Informe Seguimiento 1 TFG

Aplicación de Gestión Financiera

Rafael Gómez Pérez

Mención en Ingeniería del Software

****

**Índice de contenidos**

1. [Histórico de Revisiones](#Revisiones) 3

2. [Objetivos del Proyecto](#ProjectObjectives)  4

3. [Diseño y Arquitectura](#DiseñoArquitectura) 5

3.1 Diagrama de Clases 5

3.2 Esquema de la Base de Datos 7

3.3 Diagrama Microservicios 8

4. [Metodología empleada](#Metodologias) 11

5. Planificación 11

6. Tabla de Acrónimos 11

7. [Bibliografía y Referencias](#Referencias) 12

8. [Anexo](#Anexo) 13

**1.Histórico de revisiones**

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Versión** | **Descripción** | **Autor** |
| 09/04/2019 | 1.0 | Creación del informe de progreso 1 | Rafa |
| 10/04/2019 | 1.1 | Añadidos apartados sin Diagramas | Rafa |
| 13/04/2019 | 1.2 | Diagrama de clases añadido | Rafa |
| 14/04/2019 | 1.3 | Esquema de la BD y Microservicios añadidos | Rafa |
|  |  |  |  |

**2 Objetivos del proyecto (**ordenados de mayor a menor prioridad**)**

Tras revisar la nueva planificación se ha decidido reordenar los objetivos del proyecto respecto al informe inicial. En el apartado de Planificación se detallan los motivos de este cambio en la priorización de los objetivos.

El nuevo orden es el siguiente:

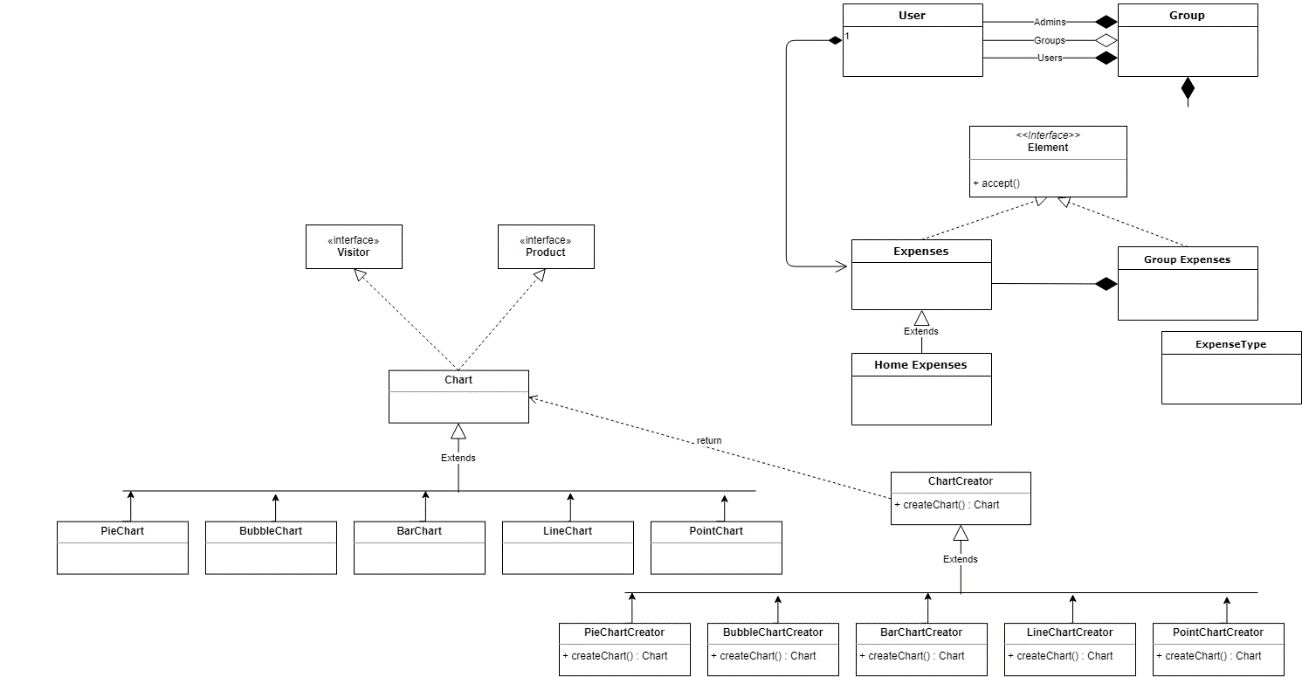
* **tfg-obj-01**: Poner en práctica los conocimientos de desarrollo del software adquiridos durante la mención.
* **tfg-obj-02**: Profundizar mis conocimientos en cada fase del desarrollo (Requisitos/Diseño/Implementación/Testing/Despliegue).
* **tfg-obj-03**: Aprender herramientas estandarizadas en el sector del desarrollo del software
* **tfg-obj-04**: Utilizar estándares de Clean Code y Design Patterns durante todo el ciclo de vida del proyecto.
* **tfg-obj-05**: Crear una aplicación que permita gestionar las finanzas de manera fácil y eficaz
  + **tfg-obj-05.1**: Poder gestionar las finanzas de diferentes grupos de amigos con la aplicación.
  + **tfg-obj-05.2**: Poder gestionar de forma fácil mis finanzas domésticas a través de la aplicación.
* **tfg-obj-06**: Realizar una arquitectura basada en microservicios y clustering en el cloud.
* **tfg-obj-07**: Crear una app multiplataforma (iOS y Android).

**3-Diseño y Arquitectura del proyecto**

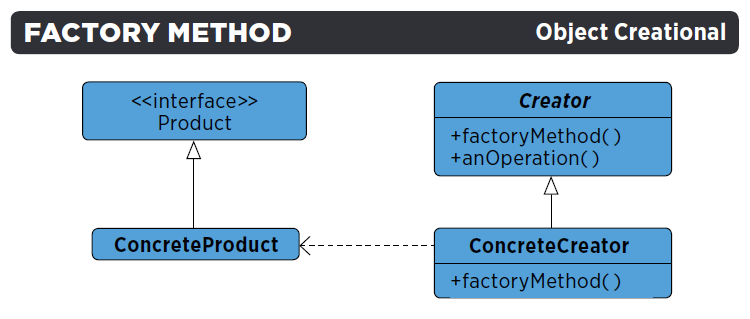
En el apartado de diseño y arquitectura cubriremos las diferentes fases del proceso de diseño que se han realizado durante el período de tiempo entre este informe y el anterior.

**3.1 Diagrama de Clases**

Respecto al diseño de las clases del proyecto, se ha decidido realizar un diagrama de clases (UML). Acorde con lo especificado en la rúbrica y, aplicando los conocimientos sobre Patrones de Diseño que se nos mostraron en asignaturas como Diseño del Software (DS), esta es la conclusión a la se ha llegado.

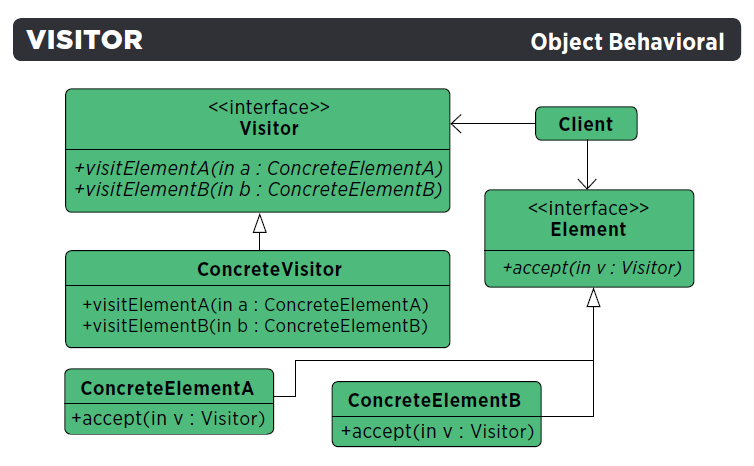


Como podemos observar en el diseño de clases, se han implantado dos patrones de diseño en el proyecto. El primero es [un Factory Method [1]](#FactoryMethod), un patrón de diseño creacional que, entre otros de sus beneficios, nos permite obtener un sistema de creación de objetos (en nuestro caso diferente tipos de diagramas) donde las subclases son las que deciden que objeto ha de ser creado.



El segundo es un [Visitor [4]](#Visitor), un patrón de diseño de comportamiento que, entre otros de sus beneficios, nos permite decidir las operaciones que hemos de realizar sin modificar la estructura de los objetos con los que trabajamos.

En nuestro caso, nuestra aplicación tiene dos tipos de gastos, gastos domésticos y gastos grupales. Debido a esto y a que en ambos casos hemos de generar diferentes tipos de gráficos, hemos decidido implementar un visitor que nos permite hacer esto de una forma cómoda y ampliable.



Aunque no se muestran en el diagrama de clases, al utilizar el framework de desarrollo Spring (y algunos de sus módulos) estaremos trabajando con otros patrones de diseño que vienen ya implícitos como [Command [3]](#Command) y [Facade [4]](#Facade) entre otros.

**3.2 Esquema de la Base de Datos (MongoDB)**

Como en todas las aplicaciones, la información ha de ser guardada en alguna Base de Datos (BD). En nuestro caso, estuvimos analizando entre utilizar una base de datos relacional como MySQL (basada en relaciones entre tablas) o una base de datos NoSQL como [MongoDB [5]](#MongoDB) (basada en documentos).

Los puntos fuertes de las bases de datos relacionales o SQL son los siguientes:

* Uso mas estandarizado por lo que es más accesible información sobre ellas.
* Atomicidad en las operaciones en la base de datos.
* Los datos han de cumplir requisitos de integridad tanto en tipo como en compatibilidad.

Los puntos flojos de las bases de datos relaciones son los siguientes:

* Garantizar la atomicidad de las operaciones afecta al rendimiento.
* Escalabilidad, aunque probada, es inferior a las bases de datos NoSQL.

Los puntos fuertes de las bases de datos NoSQL son los siguientes:

* Carácter descentralizado, son muy efectivas en estructuras distribuidas.
* Permiten adaptarse a necesidades de proyectos mucho más fácilmente que los modelos relacionales.
* Se pueden hacer cambios en el esquema sin necesidad de parar la base de datos.
* Escalabilidad horizontal, son capaces de crecer en cantidad de máquinas, en lugar de tener que utilizar máquinas con características muy altas.
* Se pueden ejecutar en máquinas con pocos recursos
* Consultas optimizadas para trabajar con grandes volúmenes de datos.

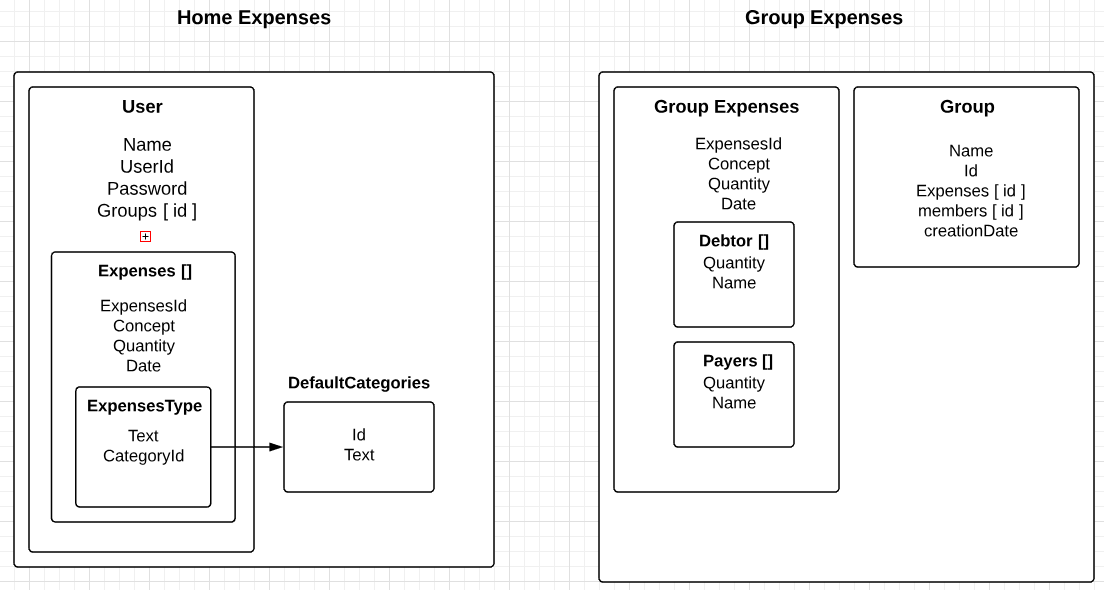
Los puntos flojos de las bases de datos NoSQL son los siguientes:

* No todas las bases de datos NoSQL contemplan atomicidad ni la integridad de datos (consistencia eventual).
* Problemas de compatibilidad con bases de datos SQL.
* Soporte multiplataforma algo pobre.

Teniendo en cuenta todos los factores comentados previamente y, dado que vamos a implementar un sistema distribuido, hemos optado por utilizar MongoDB como base de datos para nuestra aplicación y, en especial, [MongoDB Atlas [6]](#MongoDBAtlas) como nuestro [Database as a Service (DbaaS) [7].](#Dbaas)

Dado el funcionamiento de [MongoDB [5](#MongoDB)] basado en documentos y colecciones, junto con las dos principales funcionalidades del proyecto (Finanzas Domésticas y Grupales), se ha decidido

utilizar el siguiente esquema.



**3.3 Arquitectura basada en Microservicios**

Una de las corrientes más comunes en el desarrollo del software en la actualidad es el uso de servicios que proveen al usuario o cliente de lo que necesita. Incluso la mayoría de frameworks más establecidos en el sector, ya sea en backend o frontend (Spring, Angular, Flask, Django), los usan como un pilar en su forma de trabajar.

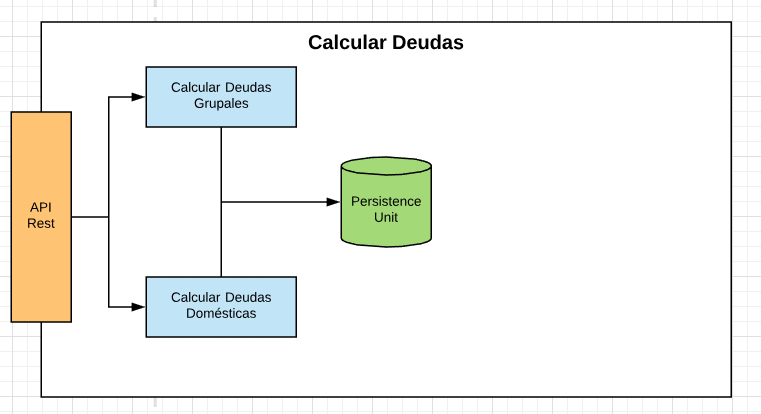
Debido a la necesidad de eliminar las dependencias que se generan a la hora de desplegar y hostear aplicaciones que contienen dichos servicios, nacen los microservicios.

Los microservicios, son una nueva forma de entender y crear arquitectura de software de modo que cada servicio se despliega de forma totalmente independiente del resto (normalmente utilizando los contenedores de Docker).

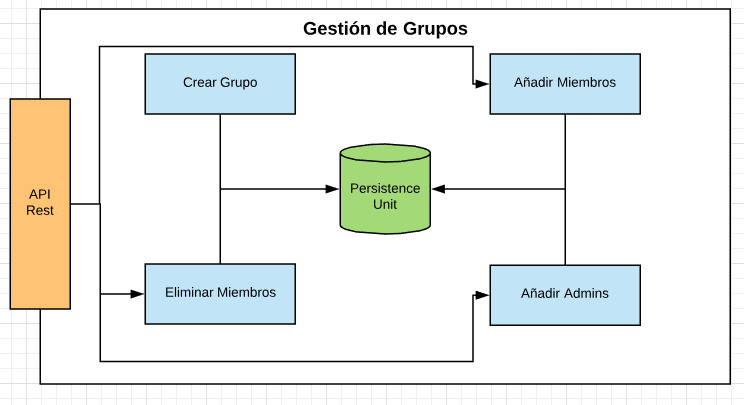
Entre las ventajas de los sistemas distribuidos basados en microservicios, se encuentran la escalabilidad, la resistencia al error (resilience) e incluso la capacidad de recuperarse de un error que en una aplicación monolítica resultaría fatal (auto-healing). Gracias a todas esas ventajas previamente mencionadas, los microservicios son utilizados normalmente cuando se trabaja con grandes volúmenes de datos o se dispone de una alta disponibilidad.

Aunque no prevemos que nuestra aplicación tenga la necesidad de una alta disponibilidad ni que vaya a tratar con un gran número de peticiones, dado la popular e interesante que nos resulta esta tendencia de arquitectura hemos decidido implementarla en nuestro proyecto.

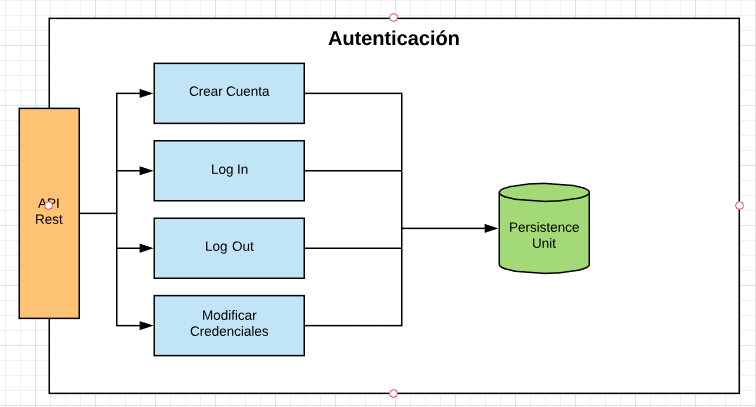
En el proyecto hemos diseñado una estructura basada en microservicios (diagrama completo en el anexo) que consiste en cuatro microservicios, los cuales contienen todas las funcionalidades que se implementarán hasta la fecha. Todos los microservicios constarán de una Persistence Unit (BD) para cada uno.

****

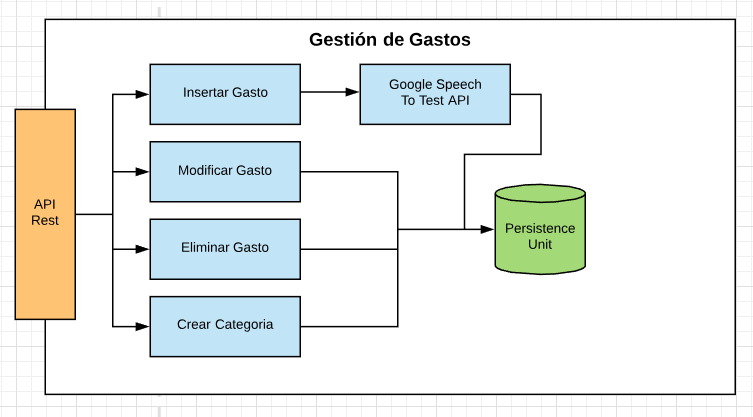
El microservicio de Cálculo de deudas será el encargado de obtener los valores de deudas, tanto para generar los diferentes gráficos de la aplicación como para hacer recomendaciones de quien ha de pagar la siguiente vez.



El microservicio de Gestión de grupos nos permitirá realizar todas las operaciones y/o cambios que el usuario necesite a los grupos (siempre y cuando tenga permisos para ello).



El microservicio de Autenticación permitirá al usuario realizar todas las acciones relacionadas con acceso a nuestros servicios (Crear cuenta, Log in etc).



El microservicio de Gestión de gastos nos permitirá, realizar operaciones con los gastos (como podemos ver en el diagrama). El caso de insertar un gasto es algo más complejo pues, al no tener acceso a una api de un banco, utilizaremos la API de Google Speech To Text para que el usuario pueda indicar con una grabación

**4-Metodología de trabajo**

Como se comentó en el informe anterior, metodología utilizada en este proyecto es una combinación de [**Kanban [8]**](#ReferenciaKanban) y [**feature-oriented-programming [9]**](#ReferenciaFOP) (FOP).

Hasta el momento la parte de Kanban ha sido la mas utilizada entre ambas, debido a que hasta este momento no se ha realizado ninguna implementación todavía. Por lo mencionado anteriormente, al no haber implementado nada aún no se ha podido comprobar la eficacia y/o utilidad de la metodología FOP.

**5-Planificación**

Después de terminar toda la fase de diseño del proyecto, tanto arquitectura de microservicios como diseño de clases, base de datos etc. Nos hemos dado cuenta que la planificación inicial que se había presentado en el anterior informe era demasiado optimista. Es decir, dada la cantidad de investigación a realizar sobre la mayoría de las tecnologías utilizadas en este proyecto previo a poder implementar cada funcionalidad, hemos visto que la cantidad de tiempo necesaria para desarrollar cada una de esas funcionalidades es (en principio) muy superior a lo que se tuvo en cuenta al realizar la primera implementación.

Teniendo en cuenta factores como tiempo disponible del equipo de desarrollo, la experiencia con estas tecnologías y los errores que aparecerán durante el proyecto se ha decidido volver a planificar el proyecto, reduciendo en principio, el número de funcionalidades a implementar.

En el [anexo](#Anexo) se puede encontrar el nuevo diagrama de Gantt acorde a la planificación mencionada.

**6-Listado de Acrónimos**

|  |  |
| --- | --- |
| Abreviatura | Significado |
| UML | Unified Modeling Language |
| BD | Base de Datos |
| SQL | Structured Query Language |
| FOP | Feature Oriented Programming |

**7-Referencias y Bibliografía**

**[1]** <https://es.wikipedia.org/wiki/Factory_Method_(patr%C3%B3n_de_dise%C3%B1o)>, Abril 2019

**[2]** <https://es.wikipedia.org/wiki/Visitor_(patr%C3%B3n_de_dise%C3%B1o)>,

Abril 2019

**[3]** <https://es.wikipedia.org/wiki/Command_(patr%C3%B3n_de_dise%C3%B1o)>,

Abril 2019

**[4]** <https://es.wikipedia.org/wiki/Facade_(patr%C3%B3n_de_dise%C3%B1o)>,

Abril 2019

**[5]** <https://docs.mongodb.com/?_ga=2.249737216.701975626.1555268920-415646804.1550786782>, Abril 2019

**[6]** <https://www.mongodb.com/cloud/atlas>, Abril 2019

**[7]** <https://www.baquia.com/emprendedores/2013-04-23-que-es-una-base-de-datos-cloud-dbaas>, Abril 2019

**[8]** <https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban/> Abril 2019

**[9]** <https://en.wikipedia.org/wiki/Feature-oriented_programming>, Abril 2019

**Anexo**

