Máster Universitario de Investigación en Ingeniería de Sotfware y Sistemas Informáticos

Itinerario de Ingeniería de Software (Código 31105128)

GenMovil: Aplicación de WebDSL para crear aplicaciones multiplataforma

Alumno: Morales Aguayo, Carolina
Director: Abad Cardiel, Ismael

Curso: 2017/2018
Convocatoria: Septiembre de 2018

Máster Universitario de Investigación en Ingeniería de Sotfware y Sistemas Informáticos

Itinerario de Ingeniería de Software (Código 31105128)

GenMovil: Aplicación de WebDSL para crear aplicaciones multiplataforma

Generador automático de código que parte del DSL WebDSL y tiene como finalidad un proyecto Apache Cordova para generación de aplicaciones multiplataforma

Alumno: Morales Aguayo, Carolina
Director: Abad Cardiel, Ismael

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 08/08/2018

Quién suscribe:

Autor(a): Carolina Morales Aguayo
D.N.I/N.I.E/Pasaporte.: 30.975.365-T

Hace constar que es la autor(a) del trabajo:

Título completo del trabajo.
GenMovil: Aplicación de WebDSL para crear aplicaciones multiplataforma

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo. 

Impreso TFdM05_Autor. Autorización de publicación
y difusión del TFdM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

Juan del Rosal, 16
28040, *Madrid*

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

Resumen:

El trabajo de investigación expuesto en este documento trata de dar visibilidad a la generación de aplicaciones móviles multiplataforma a partir de un dominio específico, WebDSL.

Existe gran cantidad de infraestructuras y herramientas sotfware que facilitan la generación de aplicaciones móviles en distintas plataformas a partir de un único dominio. Tras un breve período de decisión, se ha escogido una de ellas como objetivo del generador, Apache Cordova.

El trabajo presenta una visión general del estado del arte en cuanto a generación de aplicaciones multiplataforma. A continuación, se explican las características tanto del lenguaje de dominio WebDSL como de la estructura de un proyecto tipo Apache Cordova. Seguidamente, se explican las particularidades del lenguaje WebDSL como si de un lenguaje de dominio se tratara. Después, se analizan las transformaciones a las que ha de dar lugar el generador para la obtención de la estructura Apache Cordova. Finalmente, se realiza una demostración de los resultados obtenidos haciendo uso del generador.

Este trabajo de investigación permite la progresión en otras líneas también relacionadas con la generación automática de aplicaciones móviles multiplataforma. Las líneas de investigación parten desde añadir funcionalidades al generador diseñado, hasta ser capaces de convertir cualquier página web a una aplicación móvil, pasando por los distintos tipos de herramientas para aplicaciones multiplataforma ya existentes.

Palabras clave:

Generador automático, aplicación móvil, aplicación multiplataforma, WebDSL, HTML, JavaScript, CSS, Apache Cordova, Xamarin, android, windows, navegador

Índice

Índice de figuras

Índice de tablas

1. Introducción

El uso de dispositivos móviles tales como *tablets* o *smartphones* es indispensable hoy en día. Los usuarios de estos dispositivos esperan tener a su alcance todo tipo de aplicaciones que cubran sus necesidades en todo momento: noticias, información del tiempo o la bolsa, redes sociales, mensajería, etc. Los dispositivos móviles aportan al usuario una respuesta inmediata ante la necesidad de actuación o conocimiento.

Aunque los dispositivos móviles soportan distintos tipos de navegadores webs, el uso de aplicaciones descargables e instalables es mucho más atractivo al usuario, ya que mejora la interfaz gráfica, rendimiento y uso de los recursos en los dispositivos móviles. Surge pues la necesidad de crear o generar aplicaciones móviles con el mismo contenido que las aplicaciones web, pero incorporando las prestaciones de un dispositivo móvil.

Los dispositivos móviles más usados hoy en día utilizan como sistema operativo Android o iOS, aunque existen otros menos frecuentes como Windows o Blackberry. Cada uno de estos sistemas operativos desarrolla sus aplicaciones en un formato diferente, con lenguajes diferentes y estructuras distintas. Existe pues, la necesidad de tener una infraestructura común que permita, con un solo desarrollo, obtener aplicaciones para los distintos sistemas operativos.

La generación de aplicaciones multiplataforma supone una reducción de costes, tiempo y recursos a la hora de crear una aplicación que pueda ser soportada en los distintos dispositivos. [1]

Consecuentemente, aparece la necesidad de creación de aplicaciones móviles multiplataforma que sean equivalentes a una página o servicio web. La generación de aplicaciones multiplataforma está muy extendida, es decir, ya existen infraestructuras y diseños que permiten generar aplicaciones móviles para distintos sistemas operativos a partir de un código común. Para poder generar las distintas aplicaciones móviles a partir de una web, es necesario que este código común acepte el lenguaje web de la página en cuestión o que sea compatible con él.

Algunos generadores de aplicaciones multiplataforma que existen se enmarca en el cuadro de las aplicaciones híbridas. Estas aplicaciones híbridas son realmente páginas web dentro de un navegador que está empotrado en un aplicación móvil. En este caso, el navegador web es invisible para el usuario y el aspecto de la interfaz es similar al de una aplicación móvil nativa. Estas aplicaciones híbridas tienen dos capas: el contenedor y la interfaz. La interfaz es la página web en sí, es decir, el código en el dominio web

que crea la página. El contenedor es la parte específica de plataforma, que puede ser de tipo Android, iOS, Windows o, y aquí es donde reside el punto clave de este trabajo, ser un generador automático de aplicaciones multiplataforma. Existen contenedores de tipo multiplataforma cuyo uso está muy extendido, como Apache Cordova [2] o Xamarin [3]. Para estos casos, la generación de aplicaciones multiplataforma sólo se centra en la parte de la interfaz.

Existen gran cantidad de dominios de lenguaje web, algunos muy conocidos, tales como HTLM, JavaScript o CSS. Estos lenguajes están bien integrados en la generación de aplicaciones multiplataforma, ya que existen infraestructuras que los aceptan como código común a la hora de generar las aplicaciones móviles. Sin embargo, existen otros dominios de lenguaje menos conocidos, como WebDSL, que se encuentran con el inconveniente de no tener una conversión directa a aplicaciones multiplataforma. WebDSL supone un lenguaje web novedoso, dinámico e interactivo, capacitado para soportar aplicaciones con modelos de datos amplios y complejos, además de otros módulos que aportan funcionalidades extra (control de acceso, validación de entradas, etc.), mediante una sintaxis sencilla y concisa. [4] [5]

1.1. Alcance

El alcance del presente proyecto abarca la generación de aplicaciones multiplataforma a partir un dominio web.

El uso de aplicaciones híbridas y la existencia de contenedores multiplataforma hace factible la creación o generación automática de interfaces a partir de un dominio web específico. Esta descripción abarca varias líneas de investigación, puesto que las combinaciones posibles de generadores es alta:

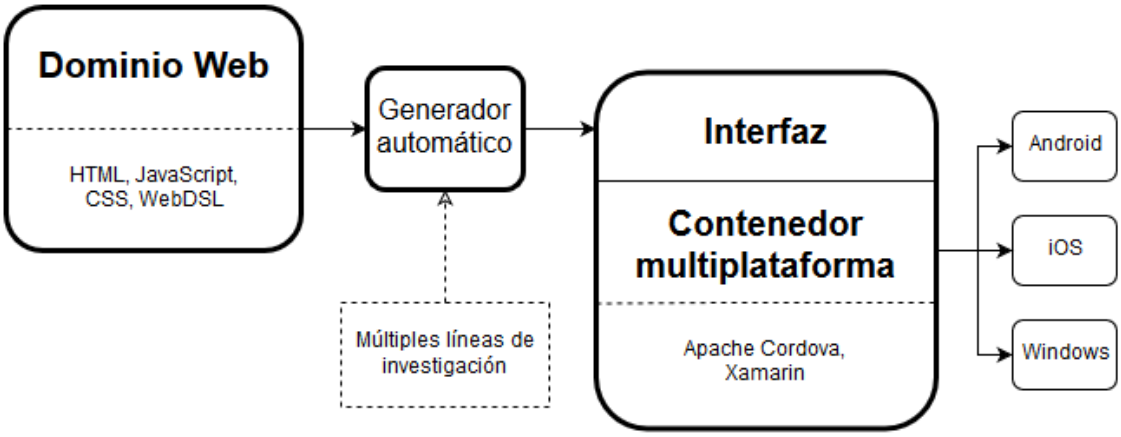

Figura 1: Diagrama de alcance

En el caso de este proyecto se ha elegido el dominio web WebDSL y el contenedor multiplataforma Apache Cordova. Un primer apunte para la investigación de este proyecto es hacer un análisis de la estructura, sintaxis y capacidad del dominio web WebDSL. Se estudia, para los casos más interesantes propios del dominio WebDSL, su capacidad de adaptación a un contenedor multiplataforma. Así pues, el proyecto trata de describir de qué forma adaptar el dominio WebDSL a la estructura y caracteristícas del contenedor multiplataforma Apache Cordova.

1.2. Objetivos

El proyecto descrito en este documento trata de dar visibilidad a la generación de aplicaciones multiplataforma haciendo uso de herramientas que ya existen, tienen una madurez alta y son bien conocidas en el mercado. Estas herramientas son capaces de, a partir de un código común, adaptarse a las diferentes plataformas que existen y crear aplicaciones independientes para los distintos tipos de dispositivos móviles. La existencia de estas herramientas facilita la conversión de aplicaciones web en aplicaciones móviles.

Sin embargo, existe una gran cantidad de páginas webs desarrolladas y escritas en multitud de lenguajes diferentes, que, idealmente, han de servir como entrada a las herramientas de conversión para obtener las aplicaciones multiplataforma y que estas queden disponibles. No todas las páginas webs existentes van a tener una conversión directa con las herramientas de conversión. Es por eso necesario dar visibilidad y credibilidad al desarrollo de otras herramientas que permitan adaptar, modificar y encajar los dominios webs

que ya existen contra las herramientas de conversión más maduras.

1.3. Plan de trabajo

La duración para la realización de este proyecto ha sido de dieciséis semanas aproximadamente. En el desarrollo del proyecto ha habido tres etapas principales a despuntar: investigación y análisis, diseño y codificación del generador y desarrollo de la memoria. Cada una de las actividades se ha separado en otras más pequeñas y específicas, indicando el tiempo que ha llevado su finalización:

1. **Investigación**: 6 semanas
    - Investigación y documentación de aplicaciones multiplataforma: 2 semanas.
    - Investigación, análisis e instalación de WebDSL: 3 semanas.
    - Investigación, análisis e instalación de Apache Cordova: 2 semana.
2. **Generador automático de código**: 7 semanas
    - Diseño del generador: 1 semana.
    - Parseo de las entradas al generador: 2 semanas.
    - Generación de salidas: 3 semanas.
    - Pruebas y depuración: 1 semana.
3. **Memoria**: 3 semanas

Las actividades no han seguido el orden lineal estricto expuesto, sino que se han visto mezcladas o interrumpidas cuando ha sido necesario.

Por ejemplo, el desarrollo de la memoria se ha hecho en paralelo con cada uno de los apartados expuestos. Así mismo, el diseño y codificación del generador se ha separado en dos partes. El parseo de las entradas y la generación de salidas se ha desarrollado de forma independiente para cada una de estas partes.

1.4. Estructura de la memoria

El presente documento comienza con un apartado de estado del arte, en el que se pretende hacer un análisis de la problemática actual, así como analizar las soluciones ya

existentes y su alcance. En este apartado también se presenta la vía de desarrollo elegida para elaboración de este proyecto y cómo esta se relaciona con el generador automático de código.

A continuación, se realiza un análisis detallado de la solución propuesta y el generador automático a desarrollar. Se describen algunas de las características estructurales de los entornos de entrada y salida del generador. El siguiente apartado muestra las características sintácticas y funcionales de WebDS.

Después se presenta el generador automático de código y como trabaja para convertir el código WebDSL en código web compatible con la estructura de Apache Cordova. Más tarde, se expone un ejemplo de la salida del convertidor y las capacidades funcionales que se heredan de WebDSL.

Para finalizar, se añade un apartado de conclusiones sobre el trabajo realizado, así como una reflexión sobre las líneas de investigación abiertas a raíz del desarrollo de este proyecto.

2. Estado del arte

Los contenedores para las aplicaciones híbridas más conocidos y usados son Xamarin y Apache Cordova. Existen diferencia de ellos, no sólo respecto a las interfaces que soportan sino a nivel de independencia que tienen respecto a las plataformas móviles. Así pues, se distinguen dos tipos de aplicaciones multiplataforma [6]:

1. **Aplicación multiplataforma web**: Se trata de aplicaciones de HTML que se ejecutan en un navegador web propio de cada plataforma.

2. **Aplicación multiplataforma pseudo-nativas**: Se trata de frameworks que generan código nativo de cada plataforma a partir de otro código (JavaScript, C#, etc).

2.1. Aplicación multiplataforma web

Las aplicaciones están escritas en HTML5 y se ejecutan en navegador nativo tipo WebView dentro de cada plataforma. El rendimiento de este tipo de aplicaciones es más lento que el de una aplicación nativa debido a la carga del navegador.

Una de las ventajas principales de este tipo de aplicación es que el desarrollador web puede utilizar todo su potencial, sin necesidad de aprender ningún otro lenguaje. Así, todas las aplicaciones que están escritas en HTML, JavaScript y CSS tiene una herramienta de conversión directa a una aplicación multiplataforma.

Para acceder a las capacidades que ofrece cada dispositivo y que no tienen las páginas webs, se utilizan APIs nativas y plugins. Con estos mecanismos se puede acceder a los sensores, cámara, estado de la red, etc.

Las APIs nativas que estén soportadas por la plataforma son accesibles desde la aplicación en JavaScript. También existen plugins ya desarrollados que facilitan el acceso a las APIs. En caso de que exista una funcionalidad nativa que no esté aún soportada en ningún plugin, el desarrollador ha de encargarse de su creación.

El contenedor Apache Cordova está dentro de esta categoría. Apache Cordova dispone de una gran cantidad de plugins disponibles tanto oficiales como desarrollados por la comunidad. [7]

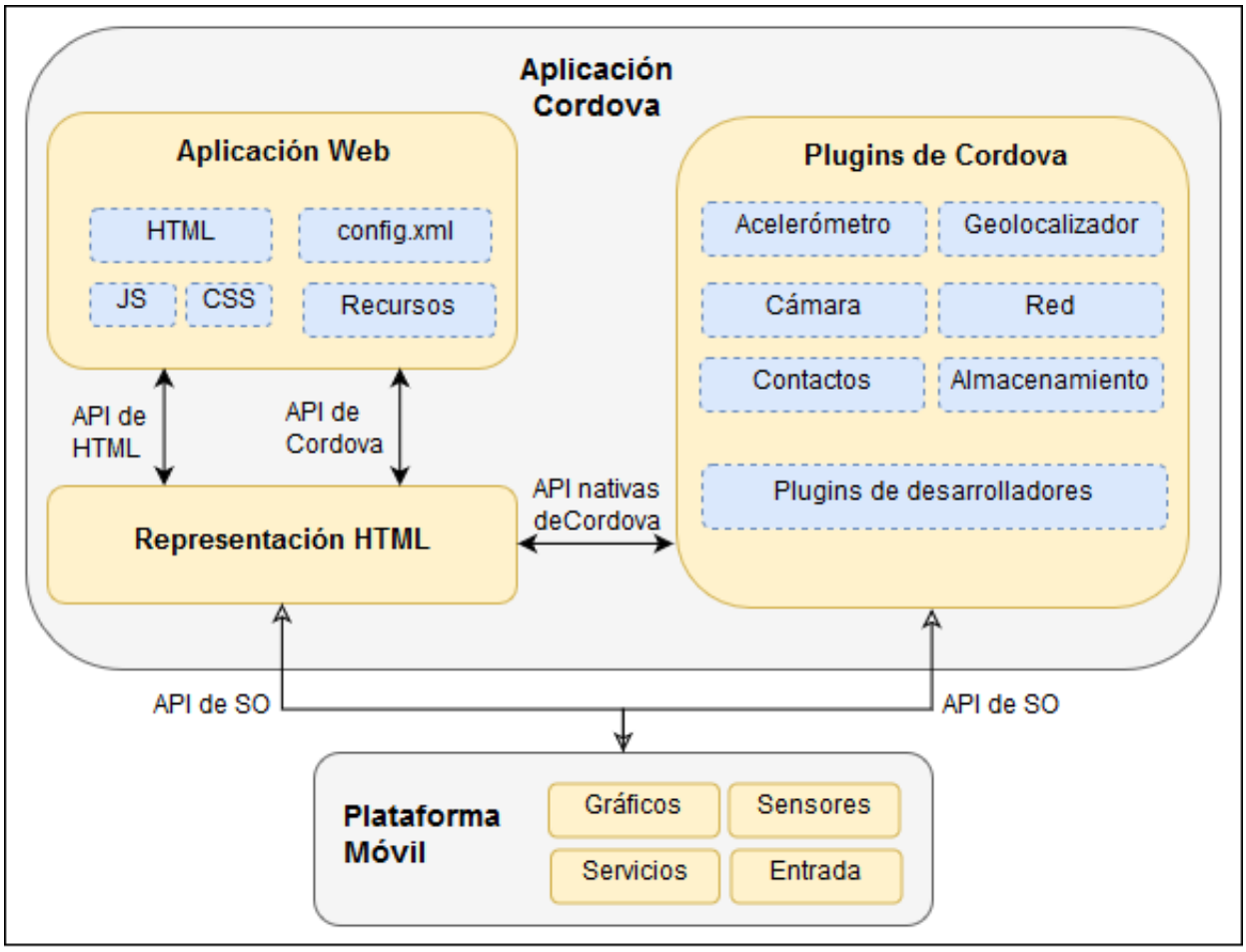

Figura 2: Estructura de Apache Cordova

Existen muchas infraestructuras de HTML5 que ofrecen su integración con Cordova:

- *PhoneGap*: es la distribución más común de Apache Cordova. Permite crear aplicaciones muy rápido a partir de código web haciendo uso de una aplicación de escritorio.
- *Telerik*: es un plataforma de desarrollo para la creación de aplicaciones nativas e híbridas con JavaScript.
- *AppBuilder*: es un entorno de desarrollo para Microsft Windows que permite crear aplicaciones HTML5 y nativas sin tener conocimientos de programación. Ofrece varios controles y acciones para incluir en las aplicaciones móviles, así como ejemplos ilustrativos.

- *GapDebug*: es una herramienta para depurar y gestionar aplicaciones híbridas para Android e iOS.

2.2. Aplicación multiplataforma pseudo-nativa

Estas aplicaciones trabajan directamente con componentes nativos por lo que el aspecto visual de la aplicación móvil final es igual que si se hubiese desarrollado de forma nativa. La ventaja es que, para cada una de las plataformas, existe un lenguaje común a más bajo nivel.

Estas aplicaciones multiplataforma tienen en común la lógica interna de la aplicación pero permiten (y a veces exigen) el desarrollo de la interfaz de usuario de forma nativa. Esta solución pierde levemente el carácter multiplataforma que se busca, pero permite la optimización máxima de cada una de ellas.

Como realmente se trabaja con componentes nativos, no se tiene ningún tipo de restricción a la hora de trabajar con recursos de los distintos dispositivos.

La aplicación más importante que se enmarca dentro de este tipo es Xamarin.

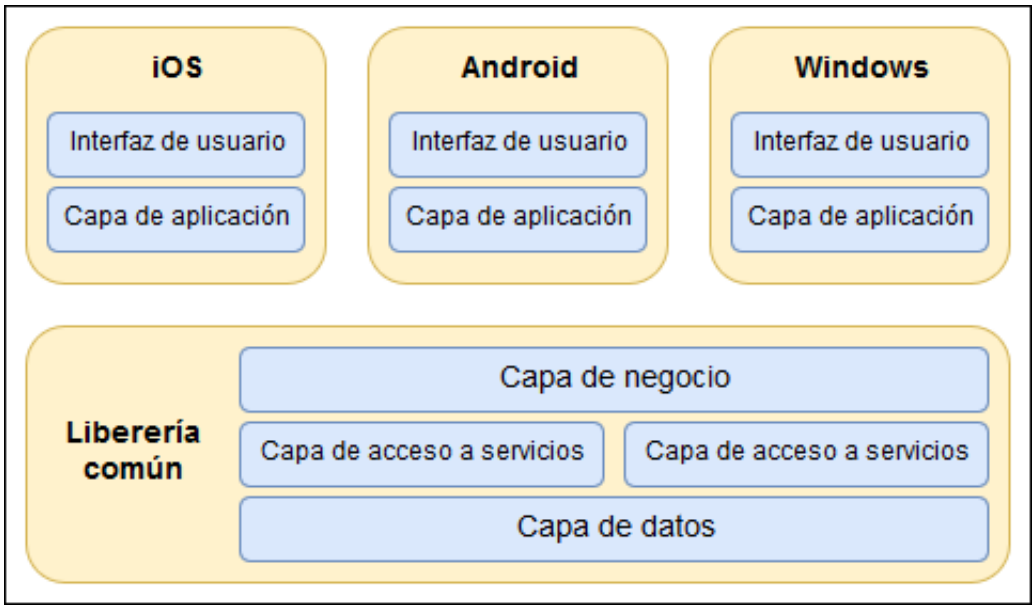

Figura 3: Estructura de Xamarin

Aún así, existen muchas otras también consideradas como pseudo-nativas:

- *Xamarin.Forms*: es una API de Xamarin que aumenta la cantidad de código común entre plataformas hasta más del 95 %.
- *AppCelerator*: esta aplicación soporta iOS, Android y Blackberry a partir de JavaScript. Es necesario adquirir una licencia para su uso.
- *NativeScript*: soporta iOS y Android a partir de JavaScript y es de libre distribución. Además, se puede utilizar con Angular 2.
- *Kivy*: haciendo uso de Python, esta aplicación está orientada a las interfaces gráficas para el desarrollo de juegos. Es de libre distribución.

2.3. Comparativa

A continuación, se incluye una tabla que muestra una comparativa entre los dos tipos de aplicaciones multiplataforma presentados centrándose en aspecto como el rendimiento o las distintas plataformas soportadas.

Con esta comparación, nace la necesidad de elegir cuál es herramienta de aplicaciones multiplataforma más conveniente a utilizar a la hora de desarrollar una aplicación móvil para distintos sistemas operativos. Haciendo uso de las descripciones expuestas anteriormente así como de la tabla comparativa, se puede obtener una serie de afirmaciones útiles que ayuden a decidir cuál es la mejor opción:

- **Aplicaciones multiplataforma web**. Su uso es más conveniente en los siguientes casos [6]:
  1. Si los desarrolladores tienen amplio conocimiento en lenguaje web.
  2. Si se busca rapidez en la obtención de aplicaciones móviles listas para instalar y utilizar.
  3. Si la aplicación móvil a desarrollar no tiene un alto contenido gráfico.
  4. Si la aplicación móvil a desarrollar no requiere de mucha capacidad de rendimiento.
- **Aplicaciones multiplataforma pseudo-nativas**. Este tipo de herramientas es más conveniente para los siguientes entornos:
  1. Si se busca un resultado igual al de las aplicaciones nativas.

Característica	App. web	App. pseudo-nativa
Lenguaje	HTML, JavaScript, CSS	C#, Python
Reutilización	Máxima	Interfaz poco común
Plataformas	Android, iOS, MAC, Window, Blackberry	Muy comunes: Android, iOS; Poco comunues: Windows, Blackberry
Functionalidad nativa	APIS, plugins; difícil para componenetes visuales	Totalmente cubierta
Rendimiento	Navegador web	Traducción de código común a código nativo.

Tabla 1: Comparación aplicaciones multiplataforma tipo web y pseudo-nativas

2. Si el rendimiento de la aplicación móvil ha de ser alto.
3. Si se trabaja con elementos gráficos pesados o complejos.
4. Si se necesita acceso a todos los recursos del dispositivo móvil.

2.4. Apache Cordova como aplicación multiplataforma

Apache Cordova, al ser una herramienta para aplicaciones multiplataforma web, permite usar los lenguajes estándares HTML, CSS y JavaScript para el desarrollo de aplicaciones móviles sin tener en cuenta las características nativas de cada una de las plataformas. Las aplicaciones se ejecutan en un contenedor y que basan en APIs estándar que permiten el acceso a los distintos elementos de los dispositivos: estado, red, datos, sensores, etc. [8]

El uso de Apache Cordova esta orientado a programadores de aplicaciones móviles que quieran obtener la misma aplicación para diferentes plataformas, pero sin codificación de

cada lenguaje específico. Apache Cordova permite mezclar componentes de las aplicaciones nativas con un navegador que puede acceder a las APIs de cada dispositivo.

Apache Cordova permite su ejecución mediante línea de comandos con la palabra reservada *cordova*, que es la más recomendada para su uso. El escenario más básico permite la creación de una aplicación Apache Cordova con un contenido por defecto, que puede ser manipulado por el usuario.

Mediante la línea de comandos, también se permite añadir las distintas plataformas sobre las que se va a generar la aplicación. Para cada plataforma concreta, es necesario la instalación de su SDK. Por supuesto, se pueden utilizar emuladores de dispositivos móviles para la ejecución de la aplicación con Apache Cordova. Es posible que la ejecución de una plataforma requiera de un sistema operativo base concreto para su funcionamiento, como es el caso de iOS.

Actualmente, Apache Cordova ofrece soporte para las plataformas Android, BlackBerry, iOS, Windows Phone 7 y 8, Windows 8 y Tizen. No todas las plataformas están preparadas para soportar el uso de la línea de comandos, el navegador web embedido o las diferentes APIs. Toda esta información se puede encontrar en la página web oficial de Apache Cordova, listadas en la bibliografía. [9]

2.5. WebDSL como lenguaje Web

La estructura de una página web suele estar separada por capas, donde cada una de ellas está codificada con un lenguaje diferente: Java, SQL, JSF, etc. Conseguir que estas capas trabajen de forma conjunta es una tarea para los desarrolladores que puede dar lugar a errores de interacción entre las diferentes capas o incluso a la existencia de agujeros en la seguridad de los sitios web.

WebDSL se postula como un lenguaje capaz de unificar estas capas en una sola, donde el desarrollador tiene el control de todos los elementos. Los proyectos en WebDSL proveen de una plataforma de desarrollo para webs integradas, reduciendo los problemas provocados por integración de las diferentes capas. Este lenguaje asegura una arquitectura modular, haciendo uso de la generación de código por transformación de modelos. [10, 11]

WebDSL se utiliza como un modulo de la herramienta *Eclipse*, en el que pueden crearse proyectos que ya tienen incorporada la estructura necesaria para el funcionamiento de la aplicación. La compilación de los proyectos tipo WebDSL se puede realizar desde el

propio *Eclipse* o haciendo uso de un compilador externo. Esta segunda opción es mucho más rápida que la primera. El compilador se encarga, además de generar el código web, de crear las bases de datos necesarias así como cualquier otro elemento necesario para la aplicación. [12]

La ejecución de WebDSL se realiza sobre un servidor web, normalmente *tomcat*.

A continuación se muestra un ejemplo sencillo, pero conciso, de un desarrollo en WebDSL. Con este ejemplo se pretende dar visibilidad a la capacidad del lenguaje.

```
 1  application ShortExample

 2  page root(){
 3          navigate createUser() {"create User"}
 4          authentication()
 5  }

 6  entity Meeting{
 7          name :: String
 8          speaker -> User
 9          attendant -> List<User>
10  }
11  entity User{
12          name :: String
13          password :: Secret
14  }
15  derive CRUD User
16  derive CRUD Meeting

17  principal is User with credentials name, password
18  access control rules
19          rule page root(){true}
20          rule page createUser(){loggedIn()}
```

El análisis de este código incluye dos de las partes que se van a utilizar en el generador automático de código, de cara a la sintaxis de WebDSL: el modelo de datos y el control de acceso. El análisis del código es:

1. Existe una aplicación web con nombre *ShortExample* (línea 1).

2. La página principal tiene un enlace a la página *createUser* (línea 3) y un formulario de autenticación(línea 4).

3. Para el modelo de datos:

   a) Con la sentencia *entity* se han definido los objetos *User* y *Meeting* (líneas 6 a 14).

   b) Cada uno de los objetos definidos contiene propiedades que pueden referenciar a otros objetos (líneas 8 y 9).

   c) La sentencia *CRUD* permite crear, modificar, visualizar y borrar instancias de los objetos definidos (líneas 15 y 16).

4. Para el control de acceso:

   a) Los objetos tipo *User* se utilizan para autenticar a los usuarios, haciendo uso del nombre y la contraseña (línea 17).

   b) Todos los usuarios tienen acceso a la página principal (línea 19).

   c) Sólo los usuarios que han iniciado sesión (línea 20).

El alcance y potencial del lenguaje WebDSL queda bien identificado en este ejemplo. Con una extensión de 20 líneas de código, es posible crear una estructura de datos de acuerdo a un modelo, páginas para la gestión de dichos datos y control de acceso para las páginas definidas.

3. Descripción de la solución

En los apartados anteriores se han mostrado dos necesidades que conciernen a las aplicaciones para dispositivos móviles:

- Aplicaciones móviles multiplataforma, debido a la necesidad de coexistencia de distintas plataformas para los distintos dispositivos móviles.
- Facilidad en la conversión de páginas web en aplicaciones para dispositivos móviles. De esta forma, cualquier empresa, organismo o compañía puede obtener su propia aplicación móvil con la certeza de conservar el mismo servicio que en su página web.

Para dar respuesta a esta necesidad, se propone realizar un convertidor de código desde el DSL para páginas web WebDSL, hasta la plataforma Apache Cordova.

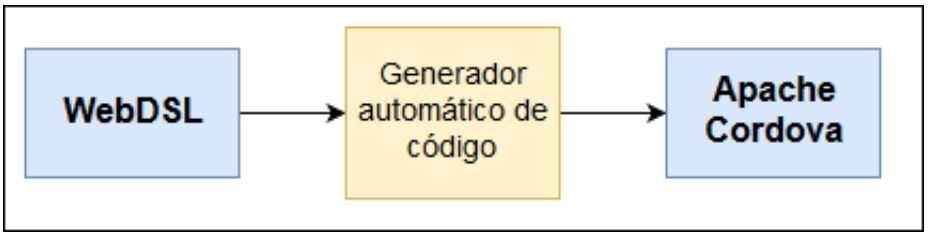

Figura 4: Entrada y salida del generador automático de código

Un generador automático de código ideal para la solución propuesta sería aquel que, ante cualquier código WebDSL como entrada, genere de forma automática un código compatible con Apache Cordova, aplicable a todas sus plataformas y con las mismas funcionalidades. WebDSL es un lenguaje de dominio que contiene módulos de diversas y amplias funcionalidades. Cada uno de estos módulos, y todas las funcionalidades que ofrecen, se pueden utilizar como entrada al generador automático de código.

Sin embargo, el presente proyecto pretende dar visibilidad y credibilidad a la generación de aplicaciones móviles multiplataforma a partir de un dominio específico. Abordar todo el contenido de WebDSL resulta poco práctico para este proyecto. Es por esto que se escogen algunas características de WebDSL para formar parte de la entrada del generador de aplicaciones multiplataforma, lo suficientemente amplias como para verificar la viabilidad del proyecto. El abordaje de todo el contenido y funcionalidad de WebDSL para el

desarrollo del generador automático queda fuera del alcance de este proyecto, dando lugar a un otro centrado en el diseño y estructura, en lugar de en la investigación.

WebDSL es un lenguaje de dominio específico para crear páginas webs dinámicas. La parte más nativa del lenguaje WebDSL define entidades, páginas y lógica a bajo nivel. Existe extensiones modulares a la parte central del lenguaje que permiten trabajar con abstracciones a alto nivel, control de acceso y flujos de trabajo. Una de las características principales del dominio WebDSL es que permite al desarrollador abstraerse de las diferentes capas que existen bajo una aplicación web. De esta forma, WebDSL se postula como un lenguaje unificado en el que intervienen todas las acciones requeridas por un estructura web heterogénea. [5]

Por otro lado, la plataforma Apache Cordova se encarga de ejecutar un navegador web y desplegar en él la página web formalizada en sus módulos de HTML, JavaScript y CSS.

El generador automático propuesto se encarga de transformar las distintas particularidades del lenguaje WebDSL (código unificado), en codificación web HTML, JavaScript y CSS.

3.1. Estructura de WebDSL

WebDSL se crea como un proyecto de la plataforma *Eclipse*. Cada workspace de WebDSL contiene varios archivos (de configuration, datos y descripción de la aplicación WebDSL) y subcarpetas. Se mencionan aquí las más importantes y las que aplican al proyecto [12]:

- Archivo "application.ini": en este archivo se encuentran los elementos de configuración principales de la página WebDSL. Se pueden configurar, entre otros los siguientes elementos:
  - Nombre de la aplicación.
  - Ruta donde se encuentra el servidor Tomcat.
  - Puertos para los protocolos HTTP y HTTPS.
  - Propiedades del servidor SMTP: máquina('localhost' por defecto), puerto, usuario y contraseña.
  - Propiedades de la base de datos: tipo('h2' o 'mysql'), nombre, servidor, modo de uso, etc.

- Archivo *nombreArchivo.app*: los archivos con extensión *app* contienen lenguaje de tipo WebDSL y definen el contenido de la propia pagina web además de los datos y todos los aspectos relacionados con ella. Los ficheros se pueden importar entre ellos y estar incluidos en rutas de carpetas para la organización del código fuente. Se muestra un ejemplo de un archivo *app* con la aplicación básica 'Hola Mundo':

```
application SimpleHelloWorld

  page root(){ "Hello world" }
```

- Carpeta *stylesheets*: en esta carpeta se incluyen los archivos con extensión *.cs* para el formato de la página.

3.2. Estructura de Apache Cordova

Una aplicación de tipo Apache Cordova contiene los siguientes elementos:

- Archivo de configuración *config.xml* donde se recogen las distintas propiedades de la aplicación:
  - Nombre de la aplicación.
  - Descripción.
  - Autor.
  - Plugins instalados.
  - Plataformas soportadas: Android, iOS, etc.

  Este fichero *config.xml* se crea de forma automática durante la generación del proyecto Apache Cordova. El contenido de este fichero es dinámico y se ve modificado a lo largo del desarrollo. Apache Cordova se encarga de la modificación de este fichero de acuerdo a las exigencia del desarrollador en cuento a los plugins instalados o las plataformas que necesitan ser soportadas.
- Carpeta de archivos con código web en lenguaje HTML, JavaScript y CSS.
- Carpeta con los plugins instalados.
- Carpeta con las plataformas instaladas.

3.3. Ámbito del generador

El generador propuesto como parte práctica del presente proyecto pretende utilizar las características de WebDSL definidas en los módulos de modelo de datos y control de acceso para resolver diferentes casos de uso con las siguientes características:

- Existe un sistema de datos, llamados *objetos*, que se almacenan en una base de datos.
- Los objetos están relacionados entre ellos por medio de sus propiedades.
- Los objetos se pueden consultar, crear, borrar, modificar o manipular de cualquier modo.
- Los permisos necesarios para acceder a estas operaciones viene dado por un sistema de control de acceso.

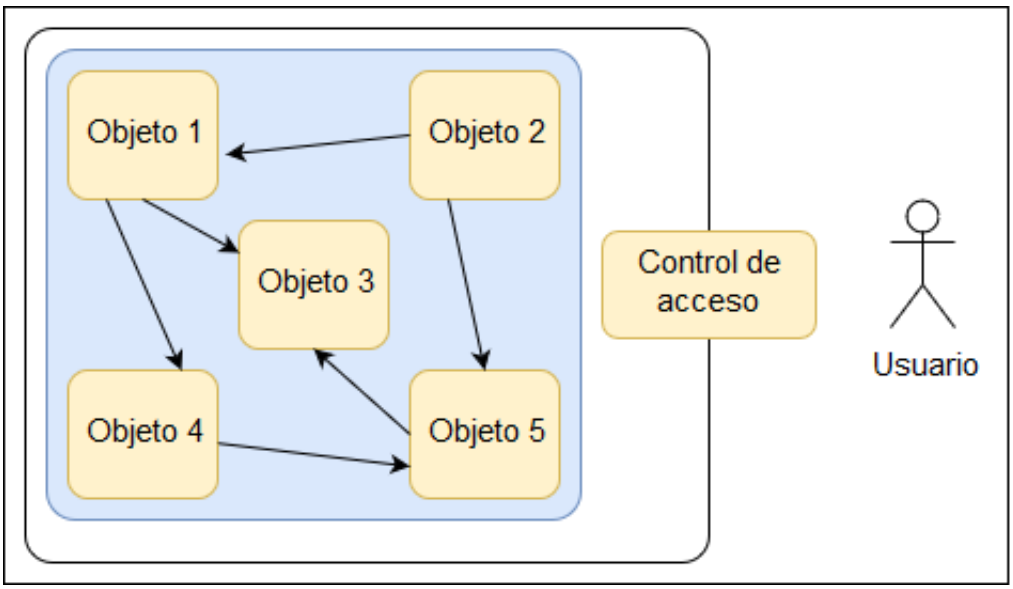

Figura 5: Esquema general de caso de uso para el generador

Existen muchos ejemplos de un modelo de datos con control de acceso que cumpla las características descritas. Como ejemplos se pueden citar:

1. En el ámbito de una institución académica, se pueden organizar la relación entre

alumnos, profesores y asignaturas. Los profesor tienen acceso a esta información, pero los alumnos no.

2. En un evento cultural, definir los diferentes eventos de que se compone, los asistentes, las fechas y lugares necesarios. Sólo el personal gestor del evento tiene capacidad de modificar alguna de las actividades.

3. En un registro de filmografía, definir películas, series, actores, directores y demás elementos que intervengan.

Así pues, si existe un conjunto de datos a organizar dentro de un sistema en el que éstos están relacionados entre sí, puede ser una buena práctica utilizar las características que ofrece WebDSL para organizar la estructura de dicho sistema. El uso de WebDSL y sus plantillas permite tener de forma inmediata y automática páginas para crear, modificar, gestionar o simplemente consultar cada uno de los datos a manejar. Además, el uso y manejo del modelo de la datos en WebDSL, permite una gestión del contenido que es muy transparente para el programador.

Hay que tener en cuenta que los conjuntos de datos definidos para trabajar con las sentencias WebDSL no necesitan ser estáticos. La aplicación se puede desarrollar sobre conjuntos de datos dinámicos en los que éstos interactúan entre sí de forma constante.

El análisis de los datos y el control de acceso va a ofrecer, por medio del generador automático de código, ficheros web compatibles con la estructura Apache Cordova, es decir, con código HTML y Javascript. Las aplicaciones móviles generadas permiten visualizar los datos en tablas o listas así como modificarlos y crearlos por medio de formularios. Los ficheros de formato CSS son compatibles tanto con WebDSL como Apache Cordova, por lo que no se realizará ninguna modificación sobre ellos, manteniendo así el formato original.

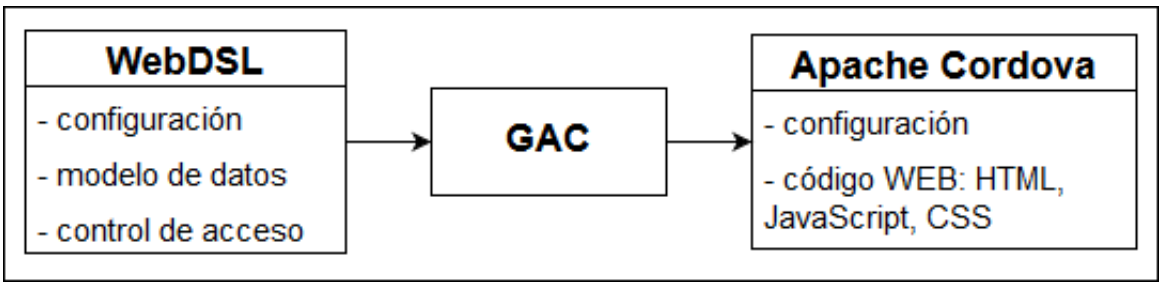

Figura 6: Esquema general del generador automático

El nombre de la aplicación móvil es equivalente al nombre incluido en el fichero *application.ini* de WebDSL.

4. WebDSL como DSL

Para la generación automática de aplicaciones multiplataforma basadas en contenedores webs, se ha elegido el lenguaje WebDSL como entrada al generador. Este lenguaje, aunque no tan conocido como otro dominios webs, contiene ciertas características de abstracción para el desarrollo que le convierte en un DSL heterogéneo y de interés para la generación automática de código.

Inicialmente, WebDSL contiene sólo tres sublenguajes: modelo de datos, interfaz de usuario y lógica. Conforme se avanza en el desarrollo de WebDSL, se añaden más sublenguajes para abstraer al programador de términos a bajo nivel: control de acceso, flujo de trabajo, validación de datos, etc. A continuación, se explican algunos de estos sublenguajes:

- El modelo de datos de una página web en WebDSL se describe usando 'entidades', que son objetos con propiedades, donde cada propiedad tiene un nombre y un tipo. Cada entidad se puede visualizar y editar en las páginas webs. La definición de una página web se caracteriza haciendo uso de las diferentes entidades.

- Para el control de acceso, WebDSL contiene una reglas que determinan la accesibilidad a los distintos elementos de la aplicación. Los datos de la aplicaciones puede ser accesibles (o no) a los usuarios. Para trabajar y controlar los permisos en el control de acceso se utilizan las expresiones del propio lenguaje, por lo que se produce un integración entre la definición de la aplicación web y la forma de especificar las comprobaciones en los accesos. La representación de cada usuario se declara haciendo uso de la definición 'principal'.

- El flujo de trabajo en WebDSL es un modelo de lenguaje orientado a objeto para aplicaciones webs. Se definen unos 'procedimientos' para operar con objetos de dominio. Los procedimiento se forman a partir de combinaciones de procesos secuenciales y concurrentes. La definición de estos flujos de trabajo resulta en páginas, listas de tareas, estado de las páginas y navegación entre ellas.

- La validación en WebDSL permite comprobar la consistencia de datos de entrada en formularios, así como proporcionar al usuario una respuesta útil para comprender el uso del sistema. Existe un módulo de validación en WebDSL que permite especificar las reglas de validación de los datos de entrada de los usuarios. Normalmente, las sentencias de validación cuentan con dos argumentos: uno que valora la veracidad

de los datos introducidos y otro que muestra un mensaje de error en caso de fallo. La comprobación de los datos de entrada se añade automáticamente a cada campo de entrada de la página.

WebDSL contiene, por tanto, una estructura de módulos en los que existe un código nativo y, más adelante, añadidos que permiten trabajar con funcionalidades extra. Esta característica es un motivo para elegir este lenguaje como DSL de entrada al generador automático de aplicaciones móviles.

Para sintetizar las funcionalidades de WebDSL de cara al generador automático, se han elegido aquellos módulos que se consideran más interesantes y necesarios para la demostración de esta investigación. Los dos módulos elegidos son los encargados de la gestión y manejo del modelo de datos y el control de acceso.

Por tanto, como lenguaje de dominio, se tienen en cuenta las siguientes características o módulos de WebDSL.

1. Páginas
2. Plantillas
3. Modelo de datos o entidades
4. Control de acceso

Para cada uno de ellos se indica, a continuación, su sintáxis y su uso, así como ejemplos válidos de entrada al generador.

4.1. Páginas

Una de las propiedades más importantes de un servicio web es el aspecto y contenido de cada una de las páginas que lo componen. Para describir este contenedor en WebDSL se utiliza la sentencia *define page* seguido del nombre de la página.

La declaración de una página puede contener argumentos de entrada, así como un título (que aparece en la cabecera de la página web) y diversas secciones y cabeceras para facilitar la estructura.

```
define page MiPagina(){
	title{"Mi pagina WebDSL"}
	section{
		"Esta es la primera seccion"
	}
	section{
		"Esta es la segunda seccion"
	}
}
```

En el ejemplo expuesto, el nombre de la página es *MiPagina* y el título *Mi pagina WebDSL*. La página contiene dos secciones.

Las distintas páginas a definir para una aplicación pueden estar contenidas en diferentes ficheros. Para el generador automático de páginas webs, se asume que no existen diferentes secciones dentro de la misma página, por lo que todo el contenido pertenece a una misma sección.

El contenido de las páginas definidas en WebDSL incluye elementos gráficos propios de WebDSL, así como ciertas sentencias HTML. Además, dentro del ámbito de cada página se pueden realizar asignaciones, comprobaciones o cualquier sentencia lógica en el ámbito de WebDSL.

4.1.1. Página principal

La página principal o *root* es la primera página a la que se accede desde el navegador web para acceder a la aplicación WebDSL. El nombre de está página ha de ser *root* en WebDSL y se define de la siguiente forma:

```
define page root()
{
	"Esta es la pagina principal"
}
```

Su estructura y uso es igual que la de una página normal.

La definición de la página principal es obligatoria para WebDSL.

4.1.2. Enlaces

La definición de páginas se utiliza también para poder crear enlaces de una páginas a otras. Así pues, para crear enlaces que permiten acceder a un página a partir de otra, se utiliza la sentencia *navigate*:

```
page root(){
     navigate page1(){"Enlace a la pagina 1"}
     navigate page2(){"Enlace a la pagina 2"}
}

define page page1(){
     "Esto es la pagina 1"
}

define page page2(){
     "Esto es la pagina 2"
}
```

En el ejemplo expuesto, aparecen dos enlaces con los nombres "Enlace a la pagina 1" y "Enlace a la pagina 2", que remiten a las páginas *page1* y *page2* respectivamente.

4.2. Plantillas

Las plantillas en WebDSL suponen sentencias de código agrupadas debajo de un alias. El uso de plantillas permite hacer llamadas a dichas sentencias y ejecutarlas como un conjunto. Las plantillas permiten la entrada de argumentos. La definición de una plantilla se realiza con la palabra reservada *define*:

```
define page root()
{
      miPlantilla()
}

define miPlantilla(){
      "Esto es una plantilla"
}
```

Invocar una plantilla tiene el mismo efecto que sustituir el código dentro de la plantilla en el lugar de la llamada.

4.3. Entidades

La representación y almacenamiento de datos en WebDSL se lleva a cabo a través de las entidades. Las entidades son, pues, modelos de datos. Haciendo una comparativa con la programación orientada a objeto, las entidades son análogas a las clases.

4.3.1. Declaración de entidades

Una entidad queda constituida con un nombre y una serie de propiedades y funciones que se atribuyen a la entidad. Las propiedades tienen las siguientes características:

- El nombre de la propiedad.
- La clase de la propiedad, que puede ser un valor, una referencia o un compuesto.
  - El valor indica que la propiedad va a adquirir una valor de tipo previamente definido (cadena de caracteres, número, etc.)
  - La referencia indica que la propiedad va a adquirir un valor de otra entidad.
  - El compuesto indica que la propiedad va a tomar un valor de la propia entidad a la que se refiere. Este tipo de propiedad no se va a utilizar en el proyecto actual.
- El tipo de la propiedad, que puede ser un *String*, *Int*, *Long*, *Text*, *Secret*, compuesto, etc.
- Las propiedades pueden contener, además, una serie de modificadores que indican ciertas características. El modificador *inverse* se utilizará más adelante en este proyecto.

En el siguiente ejemplo se muestra la definición de tres entidades:

```
entity Asignatura{
      name :: String
      code :: Int
      coordinador -> Profesor
}

entity Profesor{
      name :: String
      password :: Secret
      asignatura -> Asignatura
      alumnos -> List<Alumno>
}

entity Alumno{
      name :: String
      tutor -> Profesor
      asignaturas -> List<Asignatura>
}
```

Cabe mencionar que todas las entidades han de tener una propiedad con nombre *name* y de tipo *String*. La propiedad se crea automáticamente en caso de que el desarrollador no la incluya.

Uno de los modificadores más importantes que existen para las propiedades es el *inverse*. El modificador *inverse* permite realizar una doble asignación entre entidades con una propiedad que es del tipo de una entidad.

```
entity Asignatura{
      name :: String
      code :: Int
      coordinador -> Profesor
}

entity Profesor{
      name :: String
      password :: Secret
      asignatura -> Asignatura (inverse = Asignatura.coordinador)
      alumnos -> List<Alumno>
}

entity Alumno{
      name :: String
      tutor -> Profesor (inverse = Profesor.alumnos)
      asignaturas -> List<Asignatura>
}
```

En el ejemplo descrito, si se instancia una entidad de tipo *Profesor* y se le asigna un objeto tipo *Asignatura* en la propiedad *asignatura*, automáticamente se asigna el objeto *Profesor* a la propiedad *coordinador* de la entidad *Asignatura*. Este tipo de asignación

también se puede realizar con listas de entidades, tal como ocurre con las propiedades *alumnos* de *Profesor* y *tutor* de *Alumno*. WebDSL no implementa la relación *inverse* entre dos propiedades que contengan una lista.

Las entidades, además de propiedades, también pueden contener funciones. Las funciones quedan fuera del ámbito de este proyecto.

4.3.2. Instancia de entidades

Una vez que se han definido las entidades pertinentes, estas se pueden instanciar de la siguiente forma:

```
var asig1 := Asignatura{
        name := "Asignatura1"
        };

var p1 := Profesor{
        name := "Profesor1"
        email := "profesor1@uned.es"
        asignatura := asig1
        };

var alum1 := Alumno{
        name := "Alumno1"
        tutor := p1
        asignaturas := [asig1]
        };

var alum2 := Alumno{
        name := "Alumno2"
        tutor := p1
        };
```

En el ejemplo se han creado cuatro variables (haciendo uso de la palabra reservada *var*) de tipo *Profesor*, *Asignatura* y *Alumno*. Es importante mencionar que, durante la instancia a un objeto, no es necesario asignar valor a todas sus propiedades.

Respecto al uso de la sentencia *inverse*, cabe destacar dos asignaciones:

- En la creación del objeto *p1* se utiliza el objeto de tipo *Asignatura* para dar valor a la propiedad *asignatura*. Esta asignación provoca que el objeto *asig1* tenga al objeto *p1* como valor de la propiedad *coordinador*.
- De la misma forma, aunque en la creación del objeto tipo *Profesor* no se asigna

ningún valor a la propiedad *alumnos*, ésta queda definida en la creación de los objetos *alum1* y *alum2*.

Las asignaciones declaradas durante la creación de objetos no son permanentes en la aplicación, sino que es necesario su almacenamiento en la base de datos de WebDSL. La sentencia *save* aplicada a un objeto, sirve para almacenar su definición en la base de datos de WebDSL. Tanto si el objeto ha sido creado desde cero como si ha sido modificado, la sentencia *save* guarda los cambios realizados en la base de datos para su uso futuro.

Un ejemplo es la sentencia *save* es como sigue:

```
var p1 := Profesor{
        name := "Profesor1"
        email := "profesor1@uned.es"
        };

p1.save();
```

La sentencia *save()*, aunque necesaria en WebDSL para el almacenamiento de instancias en la base de datos, no se va a utilizar en el generador automático de código. Por defecto, se establece que todos los objetos instanciados se almacenan en la base de datos, contengan o no una sentencia *save()*.

4.3.3. Sentencia *CRUD*

Las instancias de entidades que se crean en una página WebDSL son dinámicas, es decir, el valor de sus propiedades puede ser modificado durante el uso del servicio web. La creación, modificación, borrado o simplemente, la visualización de un objeto instanciado son acciones requeridas muy habitualmente por el usuario cuando se trata de manejar modelos de datos. Un método común de crear o modificar el contenido de un objeto es haciendo uso de formularios.

La sentencia *CRUD* sirve para generar, de forma automática, páginas que permiten crear, editar, manejar y visualizar los objetos instanciados de una entidad específica. El contenido de las páginas para el manejo de las entidades depende de la acción requerida. En total, se crean cuatro páginas con los siguientes nombres:

- *create*{*nombreEntidad*}: la plantilla para crear un objeto despliega un formulario

donde se pueden rellenar cada una de las propiedades de la entidad.

- {*nombreEntidad*}: la plantilla para visualizar un objeto lista las propiedades de dicha entidad y el valor asignado para cada una de ellas.
- *edit*{*nombreEntidad*}: la plantilla para modificar un objeto también despliega un formulario en el que los valores por defecto son los ya asignados anteriormente.
- *manage*{*nombreEntidad*}: la plantilla para gestionar un objeto ofrece enlaces a las plantillas anteriormente descritas.

Un ejemplo de uso de la sentencia *CRUD* queda como sigue:

```
entity Alumno{
        name :: String
        tutor -> Profesor
        asignaturas -> List<Asignatura>
        email :: Email
}

derive CRUD Alumno
/* Equivalente a:
   define page createAlumno()
   define page alumno()
   define page editAlumno()
   define page manageAlumno()
   */
```

Cada una de las plantillas creadas automáticamente con la sentencia *CRUD* se pueden sobreescribir de forma que el desarrollador pueda ajustar su funcionalidad según los requisitos de la aplicación.

```
entity Alumno{
        name :: String
        tutor -> Profesor
        asignaturas -> List<Asignatura>
        email :: Email
}

derive CRUD Alumno
/* Equivalente a:
   define page createAlumno()
   define page alumno()
   define page editAlumno()
   define page manageAlumno()
   */

define override page alumno(a:Alumno){
        "El nombre del alumno es " output(a.name)
}
```

En el ejemplo arriba expuesto, la página *alumno* ya no corresponde con la plantilla por defecto asignada en la sentencia *CRUD*. En su lugar, se imprime por pantalla el nombre del alumno.

Para acceder a estas páginas creadas automáticamente, se puede utilizar la sentencia *navigate*.

```
navigate alumno((from Alumno)[0]) {"view alumno"}
navigate manageAlumno() {"manage alumno"}
```

4.3.4. Sentencia *init*

La sentencia *init* define un conjunto de sentencias a ejecutar al comienzo de la aplicación. La sentencia *init* se suele utilizar para la declaración de variables y asignaciones iniciales necesarias para el correcto funcionamiento de la aplicación.

La sentencia *init* no permite argumentos. La ejecución de la sentencia *init* es automático al inicio de la aplicación, no necesita ser instanciada.

Así pues, la creación de objetos tipo *Asignatura*, *Profesor* y *Alumno* se pueden, y deben, incluir dentro de la sentencia *init*.

```
init{
      var asig1 := Asignatura{
              name := "Asignatura1"
              };

      var p1 := Profesor{
              name := "Profesor1"
              email := "profesor1@uned.es"
              asignatura := asig1
              };

      var alum1 := Alumno{
              name := "Alumno1"
              tutor := p1
              asignaturas := [asig1]
      };

      var alumn2 := Alumno{
              name := "Alumno2"
              tutor := p1
      };

      p1.save();

}
```

4.4. Control de acceso

El control de acceso en WebDSL supone uno de los añadidos más importantes a la estructura inicial del DSL. El control de acceso en WebDSL trabaja con entidades y plantillas, al igual que se ha mostrado en los apartados anteriores.

El control de acceso requiere de una definición de entidad que sea la encargada de manejar el inicio y cierre de sesiones, así como permitir o denegar el acceso a ciertas páginas WebDSL o parte de ellas. En la implementación del control de acceso por parte del usuario, han de definirse las siguientes etapas básicas:

1. Definición de la entidad que se encarga del control del acceso.
2. Asignación de dicha entidad al control de acceso. En esta etapa se incluye la selección de qué propiedades de la entidad se requieren como credenciales de autenticación.
3. Definición de las reglas del control de acceso para las diferentes páginas definidas.

La entidad que va a liderar el control de acceso puede tener esta definición:

```
entity User {
      name :: String
      password :: Secret
}
```

Es necesario asignar una entidad ya definida como la encargada de garantizar el control de acceso. La sentencia *principal* se encarga de esta asignación. Así mismo, la sentencia *credentials* indica qué propiedades de la entidad se requieren para el acceso. Ambas sentencias, *principal* y *credentials* van juntas en la misma definición, de forma que el modo de autenticación de los usuarios queda completamente definido. La sintaxis para esta sentencia es:

```
principal is User with credentials name, password
```

Con esta declaración se indica que, para el control de acceso, son necesarias las propiedades *name* y *password* de cualquier objeto tipo *User* guardado en la base de datos de WebDSL.

Las sentencias *principal* y *credentials* tienen las siguientes características:

- Las propiedades requeridas de la entidad para el control de acceso han de ser de tipo *String*, *Email* o *Secret*. La entidad puede contener más propiedades o funciones, pero no son válidas para garantizar la autenticación en el control de acceso.
- Al declarar la sentencia *principal* se crean automáticamente la plantilla *authentication* y la función *loggedIn*. La plantilla *authentication* permite a cada usuario iniciar y/o cerrar sesión haciendo uso de las credenciales mediante un formulario para introducir las propiedades indicadas en las credenciales. Esta plantilla indica el acceso garantizado y/o denegado haciendo una comparación entre los valores introducidos en el formulario y los almacenados en la base de datos. La función *loggedIn* indica si algún si hay alguna sesión iniciada en el sistema.

Por último, se definen un conjunto de reglas de acceso para cada una de las páginas definidas que requiera control. Para generar las reglas se utiliza la sentencia general *access control rules* y, a continuación, las reglas separadas para cada una de las páginas:

```
access control rules
      rule page miPagina1{
              reglaACumplir1()
              }
      rule page miPagina2{
              reglaACumplir2()
      }
```

Para agrupar páginas que comparten la misma regla, se utiliza la sentencia *pointcut*:

```
access control rules
        pointcut conjuntoPaginas(){
                page pagina1(),
                page pagina2(*)
        }
        rule pointcut conjuntoPaginas(){
                reglaACumplir()
        }
```

La condición que se encuentra dentro de las reglas para cada página es una variable tipo booleano que indica *true* en caso de que el acceso sea favorable y *false* si es denegado. Es común utilizar como reglas las función *loggedIn* ya definida por la sentencia *principal*.

La existencia de reglas o de conjuntos de páginas sobre los que aplica la misma regla permite definir tanto grupos de páginas con un acceso similar, como distintos tipos de acceso. Un caso de uso práctico para estas sentencias es aquel en que todos los usuarios tienen acceso a la visualización de objetos; mientras que el acceso a la gestión de dichos objeto está restringido a usuario con una sesión iniciada.

El siguiente ejemplo muestra el proceso completo a la hora de realizar un control de acceso:

```
principal is Profesor with credentials name, password

access control rules

        rule page root(){true}

        rule page profesor(*){true}

        pointcut profesorPages(){
                page *Profesor(*)
        }

        rule pointcut profesorPages(){loggedIn()}
```

En el ejemplo expuesto, no existe control de acceso para la página principal *root*, ni para la visualización de los objetos tipo profesor. Para las demás páginas que gestionan la entidad *Profesor* es necesario haber iniciado sesión con anterioridad para que el acceso este permitido. Para todas las demás páginas de la aplicación, el acceso está denegado.

5. Generador automático de código

El generador automático de código desarrollado para este proyecto tiene como fin la generación de código compatible con la infraestructura de Apache Cordova, de forma que se pueda desplegar en distintos dispositivos móviles. Para la realización de este proyecto se han utilizado como plataformas destino un navegador web y un simulador LG5 de Android.

El generador aquí propuesto va a contar con tres partes diferenciadas:

1. **Definición de entidades e instancia de objetos**: La definición de entidades se va a llevar a cabo en un fichero independiente con extensión *app*. Este fichero va a contener todas las entidades que se vayan a utilizar durante la aplicación con la estructura propia de un WebDSL. Si, para alguno de los objetos creados se requiere la sentencia *CRUD*, también ha de aparecer en este archivo. Este fichero, por tanto, tiene sólo pretensiones declarativas, sin efectuar ningún operación concreta.

   La instancia de objetos a partir de la entidades ya definidas se incluye dentro de la sentencia *init*. Todas las definiciones iniciales de objetos se incluirán en esta sentencia, aportando valor a cada una de las propiedades según corresponda a la aplicación. Estos valores asignados inicialmente se podrán modificar después en caso de uso de la sentencia *CRUD*.

2. **Elementos de entrada**: las plantillas creadas por la sentencia *CRUD* están predefinidas. El generador permite la modificación de dichas plantillas asignadas a cada una de las entidades en un fichero con extensión *app*.

3. **Control de acceso**: El uso de la instancia *principal* así como la definición de reglas estarán incluidos en un fichero independiente tambien con extensión *app*. La entidad utilizada como la encargada de definir el control de acceso ha de estar previamente definida en el fichero de las entidades. Las reglas de acceso van a estar definidas según las páginas creadas con la sentencia *CRUD*.

5.1. Entradas y salidas generador

5.1.1. Entradas del generador

El ámbito del generador hace necesaria la creación de una sintáxis específica y, en cierto modo, estricta, en los ficheros de entradar. La salida del generador en caso de que los ficheros de entrada disten de los esperados es impredecible. Los ficheros de entrada para el generador implementado son:

- *application.ini*: este fichero contiene la configuración de WebDSL. Se utiliza para conocer el nombre de la aplicación a generar. Un ejemplo del fichero *application.ini* es:

```
appname=SimpleHelloWorld
backend=servlet
sessiontimeout=120
smtphost=smtp.gmail.com
smtpport=465
smtpuser=
smtppass=
tomcatpath=/opt/tomcat
httpport=8080
httpsport=8443
db=h2
dbfile=SimpleHelloWorld.db
dbmode=create-drop
```

- *nombreAplicacion.app*: este fichero contiene la clase principal o *root* para WebDSL. Por ejemplo, este fichero puede contener:

```
page root(){
    navigate alumno((from Alumno)[0]) {"viewAlumno"}
    " "
    navigate manageAlumno() {"manage alumno"}
    " "
    navigate editAlumno((from Alumno)[0]) {"edit alumno"}
    " "
    navigate createAlumno() {"create alumno"}
    " "
    authentication()
}
```

- *data.app*: este fichero contiene el modelo de datos a convertir por el generador, es decir, las entidades con la que la aplicación va a trabajar. Se puede ver un ejemplo del mismo en el apartado 4.3.1, Declaración de entidades.

- *initilization.app*: este fichero contiene la declaración de ciertas instancias durante la inicialización de la aplicación. Se puede ver un ejemplo del mismo en el apartado 4.3.4, Sentencia *init*.

- *ac.dat*: contiene la definición del control de acceso. Se ha incluido un ejemplo completo del mismo al final del apartado 4.4, Control de acceso.

Los ejemplos aquí incluidos o citado son aptos como archivos de entrada al generador. Estos ficheros se utilizan más adelante en el apartado de demostración del generador.

5.1.2. Salidas del generador

Tanto la cantidad como el contenido de las páginas de salida generadas por el generador automático de código varía dependiendo de las entradas proprocionadas. Aún así, se pueden establecer los siguientes criterios:

- *config.xml*: es el fichero de configuración de Apache Cordova. Se puede ver un ejemplo en el siguiente apartado Configuración de Apache Cordova.

- *index.js*: fichero JavaScript donde se encuentran las funciones de acceso a la base de datos indexada, el control de acceso, etc. Este fichero es amplio y se explica en profundidad en el apartado 5.5, Modelo de datos.

- *functions.js*: fichero JavaScript, importado por el anterior, donde se encuentran el resto de funciones definidas, relacionadas con la formación de tablas y formularios a mostrar por cada una de las páginas. Al igual que *index.js*, su contenido es amplio y se desglosa en el apartado 5.5, Modelo de datos.

- Un fichero con extensión *.html* para cada una de las páginas definidas, de forma independiente. Por ejemplo, el fichero de salida para la página principal es:

```
<!DOCTYPE html>
<html>
    <meta charset="UTF-8">
    <title></title>
    <head>
       <script src="./js/index.js" type="module"></script>
    </head>
        <body id="root">
                <a href='alumno.html?val=alumn2' id="alumno">viewAlumno</a>
                <a href='manageAlumno.html' id="manageAlumno">manage alumno</a>
                <a href='editAlumno.html?val=alumn2' id="editAlumno">edit alumno</a>
                <a href='createAlumno.html' id="createAlumno">create alumno</a>
        <p id=authentication></p>
        <p id=loginResult></p>
        </body>
</html>
```

5.1.3. Relación entre entrada y salida

El flujo del generador automático se puede definir en tres partes principales:

1. Lectura de los diferentes ficheros de entrada.
2. Análisis y procesamiento de los elementos de entrada. Se utilizan variables internas para almacenar los datos necesarios para la salida del generador.
3. Generación de los ficheros de salida.

De forma general, cada elemento de entrada da lugar a un conjunto de variables internas que contiene la información útil, no sólo en contenido, sino en formato y estructura. Los datos internos se utilizan después para la generación de los elementos de salida. Existe una relación directa entre qué elementos de entrada corresponden a los distintos elementos de salida, juntos con sus variables de internas de almacenamiento.

La siguiente tabla indica la relación entre los elementos de entrada y salida del generador automático de código:

<table>
<tr><th></th><th>Entrada</th><th>Generador automático de código</th><th colspan="2">Salida</th></tr>
<tr><td>Configuracion</td><td>application.ini</td><td>Nombre de la aplicación</td><td colspan="2">config.xml</td></tr>
<tr><td>Página root</td><td>nombreApp.app</td><td>Elementos de root</td><td colspan="2">nombreApp.html</td></tr>
<tr><td rowspan="4">Modelo de datos</td><td rowspan="4">data.app

application.app</td><td></td><td>Base de datos indexada</td><td rowspan="5">index.js

function.js</td></tr>
<tr><td>Entidades</td><td>Almacen en indexedDB</td></tr>
<tr><td>Instancias</td><td>Objetos en indexedDB</td></tr>
<tr><td>CRUD</td><td>view.html
manage.html
create
edit.html</td></tr>
<tr><td rowspan="2">Control de acceso</td><td rowspan="2">ac.app</td><td>Principal y credenciales</td><td>Autenticación</td></tr>
<tr><td>Control de Acceso</td><td></td></tr>
</table>

Tabla 2: Relación entre entrada y salida del generador

5.2. Configuración de Apache Cordova

El fichero de configuración de Apache Cordova es necesario para la ejecución de la aplicación. El fichero de configuración *config.xml* ha de contener las sentencias mínimas necesarias para poder arrancar la aplicación en las distintas plataformas requeridas.

Las distintas características que recoge el fichero de configuración se mantienen fijas en la salida del generador, a excepción del nombre de la aplicación.

El contenido del fichero de configuración es el siguiente:

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="io.cordova.hellocordova" version="1.0.0" xmlns="http://www.w3.org/ns/wid
gets" xmlns:cdv="http://cordova.apache.org/ns/1.0">
    <name>MyFirstApp</name>
    <description>
        A sample Apache Cordova application that responds to the deviceready event.
    </description>
    <author email="dev@cordova.apache.org" href="http://cordova.io">
        Apache Cordova Team
    </author>
    <content src="index.html" />
    <plugin name="cordova-plugin-whitelist" spec="1" />
    <access origin="*" />
    <allow-intent href="http://*/*" />
    <allow-intent href="https://*/*" />
    <allow-intent href="tel:*" />
    <allow-intent href="sms:*" />
    <allow-intent href="mailto:*" />
    <allow-intent href="geo:*" />
    <platform name="android">
        <allow-intent href="market:*" />
    </platform>
    <platform name="ios">
        <allow-intent href="itms:*" />
        <allow-intent href="itms-apps:*" />
    </platform>
    <plugin name="cordova-plugin-console" spec="^1.1.0" />
    <plugin name="cordova-plugin-camera" spec="^4.0.3" />
    <engine name="android" spec="^7.0.0" />
    <engine name="windows" spec="~5.0.0" />
</widget>
```

Las plataformas incluidas en la configuración por defecto son:

1. Un navegador.
2. Un dispositivo tipo *Android*.

5.3. Páginas

Para la creación de páginas en Apache Cordova, se utiliza el lenguaje HTML. Cada una de las páginas definidas en WebDSL se traduce en el generador como un fichero con extensión *html* independiente. Estas páginas acceden a los ficheros JavaScript originados que contiene la lógica de la aplicación final. [13]

Por defecto, la página de inicio de la aplicación Apache Cordova coincide con el nombre de la página principal en WebDSL.

La estructura general de una página tiene las siguientes partes:

- Cabecera: la cabecera no contiene ningún nombre o título asociado. Sin embargo, en ella se encuentra la sentencia *script* que contiene nombre del archivo JavaScript a cargar al inicio de la página. Algunas páginas también incluyen código JavaScript directamente en este elemento. Esta parte se desarrolla más adelante en este mismo apartado.

- Título: para algunas páginas, dependiendo de su naturaleza, se define un título que resume el contenido de la propia página. Esta parte no se refiere al elemento *title* de HTML, sino de un texto remarcado a modo de resumen en el contenido de la página.

- Cuerpo: en el cuerpo de la página se encuentra su contenido en sí: enlaces a otras páginas, formularios, botones, etc.

Por tanto, un ejemplo de página principal quedaría:

```
<!DOCTYPE html>
<html>
    <meta charset="UTF-8">
    <title></title>
    <head>
       <script src="./js/index.js" type="module"></script>
    </head>
    <body id="root">
    //Contenido de la pagina
    </body>
</html>
```

Todas las páginas *HTML* incluyen el código JavaScript definido en *index.js*. Dentro de está página existen una serie de sentencias que se ejecutan siempre que se carga una nueva página. Para realizar dicha operación, el código JavaScrip de *index.js* está principalmente marcado por la siguiente sentencia:

```
window.onload = function(){
   //Funciones a ejecutar al inicio de cada pagina
};
```

Dentro de este código se encuentran todas las funciones a ejecutar al inicio de cada página y que, en casi todos los casos, cargan su contenido de forma dinámica.

5.4. Navegación

Los enlaces entre páginas se codifican en HTML de la siguiente forma:

```
<a href='URLenlace' id="identificador">nombreEnlace</a>
```

El generador permite la inclusión de argumentos en la navegación entre páginas. Para el caso de HTML, los argumentos se añaden a continuación de la URL del enlace, tal y como ocurre en el código HTML clásico:

```
<a href='URLenlace?arg1=val1&arg2=val2' id="identificador">nombreEnlace</a>
```

El análisis de los argumentos se realiza en el código Javascript; su uso depende de la funcionalidad de cada página.

5.5. Modelo de datos

El modelo de datos en la aplicación móvil se gestiona con código Javascript, en las páginas generadas para la lógica de la aplicación: *index.js* y *functions.js*. Al igual que en WebDSL, el modelo de datos en Javascript está construido sobre una base de datos.

Ya que Apache Cordova utiliza un navegador como contenedor para la creación de aplicaciones móviles, se ha utilizado una base de datos indexada *indexedDB* para la gestión del modelo de datos. La base de datos indexada es una API a bajo nivel que permite almacenamiento de información en el lado del cliente.

5.5.1. Base de datos indexada

La base de datos indexada se postula como una API que combina las ventajas más representativas de otros tipos de almacenamiento web, como son *WebSQL* y *localStorage*. Una base de datos indexeda contiene casi todos los beneficios de las tablas SQL, sin necesidad de definir su estructura al inicio del programa. Además, el modelo de datos de

indexedDB es simple, al igual que *localStorage*, y permite la creación de varias bases de datos de manera simultánea.

El funcionamiento de la base de datos indexada es robusto y asíncrono, por lo que la interfaz de usuario no se queda bloqueada por su uso. Su estructura es más flexible que la de *WebSQL* y soporta versiones.

Como punto negativo cabe destacar que, al tratarse de una aplicación asíncrona, se puede dar lugar a complejas estructuras de retorno cuando terminan las peticiones a la base de datos.

La base de datos indexada está soportada de forma natural por todos los navegadores actuales. Para las plataformas móviles, Apache Cordova contiene un plugin para manejar la base de datos indexada, compatible con iOS, Android, Windows, así como para navegadores más antiguos. [14] [7]

El acceso a la base de datos indexada se realiza cada vez que se carga una nueva página, ya que la mayoría de ellas necesitan de los datos almacenados para su correcto funcionamiento. El código Javascrip para el acceso a base de datos quedaría así:

```
var db;

function startDB() { //Open indexedDB

        var indexedDB = window.indexedDB || window.mozIndexedDB ||
                        window.webkitIndexedDB || window.msIndexedDB;

        var req = indexedDB.open("DatabaseName", 1);

        req.onupgradeneeded = function(evt){
                db = this.result;
                console.log("Needs upgrade");
        };

        req.onsuccess=function(evt){
                db = this.result;
                console.log("DB opened!");
        };

        req.onerror = function(evt){
                console.error("openDB:", evt.target.errorCode);
        };
};
```

Este código se encuentra en la página *index.js*, dentro de la sentencia *onload*. La variable *db* que almacena la instancia de la base de datos está declarada fuera del código de apertura de la base de datos, por lo está accesible para todas las demás funciones que requieran de la instancia. Así pues, en esta página se encuentran todas las funciones que necesitan de los datos almacenados para ejecutar su funcionamiento. [15] [16] [17]

Una de las partes más interesantes de la apertura de la base de datos son los elementos de entrada en la llamada: el nombre de la base de datos y la versión que aplica. El nombre de la base de datos coincide con el nombre de la aplicación, mientras que la versión se va a mantener constante.

5.5.2. Creación de la estructura

La creación de estructura para la base de datos se lleva a cabo cuando se carga la página principal o root. La definición de la estructura ha de estar contenida dentro del evento *onupgradeneeded*, tras la apertura de la base de datos. La ocurrencia de este evento tiene lugar:

1. Cuando la base de datos con el nombre especificado no existe en el navegador.
2. Cuando la base de datos modifica su versión.

En la aplicación de salida del generador, ya que la versión se mantiene constante, la base de datos sólo se crea cuando la página principal se carga por primera vez. El contenido de la base de datos indexada se mantiene para instancias consecutivas del navegador.

Para cada una de las entidades de WebDSL se crea un almacén en la base de datos indexada (similar a una tabla en SQL). Por cada una de las propiedades de las entidades, se crea una columna en la base de datos indexada, además de una columna extra que indica el nombre de la instancia (y además sirve como clave primaria).

Para cada una de las entidades definidas, se ejecutan las siguientes sentencias en JavaScript.

```
var transaction = evt.target.transaction;

//Create Entity
var storeEntity = db.createObjectStore('EntityName', {keyPath: 'instanceName'});

storeAsignatura.createIndex('propiedad1', 'propiedad1', {unique : false});
storeAsignatura.createIndex('propiedad2', 'propiedad2', {unique : false});
storeAsignatura.createIndex('propiedad3', 'propiedad3', {unique : false});
var Entity = transaction.objectStore('EntityName');
```

5.5.3. Inserción de instancias

La inserción de instancias en la base de datos indexada previamente definidas en WebDSL se realiza en conjunto con la creación de los almacenes. Los datos a almacenar en la base de datos son objectos Javascript (entidades con pares clave-valor), con una estructura similar a la de las entidades en WebDSL.

El objeto Javascript tiene una estructura clave-valor, que contiene el nombre de la propiedad y el valor. Las propiedades a almacenar por cada instancia son aquellas que contiene la estructura, además del propio nombre de la instancia.

Para cada una de las instancias, se utiliza la siguiente sentencia:

```
var EntidadData = [
        { instanceName : "instance1", propiedad1: "valor1Instacia1"},
          instanceName : "instance2", propiedad1: "valor1Instacia2"}];

EntidadData.forEach(function (Entity) {
        storeEntity.add(Entity);
});
```

Existen algunas características para en la inserción de objetos, listadas a continuación:

1. Algunas instancias no tiene valor para ciertas propiedades. En este caso, el objeto Javascript no contiene dicha propiedad y esta se queda vacía en la base de datos indexada.

2. Si una propiedad puede contener una lista de elementos, el valor de dicha propiedad en la base de datos indexada es una concatenación de los diferentes objetos separados por comas.

3. Una entidad puede contener a otra entidad como propiedad. En este caso, el valor de la propiedad es igual al nombre de la instancia de la entidad contenida.

5.5.4. Uso dinámico de la base de datos indexada

Las operaciones que, hasta ahora, se han descrito para acceder a la base de datos indexada son estáticas y se ejecutan al inicio de cada una de la página principal si y sólo si ésta no ha sido creada con anterioridad.

El resto de páginas de la aplicación necesitan acceder a los objectos contenidos en la base de datos de forma dinámica y según lo requiera la propia aplicación. Para manejar la base de datos se han definido cinco funciones en JavaScript estándares que realizan las peticiones necesarias a la base de datos indexada. Estas cinco funciones son:

1. **Obtención de un almacén**: Para acceder a un almacén (equivalente a una tabla en el lenguaje SQL tradicional), se realiza una petición aportando el nombre del almacén requerido y el tipo de transacción a realizar: sólo lectura o lectura y escritura.

```
//Function to select the store to access
function getObjectStore(store_name, mode) {
        var tx = db.transaction(store_name, mode);
        return tx.objectStore(store_name);
};
```

2. **Obtención de un objeto**: para obtener un objeto de la base de datos es necesario conocer el almacén en que éste se encuentra guardado, así como la clave primera del objeto, es decir, el nombre de la instancia.

   La petición de un objeto a la base de datos es una función asíncrona. Para que el retorno de la página se produzca cuando los datos han sido obtenidos se introduce un objeto *Promise* de retorno, que espera a la finalización de la petición para devolver un valor, mediante la función *resolve()*. [18]

```
var requestObject;

function getSingleFromIndexedDB(store, requestedValue){
        return new Promise(resolve => {

        var objectStore = getObjectStore(store, 'readonly')
        var request = objectStore.get(requestedValue);

        request.onerror = function(event) {
                console.log("error while getting " + defaultValue + " from " + store);
                        // Handle errors!
                        resolved(false);
                };

                request.onsuccess = function(event) {
                        requestObject = request.result;
                        resolve(true);
                };
        });
};
```

   El valor devuelto por la base de datos se almacena en una variable definida fuera de la función, de forma que los datos quedan accesibles en otros ámbitos del código.

3. **Obtención de todos los objetos de un almacén**: la función para obtener todo el contenido de un almacén es similar al de obtención de un sólo objeto.

```
var requestObjects;

//Function to return all objects in the same store
function getAllFromIndexedDB(store){
        return new Promise(resolve => {
                var objectStore = getObjectStore(store, 'readonly');

                var request = objectStore.getAll();

                request.onerror = function(event) {
                        console.log("error while getting data from " + store);
                        resolve(false);
                };
                request.onsuccess = function(event) {
                        requestObjects = request.result;
                        resolve(true);
                };
        });
};
```

El resultado de la petición se almacena en otra variable definida fuera del ámbito de la función. *Inserción de un objeto*: los datos necesarios para insertar un objeto en la base de datos son el nombre del almacén correspondiente y el contenido del objeto a almacenar. Además es necesario indicar si los datos a guardar pertenecen a un objeto ya existente en la base de datos, o se trata de un objeto nuevo.

```
//function to insert (add or update) a single object in indexedDB
function insertItemToIndexedDB(storeName, item, newItem){

        var store = getObjectStore (storeName, 'readwrite');
        var request;

        if (newItem){
                request = store.add(item);
        }else{
                request = store.put(item);
        }

        request.onsucess = function(evt){
                console.log("Item " + item.name + " inserted into indexeddb");
        };
        request.onerror = function(evt){
                console.log("Error while inserting " + item.name + " into " + storeName);
        };
};
```

4. **Borrado de un objeto**: para borrar un objeto de la base de datos, al igual que para solicitarlo, es necesario tener el nombre del almacén y la clave primaria del objeto.

```
//function to delete an object from indexedDB
function deleteItemFromIndexedDB(storeName, item){

        var store = getObjectStore (storeName, 'readwrite');
        var request = store.delete(item);

        request.onsucess = function(evt){
                console.log("Item " + item.name + " deleted from indexeddb");
        };
        request.onerror = function(evt){
                console.log("Error while deleting " + item.name + " from " + storeName);
        };
};
```

Las cinco funciones definidas se encuentran dentro del fichero *index.js* y facilitan el acceso y manejo de la base de datos para todas las demás partes de la aplicación.

5.5.5. Sentencia CRUD

La sentencia CRUD en WebDSL se traduce en el generador automático en la creación de cuatro páginas para el manejo de las instancias de una entidad concreta. Las cuatro páginas generadas son sirven para:

1. Visualización de los objetos ya creados.
2. Creación de nuevos objectos.
3. Modificación de objetos ya creados.
4. Gestión de la entidad.

Cada una de estas páginas está compuesta de tres partes fundamentales:

1. **Acceso a la base de datos indexada.** Se trata de código Javascript a ejecutar cuando se carga la página; se accede a la base de datos indexada para obtener los datos necesarios.
2. **Presentación de los datos.** El contenido de la página depende de tipo de entidad sobre el que se va a trabajar y, sobre todo de sus propiedades. En general, la parte visual de la página contiene una tabla con la información y/o formularios necesarios para su funcionamiento.

3. **Almacenamiento de las acciones.** Se trata de código Javascript a ejecutar cuando se realiza una acción concreta sobre la página. Esta acción es, en casi todos los casos, añadir, modificar o borrar un objeto de la base de datos indexada.

El contenido de cada una de las página generadas con *CRUD* es como sigue:

1. **Visualización de objetos**

   Para visualizar un objeto ya creado se accede a la página de visualización del almacén, pasando como argumento el nombre de las instancia a visualizar. Para acceder a la instancia *instance1* del almacén *entity*:

```
<a href='entity?val=instance1' id="entity">Ver entity</a>
```

   Cuando se accede a la visualización de un objeto, este se busca en la base de datos para listar sus propiedades y valores. Para buscar el contenido de una instancia en la base de datos indexada:

```
//Get argument from URL
var argURL = location.search.substr(1);

//Get object to view from IndexedDB
await getSingleFromIndexedDB("NombreAlmacen", argURL.split("=")[1]);
var instanceValue = requestObject;
```

   Cuando una de las propiedades contiene el valor de otra instancia, es necesario realizar una segunda búsqueda para conocer el nombre de esta segunda instancia. En esta búsqueda, se pretende conoce la propiedad *name* de la entidad secundaria, para mostrarla como valor de la propiedad.

   Para buscar las instancias secundarias en la base de datos indexada:

```
//Get data from Entity2 which is linked to Entity
await getAllFromIndexedDB("NombreAlmacen2");
for (var i = 0; i < requestObjects.length; i++){
        linkInstancesValue.push(requestObjects[i]);
}
```

   Cuando se llama a la función para obtener un objeto dentro de la base de datos, se utiliza la sentencia *await* para esperar el final de su ejecución. La función a la que se llama ha de devolver un objeto de tipo *Promise*.

El contenido de la página es una tabla con dos columnas, en las que se muestran los valores obtenidos de las peticiones a la base de datos. El valor de cada propiedad es aquel que se ha almacenado junto con el objeto en la base de datos indexada. Si el valor es la instancia de otra entidad con una página de visualización definida, esta se muestra como un link para visualizar esta segunda instancia; en caso contrario, el valor se muestra como un texto.

2. **Creación de objetos**

Para crear un objeto de una entidad específica, se accede a la página *createEntity.html*, sin argumentos, de la siguiente forma:

```
<a href='createEntity.html' id="createEntity">create entity</a>
```

Al inicio de la página de creación de un objeto, se accede a la base de datos para obtener un listado de las instancias que pueden formar parte de algunas de las propiedades del nuevo objeto a crear. Para acceder a esta lista de entidad se utiliza el mismo formato que durante la visualización de objetos.

El contenido de la página de creación muestra un formulario para rellenar las distintas propiedades de que se compone un objeto. Si la propiedad ha de contener una instancia de otra entidad, se genera un desplegable con las distintas opciones a elegir.

Así mismo, si una propiedad puede contener una lista de elementos, estos se pueden añadir y quitar en la página de creación a petición del usuario. Para añadir elementos de una lista de forma temporal a un objeto que está en proceso de creación, se utiliza el almacenamiento *sessionStorage*. Para almacenar una valor en *sessionStorage*, de forma reducida:

```
function addEntityPropiedad(){

        var sel = document.querySelector("#inputpropiedad");
        var item= sel.options[sel.selectedIndex].text;

        var oldItems = sessionStorage.getItem("EntityPropriedad");
        var newItems;
        if (oldItems == null){
                newItems = newItem
        }
        else{
                newItems = oldItems + ", " + newItem;
        }
        sessionStorage.setItem("EntityPropriedad", newItems);
}
```

De la misma forma, cuando un elemento se elimina de forma temporal de la propiedad de otro elemento, éste se elimina de *sessionStorage*:

```
function removeEntityPropiedad(item){

        var oldItems = sessionStorage.getItem("EntityPropiedad").split(", ");
        var itemPosition = oldItems.indexOf(item);
        oldItems.splice(itemPosition, 1);

        if(oldItems.length > 0){
                sessionStorage.setItem("EntityPropiedad", oldItems.join(", "));
        }else{
                sessionStorage.removeItem("EntityPropiedad");
        }
}
```

Este tipo de almacenaje, similar a *localStorage*, recoge una lista de los elementos que se añaden para formar parte del valor asignado a una propiedad concreta. El generador se asegura de que la información contenida en *sessionStorage* se elimina al salir de la página. De forma natural, el contenido de *sessionStorage* se elimina del navegador al salir de éste.

Estas dos funciones de inserción y eliminación de objetos temporales a una propiedad de un objeto que puede ser una lista de otros objetos se encuentran dentro de cada archivo HTML, no de los archivos con código JavaScript.

En la visualización de la página, el usuario tiene accesible un formulario en el que puede definir las propiedades de la entidad. La única propiedad no accesible es el nombre de la instancia, que es añadida de forma automática por la propia aplicación. Así, el formulario para una instancia contiene diferentes elementos según la entrada sea una cadena de caracteres, un enlace a otra instancia o una lista de objetos. Un ejemplo de formulario, con los tres casos indicados es:

```
<form>
<table id="createEntity">
        <tr><td>Propiedad1: </td>
        <td><input type="text" id="inputpropiedad1"/></td></tr>
        <tr><td>Propiedad2: </td>
        <td><select name="inputpropiedad2" id="inputpropiedad2"></select></td></tr>
        <tr><td>Propiedad3: </td>
        <td><select name="inputpropiedad3" id="inputpropiedad3"></select></td>
        <td><p id=addentity></p>
            <p><button onclick="addEntityPropiedad();">Add</button></p>
        </td></tr>
</table>
<button name="saveEntity" id=saveEntity onclick="return false;">Save</button>
</form>
```

Una vez pulsado el botón de creación de objetos, se llama a una función de Javascript que se encarga de leer los valores introducidos para cada una de las entradas del

formulario, así como de *sessionStorage* en el caso de valores listados. El enlace de este botón se define con un *eventListener* en JavaScript. Esta función accede a la base de datos para añadir la nueva instancia creada en el almacén correspondiente. Para almacenar los datos en la base de datos indexada:

```
var newInstanceName = "Entity" + (new Date()).getTime(); //unique name
var newData = {instanceName: newInstanceName};

//Add properties depending on sessionStorage
if("EntityPropiedad1" in sessionStorage){
        newData['propiedad1'] = sessionStorage.getItem("EntityPropiedad1");
        sessionStorage.removeItem("EntityName");
}else{
        newData['name'] = document.querySelector("#inputname").value;
}

insertItemToIndexedDB("EntityName", newData, true); //Insert item to indexedDB
```

Una vez está creado el objeto, el código Javascript carga automáticamente la página que corresponde a la visualización de este objeto.

```
//Link to view the just created object
window.location.href = window.location.protocol + '//'
                     + window.location.host
                     + "/entityName.html?val=" + newInstanceName;
```

3. **Modificación de objetos**

Para modificar un objeto ya almacenado en la base de datos indexada, se accede a la página de modificación, pasando como argumento el nombre de la instancia a visualizar. Para modificar la instancia *instance1* del almacén *entity*:

```
<a href='editEntity.html?val=instance1' id="editEntity">edit entity</a>
```

En general la página de modificación de objetos es igual que la página de creación de objetos, con las siguientes diferencias:

- Al igual que la visualización, al comienzo de la página de modificación se accede a la base de datos para extraer la información contenida sobre la instancia a modificar.
- Al igual que la creación, también se extrae de la base de datos indexada los posibles valores de las propiedades que contiene otra instancia.

- El contenido de la página se muestra como un formulario similar al de la página de creación, con la salvedad de que existen uno valores definidos por defecto. De hecho, la función a la que se accede es la misma para las operaciones de creación y modificación. La única salvedad es que el valor por defecto a mostrar en el formulario es nulo para la creación, pero no para la modificación.
- El botón *Update* para actualizar el objeto modificado, está conectado con un *eventListener* a la función de modificación.
- La inclusión y/o borrado de valores en las propiedades que permiten una lista de elementos también se realiza con *sessionStorage*.
- El enlace tras la modificación del objeto en la base de datos indexada también es la página de visualización de dicho objeto.

Para mostrar un objeto por defecto en el formulario de modificación, se utiliza la siguiente acción:

```
//Set default value in the form -- for 'edit'
if (defaultValue != null){
        if(!("currentInstancePropiedad1" in sessionStorage)){
                sessionStorage.setItem("currentInstancePropiedad1",
                                        defaultValue.propiedad1);
        }
}
```

4. **Gestión de objetos**

Para acceder a la página de gestión de objetos se utiliza el siguiente enlace, sin argumentos:

```
<a href='manageEntity.html' id="manageEntity">manage entity</a>
```

La página de gestión permite el acceso directo, mediante enlaces y botones, a la creación, modificación y borrado de cada uno de los objetos que pertenezca a un almacén concreto. Durante el inicio de la página, se realiza una petición a la base de datos indexada para obtener un listado de sus objetos almacenados.

El contenido de la página contiene varios elementos:

- Un enlace la página de creación de objetos.
- Un listado de los objetos de dicho almacén. Para cada uno de ellos se crear:

a) Un enlace para su visualización.

b) Un enlace para su modificación.

c) Un botón de borrado.

Los enlaces a las distintas páginas son similares a los ya explicados anteriormente.

Por cada botón de borrado se crea un *eventListener* que accede la función de borrado del objeto en cuestión:

```
//Get all objects from Profesor store
var values = [];

await getAllFromIndexedDB("EntityName");
for (var i = 0; i < requestObjects.length; i++){
        values.push(requestObjects[i]);
        }

//Set a different event Listener for each 'remove' button.
for (var i = 0; i < values.length; i++){
        document.getElementById("deleteInstance"+values[i].instanceName)
                .addEventListener("click", function(){
                deleteInstanceEntity();
        });
}
```

Cuando se pulsa el botón de borrado, el elemento a borrar se almacena en la sesión(*sessionStorage*) y se ejecuta una función de JavaScript que accede a la base de datos para buscar el elemento seleccionado. Después, el elemento se elimina.

```
var data = active.transaction(['Entity'], 'readwrite');
var objectStore = data.objectStore("Entity");
var request = objectStore.delete(deleteEntityName);

request.onerror = function (evt){
    console.error("Error while removing " + deleteEntityName + " from database");
};
request.onsuccess = function (evt){
    console.log("Element " + deleteEntityName + " removed from Entity");
};
```

Es posible que algunos otros objetos contengan una propiedad con el valor de la instancia borrado. El código Javascript realiza una búsqueda de estos objetos y los elimina de la base de datos indexada. De esta forma, el objeto borrado queda totalmente eliminado del sistema, tanto su propia instancia, como los enlaces que contienen otros objetos.

La página de enlace tras la acción de borrado sigue siendo la de gestión de los objetos.

5.5.6. Sentencia inverse

La sentencia inverse, en WebDSL, supone el enlace de los valores de dos propiedades diferentes que pertenecen a entidades distintas. De esta forma, si un objeto instancia a otro en una de sus propiedades, el segundo objeto instancia automáticamente al primero, sin que el usuario se percate de ello.

Esta funcionalidad está incluida en WebDSL de forma natural. A la hora de implementar esta sentencia, aparecen una gran cantidad de casos a tener en cuenta para abordar la generación automática de código. Es por ello que se ha decidido implementar la sentencia *inverse* en un formato reducido, de forma que no sea necesario tener en cuenta todas los casos existentes a los que puede dar lugar la implementación en WebDSL.

La sentencia *inverse* implementada tiene las siguientes características:

1. Si una propiedad se marca en WebDSL como inversa de otra (contiene la sentencia *inverse*), la segunda propiedad deja de ser editable para el usuario final. Así pues, esta segunda propiedad deja de aparecer en los formulario de creación y edición de objetos y el usuario no tiene acceso a ella.

2. Para esta segunda propiedad, no se atribuye ningún valor durante la inicialización de la base de datos y la creación de instancias.

3. Las propiedades no editables sí que están disponibles durante la visualización de un objeto. Cuando se accede a un objeto que tiene una o más propiedades de este tipo, se realiza una búsqueda en la base de datos indexada para conocer qué instancias de otros objetos contienen a ésta como valor en alguna de las propiedades inversas. Estos datos permiten pues mostrar al usuario todas las propiedades de la entidad, independientemente de si son editables o no.

Con este método, el valor de las propiedades inversas sólo se refleja en uno de los objetos guardados en la base de datos. El segundo objeto que tiene la propiedad inversa, no tiene asignado el valor del primer objeto en base de datos, sino que se realiza una búsqueda dinámica cada vez que se requiere su visualización.

La implementación de *inverse* se realiza en código JavaScript, en el archivo *function.js* y está disponible sólo en las páginas de visualización.

Así mismo, cuando se crea o edita un objeto con una propiedad inversa, es posible que ya existe otro objeto con esa propiedad definida. Los valores a elegir en la creación

o edición se ven reducidos a aquellos que aún no han sido asignados a ningún objeto del mismo tipo.

5.6. Control de acceso

El control de acceso de una página en WebDSL se realiza mediante la autenticación de usuarios y los permisos que tienen dichos usuarios para acceder a las páginas.

El control de acceso es independiente del modelo de datos. En el modelo de datos ya descrito, el acceso a los distintos enlaces y el manejo de datos ha de estar supervisado por un módulo de autenticación que otorga, en cada caso, el acceso o no a cada uno de los elementos incluidos en el modelo de datos. El uso de un control de acceso que gestione a lo diferentes usuarios supone un añadido al uso del modelo de datos y, por tanto, es opcional para el generador.

De entre todas las entidades definidas en WebDSL, existe una que es la encargada de manejar la autenticación de usuarios. Algunas de las propiedades de esta entidad, llamada *principal*, son requeridos para demostrar la validez del usuario que inicia la sesión. Esta autenticación se utiliza para verificar si un usuario del sistema tiene acceso o no a las distintas páginas generadas.

5.6.1. Autenticación

Para que un usuario pueda acceder a las distintas páginas de la aplicación a un usuario, se utiliza un formulario para introducir los datos requeridos por la autenticación y un botón de login. Cuando se pulsa el botón, se accede a una función para comprobar si los datos introducidos pertenecen a algún usuario dentro de la base de datos, es decir, una instancia dentro del almacén de la entidad principal.

```
function login(){

        var credPropiedad1 = document.getElementById("credPropiedad1").value;
        var credPropiedad2 = document.getElementById("credPropiedad2").value;

        await getAllFromIndexedDB("Principapl");

        var resultPrincipal = requestObjects;
        var loggedIn = false;

        resultPrincipal.forEach(function(result){
                //Check all possibilities to log in
                if ( credPropiedad1 == result.propiedad1 &&  credPropiedad2 == result.propiedad2 ){
                        sessionStorage.setItem("loginUser", result.name);
                        loggedIn = true;
                }
        });

    //Set an error if not logged in
    if (!loggedIn){
        sessionStorage.setItem("loginResult", "Name or Password are not correct.");
        }
};
```

Esta información se almacena en la sesión para luego ser mostrada por los elementos HTML. Si el resultado de la comprobación es correcto, aparece un mensaje de confirmación con el nombre del usuario autenticado y un botón de logout.

```
//Authentication element
//If there is already a user logged in
if("loginUser" in sessionStorage){
        document.getElementById("loginResult").innerHTML =
                "You are now logged in as " + sessionStorage.getItem("loginUser");

        document.getElementById("authentication").innerHTML = authForm =
                "<form><button id=logoutButton>Log out</button></form>";
}
```

Si el resultado de la autenticación resulta fallido, aparece un mensaje de error indicando que las credenciales introducidas no son correctas, además de volver a mostrar el formulario.

5.6.2. Accesos

El control de accesos a la página web se realiza al inicio de cada página, cuando éstas se cargan. Es un código genérico, fuera de la sentencia *onload* de JavaScript.

Una página web generada automática puede contener (o no) una regla que muestre los permisos necesarios para acceder a ella. Los permisos se clasifican en tres grupos:

1. **Acceso completo**: cualquier usuario puede acceder a la página web. No es necesario estar logueado para acceder al contenido.

2. **Acceso restringido**: sólo los usuarios logueados pueden acceder al contenido de la página web.

3. **Acceso prohibido**: ningún usuario puede acceder a la página.

Para comprobar los permisos de cada página, existe una lista, con valores clave-valor, en lo que se indica las reglas a marcar para el acceso a cada página.

```
//Set access control before loading pages.
var rules = {
        "pagina1":"regla1",
        "pagina2":"regla2"};

//Check the rule
var pathName = window.location.pathname.substring(1, window.location.pathname.length-5);
var rule = rules[pathName];
if (rule == null || rule == "false"){
        window.location.href =
                window.location.protocol + '//' + window.location.host + "/accessDenied.html";
}
else if (rule == "loggedIn" && !("loginUser" in sessionStorage)){
        window.location.href =
                window.location.protocol + '//' + window.location.host + "/accessDenied.html";
}
```

Si un usuario no puede acceder a una página, se carga automáticamente una página de denegación de acceso, que contiene un enlace a la página principal.

6. Demostración del generador

El generador automático de código implementado trata de dar cobertura a los casos de uso de modelo de datos y control de acceso ya explicado en anteriores apartados. El objetivo de este apartado es demostrar que el generador cumple con las funcionalidades expuestas en distintas plataformas móviles soportadas por Apache Cordova.

La demostración parte de un código en WebDSL con las siguientes características:

- Un conjunto de entidades relacionadas entre ellas a través de sus propiedades.
- Varias de estas entidades definen la sentencia *CRUD*.
- Página principal o *root* con llamadas a las funciones definidas por *CRUD*.
- Control de acceso para las páginas definidas, en base a una de las entidades declaradas.

Un código WebDSL con las características descritas cumple todos los requisitos necesarios que requiere la entrada para el generador automático, lo que posibilita su salida como un proyecto completo de Apache Cordova.

El archivos de entrada utilizado son similares a los descritos en el apartado 4, WebDSL como DSL y el Entradas del generador. Este código tiene las siguientes características:

- Entidades definidas. Con las entidades definidas, se aprecia que todas tienen diferentes propiedad, algunas de ellas instanciando a otras entidades. A su vez, se identifican dos pares de propiedades inversas.
- Sentencia *CRUD* asignada a varias entidades (en este caso todas las entidades contienen la sentencia *CRUD*).
- Instancias predefinidas de WebDSL. La definición previa de las instancias en WebDSL es opcional para el generador.
- Página principal o *root*. La página de inicio ofrece enlaces para crear, visualizar, gestionar y editar los objetos de tipo *Alumno*.
- Control de acceso. Los usuarios *Profesor* pueden iniciar sesión en la aplicación. La autenticación o no de usuario permite o deniega el acceso a las distintas páginas.

Se muestran a continuación el resultado del simulador para cada una de las funcionalidades implementadas:

6.1. Inicialización

Una vez se ha ejecutado el generador automático de código, se obtiene una aplicación en Apache Cordova cuya página principal queda como sigue:

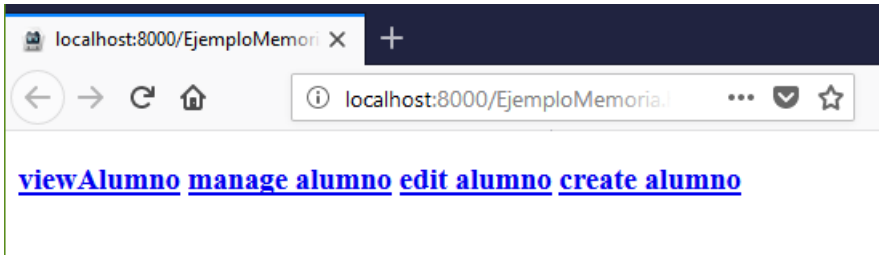

Figura 7: Página principal

Si se accede a la base de datos indexada del navegador, se aprecia que ésta se ha creado correctamente:

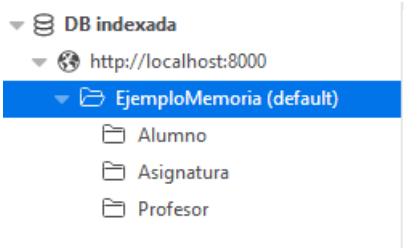

Figura 8: Creación de base de datos indexada

La base de datos indexada contiene un almacén por cada una de las entidades definidas, con todas sus propiedades:

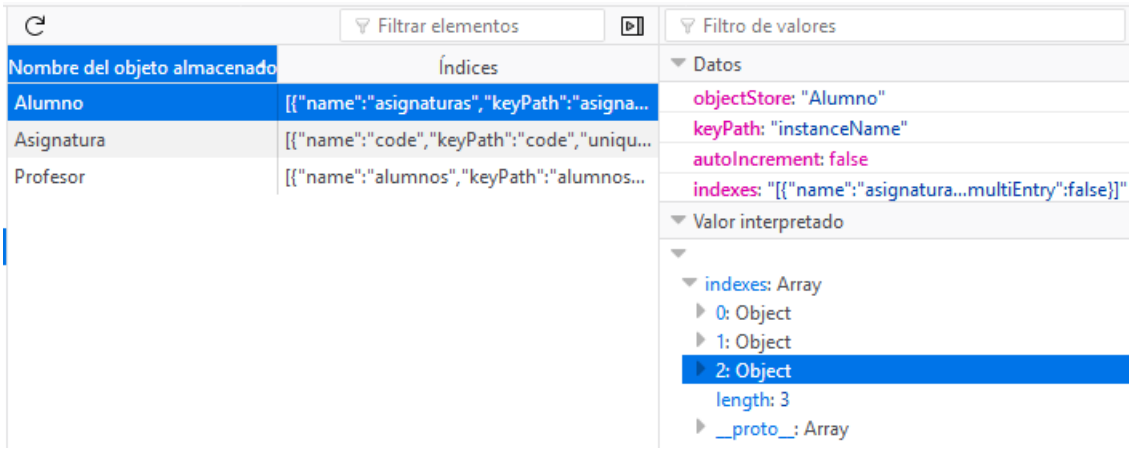

Figura 9: Almacenes en base de datos indexada

Además, para cada uno de los almacenes, se crean las instancias definidas en WebDSL como objetos independientes:

Clave	Valor
alumn1	{"instanceName":"alumn1","name":"Alumno1","asignaturas":"asig1, asig2","tutor":"prof1"}
alumn2	{"instanceName":"alumn2","name":"Alumno2","asignaturas":"asig1, asig2","tutor":"prof1"}

Figura 10: Almacenes en base de datos indexada

6.2. Objetos insertados en la base de datos indexada

Si se accede al enlace de visualización de la página principal, esta se traslada a la página de visualización de un objeto tipo *Alumno* concreto:

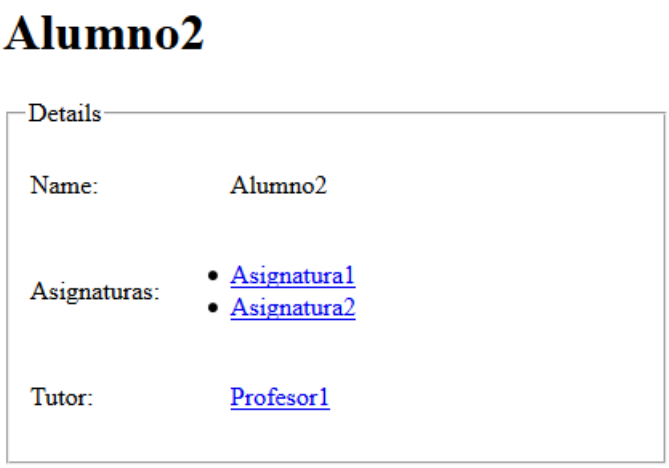

Figura 11: Vista de *Alumno*

Si alguna de las propiedades de este objeto está enlazada con otro objeto, éste aparece como un enlace para su visualización. Por ejemplo, si se pulsa en el enlace de .ªsignatura1":

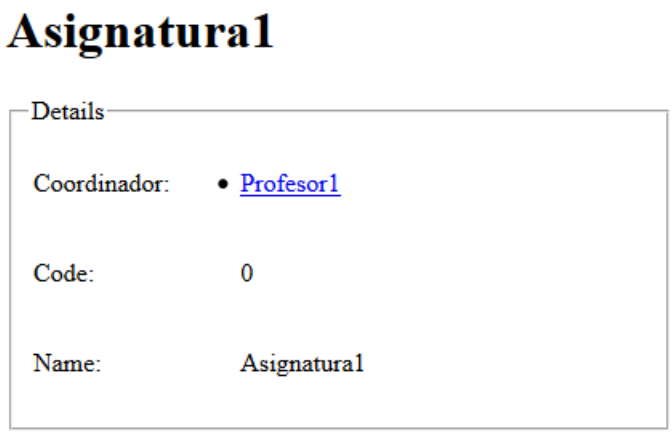

Figura 12: Vista de asignatura

6.3. Creación de objetos

En la página de creación de objeto de tipo *Alumno* tiene el siguiente contenido:

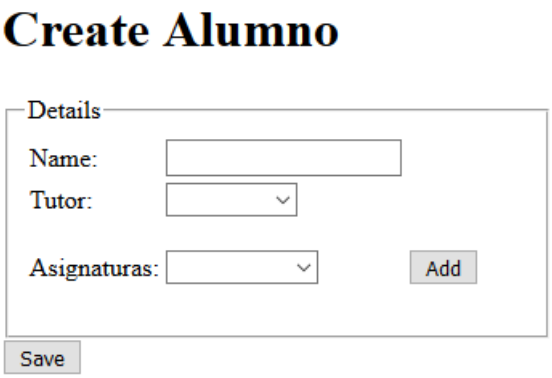

Figura 13: Página para la creación de *Alumno*

El usuario puede rellenar los campos disponible con diferentes opciones:

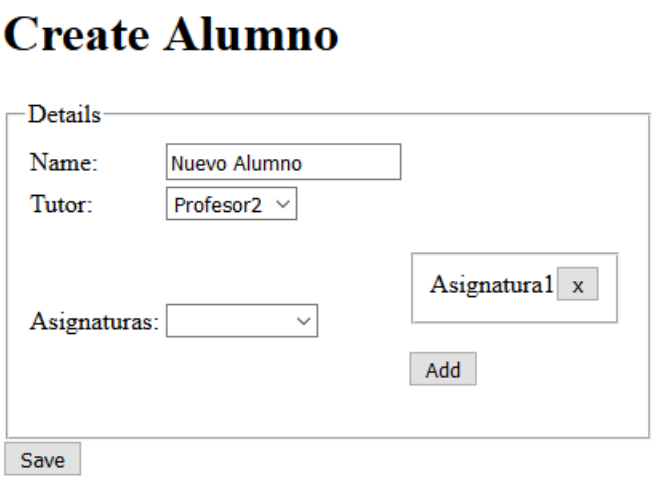

Figura 14: Inserción de valores para la creación de *Alumno*

Si se pulsa el botón de guardar, el usuario es redirigido a la página de visualización del nuevo alumno.

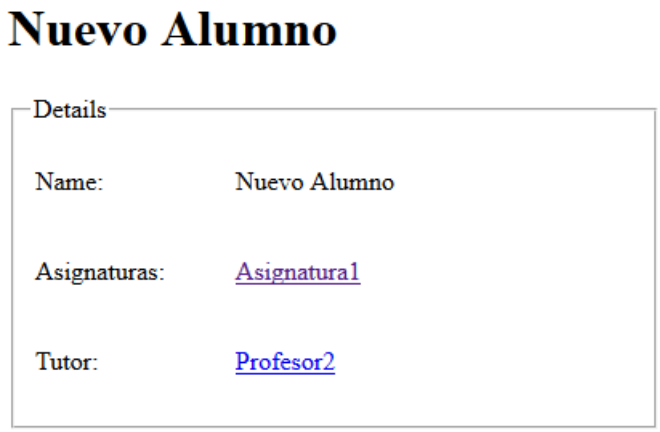

Figura 15: Vista del nuevo objeto *Alumno*

6.4. Edición de objetos

Para modificar un objeto de tipo *Alumno*, se accede a la página de edición de dicho objeto, pasándolo como parámetro. La página muestra, por defecto, los valores ya asociados a dicho objeto:

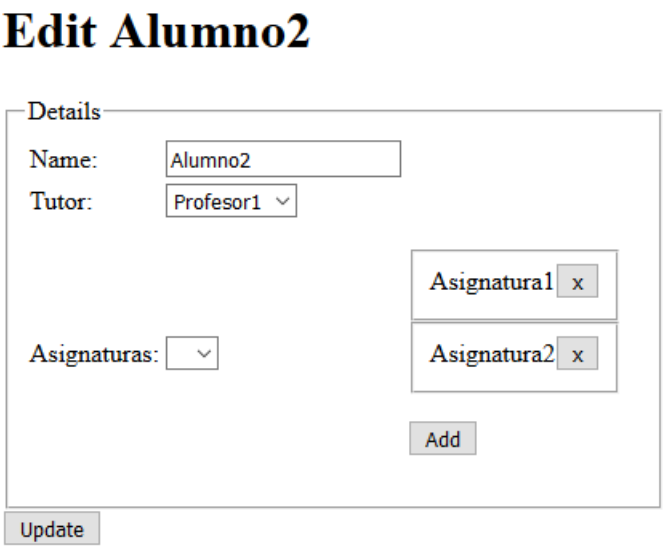

Figura 16: Página de modificación de *Alumno*

Una vez el objeto está modificado (se cambia, por ejemplo, las propiedades *Name* y *Tutor*) se pulsa el botón *Update*. Esta acción se redirije a la página de visualización del

objeto en cuestión, en la que pueden apreciarse las modificaciones:

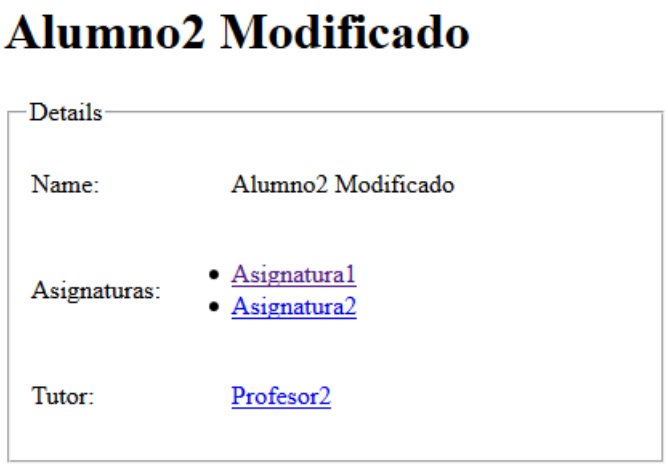

Figura 17: Vista de *Alumno* modificado

6.5. Gestión de objetos

La página de gestión de objetos de tipo *Alumno* tiene el siguiente contenido:

Figura 18: Página de gestión de *Alumno*

La mayoría de los enlaces que de la página de gestión ya se han visto anteriormente. En el caso del botón de borrado, este implica que dicho objeto se va a eliminar de la base

de datos. Así pues, si se borra el objeto *Alumno1*:

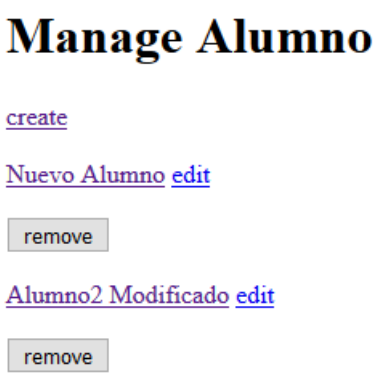

Figura 19: Página de gestión de *Alumno* tras el borrado

Al borrar un objeto, la aplicación no sólo se encarga de borrarlo de la base de datos indexada, sino de recorrer las posibles instancias que puedan tener alguna referencia a este objeto. Estas referencias también se borrar.

Como ejemplo ilustrativo, se crea un objeto nuevo de tipo *Asignatura*:

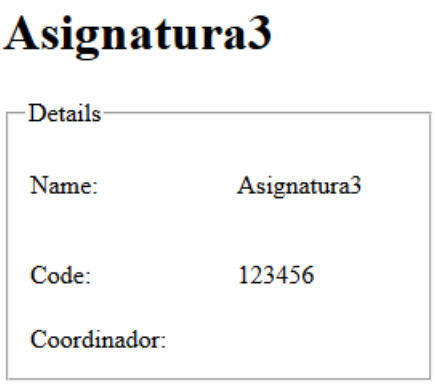

Figura 20: Objeto de tipo *Asignatura*

A continuación, se crea un objeto de tipo *Profesor*, que es el coordinador del objeto *Asignatura*:

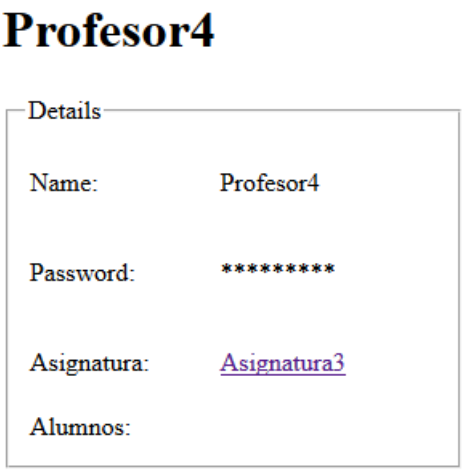

Figura 21: Objeto *Profesor* que instancia a un objeto *Asignatura*

En la página de gestión de los objetos *Asignatura*, se borra el objeto que ha sido enlazado con *Profesor*:

Figura 22: Borrado de objeto *Asignatura*

Si se retorna a la página de visualización del objeto tipo *Profesor*, se comprueba que éste ha perdido su referencia a *Asignatura*:

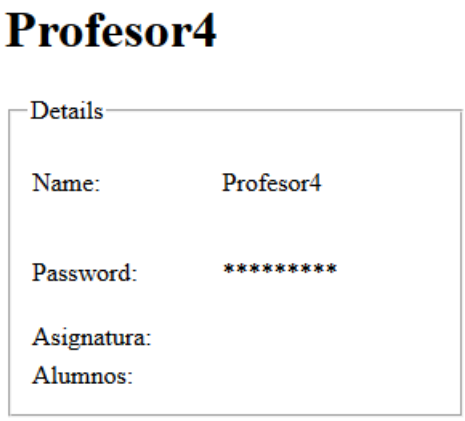

Figura 23: Objeto *Profesor* sin enlace a *Asignatura*

6.6. Inverse

La validación de la sentencia *inverse*, se visualiza primero el contenido del objeto *Profesor2*:

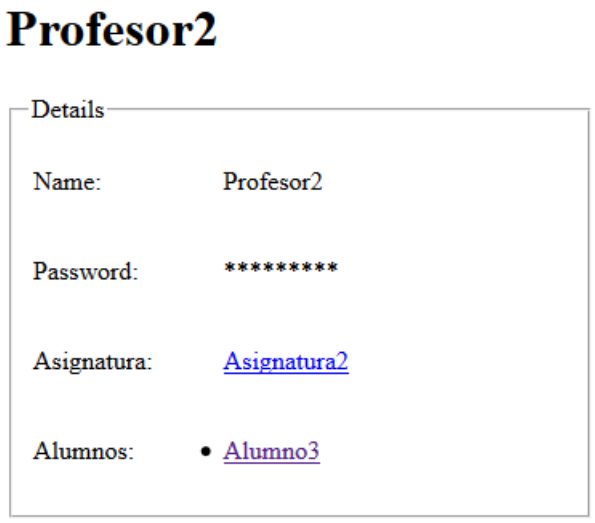

Figura 24: Vista de *Profesor*

Se crea un alumno nuevo cuya propiedad *Tutor*, sea *Profesor2*.

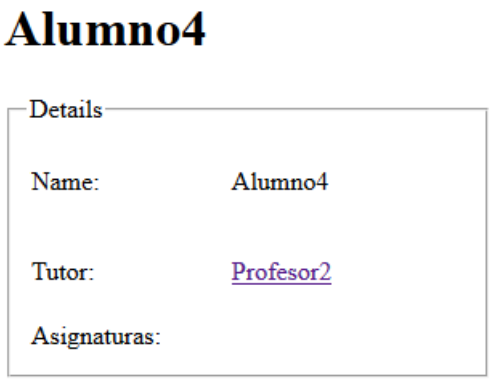

Figura 25: Vista de un nuevo *Alumno*

Cuando se accede de nuevo al objeto *Profesor2*, su propiedad *alumnos* se ha modificado, pues ahora también contiene al nuevo objeto *Alumno4*:

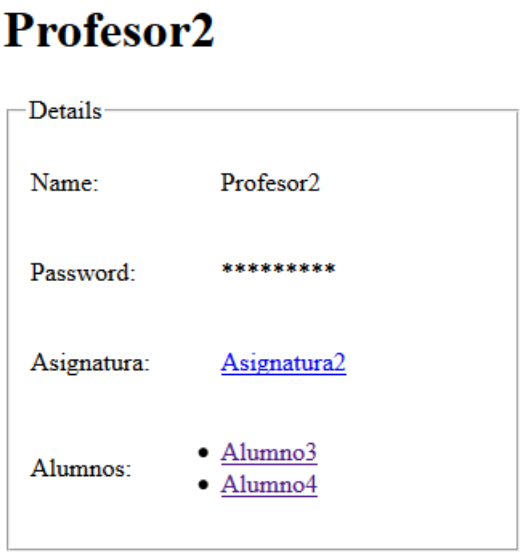

Figura 26: Vista de *Profesor2* con *inverse*

6.7. Control de acceso

En el ejemplo actual, el control de acceso viene dado por la entidad *Profesor*, por lo que autenticación de usuario se realiza con un formulario para introducir propiedades de dicha entidad. Las propiedades elegidas para autenticar a un usuario *Profesor* son el nombre y la contraseña.

En la página de inicio de la aplicación, se muestra el formulario de inicio de sesión para los usuarios. El formulario de entrada queda así:

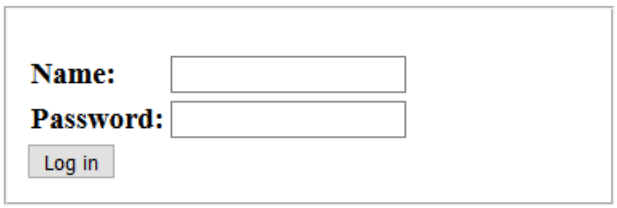

Figura 27: Formulario de autenticación

Si las credenciales son correctas, aparece un mensaje indicando qué usuario ha iniciado la sesión:

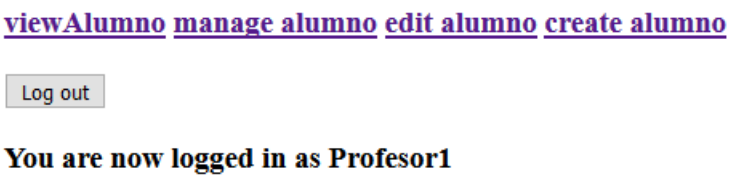

Figura 28: Inicio de sesión

En caso de que las credenciales sean incorrectas, aparece un mensaje de error:

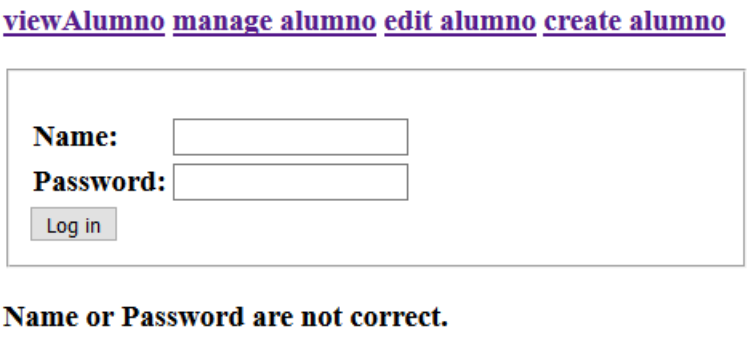

Figura 29: Inicio de sesión fallido

En cualquier caso, si se quiere acceder a una página y el usuario en cuestión no tiene

permitido su acceso, aparece una página indicado la denegación del permiso, con un enlace a la página principal:

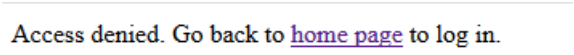

Figura 30: Inicio de sesión

7. Conclusiones y trabajos futuros

7.1. Conclusiones

Un generador automático de código a nivel general, tiene una implementación compleja, debido a la cantidad de casos de elementos de entrada (y su disposición) que se pueden dar. Como práctica clave, se establece un ámbito de alcance y unas características de entrada más o menos estrictas, que hacen posible la definición del generador.

Los conocimientos necesarios para el entendimiento, abordaje y uso de un generador automático de código fueron adquirido durante el curso de la asignatura *Generador automático de código* (en el caso de la autora de este proyecto, en el año académico 2016/2017), perteneciente al plan de estudio de este máster de investigación.

En el trabajo abordado sobre conversión de código entre el lenguaje de dominio WebDSL y la plataforma WebDSL, se ha abordado la conversión del modelo de datos y el control de acceso. El lenguaje de dominio WebDSL permite, con muy pocas líneas, definir un modelo de datos complejo, en el que las entidades pueden interactuar entre ellas, afectándose mutuamente. Así mismo, el control de acceso ofrece, en una sola página, un método conciso de administración y gestión de entrada en las distintas páginas.

La entrada para el generador automático de código elegido es el dominio web WebDSL. El formato y análisis de los elementos de entrada influyen de manera directa en la implementación del generador, así como en la disposición de los elementos de salida. La elección de cualquier otro dominio de entrada (sea web o no), da lugar a un generador automático de código para nada parecido al implementado en este proyecto.

El lenguaje WebDSL contiene un módulo central, al que pertenece el modelo de datos. Por otro lado, existen varios módulos adicionales, con nuevas funcionalidades añadidas, como el control de acceso o el control de flujo, que son independientes del módulo central pero lo complementan. Cada uno de estos módulos son independientes entre sí, por lo que pueden o no utilizarse para un desarrollo WebDSL. De cara al generador automático de código, cada uno de estos módulos puede ser implementado de forma diferente e independiente, haciendo un generador incremental que puede implementar cada uno de estos módulos (como una estructura tipo ágil).

7.2. Trabajos futuros

Se ha elegido Apache Cordova como plataforma de salida del generador. Sin embargo, al principio del documento (apartado 2.3, Comparativa), se desarrollan las diferencias entre las aplicaciones móviles multiplataforma de tipo web (a la que pertenece Apache Cordova) y las aplicaciones móviles multiplataforma de tipo pseudo-nativas (a la que pertenece, por ejemplo, Xamarin). Cómo parte del generador automático, se puede modificar la plataforma destino sobre la que está versado el generador automático. El análisis sintáctico y funcional de los elementos de entrada es el mismo en los dos casos, siendo diferente la plataforma de salida. En el caso de elegir Xamarin, o cualquier otra aplicación pseudo-nativa, hay que tener en cuenta que la interfaz gráfica no es común para las diferentes plataformas, por lo que existe un punto extra de generación de código, diferentes para cada una de las plataformas a implementar.

La necesidad de crear aplicaciones móviles de forma automática para diferentes plataformas a partir de un código común es cada vez mayor. Si se pretende abordar este problema con la generación automática de código, existen multitud de combinaciones entrada-salida para las cuales se puede implementar un generador.

8. Bibliografía

[1] Vanessa Estorach, "Desarrollo de aplicaciones móviles: nativo o multiplataforma." http://www.vanessaestorach.com/desarrollo-de-aplicaciones-moviles-nativo-o-multiplataforma/.

[2] "Apache Cordova." https://cordova.apache.org/.

[3] "Xamarin." https://www.xamarin.com/.

[4] Zef Hemel, "A Point of WebDSL." https://zef.me/the-point-of-webdsl-4c9874e07a94, 2010.

[5] E. Viser, "WebDSL: A Case Study in Domain-Specific Language Engineering," tech. rep., Delf University Of Technology, 2008.

[6] Alfonso Marín, "Alternativas para desarrollar aplicaciones moviles multiplataforma." https://alfonsomarin.com/desarrollo-movil/articulos/alternativas-para-desarrollar-aplicaciones-moviles-multiplataforma, 2016.

[7] "Plugins para Apache Cordova." https://cordova.apache.org/plugins/.

[8] "Apache Cordova Tutorial." https://ccoenraets.github.io/cordova-tutorial/.

[9] "Soporte de Plataformas para Apache Cordova." https://cordova.apache.org/docs/en/latest/guide/support/index.html.

[10] D. Groenewegen, Z. Hemel, L. Kats, and E. Visser, "WebDSL: A Domain-Specific Language for Dynamic Web Application," tech. rep., Delf University Of Technology, 2008.

[11] Zef Hemel, "Methods and Techniques for the Design and Implementation of Domain-Specific Languages," tech. rep., Delf University Of Technology, 2012.

[12] "WebDSL." http://webdsl.org.

[13] Rolando Caldas, "Cómo incluir Javascript en HTML5." https://rolandocaldas.com/html5/como-incluir-javascript-en-html5.

[14] "Almacenamiento en Apache Cordova." https://cordova.apache.org/docs/en/latest/cordova/storage/storage.html.

[15] Rolando Caldas, "IndexedDB: tu base de datos local en HTML5." https://rolandocaldas.com/html5/indexeddb-tu-base-de-datos-local-en-html5.

[16] "IndexedDB API." https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API.

[17] "Web Storage concepts and usage." https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API.

[18] "Indexed DB with promises and async/await." https://medium.com/@filipvitas/indexeddb-with-promises-and-async-await-3d047dddd313.

9. Siglas

SDK: Software Development Kit

DSL: Domain Specific Language

SQL: Structured Query Language

JSF: JavaServer Faces

HTML: HyperText Markup Language