Trabajo de Fin de Grado

Grado en Ingeniería Informática

Gestión de información e interoperabilidad de información de edificaciones.

Information management and interoperability of building information

Francisco Javier Palacios Rodríguez

La Laguna, 5 de septiembre de 2017

Dña **Isabel Sánchez Berriel**, con N.I.F. 42.885.838-S profesora contratada Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

Dña. **Norena Martín Dorta**, con N.I.F. 78.674.114-S profesora contratada Doctora adscrita al Departamento de Técnicas y Proyectos en Ingeniería y Arquitectura de la Universidad de La Laguna, como cotutora.

C E R T I F I C A (N)

Que la presente memoria titulada:

"Gestión de información e interoperabilidad de información de edificaciones."

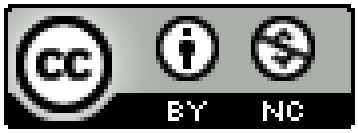
ha sido realizada bajo su dirección por **Francisco Javier Palacios Rodríguez**, con N.I.F. 45.895.806-L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de septiembre de 2017.

Agradecimientos

En especial a mi familia, por su paciencia y apoyo incondicional, mi pareja que es la que más horas ha aguantado de “seguro falta algún -1 por aquí” y por supuesto a las tutoras e involucrados en este proyecto de final de grado que cierra una importante etapa en mi vida.

Licencia



Resumen

El presente proyecto, "Gestión de información e interoperabilidad de información de edificaciones" pretende desarrollar una aplicación web sencilla e intuitiva que permita a personas interesadas en el ámbito de la arquitectura y edificación gestionar y compartir la información e incidencias de sus proyectos.

Se parte de la plataforma de GitHub, a través de la que se gestionará la información del proyecto en los formatos estándar propios del flujo de trabajo bajo la metodología BIM. Para lograr este propósito se aprovecha la API que brinda para dar soporte a la gestión de todos los recursos a través de una aplicación web sencilla e intuitiva.
A grandes rasgos la aplicación permitirá a los usuarios compartir el fichero del proyecto y sus incidencias. Se podrán visualizar las incidencias con información detallada y una vista previa de los elementos afectados. También esta aplicación permite comentar en tiempo real dichas incidencias y notificar a los implicados en el proyecto. Todos los recursos del proyecto serán accesibles a cualquier versión del mismo gracias a la tecnología de GitHub.

Palabras clave: BIM, IFC, BCF, Github, Javascript, Nodejs, Expressjs, Edificación, Modelado de Información de Construcción.

Abstract

This project, "Information management and interoperability of building information", looks for a simple and intuitive web application that allows people interested in the building and architecture field to share and manage information and issues about their projects.

Starting from the GitHub's platform, through which project information is managed in the standard workflow formats under the BIM methodology. To achieve this purpose it takes advantage of the API that provides to support the management of all resources through a simple and intuitive web application.

Broadly the application allows users to share the project file and its issues. Incidents are displayed with detailed information and a preview of affected items. Also this application allows to comment in real time the incidents and to notify those users involved in the project. All the resources of the project are accessible to any version of the same due to the technology of GitHub.

Keywords: *BIM, IFC, BCF, Github, Javascript, Nodejs, Expressjs, Building, Building Information Modelling.*

Índice General

1. Introducción

En los últimos años la industria de la construcción está evolucionando con mayor rapidez gracias a los avances tecnológicos y a nuevas metodologías de trabajo como Building Information Modeling (BIM).

Este concepto engloba a todas las disciplinas que intervienen en el desarrollo de un proyecto: el diseño arquitectónico y de ingeniería, la construcción, el mantenimiento, abarcando todo el ciclo de vida de un proyecto. BIM se define como la "forma de trabajo en el que mediante herramientas informáticas se elabora un modelo de un edificio al que se incorpora información relevante para el diseño, construcción o mantenimiento del mismo".

La investigación viene demostrando que Building Information Modeling (BIM) se está convirtiendo en un factor clave en la eficiencia y la competitividad internacional de la industria de la construcción.

Históricamente los arquitectos, ingenieros y diseñadores utilizan tableros de dibujo y técnicas de dibujo manual para transmitir sus propuestas de diseño a los clientes. Entre los años 1970 y 1980, la industria AEC (Architecture, Engineering and Construction) progresó hacia el uso de los sistemas CAD, de tal forma que los ordenadores asistían en la elaboración, diseño y redacción de obras de ingeniería y edificación. Comparado con el dibujo manual, el Diseño Asistido por Ordenador (Computer Aided Design, CAD) produjo mejoras en la productividad y en la reducción de los costes al permitir desarrollar mayor cantidad de trabajo en menor tiempo y con un menor esfuerzo. Sin embargo, el reto se convirtió en asegurar la coherencia entre todos los documentos de un proyecto.

Las herramientas tradicionales CAD carecen, entre otras, de las funciones que se detallan a continuación:

- Los modelos sólo contienen geometría tridimensional, pero no los atributos o características de estos objetos, es decir que no permiten que se puedan realizar un análisis de distintos tipos: mediciones, presupuestos, planificación, análisis de conflictos de disciplinas, simulaciones energéticas y de sostenibilidad, gestión del mantenimiento de las infraestructuras, etc.

- Los modelos tradicionales CAD no son paramétricos. Esto hace muy difícil realizar cambios y seguir las modificaciones que se producen en una obra y contribuye a que se produzcan errores e inconsistencias en los proyectos.

- Es habitual que los archivos de un proyecto sean distintos en función de las disciplinas (arquitectura, instalaciones, estructura, etc.) y el software utilizado. Esto hace que sea muy difícil asegurar que el proyecto resultante sea consistente y no contenga errores.

- Los proyectos realizados con metodología tradicional CAD permiten realizar cambios parciales que no se replican automáticamente en el conjunto del proyecto. Esto provoca errores que pueden ser difíciles de detectar dependiendo del nivel de complejidad.

- Carecen de un sistema de seguimiento y constancia de los errores en un proyecto, ya que al usar herramientas diferentes para las distintas áreas de trabajo involucradas no podemos garantizar la fiabilidad del diseño.

En la década de los 90, la International Alliance for Interoperability (actualmente conocida como BuildingSmart) desarrolló un modelo de datos para garantizar la interoperabilidad, diseño y desarrollo de proyectos relacionados con la construcción. Dicho modelo se denominó como ***Industry Foundation Classes*** (IFC), el cual contiene una serie de elementos que van desde la geometría de los componentes de la estructura hasta los materiales de los mismos, así como las relaciones entre ellos.

Hoy en día los softwares más populares soportan la edición de los ficheros IFC, entre ellas destacamos el ***Revit*** y el ***Archicad*** porque son las que más se adaptan a la tecnología BIM. Sin embargo otras herramientas como el Autocad se usan con frecuencia pero tienen que complementarse con otros programas como el ***CYPE*** para abarcar más disciplinas de la profesión.
Hay cierta complejidad para comprender la estructura y las relaciones entre los elementos, por lo que resulta útil e interesante tener una herramienta que permita al usuario visualizar los elementos que forman parte del mismo y cómo se relacionan entre ellos, entre los que destacamos xBimXplorer que es un herramienta de código abierto para la visualización de ficheros IFC.

En cuanto a la gestión de informes de errores, aparece el término ***BCF*** (BIM Collaboration Format) lo encontramos en la revisión de un proyecto y en el control de calidad del mismo. Su uso lo podemos realizar en cualquier fase del proyecto en que nos encontremos, en donde sea necesario realizar un control y chequeo del modelo. Es un formato Open BIM adoptado como un estándar por Building Smart International. Su uso va de la mano del IFC, en él traspasamos el modelo y mediante ficheros en formato BCF comunicamos incidencias y revisiones realizadas a ciertos elementos del modelo. Los BCF se generan a partir de análisis del IFC en softwares como ***Revit, Tekla, Archicad...*** Sin embargo existen softwares autónomos y plugins para las herramientas CAD que también generan dichos informes, donde destacamos **BCFier** que es el que se usó para generar los ejemplos de este proyecto.

1.1. Objetivo

El objetivo principal del presente proyecto consiste en ofrecer una herramienta sencilla e intuitiva para la visualización y distribución de datos relativa al modelo de datos de comunicación de incidencias BCF, de forma que los usuarios puedan compartirlos y llevar un control de versiones sobre el proyecto.

1.2. Alcance

Para dar respuesta a las necesidades planteadas en el apartado anterior, se requiere del desarrollo de una aplicación web que permita la visualización de la información contenida en ficheros de tipo BCF.

Además, la herramienta desarrollada ha de ser fácil e intuitiva. Dicha aplicación se estructurará de forma que permita realizar las siguientes operaciones:

- Cargar un archivo IFC que será almacenado de manera permanente, se podrá acceder a cualquier versión suya siempre que se requiera.
- Cargar archivos BCFZIP que contendrán todas las incidencias relacionadas con el correspondiente fichero IFC
- Visualizar y editar las incidencias.
- Notificar a los implicados en el proyecto.

1.3. Antecedentes

Durante la fase de investigación de este proyecto se han analizado diversas herramientas que permiten la visualización y modificación de la información contenida en ficheros BCF. Básicamente todas ellas son capaces de lo mismo, crear incidencias, añadir vistas previas, comentarlas, ordenarlas, filtrarlas..etc. En lo que se diferencian es que algunas son de pago y otras de código abierto.

De todas ellas se destacan las siguientes:

- KUBUS BCF Manager. Es de pago, compatible con Archicad, Revit, Tekla, Solibri, SimpleBim y una versión autónoma.

Figura 1. Captura del BCFManager de KUBUS. Fuente: https://www.solibri.com/wp-content/uploads/2014/01/2013_KUBUS_launches_BCF_Manager_03-1200x878.jpg

- BCFier desarrollado por Matteo Cominetti. Es de código abierto, tiene una versión autónoma y plugins para Revit y XbimXplorer, aunque se están desarrollando plugins para Archicad y Naviswork.

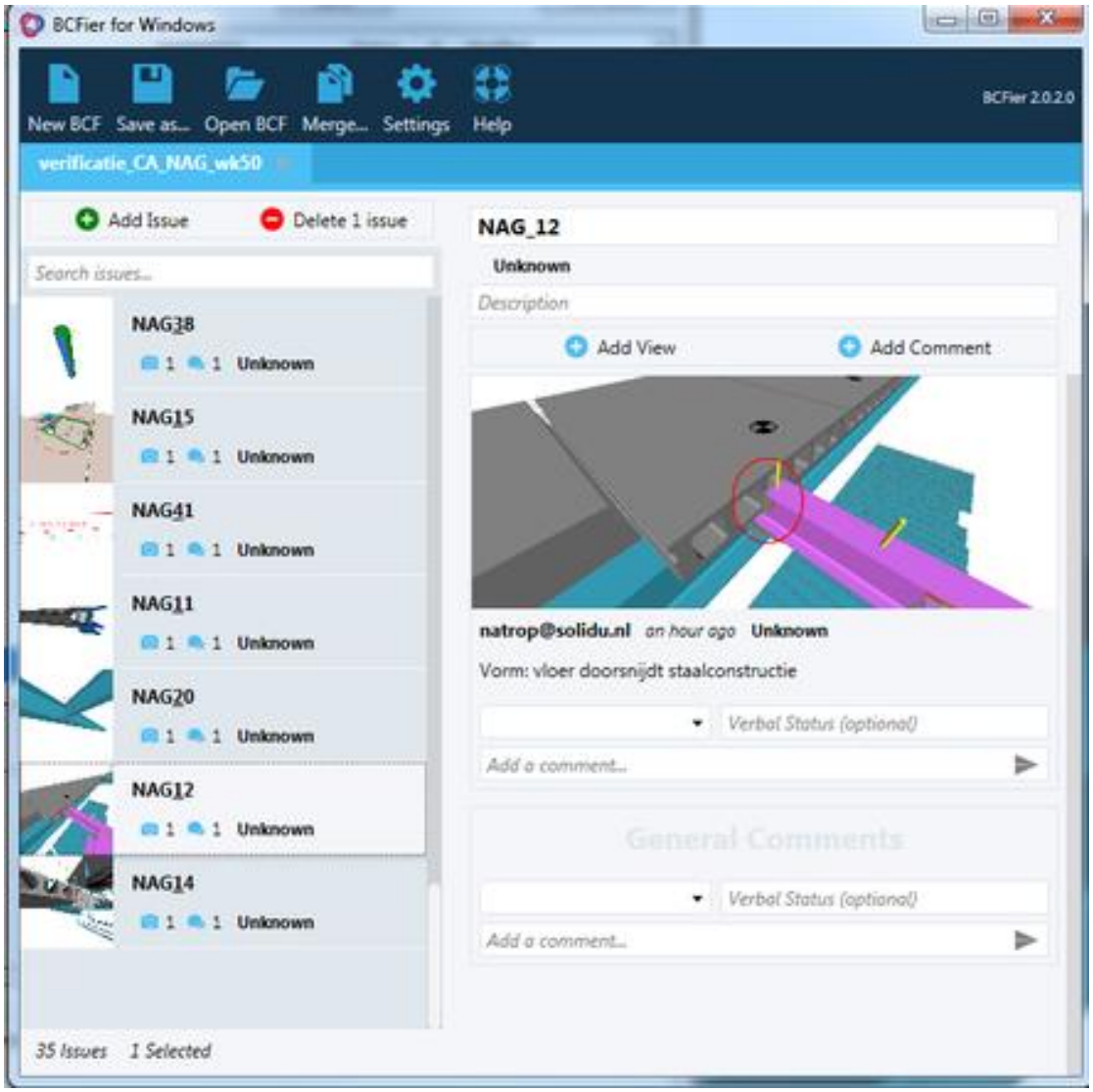


Figura 2. Captura del BCFier de Matteo Cominetti. Fuente: https://pbs.twimg.com/media/CV8MdxqUEAExD9a.png:large

- Tekla BIM Sight. Es gratuita, pero no de código abierto. Esta herramienta sirve también para visualizar el fichero IFC y completa su funcionalidad añadiendo el manejo de incidencias como parte del mismo.También está disponible para iOS y Android.

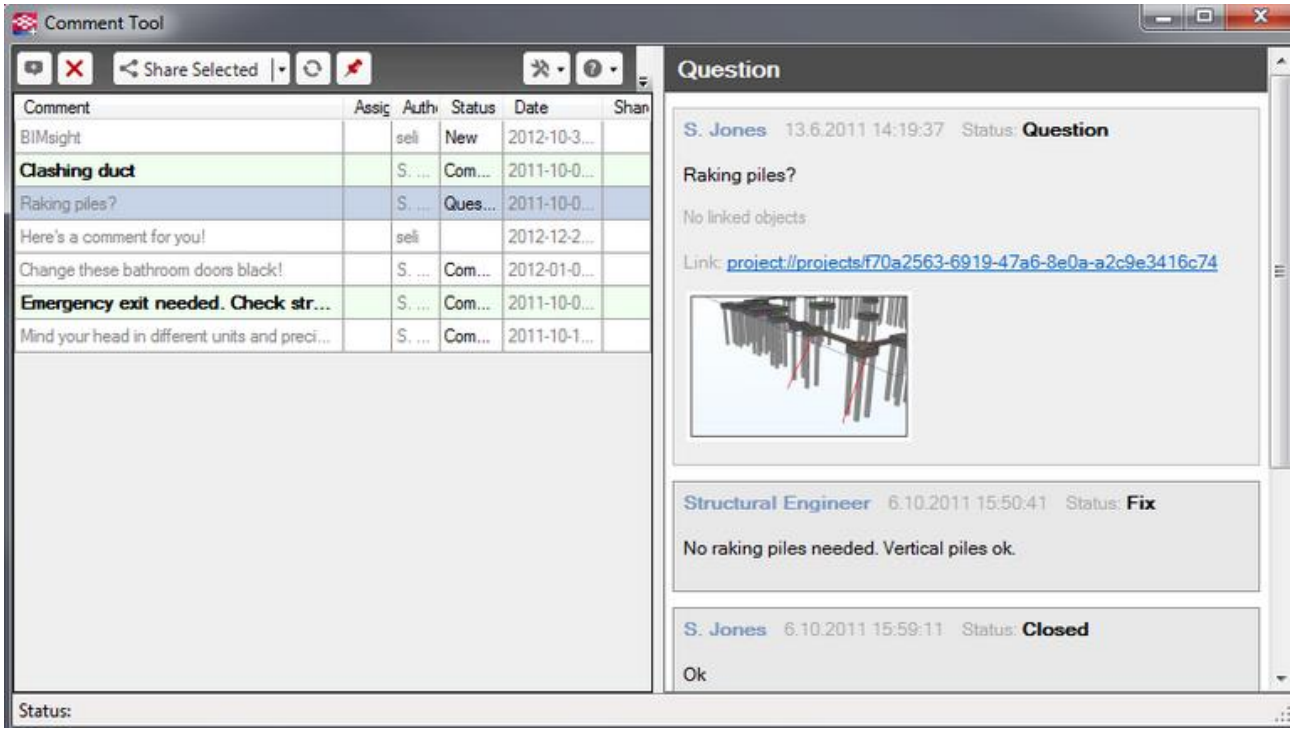


Figura 3. Captura de la herramienta Tekla BIM Sight. Fuente: http://teocomi.com/assets/2013/08/CommentTool_image1-470x305.png

1.4. Destinatarios

Este trabajo está dirigido al sector de la arquitectura, ingeniería, construcción y mantenimiento (architecture, engineering, construction and operations AEC/O). Un campo muy específico pero con aplicaciones transversales: desde los profesionales del sector que desarrollan y ejecutan proyectos (arquitectos, ingenieros de distintas disciplinas, arquitectos técnicos, constructores) hasta la administración pública que tendrá que chequear que cumplen las exigencias normativas.

Puede ser una herramienta útil para todo aquél que disponga de un equipo de profesionales en distintas disciplinas del AEC y requiera trabajar con ficheros de tipo IFC, ergo BCF también.

2. Estudio previo

A continuación se estudiarán los ficheros de trabajo, las posibles herramientas y técnicas de desarrollo para el proyecto.

2.1. Ficheros IFC

El formato IFC es un formato de datos de especificación abierta, definido con el propósito de convertirse en un estándar que facilite la interoperabilidad entre programas del sector de la construcción.

Es el fichero que contiene toda la información relacionada con la construcción y diseño del proyecto. Las clases y objetos IFC representan un modelo de información tanto geométrico como alfanumérico, formado por un conjunto de más de 600 clases y en continua ampliación. Todos los programas de software que soportan IFC pueden leer y escribir información e intercambiarla con otros programas. De este modo comunicamos "objetos", con funcionalidad y propiedades.

Obviamente la funcionalidad no es total entre aplicaciones de software, pues cada programa puede tener su parcela propia: yo puedo leer información de un muro, pero no sus propiedades acústicas (por ejemplo). Sin embargo el sólo hecho de poder traspasar de un programa a otro un muro y sus relaciones geométricas ahorra muchísimo tiempo y es una herramienta eficaz para el desarrollo del proyecto, la entrega, la documentación as-built o la gestión del mantenimiento.

Este fichero se genera a partir de las herramientas CAD, y en pocas palabras es una representación del modelo, plano, instalación o las disciplinas con las que se esté trabajando en un formato de texto que suele ser más ligero.

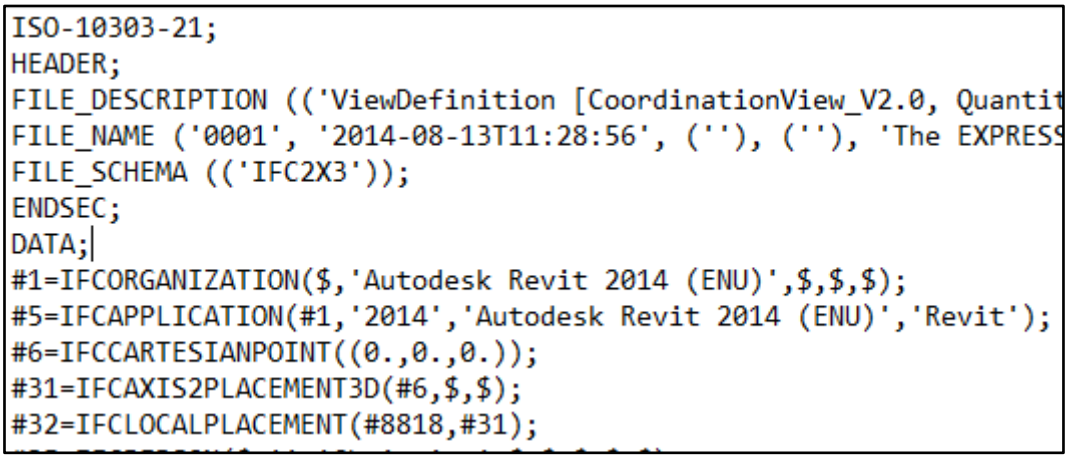


Figura 4. Ejemplo de fichero IFC generado desde Revit. Fuente: Elaboración propia

2.2. Ficheros BCF

El fichero BCF es un formato de archivo abierto donde se almacenan las incidencias con el objetivo de facilitar la comunicación de incidencias entre los miembros del proyecto. Se consigue mediante la creación de unas imágenes en donde se han detectado las incidencias en el modelo. Estas diapositivas van acompañadas de unos comentarios por parte del revisor que indican el tipo de corrección a realizar (por ejemplo, mover una distancia un conducto de ventilación que está "chocando" con un pilar). Por otro lado, capturamos el elemento que forma parte de la incidencia. En la misma diapositiva en donde generamos el problema a solucionar indicamos la especialidad afectada (instalaciones, arquitectura, estructura), quién genera el informe y a quién va dirigida.

La estructura de una incidencia viene dada generalmente por 3 ficheros:

- **Markup.bcf** : Es el que contiene el autor, los comentarios, a quien va dirigido, etc... En realidad consiste en un fichero XML común, es decir un lenguaje de etiquetas donde cada palabra clave tiene su utilidad.
  A continuación se detallarán las etiquetas más relevantes.

Etiqueta	Función
<Markup> </Markup>	Engloba todo el contenido de la incidencia
<Header> </Header>	Define la cabecera
<Filename> </Filename>	Ruta del fichero .IFC
<Topic> </Topic>	Define una incidencia
<CreationAuthor> </CreationAuthor>	Autor del creador de la incidencia
<AssignedTo> </AssignedTo>	A quién se le asigna la incidencia
<Description> </Description>	Descripción de la incidencia
<Comment> </Comment>	Define la declaración de un comentario
<Status> </Status>	Estado de la incidencia en el comentario
<Author> </Author>	Autor del comentario
<Comment> </Comment>	Contenido del comentario
<Date> </Date>	Fecha del comentario ó de la incidencia

Las que no se mencionan son opcionales o no se toman en cuenta a la hora de leer el fichero.

```
<?xml version="1.0"?>
<Markup xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="
  <Header>
    <File>
      <Filename>C:\Users\Fran-PC\Documents\GradoInformatica\TFG\
      <Date>2017-07-18T17:57:29.1793446+01:00</Date>
    </File>
  </Header>
  <Topic Guid="123469fe-c5fe-4359-b602-a1a204cc51e4">
    <Title>Incidencia en la ventana</Title>
    <CreationAuthor>franjpr</CreationAuthor>
    <AssignedTo>alguien@delproyecto</AssignedTo>
    <Description>Descripcion de la incidencia</Description>
  </Topic>
  <Comment Guid="7572015f-f6f7-44b6-ad09-7643a5654ec6">
    <VerbalStatus />
    <Status>Abierto</Status>
    <Date>2017-07-18T17:59:51.41069+01:00</Date>
    <Author>Francisco Palacios</Author>
    <Comment>Mensaje 1</Comment>
```

Figura 5. Ejemplo de un fichero markup.bcf. Fuente: Elaboración propia

- **Snapshot.png** : Es la imagen capturada de la incidencia, puede ser las piezas que colisionan, algún elemento que sobresale, etc... Es la imagen que se usará como vista previa. Va de la mano del viewpoint.bcfv.

- **Viewpoint.bcfv** : Contiene la información relacionada con la posición y ángulo de la 'cámara' en el proyecto desde la cual se observa la incidencia.

```
<?xml version="1.0"?>
<VisualizationInfo xmlns:xsd="http://v
  <PerspectiveCamera>
    <CameraViewPoint>
      <X>-6.824388022229007</X>
      <Y>-7.79982614478395</Y>
      <Z>-1.2286300123474043</Z>
    </CameraViewPoint>
    <CameraDirection>
      <X>-3.9120848058795588</X>
      <Y>4.6622411219402009</Y>
      <Z>-2.0928554156450594</Z>
    </CameraDirection>
    <CameraUpVector>
```

Figura 6. Ejemplo de viewpoint.bcfv. Fuente: Elaboración propia

Cuando se generan incidencias, éstas vienen separadas por carpetas y se comprimen en un fichero de extensión **.bcfzip** el cual también incluye 2 ficheros para añadir información del proyecto y la versión.

- **bfc.version** : Contiene la información que especifica la versión del plugin o herramienta que procesa las incidencias.

- **project.bcfp** : Contiene el nombre del proyecto en el procesador de incidencias.

```
<?xml version="1.0"?>
<ProjectExtension xmlns:xsd="http://www.w3.org/2001/X
  <Project ProjectId="228d1e4f-d37b-4607-9225-647ee1d
    <Name>Nombre del proyecto de incidencias</Name>
  </Project>
  <ExtensionSchema />
</ProjectExtension>
```

Figura 7. Ejemplo de project.bcfp. Fuente: Elaboración propia

Al final el fichero de incidencias generado tendrá la siguiente estructura.

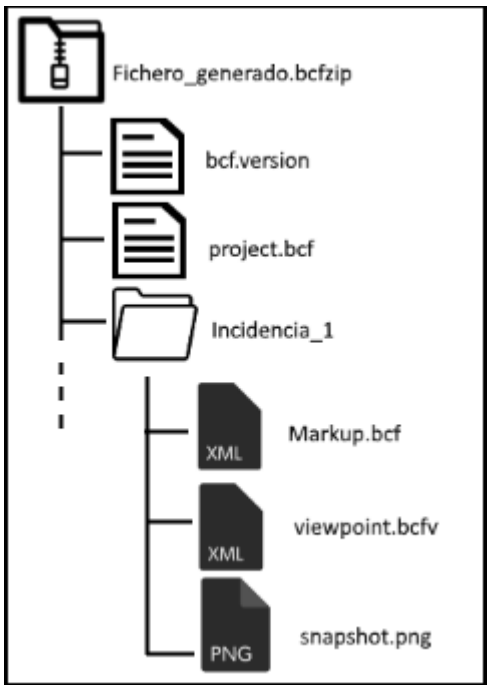


Figura 8. Ejemplo de la estructura de un fichero bcfzip. Fuente: Elaboracion propia

2.3. Análisis de las herramientas

Este proyecto se desarrolló en NodeJS usando el framework ExpressJS y se publicó en Heroku. Se han elegido ambas herramientas porque son gratuitas y por tener experiencia previa al haber trabajado con ellas en algunas de las asignaturas cursadas. También se necesitaba almacenar ciertos datos, para lo que se optó por usar una base de datos de fácil acceso y gratuita como lo es mLab. Para trabajar con los ficheros de la incidencia se recurrió a con 2 librerías, una es JSZip para poder ver el contenido de los *.bcfzip* y la otra es x2js que me permitía convertir un XML en un objeto JSON que es más cómodo de manipular a la hora de escribir en javascript.
Por último, dada la necesidad de trabajar con la API de github en nodejs, se eligió node-github ya que de las herramientas que ofrecían para javascript es la más completa.

- ***NodeJS*** es una plataforma de código abierto que fue desarrollada en 2009 por *Ryan Dahl*, está basado en el motor V8 de Google y tiene como función principal desarrollar servidores web, en pocas palabras, es una plataforma que nos permite correr código javascript en un ordenador o servidor.
  La gran ventaja que ésto nos aporta es que ya no hace falta aprender un lenguaje de programación para escribir el comportamiento del servidor como *Ruby* o *PHP*, sino que nos permite desarrollar una aplicación web al completo solamente con *Javascript*.

  NodeJS es una plataforma que nos permite desarrollar aplicaciones altamente escalables y para ello incorpora varios "módulos básicos" como por ejemplo, un módulo de red que proporciona una capa para la programación asíncrona entre otros módulos fundamentales, como *Path, FileSystem, Buffer, Timers* y el de propósito más general *Stream*.

Además cuenta con NPM *(Node Package Manager)* que nos da la posibilidad de instalar los módulos que necesitemos para cumplir nuestros objetivos, el npm se sirve de *npmjs* que es donde se aloja una lista muy extensa de módulos para NodeJS.

- **ExpressJS** Es un módulo de NodeJS que nos permite crear aplicaciones web de una manera rápida y sencilla, en su API dispone de miles de métodos que nos simplifican el trabajo. ExpressJS gestiona el direccionamiento, los motores de renderizado de vistas y los middlewares.
  El direccionamiento hace referencia a cómo responde la aplicación a la solicitud del cliente mediante una vía de acceso (URI) y un método de solicitud HTTP (get, post, etc), este comportamiento se define en un *manejador* que es una función de javascript que recibe normalmente 3 argumentos para manipular la petición (***req****uest*), la respuesta (***res****ponse*) y pasa el control al siguiente middleware (*next*).

  Por ejemplo:

```
app.get('/contacto', function(req, res, next) {
     res.send('Hola mundo');
});
```

  Cuando el usuario quiera acceder a la ruta miweb.com/contacto, el navegador envia una petición de tipo GET al servidor y este le enviará como respuesta el texto del ejemplo. Las rutas también se pueden escribir como expresiones regulares de javascript.

  Los middlewares son funciones que tienen acceso a los objetos *request*, *response* y *next*. Pueden realizar las siguientes tareas:

  - Ejecutar cualquier código.
  - Realizar cambios en la solicitud y los objetos de respuesta.
  - Finalizar el ciclo de solicitud/respuestas.
  - Invocar el siguiente middleware en la pila.

Si la función de middleware actual no finaliza el ciclo de solicitud/respuestas, debe invocar next() para pasar el control a la siguiente función de middleware. De lo contrario, la solicitud quedará colgada.

Ejemplo de definición y uso de un middleware.

```
var checkSession = (req, res, next) => {
  if(req.session.username && req.session.password)
     next();
  else
     res.redirect('/?badLogin=true');
};

app.get('/:usuario', checkSession, (req, res, next) => {
   res.send(`Hola ${req.params.usuario}!`);
});
```

Si el middleware *checkSession* ejecuta *next()* se ejecutará la siguiente función, que en este caso es la que envía al cliente el texto "Hola <nombreUsuario>!", en caso contrario, finaliza el ciclo de solicitud/respuesta redirigiendo al cliente a la ruta especificada (badLogin).

Las vistas se renderizan según el motor elegido, existen varios como *handlebars*, *pug*, *ejs*, entre otros... En este proyecto se ha elegido *EJS*. Para su uso se debe definir mediante el método ***.set*** el motor elegido y la ruta donde se encuentran los ficheros de las vistas.

```
app.set('viewengine', 'ejs'); //Motor ejs
app.set('views', __dirname + '/views'); //Carpeta 'views'
```

Finalmente para renderizar una vista, en el *manejador* de una ruta se debe llamar al método ***render()*** desde el objeto *response*, además se puede enviar información adicional en formato *JSON* a la vista, ya que EJS nos permite ejecutar código javascript dentro de la misma.

Ejemplo:

```
app.get('/contacto', (req, res, next) => {
   res.render('ruta/hasta/contacto.ejs', { información: Adicional });
});
```

- **Heroku** es una plataforma en la nube que permite a las empresas construir, publicar, supervisar y escalar aplicaciones desarrolladas ya sea en NodeJS, Ruby, Java, Python.. entre otros lenguajes de programación, además es donde se aloja este proyecto.

- **Mlab** es un servicio de base de datos en la nube que provee soporte para MongoDB, copias de seguridad, herramientas de administración basadas en web y soporte experto entre otras cosas. En este proyecto se almacenan ciertos datos en la base de datos de mLab, a la cual nos conectamos mediante un paquete de node llamado mongoose.

- **JSZip** Es una librería de javascript que nos permite crear, leer y editar archivos comprimidos en .ZIP que dispone de una API de fácil comprensión y varios ejemplos en su web.

- **X2JS** Es una librería de javascript que nos permite convertir cadenas de texto con estructura XML a JSON y viceversa.

- **node-github** Es un módulo de NodeJS que nos facilita muchísimo la interacción con la API de github, sólo necesitamos instalarlo con npm e incluirlo con un require('node-github') en nuestro fichero de trabajo. Gracias a su extensa API se pueden hacer multitud de operaciones con nuestras cuentas de github con una simple e intuitiva función como por ejemplo api.repos.create({...}), api.users.getForUser({ ... }), etc... El único inconveniente de esta herramienta frente a las demás es que hay que autenticarse cada vez que se haga una petición, pero es mucho más completa que el resto.

- **email-via-gmail** Es un módulo de NodeJS que nos permite enviar emails desde la aplicación a uno o más destinatarios, sólo con 2 funciones muy simples, este módulo se usa para el sistema de notificaciones.

2.4. Lenguajes de programación

El proyecto se desarrolla al completo en *Javascript*, gracias a la plataforma NodeJS y el módulo ExpressJS se pudo hacer el funcionamiento de la aplicación web al completo sólo con javascript, tanto el comportamiento en el cliente (navegador) como el del servidor.
También se usó el lenguaje de etiquetas HTML para elaborar la estructura visual de la aplicación web y un preprocesador de CSS *(SASS)* para agilizar la codificación de la hoja de estilos.

2.5. Entornos de desarrollo

Para el desarrollo de la aplicación web se usó el editor Atom, es un editor de texto muy polivalente a la hora de programar, ya que dispone de opciones de autocompletado, funciona con github, e infinidad de plugins gratuitos para la personalización del mismo.

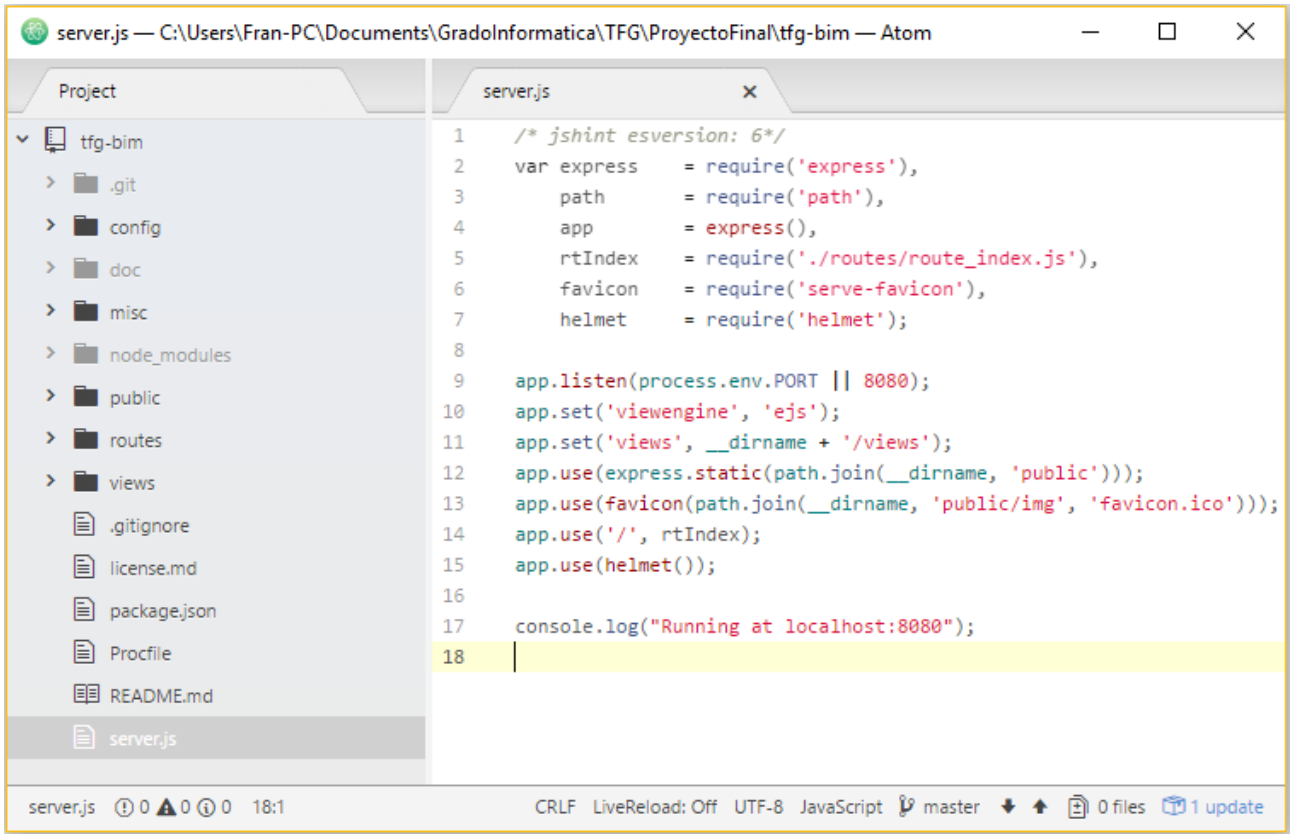


Figura 9. Captura del editor Atom y fichero principal del servidor. Fuente: Elaboración propia

3. Diseño e implementación

En primer lugar se plantea el esquema de funcionamiento de la aplicación, donde el cliente a través de la interfaz de la misma gestiona sus proyectos alojados en los servidores de github, y con la base de datos de mLab mantiene una lista actualizada de sus proyectos activos.

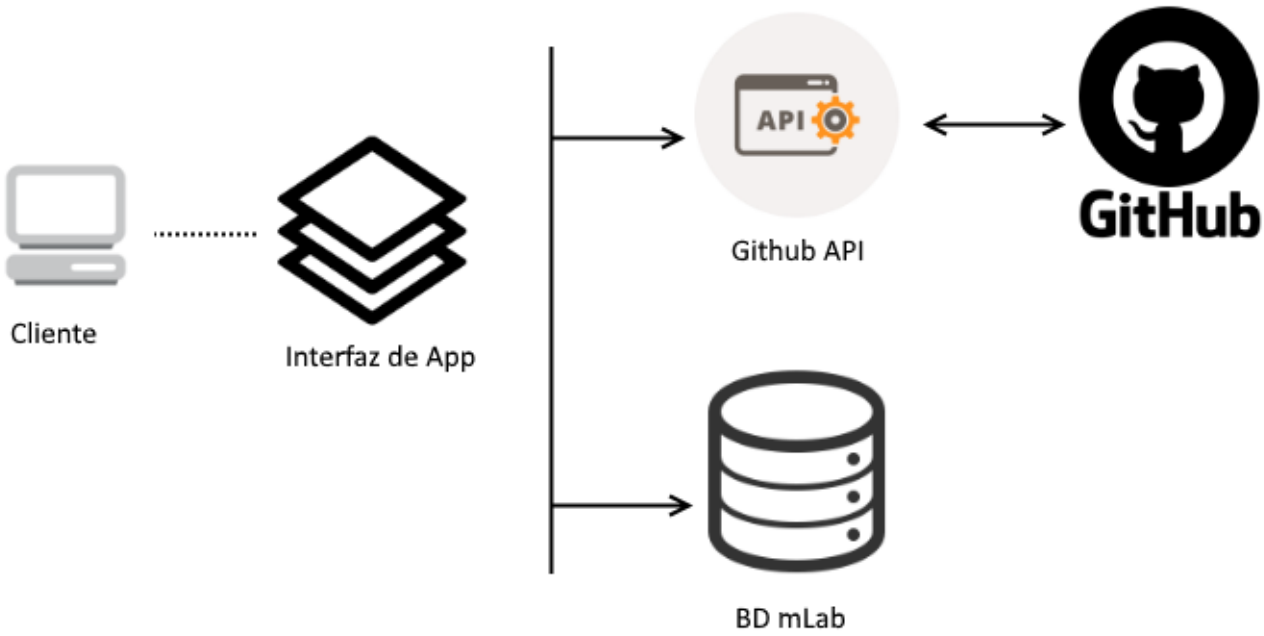


Figura 10. Esquema de funcionamiento. Fuente: Elaboración propia

El diseño e implementación de este proyecto se ha estructurado en 2 partes. La primera, el comportamiento del servidor, donde tratamos la comunicación con la API, la BD y las respuestas al cliente. Y la segunda es el comportamiento del cliente donde se diseña la interfaz y visualización de los recursos solicitados.

3.1. Principios de programación

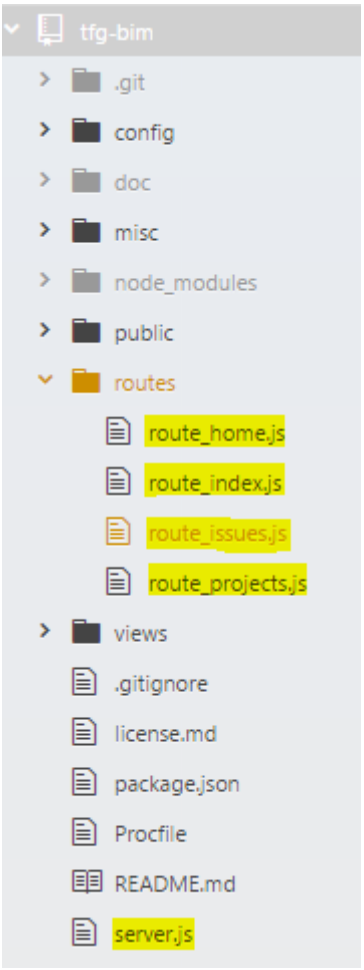
A continuación detallaré los principios de programación usados en este proyectos y se acompañarán de ejemplos de su uso en el código.

Separación de intereses: Es un principio de diseño para separar un programa informático en secciones distintas, tal que cada sección enfoca un interés delimitado. El valor de la separación de intereses es simplificar el desarrollo y mantenimiento de programas informáticos. Cuando los intereses están bien separados, se pueden reutilizar, desarrollar y actualizar las distintas secciones individuales de forma independiente. La posibilidad de modificar una parte del código del programa sin tener que revisar y modificar las demás es de gran valor en el mantenimiento de software.

El código que define la función del servidor se ha separado en 5 intereses, siendo ellos los que se ven resaltados en la figura 9.

En el fichero **server.js** es donde se declaran las herramientas que usará como por ejemplo el puerto, motor para renderizar las vistas, los ficheros que se van a servir, etc.

Luego cada una de las rutas sólo dispone de la descripción de su comportamiento y de las herramientas que necesiten como pueden ser las sesiones, cookies, conexiones con bases de datos y sobretodo la conexión con la API de github.



- **route_index.js** Responde a la ruta *"/" siendo la raíz de la aplicación,* comprueba el inicio de sesión y define un middleware que guarda las credenciales para futuras peticiones.

- **route_home.js** Responde a la ruta *"/u/usuario"* en esta vista se consulta la base de datos de mLab y muestran los proyectos activos del usuario.

- **route_projects.js** Responde a la ruta *"u/usuario/projects"* en la vista se muestran todos los proyectos (repositorios) en los que trabaja el usuario, dónde es capaz de crear y eliminar repositorios asociados a su cuenta en github y también modificar su lista de proyectos activos.

- **route_issues.js** Responde a la ruta *"u/usuario/p/proyecto-dueño"* en esta parte el usuario es capaz de descargar y subir ficheros BCF o IFC al proyecto, también puede añadir colaboradores y añadir mensajes a las incidencias.

Figura 11. Separación de intereses. Fuente: Elaboración propia

Encapsulamiento: Es el ocultamiento del estado de un objeto. Cada objeto está aislado del exterior y la aplicación entera se reduce al uso combinado de objetos. El aislamiento protege a los datos asociados de un objeto contra su modificación por quien no tenga derecho a acceder a ellos, y por lo tanto se evita que el usuario pueda cambiar su estado de manera imprevista e incontrolada.

El comportamiento en el cliente también se ha separado por intereses pero aplicando encapsulamiento (también conocido como *funciones anónimas de auto-invocación*) para evitar lo mencionado en su definición.
Ejemplo de encapsulamiento.

```
(() => {
  //Cuerpo de la función.
  //Se crea un ámbito temporal que no afecta el
  //espacio de nombres global
}).call(this)
```

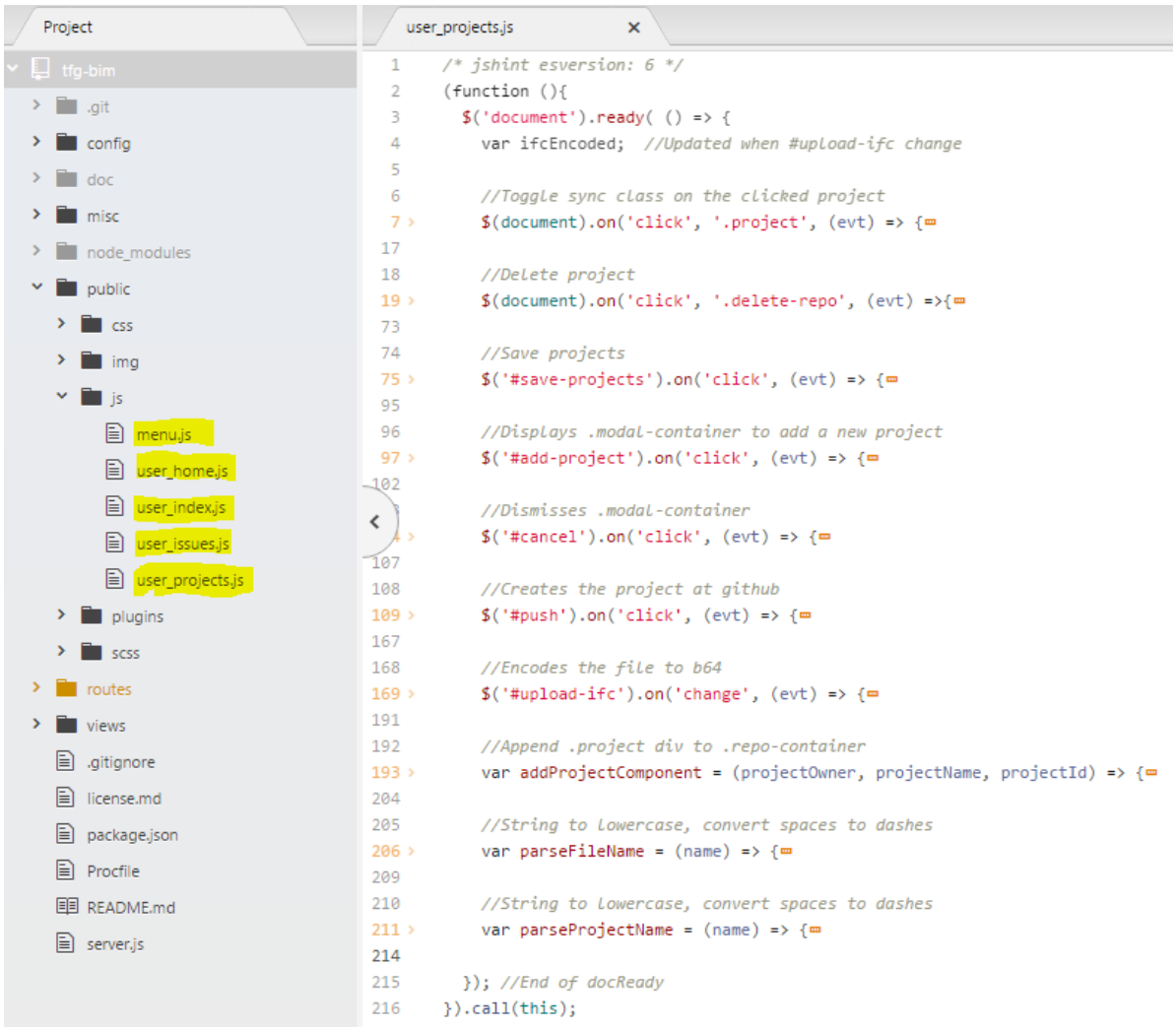


Figura 12. Encapsulamiento. Fuente: Elaboración propia

3.2. Manipulación del BCF

Se divide en 2 tareas, que son la subida del fichero a github y la descarga (lectura) del mismo. En primer lugar recordemos que las incidencias se han generado con BCFier y son en formato **.bczip** que realmente sigue siendo un comprimido .ZIP.

En la primera tarea, la aplicación web recibe el fichero de la incidencia (bcfzip) y con la ayuda de la librería JSZip la descomprime en el navegador, luego la envía al servidor para tenerlas ordenadas por carpetas.

Pseudocódigo para descomprimir un fichero zip con JSZip:

```
JSZip.loadAsync(CONTENIDO DEL ZIP).then((FICHERO) => {
  //Se lee el zip y cada fichero se guarda en una lista temporal
}).then(() => {
  Promise.all(LISTA TEMPORAL.map((FICHERO) => {
  //Se itera sobre la lista diferenciando si el fichero es imagen o
  //texto para poder codificarlo en el formato correspondiente BASE64
  // o STRING y luego se guardan en otra lista que es la que se
  // enviará al servidor
  })).then(() => {
     //Cuando se termine de codificar los ficheros se envían a
     // la ruta '/u/USUARIO/p/PROYECTO/ulbcf'
     //mediante una petición ajax de tipo POST
  });
});
```

El servidor recibe estos ficheros en forma de *JSON* con la siguiente estructura:

```
ListaFicheros:
[
  { fichero1
    name: nombre (cualquiera de los contenidos en el .bcfzip),
    content: contenido en string para xml o base64 para las imágenes
  },
 {fichero2},
 {fichero3},
 ...]
```

Cuando el navegador hace la petición POST al servidor, este ejecuta el código correspondiente para crear el commit y enviarlo a github.

Pseudocódigo de los pasos para crear un commit mediante la api:

```
rtIssues.post('/:projectname/ulbcf', (req, res, next) => {
  //Se Autentica al usuario mediante la API
  api.authenticate({username: USUARIO, password: CONTRASEÑA});
  let listaFicheros = listaEnvidadaDesdeElCliente.

  //Se itera sobre los ficheros
  Promise.all(listaFicheros.map((file) => {
     //Primero. Se crea un blob por cada fichero
     return api.gitdata.createBlob({});
  })).then((blobs) => {
     //Segundo. Se crea un árbol que contiene los blobs
     return api.gitdata.createTree({});
  }).then((tree) => {
     //Tercero. Se crea el commit
     return api.gitdata.createCommit({});
  }).then((commit) => {
     //Cuarto. Finalmente se actualiza la referencia a que apunte al
     // último commit
     return api.gitdata.updateReference({});
  }).then((resp) => {
     //Se ha creado correctamente, se envia un OK al navegador
     // e información sobre el commit
     res.status(200).send(commitInfo);
  }).catch((err) => {
     //Se trata el error
     res.sendStatus(409);
  });
});
```

Una vez culminado este proceso ya estará almacenada en nuestra cuenta de github la incidencia asociada al repositorio en el que se esté trabajando.

La segunda tarea es la lectura de una incidencia, para ello primero tenemos que 'descargarla' de github (en el servidor) y luego manipularla (en el navegador).

Cuando se solicita ver las incidencias primero se selecciona el commit que se desee y luego se da al símbolo de actualizar.

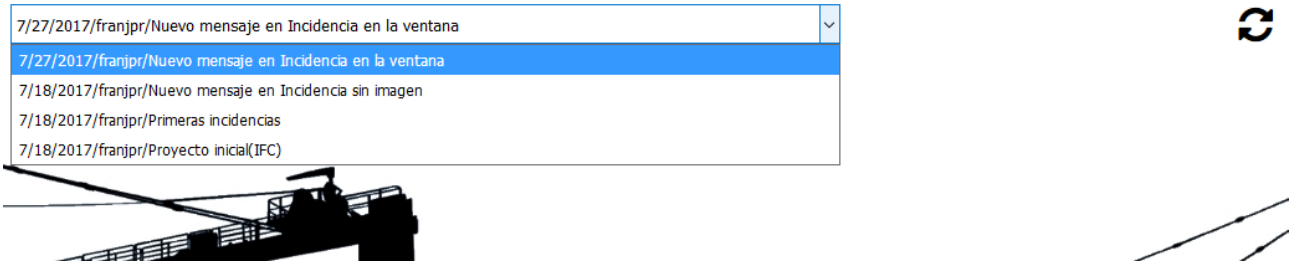


Figura 13. Selección del commit. Fuente: Elaboración propia

Cada commit tiene asociado un ID que identifica el estado en ese momento del repositorio, así tendremos la posibilidad de ver el proyecto en cualquier versión en el tiempo. El navegador envía dicho ID al servidor y este lo pide a github.

Pseudocódigo de la petición de incidencias:

```
rtIssues.post('/:projectname/getissues', (req, res, next) => {
  //Autenticación del usuario
  api.authenticate({username: USUARIO, password: CONTRASEÑA});

  api.gitdata.getTree(idRecibidoDelNavegador).then((res) => {
     //Se pide el contenido del arbol segun el ID
     //Se recorre el árbol pidiendo cada fichero de la incidencia
     return listaDeFicheros;
  }).then((issues) => {
     //Se recorre la lista de ficheros y se
     //ordenan quedando de la siguiente forma
     //{name:'nombreFichero', markup: base64, snapshot: base64}
     //y se envian al navegador
     res.status(200).send(incidenciasOrdenadas);
  }).catch((err) => {
     //Se trata el error
     res.status(409).send(err);
  });
});
```

Luego el navegador recibe dicha lista y la recorre para tratar cada incidencia por separado

Pseudocódigo de cómo se leen las incidencias:

```
let agregarIncidencia = (incidencia) => {
  //La incidencia se parsea de XML a JSON para su manipulación
  let x2js = new X2JS();
  //Se decodifica con la función atob()
  //porque github siempre almacena los datos en base64
  let jsonbcf = x2js.xml_str2json(atob(incidencia.markup));
  let vistaPrevia = atob(incidencia.snapshot);
  // Luego se crean los distintos elementos HTML para
  // mostrarlos en pantalla
};
```

Como se ve en el pseudocódigo tenemos que usar la herramienta x2js que lee una cadena de texto con estructura XML y la convierte en un objeto JSON.

Por ejemplo si tenemos este XML:

```
<Etiqueta>
     <Nombre> minombre </Nombre>
     <Email> mi@email.com </Email>
     <Contactos>
        <cont> cont1@mail.com </cont>
        <cont> cont2@mail.com </cont>
        <cont> cont3@mail.com </cont>
     </Contactos>
</Etiqueta>
```

Lo convierte a JSON quedando de la siguiente forma:

```
{
  Nombre: minombre,
  Email: mi@email.com
  Contactos: [ cont1@mail.com, cont2@mail.com, cont3@mail.com]
}
```

De esta manera es más fácil acceder a los campos de información en javascript, por ejemplo para leer el autor de una incidencia si recordamos cómo era la estructura de markup.bcf sería algo así como $Markup.Topic.CreationAuthor$ y esto nos devolvería el valor de ese campo.

3.3. Diseño de la interfaz

En esta sección se describirán la interfaz y funcionamiento de la aplicación web.

- El usuario debe tener una cuenta de github. Esta cuenta con 4 vistas que desempeñan las siguientes funciones:
    - **Inicio de Sesión**: Se le pide al usuario su nombre de usuario y contraseña de github, estos datos se envían al servidor y a través de la API se comprueban que son correctos.

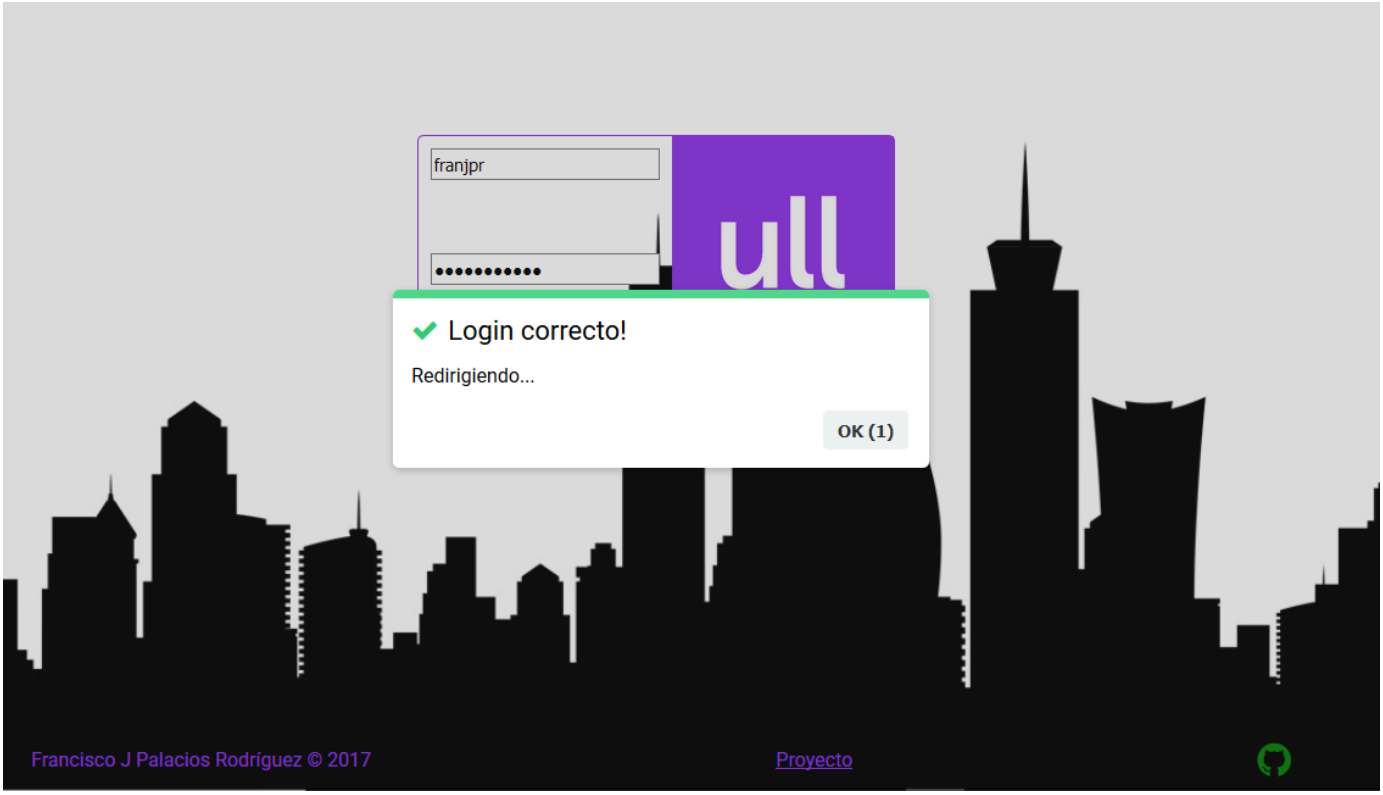


Figura 14. Inicio de sesión. Fuente: Elaboración propia

- **Proyectos Activos**: En esta vista se mostrarán los proyectos marcados como activos, en caso de que no haya ninguno se mostrará un mensaje de información que describe como marcar los proyectos. Cada proyecto tendrá un icono para su edición, que redirigirá al cliente a la vista de manipulación de incidencias.

Figura 15. Proyectos activos. Fuente: Elaboración propia

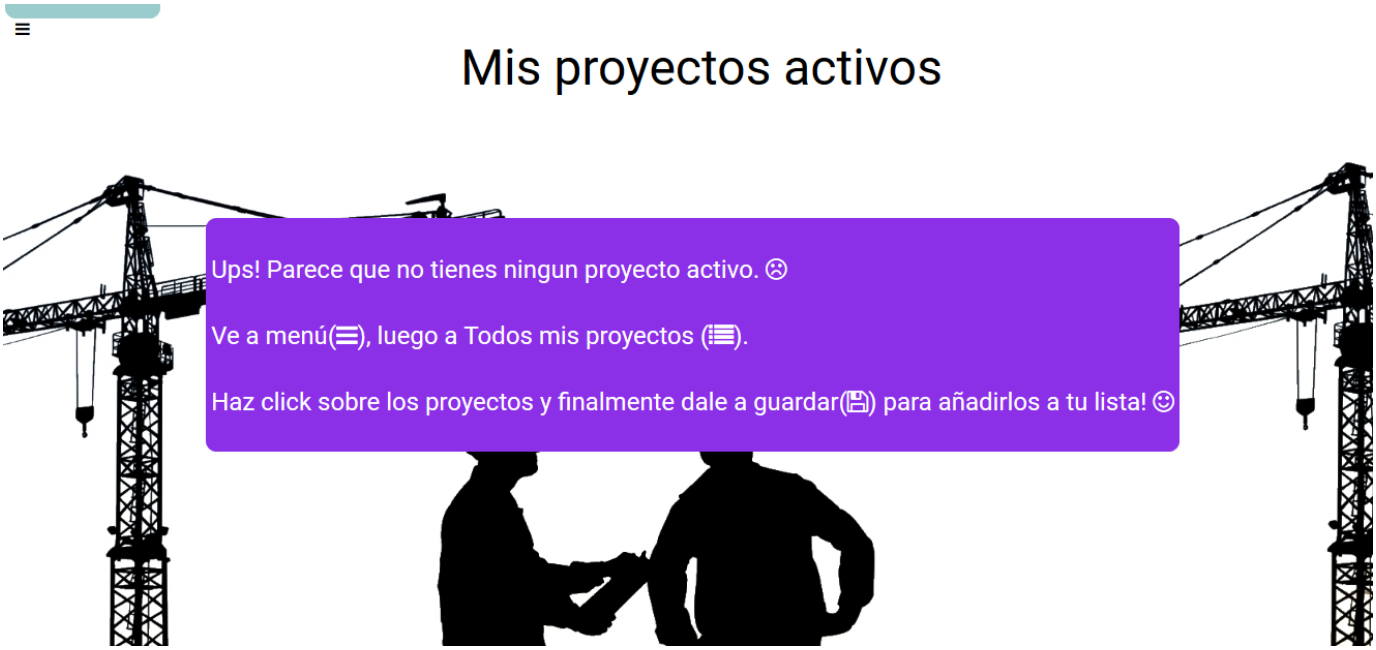


Figura 16. Ejemplo al no tener proyectos activos. Fuente: Elaboración propia

- **Todos mis proyectos**: Aquí se muestran todos los proyectos (repositorios) que pertenecen al usuario o de los que forma parte como colaborador. Permite crear y eliminar los repositorios, también al hacer click sobre ellos se marcarán con un icono de visto que los clasifica como proyectos activos, esta clasificación no será permanente hasta que se clickee en el icono de guardar.

Figura 17. Todos los proyectos. Fuente: Elaboración propia

Figura 18. Ejemplo de crear un proyecto. Fuente: Elaboración propia

- **Manejo de incidencias**: Se podrá añadir o quitar colaboradores al proyecto, también dispone de una lista desplegable que muestra los commits para visualizar las incidencias de la fecha, al pasar el cursor por encima de la incidencia se mostrará información sobre la misma, además se puede descargar la *versión* del proyecto o incidencia del commit seleccionado, también se puede en cualquier momento subir los ficheros **.ifc** o **.bcfzip** para actualizarlos y por último se pueden ver y añadir mensajes sobre las incidencias.

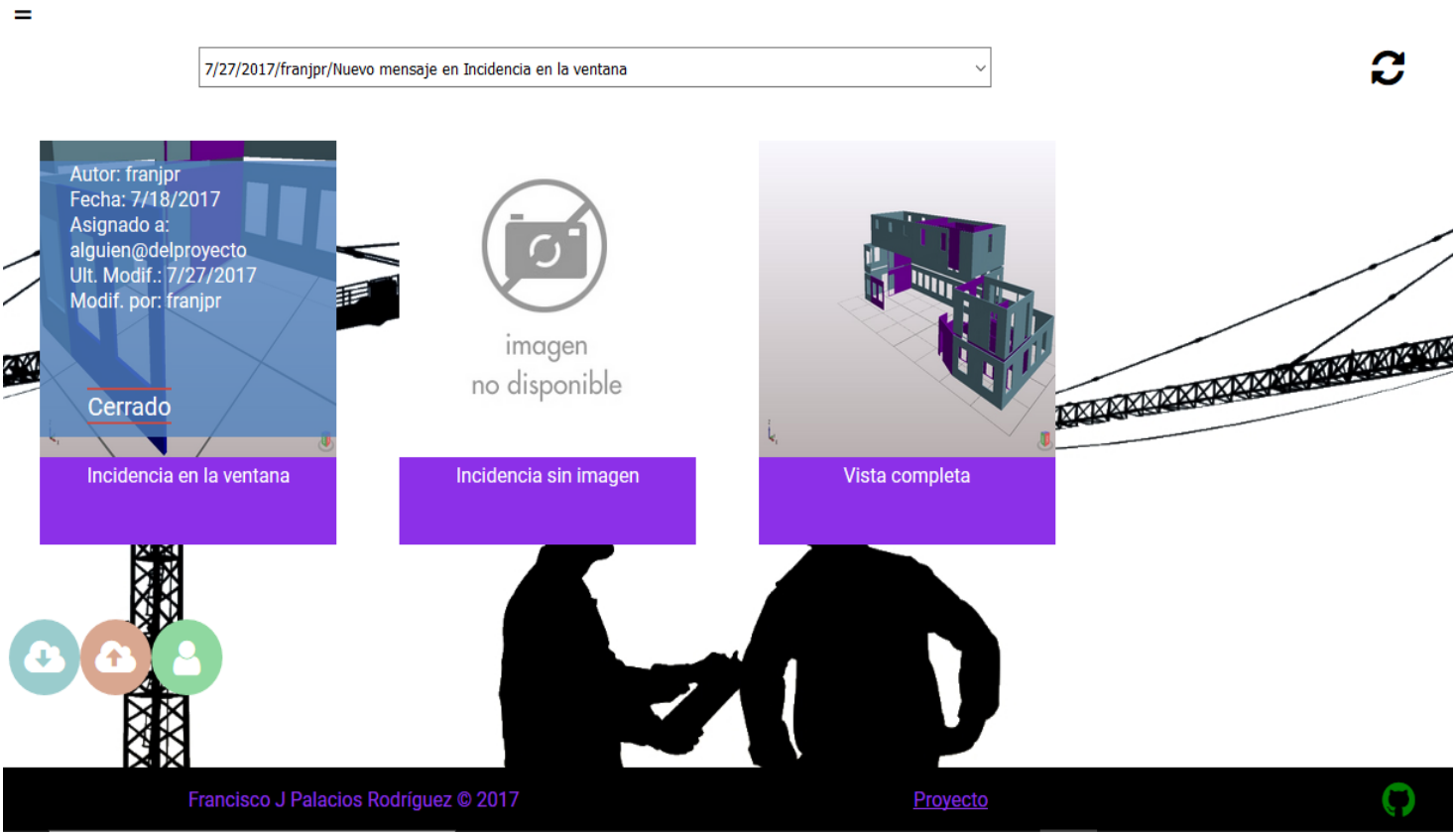


Figura 19. Vista del manejo de incidencias. Fuente: Elaboración propia

Sólo el dueño del proyecto tiene la posibilidad de agregar colaboradores.

Figura 20. Ventana al agregar colaboradores. Fuente: Elaboración propia

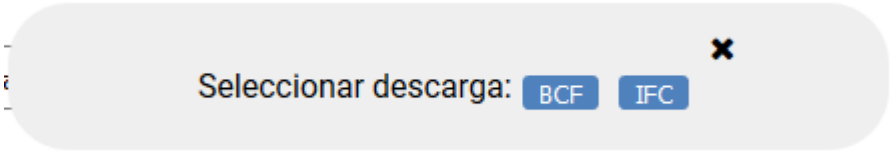


Figura 21. Ventana para descargar el proyecto o las incidencias. Fuente: Elaboración propia

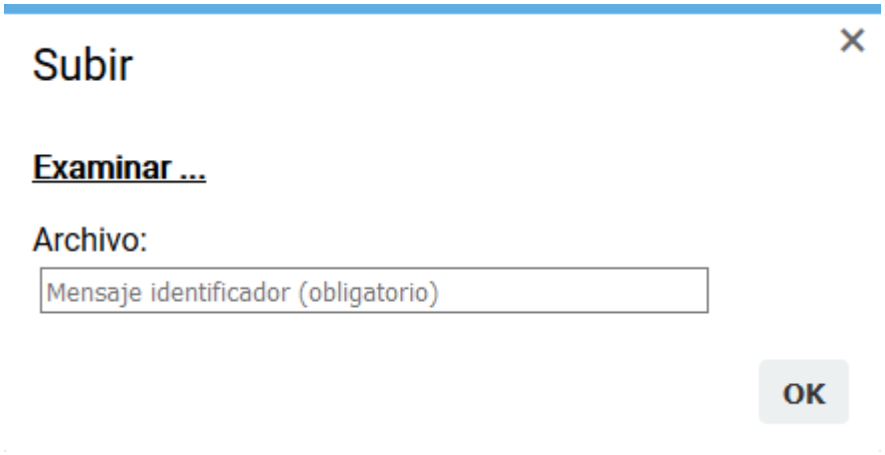


Figura 22. Ventana para actualizar el proyecto o incidencias. Fuente: Elaboración propia

Al clickear sobre la incidencia se mostrará una ventana similar a un chat donde estarán los comentarios sobre la incidencia, también se puede cambiar el estado de la misma a través de la lista desplegable al enviar un nuevo mensaje.

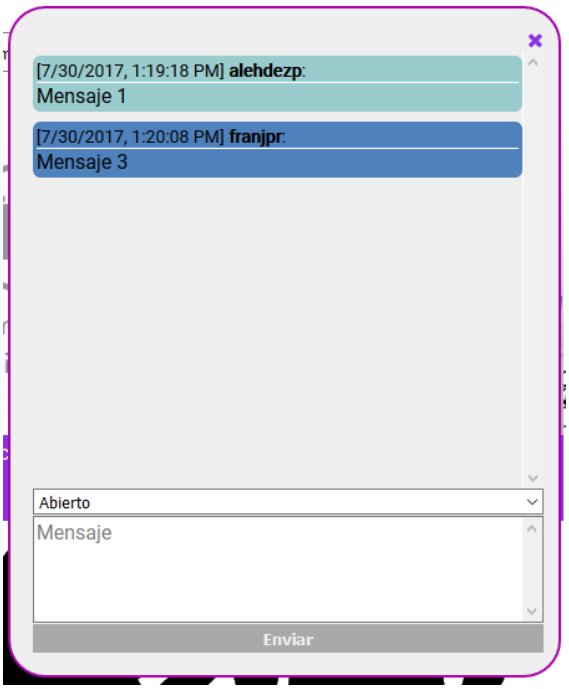


Figura 23. Vista de los comentarios en las incidencias. Fuente: Elaboración propia

Cada acción al enviar un mensaje o subir un fichero crea un commit y lo añade automáticamente a la lista desplegable de selección de commits de la figura 13.

Finalmente se puede navegar de una vista a otra a través del menú desplegable en la parte superior izquierda, que muestra la imagen de github, el nombre de usuario y 3 iconos con su acción correspondiente.

Figura 24. Menú de la aplicación web. Fuente: Elaboración propia

3.4. Sistema de notificaciones

La aplicación cuenta con un sistema de notificaciones muy simple, al añadir a un usuario como colaborador a un proyecto, mediante el módulo *email-via-gmail* se les envía un correo a su dirección de email con la que se hayan registrado en la plataforma de Github, además dicho email deberá ser visible, ya que sin esto tampoco será visible desde la API. Para ello deben ir a la web de github , y en configuración, seleccionarlo como público como se ilustra en la figura 25.

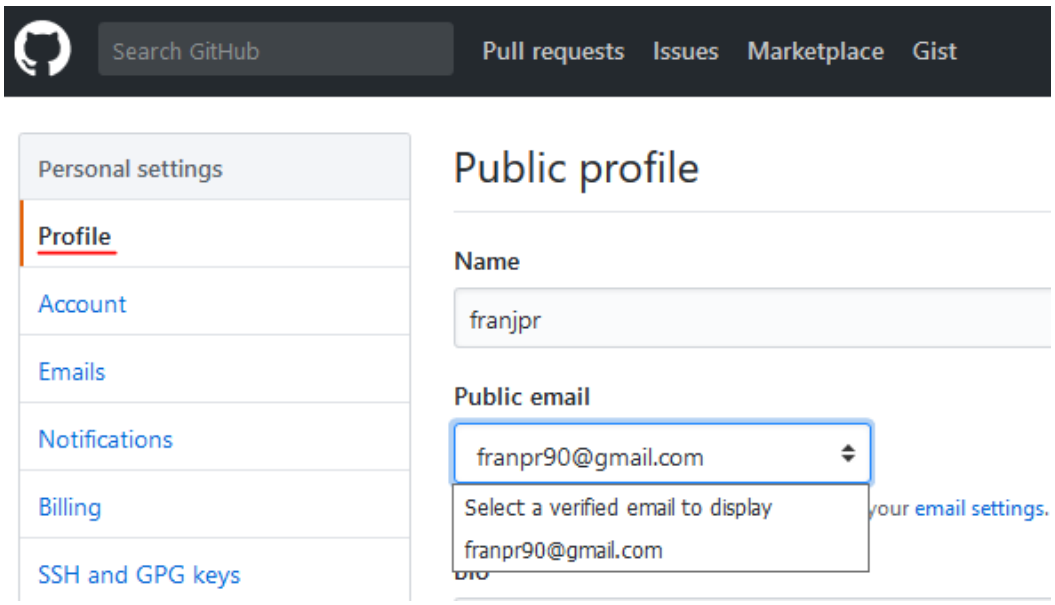


Figura 25. Configuración de perfil de github. Fuente: Elaboración propia

Finalmente en cada incidencia se dispone de un icono de aviso para notificar a los que tengan asignaciones, se les notificará de la misma manera, mediante el email con el que se hayan registrado en github, se podrá notificar a uno o más implicados, siempre que sus nombres de usuario estén separados por comas tal y como se observa en la figura 26.

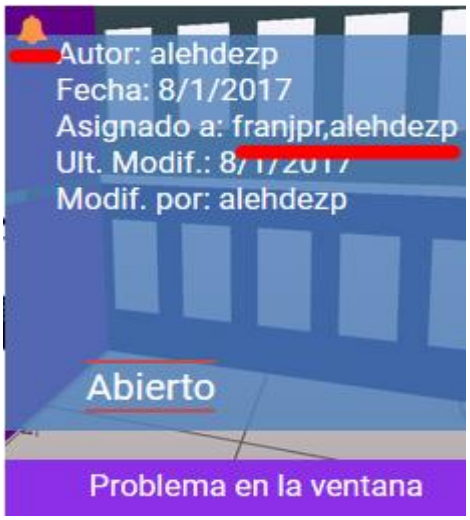


Figura 26. Notificar a los implicados en una incidencia. Fuente: Elaboración propia

4. Líneas futuras

Este proyecto se puede ampliar en distintas líneas de trabajo y nuevas funcionalidades como la posibilidad de automatizar el sistema de notificaciones en cuanto a los asignados se refiere o añadir menciones en los mensajes, otra mejora sería añadir algún tipo de filtrado para la búsqueda de proyectos.

5. Conclusiones

During the realization of this work I have learned that it's very important to establish a good foundation and focused on the final state of the project to avoid having to change the structure very often if there were adversities along the way.

It has also been a novelty to work with the github's API, I have seen the great potential it has and how useful it can be.

The application developed can help you to understand issues files and speed up the workflow when communicating through a simple and intuitive process.

This project constitutes a first approximation to the communication and distribution of work's issues.

6. Bibliografía

Principios de programación. (s.f). En *Wikipedia.* Recuperado el 02 de Agosto de 2017 .
https://es.wikipedia.org/wiki/Categor%C3%ADa:Principios_de_programaci%C3%B3n

Códigos de estado HTTP. (s.f). En *Wikipedia.* Recuperado el 02 de Agosto de 2017 .
https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP

Margalef, A.J. 19/08/2015. El BCF como formato OpenBIM para la colaboración. Lugar de publicación:
http://www.apogeavirtualbuilding.com/el-bcf-como-formato-openbim-para-la-colaboracion/

Redacción. 20/03/2017. IFC Qué es, a qué sirve y cuál es su relación con el BIM? Lugar de publicación:
http://biblus.accasoftware.com/es/ifc-que-es-y-relacion-con-el-bim/

Jeremy A. Documentación Underscore. Lugar de publicación:
http://underscorejs.org/

NodeJS Foundation. Documentación ExpressJS. Lugar de publicación:
https://expressjs.com/en/4x/api.html#express

Stuart K. Documentación JSZip. Lugar de publicación:
https://github.com/Stuk/jszip

Abdmob. Documentación X2JS. Lugar de publicación:
https://github.com/abdmob/x2js

Matthew E. Documentación EJS. Lugar de publicación: http://ejs.co/

Automatic. Documentación Mongoose. Lugar de publicación:
http://mongoosejs.com/docs/guide.html

GitHub. Documentación Github API. Lugar de publicación:
https://developer.github.com/v3/

jQueryFoundation. Documentación jQuery. Lugar de publicación:
http://api.jquery.com/

jQueryFoundtaion. Documentación jQueryUI. Lugar depublicación:
http://api.jqueryui.com/

Boniface Pereira. Documentación jQueryConfirm. Lugar de publicación:
https://craftpip.github.io/jquery-confirm/#getting-started

Múltiples consultas en MozillaDevelopers.
Fuente: https://developer.mozilla.org/es/docs/Web

Múltiples consultas en Stackoverflow.
Fuente: https://stackoverflow.com