

GGEZdb

Proxecto DAW 2020

José Antonio Alonso Chamorro

ÍNDICE

INTRODUCCION.....	5
ENTORNO DE DESARROLLO.....	6
BACKEND.....	6
INSTALACIÓN DEL ENTORNO LARAGON Y LARAVEL.....	7
FRONTEND.....	8
CONTROL DE VERSIONES.....	8
MANUAL DE USUARIO.....	9
API DE TWITCH.....	9
ESTRUCTURA DE LA APLICACIÓN.....	11
CONTROLADORES.....	11
CONTROLADORES EN LA APLICACIÓN.....	11
VISTAS.....	12
VISTAS EN LA APLICACIÓN.....	13
RUTAS.....	13
CASOS DE USO.....	14
BASES DE DATOS.....	24
DESPLIEGUE	25
PRESUPUESTO.....	28
MEJORAS.....	29
APÉNDICE I. Estructura de carpetas de Laravel.....	30

INTRODUCCION

El proyecto GGEZDB busca ofrecer un lugar donde los aficionados al mundo del videojuego puedan acceder a una página web en la que puedan consultar cualquier detalle de sus juegos favoritos, descubrir, clasificar y calificar títulos según sus preferencias, hábitos de juego, popularidad y muchas otras opciones, principalmente dedicado al público de habla hispana.

Mediante un registro en la página, los usuarios podrán acceder a una variedad de utilidades, entre las que se encuentran, pero no se reducen a: establecer relaciones entre otros usuarios, crear y compartir listas personalizadas, asignar una calificación a los juegos asociados a su cuenta o consultar los juegos más populares en Twitch.tv.

Podemos resumir GGEZDB como una página web a través de la que se pueden realizar consultas de todo tipo sobre una base de datos, que ofrece un sistema de registro de usuario, a partir del cual se podrá acceder a la mayor parte de las funcionalidades de la web. No es necesaria ninguna instalación, y cualquiera con acceso a internet y un navegador podrá disfrutar de todo lo que la web ofrece. En el servidor se alojarán las herramientas con las que se desplegará el proyecto.

La integración de la aplicación con la API de Twitch nos permite acceder a la base de datos IGDB, que cuenta con cientos de miles de títulos, y que nos permitirá disfrutar de una base de datos completa desde el comienzo del desarrollo.

El mundo de los videojuegos está en auge comercial. Su facturación supera a la de la industria del cine y la música *combinadas*. Es un sector que levanta pasiones, y que se beneficia especialmente de las particularidades de la red, sea a través de *streams*, foros de opinión, contenido de vídeo, etc. Si bien ya existen medios que recopilan la opinión de usuarios y/o prensa, no existe un portal en español que englobe este tipo de tráfico, y el fin de esta aplicación es captar ese sector del público.

ENTORNO DE DESARROLLO

Para este proyecto se ha elegido el entorno de desarrollo local Laragon. Laragon es un entorno de desarrollo universal para **PHP** y **Node.js** entre otros, y que ofrece muchas funcionalidades, destacando en este caso las que agilizan el proceso de desarrollo, como son su ligereza (menos de 4MB de RAM), la portabilidad de la carpeta del proyecto¹, disponer de un entorno aislado de modo que no afecte al sistema operativo² y la velocidad de ejecución de los servicios del entorno³.

Además, el paquete incorpora MySQL, el gestor bases de datos HeidiSQL, el servidor web Apache y la terminal de comandos **Cmder**. Otra característica de Laragon es que muchas funcionalidades vienen auto configuradas, lo que permite comenzar el desarrollo al momento.

También incluye el gestor de dependencias **Composer**, que permite declarar las librerías de las cual depende el proyecto, instalándolas y actualizándolas por su cuenta y el sistema de gestión de paquetes **npm** (por defecto, viene con Node.js). Cuenta también con utilidades para la creación de proyectos, cambiar la versión de PHP, instalación de nuevos hosts virtuales o aplicaciones como **WordPress** o **Laravel**, el cual emplearemos en este proyecto.

BACKEND

Para el backend, se ha empleado el framework Laravel, que ofrece entre varias características que simplifican la realización de tareas comunes como la gestión de sesiones, la autenticación de usuarios o los enrutamientos. Algunas de sus características destacables son:

- Un interface para la interacción con las bases de datos. Este constructor de consultas está encargado de generar cualquier consulta a la base de datos, ya sea para traer, insertar, actualizar, o eliminar datos. También usa la vinculación de parámetros de PDO⁴ (PDOStatement::bindParam) para proteger la inyección de SQL externas.
- El motor de gestión de plantillas Blade: permite crear vistas con facilidad estableciendo diferentes secciones (@section) y el uso de plantillas desde las que pueden heredar otras vistas. Con Blade, las vistas se compilan en PHP plano y se almacenan en caché hasta que son modificadas, con lo que se muestran con gran velocidad.
- Un sistema de migraciones de bases de datos que permite control de versiones y almacenamiento de copias de los esquemas de la base de datos, para poder modificarlas sin riesgo de pérdida de datos.

1 <https://laragon.org/docs/directory-structure.html>

2 <https://laragon.org/docs/isolated.html>

3 <https://laragon.org/docs/fast.html>

4 **PDO** significa **PHP Data Objects, Objetos de Datos de PHP**, una extensión para acceder a bases de datos. PDO permite acceder a diferentes **sistemas de bases de datos** con un controlador específico (**MySQL, SQLite, Oracle...**) mediante el cual se conecta. Independientemente del sistema utilizado, **se emplearán siempre los mismos métodos**, lo que hace que cambiar de uno a otro resulte más sencillo.

- El intérprete de línea de comandos (CLI) Artisan, que simplifica muchas de las tareas rutinarias introduciendo texto desde el terminal, como la creación de controladores, consulta de rutas o migraciones.

INSTALACIÓN DEL ENTORNO LARAGON Y LARAVEL

Para instalar Laragon, simplemente descargamos el paquete de su web y ejecutamos la aplicación. Laragon es muy sencillo de instalar y no es necesario más que elegir la ruta de instalación y seguir las instrucciones. Por defecto, Laragon se instala en [C:\Laragon](#). Como comentábamos anteriormente, Laragon es portable, está aislado y contenido, con lo que se puede cambiar la ruta cuando queramos, llevárnosla a otro equipo, etc.

En la ruta de instalación, crea una estructura de carpetas, de las cuales las únicas imprescindibles son:

data\	Directorio de datos de MySQL, MariaDB, PostgreSQL y MongoDB. Si este directorio no existe, Laragon se encargará de crearlo y de crear datos para cada uno de estos servicios.
www\	Directorio raíz (donde se almacenan los proyectos). Laragon tiene 'auto creación de hosts virtuales', y al crear una carpeta 'proyecto' en www\ se puede acceder a este proyecto con la url: http://proyecto.test
usr\	Directorio del usuario (de esta manera si se reinstala o actualiza Laragon, se mantendrán sus ajustes).

Cualquiera de las otras carpetas es prescindible. Por ejemplo, si usamos el servidor web Apache, podemos borrar la carpeta donde se encuentra el servidor web nginx (bin\nginx) y la aplicación seguiría funcionando perfectamente.

El próximo paso sería crear nuestro proyecto Laravel, y lo haremos desde la terminal, mediante composer ejecutando el siguiente comando desde la ruta laragon\www:

```
composer create-project --prefer-dist laravel/laravel [nombre-app]
```

Y ya podremos acceder a nuestro proyecto accediendo desde un navegador a la url <https://nombre-app.test>.

FRONTEND

Para el código frontend se ha empleado Javascript, y se emplea el *scaffolding* [Jetstream](#) para Laravel. Jetstream proporciona varias utilidades ya programadas entre las que se incluyen el login, registro, verificación de emails, autenticación de dos factores o manejo de sesiones.

Concretamente, se ha usado el *scaffolding* [Inertia](#), que usa Vue.js como su lenguaje de plantillas. La principal ventaja de Inertia es que permite renderizar los componentes del backend de manera sencilla y sin necesidad de recargar las páginas completas. Para esto se emplea Vue, mientras que seguimos usando los enrutados, controladores y middleware de Laravel.

Jetstream construye la aplicación integrándola con el framework Tailwind CSS, aunque permite trabajar con otros, como PurgeCSS.

Para compilar, disponemos de los comandos npm incluidos en el archivo package.json:

npm run dev	lanza la aplicación, permitiendo abrirla en el navegador. Se usa durante el desarrollo local.
npm run prod	minimiza los archivos js/css, de manera que ocupen menos memoria y sean más rápidos a la hora de acceder a ellos. Se utiliza a la hora de desplegar la aplicación.
npm run watch	permite aplicar los cambios 'en caliente', sin necesidad de recompilar la aplicación. Imprescindible durante el desarrollo.

CONTROL DE VERSIONES

El desarrollo se realiza en un entorno local, y se emplea GitHub como repositorio remoto para el control de versiones. El código del proyecto se puede consultar en:

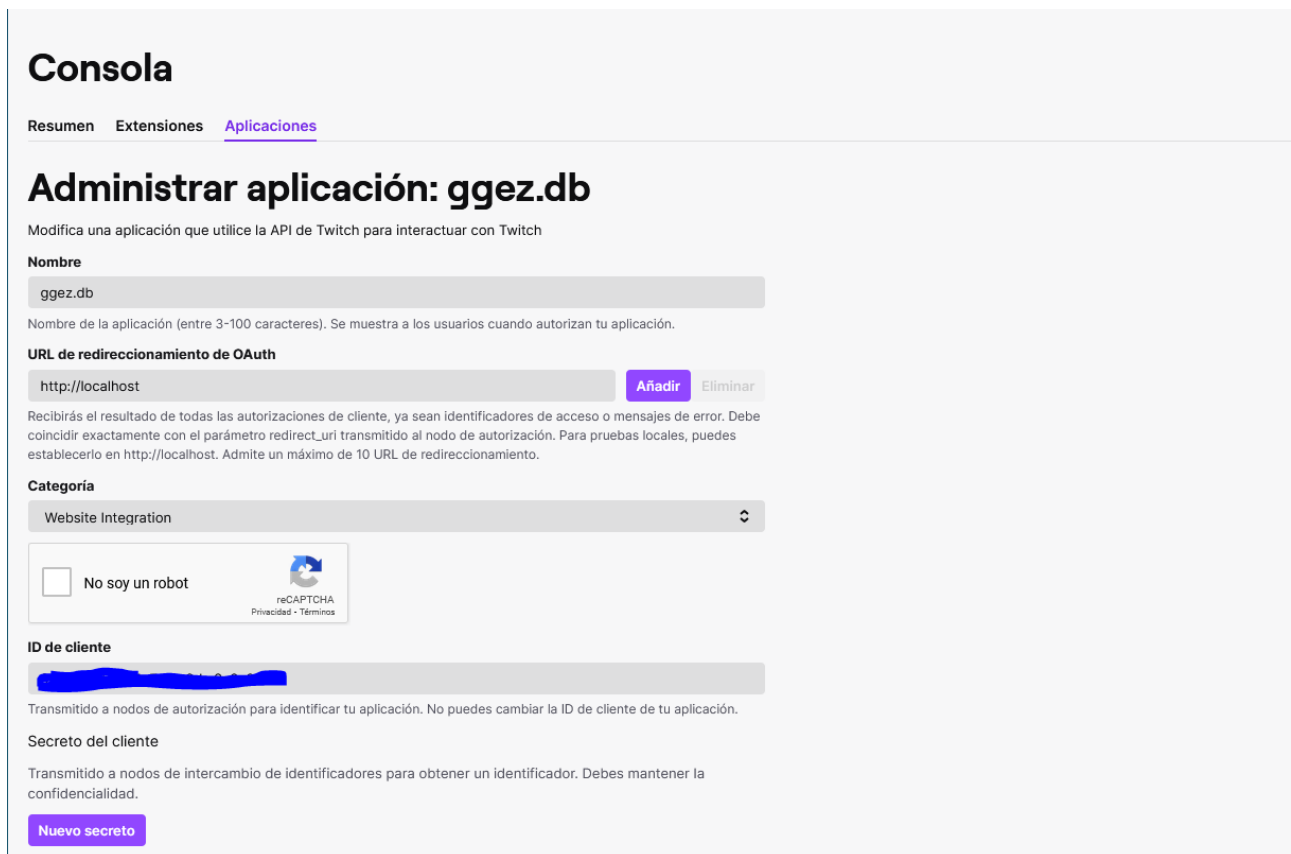
<https://github.com/DregoZ/ggezdb>

MANUAL DE USUARIO

API DE TWITCH

La API de Twitch proporciona las herramientas necesarias para el desarrollo de aplicaciones integradas con Twitch. En nuestro caso, esto nos permite hacer peticiones a su base de datos, que serán procesadas por nuestra aplicación.

Para ello, **es preciso inscribir nuestra aplicación en la consola de desarrolladores de Twitch**, donde se nos asignará un ID Cliente y un Cliente Secreto. Estos datos nos servirán para recibir un Token de autorización, que será necesario incluir en la cabecera de cualquier consulta que hagamos a la API.



The screenshot shows the 'Administrar aplicación' page in the Twitch Developer Console. The page title is 'Administrar aplicación: ggez.db'. Below the title, there is a description: 'Modifica una aplicación que utilice la API de Twitch para interactuar con Twitch'. The page is divided into several sections:

- Nombre:** A text input field containing 'ggez.db'. Below it, a note says: 'Nombre de la aplicación (entre 3-100 caracteres). Se muestra a los usuarios cuando autorizan tu aplicación.'
- URL de redireccionamiento de OAuth:** A text input field containing 'http://localhost'. To the right are 'Añadir' and 'Eliminar' buttons. Below it, a note says: 'Recibirás el resultado de todas las autorizaciones de cliente, ya sean identificadores de acceso o mensajes de error. Debe coincidir exactamente con el parámetro redirect_uri transmitido al nodo de autorización. Para pruebas locales, puedes establecerlo en http://localhost. Admite un máximo de 10 URL de redireccionamiento.'
- Categoría:** A dropdown menu showing 'Website Integration'.
- reCAPTCHA:** A checkbox labeled 'No soy un robot' with a reCAPTCHA logo and links for 'Privacidad' and 'Términos'.
- ID de cliente:** A text input field containing a blue-redacted string. Below it, a note says: 'Transmitido a nodos de autorización para identificar tu aplicación. No puedes cambiar la ID de cliente de tu aplicación.'
- Secreto del cliente:** A text input field containing a blue-redacted string. Below it, a note says: 'Transmitido a nodos de intercambio de identificadores para obtener un identificador. Debes mantener la confidencialidad.' There is a 'Nuevo secreto' button below this field.

Pantallazo de la consola de Twitch Developers

Para no tener que hacer esto, instalamos mediante composer el Wrapper para Laravel [IGDB-Laravel](#). El paquete registrará automáticamente su proveedor de servicios, y permite publicar la configuración mediante comandos de consola. Durante el desarrollo, simplemente incluiremos en el archivo .env las credenciales de desarrollador (Cliente ID y Secreto Cliente) y ya podremos consultar la base de datos de Twitch:

```
TWITCH_CLIENT_ID="nuestro_id_cliente"  
TWITCH_CLIENT_SECRET="nuestro_secreto_cliente"  
TWITCH_REDIRECT_URI="http://localhost"
```

Las consultas a la base de datos de Twitch por medio del Wrapper son sencillas y se basan en modelos predefinidos con los que podremos acceder a diferentes *endpoints* de la API. Su semántica es similar al sistema Eloquent y al Query Builder de Laravel.

Para obtener una lista de todos los juegos simplemente podemos hacer:

```
$todo = Game::all();
```

Pero para acotar un poco la búsqueda, podemos concatenar métodos que nos harán las veces de condición en las queries. Por ejemplo, si queremos acceder a una lista de los 50 primeros juegos ordenados por su calificación:

```
$juegos = Game::orderBy('rating', 'desc')->where('rating', '!=',  
null)->limit(50)->get();
```

o si queremos acceder a la portada de un juego a partir de su `game_id`:

```
$portada = Cover::where('id', $game_id)->get();
```

Una vez recibidos los datos en nuestro *backend*, podemos procesarlos para enviarlos a las vistas, donde se mostrarán maquetados según se defina.

ESTRUCTURA DE LA APLICACIÓN

La aplicación se basa en el modelo vista controlador (MVC). Este estilo de arquitectura separa los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.

Cuando un usuario interactúa con la aplicación (pulsando un botón, enlace, búsqueda...), un controlador se encarga de procesar esa solicitud, accediendo al modelo y actualizándolo si es necesario, para luego mostrar los resultados a través de una de las vistas.

CONTROLADORES

Los controladores determinan la lógica de comportamiento de las peticiones. Al organizar estos comportamientos en una clase Controlador para cada petición (almacenadas en `app/http/Controllers`), podemos mantener un código organizado y separado de los archivos de enrutado, en los que nos interesa solamente tener rutas. Es decir, si queremos pasar una variable a una vista, la declararemos y la manejaremos en su controlador, en lugar de en el archivo de rutas.

Dependiendo de la cantidad de acciones que queramos que ejecute un controlador determinado, podemos definir varios tipos, de los cuales principalmente usaremos:

- **Controladores de única acción:** en estos controladores bastará con crear una función `__invoke()` para manejar el comportamiento. Así, a la hora de determinar la ruta para un controlador de única acción no será necesario especificar el método a ejecutar. Crearemos estos controladores añadiendo la opción `-invokable` al comando de creación.
- **Controladores de recursos:** estos controladores asignan las rutas típicas de los procedimientos 'CRUD' con una sola línea de código, simplemente añadiendo la opción `-resource` al crear el controlador. Así, el controlador se generará con un método para cada una de las operaciones disponibles, a saber: `index`, `create`, `store`, `show`, `edit`, `update` y `destroy`.
- **Controladores de recursos API:** para manejar las respuestas JSON que se reciben de una API, empleamos estos controladores. Añadiremos la opción `-api`, y el controlador se creará sin los métodos `create` y `edit`.

CONTROLADORES EN LA APLICACIÓN

Como se comentaba anteriormente, los controladores para la aplicación se almacenarán en `app/http/Controllers`. En ellos se determina el comportamiento de los datos que procesamos, y en

ellos programaremos la lógica de las búsquedas a la base de datos y los datos que se pasan a las diferentes vistas. Los principales controladores de la aplicación son los siguientes:

- **Controlador SearchController**

Realiza varios queries a la API de Twitch en los que se reciben los datos de los juegos buscados a través del formulario de búsqueda de la aplicación. Se pasan a la vista `Queries/Search.vue` los juegos, portadas y plataformas que coincidan con esta búsqueda, además del término de búsqueda introducido.

- **Controlador GameController**

Realiza varios queries a la API de Twitch en los que se recibe los datos de un juego concreto, junto con su portada, además de las plataformas, pantallazos y vídeos asociados al mismo. Estos datos se pasan a la vista `Queries/Games.vue`.

- **Controlador ExplorarController**

Este controlador llama al método `getTopGames()` (definido en el archivo de métodos `app/helper.php`) y los muestra la vista `Explorar.vue`. Esta vista se muestra al logear en la aplicación y es el punto de partida para los usuarios identificados.

VISTAS

Las vistas contienen el HTML servido por la aplicación, y sirven para separar la lógica de los controladores de la presentación.

En Laravel se usa el motor de plantillas Blade, que permite realizar muchas operaciones con los datos, como establecer herencias entre plantillas, definir una plantilla básica o sustituir secciones enteras o parte de ellas por otro contenido. Sin embargo en esta aplicación se emplean plantillas Vue para procesar y mostrar los datos en el *frontend*.

El funcionamiento de una plantilla Vue es similar a Blade: en ambas se reciben los datos desde un controlador, y en ellas se programa el uso que se les da a esos datos.

En `storage/framework/views/` se almacena la compilación de las vistas Blade, de manera que el paso de Blade a php plano solo se ejecuta una vez por cada cambio y no cada vez que se recarga la página. Esto protege también de la inyección de scripts.

Las vistas y componentes de Vue reciben datos de la misma manera que Blade (aunque se procesen de diferente modo), y se almacenan en `resources/js`.

VISTAS EN LA APLICACIÓN

En esta aplicación se manejan varias vistas:

- Search

En esta vista se muestran los juegos que coincidan con el criterio de búsqueda introducido, y se muestran los datos relevantes: título, portada, calificación, resumen y plataformas en las que está disponible.

- Games

Esta vista muestra la información de un título concreto, al que se accede desde los enlaces mostrados (en el título de cada juego) en Search.vue.

- Trending

Esta vista es un componente que se puede incrustar en otras vistas, y que muestra los datos de los juegos más populares en Twitch.tv.

- Explorar

Esta vista recibe el componente Trending y lo muestra.

RUTAS

En Laravel, los archivos de rutas se guardan en `app/routes`. En `web.php` definiremos las rutas de la aplicación, y en `api.php` las rutas para las APIs que empleemos. Estas rutas son cargadas automáticamente por el framework.

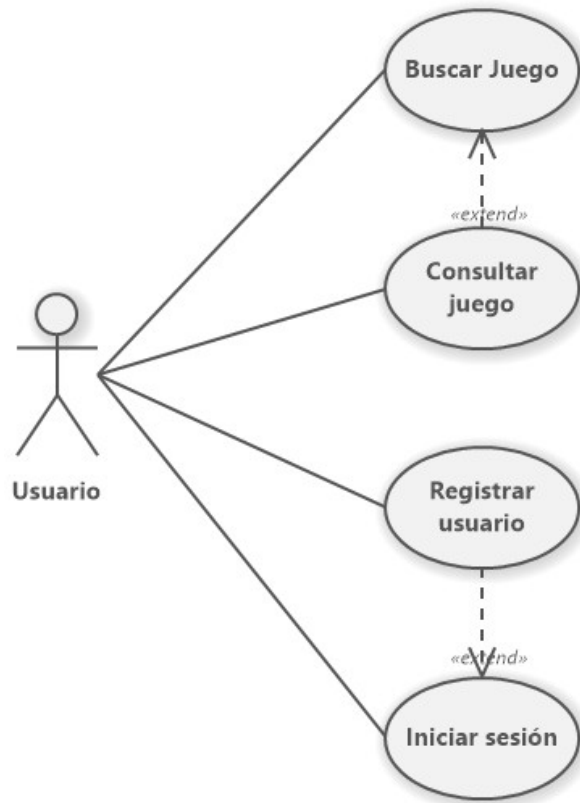
Gracias al sistema de enrutado, podemos definir parámetros en los controladores, o en las mismas rutas, para almacenar datos dinámicos. Si declaramos en un controlador cierta variable y esta se pasa a la vista, podremos referenciar a esta variable desde esa vista.

En este ejemplo, la ruta declarada usa un middleware que verifica si el usuario está autenticado, y de estarlo, lo redirige a la vista 'explorar'.

```
Route::middleware(['auth:sanctum', 'verified'])->get('/explorar',  
ExplorarController::class)->name('explorar');
```

CASOS DE USO

Los casos de uso describen las actividades que el usuario puede llevar a cabo en la aplicación, y sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con dicho usuario o con el mismo sistema. Para nuestra aplicación, se plantean los siguientes casos de uso al acceder un usuario a su página principal:



i. Buscar Juego

El cuadro de búsqueda permite al usuario hacer consultas básicas a la base de datos sin necesidad de identificarse.

ii. Consultar juego

A través de los enlaces mostrados tras el caso buscar juego, el usuario puede acceder a la ficha de un juego en concreto.

iii. Registro

Para usar la mayoría de las funcionalidades de la aplicación, es necesario aportar datos de registro y crear un usuario con el que identificarse.

iv. Iniciar sesión

Se pedirán los datos de inicio de sesión. El inicio de sesión permite acceder a la mayoría de las funcionalidades de la aplicación.

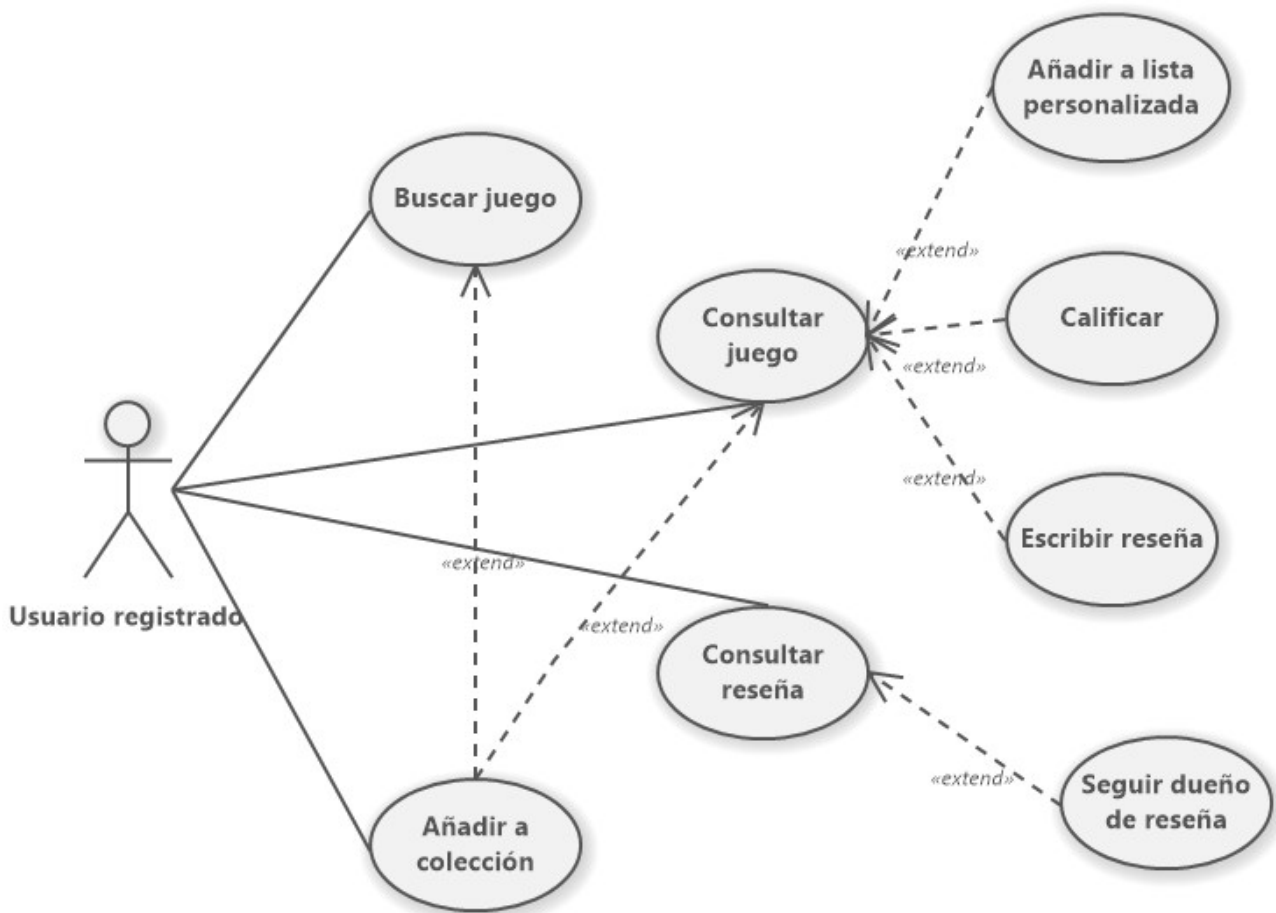
Caso ID	i
Nombre del Caso	Buscar Juego
Actores	Usuario
Descripción	Cualquier usuario puede realizar una búsqueda sencilla por título de juego.
Desencadenante	El usuario escribe algo en el formulario de búsqueda
Condiciones previas	-
Condiciones posteriores	El usuario es redirigido a /explorar
Flujo normal	Se muestra la lista de juegos que coinciden con la búsqueda
Flujo Alternativo	No se muestra ninguna coincidencia con el término buscado

Caso ID	ii
Nombre del Caso	Consultar Juego
Actores	Usuario
Descripción	Accede a la ficha de un juego concreto
Desencadenante	El usuario pincha en un enlace de un juego
Condiciones previas	Debe haberse realizado una búsqueda (caso i) o pinchado un enlace
Condiciones posteriores	El usuario accede a la ficha del juego.
Flujo normal	1. El usuario pincha en un enlace de un juego 2. Se muestra la ficha del juego
Flujo Alternativo	-

Caso ID	iii
Nombre del Caso	Registrar usuario
Actores	Usuario
Descripción	Registra al usuario en la base de datos, lo que le permitirá identificarse en un futuro.
Desencadenante	El usuario pincha el enlace 'Registro'
Condiciones previas	El usuario no debe existir en la base de datos.
Condiciones posteriores	El usuario es creado en la base de datos.
Flujo normal	<ol style="list-style-type: none"> 1. El usuario accede al formulario de registro 2. El usuario introduce los datos requeridos 3. El sistema comprueba que los datos son válidos 4. Crea al usuario en la base de datos.
Flujo Alternativo	<ol style="list-style-type: none"> 3. El sistema comprueba los datos requeridos, y estos no son válidos. 4. El usuario no es creado en la base de datos (datos erróneos o usuario ya registrado).

Caso ID	iv
Nombre del Caso	Iniciar Sesión
Actores	Usuario
Descripción	El inicio de sesión permite acceder a la mayoría de las funcionalidades de la aplicación.
Desencadenante	El usuario introduce sus credenciales de sesión
Condiciones previas	El usuario debe existir en la base de datos y no estar identificado todavía.
Condiciones posteriores	El usuario queda identificado en la aplicación
Flujo normal	<ol style="list-style-type: none"> 1. El usuario introduce sus credenciales 2. El sistema comprueba los credenciales introducidos 3. El usuario tiene acceso a las funcionalidades de la aplicación
Flujo Alternativo	<ol style="list-style-type: none"> 2. Los credenciales son erróneos 3. El usuario no puede acceder a su sesión 4. Vuelve al paso 1

Tras el registro e identificación del usuario, se podrá acceder al resto de funcionalidades de la web. Es a partir de este paso en el que el usuario podrá comenzar a crear listas, ampliar su lista de usuarios seguidos o añadir juegos a su colección. Se presentan los siguientes casos de uso



i. Buscar juegos

A través del cuadro de búsqueda, se puede consultar la base de datos de juegos.

ii. Consultar juego

Se puede consultar un juego directamente de las listas mostradas (habitualmente, los juegos más populares).

O bien se puede consultar un juego que se muestre tras un caso de búsquedas

iii. Consultar reseñas

Se puede acceder a una de las reseñas destacadas (normalmente las mejor valoradas recientemente).

O bien se puede consultar las reseñas específicas de un juego tras el Caso Consultar juego.

iv. Acceder perfil dueño de reseña

Accede al perfil del redactor de la reseña.

v. Seguir dueño de reseña

Incluye al dueño de la reseña en la lista de 'Seguidos'.

vi. Añadir a colección

Añade a la colección personal un juego de las listas mostradas.

O bien, se puede añadir un juego que se consulte tras un caso de Consultar juego.

vii. Añadir a lista personalizada

Desde la página de un juego se puede añadir a una lista personalizada por el usuario (por ejemplo, Favoritos).

viii. Calificar

Desde la página de un juego, se le puede asignar una calificación al mismo.

ix. Escribir reseña

Desde la página de un juego, se le puede añadir una reseña.

x. Valorar reseña

Se puede valorar una reseña de manera positiva o negativa.

Caso ID	i
Nombre del Caso	Buscar Juego
Actores	Usuario
Descripción	Cualquier usuario puede realizar una búsqueda sencilla por título de juego.
Desencadenante	El usuario escribe algo en el formulario de búsqueda
Condiciones previas	-
Condiciones posteriores	El usuario es redirigido a /explorar
Flujo normal	Se muestra la lista de juegos que coinciden con la búsqueda
Flujo Alternativo	No se muestra ninguna coincidencia con el término buscado

Caso ID	ii	
Nombre del Caso	Consultar Juego	
Actores	Usuario	
Descripción	Accede a la ficha de un juego concreto	
Desencadenante	El usuario pincha en un enlace de un juego	
Condiciones previas	Debe haberse realizado una búsqueda (caso i) o pinchado un enlace	
Condiciones posteriores	El usuario accede a la ficha del juego.	
Flujo normal	<ol style="list-style-type: none"> 1. El usuario pincha en un enlace de un juego 2. Se muestra la ficha del juego 	
Flujo Alternativo	-	

Caso ID	iii	
Nombre del Caso	Consultar Reseñas	
Actores	Usuario	
Descripción	Accede a una reseña escrita por un usuario sobre un juego	
Desencadenante	El usuario pincha en un enlace de la reseña	
Condiciones previas	Debe mostrarse una lista de reseñas destacadas o reseñas de un juego concreto	
Condiciones posteriores	El usuario accede a la reseña del juego.	
Flujo normal	<ol style="list-style-type: none"> 1. El usuario pincha en un enlace de reseña 2. Se muestra la reseña del juego 	
Flujo Alternativo	-	

Caso ID	iv
Nombre del Caso	Acceder perfil dueño de reseña
Actores	Usuario
Descripción	Accede al perfil de un usuario del sitio
Desencadenante	El usuario pincha en un enlace de un perfil
Condiciones previas	Debe haberse mostrado un nombre de perfil en una lista de reseñas
Condiciones posteriores	El usuario accede perfil del usuario
Flujo normal	1. El usuario pincha en un enlace de perfil 2. Se muestra la ficha del juego
Flujo Alternativo	-

Caso ID	v
Nombre del Caso	Seguir a otro usuario
Actores	Usuario identificado
Descripción	Seguir a un usuario significa que veremos notificaciones sobre la actividad de los usuarios que seguimos
Desencadenante	Un usuario identificado pincha en un botón de 'Seguir' desde el perfil de otro usuario identificado
Condiciones previas	El usuario debe estar identificado. Debe haberse mostrado un nombre de perfil en una lista de reseñas.
Condiciones posteriores	El usuario seguido está en la lista de Seguidos del usuario 'seguidor'.
Flujo normal	1. El usuario pincha en el botón 'Seguir' 2. El usuario seguido pasa a estar en la tabla 'usuario_sigue_usuario'
Flujo Alternativo	-

Caso ID	vi
Nombre del Caso	Añadir a la colección
Actores	Usuario identificado
Descripción	Añade a la colección del usuario un título determinado, al que debe asignarle un estado (ver tabla 'estado'***)
Desencadenante	Un usuario identificado pulsa el botón 'Añadir a colección' de un juego.
Condiciones previas	El usuario debe estar identificado El juego no puede estar en la colección de ese usuario
Condiciones posteriores	El juego pasa a la colección del usuario, con un estado asignado
Flujo normal	1. El usuario pincha en el botón 'Añadir a colección' 2. El juego y el usuario se almacenan en la tabla 'usuario_colecciona_juego'
Flujo Alternativo	-

Caso ID	vii
Nombre del Caso	Añadir a lista personalizada
Actores	Usuario identificado
Descripción	Los usuarios pueden crear listas personalizadas que engloben diferentes conceptos, como por ejemplo una lista de favoritos.
Desencadenante	Un usuario identificado pincha en el botón de 'Añadir a lista' desde la ficha de un juego.
Condiciones previas	El usuario debe estar identificado. Debe encontrarse en la ficha de un juego.
Condiciones posteriores	El usuario añade un juego a una lista personalizada, existente o nueva
Flujo normal	1. El usuario pincha en el botón 'Añadir a lista' 2. El usuario selecciona una lista de entre las existentes 3. El juego pasa a formar parte de la lista
Flujo Alternativo	2. El usuario crea una lista nueva, la nombra y añade el juego a la lista 3. El juego pasa a formar parte de la nueva lista

Caso ID	viii
Nombre del Caso	Calificar un juego
Actores	Usuario identificado
Descripción	Los usuarios pueden calificar un juego que esté en su colección. Esta calificación se puede cambiar cuando el usuario quiera.
Desencadenante	Un usuario selecciona un valor en el selector 'Califica este juego'
Condiciones previas	El usuario debe estar identificado. Debe encontrarse en la ficha de un juego.
Condiciones posteriores	La calificación del juego se guarda en la base de datos.
Flujo normal	1. El usuario selecciona un valor en el selector 'Califica este juego' 2. Se guarda la calificación en la base de datos
Flujo Alternativo	-

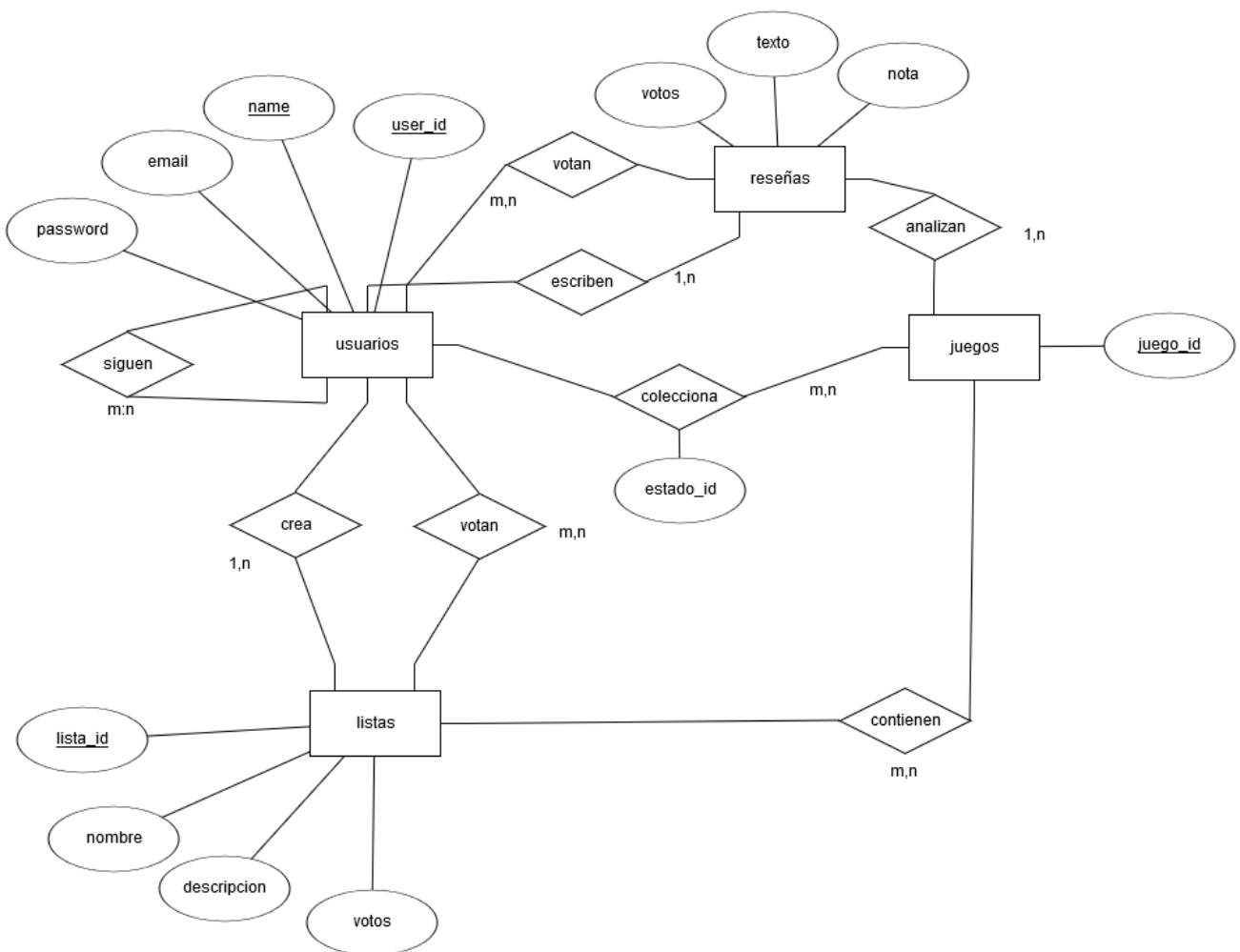
Caso ID	ix
Nombre del Caso	Escribir una reseña
Actores	Usuario identificado
Descripción	Los usuarios pueden escribir una única reseña de un juego que esté en su colección. Esta reseña se puede modificar cuando el usuario quiera.
Desencadenante	Un usuario pincha en el enlace 'Escribir una reseña'
Condiciones previas	El usuario debe estar identificado. Debe encontrarse en la ficha de un juego.
Condiciones posteriores	La reseña del juego se guarda en la base de datos.
Flujo normal	1. El usuario pincha en el enlace 'Escribir una reseña' 2. Se guarda la reseña en la base de datos
Flujo Alternativo	-

Caso ID	x
Nombre del Caso	Valorar una reseña
Actores	Usuario identificado
Descripción	Los usuarios pueden valorar de manera positiva o negativa una reseña de otro usuario.
Desencadenante	Un usuario pincha en un enlace 'Valorar positivamente' o 'Valorar negativamente' de una reseña
Condiciones previas	El usuario debe estar identificado. Debe encontrarse en la reseña de un juego.
Condiciones posteriores	La valoración de la reseña se guarda en la base de datos.
Flujo normal	1. El usuario pincha en uno de los enlaces 'Valorar reseña' 2. Se guarda la valoración en la base de datos
Flujo Alternativo	-

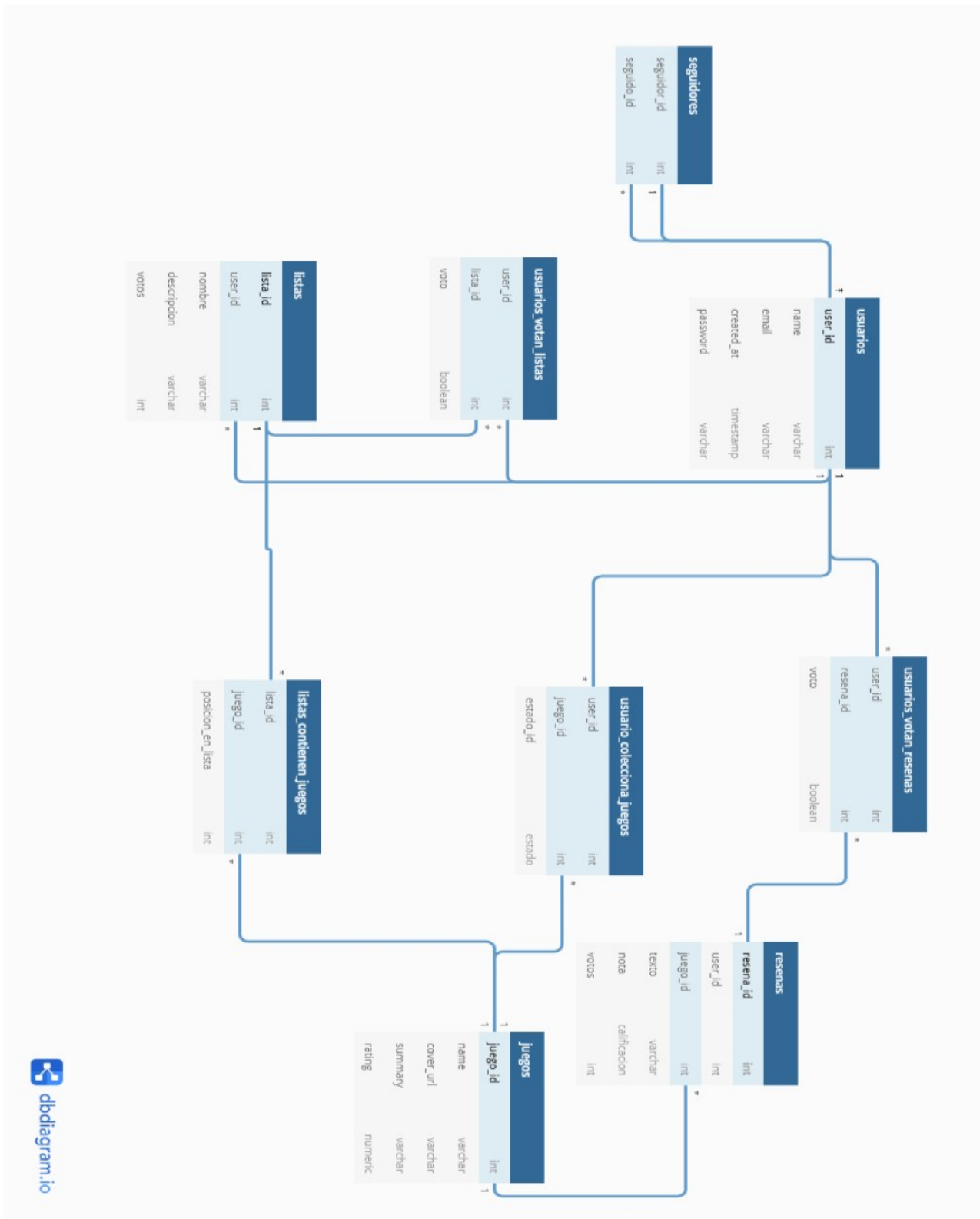
BASES DE DATOS

A continuación se presenta el diagrama entidad-relación de los actores que componen la base de datos. Los atributos que se muestra, si bien existen en cada entidad, no son los únicos que una entidad puede tener. Por ejemplo, en la tabla *users* contiene otros atributos como las recuperaciones de contraseña, añadidas por el framework Jetstream. Para esta presentación se emplearán únicamente solo los datos esenciales para representar las relaciones.

La entidad *juegos* representa los datos recibidos desde la API de Twitch. Se muestra *juego_id* como único atributo y clave primaria para representar la relación con los otros elementos de la base de datos. La cantidad de atributos de cada juego, y de otras entidades de la Api de Twitch, se pueden consultar en <https://api-docs.igdb.com/#endpoints>



A partir de este diagrama, se diseñan las siguientes tablas:



Las entidades *calificación* y *estado* son constantes que se usan para determinar el valor de los atributos *nota* y *estado* en las entidades *usuarios_votan_reseñas* y *usuarios_coleccionan_juegos*:

estado ENUM ("deseado", "jugando", "terminado", "completado", "pospuesto", "abandonado")

calificacion ENUM("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10")

DESPLIEGUE

Llegado el momento de desplegar la aplicación al entorno de producción, es necesario tener en cuenta los requisitos para el servidor que pide Laravel (versión de PHP, JSON, PDO y [más](#)). La manera más sencilla es utilizar Laravel Forge, pues configurará los servicios necesarios, y puede crear servidores en varios proveedores de infraestructuras, como DigitalOcean, Linode o Amazon. Además, ofrece integración con varios sistemas de control de versiones.

Para el propósito de este proyecto, se describe el despliegue manual en un servidor de DigitalOcean, que simplifica bastante las tareas de administración. DigitalOcean ofrece máquinas virtuales (droplets) a las que accederemos via consola por medio de SSH.

Para crear un droplet, elegimos el stack que usaremos (por ejemplo, LAMP Ubuntu 20.04). Estos droplets son fácilmente configurables, y solo se cobran mientras estén activos.

Hay que tener en cuenta que para conectarnos al droplet desde un sistema Windows, es necesaria la herramienta [PuTTY](#).

Una vez conectados, seguimos los siguientes pasos. Téngase en cuenta que estos son pasos 'simplificados' que no pretenden ahondar en las complejidades técnicas de cada uno de ellos.

i. Actualizar el sistema: en principio todo debería estar configurado al elegir el stack LAMP, pero conviene cerciorarse.

ii. Activar el módulo de Apache mod_rewrite, que permite el funcionamiento de las rutas.

iii. Configurar MySQL e instalar phpMyAdmin de así desearlo.

iv. Configuración específica para Laravel: instalar composer, y git (esto nos permite desplegar desde un repositorio). De nuevo, el stack LAMP debería instalar estos componentes.

v. Clonamos nuestro proyecto Laravel a partir de un repositorio remoto (en nuestro caso, GitHub).

vi. Instalamos las dependencias de Laravel (composer.json) y las dependencias de Javascript (npm).

Vii. Creamos nuestro archivo .env (no se clonará automáticamente, pues se encuentra en .gitignore).

Viii. Apuntamos a la carpeta /public: editando 000-default.conf de Apache, elegimos la carpeta public como ruta base del proyecto.

ix. Creamos nuestra base de datos (podemos usar migraciones).

x. Listo. La aplicación estaría desplegada.

PRESUPUESTO

Este proyecto no necesita una gran inversión inicial, puesto que no es necesario invertir en hardware de ningún tipo, y solo deberemos afrontar los costes de desarrollar y mantener la aplicación funcionando a través de un servidor de despliegue.

Aunque la escala del proyecto es en un principio muy pequeña, se ha elegido un servidor virtual privado (VPS) para alojar el proyecto. Esto ofrece más seguridad en el tratamiento de datos, y una personalización mayor a la hora de asignar los recursos a nuestro proyecto, además de proporcionarnos un aislamiento para la base de datos, lo que es recomendable en términos de seguridad.

A continuación se muestra una tabla con los costes estimados. Los precios pueden variar según el proveedor, y se ha optado por un coste aproximado.

COSTES DE INFRAESTRUCTURA

	Coste mensual	Coste anual
Alojamiento en VPS (incluye dominio)	12,00 €	144,00 €
Servidor de correo	3,00 €	36,00 €
		180,00 €

Los costes de diseño y desarrollo se han calculado para un despliegue inicial con la aplicación funcionando de manera básica. Estos costes podrían aumentar dependiendo de como evolucione el proyecto.

COSTES DE DISEÑO Y DESARROLLO (20€/hora, impuestos incluidos)

	Tiempo (horas)	Coste
Planificación y diseño	10	200,00 €
Codificación	50	1000 €
Despliegue e instalación	5	100,00 €
Soporte y mantenimiento	~10 horas / mes	200,00 €
		1.500,00 €

MEJORAS

La aplicación se encuentra en una etapa muy temprana del desarrollo, y se le añaden nuevas funcionalidades constantemente. Actualmente funcionan el sistema de registro de usuarios y login, además de las búsquedas simples a la API de Twitch con las que se pueden consultar los juegos y consultar varios de sus datos. El desarrollo se intentará enfocar en los siguientes puntos:

- Creación de las migraciones y tablas de la base de datos (actualmente funciona con una tabla básica de users, con datos ficticios creados con Tinker).
- Programar los controladores para la creación de listas, colecciones y otros.
- Configuración de la vista home, para mostrar un frontpage atractivo e interactivo sin necesidad de identificación. Diseño de un logo.
- Tratamiento de datos según la Ley de Protección de datos.
- Maquetado y aspecto general de la web (el maquetado actual es muy básico).
- Tras el maquetado, programar de manera *responsive* las vistas que se muestran, y la web en general.
- Mejoras QOL en las fichas de los juegos: posibilidad de consultar las listas en las que se encuentra, y que jugadores lo están jugando actualmente.
- Estudiar la API de *Twitch* para vincular el *login* de la aplicación con el *login* de Twitch, y evaluar su viabilidad.
- Creación de un plan de para usuarios *Premium*, a través del cual podrán disfrutar de varias ventajas exclusivas asociadas a un plan de suscripción, tales como: personalización del perfil, estadísticas avanzadas, manejo avanzado de listas, acceso anticipado a nuevas funcionalidades y más.

APÉNDICE I. Estructura de carpetas de Laravel

Una vez creado el proyecto, podremos comprobar que Laravel ha creado una estructura de directorios para nuestro proyecto. Destacan las siguientes carpetas:

/app: contiene el código fuente de la app y casi todas las clases que creemos. Podremos crear carpetas según sea necesario, como por ejemplo una carpeta *app/mails* para almacenar los archivos que tengan relación con los correos electrónicos.

/console: aquí se almacenarán los comandos creados por nosotros en la consola con `make:command`. Este directorio se creará si no existe en el momento de crear un comando.

http/controllers: aquí se guarda la clase `Controllers`, y podremos crear y almacenar aquí controladores personalizados que descienden de esta clase `Controller`.

/exception: aquí se encuentran los manejadores de errores, y podremos almacenar excepciones personalizadas.

/http: esta carpeta almacenará los controladores, middleware y el kernel para registrarlos.

/providers: guarda los `_service providers` de la aplicación, es decir los servicios que arrancan tanto la aplicación como los servicios básicos de Laravel.

/bootstrap: contiene el archivo `app.php`, el cual arranca el framework. En su interior hay un directorio `cache` en el que se almacenan archivos generados por el framework que optimizarán el rendimiento de la aplicación, pues no tendrá que volver a procesarlos cada vez.

/config: en esta carpeta se guardan los archivos de configuración de la aplicación, como por ejemplo el nombre de la misma. También tenemos archivos de autenticación o de acceso a diferentes bases de datos. Se pueden cambiar gran cantidad de parámetros, pero es recomendable hacer esto en el archivo `.env` de la carpeta `root`. El archivo `.env` almacenará las variables de entorno local, pudiendo así ajustar la aplicación del modo que más nos convenga según el entorno en el que la estemos configurando.

/database: contiene las carpetas los modelos de factories, migrations de la base de datos y seeds para almacenar diferentes archivos relacionados con la base de datos. De usar SQLite podremos almacenar aquí la base de datos.

/public: contiene el archivo `index.php`, que es el punto de entrada de todas las peticiones que hagamos a la aplicación. Es el único directorio accesible públicamente, y almacenaremos aquí los archivos `css`, `js` e imágenes públicas.

/resources/views: almacenaremos aquí las vistas de la aplicación, y puede contener también archivos `css`, `js` e imágenes no compiladas que luego pasarán a la carpeta `public/`

`/lang`: para los archivos de traducción. Por defecto Laravel trae la configuración en inglés, pero podremos descargar fácilmente un archivo en cualquier idioma que queramos desde **este repositorio de github**.

/routes: contiene las rutas de la aplicación. Por defecto Laravel incluye varios archivos de rutas: *Api.php*, *console.php*, *channels.php* y *web.php*. Este último es el más que nos interesa, y en el registraremos las rutas que creemos para navegar por la aplicación.

/storage: contiene las plantillas Blade compiladas, archivos en cachés y otros archivos generados por el framework. Aquí podremos consultar el log de la aplicación en el archivo `logs/laravel.log`