Universidad Politécnica de Cataluña

Facultad de Informática de Barcelona

Ingeniería de software

Repositorio de árboles genealógicos en BD NoSQL

Autor:
Daniel Albarral Nuñez

Director:
Enric Mayol

Índice

1. Contextualización del proyecto

1.0.1. Los arboles genealógicos

Un arbole genealógico, también llamado genorama, es la representación gráfica de los antepasados y descendientes de un individuo.Para su representación se suelen usar tablas o arboles, siendo esta ultima la forma más común y la que se usara en el proyecto.

1.0.2. Uso y aplicación de los arboles genealógicos.

Los arboles genealógicos se usan como herramienta en la genealogía, que se encarga de estudiar y seguir la ascendencia y descendencia de una persona o familia. La genealogía es una ciencia auxiliar de la Historia y es trabajada por un genealogista. Uno de los objetivos del software a desarrollar es dar soporte a los genealogistas. Por otro lado hay varias comunidades de aficionados que llevan sus propios arboles genealógicos, el software creado también les podrá dar servicio a esta tipología de usuarios.

1.1. Perspectiva general del software actual.

Todo software genealógico, como mínimo permite almacenar la siguiente información de un individuo: fecha y lugar de nacimiento, fecha de casamiento, muerte y relaciones familiares, contra más flexible es el programa más información te permite introducir acerca de un individuo. También proporcionan diferentes maneras de representar la información y permiten exportar a GEDCOM la información representada.

GEDCOM [1] (**G**enealogical **D**ata **COM**munication): Es un formato de archivo de datos, proporciona un formato flexible y uniforme para el intercambio de datos genealógicos computarizados.

La mayor parte del software genealógico actual esta basado en soluciones de escritorio, pero en los últimos años han proliferado diferentes soluciones web como myheritage o familysearch, que no solo sirven como plataforma de edición sino que también son grandes plataformas *cloud* en las que se almacenan los arboles genealógicos.

Las soluciones más avanzadas, aparte de la gestión de arboles también ofrecen herramientas más orientadas a la investigación, como podrían ser sistemas de búsqueda de individuos basados en sus relaciones o herramientas estadísticas.

1.2. Tecnología

Dada la naturaleza social del tipo de software que se busca desarrollar, nacen ciertas complicaciones tecnológicas que en los últimos tiempos se han sido considerablemente investigadas, debido a la gran repercusión de las redes sociales. Las tecnologías clásicas orientadas a la persistencia de datos como las bases de datos SQL, plantean la dificultad de tener un coste muy alto de consulta cuando se pregunta acerca de datos con un alto nivel de relación entre ellos. Una de las soluciones más usadas para solucionar este problema son las bases de datos orientadas a grafos, uno de los casos de éxito es el caso de Twitter, que desarrollo su propia solución, FolckDB, una base de datos orientada a grafos, tolerante a fallos, diseñada para tratar con grandes conjuntos de datos, con información no critica.

En el libro **Neo4j in Action** [5], Partner and Vukotic realizan el siguiente experimento:
El experimento consiste en comparar una base de datos relacional con neo4j *GraphDB*, en ambas se representa el mismo modelo donde diferentes personas tienen múltiples amigos. Se realizan diferentes consultas donde se pretende averiguar si existe un camino entre dos personas escogidas al azar que los una en una profundidad de como mucho cinco relaciones. Las bases de datos constan de 1000000 personas, cada una con 50 amigos aproximadamente. Los resultados fueron los siguientes:

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Figura 1: Comparativa entre neo4j y base de datos relacional.

Con este experimento podemos ver como en las consultas donde la profundidad es mayor, o dicho de otra forma, donde la conectividad juega un papel más importante, la base de datos orientada a grafos es mucho más rápida. Esto es debido, en gran parte, a que las relaciones en las bases de datos orientadas a grafos son *ciudadanos de primer orden* a diferencia de las bases de datos relacionales que tratan las relaciones mediante claves foráneas, lo que provoca que tengan que usar una gran cantidad de *joins* para resolver el problema, elevando mucho el coste de resolución.

En el articulo Benchmarking Graph Databases de Alekh Jinda [2] realiza una comparación entre diferentes bases de datos con diferentes naturalezas.

Neo4j : Orientada a grafos.

MySQL : Orientada al tratamiento de filas.

Vertica : Orientada al tratamiento de columnas.

VoltDB : Base de datos en memoria principal.

Los conjuntos de datos usados son los siguientes:

- Conjunto de datos de Facebook, con 4k nodos y 88k aristas.
- Conjunto de datos de Twitter, con 81k nodos y 1.8M aristas.

Las querys aplicadas son *Pagerank* para analizar el comportamiento bajo la necesidad de un *fullscan* y *Dijkstra* para encontrar el camino más corto entre dos nodos. Los resultados los encontramos en la figura 2

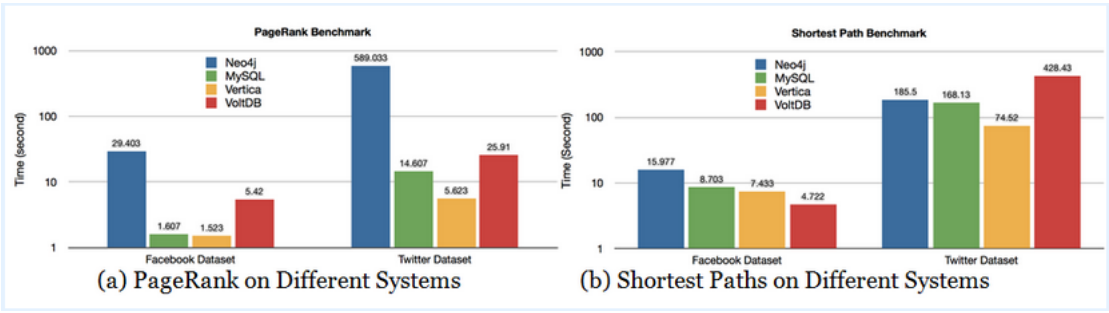


Figura 2: Comparativa entre neo4j y base de datos relacional.

Podemos ver que los resultados son contradictorios con el estudio anterior, estaría pendiente de un análisis futuro explicar el por que es estos resultados contradictorios. A pesar de esto neo4j sigue siendo una buena opción, ya que el lenguaje cypher puede facilitar la creación de consultas sobre el modelo a trabajar, dado que un árbol se expresa en forma de grafo de forma natural. También hay que tener en cuenta que este estudio solo examina la rapidez, no el uso de memoria y posiblemente VoldDB al estar orienta a el uso intensivo de memoria principal saldría muy perjudicada.

1.2.1. Bases de datos orientadas en grafos.

La gran cantidad de proyectos que han nacido en los últimos años hace prácticamente imposible hacer una comparativa concreta de todas las tecnologías existentes. Pero podemos diferenciar el panorama actual haciendo dos generalizaciones:

Tecnologías usadas para propósitos transaccionales .

Orientadas a dar un servicio *online* en tiempo real a una aplicación. Simpli-

fican la diseño del modelo de datos cuando este puede representarse de forma más intuitiva con un grafo.
Estas tecnologías son llamadas bases de datos orientadas en grafos. Son el equivalente a las OLTP en el modelo transaccional.

> OLTP (**O**n**L**ine **T**ransaction **P**rocessing):
> Es un tipo de procesamiento que facilita y administra aplicaciones transaccionales, usualmente para entrada de datos y recuperación y procesamiento de transacciones (gestor transaccional). Los paquetes de software para OLTP se basan en la arquitectura cliente-servidor ya que suelen ser utilizados por empresas con una red informática distribuida..

Tecnologías usadas principalmente para el análisis de grafos .

Llamados Motores de procesamiento de grafos, siguiendo el mismo símil que antes, podríamos pensarlos como herramientas de *data mining* y análisis de procesos(OLAP)

Las bases de datos orientadas en grafos son sistemas de bases de datos que permiten operaciones CRUD (*Create, Read, Update y Delete*) sobre los objetos representados en ellas. Suelen estar orientadas a funcionar con sistemas transaccionales (OLTP), como aviamos mencionado anteriormente. Sus principales propiedades son, las relaciones son *ciudadanos de primer orden*, a diferencia de otros modelos, como el relacional que tienen que usar claves foráneas. En este tipo de base de datos para representar el dominio de nuestro problema simplemente nos b asta con definir los nodos y las relaciones que lo componen.

En el libro Graph Databases [3], encontramos el siguiente gráfico (Figure 3), que nos da una idea de las principales tecnologías orientadas en grafos y a que enfocan su potencial.

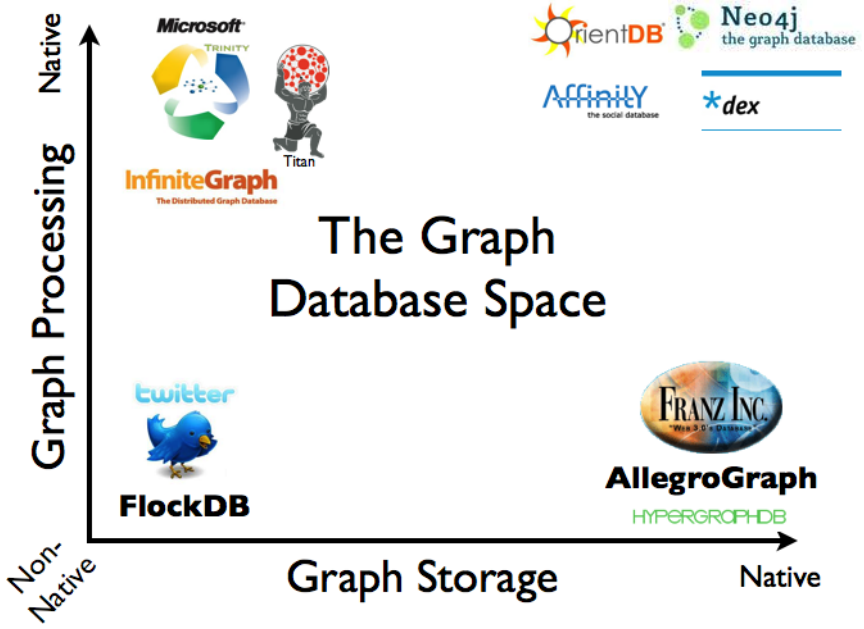


Figura 3: Visión del conjunto de tecnologías

2. Metodología y rigor

Las metodologías usadas son las siguientes:

2.1. SCRUM

Para el desarrollo del proyecto se ha escogido como metodología de trabajo SCRUM, los *sprints* dado la naturaleza del proyecto se han adaptado esta metodología ágil para ser usada por una sola persona. La metodología de trabajo se ha definido de la siguiente forma:

Product backlog :

Para crear el *product backlog* se han definido un seguido de historias de usuario, de acuerdo con el tutor del proyecto, que asume el rol de *product owner*. Estas historias de usuario se revisaran constantemente en los *sprint backlogs*.

Sprints :

Se desarrollaran a lo largo de dos semanas, en este tiempo se desarrollaran las historias de usuario escogidas.

Sprint backlog :

Se realizaran al final de cada *sprint*, dado que esta tarea estará limitada por la disponibilidad del tutor y el alumno, se dara flexibilidad pudiendo realizar el *sprint backlog* de varios *sprints* en una reunión. Durante estas reuniones se revisaran las historias de usuario, reorganizando su prioridad, modificando su contenido, añadiendo nuevas o eliminando historias que no se crean necesarias.

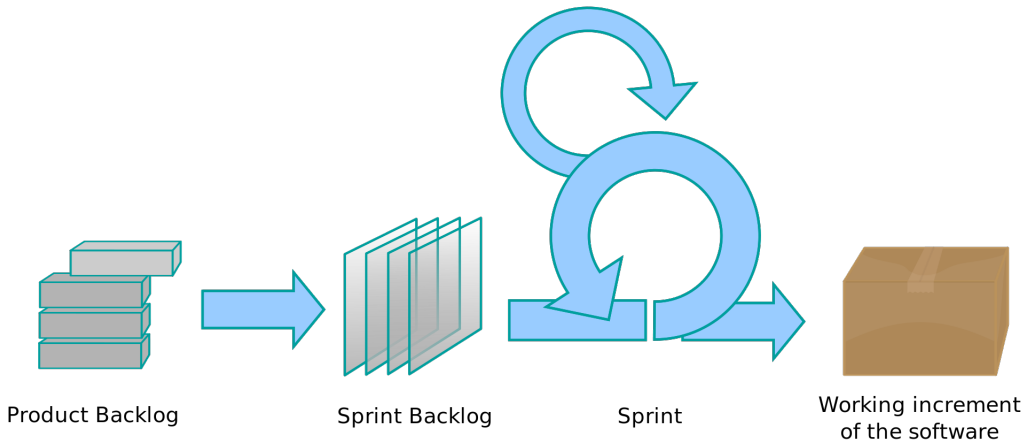


Figura 4: Metodología SCRUM

2.2. Desviaciones

Dado al carácter flexible de la metodología, esta no se ha cambiado. Por otro lado durante los *sprints backlogs* se han modificado considerablemente las historias de usuario, lo que ha repercutido en una re-orientación del proyecto, como se explicara en el análisis de alternativas.

3. Planificación

Para el desarrollo del proyecto se usara *Scrum*, por ello la planificación temporal estará estructurada para facilitar la consecución de esta metodología. El final del proyecto se ha establecido el 22 de abril del 2016.

3.1. Gantt

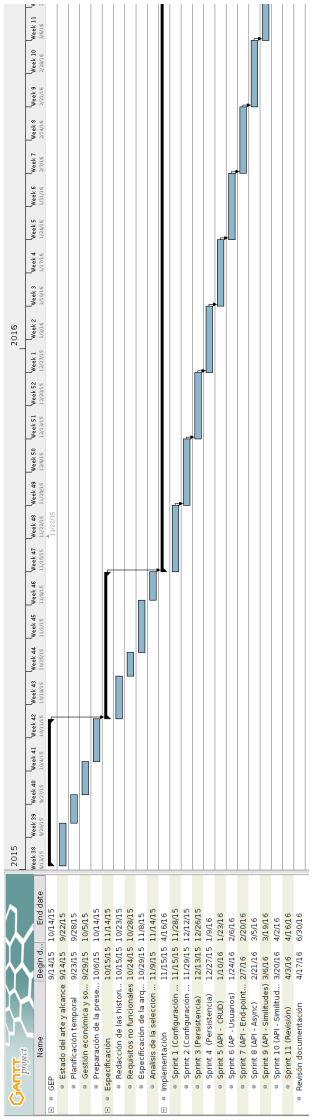
El gantt adjuntado a continuación tiene varias peculiaridades:

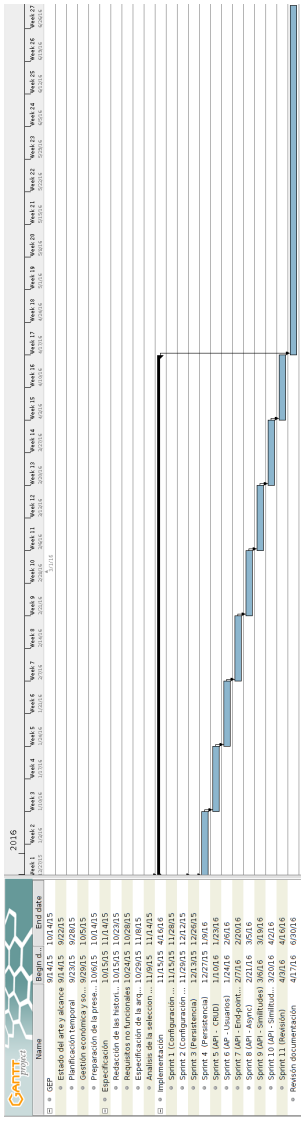
Dependencias de las tareas .

Dado que en el proyecto se desarrolla íntegramente por una sola persona, la secuencialidad del diagrama es absoluta.
Durante la fase de desarrollo simplemente se especifican los *Sprints* que se llevaran a cabo hasta conseguir la consecución del proyecto, a medida que las historias de usuario se vayan redactando y priorizando se irán asignando a los *sprints*.

3.1.1. Tareas y Gráficos gantt

Name	Begin date	End date
GEP	9/14/15	10/14/15
Estado del arte y alcance	9/14/15	9/22/15
Planificación temporal	9/23/15	9/28/15
Gestión económica y sostenibilidad	9/29/15	10/5/15
Preparación de la presentación final de GEP	10/6/15	10/14/15
Especificación	10/15/15	11/14/15
Redacción de las historias de usuario	10/15/15	10/23/15
Requisitos no funcionales	10/24/15	10/28/15
Especificación de la arquitectura	10/29/15	11/8/15
Analisis de la seleccion de tecnologias	11/9/15	11/14/15
Implementación	11/15/15	4/16/16
Sprint 1 (Configuración del entorno de pre y pro)	11/15/15	11/28/15
Sprint 2 (Configuración del entorno de pre y pro)	11/29/15	12/12/15
Sprint 3 (Persistencia)	12/13/15	12/26/15
Sprint 4 (Persistencia)	12/27/15	1/9/16
Sprint 5 (API - CRUD)	1/10/16	1/23/16
Sprint 6 (AP - Usuarios)	1/24/16	2/6/16
Sprint 7 (API - End-points ext)	2/7/16	2/20/16
Sprint 8 (API - Async)	2/21/16	3/5/16
Sprint 9 (API - Similitudes)	3/6/16	3/19/16
Sprint 10 (API - Similitudes)	3/20/16	4/2/16
Sprint 11 (Revisión)	4/3/16	4/16/16
Revisón documentación	4/17/16	6/30/16


GANTT project
2015
2016
GEP
Estado del arte y alcance
Planificación temporal
Gestión económica y so...
Preparación de la prese...
Especificación
Redacción de las histori...
Requisitos no funcionales
Especificación de la arq...
Análisis de la selección ...
Implementación
Sprint 1 (Configuración ...
Sprint 2 (Configuración ...
Sprint 3 (Persistencia)
Sprint 4 (Persistencia)
Sprint 5 (API - CRUD)
Sprint 6 (AP - Usuarios)
Sprint 7 (API - End-point...
Sprint 8 (API - Async)
Sprint 9 (API - Similitudes)
Sprint 10 (API - Similitud...
Sprint 11 (Revisión)
Revisión documentación


GANTT project
2016
GEP
Estado del arte y alcance
Planificación temporal
Gestión económica y so...
Preparación de la prese...
Especificación
Redacción de las histori...
Requisitos no funcionales
Especificación de la arq...
Análisis de la selección ...
Implementación
Sprint 1 (Configuración ...
Sprint 2 (Configuración ...
Sprint 3 (Persistencia)
Sprint 4 (Persistencia)
Sprint 5 (API - CRUD)
Sprint 6 (AP - Usuarios)
Sprint 7 (API - End-point...
Sprint 8 (API - Async)
Sprint 9 (API - Similitudes)
Sprint 10 (API - Similitud...
Sprint 11 (Revisión)
Revisión documentación

3.2. Recursos

Las necesidades en recursos serán muy escuetas dado que todo el software que se usara es libre y las maquinas en las que se trabaja en pre-producción serán virtuales. La única necesidad de tener unos recursos que se necesiten planificar, ya que requieren ser contratados, son los servidores en los que se hará el *deploy*, se tendrán que contratar durante las semanas que dures los *sprints* en los que se haga el *deploy*. Por otro lado en cuanto a recursos humanos el proyecto se llevara a cabo por un solo desarrollador que se encargar tanto de la programación como la documentación.

3.3. Alteativas y plan de acción

Dado que el proyecto se desarrollara usando *Scrum*, al final de cada *sprint*, durante el *sprint backlog* se determinara si se han cumplido las expectativas del *sprint*. Por otro lado en el *sprint planning* se tendrá en cuenta si han habido carencias o *bugs* en los anteriores *sprints* para incorporarlos como historias de usuario al siguiente *sprint*.

Como se ha comentado en el alcance, se diseñara un software abierto orientado al desarrollo continuado. Por lo que en el *sprint* 13 se dará por concluido el proyecto, y se documentara el estado en el que se encuentre y las historias de usuario pendientes.

3.4. Desviaciones

Del planteamento inicial del proyecto se ha decidido prescindir del desarrollo del *frontend* y centrar se en la parte del *backend*. Se ha creado *endpoints* asíncronos par las funcionalidades más pesadas y se ha añadido un endpoint para subir archivos GEDCOM.

3.5. Estado actual de ejecución

En el punto actual de ejecución del proyecto se da por concluida la implementación de todas las funcionalidades.

4. Análisis de alternativas

4.1. Tecnologias

Se ha establecido como condición necesaria para todas las tecnologías que sean *opensource* o *freeware*.

4.1.1. Base de datos

Se han valorado las siguientes bases de datos:

HyperGraphDB (`http://hypergraphdb.org/`) :
: Es un base de datos orientada a grafos de propósito general, extensible, potable, distribuida y integrable. Diseñada especialmente para proyectos sobre inteligencia artificial y web semántica. También se pude usar como base de datos orientada a objetos.

InfoGrid (`http://infogrid.org`) :
: Es una base de datos orientada a grafos especialmente enfocada al desarrollo de aplicaciones *backend* REST-ful. Hace especial énfasis en la orientación a objetos. De este proyecto destaca su modularidad.

Neo4j (`http://neo4j.com/`) :

Esta base de datos basada en grafos esta diseñada como una base de datos de propósito general, cumple los requisitos de las dos anteriores a pesar que no se pueden crear hyperedges como tal a diferencia de HyperGraphDB, por otro lado se pueden usar técnicas en la representación de los grafos que los simulan. También aporta mejoras en el almacenamiento de los datos respecto a InfoGrid, ya que el almacenamiento esta implementado en grafos de forma nativa.

Decisón:
Final mente se ha seleccionado Neo4j. Se ha descartado el uso de HyperGraphDB, dado que pudiéndose acomodar a las necesidades del proyecto no es tan eficiente como InfoGrid y Neo4j cuando usamos la base de datos como una base de datos orientada a objetos puramente. InfoGrid se ha descartado al no ser tan eficiente como neo4j en el tratamiento de datos en una lógica de grafos, esto se debe a que en InfoGrid los datos no se guardan nativa-mente en estructuras de grafo [4].

4.1.2. Lenguaje de programación

Los lenguajes escogidos para el análisis han estado seleccionados con criterios subjetivos, seleccionándolos de tal forma que aporten alternativas a la hora tanto de dar agilidad a la implementación como a ser eficientes en el rendimiento. También se ha tendió en cuenta los *frameworks* que se pueden usar en estos lenguajes para la consecución de una API rest.
Para desarrollar este proyecto se ha optado entre los siguientes lenguajes de programación.

Scala (`www.scala-lang.org/`) :

Scala es un lenguaje de propósito general multi-paradigma, da soporte completo para programación funcional y orientada a objetos, también dispone de un sistema de inferencia de tipo estático muy potente. Se compila a Java byte code y se ejecuta sobre la maquina virtual de Java. Es totalmente interoperable con Java por lo que se pueden usar librerías Java y viceversa.

Frameworks:

Scarlata(`www.scalatra.org/`) :

Es un *Micro-framework* para scala, consta de librerías para la creación de una API rest completa, como *micro-framework* no fuerza ningún tipo de arquitectura. Se usa para desarrollar bajo la siguiente filosofa:

Empezar desde lo pequeño, desarrolla hacia arriba :

Empieza construyendo un pequeño nucleo y integra módulos simples para solucionar las tareas.

Libertad :

Permite al programador usar la estructura y las librerías que crea convenientes para su aplicación.

Solido pero no terco :

Usa componentes sólidos para la creación de la API, por ejemplo, los serverlets no son una solución muy moderna a la gestión de la comunicación cliente servidor, pero tienen una gran comunidad detrás y son extremadamente estables. Por otro lado se anima a usar la flexibilidad del framework para usar nuevas técnicas y aproximaciones en diferentes ámbitos del servicio.

Ama HTTP :

HTTP es un protocolo de naturaleza carente de estado, no se ha de pretender usar mecanismos complejos que den la apariencia de estado.

Lift(`liftweb.net/`) :

Es un *framework* para scala, esta diseñado para seguir la arquitectura 'View First". Clama ser uno de los *frameworks* más seguros y destaca por los siguientes puntos:

Seguridad :

Lift es resistente a casi todas las vulnerabilidades comunes incluyendo la mayoría de la lista "OWASP *top 10* de".

Amigable para el desarrollador :

Lisft esta diseñado para facilitar la creación de aplicaciones de forma rápida y sencilla.

Escalable :
Esta diseñado para dar soporte a grandes cantidades de trafico.

Modular :
El programa se compone de módulos, esto da mucha agilidad a la hora ampliar las funcionalidades de los proyectos y da una arquitectura muy limpia y fácil de mantener.

Python(`python.org/`) :
Python es un lenguaje de propósito general, interpretad, y de tipado dinámico. Es un lenguaje multi-paradigma, combina la orientación a objetos con la programación funcional. A diferencia de escala la parte funcional no esta totalmente desarrollada ya que no permitite valores inmutables o técnicas como curring. Python esta diseñado para agilizar el proceso de desarrollo y facilitar la legibilidad y comprensión del código.

Frameworks:

Falsk(`flask.pocoo.org/`) :
Es un *micro-framework*, esta diseñado permitir el desarrollo de aplicaciones web de forma flexible. Consta de los siguientes herramientas:

- Incorpora servidor y debugger
- Integra soporte para unit test
- Gestor de peticiones RESTful
- Usa Jinja2 templating
- Soporte para "Cookies" con seguridad
- Cumple al 100 % el WSGI 1.0 (Estándar para la comunicación de las aplicaciones con el servidor)
- Basado en unicode
- Extensamente documentado

Django(`www.djangoproject.com/`) :
Es un *framework*, esta diseñado permitir el desarrollo de aplicaciones rápida-

mente y con una estructura limpia. Consta de diferentes capas que aportan seguridad la aplicación. Esta basado en la arquitectura modelo vista controlador. Entre sus virtudes destacan la gran facilidad para modular las aplicaciones y facilitar su integración y una comunidad muy extensa, también dispone de una documentación clara, bien estructurada y muy completa. Django destaca en su web estos tres puntos:

- Ridicualemtne rapido de desarrollar
- Altamente seguro
- Escalabilidad asombrosa

Decisión: Finalmente se ha escojido Django, ya que es el framework que permite más rapidez en el desarrollo y consta de herramientas que se adhieren altamente a las necesidades del proyecto. A destacar:

Django Rest Framework :

Framework que se integra con Django y proporciona herramientas para facilitar la seralización, enrutamientó teniendo en cuenta el protocolo REST-ful, autorización y autenticación, creación de vistas para las operaciones CRUD, etc.

Django OAuth Toolkit :

Conjunto de herramientas para crear autenticación y autorización con permisos, basado en el protocolo oAuth2.

Celerys :

Librería de python que permite combinado con una tecnología *"message broker"* permite tarear funcionalidades de la aplicación o crear tareas asíncronas.

neomodel OGM(Object graph maper) :

Permite crear modelos en la base de datos neo4j basándose en los modelos de la aplicación.

5. Integración de conocimientos

En el proyecto se integran 3 grandes bloques:

5.1. Base de datos orientada a grafos

Se ha diseñado el modelo y de una DB orientada a grafos que represente un árbol genealógico. Para la cración del modelo se han usado diferentes técnicas, como los arboles temporales [3]. Para las localizaciones se ha creado un sistema que almacena direcciones in-equivocas forma única en forma de árbol, esto se ha conseguido por medio de la API de google maps haciendo uso de los geocomponentes. De esta forma tanto las fechas como las direcciones pueden funciónar como un *"hyperedge"* ya que conseguimos relacionar los diferentes subgrafos por medio de estos nodos.

5.2. Backend

Creación de un *backend* rest-ful con orientación social, para gestionar la base de datos orientada a grafos. A la hora de construir este *backend* se ha tenido en cuenta patrones arquitectónicos que faciliten la mantenibilidad y ampliabilidad de la aplicación. Este *backend*constara principalmente de:

Gestión de usuarios

Para la gestión de usuarios no se creara ningún *endpoint* en la API por cuestiones de seguridad, se proporcionara una interfaz para poder gestionar los usuarios desde un *frontend* . Los usuarios para darse de alta tendrán que confirmar su correo electrónico.

Autorización y autenticación

Los usuarios, para poder realizar peticiones a los *endpoints* de la API, tendrán que estar autenticados y autorizados, este proceso se hará mediante el protocolo

oAuth2, concretamente con *bearer token*.

Gestión asincrona

En los *end-points* más pesados la peticiónes seran asincronias. Esto se conseguira con Rabit MQ como *message broker* y celerys, una librería python que nos permite enviar menssages a la cola de Rabit MQ y gestionar los *workers* que trabajaran sobre ella.

Endpoints

El *backend* consta de todos los *endpoints* necesarios para realizar todas las operaciónes crud sobre el modelo de forma rest-ful. Aparte se incluirán otros *endpoints* para realizar operaciones adicionales.

5.3. Busqueda de personas similares

El programa permitirá encontrar coincidencias entre las personas de la DB. Para ello se aprovechara el potencial de la base de datos orientada a grafos. Para ello se buscaran personas que compartan fechas y lugares en diferentes eventos, nacimiento, muerte, etc. Una vez encontrados estos candidatos se aplicaran heuristicos para determinar en que medida esas dos personas pueden ser la misma.

6. Integración de leyes y regulaciones

Al desarrollar una aplicación *backend* donde se almacenaran datos de usuarios y personas se tendrá que respetar la **Ley Orgánica 15/1999 de Protección de Datos de Carácter Personal (Lopd)**. Por ello se tendrá que tener en cuenta:

6.1. Derecho de información

El derecho de información esta regulado en la **Ley Org ánica 15/1999 de Protección de Datos de Carácter Personal (Lopd)**. Regula las condiciones en que se ha de recoger, tratar y ceder los datos personales, su fin es proteger la intimidad y demás derechos ciudadanos. Para adecuarse a esta ley, si un usuario introduce datos personales de otra persona se tendrá que se notificar de la posesión de estos datos al afectado y pedir su aprobación para su almacenamiento.

6.2. Consentimiento del afectado

EL articulo 6 de la Lopd, regula que para tratar datos personales se ha de tener el consentimiento expreso del damnificado, por ello al registrarse los usuarios tendrán que aceptar una licencia de acuerdo con el usuario final (EULA). Por el momento al ser una plataforma en fase beta donde no se registraran los usuarios no se redactara dicho acuerdo ni se realizara ninguna confirmación a la hora del registro.

Referencias

[1] Ryan Heaton. About gedcom, 2014.

[2] Alekh Jindal. Benchmarking graph databases, 2013.

[3] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly Media, Inc., 2013.

[4] Richard Stallman. Java trap, 2004.

[5] Aleksa Vukotic, Nicki Watt, Tareq Abedrabbo, Dominic Fox, and Jonas Partner. *Neo4j in action*. Manning, 2015.