Diseño de la Base de Datos: Wordle+.

Alumno: David Correa Rodríguez
Tutor: Juan Manuel Fernández Luna

Análisis del contexto

Para realizar un correcto diseño de la base de datos, previamente a caracterizar éste a la tecnología a utilizar, es necesario hacer un análisis de toda la información necesaria a almacenar y sus respectivas relaciones.

Para ello, y recapitulando todas las características principales del proyecto, existen varias entidades básicas de las que tenemos que almacenar información:

- Usuarios
- Partidas
- Torneos

De los usuarios derivan: los jugadores, los gestores de eventos, la lista de amigos, las peticiones de amistad y las notificaciones.

De los torneos derivan: las participaciones de los jugadores en los torneos y las rondas de cada torneo.

Con esta información, podemos identificar algunos elementos que se darán en la base de datos, añadiendo también los campos asociados:

- **Usuarios**: son los usuarios que utilizarán la aplicación. Sus campos asociados son: nombre de usuario, correo, nombre, apellidos, fecha de registro, si es mánager o no, y el avatar):

    - **Jugadores**: son aquellos usuarios que utilizarán la plataforma para realizar las actividades relacionadas con el juego. Sus campos relacionados son: nombre de usuario, email, wordles completados, partidas 1vs1 ganadas, torneos ganados, experiencia y categoría)

    - **Gestores de eventos**: son aquellos usuarios que se encargan de la gestión de los torneos de la plataforma. Sus

campos asociados son: nombre de usuario, correo, nombre, apellidos, fecha de registro

- **Lista de amigos**: representa la lista de amigos de un jugador. Sus campos asociados son: nombre de usuario de un jugador, nombre de usuario de otro jugador.

- **Solicitudes de amigos:** representa las solicitudes de amistad de un jugador. Sus campos asociados son: nombre de usuario del remitente, nombre de usuario del destinatario, la fecha de creación y si es válida o no.

- **Notificaciones:** representa todas las notificaciones relacionadas con un jugador. Sus campos asociados son: identificador de la notificación, nombre de usuario asociado a la notificación, texto asociado y link asociado.

- **Partidas:** representa las partidas 1 contra 1 entre jugadores. Sus campos asociados son: identificador de la partida, nombre de usuario del jugador 1, nombre de usuario del jugador 2, si el jugador 1 ha jugado, si el jugador 2 ha jugado, y quién es el ganador.
Nota: las partidas serán asíncronas, es decir, no es necesario que ambos jugadores estén al mismo tiempo jugando la partida, por lo que es necesario los campos de si cada jugador ha jugado la partida.

- **Torneos:** representa los eventos entre jugadores cuya competición está relacionada con partidas de Wordle 1 contra 1. Sus campos asociados son: identificador del torneo, nombre, descripción, número máximo de jugadores y si está cerrado o no. Un torneo se considerará cerrado si:
    - De por sí, es un torneo cerrado, es decir, si es un torneo el cual los jugadores de la plataforma no se pueden unir por voluntad propia. Los participantes de este tipo de torneo estarán seleccionados por los gestores de eventos.
    - El número de jugadores inscritos en el torneo es igual a su número máximo permitido. Este caso se da en los torneos abiertos, en los que cualquier jugador puede participar.

- **Rondas:** los torneos se subdividen en rondas, en donde en cada ronda se obtiene el jugador ganador de cada partida 1 contra 1. Sus campos asociados son: identificador de la ronda, identificador del torneo asociado, número de la ronda, partida asociada a dicha ronda

En el análisis podemos notar lo siguiente:

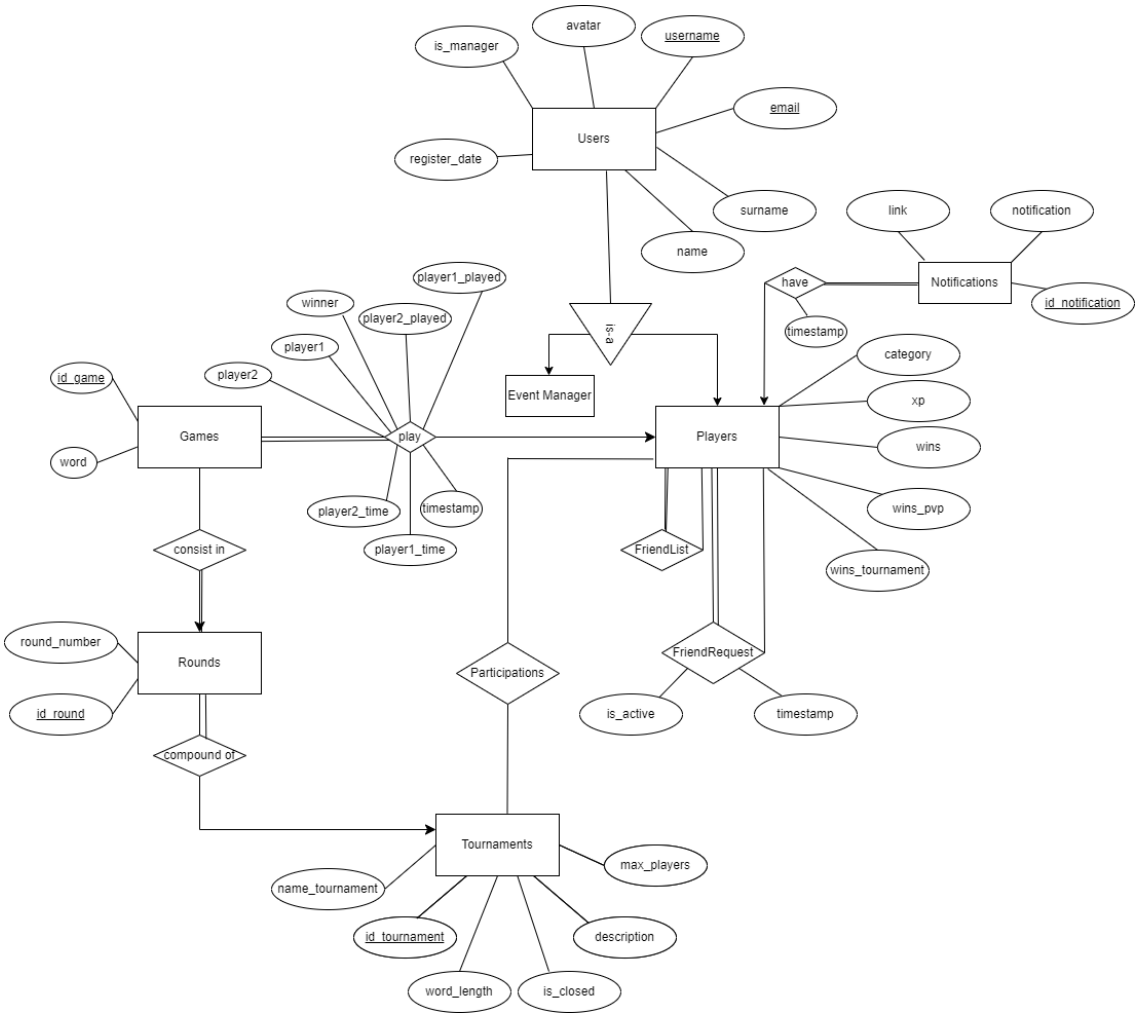
- Las entidades **Jugadores** y **Gestores de eventos** son entidades heredadas de **Usuarios**, por lo que tendrán una relación de herencia *es-un.*
- La entidad **Gestores de eventos** resulta innecesaria al tener un subconjunto (y solamente dicho subconjunto) de los campos de la entidad **Usuarios**.

- Existen varias entidades que dependen existencialmente de otras, es decir, la existencia de una entidad es obligatoria para la existencia de otra entidad. Estas restricciones se detallarán más adelante, una vez diseñado el diagrama Entidad-Relación.

Diagrama Entidad-Relación

Un modelo de datos entidad-relación está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados entidades, los cuales pueden tener propiedades o atributos asociados; y de relaciones, que son conexiones entre las entidades. [29]

Teniendo en cuenta todos los aspectos destacados en el previo análisis, un posible diagrama Entidad-Relación sería el siguiente:



Algunas aclaraciones a tener en cuenta son:

1. Como se ha comentado anteriormente, la entidad *Event Manager* es una especialización de la entidad *Users* sin ningún atributo extra, por lo que añadiendo el campo *is_manager* a la entidad *Users* no es necesario crear una tabla nueva para *Event Manager.* La representación de los gestores de eventos en la base de datos puede hacerse con el campo *is_manager* de la tabla *Users.* En cambio, la entidad *Players*, al tener muchos más campos, si requiere una tabla aparte en donde almacenar dicha información.

2. Respecto a la entidad *Notifications* y la relación *Have,* un jugador puede tener muchas notificaciones, pero esa notificación solamente puede pertenecer a un jugador, por lo que la relación *have* entre *Players* y *Notifications* es de 1:N. Además, la entidad de notificaciones depende existencialmente de la entidad *Players*, ya que no tiene sentido una notificación que no tenga asociado un jugador. La entidad *Notifications* heredará como clave externa la clave de la entidad *Player.*

3. Respecto a la relación recursiva *FriendList* entre la entidad *Players*, un jugador puede ser amigo de muchos jugadores, y también viceversa, por lo que presenta una cardinalidad N:M. Además, la tabla resultante de esta relación presenta una obligatoriedad con la entidad *Players* ya que las relaciones de amistad solamente tienen sentido entre instancias de la entidad *Players*. Esta relación heredará las claves primarias de la entidad *Players.*

4. Respecto a la relación *FriendRequest*, presenta ciertas similitudes con la relación anterior, pero con ciertos atributos extra. Un jugador puede tener varias peticiones de amistad, y una misma petición de amistad puede pertenecer a varios jugadores, por lo que presenta una cardinalidad N:M, pero en este caso solamente puede pertenecer a uno de los dos jugadores, pero no de forma simultánea. Esta condición deberá comprobarse antes de introducir valores a la tabla resultante. La tabla resultante de esta relación exige una obligatoriedad con la entidad *Players* con la misma justificación que el punto anterior.

5. Respecto a la relación *Play* entre las entidades *Players* y *Games*, un jugador puede jugar muchas partidas, y esa partida puede ser jugada exactamente por dos jugadores, por lo que la relación presenta una cardinalidad N:2, con obligatoriedad de las partidas hacia los jugadores, ya que sin jugadores las partidas no pueden ser jugadas.

6. Respecto a la relación *Consist in* entre las entidades *Games* y *Rounds*, una ronda consiste en varias partidas, pero una partida de un torneo en concreto solamente puede pertenecer a una ronda, ya que en un torneo no se repiten enfrentamientos entre los mismos jugadores en diferentes momentos (en torneos eliminatorios), por lo que la relación presenta una cardinalidad 1:N. Existe obligatoriedad de las rondas hacia las partidas, es decir, una ronda debe estar compuesta por partidas, ya que no tiene sentido que una ronda no tenga partidas.

7. Respecto a la relación *Compound of* entre las entidades *Tournaments* y *Rounds*, un torneo se compone de varias rondas, pero dicha ronda solamente puede pertenecer a un torneo en concreto, por lo que la relación presenta una cardinalidad 1:N. Además, existe obligatoriedad entre las rondas y los torneos, ya que una ronda no tiene sentido sin antes existir un torneo que la contenga.

8. Respecto a la relación *Participations* entre las entidades *Tournaments* y *Players*, un jugador puede participar en muchos torneos, y en un mismo torneo pueden participar muchos jugadores, por lo que la relación presenta una cardinalidad N:M. No existe obligatoriedad entre estas

entidades, ya que la existencia de los torneos y los jugadores son independientes.

Paso a tablas

Una vez está diseñado el esquema conceptual de la base de datos, podemos realizar un modelo lógico de la base de datos realizando el “paso a tablas”. En este proceso, se deben tener las siguientes consideraciones:

- Los atributos de las **entidades fuertes** se representarán por medio de una tabla en donde cada tupla es una ocurrencia del conjunto de entidades y está caracterizada por *n* columnas distintas, una por cada atributo. La clave primaria de la tabla está constituida por los atributos que forman la clave primaria en el conjunto de entidades.

- Los atributos de las **entidades débiles** se representarán de forma similar a las entidades fuertes, pero considerando que la clave primaria de la tabla estará constituida por los atributos que forman la clave primaria de la entidad de la que depende, más los atributos marcados como discriminadores o claves parciales de la entidad débil si existen. Además, hay que generar una clave externa que referencia a la entidad de la que depende. En este caso, no existen entidades débiles.

- Los atributos de las **relaciones** se representarán en tablas en donde la clave primaria dependerá de la cardinalidad de dicha relación:
    - Si la relación es de muchos a muchos, la clave primaria estará formada por la unión de las claves primarias de los conjuntos de entidades que intervienen en la relación, con posibilidad de añadir algunos atributos de la relación.
    - Si la relación es de muchos a uno, la clave primaria estará formada por la clave primaria de la entidad con la cardinalidad de muchos.
    - Si la relación es de uno a uno, existirán dos claves candidatas de las cuales una de ellas será la clave primaria.

- Los atributos de las relaciones de **herencia** se representarán en tablas en donde el conjunto de entidades más general pasa a ser una tabla según su tipo de entidad, y cada conjunto de entidades de nivel inferior será una tabla constituida por los atributos propios más la clave primaria de la entidad superior.

En cualquier caso, los atributos que identifican a las claves de otras entidades hay que establecerlos como claves externas a las claves primarias de dichas entidades.

Con esto, las tablas resultantes del diagrama Entidad-Relación son las siguientes:

Entidades:

Users
username - PK
email - UNIQUE
name
surname
register_date
is_manager
avatar

Players
username - PK, FK (Users)
email - UNIQUE, FK (Users)
wins
wins_pvp
wins_tournament
xp
category

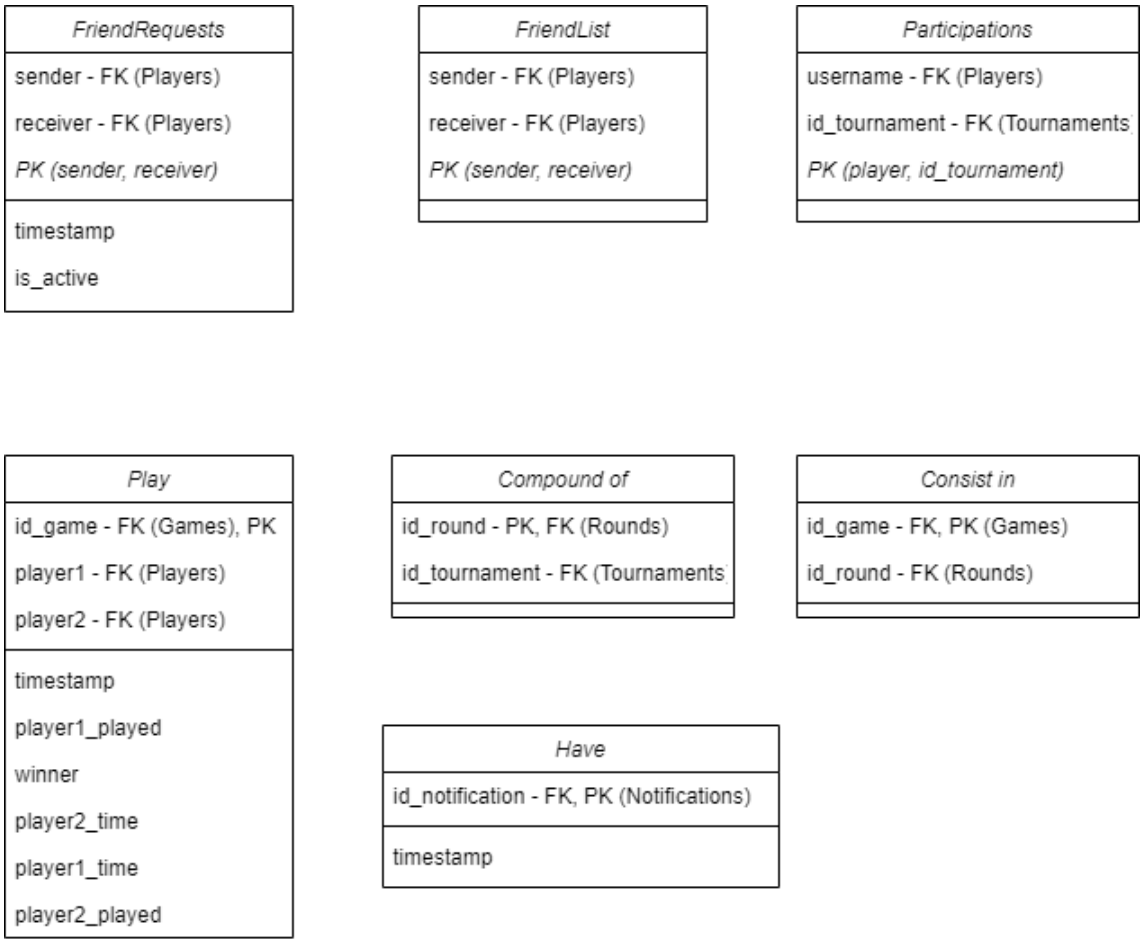
Games
id_game - PK
word

Notifications
id_notification - PK
username - FK (Users)
text
link

Tournaments
id_tournament - PK
name_tournament
description
max_players
word_length
is_closed

Rounds
id_round - PK
round_number

Relaciones:



Fusiones

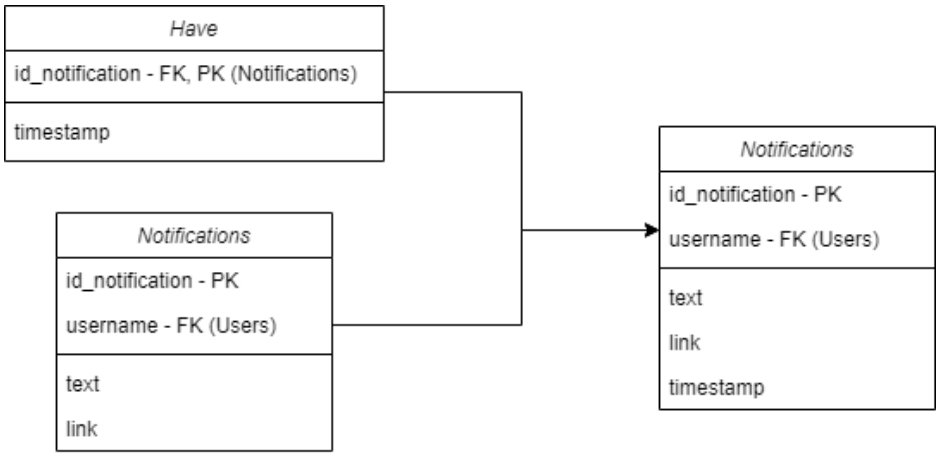
Las fusiones consisten en combinar varias tablas, permitiendo una reducción del número de estas, siempre y cuando no se pierda información (tanto de datos como de restricciones). Las fusiones también permiten mejoras de eficiencia a nivel de almacenamiento, es decir, se ahorra espacio de almacenamiento; y rendimiento del sistema, ya que las búsquedas pueden verse aceleradas al compactar la información.

Las condiciones necesarias que se deben dar para realizar una fusión son:

- Que las tablas tengan la misma clave primaria o candidata.
- Que ninguna de las tablas proceda de herencia.
- Que semánticamente la fusión tenga sentido.

1. La fusión entre las tablas *Users* y *Players* no puede darse, al pertenecer la tabla *Players* de una herencia de la Tabla *Users*.

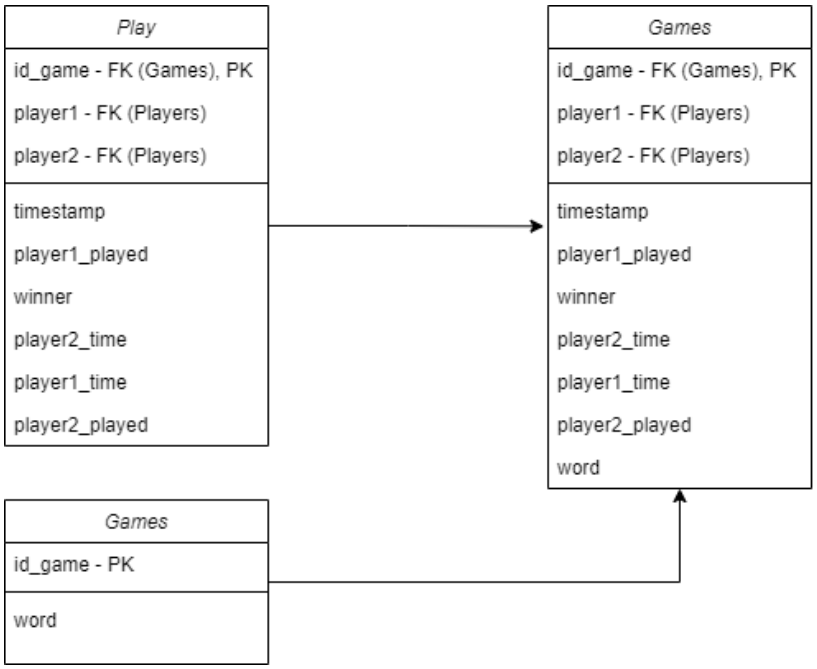
2. La fusión entre las tablas *Have* y *Notifications* puede darse, al tener la misma clave primaria (*id_notification* de *Notifications*) y ser semánticamente compatibles, ya que una notificación siempre va a estar asociada a un jugador. Cabe mencionar que no se puede dar la fusión entre *Have* y *Players* ya que un jugador puede que no tenga notificaciones en ningún momento.



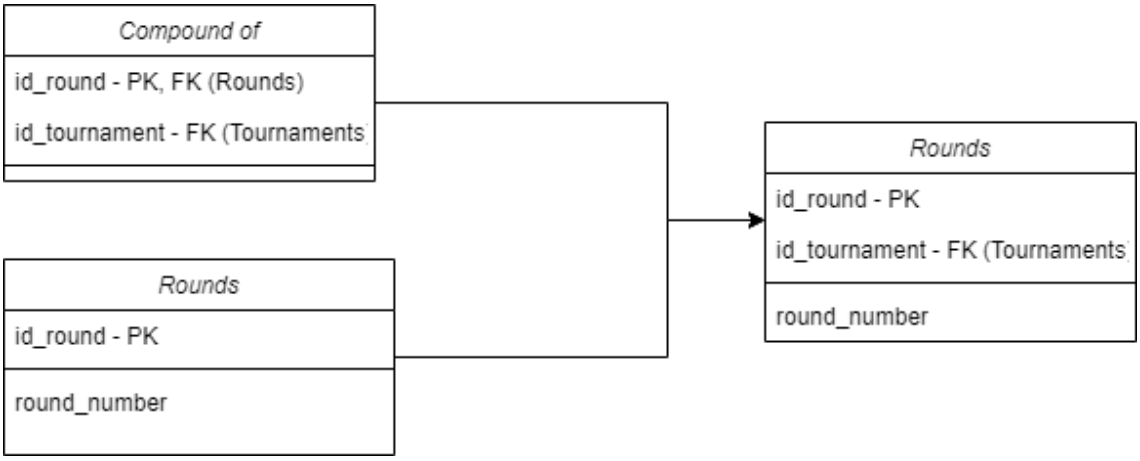
3. La fusión entre las tablas *Players* y *FriendList* no puede darse, ya que un jugador no tiene por qué tener una lista de amigos. De forma similar, tampoco puede darse entre *Players* y *FriendRequest*.

4. La fusión entre las tablas *Players* y *Participations* no puede darse, ya que no siempre un jugador va a participar en un torneo. De forma similar, tampoco puede darse entre *Tournaments* y *Participations* ya que pueden existir torneos sin participantes. Este es el caso temporal de los torneos abiertos, en los que en un momento determinado (en su creación) los torneos abiertos no tienen participantes hasta que éstos no ingresan en el torneo.

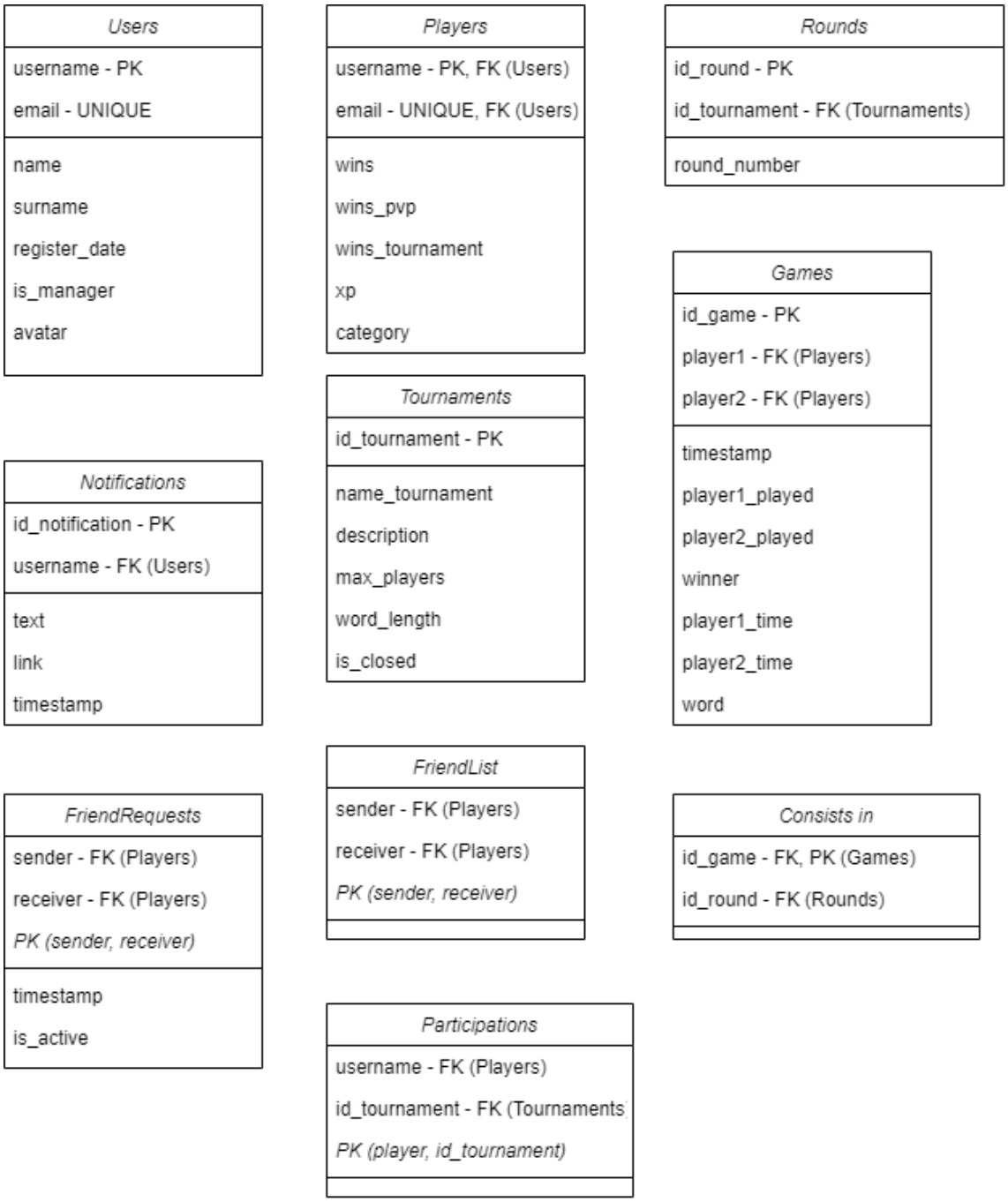
5. La fusión entre las tablas *Games* y *Play* puede darse, al tener la misma clave primaria (*id_game* de *Games*) y ser semánticamente compatibles, ya que una partida siempre va a estar relacionada directamente con dos jugadores. Cabe mencionar que no se puede dar la fusión entre *Players* y *Play*, ya que podrían existir jugadores que nunca han jugado a ninguna partida.



6. La fusión entre las tablas *Rounds* y *compound of* puede darse, al tener la misma clave primaria (*id_round* de *Rounds*) y ser sermánticamente compatibles, ya que una ronda siempre estará relacionada con un torneo existente. Cabe mencionar que no se puede dar la fusión entre *Rounds* y *consist in* ya que no tienen la misma clave primaria.



Por tanto, y como resultado de las fusiones anteriores, el modelado de tablas de la base de datos queda de la siguiente manera:


Users
username - PK
email - UNIQUE
name
surname
register_date
is_manager
avatar
Players
username - PK, FK (Users)
email - UNIQUE, FK (Users)
wins
wins_pvp
wins_tournament
xp
category
Rounds
id_round - PK
id_tournament - FK (Tournaments)
round_number
Games
id_game - PK
player1 - FK (Players)
player2 - FK (Players)
timestamp
player1_played
player2_played
winner
player1_time
player2_time
word
Tournaments
id_tournament - PK
name_tournament
description
max_players
word_length
is_closed
Notifications
id_notification - PK
username - FK (Users)
text
link
timestamp
FriendList
sender - FK (Players)
receiver - FK (Players)
PK (sender, receiver)
FriendRequests
sender - FK (Players)
receiver - FK (Players)
PK (sender, receiver)
timestamp
is_active
Consists in
id_game - FK, PK (Games)
id_round - FK (Rounds)
Participations
username - FK (Players)
id_tournament - FK (Tournaments
PK (player, id_tournament)

Normalización

El proceso de normalización de base de datos consiste en aplicar una serie de procedimientos a las tablas obtenidas tras el paso del modelo entidad-relación al modelo relacional (o tablas) para conseguir minimizar la redundancia de los datos. [30]

Los objetivos de la normalización son:

- Minimizar la redundancia de los datos.
- Reducir los problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

Existen varios tipos de normalización, que incluyen la Primera Forma Normal (1FN), Segunda Forma Normal (2FN), Tercera Forma Normal (3FN), Forma normal de Boyce-Codd (FNBC), Cuarta Forma Normal (4FN) y Quinta Forma Normal (5FN). En general, las primeras tres formas normales son el mínimo que deben cubrir todas las bases de datos, y es recomendable normalizarlas hasta la Forma Normal de Boyce-Codd. Se dice que una base de datos está normalizada en la forma normal N si todas sus tablas están en la forma normal N.

Es necesario mencionar que un atributo no primo (o no primario) es aquel que no pertenece a ninguna clave candidata.

De esta manera, las tablas obtenidas en el punto anterior serán normalizadas hasta FNBC. [30]

Primera Formal Normal (1FN)

Las reglas de la 1FN son:

1) Todos los atributos son atómicos, es decir, los elementos del dominio son simples e indivisibles.
2) No existe variación entre el número de columnas.
3) Los campos no clave se identifican por la clave.
4) Hay una independencia del orden tanto de las filas como de las columnas.

Si analizamos todas las tablas resultantes del paso a tablas, observamos que todas las tablas están en 1FN:

- ***Users***:

1) Sus atributos asociados (*username, email, name, surname, register_date, is_manager, avatar*) son de dominios simples.
2) El número de columnas no varía.
3) Todos los campos que no son clave están identificados por la clave.
4) Si los datos cambian de orden no cambian sus significados.

- ***Players***:
  1) Sus atributos asociados (*username, wins, wins_pvp, wins_tournament, xp, category*) son de dominios simples.
  2) El número de columnas no varía.
  3) Todos los campos que no son clave están identificados por la clave.
  4) Si los datos cambian de orden no cambian sus significados.

- ***Rounds***:
  1) Sus atributos asociados (*id_round, id_tournament, round_number*) son de dominios simples.
  2) El número de columnas no varía.
  3) Todos los campos que no son clave están identificados por la clave.
  4) Si los datos cambian de orden no cambian sus significados.

- ***Notifications***:
  1) Sus atributos asociados (*id_notification, username, text, link, timestamp*) son de dominios simples.
  2) El número de columnas no varía.
  3) Todos los campos que no son clave están identificados por la clave.
  4) Si los datos cambian de orden no cambian sus significados.

- ***Tournaments***:
  1) Sus atributos asociados (*id_tournament, name_tournament, description, max_players, word_length, is_closed*) son de dominios simples.
  2) El número de columnas no varía.
  3) Todos los campos que no son clave están identificados por la clave.
  4) Si los datos cambian de orden no cambian sus significados.

- ***Games***:
  1) Sus atributos asociados (*id_game, player1, player2, timestamp, player1_played, player2_played, winner, player1_time, player2_time, word*) son de dominios simples.
  2) El número de columnas no varía.
  3) Todos los campos que no son clave están identificados por la clave.

4) Si los datos cambian de orden no cambian sus significados.

- ***FriendRequests***
  1) Sus atributos asociados (*sender, receiver, timestamp, is_active*) son de dominios simples.
  2) El número de columnas no varía.
  3) Todos los campos que no son clave están identificados por la clave.
  4) Si los datos cambian de orden no cambian sus significados.

- ***FriendList***
  1) Sus atributos asociados (*sender, receiver*) son de dominios simples.
  2) El número de columnas no varía.
  3) Todos los campos que no son clave están identificados por la clave.
  4) Si los datos cambian de orden no cambian sus significados.

- ***Consist in***
  1) Sus atributos asociados (*id_game, id_round*) son de dominios simples.
  2) El número de columnas no varía.
  3) Todos los campos que no son clave están identificados por la clave.
  4) Si los datos cambian de orden no cambian sus significados.

- ***Participations***
  1) Sus atributos asociados (*username, id_tournament*) son de dominios simples.
  2) El número de columnas no varía.
  3) Todos los campos que no son clave están identificados por la clave.
  4) Si los datos cambian de orden no cambian sus significados.

Al encontrarse todas las tablas en 1FN, se concluye que la base de datos se encuentra en 1FN.

Segunda Formal Normal (2FN)

La condición necesaria de la 2FN es que la base de datos esté en 1FN, y, además, que los atributos que no forman parte de ninguna clave dependan de forma completa de la clave principal, es decir, que no haya dependencias parciales. Todos los atributos que no pertenezcan a la clave deben depender

únicamente de la clave. Este requisito está basado en el concepto de dependencia funcional. [30]

Cabe recalcar que las tablas que están en 1FN, y que además disponen de una clave primaria formada por una única columna con valor indivisible, cumple con la 2FN.

Con esta aclaración, las tablas que cumplen dicha condición son: *Users, Players, Rounds, Notifications, Tournaments, Games,* y *Consist in,* ya que su clave primaria solamente está formada por un único campo.

Las tablas restantes, las cuales son necesarias analizar si están en 2FN son: *FriendRequests*, *FriendList* y *Participations*.

- ***FriendRequests***:
  1) La clave primaria de esta tabla está formada por la combinación de los usuarios involucrados en la petición de amistad (*sender, receiver*).
  2) El atributo *timestamp* representa la fecha de creación de la petición de amistad, y es totalmente dependiente de toda la clave primaria. Este campo no puede depender solamente del campo *sender* ya que no se conocería quién es el que recibe la petición; y tampoco puede depender solamente de *receiver* ya que no se conocería quién es el que la manda.
  3) El atributo *is_active* representa si la solicitud de amistad está activa o no. Controla si la petición se ha cancelado, rechazado o aceptado. Este campo es totalmente dependiente de la clave primaria por la misma razón que el atributo *timestamp,* ya que es un atributo que relaciona a ambos jugadores.

     Por tanto, esta tabla se encuentra en 2FN.

- ***FriendList***:
  1) La clave primaria de esta tabla está formada por la combinación de los usuarios involucrados en la relación de amistad (*sender, receiver).*
  2) Esta tabla no tiene más atributos asociados, por lo que no existen dependencias parciales.

     Por tanto, esta tabla se encuentra en 2FN.

- ***Participations***:

1) La clave primaria de esta tabla está formada por la combinación del nombre de usuario del jugador y el identificador de torneo (*username, id_tournament*).
2) Esta tabla no tiene más atributos asociados, por lo que no existen dependencias parciales.

   Por tanto, esta tabla se encuentra en 2FN.

Al encontrarse todas las tablas en 2FN, se concluye que la base de datos se encuentra en 2FN.

Tercera Formal Normal (3FN)

La condición necesaria para que una base de datos se encuentre en 3FN es que se encuentre en 2FN y, además, no contenga ninguna dependencia transitiva entre los atributos que no son clave.

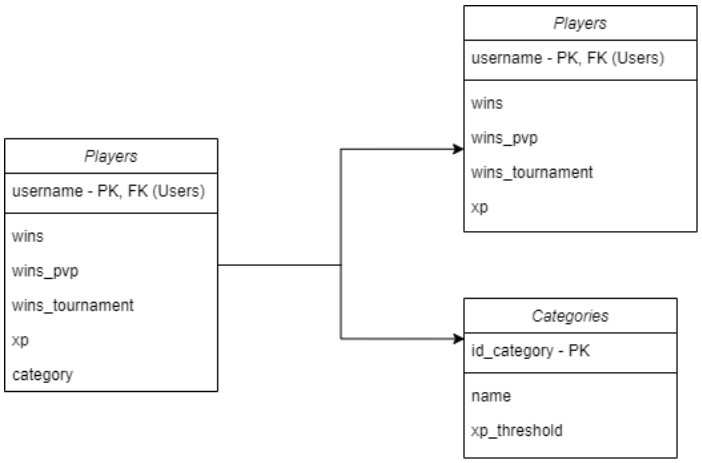
Una dependencia transitiva es una dependencia funcional X → Z en la cual Z no es inmediatamente dependiente de X, pero sí de un tercer conjunto de atributos Y, que a su vez depende de X (y siempre que no ocurra que X sea también dependiente de Y). Es decir, X → Z por virtud de X → Y e Y → Z (y no ocurre que Y → X). Dicho de otra manera, las columnas que no pertenezcan a la clave primaria deben depender solamente de la clave, y no de otra columna que no sea clave. [31]

A continuación, se expone un análisis de cada tabla de la base de datos, comprobando si se cumplen las condiciones de la 3FN:

- ***Users***:
  1) Sus atributos no primos (*name, surname, register_date, is_manager, avatar*) solamente dependen de la clave primaria (*username*) o de la clave candidata (*email*), y no existen dependencias entre ellos.

- ***Players***:
  1) Sus atributos no primos (*wins, wins_pvp, wins_tournament, xp, category*) solamente dependen de la clave primaria (*username*) o de la clave candidata (*email*), y no existen dependencias entre ellos, excepto el campo *category*.
  2) El campo *category* depende del campo *xp*, y el campo *xp* depende de la clave primaria *username*. Por tanto, existe una dependencia transitiva que es necesario eliminar para cumplir las condiciones de la 3FN.

3) Para ello, esta tabla se divide en dos, eliminando la dependencia transitiva. De esta manera, el campo *category* de la tabla *Players* se elimina, y se crea una nueva tabla *Categories* que contiene la información asociada (el identificador, el nombre de la categoría y el umbral de experiencia de dicha categoría)



4) Con esta división, ya no existe la dependencia transitiva.

- ***Rounds***:
  1) Sus atributos no primos (*id_tournament, round_number*) solamente dependen de la clave primaria (*id_round*), y no existen dependencias entre ellos.

- ***Notifications***:
  1) Sus atributos no primos (*username, text, link, timestamp* solamente dependen de la clave primaria (*id_notification*), y no existen dependencias entre ellos.

- ***Tournaments***:
  1) Sus atributos no primos (*name_tournament, description, max_players, word_length, is_closed*) solamente dependen de la clave primaria (*id_tournament*), y no existen dependencias entre ellos.

- ***Games***:
  1) Sus atributos no primos (*player1, player2, timestamp, player1_played, player2_played, winner, player1_time, player2_time, word*) solamente dependen de la clave primaria (*id_game*), y no existen dependencias entre ellos.

- ***FriendRequests***

1) Sus atributos no primos (*timestamp, is_active*) solamente dependen de la clave primaria (*sender, receiver*), y no existen dependencias entre ellos.

- ***FriendList***
  1) No contiene atributos no primos, solamente contiene su clave primaria (*sender, receiver*).

- ***Consist in***
  1) El único atributo no primo (*id_round*) depende de la clave primaria (*id_game*).

- ***Participations***
  1) No contiene atributos no primos, solamente contiene su clave primaria (*username, id_tournament).*

Al encontrarse todas las tablas en 2FN, se concluye que la base de datos se encuentra en 3FN.

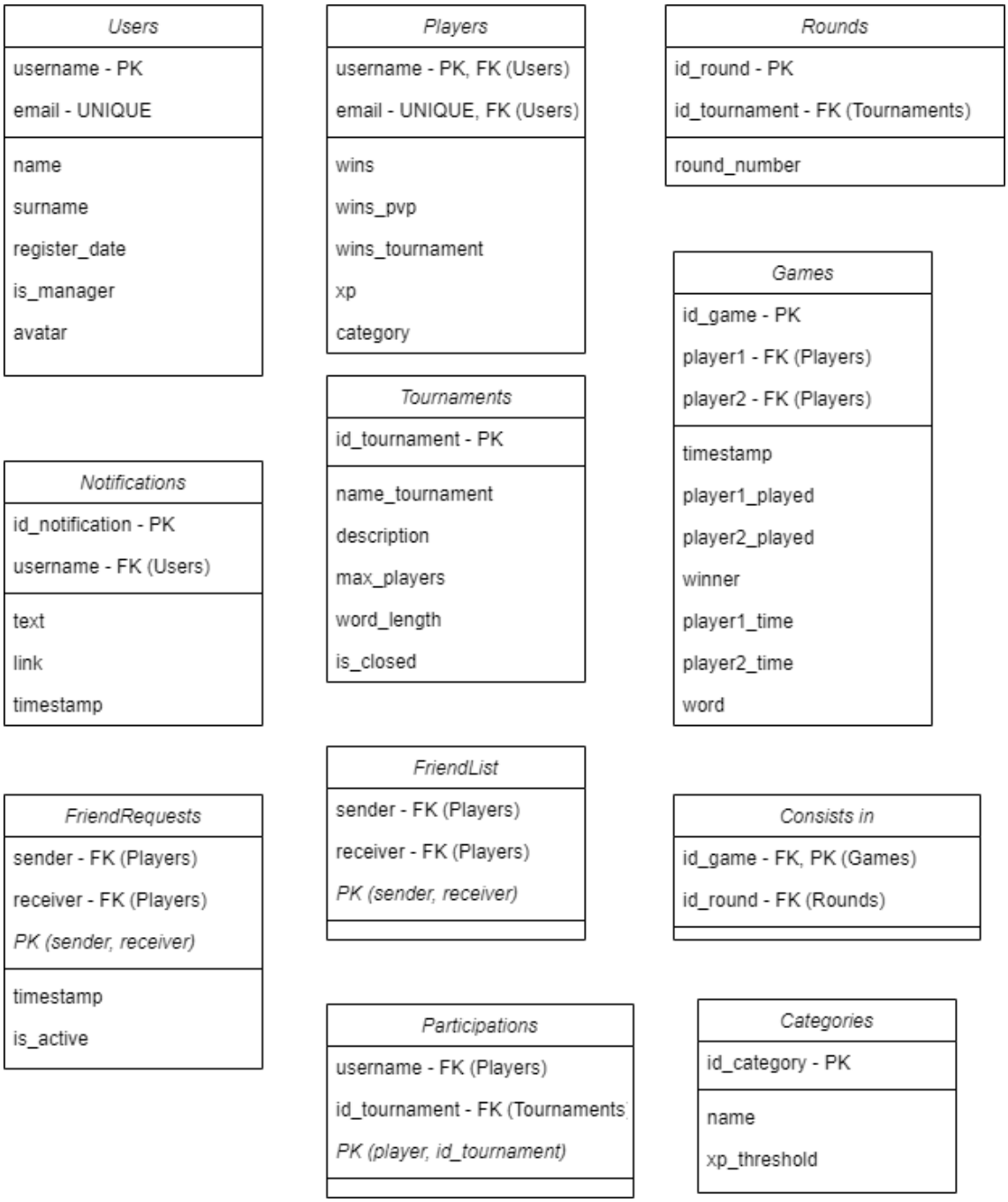
Forma Normal de Boyce-Codd (FNBC)

La FNBC es una versión ligeramente más restrictiva que la 3FN, que impone que una tabla estará en FNBC si está en 3FN, y además no existen dependencias funcionales no triviales de los atributos que no sean un conjunto de la clave candidata. Dicho de otra manera, se cumple la 3FN si y sólo si todo determinante es una clave candidata, donde determinante de una relación es todo conjunto de atributos del cual depende de forma completa otro atribulo de la relación.

Una forma sencilla de comprobar si una tabla se encuentra en FNBC es que no tenga clave candidatas compuestas, es decir, con varios atributos. [32]

En esta base de datos, las únicas tablas con claves candidatas son *Users* y *Players* con el atributo *email*, que no es una clave candidata compuesta, sino simple. Por lo que se concluye que la base de datos se encuentra en FNBC.

Como conclusión, y tras el paso a tablas y la normalización, las tablas resultantes de dichos procesos son las siguientes:



Bibliografía y referencias: Wordle+.

[1] Wikipedia. *es.wikipedia.org.* 22 de febrero de 2023. https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software.

[2] SA, ERIKA. *atlassian.com.* s.f. https://www.atlassian.com/es/agile/scrum/scrum-metrics (último acceso: 2 de febrero de 2023).

[3] Kabanize. *¿Qué es Kanban?* s.f. https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban#:~:text=Kanban%20es%20un%20m%C3%A9todo%20Lean,la%20eficiencia%20y%20mejorar%20continuamente.

[4] Unity. *¿Qué es CICD?* s.f. https://unity.com/es/solutions/what-is-ci-cd#:~:text=La%20CI%2FCD%2C%20o%20integraci%C3%B3n,la%20entrega%20continua%20de%20c%C3%B3digo.

[5] Google. *developer.android.com.* s.f. https://developer.android.com/studio (último acceso: 25 de febrero de 2023).

[6] Flutter. *https://flutter.dev/.* s.f. https://flutter.dev/ (último acceso: 25 de febrero de 2023).

[7] IBM. *www.ibm.com.* 9 de mayo de 2019. https://www.ibm.com/es-es/cloud/learn/lamp-stack-explained.

[8] XenForo. *linux.org.* s.f. https://www.linux.org/.

[9] Apache. *httpd.apache.org/.* s.f. https://httpd.apache.org/ (último acceso: 25 de febrero de 2023).

[10] Oracle. *mysql.com.* s.f. https://www.mysql.com/ (último acceso: 25 de febrero de 2023).

[11] PHP. *php.net.* s.f. https://www.php.net/ (último acceso: 25 de febrero de 2023).

[12] JavaScript. *javascript.com.* s.f. https://www.javascript.com/ (último acceso: 25 de febrero de 2023).

[13] MongoDB. *What is the MERN stack?* s.f. https://www.mongodb.com/mern-stack#:~:text=MERN%20stands%20for%20MongoDB%2C%20Express,MongoDB%20%E2%80%94%20document%20database.

[14] Inc., MongoDB. *mongodb.com.* s.f. https://www.mongodb.com/ (último acceso: 25 de febrero de 2023).

[15] Platforms, Meta. *reactjs.org.* s.f. https://es.reactjs.org/.

[16] Angular. *angular.io.* s.f. https://angular.io/ (último acceso: 25 de febrero de 2023).

[17] Foundation, OpenJS. *nodejs.org.* s.f. https://nodejs.org/en/ (último acceso: 25 de febrero de 2023).

[18] Dart. *dart.dev.* s.f. https://dart.dev/ (último acceso: 25 de febrero de 2023).

[19] Native, React. *https://reactnative.dev/.* s.f.

[20] Wikipedia. *Objective-C definition.* s.f. https://es.wikipedia.org/wiki/Objective-C.

[21] Inc, Apple. *swift.* s.f. https://www.apple.com/es/swift/.

[22] Oracle. *java.com.* s.f. https://www.java.com/es/ (último acceso: 25 de febrero de 2023).

[23] Ionic. *https://ionicframework.com/.* s.f.

[24] You, Evan. *vuejs.org.* s.f. https://vuejs.org/ (último acceso: 25 de febrero de 2023).

[25] PostgreSQL. *postgresql.org.* s.f. https://www.postgresql.org/ (último acceso: 25 de febrero de 2023).

[26] Python. *python.org.* s.f. https://www.python.org/ (último acceso: 25 de febrero de 2023).

[27] Framework, Django Rest. *django-rest-framework.org.* s.f. https://www.django-rest-framework.org/ (último acceso: 25 de febrero de 2023).

[28] Docker. *docker.com.* s.f. https://www.docker.com/ (último acceso: 25 de febrero de 2023).

[29] Wikipedia. *Modelo entidad-relacion.* s.f. https://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n (último acceso: 13 de febrero de 2023).

[30] Wikipedia. *Normalización de base de datos.* s.f. https://es.wikipedia.org/wiki/Normalizaci%C3%B3n_de_bases_de_datos (último acceso: 16 de marzo de 2023).

[31] Wikipedia. *Tercera forma normal.* s.f. https://es.wikipedia.org/wiki/Tercera_forma_normal (último acceso: 21 de marzo de 2023).

[32] Wikipedia. *Forma normal de Boyce Codd.* s.f. https://es.wikipedia.org/wiki/Forma_normal_de_Boyce-Codd (último acceso: 21 de marzo de 2023).