ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

FUNDAMENTOS DE INGENIERÍA DEL SOFTWARE
PARA SISTEMAS CLOUD

ANIMEA

Microservicio Friends

Realizado por
Rafael Fresno Aranda
David Posada Mena

Índice general

Índice de figuras

Índice de cuadros

Parte I

INTRODUCCIÓN

1. Introducción

Vivimos en una sociedad donde cada vez es más frecuente el uso de aplicaciones web para cualquier tipo de interacción con el resto de amigos y/o conocidos. Ya no solo el hecho de conocer nuevas personas sino también para comunicación, compartir gustos, realizar planes conjuntos...

El uso de internet se ha vuelto indispensable a la hora de entablar relación con el resto de personas, concretamente el uso de aplicaciones como GameTree, Meet Me, Tinder es cada vez más extendido entre personas de todas las edades debido a la facilidad para encontrar nuevas personas que conocer cada día y por la posibilidad de poder filtrar entre todas las posibilidades de forma sencilla acudiendo a recursos como los gustos personales, por ejemplo.

Actualmente existen muchas aplicaciones que cumplan este cometido pero no había ninguna enfocada en aquellas personas que sintiesen especial apego al mundo de los animes o el manga.

Es aquí donde nace *Animea*. Una aplicación donde las personas no solo podrán compartir sus diferentes gustos en cuanto a animes sino también poder conocer nuevas personas con sus mismos gustos y, además, poder realizar quedadas con esas personas con un único objetivo en mente, la diversión.

El microservicio *Friends* se encarga de gestionar los amigos de cada usuario, así como las peticiones de amistad. Mediante su API se pueden consultar y borrar amigos, y también obtener, crear, actualizar y borrar peticiones.

2. Estructura del documento

Este documento se divide en seis partes principales:

- Introducción.
- Descomposición en microservicios.
- Aspectos relevantes.
- Requisitos.
- Esfuerzos.
- Lecciones aprendidas.

En la **Introducción**, se expone de forma concisa qué proyecto se ha llevado a cabo, explicando brevemente el por qué de éste.

En la parte de **Descomposición en microservicios** se exponen los diferentes microservicios que componen la base de la aplicación a desarrollar, métodos disponibles, códigos de estado, ...

En la parte de **Aspectos relevantes**, se desarrollan los diferentes aspectos y/o características referentes al proyecto que resulten de interés y que no tengan una categoría definida.

Por último, está la sección de **Lecciones aprendidas**, donde se recogen las conclusiones finales tras el desarrollo del proyecto.

Parte II

ANIMEA

3. Introducción

Animea es un proyecto que ha surgido de la necesidad de miles de personas por tener un lugar concreto al que acudir, donde sentirse cómodos y arropados, acompañados y rodeados de personas con las mismas inquietudes. Un lugar donde *compartir* gustos, *conocer* nuevas amistades, *organizar* actividades y, sobretodo, *divertirse*.

A continuación se recogen las características más destacadas acerca de la base sobre la que se desarrolla toda la aplicación:

- Arquitectura basada en microservicios.
- Express.js para el desarrollo de las API.
- React junto a Materialize para el desarrollo del frontend.
- MongoDB como base de datos, hosteado en la nube utilizando Mongo Atlas.
- Travis CI como servicio de integración continua.
- Heroku para el despliegue de la aplicación.

En la lista anteriormente descrita se recogen los aspectos básicos y fundamentales de la aplicación. No obstante, hay otros aspectos adicionales como el *API Gateway*, que han sido implementados y serán explicados dentro de la sección *Aspectos relevantes* para, de esta forma, mantener una distinción entre el núcleo de la aplicación y el resto de módulos.

4. Descomposición en microservicios

Los microservicios que conforman el núcleo de **Animea** son:

- Animes
- Meetings
- Profile
- Friends

Son los cuatro pilares fundamentales bajo los que se sustenta toda la aplicación y, a continuación, se desarrollarán de forma más extendida cada uno de ellos.

4.1. Animes

Este microservicio es el encargado de gestionar los animes. Se encarga de tareas como: obtener lista de animes, eliminar un anime, guardar un anime... Para la obtención de los animes se ha utilizado una API externa: *Kitsu API*.

La unidad mínima de este microservicio es *anime*.

4.2. Meetings

Este microservicio es el encargado de gestionar los meetings, es decir, las diferentes reuniones o actividades que organicen los propios usuarios. Se encarga de tareas como: obtener lista de meetings, eliminar un meeting, crear un meeting...

La unidad mínima de este microservicio es *meeting*.

4.3. Profile

Este microservicio es el encargado de gestionar los profiles de los diferentes usuarios. Se encarga de la visualización y el mostrado de las diferentes características del perfil de cada uno de los usuarios.

La unidad mínima de este microservicio es *profile*.

4.4. Friends

Este microservicio es el encargado de gestionar los friends de los diferentes usuarios, es decir, los amigos. Se encarga de tareas como: listar la lista de amigos, añadir amigos, eliminar amigos... También gestiona las peticiones de amistad: listarlas, crearlas, borrarlas... Se consume la API de SendGrid para enviar un correo electrónico cuando se crea una nueva petición. Además, el microservicio se comunica con profile y con anime para recibir información sobre usuarios y animes respectivamente. El código del microservicio está disponible en `https://github.com/animea-FIS/animea-friends`.

Las unidades mínimas de este microservicio son *friends* y *request*. El recurso *friends* hace referencia a una lista de amigos, ya que un amigo en concreto es equivalente a una instancia del recurso *profile*. En la base de datos se guardará la siguiente información del recurso friends:

- **userId**: ID único del usuario.
- **friends**: conjunto de números donde cada uno de éstos referencia el ID de usuario de los amigos.

Respecto a request, la información que se guardará es la siguiente:

- **userId**: ID único del usuario que creó la petición.
- **friendId**: ID único del usuario que recibe la petición.
- **message**: mensaje de texto que el creador de la petición puede escribir.

Cabe destacar que aunque se almacenen únicamente identificadores, la API devuelve el perfil completo de cada usuario tanto en las listas de amigos como en las peticiones de amistad.

Los métodos disponibles en este microservicio son:

- GET /users/:id/friends: Obtiene la lista de amigos del usuario con *id* indicado.
  - Códigos de estado
    - 200 OK
    - 401 Unauthorized
    - 404 Not Found
- GET /users/:id/friends/animes: Obtiene la lista de animes de los amigos del usuario cuyo *id* es el indicado.
  - Códigos de estado
    - 200 OK
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found
- DELETE /users/:id/friends: Elimina todos los amigos de la lista de amigos del usuario con *id* indicado.
  - Códigos de estado
    - 204 No content
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found
- DELETE /users/:id/friends/:friendId: Elimina un amigo de la lista de amigos del usuario con *id* indicado.
  - Códigos de estado
    - 204 No content
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found

- GET /users/:id/requests: Obtiene la lista de peticiones de amistad del usuario con *id* indicado.
  - Códigos de estado
    - 200 OK
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found
- GET /users/:id/requests/:reqId: Obtiene la petición de amistad con *id* indicado de la lista de peticiones del usuario con *id* indicado.
  - Códigos de estado
    - 200 OK
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found
- GET /users/:id/requests/:reqId/accept: Acepta la petición de amistad con *id* indicado de la lista de peticiones del usuario con *id* indicado.
  - Códigos de estado
    - 204 No content
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found
- POST /users/:id/requests: El usuario con *id* indicado realiza una nueva petición de amistad.
  - Request
    - Body (Petición de amistad a crear)
  - Códigos de estado
    - 201 Created
    - 400 Bad Request
    - 401 Unauthorized

    - 403 Forbidden
    - 404 Not Found

- PUT /users/:id/requests/:reqId: Modifica una petición de amistad del usuario con *id* indicado.

  - Request
    - Body (Petición de amistad a crear)

  - Códigos de estado
    - 204 No content
    - 400 Bad Request
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found

- DELETE /users/:id/requests: Elimina todas las peticiones de amistad del usuario con *id* indicado.

  - Códigos de estado
    - 204 No content
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found

- DELETE /users/:id/requests/:reqId: Elimina la petición de amistad con *id* indicado de la lista de peticiones de amistad del usuario con *id* indicado.

  - Códigos de estado
    - 204 No content
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found

5. Aspectos relevantes

En este apartado se recogen diferentes aspectos del desarrollo del proyecto, ajenos a los **cuatro** microservicios principales de los que se ha hablado anteriormente, dignos de mención.

5.1. Microservicio Auth

Pese a ser un microservicio como tal, no se ha añadido a la lista de **principales** debido a que es totalmente básico para cualquier aplicación y, por lo tanto, no tiene sentido tenerlo como núcleo dentro de nuestra aplicación.

Este microservicio se encarga de tareas como:

- Login de usuarios.
- Autenticación en las transacciones de los usuarios.

La API correspondiente a este microservicio también se ha desarrollado con *express.js* haciendo uso del estándar **JWT**, es decir **JSON Web Token** para las comunicaciones cliente-servidor.

5.2. API Gateway

De nuevo tenemos otra implementación que, perfectamente, podría considerarse como un microservicio pero que, por razones obvias, tampoco ha sido añadido a la lista de microservicios principales.

En nuestra aplicación, el **API Gateway** actúa como única puerta de comunicación para los clientes que quieran hacer uso de **Animea**. Las principales características del API Gateway desarrollado son:

- Redirección de peticiones a los diferentes microservicios según el contexto.
- Implantación de políticas de seguridad.
- Implantación de políticas de consumo, riesgo...

5.3. Despliegue

El despliegue de todo el sistema se realiza en la plataforma Heroku. Las URL en las que se encuentra disponible *Animea*, y en particular el microservicio de friends, son las siguientes:

- Frontend: `https://animea-frontend.herokuapp.com` (común a toda la aplicación).
- Gateway: `https://animea-gateway.herokuapp.com`.
- Friends: `https://animea-friends.herokuapp.com`, también disponible en `https://animea-gateway.herokuapp.com/friends` a través del gateway.

5.4. Planificación inicial

Para la planificación del proyecto se ha tenido en cuenta el número de integrantes del grupo, *8*, y el número de microservicios a realizar, *4*, es decir, **2** personas por microservicio, con la excepción de profile y auth donde ha participado **1** persona en cada uno. También se han tenido en cuenta el total de días disponibles hasta la fecha de entrega, unos *40* días.

PLANIFICACION INICIAL				
Reparto de Microservicios				
Anime	**Meetings**	**Profile**	**Friends**	**Auth**
Camila Reyes y Jorge Gordo	Alicia Viñas y María Alcoba	Antonio Rodríguez	David Posada y Rafael Fresno	Antonio Brenes
Hitos a conseguir				
20/11	**05/12**	**19/12**	**30/12**	**05/01**
Decidir tema del proyecto a desarrollar	Primer desarrollo de las APIs para desmotración	Desarrollo completo de APIs principales	Finalización frontend	Correcciones finales

Asociar un microservicio por cada dos personas	Desarrollo de API Gateway	Implementar sistema de integración continua	Integrar todos los microservicios en el frontend	Revisión completa del entregable
Elegir tecnologías a usar en el proyecto	Realizar integración entre las diferentes APIs	Desarrollo del circuit breaker	Desarrollo de la documentación	Cierre del proyecto

Cuadro 5.1: Animea - Planificación inicial

Siguiendo con la planificación anterior se esperaba tener una primera muestra de la aplicación ligeramente funcional para el seguimiento del día *19 de diciembre* y la aplicación completamente desarrollada el día *5 de enero*, de forma que se tenga un margen de 2 días con la fecha final de entrega para corregir cualquier tipo de problema o retraso que pueda surgir. Sin embargo, la planificación no pudo seguirse completamente debido a diversos problemas y retrasos, por lo que la finalización del proyecto se aplazó.

6. Requisitos

El nivel de acabado del microservicio de friends es el que opta a obtener **hasta 9 puntos**. Los requisitos básicos que se han cumplido son los siguientes:

- **RB1: El backend debe ser una API REST tal como se ha visto en clase implementando al menos los métodos GET, POST, PUT y DELETE y devolviendo un conjunto de códigos de estado adecuado**. Como se ha comentado en secciones anteriores, el microservicio es una API REST desarrollada en Node.js utilizando el módulo Express. Existe al menos un endpoint para cada uno de los métodos, y se devuelven códigos de estado de acuerdo a las convenciones habituales.
- **RB2: Debe tener un frontend que permita hacer todas las operaciones de la API (este frontend puede ser individual o estar integrado con el resto de frontends)**. Como también se ha comentado en secciones anteriores, existe un frontend común desarrollado en React para toda la aplicación que permite interactuar con todos los endpoints de la API. Cabe mencionar que los parámetros que algunos enpdoints aceptan en la query no se han utilizado, ya que fueron desarrollados con el objetivo de facilitar la depuración del código.
- **RB3: Debe estar desplegado en la nube y ser accesible en una URL**. El microservicio se encuentra disponible en la plataforma Heroku.
- **RB4: La API que gestione el recurso también debe ser accesible en una dirección bien versionada**. Todos los endpoints de la API están precedidos por */api/v1* para indicar que se trata de la primera versión de la aplicación.
- **RB5: Se debe tener un conjunto de ejemplos de uso del API en Postman de todas las operaciones de la API**. Se ha creado una colección con un ejemplo para cada endpoint de la API. Esta colección se ha parametrizado utilizando entornos, de forma que se puede alternar fácilmente entre la aplicación en local y desplegada.

- **RB6: Debe tener persistencia utilizando MongoDB**. El microservicio utiliza una base de datos en la nube, alojada de forma gratuita en Mongo Atlas.
- **RB7: Utilizar gestión del código fuente y mecanismos de integración continua: el código debe estar subido a un repositorio de Github siguiendo Github flow, y debe compilarse y probarse automáticamente usando Travis CI en cada commit**. El código está disponible en un repositorio público, alojado en GitHub en `https://github.com/animea-FIS/animea-friends`. Existe una rama principal, *master*, con el código actualmente desplegado. La rama *develop* sirve para integrar y probar cambios antes de pasarlos a *master*. Desde *develop* se crean ramas para cada característica que se esté desarrollando. Cuando está lista, se hace un merge a *develop*, y, cuando se quiere realizar un nuevo despliegue, se crea una pull request a *master* y si no hay errores se acepta. Cada commit en cualquier rama, así como cada pull request, dispara la ejecución de los tests en Travis y la construcción de la imagen de Docker. Aceptar una pull request (equivalente a hacer un commit en *master* en nuestro caso) dispara además el despliegue automático del microservicio en Heroku. El frontend, disponible en `https://github.com/animea-FIS/animea-frontend`, sigue una dinámica similar.
- **RB8: Debe haber definida una imagen Docker del proyecto**. Existe un Dockerfile que permite crear una imagen del microservicio. Esta imagen se crea automáticamente en cada commit, aunque no se publica en Docker Hub. Se ha publicado manualmente y está disponible en `https://hub.docker.com/r/raffrearaus/animea-friends`.
- **RB9: Debe haber pruebas unitarias implementadas en Javascript para el código del backend utilizando Jest (el usado en los ejercicios) o Mocha y Chai o similar**. Hay una carpeta de tests en el proyecto donde se han implementado diversas pruebas utilizando Jest. Se han probado todos los métodos de la API, y en algunos de ellos se han incluido casos negativos que provocan distintos códigos de estado.
- **RB10: Debe haber pruebas de integración con la base de datos**. En la misma carpeta junto a las pruebas unitarias se encuentran algunas

pruebas de integración. Se prueba que los datos se crean, actualizan y borran. Cabe mencionar que se utilizan colecciones de Mongo distintas a las que se usan en producción, de forma que se pueden borrar al finalizar las pruebas.

- **RB11: Debe tener un mecanismo de autenticación en la API**. Todos los endpoints requieren autenticación mediante JWT. Este token se genera en el microservicio de auth y se debe pasar en una cabecera en cada petición, *x-access-token*. Si no se envía un token o no es válido, se devuelve el código de estado 401. La mayoría de los endpoints requieren además que el usuario logeado sea el propietario del recurso para acceder a él; de lo contrario, se devuelve un código 403.

En cuanto a los requisitos avanzados, se han implementado los siguientes:

- **RA1: Implementación de pruebas unitarias utilizando mocks y/o stubs**. Todas las pruebas unitarias utilizan mocks de la base de datos para evitar que se modifiquen los datos en producción. Para las llamadas a otras APIs no se han implementado mocks.
- **RA2: Consumo de algún API externa (distinta de las de los grupos de práctica)**. Se consume la API de SendGrid, un servicio de envío de correo electrónico. Cada vez que se crea una petición de amistad, se envía un correo al destinatario indicándole el usuario que ha creado la petición y el enlace donde puede aceptarla o rechazarla.
- **RA3: Implementación de cachés o algún mecanismo para optimizar el acceso a datos de otros recursos**. Cuando se devuelve una petición de amistad, se muestran los perfiles completos del usuario que la creó y del destinatario. Esto se traduce en dos peticiones a la API de profile por cada request que se quiera obtener. Para reducir la carga en dicha API, se ha implementado una caché simple, mantenida en memoria, que almacena los perfiles de los usuarios para reutilizarlos y no realizar varias llamadas para un mismo perfil.
- **RA4: Uso de Kubernetes para el despliegue completo del microservicio**. Se ha realizado un depsliegue en local, utilizando *minikube*. Los archivos necesarios se encuentran en una carpeta dentro del proyecto llamada kubernetes.

- **RA5: Añadir validación a los formularios del frontend**. El microservicio de friends tiene únicamente un formulario, la creación de peticiones de amistad, con un único campo, el mensaje. En la propia etiqueta *textarea* de HTML se ha especificado que el mensaje es obligatorio y que no debe tener más de 500 caracteres.
- **RA6: Tener el API REST documentado con swagger**. Se ha documentado la API completa en OpenAPI 3. No se ha detallado la estructura de aquellos recursos que dependen de otros microservicios (profile y anime), ya que contienen una gran cantidad de atributos y además podrían cambiar sin nuestro conocimiento.

Los requisitos avanzados de la aplicación completa que se han desarrollado son:

- **RC1: Interacción completa entre todos los microservicios de la aplicación integrando información**. Los distintos microservicios se comunican entre ellos para obtener la información que necesitan. En nuestro caso, friends consulta anime y profile, mientras que anime también nos consulta a nosotros.
- **RC2: Tener un front end común común que integre los front ends de cada uno de los microservicios**. El frontend, que está disponible en `https://animea-frontend.herokuapp.com`, contiene las interacciones de cada microservicio y es común a todos ellos.
- **RC3: Hacer uso de un API Gateway**. Existe un gateway, desplegado en `https://animea-gateway.herokuapp.com`, que direcciona cada petición a su microservicio correspondiente. Para acceder a los microservicios, se añade su nombre a la URL (por ejemplo, `https://animea-gateway.herokuapp.com/friends`).
- **RC4: Implementación de un mecanismo de autenticación homogéneo para todos los microservicios**. Todos los microservicios utilizan el mismo sistema basado en JWT para autenticar las peticiones.

7. Esfuerzos

En la siguiente tabla se recogen las horas invertidas por cada uno de los miembros del microservicio de friends en cada requisito. Los códigos de los requisitos se corresponden con los de la sección anterior, y se incluye un breve texto para identificarlos sin necesidad de volver a dicha sección. Para el requisito de GitHub y CI se contabilizan solo las horas dedicadas a establecer la dinámica de trabajo.

	Rafael Fresno	**David Posada**
RB1: API REST	10	0
RB2: Frontend	8	0
RB3: Despliegue	2	0
RB4: API versionada	1	0
RB5: Postman	4	0
RB6: Persistencia	2	0
RB7: GitHub y CI	2	0
RB8: Docker	1	0
RB9: Pruebas unitarias	5	0
RB10: Pruebas integración	2	0
RB11: Autenticación	4	0
RA1: Mocks	5	0
RA2: API externa	2	0
RA3: Caché	2	0
RA4: Kubernetes	3	0
RA5: Validación frontend	1	0
RA6: Swagger	3	0
RC1: Interacción microservicios	4	0
RC2: Frontend común	8	0
RC3: API gateway	1	0
RC4: Autenticación homogénea	4	0

8. Lecciones aprendidas

En este capítulo se recogen, de forma breve, comentarios y pensamientos de los diferentes miembros del equipo surgidos durante el desarrollo del proyecto.

En la lista descrita a continuación se recogen las *lecciones aprendidas* de los diferentes miembros del equipo de **Animea**:

- **Realizar planificación más precisa**: Debido a la falta de una planificación detallada, el seguimiento era muy irregular y complicado de llevar a cabo provocando, de esta forma, un atraso en las tareas.
- **Falta de disponibilidad**: Teniendo en cuenta el número de personas del equipo, muchos trabajadores, se ha vuelto bastante complicado poder realizar reuniones en las que todos hayamos podido participar. El uso de aplicaciones como **Doodle Calendar** podría haber sido de ayuda para elegir los días con mejor disponibilidad.
- **Arriesgar es arriesgado**: Puede parecer superfluo pero, más veces de la cuenta tendemos a crearnos obstáculos por el simple hecho de querer abarcar más de lo que podemos o debemos. Concretamente, decidimos usar **React** para la elaboración del frontend cuando ninguno teníamos un conocimiento pleno en la tecnología, algo que nos ha retrasado enormemente.

En cuanto a los miembros del microservicio de friends, las *lecciones aprendidas* en este caso son:

- **Priorizar las tareas**: Dado que la evaluación del proyecto se divide en distintos niveles en los que hay que completar varios requisitos incrementalmente, debíamos haber priorizado tareas concretas para ir alcanzando los niveles de uno en uno. Nos hemos dedicado a trabajar en varias tareas a la vez y sin ningún orden, de forma que incluso trabajamos en tareas opcionales antes de haber terminado todas las obligatorias.
- **Más formación y transferencia de conocimiento**: Algunas de las tecnologías utilizadas en el proyecto eran desconocidas para nosotros.

Sin embargo, otros compañeros sí las conocían, por lo que podría haber sido conveniente dedicar algún día a formarnos en estas tecnologías.