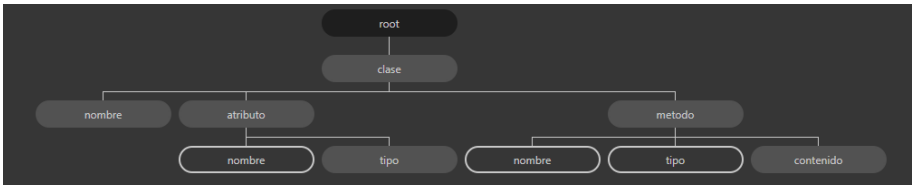


Fig. 95 Ejemplo de dominio

Supongamos que se quiere obtener el modelo a partir del código de la siguiente clase:

```
public class Example {
    public String var1;
    public Char var2;

    public void method1() {
    }

    public void method2() {
    }
}
```

En primer lugar, se crearía una expresión regular para el elemento "clase" que obtenga el código de la clase, que, en este caso, sería todo el código. Seguidamente a partir de ese código, se crearía una expresión regular para el elemento "atributo" que obtenga únicamente el código de los atributos, en este caso encontraría dos atributos:

```
public String var1;

public Char var2;
```

Sobre este código se creará una expresión regular para el elemento "nombre" que obtenga el código del nombre del atributo y de igual forma para el elemento "tipo". El analizador creará el modelo en base a los valores de los elementos hoja. En este caso crearía el modelo de la siguiente forma:

```
clase(
   atributo(
      nombre("var1"),
      tipo("String")
   ),
   atributo(
      nombre("var2"),
      tipo("Char")
   )
)
```

Del mismo modo se configuraría el analizador para el resto de elementos.

Es importante, que el analizador sea independiente y para cada dominio, puedan implementarse varios analizadores, ya que el origen de los modelos puede variar.

5.2.2.4. Control de versiones y entornos

La construcción de LPS utilizando SPL Studio no dispone de un sistema de control de versiones. Sería interesante incorporar esta característica, mediante la cual, se puedan publicar diferentes versiones de la LPS, por un lado, permitiría seguir trabajando sobre ella sin afectar a otros usuarios, y por otro serviría de punto de seguridad al que poder volver si algo sale mal durante el desarrollo. Este control de versiones podría

combinarse con una gestión de entornos, donde las versiones puedan ser publicadas en un entorno especifico, así, se podrían integrar entornos de pruebas en el flujo de trabajo.

5.2.2.5. Tablas de valores para los elementos

Una característica principal de la LPS es la libertad a la hora de crear el dominio, pudiendo establecer sin restricciones la jerarquía que toman los elementos, sin embargo, lo que facilita la creación de este dominio, en algunos casos puede complicar la construcción de las plantillas, por ejemplo, supongamos que tenemos una LPS para la creación de algún tipo de software que trata con tallas. En este caso tenemos un elemento del dominio que es "talla", donde actualmente el usuario puede libremente en su modelo colocar el atributo que desee, por ejemplo "M", "S", "XL", etc. En el momento de realizar la construcción de la plantilla para este caso, se debe controlar que el usuario haya introducido alguno de los valores que se permiten, lanzando una excepción desde la plantilla de forma manual cuando se recupera un valor del modelo erróneo, además, si dentro de la plantilla se ha de transformar este valor en otro tipo de elemento, por ejemplo numérico, nos encontramos ante la necesidad de implementar bastante código solo para la validación de datos. Por todo esto, se plantea la creación de un sistema, que, en estos casos, facilite la gestión por parte de la plantilla atendiendo a las siguientes necesidades:

- Establecer restricciones de forma sencilla para los valores de diferentes elementos.
- Establecer conversiones de forma sencilla entre los valores del modelo y valores a utilizar en las plantillas, ya sean directos o a través de funciones que los calculen.
- Establecer la posibilidad de obtener valores por defecto cuando no se informa en el modelo.
- Una vez establecido lo anterior, posibilitar la reutilización en diferentes plantillas sin que sea necesario redefinir las propiedades.

De esta forma, se pretende reducir el tiempo de construcción de la LPS, así como mejorar la limpieza de la misma, excluyendo este tipo de lógica que no tiene relación con el producto final.

5.2.2.6. Mejora de la información sobre errores

Durante las pruebas de SPL Studio, se ha detectado que la información que el microservicio compilador proporciona sobre los errores que se han producido en ocasiones es escasa. Si bien, indica el tipo de error que se ha producido y el fichero donde se encuentra, no muestra la línea de código correspondiente, lo que en ocasiones lleva a confusión e impacta directamente en el tiempo de construcción. Completar esta información puede ayudar a reducir este tiempo de forma considerable.

Capítulo 6: Referencias

1. Weiss, D. M. and Lai, C. T. R. (1999) Software product-line engineering: A family-based software development process. Boston, MA: Addison Wesley.

2. Ingeno, J. (2018) Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts. Birmingham, England: Packt Publishing.

3. Baldonado, M., Chang, C.-C.K., Gravano, L., Paepcke, A. (1997) The Stanford Digital Library Metadata Architecture. Int. J. Digit. Libr. 1 108–121

4. Montilva C., J. A., Arapé, N. and Colmenares, y. J. A. (Sin fecha) Desarrollo de Software Basado en Componentes, Juancol.me. Available at: http://juancol.me/rsrc/sw-basado-en-comp-CAC2003.pdf (Acceso: Agosto 5, 2021).

5. Director, L. N. and Jones, L. (Sin fecha) Software Product Line Adoption, Dtic.mil. Available at: https://apps.dtic.mil/sti/pdfs/ADA634119.pdf (Acceso: Agosto 5, 2021).

6. Gomaa, H. (2004) Designing software product lines with UML: From use cases to pattern-based software architectures. Boston, MA: Addison Wesley.

7. Fowler, M. and Parsons, R. (2010) Domain-Specific Languages. Boston, MA: Addison-Wesley Educational.

8. Herrington, J. (2003) Code Generation in Action. New York, NY: Manning Publications.

9. Ferguson, R. (2018) Decisions for Sustaining a Software Product Line [Blog]. Recuperado de http://insights.sei.cmu.edu/blog/decisions-for-sustaining-a-software-product-line/

10. Polo Usaola, M. (2013) Desarrollo de software basado en reutilización: UOC

Anexo 1: Anexo 1. Acrónimos

LPS: Línea de productos de software.

PP: Punto de pago.

IDE: Entorno de desarrollo integrado.

DSL: Lenguaje especifico de dominio (Domain specific language).

EDP: Ingeniería de dominio / Espacio del problema.

EDS: Ingeniería de dominio / Espacio de la solución.

EAP: Ingeniería de aplicación / Espacio del problema.

EAS: Ingeniería de aplicación / Espacio de la solución.

Anexo 2: Código e instalación

Repositorios de código

Cada componente de la aplicación SPL Studio cuenta con su propio repositorio. Es posible acceder a cada repositorio a través de los siguientes links:

Servicio Eureka:

https://JarracedoHz@bitbucket.org/lobosolitario/spl-servicio-eureka.git

Servicio getway:

https://JarracedoHz@bitbucket.org/lobosolitario/spl-servicio-getway.git

Servicio Proyectos:

https://JarracedoHz@bitbucket.org/lobosolitario/spl-servicio-proyectos.git

Servicio elementos:

https://JarracedoHz@bitbucket.org/lobosolitario/spl-servicio-elementos.git

Servicio compilador:

https://JarracedoHz@bitbucket.org/lobosolitario/spl-servicio-compilador.git

Aplicación web SPL Studio:

https://JarracedoHz@bitbucket.org/lobosolitario/spl-splstudio-react.git

Instalación

SPL Studio se puede probar mediante un contenedor de docker. Los requisitos necesarios para ellos son:

- Tener instalado el software de virtualización Docker. Disponible en https://www.docker.com/.
- Disponer de una máquina con arquitectura amr64.
- Tener acceso a internet para que Docker pueda descargar la imagen de forma remota.

Para crear un contendor con la aplicación SPL Studio, solo es necesario ejecutar el siguiente comando:

```
docker run -p 3100:3100 -p 3101:3101 -p 8761:8761 -p 8095:8095 bovedano/splstudioserver:v1
```

Una vez ejecutado, comienza el proceso de descarga y extracción, por lo que la creación del primer contenedor se demora unos minutos.

El contendor creado, puede ser gestionado como cualquier otro contenedor de Docker (Documentación Docker: https://docs.docker.com/engine/reference/run/)

El proceso de arranque del contenedor es aproximadamente de un minuto, una vez terminado, se tendrá acceso a la aplicación a través de la url: http://localhost:3100/.

Una vez dentro de la aplicación, aparecerá la pantalla que se muestra en la figura **Fig. 96**. No es necesario insertar ningún usuario para probar la aplicación, simplemente bastará con pulsar en el botón "Iniciar sesión".

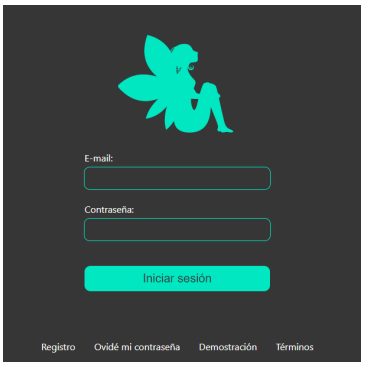


Fig. 96 Pantalla inicial de SPL Studio

Todos los datos que el usuario cree dentro de la aplicación son almacenados en el contenedor de Docker, por lo tanto, si se elimina este contenedor, se borrarán los datos de la aplicación.