Δchan

Proyecto fin de ciclo
Desarrollo de Aplicaciones Multiplataforma

Alfredo Rodríguez Gracía

21 de mayo de 2021

última revisión
1 de junio de 2021

Índice

1. Descripción del proyecto

Δchan (pronunciado como *dichan*) es un proyecto de tablón de imágenes [1], centrado en el anonimato y la libertad de expresión *on-line*, dónde los usuarios pueden subir imágenes y vídeos cortos para iniciar un debate. Está inspirado en otros tablones existentes como *4chan* y *2channel*, sitios que, a pesar del enorme auge de las redes sociales, siguen siendo el refugio de muchos internautas hoy en día.

Un tablón de imágenes (también conocido por su nombre en inglés: *imageboard*) es un tipo de página web anónima donde la publicación de imágenes y pequeños vídeos cobra una gran importancia. Los primeros tablones de imágenes fueron creados en Japón a finales de los 90, y se basan en el concepto de los foros de texto. En términos generales ambos comparten la misma estructura, incluyendo la separación de los debates (*threads*) de diferentes temáticas en secciones, llamadas tablones o *boards*. Sin embargo, los *threads* en los *imageboards* pueden llegar a ser mucho más esporádicos que en los foros convencionales, donde el tiempo de vida de uno puede ser inferior a varias horas. Los tablones de imágenes más populares en occidente tienden a estar relacionados en su mayoría con la cultura japonesa, como son la temática del *anime* y *manga*. Sin embargo, en Japón son más populares y sus tópicos abarcan una gran variedad de temas.

El proyecto Δchan intenta emular a estos tablones haciendo muy sencillo que cualquiera que lo desee pueda montar su propia instancia en un equipo, incluso con muy pocos recursos. La estructura de la página es muy simple, consta principalmente de dos partes bien diferenciadas: la portada, donde se visualizará la lista de *boards* activos en la página; y los *boards* en sí, cada uno de su temática particular y limitado a nueve páginas de contenido. Cada página de un *board* contendrá cinco *threads* ordenados por fecha de actualización mas reciente, es decir, en el primer puesto de la primera página se colocará el *thread* que ha recibido el último comentario y en el último puesto de la novena página estará el *thread* que ha pasado mas tiempo sin comentarios. En el momento que un usuario decida abrir un nuevo *thread* ese último se borrará y el nuevo aparecerá en el primer puesto. De esta forma se consigue ese dinamismo tán característico de los *imageboards* dónde tienes la certeza de que lo primero que ves al entrar es de lo que se está hablando actualmente, es el tema del momento.

La intención del proyecto mira hacia un futuro colaborativo, donde muchas personas puedan aportar sus opiniones y mejoras al mismo. Este es el motivo por el que se publica bajo la *GNU General Public License version 3* [2], para garantizar que forme parte del movimiento del *software libre* definido por M. Stallman [3]. El código fuente será accesible desde un repositorio *Git*

de libre acceso, donde cualquier persona podrá proponer cambios a través de los procedimientos establecidos. De esta forma también es posible que surjan *forks* [4] donde, utilizando este proyecto como base, cualquiera puede cambiar radicalmente el propósito original y aportar un enfoque completamente diferente.

1.1. Partes del proyecto

El proyecto consta de tres partes muy bien diferenciadas: la **página web**, la cual constituye el *frontend*, desarrollada utilizando *HTML5*, *CSS3* y *ECMAScript 6* (*JavaScript*), es la encargada de lidiar cara a cara con el usuario final, mostrar los resultados de las llamadas a la *API* y distribuir esas respuestas adecuadamente en la interfaz gráfica (*GUI*); la ***API***, una de las dos partes del *backend*, desarrollada utilizando el lenguaje *Rust* para garantizar una mayor eficiencia en la respuesta, es la intermediaria entre la web y los datos almacenados, ofreciendo un estándar a la hora de consultar y modificar la información ofrecida por los usuarios; y la segunda parte del *backend* es la **base de datos** que, corriendo sobre un servidor *MariaDB*, almacenará toda la información de la aplicación en tablas relacionadas para poder acceder a ella de la manera mas eficiente posible, garantizando siempre la integridad y la alta disponibilidad de los datos. Observando la figura 1 se puede observar de forma gráfica cómo se relacionan entre sí estas tres partes.

Figura 1: Relaciones entre las partes del proyecto

Webpage $\longleftrightarrow$ API $\longleftrightarrow$ Database

La grán ventaja de utilizar un *API* en el *backend* es que, de esta forma se abre la posibilidad a la aparición de clientes programados por terceros que interactúen con los datos a través de un interfaz común para todos ellos. Haciendo muy portable el proyecto hacia nuevos entornos, como por ejemplo, en forma de app movil, aplicación de escritório u otras interfaces web. Asimismo, el hecho de independizar por completo el cliente del servidor permite que futuras versiones puedan prescindir de alguna de las partes con mucha facilidad y re-implementarla al gusto, facilitando al máximo la re-utilización del código.

2. Ámbito de implantación

Δchan será administrado por una organización sin ánimo de lucro (*The*

Δ*chan Foundation*), creada específicamente con la finalidad de asegurar la libertad, independencia y neutralidad del proyecto, así como mantenerlo ajeno a todo interés económico, político y personal que pueda derivar de una administración centrada en una única o un reducido número de personas. Imitando así modelos como los implementados en *Wikimedia* [5] o *The Internet Archive* [6]. Esta será la encargada tanto de la parte financiera, administrando las donaciones que se puedan recibir por parte de los usuarios, como la parte tecnológica, ofreciendo al proyecto de la infraestructura física y el mantenimiento necesario para asegurar su correcto funcionamiento.

El objetivo final pasa por establecer un lugar de debate e intercambio de opiniones abierto, libre y neutral; donde cada usuario pueda encontrar su rincón y hablar de lo que le apetezca con personas que comparten sus mismos gustos y aficiones. Y al mismo tiempo ofrecer espacios donde posiciones opuestas se enfrenten para que eventualmente se alcance un consenso común y beneficioso para ambas partes.

Este proyecto va dirigido a usuarios que les importan mas las ideas –el debate en sí– y no las personas que sostienen esas ideas, gente de todas las edades que quiera compartir sus opiniones y experiencias con una gran comunidad de distribuida por todo el mundo. El anonimato hace que la gente se pueda expresar sin tapujos y sin esperar consecuencias en el ámbito personal, instigadas por lo que uno piensa.

3. Recursos de *hardware* y *software*

Se describirán los requisitos mínimos y los requisitos recomendados de *hardware*, tanto para el desarrollo de la aplicación, como para su instalación y ejecución.

Se describirán las necesidades de *software* requeridas para el desarrollo de la aplicación.

Puesto que Δchan es un *aplicación web* existen tres escenarios a tratar con respecto a los requisitos de *hardware* y software: **desarrollo**, **despliegue** e **instalación por parte del usuario**. Inicialmente se pretende que en cualquier de ellos, estos recursos, sean lo más limitados y gratuitos posibles haciendo que el proyecto pueda ser accesible por cualquiera sin importar la calidad del *hardware* disponible.

3.1. Requisitos de desarrollo

Los requisitos para el desarrollo de este proyecto no son para nada exigentes, todas las partes se pueden montar e interconectar en un mismo equipo con, prácticamente, cualquier especificación de *hardware*. Unos requisitos mínimos podrían ser, un *Intel Core i5* o *i3* con al menos 4 GB de memoria *RAM*, 5 GB de espacio libre en el dísco duro y, como requisito a parte del hardware, es indispensable una conexión estable a internet en el puesto de trabajo.

Además del *hardware*, los siguientes programas son necesarios para poder montar un entorno de desarrollo adecuado y funcional: un servidor de base de datos *MariaDB* o *MySQL*, para poder ejecutar consultas *SQL* contra él en el entorno local, emulando de la forma mas directa como se realizarán una vez se haya desplegado toda la infraestructura; un servidor *HTTP*, como puede ser *NGINX* o *Apache2*, para poder simular un entorno similar al servidor donde se va a desplegar la web al final del desarrollo. Por otra parte, se recomiendan algunos programas opcionales que quedan a elección del desarrollador, com puede ser, un editor de código adaptado para las diferentes tecnologías y lenguajes utilizados en el proyecto (*Visual Studio Code OSS*, *Atom*, *Geany*...); un cliente de bases de datos como *DBeaver* o *MySQL Workbench*, para administrar de manera mas sencilla la base de datos en las fases mas tempranas del desarrollo. Y como apunte final sería aconsejable realizar todo el proceso de desarrollo sobre una distribución de *GUN/Linux* para simplificar al máximo la portabilidad al entorno del servidor cuando el proyecto se pase a fase de *producción*.

3.2. Requisitos de despliegue

Algunos mas. (mínimos y recomendados)

Tanto el diseño como el desarrollo de la base de datos se realizarán pensando en un servidor *MariaDB*, dejando así la puerta abierta a una mayor compatibilidad con *MySQL* y otras tecnologías similares. Estará pensada para ser lo mas ligera posible, permitiendo que pueda ser alojada en el mismo equipo que el servidor *HTTP*. Pero siempre permitiendo que pueda instalarse en un equipo diferente, no cerrando la puerta a incrementar las capacidades de la instancia para soportar mayores cantidades de tráfico.

3.3. Requisitos de instalación por parte del usuario

Al ser una aplicación web no requiere de una instalación, como tal, en el equipo del usuario final. Pero si que necesita de unos requisitos mínimos que vienen definidos por las tecnologías utilizadas en la parte del *frontend*.

Por ejemplo, el usuario necesita tener instalado en su equipo por lo menos un navegador web que sea capaz de soportar código *ECMAScript 6* para que el *JavaScript* pueda funcionar sin ningún impedimento. Los navegadores mas modernos con motores como *Gecko* (*Mozilla Firefox*) o *Webkit* (*Google Chrome*, *Opera*, *Brave*...) son perfectamente compatibles con esta reciente tecnología.

Con respecto a los requisitos *hardware*, no se requiere nada fuera de lo común, se considera que hoy en día todo dispositivo (*PC* o *smartphone*) posee las capacidades básicas para abrir un navegador, acceder a una página web y leer texto, visualizar imágenes o pequeños vídeos.

4. Temporalización del desarrollo

Lista de tareas a realizar y cuanto tiempo conlleva cada una:

1. (2d[1]) *Setup* del entorno de desarrollo (*hardware* y *software*).
2. (5d) Diseño de *mockups* para las pantallas de la web.
3. (5d) Definición de los *endpoints* de la *API* con un *swagger*.
4. (5d) Implementación de la base de datos (*DDL*).
5. (3d) Desarrollo de la conexión de la *API* con la base de datos.
6. (4d) Implementación de los *endpoints* de la *API*.
7. (2d) Desarrollo de la estructura y estilos básicos de la web.
8. (3d) Desarrollo de la conexión de la web con la *API*.
9. (3d) Finalización de los estilos en la web.
10. (5d) Setup del entorno del servidor (*hardware* y *software*).
11. (5d) Despliegue de la aplicación en el entorno del servidor.

Diagrama de Gantt...
Diagrama PERT...

[1]Unidades de tiempo en días.

5. Descripción de los datos base y resultados

Se describirán el tipo de campo (en caso de java serían: String, char, int, double, long...), que se utilizará para recoger los diferentes datos.

Posibles restricciones y/o estructuras utilizadas (clases). Lo mismo para los datos resultantes de los procesos.

Texto introductorio...

5.1. Página web

Texto introductorio, alguna clase de JS o alguna estructura a de código que se comunique con la API...

5.2. *API*

Definición de los *endpoints* de la *API* basada en el estándar *Open API* versión 3.0.3 [7].

- (GET) `/board` – Devuelve la lista de todos los *boards*.
  - Respuestas
    - `200` – `application/json` – Un array con los *boards* y sus atributos.
- (GET) `/{slug}/thread` – Devuelve la lista de todos los *threads* dentro de un *board*, ordenados desde el mas reciente al mas antiguo.
  - Parámetros
    - `division` – number, query – Número de *threads* que quieres que aparezcan por página.
    - `page` – number, query – Número de página del *board* que quieres ver, en base a la división anterior.
  - Respuestas
    - `200` – `application/json` – Un array con los *threads* y su primer y últimos cinco *posts* (a modo de resumen).
    - `400` – `application/json` – Cuando los *query params* tienen un formato erróneo.
    - `404` – `application/json` – Cuando solicitas un *slug* o una página que no existe.

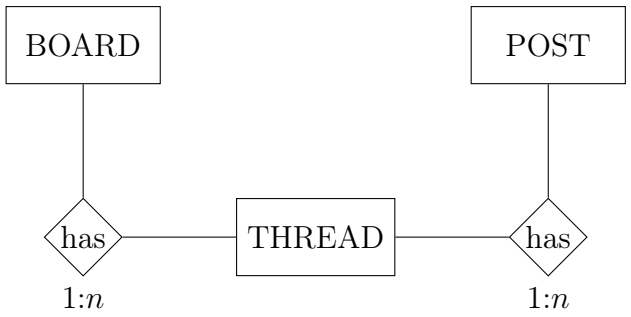
- (POST) `/{slug}/thread` – Permite publicar un nuevo *thread* dentro de un *board* dado.
    - *Request body* (*required*) – `application/json`
        - **subject** – string – El asunto del *thread*.
        - **author** – string – El nombre del autor del *thread*, si se deja en blanco aparecerá *Anonymous*.
        - **comment** (*required*) – string – El texto del primer *post* del *thread*.
        - **fileurl** – string – La *URL* del archivo adjunto al *thread*.
        - **slug** (*required*) – string – El *slug* del *board* dónde se va a publicar el *thread*.
    - Respuestas
        - `201` – `application/json` – El *thread* se ha publicado correctamente.
        - `400` – `application/json` – Cuando el *request body* tiene un formato erróneo.
        - `404` – `application/json` – Cuando se envía un *slug* que no existe.
- (GET) `/{slug}/thread/{id}` – Devuelve la lista de todos los *posts* de un *thread* dado.
    - Respuestas
        - `200` – `application/json` – Un array con los *posts* ordenados de mas antíguo a mas reciente.
        - `404` – `application/json` – Cuando solicitas un *slug* o un *thread* que no existe.
- (POST) `/{slug}/thread/{id}` – Permite publicar un nuevo *post* dentro de un *thread* dado.
    - *Request body* (*required*) – `application/json`
        - **author** – string – El nombre del autor del *post*, si se deja en blanco aparecerá *Anonymous*.
        - **comment** (*required*) – string – El texto del *post*.
        - **fileurl** – string – La *URL* del archivo adjunto al *post*.
        - **thread** (*required*) – number – El número del *thread* dónde se va a publicar el *post*.

- Respuestas
  - `201` – `application/json` – El *post* se ha publicado correctamente.
  - `400` – `application/json` – Cuando el *request body* tiene un formato erróneo.
  - `404` – `application/json` – Cuando se envía un *slug* o un *thread* que no existe.

5.3. Base de datos

El diseño de la base de datos, siempre enfocado hacia una mayor eficiencia y rapidez en la lectura y escritura de los datos, está dividida en tres tablas: *BOARD*, *THREAD* y *POST*. Este diseño intenta compartimentar al máximo las diferentes partes de la página con el mínimo acoplamiento entre la s tablas para evitar hacer *joins* innecesarios, que supondrían una mayor carga para el servidor.

Figura 2: Diagrama Entidad-Relación de la base de datos de Δchan



En el diagrama *ER* de la figura 2 no se han dibujado los atributos de las entidades con el fin de profundizar en ellos a continuación con el mayor detalle posible.

BOARD Cada uno de los tablones que dividen la página en diferentes temas.

- **slug** VARCHAR(4) PRIMARY KEY
- **name** VARCHAR(256) NOT NULL

THREAD Cada uno de los debates, o *hilos* de discusión, que se inician dentro de un tablón.

- **id** BIGINT PRIMARY KEY
- **subject** VARCHAR(256) DEFAULT '' NOT NULL
- **author** VARCHAR(50) DEFAULT 'Anonymous' NOT NULL
- **comment** VARCHAR(512) DEFAULT 'Anonymous' NOT NULL
- **fileurl** VARCHAR(512) DEFAULT NULL
- **timestamp** DATETIME DEFAULT NOW() NOT NULL
- **sticky** BOOLEAN DEFAULT FALSE NOT NULL
- **closed** BOOLEAN DEFAULT FALSE NOT NULL
- **deleted** BOOLEAN DEFAULT FALSE NOT NULL
- **board** VARCHAR(4) FOREIGN KEY REF. BOARD(slug) NOT NULL

POST Cada comentario de un usuario dentro de un debate (*thread*), formado por un texto y una foto o pequeño vídeo opcional.

- **id** BIGINT PRIMARY KEY
- **author** VARCHAR(50) DEFAULT 'Anonymous' NOT NULL
- **comment** VARCHAR(512) DEFAULT 'Anonymous' NOT NULL
- **fileurl** VARCHAR(512) DEFAULT NULL
- **timestamp** DATETIME DEFAULT NOW() NOT NULL
- **deleted** BOOLEAN DEFAULT FALSE NOT NULL
- **thread** BIGINT FOREIGN KEY REF. THREAD(id)

Ejemplo 1: Procedimiento almacenado para la inserción de un nuevo *post*

```
DELIMITER $$
  DROP PROCEDURE IF EXISTS insert_post;$$
  CREATE PROCEDURE insert_post(
    IN new_author VARCHAR(50),
    IN new_comment VARCHAR(512),
    IN new_fileurl VARCHAR(512),
    IN new_thread BIGINT
  ) BEGIN
    SET @last_id = 0;
    SELECT id INTO @last_id FROM POST
      ORDER BY id DESC LIMIT 1;
    IF (@last_id IS NULL) THEN
      SET @last_id = 0;
    END IF;
    INSERT
      INTO POST (id, author, comment, fileurl, thread)
      VALUES (@last_id + 1, new_author, new_comment,
        new_fileurl, new_thread);
  END;$$
DELIMITER ;
```

6. Relación entre dispositivos y programa o rutinas

Se identificarán los componentes que comunican el paquete o aplicación *software* desarrollado con el resto de actores relevantes fuera de la máquina. Es decir, interfaces persona-máquina para entrada y/o salida de datos, interfaces de red u otros medios para comunicación con máquinas remotas, periféricos específicos o componentes concretos de plataformas móviles, etc.

Se identificarán los componentes *software* (clases, procedimientos) representativos y se vincularán con los anteriormente mencionados a través de texto y/o diagrama(s) que ayuden a comprender el funcionamiento general de la aplicación.

---

etc...

Ejemplo: llamada a la API desde JS hasta MySQL.

Ejemplo 2: Obtengo la lista de *threads* que hay en un *board*

```
const getBoardThreads = async (slug) => {
```

```
2    const response = await fetch('https://dchan.org/${slug
       }/threads');
3    const boarThreadsJSON = await response.json();
4  }
```

Ejemplos de código

Referencias

[1] Wikipedia, *Imageboard*, n.d. dirección: `https://en.wikipedia.org/wiki/Imageboard` (visitado 21-05-2021).

[2] F. S. F. Inc., *GNU General Public License version 3*, 2007. dirección: `https://www.gnu.org/licenses/gpl-3.0.html` (visitado 23-05-2021).

[3] R. M. Stallman, «3. La definición del software libre,» en *Software libre para una sociedad libre*, Traficantes De Sueños, 2004. dirección: `https://www.gnu.org/philosophy/fsfs/free_software2.es.pdf` (visitado 23-05-2021).

[4] Wikipedia, *Fork (software)*, n.d. dirección: `https://en.wikipedia.org/wiki/Fork_(software_development)` (visitado 21-05-2021).

[5] T. W. Foundation, *About the Wikimedia Foundation*, n.d. dirección: `https://wikimediafoundation.org/about/` (visitado 29-05-2021).

[6] T. I. Archive, *About the Internet Archive*, n.d. dirección: `https://archive.org/about/` (visitado 29-05-2021).

[7] S. Software, *OpenAPI Specification*, 20 de feb. de 2020. dirección: `https://swagger.io/specification/` (visitado 30-05-2021).