UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Plataforma Web para modelar comportamiento server-side en augmentaciones
Autor: Franco Omar Mahl
Director: Matías Urbieta
Carrera: Licenciatura en Sistemas

Resumen

La aumentación web es un conjunto de técnicas que permiten a los usuarios definir y ejecutar software que depende de la capa de presentación de una página web concreta. De esta manera, a través del uso de artefactos de aumentación web, los usuarios finales pueden satisfacer varios tipos de requisitos que no fueron considerados por los analistas, desarrolladores y otras partes interesadas que construyeron la aplicación.
Aunque hay algunos enfoques de aumentación que contemplan una contraparte de servidor (para soportar aspectos tales como colaboración, gestión de sesión de explorador cruzado, etc.), los artefactos de aumentación suelen ser puramente del lado del cliente.
Este soporte del lado del servidor mejora las capacidades de las aumentaciones, ya que puede permitir compartir información entre usuarios e incluso entre las mismas aplicaciones.
Hasta ahora, este apoyo se define a menudo y se desarrolla de una manera ad-hoc.
Aunque está claro que el soporte del servidor aporta nuevas posibilidades, también es cierto que el desarrollo y despliegue de aplicaciones web del lado del servidor es una tarea compleja que los usuarios finales difícilmente pueden manejar.
Este trabajo presenta una herramienta CASE Web en fácil aprendizaje y uso, reemplazando las actuales herramientas desktop que se utilizan en estos casos para desarrollar el comportamiento del lado del servidor mediante el modelado conceptual y navegacional, brindando los elementos para el desarrollo de la interfaz de usuario y la persistencia de los modelos.

Palabras Claves

Aumentación web.
Desarrollo de aumentaciones web
Comportamiento server side de aumentaciones.
WebRatio web.
Diagramado de aumentaciones.
Aumentaciones web basada en modelos.
MDWE.
Standard IFML.
Modelado de Aumentaciones web server side.
SSWMFA.

Conclusiones

Este trabajo se basa en el enfoque para diseñar aumentaciones web basada en modelos (MDWE) con comportamientos del lado del cliente y del lado del servidor.
Para implementar el comportamiendo del lado del servidor se presenta en este trabajo la herramienta Server Side Web Modeling for Augmentations (SSWMFA)
Actualmente los aumentos se modelan utilizando IFML o con otros enfoques como UWE u OOHDM, pero esto puede traer muchos problemas como se explica en la Motivación de este trabajo.
El enfoque MDWE se basa en la separación de los principios de los conceptos.
Por lo tanto, se proporciona el mecanismo de composición para cada modelo (conceptual, navegacional y la interfaz de usuario), las formas de implementarlos del lado del servidor y distintas formas de desarrollar la interfaz con el lado del cliente.

Trabajos Realizados

Se desarrolló completamente la herramienta SSWMFA (Server-Side Modeling for Web Augmentations) siguiendo el enfoque de las aumentaciones web basadas en modelos (MDWE) siguiendo las bases del estándar IFML. La herramienta permite desarrollar comportamiento server-side de aumentaciones web y provee los medios para comunicarse con herramientas client-side para ser consumida como un plug-in. Esto permite reemplazar las herramientas desktop utilizadas hasta el momento. Se realizaron pruebas suiguiendo casos de uso reales simulando el comportamiento client-side.

Trabajos Futuros

Implementar en un caso real integrando con una herramienta client side.
Realizar pruebas con usuarios finales.
Mejoras en la herramienta.
Implementar mensajes claros en la interfaz gráfica que comuniquen al usuario claramente qué está sucediendo en el backend.
Manejo de errores: mejorar las respuestas especificando claramente qué acaba de suceder al realizar el request.
Relaciones y herencia en el diagrama Entidad/Relación: Implementar la funcionalidad de relación entre entidades. Agregar herencias entre Clases.
Persistir los diagramas en una base de datos.
Permitir que pueda haber múltiples formularios/listados de una misma clase dentro de una misma página.
Agregar más funcionalidades provistas por GoJS.

Fecha de la presentación: MARZO 2019

Índice

Índice de figuras

Resumen

La aumentación web es un conjunto de técnicas que permiten a los usuarios definir y ejecutar software que depende de la capa de presentación de una página web concreta. De esta manera, a través del uso de artefactos de aumentación web, los usuarios finales pueden satisfacer varios tipos de requisitos que no fueron considerados por los analistas, desarrolladores y otras partes interesadas que construyeron la aplicación.

Aunque hay algunos enfoques de aumentación que contemplan una contraparte de servidor (para soportar aspectos tales como colaboración, gestión de sesión de explorador cruzado, etc.), los artefactos de aumentación suelen ser puramente del lado del cliente.

Este soporte del lado del servidor mejora las capacidades de las aumentaciones, ya que puede permitir compartir información entre usuarios e incluso entre las mismas aplicaciones.

Hasta ahora, este apoyo se define a menudo y se desarrolla de una manera ad-hoc.

Aunque está claro que el soporte del servidor aporta nuevas posibilidades, también es cierto que el desarrollo y despliegue de aplicaciones web del lado del servidor es una tarea compleja que los usuarios finales difícilmente pueden manejar.

Este trabajo presenta una herramienta CASE [23] Web para desarrollar este comportamiento del lado del servidor mediante el modelado conceptual y navegacional, y brindando los elementos para el desarrollo de la interfaz de usuario.

1. Motivación

Las aplicaciones web son diseñadas y desarrolladas considerando, generalmente, un conjunto muy acotado de partes interesadas (managers, usuarios internos, propietarios del producto, desarrolladores y clientes) sin tener en cuenta el feedback de los usuarios finales y sus verdaderas necesidades.

Si bien hoy en día se cuenta con herramientas y mecanismos de adaptación para personalizar las aplicaciones de terceros, sus usuarios pueden seguir teniendo requerimientos no satisfechos, ya que no es realista que este grupo reducido de personas pueda cubrir todas y cada una de las necesidades de los usuarios.

Un mecanismo popular de hoy en día es modificar los sitios web una vez cargados en el navegador (client-side). Cualquier sitio web puede ser mejorado a través de la manipulación de su DOM (Document Object Model). Es posible agregar, eliminar o editar contenido, estilos y funcionalidades. Esta técnica es parte de lo que se conoce como Web Augmentation o Aumentación Web [8].

El enfoque de aumentación web basada en modelos (MDWE) [29] contempla una aplicación cliente-servidor que abstrae al usuario de la complejidad del backend, facilitando la generación de aumentaciones web complejas basadas en mejoras client-side pero soportadas por artefactos server-side.

Los componentes que provee este modelado de aumentación web son:

1. Enfoque basado en modelos que soporta la producción de aumentaciones que se beneficia de la combinación de artefactos en los dos lados, del cliente y del servidor.

2. Un soporte completo para facilitar la introducción de este tipo de aumentaciones

3. Un proceso de desarrollo híbrido web-desktop

El tercer punto generó dificultades al tener que utilizar una herramienta desktop como lo es WebRatio [35].

En una experiencia de desarrollo, se comenzó a tener problemas desde el momento de la instalación de la aplicación ya que corre sobre una determinada versión de Java [34] que no es compatible con la última versión disponible.

Otra observación realizada sobre WebRatio es que es una aplicación de propósito muy amplio que excede en gran medida el propósito de uso que se requiere para desarrollar artefactos de aumentaciones web server-side. Esto deriva en que se invierta mucho tiempo de investigación y aprendizaje para utilizar solo una porción pequeña de las herramientas que ésta brinda.

En este trabajo se presenta una herramienta CASE [23] Web para modelar comportamiento server-side para aumentaciones web basadas en modelos (MDWE) [29] proveyendo solo la funcionalidad necesaria e intentando que sea de rápido aprendizaje y fácil manejo para usuarios sin un profundo conocimiento en programación.

El enfoque de herramienta de diagramado web (que corre sobre un navegador) ayuda a que el usuario se abstraiga del sistema operativo adyacente y no se encuentre con problemas a la hora de descargar - instalar - correr un programa desktop que puede variar según el sistema operativo, la versión y las distintas distribuciones que se ofrecen.

Acerca del código, se decidió que se realizará en lenguajes populares, robustos y con una buena documentación para facilitar la comprensión y mantenimiento del mismo que, a su vez, se liberará para que desarrolladores con conocimiento avanzado puedan contribuír con el desarrollo del core de la herramienta y personalizar las funcionalidades requeridas en cada caso. Por ejemplo, es posbile agregar fácilmente endpoints que ayuden a integrar el comportamiento server-side como un complemento (plug-in)[1] de la herramienta client-side. A esto se agrega que el código de SSWMFA corre en un container de Docker [21], lo que facilita a los desarrolladores la tarea de descargar

[1] https://es.wikipedia.org/wiki/Complemento_(informática)

el código y ejecutarlo localmente ya que no necesitan realizar ninguna instalación aprovechando los beneficios de la abstracción que provee Docker.

Podemos decir que buscamos que SSWMFA les facilite la vida tanto a los usuarios (quienes consumirán la herramienta desde el lado del cliente) como a la comunidad de desarrolladores que se encarga de realizar comportamiento server-side de aumentaciones web.

2. Marco teórico y estado del arte

2.1 Aumentación de aplicaciones web

Aumentación web es un conjunto de técnicas utilizadas por un gran número de usuarios. Se pueden encontrar varias extensiones para adaptar el contenido de una web en tiendas de navegadores donde un número significativo de comunidades apoyan algunas de estas herramientas.

Los usuarios finales y otras partes interesadas con habilidades de programación pueden interactuar en dichas comunidades participando en la creación, intercambio y mejora de artefactos de aumentación específicos. Por ejemplo, la comunidad Userstyles (http://userstyles.org) ofrece un gran número de secuencias de comandos que aumentan los sitios web al agregar especificaciones CSS que cambian la presentación del contenido. Las comunidades de scripts de usuarios, como Greasyfork (https://greasyfork.org/), ofrecen repositorios de scripts con un espectro más amplio de propósitos, ya que admiten diferentes tejedores de código JavaScript (por ejemplo, GreaseMonkey o TamperMonkey). Por lo tanto, es posible modificar no sólo el estilo sino el contenido y el comportamiento de una página web.

En todas estas comunidades, independientemente de la herramienta que soporten, existe una dependencia entre usuarios con y sin habilidades de programación, ya que no todos ellos pueden implementar las soluciones que necesitan y deben pedir ayuda a otros usuarios. En este sentido, algunas investigaciones proponen enfoques de Desarrollo de Usuario Final (EUD) para permitir a los usuarios especificar sus propios artefactos de aumentación. Una de ellas es la aumentación basada en modelos (MDWA) [29] la cual es la base de este trabajo donde se detalla una herramienta CASE [23] Web para modelar el comportamiento server-side.

2.2 Ingeniería de enfoques orientados a modelos

En la mayoría de los enfoques de diseño web avanzados, como UWE, WebML, UWA, Hera, OOWS u OOHDM, una aplicación web está diseñada con un proceso iterativo que comprende por lo menos el modelado conceptual y de navegación.

De acuerdo con las técnicas de ingeniería de enfoques orientados a modelos (MDWE) [9] [1], estos métodos producen un modelo independiente de la implementación que puede ser posteriormente mapeado a diversas plataformas de ejecución.

Para que sea más fácil de comprender, me centraré en los modelos conceptuales, de navegación y de interfaz, ya que son bastante similares en distintos enfoques de diseño.

2.2.1 Diseño conceptual

El modelo conceptual de una aplicación web (conocido como dominio de aplicación o modelo de contenido) se centra en expresar los conceptos específicos de dominio involucrados en el problema que resuelve, con sus atributos, relaciones y comportamiento asociado. Cuando se define mediante algunas metodologías, como la metodología de diseño de hipermedia orientada a objetos (OOHDM) o UWE, este modelo es un modelo orientado a objetos descrito con UML y compuesto de clases con sus atributos, métodos y asociaciones.

2.2.2 Diseño navegacional

El diseño de navegación de una aplicación web tiene como objetivo definir vistas, estructuras de acceso y rutas de navegación hacia los contenidos para permitir al usuario acceder y navegar fácilmente. La mayoría de los métodos de ingeniería web basan su modelo de navegación en dos primitivas de modelado: Nodo y Link. El Interaction Flow Modeling Language (IFML) [2] no es una excepción a esto, ya que define páginas como vistas lógicas en clases de modelo de aplicación, y enlaces como la

realización hipermedia de asociaciones de modelos de aplicación que define capacidades de navegación o activación de comportamiento del sistema.

2.2.3 Diseño de interface abstracta

En OOHDM, la interfaz de usuario se especifica mediante vistas de datos abstractos (ADVs). Soporta un modelo orientado a objetos para objetos de interfaz. Para cada clase de nodo, se debe definir un ADV para indicar cómo cada uno de sus atributos o subnodos debe ser presentado al usuario. En este contexto, un ADV puede ser visto como un Observer [17] del nodo, y es posible expresar sus atributos y operaciones usando un diagrama de configuración [33].

Los ADV también se usan para especificar cómo se desarrollará la interacción como resultado de eventos generados por el usuario. Estos aspectos de comportamiento se especifican mediante ADV-charts [33], una especie de diagramas de estados que representan los estados y sus transiciones para un ADV dado. Son útiles cuando se necesita modelar comportamientos de interfaz enriquecidos como el de Aplicaciones de Internet Enriquecidas (RIA) [22]. ADV-Charts son diagramas de máquinas de estado que permiten expresar las transformaciones de la interfaz que ocurren como resultado de la interacción del usuario en un ADV determinado a través de las reglas de condición de evento.

2.3 Trabajos relacionados

2.3.1 Aumentación de aplicaciones web

El desarrollo del usuario final (EUD) fue explorado en el campo del aumento de la web para capacitar a los usuarios sin habilidades de programación en la especifición de artefactos de aumento.

En un trabajo anterior, se presentó una herramienta denominada WOA (Web Object Ambient) que permite a los usuarios finales recrear un modelo de objeto de

sitios web en el lado del cliente [12]. Sin embargo, puesto que el foco está en soportar complejas aplicaciones de aumentación que no podrían funcionar solo con componentes del lado del cliente, se desea que los usuarios con habilidades de modelado sean capaces de crear la contrapartida back-end de estas aumentaciones. Este soporte generalizado de back-end potenciará los artefactos de aumentación ya que podrían contemplarse otras características, tales como sincronización de datos, lógica de negocio compleja, almacenamiento o aspectos sociales.

Varios aspectos sobre el aumento se han abordado a través de actividades de modelado, como la especificación de requerimientos [10]. Estos aspectos suelen ser dirigidos a modelar la capa de presentación, dejando de lado la especificación de comportamiento que puede tomar ventaja de los componentes del servidor. Tales modelos no son suficientes para representar la lógica de back-end de una aplicación.

En este sentido, este trabajo propone la integración del enfoque para extraer un objeto modelo de las páginas web existentes [12] y una aplicación back-end creada utilizando los lenguajes de modelado web existentes. La aumentación web podría ser considerada como un mecanismo de aplicación externa, ya que puede adaptar los sitios web de terceros en el lado del cliente. Sin embargo, las adaptaciones del lado del cliente también podrían ser vistas como una preocupación central durante el diseño de la aplicación.

Un enfoque interesante bajo este punto de vista fue la utilización de dicho mecanismo en un sistema de e-learning [4]. Esto propone una capa de modelado para la adaptación al cliente, pero no se utiliza desde el punto de vista de la aumentación y su uso está restringido a los propietarios de las aplicaciones.

2.3.2 Separación de conceptos en MDWE

Varios enfoques de la ingeniería web orientada a modelos (MDWE) permiten componer sin problemas las preocupaciones de las aplicaciones web, tales como Funcionalidades Volátiles en Aplicaciones Web [32, 15], la separación de modelado de preocupaciones en Workflows [31], o la separación de las preocupaciones de Web-GIS en

aplicaciones web [30]. Otros enfoques soportan requisitos evolutivos, como el Modelo de Proceso de Composición Web [16], el Distributed Concern Delivery [5] o el apoyo a principios más generales, como refactoring [14] y patrones de diseño [17]. Sin embargo, todos estos enfoques se centran en la composición de las preocupaciones en el lado del servidor y sólo cuando el desarrollador tiene acceso al núcleo de la aplicación.

Este escenario no es el objetivo de la aumentación web.

2.4 Enfoque de aumentación web basado en modelos (MDWA)

En esta sección, se presenta una breve caracterización de los requerimientos de aumentación que necesitan soporte de cliente y servidor. Además se presenta el enfoque donde cada uno de sus pasos se explora en detalle.

2.4.1 Justificación sobre el uso del enfoque

La práctica más común es agregar esta capa de aumentación a las aplicaciones web existentes en el lado del cliente. Como ya se ha mencionado en este documento, esto se logra manipulando el DOM de las páginas web una vez que se cargan en el navegador. Esto se logra utilizando artefactos de software independientes que los usuarios finales corren en el navegador web de su dispositivo (de escritorio o móvil).

El soporte del lado del servidor para la aumentación de la web proviene de la necesidad de adaptar los sitios web existentes más allá de la simple manipulación del DOM. Hay varios aspectos que respaldan esta idea:

- **Alcance de ejecución**: los artefactos típicos de aumentación web [8] se ejecutan en el navegador una vez que se carga una página web en particular. Esto limita el contexto de ejecución dado que el artefacto de aumentación, denominado aumentador, solo se ejecuta cuando se carga la página web.

Es posible que algunas aumentaciones requieran continuar trabajando incluso si el sitio web de destino no se está utilizando porque, por ejemplo, se basa en la información que se rastrea desde otras fuentes o sitios web.

Por lo general, existe la posibilidad de ejecutar un artefacto de aumentación en cada página web cargada (incluso si ésta página web no está aumentada) y que el aumentador continúe trabajando [36]. Sin embargo, este escenario todavía depende del hecho de que el navegador web esté abierto. En los casos en que se requiera, por ejemplo, procesar flujos de información para realizar la capa de aumentación, se requerirá un enfoque del lado del servidor para hacerlo.

- **Integración entre dispositivos**: en una estrategia del lado del cliente, los aumentadores se ejecutan en un único dispositivo, independientemente de otras ejecuciones/instalaciones del mismo aumentador en otros navegadores web.

  Si un usuario requiere algún tipo de integración/interacción entre sus dispositivos, entonces se necesitaría un back-end del lado del servidor. Por ejemplo, este es el caso de aumentadores para interacciones de interfaz de usuario distribuídas [11].

- **Colaboración**: hay varias herramientas de colaboración (como Evernote, Diigo, etc.) que proponen capas de aumentación basadas en patrones de colaboración en las que los usuarios finales pueden anotar sitios web, agregar comentarios, etc.

  Este tipo de requerimientos de aumentación web obviamente necesitan una base de datos común donde se almacenarán todas las contribuciones de los usuarios.

  Tener en cuenta a otros usuarios para las mismas capas de aumentación también permite la posibilidad de ir más allá de la adaptación de UI, ya que se pueden lograr otros aspectos relacionados con la personalización, como el filtro colaborativo [18].

- **Cómputo elevado y grandes requisitos de almacenamiento**: si la aumentación apunta a satisfacer requisitos que exigen una gran carga de trabajo (es decir, procesamiento de imágenes o aprendizaje profundo), el navegador web

puede no ser la mejor opción para ejecutar el aumentador y, debido a que los recursos son limitados, la experiencia de la navegación web puede verse afectada negativamente. En casos como este, los artefactos que requieren recursos de alto rendimiento pueden implementarse en el lado del servidor y luego su resultado se entreteje en el lado del cliente.

Es importante tener en cuenta que estas cuatro dimensiones no son ejemplos de aumentadores, sino que representan diferentes preocupaciones que pueden estar involucradas en varios tipos de requisitos de aumentación.

2.4.2 Descripción del enfoque

El enfoque se basa en la idea de que incluso el comportamiento más simple (por ejemplo, una nueva funcionalidad para insertar comentarios) debe considerarse como una funcionalidad de primera clase y, como tal, diseñada en consecuencia. Su diseño e implementación tienen que desarrollarse separadas de la aplicación principal, y disociadas lo más posible de las funcionalidades básicas y estables de la aplicación principal. Las aumentaciones no pueden ser introducidas al código fuente de la aplicación porque el analista aumentador no forma parte del equipo de desarrollo.

Basándonos en estas ideas, el enfoque se puede resumir con las directrices de diseño que se muestran en la Figura 2.1:

1. Se desacopla la aumentación de la aplicación principal mediante la introducción de una capa de diseño (denominada capa de aumento), que comprende un modelo conceptual, un modelo de navegación y un modelo de interfaz.

2. Se captura el modelo conceptual básico clasificando los datos en las páginas de la aplicación principal usando un mecanismo de abstracción y estructuración [12]. La falta de acceso a los modelos subyacentes de la aplicación principal requiere la extracción perceptible del modelo conceptual. En este proceso, los elementos de datos en la página son clasificados y agrupados en una definición de entidad por

un analista de aumento de tal manera que se obtiene un modelo conceptual simplificado. El analista de aumento es un usuario final con conocimientos avanzados de aplicación web que tiene como objetivo mejorar la aplicación principal. En otros pasos, la instanciación del modelo en una sesión de usuario particular se utilizará para proporcionar información contextual al motor de aumento proporcionando información de instancias de modelo al activar el aumento.

3. Los requisitos de aumento se modelan utilizando las anotaciones de ingeniería web (por ejemplo, casos de uso, diagramas de interacción del usuario, etc.) y se mapean por separado en los siguientes modelos usando las heurísticas definidas por el enfoque de diseño (véase por ejemplo [24]). Se puede observar que, como se muestra en la Figura 2.2, los requisitos de aumento no están integrados en el modelo de requerimientos principales, por lo tanto, se deja su integración a otras actividades de diseño.

   (a) Los nuevos comportamientos, es decir, aquellos que pertenecen a la capa de aumento, se modelan como objetos de primera clase en el modelo conceptual de aumento. Define todos los objetos y comportamientos correspondientes a los nuevos requisitos. Además, este modelo puede incluir clases conceptuales de aplicaciones centrales (capturadas en el Paso 2) percibibles por el usuario final que permiten definir relaciones entre el modelo de negocio de aumento y la aplicación principal. Observe que esta estrategia puede aplicarse a cualquier método orientado a objetos, es decir, a cualquier método que utilice un enfoque de especificación similar a UML.

   (b) Los nodos y los enlaces pertenecientes al modelo de navegación de aumento pueden o no tener vínculos con el modelo de navegación principal. El modelo de navegación central también es ajeno a las clases de navegación de aumento, es decir, no hay enlaces u otras referencias desde el núcleo a la capa de aumento. Este principio se puede aplicar en cualquier enfoque de diseño web.

(c) Se utiliza una especificación de integración independiente para especificar la conexión entre nodos de núcleo y de aumento.

(d) Se diseñan (e implementan) las interfaces correspondientes a cada concern (principal y aumento) por separado. El diseño de la interfaz de las clases principales (descrito en OOHDM utilizando Vistas Abstractas de Datos (ADV) [33]) están desconectadas de la interfaz de concerns de aumento. Como en la capa de navegación, este principio es independiente del enfoque de diseño.

4. Las interfaces principales y de aumento (en las capas ADV y de implementación) se tejen ejecutando una especificación de integración, que se realiza mediante transformaciones DOM. Una vez más, la idea de tejido del modelo es genérica y por lo tanto el mismo resultado se puede obtener utilizando otra solución técnica.

Una vez que los requisitos de aumento son modelados en el Paso 3, el aumento de otro sitio será bastante sencillo, requiriendo encontrar y mapear las clases virtuales (Paso 2) y especificar cómo activar los artefactos de UI de aumentación (Paso 4).

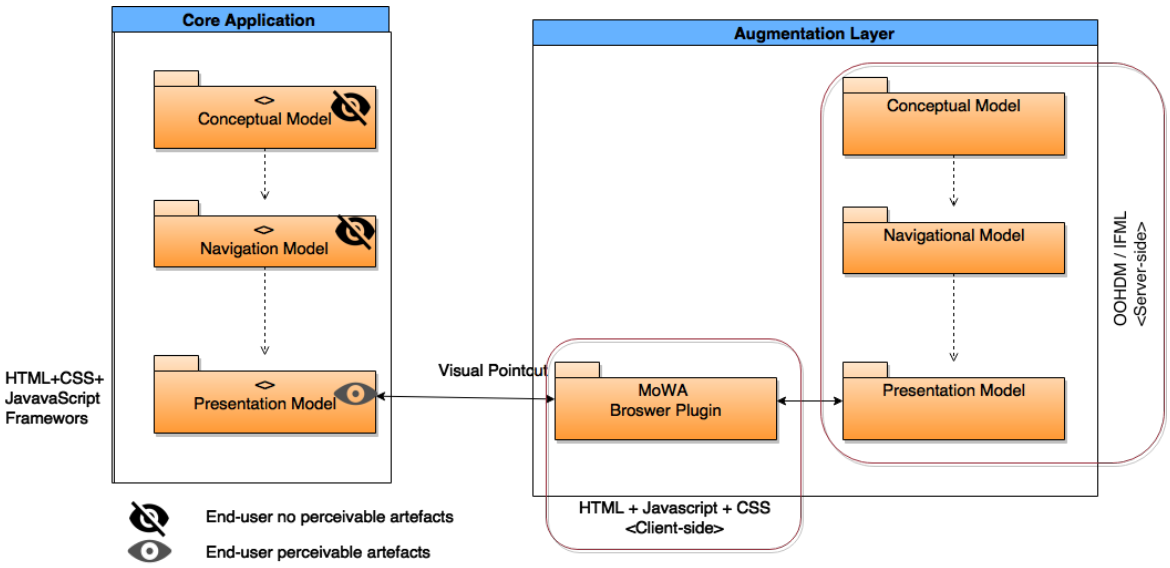


Fig. 2.1 Esquema del enfoque

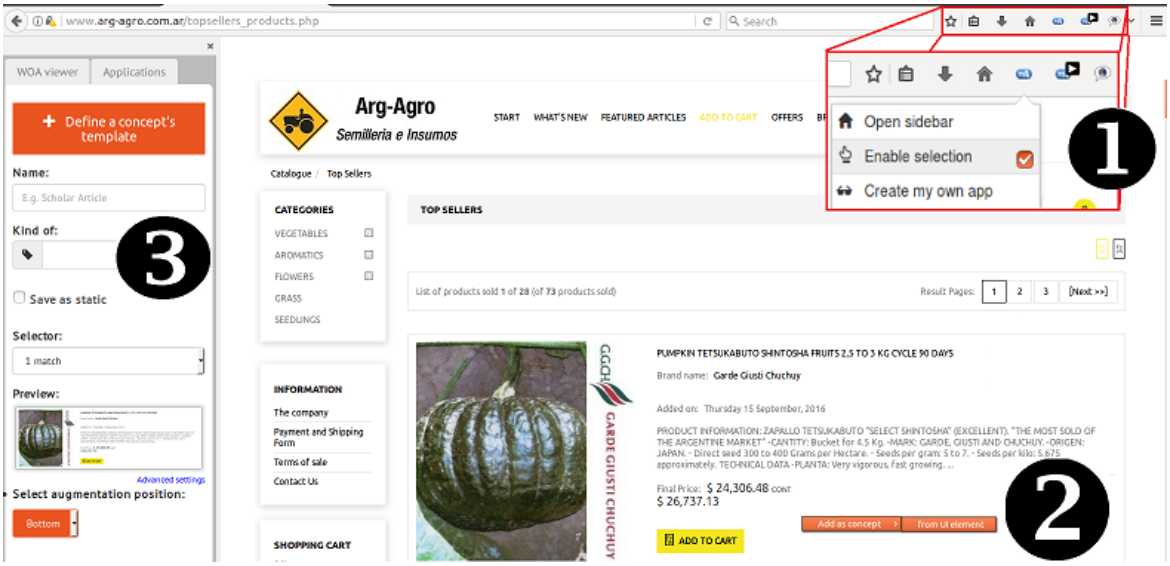


Fig. 2.2 Definición del concepto

3. La herramienta propuesta (SSWMFA)

En este capítulo se decribe la herramienta propuesta, SSWMFA (Server-Side Web Modeling for Augmentations), cuyo objetivo es facilitar el desarrollo de artefactos server-side para aumentaciones web.

Como se mencionó anteriormente, el enfoque de aumentación web basada en modelos (MDWE) contempla una aplicación cliente-servidor que abstrae al usuario de la complejidad del backend, facilitando la generación de aumentaciones web complejas basadas en mejoras client-side soportadas por artefactos server-side. Actualmente estos artefactos server-side se desarrollan en una herramienta desktop como lo es WebRatio [35].

La herramienta Server-Side Web Modelling for Augmentations (SSWMFA) presenta un enfoque de desarrollo web utilizando un navegador.

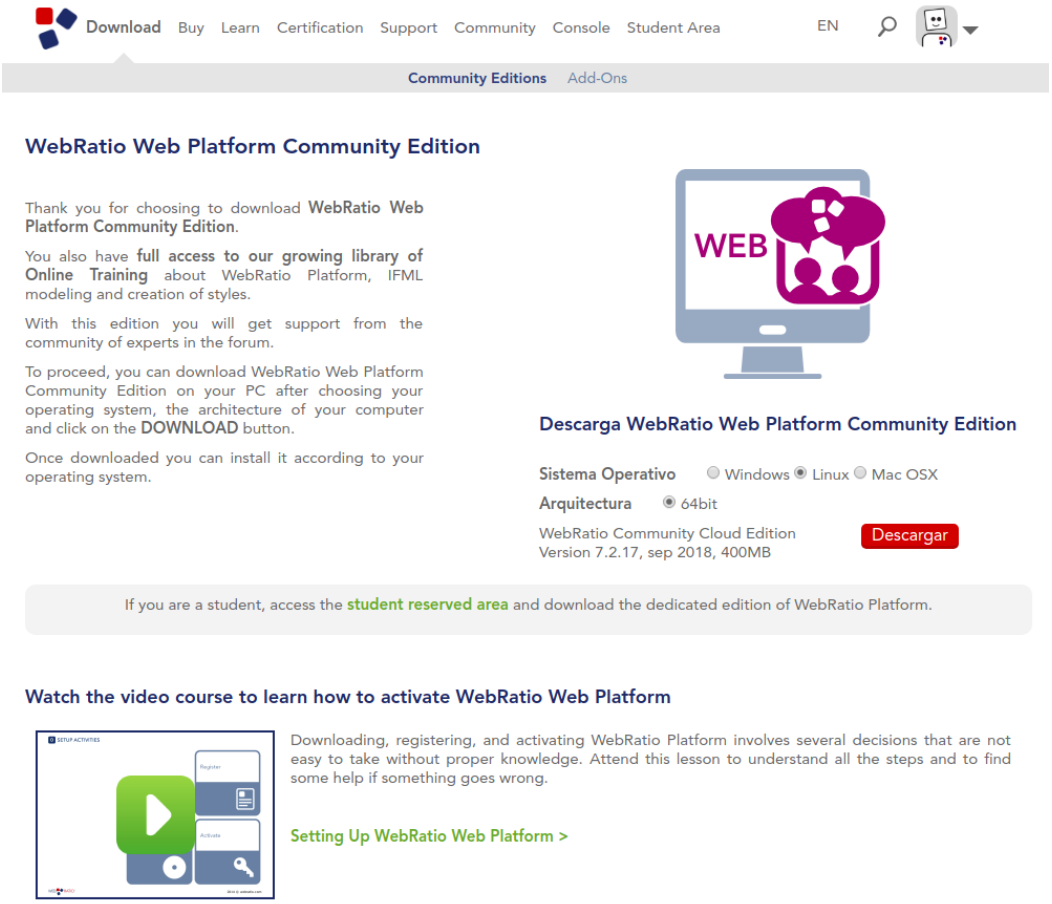
3.1 Problema a resolver

La necesidad de la creación de esta herramienta con propósito específio para desarrollar artefactos de aumentación web del lado del servidor nace de que WebRatio prensenta las desventajas detalladas a continuación.

Su instalación puede ser muy engorrosa.

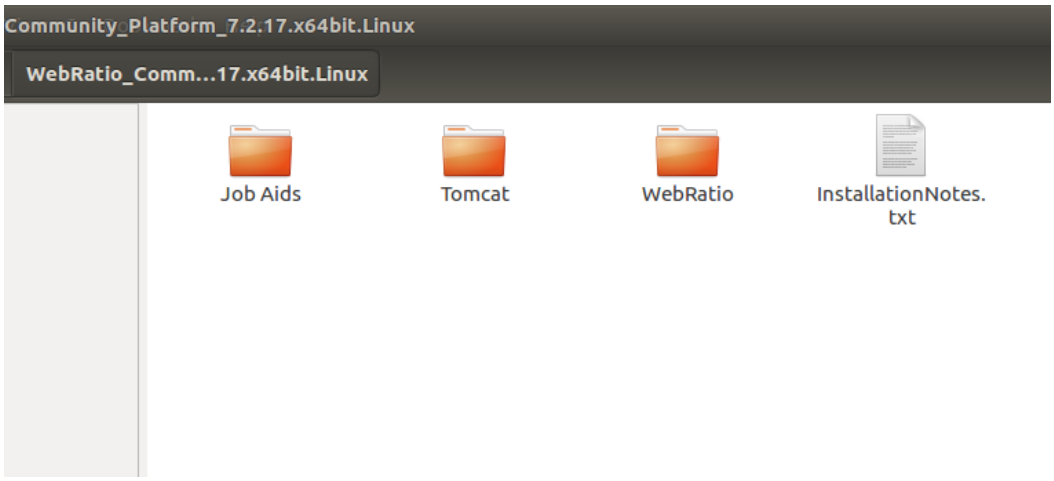
El primer paso es descargar la versión **WebRatio Web Platform Community Edition** desde el sitio oficial [1] para lo cual hay que registrarse creando una cuenta en el sitio.

[1] https://my.webratio.com/



Una vez descargado el paquete y extraído su contenido dentro de nuestro file system podemos comenzar con la instalación de la herramienta.

Dentro de la carpeta de WebRatio se encuentra un archivo de texto con los pasos para la instalación `InstallationNotes.txt`.



Aquí podemos observar la complejidad de los pasos a seguir, por ejemplo, al tener que instalar alguna de las versiones de Java Development Kit 1.5, 1.6 o 1.7 la cual dejó de actualizarse en abril del 2015[2] y, por ejemplo, ya no se encuentra disponible para versiones de Ubuntu mayores a 16.10. La versión de JDK actual es la 1.9, por lo que podemos ver que funciona con versiones antiguas.

El contenido del archivo con los pasos es el siguiente.

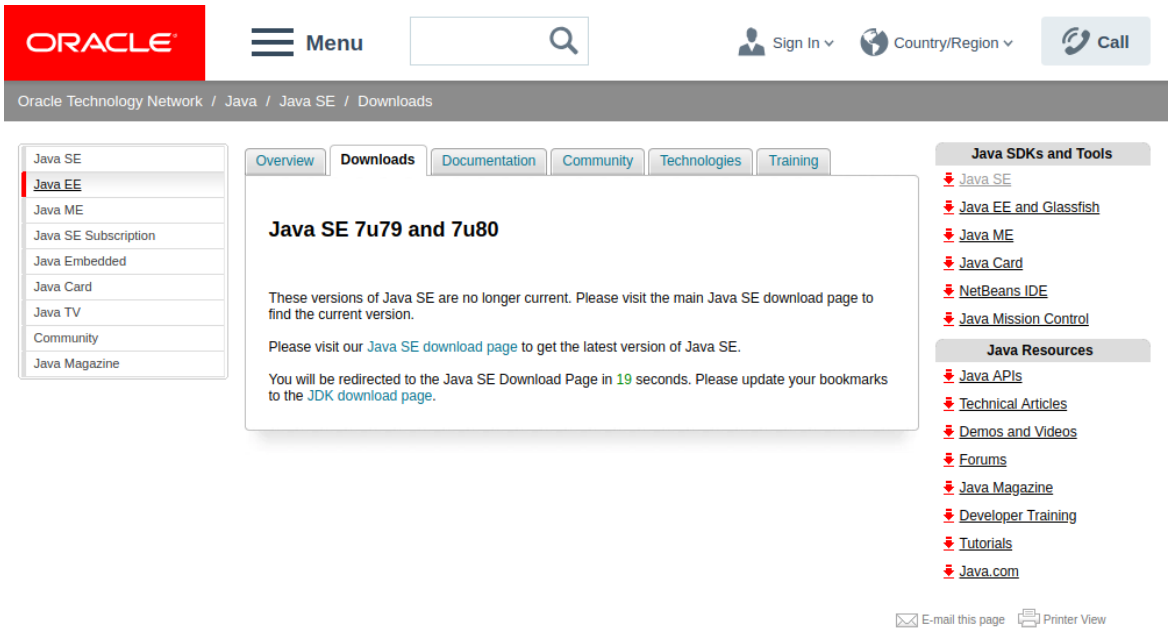
```
If you run WebRatio and you got the message saying that the
   Java version used is not suitable, you need to configure
   another JDK

Just follow these steps:

1. Check the current installed Java version. Open a shell and
   execute the command

   Java -version

   You get something like
```

[2]https://en.wikipedia.org/wiki/Java_version_history

```
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed
    mode)

Then execute the command

echo $JAVA_HOME

If the result is 1.5, 1.6 or 1.7 then you can execute
    WebRatio.
If you got 1.8 you have to install one of the
    abovementioned JDK.

2. Download from the Oracle website the JDK 1.7 for your
operating system and architecture (http://www.oracle.com/
technetwork/java/javase/downloads/jre7-downloads-1880261.
html).

3. Install the downloaded JDK.
The tar.gz provided by Oracle don't have an actual
    installation process.
You just extract those files to a location you want and add
     them to your path. So the process is the following:

- Extract the .tar.gz file to somewhere;
- Move the extracted folder to /usr/lib/jvm. This is not
    required but it is the place where Java runtime
```

```
    sudo mv /path/to/jdk1.7.0_79 /usr/lib/jvm/oracle_jdk7

  - Create a file /etc/profile.d/oraclejdk.sh with the
    following content (adapt the paths to reflect the path
    software is installed):

    export J2SDKDIR=/usr/lib/jvm/oracle_jdk7
    export J2REDIR=/usr/lib/jvm/oracle_jdk7/jre
    export PATH=$PATH:/usr/lib/jvm/oracle_jdk7/bin:/usr/lib/
      jvm/oracle_jdk7/db/bin:/usr/lib/jvm/oracle_jdk7/jre/
      bin
    export JAVA_HOME=/usr/lib/jvm/oracle_jdk7
    export DERBY_HOME=/usr/lib/jvm/oracle_jdk7/db

    Those paths will only be recognized after you logout or
      restart, so if you want to use them right away run
      source /etc/profile.d/oraclejdk.sh.

  NOTE. You may have more than one JDK running on your
    computer. You don't need to uninstall them.

4. Run WebRatio.

If you don't succeed with the previous procedure you can
  always extract the .tar.gz file in the same folder of
  WebRatio and start it.
This is not the suggested procedure, please use it when the
  suggested one does not work for you.
```

Al comenzar con el primer y segundo paso ya nos encontramos con algo inesperado, la url que nos brinda (http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html) nos lleva a un sitio de descargas de Java en donde se detalla que la versión 1.7 ya no se encuentra disponible.

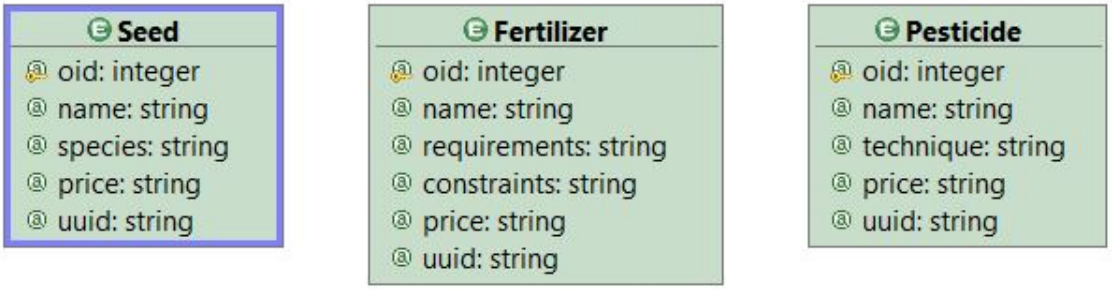


Buscando en google se puede encontrar una página de archivo de Java donde es posible descargar la versión JDK7 (1.7) https://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html.
Luego de completados los pasos es posible correr la herramienta.
Debido a la complejidad de la instalación de WebRatio es que se pensó **SSWMFA** como una herramienta web que corre sobre un navegador, lo que no requiere descargas, instalación ni un sistema operativo específico para trabajar.
Es una herramienta de propósito mucho más amplio de lo necesario para desarrollar artefactos de aumentación web.
Esto hace que su uso requiera de muchas horas de aprendizaje para luego utilizar un conjunto muy acotado de sus funcionalidades.

Crear una aplicación diseñando en WebRatio es muy complicado y lleva mucho tiempo ya que su documentación no es muy completa ni amigable para los nuevos

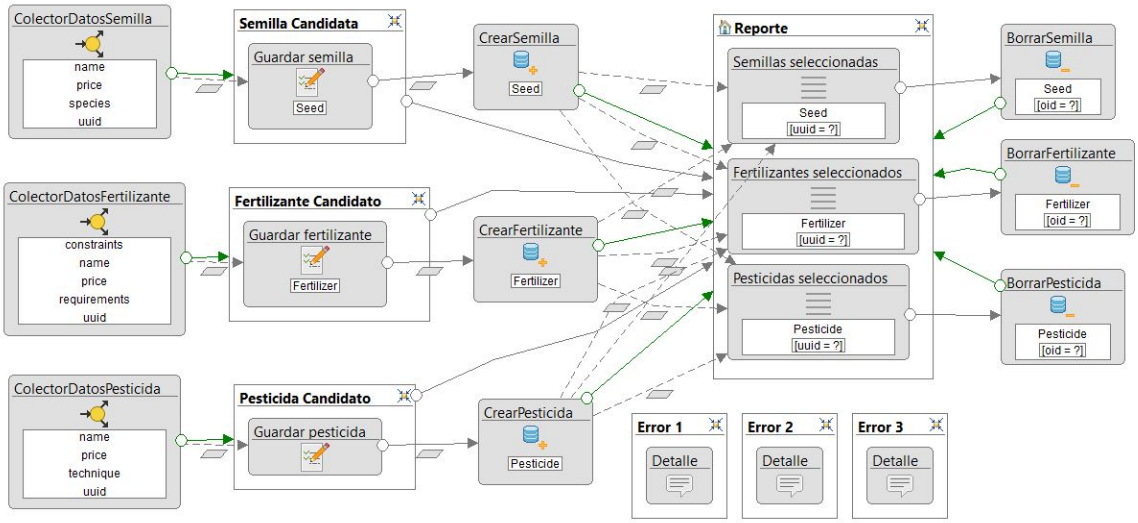
desarrolladores. Por ejemplo, una herramienta de aumentación sobre un sitio de agricultura tiene que desarrollar lo siguiente.

Modelo conceptual: Desarrollar el diagrama de Entidad/Relación de la Base de Datos adyacente:



Modelo navegacional: Para realizar las acciones necesarias de colectar la información del sitio, persisitir los datos, editar, eliminar y listar la información almacenada en la base de datos se deben crear muchas páginas interconectadas de forma correcta (ya que hay múltiples formas de conectar páginas) y completar con todos los datos requeridos para cada página. Las opciones son muchas lo que puede resultar abrumador y confuso para usuarios inexpertos.

Luego de varias horas de buscar en internet, consultar ejemplos e investigar la herramienta se puede obtener un modelo navegacional como el siguiente:



Podemos observar la gran cantidad de páginas que requiere realizar un simple manejo de ABLM (Alta, Baja, Listado y Modificación) de elementos en la base de datos, por lo que uno de los objetivos de **SSWMFA** es brindar la posibilidad de, si se es deseado, realizar todas estas acciones en una sola página de forma sencilla y en un nivel de abstracción lo suficientemente alto como para no necesitar diagramar cada paso, como por ejemplo podemos ver el paso con el ícono de Base de Datos que especifica persistencia de un elemento; o el paso de error por cada uno de los tipos de elementos. Estos pasos deberían estar implicitos en el manejo interno de la herramienta.

Es de código propietario (cerrado) lo que no permite experimentar con el core de la aplicación ni intentar crear o modificar funcionalidades.

SSWMFA será de código abierto permitiendo a la comunidad de desarrolladores seguir expandiendo la herramienta, o incluso, desarrollar sus propias distribuciones.

Otra ventaja de que sea de código libre es que es posible personalizar la herramienta para cada caso, permitiendo crear endpoints para consumir desde el frontend hacia el backend funcionalidad específica para el caso a aumentar.

En resumen, lo que se busca es facilitar y agilizar el desarrollo del comportamiento Server-Side para aumentaciones web, brindando una alternativa a WebRatio que no requiera descarga ni instalación, ni muchas horas de aprendizaje, abstrayendo al desarrollador del comportamiento de bajo nivel (amigable para desarrolladores inexpertos) y creando los modelos conceptual y navegacional de forma intuitiva y user friendly, pero a la vez brindando la posibilidad de modificar el core de la herramienta para personalizar y potenciar las funcionalidades (código abierto para desarrolladores expertos). Adicionalmente se busca que se pueda incluir como un plug-in a herramientas client-side de aumentación web mediante el llamado a endpoints que crea la herramienta dinámicamente para cada caso.

3.2 Objetivo de la herramienta

El objetivo de este trabajo es realizar una herramienta de diagramado web liviana para modelar el comportamiento server-side de aumentaciones web siguiendo lo detallado por la ingeniería de enfoques orientados a modelos (MDWE).

Para esto, la herramienta proveerá las siguientes funcionalidades:

- Diagramado del modelo conceptual: Desarrollo del diagrama de Entidades y Relaciones (backend).
- Creación de la Base de Datos y tablas derivadas del modelo conceptual de Entidad-Relación donde las entidades se mapean a tablas y las relaciones a claves foráneas (en los casos de relaciones *uno a uno* o *uno a muchos*) o tablas intermedias (en los casos de relaciones *muchos a muchos*) .
- Diagramado del modelo de navegación utilizando como base IFML y el modelo Nodo/Link donde las páginas serán los nodos y los enlances o hipervínculos entre ellas, los links (frontend).
- Renderizar el modelo de navegación creando las páginas con los elementos contenidos en ellas: Listas, Forms y Scripts que se describirán luego en este artículo. Los links entre las páginas se renderizarán como hipervínculos entre ellas.
- Posibilidad de conexión a la herramienta mediante plugins realizando llamados del tipo HTTP Request a la API REST [25]. La herramienta contará con algunos endpoints creados dinámicamente en base al modelo de navegación, pero además tendrá la posibilidad de que desarrolladores con experiencia básica en el lenguaje JavaScript y en desarrollos de API REST puedan crear sus propios endpoints personalizados y así enviar y consumir información del lado del servidor desde el lado del cliente.

Como se describe en los puntos anteriores, una vez obtenidos los diagramas se permitirá crear la base de datos derivada del modelo conceptual, y crear y renderizar las páginas modeladas con sus links y funcionalidades detalladas para cada una.

Para esto se divide la herramienta en dos secciones: modelado conceptual y modelado navegacional o de navegación como se muestra en la Figura 3.1.

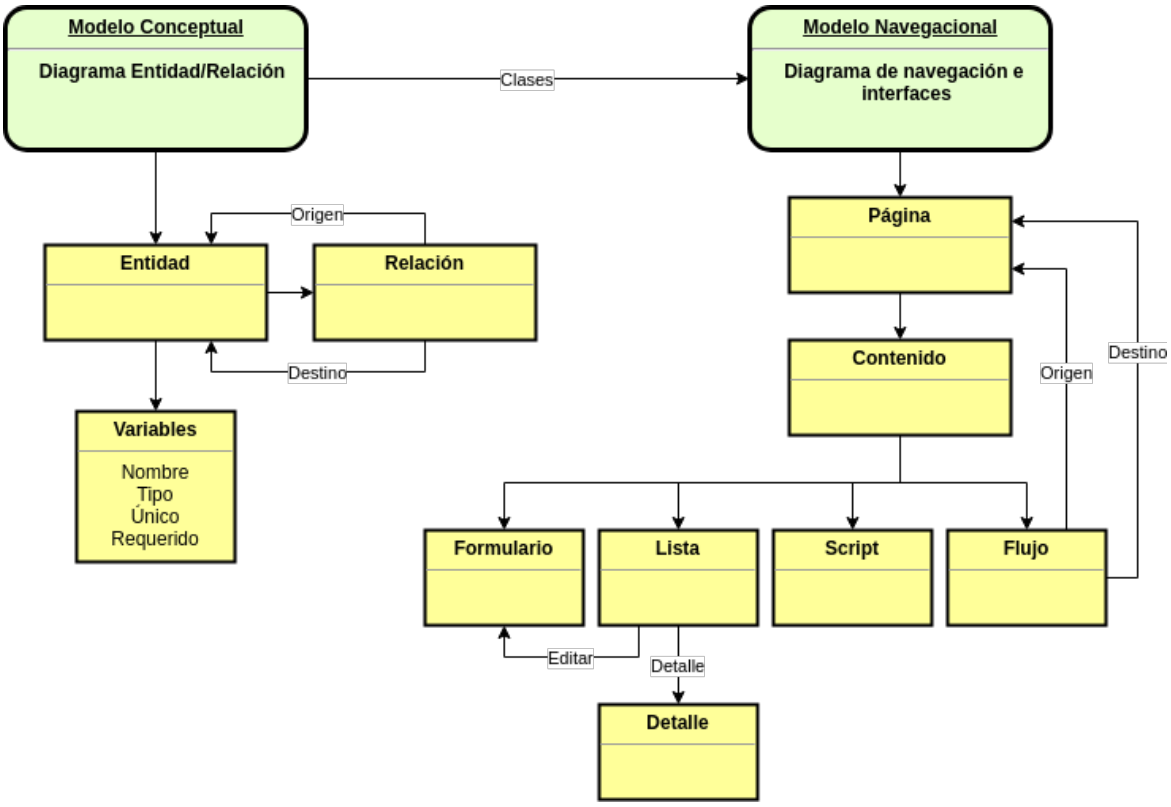


Fig. 3.1 Metamodelo de la aplicación SSWMFA

3.2.1 Diagramado del Modelo Conceptual

La herramienta proveerá la funcionalidad de crear un diagrama UML de Entidad Relación [6] donde se podrá representar cada propiedad a persistir como una entidad con sus atributos.

Se deberá especificar por cada atributo el **tipo** (numérico, texto, fecha y otros tipos permitidos por la Base de Datos adyacente), si es de valor **único** (determina que no puede haber dos registros con el mismo valor en este atributo) y si es **requerido**

insertar un valor (determina que de no insertarse un valor se muestre un mensaje de error diciendo que el campo no puede estar vacío).

Además de las entidades se podrán graficar, en caso de ser necesario, las relaciones entre éstas. Las relaciones entre dos entidades puede ser uno a uno (1..1 - 1..1), uno a muchos (1..1 - 1..*), o muchos a muchos (1..* - 1..*). En caso de ser optativa se representa con un 0 en lugar del 1.

Un ejemplo de un diagrama de Entidad Relacion puede verse en la Figura 3.2.

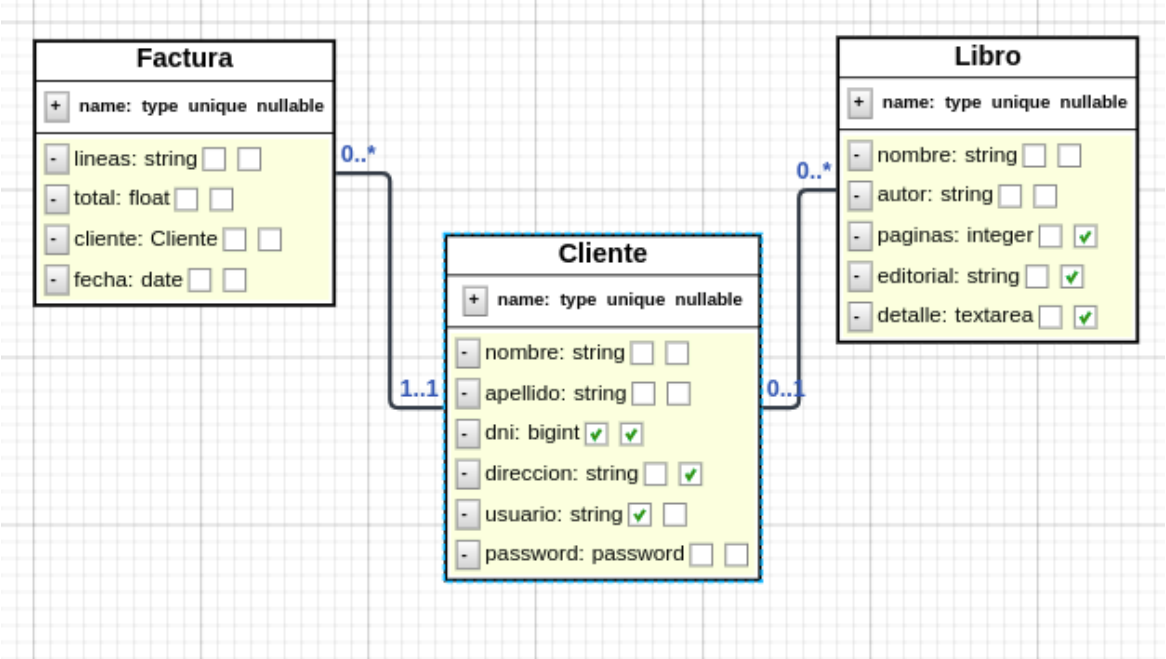


Fig. 3.2 Ejemplo de diagrama UML de Entidad Relación

Una vez finalizado el diagrama conceptual se permitirá la creación la base de datos correspondiente donde cada Entidad será mapeada a una tabla y sus atributos serán campos en dicha tabla. Además, las relaciones entre entidades se representarán como claves foráneas entre las tablas o, en el caso de relación muchos a muchos, como una tabla intermedia.

3.2.2 Diagramado del Modelo de Navegación

Luego de finalizado el diagramado del modelo conceptual y habiendo especificado las entidades con sus atributos y relaciones, será posible modelar la navegación de la aplicación web usando como base el estándar IFML de Nodos y Links de una forma simplificada, donde los Nodos serán las páginas y los Links, los hipervínculos entre éstas.

Las páginas se diagraman como elementos individuales (nodos) que contienen otros elementos.

En la figura 3.3 se pueden observar tres páginas sin contenido donde `Home` tiene links (hipervínculos) hacia `Página 1` y `Página 2`. En caso de renderizar este diagrama se obtendrán tres páginas sin contenido y los links correspondientes.

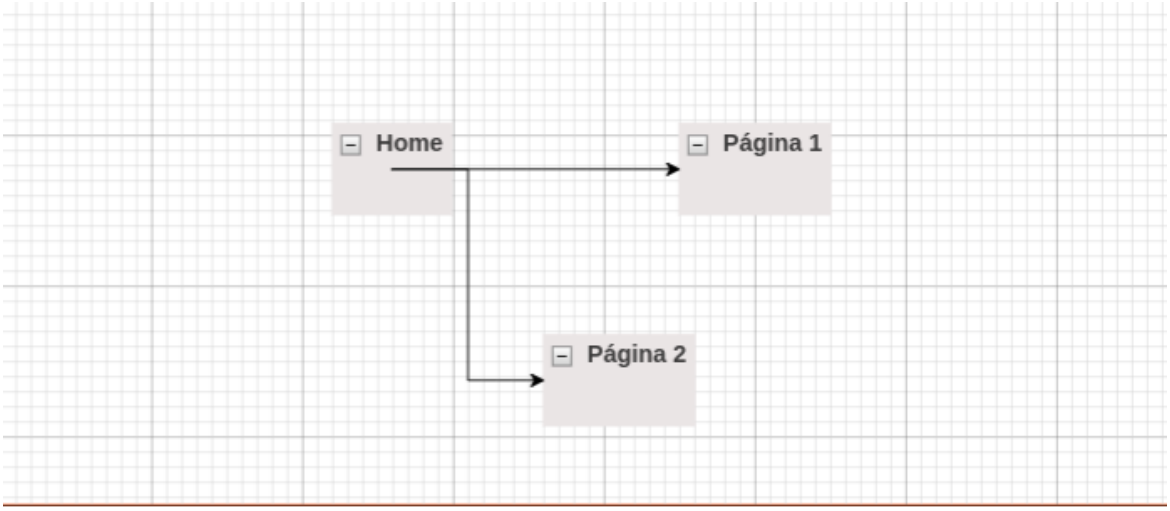


Fig. 3.3 Páginas sin contenido

Los tres tipos de elementos que puede contener una página son: **formularios**, **listas** y **scripts**, adicionalmente a los **links** hacia otras páginas.

3.2.2.1 Formularios

Los formularios deben relacionarse con alguna (solo una) entidad del diagrama conceptual y se utilizan para crear o editar registros, de la clase relacionada, en la base de datos.

Los **formularios de creación** de registros figuran explícitamente en el diagrama dentro de una página, en cambio los **formularios de edición** de registros se crean como consecuencia de la posibilidad de editar registros en las listas, como se describirá en la siguente sección.

En la figura 3.4 se modeló una `Página de Clientes` que contiene un formulario de alta de clientes.

Al hacer la relación del elemento formulario con la entidad `Cliente` del modelo conceptual automáticamente se listarán los atributos de la entidad permitiendo tildar aquellos que serán incluídos como campos del formulario.

En este ejemplo se decidió que el campo `num_cliente` no se mostrara ya que es un campo autoincremental que se completará automáticamente por el motor de base de datos a la hora de insertar el registro.

Una vez renderizada la `Página de Clientes` se obtendrá una página con el formulario `Dar de alta un cliente` cuyos campos serán los atributos tildados en el modelo, como se muestra en la Figura 3.5. Cabe que aclarar que los campos marcados como requeridos en el modelo conceptual serán siempre incluídos en el formulario, de otra manera fallará la inserción o actualización de registros en la base de datos. Los campos requeridos tendrán un estilo especial en la vista, en el ejemplo figuran con un ***** luego del nombre y contarán con la validación en el navegador que provee HTML5 [7].

3.2.2.2 Listas

Al igual que los formularios, las listas deberán relacionarse con una, y solo una, entidad del diagrama conceptual tildando los atributos que se mostrarán como campos en el listado.

Un elemento Lista se inserta dentro de una página y tiene las siguientes funcionalidades sobre los registros de la entidad relacionada:

- Listar los registros en forma de lista HTML

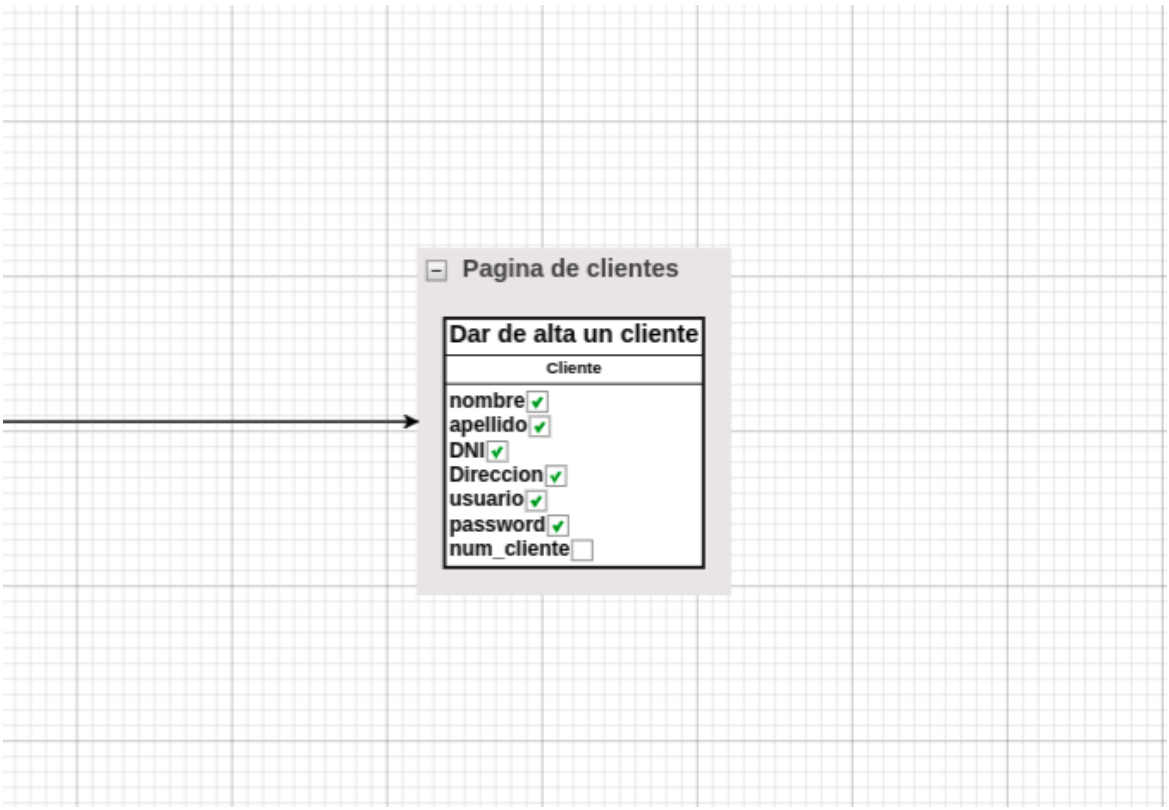


Fig. 3.4 Formulario relacionado con la entidad Clientes

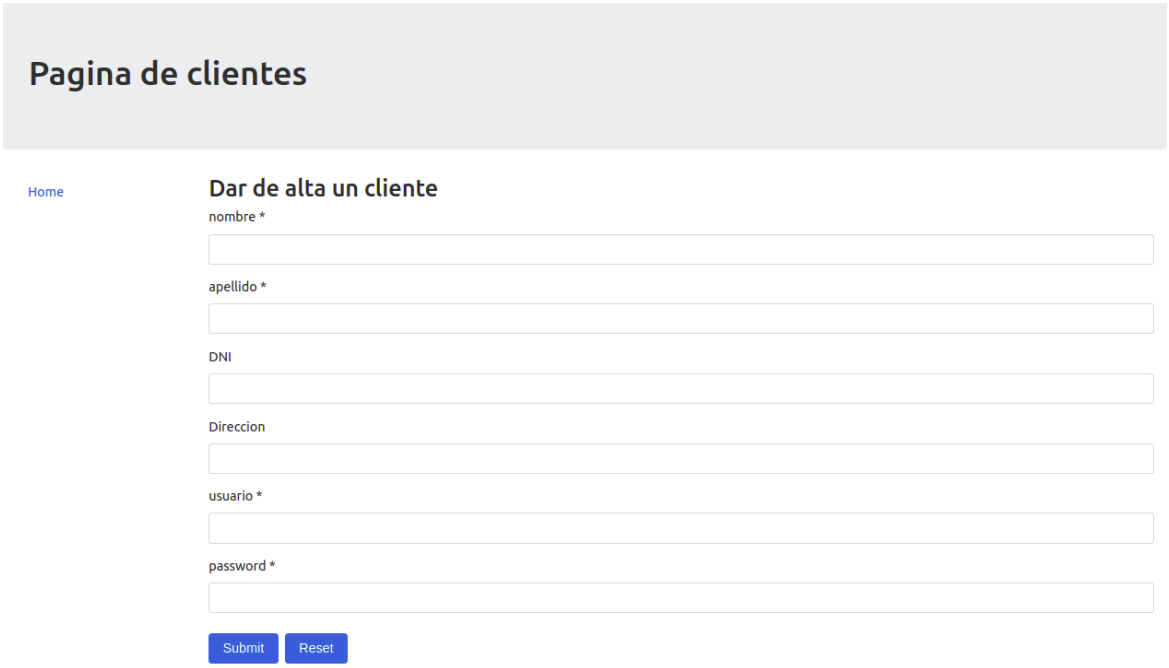


Fig. 3.5 Formulario renderizado

- Eliminar registros (opcional). Se muestra un botón por cada registro de la lista que hace un llamado al backend eliminándolo de la base de datos.

- Editar registros (opcional). Esta funcionalidad se provee a través de un link por cada registro que redirige a una página que contiene un formulario de edición del registro con los valores actuales precargados en los campos del formulario, se permite editarlos y guardar los cambios. Esto dispara una acción al backend que actualiza el registro en la base de datos.

- Mostrar detalle del registro (opcional). Al activar esta funcionalidad se muestra un link por cada registro que redirige a una página de detalle donde se listan todos los campos del registro con sus valores. Esto puede ser utilizado en caso de que en el listado se muestren solo algunos campos y el resto de la información del registro esté disponible en la página del detalle.

  En la Figura 3.6 se muestra el modelo de una página que contiene un formulario (alta) de clientes y un listado de los clientes insertados, al renderizar esta página se obtiene el formulario de alta y el listado modelados.

  En la Figura 3.7 se puede ver que se dieron de alta dos clientes, `Foo` y `Bar`, y el listado solo muestra los campos tildados en el modelo con las acciones marcadas para cada registro: Eliminar, Editar y Detalle.

3.2.2.3 Scripts

La herramienta proveerá la posibilidad de insertar un elemento del tipo Script que contenga código JavaScript que, al renderizar el modelo, será ejecutado cuando se acceda a la página que lo contiene.

En la Figura 3.8 se puede ver un elemnto Script dentro de una paǵina, este script lo que hará será imprimir el mensaje `"hello"` en la consola del navegador cuando la página sea renderizada y abierta en el navegador web.

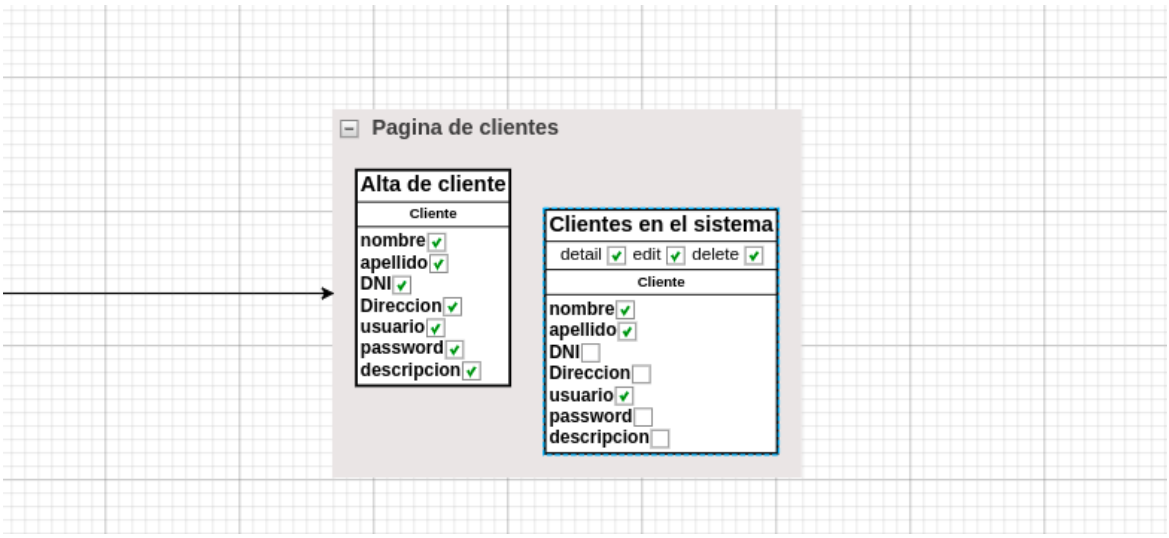


Fig. 3.6 Modelo de página con elemento Lista relacionada con la entidad Clientes

Fig. 3.7 Listado renderizado

3.2.2.4 Links

En el diagrama, las páginas (nodos) pueden vicularse entre ellas con links (lazos direccionados) especificando las páginas origen y destino. Un link desde una página hacia otra se renderiza como un link o hipervínculo en la página origen.

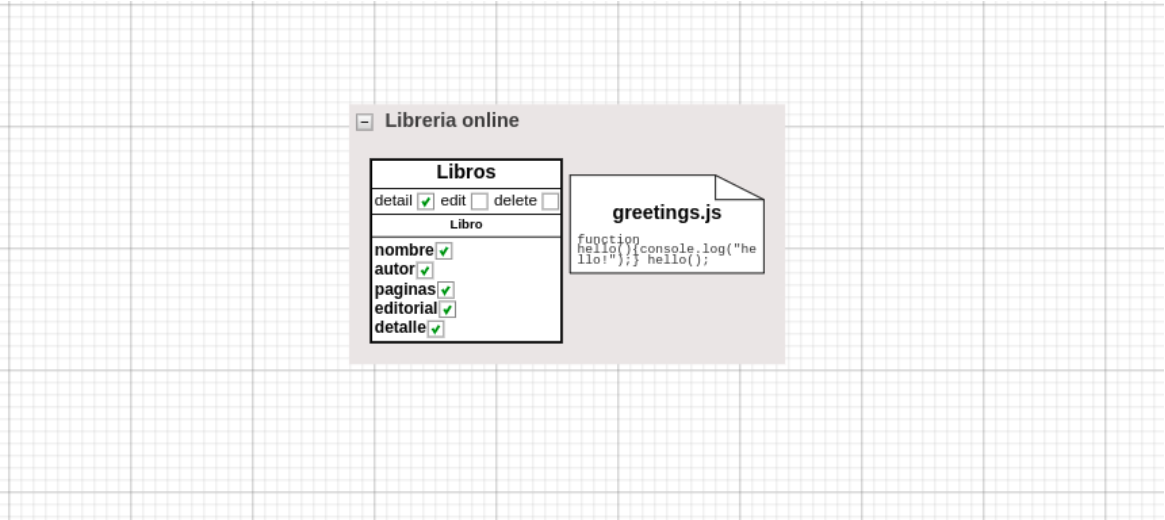


Fig. 3.8 Elemento Script dentro de una página

En la Figura 3.9 se muestra el ejemplo de la página `Libreria online` que tiene links hacia `Pagina de clientes`, `Pagina de libros` y `Pagina de facturacion` lo que, luego de renderizado, se transformarán en tres hipervínculos hacia estas páginas como se muestra en la Figura 3.10.

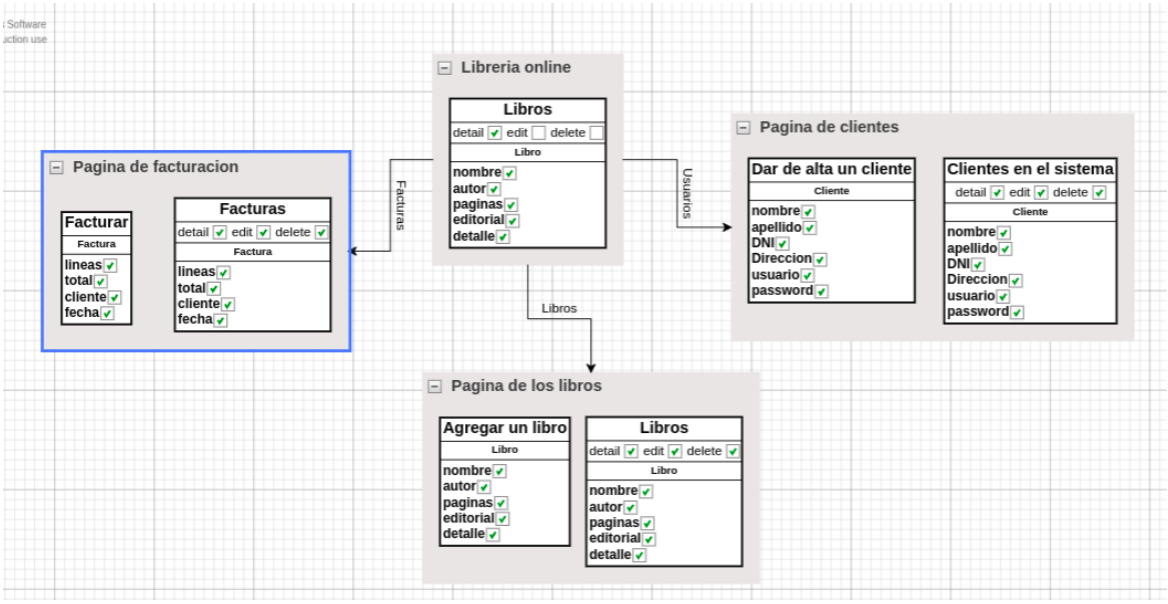


Fig. 3.9 Modelo de página con links a otras páginas

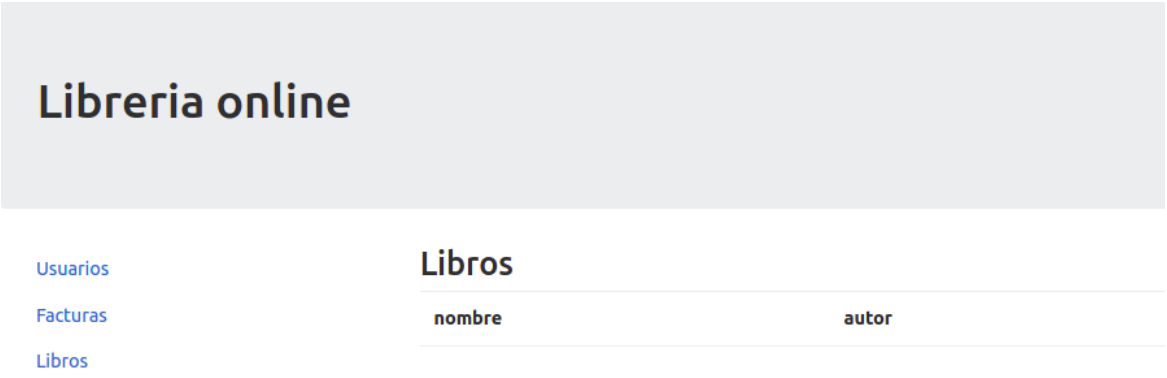


Fig. 3.10 Links renderizados

3.2.3 Renderizado

El renderizado es el último paso en el modelado de aumentaciones web en el lado del servidor.

Una vez modelada la navegación se podrán renderizar las páginas web con los elementos descriptos en la sección anterior. Lo que hará el renderizado será crear el código HTML de las páginas y la funcionalidad de los elementos configurando la relación de la vista con la base de datos a través de un ORM (mapeo objeto-relacional) [19].

Habiendo renderizado el modelo navegacional será posible navegar a través de las páginas e interactuar con la base de datos creando (persistiendo), listando, editando y eliminando registros.

El código de las páginas será lo suficientemente simple como para que una persona con conocimientos básicos de programación web pueda modificarlas agregando, editando o quitando funcionalidades y estilo (CSS).

4. Evaluación de la herramienta

El objetivo es demostranr cómo SSWMFA permite de modo fácil y rápido desarrollar el comportamiento server-side de una aumentación. A modo de ejemplo veremos un tipo de aumentación que mejora un sitio web en el dominio de la agricultura que vende diferentes tipos de semillas. Además se muestran las maneras de implementar la interfaz con el comportamiento client-side de la aumentación brindando endpoints para el desarrollo de plugins.

4.0.1 Transmisión de contenido desde el servidor

4.0.1.1 Extraer el modelo conceptual de la aplicación principal

El primer paso en el enfoque MDWE es capturar el modelo subyacente percibido por el usuario final que se utilizará más adelante para mejorar la aplicación web. Por ejemplo, la figura 4.1 destaca una parte específica del DOM que contiene información acerca de una instancia de la clase `Product`. También es posible ver que otros elementos DOM representan más instancias de la misma clase. Estas instancias parecen presentar información similar relacionada con diferentes propiedades de la clase virtual Product como su nombre, precio y una breve descripción.

La idea principal es definir el concepto de producto como una clase, con sus atributos y sus relaciones con los elementos DOM, por lo que es posible extraer los valores de las propiedades para cada instancia concreta.

4.0.2 Modelando el comportamiento server side

Una vez obtenidas las clases y sus atributos es posible comenzar el modelado del comportamiento server side de la aumentación utilizando la herramienta SSWMFA.

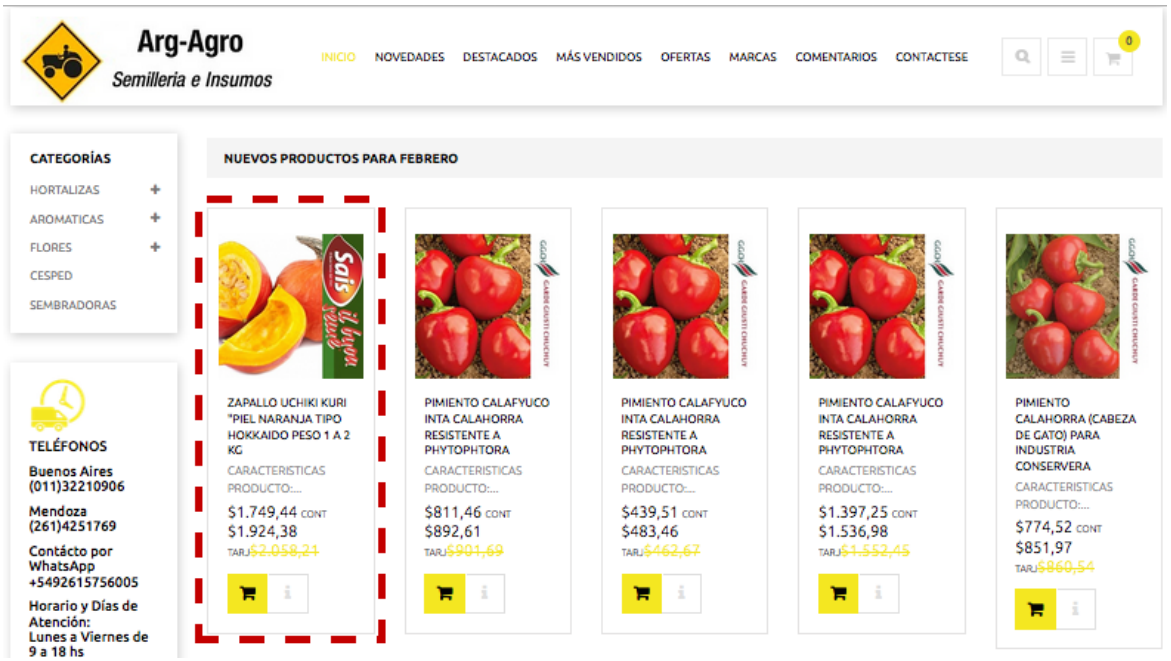


Fig. 4.1 Identificación de la instancia y concepto en contenido web existente

4.0.2.1 Creación el modelo conceptual

Se comienza con el diseño del modelo conceptual detallado en la sección 3.2.1 (Figura 4.2) que documenta cómo el aumento enriquece el modelo de la aplicación central.

En el ejemplo, una clase virtual de productos está enriquecida con un conjunto de comentarios que representan las revisiones de la comunidad. Cabe señalar que la clase virtual modelada en SSWMFA solo requiere los atributos de la página que se manejarán en el lado del servidor y los nuevos atributos que mejoran su definición. El OID (Object ID) que denota el identificador único es creado automáticamente por la herramienta a través del ORM cuando se inserta un nuevo registro a la base de datos.

El primer paso es la descripción del modelo conceptual utilizando el modelado Entidad/Relación como se observa en la Figura 4.3. La clase virtual producto se aumenta con una referencia multivaluada a la Entidad `Comment`. La referencia se denomina por la relación en el diagrama que especifica que un producto puede tener

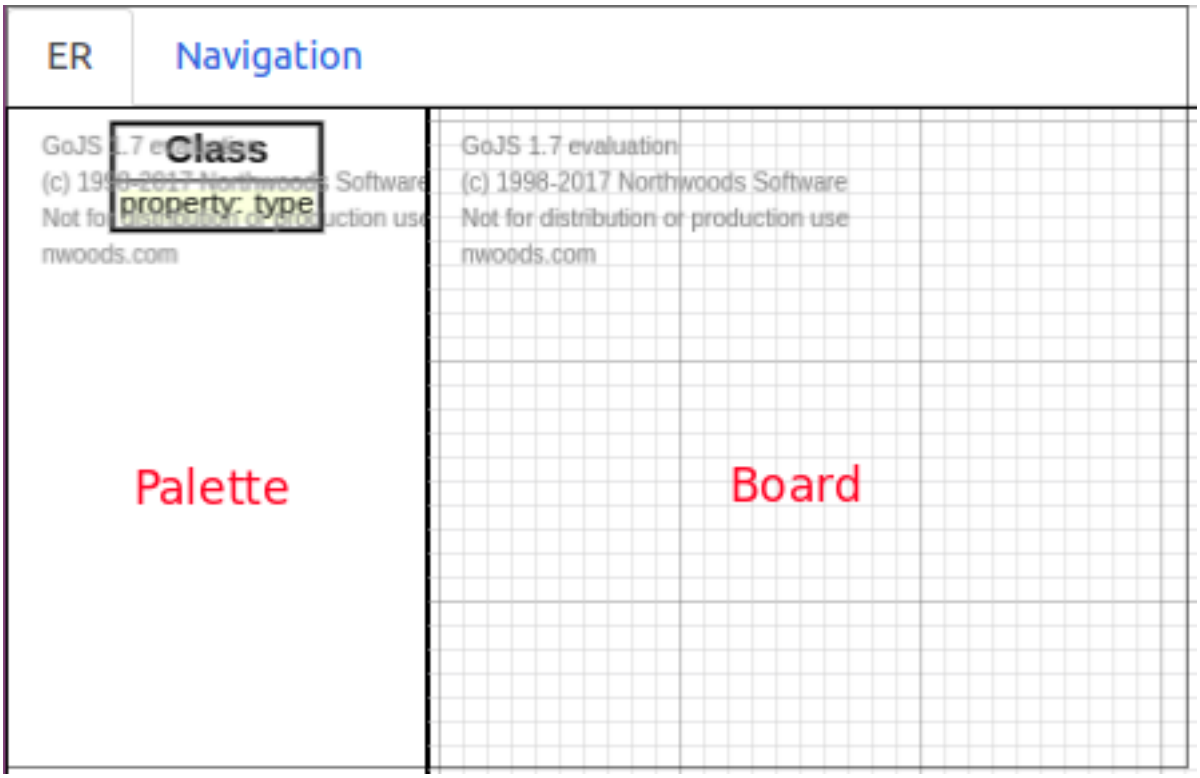


Fig. 4.2 Sección ER - Palette y Board

cero o varios comentarios y un comentario tiene solo un producto asociado donde la clave foránea se almacena en el campo `product`.

Es de destacar que la entidad `Comment` no necesita indicar toda la información disponible en la página, de hecho, solo se requiere un conjunto básico de información (por ejemplo, cualquier información que identifique el objeto, como un identificador o un nombre) que permita una implementación de búsqueda básica.

El diagrama de Entidad/Relación se modeló en la sección **ER** de SSWMFA. Este diagrama permite agregar de la paleta un solo tipo de nodo: Class (Clase), que representa a una Entidad y posteriormente a una tabla en la Base de Datos. Al crear una Clase se deben completar:

- **Nombre**: Un string con un valor único en el diagrama que especifique el nombre de la Entidad.

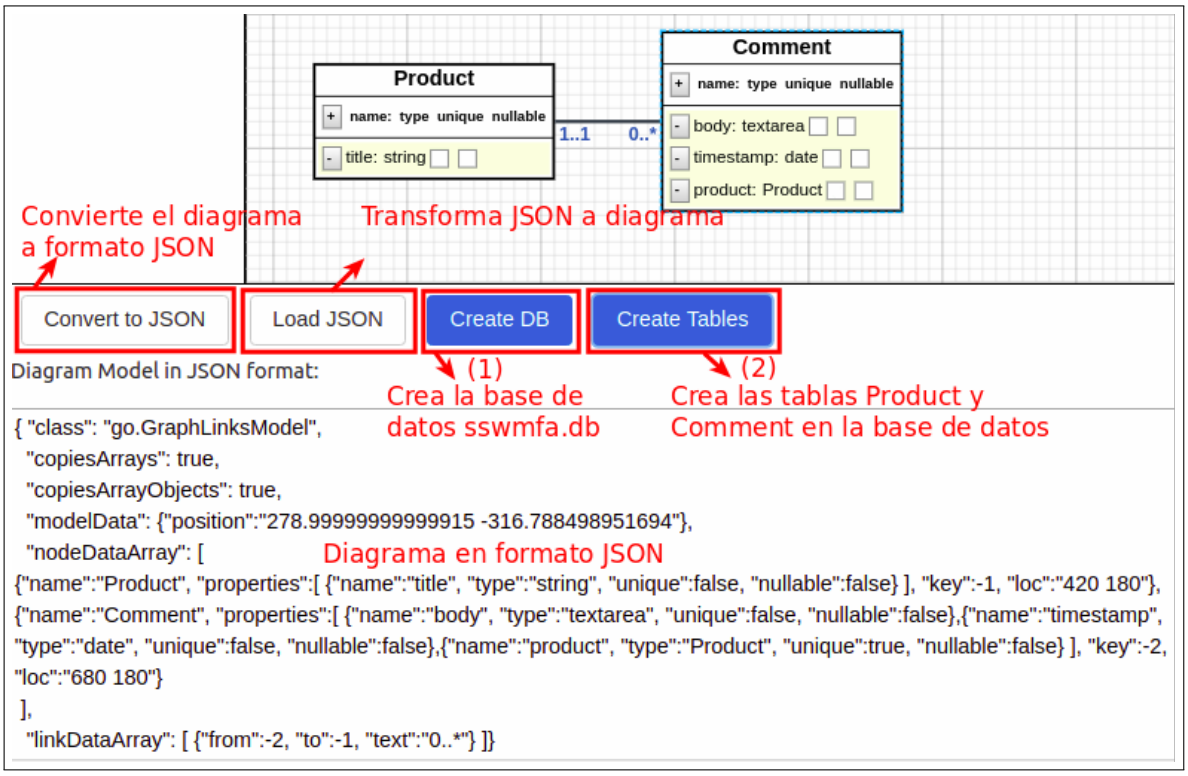


Fig. 4.3 Modelo Entidad-Relación

- **Propiedades**: Los atributos de la Entidad. Mediante el botón `+` se pueden agregar atributos a la Entidad, y cada atributo puede eliminarse mediante el botón `-`.
- **Relación a otras clases**: Determina la relación entre Entidades.

Las propiedades de una clase tienen las siguientes características:

- **Nombre**: Un string con un valor único dentro de la Clase que especifique el nombre del atributo de la Entidad a la que pertenece.
- **Tipo**: Un string que determina el tipo del atributo en la Base de Datos. Los tipos soportados por SSWMFA son: integer, number, bigint, text, textarea, date, datetime y string. El tipo por defecto es string por lo que si se inserta un valor que no está dentro de los especificados el atributo será creado como un campo en la Base de Datos de tipo `STRING`.

- **Único**: Un checkbox booleano donde chequeado es "true" y no chequeado es "false". Si se especifica como "true" el atributo se creará como un campo en la Base de Datos como `UNIQUE=TRUE`, lo que determina que ese atributo no puede repetir su valor dentro de la tabla. Puede utilizarse, por ejemplo, para el DNI de una persona ya que es un valor que no se repite entre personas, sino que es único. Si se intenta insertar un valor duplicado (que ya existe en otro registro en la tabla), el ORM retorna un mensaje de error.

- **Requerido**: Un checkbox booleano donde chequeado es "false" y no chequeado es "true". En este caso es invertido el valor ya que el nombre es `nullable` que quiere decir que permite que el atributo no tenga valores (no requerido). Si un atributo se especifica como requerido (not nullable), el campo de la Base de Datos se creará como `NOT NULL` y si se intenta guardar vacío, el ORM retorna un mensaje de error.

Las relaciones entre Entidades deben especificar su cardinalidad las cuales pueden ser puede ser uno a uno (1..1 - 1..1), uno a muchos (1..1 - 1..*), o muchos a muchos (1..* - 1..*). En caso de ser optativa se representa con un 0 en lugar del 1.

Una vez realizado el diagrama de Entidad/Relación se puede proceder con las funcionalidades de la sección **ER** de SSWMFA, que se pueden observar en la Figura 4.3:

- **Convert to JSON**: El diagrama dibujado puede convertirse a formato JSON [3] que puede ser guardado en un archivo de texto o ser consumido por una API externa. Los campos que se guardan en el JSON son

  - `"class"`: Este campo le indica a la librería GoJS que se trata de un diagrama y que puede interpretarlo, por lo tanto siempre será `"go.GraphLinksModel"`.
  - `"modelData"`: Contiene las coordenadas del diagrama en el board, esto sirve para mantener los elementos en la misma posición dentro del board cuando sean cargados (transformados de JSON a diagrama).

  - `"nodeDataArray"`: Contiene una lista de los elementos del diagrama, a partir de ahora los llamaremos nodos. Cada nodo representa una clase y tiene el nombre (`"name"`: string), una clave única de nodo en el diagrama (`"key"`:integer), una posición en el board (`"loc"`: coords) y una lista de atributos (`"properties"`), donde por cada atributo se especifica su nombre (*"name"*: string), tipo (*"type"*: string), si solo permite valor único en la tabla (*"unique"*: boolean) y si es requerido (*"nullable"*:boolean)
  - `"linkDataArray"`: Representa las relaciones entre Entidades. Contiene los valores `"from"`: key del nodo origen; `"to"`: key del nodo destino; `"text"`: cardinalidad de la relación.

- **Load JSON**: Puede cargarse un diagrama en formato JSON insertando el texto en el textarea en la zona *Diagram Model in JSON format* y llamando a la función que traduce el texto en diagrama a través del botón Load JSON. Esta función es nativa de la librería GoJS.

- **Create DB**: Crea la Base de Datos `sswmfa.db` vacía. En este ejemplo se configura el ORM para trabajar con una Base SQLite3 [1] como se puede observar en la Figura 4.4

Fig. 4.4 Archivo que contiene la Base de Datos en SQLite3

- **Create Tables**: Mapea las Entidades a Tablas en la Base de Datos y las crea haciendo uso de las funciones del ORM. Este mapo se puede observar en el archivo **models/db.js** Figura 4.5.

[1] https://www.sqlite.org

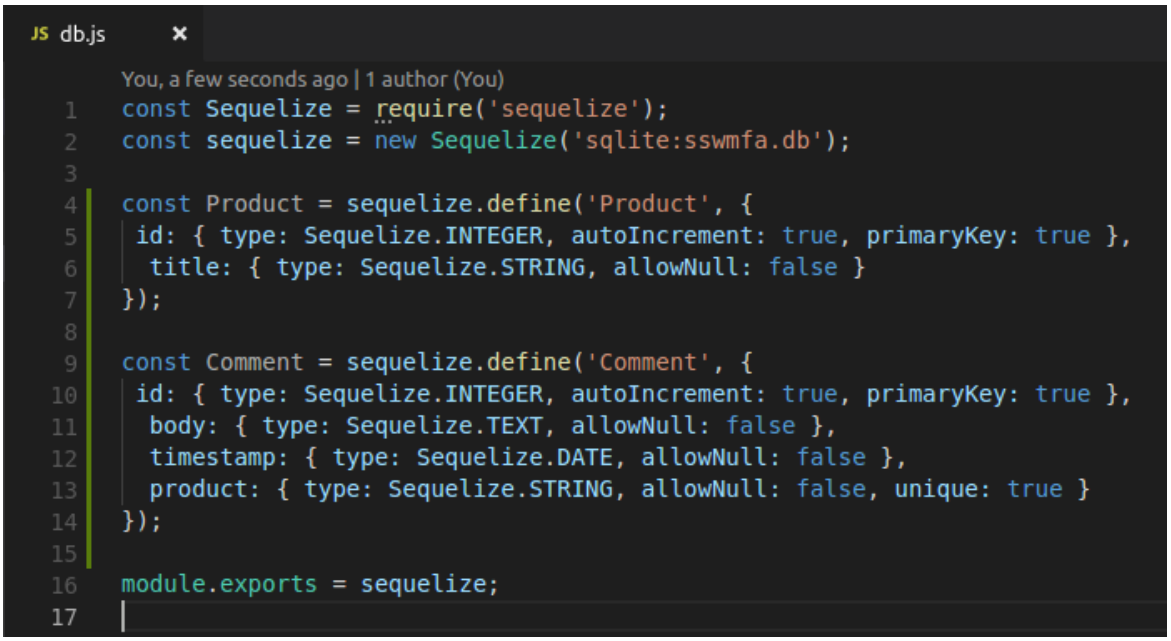


Fig. 4.5 Archivo de mapeo de Entidades a Tablas

Habiendo creado la Base de Datos y las tablas se puede proceder con el diagrama navegacional.

4.0.2.2 Plugin Create Entity

Para facilitar la integración con programas de aumentación, SSWMFA posee un endpoint `createEntity` que espera recibir desde el lado del cliente un objeto JSON con información de una Entidad. Esto puede ser utilizado a la hora de capturar el modelo subyacente del sitio a aumentar mediante un plugin en la herramienta de aumentación que permita seleccionar entidades del DOM del sitio, extraer las propiedades (atributos) de cada entidad y realizar un llamado al endpoint `createEntity` con el objeto extraído.

Una vez recibida la información, SSWMFA parsea a la representación JSON del diagrama ER agregando esa Entidad en su lista de nodos `nodeDataArray`.

El formato de contenido del mensaje debe especificar el nombre de la clase `"class"` y una lisa de `"fields"` especificando por cada campo su `"name"`, `"type"`, `"unique"` y `"nullable"`:

```
{
 "class": "Product",
 "fields": [
     {
      "name": "name",
            "type": "string",
            "unique": "true",
            "nullable": "true"
      },
      {
       "name": "description",
             "type": "textarea",
             "unique": "false",
             "nullable": "true"
       }
   ]
}
```

Al enviar el mensaje al endpoint se puede observar en el modelo ER que, luego de cargar el JSON al diagrama, se dibujó la entidad como una Class en el board del diagrama como muestra la figura 4.6

Cada entidad que se envíe al endpoint agregará un nuevo nodo Class en el diagrama ER, por lo que, para agregar muchas entidades, deberán realizarse múltiples llamados al endpoint uno con cada entidad a agregar.

Continuando con el ejemplo, los mensajes a enviar desde la herramienta hacia el backed para crear las Entidades `Product` y `Comment` son:

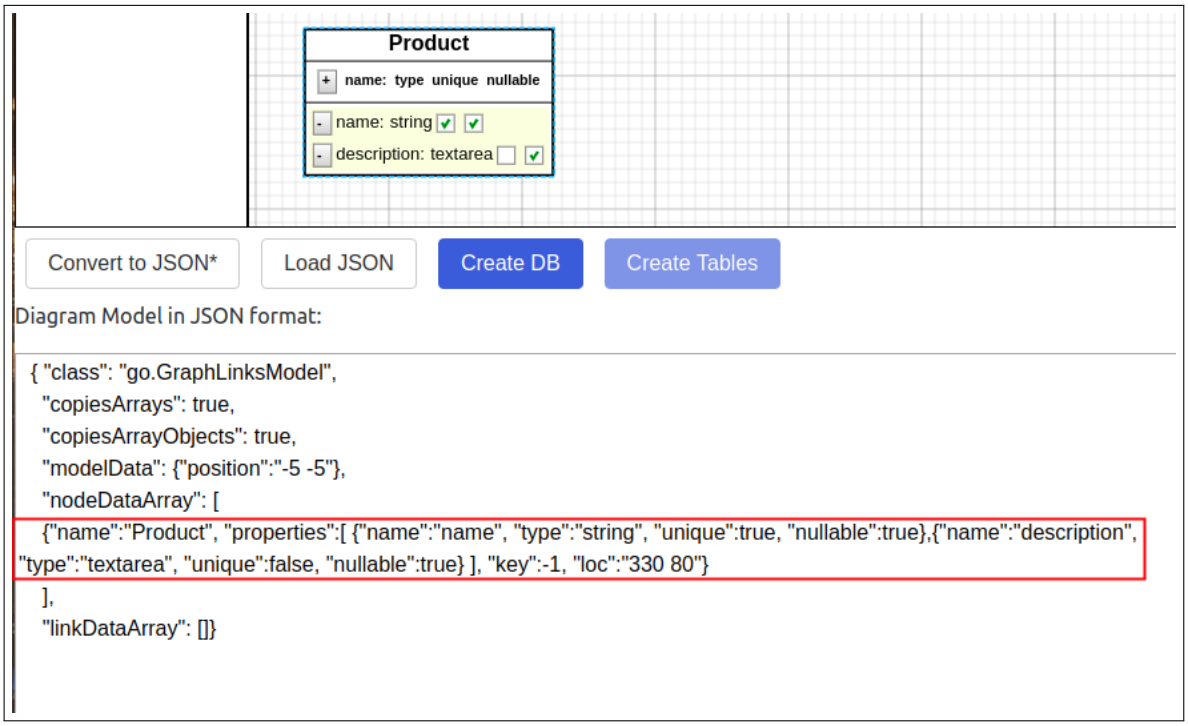


Fig. 4.6 Representación de los comentarios en el lado del cliente

```
{
 "class": "Product",
 "fields": [
     {
      "name": "name",
           "type": "string",
           "unique": "false",
           "nullable": "false"
      }
   ]
}
```

```
{
 "class": "Comment",
 "fields": [
     {
      "name": "body",
           "type": "textarea",
           "unique": "false",
           "nullable": "false"
      },
      {
       "name": "timestamp",
            "type": "date",
            "unique": "false",
            "nullable": "false"
       },
      {
       "name": "product",
            "type": "Product",
            "unique": "false",
            "nullable": "false"
       }
   ]
}
```

Adicionalmente SSWMFA provee la posibilidad de borrar todos los nodos del diagrama ER simplemente llamando al endpoint `clearEntities`. Esto limpia la variable en memoria que contiene todas las Entidades recibidas externamente.

4.0.2.3 Creación el modelo de navegación

Las acciones en el lado del cliente requieren, además de la mejora del modelo conceptual, mejoras en los modelos de navegación para describir el comportamiento de la aplicación.

Se modela utilizando nodos, elementos y enlaces (detallados en 3.2.2) en la sección **Navigation** de SSWMFA.

La Figura 4.7 muestra que la paleta contiene los cuatro nodos: Formulario, Lista, Script y Página.

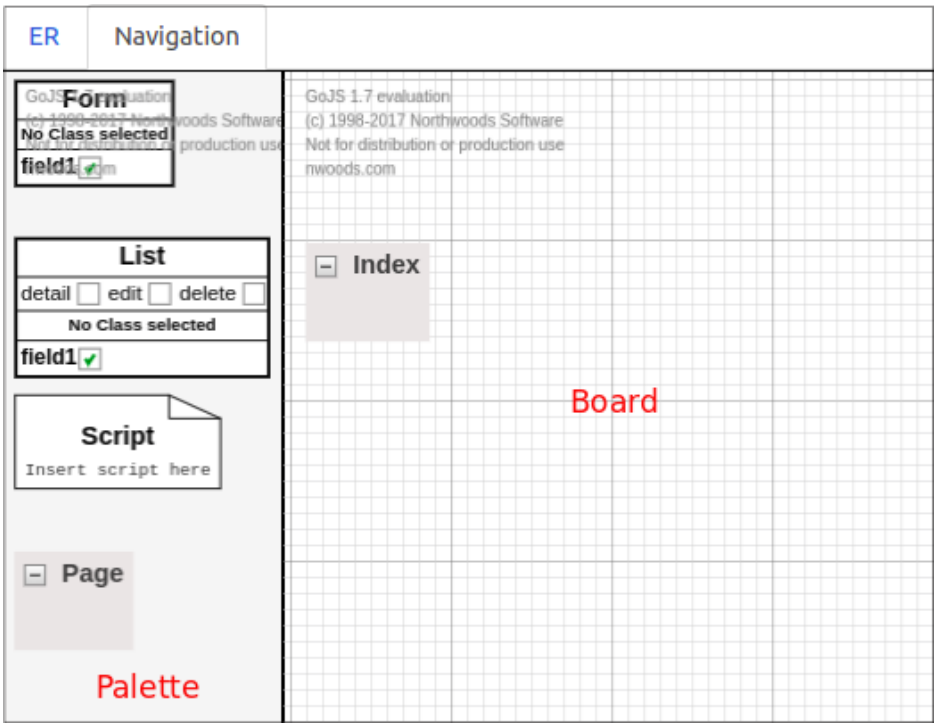


Fig. 4.7 Sección Navigation - Palette y Board

En el board se puede ver que hay una página `Index`, esta página se crea por defecto y no puede ser eliminada, pero sí renombrada, ya que será la página principal de la navegación cuya url será la raíz **/render**.

En la Figura 4.8 se definen dos páginas: `Comments` que provee la funcionalidad de crear comentarios nuevos utilizando un formulario (detallado en 3.2.2.1) y un listado (detallado en 3.2.2.2) de los comentarios acerca de los productos. El listado

permite además ver detalles de un comentario mediante un link a una página de detalle de comentario, editar comentarios mediante un link a una página que contiene un formulario de edición de comentario y eliminar comentarios; la otra página es `Products` que permite las mismas acciones de alta, baja, modificación, listado y detalle de productos. Las dos páginas se encuentran enlazadas mediante links desde una hacia la otra y viceversa.

Las opciones al momento de diagramar una página son:

- Elegir un nodo `Página` desde la paleta y llevarlo al board. La página puede renombrarse, esto sirve para establecer la url que será `/nombre_de_la_pagina`. En el ejemplo, `Comments` es la página Index por lo que su url será la raíz `/render`, pero la url de Products será `/render/products`.

- En caso de ser necesario agregar un formulario de alta se deberá elegir un nodo `Form` de la paleta y llevarlo dentro de la página, luego hacer click derecho sobre el formulario y elegir a cuál Entidad del diagrama conceptual se va a relacionar.

  En este ejemplo uno de los formularios fue asociado a la Entidad `Product` y el otro a `Comment`. Al asociarlo a una Entidad automáticamente se listarán dentro del formulario las propiedades de la clase y cada una tendrá un checkbox que al estar tildado significa que ese campo figurará en el formulario renderizado, en caso contrario el campo no se motrará en la navegación y todo elemento creado tendrá ese campo como `NULL` dentro de la Base de Datos. En este caso hay que tener en cuenta que los campos establecidos como `NOT NULL` figurarán siempre en el formulario renderizado (marcado con un `*`) ya que no es posible insertar un valor `NULL` en ellos como lo indica el parámetro.

- De forma análoga al paso anterior, de ser necesario insertar un Listado se deberá elegir un nodo `List` de la paleta e insertarlo en la página. Un listado debe ser asociado a una Entidad de la misma manera que un formulario, donde se listarán los campos de la clase y se podrá elegir si se muestran en el listado o no mediante el checkbox de cada campo.

Como se explicó en 3.2.2.2, una lista provee las funcionalidades de editar, eliminar y mostrar el detalle de cada registro. Estas funcionalidades se habilitan marcando los checkboxes correspondientes en el nodo como se muestra en la Figura 4.8

- En caso de ser necesario agregar un script que se ejecute en el momento de cargar una página en el navegador se deberá elegir un nodo `Script` e insertarlo dentro de la página. En el cuerpo del nodo debe ir el código del script que, al momento del renderizado, será creado como un archivo de la extensión especificada en el nombre del nodo (por ejemplo "hello.js" será creado como un archivo de nombre *hello* con extensión *.js* de JavaScript) y agregado en la carpeta `public/rendered` del proyecto, finalmente se importará en el header de la página.

- Las páginas pueden enlazarse entre sí creando links entre ellas como se muestra en la Figura 4.8. Cada link tiene un origen, un destino y un texto que será el texto del link en la página renderizada. En este ejemplo se creó un link desde `Comments` hacia `Products` "products", y uno inverso "comments".

Una vez realizado el diagrama de Navegación se puede proceder con las funcionalidades de la sección **Navigation** de SSWMFA, que se pueden observar en la Figura 4.8:

- **Convert to JSON**: Funciona de la misma manera que en **ER** donde `"class"` es siempre `"go.GraphLinksModel"`, `"modelData"` contiene las coordenadas del diagrama dentro del board y `"linkDataArray"` contiene los links entre las páginas y su descripción.

  A diferencia de **ER** en **Navigation** `"nodeDataArray"` tiene la lista de elementos en el diagrama dividido por dos tipos:

  - Páginas: Como se explicó anteriormente en un diagrama hay una página principal por lo que siempre hay un elemento de `"category": "MainPage"`. El resto de las páginas tendrá `"category": "Page"`. Los otros campos de las páginas son `"name"`: string que contiene el nombre de la página;

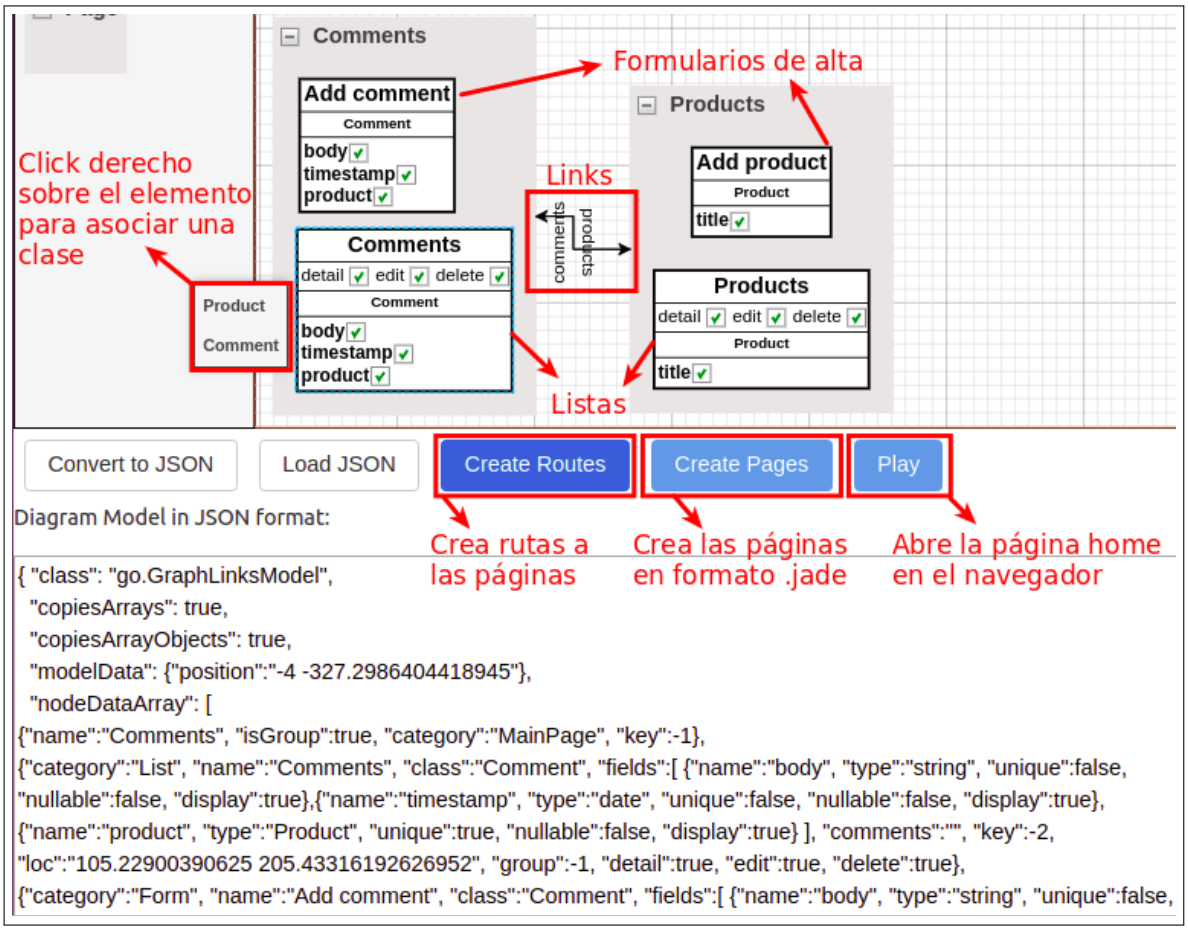


Fig. 4.8 Modelo de navegación

`"isGroup":true` que especifica que es un nodo que contendrá otros nodos; y `"key"`: integer que contiene el id del nodo en el diagrama.

– Elementos: Tienen `"category"`: String que determina la categoría del nodo (Form, List o Script); `"name"`: String que contiene el nombre del nodo; `"class"`: clase asociada del diagrama conceptual; `"key"`: integer que contiene el id del nodo en el diagrama; `"group"`: el valor de `"key"` de la página en la que está contenido; `"fields"`: los campos de la clase asociada con sus respectivos valores de `"name"`, `"type"`, `"unique"` y `"nullable"` explicados en el diagrama conceptual, y `"display"`: checkbox que determina si el campo será mostrado en el listado renderizado o no.

- **Load JSON**: Funciona de la misma manera que en **ER**.

- **Create Routes**: Crea las rutas a las páginas. El routeo de Node/Express funciona definiendo en un archivo de routeo el direccionamiento [2] que hace referencia a la definición de puntos finales de aplicación (URI) y cómo responden a las solicitudes de cliente. Cada página debe tener su ruta definida en el archivo como se puede observar en la Figura 4.9.

render.js

```
You, a few seconds ago | 1 author (You)
var express = require('express');
var router = express.Router();
var sequelize = require('../models/db');
//below autogenerated routes
router.get('/', function (req, res) {
  sequelize.models.Comment.findAll().then(recordsFromComment => {
    res.render('rendered/comments', { title: 'Comments',   recordsFromComment : recordsFromComment });
  });
});

router.get('/products', function(req, res) {
  sequelize.models.Product.findAll().then(recordsFromProduct => {
    res.render('rendered/products', { title: 'Products',   recordsFromProduct : recordsFromProduct });
  });
});

router.post("/insertOneInProduct", function(req, res){
  sequelize.models.Product.create({
    title: req.body.title
  })
  .then(()=> {res.redirect('back')})
  .catch(error => {console.log(error);res.redirect('back')});
});

router.get("/updateOneInProduct/:id", function(req, res){
  sequelize.models.Product.findById(req.params.id).then(record => {
    res.render("rendered/formUpdateOneInProduct", { title: "Update in Product", record : record })
  });
});

router.post("/updateOneInProduct", function(req, res){
  sequelize.models.Product.findById(req.body.id).then(record => {
    record.title = req.body.title;
    record.save()
    .then(()=> {res.render("rendered/formUpdateOneInProduct", { title: "Update in Product", record : record })})
    .catch(error => {console.log(error);res.redirect('back')});
  });
});

router.get("/deleteOneFromProduct/:id", function(req, res){      You, a few seconds ago • Uncommitted changes
  sequelize.models.Product.findById(req.params.id).then(record => {
    record.destroy().then(()=> {res.redirect('back')})
  });
});

router.get("/detailOfOneFromProduct/:id", function(req, res){
  sequelize.models.Product.findById(req.params.id).then(record => {
    res.render("rendered/detailFromProduct", { title: "Detail of record from Product", record : record });
  });
});
```

Fig. 4.9 Archivo de ruteo render.js

Las rutas que se crean son:

[2] http://expressjs.com/es/guide/routing.html

  - A las páginas: En los ejemplos son `/` y `/products` que son rutas relativas a `/rendered`.
  - Al detalle de registros: Por cada listado que tenga la funcionalidad de detalle de registro se crea una ruta que comienzan con el prefijo `/detailOfOneFrom{ nombre_de_la_tabla }/:id`, donde `id` es el id del registro a mostrar el detalle. En el ejemplo se crearon las rutas `/detailOfOneFromProduct/:id` y `/detailOfOneFromComment/:id`.
  - Al formulario de edición de registros: Por cada listado que tenga la funcionalidad de edició de registro se crea una ruta que comienzan con el prefijo `/updateOneInProduct{ nombre_de_la_tabla }/:id`, donde `id` es el id del registro a editar. En el ejemplo se crearon las rutas `/updateOneInProduct/:id` y `/updateOneInComment/:id`.

- **Create Pages**: Crea las páginas en formato `.jade` dentro de la carpeta `views/rendered`. Figura 4.10. En el ejemplo se crearon las páginas de `comments` y `products`, además se crearon las páginas de detalle y edición de registros para cada una de las clases `detailFromComment`, `detailFromProduct`, `formUpdateOneInComment` y `formUpdateOneInProduct`.

  Las páginas son generadas utilizando el template engine Jade y heredan de `views/layout.jade` que contiene los headers necesarios para que las páginas puedan renderizarse.

- **Play**: Abre en una nueva pestaña del navegador la página Index.

4.0.2.4 Renderizando y navegando las páginas de administración

Luego de realizados los modelos conceptuales y de navegación del comportamiento server side de la aumentación es posible renderizar las páginas para así obtener una interfaz de usuario amigable que funciona como una sección administrador. En estas páginas se pueden agregar, editar, eliminar, listar y detallar los productos y comentarios a los productos extraídos del DOM del sitio a aumentar.

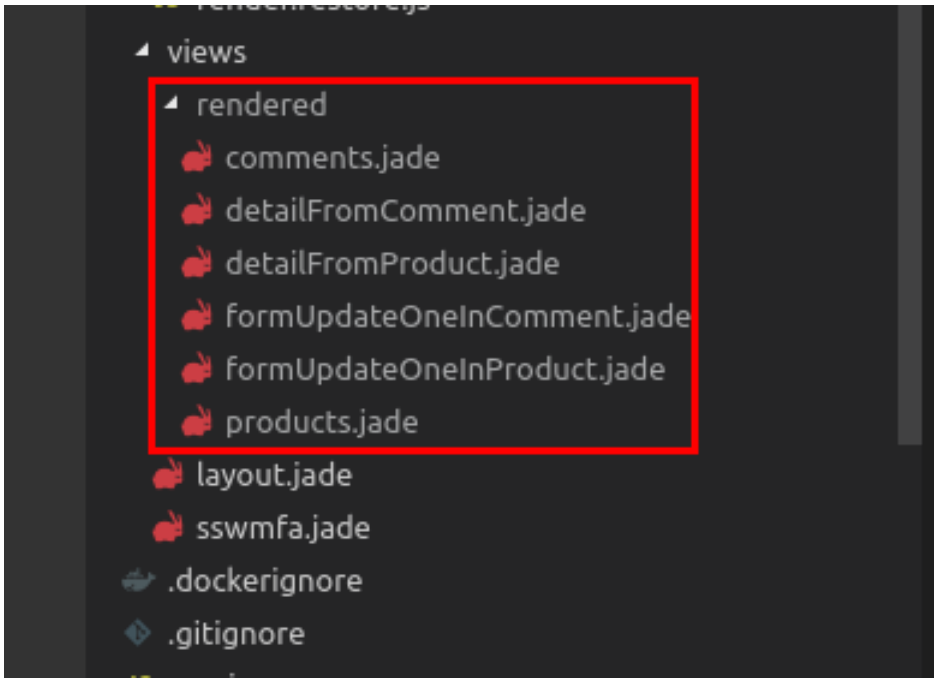


Fig. 4.10 Páginas renderizadas

Como se explicó anteriormente, en la sección **Navigation** de SSWMFA es posible renderizar estas páginas y, luego de disparar la función **Play** comenzar a navegarlas.

En la Figura 4.11 se renderizó y se comenzó a navegar desde la página principal que contiene un link a la página de productos y el formulario de alta y el listado de comentarios. Se puede observar que el listado de comentarios no muestra registros, esto es porque la tabla `Comments` está vacía.

Para agregar un comentario primero es necesario tener un producto, por lo que en la Figura 4.12 se agregó un producto extraído del sitio web a aumentar: *"Hurricane pumpkin F1 "Precocious Waltham, High Quality""*.

Ahora estamos en condiciones de agregar un comentario desde el administrador. En la Figura 4.13 se agregaron comentarios al producto realizando la asociación por `id`.

4.0.3 Consumiendo la información en el lado del cliente

Finalmente, del lado del cliente, el diseñador debe estudiar el mejor diseño UX para presentar la lista de comentarios de productos extraídas de SSWMFA.

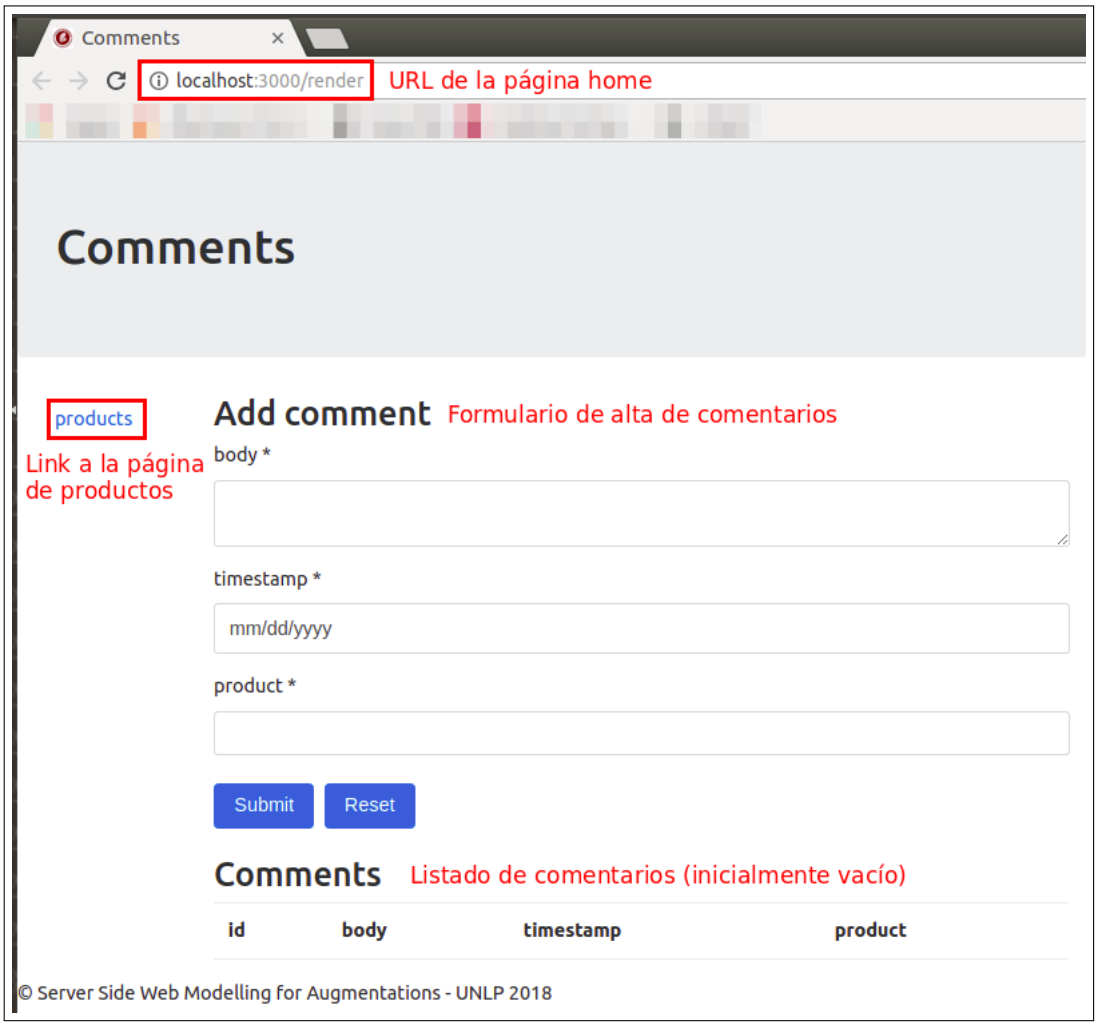


Fig. 4.11 Página de comentarios - Home Page

La forma de obtener el código HTML de las páginas renderizadas es hacer un HTTP Request con el método GET a la ruta de la página deseada. En el caso del ejemplo, para obtener el código de la página de **Comments** se debe realizar una llamada a `localhost:3000/render` y capturar el código HTML de la respuesta como se puede observar (en parte) en la imagen 4.14.

Esta forma de obtener el código HTML de las páginas renderizadas puede servir para implementar el diseño UX de la herramienta de aumentación, ya que puede tomarse el código y editarse según lo deseado.

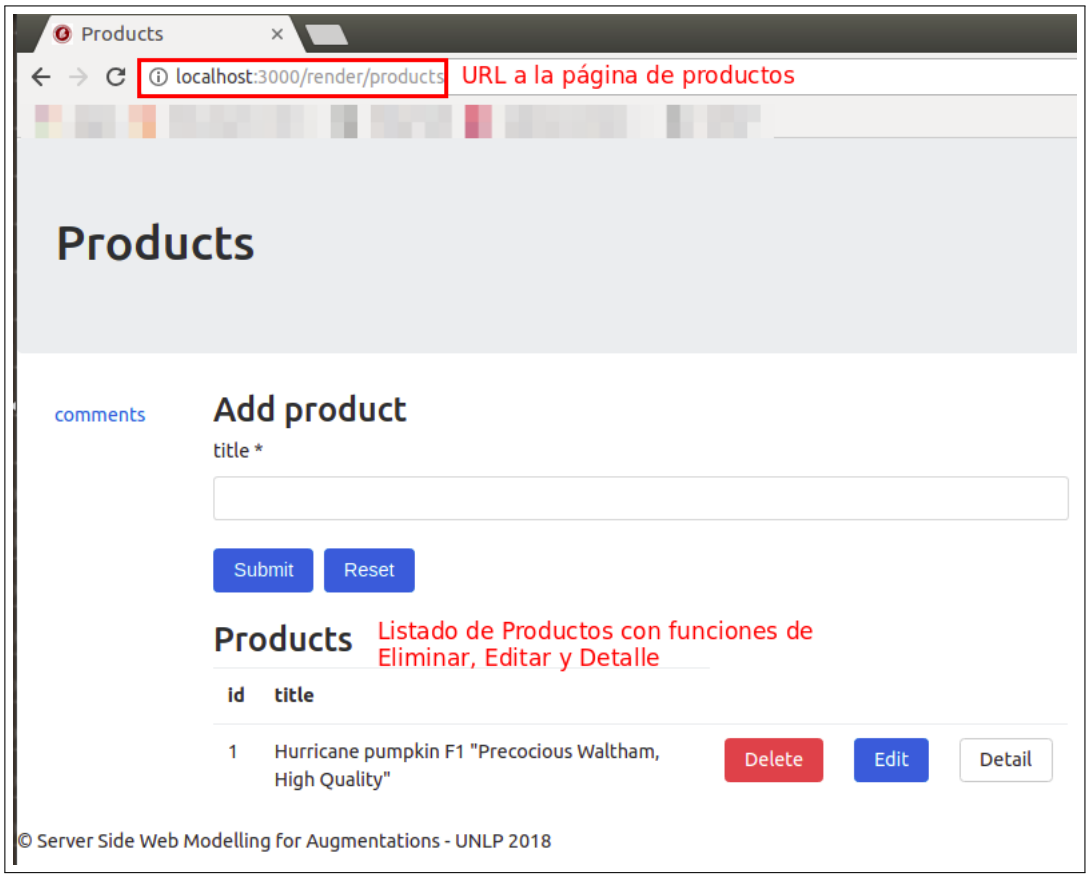


Fig. 4.12 Página de productos

Otro enfoque, diferente al utilizar las páginas de adminstración, para comunicarse con el backend para crear, editar, eliminar y listar los datos de la Base de Datos es implementar plugins en la herramienta de aumentación en el frontend que realicen llamados HTTP Requests a los endpoints que provee SSWMFA. Estos endpoints se crean en el momento de crear las rutas del modelo de navegación y se encuentran definidos en el archivo `render.js`.

Siempre existe la posiblidad de editar el archivo `render.js` para crear endpoints personalizados, por ejemplo, si se desea tener un endpoint que devuelva un **Producto** de la base de datos sin el HTML que lo envuelve especificando el `id`, podría editarse el archivo `render.js` agregando el siguiente código:

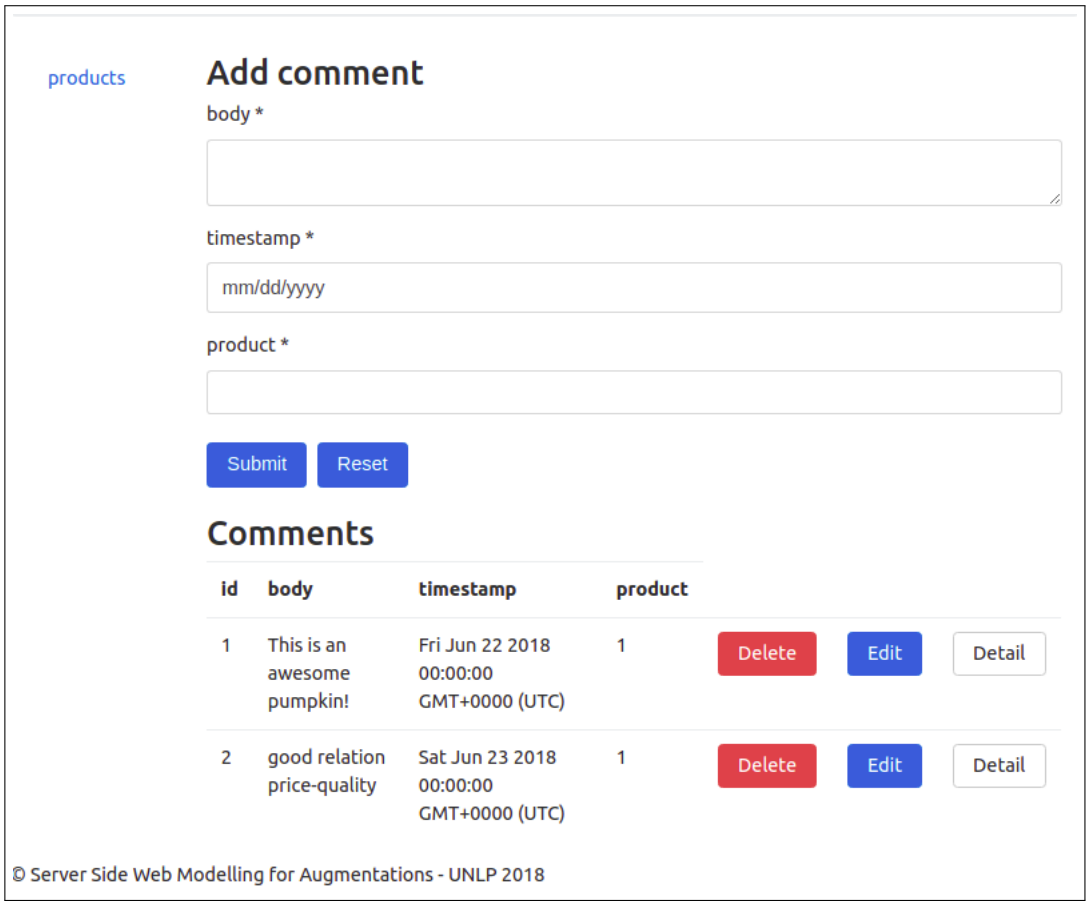


Fig. 4.13 Página de comentarios con comentarios

GET http://localhost:3000/render/

Pretty Raw Preview HTML

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Comments</title>
        <script src="../../src/gojs/assets/js/jquery.min.js"></script>
        <script src="../../src/gojs/assets/js/jquery-ui.min.js"></script>
        <link rel="stylesheet" href="../../src/css/style.css">
        <script src="https://npmcdn.com/tether@1.2.4/dist/js/tether.min.js"></script>
        <link rel="stylesheet" href="../../src/bootstrap-4.0.0-alpha.6-dist/css/bootstrap.min.css">
        <script src="../../src/bootstrap-4.0.0-alpha.6-dist/js/bootstrap.min.js"></script>
    </head>
    <body>
        <div class="jumbotron">
            <h1>Comments</h1>
        </div>
        <div class="container-fluid">
            <div class="row">
                <div class="col-sm-2">
                    <nav class="nav flex-column">
                        <a href="/render/products" class="nav-link">products</a>
                    </nav>
                </div>
                <div class="col-sm-10">
                    <h3>Add comment</h3>
                    <form id="formComment" action="/render/insertOneInComment" method="POST">
                        <div class="form-group">
                            <label>body *</label>
                            <textarea id="body" name="body" required="" class="form-control"></textarea>
                        </div>
                        <div class="form-group">
                            <label>timestamp *</label>
                            <input id="timestamp" name="timestamp" type="date" required="" class="form-control">
                        </div>
                        <div class="form-group">
                            <label>product *</label>
                            <input id="product" name="product" type="text" required="" class="form-control">
                        </div>
                        <div class="form-group">
                            <button type="submit" class="btn btn-primary">Submit</button>
                            <button type="reset" class="btn btn-primary margin-btn">Reset</button>
                        </div>
                    </form>
                    <h3>Comments</h3>
                    <table class="table">
                        <thead class="thead-dark">
                            <tr>
                                <th scope="col">id</th>
                                <th scope="col">body</th>
                                <th scope="col">timestamp</th>
                                <th scope="col">product</th>
                            </tr>
                        </thead>
                        <tbody>
                            <tr>
                                <td>1</td>
                                <td>This is an awesome pumpkin!</td>
                                <td>Sun Sep 02 2018 00:00:00 GMT+0000 (UTC)</td>
                                <td>1</td>
                                <td>
                                    <a href="/render/deleteOneFromComment/1" role="button" class="btn btn-danger">Delete</a>
                                </td>
                                <td>
                                    <a href="/render/updateOneInComment/1" role="button" class="btn btn-primary">Edit</a>
                                </td>
                                <td>
```

Fig. 4.14 Código HTML en la respuesta

```
router.get("/getProduct/:id", function(req, res){
  sequelize.models.Product
  .findById(req.params.id)
  .then(record => {
    res.json(record);
  });
});
```

Para comprender un poco más sobre cómo deben ser las respuestas en Express referirse al sitio oficial [3].

Una vez creado el nuevo endpoint ya puede llamarse mendiante un HTTP Request con el método GET a `http://localhost:3000/render/getProduct/1` y la respuesta, en este ejemplo, será:

[3] http://expressjs.com/es/api.html#express.json

```
{
 "id": 1,
 "title": "Hurricane pumpkin F1 \"Precocious Waltham, High Quality\"",
 "createdAt": "2018-09-08T21:51:44.516Z",
 "updatedAt": "2018-09-08T21:51:44.516Z"
}
```


En la Figura 4.15 se muestra cómo podría ser un ejemplo de la UX de la información obtenida de SSWMFA sobre los comentarios de los productos. En el ejemplo se pueden ver los dos comentarios agregados desde el administrador de la aumentación server side.

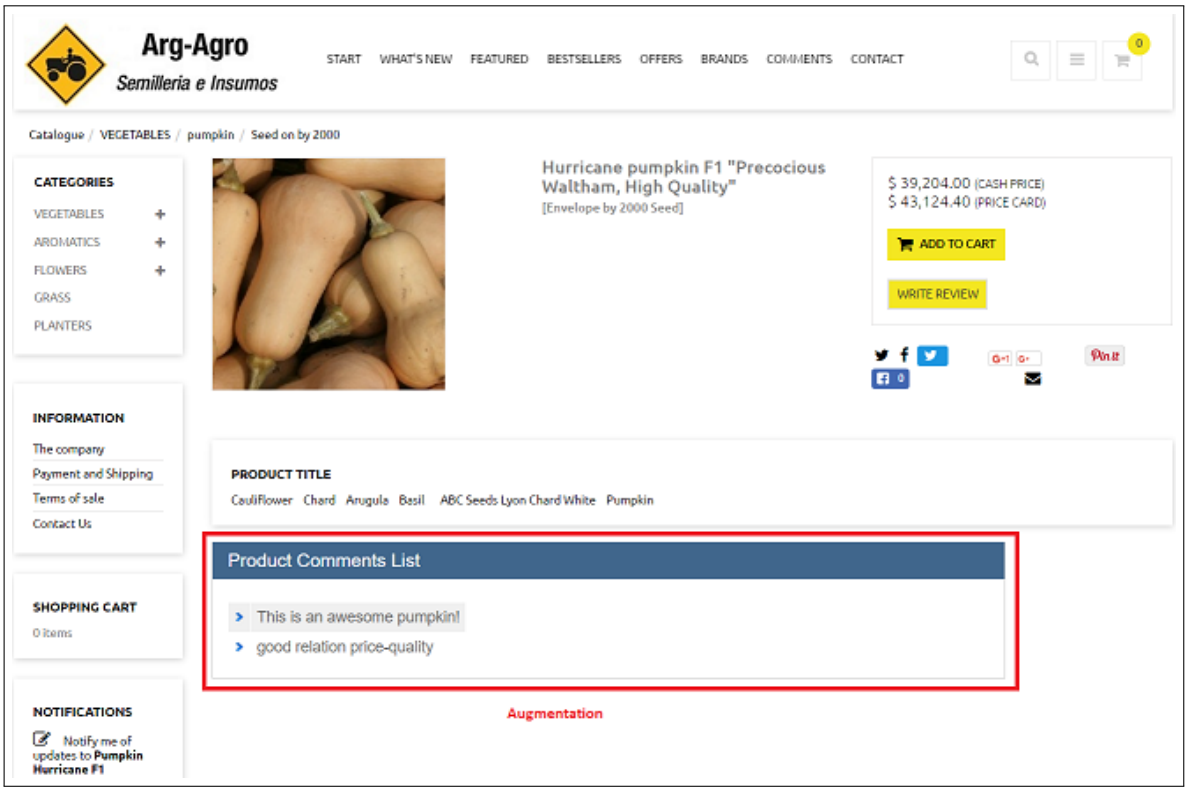


Fig. 4.15 Representación de los comentarios en el lado del cliente

5. Un ejemplo comprensivo

Siguiendo los pasos detallados en el Capítulo 4, y utilizando la extensión de Chrome MDWA Collector [1] para desarrollar el comportamiento client-side y SSWMFA para modelar el comportamiento server-side, se procederá a realizar un ejemplo concreto de aumentación del sitio de venta online de electrodomésticos de la marca Frávega [2].

El objetivo es extraer el modelo conceptual de la aplicación principal utilizando la extensión MDWA Collector; enviar la información al lado del servidor mediante una llamada al endpoint de SSWMFA; modelar el comportamiento server side realizando los modelos conceptuales y de navegación; y finalmente recibir los datos desde el lado del cliente para almacenarlos y manipularlos en la base de datos de la aumentación.

5.1 Transmisión de contenido desde el servidor

5.1.1 Extraer el modelo conceptual de la aplicación principal

Como se explicó en el Capítulo 4, el primer paso en el enfoque MDWE es capturar el modelo subyacente percibido por el usuario final.

Para realizar esto, primero ingresamos al sitio web de Frávega y nos dirigimos a la sección de televisores[3]. Una vez allí activamos la extensión de Chrome MDWA Collector como se muestra en la Figura 5.1.

La extensión abrirá un cuadro de bienvenida y nos avisará que no existen elementos anotados en el sitio y nos alienta a crear un nuevo template anotando el DOM de la aplicación principal (Figura 5.2).

[1] https://github.com/pauerbino/extensionIAW
[2] https://www.fravega.com
[3] https://www.fravega.com/tv-y-video/tv/smart%20tv

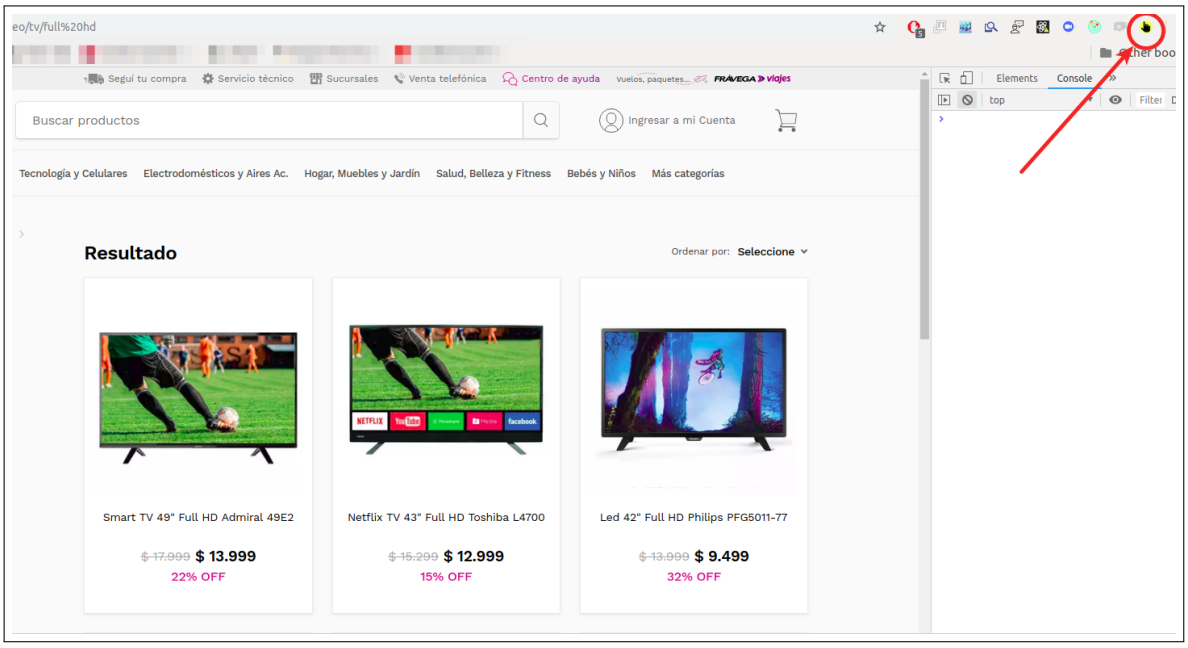


Fig. 5.1 Una vez en el sitio a aumentar comenzamos con el proceso haciendo click en la extensión MDWA

El siguiente paso es ingresar el nombre del concepto o elemento que queremos anotar en el sitio y luego insertar un tag. En este caso el concepto es **"Product"** y el tag es **"television"** (Figura 5.3). Esto se mapeará en el lado del servidor como una tabla llamada **Television**.

Luego, se selecciona el elemento de la aplicación principal que contiene el o los conceptos a anotar utilizando el mouse y clickeando en el elemento. Una vez seleccionado el elemento del DOM se lo marcará en amarillo en el sitio principal.

También se deben elegir las ocurrencias del concepto a anotar. En este caso sólo elegiremos solo una (Figura 5.4).

Finalmente elegiremos las propiedades que tendrá el elemento a aumentar. Estas propiedades serán mapeadas como atributos de la tabla **Television**.

En este caso solo tomaremos el título del producto y lo almacenaremos como **description**. Podemos ver en la Figura 5.5 cómo se muestra un ejemplo del valor que puede tomar esa propiedad.

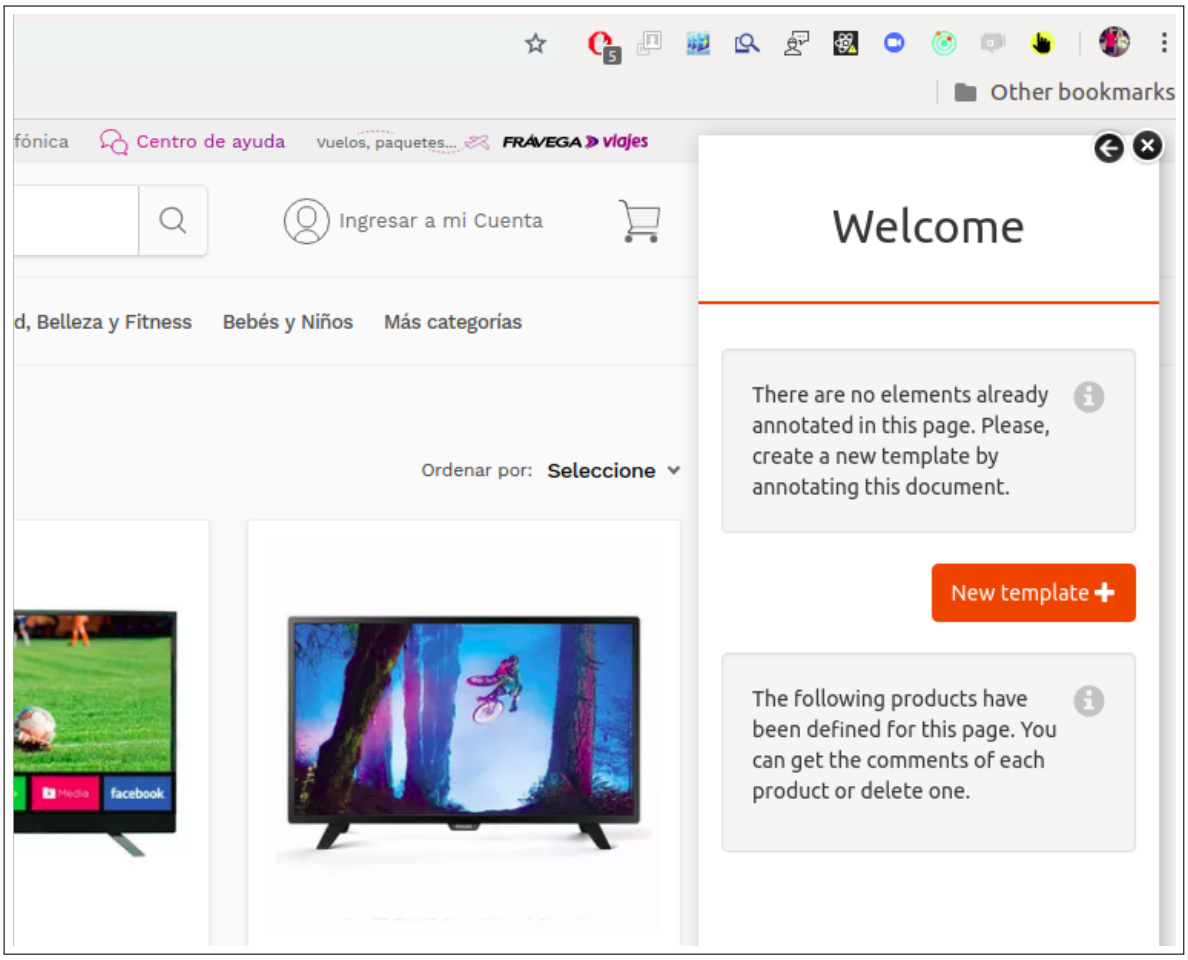


Fig. 5.2 Comenzando con la aumentación

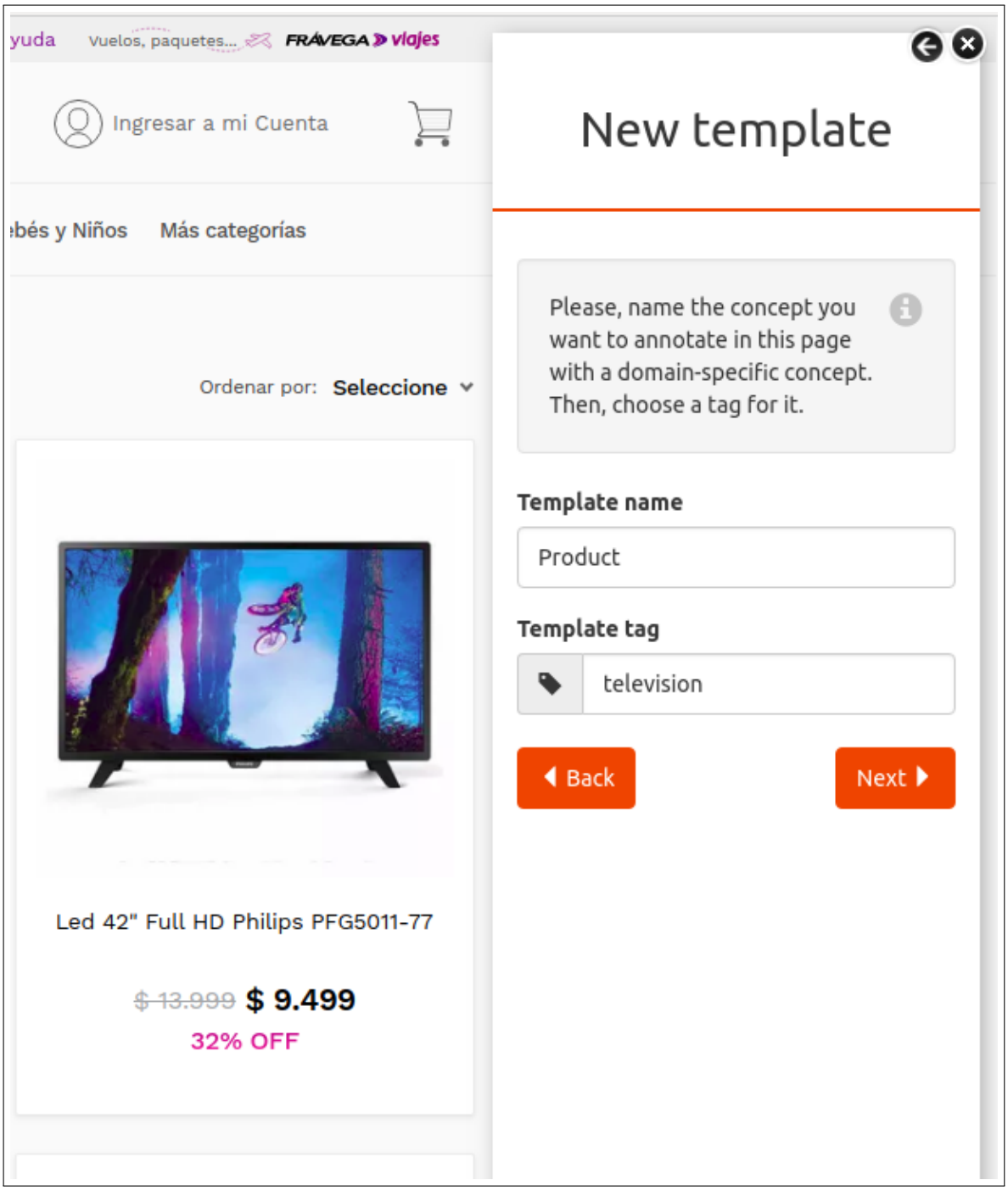


Fig. 5.3 Se inserta nombre y tag del elemento del DOM a capturar

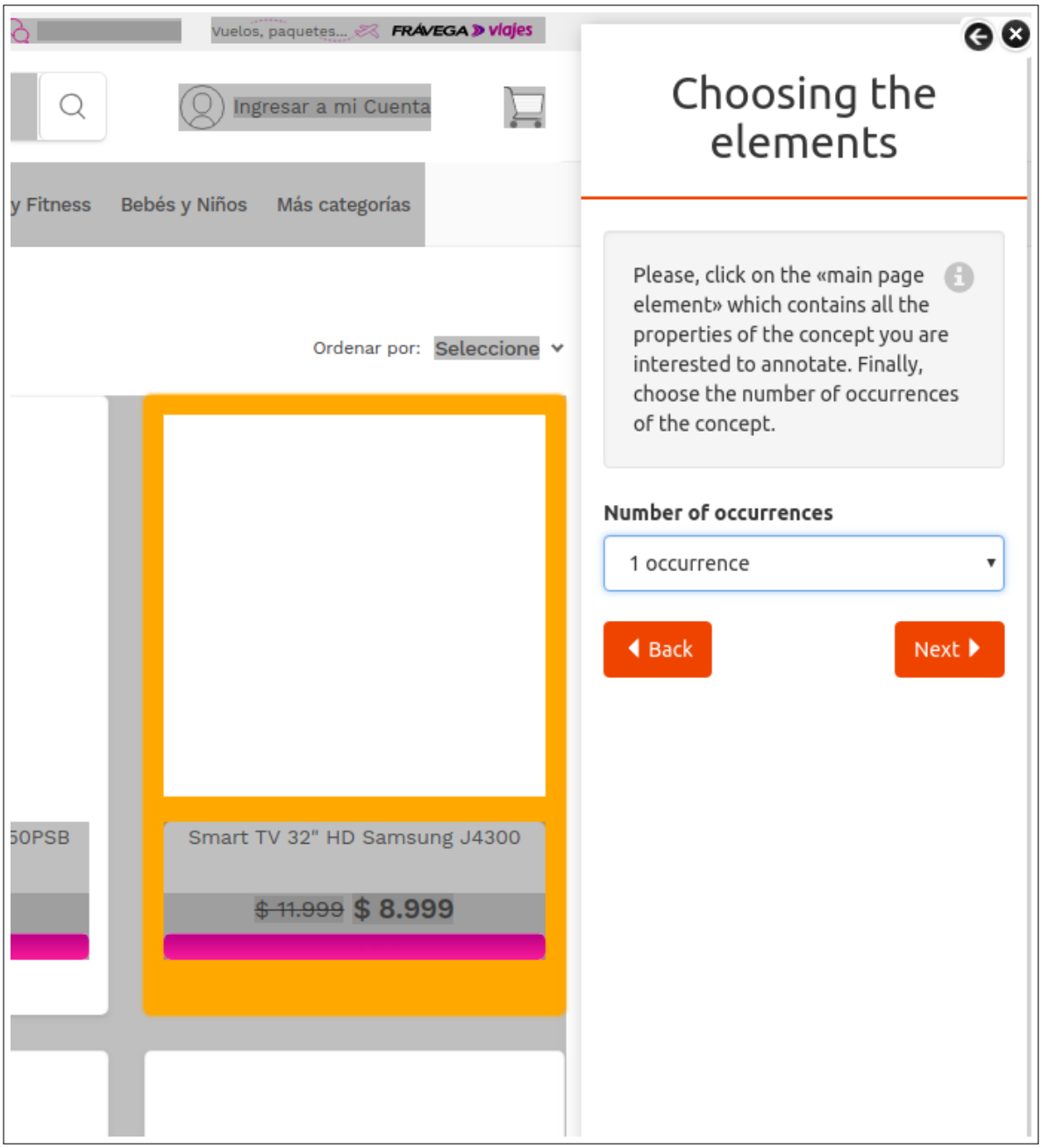


Fig. 5.4 Selección del elemento a aumentar

En la Figura 5.6 podemos ver el mensaje que nos muestra la extensión avisándonos que el template fue creado exitosamente y la información ya fue enviada al servidor (Figura 5.6).

Esta información fue recibida desde el lado del servidor por SSWMFA y mapeada a una tabla del modelo conceptual.

5.2 Modelando el comportamiento server side

Una vez recibida la información del elemento a aumentar podemos comenzar con la creación de los modelos conceptual y de navegación.

5.2.1 Creación el modelo conceptual

Como se explicó al final de la sección anterior, la información del elemento a aumentar (nombre y propiedades) fueron enviadas a SSWMFA a través del endpoint que éste provee y mapeado a una tabla del modelo conceptual.

Para visualizar esto abrimos SSWMFA y nos dirigimos a la pestaña del modelo conceptual.

En el board no se verá nada, ya que la información se encuentra en formato JSON esperando para ser cargada al diagrama:

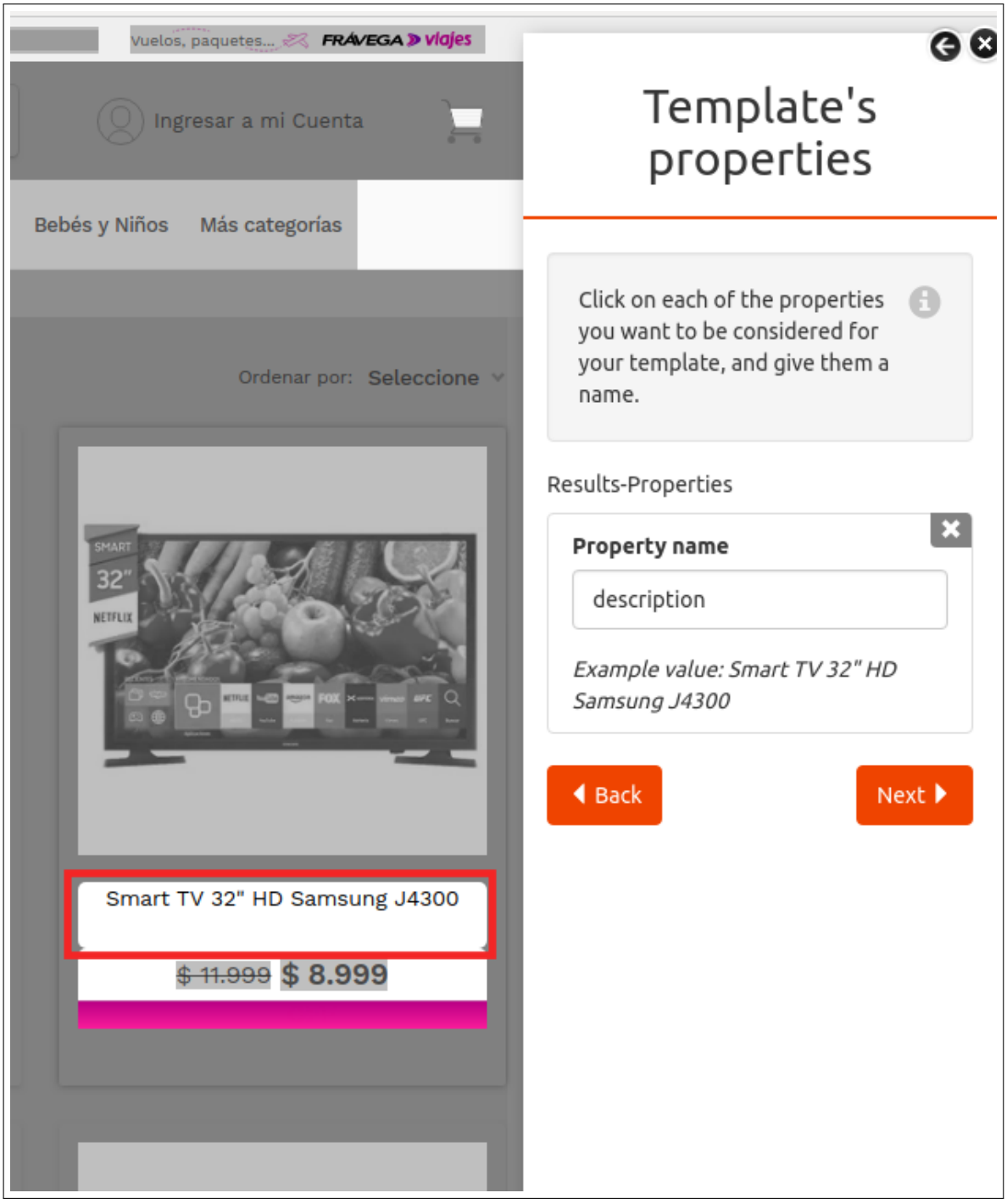


Fig. 5.5 Selección de las propiedades del elemento

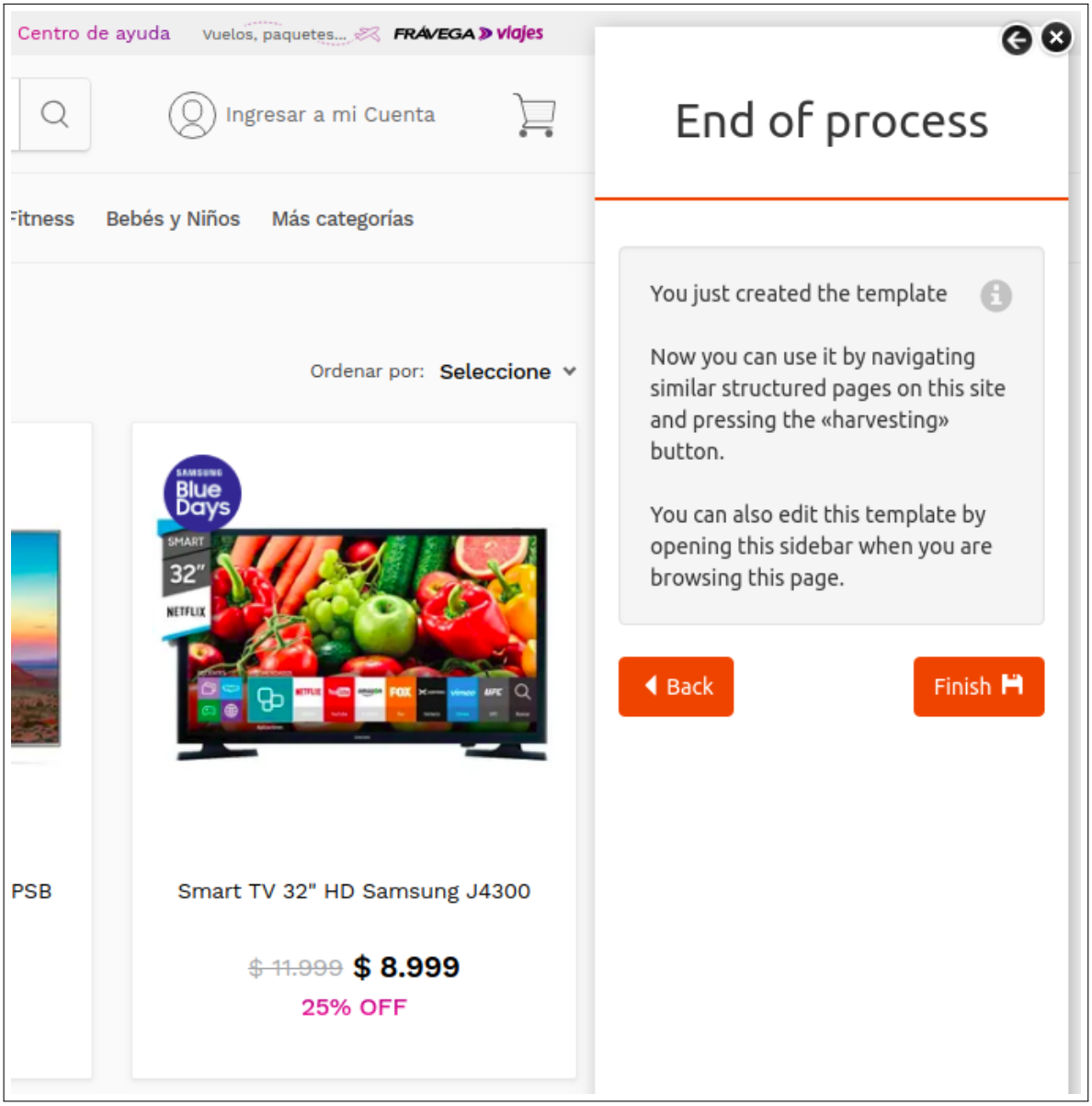


Fig. 5.6 Información enviada al server-side

```
{ "class": "go.GraphLinksModel",
    "copiesArrays": true,
    "copiesArrayObjects": true,
    "modelData": {"position":"-5 -5"},
    "nodeDataArray": [
    {"name":"television", "properties":[
        {"name":"description",
        "type":"string",
        "unique":false,
        "nullable":false} ],
    "key":-1,
    "loc":"330 80"}
    ],
    "linkDataArray": []}
```

Los pasos a seguir son, como se muestra en la Figura 5.7, primero cargar el JSON al diagrama, luego crear la Base de Datos y finalmente crear la tabla correspondiente.

Podemos observar en el archivo `db.js` que se creó la tabla `television` con el atributo `description`:

```
const television = sequelize.define('television', {
 id: { type: Sequelize.INTEGER,
        autoIncrement: true,
        primaryKey: true },
  description: { type: Sequelize.STRING, allowNull: false }
});
```

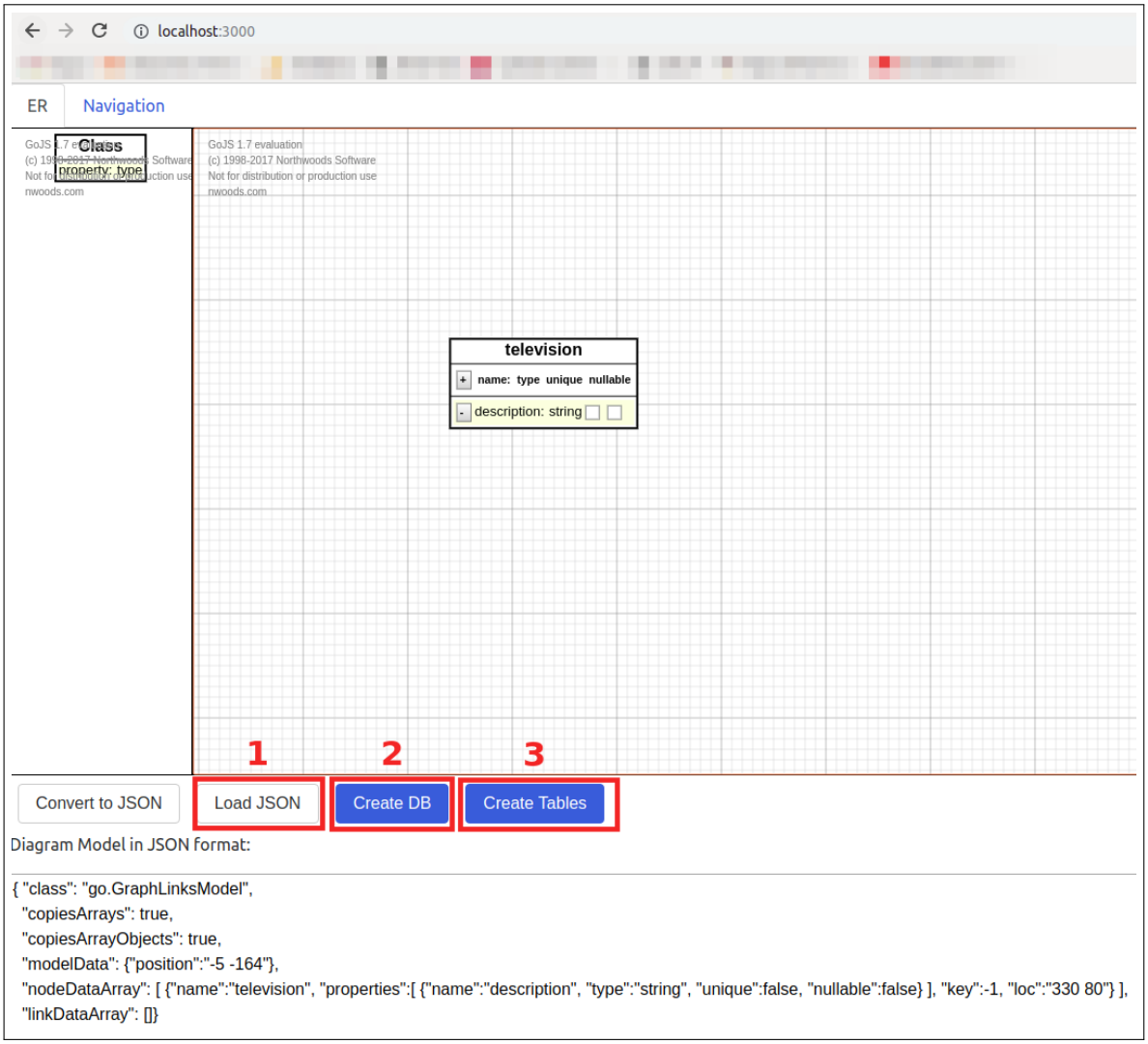


Fig. 5.7 Modelo conceptual

5.2.2 Creación el modelo de navegación

Para el ejemplo crearemos una página básica que contenga solamente un formulario para creación de nuevos elementos y un listado que permita ver, editar, y eliminar elementos de la base de datos.

Para esto nos dirigimos a la pestaña **Navigation** de SSWMFA y arrastramos un objeto **Form** y un objeto **List** dentro de la página **index**.

También es posible (y recomendable) nombrar tanto los objetos como la página para hacer la vista más amigable al usuario.

Como se muestra en la Figura 5.8 los pasos a seguir, luego de modelar el diagrama, son: convertir el diagrama a formato JSON para poder exportarlo; crear las rutas que necesita SSWMFA internamente (Figura 5.9); crear la página modelada y las páginas derivadas, como las de edición y detalle de los elementos (Figura 5.10); y finalmente renderizar y abrir la página de inicio creada en el modelo.

5.2.3 Renderizado de la página y manipulación de los datos

Finalmente podemos acceder a nuestro administrador de la información del elemento a aumentar. En la Figura 5.11 podemos observar el formulario para crear nuevas descripciones y el listado de las descripciones extraídas desde el lado del cliente utilizando la extensión MDWA Collector.

5.2.4 Conclusiones

En este ejemplo vimos cómo extraer informacón de un elemento de un sitio web a aumentar del lado del cliente utilizando la extensión MDWA.

Como esta extensión es un trabajo que no está terminado en la actualidad y no tiene implementado la funcionalidad de leer la información generada server-side por SSWMFA, tal como podría ser una descripción editada, o insertar una nueva propiedad `comentarios` que permita a usuarios finales ingresar sus opiniones sobre los televisores.

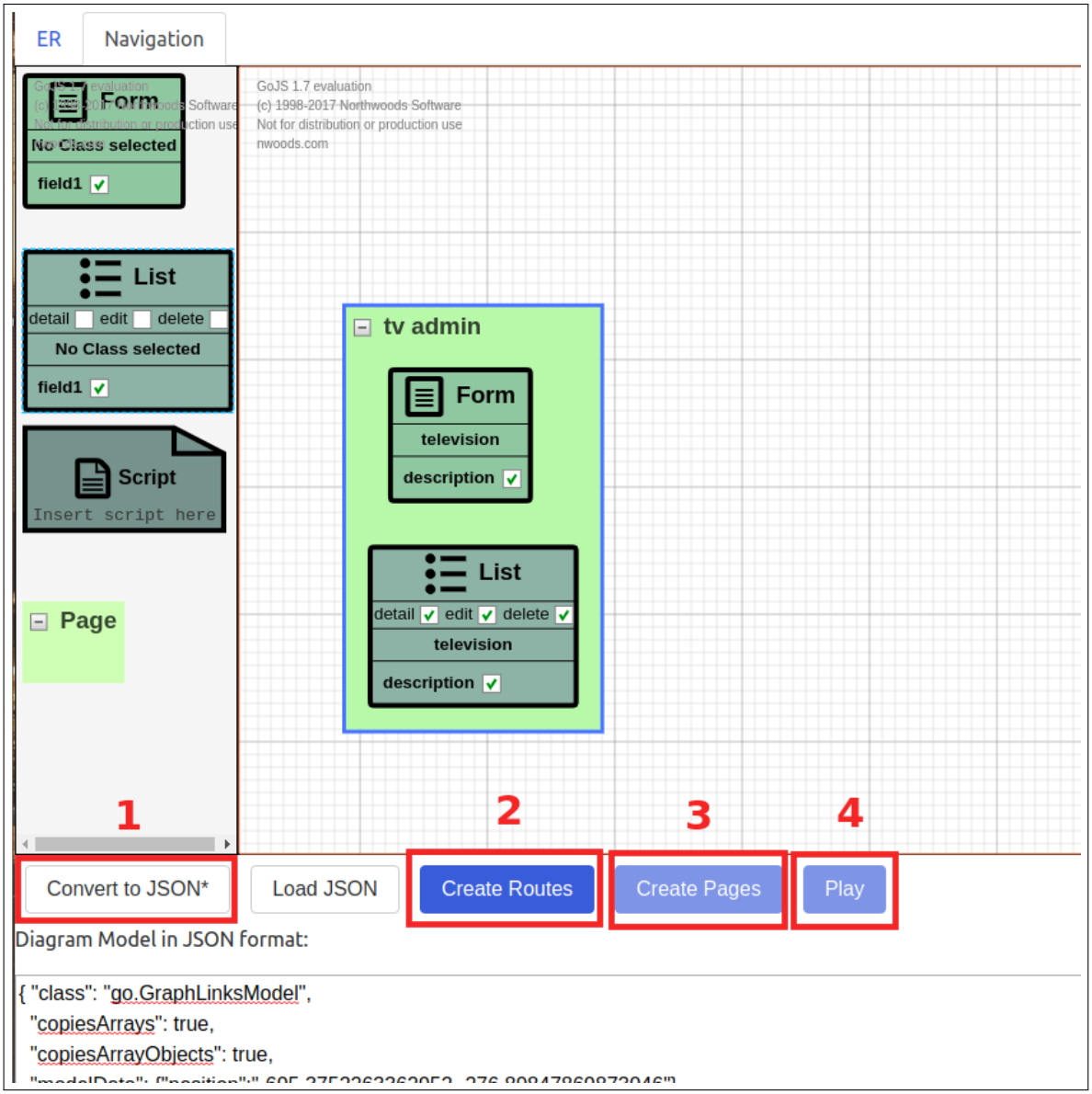


Fig. 5.8 Modelo de navegación

render.js

```
var express = require('express');
var router = express.Router();
var sequelize = require('../models/db');
//below autogenerated routes
router.get('/', function (req, res) {
  sequelize.models.television.findAll().then(recordsFromtelevision => {
   res.render('rendered/tv_admin', { title: 'tv admin',   recordsFromtelevision : recordsFromtelevision });
  });
});

router.post("/insertOneIntelevision", function(req, res){
  sequelize.models.television.create({
    description: req.body.description
  })
  .then(()=> {res.redirect('back')})
  .catch(error => {console.log(error);res.redirect('back')});
});

router.get("/updateOneIntelevision/:id", function(req, res){
  sequelize.models.television.findById(req.params.id).then(record => {
    res.render("rendered/formUpdateOneIntelevision", { title: "Update in television", record : record })
  });
});

router.post("/updateOneIntelevision", function(req, res){
  sequelize.models.television.findById(req.body.id).then(record => {
    record.description = req.body.description;
    record.save()
    .then(()=> {res.render("rendered/formUpdateOneIntelevision", { title: "Update in television", record : record })})
    .catch(error => {console.log(error);res.redirect('back')});
  });
});

router.get("/deleteOneFromtelevision/:id", function(req, res){
  sequelize.models.television.findById(req.params.id).then(record => {
    record.destroy().then(()=> {res.redirect('back')})
  });
});

router.get("/detailOfOneFromtelevision/:id", function(req, res){
  sequelize.models.television.findById(req.params.id).then(record => {
    res.render("rendered/detailFromtelevision", { title: "Detail of record from television", record : record });
  });
});


module.exports = router;
```

Fig. 5.9 Archivo render.js con las rutas internas

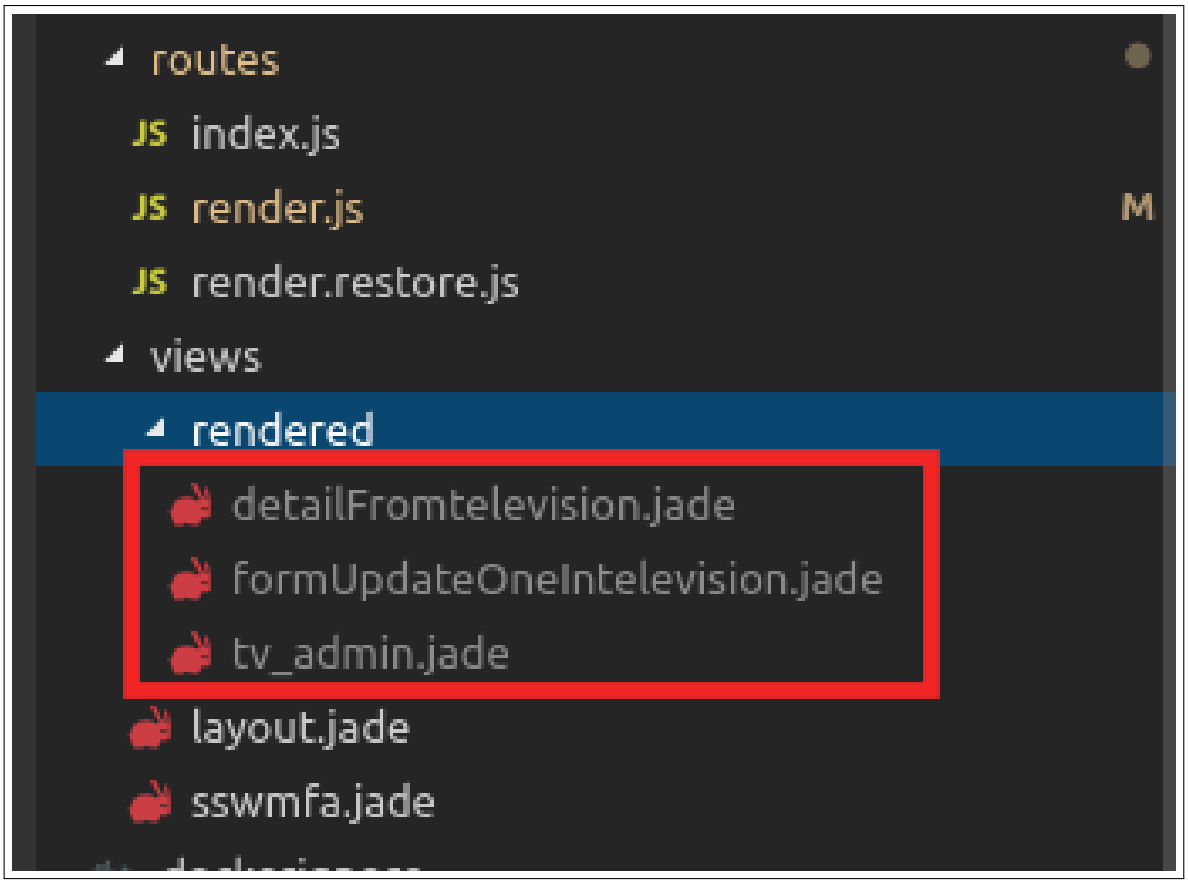


Fig. 5.10 Páginas creadas a partir del modelo de navegación

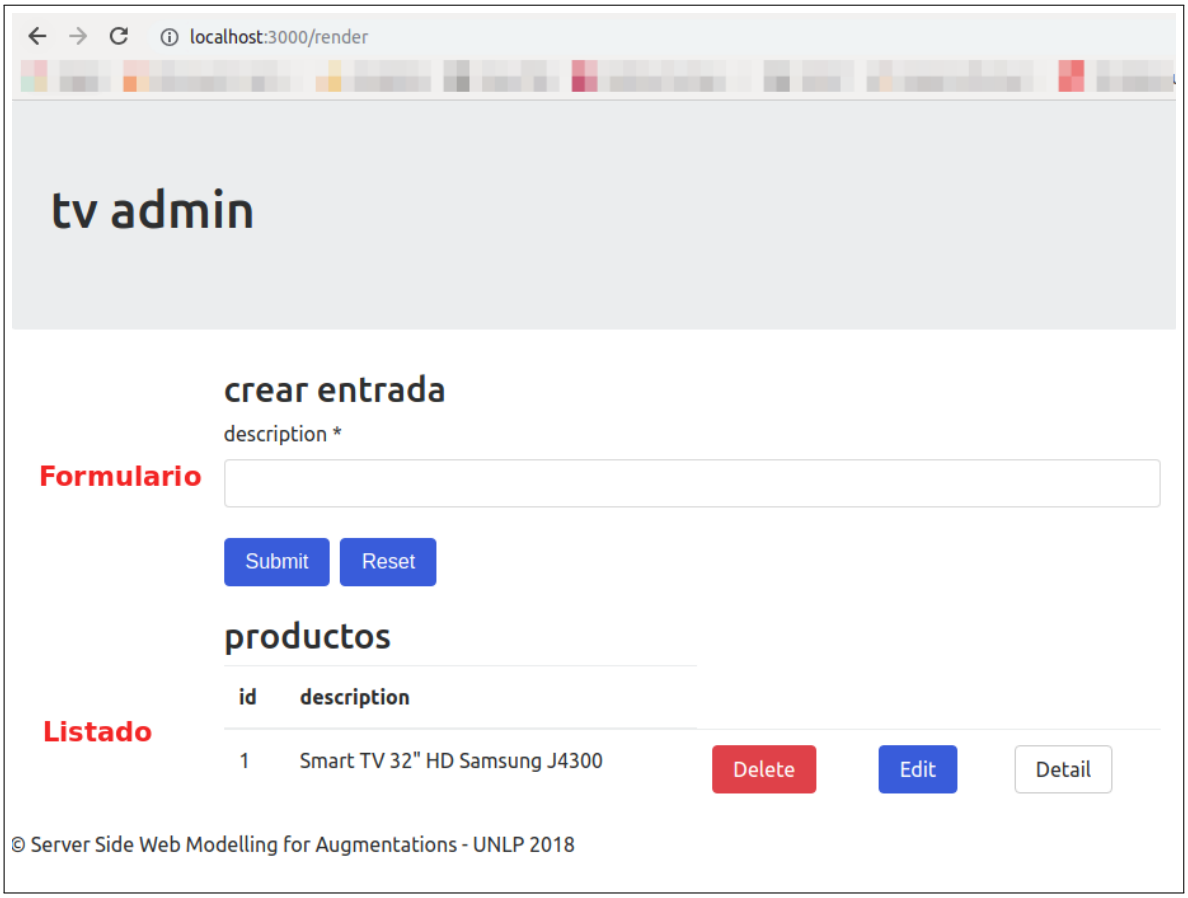


Fig. 5.11 Página de administración de las descripciones de los elementos televisión

A lo que concierne al modelado del comportamiento server-side de aumentaciones, SSWMFA provee todo lo necesario a las aplicaciones client-side para extraer, persistir y modificar los elementos a aumentar de alguna aplicación.

6. Discusión técnica

Al momento de idear una herramienta para cumplir con estos objetivos se pensó en un enfoque web para abstraer a los desarrolladores del sistema operativo y lograr una mejor portabilidad de los artefactos.

El lenguaje elegido para desarrollar SSWMFA fue JavaScript con el framework NodeJS [1] y el ORM Sequelize [2]. La librería elegida para el desarrollo de los diagramas fue GoJS [3].

6.0.1 Elección del Lenguaje

A la hora de elegir un lenguaje la primera aproximación fue **Python** con el framework **Django** [13] ya que cuento con experiencia en esa tecnología desarrollando diversas aplicaciones y sistemas web, pero fue rápidamente descartado porque SSWMFA debe ser una aplicación muy dinámica donde se permita crear un sistema web desde cero a partir de los diagramas conceptual y de navegación que pueden ser editados por el desarollador y estos cambios deben ser reflejados en tiempo real en el sistema web renderizado. Esto si bien es posible de hacer con Python, la complejidad sería muy alta ya que Django espera que, al menos, el modelo de la base de datos esté especificado a la hora de comenzar con el proyecto. Para SSWMFA esto no es posible porque la Base de Datos cambiará a medida que se actualice el modelo conceptual.

La segunda aproximación, por experiencia personal con el lenguaje, fue JavaScript que cuenta con varios frameworks y una comunidad de desarrolladores muy grande en todo el mundo.

[1] https://nodejs.org/es/
[2] http://docs.sequelizejs.com/
[3] https://gojs.net/latest/index.html

Luego de especificado el alcance de SSWMFA se decidió realizar tanto el backend como el frontend en este lenguaje aprovechando que provee la posiblidad de un desarrollo `full stack` [20] lo que hace que la interacción entre la vista, el controlador y el modelo sea fácil de implementar, robusta y performante.

El framework elegido fue **Node JS** [27] por su popularidad y la completa documentación que posee. Node JS cuenta un template engine [28] por defecto muy simple y rápido de aprender: **Jade** [4].

Con Node JS para el controlador y Jade para la vista, solo quedaba elegir el ORM (mapeo objeto-relaciona) [19] para la base de datos.

Como se dijo anteriormente, la base de datos debe ser creada y modificada dinámicamente en tiempo de ejecución, por lo que debe ser posible destruirla y generarla nuevamente de forma rápida y segura, además debe poder soportar distitos motores de bases de datos como mysql, sqlite o postgres para brindar más herramientas a los desarrolladores de artefactos de aumentaciones web.

Sequelize [5] fue el ORM elegido por su simpleza, su fácil integración a un proyecto Node JS, la facilidad de crear cosultas genéricas y abstraídas al motor de base de datos elegido, y por la posibilidad de destruir y crear una base de datos sobre la marcha (on the fly).

6.0.2 Elección de la librería de diagramado

Existen varias librerías JavaScript de código abierto que pueden utilizarse para realizar diagramas del tipo que se necesitan para los modelos conceptual y de navegación.

A continuación se listan las que fueron evaluadas con su resumen y, finalmente, la razón de por qué se eligió GoJS.

[4] http://jade-lang.com/
[5] http://docs.sequelizejs.com/

6.0.2.1 mxGraph

mxGraph [6] es una librería JavaScript de diagramado en el lado del cliente (client-side) que utiliza SVG (Gráficos vectoriales escalables) y HTML para renderizar los diagramas.

El paquete mxGraph contiene un software cliente escrito en JavaScript y una serie de backends en distintos lenguajes: .NET, Java y PHP. El software cliente es un componente graph con una aplicación wrapper opcional que está integrada en una interface web existente. El cliente puede requerir un webserver que le envíe los archivos requeridos o puede ser ejecutado directamente en el filesystem local.

El backend puede ser utilizado como es o puede ser embebido en una aplicación servidora desarrollada en uno de los tres lenguajes soportados.

Si el backend existe, entonces el cliente puede ser configurado para ser usado para, por ejemplo, crear imágenes, importar y exportar diagramas o crear una representación de un graph.

Estos escenarios pueden combinarse de distintas maneras, como por ejemplo enviar un archivo XML con cada cambio al backend en tiempo real o auto guardar el diagrama para evitar pérdida de información en el cliente.

mxGraph provee la posibilidad de que el cliente opere en modo offline donde no se requiere un backed o un webserver, simplemente corre en el navegador permitiendo crear gráficos pero sin la posibilidad de crear archivos o guardar cambios.

Si bien esta solución es bastante robusta y viene bien integrada, fue descartada debido a, fundamentalmente, dos motivos.

Primero, la forma recomendada (por lo tanto documentada) por la librería es realizar el desarrollo con un web server y backend en Java, .Net o PHP. Como se explicará más adelante el lenguaje utilizado para desarrollar SSWMFA, tanto para frontend como backed, es JavaScript y si bien hay formas de realizarlo con este lenguaje utilizando mxGraph, la implementación sería muy engorrosa y no existe buena documentación en caso de tener dudas o errores.

[6] https://jgraph.github.io/mxgraph/docs/tutorial.html

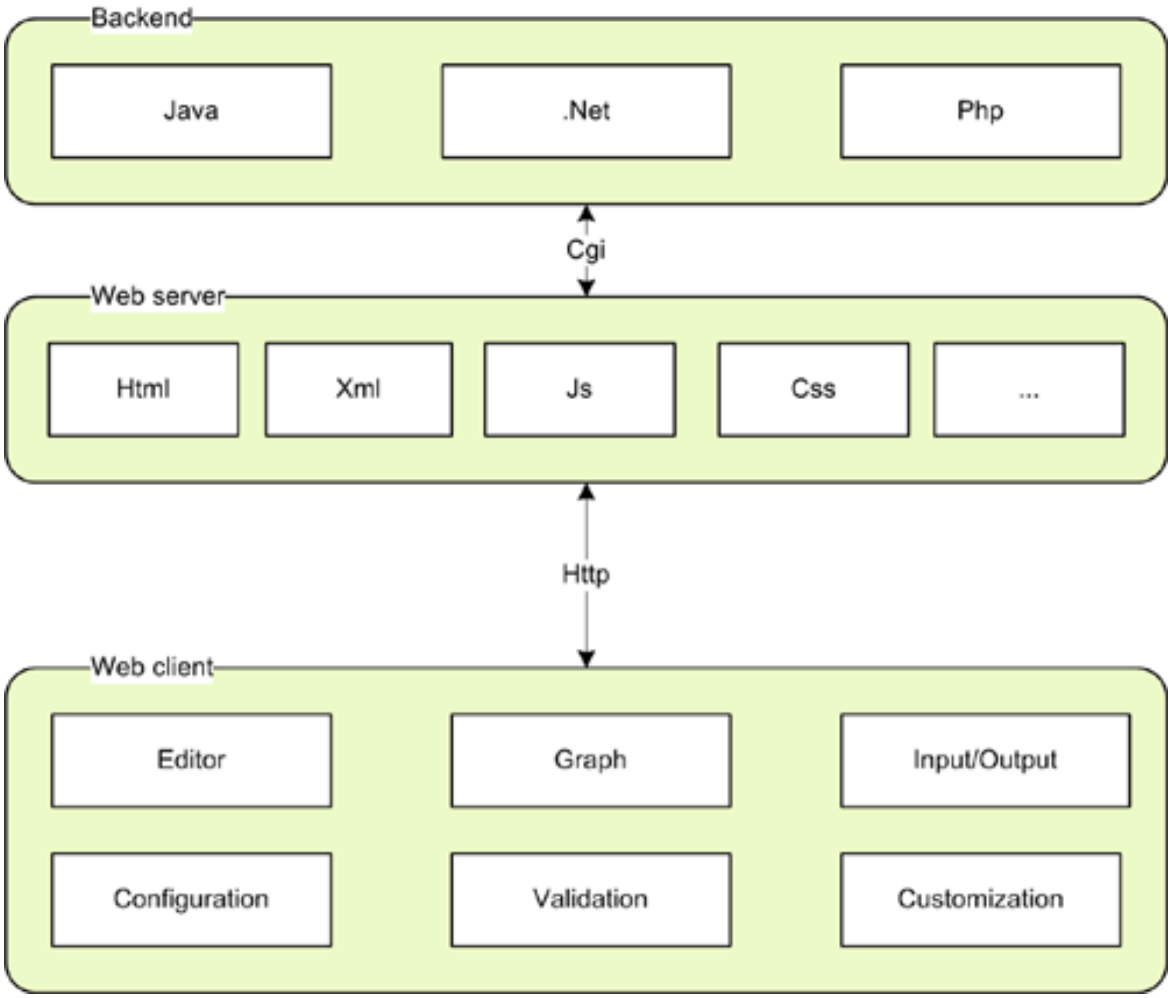


Fig. 6.1 Arquitectura de mxGraph

Segundo, es una librería demasiado grande para el alcance de este trabajo y solo se haría uso de una porción muy reducida de ésta, por ejemplo se ha utilizado para desarrollar aplicaciones complejas y de propósito muy amplio como draw.io [7] que permite realizar muchos tipos de diagramas.

[7] https://www.draw.io/

6.0.2.2 Raphaël

Raphaël [8] es una pequeña librería JavaScript que simplifica el trabajo con gráficos vectoriales en la web. Se puede utilizar para crear gráficos personalizados o recortar y rotar imágenes.

Raphaël utiliza la recomendación de la W3C para SVG [9] (Gráficos vectoriales escalables) y VML (Vector Markup Language - formato de archivos basado en XML para gráficos vectoriales de dos dimensiones) como base para crear gráficos. Esto significa que cada objeto gráfico creado es, además, un objeto DOM, por lo que es posible asignar manejadores de eventos JavaScript y luego modificarlos.

El objetivo de esta librería es proveer un adaptador que permita dibujar arte vectorial fácilmente y en distintos navegadores.

En caso de desarrollar una herramienta de diagramado desde cero esta librería sería muy útil para dibujar los elementos del diagrama y agregar funcionalidad a cada uno, así como a las relaciones y agrupamientos de estos elementos.

Ejemplo de uso:

```
// Creates canvas 320 × 200 at 10, 50
var paper = Raphael(10, 50, 320, 200);

// Creates circle at x = 50, y = 40, with radius 10
var circle = paper.circle(50, 40, 10);
// Sets the fill attribute of the circle to red (#f00)
circle.attr("fill", "#f00");

// Sets the stroke attribute of the circle to white
circle.attr("stroke", "#fff");
```

Fig. 6.2 Raphaël - Ejemplo de uso

[8] http://dmitrybaranovskiy.github.io/raphael/
[9] https://www.w3.org/TR/SVG11/

6.0.2.3 D3JS

D3.js [10] es una librería JavaScript para manipular documentos basados en datos permitiendo representarlos gráficamente utilizando HTML, SVG y CSS (Hoja de estilos en cascada).

El énfasis de D3 en los estándares web permite utilizar al máximo las capacidades de los navegadores modernos evitando depender de un framework propietario, combinando componentes de visualización poderosos y el enfoque basado en datos para manipular el DOM (Document Object Model).

D3 permite relacionar datos arbitrarios a un DOM y luego aplicar transformaciones basados en los datos al documento, estas transformaciones pueden ser en respuesta a interacciones con el usuario, animaciones en el tiempo, o incluso por notificaciones asíncronas recibidas desde una fuente externa. Por ejemplo se puede utilizar D3 para generar una tabla HTML a partir de un array de números. O usar los mismos datos para crear un gráfico SVG interactivo.

Un ejemplo de un gráfico realizado con D3 es el que se muestra en la Figura 6.3, que detalla el porcentaje de nativos en cada región de Alaska, la información que se muestra a la izquierda es de la zona apuntada con el cursor, donde se puede observar un segundo gráfico que se actualiza en tiempo real cuando se mueve el cursor apuntando a otra región [11].

6.0.2.4 GoJS

GoJS [12] es una libería JavaScript que permite desarrollar diagramas interactivos y visualizaciones complejas en distintos navegadores web y plataformas.

Esta librería ofrece muchas funcionalidades para la interacción del usuario como drag-and-drop (arrastrar y soltar), copiar y pegar, edición de texto en el lugar, tooltips,

[10] https://d3js.org/

[11] https://www.theguardian.com/environment/interactive/2013/may/14/alaska-villages-frontline-global-warming

[12] https://gojs.net

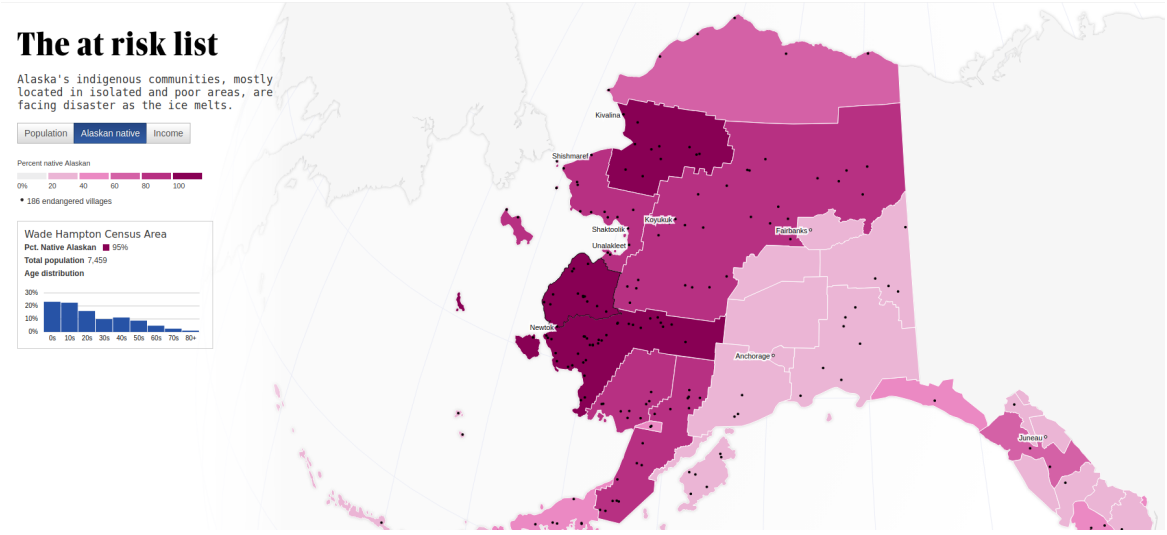


Fig. 6.3 D3JS - Ejemplo de uso

plantillas, modelos relacionados con datos, paletas, vista previa, manejadores de eventos, la posibilidad de representar los diagramas en formato JSON, entre otros.

GoJs es una librería JavaScript pura, por lo que los usuarios pueden interactuar con ella sin necesidad de que la aplicación se conecte a un servidor ni de instalar plugins adicionales al navegador. Corre completamente en el navegador web y renderiza un Canvas de HTML5 o imagenes SVG sin necesidad de requerimientos server-side.

El hecho de que esta librería no requiera de un back-end y sea fácil de implementar en cualquier framework (incluso sin uno) hizo que fuera la elegida para desarrollar SSWMFA.

Otra gran ventaja es la posibilidad de representar los diagramas en formato JSON lo que permite crear fácilmente interfaces para la comunicacón con herramientas externas, para esto solo se necesita informarles a las herramientas el tipo de objeto JSON esperado, una vez recibido, transformarlo y representarlo en forma gráfica.

Las ventajas por sobre las otras librerías evaluadas son:

- mxGraph: GoJS es más liviana y simple. No requiere implementar un servicio backend.

- Raphaël: GoJS ya contiene una librería para realizar gráficos vectoriales por lo que no es necesario utilizar Raphaël.

- D3JS: GoJS ya contiene una librería para realizar gráficos vectoriales. Además D3 tiene una finalidad más orientada a representar datos (que pueden ir variando o no) en forma gráfica y no tanto a los diagramas UML como son los de flujo o Entidad Relación que son los que necesitaríamos para realizar los diagramas conceptual y de navegación.

6.0.3 Tecnología complementaria

Se decidió incluír Docker [21] Compose [13] para lograr un nivel más de abstracción para los desarrolladores. Docker provee la posibilidad de correr el código de SSWMFA de forma local sin necesidad de contar con las versiones correspondientes de JavaScript, Node, y otros paquetes necesarios ya que las instalaciones requeridas se realizan en un container de Docker y allí corre el códgo de la herramienta.

6.0.4 Código de la herramienta finalizada

Una vez finalizado el desarrollo de SSWMFA el código fue subido a GitHub [14] con las instrucciones de cómo crear el ambiente de forma local y cómo correr la aplicación.

[13] https://docs.docker.com/compose/
[14] https://github.com/francomahl/sswmfa

7. Trabajos futuros

Como se especificó durante el documento, SSWMFA es una herramienta para implementar el comportamiento server-side (backend) de herramientas de aumentación de sitios web.

En el ejemplo comprensivo de la sección 5 se presentó una caso de implementación de la interfaz entre la herramienta de aumentación y SSWMFA para implementar el comportamiendo del lado del servidor pero del lado client-side la herramienta no está completa, por lo que queda pendiente probar el último paso que es editar y agregar comentarios desde SSWMFA (server-side). Un trabajo futuro sería implementar esta interfaz con el comportamiento client-side (MDWA Collector) finalizado.

Una vez implementada una solución completa client-side y server-side pueden realizarse pruebas con usuarios finales.

7.1 Mejoras en la herramienta

SSWMFA presenta una primera aproximación a lo que es la implementación del comportamiento server-side de herramientas de aumentación web, y como primera aproximación puede ser mejorada en varios aspectos.

Mientras fui desarrollando esta herramienta me surgieron ideas para mejoras futuras que paso a listar a continuacipon.

- Comunicación con el usuario: Actualmente los mensajes de éxito o error son mostrados en la consola del navegador, por ejemplo, al crear la base de datos, tablas, rutas, páginas, etc. Una mejora sería implementar mensajes claros en la interfaz gráfica que comuniquen claramente qué está sucediendo en el backend.

- Manejo de errores: Si bien las llamadas a los endpoints devuelven en sus respuestas si el request fue exitoso o si falló, lo ideal sería mejorar las respuestas especificando claramente qué acaba de suceder al realizar el request.

- Relaciones y herencia en el diagrama Entidad/Relación: actualmente no está implementada la funcionalidad de relación entre entidades y debe realizarse manualmente especificando un campo con el nombre de la entidad a la cual realizar una clave foránea del tipo `integer` y hacer la relación insertando manualmente el `id` del objeto a relacionar. Debería poder representarse una relación entre dos entidades en los formularios y listados. Igualmente con las herencias entre Clases.

- Persistir los diagramas en una base de datos.

- Actualmente solo puede haber un formulario o listado de la misma clase en una misma página: Permitir que pueda haber múltiples formularios/listados de una misma clase dentro de una misma página.

- Funcionalidades en los diagramas: GoJS permite cada vez más funcionalidades para implementar en los diagramas. Por ejemplo:

  - Drag and drop de fields para cambiar el orden de los atributos de una Entidad o de los campos en un Form o List [1].
  - Minimapa del diagrama [2].
  - Exportar diagrama en formato SVG [3].
  - Cargar diagramas en formato XML [4] (actualmente se realiza en JSON).

  Para más información referirse al listado de todos los ejemplos de las funcionalidades que brinda GoJS [5].

[1] https://gojs.net/latest/samples/dragDropFields.html
[2] https://gojs.net/latest/samples/network.html
[3] https://gojs.net/latest/samples/minimalSVG.html
[4] https://gojs.net/latest/samples/minimalXML.html.
[5] https://gojs.net/latest/samples/all.html

8. Conclusión

Este trabajo se basa en el enfoque para diseñar aumentaciones web basada en modelos (MDWE) con comportamientos del lado del cliente y del lado del servidor. Para implementar el comportamiendo del lado del servidor se presenta en este trabajo la herramienta Server Side Web Modeling for Augmentations (SSWMFA)

Actualmente los aumentos se modelan utilizando IFML o con otros enfoques [26] como UWE u OOHDM, pero esto puede traer muchos problemas como se explica en la Motivación de este trabajo 1.

El enfoque MDWE se basa en la separación de los principios de los conceptos. Por lo tanto, se proporciona el mecanismo de composición para cada modelo (conceptual, navegacional y la interfaz de usuario), las formas de implementarlos del lado del servidor y distintas formas de desarrollar la interfaz con el lado del cliente.

Para describir las posibilidades del enfoque MDWE, se describió un ejemplo que respalda algunas de las necesidades y complejidades de los usuarios en un sitio de comercio electrónico agrícola haciendo una simulación del desarrollo del lado del cliente, simulando una interfaz con el backend y mostrando las funcionalidades de SSWMFA para el desarrollo del lado del servidor. El objetivo de este ejemplo era mejorar las actividades de toma de decisiones en el sitio extrayendo las entidades que pueden interesar y aumentándolas con distintas funcionalidades, en el ejemplo, agregando comentarios a productos, y persisitiendo estas aumentaciones en una base de datos.

Este trabajo presenta una alternativa a los enfoques de desarrollo del comportamiento server-side de aumentaciones que se vienen utilizando en enfoques como MDWE. Se mostró cómo SSWMFA es más simple y de propósito más específico para aumentaciones que, por ejemplo, WebML detallando que es más fácil de aprender y permite una personalización de las funcionalidades de forma sencilla para programadores

con conocimientos básicos de JavaScript, NodeJS y el concepto de API Rest. Sin embargo, un estudio completo debería comprender un experimento controlado donde evaluar el enfoque con un conjunto más amplio de sujetos y así analizar la experiencia de los usuarios.

Referencias

[1] Aragón, G., Escalona, M. J., Lang, M., and Hilera, J. R. (2013). An analysis of model-driven web engineering methodologies.

[2] Brambilla, M. and Fraternali, P. (2014). *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. Morgan Kaufmann, Burlington, Massachusetts,USA.

[3] Bray, T. (2017). The javascript object notation (json) data interchange format.

[4] Ceri, S., Dolog, P., Matera, M., and Nejdl, W. (2004). Model-Driven Design of Web Applications with Client-Side Adaptation. In Koch, N., Fraternali, P., and Wirsing, M., editors, *Web Engineering: 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004. Proceedings*, pages 201–214. Springer Berlin Heidelberg, Berlin, Heidelberg.

[5] Cerny, T., Macik, M., Donahoo, M. J., and Janousek, J. (2015). On distributed concern delivery in user interface design. *Computer Science and Information Systems*, 12(2):655–681.

[6] Chen, P. P.-S. (1988). The entity-relationship model—toward a unified view of data. In *Readings in artificial intelligence and databases*, pages 98–111. Elsevier.

[7] Crowther, R., Lennon, J., Blue, A., Wanish, G., and Heilmann, C. (2014). *HTML5 in Action*. Manning.

[8] Díaz, O. and Arellano, C. (2015). The Augmented Web: Rationales, Opportunities, and Challenges on Browser-Side Transcoding. *TWEB*, 9(2):8.

[9] Domínguez-Mayo, F. J., Escalona, M. J., Mejías, M., Ross, M., and Staples, G. (2014). Towards a Homogeneous Characterization of the Model-driven Web Development Methodologies. *J. Web Eng.*, 13(1&2):129–159.

[10] Firmenich, D., Firmenich, S., Rivero, J. M., Antonelli, L., and Rossi, G. (2016a). CrowdMock: an approach for defining and evolving web augmentation requirements. *Requirements Engineering*, pages 1–29.

[11] Firmenich, S., Bosetti, G., Rossi, G., and Winckler, M. (2016b). Flexible Distribution of Existing Web Interfaces: An Architecture Involving Developers and End-Users. In *Current Trends in Web Engineering - {ICWE} 2016 International Workshops, DUI, TELERISE, SoWeMine, and Liquid Web, Lugano, Switzerland, June 6-9, 2016, Revised Selected Papers*, pages 200–207.

[12] Firmenich, S., Bosetti, G. A., Rossi, G., Winckler, M., and Barbieri, T. (2016c). Abstracting and Structuring Web Contents for Supporting Personal Web Experiences. In *Web Engineering - 16th International Conference, {ICWE} 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, pages 77–95.

[13] Forcier, J., Bissex, P., and Chun, W. J. (2008). *Python web development with Django*. Addison-Wesley Professional.

[14] Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, Boston, MA, USA.

[15] Frajberg, D., Urbieta, M., Rossi, G., and Schwinger, W. (2016). Volatile Functionality in Action: Methods, Techniques and Assessment. In Bozzon, A., Cudre-Maroux, P., and Pautasso, C., editors, *Web Engineering: 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, pages 59–76. Springer International Publishing, Cham.

[16] Gaedke, M. and Gräf, G. (2001). Development and Evolution of Web-Applications Using the WebComposition Process Model. In *Web Engineering*, volume 2016, pages 58–76.

[17] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[18] Herlocker, J. L., Konstan, J. A., and Riedl, J. (2000). Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work - CSCW '00*, pages 241–250.

[19] Keith, M. and Schnicariol, M. (2009). Object-relational mapping. In *Pro JPA 2*, pages 69–106. Springer.

[20] Mardan, A. (2015). *Full Stack JavaScript*. Springer.

[21] Nickoloff, J. (2016). *Docker in action*. Manning Publications Co.

[22] Niederhausen, M., Sluijs, K., Hidders, J., Leonardi, E., Houben, G.-J., Meißner, K., Van Der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.-J., and Meißner, K. (2009). Harnessing the power of semantics-based, aspect-oriented adaptation for AMACONT. In Gaedke, M., Grossniklaus, M., and Díaz, O., editors, *Web Engineering*, chapter Harnessing, pages 106–120. Springer, Berlin, Heidelberg.

[23] Orlikowski, W. J. (1993). Case tools as organizational change: Investigating incremental and radical changes in systems development. *MIS quarterly*, pages 309–340.

[24] Popovici, A., Gross, T., and Alonso, G. (2002). Dynamic Weaving for Aspect-oriented Programming. In *Proceedings of the 1st International Conference on Aspect-oriented Software Development*, AOSD '02, pages 141–147, New York, NY, USA. ACM.

[25] Rodríguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J. C., Canali, L., and Percannella, G. (2016). Rest apis: a large-scale analysis of compliance with principles and best practices. In *International Conference on Web Engineering*, pages 21–39. Springer.

[26] Rossi, G., Pastor, O., Schwabe, D., and Olsina, L. (2008). *Web Engineering: Modelling and Implementing Web Applications*, volume 12. Springer-Verlag London.

[27] Tilkov, S. and Vinoski, S. (2010). Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83.

[28] Tso, M. M.-H. (2000). Context-sensitive template engine. US Patent 6,085,201.

[29] Urbieta, M., Firmenich, S., Maglione, P., Rossi, G., and Olivero, M. A. (2017). A Model-driven Approach for Empowering Advance Web Augmentation - From Client-side to Server-side Support. In *APMDWE*. INSTICC, ScitePress.

[30] Urbieta, M., Oliveira, A., Araújo, J., Rodrigues, A., Moreira, A., Gordillo, S., and Rossi, G. (2014). Web-GIS models: accomplishing modularity with aspects. *Innovations in Systems and Software Engineering*, 10(1):59–75.

[31] Urbieta, M., Retschitzegger, W., Rossi, G., Schwinger, W., Gordillo, S., and Luna, E. R. (2012a). Modelling adaptations requirements in web workflows. *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services - IIWAS '12*, page 72.

[32] Urbieta, M., Rossi, G., Distante, D., and Ginzburg, J. (2012b). Modeling, Deploying, and Controlling Volatile Functionalities in Web Applications. *International Journal of Software Engineering and Knowledge Engineering*, 22:129–155.

[33] Vilain, P., Schwabe, D., and de Souza, C. S. (2000). A Diagrammatic Tool for Representing User Interaction in UML. In Evans, A., Kent, S., and Selic, B., editors, *International Conference on the Unified Modeling Language*, volume 1939 of *Lecture Notes in Computer Science*, pages 133–147. Springer.

[34] WebRatio (2013). Setting Up WebRatio Platform. URL https://my.webratio.com/learn/learningobject/setting-up-webratio-platform-v-72.

[35] WebRatio (2017). WebRatio Platform. URL http://www.webratio.com/site/content/en/web-application-development.

[36] Wischenbart, M., Firmenich, S., Rossi, G., and Wimmer, M. (2015). Recommender Systems for the People - Enhancing Personalization in Web Augmentation. In *Proceedings of the Joint Workshop on Interfaces and Human Decision Making for Recommender Systems, IntRS 2015, co-located with {ACM} Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 19, 2015.*, pages 53–60.