PEC1

Propuesta formal del proyecto

M1.926 – Trabajo Final de Máster
Máster de aplicaciones multimedia | UOC

Juan Ramón Cárceles Román

Octubre 2019

Juan Ramón Cárceles Román

Propuesta de título

Aplicación para consulta, revisión y colaboración de proyectos BIM

Palabras clave

Viewer, BIM, architecture, drawings, blueprints, 3D model, documentation.

Tengo la duda de si incluir la palabra clave VisualARQ o Rhino3d ya que en principio el exportador existiría solo para ese software, no obstante lo dejo al margen para hacerlo genérico ya que en teoría podría existir exportador para otros programas similares.

Resumen de la propuesta

CONTEXTO

Rhinoceros3D + VisualARQ es un paquete de software que permite crear un modelo 3D de un edificio o infraestructura con información de sus componentes, es decir un modelo BIM (Building Information Modeling). Por ejemplo en el caso de un muro el programa te indica su composición, su área, su longitud… en el caso de una biga te indica su perfil, sus dimensiones… Gracias a toda esta información se consigue hacer más eficiente todo el proceso de diseño hasta la construcción y agiliza muchos de los cálculos necesarios.

PROPUESTA

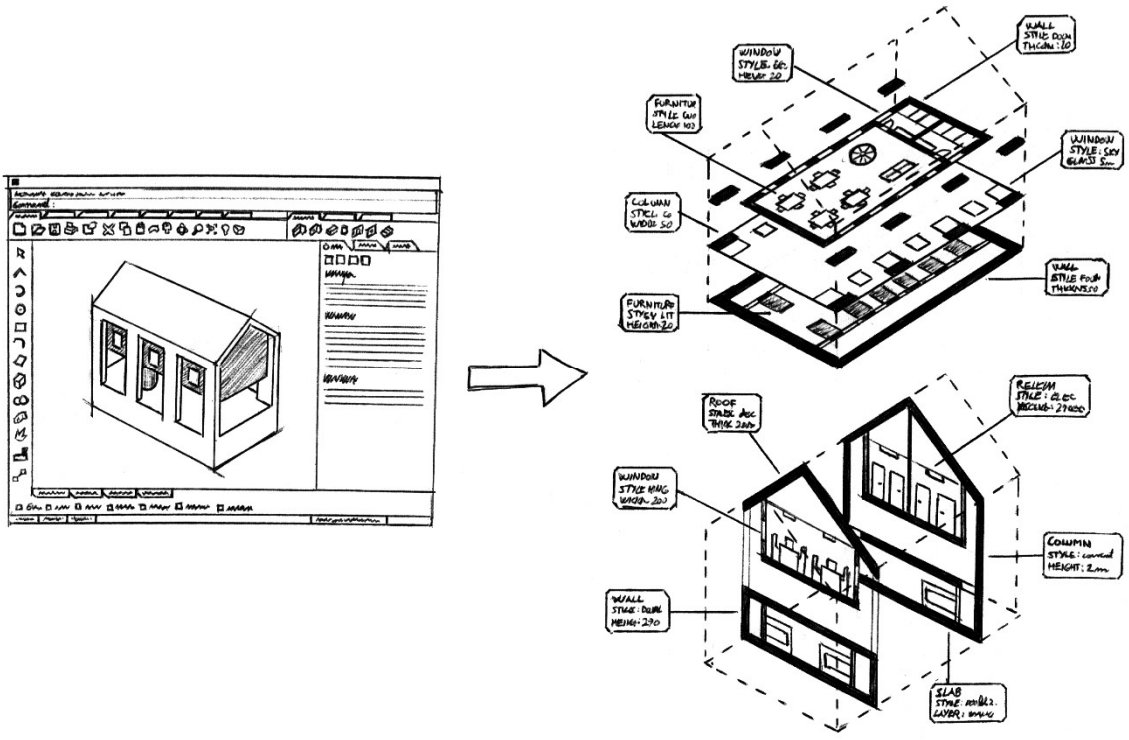
Mi propuesta de proyecto consiste en realizar una aplicación web que permita visualizar proyectos modelados en Rhinoceros3D + VisualARQ. Esta aplicación interpretará un archivo o conjunto de archivos generados por un plugin exportador. Los archivos contendrán toda la información del modelo que la aplicación web necesitará.

El objetivo principal de la aplicación es el de permitir a los usuarios y sus colaboradores visualizar el proyecto desde cualquier dispositivo, además de poder realizar tareas de consulta, revisión y colaboración. El objetivo de la aplicación web por lo tanto no es el de poder modificar el archivo original, sino el de realizar tareas paralelas.

El proyecto real por lo tanto consiste en dos partes:

- El exportador que generaría el archivo/s con todos los datos.
- La aplicación web que interpretaría el archivo/s exportado para ofrecer el servicio al usuario.

Para el TFM solo voy a tratar la parte correspondiente a la aplicación web. La parte del exportador se desarrollará más adelante. Para desarrollar la aplicación se trabajará con archivos generados manualmente que imitarán a los que se exportarían. Se trata de archivos que contendrán información en SVG y JSON principalmente.



Concepto de exportación del modelo BIM con toda la información desde Rhino + VisualARQ

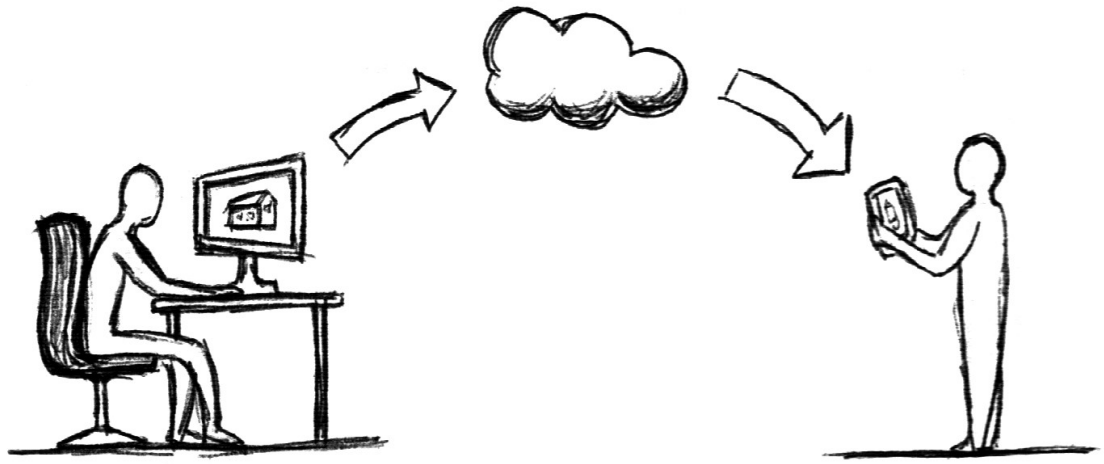
PRINCIPALES FUNCIONES

- Almacenar varios proyectos. Estos se podrán cargar desde la web o directamente desde el exportador.
- Visor 2D de dibujos vectoriales: plantas, secciones y axonometrías.
- Visor 3D del modelo.
- Consultar información de los objetos del proyecto (muros, puertas, arboles…). Es decir al seleccionar los objetos aparecería su información asociada en una tabla de datos. Se podrá realizar tanto desde el 2D como desde el 3D.
- Realizar cálculos con la información de los objetos, por ejemplo cantidad de metros cuadrados de muro del tipo XXX, o cantidad de ventanas de cada tipo.
- Medir distancias, áreas y ángulos en los dibujos. Solo disponible en los dibujos (visor 2D). Esto incluye poder hacer *snap* en los puntos del dibujo como en los programas de CAD.
- Añadir comentarios asociados a objetos los cuales servirían para indicar por ejemplo correcciones.
- Añadir comentarios generales asociados a dibujos o a todo el proyecto.
- Enviar archivo con comentarios a otros colaboradores que dispongan de la aplicación para poder colaborar a distancia.
- Responder a los comentarios que han añadido otros usuarios y crear notificaciones.
- Cambiar representación de los dibujos: color de línea, grosor…
- Imprimir en pdf los dibujos y las tablas de cálculos.
- Almacenar imágenes relacionadas con un proyecto.

- Vincular las imágenes con elementos del modelo o indicar desde donde se han tomado.
- Hacer otros tipos de indicaciones sobre los dibujos con elementos predefinidos como flechas o incluso pequeños bocetos.

Funciones avanzadas que van a ser ignoradas:

- Transmitir la información añadida mediante comentarios de vuelta a VisualARQ y que aparezcan como notificaciones en los objetos además de en una tabla.



ESTRUCTURA

La estructura de la aplicación a trazos generales constará de una pantalla con un listado de proyectos que el usuario habrá exportado y guardado en esta y una pantalla de consulta del proyecto en la cual se encontrará el visor 2D/3D con sus varias herramientas y sub apartados como la galería de imágenes.

USUARIOS OBJETIVO

El principal usuario es el usuario actual de Rhinoceros3D y VisualARQ, es decir mayoritariamente arquitectos e ingenieros. Además sería usada por otros actores relacionados a su trabajo como otros técnicos y con clientes.

MONETIZACIÓN Y GESTION DE USUARIOS

Partiendo de que se trata de una aplicación que depende de archivos generados por un exportador concreto de un software de pago y que yo ya trabajo para la empresa que lo desarrolla la aplicación web por si sola será gratuita y servirá para añadir características al software base. No obstante habrá dos posibilidades de uso:

- Totalmente libre sin necesidad de crear una cuenta. Esta opción no permitirá almacenar proyectos, tampoco añadir información en ellos ni compartirlos, sino tan solo visualizarlos y consultar información cargando el archivo a la web.
- Por otra parte accediendo con una cuenta se podrán almacenar proyectos, añadir información como comentarios e imágenes y se podrá compartir mediante un enlace con otros usuarios. Además hacer uso de una cuenta permitirá que en el momento de exportar se pueda elegir si se quiere guardar el archivo directamente en la aplicación web (introduciendo las credenciales en el momento de exportar).

Teniendo en cuenta que como he mencionado este proyecto tiene relación con la empresa en la cual trabajo y que si la primera versión que desarrollo para el TFM les interesa puede que el proyecto prospere hay aspectos que he planteado de antemano para saber la cantidad de recursos disponibles. Es por eso que con la finalidad de ahorrar y hacerlo más cercano a la realidad he decidido aprovechar el API de Google para realizar la autenticación y el almacenamiento de los datos de los usuarios. Es decir el usuario accederá a la plataforma mediante su cuenta de Google y todos los proyectos que guarde y consuma se almacenaran en una carpeta de su Google Drive específica para la aplicación (teniendo en cuenta que cada usuario dispone de 15GB gratuitos). El usuario tendrá que consentir el acceso a Google Drive pero la gestión de los archivos la realizará totalmente la aplicación.

Una opción de pago no se contempla pero se podría valorar para acceder a opciones avanzadas como poder mandar información de comentarios de vuelta a VisualARQ.

Justificación y motivación

Estudié arquitectura y hasta hace poco trabajé como arquitecto. Actualmente trabajo en una empresa de informática dando soporte y gestionando un producto de arquitectura, concretamente se trata de VisualARQ (https://www.visualarq.com/), un plugin para Rhino3d (https://www.rhino3d.com/), el cual añade características BIM (Building Information Modeling).

Este proyecto lo empecé a esbozar hace pocos meses en el trabajo ya que la idea me interesaba mucho. A dia de hoy he podido realizar algunas pruebas y me ha servido más que nada para irlo planteando bien y comentarlo en el trabajo en donde siempre ha sido bien recibido como algo que sumaría al producto actual.

PEC2

Mandato del proyecto y planificación

M1.926 – Trabajo Final de Máster
Máster de aplicaciones multimedia | UOC

Juan Ramón Cárceles Román

Octubre 2019

ÍNDICE

1. ESTADO DEL ARTE

El tema central en el cual se apoya mi proyecto es el Building Information Modeling, ya que se trata de una aplicación complementaria a un software de modelado BIM.

El BIM es una tecnología utilizada en el campo de la arquitectura, ingeniería y construcción que goza actualmente de un crecimiento continuado. El motivo es la conveniencia de aprovechar las ventajas que ofrece en comparación a los métodos de trabajo tradicionales (software CAD) más parecidos a la manera de trabajar anterior a la aparición de los ordenadores.

La tecnología BIM consiste en la creación de un modelo digital de un edificio o infraestructura el cual incluye toda la información del proyecto. Esto se consigue gracias a realizar el modelo 3D mediante elementos constructivos (muros, puertas…) que contienen información como dimensiones, materiales… Las principales ventajas de trabajar con la tecnología BIM son:

- Facilitar tareas de documentación, ya que a partir del modelo 3D se pueden obtener automáticamente los planos y secciones (se puede trabajar en 3D y 2D en cualquier momento).
- Visualizar el proyecto en 3D continuamente y así tomar mejores decisiones de diseño arquitectónico.
- Permite cuantificar información y realizar varios tipos de cálculos (mediciones, costes, cálculos energéticos…) los cuales ayudan a optimizar el proceso y el resultado.
- Aunque se trata de un aspecto avanzado también la posibilidad de colaborar es importante, de tal manera que el modelo 3D funcione como una base de datos común en la cual participen los varios actores de las varias disciplinas implicadas en el proyecto: arquitectos, ingenieros de estructuras, ingenieros de instalaciones, constructores…

Debido a la magnitud y diversidad de las tareas no basta con un solo software para poder llevarlas a cabo todas y normalmente se requiere más de uno de tal manera que entre ellos se complementen. Cada uno estará enfocado a una tarea diferente del proceso BIM.

Actualmente los softwares BIM líderes que desarrollan la parte principal del proceso (modelado) y sirven de base a otros complementarios son Revit (Autodesk), ArchiCAD (Graphisoft) y Allplan (Nemetschek).

Teniendo en cuenta que el proyecto que planteo consiste en una aplicación BIM complementaria a un software principal como los que he mencionado, voy a analizar las aplicaciones complementarias de las que dispone cada uno de estos. Además voy a incluir un par de aplicaciones independientes también relacionadas con tareas complementarias BIM pero especializadas en archivos 2D, y por último un par especializadas en visualizar archivos IFC (formato estándar en el campo AEC).

BIM 360 de Revit (Autodesk) (www.autodesk.com/bim-360/)

BIM 360 es un servicio complementario a los productos BIM de Autodesk, lo cual es casi como decir Revit (también existen otros como Navisworks). Se trata de una plataforma el objetivo de la cual es ofrecer herramientas de colaboración y gestión del proyecto. De hecho una de las características más importantes es el *Revit Cloud Worksharing*, el cual permite a todos los actores vinculados en el proyecto encontrarse en un punto.

Características destacadas:

- Acceso desde cualquier lugar (es una aplicación para dispositivos móviles). Utiliza el servicio Cloud de Autodesk para alojamiento y sincronización de proyectos.
- Almacena varios tipos de archivos del proyecto en carpetas: modelo, láminas, fotos…
- Revisión de documentos (se pueden añadir comentarios y anotaciones para notificar de errores).
- Tareas de colaboración y coordinación entre todos los participantes. Se especifica quien participa y en qué grado y tarea.
- Se puede ver el modelo 3D ver en qué parte está trabajando cada uno y que cambios realiza.
- Incluye un línea del tiempo en la cual se especifica cada equipo en qué fase actúa.

Coste:
A partir de $420/año por usuario.

BIMx de ArchiCAD (Graphisoft) (www.bimx.archicad.com/en/)

ArchiCAD por si solo dispone de una herramienta de colaboración llamada *team work*. Esta permite que varias personas trabajen sobre un mismo modelo simultáneamente. Este servicio no requiere ninguna aplicación externa pero utiliza el servicio BIM Cloud de Graphisoft.

La aplicación BIMx se trata de un complemento con el principal objetivo de poder visualizar el proyecto en dispositivos móviles, pensada tanto para los técnicos como para los clientes. Está disponible para Android y para iOS, y desde hace poco también como una aplicación web para poder utilizarla desde el ordenador. BIMx también utiliza el servicio de BIM Cloud.

Características destacadas:

- Exportación integrada desde ArchiCAD, se eligen los documentos que se quieren subir a la aplicación (será el 3D más los layouts elegidos) y estos se cargan automáticamente.
- Posibilidad de navegar el 3D como si de un videojuego se tratara, incluso con realidad virtual lo cual es muy útil de cara a mostrar el proyecto a los clientes.
- La característica principal que lo diferencia es el hecho de disponer en el 3D de botones mediante los cuales se puede acceder a los layouts. También funciona al contrario, es decir desde un dibujo se puede acceder mediante un botón en el dibujo al 3D con una vista concreta. De este tipo de enlaces entre el modelo 3D y los layouts 2D sale el nombre de *hyper-model* que es el que se le da al modelo que se exporta.
- El modelo 3D se puede seccionar dinámicamente.
- En el modelo 3D se pueden seleccionar los elementos constructivos para ver su información. Esto no está disponible en los dibujos 2D.
- Medir (distancia, área y ángulo) haciendo snap. Esto no está disponible en los dibujos 2D.
- En los dibujos se pueden añadir anotaciones y comentarios como se haría sobre un PDF.
- Servicio para guardar y compartir modelos mediante un enlace (BIMx Model Transfer).
- Los documentos cargados se pueden actualizar individualmente con nuevas versiones, no es necesario exportar todo de nuevo.

BIMx Pro es solo para acceder a características de colaboración avanzadas, es decir especificar lo que hace cada uno y sincronizar con el modelo central. Además permite comunicarse con otras aplicaciones para mandar o recibir información.

Coste:
La versión estándar es gratuita.
La versión pro cuesta alrededor de $50.

BIMPLUS de Allplan (Nemetschek) (www.bimplus.net/es/)

Se trata de una aplicación web, por lo tanto accesible desde cualquier dispositivo, el principal objetivo de la cual es que las varias disciplinas implicadas en el proyecto pueden colaborar. Sirve para implementar en Allplan un *Common Data Environment*, es decir un centro común con todos los datos del proyecto BIM. Allplan Share es el servicio cloud que utiliza para compartir datos y almacenar.

Características destacadas:

- Visor 3D llamado BIM Explorer.
- Visualización de todos los modelos en los cuales trabaja cada equipo simultáneamente. Esto permite detectar errores entre las partes, por ejemplo con el control de colisiones.
- Coordinación de tareas entre equipos mediante el *Task Board*.
- Se pueden añadir archivos complementarios como imágenes.
- Se pueden seleccionar los objetos en el 3D y ver sus datos.
- Se pueden ocultar por categorías.
- Se pueden crear anotaciones sobre una vista del 3D en concreto. Se puede especificar a quien se asigna y la fecha límite entre otros aspectos.
- Visualización del 3D como realidad virtual.
- Posibilidad de incorporar datos desde cualquier otro software mediante el API.

Coste:
El precio depende del almacenamiento, va desde 2GB gratuitamente hasta 100GB por 70€ por usuario al mes.
También dispone de Addons, como la conexión con Excel. Cada uno de los módulos extra tiene un precio de 10€ por usuario al mes.

Revu de Bluebeam (www.bluebeam.com/solutions/revu)

Descubrir esta aplicación fue para mí una sorpresa ya que trabaja específicamente con documentación 2D. Se trata principalmente de una aplicación de escritorio con una versión para iPad con menos características. El hecho de que sea principalmente para ordenador no sorprende ya que no complementa a ningún otro gran software BIM.

Se puede entender como un visor de archivos PDF especialmente pensado para cualquier tipo de dibujos técnicos ya que contiene gran cantidad de herramientas para consultar, anotar en PDFs y gestionarlos.

Esta misma empresa produce también una aplicación llamada *Bluebeam Drawings* para dispositivos móviles pero es bastante simple y no tiene mucho éxito así que no la voy a tratar.

También se vende como herramienta de colaboración con otros agentes del proyecto con lo que llaman *Studio Projects* y *Studio Sessions*:

- *Studio Projects*: permite almacenar documentación de proyectos.
- *Studio Sessions*: permite a equipos colaborar en las tareas de revisión (comentar, modificar y actualizar los archivos).

Al trabajar con dibujos en formato PDF estándar y no estar vinculado a ninguna aplicación BIM directamente los dibujos no contienen información de los objetos que aparecen, sino que se trata de líneas y sombreados como un dibujo CAD convencional.

Coste:
De $349 a $599 según la versión.

Building Information Drawings de Thorton Tomasetti y Core Studio

El estudio de arquitectura Thorton Tomasetti de Nueva York dispone de un departamento de I+D llamando Core Studio que ha creado una aplicación interna que a partir de un modelo de Revit y usando el formato DXF genera unos dibujos con información asociada a la cual se accede seleccionando los elementos constructivos que aparecen en el dibujo (de ahí el concepto de Building Information Drawings). Esto fue requerido por el estudio de arquitectura ya que les resultaba realmente útil.

Toda la información que se de este proyecto ha llegado a mí a través de los programadores de la empresa en la cual trabajo ya que han colaborado a veces en talleres realizados por Core Studio.

Visores IFC genéricos: Tekla BIMsight y Solibri Model Viewer (Nemetschek)

El formato IFC es un formato de archivo estándar para el intercambio de información entre softwares BIM. Este incluye principalmente información geométrica y de los tipos de elementos constructivos. Es también el formato requerido para entregar proyectos BIM.
Debido a la difusión de este formato en el sector no es de extrañar que existan gran cantidad de visores de archivos IFC, dos de estos son BIMsight de Tekla y Solibri MV.

Estos están disponibles solo para ordenador y a veces disponen de versiones simplificadas para dispositivos móviles pero las cuales no tienen mucho éxito.

Características destacadas:

- Este tipo de software se centra en poder visualizar el modelo 3D y consultar a su información mediante la selección de sus elementos constructivos. No incluyen dibujos 2D ya que no forman parte del IFC.
- También ofrece la posibilidad de poder esconder los elementos por categorías las cuales se corresponden con las categorías IFC. Esto suele estar disponible en cualquier aplicación que disponga de visor 3D.

Coste:
Estas aplicaciones acostumbran a ser gratuitas.

2. DEFINICIÓN DE LOS OBJETIVOS Y DEL ALCANCE

Las aplicaciones analizadas son por lo general proyectos grandes desarrollados por grandes empresas y que incluyen muchas funciones avanzadas. Mi intención no es la de intentar copiar uno de esos productos ya que no sería un objetivo realista, sino más bien entender cuáles son las características más importantes que ofrecen y pensar cómo estas pueden ser aplicadas de manera sencilla en mi TFM para crear una aplicación que disponga de estas.

Las más importantes son:

- Disponibilidad
- Visualización
- Colaboración
- Consulta
- Revisión

La disponibilidad en cualquier lugar es una característica muy importante ya que sin esta opción todo el resto no tendría sentido. En mi caso esto se conseguirá creando la aplicación como una página web, a poder ser instalable. Además tendrá que ser *responsive* y mostrar un formato adecuado para poderla usar cómodamente tanto desde un Smartphone como desde una Tablet. El almacenamiento de los archivos de cada usuario se hará mediante el API de Google Drive, la cual al mismo tiempo me resuelve el aspecto de la autenticación.

La visualización del proyecto es un aspecto indispensable que en las aplicaciones directamente vinculadas a un producto BIM, como son las tres primeras que mencioné (BIM 360, BIMx y BIMPLUS), siempre está presente tanto en 3D como en 2D. En mi caso tengo la intención de centrarme más en la documentación 2D ya que creo que esta es especialmente importante para los implicados en la fase constructiva ya que tratándose de dibujos previamente preparados para mostrar una parte específica del edificio resultan más útiles técnicamente que el 3D. Además en las tres primeras aplicaciones comentadas los dibujos son tratados como simples PDFs sin funcionalidad añadida. Es por este motivo que mi intención es crear una documentación 2D que tenga funciones avanzadas más parecidas a las dos últimas aplicaciones analizadas. Estas características las mencionaré en el apartado de consulta y en el de revisión.
La vista 3D también estará presente, ya que de cara a tener una idea general o a mostrar el proyecto al cliente es importante, pero no será desarrollada más que a nivel de prototipo.

La colaboración es el aspecto más importante que tratan prácticamente todas las aplicaciones BIM. Es un tema muy importante ya que es muy normal trabajar en equipo y las herramientas de colaboración pueden hacer que el trabajo sea mucho más eficiente. Desgraciadamente son temas complejos de desarrollar ya que requieren la sincronización de varios usuarios sobre una base de datos centralizada a la cual puedan acceder todos para aportar su trabajo y poder ver lo que hacen los demás. Esta característica requiere de un buen servicio de backend con base de datos y por eso cada una de las empresas mencionadas dispone de uno. En mi caso ya se de antemano que no voy a disponer de este servicio en el trabajo y por lo tanto como alternativa utilizaré el servicio de Google Drive mediante su API. Si después de la primera versión el resultado es bueno probablemente consiga convencer a la empresa de empezar a implementar un servicio de almacenamiento propio. El aspecto de la colaboración se puede implementar a varios niveles, lo importante es que esté presente. Mi intención es aplicarlo inicialmente a una escala pequeña para compartir tareas de revisión que comentaré más adelante y para compartir archivos.

La consulta de datos del modelo es otro aspecto importante. Esta es más fácil de implementar que la colaboración y mi intención es que sea una de las características destacadas de mi TFM. Como he comentado en el apartado de visualización, los dibujos 2D tendrán funciones avanzadas y una de ellas será la consulta de información de los elementos que se encuentren en este, es decir se podrán seleccionar los objetos de los dibujos (ej: ventana) y se verá en una tabla todos sus datos como el área, el tipo… Otra función relacionada con la consulta será la posibilidad de medir distancias, áreas y ángulos. Ni la consulta de información ni las herramientas de medición estarán disponibles en el 3D. Por último habrá un apartado de consulta de datos del proyecto el cual no requerirá de ningún tipo de vista. Con este se podrán realizar cómputos como por ejemplo la superficie total de muros de un tipo.

La revisión de los documentos de forma activa es el último aspecto importante que voy a tratar. La finalidad es la de poder indicar si hay errores y poder opinar sobre ellos. Todas las aplicaciones disponen de esta función, cada una en un nivel diferente. Lo más típico es que esta función se lleve a cabo como si de un trabajo manual se tratase, es decir sobre un dibujo poder añadir anotaciones. En mi caso quiero poder aprovechar la posibilidad de disponer de la información de cada objeto en los dibujos para poder realizar anotaciones específicas sobre un objeto o sobre más de uno, es decir etiquetándolos con un comentario. Además se podrán añadir comentarios no ligados a objetos concretos sino a partes del dibujo indicadas con una forma geométrica. A los comentarios se les podrá aplicar un nivel de prioridad.
Para que esto funcione correctamente es importante que desde la aplicación web se pueda guardar la nueva información de los comentarios que se haya añadido. Es aquí donde entra en mi proyecto la necesidad de poder almacenar información generada por el usuario, lo cual se hará mediante el API de Google Drive. Además estará disponible la posibilidad de poder compartir archivos para ofrecer funciones de colaboración. La colaboración estará basada en la posibilidad de mandar el dibujo con las anotaciones y que otro usuario pueda modificarlas y/o contestarlas.

Resumen de los objetivos:

- Implementar una aplicación web en la cual se pueda almacenar documentación de proyectos BIM realizados con Rhino3D + VisualARQ.
- La aplicación ha de ser fácilmente utilizable desde dispositivos móviles, especialmente tabletas.
- Permitir visualizar el modelo 3D y los dibujos 2D.
- Permitir obtener la información de los elementos que componen los dibujos 2D.
- Permitir realizar tareas de medición sobre los dibujos 2D (distancia, área y ángulo).
- Permitir almacenar documentos extra relacionados con el proyecto como imágenes.
- Permitir añadir comentarios a los dibujos ya sea marcando un elemento concreto o simplemente marcando una zona.
- Permitir compartir con otros usuarios dibujos con comentarios para que puedan colaborar.

Sobre el alcance:

La interfaz gráfica se desarrollará por completo con HTML y CSS. Cualquier aspecto que tenga que ver con el 3D y con archivos extra como imágenes aparecerá pero no será funcional. Tan solo lo será todo lo relativo a la documentación 2D, es decir la consulta de información de los elementos y las herramientas de medición. Por último sí que se implementará el sistema de almacenamiento mediante el API de Google Drive.

3. PLANIFICACIÓN

	PEC 3							PEC 4							PEC5					
	15 Octubre - 11 Noviembre (28 días)							12 Noviembre - 9 Diciembre (28 días)							10 Diciembre - 3 Enero (25 días)					
	15 - 18	19 - 22	23 - 26	27 - 30	31 - 3	4 - 7	8 - 11	12 - 15	16 - 19	20 - 23	24 - 27	28 - 1	2 - 5	6 - 9	10 - 13	14 - 17	18 - 21	22 - 25	26 - 29	30 - 3
ARQUITECTURA DE LA APLICACIÓN																				
DIAGRAMAS DE NAVEGACIÓN																				
ESQUEMA WIREFRAMES PANTALLAS																				
INTERFAZ GRÁFICA PANTALLAS (HTML + CSS)																				
PRUEBA VISUALIZACIÓN DOCUMENTOS 2D Y 3D																				
PRUBEA GOOGLE API PARA AUTENTICACIÓN Y ALMACENAMIENTO																				
CONSULTA INFORMACIÓN OBJETOS (2D)																				
DESARROLLO HERRAMIENTAS MEDICIÓN EN DIBUJOS																				
AÑADIR COMENTARIOS EN LOS DIBUJOS																				
GUARDAR INFORMACIÓN DE COMENTARIOS																				
COMPARTIR DOCUMENTOS CON INFORMACIÓN AÑADIDA																				
TEST CON USUARIOS REALES Y REVISIÓN																				
DETALLES GRAFICOS FINALES																				
DETALLES IMPLEMENTACIÓN FINALES																				
MEMORIA FINAL																				
DOCUMENTO PRESENTACIÓN																				

4. BIBLIOGRAFÍA Y RECURSOS

Rodríguez, José Ramón. Definición de objetivos (2.1). En ***La gestión del proyecto a lo largo del trabajo final***. (pp.10-11). Universitat Oberta de Catalunya.

Rodríguez, José Ramón. Definición del alcance (2.2). En ***La gestión del proyecto a lo largo del trabajo final***. (pp.11-13). Universitat Oberta de Catalunya.

Rodríguez, José Ramón. Planificación (3). En ***La gestión del proyecto a lo largo del trabajo final***. (pp.17-23). Universitat Oberta de Catalunya.

What is the state-of-the-art of BIM (Building Information Modeling) today?
https://www.youtube.com/watch?v=cgSeRxTKteU

BIM Software Tools for all Occasions
https://www.e-zigurat.com/blog/en/bim-software-tools-for-all-occasions/

The Ultimate BIM Software List for 2019
https://www.lodplanner.com/bim-software/

PEC3

Entrega 1. Informe de trabajo

M1.926 – Trabajo Final de Máster
Máster de aplicaciones multimedia | UOC

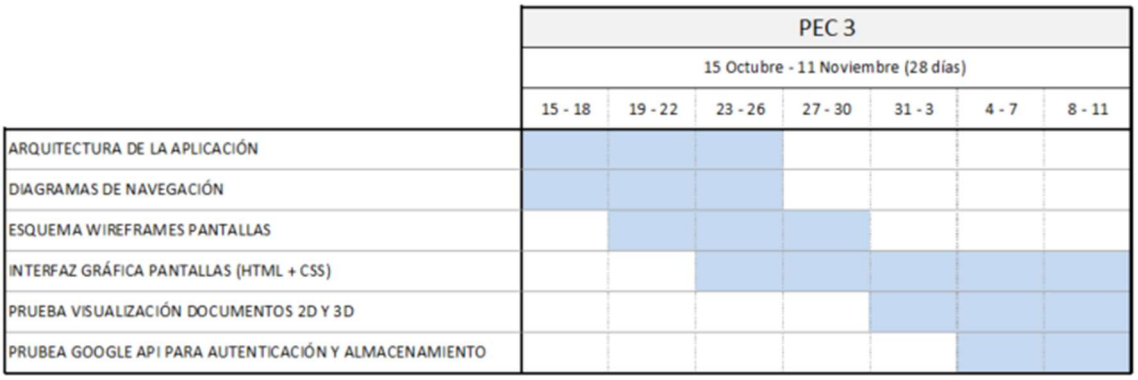
Juan Ramón Cárceles Román

Noviembre 2019

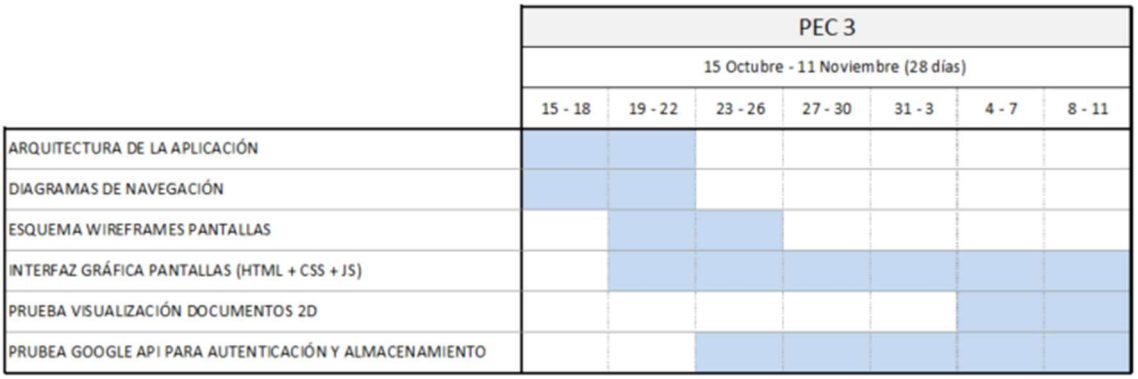
ÍNDICE

1. TAREAS DE LA PLANIFICACIÓN REALIZADAS EN LA ETAPA

La planificación prevista para la etapa era la siguiente:



Esta ha sido aproximadamente la versión real de la planificación una vez finalizada la etapa:



La duración y fechas de inicio y fin pueden ser consultadas en la imagen superior. A continuación hago una breve descripción sobre cada una de ellas.

Nombre	Estado	Descripción
Arquitectura de la aplicación	Concluida	A esta tarea le he dedicado menos tiempo del previsto. No porque no sea importante sino porque ha sido una tarea que se ha mezclado continuamente con la de diseño de las pantallas y la del inicio del desarrollo. Solo a medida que han ido avanzando las demás he ido clarificando mis ideas. En cuanto a la estructura de archivos la he mantenido simple, y por ahora no he hecho uso de ningún *bundler*, ni tampoco de SASS ni de Typescript como tenía previsto.
Diagramas de navegación y diseño de la interfaz gráfica con *wireframes*	Concluida	He unificado estas dos tareas ya que han formado parte de la misma sesión de trabajo. Debido a que la aplicación presenta mucha más dificultad en su funcionamiento que en su estructura de pantallas la duración de esta tarea se ha reducido bastante.
Desarrollo con HTML, CSS y JavaScript	Casi concluida	Esta se ha solapado más de lo previsto con el diseño de los *wireframes*. Lo he llevado en paralelo para ir comprobando si lo que dibujaba era más o menos fácil de desarrollar. Se ha extendido hasta el final de la etapa tal y como tenía previsto.

		Ahora aún faltan varios aspectos que espero concluir durante la próxima PEC.
Visualización documentos 2D	En curso	La prueba de visualización de documentos ha ocupado menos ya que lo he reducido a archivos muy simples como representación simbólica de los que deberá crear el exportador. Estos tienen la misma estructura para poder comprobar que la aplicación funciona. En cuanto al contenido de los archivos será durante la próxima PEC cuando añadiré ejemplos de arquitectura más interesantes, por ahora se ha limitado a archivos muy simples con pocas formas geométricas. En cuanto al tema de archivos 3D, este se ha descartado debido a que la aplicación se vuelve demasiado compleja para el poco tiempo del que dura el TFM y además ya hay suficientes aspectos relacionados con el 2D que hay que resolver.
Conexión con API Google Drive	En curso	La prueba del API de Google Drive empezó antes de lo previsto ya que necesitaba entender mejor su funcionamiento para prever mejor lo que podía implicar para las demás tareas. Me he dado cuenta que había subestimado su importancia y voy a tener que dedicar tiempo a lo largo del resto del proyecto ya que la correcta gestión de los archivos es importante.

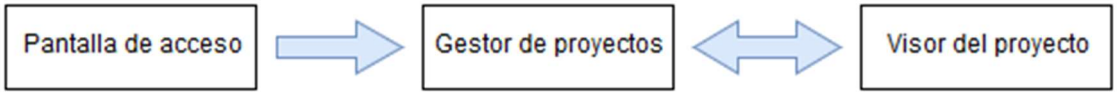
En general concluyo que todas las tareas se han modificado en mayor o menor medida. Unas se han fragmentado, otras se han unificado y algunas se han desarrollado durante toda la etapa de manera puntual. Preveo que esto será lo que pasara también con las tareas de las siguientes etapas.

2. ESTRUCTURA Y PANTALLAS DE LA APLICACIÓN

2.1. Flujo de la aplicación

El flujo de la aplicación es bastante simple ya que trata de imitar lo que sería un programa tradicional de un ordenador de escritorio. Es en la pantalla del visor del proyecto en la cual se encontrará la mayor parte de la lógica de la aplicación.

Existen tres pantallas o vistas diferentes:



Pantalla de acceso	**Gestor de proyectos**	**Visor del proyecto**
Esta pantalla tan solo incluirá tres opciones. La primera será la autenticación del usuario la	Esta pantalla incluye principalmente el listado de proyectos del usuario. Además es desde esta pantalla desde	Esta es la pantalla más importante de la aplicación ya que es desde la cual se realizan las operaciones de

cual se indicará que se tendrá que hacer mediante una cuenta de Google. La segunda opción será la de subir un proyecto para visualizarlo sin necesidad de acceder y por lo tanto sin opción de poder guardarlo. La última será la posibilidad de consultar un proyecto de muestra.	donde se llevará a cabo la operación de carga de un nuevo proyecto a la aplicación. Se accederá desde aquí también a acciones generales de cada proyecto como cambiarle el nombre o eliminarlo, las cuales también deberán estar disponibles desde dentro del propio proyecto.	consulta sobre los documentos de un proyecto. Incluye el *viewport*, la lista de dibujos del proyecto, la barra de herramientas y un cuadro en el cual se mostrará información si es requerido por la operación llevada a cabo.

Teniendo en cuenta las tres posibles pantallas me surgió la duda de como estructurar el proyecto, es decir de si crear tres páginas diferentes o si crear todo en una sola página.

Después de valorar los pros y los contras me decidí por desarrollar toda la aplicación en una sola página ya que para el usuario resulta más cómodo, especialmente se agiliza el paso del gestor de proyectos al visor, ya que es posible que mientras uno trabaja en un proyecto necesite acceder al gestor de proyectos varias veces. Así se evita tener que cerrar el proyecto para acceder a la lista de proyectos en otra página.

Además en el caso de querer hacer uso de las flechas del navegador se podría igualmente modificando la historia del navegador dentro de la misma página.

Por último en el caso de querer ver varios proyectos a la vez usando varias pestañas del navegador simplemente habría que introducir la url específica de cada proyecto en una pestaña nueva.

2.2. Prototipos de las pantallas de la aplicación

Para realizar los *wireframes* he utilizado sobre todo papel y lápiz y finalmente Balsamiq en su versión de prueba.

Consultar la carpeta Wireframes para ver los archivos de imagen creados. Estos corresponden a cada una de las tres pantallas mencionadas para la versión para ordenador y para teléfono, y dos más del visor de proyectos con un desplegable abierto.

El mayor cambio en la versión para Smartphone (menos de 500px) es que la barra de navegación se sitúa debajo y el contenedor del *viewport* arriba. Esto lo he hecho así con el objetivo de acercar los botones a la parte inferior, cercano al dedo pulgar cuando el teléfono se coge con una sola mano.

3. DESARROLLO ESTRUCTURA E INTERFAZ (HTML, CSS Y JS)

La estructura HTML se compone de una barra de navegación (*header*) y de un cuerpo (*main*). Además existen otros contenidos secundarios posicionados de manera flotante que aparecen solo cuando son requeridos como cuadros de dialogo emergentes.

3.1. El contenedor de dibujos

Dentro del *main* es donde se encuentra el contenedor de los dibujos que es uno de los elementos más importantes. Este lo he intentado implementar de la manera más sencilla posible. La gestión de los dibujos (SVG) se realiza de la siguiente manera:

El dibujo solicitado desde la lista de dibujos del visor se pide mediante el API, una vez este se recibe se guarda su contenido (SVG) en un objeto que alberga varios datos del proyecto y finalmente el contenido del dibujo se añade al contenedor HTML. Cualquier otro dibujo que sea solicitado seguirá el mismo proceso y por lo tanto será añadido al objeto con los datos del proyecto y al contenedor. Cuando en el contenedor ya hay más de un dibujo o cuando se solicite un dibujo que ya se solicitó antes, lo que se hará es ocultar el anterior y mostrar el actual mediante css.

Otra opción que plantee fue la de igualmente disponer del objeto que almacene los datos del proyecto pero sustituir cada vez el contenido del contenedor HTML con el contenido del dibujo actual, es decir que dentro del contenedor solo pudiese haber un dibujo. Finalmente descarté esta opción ya que más adelante tendré que añadir los comentarios a los dibujos, los cuales serán contenido que incluirá *event listeners* propios y por lo tanto sería muy costo cada vez que se esconde o muestra un dibujo tener que quitar y ponerlos todos.

3.2. La lista de proyectos

Para la distribución de los ítems de la lista de proyectos utilicé css *flexbox*. Esté me ayudó mucho pero me encontré con la dificultad de distribuir correctamente los ítems de la última fila cuando esta no estaba completa, ya que estos tomaban una dimensión diferente al resto. Tras comprobar en internet que se trataba de un problema propio de *flexbox* decidí utilizar JavaScript para solucionarlo.

La solución consiste principalmente en la creación de un último *seudo elemento* con ancho equivalente a todos los que faltan para completar la fila. Complementariamente mediante *media queries* especifico cuantos ítems han de caber en horizontal.

Esto parece haber funcionado correctamente, el único problema que he encontrado es que en Edge el *seudo elemento* no se crea cuando la pantalla es mayor de 1000px. Curiosamente el último *media quey* que actua sobre este aspecto es hasta los 1000px y a partir de ahí hay un valor fijo de 6 ítems en horizontal *(--items-h: 6*), no obstante sigo sin poder solucionarlo.

```
#projectsList {
  display: flex;
  flex-wrap: wrap;
  margin-left: 1rem;

  --items-h: 6; /* Read with JS */
  --item-width: calc((100% / var(--items-h)) - 1rem);
  --remaining-items: unset; /* it will be updated with JS */
}

/* Values read with JS */
@media (max-width: 1000px) {
  #projectsList {--items-h: 5;}
}

@media (max-width: 800px) {
  #projectsList {--items-h: 4;}
}

@media (max-width: 600px) {
  #projectsList {--items-h: 3;}
}

@media (max-width: 400px) {
  #projectsList {--items-h: 2;}
}

@media (max-width: 280px) {
  #projectsList {--items-h: 1;}
}

#projectsList::after {
  content: "";
  flex-grow: var(--remaining-items);
  width: calc((var(--item-width) + 1rem) * var(--remaining-items));
}
```

4. CONEXIÓN API GOOGLE DRIVE

Utilizar el API de Google Drive ha resultado muy útil ya que ha resuelto a la vez el aspecto de la autenticación de los usuarios. Además la librería que proporcionan simplifica mucho cualquier tarea relacionada con el API.

Para aprender a utilizarla he consultado la documentación oficial, la cual incluye varios ejemplos y he consultado repetidamente Stackoverflow cuando esta no era suficientemente clara. Un aspecto interesante es que las carpetas se consideran archivos también, solo que con un formato propio.

4.1. Permisos

Uno de los aspectos incomodos para el usuario de usar Google Drive es el hecho de que la primera vez que accede a la aplicación con su cuenta de Google se le solicitara que de varios permisos a la aplicación para que esta puede leer, modificar y editar archivos de su drive. Esto podría echar atrás a los usuarios. Lo mejor será aprender a pedir permisos de forma progresiva según el usuario lo necesite y de esta manera entenderá porque se le está pidiendo eso.

4.2. Funciones del API implementadas

Las operaciones básicas que he implementado durante esta etapa han sido:

- **Listar archivos**: a partir de una *query* con varios parámetros esta retorna un *array* de objetos con los datos solicitados, por ejemplo nombre e id.
- **Obtener contenido de un archivo (de tipo texto)**: a partir de un id devuelve el contenido, este lo utilizo para archivos JSON y SVG los cuales llegan como un *string*.
- **Obtener datos de un archivo específico**: a partir de su id, se reciben los campos que se necesiten de los que pone a disposición el API. Son datos del archivo pero no su contenido.
- **Creación de una carpeta**: a partir de su nombre y padre. Devuelve su id.
- **Carga de un archivo**: esta ha sido una de las más difíciles ya que el ejemplo de la documentación no es fácil ya que incluye tres opciones según el tamaño del archivo. Finalmente encontré en Internet un ejemplo con el método para archivos pequeños, de hasta 5MB. Más adelante (probablemente después del TFM) tendré que implementar otro método que permita más capacidad, sobre todo cuando haya una función para cargar imágenes del proyecto.

Las más importantes que me faltan por implementar son:

- **Eliminar un archivo o carpeta**
- **Actualizar un archivo**

A partir de las funciones básicas mencionadas he creado por ahora tres funciones que incluyen varias de estas para realizar tareas más complejas específicas de la aplicación. Estas funciones las he tenido que crear también como asíncronas ya que siempre alguna parte de la aplicación las ha de esperar para dar una respuesta al usuario. Las funciones son:

Creación de un proyecto, a partir de un archivo subido a la aplicación se realizara primero una comprobaciones de su validez y si existe ya un proyecto con ese nombre, si el proceso es exitoso

se procede fragmentando cada una de sus partes (principalmente dibujos SVG y datos de elementos JSON) y se envía a Drive. Todos los datos se guardan formando una estructura concreta de carpetas.
Esta función además almacena los contenidos del proyecto n el objeto global *lastUploadedProject*. Esto evita que se tengan que ir a buscar los contenidos al *backend* si se acaban de subir. El hecho de guardarlos en el objeto permite también poderlos consultar sin conexión a internet.

Listar proyectos, toma los nombres e id de los proyectos, con estos datos se crea la lista de proyectos en HTML.

Coger proyecto, a partir del id del proyecto devuelve los nombres e id de los archivos de este, dibujos SVG y JSON con datos de elementos.

4.3. Estructuración de archivos en Google Drive

Todos los contenidos de la aplicación los almaceno en una carpeta que se crea la primera vez que se guarda un proyecto. Por ahora esto lo he hecho con una carpeta estándar visible por el usuario en vez de con una carpeta oculta específica para aplicaciones. He decidido hacerlo así para que el usuario pueda gestionar los contenidos de sus proyectos también accediendo a su Google Drive directamente.

Para identificar esta carpeta uso un nombre específico, no obstante tendré que buscar una alternativa ya que si existiese otra carpeta con ese nombre o si se le cambiase el nombre la aplicación dejaría de funcionar correctamente. Una opción sería almacenar el id de esa carpeta visible por el usuario en una carpeta oculta para la aplicación.

Dentro de la carpeta de la aplicación que se crea actualmente se encuentran las carpetas de los proyectos y una carpeta llamada *appSettings*. Dentro de *appSettings* habrán datos de la aplicación como por ejemplo las miniaturas de cada proyecto y un archivo JSON con ajustes generales que se creará la primera vez que este se necesite.

Dentro de cada carpeta de proyecto la estructura será similar. Cada una incluirá una carpeta *drawings* para los archivos SVG de los dibujos y una *elementsData* para los JSON con datos de los elementos del dibujo. Además más adelante habrá un archivo JSON para ajustes específicos del proyecto y una carpeta para imágenes.

Los archivos JSON relativos a ajustes de la aplicación que he mencionado se podrían guardar en una carpeta oculta para la aplicación ya que el usuario no necesita para nada acceder a ellos directamente y podría borrarlos accidentalmente.

4.4. Estructura del archivo exportado que lee la aplicación

Este sería el archivo que generaría el exportador y que es leído por el formulario de la aplicación. Tras ser leído su contenido se fragmenta en cada una de sus partes elementales y estas se envían a Google Drive.

El archivo tiene estructura JSON. Está compuesto por tres entradas, la primera “projectInfo” que contiene un objeto donde actualmente se encuentra solo el título del proyecto aunque más adelante incluirá otros datos como el propietario. Las otras dos entradas que incluye el archivo son “drawings” y “elementsData”, cada una de las cuales tiene asociada un objeto que contiene los dibujos i los datos de los elementos respectivamente. En el caso de los datos de los elementos

se utiliza el nombre de la categoría como *key*, por ejemplo *columns* o *windows*. En el caso de los dibujos se utiliza el nombre de cada dibujo. Si el hecho de usar el nombre del dibujo como *key* se volviese un inconveniente ya que dos dibujos tienen el mismo nombre entonces habrá que organizarlo en un *array* de objetos, cada uno de los cuales contendrá el nombre del dibujo y su contenido. De momento no lo he hecho así para simplificar el proceso de lectura.

Un aspecto que cabe mencionar es que para poder almacenar correctamente el contenido SVG como *strings* es necesario *escapar* las comillas (`\"`) que contiene.

Consultar el archivo de muestra que se incluye con los archivos del proyecto.

4.5. Almacenamiento de datos en el *front* (optimización de peticiones al API)

Existe la variable *appData* que almacena datos de los proyectos (principalmente nombres e ids, no los contenidos de los archivos ya que podría ser demasiado grande) y sirve para reducir al máximo la necesidad de realizar peticiones, si un recurso ya se guardó ya no se vuelve a pedir.

Existen también dos variables con igual estructura que almacenan datos del proyecto actual incluyendo el contenido de los archivos ya cargados. La otra almacena el último proyecto cargado, lo cual resulta útil para evitar ir a buscar recursos que se acaben de subir y para permitir consultarlo sin conexión a internet.

Se puede ver la estructura de estos objetos en las primeras líneas del archivo *index.js*.

5. SELECCIÓN DE ELEMENTOS EN LOS DIBUJOS

He intentado desarrollarlo de la manera más simple posible. He organizado el contenido del archivo SVG en grupos con un atributo inventado que lo identifica como conjunto seleccionable y un atributo data con el id (no utilizo el id estándar ya que habría en el documento más de un elemento con el mismo id).

```
<g selectable data-id="wall-1"> . . . </g>
```

En vez de añadir un *click event listener* a cada entidad seleccionable lo que he hecho ha sido crear un solo *event listener* al cargar la aplicación en el contenedor general de todos los dibujos (apartado 3.1), el cual está siempre presente en la aplicación. Así evito tener que añadir gran cantidad de *event listeners* y peor aún tener que quitarlos todos al cambiar de proyecto.

El único *event listener* del contenedor detecta si se ha hecho clic en algún hijo y el evento sube hasta que se encuentra con un elemento con atributo *selectable*, es entonces cuando sabe que ha encontrado un elemento seleccionable así que le añade una clase css para mostrarlo seleccionado y guarda el valor de su `data-id` en una variable. Si se hace clic fuera de un elemento con ese atributo simplemente se deselecciona cualquier elemento que pueda estar seleccionado.

El valor del id del elemento seleccionado guardado en la variable se utiliza para gestionar la selección al cambiar de dibujo. Al cambiar de dibujo permite deseleccionar el elemento en el dibujo actual si este está presente y añadir la selección al elemento en el nuevo dibujo si este está presente.

6. BIBLIOGRAFÍA Y RECURSOS

Google Drive Api
https://developers.google.com/drive

Google Drive Api > V3 > Browser Quickstart
https://developers.google.com/drive/api/v3/quickstart/js

Google Drive API > V3 > File reference
https://developers.google.com/drive/api/v3/reference/files

MDN web docs | Using Promises
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

Styling & Customizing File Inputs the Smart Way
https://tympanus.net/codrops/2015/09/15/styling-customizing-file-inputs-smart-way/

PEC4

Entrega 2. Informe de trabajo

M1.926 – Trabajo Final de Máster
Máster de aplicaciones multimedia | UOC

Juan Ramón Cárceles Román

Diciembre 2019

ÍNDICE

1. TAREAS DE LA PLANIFICACIÓN REALIZADAS EN LA ETAPA

La planificación prevista para la etapa era la siguiente:

	PEC 4						
	12 Noviembre - 9 Diciembre (28 días)						
	12 - 15	16 - 19	20 - 23	24 - 27	28 - 1	2 - 5	6 - 9
INTERFAZ GRÁFICA PANTALLAS (HTML + CSS + JS)							
PRUEBA VISUALIZACIÓN DOCUMENTOS 2D							
PRUBEA GOOGLE API PARA AUTENTICACIÓN Y ALMACENAMIENTO							
CONSULTA INFORMACIÓN OBJETOS (2D)							
DESARROLLO HERRAMIENTAS MEDICIÓN EN DIBUJOS							
AÑADIR COMENTARIOS EN LOS DIBUJOS							
GUARDAR INFORMACIÓN DE COMENTARIOS							
COMPARTIR DOCUMENTOS CON INFORMACIÓN AÑADIDA							
TEST CON USUARIOS REALES Y REVISIÓN							

Esta ha sido aproximadamente la versión real de la planificación una vez finalizada la etapa:

	PEC 4						
	12 Noviembre - 9 Diciembre (28 días)						
	12 - 15	16 - 19	20 - 23	24 - 27	28 - 1	2 - 5	6 - 9
ARQUITECTURA DE LA APLICACIÓN							
INTERFAZ GRÁFICA PANTALLAS (HTML + CSS + JS)							
PRUEBA VISUALIZACIÓN DOCUMENTOS 2D							
PRUBEA GOOGLE API PARA AUTENTICACIÓN Y ALMACENAMIENTO							
CONSULTA INFORMACIÓN OBJETOS (2D)							
AÑADIR COMENTARIOS EN LOS DIBUJOS							
GUARDAR INFORMACIÓN DE COMENTARIOS							

Esta vez la planificación prevista se ha modificado más que en el caso de la PEC anterior. Esto ha sido debido principalmente a una importante tarea de reestructuración del código JavaScript. Esto es algo que tendría que haber hecho desde el principio pero que debido a mí limitada experiencia dejé hasta verlo más claro. Supongo que a poco a poco me acostumbraré a plantear desde el principio una buena estructura del proyecto.

Por otra parte he subestimado el tiempo requerido para llevar a cabo ciertas tareas como implementar una herramienta o compartir proyectos. Las cuales finalmente he dejado parcial o totalmente para la próxima PEC.

Aspectos sobre las tareas principales llevadas a cabo:

Nombre	Estado	Descripción
Arquitectura de la aplicación	En curso	Esta tarea fue la primera y la que he llevado en paralelo todo el tiempo. Continuará durante la próxima PEC hasta que considere que está consolidada y el proyecto puede crecer manteniendo la misma estructura.

Prueba visualización documentos 2D	Concluida	Esta tarea ha tomado el tiempo que tenía previsto aproximadamente. He podido realizar pruebas con dibujos que equivalen a dibujos reales, lo cual ha resultado muy útil para plantear una estructura definitiva del svg que tendrá que generar el exportador y sobre todo hacer que la aplicación soporte el formato.
Conexión con API Google Drive	En curso	He continuado añadiendo funciones del API de Google Drive que son necesarias para la aplicación. Esto es algo que he tratado puntualmente en momentos diferentes a medida que he ido necesitando las funciones.
Consulta información elementos	Concluida	Esta tarea se puede considerar concluida ya que actualmente se puede consultar mediante una herramienta la información asociada a los elementos del dibujo.
Añadir comentarios	Concluida	Tarea completada en el tiempo previsto ya que se ha implementado correctamente la herramienta que permite añadir un comentario a un elemento del dibujo. No obstante se podría considerar incompleta ya que todavía no se pueden editar ni eliminar los comentarios, lo cual es imprescindible para que tenga sentido añadirlos.
Guardar información de comentarios	En curso	Esta tarea la considero concluida ya que la información de los comentarios se guarda. Esta tarea está muy relacionada con la del API de Google Drive.

2. REESTRUCTURACIÓN DEL CÓDIGO

2.1. Programación orientada a objetos

Tras haber finalizado la PEC3 con una idea más clara del funcionamiento de la aplicación he decidido que era el momento de empezar a reestructurar el código antes de que el proyecto creciese demasiado. La reestructuración se ha hecho siguiendo los principios de la programación orientada a objetos, lo cual me ayudará a que el proyecto sea más entendible y fácil de mantener, lo cual es imprescindible de cara al futuro.

Como los navegadores todavía no soportan la sintaxis `import/export`, que es la que permite fraccionar el código JavaScript en archivos con dependencias entre ellos, lo que he hecho ha sido utilizar un *bundler*. He elegido Webpack ya que lo he usado en otras ocasiones y lo he configurado por ahora para que genere un único archivo `main.js` que es el que vinculo al final del archivo `index.html`. Por ahora he mantenido la configuración bastante básica, más adelante si fuese necesario añadiré complementos para por ejemplo hacer el JavaScript compatible con versiones de navegadores anteriores, para minificar el código…

El primer paso que he seguido ha sido el de empezar a pasar el código del archivo `index.js` a una clase general llamada `Application` (todavía no he terminado de pasarlo todo). Esta clase es la que es instanciada una vez y se llama a su método `start()` para arrancar la aplicación.

Un aspecto importante es que organizando el código de esta manera se tiende hacia la modularidad, lo cual podría permitir fácilmente en el futuro abrir más de un proyecto a la vez sin tener que hacer grandes cambios en la estructura del código.

Progresivamente he ido creando otras clases correspondiente a entidades menores que en muchos casos corresponden a elementos gráficos, como por ejemplo la clase `Dibujo` o la clase `MainPanel`.

2.2. Clases destacadas

Clase `Tool`

He creado clases para las herramientas de la aplicación que he creado hasta ahora.

Lo primero que he hecho para implementar las herramientas ha sido crear una clase abstracta base de la cual heredaran todas las herramientas. Esta clase contiene información del nombre de la herramienta y del botón desde la cual se activa. Luego he creado otra clase abstracta (`ElementSelection`) de la cual heredaran todas aquellas herramientas que requieran seleccionar elementos. Si JavaScript soportase la herencia múltiple no hubiese hecho que `ElementSelection` herede de `Tool`, sino que la herramienta herede directamente de ambas. Es a partir de la clase abstracta `ElementSelection` que por lo tanto he implementado las dos clases correspondientes a las herramientas que hay actualmente:

- `ElementData`
- `AddComment`

Clase `Workspace`

El `Workspace` corresponde al espacio de trabajo de un proyecto. Se crea una instancia de esta clase cuando se accede a un proyecto, ya sea desde la lista de proyectos como directamente con su url.
Tiene la función de preparar el espacio de trabajo por ejemplo creando los botones con los dibujos del proyecto y de gestionarlo por ejemplo controlando las herramientas activas y gestionando el cambio de dibujos.

Actualmente el usuario no puede crear más de un `Workspace` a la vez.

Clase `Drawing`

Es una clase que tiene una referencia al elemento svg correspondiente.

Cuando se accede a un proyecto (cuando se crea un `Workspace`) es cuando se crean tantas instancias como dibujos tenga. Los objetos se crean solo con el nombre e id del dibujo, es decir sin contenido gráfico, y se añaden a una propiedad del objeto `Application` que guarda información de los proyectos. Quizás sería más coherente crear los dibujos y guardarlos directamente en el array `drawings` del objeto `Workspace` que se acaba de crear. Si no lo hago es porque cuando se cierra el `Workspace` me conviene conservar por lo menos los nombres e ids de los dibujos para ahorrarme volverlos a pedir si el usuario vuelve a entrar al proyecto en esa misma sesión.

Es en el momento que se hace click en un dibujo de la lista, cuando se añade el contenido gráfico al objeto dibujo (en el caso de que no lo tenga) mediante el método `setContent()`.

Los objetos de tipo dibujo son los que más adelante incluirá también el método para hacer zoom en los svg.

Clase Main Panel

El panel es un elemento muy importante de la aplicación. Este es una ventana que aparece al lado o encima de los dibujos y que muestra contenido relacionado con la acción que se está llevando a cabo. Por ejemplo si se quieren ver las propiedades de un elemento, al seleccionarlo estas aparecerán en el panel, o si se comenta un elemento, el formulario para introducir el texto aparecerá también en el panel.

Se ha creado una clase `MainPanel` con toda su funcionalidad. Esta tiene métodos para añadir y quitar contenido así como para anclarlo en diferentes sitios (este último no se puede utilizar todavía).

Un aspecto importante es que está preparado para mostrar varios contenidos de categorías diferentes simultáneamente los cuales se organizan en pestañas. Esto será útil ya que más adelante se podrá seleccionar un elemento para ver su información y en el caso de que tenga un comentario en el panel aparecerá la pestaña Propiedades y una Comentario.

3. FUNCIONES AÑADIDAS AL API

He implementado tres acciones más mediante el API de Google Drive.

Eliminar un archivo o carpeta

Eliminar de momento lo he utilizado para eliminar proyectos, lo cual se puede llevar a cabo desde el menú contextual que aparece sobre los ítems proyecto de la lista.

Un aspecto importante era reflejar los cambios en el *front-end* tras haber recibido la respuesta del servidor de que la carpeta del proyecto se eliminó correctamente.

No obstante la eliminación de un proyecto no está del todo implementada ya que tendré que ampliarla para que haga más cosas como borrar su miniatura en el caso de que disponga de una, ya que estas se encuentran en una carpeta separada. También falta gestionar las variables `currentProject` y `lastUploadedProject` en el caso de que el proyecto esté en estas. Además si se quisiese borrar el proyecto actual que está abierto primero habría que poder cerrarlo y poner la aplicación en un estado equivalente al inicial.

Actualizar un archivo

Actualizar el contenido de un archivo lo he utilizado para guardar los comentarios añadidos y es en el apartado que trato este tema donde lo explico en más detalle.

Renombrar un archivo

El método para renombrar lo he implementado pero todavía no lo he utilizado en ningún sitio. Lo más probable es que haga una prueba para cambiar el nombre a un proyecto.

4. LOS COMENTARIOS

La aplicación va a permitir por lo menos dos tipos de comentarios. Comentario sobre un elemento y sobre una zona del dibujo. Ambos se añadirán mediante sus correspondientes herramientas. He empezado implementando el comentario sobre un elemento.

4.1 Comentario sobre un elemento

El comentario sobre un elemento consiste en un comentario que se crea tras indicar el elemento del dibujo sobre el cual se quiere comentar. Una característica importante que lo diferencia del comentario sobre una zona es el hecho que la representación gráfica que se añade al elemento comentado para indicar que tiene un comentario ha de aparecer en cualquier otro dibujo del proyecto en el cual aparezca el elemento. Por ejemplo una misma columna que aparezca en una planta primera y en una sección siempre ha de tener la indicación correspondiente.

Cuando se crea un comentario se crea una nueva instancia de la clase `Comment` con los datos del comentario y este se añade al array de comentarios, que es una propiedad del `Workspace` actual. Además mediante un método de ese nuevo objeto comentario se crea su primera representación gráfica, la cual corresponde a la del dibujo actual. Las representaciones gráficas se colocan en un `<g>` (`SVGGElement`) al final del `<svg>` (`SVGSVGElement`) correspondiente.

El resto de dibujos no se modifican ya que de hecho los dibujos se cargan a medida que se solicitan y la mayoría de ellos no estarán ni siquiera en el navegador.

Es por este motivo que es solo en el momento de cambiar de dibujo (lo cual se hace mediante el método `setDrawing()` del `Workspace`) que si al nuevo dibujo le faltan representaciones de comentarios existente estas serán creadas. Cualquier nueva representación que se cree será añadida a un array de representaciones del objeto comentario correspondiente. Para optimizar la comprobación cada vez que se muestra un dibujo se hace uso de una propiedad con valor booleano del objeto dibujo actual llamada *commentsChanged*.

Dificultades que he encontrado:

El elemento `<svg>` que actualmente representa un comentario es un rectángulo alrededor del elemento. Este se crea calculando el *bounding box* del elemento a comentar mediante el método nativo *SVGGraphicsElement.getBBox()*.

Mi intención era que al cambiar de dibujo el nuevo dibujo no se mostrarse hasta que las posibles representaciones graficas se hubiesen añadido. Es por este motivo que en el caso de que el dibujo todavía no estuviese "descargado" lo cogía del *backend*, lo *parseaba* mediante el *DOMParser* y le añadía las representaciones antes de añadirlo al DOM. Por otra parte en el caso de que ya estuviese en el DOM le añadía las representaciones antes de quitarle el `display:none`. Desafortunadamente ninguno de estos procesos me dio un buen resultado ya que, por lo menos en Chrome, hasta que un `<svg>` no se encentra en el DOM o si tiene `display:none` no se puede realizar ningún cálculo sobre este, lo cual hacia que no se pudiese calcular el *bounding box*.

Esto me obligó a añadir siempre el dibujo al DOM en el caso de que todavía no estuviese y a añadirle `visibility:hidden` y quitar el `display:none` antes de realizar las operaciones. *Visibility* tiene la ventaja de que sí permite realizar cálculos aunque no se vea ya que de hecho el elemento ocupa espacio en el documento, es por esto que tuve que asegurarme de que al

dibujo anterior se le aplicara `display:none` previamente ya que ambos no cabrían simultáneamente.

4.2. Herramienta para crear comentarios

La herramienta tiene la tarea principal de gestionar el proceso de creación de un objeto comentario. Es decir permite seleccionar el elemento y muestra el formulario en el cual se añade el contenido textual.

Esta parcialmente implementada ya que todavía no se puede cancelar un comentario que este a medias ni cambiar de objeto comentado si ya se ha seleccionado uno.

4.3. Guardar comentarios creados

La representación gráfica de los comentarios es por ahora el único contenido que se añade dinámicamente a los dibujos mediante la aplicación.

La propiedad *commentsChangesUnsaved* del `Workspace` servirá para indicar que hay que guardar los datos en Drive. Cada vez que se cree o se borre un comentario se cambiará el valor de *commentsChangesUnsaved* a true y el botón guardar dejará de estar inactivo.

Los dibujos svg no se guardaran con las representaciones añadidas sino que se guardará un archivo *json* con la información de los comentarios y estos se pintaran cuando el proyecto se vuelva a abrir en el futuro.

El archivo *json* con la información de los comentarios se creará dentro de la carpeta del proyecto y tendrá una estructura de array de objetos de este tipo:

```
{
  elementId: "45763453",
  content: "Hola"
}
```

Dificultades que he encontrado:

Un inconveniente importante que he encontrado ha sido con el uso de Google Drive. Ya que al tener limitado control sobre el *back-end* lo único que puedo hacer es actualizar el contenido de un archivo. Esto implica que cada vez que guardo tengo que enviar todo el contenido de los comentarios del proyecto en vez de enviar solo la diferencia. Esto podría ser modificado en el futuro si el *back-end* fuese también desarrollado.

4.4. Tareas pendientes

Los aspectos más importantes que me han quedado pendientes respecto a la herramienta comentarios ha sido poderlos eliminar y editar su contenido.

Otras tareas que serían útiles sería la otra herramienta para añadir comentarios en zonas del dibujo las cuales se podrían indicar mediante una polilínea. Por último resultaría útil poder ocultar todos los comentarios de los dibujos.

5. LOS DIBUJOS

5.1. La estructura *svg* de los dibujos

El hecho de haber preparado dibujos que representan un proyecto arquitectónico me ha ayudado a determinar la estructura interna que han de tener estos y a adaptar la aplicación para que pueda interpretarla. Los elementos se agrupan en entidades (elementos seleccionables) cada uno de los cuales tiene una serie de hijos que son grupos de elementos con igual representación gráfica:

```
<g selectable data-id="column-7" data-category="columns">
   <g class="column-sectioned solid">
       <rect x="2041.154" y="969.479" width="70.866" height="70.866" />
   </g>
   <g class="column-sectioned curve">
       <rect x="2070.324" y="998.648" width="12.527" height="12.527" />
   </g>
</g>
```

El motivo de esta decisión se explica en los siguientes apartados.

5.2. Los estilos *css* de los dibujos

El css que se aplica al svg de los dibujos y que se exporta con el archivo, se sitúa en un único lugar desde el cual todos los dibujos lo reciben y lo aplican mediante clases, de esta manera se reduce el código y se controlan los estilos css desde un solo punto. Este último aspecto es importante en el caso de que más adelante se permita modificar la representación de los elementos desde la aplicación para por ejemplo imprimir un pdf.

Esta serie de estilos css se cargan la primera vez que se solicita un dibujo, aunque la otra posibilidad será al entrar en el espacio de trabajo. Actualmente se inyectan en el `<head>` en una etiqueta `<style>`. La intención era situarlos con la etiqueta `<style>` dentro de un contenedor específico del proyecto como por ejemplo el `<main>`, de tal manera que si más adelante se añade la posibilidad de tener más de un proyecto abierto a la vez estos al estar dentro de cada uno de los contenedores para cada proyecto y con el atributo `scope` permitiría apuntar solo a los dibujos de cada proyecto. Desafortunadamente el atributo `scope` se eliminó y si realmente más adelante permito abrir más de un proyecto a la vez tendré que buscar una alternativa.

Ahora al cerrarse un *workspace* se elimina la etiqueta `<style>` del `<head>` de tal manera que al abrir otro no haya más de una simultáneamente.

5.3. Selección de elementos del dibujo

La representación gráfica de la selección de los elementos del dibujo ha sido modificada ligeramente para adaptarse a la estructura final de los dibujos svg. La versión de la aplicación presentada en la PEC anterior ya realizaba la selección sobre los elementos de los dibujos proporcionados. No obstante debido a la simplicidad de aquellos el sistema ha tenido que ser modificado.

La selección grafica se sigue realizando mediante la adición de una clase css. Una diferencia es que debido al cambio en la estructura de los dibujos ahora en vez de situarse en el propio

elemento con el atributo *selectable* que engloba todo el elemento, se sitúa una clase en cada uno de sus hijos, los cuales corresponden a grupos de elementos con representación similar. Es en ese mismo nivel donde se encuentran las clases que dan el estilo al svg (ver punto 6.1).

Pese a tener todas las clases el mismo nivel de especificidad, debido a que el estilo de los dibujos se añade dinámicamente (cuando se solicita el primer dibujo) en un `<style>` al final de `<head>` este sobrescribe la hoja de estilos general que incluye el css para indicar selección y esta deja de tener efecto. Para evitar este problema y no tener que preocuparme por situar cada vez el `<style>` de los dibujos delante del `<link>` css lo que he hecho ha sido añadir mayor especificidad a las reglas de selección añadiendo otro nivel que indica los posibles elementos a los que afecta en su interior, por ejemplo `.selected line`.

6. OTRAS TAREAS MENORES

6.1. El menú contextual

El primer sitio donde lo he probado ha sido en los ítems de la lista de proyectos para poderlos borrar.

Para crear el menú he sobrescrito el `contextmenu` *event* del objeto `window`. Por otra parte cualquier click en `Window` o un evento `contextmenu` hace que si hay alguno abierto este se cierre.

El contenedor principal del menú contextual vacío lo he añadido directamente al HTML como otro más de los que están fuera del `<header>` y del `<main>` y que por defecto están ocultos. Este contenedor se rellena dinámicamente con los botones y acciones correspondientes según donde sea requerido. Ha sido importante tener en cuenta que los *event listeners* de los botones sean eliminados al cerrar el menú ya que podrían irse acumulando en la memoria.

6.2. Iconos

En la versión anterior los iconos que aparecían repetidos más de una vez se añadían mediante la etiqueta `<img>`.

Con la finalidad de reducir la cantidad de peticiones y mejorar la carga de la página he utilizado otro sistema basándome en lo que he visto en la aplicación Pipedrive. Esto ha afectado únicamente a la estructura del archivo `index.html` y consiste en lo siguiente:

He creado un contenedor `<div>` en el cual he creado un `<svg>` en línea en el interior del cual he creado un elemento `<symbol>` para cada icono. Posteriormente estos son solicitados desde cualquier otro punto del DOM mediante un `<use>` dentro de un `<svg>` en línea.

Esto me ha servido incluso para elementos `<svg>` animados como es el caso del que aparece mientras se espera una acción.

7. BIBLIOGRAFÍA Y RECURSOS

MDN web docs | JavaScript modules
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

MDN web docs | DOMParser
https://developer.mozilla.org/en-US/docs/Web/API/DOMParser

MDN web docs | Parsing and serializing XML
https://developer.mozilla.org/en-US/docs/Web/Guide/Parsing_and_serializing_XML

MDN web docs | Document.documentElement
https://developer.mozilla.org/en-US/docs/Web/API/Document/documentElement

HTML scoped Attribute
https://www.w3schools.com/tags/att_scoped.asp

Google Drive API V3 | Upload files
https://developers.google.com/drive/api/v3/manage-uploads

How to create a custom context menu for your web application
https://dev.to/iamafro/how-to-create-a-custom-context-menu--5d7p

PEC5

Entrega 3. Informe de trabajo

M1.926 – Trabajo Final de Máster
Máster de aplicaciones multimedia | UOC

Juan Ramón Cárceles Román

Enero 2020

ÍNDICE

1. TAREAS DE LA PLANIFICACIÓN REALIZADAS EN LA ETAPA

La planificación prevista para la etapa era la siguiente:

	PEC5					
	10 Diciembre - 3 Enero (25 días)					
	10 - 13	14 - 17	18 - 21	22 - 25	26 - 29	30 - 3
COMPARTIR DOCUMENTOS CON INFORMACIÓN AÑADIDA						
TEST CON USUARIOS REALES Y REVISIÓN						
DETALLES GRAFICOS FINALES						
DETALLES IMPLEMENTACIÓN FINALES						
MEMORIA FINAL						
DOCUMENTO PRESENTACIÓN						

Esta ha sido aproximadamente la versión real de la planificación una vez finalizada la etapa:

	PEC5					
	10 Diciembre - 3 Enero (25 días)					
	10 - 13	14 - 17	18 - 21	22 - 25	26 - 29	30 - 3
COMPARTIR PROYECTOS (COLABORACIÓN)						
TEST CON USUARIOS REALES Y REVISIÓN						
DETALLES GRAFICOS FINALES						
SERVICE WORKER Y NOTIFICACIONES						
DETALLES IMPLEMENTACIÓN FINALES						
MEMORIA FINAL						
DOCUMENTOS PRESENTACIÓN						

En esta última PEC las tareas realizadas han seguido prácticamente en su totalidad las previstas. La diferencia principal ha sido que he dedicado menos tiempo del previsto a realizar la memoria y más a terminar aspectos funcionales de la aplicación que estaban a medias e incluso añadir algunos que no tenía previstos inicialmente, con el objetivo de conseguir un producto más completo y por lo tanto con más valor.

Las tareas de desarrollo más relevantes han sido las siguientes:

- Compartir un proyecto para poder colaborar.
- Editar un comentario existente.
- Borrar un comentario existente.
- Hacer que la aplicación se pueda instalar.
- Poder recibir notificaciones.

2. COMPARTIR UN PROYECTO

Para conseguirlo he utilizado nuevos métodos del API de Google Drive, concretamente los relacionados con los permisos. El concepto *permiso* es el que utiliza Google Drive para gestionar la propiedad de los archivos. Como mínimo cada archivo tiene siempre un permiso que es el del creador, el cual por lo tanto tiene un permiso con rol propietario. Sobre un mismo archivo se pueden crear varios permisos, cada uno referente a un usuario, de tal manera que este es accesible a todos, es decir que está compartido. Si el elemento es una carpeta todos los archivos internos también son accesibles a los usuarios que tengan permiso sobre esta.

He necesitado por lo tanto poder crear nuevos permisos en las carpetas de los proyectos, de tal manera que los usuarios añadidos tengan acceso a todo su contenido. Para ello lo imprescindible es saber el email del usuario con el cual se quiere compartir.

También he necesitado añadir la función de eliminar permisos para dejar de compartir una carpeta de un proyecto. Cualquier persona puede eliminar cualquier permiso excepto el del propietario de la carpeta. Para eliminar un permiso necesito el id del permiso, este se genera cuando se crea.

La estructura del objeto permiso que ofrece Google Drive me da alguna información del usuario. Esto lo he aprovechado para mostrar el nombre y la imagen de perfil en la lista de miembros que colaboran en un proyecto.

Dificultades encontradas

La principal dificultad la encontré en el momento de intentar crear varios permisos simultáneamente. Al enviar varias peticiones a la vez solo se realizaba la primera y las siguientes devolvían error indicando que había llegado al límite de peticiones. Esta limitación la tenía que solucionar sí o sí ya que sino la aplicación solo podría ofrecer la opción de añadir uno a uno cada miembro.

Desgraciadamente no he conseguido todavía encontrar ninguna solución fácil así que para intentar evitarlo temporalmente lo que he hecho ha sido que en el método que crea permisos, si se solicita crear más de uno, este deja un breve espacio de tiempo entre cada petición al API, así evito que salte el error. No obstante incluso dejando tiempo entre uno y otro a veces falla. Tengo que continuar mirando si el API ofrece métodos avanzados para enviar varias peticiones simultáneamente (batch request) o si la única solución sería pagar para quitar las limitaciones.

3. GESTIÓN DE LOS COMENTARIOS

Para poder trabajar sobre comentarios existentes añadidos a elementos el primer paso fue poderlos seleccionar. Para ello decidí que la selección del comentario se haría por ahora seleccionando el elemento comentado.
Para poder conseguirlo lo que hice fue añadir al elemento comentado un atributo data con el id de su comentario, de tal manera que si este es seleccionado compruebo si tiene un cometario y en caso afirmativo lo busco también.
Esta técnica puede que no sea la mejor cuando se puedan añadir varios comentarios a un mismo elemento así que tendré que mejorarla.

Tanto en el caso de editar o borrar que comentaré a continuación lo más importante ha sido crear una clase *CommentForm* que es la encargada de gestionar el contenido de un comentario. Esto lo he planteado así ya que por ahora es la única interfaz posible para poder visualizar y por lo tanto trabajar sobre un comentario existente.

3.1. Edición de un comentario existente

Dese el panel de visualización del contenido del comentario si se hace clic en Edit se habilita el formulario en el cual se está mostrando el contenido dando la oportunidad de modificarlo.

Una vez se hayan realizado los cambios pertinentes hay un botón para confirmarlos, es cuando se hace clic en este que los añado al objeto comentario que se está editando y activo el botón guardar de la aplicación para indicar que hay cambios por guardar, ya que sino estos no estarán disponibles la próxima vez que se acceda a la aplicación.

3.2. Borrado de un comentario

Eliminar un comentario por ahora lo he hecho con un proceso similar al de editar que he explicado en el punto anterior. El botón para borrar también se encuentra en el panel de visualización del contenido del comentario seleccionado.

Esto lo he hecho así ya que era la manera más fácil de solucionarlo, no obstante no creo que sea la más normal, y tendré que ofrecer la opción de poder seleccionar el comentario directamente y borrarlo con una herramienta de borrado, o bien desde un menú contextual haciendo clic secundario o con el teclado.

Cuando se borra lo primero que se hace es borrar todas las representaciones svg que haya de este en los dibujos, las cuales estarán referenciadas en un array que es una propiedad del objeto comentario. Tras finalizar se procede a quitar el objeto comentario del array de comentarios del workspace y se indica que hay cambios por guardar.

4. PWA: MANIFEST Y SERVICE WORKERS

Gracias a haberla convertido en una *progressive web app* he podido añadir ciertas características muy útiles como la posibilidad de instalarla y de recibir notificaciones.

En este caso me resultó muy útil todo lo aprendido en el curso de *Tecnologías y Herramientas para el desarrollo web* del máster.

Para comprobar la calidad de la aplicación he utilizado Lighthouse de Chrome para ejecutar Audits e irla mejorando.

4.1 Instalación

Una vez añadido el archivo *manifest.json* y el *service worker* ambos cumpliendo unos requisitos mínimos la aplicación ofrece la posibilidad de ser instalada en los navegadores que son compatibles. El más compatible es Chrome aunque también funciona en otros como Firefox.

La tarea de transmitir al usuario que la aplicación se puede instalar he elegido personalizarla evitando que el navegador pueda mostrar automáticamente la alerta por defecto. Concretamente lo que he hecho ha sido crear un botón en el menú lateral, el cual se muestra solo si el navegador ofrece la posibilidad de instalar.
Tendré que pensar más en detalle cual es la estrategia que quiero llevar a cabo para convencer a los usuarios de que la instalen, ya que si lo hacen es más posible que la usen con mayor frecuencia.

Ya que una de las características principales que se espera un usuario si la instala es que pueda utilizarla offline y que sea más rápida he puesto en cache algunos archivos, no obstante este es un tema en el cual no he profundizado y que tengo pendiente ya que es muy importante.

La posibilidad de recibir notificaciones también se obtiene gracias a añadir un *service worker* a la aplicación, este es el que es capaz de recibirlas incluso cuando la aplicación no esté abierta. Este tema es el que explico en el siguiente puto.

4.2. Notificaciones push

Pese a la posibilidad de poder recibir notificaciones gracias a haber añadido un *service worker* me di cuenta que al parecer para ello era necesario disponer también de un back-end que las enviase, un motivo era la necesidad de un ambiente seguro para realizar la solicitud de envío. Esto hizo que la situación se complicara bastante y que tuviese que investigar mucho para buscar una solución, ya que una de las condiciones de la aplicación era que funcionase sin back-end.

Tras buscar mucho encontré algunos servicios de Firebase como FCM (Firebase Cloud Messaging) que podían ayudarme a resolver el problema, ya que muchos desarrolladores decían que los utilizaban en sus aplicaciones Android cuando estas no disponían de back-end. Puesto que añadir estos servicios de Firebase ha sido un paso importante he decidido explicarlo más en detalle en el siguiente apartado.

Algunos aspectos de la recepción y la gestión de notificaciones no los resolvía FCM así que los he tenido que resolver sin esta librería. El principal es que cuando se recibe una notificación el comportamiento es diferente si la aplicación esta activa, es decir si la ventana está abierta y visible o si está escondida o cerrada. Si está abierta el mensaje es recibido por el código principal en un *event listener onmessage* y por lo tanto se puede gestionar como se quiere, por otra parte si la app no está visible lo recibe el archivo *service worker* y se produce una notificación del sistema. La dificultad era que en este segundo caso necesitaba hacer llegar el mensaje desde el *service worker* al código de la aplicación, lo cual no es tan fácil ya que son archivos que se ejecutan en paralelo y por ejemplo el *service worker* no tiene acceso al objeto *window* global.

Para conseguir que si la aplicación estaba en segundo plano le llegase el mensaje tenía que conseguir que el *service worker* se comunicara con el código principal de la aplicación para hacerle llegar también el mensaje y que este apareciese en la bandeja de notificaciones. Esto lo

conseguí gracias al elemento nativo *BroadcastChannel*, este dispone del método *postMessage()* y del evento *onmessage*. *BroadcastChannel* no es compatible con todos los navegadores pero me sirvió por el momento ya que no encontré soluciones fáciles.

La tarea de gestionar las notificaciones es más amplia y complicada de lo que pensaba inicialmente así que se quedó a medias y ahora tendré que continuar hasta completarla satisfactoriamente. Por ejemplo es importante que una vez recibida en la bandeja de notificaciones cuando el usuario haga clic sobre ella, este sea dirigido a la parte del proyecto que es trata en la notificación, de la misma manera que una notificación de YouTube te lleva al vídeo que se menciona.

5. FIREBASE

El uso de los servicios de Firebase ha sido tan importante para poder dar solución a algunos problemas y limitaciones que a continuación voy a explicar algunos aspectos en más detalle de los tres que hasta ahora he tenido que añadir.
Firebase ofrece varios servicios para aplicaciones (base de datos, autenticación…) los cuales se pueden utilizar por separado o conectar entre ellos como se desee.

5.1 Firebase Cloud Messaging

Para poder enviar notificaciones a los usuarios Firebase Cloud Messaging hace uso de unos tokens que identifican a los dispositivos / navegadores en los cuales se está utilizando la aplicación. Estos tokens se crean mediante un método de su librería *getToken()*, y solo son generados satisfactoriamente si el usuario permite recibir notificaciones, es decir si acepta el mensaje que muestra el navegador sobre permitir notificaciones por parte de la aplicación.

En cuanto a cuando hacer aparecer la alerta del navegador que solicita poder recibir notificaciones, con la finalidad de evitar que los usuarios indiquen que no, he organizado el código de tal manera que solo aparece la primera vez que un usuario decide añadir colaboradores a un proyecto, de tal manera que puede entender mejor porqué se le está pidiendo y haya más posibilidad de que acepte. Esto incluye también la situación en que uno recibe una invitación de colaboración ya que en ese caso aunque sea la primera vez que acceda directamente ya tendrá un proyecto con más de un usuario.

Si se acepta permitir notificaciones habrá que tener en cuenta que el token más adelante podría cambiar y que por lo tanto tendrá que ser generado de vez en cuando. Lo que he hecho para reducir al máximo la creación del token ha sido utilizar el *Session Storage* que incluyen los navegadores para almacenar el token. El *Session Storage* tiene la característica de permitir almacenar datos los cuales persisten durante toda una sesión, es decir no se borran si se recarga la página sino solo cuando se cierra la ventana (pestaña o navegador). De esta manera solo ejecuto el código si la aplicación se abre en una ventana nueva.

5.2 Cloud Firestore (base de datos)

Era imprescindible disponer de un lugar centralizado en el cual almacenar todos los tokens generados, de tal manera que pudiese acceder para buscar el token del usuario al cual le quiero enviar la notificación.

Para conseguirlo añadí otro producto de Firebase, en este caso Firestore que es una base de datos.

El proceso por lo tanto es que cuando se genera un token este se envía a Firestore junto con el email. Allí son guardados usando el propio token como id del documento y el email como contenido. Esto me permite disponer de todos los tokens de un usuario. Por motivos de seguridad, en el caso de que alguien accediese a la base de datos, más adelante puede que cifre de alguna manera las direcciones de email.

Para evitar que se acumulen tokens viejos que ya no son válidos lo que se hace es que si cuando se envía una notificación utilizando el token de un usuario, si este devuelve una respuesta negativa de un tipo concreto quiere decir que este ya no es válido y por lo tanto este se borra de la base de datos.

5.3 Firebase Cloud Functions

Una vez resuelto el almacenamiento de tokens, para poder ejecutar el envío de notificaciones mediante FCM tenía que disponer de todos modos de un servidor, en parte por la seguridad. Ante la falta de este tuve que añadir al proyecto un último producto de Firebase, las *Cloud Functions*. Este servicio ofrece la posibilidad de crear un archivo con varias funciones, cada una de las cuales se ejecuta mediante una petición HTTP a una dirección específica y a las cuales se les pueden enviar parámetros en el cuerpo de la petición.

Desde una función *cloud* se puede acceder fácilmente a otros servicios de Firebase, lo cual era imprescindible en este caso ya que necesitaba FCM para enviar las notificaciones y Firestore para obtener todos los tokens a los cuales realizar el envío.

Esta función la diseñe de tal manera que uno de sus parámetros es una lista de emails a los cuales quiero enviar la notificación y los demás son el contenido de esta. En la primera parte de la función utilizo los emails para consultar la base de datos (Firestore) y obtener todos los device tokens vinculados a cada email. Una vez obtenidos los tokens en la segunda parte es cuando mediante Firebase Cloud Messaging realizo el envío de las notificaciones push a todos los usuarios con el contenido también proporcionado a la función *cloud*.

Aquí se puede consultar el código del archivo que incluye la función:

https://github.com/juanramoncarceles/visualarqdv-cloud-functions/blob/master/functions/src/index.ts

5.4 Mandar un email de invitación (Cloud Function)

Gracias a haber añadido al proyecto el servicio Cloud Functions de Firebase que he comentado en el punto anterior pude solucionar otra limitación que también encontré por no disponer de back-end, concretamente el envío del email de invitación al añadir colaboradores a un proyecto.

Para conseguirlo cree una segunda función para la cual añadí el paquete *nodemailer* con el cual ya había trabajado anteriormente. El código se encuentra en el mismo archivo enlazado al final del punto anterior.

5.5 Otros posibles servicios

Descubrir los productos de Firebase también me hizo pensar que si más adelante quisiese ofrecer una alternativa al almacenamiento en Drive podría hace uso de Cloud Storage que permite almacenar archivos. Para manipularlos en cualquier caso también sería necesario crear más funciones Cloud.

6. VER UN PROYECTO DE MUESTRA

Este es un aspecto que ya terminé para la PEC anterior, no obstante no explique su funcionamiento. Implementar esta parte ha sido relativamente fácil.

La tarea principal ha consistido en crear una carpeta pública en mi Drive. Esta carpeta accesible por cualquiera tiene una estructura de proyecto con todos sus archivos pertinentes. Lo que he hecho ha sido ejecutar la función *start(projectId)* que arranca la aplicación directamente con el id de esa carpeta de proyecto, de tal manera que al ser publica no se solicita autenticación.

Además he adaptado la interfaz para no permitir realizar ciertas acciones que requerirían permisos y por lo tanto harían saltar errores, como por ejemplo ver la lista de proyectos y guardar los cambios. Esto tengo que mejorarlo para que por ejemplo si el usuario realiza un cambio y quiere guardar le aparezca un mensaje del tipo “créate una cuenta para poder guardar el proyecto”.

7. BIBLIOGRAFÍA Y RECURSOS

Google Drive API | Push Notifications
https://developers.google.com/drive/api/v3/push

Google Drive API | Share files, folders and drives
https://developers.google.com/drive/api/v3/manage-sharing

Google Drive API | Permissions
https://developers.google.com/drive/api/v3/reference/permissions

Web Fundamentals | Adding Push Notifications to a Web App
https://developers.google.com/web/fundamentals/codelabs/push-notifications

Codelabs Google | Your First Progressive Web App
https://codelabs.developers.google.com/codelabs/your-first-pwapp

Codelabs Google | Firebase web app
https://codelabs.developers.google.com/codelabs/firebase-web

Codelabs Google | Cloud Functions for Firebase
https://codelabs.developers.google.com/codelabs/firebase-cloud-functions

Firebase | Cloud Functions
https://firebase.google.com/docs/functions

Firebase | Cloud Firestore
https://firebase.google.com/docs/firestore

Medium | Cloud Functions for Firebase, sending email
https://medium.com/@edigleyssonsilva/cloud-functions-for-firebase-sending-e-mail-1f2631d1022e

Nodemailer | Mailcomposer
https://nodemailer.com/extras/mailcomposer/

Web Fundamentals | BroadcastChannel API: A Message Bus for the Web
https://developers.google.com/web/updates/2016/09/broadcastchannel

Web Fundamentals | Add to Home Screen
https://developers.google.com/web/fundamentals/app-install-banners/

Web Fundamentals | Progressive Web Apps on Desktop
https://developers.google.com/web/progressive-web-apps/desktop

Web Fundamentals | Patterns for Promoting PWA Installation (mobile)
https://developers.google.com/web/fundamentals/app-install-banners/promoting-install-mobile