Te lo traigo de mi Pueblo

Grado de Ingeniería Informática

Jorge Aguilera

Version v38-23-gdb0740e, 2017-01-01

Table of Contents

***TelotraigodemiPueblo corresponde al Trabajo Fin de Grado de** Jorge Aguilera.*

> *A Miriam y Dani, que siempre estuvieron ahí*

> *"No existe una mejor prueba del progreso de la civilización que la del progreso de la cooperación."*
>
> **— Oscar Wilde, 1854-1900**

Título del trabajo	Te lo traigo de mi Pueblo
Nombre del autor	Jorge Aguilera
Nombre del consultor	Albert Grau Perisé <agrau@uoc.edu> Santi Caballe Llobet <scaballe@uoc.edu>
Fecha de entrega	01/2017
Area del Trabajo Final	Aplicaciones J2EE
Titulación	Grado de Ingenieria Informática

Resumen del trabajo

El proyecto consiste en un prototipo de una red social en la cual cada usuario puede compartir los productos típicos de cada pueblo que visita de manera frecuente. Esta información estará disponible para la red de sus amigos y estos podrán realizarle encargos. En el desarrollo se ha aplicado una metodología de Entrega Contínua y herramientas de Código Abierto.

Abstract

The project consists of a prototype of a social network in which each user can share the typical products of every town that visits frequently. This information will be available to his network and they will be able to order products. In this development a Continuous Delivery methodology, plus Open Source tools, has been applied.

Palabras clave (entre 4 y 8):

Red social, Grafos, Git, Grails, Asciidoctor, Entrega Contínua

1. Introducción

1.1. Contexto y justificación del Trabajo

TelotraigodemiPueblo es una red social donde conectar a personas que viajan frecuentemente a sus pueblos con sus amigos para permitirles realizar encargos de productos típicos de su tierra.

De esta forma TelotraigodemiPueblo intenta ayudar en diferentes áreas: mejorar la relación entre amigos y compañeros al servir como recordatorio a ambas partes de los pedidos que se hicieron, aprovechar el mayor volumen de compra para poder conseguir precios ventajosos y por último servir como ayuda para potenciar el consumo de productos autóctonos y más auténticos.

1.2. Objetivos del Trabajo

El trabajo busca cumplir principalmente los siguientes objetivos:

- desde una perspectiva técnica, desarrollar un producto totalmente funcional que sirva como punto de partida para una aplicación comercial.
- desde una perspectiva académica, demostrar las bondades de una metodología de Entrega Contínua en un Trabajo Fin de Grado (y por ende en el resto de asignaturas de una carrera)

1.3. Enfoque y método

Desde el primer momento TelotraigodemiPueblo ha sido un proyecto de código abierto y de cara al público, de tal forma que se puede consultar desde la primera aportación que se realizó y ver la evolución que ha ido teniendo.

Así mismo se ha intentado aplicar una metodología de Entrega Contínua de tal forma que toda aportación al proyecto, ya fuera en forma de código o de documentación, fuera puesta en producción en el menor tiempo posible.

Considero que este enfoque es muy provechoso en un TFG porque, por un lado, mantiene el ritmo de trabajo constante y por otro nos acostumbra a un entorno parecido al laboral orientado a entregas con valor cada poco tiempo.

1.4. Planificación

(Para una mayor comprensión del calendario se recomienda revisar el apartado de épicas donde se describen en detalle cada una de ellas y sus dependencias)

El desarrollo del producto debe adaptarse al calendario de entregas propio del TFG tal que en las fechas propuestas por el mismo se hayan alcanzado los hitos propuestos.

Hito	Fecha	Descripcion
PAC1	5/10/2016	Definición del proyecto.
PAC2	9/11/2016	Requerimientos funcionales.
PAC3	23/12/2016	Implementación.
PAC4	12/01/2017	Memoria final.

Para ello, y teniendo en cuenta que deseamos utilizar una metodología ágil, debemos establecer un calendario de sprints que permita cumplir con los hitos previos. Se establece como primera aproximación la siguiente planificación:

Sprint	Fecha Inicio	Descripcion
1	3/10/2016	Aprovisionamiento de infraestructura y depliegue contínuo.
2	10/10/2016	Diseño modelo de Negocio mediante tests.
3	17/10/2016	HU1 HU2.
4	31/10/2016	HU6 HU7 HU8 HU9.
5	14/11/2016	HU3 HU4 HU5.
6	28/11/2016	HU10 HU11 HU12.
7	12/12/2016	HU13 HU14 HU15 HU16.
8	2/1/2017	Memoria y Presentación.

Cada sprint consta de un período de 15 días en el cual se revisará el trabajo realizado en el anterior sprint para evaluar si la estimación ha sido correcta, tanto en exceso como en defecto, para poder tomar las medidas necesarias. Así mismo se realizará una revisión del trabajo a realizar.

1.5. Productos obtenidos

- Aplicación J2EE (war), desplegado y accesible desde Internet
- Documentación del API accesible desde Internet

1.6. Estructura del documento

En los capítulos sucesivos de este documento se describe

- el origen de la idea
- las historias de usuario mínimas a desarrollar para disponer de un MVP
- aspectos técnicos como elección de la base de datos, diseño, etc.

2. Incepción

"Te lo traigo de mi pueblo" pretende ser el aplicativo donde personas con relaciones de confianza puedan indicar a su grupo que van a ir de visita a un lugar típico y pueden traer productos de ese lugar.

La aplicación permitirá a cada usuario gestionar un número indeterminado de lugares y productos locales del mismo, de tal forma que cuando publique en su red que va a ir a ese lugar en una fecha determinada, los miembros de la misma puedan realizarle pedidos. El usuario podrá decidir qué pedidos puede comprometerse a traer y de esa forma optimizar su viaje.

El viajante queda bien con sus amistades al poder cumplir con los encargos, las amistades contentas por disponer de productos locales y los vendedores locales por el incremento de sus ventas.

La red de amistades tendrá una forma típica de red social donde un usuario puede estar enlazado con un número indeterminado de usuarios. Para realizar este enlace un usuario enviará a otro una solicitud de enlace que el segundo usuario podrá aceptar o ignorar. A partir del momento en que se encuentren enlazados un usuario podrá ver los viajes futuros que publique el otro.

Para este Minimum Value Product (MVP) se busca la sencillez por lo que las pantallas y navegación estarán enfocadas hacia la funcionalidad del sistema y no a su presentación estética.

La evolución lógica del aplicativo, que queda fuera del MVP, sería buscar la monetirización de la misma, incluyendo una pasarela de pago, publicidad de los vendedores locales, etc.

3. Objetivo

Como podemos ver por la incepción de la idea, el producto a conseguir tiene por objetivo obtener un producto mínimo viable (MVP), en un tiempo reducido, que sirva como presentación ante un posible grupo de inversores para seguir evolucionándolo. Para la consecución de este MVP se deberá implementar una serie de funcionalidades imprescindibles, de primer nivel, así como otras básicas, de segundo nivel, que deberán ser definidas al inicio del proyecto.

Así mismo presenta un alto componente tecnológico (gestionar redes de contactos, indeterminado número de lugares y productos, etc) lo que implica definir una arquitectura y herramientas que permitan un desarrollo rápido pero fiable.

4. Épicas

Entendemos por épica, dentro del marco de desarrollo ágil, aquellas historias de usuario demasiado genéricas y que nos sirven para delimitar el producto en áreas. De la división de estas en elementos más pequeños obtenemos las historias de usuario y de estos a su vez los items a desarrollar.

Así para el caso de TelotraigodemiPueblo encontramos las siguientes épicas y para cada una de ellas las historias de usuario.

4.1. Red Social (EP1)

Como red social que es, TelotraigodemiPueblo permitirá registrar a usuarios nuevos, los cuales deberán proporcionar su email como identificador así como establecer una contraseña para validar el acceso. Como esperamos un alto grado de aceptación del aplicativo, no nos centraremos en el borrado del usuario hasta que el departamento de asuntos legales nos confirme su necesidad. Mediante formularios de búsqueda, el usuario encontrará personas que no están en su red y a las que podrá enviar una invitación de contacto.

- Registro de usuarios (HU1)

Como persona ajena al sistema deseo registrarme en el mismo introduciendo mi email. Tras la comprobación de aquél podré añadir información en lo que será mi perfil como mi password, nombre, edad, etc.

- Búsqueda de gente (HU2)

Como usuario identificado en cualquier momento puedo realizar búsquedas en la red global para buscar conocidos. Para ello introduciré un nombre y el sistema me buscará los que concuerden con el mismo.

- Búsqueda de lugares (HU3)

Como usuario identificado podré buscar por lugares lo que me ofrecerá un listado de personas que tienen el nombre del lugar introducido entre sus sitios de viaje.

- Solicitud de enlace (HU4)

En ambos casos, una vez obtenido el listado de usuarios podré enviar una solicitud de enlace en el caso de que no se encuentre en mi red de contactos. Si ya se encuentra el sistema no hará nada.

- Aceptación de solicitud (HU5)

Como usuario registrado quiero ver las solicitudes de contacto que tengo pendientes y si decido aceptarla se creará un enlace entre el demandante y yo.

4.2. Lugares y productos (EP2)

Como usuario "telotraigo" quiero poder gestionar un número indeterminado de lugares que visito, como puede ser el pueblo de mi padre, de mi madre, novia, etc. Básicamente quiero poder decirle a mi red cómo se llama el lugar y dar alguna indicación para que puedan buscarlo, por ejemplo mediante geolocalización. En versiones futuras se podría incluir otros elementos a cada lugar que ayuden a describirlo mejor e incluso abrir una vía de monetarización.

Cada lugar tiene productos típicos que quiero poder mostrar a mi red. En este MVP vale una pequeña descripción y un precio aproximado de cada uno de ellos aunque lo ideal sería poder incluir fotos, enlaces, etc. El número de productos que puedo indicar por cada lugar es variable e indeterminado.

- ver mis lugares (HU6)

Como usuario identificado quiero poder ver un listado de mis lugares con la información necesaria para poder identificarlos

- crear un lugar (HU7)

Como usuario identificado quiero poder crear un nuevo lugar indicando el nombre, latitud y longitud.

- borrar un lugar (HU8)

Como usuario identificado con al menos un lugar añadido, quiero poder borrarlo de mi lista, siempre y cuando no tenga ningún viaje con pedidos pendientes al mismo.

- modificar un lugar (HU9)

Como usuario identificado quiero poder modificar la información de cualquiera de mis lugares.

- añadir productos a un lugar (HU10)

Como usuario identificado quiero poder añadir un producto a un lugar por lo que tras seleccionar el lugar añado un producto indicando la descripción y precio aproximado

- modificar un producto a un lugar (HU11)

Como usuario identificado quiero poder modificar un producto existente por lo que tras seleccionarlo puedo modificar los datos del mismo. Si hubiera un pedido pendiente que incluyera este producto debería enviarse un mensaje al solicitante notificandole del cambio.

- borrar un producto a un lugar (HU12)

Como usuario identificado quiero poder eliminar un producto existente por lo que tras

seleccionarlo puedo eliminarlo de la lista. Si hubiera un pedido pendiente que incluyera este producto debería enviarse un mensaje al solicitante notificandole de la situación y eliminandolo de la lista.

4.3. Telopido (EP3)

Cuando voy a realizar un viaje a uno de mis lugares quiero hacerselo saber a mi red de contactos mediante una nota en la que indique además del sitio al que voy, la fecha en la que volveré (no importa cúando me voy, sino cúando vuelvo). Además indicaré cuántos pedidos puedo aceptar en ese viaje de una forma orientativa y estableceré una fecha tope para aceptarlos.

De esta forma, mis contactos podrán apuntarse enviando una petición de qué producto y la cantidad que quieren pedirme. Yo recibiré estas peticiones y las podré aceptar o rechazar en función de la capacidad disponible. Tanto si acepto como rechazo, quien me ha hecho el pedido recibirá una notificación sobre su petición.

- planificar un viaje (HU13)

Como usuario identificado quiero poder planificar un viaje a alguno de mis lugares para lo cual proporcionaré una fecha de retorno y un número de pedidos máximos que puedo aceptar. Mi red recibirá un aviso de esta situación

- realizar pedido (HU14)

Como usuario seguidor recibiré mensajes en mi tablón sobre los viajes de mi red y al seleccionar uno podré realizar un pedido, indicando el producto y la cantidad que deseo. Hasta no recibir la confirmación (o rechazo) el pedido no será efectivo. En caso de que el viaje tuviera una fecha tope de aceptación de pedidos y esta se haya sobrepasado el sistema no me permitirá realizar el pedido.

- aceptar pedido (HU15)

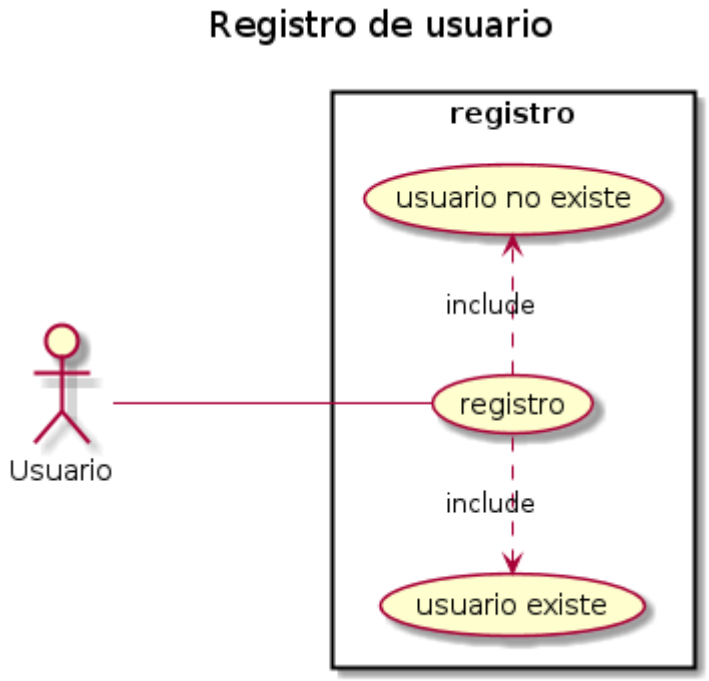
Como usuario "telotraigo" recibiré mensajes en mi tablón sobre los pedidos que me realizan mis contactos y tras seleccionarlos podré aceptarlos o rechazarlos. En caso de aceptarlos el pedido será adjuntado a mi viaje mientras que si es rechazado no se hará nada. En ambos casos mi seguidor recibirá un mensaje sobre su pedido.

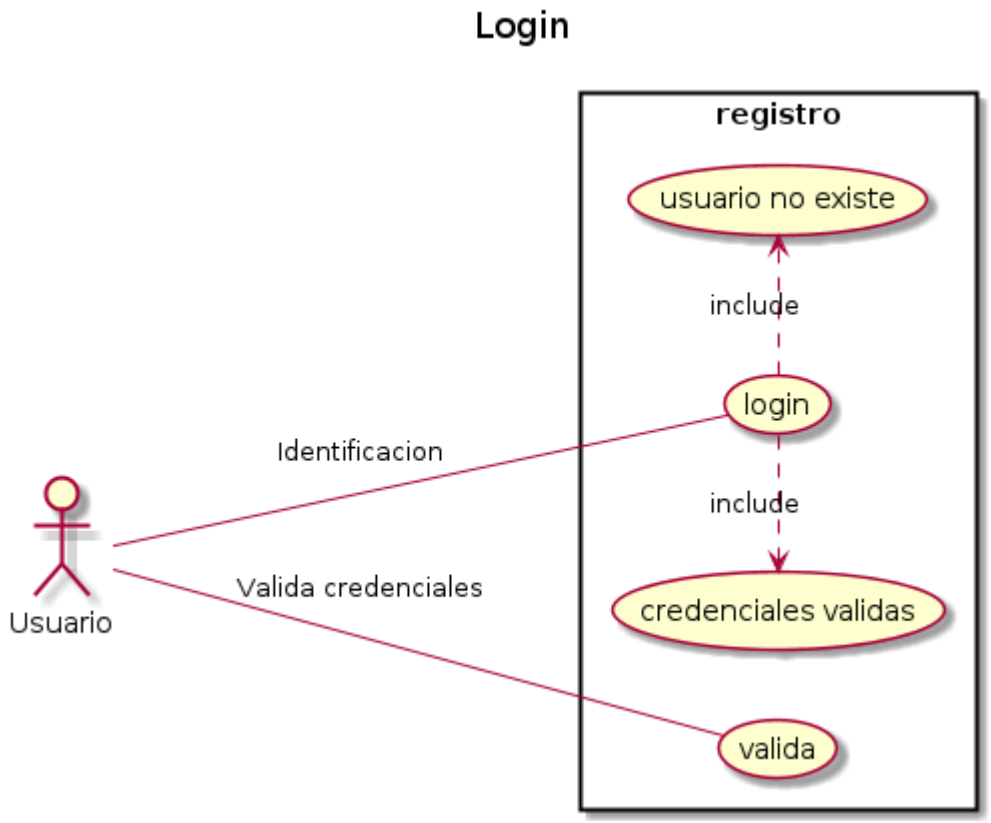
- ver situación de un viaje (HU16)

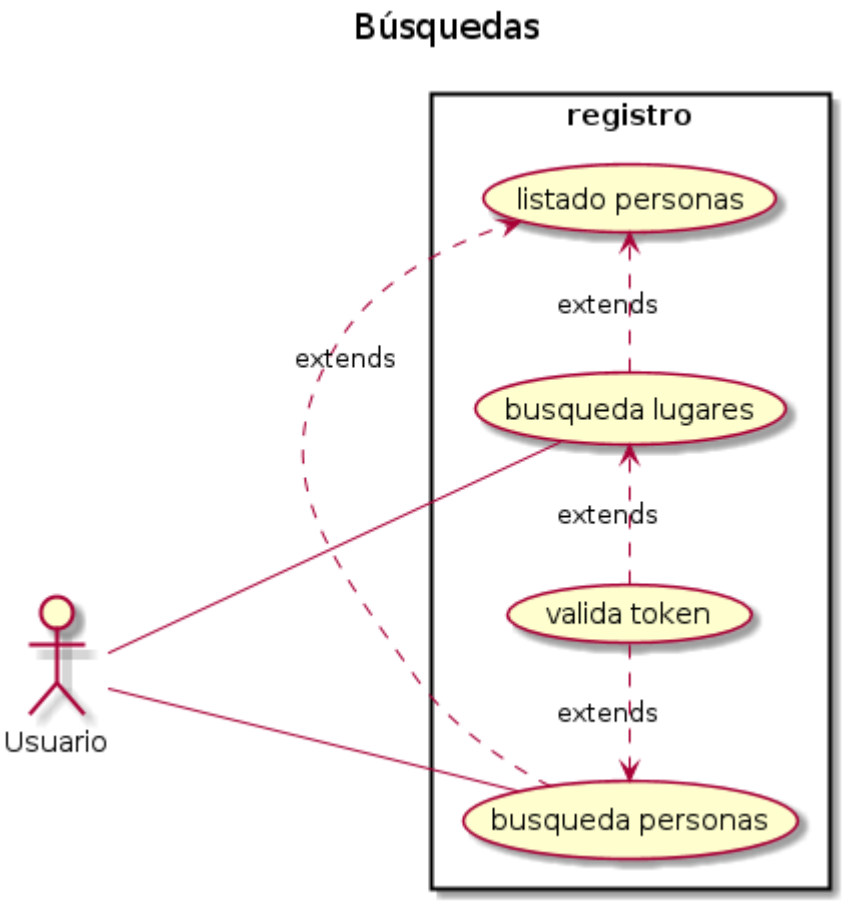
En cualquier momento como usuario "telotraigo" podré comprobar los pedidos realizados para un viaje. En una versión futura podría disponer de un mecanismo de exportarlo o enviarlo por correo, etc.

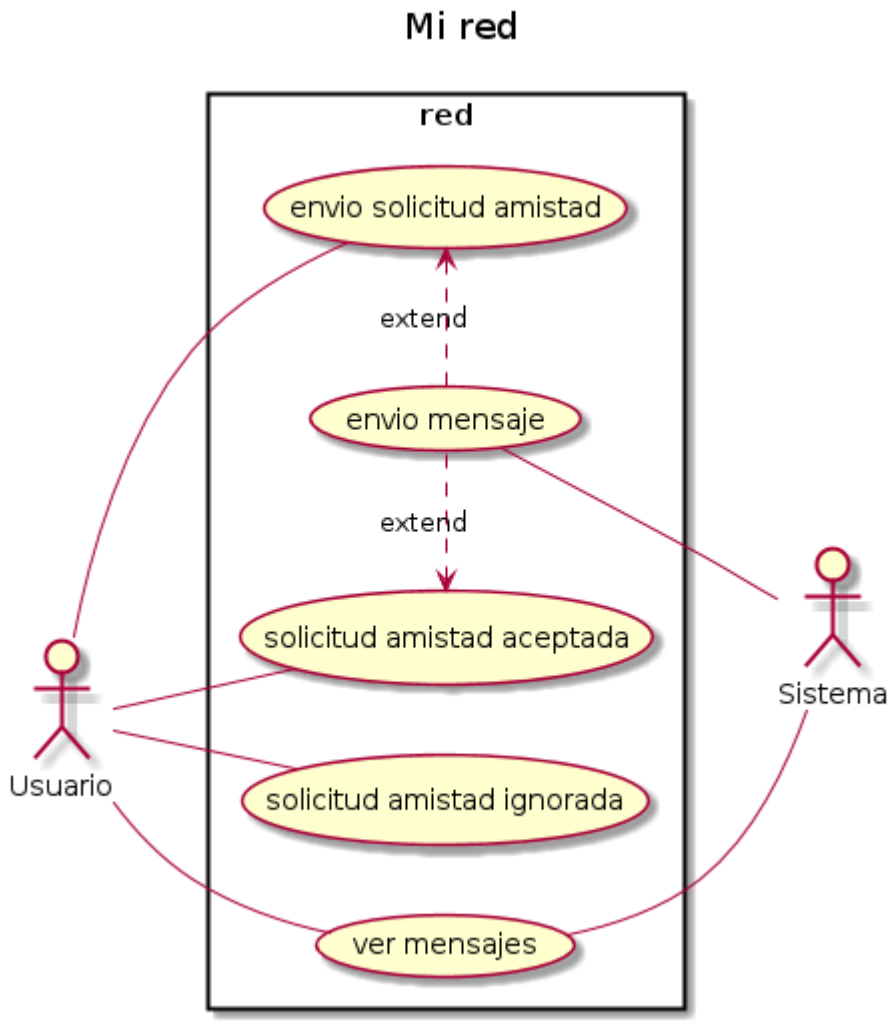
5. Casos de uso

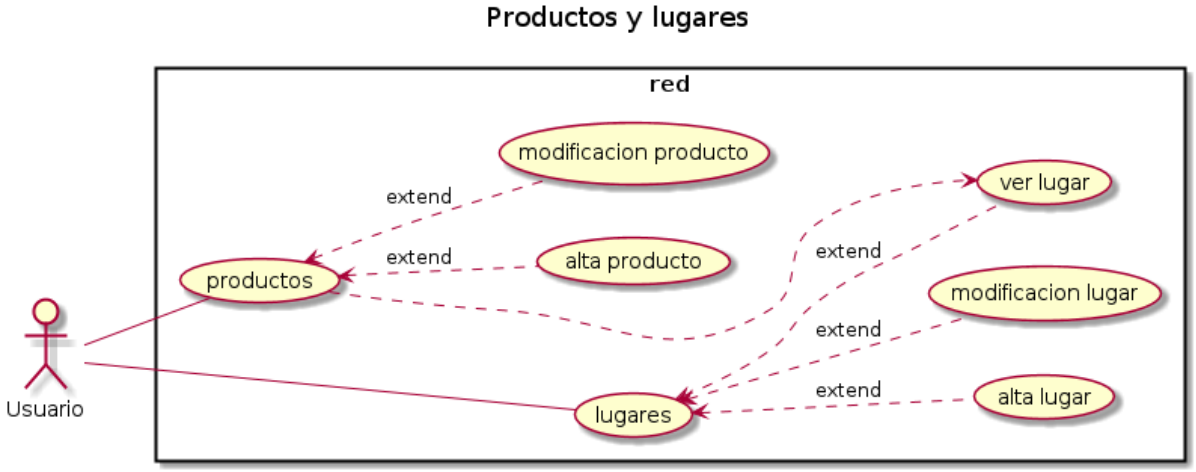
A partir del enunciado de los casos de uso iniciales se han creado los siguientes diagramas





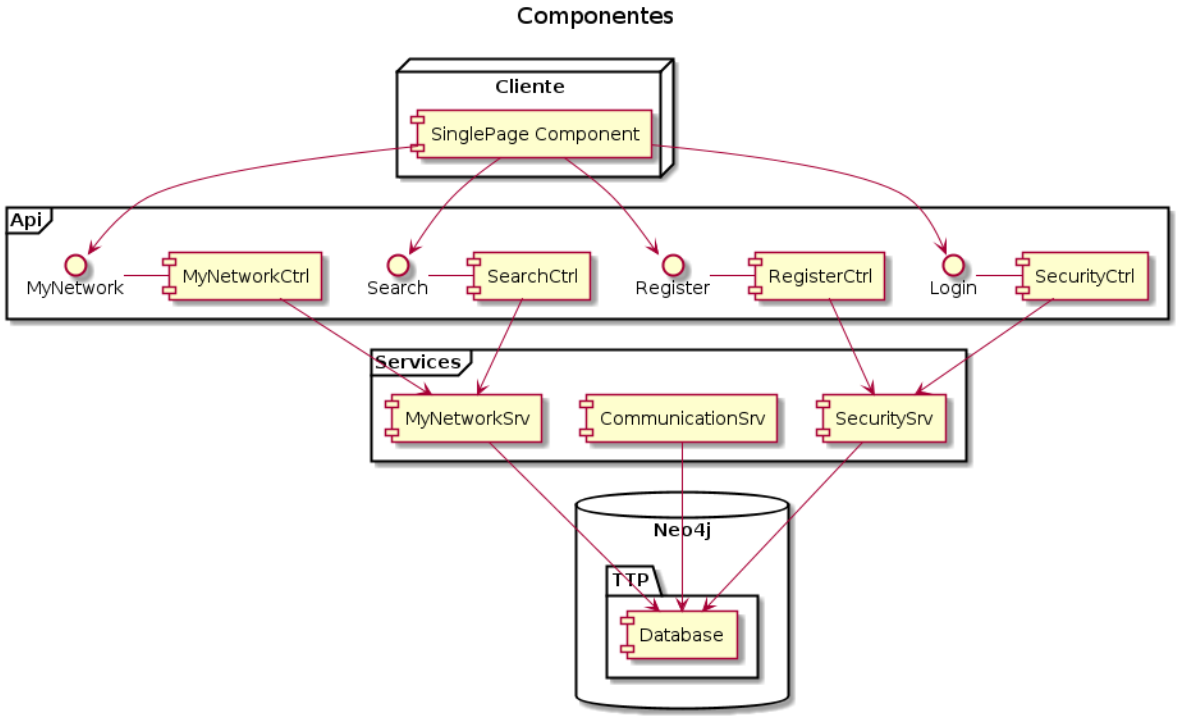

Búsquedas
registro
listado personas
extends
extends
busqueda lugares
extends
valida token
extends
busqueda personas
Usuario


Mi red
red
envio solicitud amistad
extend
envio mensaje
extend
solicitud amistad aceptada
solicitud amistad ignorada
ver mensajes
Usuario
Sistema


Productos y lugares
red
modificacion producto
extend
ver lugar
extend
alta producto
extend
productos
modificacion lugar
extend
extend
alta lugar
lugares
Usuario

6. Componentes

En esta fase realizamos una definición de componentes de alto nivel donde distinguimos un cliente que comunicará con diferentes partes del API, las cuales a su vez delegarán en los servicios oportunos la resolución de la petición. Todos los servicios utilizarán el mismo repositorio (Neo4j).



7. Diagramas de Secuencia

A continuación se representan los tres diagramas de secuencia principales de la aplicación.

Diagrama de secuencia donde un usuario anónimo desea registrarse en el sistema. Al no estar registrado el API no puede securizarse mediante el intercambio de tokens

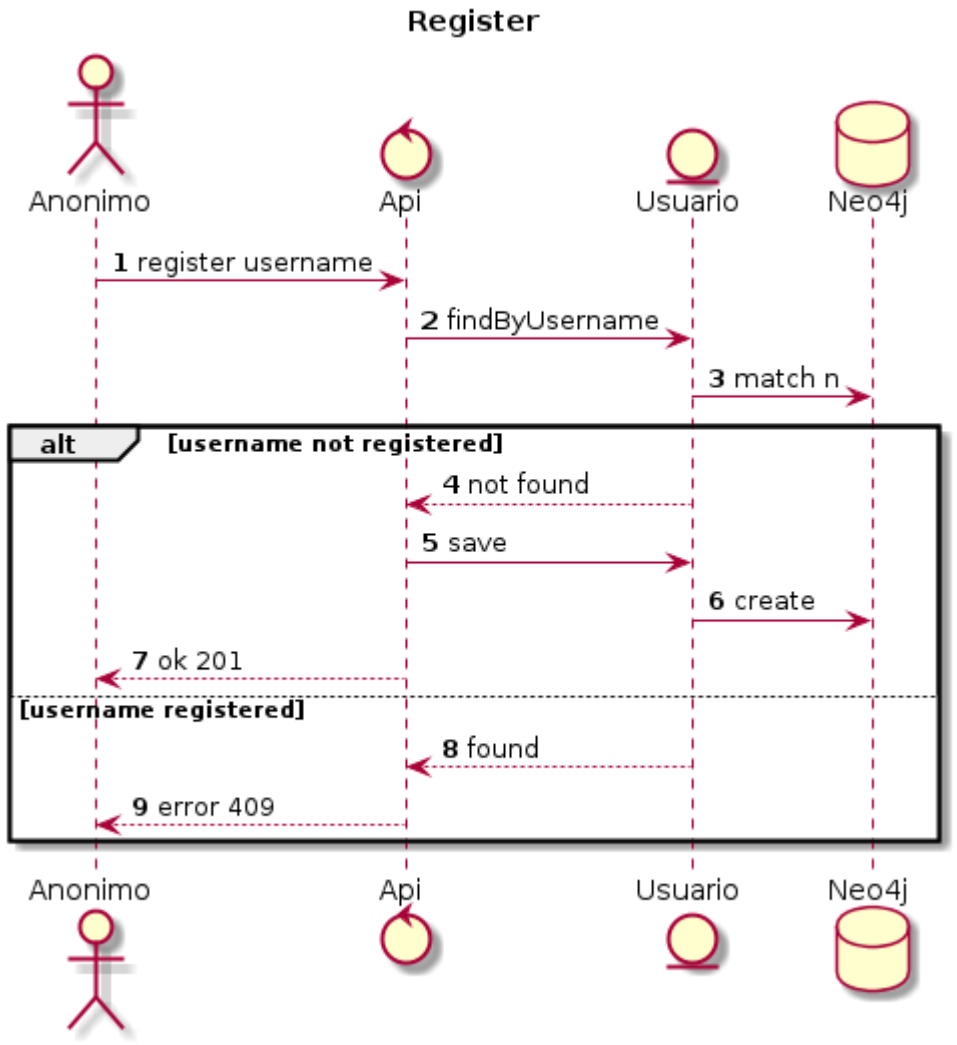


Diagrama de secuencia donde un usuario anónimo desea identificarse en el sistema. Si la identificación es correcta obtendrá un token para usar en futuras peticiones

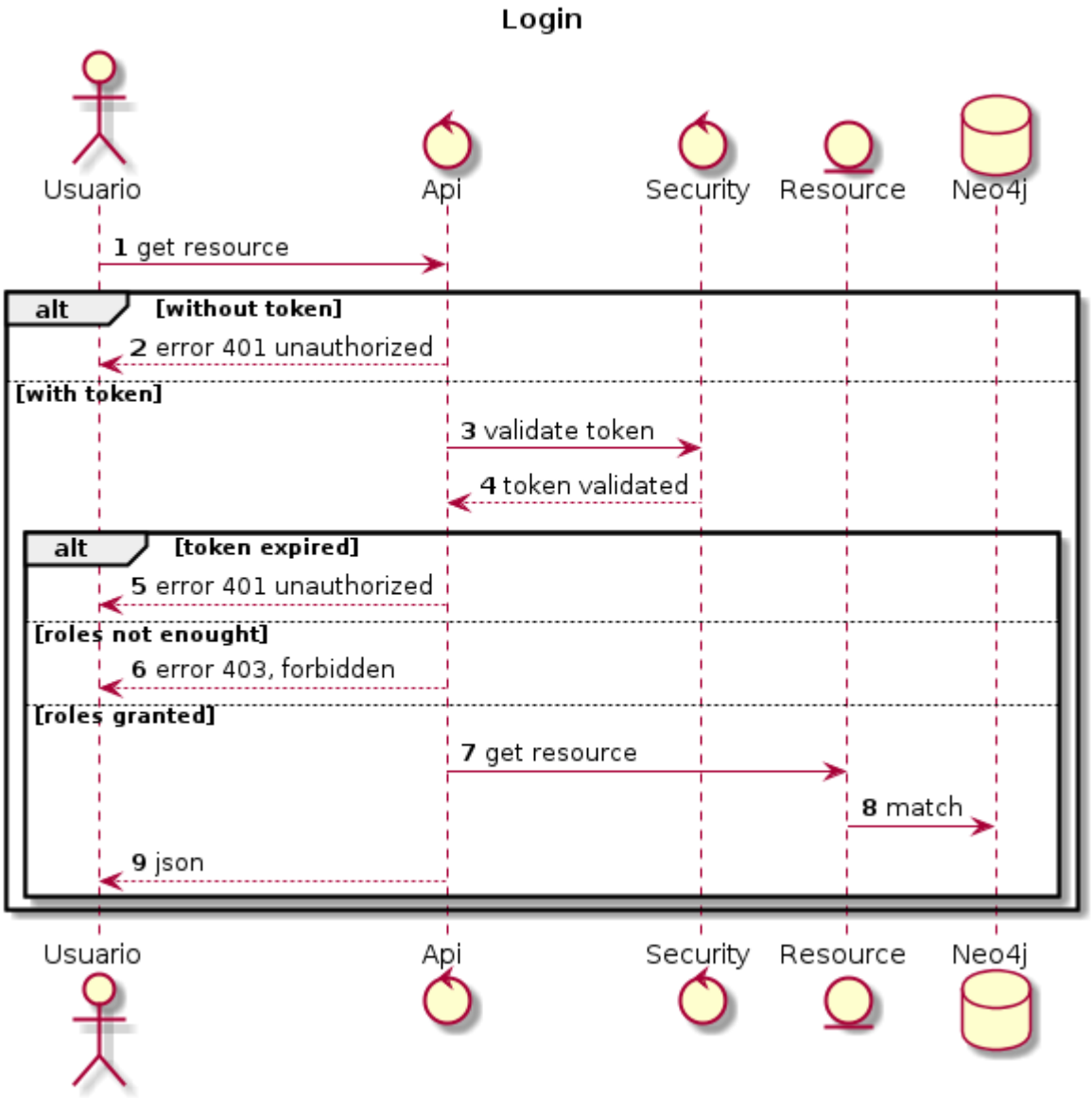
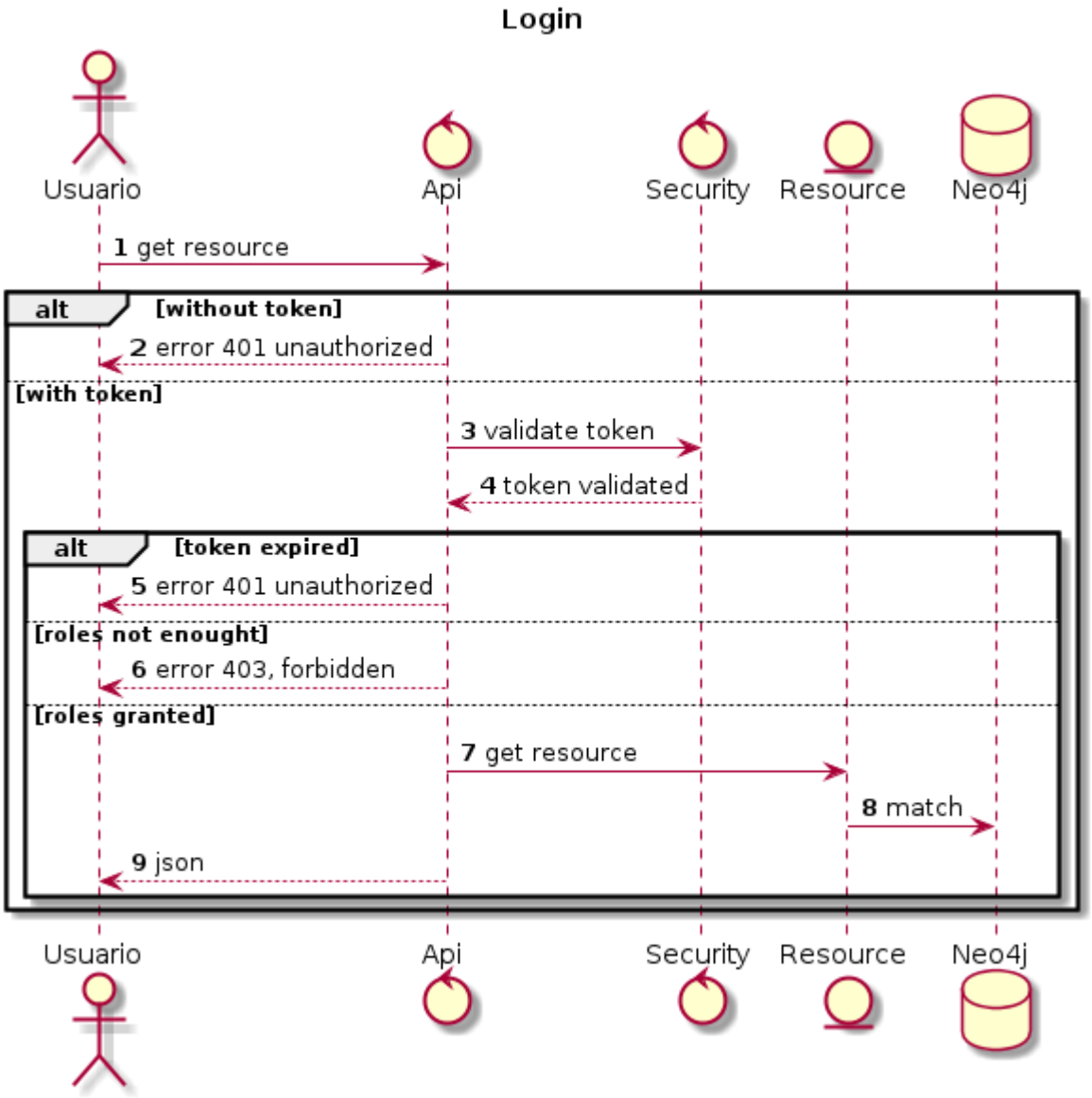


Diagrama de secuencia donde un usuario identificado previamente y que disponde de un token, desea acceder a un recurso protegido (por ejemplo ver sus mensajes)



8. Aprovisionamiento de infraestructura

Una vez establecidas las líneas generales del proyecto y con una idea sobre qué herramientas y tecnología vamos a trabajar procedemos a aprovisionar aquellos recursos que necesitaremos para realizar un ciclo completo inicial. La idea es conseguir, de una forma rápida, un pipeline completo desde el desarrollo hasta el despliegue que nos ayude a identificar todos los elementos que intervendrán y así reducir riesgos.

En nuestro caso, para el desarrollo y de cara a infraestructura, vamos a necesitar:

- un repositorio de códigos y artefactos accesible desde diferentes sitios (Gitlab)

Así mismo para el despligue requeriremos:

- registrar un nombre de dominio
- proveedor de cuentas de correo.
- un servidor de aplicaciones accesible desde Internet (OpenShift, Heroku, ..)
- un servidor de base de datos Neo4j. Crearemos uno desde cero o usaremos servicios como GrapheneDB que permiten crear una base de datos Neo4j para pruebas de concepto.

8.1. Dominio y correo

El registro de un dominio nos permite principalmente tener presencia en Internet de nuestro producto. Hoy en día existen sufijos de dominio (.com, .es, .info, etc) por unos 12 euros pero recientemente se han creado nuevos sufijos que nos permiten registrar nuestro nombre por menos de 1 euro. En nuestro caso el dominio registrado sera http://telotraigodemipueblo.club

La mayoría de proveedores de dominios suelen ofrecer planes donde incluyen un espacio para alojar contenido estático y alguna cuenta de correo. Mientras que el contenido estático para nuestro caso no reporta grandes ventajas, el poder disponer de alguna cuenta de correo nos va a permitir poder acceder a servicios de infraestructura PaaS, SaaS, IaaS, etc y asociarlo al mismo, de tal forma que podremos disponer de un repositorio de código Git, instancias donde correr la aplicación, etc (si bien hay que tener en cuenta que algunos de ellos no ofrecerán capacidad como para desplegar nuestro producto al mercado sino más bien a validar la idea de negocio).

8.2. Repositorio de código Git

Hoy en día el sistema por excelencia para el control del código es sin duda Git, por encima de Subversion, Mercurial, etc.

Git puede ser instalado de una manera fácil en un equipo alojado en nuestra propia red (incluso en el mismo equipo que usamos para trabajar) aunque existen proveedores que

ofrecen este servicio de forma gratutita o a un precio asequible. La ventaja que ofrecen estos servicios no es sólo que nos evitamos el tener que instalar y mantener el sistema sino la posibilidad de dar visibilidad a nuestro código a la comunidad permitiendo así tener feedback e incluso colaboraación de la misma.

Como servicios principales podemos destacar a:

- Bitbucket, ofrece alojamiento gratuito para equipos de menos de 5 miembros.
- Github, ofrece alojamiento gratuito para proyectos OpenSource
- Gitlab, igual que los anteriores pero también permite crear repositorios privados con un límite de espacio.

Además del servicio básico de repositorio de código suelen incluir herramientas de gestión de incidencias, procesos de Integración Contínua (CI), generación de páginas estáticas (útiles para documentar el proyecto), etc.

Para este proyecto **hemos seleccionado Gitlab básicamente por experiencias anteriores positivas** aunque cualquiera de los anteriores hubieran sido válidos.

8.3. Servidor de aplicaciones

Debido a que nuestro proyecto será una aplicación J2EE Grails debemos buscar proveedores que permitan el despliegue de aplicaciones de este tipo, lo cual no es fácil. A diferencia de aplicaciones PHP (por ejemplo) los requisitos de un servidor J2EE son mayores por lo que los proveedores habituales no ofrecen este servicio.

Aunque desactualizada, la página https://grails.org/wiki/hosting nos proporciona un listado de proveedores que ofrecen servicio de alojamiento para este tipo de aplicaciones. Sin embargo de todos ellos los únicos que proporcionan un plan free son:

- Heroku, ofrece una aplicación free con 512 Mb de memoria y con un sistema integrado en Git de despliegue. Para probar este servicio se ha creado un aplicación Grails de prueba que se encuentra en el directorio "testheroku" de este proyecto.
- Cloudbees ofrece el mismo servicio pero con sólo 128 Mb de memoria las cuales son insuficientes para una aplicación mínima.

Así mismo, y con la madured de la tecnología de la "Nube" existen proveedores que nos ofrecen crear máquinas virtuales donde alojar nuestras aplicaciones con unos requisitos mínimos como pueden ser:

- Amazon Web Services (AWS), ofrece un año de prueba gratuito en casi todos sus productos, incluido una instancia EC2. Sin embargo la máquina que se permite en dicho plan es realmente escasa y deberíamos optar a una instancia superior, aunque a un precio muy económico.
- OpenShift de RedHat. Hasta hace poco ofrecía un servicio en su version 2 donde podíamos aprovisionar hasta 3 aplicaciones de forma gratuita. Sin embargo dicho plan

se encuentra en proceso de desaparición por el nuevo plan 3 basado en Docker. Aún así se ha probado dicho plan con una instancia DIY (DoItYourself) donde hemos podido desplegar una aplicación Grails para probar el proceso de despliegue. Dicha aplicación se encuentra en el directorio "testopenshift" de este proyecto. La idea básica para realizar una integración contínua en este contenedor reside en los ficheros que se encuentran en el directorio .openshift y que actuan como "hooks"

Tras evaluar las diferentes posibilidades **la que mejor se ha adaptado a nuestro proyecto ha sido Heroku** la cual nos ha permitido desplegar la aplicación sin mayores complicaciones en un tiempo corto y de forma integrada en nuestro pipeline. Básicamente la idea de Heroku es que una vez que realizamos un push en la rama "heroku" de nuestro proyecto se lanza un deploy del mismo y si el resultado es satisfactorio se reemplaza la instancia en poco más de un minuto. Para ello debemos incluir un fichero Procfile que le indica el tipo de proyecto , y el comando a ejecutar para desplegar nuestra aplicación.

Para más detalle se puede seguir este tutorial: https://devcenter.heroku.com/articles/getting-started-with-grails#create-a-heroku-app-and-deploy

8.4. Proveedor Neo4j

Al ser Neo4j un motor de base de datos un tanto atípico, encontrar una solución donde alojarle resulta un tanto problemática. Actualmente multitud de servicios de hosting ofrece MySql, PostgreSQL, o similares de forma prácticamente gratuita y con recursos más que suficientes para desarrollar un producto listo para el mercado.

Así pues la elección de Neo4j en un primer momento ha sido desplegar una instancia en alguno de los proveedores citados anteriormente. De esta forma hemos conseguido crear una instancia en OpenShift donde poder alojar un motor de Neo4j pero a costa de demasiado esfuerzo (gestión de puertos, paradas y arranques, etc)

La segunda elección ha sido optar por un proveedor de Neo4j, **GrapheneDb** que ofrece un plan gratuito orientado al desarrollo de prototipos donde alojar hasta 1.000 nodos y 10.000 relaciones, y con interface REST lo que nos permitirá conectar nuestra aplicación alojada en Heroku dialogar con la base de datos alojada en GrapheneDb.

El proyecto testheroku incluye algunas clases de Dominio así como un Controller y test de integración, destinados a validar dicha arquitectura.

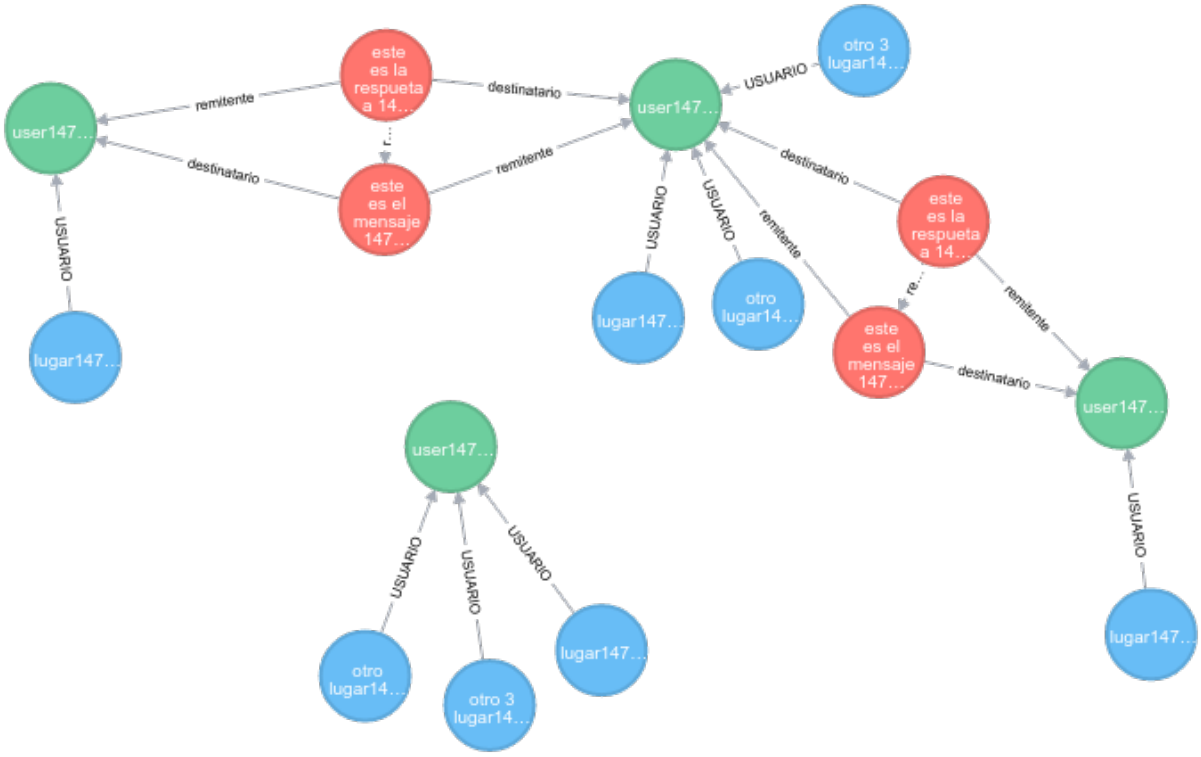
9. Base de datos de Grafos

9.1. Conceptos básicos

El diseño del modelo de datos en una aplicación de grafos como es Neo4j difiere del diseño tradicional de una base de datos relacional.

En una base de datos de grafos las entidades se identifican como nodos con atributos que pueden tener un número indeterminado de relaciones entre sí. Por ejemplo en el siguiente diagrama podemos ver:

- entidad Usuario (4 nodos verdes)
- entidad Mensaje (4 nodos rojos)
- entidad Lugar (8 nodos azules)
- relaciones diferentes entre nodos (remitente, destinatario, usuario) que nos permiten navegar entre ellos



Así mismo la forma de consultar a la base de datos difiere de la tradicional SQL. En este tipo de motores la consulta básica se realiza mediante "MATCH":

```
MATCH (n) return n  ①
MATCH (n:Usuario) return n  ②
MATCH (n:Usuario {nombre:'name'}) return n ③
MATCH (n:Usuario {nombre:'name'})-[:VISITO]->(l:Lugar) return n,l ④
```

① retorna todas las entidades de la bbdd

② retorna todas las entidades del tipo Usuario

③ retorna las entidades del tipo Usuario con el atributo nombre='name'

④ retorna un usuario y todos los lugares con los que tiene una relacion de 'VISITO'

9.2. Grails y Neo4j

Para manejar nuestro modelo de datos utilizaremos el plugin de Grails Neo4j el cual nos facilitará la creación de las entidades, sus relaciones y consultas.

De esta forma nuestro trabajo consistirá en definir nuestras entidades programáticamente y delegar en dicho plugin la gestión de las mismas creando una clase *Domain.groovy* en la carpeta **domain**

Así por ejemplo para definir una entidad de dominio Usuario la definiríamos en el propio código:

Listing 1. Usuario.groovy

```
package ttp
import grails.persistence.Entity
import grails.rest.Resource

@Entity

class Usuario {

    String nombre       ①
    String username
    String password

    boolean enabled = true
    boolean accountExpired
    boolean accountLocked
    boolean passwordExpired

    static hasMany = [          ②
            confia : Usuario,
            conoce : Lugar
    ]

    Set<Role> getAuthorities() {
        UserRole.findAllByUser(this)*.role
    }

    def beforeInsert() {
        encodePassword()
    }
```

```
    def beforeUpdate() {
        if (isDirty('password')) {
            encodePassword()
        }
    }

    protected void encodePassword() {
        password = springSecurityService?.passwordEncoder ?
springSecurityService.encodePassword(password) : password
    }

    transient springSecurityService

    static transients = ['springSecurityService']

    static constraints = {          ③
        nombre nullable:false
        username nullable: false, email:true
        password nullable:false, password:true
    }

    static mapping = {              ④
        username index:true, unique:true
    }


    Usuario amigos(Usuario amigo){ ⑤
        this.addToConfia(amigo)
        amigo.addToConfia(this)
        this
    }

}
```

① Definimos propiedades de la entidad

② Definimos relaciones con otras entidades que gestionará Grails

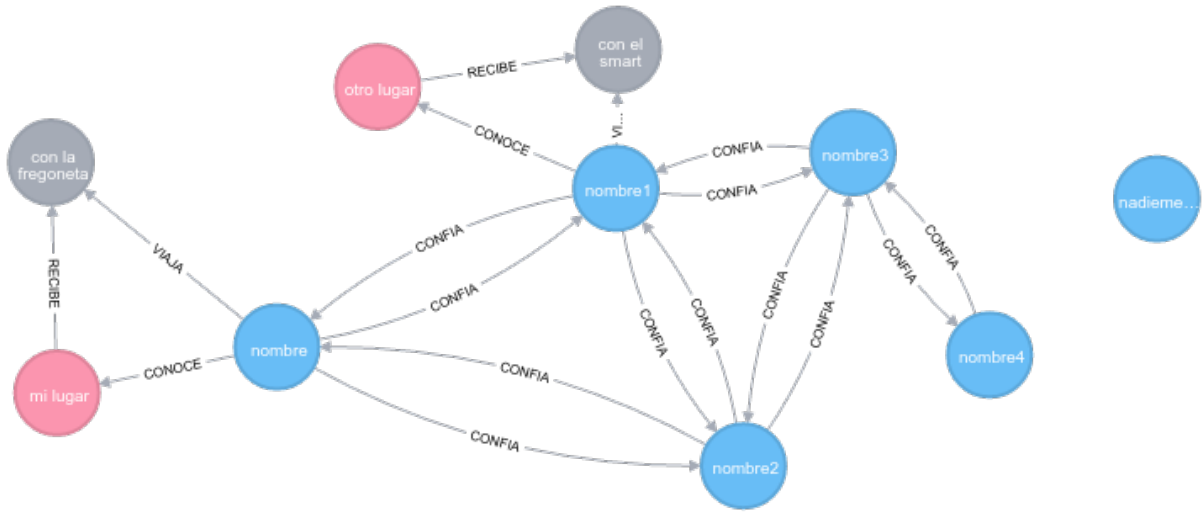
③ Podemos indicar que restricciones tiene nuestra entidad

④ Ajustamos nuestro modelo al motor

⑤ Podemos incluir funciones en una entidad que nos ayuden a centralizar la lógica

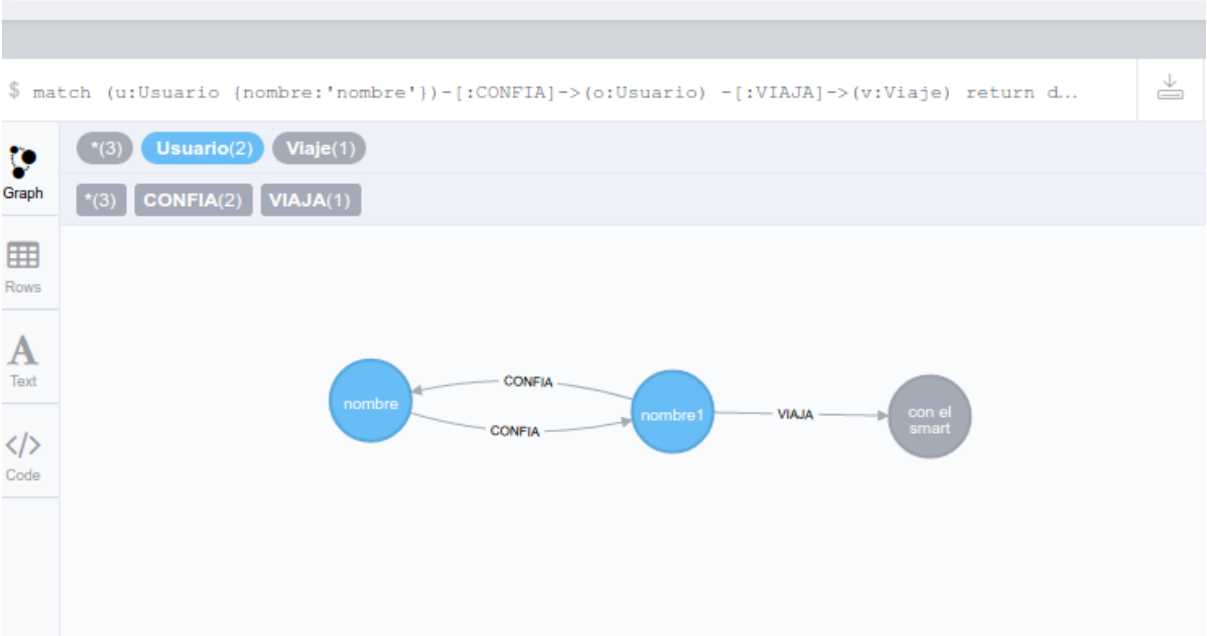
9.3. Ejemplos de diagramas de nodos

Ejemplo de una red de amigos donde dos de ellos tienen previsto viajar a lugares diferentes:



Ejemplo de una consulta donde dado un usuario 'nombre' podemos ver los viajes de su red de amistades (extraído del ejemplo anterior):

```
1 match (u:Usuario {nombre:'nombre'})-[:CONFIA]->(o:Usuario)
2                 -[:VIAJA]->(v:Viaje)
3                 return distinct(v),o,u
```



10. Metodología y herramientas

Desde el primer momento se ha pretendido aplicar una metodología ágil orientada a la ejecución de sprints cortos donde se aporte valor en cada uno de ellos, buscando conseguir un procedimiento de despliegue contínuo donde cada entrega pueda ser puesta en producción al momento y con el menor esfuerzo, sin olvidar por ello la calidad de la entrega.

Así mismo, y como parte del proyecto, se ha buscado generar toda la documentación relativa al mismo en un formato abierto que permita su publicación de la misma manera por lo que la generación de esta se ha integrado en el propio proceso de despligue contínuo.

Con estas premisas, y una vez finalizado el proyecto, podemos destacar:

- El uso intensivo de Git como control de versiones ha permitido trabajar en diferentes partes del proyecto a la vez, incluso en diferentes lugares, aprovechando al máximo el tiempo del que se diponía.
- Gitlab como proveedor tanto del repositorio Git como de las herramientas asociadas (gestión de Issues, páginas estáticas, ejecución de Pipelines, despliegue de aplicaciones, etc). Igualmente podíamos haber usado Github o alguno similar.
- El uso de Asciidoctor como herramienta de documentación ha servido para que esta estuviera controlada, organizada y que pudiera ser reutilizada en diferentes partes del proyecto. Su formato abierto y orientación al contenido sobre el aspecto ha permitido generar una extensa documentación del proyecto. Así mismo su integración con los test de integración han permitido incluso generar la documentación del API de forma automática
- Neo4j y GrapheneDB como motor de base de datos y proveedor de la misma han permitido reducir el riesgo que suponía utilizar un paradigma diferente al típico relacional.
- Groovy, Grails y Gradle como lenguaje, framework y gestión de tareas respectivamente, tras una curva de aprendizaje, ofrecen un entorno realmente productivo donde el desarrollo de aplicaciones completas se reducen de meses a semanas.

El enfoque de proyecto abierto hace que cualquiera pueda consultar la evolución del mismo desde el primer commit estando alojado de forma pública en https://gitlab.com/telotraigodemipueblo/tfg

11. Entorno de desarrollo

Para la gestión del proyecto (compilado, dependencias, despliegue, etc) utilizaremos http://gradle.org

Gradle es similar a Maven en el sentido de que nos proporciona herramientas para organizar nuestros proyectos, declarar las dependencias de los mismos, etc.

En nuestro caso la estructura del proyecto será una estructura de multiproyecto donde tenemos un proyecto principal vacío que simplemente declara dependencias a los subproyectos.

11.1. Subproyecto docs

El proyecto **docs** es un proyecto donde hacemos uso del plugin asciidoctor de gradle (entre otros) para que al ser invocado se genere la documentación tanto HTML como Pdf. Esta documentación hace referencia a las diferentes entregas del TFG, como es la planificación, hitos, memoria, etc

Dicha documentación se encuentra estructura en múltiples ficheros .asciidoc para poder ser reutilizados tanto a la hora de generar un HTML como a la hora de generar los Pdfs a entregar siendo el documento *src/docs/asciidoc/index.adoc* el fichero principal para la parte HTML y los ficheros *src/docs/asciidoc/pec2.adoc* y *src/docs/asciidoc/pec3.adoc* ficheros principales para la generación PDF.

Cuando se invoca la tarea **build**, este proyecto ejecuta dos tareas (buildHtml y buildPdf) que generan un HTML **index.html** y diferentes PDFs, todos ellos en la carpeta de salida **buid/asciidoctor**. Dicha carpeta será utilizada para su publicación en las páginas públicas de Gitlab haciendolos accesibles por Internet.

11.2. Subproyecto telotraigodemipueblo

Este proyecto corresponde al aplicativo a generar y contiene básicamente el código groovy a generar junto con los test de integración para asegurar la calidad del software.

Cuando se invoca la tarea **build** en este proyecto se ejecuta todo el pipeline propio de Grails donde se pasan por las fases propias de compilado, ensamblado, test de integración, y generación del producto final. Este producto final es utilizado por la herramienta de despliegue de Heroku para reemplazar la versión actual por la nueva de forma automática.

11.3. Subproyecto api

Este proyecto en realidad es un proyecto "virtual" en el sentido de que no existe como tal sino que es un proyecto creado en Gitlab con el nombre **api** para disponer de una URL donde poder publicarla. Cuando se hace un versionado de la aplicación se ejecuta un

trigger que descarga la última versión del proyecto y ejecuta los test los cuales a su vez generan la documentación del API lista para ser publicada en http://telotraigodemipueblo.gitlab.io/api

11.4. Subproyecto reports

Al estilo del anterior, este proyecto también es un proyecto "virtual" con la particularidad de que, a diferencia del anterior, permite errores en los test pues está destinado a generar un report de los test tanto satisfactorios como erróneos en http://telotraigodemipueblo.gitlab.io/reports

12. Database (Neo4j)

La persistencia de la aplicación se hace enteramente sobre un motor Neo4j (base de datos de grafos, diferente a la tradicional relacional). Dicho motor puede ser instalado siguiendo el tutorial de la página oficial https://neo4j.com/download/ o bien si tenemos Docker instalado podemos usar la imagen oficial https://hub.docker.com/_/neo4j/ .

Así mismo http://www.graphenedb.com/ ofrece alojamiento gratuito donde crear instancias de Neo4j aunque limitadas a un número de nodos y relaciones (suficientes para el objetivo de este proyecto). Tras probar las diferentes opciones al final se ha optado por utilizar este servicio por las siguientes razones:

- aunque ofrece un número limitado de nodos y relaciones son suficientes para poder mostrar la funcionalidad del aplicativo
- accesible desde cualquier estación de trabajo con conexión a Internet, lo que permite el desarrollo en diferentes sitios incluso simultáneos
- fácil de instalar (sólo requiere de una cuenta de correo para confirmar las credenciales) y de administrar.

Gracias a que GrapheneDB nos permite crear varias bases de datos (todas con la misma limitación de nodos) se han creado tres instancias:

- telotraigodemipueblo_dev, para el entorno de desarrollo
- telotraigodemipueblo_ci, utilizada por los test de integración y despligeue contínuo
- telotraigodemipueblo, utilizada por el entorno de real

A continuación se detalla la configuración necesaria para el entorno de desarrollo y de test:

Listing 2. ${HOME}/.gradle/gradle.properties

```
NEO4J_URL_DEV=http://hobby-
pdjgpmbeoeaggbkeccahdkol.dbs.graphenedb.com:24789/db/data/
NEO4J_USER_DEV=telotraigodemipueblo_dev
NEO4J_PWD_DEV=XXXXXXXXXXXXXX  ①

NEO4J_URL_TEST=http://hobby-
ocnnamhijildgbkeiaigdgol.dbs.graphenedb.com:24789/db/data/
NEO4J_USER_TEST=telotraigodemipueblo_ci
NEO4J_PWD_TEST=XXXXXXXXXXXX ①
```

① Configurar la password correspondiente

En el entorno de producción (Heroku) se han configurado las variables correspondientes al entorno de producción mediante el interface que proporciona dicha plataforma.

```
NEO4J_URL=http://hobby-xxxxxxxxxxx.dbs.graphenedb.com:24789/db/data/
NEO4J_USER=telotraigodemipueblo_dev
NEO4J_PWD=XXXXXXXXXXXXXX  ①
```

① Configurar la password correspondiente

13. Grails

Para el backend hemos usado Grails (http://www.grails.org) como framework de desarrollo. Grails nos proporciona un entorno donde la convención prevalece sobre la configuración de tal forma que si seguimos las convenciones establecidas (nombre, rutas, etc) prácticamente no tendremos que realizar ninguna tarea de configuración.

Así mismo Grails proporciona todo un stack J2EE (spring-framework, hibernate, sitemesh, etc) junto con un ecosistema de plugins muy amplio destinado a proporcionar un desarrollo muy rápido.

Por último comentar que Grails se basa en el lenguaje Groovy, el cual es un lenguaje dinámico que corre sobre una JVM y que cuenta con una gran popularidad actualmente.

Respecto de nuestra aplicación, a continuación se describen los artefactos de interés:

- grails-app
  - controllers
    - telotraigodemipueblo.UrlMappgings: configuración de puntos de entrada a los diferentes controllers
    - telotraigodemipueblo/api/core: Controllers de uso básico (registrarse, mi perfil, invitar a unirse, etc)
    - telotraigodemipueblo/api/crud: Controllers orientados al CRUD de diferentes recursos (lugares, productos,etc) relacionados con un usuario logeado
    - telotraigodemipueblo/api/network: Controllers destinados a interconectar a un usuario con su red (hacer un pedido, ver viajes en mi red, etc)
  - domain
    - ttp: Objetos de dominio con etiquetas necesarias para la persistencia. Siguiendo las convenciones de Grails las propiedades de estos serán persistidas en el motor de datos.
  - views: Siguiendo las conveciones de Grails cada carpeta está destinada a renderizar los diferentes objetos en formato json
  - services
    - telotraigodemipueblo/EventToMessageService: Servicio destinado a recibir en un solo punto los eventos que genera la aplicación (como por ejemplo LugarCreado, PedidoSolicitado, etc) y convertirlos en Mensajes hacia los usuarios interesados.
- src
  - integration-test/groovy/ttp
    - domain: Test de integración (Spock Framework) destinados a validar las querys

básicas de los objetos de dominio.

- docuapi: Test de integración (Spock Framework) destinados a validar las diferentes APIs. Estos test son usados así mismo para generar la documentación del API creando cada uno de ellos fragmentos de documentación relativos a la petición REST (parámetros de envío, cabeceras, respuesta esperada, etc). En caso de que un test falle o no la respuesta recibida no coincida con la espera se aborta la generación del aplicativo. Debido a que estos test se ejecutan a través de HTTP contra el propio servidor se han desarrollado una serie de utilidades que nos permitan realizar la autentificación, crear elementos para pruebas, etc.

- plugins: en el fichero **build.gradle** definimos los plugins necesarios para nuestra aplicación. En concreto para TelotraigodemiPueblo hemos añadido "org.grails.plugins:spring-security-rest:2.0.0.M2" (para securizar las llamadas al API), 'org.springframework.data:spring-data-neo4j-rest:3.4.6.RELEASE' (para conectar con una base de datos Neo4j remota), y "org.springframework.restdocs:spring-restdocs-core:$springRestdocsVersion" (para generar la documentación del API)

14. Cliente (Javascript+HTML con AngularJS)

Para la parte cliente hemos utilizado el framework Javascritp Angular 1.5 el cual nos permite por un lado una interacción con el cliente y por otro nos ofrece funcionalidades de comunicación con el backend vía REST.

Respecto de nuestra aplicación, a continuación se describen los artefactos de interés:

- grails-app
  - assets/javascripts/telotraigodemipueblo
    - core: Módulo AngularJS donde definimos componentes de uso genérico como el buzón del usuario, el perfil de este, y los Resource mapeados con los diferentes REST del backend
    - index: Módulo AngularJS donde definimos componentes de inicio, como la bienvenida, registro y login
    - network: Módulo AngularJS donde definimos componentes de gestión de la red, como la búsqueda de usuarios o los viajes de mis amigos

Para cada módulo organizamos los componentes de la misma forma:

- controllers: donde ubicamos los controllers característicos de Angular
- domain: donde mapeamos los Resouces REST
- templates: donde diseñamos los componentes HTML

Grails nos permite crear test de la parte cliente pero para esta fase se han suprimido por no complicar más el desarrollo.

15. Integración contínua (Gitlab y Heroku)

Desde el principio el proyecto se ha orientado hacia un proceso de Integración Contínua, donde la automatización de todos los procesos pudieran permitir el despliegue no sólo de la aplicación (Heroku) sino de la propia documentación (Gitlab).

El diseño de dicho proceso sigue los siguientes pasos:

- Tras realizar un desarrollo determinado (añadir nueva documentación, resolver una incidencia, o añadir una nueva funcionalidad) en la rama git correspondiente se procede a integrarla en la rama **master**.
- Se ha añadido un hook en el repositorio Git que se ejecuta en local cada vez que se hace un commit en la rama master, el cual ejecuta la task **check** para validar que lo que se intenta versionar pasa los test. Si es correcto permitirá realizar un **pull** al repositorio **origin** (Gitlab)
- Mediante el fichero **.gitlab-ci.yml** configuramos los hooks de servidor en Gitlab que le indicarán lo que hay que hacer en cada caso. Básicamente ejecutaremos dos tipos de tareas:
  - una de ellas destinada a generar la documentación y publicarla en el espacio que Gitlab nos ofrece (al estilo de Gh_pages de Github)
  - otra de ellas destinada a empaquetar la aplicación y enviarla a Heroku para su despliegue. Esta se ejecuta únicamente cuando etiquetamos el proyecto para poder asociar versión desplegada con su etiqueta.
- Mediante el fichero **Procfile** Heroku puede conocer los pasos que tiene que realizar para desplegar la aplicación. Previamente al ser un proyecto Gradle, ejecutará la tarea **stage** encargada de validar la aplicación descargada y si es correcta generar el aplicativo a desplegar.

16. Valoración económica

Tal como se ha venido explicando a lo largo de este documento, para la consecución de este MVP se han utilizado diferentes herramientas y servicios de libre uso o en su caso el plan de servicio gratuito que permitiera su uso a un nivel adecuado.

Así, en esta entrega se puede afirmar que el producto obtenido **ha tenido un coste de 1 euro** por el registro del dominio **telotraigodemipueblo.club** y que es totalmente funcional aunque cuenta con con unas limitaciones de recursos. Por ejemplo la base de datos está limitada a 1.000 nodos (es decir el conjunto de usuarios, lugares, productos, viajes y pedidos) y el servidor de aplicaciones detiene la instancia tras un período de inactividad.

Ambos componentes pueden ser ampliados según demande el negocio sin tener que modificar ni la arquitectura ni el aplicativo, simplemente actualizando el plan de precios de cada uno.

Así por ejemplo podemos incrementar la capacidad de la base de datos a una instancia reservada con 1GB de capacidad por 50 euros (http://www.graphenedb.com/pricing) o ampliar el servidor de aplicaciones por 25 euros al mes (https://www.heroku.com/pricing)

Por otra parte, gracias al diseño REST stateless, sería perfectamente fácil desarrollar la parte cliente para móviles (y/o tablet) en un período corto de tiempo al contar con un API perfectamente documentada y probada.

Aunque en su estado actual la aplicación NO puede ser comercializada (por estética y limitada de recursos) sí es válida para ser expuesta ante un grupo de inversores donde se definiría su potencial real.

17. Conclusiones

17.1. Académicas

Desde el lado académico, el enfoque de proyecto abierto con repositorio público y uso de herramientas de edición "no binarias" han aportado al trabajo:

- Aprovechamiento del tiempo disponible en diferentes lugares para mantener un ritmo constante de trabajo. Esto me ha permitido una mayor concentración y revisión contínua de cada progreso.
- Utilizar una herramienta de generación de documentación (AsciiDoctor) totalmente contraria a los típicos editores como son Word o LibreOffice me ha permitido trabajar sobre el contenido y no sobre la visualización. Su simplicidad en la escritura contrasta con su potencia para generar el mismo contenido en diferentes formatos de forma automática.
- Al uso de git como solución de repositorio se han añadido los extras que proporcionan los servicios como Gitlab o Github, como gestión de incidencias, integración de ramas, ejecución de pipelines automáticos que permiten generar artefactos (documentación, ejecutables, etc). Mediante estos he podido trabajar en diferentes partes del proyecto e integrarlos todos de una forma cómoda.
- La apuesta de realizar el trabajo utilizando una metodología más ágil me ha permitido reducir los riesgos sobre el producto final así como tener una mayor motivación. El compromiso de tener cada dos semanas un valor que añadir me ha obligado a estar totalmente enfocado en el trabajo

17.2. Personales

Si bien es cierto que al inicio del proyecto conocía la mayoría de lenguajes y herramientas que iba a usar, no es menos cierto que tenía cierto temor sobre cómo encajaría todas ellas unido al desconocimiento de las otras.

Así por ejemplo nunca había trabajado con bases de datos de grafos y las primeras semanas las tuve que dedicar a entender su sintaxis, requisitos, etc. En mi opinión fue un acierto que desde el primer momento lo enfocará mediante la creación de unos Test Unitarios orientados a entender cómo definir las relaciones y ejecutar las consultas. Esto me permitió ir ganando confianza y comprensión del sistema.

Por otra parte mi planteamiento inicial sobre cómo manejaría las versiones ha ido evolucionando también. Mientras que al principio pretendía tener una rama por cada sprint a realizar (visión un tanto estática) poco a poco fui adoptando una metodología más orientada al *issue* donde, usando las herramientas de gestión de incidencias, creaba una rama para cada una de ellas sobre la que trabaja e integraba con *master* de forma más contínua.

17.3. Evolución del producto

A lo largo de este proyecto he podido comprobar que la brecha entre la concepción de una idea de negocio y su realización, a un nivel más alla de un prototipo vacío, hoy en día es muy reducida. Disponemos de muchas herramientas, lenguajes y servicios que nos permiten desarrollar un MVP en un par de meses frente a los desarrollos típicos donde el primer hito se lograba tras varios meses de trabajo.

Sin pretender ser la aplicación que revolucione el mercado, en esta fase TelotraigodemiPueblo cuenta, por ejemplo, con un API accesible vía dispositivos móviles por lo que sería muy fácil desarrollar la parte cliente para ellos. Así mismo al implementar un modelo stateless (sin sesión de usuario) permite desde su inicio la escalabilidad horizontal.

18. Agradecimientos

Prácticamente estoy terminando una etapa que inicié hace ya unos años y en la que he tenido la oportunidad de aprender de muchas personas tanto docentes como alumnos y sería injusto nombrar a unos y olvidarse de otros, por lo que creo que me limitaré a mostrar mi gratitud a una sóla persona de ellas con la convicción de que el resto lo entenderán:

Quiero agradecer a **Antonio Cauto**, mi tutor durante todos estos años, su apoyo contínuo en esta andadura. Estoy seguro que sin él y sus consejos, hace tiempo que habría pospuesto esta aventura de forma indefinida.

19. Glosario de términos

MVP

Minimum Viable Product. La versión de un nuevo producto que permite a un equipo recolectar, con el menor esfuerzo posible, la máxima cantidad de conocimiento validado sobre sus potenciales clientes. (https://es.wikipedia.org/wiki/Producto_viable_m%C3%ADnimo)

API

Application Programming Interface. Conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción (https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones)

REST

Transferencia de Estado Representacional Estilo de arquitectura software para sistemas hipermedia distribuidos (https://es.wikipedia.org/wiki/Representational_State_Transfer)

Backlog

Es el conjunto de todos los requisitos de proyecto, el cual contiene descripciones genéricas de funcionalidades deseables (https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)#Product_backlog)

Sprint

El Sprint es el período en el cual se lleva a cabo el trabajo en sí (https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)#Sprint)

Sprint backlog

subconjunto de requisitos que serán desarrollados durante el siguiente sprint (https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)#Sprint_backlog)

Groovy

Lenguaje de programación dinámico para la máquina virtual Java. http://www.groovy-lang.org/

Grails

Framework basado en Groovy que aúna diferentes librerías para el prototipado rápido de aplicaciones https://grails.org/

Grafo (Base de datos)

"Una base de datos orientada a grafos (BDOG) representa la información como nodos de un grafo y sus relaciones con las aristas del mismo ..." https://es.wikipedia.org/wiki/Base_de_datos_orientada_a_grafos

20. Screenshots

A continuación se adjuntan capturas de pantallas de todos los casos contemplados por la aplicación, desde el registro , invitación y aceptación de amistad, mantenimiento de lugares, productos y lugares, así como solicitud de pedido y aceptación del mismo.

Figure 1. Inicio

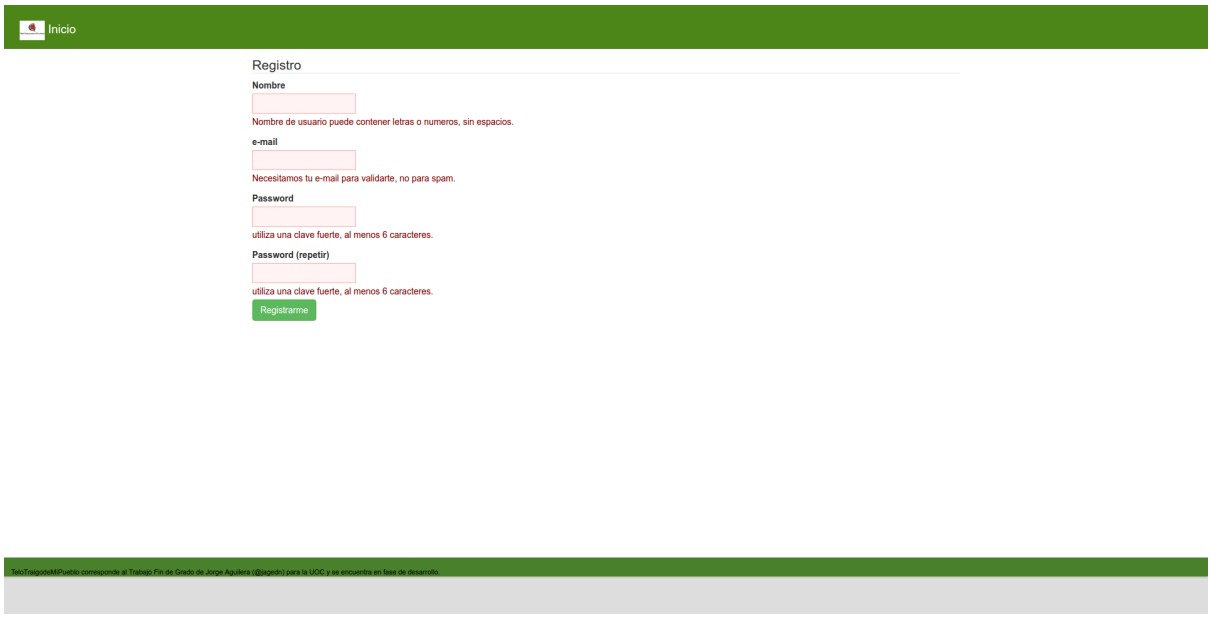


Figure 2. Registro

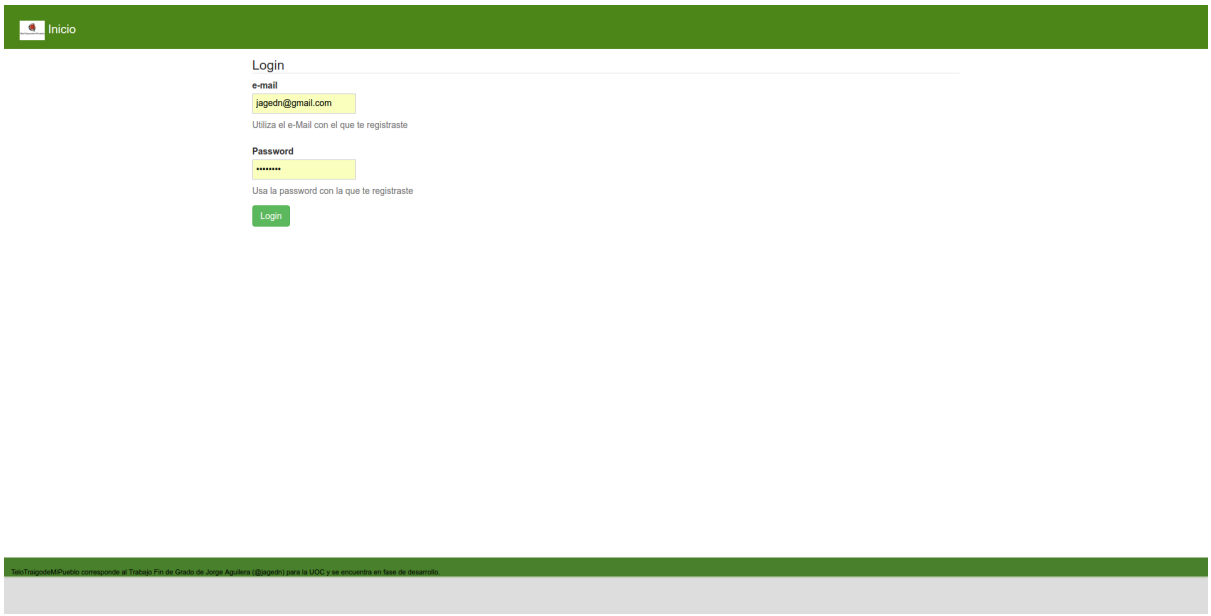


Figure 3. Login

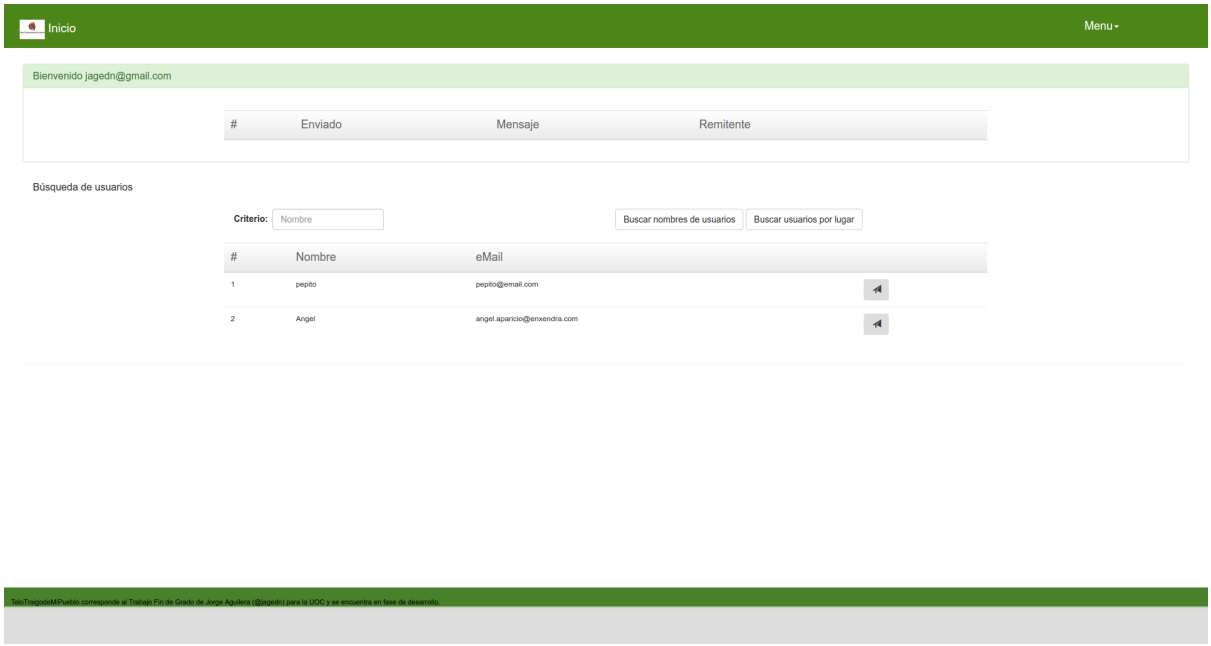


Figure 4. Welcome

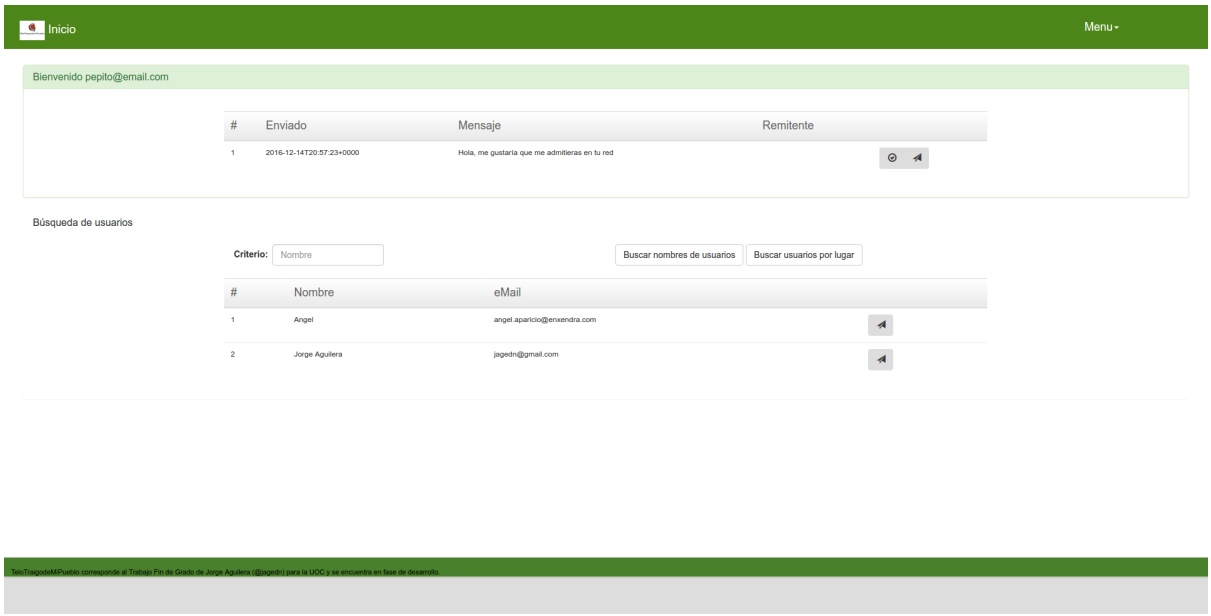


Figure 5. Nuevo amigo

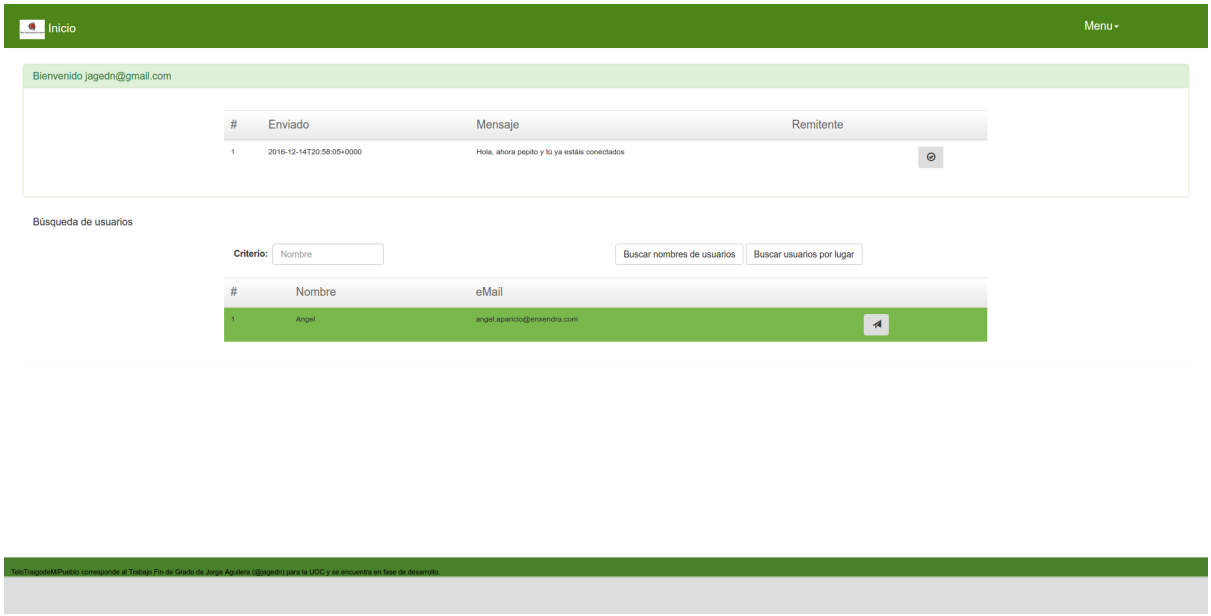


Figure 6. Aceptado

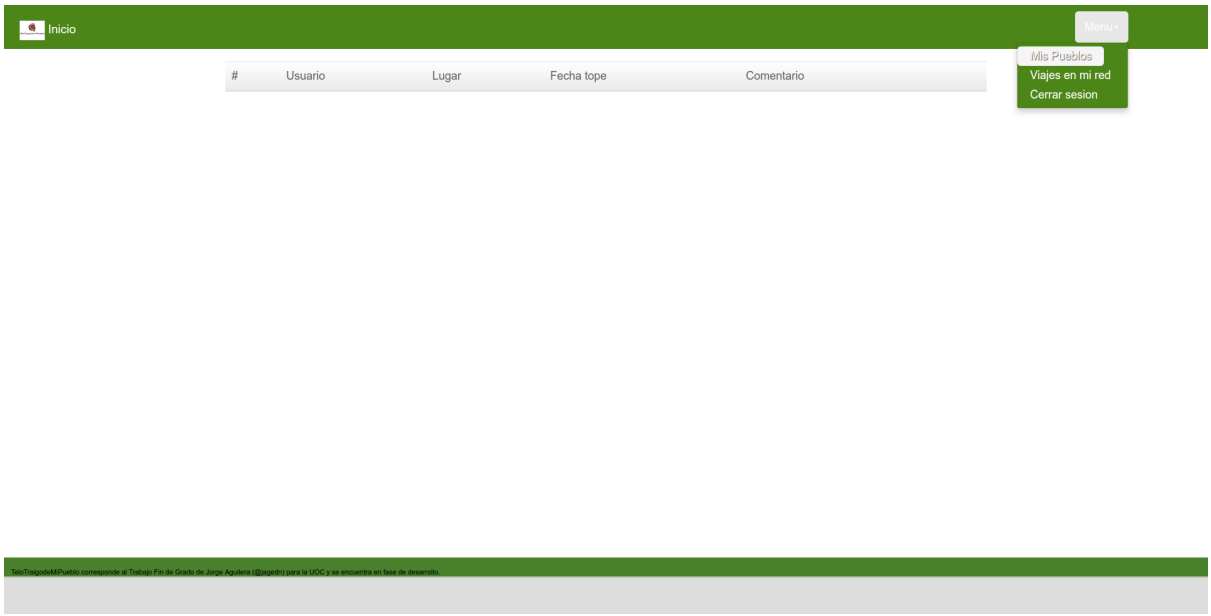


Figure 7. Menú

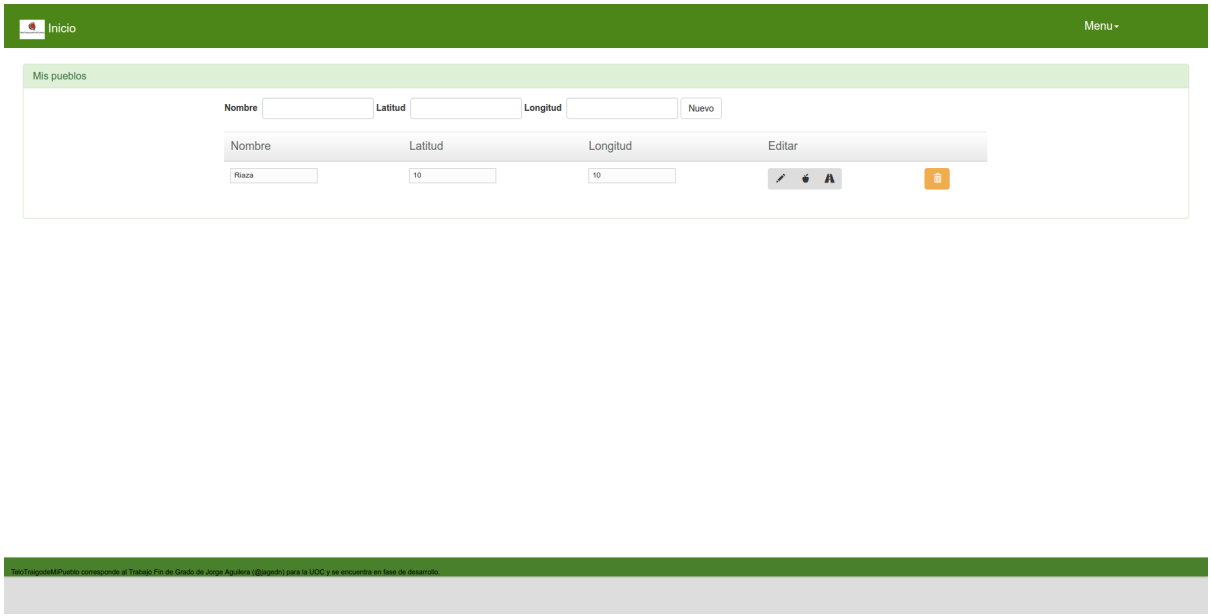


Figure 8. Mis pueblos

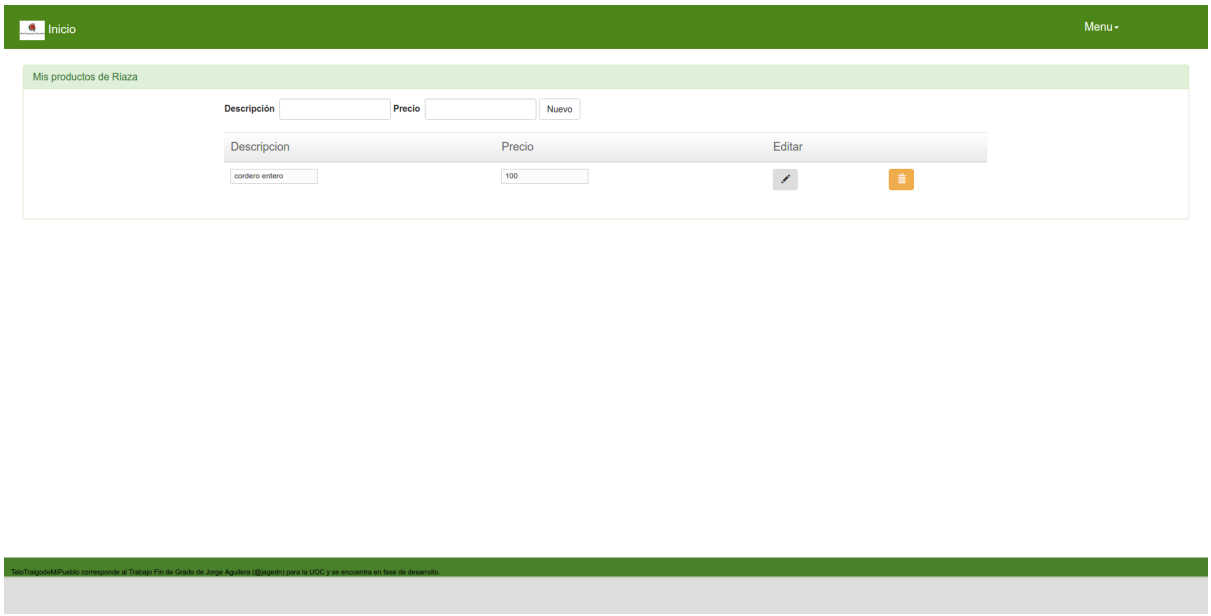


Figure 9. Productos de un pueblo

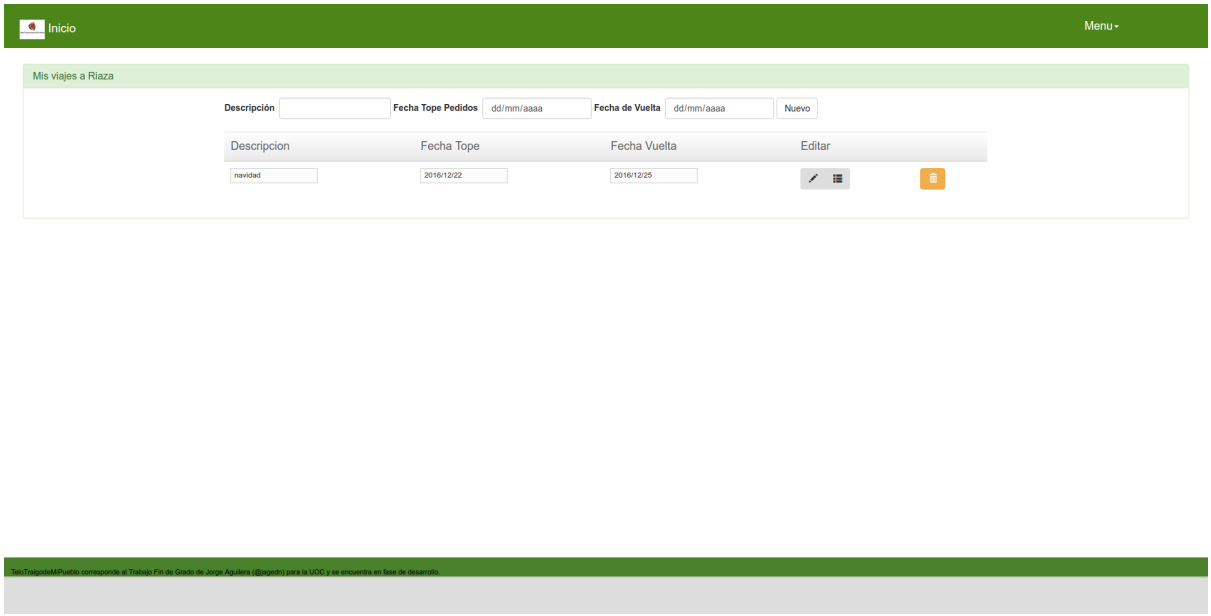


Figure 10. Viajes a un pueblo

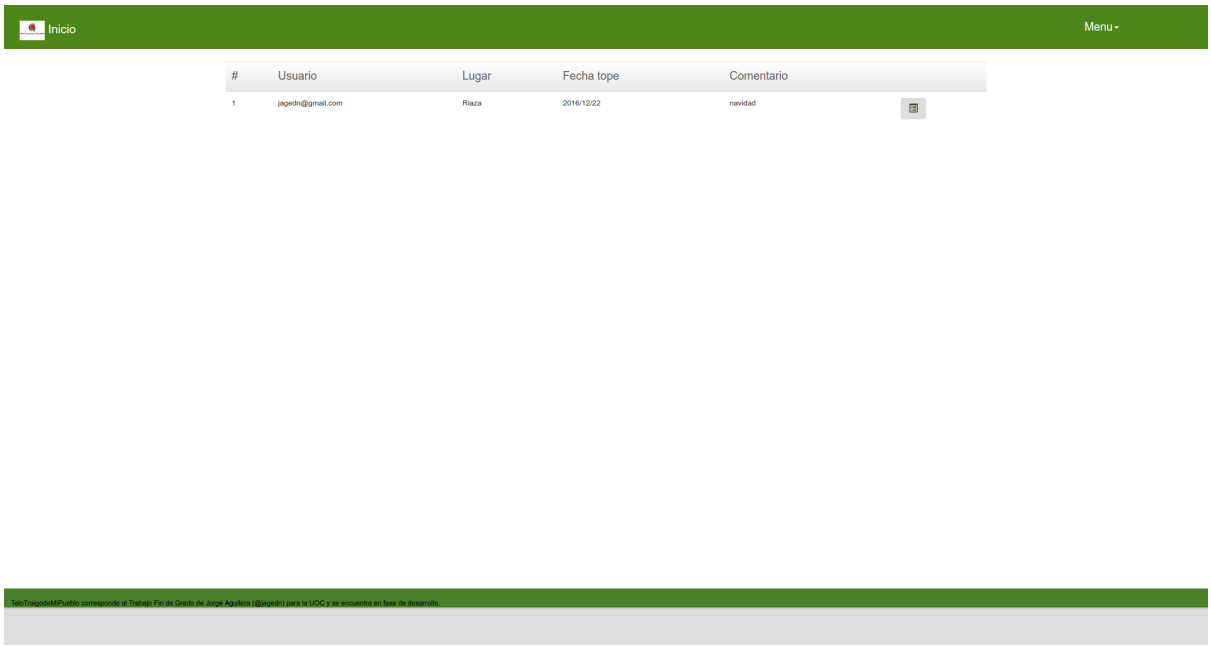


Figure 11. Viajes en mi red

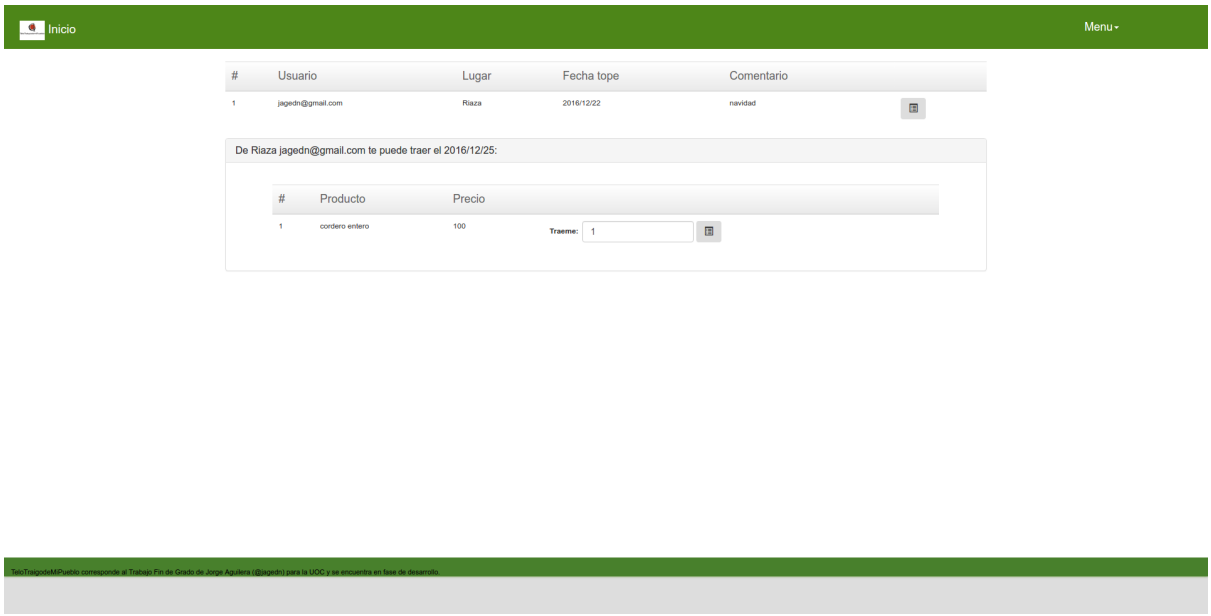


Figure 12. Solicitar pedido

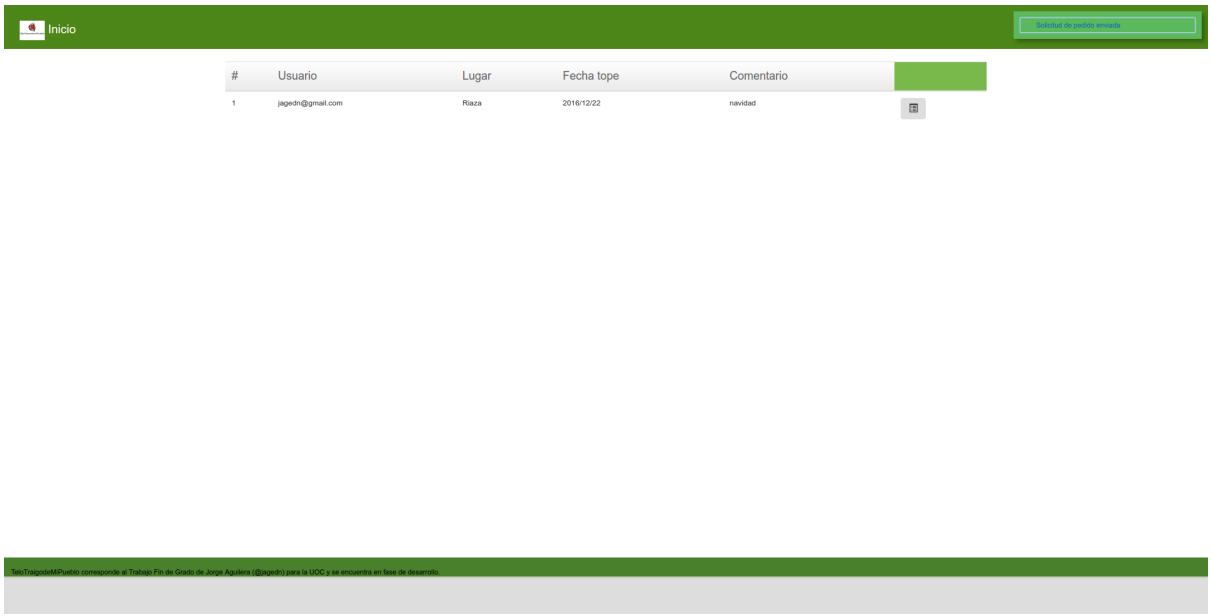


Figure 13. Pedido solicitado

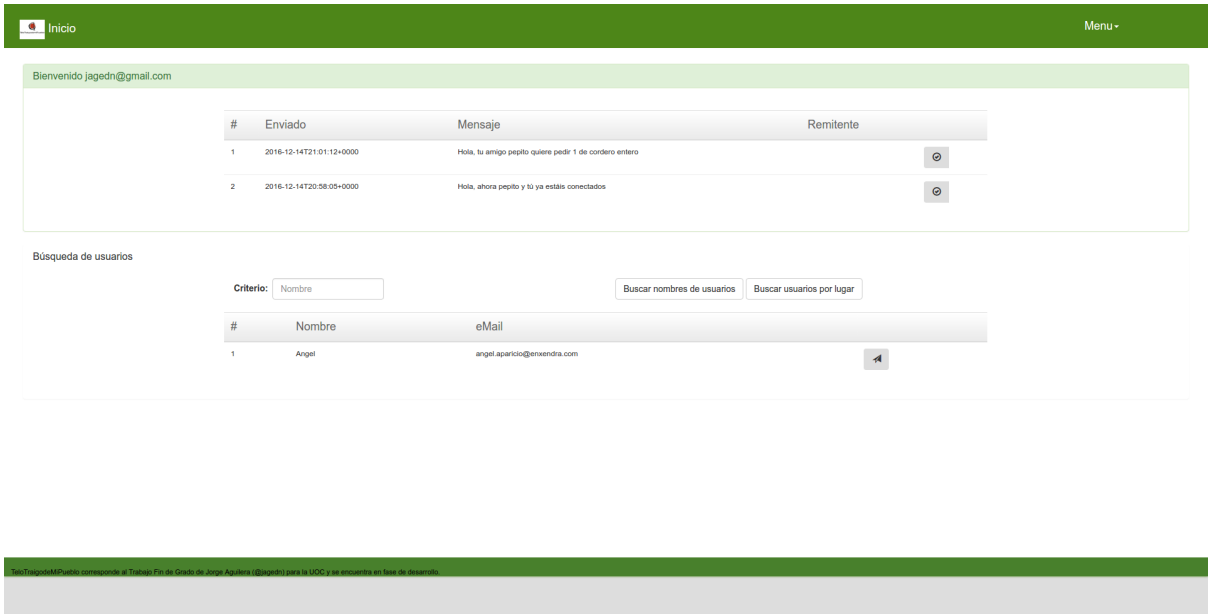


Figure 14. Notificación de pedido

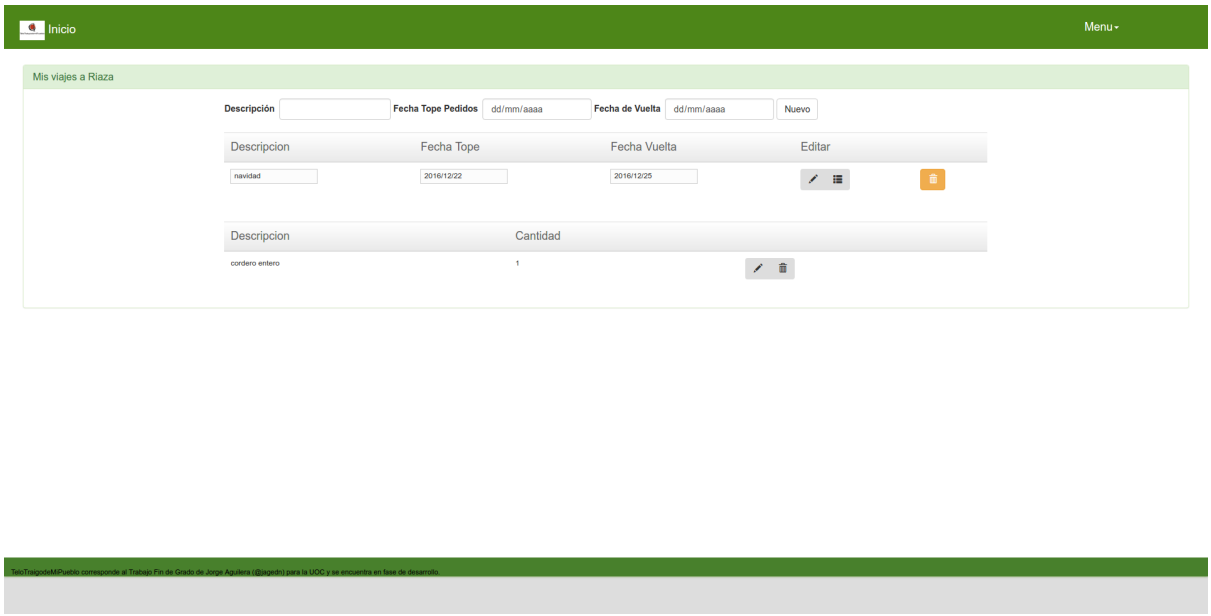


Figure 15. Lista de pedidos