

1 Introducción:	3
1.1 Objeto del proyecto:	3
1.1.1 Justificación y sentido del proyecto:	3
1.1.2 Descripción breve del proyecto:	3
1.1.3 Elementos diferenciadores dentro del nicho de mercado:	3
1.2 Lenguajes empleados:	3
1.3 Distribución:	4
1.4 Requisitos de los clientes:	5
2 Recursos:	6
2.1 Hardware:	6
2.2 Software:	6
2.3 Recursos humanos:	8
2.4 Previsión económica del coste del proyecto:	8
3 Descripción de la aplicación:	9
3.1 Funcionalidad:	9
3.1.1 Requisitos funcionales:	9
3.1.2 Diagramas de flujo:	12
3.1.2.1 Diagrama de flujo de inicio de sesión:	12
3.1.2.2 Diagrama de flujo de usuarios no administradores:	13
3.1.2.3 Diagrama de flujo de usuarios administradores:	14
3.1.3 Diagrama de casos de uso:	15
3.1.4 Funcionamiento general:	15
3.2 Arquitectura:	22
3.2.1 Arquitectura aplicación front-end Angular:	22
3.2.1.1 Estructura de carpetas y archivos:	22
3.2.1.2 Rutas:	22
3.2.1.3 Layout:	23
3.2.1.4 Comunes y variables de entorno:	25
3.2.1.5 Servicios:	25
3.1.2.6 Interfaces:	27
3.2.1.7 Páginas de juegos:	28
3.2.1.8 Página de creación de juegos:	29
3.2.2 Arquitectura aplicación back-end Spring Boot:	31
3.2.2.1 Estructura del proyecto:	31
3.2.2.2 Comunicación entre elementos:	33
3.2.2.3 Creación de palabras:	34
3.2.2.3.1 APIs externas:	34
3.2.2.3.2 Requesters:	36
3.2.2.3.3 Validadores:	37
3.2.2.3.4 Servicios de creación:	37
3.2.2.3.4.1 Creador de fonética:	38
3.2.2.3.4.2 Creador de definición:	40
3.2.2.3.4.3 Creador de similares a fonética:	40

3.2.2.3.4.4 Creador de similares a definición:.....	42
3.2.2.3.4.5 Creador de sinónimo y similares:.....	43
3.3 Arquitectura de seguridad:.....	44
3.4 Arquitectura de emails:.....	48
3.5 Arquitectura de procesos periódicos:.....	51
3.6 Arquitectura de pruebas:.....	52
3.6.1 Tests estándar:.....	52
3.6.2 Tests de componentes:.....	53
3.6.3 Test de controladores:.....	54
4 Diseño de la base de datos:.....	55
4.1 Arquitectura de la base de datos:.....	55
4.2 Diagrama entidad relación:.....	55
4.3 Diagrama de clases:.....	56
4.4 Clases Java y su almacenamiento:.....	56
5 Interfaz:.....	60
5.1 Características generales:.....	60
5.1.1 Mock-ups:.....	60
5.2 Adaptación a dispositivos móviles:.....	63
5.3 Usabilidad:.....	65
5.4 Accesibilidad:.....	66
6. Autoevaluación y conclusiones:.....	67
6.1 Valoración del trabajo y dificultades encontradas:.....	67
6.2 Valoración de la herramienta o aplicación desarrollada:.....	68
6.3 Conclusiones finales:.....	68
7 Bibliografía:.....	70

1 Introducción:

1.1 Objeto del proyecto:

1.1.1 Justificación y sentido del proyecto:

Durante los dos años de estudio del grado superior de desarrollo de aplicaciones web me volví muy aficionado a un juego online llamado *Wordle* (<https://www.nytimes.com/games/wordle/index.html>).

Su funcionamiento es muy sencillo: Hay una palabra al día y el usuario debe intentar descubrirla. Para ello, cuenta con seis oportunidades. La gracia del juego radica en que con cada palabra introducida se marcarán las letras que comparta con la palabra a descubrir y si se encuentran o no en la misma posición.

Para mí se volvió una especie de ritual hacer camino a clase la versión española (<https://lapalabradeldia.com/>), que también cuenta con una versión con tilde, además de la original (en inglés) y cuando comencé a pensar en lo que quería hacer en mi TFG me di cuenta de que quería que tuviera que ver de una u otra manera con esto.

1.1.2 Descripción breve del proyecto:

Daytionary es algo muy parecido a *Wordle*: Un juego online en el que de nuevo nos encontramos con una palabra al día. La gran diferencia en el caso de mi aplicación es que la conocemos desde el principio y lo que hay que adivinar son varias cosas en relación a ella: su fonética, una definición, y un sinónimo de entre un conjunto de alternativas.

Estas opciones se diseñan para confundir al usuario a partir de varios puntos en común como el tipo (nombre, adjetivo, etc.), el número de sílabas de la palabra y sus letras concretas.

Todo generado automáticamente a través de un conjunto de algoritmos inventados y otros reutilizados, la información proveniente de varias APIs de acceso gratuito y cierto nivel de aleatoriedad. Mi único trabajo es revisar lo creado para asegurar que no exista ningún error o errata en él antes de publicarlo.

1.1.3 Elementos diferenciadores dentro del nicho de mercado:

Es un hecho que existen multitud de juegos en línea muy parecidos a *Wordle*, yo mismo he pasado mucho más tiempo del que me gustaría admitir en sudokus, sopas de letras y otros minijuegos. Sin embargo, en el tema educativo el marco se reduce bastante ya que aunque, por ejemplo, *Wordle*, pueda ser utilizado como una fuente de aprendizaje de vocabulario creo que se centra más en la diversión que en ese aspecto. Lo mismo sucede con la mayoría de juegos online de este tipo dejando al usuario que busca aprender sin mucha opción aparte de cursos, videos y otras alternativas más serias, lentas y aburridas.

Dayctionary llena este hueco de mercado, diferenciándose del resto en un primer momento con su aspecto visual, inspirado en exámenes tipo test, apuntes estudiantiles y libros de texto; y a la hora de jugar en su sistema de contestaciones, más enfocado a informar que ha registrar las victorias o las derrotas con detalles como el que en los juegos una vez se ha contestado se muestran las palabras a las que pertenecen las opciones incorrectas y no un marcador de ningún tipo.

Pero estas alternativas son famosas y tienen tantos seguidores por varios motivos que *Dayctionary* no pasa por alto, por ejemplo, su formato diario que invita a visitar la página con frecuencia y la rapidez, en la mayoría de casos los juegos no llevan más de unos minutos.

1.2 Lenguajes empleados:

El proyecto se creará usando dos lenguajes principales:

- Java 20, para la API de Spring Boot 3.1.15 con maven 3 que tendrá acceso directo a la información de la base de datos.
- Typescript, usado con angular v16.2.1 para realizar la aplicación cliente que utilizará la información recibida de la API y le indicará las operaciones a realizar. Manejado a través de Node v18.16.

Además, se utiliza HTML5 junto a CSS para dar estilo a los emails enviados por la API y con SASS en todo el apartado front end de angular.

1.3 Distribución:

El proyecto no ha sido distribuido oficialmente, se ha desplegado un prototipo totalmente funcional de los repositorios de git a través de Render (<https://dashboard.render.com/>) con una base de datos online creada con MongoDB Atlas para probar el correcto funcionamiento e interacción de los proyectos.

Para este despliegue hicieron falta algunos cambios en el proyecto Spring Boot:

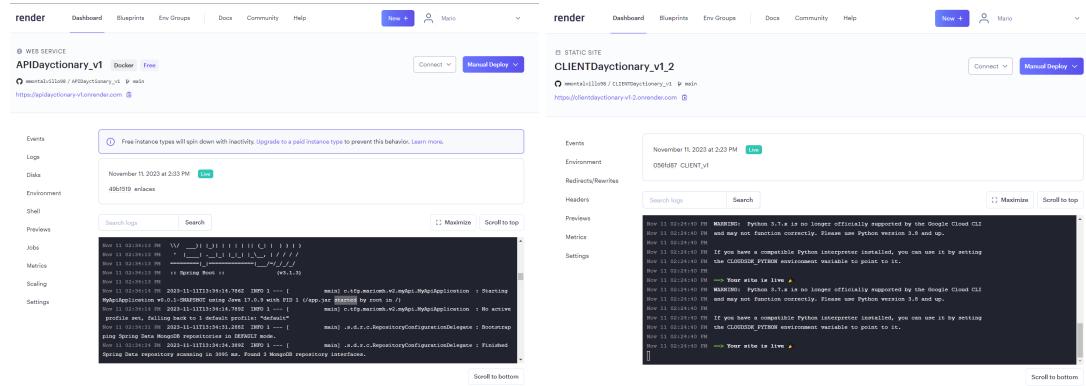
- Bajar la versión de Java a 17 porque era el máximo manejado gratuitamente por la página.
- Inclusión de la etiqueta `@EnableAutoConfiguration(exclude = {MongoReactiveAutoConfiguration.class})` en la clase que contiene el método main (MyApiApplication.java).
- Crear un archivo Docker con la siguiente información:

```
FROM eclipse-temurin:17-jdk-alpine
VOLUME /tmp
COPY target/*.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
EXPOSE 8080
```

- Añadir en application.local los datos de la base de datos MongoDB Atlas:

```
# MONGO BBDD CONFIGURATION:
# LOCAL
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=MyApiDatabase
# MONGO ATLAS
spring.data.mongodb.uri=mongodb+srv://mmontalvillo98:Wlsv5TnxTGdNNKPZ@cluster0.cpdonss.mongodb.net/
spring.data.mongodb.database=Dayctionary_v1
spring.data.mongodb.auto-index-creation=true
# spring.data.mongodb.username=admin
# spring.data.mongodb.password=password
```

Render permite un seguimiento online del log de las aplicaciones así como despliegues de commits específicos del proyecto, lanzamiento de comandos, etc:



1.4 Requisitos de los clientes:

Dayctionary cuenta con dos tipos de acceso:

- Público (de usuario): que da acceso a los juegos diarios y que solo cuenta con el requisito de registrarse con un email verificado y una contraseña.
- Privado (de administrador): que además del acceso a los juegos del día cuenta con la opción de ver toda la información a partir de la que se crean estos no solo de la fecha en cuestión sino de cualquier otro. También tendrá la opción de crear juegos aleatoriamente o a partir de la palabra que deseé.

2 Recursos:

2.1 Hardware:

Para el desarrollo se usó un ordenador portátil Lenovo IdeaPad Z580 de 8 gb de ram con procesador: Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, 2501 Mhz, 2 procesadores principales, 4 procesadores lógicos modificado con un disco SSD de almacenamiento interno Kingston SA400S37240G de 500 gb y un ratón inalámbrico silencioso recargable INPHIC.

2.2 Software:

- Sistema operativo Microsoft Windows 10 Home.
- GitHub desktop: Permite el manejo de GitHub y el control versiones sin necesidad de comandos.
- MongoDB Compass: Interfaz para hacer el acceso y las operaciones en bases de datos MongoDB más sencillas e intuitivas.
- Postman: Utilizado para probar las llamadas a la API.
- Scribbr: generador de citas en línea para la memoria.
- SwaggerEditor: editor de código YAML en línea para diseñar, crear y documentar servicios web Restful.
- MJML editor: editor de código MJML en línea para diseñar emails responsive.
- MongoDB Atlas: utilizado para el despliegue online de la base de datos.
- Render: utilizado para el despliegue online gratuito de las aplicaciones.
- Favicon.cc: creador de favicons gratuito y en línea.
- Visual Studio Code: entorno de desarrollo integrado front-end. Con los plugins:
 - Activitus bar vo.0.47: reduce el tamaño de la barra de actividad de visual studio y permite su personalización dejando un IDE más limpio.
 - Angular language service v16.1.8: Ayuda a la hora de crear código en angular por el lado de las plantillas y clases.
 - Angular schematics v5.4.2: Permite la creación de todo tipo de componentes de angular con sin necesidad de introducir comandos en el terminal.
 - Angular snippets v16.0.1: Provee una serie de snippets (pequeños bloques de texto) que permiten la creación de la base de código de clases, componentes, servicios, etc. de angular.
 - Auto close tag v0.5.14: Actualiza y cierra automáticamente las etiquetas HTML.
 - Better comments v3.0.2: Aporta un conjunto de snippets que agregándolos a los comentarios permiten categorizarlos por tipo, adoptando un color acorde.
 - Error lens v3.14.0: Da un mensaje informativo a los errores marcados por visual studio de manera que son más fáciles de detectar y entender.
 - Paste JSON as code v12.0.46: Genera código (clases, interfaces, etc.) a partir de objetos JSON.
 - Typescript importer v2.0.1: Facilita la codificación realizando todo el trabajo de búsqueda e importación de clases en el proyecto con pocas teclas.
 - Yaml v1.14.0: Provee soporte, formateo y validación del lenguaje YAML (utilizado para la documentación de la API con swagger).
- IntelliJ IDEA: entorno de desarrollo integrado back-end. Plugins:

- CodeMetrics v1.0.4: Encargado de indicar mediante un valor numérico el grado de complejidad de las clases y métodos.
- Conventional commit v0.22.0: Ayuda en la creación de commits para GitHub con un formato comprensible y semi-estándar.
- GitToolBox: Utilizado para poder realizar desde el IDE toda la gestiones de versiones de GitHub.
- SonarLint: Orientado a la creación de código limpio y carente de complejidad innecesaria.
- Librerías proyecto Angular:
 - Angular Material: Provee una lista de componentes y estilos muy útiles para conseguir un aspecto uniforme y atractivo con poco código.
 - Animations CSS: Conjunto de clases CSS para agregar animaciones sencillas a las páginas.
 - SweetAlert2: Creador de avisos estilizados y complejos.
 - SweetAlert2 Theme Dark: Aspecto de SweetAlert modo nocturno u oscuro.
 - Moment: Librería estandarizada para el trabajo con fechas en angular.
 - Compodoc: Generador de documentación automático basado en los componentes y comentarios muy parecido a Javadoc.
- Librerías proyecto Spring Boot:
 - gson (com.google.code.gson) v2.10.1: Utilizada para mapear las respuestas recibidas de las peticiones a las apis externas a las clases creadas.
 - json-simple (com.googlecode.json-simple) v1.1.1: Orientada al manejo de objetos json en java.
 - jjwt-api (io.jsonwebtoken) v0.11.5: Necesaria en todo el proceso de validación y creación de tokens para la seguridad de la API.
 - jjwt-impl (io.jsonwebtoken) v0.11.5.
 - jjwt-jackson (io.jsonwebtoken) v0.11.5.
 - spring-boot-starter (org.springframework.boot).
 - spring-boot-starter-mail (org.springframework.boot): Encargada de gestionar el envío y la creación de emails.
 - spring-boot-starter-thymeleaf (org.springframework.boot): Generador de plantillas híbrido entre java y HTML aprovechado en la creación y el estilizado de los emails enviados por la API.
 - spring-boot-starter-parent (org.springframework.boot) v3.1.3.
 - spring-boot-starter-security (org.springframework.boot).
 - spring-boot-test (org.springframework.boot).
 - spring-boot-web (org.springframework.boot).
 - spring-boot-webflux (org.springframework.boot): Cuenta con una serie de clientes muy útiles para realizar peticiones externas y tests de los propios controllers.
 - lombok (org.projectlombok) v1.18.28: Generador de código sencillo, como constructores, getters o setters; por etiquetas.
 - mapstruct (org.mapstruct) v1.4.2.Final: Mapeador de objetos que no fuí capaz de implementar directamente debido a conflictos con lombok pero cuya lógica utilzo para mis mapeadores de objetos.
 - mockito-core (org.mockito) v2.21.0: Que permite el testeo de la lógica de los servicios sin preocuparse por los datos.

- opennlp-tools (org.apache.opennlp) v1.9.0: Conjunto de herramientas que proveen una serie de métodos útiles para todo el tratamiento y comparación de palabras a la hora de crear los juegos.
- mailo (dev.distche): Librería alternativa para crear y enviar emails que usa MJML (API con un lenguaje dedicado a simplificar la maquetación y el estilizado de emails). Preferí usar la librería mail de spring en conjunto con thymeleaf como alternativa porque no me permitía tanta lógica. MJML, por ejemplo, no ofrece tratamiento de bucles.

2.3 Recursos humanos:

Diferenciados en tres tipos:

- Formativos: entre los que destacan los siguientes cursos de *udemy*:
 - Angular de cero a experto: 60 h.
 - Spring framework y Spring Boot desde cero a experto: 50 h.
 - Microservicios Spring cloud Eureka y Angular Full-stack: 30 h.
 - Angular & Spring Boot: Creando una web app Full-stack: 25 h.
 - Learn Swagger and OpenAPI specification: 2 h.
- Código: dividido por proyectos:
 - Spring Boot: previstas 100 / realizadas 175 h.
 - Angular: previstas 75 / realizadas 100 h.
- Documentales: previstas 25 / realizadas 50 h.

2.4 Previsión económica del coste del proyecto:

El proyecto tuvo un coste total de 0 € ya que no utilicé ninguna herramienta de software o hardware de pago con la que no contara y las APIs utilizadas para recibir la información necesaria para crear los juegos son todas públicas, así como las aplicaciones usadas para el despliegue del proyecto.

3 Descripción de la aplicación:

3.1 Funcionalidad:

3.1.1 Requisitos funcionales:

Id	Nombre del requisito	Descripción	Observaciones
RF-01	Iniciar sesión usuario	El sistema debe permitir a los usuarios iniciar sesión.	Credenciales: email y contraseña.
RF-02	Registrar usuario	El sistema debe permitir a los usuarios registrarse introduciendo un email y una contraseña.	Se validará que el email no está en uso, habrá un segundo campo de confirmar contraseña para asegurar que el usuario introduce los datos deseados y existirá un checkbox que permitirá al usuario activar las notificaciones para recibir un email con la publicación de cada nuevo juego.
RF-03	Iniciar sesión para jugar	El sistema debe impedir a los usuarios acceder a los juegos a no ser que haya iniciado sesión.	En caso de que intenté acceder sin haberse autenticado correctamente se le reenviará al login.
RF-04	Autenticación no accesible para autenticados.	El sistema debe impedir a los usuarios acceder al login o a registrarse una vez el usuario ha iniciado sesión o se ha registrado.	La sesión permanecerá iniciada hasta que el usuario la cierre o el token jwt guardado en el local storage caduque.

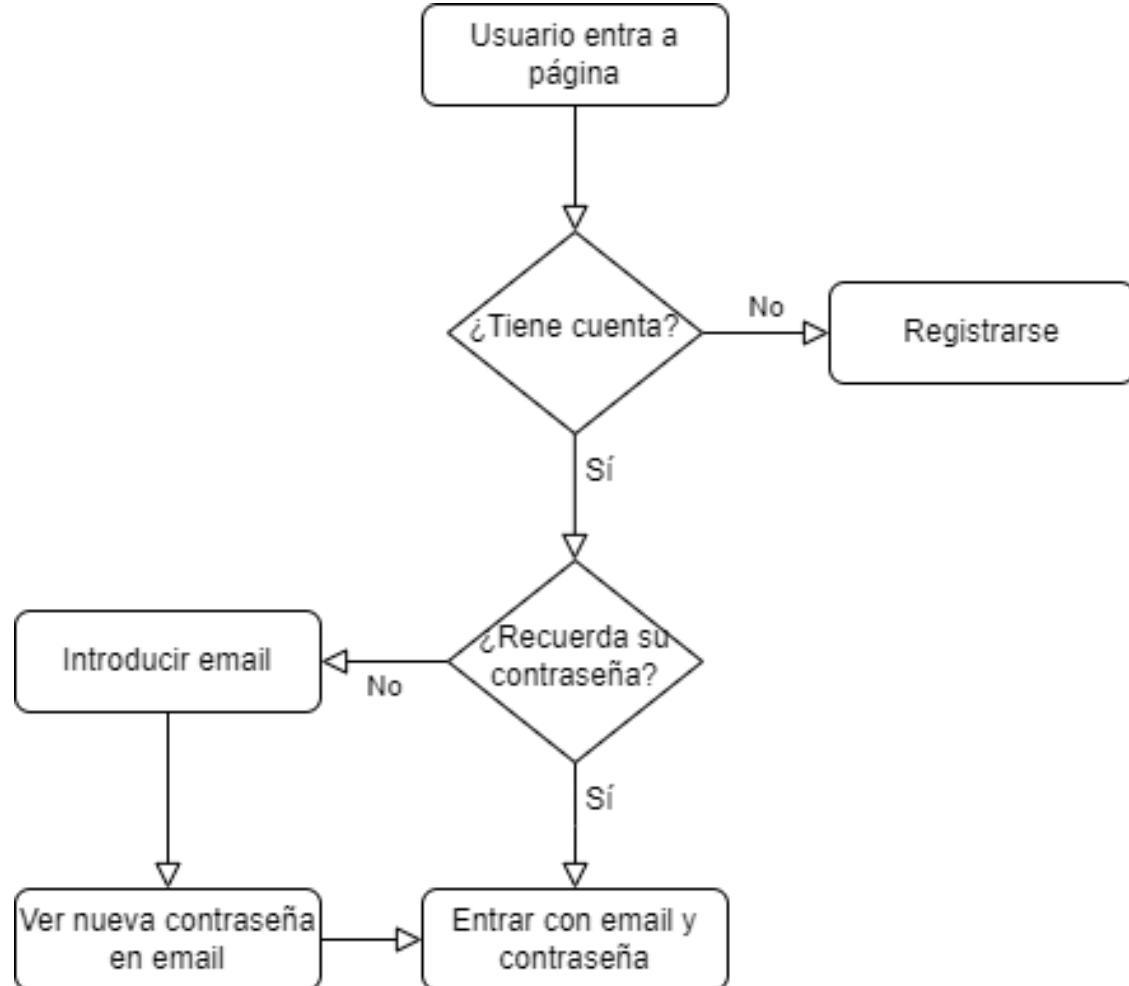
RF-05	Cambiar datos cuenta	<p>El sistema permitirá al usuario una vez ha iniciado sesión modificar sus datos.</p>	<p>Los datos modificables son: email, contraseña, notificaciones. Tienen las mismas validaciones que al crear usuario.</p>
RF-06	Borrar usuario	<p>El sistema permitirá al usuario una vez ha iniciado sesión borrar su cuenta.</p>	<p>El botón contará con un pop-up para preguntar si el usuario está seguro. Una vez borrada una cuenta no se puede recuperar. El email utilizado en esa cuenta sin embargo pasa a estar disponible para el uso en otra cuenta.</p>
RF-07	Envío de código de confirmación de email	<p>El sistema enviará un código al email del usuario antes de dejarle registrarse para confirmar que es suyo.</p>	<p>El código aleatorio deberá introducirse exactamente igual que en el email. El código puede tener un tiempo limitado de validez.</p>
RF-08	Contraseña olvidada	<p>El sistema permitirá al usuario recibir una nueva contraseña a su email en caso de haber olvidado la suya</p>	<p>La nueva contraseña se genera aleatoriamente. Es un código con el mismo formato que el de confirmación.</p>

RF-09	Elegir juego	El sistema permitirá al usuario elegir el juego que quiera de los disponibles.	Cada uno de los juegos está relacionado con algo de la palabra. Planeados: Definición, fonética y sinónimo.
RF-10	Jugar	El sistema permitirá al usuario elegir de entre varias opciones en cada juego la que él crea que pertenece a la palabra del día en cuestión.	Por ejemplo: se presentan varias definiciones y el usuario elige la que cree que es la de la palabra.
RF-11	Ver respuesta correcta	El sistema permitirá al usuario comprobar una vez contestado un juego si la respuesta elegida es correcta o no e información referente a cada opción.	Por ejemplo: en el caso de las definiciones se mostrará la palabra a la que pertenece cada definición.
RF-12	Notificaciones	El sistema enviará al email facilitado por el usuario (en caso de que él lo haya marcado) un email cada vez que haya disponible un set de nuevos juegos.	El email contendrá un enlace para acceder con un solo click a la página y jugar.
RF-13	Administrador	El sistema dará la opción a los usuarios con rol de administrador de realizar varias acciones no disponibles para los usuarios normales.	Estas opciones estarán disponibles como un botón del menú que no se muestra a los usuarios normales.

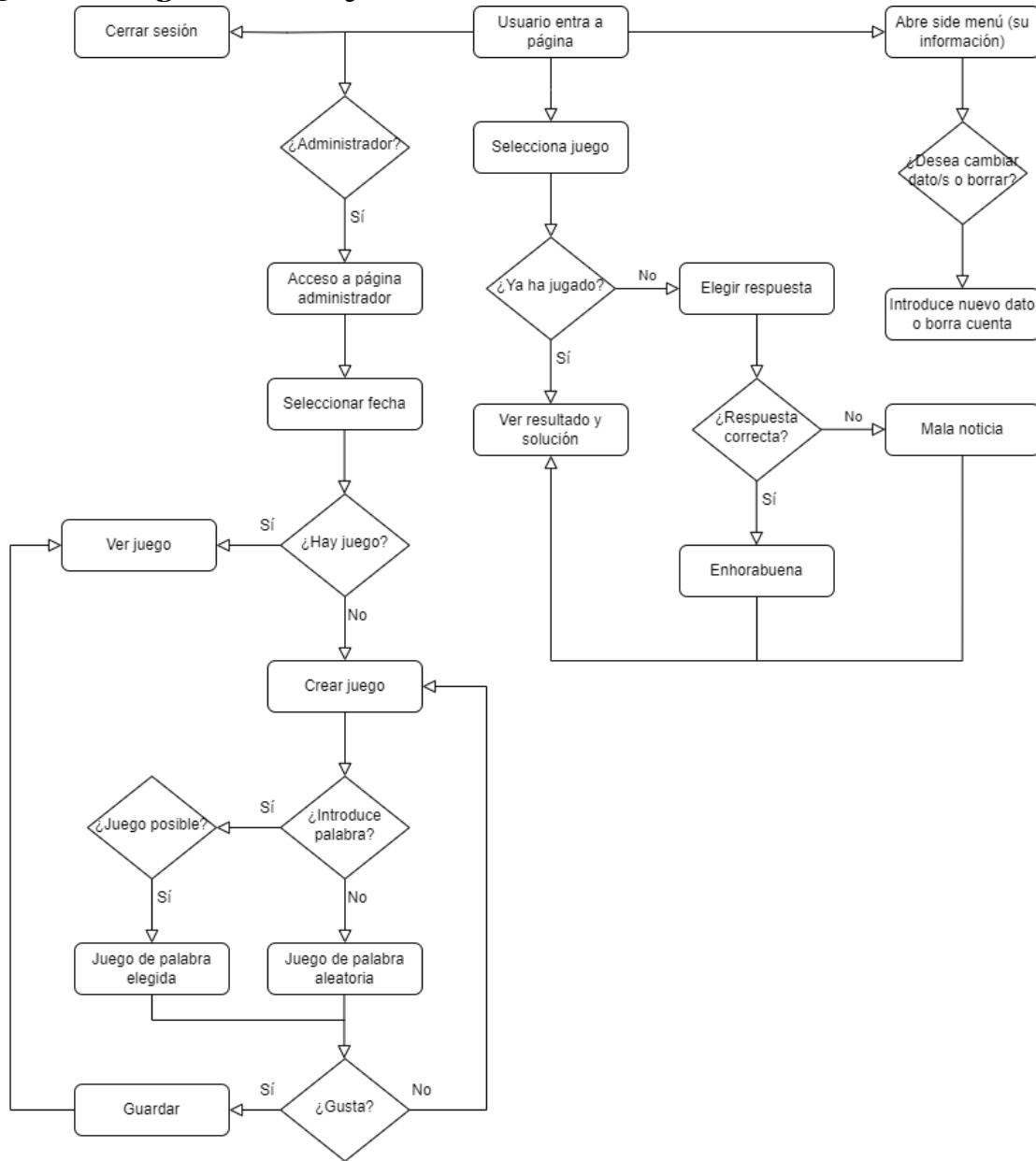
RF-14	Crear juego	<p>El sistema permitirá a los usuarios administradores solicitar nuevos juegos a la API, revisarlos y preparar su publicación para una fecha.</p>	<p>Se podrá crear a partir de una palabra , en cuyo caso puede crearse o no dependiendo de si hay o no información suficiente; o aleatoriamente. El proceso de generación puede llevar un tiempo.</p>
-------	-------------	---	---

3.1.2 Diagramas de flujo:

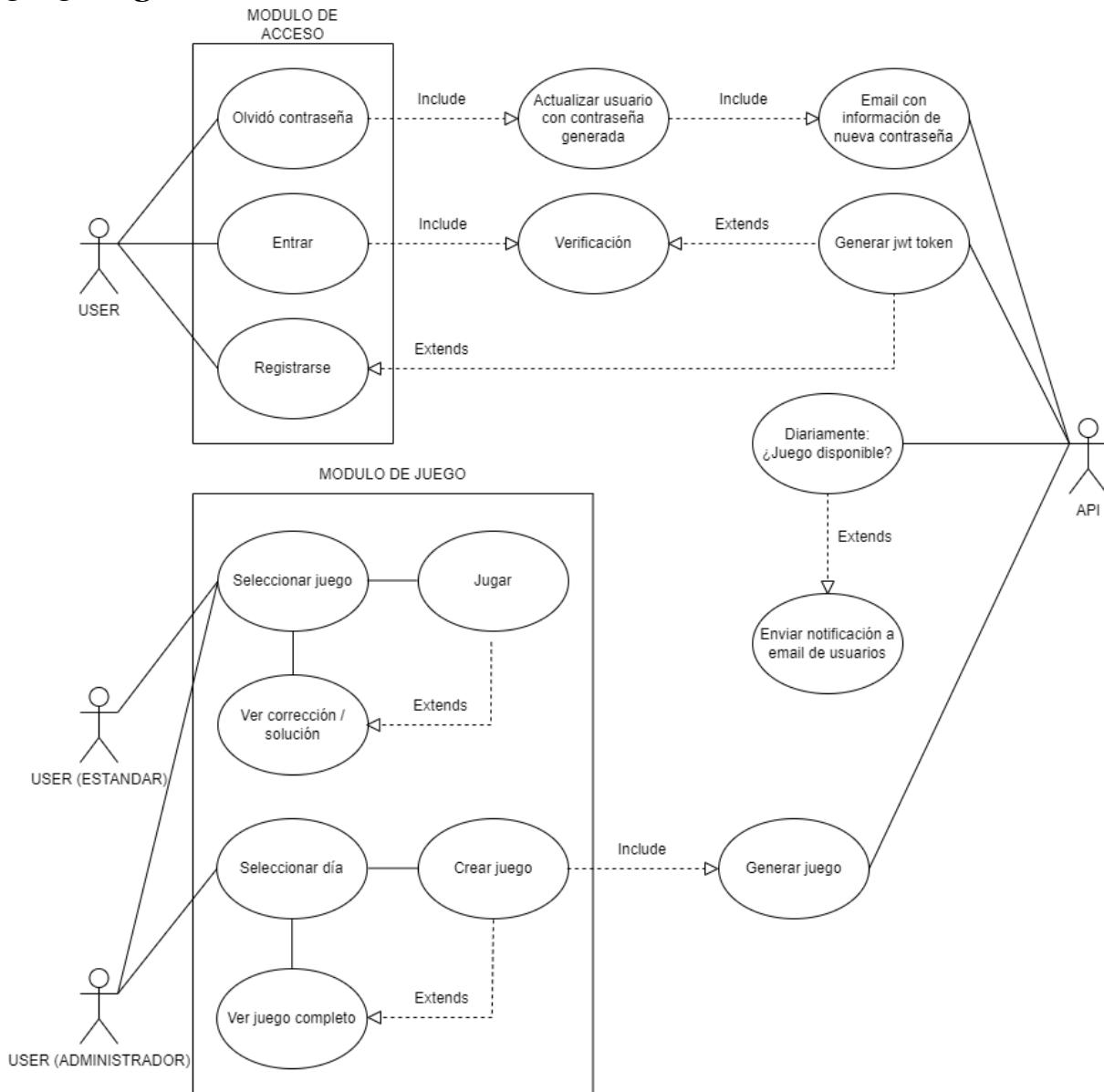
3.1.2.1 Diagrama de flujo de inicio de sesión:



3.1.2.2 Diagrama de flujo de usuarios:



3.1.3 Diagrama de casos de uso:



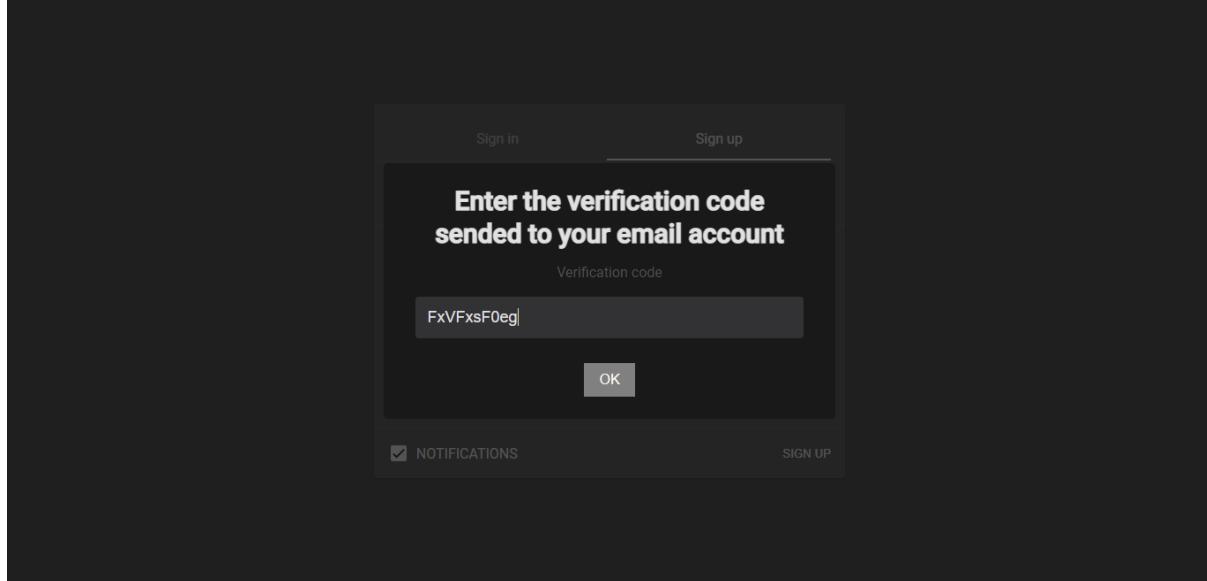
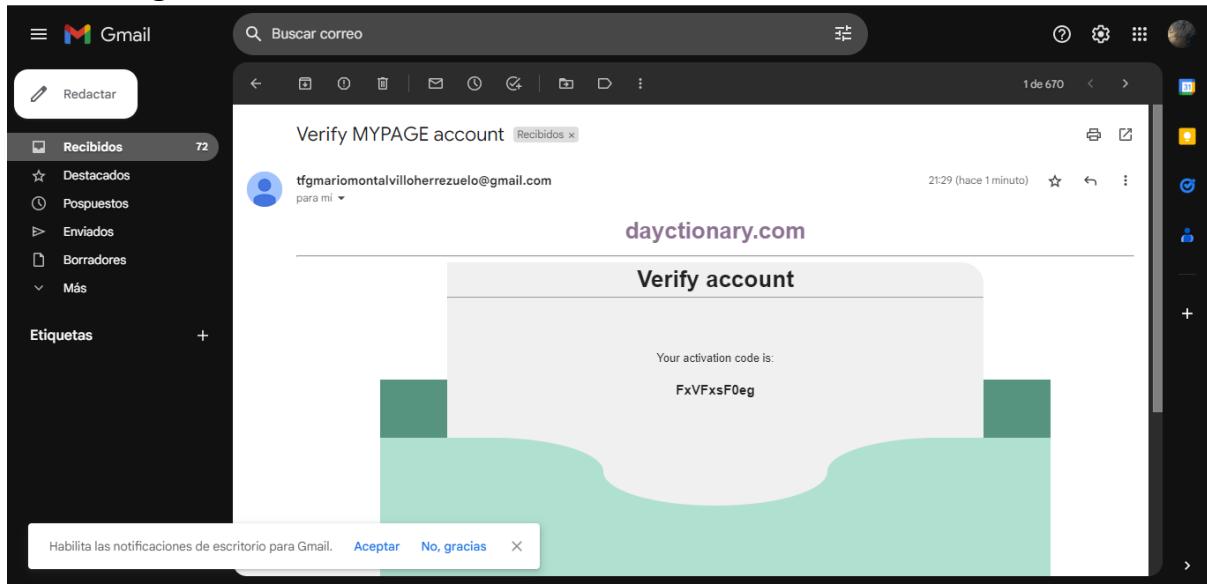
3.1.4 Funcionamiento general:

Para acceder a *Dayctionary* y poder disfrutar de los juegos es necesario iniciar sesión (explicación extendida del funcionamiento de la seguridad en [3.3 Arquitectura de seguridad](#) y en el manual de usuario adjuntado junto a la memoria).

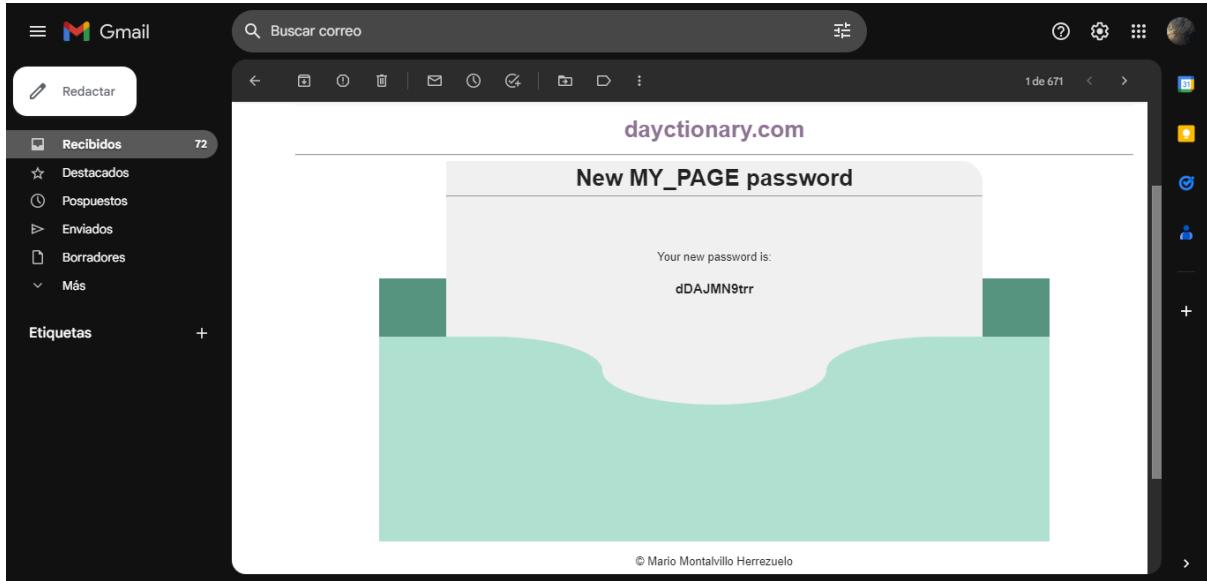
En caso de no contar con una cuenta, para crearla solo hará falta un email al que se tenga acceso, introducir una contraseña válida y si se quiere recibir notificaciones cada día de los juegos disponibles marcar la casilla correspondiente.

Es necesario un email funcional por dos motivos:

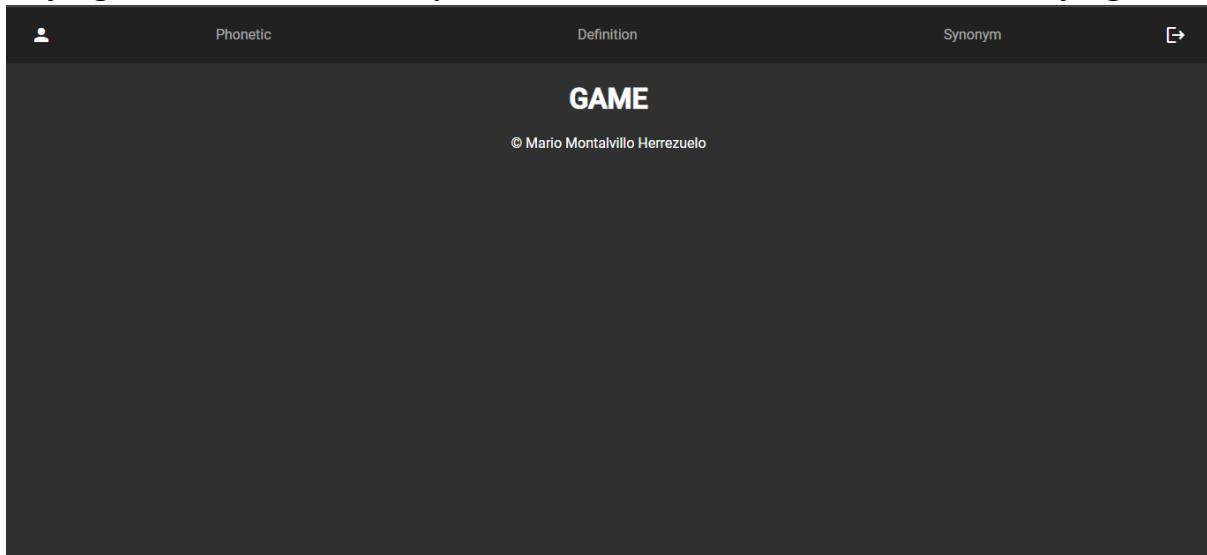
- Al registrarnos se enviará por seguridad un código de verificación aleatorio que el usuario deberá introducir en una ventana emergente que surgirá al dar a registrarse.



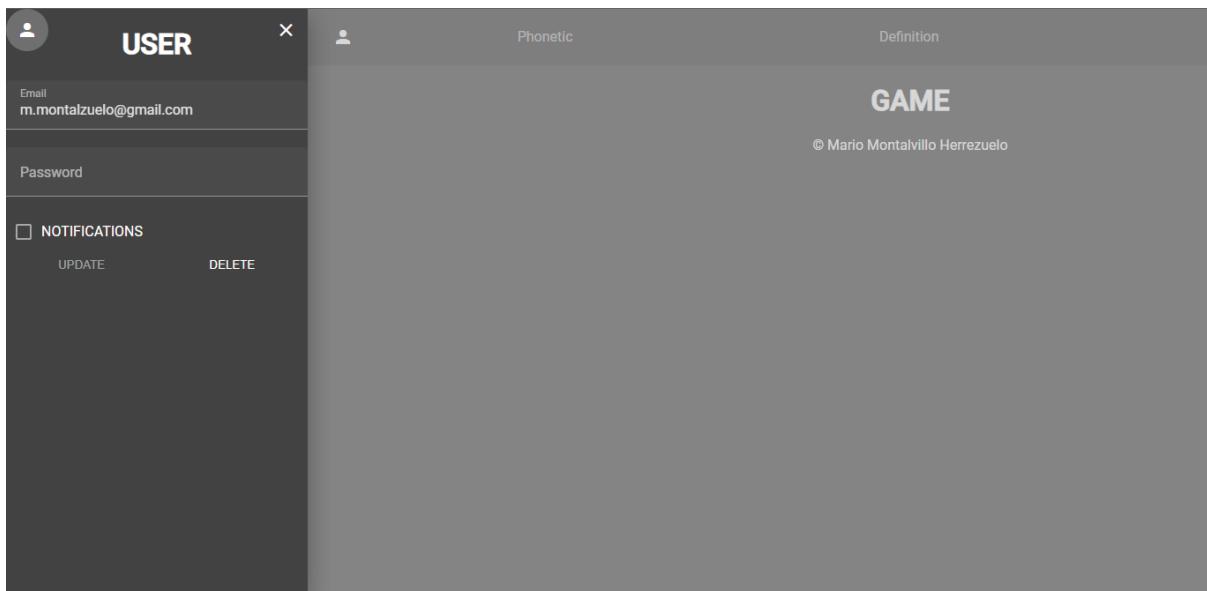
- En caso de no recordar la contraseña el usuario contará con la posibilidad de acceder pidiendo un cambio de contraseña, el nuevo credencial se enviará a su email de una manera muy similar a la del código de verificación de cuenta. Para acceder el usuario deberá introducir esa nueva contraseña al iniciar sesión.



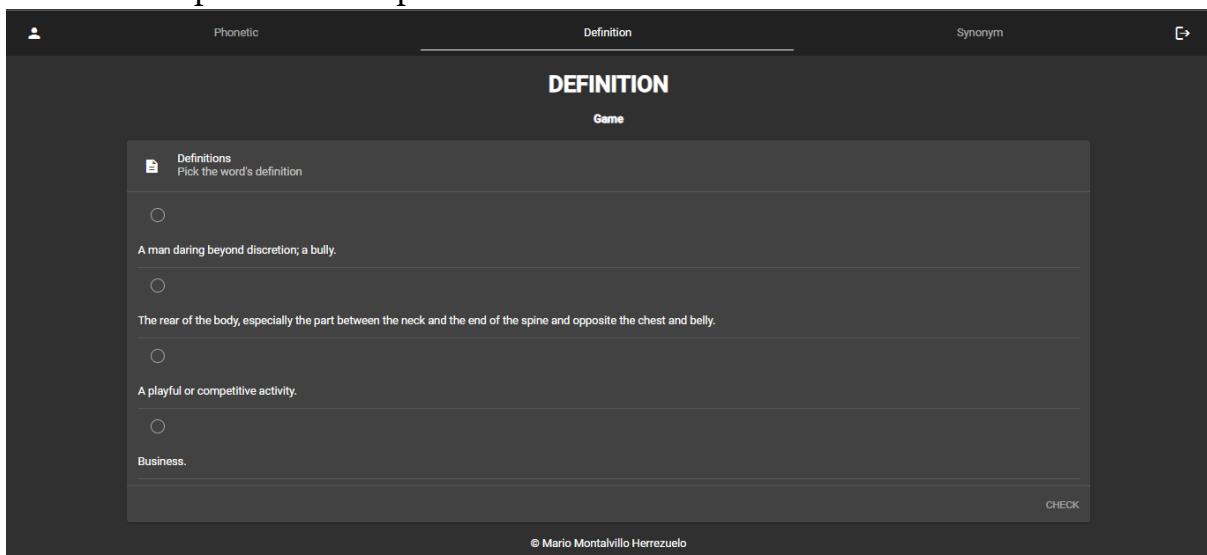
Una vez iniciada sesión el usuario contará con un token JWT y tendrá acceso a la página principal desde la que podrá ver la palabra del día (en el caso de la captura más abajo “game”, hay una palabra al día y los juegos tratarán sobre ella) y acceder a los juegos, al menú de usuario y en caso de ser administrador al creador de juegos:



- Menú de usuario (accesible desde el ícono en la esquina superior derecha de la pantalla): un formulario donde se muestra la información del usuario (email y notificaciones activas/inactivas) y se modificar esos datos y la contraseña o en caso de que el usuario lo desee borrar su cuenta.



- Juegos (ubicados junto a la derecha del ícono de menú de usuario): existen tres. Definición, fonética y sinónimo. El funcionamiento en todos los casos será el mismo. Se presentan varias opciones al usuario y este tendrá que elegir la que cree que pertenece la palabra del día (en el ejemplo: "game").
 - Definición: se muestran cuatro definiciones de las que solo una pertenece a la palabra del día.



- Fonética: se ponen a disposición del usuario cuatro audios que podrá reproducir y escuchar con los botones a la derecha. Solo uno dice la palabra del día.

The screenshot shows the 'PHONETIC' game interface. At the top, there are tabs for 'Phonetic', 'Definition', and 'Synonym'. Below the tabs, the word 'PHONETIC' is displayed in large letters. Underneath it, the word 'Game' is shown. A section titled 'Audios' with the subtitle 'Pick the word's audio' contains four options, each with a radio button and a speaker icon. At the bottom right of this section is a 'CHECK' button.

- Sinónimo: cuatro palabras de las que solo una es un sinónimo de la palabra del día.

The screenshot shows the 'SYNONYM' game interface. It has the same tab structure as the previous screen. Below the tabs, the word 'SYNONYM' is displayed in large letters. Underneath it, the word 'Game' is shown. A section titled 'Related words' with the subtitle 'Pick the word's synonym' contains four options, each with a radio button. At the bottom right of this section is a 'CHECK' button.

Al responder se mostrará al usuario un mensaje emergente en el que se anuncia si ha respondido correctamente o no al juego y se modificará el formulario para indicar la opción correcta y, en caso de haberla, información adicional sobre las opciones (en las definiciones y fonéticas se muestra la palabra a la que pertenecen).

The image contains two side-by-side screenshots of the application's feedback mechanism. The left screenshot shows a 'Good job!' message with a checkmark, indicating a correct answer. The right screenshot shows a 'Too bad...' message with a cross, indicating an incorrect answer. Both messages include a 'OK' button at the bottom.

DEFINITION

Game

Definitions
Pick the word's definition

brave

A man daring beyond discretion; a bully.

game

A playful or competitive activity.

back

The rear of the body, especially the part between the neck and the end of the spine and opposite the chest and belly.

biz

Business.

✓

CHECK

© Mario Montalvillo Herrezuelo

PHONETIC

Game

Audios
Pick the word's audio

gain

game

gone

gum

✗

CHECK

© Mario Montalvillo Herrezuelo

- Creador de juegos (ubicado a la derecha de los juegos bajo el nombre “admin” solamente para usuarios con el rol administrador):

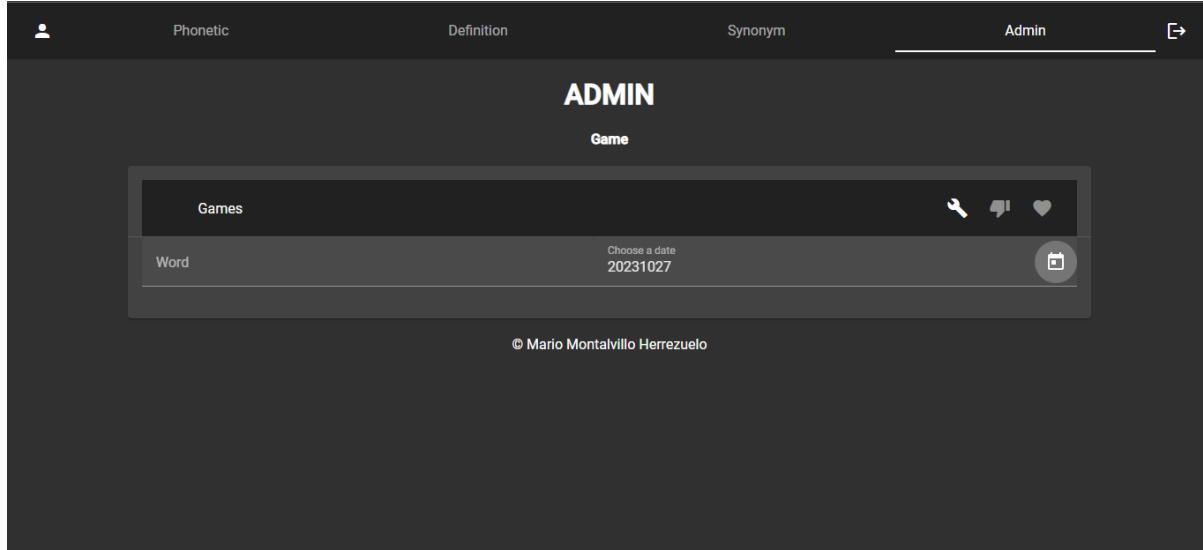
ADMIN

Game

Games		Word	Date	Actions
Syl 2	Word game	game	Choose a date 20231026	🔍 🗑️ ❤️
Syl 1	Word gain	gain		🔊 Synonym racket
Syl 1	Word gum	gum		🔊 Similar drudgery
Syl 2	Word gone	gone		🔊 Similar sega

Cuenta con un formulario que proveerá toda la información relativa a una palabra y sus juegos. Al entrar el formulario mostrará la información del día actual pero con el campo fecha se podrá elegir el día deseado.

En caso de haber un juego planeado para ese día, sucederá lo mismo, el usuario podrá ver su información, en caso de que no el formulario pasará a contar solamente con dos campos: La palabra y la fecha.

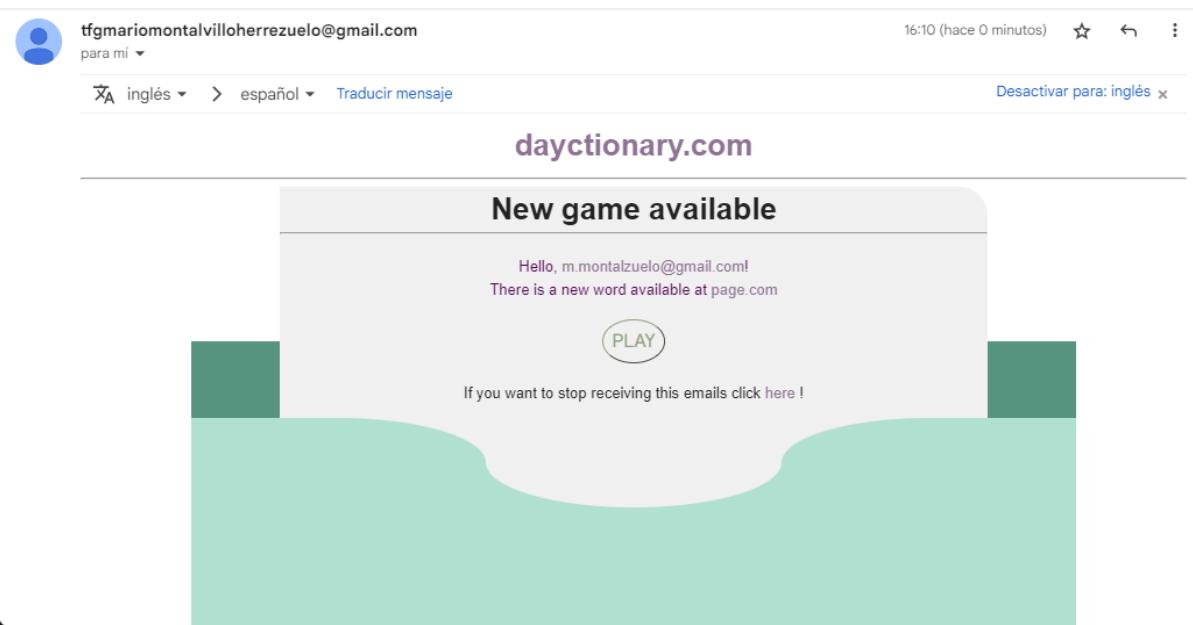


El usuario podrá crear un juego para ese día de dos maneras:

- Introduciendo una palabra, la palabra que será la palabra de ese día si es que hay información suficiente información suficiente para crear los juegos.
- Sin introducir palabra. En cuyo caso siempre se devolverá un juego pero de una palabra aleatoria (a no ser que se supere el número de peticiones diarias a una API y una petición devuelva un 429 “Too many requests”).

Una vez creado el juego se mostrará al usuario. Los campos del formulario estarán habilitados para que pueda realizar correcciones si es necesario.

En caso de que no le guste el juego el usuario puede borrarlo y crear otro, en caso de que sí, se guardará y estará disponible para jugar en la fecha señalada. Este día se informará a los usuarios que tengan habilitadas las notificaciones de que hay un juego disponible mediante el siguiente email:



3.2 Arquitectura:

3.2.1 Arquitectura aplicación front-end Angular:

3.2.1.1 Estructura de carpetas y archivos:

```

    < src
      < app
        > guards
        > interfaces
        > modules
        > pages
        > services
        > shared
        TS app-routing.module.ts
        <> app.component.html
        <> app.component.scss
        TS app.component.spec.ts
        TS app.component.ts
        TS app.module.ts
  
```

Elegí esta distribución porque reducía al mínimo el número de carpetas duplicadas y utilizadas en general haciendo que moverse por el proyecto sea lo más sencillo posible. Este sistema, acompañado de unos nombres que indican el tipo con el siguiente formato: nombre.tipo.extensión donde tipo puede ser guard, interface, module, routing, page, component o service ayuda a ser consciente en todo momento qué objeto se está tratando, dónde empieza y acaba su funcionalidad y qué debe hacer.

Las páginas cuentan además con otra carpeta a mayores además del archivo typescript, el html y los estilos scss (si lo tienen): _components. Donde se almacenan todos los componentes utilizados en esa página.

3.2.1.2 Rutas:

El cliente de *Dayctionary* se divide en dos partes diferenciables gracias a su sistema de rutas base, en app-routing.module.ts: autentificación (/auth), gestionada por auth.module y juegos (/word), por word.module.

En caso de que el usuario intente entrar a alguna ruta que no exista se le redirigirá a los juegos ({path: '**', redirectTo: "word"}).

- Autenticación: Cuenta con dos páginas: inicio de sesión (/auth/signIn) y registro (/auth/signUp). Y en caso de encontrar una ruta desconocida redirigirá a inicio de sesión.

```
const routes: Routes = [
  {
    path: '',
    component: AuthPage,
    children: [
      { path: 'signIn', component: SignInComponent },
      { path: 'signUp', component: SignUpComponent },
      { path: '**', redirectTo: 'signIn' }
    ]
  }
]

@NgModule({
  declarations: [],
  imports: [
    RouterModule.forChild(routes)
  ],
  exports: [
    RouterModule
  ]
})
export class AuthRoutingModule { }
```



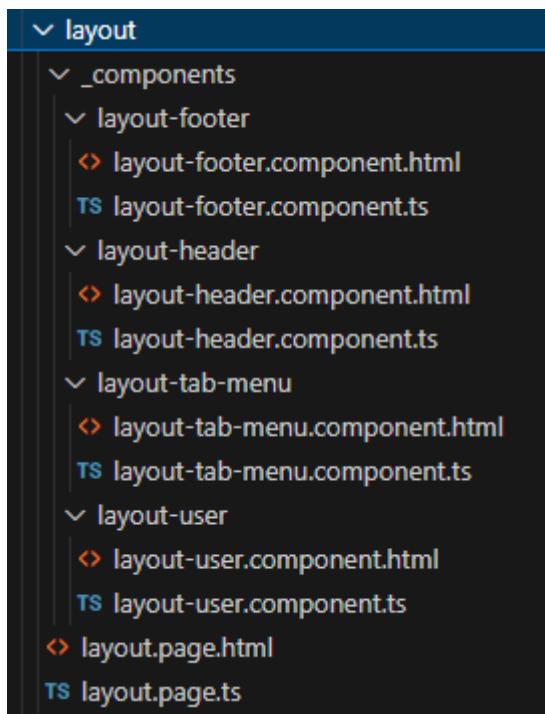
```
const routes: Routes = [
  {
    path: '',
    component: LayoutPage,
    children: [
      { path: '', component: WordPage},
      { path: 'definition', component: DefinitionPage },
      { path: 'phonetic', component: PhoneticPage },
      { path: 'synonym', component: SynonymPage },
      { path: 'admin', component: AdminPage },
      { path: '**', redirectTo: '' }
    ]
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule],
})
export class WordRoutingModule { }
```

- Juegos: donde solo tendrán acceso usuarios que han iniciado sesión, es decir, que cuentan con un token JWT válido ([3.3 Arquitectura de seguridad](#)). Cuenta con la página base (WordPage en /word), a la que se redirigirá al usuario en caso de introducirse una ruta desconocida; una página por cada juego: definición (DefinitionPage en /word/definition), fonética (PhoneticPage en /word/definition), sinónimo (SynonymPage en /word/synonym); y la página de gestión de juegos del administrador (AdminPage en /word/admin).

Todo bajo el marco del componente LayoutPage ([3.2.1.3 Layout](#)).

3.2.1.3 Layout:



Además de contener todo el marco visual de las páginas pone a disposición del usuario, a través de un solo click, su menú lateral de información (layout-user), donde modificar los datos de su cuenta o borrarla; y los distintos juegos y en caso de tener el acceso el menú de creación de juegos (layout-tab-menu).

Su estructura HTML, gracias a la librería angular material, es muy sencilla: Simplemente cuenta con un elemento mat-sidenav-container que ocupa toda la pantalla con la directiva fullscreen y con un mat-sidenav donde se introduce el menú de usuarios y toda su lógica; un mat-toolbar o barra de herramientas con varios iconos que permiten, de izquierda a derecha: acceder al menú de usuario, acceder a los juegos (explicado más adelante), al menú de administrador; y cerrar sesión. Y un sencillo div donde se introduce junto a la cabecera y el pie de la página la etiqueta router-outlet, es decir la etiqueta del componente a mostrar indicado en las rutas.

el pie de la página la etiqueta router-outlet, es decir la etiqueta del componente a mostrar indicado en las rutas.

```

<mat-sidenav-container fullscreen>
  <mat-sidenav #sidenav mode="push" class="w-25 m-w-25">
    <div class="row">
      <button mat-icon-button (click)="sidenav.close()">
        <mat-icon>person</mat-icon>
      </button>
      <h1 class="text-center spacer">USER</h1>
      <button mat-icon-button (click)="sidenav.close()">
        <mat-icon>close</mat-icon>
      </button>
    </div>
    <layout-user></layout-user>
  </mat-sidenav>
  <mat-toolbar>
    <button mat-icon-button (click)="sidenav.toggle()">
      <mat-icon>person</mat-icon>
    </button>
    <layout-tab-menu class="spacer"></layout-tab-menu>
    <button mat-icon-button (click)="logout()">
      <mat-icon>logout</mat-icon>
    </button>
  </mat-toolbar>
  <div>
    <layout-header [game]="game" [word]="todayWord"></layout-header>
    <router-outlet></router-outlet>
    <layout-footer></layout-footer>
  </div>
</mat-sidenav-container>

```

```

export class LayoutTabMenuComponent {
  public menuItems = [
    {
      route: '/word/phonetic',
      title: 'Phonetic',
      icon: 'headphones'
    },
    {
      route: '/word/definition',
      title: 'Definition',
      icon: 'description'
    },
    {
      route: '/word/synonym',
      title: 'Synonym',
      icon: 'invert_colors'
    }
  ];
  get isAdmin(): boolean | undefined {
    return this._authService.user?.roles?.some(role => role === "ADMIN")
  }
  constructor(
    private _authService: AuthService
  ) {}
}

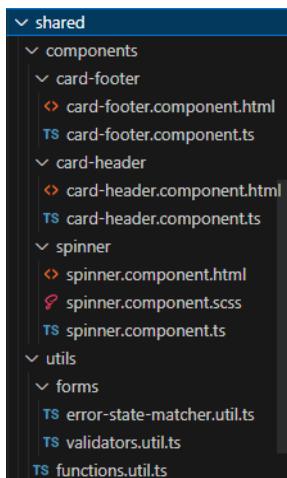
```

Un componente digno de mención que forma parte del layout es el navegador de juegos. Implementado mediante la directiva de angular mat-nav-tab-bar permite mediante muy poco código, simplemente un array de objetos menuItem como los que se pueden observar a la izquierda; la implementación de un menú de navegación muy estilizado y perfectamente funcional.

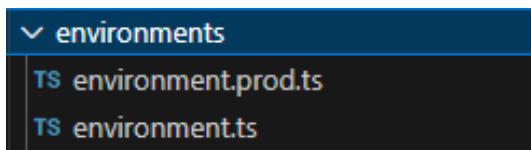
Ese sistema de objetos es en el que me fijaría para realizar todo mi manejo de cabeceras y pies de página.

```
<nav mat-tab-nav-bar [tabPanel]="tabPanel">
  <a mat-tab-link *ngFor="let item of menuItems" routerLinkActive #rla="routerLinkActive" mat-tab-link
    [active]="rla.isActive" [routerLinkActiveOptions]="{{exact: true}} [routerLink]="item.route">
    <mat-icon>{{item.icon}}</mat-icon>
    <span>{{item.title}}</span>
  </a>
  <a *ngIf="isAdmin" routerLinkActive #rla="routerLinkActive" mat-tab-link [active]="rla.isActive"
    [routerLinkActiveOptions]="{{exact: true}} [routerLink]="/word/admin">
    <mat-icon>supervisor_account</mat-icon>
    <span>{{'Admin'}}</span>
  </a>
</nav>
<mat-tab-nav-panel #tabPanel></mat-tab-nav-panel>
```

3.2.1.4 Comunes y variables de entorno:



A la hora de trabajar en proyectos grandes es muy común encontrar funcionalidad repetida y compartida por varios elementos. Con fin de no repetir ese código creé la carpeta comunes donde se hayan componentes como el cabecera o el pie de los juegos (card-header y card-footer respectivamente) y varios elementos útiles como los validadores de formularios (validators.util) y varias funciones repetidas a lo largo de toda la aplicación entre las que destacan: *todayDate()*, que a través de la librería moment devuelve la fecha con el formato que utilizo en el proyecto; y *shuffle()*, un ordenador aleatorio de arrays, muy útil para barajar las opciones recibidas por la API y no mostrarlas siempre en el mismo orden donde la primera es la correcta.



```
export const environment = {
  production: false,
  API_PATH: "http://localhost:8888"
}
```

También, de cara a trabajar en producción y desarrollo si se daba la situación y alternar la información necesaria creé la carpeta “enviroments” que cuenta con un archivo para cada una de estas opciones con un objeto constante llamado enviroment que tendrá una propiedad por cada dato necesario. Esta alternancia entre archivos se llevará a cabo automáticamente gracias a la siguiente información introducida en el archivo raíz del proyecto angular.json.

```
"configurations": {
  "production": {
    "fileReplacements": [
      {
        "replace": "src/enviroments/enviroments.ts",
        "with": "src/enviroments/enviroments.prod.ts"
      }
    ],
  }
},
```

3.2.1.5 Servicios:

No está plasmado en la carpeta servicios de ninguna forma porque pensaba que solo añadía complejidad debido al bajo número de servicios con los que cuenta el proyecto pero existen tres tipos:

- De peticiones: la mayoría, los encargados de realizar las llamadas a la API y que en la mayor parte de casos no almacenan ningún tipo de lógica o

información más allá de las rutas. Uno de estos sería `synonym.service.ts`, que solo cuenta con el método que realiza la petición de los sinónimos del día.

```
@Injectable({ providedIn: 'root' })
export class SynonymService {

  http = inject(HttpClient);

  public getTodaySynonyms(): Observable<SynonymOption[]> {
    return this.http.get<SynonymOption[]>(`${environment.API_PATH}/mini/synonyms/${todayDate()}`);
  }
}
```

- De información: proporcionan apoyo visual de lo sucedido en un proceso externo a ellos. A esta categoría pertenecen `loading.service.ts`, que se encarga de bloquear la pantalla y mostrar un spinner, muy útil para informar el usuario de un proceso de carga; y `response.service.ts`, que se sirve de `sweetalert2` para crear ventanas emergentes con información, avisos y hasta formularios sencillos como el utilizado para pedir el código de verificación de email de usuario. Destaca el método `showMessage`, que crea una ventana emergente con un título (`title`), un texto (`info`) y un ícono (“`type`”, entre sus opciones están `success`, `warning` y `error`); y `showError`, que mostrará la información de una excepción lanzada por la API.

```
public showMessage(title: string, info: string, type: SweetAlertIcon) {
  Swal.fire(title, info, type);
}
```

- De control: `day.service.ts`. El encargado de manejar el servicio de usuarios, juegos y respuestas, permitiendo que los demás componentes no tengan que implementar diversos servicios para acceder a esta información ni para realizar las peticiones que necesiten. Permite, entre otras cosas, unificar las respuestas de los usuarios en un solo método compartido con toda la lógica.

```
@Injectable({ providedIn: 'root' })
export class DayService {

  private _word!: string;
  private _gameId!: string;
  private _answer!: AnswerDone | null;
  private _loading: boolean = false;
  private _available: boolean = true;

  constructor(
    protected authService: AuthService,
    protected gameService: GameService,
    protected answerService: AnswerService,
    protected loadingService: LoadingService,
    protected responseService: ResponseService
  ) {}

  get date(): string | null {
    return localStorage.getItem('date');
  }

  get todayWord(): string {
    if (!this._word || !this.isToday()) {
      this.reload();
    }
    return this._word;
  }

  get todayAnswers(): AnswerDone | null {
    return this._answer;
  }

  get user(): User {
    return this.authService.user!;
  }
}
```

Su funcionamiento comenzará después de que un usuario inicie sesión. La página intentaría encontrar la palabra del día (`todayWord`) y al no encontrarla se desencadenaría todo el proceso de descarga (`reload()`).

En este proceso el primer paso, después de lanzar el spinner para evitar que el usuario pueda realizar ninguna acción, será guardar la fecha actual en el almacenamiento local (`localStorage`).

Después se procederá a buscar la palabra del día y el id de juego y, en caso de que exista, la respuesta hecha por el usuario (`reloadAnswer()`, petición separada debido a que se lanza también al entrar a un juego y al responder).

Una vez recibida la información se almacenará en este servicio hasta que o bien el día acabe (validado con el método `isToday()`) o el usuario recargue la página, momento en el que se reiniciará el proceso.

```

private reload(): void {
    if (!this._loading && this._available && !this.user) {
        this._loading = true;
        this.loadingService.loadingShow();
        localStorage.setItem('date', todayDate());
        this.gameService.getTodayWord().subscribe((game) => {
            this._word = game.word;
            this._gameId = game.id!;
            this._loading=false;
            this.loadingService.loadingHide();
            this.reloadAnswer();
        }, (error) => {
            this._available = false;
            this.responseService.showResponse(error);
            this.loadingService.loadingHide();
            this._loading=false;
        });
    }
}

public reloadAnswer(): void {
    this.loadingService.loadingShow();
    this.answerService.getTodayAnswer(this.user.id!, this._gameId).subscribe((answer) => {
        this._answer = answer;
        this.loadingService.loadingHide();
    }, (error) => {
        this._answer = null;
        this.loadingService.loadingHide();
    });
}

public answer(miniGame: Minigame, choice: string) {
    this.loadingService.loadingShow();
    if (this.isToday()) {
        this.answerService.answerMiniGame(this.user.id!, miniGame, choice, this._gameId!)
            .subscribe(data => [
                this._answer = data;
                this.loadingService.loadingHide();
                this.responseService.showResult(choice === this._word);
            ], (error) => {// The user already answer or not exists game
                console.log(error);
                this.loadingService.loadingHide();
                this.reload();
                this.responseService.showResponse(error);
            });
    } else {
        this.reload();
    }
}

private isToday(): boolean {
    return this.date === todayDate();
}

```

3.1.2.6 Interfaces:

```

export interface User {
    id?: string,
    mail?: string,
    password?: string,
    notifications?: boolean,
    version?: number,
    roles?: string[]
}

```

Debido a que Typescript es un lenguaje estricto fuertemente tipado, es decir, que no permite violaciones de tipos de datos; me fue necesario integrar en mi proyecto un conjunto de interfaces.

Estos elementos no son nada más que pequeñas guías que indican Typescript las propiedades que pueden tener los objetos con los que se trabaja. Por ejemplo, la interfaz de usuario tendrá la información mostrada a la izquierda.

La existencia de index.interface.ts se debe a reducir en los componentes la cantidad de importaciones a utilizar. Este archivo importa y exporta todas las interfaces de manera que un componente solo tenga que hacer referencia a este archivo para utilizar todas las interfaces que deseé.

3.2.1.7 Páginas de juegos:

Todas las páginas de juego tienen el mismo formato HTML: un formulario dinámico con un mat-card dentro que incluirá el cabecera y el pie mencionados en la parte de comunes ([3.2.1.4 Comunes y variables de entorno](#)) y entre ellos una mat-selection-list con un bucle de mat-list-options en el que se incluirá la información a mostrar. Ejemplo del HTML de la página de sinónimos:

```
<form [formGroup]="myForm" (ngSubmit)="onSubmit()">
  <mat-card class="card">
    <card-header [information]="cardHeaderInformation" />
    <mat-card-content>
      <mat-selection-list formControlName="selected" [multiple]="false" (selectionChange)="select()">
        <div *ngFor="let option of synonymOptions; let i = Index" [ngClass]="'correct' : (!todayAnswer && option.word==todayWord)">
          <mat-list-option checkboxPosition="before" [selected]="answeredOption(i)" [value]="i+1"
            [disabled]!="!todayAnswer">
            <div class="row centered">
              <div>
                {{option.synonym}}
              </div>
              <span class="spacer"></span>
            </div>
          </mat-list-option>
          <mat-divider></mat-divider>
        </div>
      </mat-selection-list>
    </mat-card-content>
    <card-footer [gameAnswer]="todayAnswer" [disabled]="disableCheck()" />
  </mat-card>
</form>
```

El archivo typescript del componente también será muy parecido. Contará con:

- El servicio del juego en concreto (phonetic.service.ts, definition.service.ts o synonym.service.ts) y que utilizará para descargarse las opciones disponibles.
- day.service.ts al que llamará al iniciarse para asegurarse de que cuenta con la última respuesta realizada por el usuario, si es que existe, y cuando el usuario envíe su respuesta.

```
ngOnInit() {
  this._synonymService.getTodaySynonyms().subscribe((data) => {
    this.synonymOptions = shuffle(data);
  });
  this._dayService.reloadAnswer();
}
```

```
onSubmit() {
  this._dayService.answer('SYNONYM', this.synonymOptions[this.selected - 1].word);
}
```

- La información a enviar al componente card-header: un ícono, un título y una descripción.

```
public cardHeaderInformation: CardHeaderInformation = {
  icon: "invert_colors",
  title: "Related words",
  description: "Pick the word's synonym"
}
public myForm: FormGroup = this._fb.group({
  selected: [0, [Validators.required]]
});
```

- El formulario en el que se registrará la opción decidida por el usuario (manejado a través de índices para que el usuario no tenga la respuesta abriendo el explorador de HTML de su navegador).
- Varios métodos utilizados para validar y elegir qué información mostrar en cada caso según las acciones que ha llevado a cabo el usuario.

```

answeredOption(i: number): boolean {
    return !(this.synonymOptions[i].word == this.todayAnswer?.word);
}

disableCheck() {
    return !this.selected || !this.todayAnswer;
}

select(){
    this.selectedIndex = this.myForm.controls["selected"].value-1;
}

```

La única página que se diferencia del resto es la de la fonética y se debe a que en vez de contener toda su lógica e información en un solo componente lo hace en tres.

El principal motivo de esta división es que al contrario que en el juego de la definición o del sinónimo no contiene simplemente texto, sino audio por lo que hay que introducir el botón para reproducirlo, elemento que transformé en otro componente ya que no encontré forma de dar estilo a la etiqueta existente en HTML5.

El primer componente, entonces sería phonetic-option, un sencillo componente que alberga la información a mostrar en cada línea de opción y el segundo componente phonetic-audio.component, que sería un poco más complejo y cuyos archivos HTML y typescript contendrán lo siguiente:

```

<button mat-icon-button matSuffix (click)="play($event)" (keydown.enter)="play($event)">
    <mat-icon>volume_up</mat-icon>
</button>

```

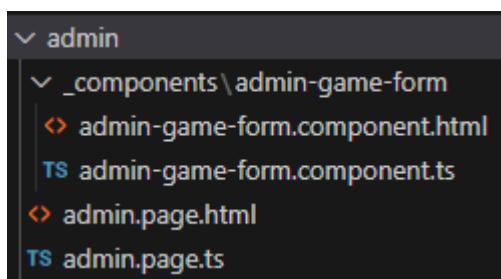
```

ngOnInit() {
    this._audioElement = new Audio(this.audio);
}

play(event: Event){
    event.preventDefault();
    event.stopPropagation();
    this._audioElement.play();
}

```

3.2.1.8 Página de creación de juegos:



Está página, solo visible para los usuarios con el rol administrador gracias a la lógica implementada en el layout ([3.2.1.3 Layout](#)) y la seguridad de la API ([3.3 Arquitectura de seguridad](#)), permite visualizar toda la información referente a los juegos diarios y, en caso de no existir uno, crearlo (Para más información de sobre su uso echar un vistazo al manual de uso adjuntado junto a la memoria).

Se divide en dos componentes, uno dentro del otro, cada uno con un formulario.

- admin.page será el componente padre y manejará la descarga de los juegos y la creación. Contiene un formulario con dos campos, uno de texto y otro de fecha, y varios botones. A través de los que se llevará a cabo la creación de juegos.
- admin-game-form: cuyo formulario contendrá un campo por cada propiedad de un objeto juego a excepción de la palabra y la fecha. Es decir: la fonética de la palabra del día y las tres similares, la definición y sus similares, y el sinónimo y sus similares. Permitirá revisar y modificar la información de todos estos datos en caso de encontrarse errores.

Cuenta con tres atributos, una fecha, una palabra y un juego. Buscará estos dos últimos a través del servicio de juegos (GameService), pasando como parámetro la fecha, que será la actual al iniciar la página o la indicada en el campo de fecha al tocarse este input.

Y un método por cada uno de sus botones:

- *build()*: será el del botón crear y se limitará a hacer la petición de crear un juego aleatorio o en caso de que el usuario haya introducido una palabra un juego a partir de la palabra indicada.

```
build() {
  this._loadingService.loadingShow();
  let request = this._gameService.getNewGame();
  if (this.wordPicker.value !== null && this.wordPicker.value.trim() !== "") {
    request = this._gameService.getNewGameFromWord(this.wordPicker.value);
  }
  this.game = null;
  this.wordPicker.setValue("");
  this.wordPicker.enable();
  request.subscribe(data => {
    this.game = data
    this.wordPicker.setValue(this.game.word)
    this._loadingService.loadingHide();
  }, error => {
    this._loadingService.loadingHide();
    this._responseService.showResponse(error);
  });
}
```

```
other() {
  this.wordPicker.setValue("");
  this.game = null;
}
```

- *other()*: el más sencillo de todos se limita a resetear el campo palabra y borrar el juego que se haya encontrado.

- *save()*: juntará ambos formularios en un objeto (“game”), que será lo que envíe a la API para realizar el guardado y la posterior publicación. Esto se realiza gracias a la propiedad `@ViewChild` de Angular, que permite a un objeto acceder a la información de un componente en su interior (siempre que sea pública).

```

export class AdminPage implements OnInit {

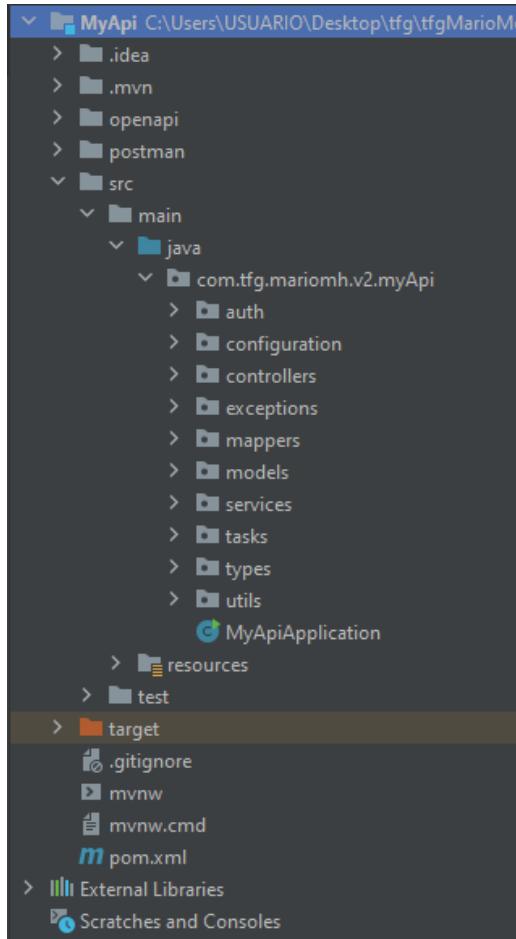
    @ViewChild(AdminGameFormComponent) adminGameFormComponent!: AdminGameFormComponent;

    save() {
        this._loadingService.loadingShow();
        let game: any = {
            word: this.wordPicker.value,
            date: this.datePicker.value.format("YYYYMMDD"),
        }
        Object.keys(this.adminGameFormComponent.myForm.controls)
            .forEach(k => game[k] = this.adminGameFormComponent.myForm.controls[k].value);
        this.game = null;
        this.wordPicker.setValue("");
        this.wordPicker.enable();
        this._gameService.save(game).subscribe((data) => {
            this.game = data;
            this.wordPicker.setValue(this.game.word)
            this.wordPicker.disable();
            this._loadingService.loadingHide();
            this._responseService.showMessage("Game created", "", "success");
        }, error => {
            this._loadingService.loadingHide();
            this._responseService.showResponse(error);
        })
    }
}

```

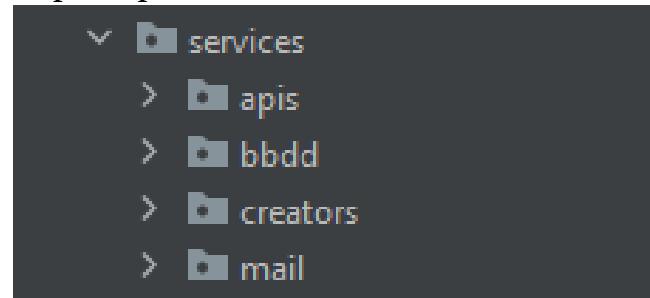
3.2.2 Arquitectura aplicación back-end Spring Boot:

3.2.2.1 Estructura del proyecto:



Los archivos de la API Spring Boot se categorizan en una división por tipo muy parecida a la de la aplicación cliente angular con algunas pequeñas excepciones donde pensé que era más útil e intuitivo en todos los ámbitos optar por un orden más orientado a la funcionalidad.

De manera que, por ejemplo, aunque la carpeta “auth” contenga servicios y una clase de configuración, al formar parte todo de algo tan concreto como es la seguridad y el manejo de credenciales preferí mantenerlo en una carpeta aparte.



Algo similar sucede dentro de la carpeta “services”, a lo que se suma el gran número de clases y su desconexión. Por ello creé cuatro subcategorías a mayores:

- apis: donde se almacenan los servicios encargados de realizar las peticiones a APIs externas.
- bbdd: que gestiona todo el guardado, modificado y trato en general de elementos de la base de datos.
- creators: conjunto de servicios utilizados para generar los juegos.
- mail: incluye el servicio de creación y el de envío de emails.

Por otro lado, a la hora de desarrollar los componentes establecí un sistema de interfaces e implementaciones que me permite definir lo que quiero que haga un clase antes de crearla y en casos como los controllers separar parte de la información de modo que puedo indicar en la interfaz del usuario controller, por ejemplo, todas las rutas (/users de base path, /used para la llamada encargada de responder si un email está usado, etc), el método (get, post, put o delete) y la procedencia de los atributos que recibirán (path, query o body) dejando a la implementación de la interfaz simplemente centrarse en el contenido de los métodos. Interfaz e implementación del servicio de respuestas:

```

@RequestMapping("/answers")
public interface IAnswerController {

    @PostMapping("/answer")
    ResponseEntity<AnswerDTO> answerGame(@RequestBody GameAnswerDTO gameAnswerDTO);

    @GetMapping("/answer/{userId}/{gameId}")
    ResponseEntity<AnswerDTO> getAnswerByUserGame(@PathVariable String userId, @PathVariable String gameId);
}

@RestController
@AllArgsConstructor
public class AnswerControllerImpl implements IAnswerController { Complexity is 3 Everything is cool!

    private final IAnswerService answerService;

    @Override
    public ResponseEntity<AnswerDTO> answerGame(GameAnswerDTO gameAnswerDTO) {
        return new ResponseEntity<>(answerService.answerGame(gameAnswerDTO), HttpStatus.CREATED);
    }

    @Override
    public ResponseEntity<AnswerDTO> getAnswerByUserGame(String userId, String gameId) {
        return new ResponseEntity<>(answerService.getAnswerByUserGame(userId, gameId), HttpStatus.OK);
    }
}

```

Además de todo esto el proyecto Spring Boot cuenta con dos carpetas añadidas a la raíz del proyecto con el fin de proveer información que puede ser útil para el usuario:

- openapi: conjunto de archivos YAML que proveen documentación sobre las peticiones existentes, su utilidad y sus respuestas.
- postman: contiene un archivo a importar en la plataforma con el mismo nombre con plantillas y ejemplos de todas las peticiones a realizar para que el desarrollador pueda probarlas.

3.2.2.2 Comunicación entre elementos:

Al comenzar la API decidí diferenciar la lógica de los distintos elementos involucrados en una petición muy estrictamente para, en caso de error, poder localizar el archivo donde está sucediendo el problema rápidamente.

Estos elementos son tres, aunque cuentan en algunos casos con subdivisiones:

- Controlador: encargado de recibir y responder las peticiones. En este caso, como hemos podido observar en la captura del controlador de respuestas ([3.2.2.1 Estructura del proyecto](#)) se encuentra dividido en dos:
 - Interfaz: donde se definen las rutas a las peticiones y los distintos parámetros que deberán o podrán incluir.
 - Implementación: donde se crearán las respuestas con su código de respuesta y se llamará al servicio encargado de proveer el contenido de esa respuesta.
- Servicios: divididos también en interfaz e implementación pero donde este primer elemento tiene poco más que carácter informativo. Aquí se lleva a cabo la lógica de las peticiones. Estos elementos serán los encargados de realizar las validaciones, comunicaciones y operaciones necesarias para crear, encontrar o borrar lo requerido.
- DAOs: Conjunto de interfaces que heredan de MongoRepository, el equivalente a JpaRepository para MongoDB. Une un objeto Java con la tabla en la que se almacena de la base de datos y contiene un conjunto de métodos para realizar operaciones como el borrado o la búsqueda además de permitir fácilmente la creación de consultas. Es, resumiendo, el encargado de realizar las operaciones directas sobre la base de datos.

```
public interface UserDao extends MongoRepository<User, String> {

    1 usage
    final static String FIND_ALL = "{deleted: false}";
    1 usage
    final static String FIND_BY_ID = "{id: '?0', deleted: false}";
    1 usage
    final static String FIND_BY_MAIL = "{mail: '?0', deleted: false}";
    1 usage
    final static String FIND_WANT_NOTIFICATION = "{notifications: true, deleted: false}";
    @Query(FIND_ALL)
    List<User> findAll();
    12 usages
    @Query(FIND_BY_ID)
    Optional<User> findById(String id);
    6 usages
    @Query(FIND_BY_MAIL)
    Optional<User> findByMail(String mail);
    1 usage
    @Query(FIND_WANT_NOTIFICATION)
    List<User> findAllWantNotifications();
```

3.2.2.3 Creación de palabras:

3.2.2.3.1 APIs externas:

Para entender el proceso de creación de juegos es necesario previamente comprender varios elementos, entre ellos, las APIs de terceros de las que me sirvo para obtener la información usada. Son tres:

- Random Word API (<https://random-word-api.herokuapp.com/home>): Cuya función es generar palabras aleatorias.
Parámetros: number, para indicar el número de palabras que se requieren (en caso de no introducirlo devuelve una); language, combinación de dos letras que señalan el lenguaje de la palabra/s pedidas (por defecto inglés); y length, o número de letras (si no se indica no se realizará ningún tipo de filtro).
Respuesta: array de Strings.
- Datamuse API (<https://www.datamuse.com/api/>): Proveedora de palabras parecidas, relacionadas, que suenan igual, etc.
Parámetros:

Key	Value	Description	...	Bulk Edit
ml	love	Significado parecido a...		
sl	hurted	Suena parecido a...		
sp	love	Spelled similarly to...		
sp	c*	Empieza por... (a*) Acaba por... (*a) Empie...		
lc	word	Suelen seguir en una frase a...		
rel_rhy	closer	Rima con...		
rel_jjb	word	Adjetivos usados para describir...		
rel_trg	love	Asociadas con...		
sug	wo	Ayudas...		
max	100	Numero de registros		

Respuesta: array de objetos que contienen la palabra (word), un número que indica, por ejemplo, en palabras que suenan parecido, el grado de similaridad (score); y el número de sílabas de la palabra (numSyllables).

- Free Dictionary API (<https://dictionaryapi.dev/>): Encargada de enviar las definiciones, sinónimos y audios que utilizo para los juegos.

Parámetros: uno, la palabra sobre la que quiere buscar información.

Respuesta: array de objetos con multitud de propiedades pero de las que yo solamente uso la palabra (word), fonética (phonetic), otras fonéticas, con acento y similares (phonetics) y los significados (meanings), que son otro array de objetos con sinónimos, antónimos, el tipo de la palabra, por ejemplo, nombre o adjetivo, y las definiciones en sí.

Para obtener la información de ellas elegí utilizar, a pesar de que sé que existen librerías que facilitan mucho el trabajo, como WebFlux; el proceso paso a paso, manualmente, porque quería entender qué realizaban todas estas librerías por detrás del mismo modo que aprendimos a hacer consultas SQL en programación de primero del grado superior a pesar de que existieran JPA y otros métodos mucho más simples y sencillos que automatizan todo ese proceso y lo reducen al mínimo.

Existe un servicio por cada API que contiene las peticiones posibles y que hereda de una clase abstracta llamada: External ApiService.

Esta es la encargada de realizar todo ese proceso manual del que he hablado de modo que realizar una petición se reduciría a llamar a este método incluyendo en los parámetros el método (GET, POST, PUT, etc) y la url.

```

private String request(String method, URL url) throws RequestApiError { Complexity is 6
    String response = "";
    Integer status = 500;
    try {
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod(method);
        status = connection.getResponseCode();
        response = readResponse(connection);
    } catch (IOException e) {
        if(status.equals(TOO_MANY_REQUESTS_STATUS)) {
            throw new TooManyRequestsResponseException(url.toString());
        }
        throw new RequestApiError(status, External ApiService.class, e.getMessage());
    }
    return response;
}

```

Otros métodos importantes de esta clase serían:

- CreateURL: que crea un objeto URL a partir de un String.
- ReplaceURL: cuyo uso se entiende sabiendo que todos los servicios de las APIs que requieren de alguna manera en sus URLs la palabra a buscar usan la constante URL_PLACEHOLDER, sustituida mediante este método por la palabra en cuestión.
- ReadResponse: que interpreta el texto recibido de una petición, transformándolo en un String que luego los objetos mapean a sus propiedades mediante la librería GSON gracias al método del objeto padre de todos los objetos respuesta (Response) al que se le pasa la respuesta en formato String y se le indica la clase de los objetos de dentro, en el caso de la Random Word API String.

```

protected static <T> List<T> stringToArray(String s, Class<T[]> clazz) {
    return new ArrayList<>(Arrays.asList(new Gson().fromJson(s, clazz)));
}

```

Otra pareja de objetos importantes y creados para simplificar el trabajo con URLs son: QueryParams yQueryParam.

```

@Data
public class QueryParams { Complexity is 6 It's time to do something...
    private List<QueryParam> parameters;
    2 usages
    public QueryParams(QueryParam ...parameters) {
        this.parameters = new ArrayList<>();
        for (QueryParam param: parameters) {
            add(param);
        }
    }
    /**
     * Crea los query params de una petición
     * @return String cadena de texto a añadir a la url
     */
    1 usage
    public String getQueryParams(){ Complexity is 5 Everything is cool!
        StringBuilder queryParams = new StringBuilder("?");
        parameters.forEach(param->{
            queryParams.append(param.getQueryParam()+"&");
        });
        queryParams.delete(queryParams.length()-1, queryParams.length());
        return queryParams.toString();
    }
    public void add(QueryParam param) { this.parameters.add(param); }
}

```

```

@Data
@AllArgsConstructor
public classQueryParam { Complexity is 3 Everything is cool!
    private String key;
    private String value;
    1 usage
    public String getQueryParam() { return key+"="+value; }
}

```

Se utilizan para trabajar con parámetros y contienen toda la lógica de creación de los strings con clave valor que luego se añade al final de las URLs.

Mediante estos objetos y estos métodos es posible que en los servicios API la complejidad se reduzca mucho ya que solo se incluyen las URLs, los

parámetros y su manejo para realizar las distintas peticiones de manera que por ejemplo en el servicio de peticiones a la API diccionario y de palabras aleatorias solo necesite introducir este código:

```
@Service
public class DictionaryApiServiceImpl extends External ApiService { Complexity is 3 Everything is cool!
    1 usage
    private final static String URL = "https://api.dictionaryapi.dev/api/v2/entries/en/" + URL_PLACEHOLDER;
    public DictionaryApiResponseDTO get(String word) throws RequestApiError {
        return new DictionaryApiResponseDTO(requestGet(replacePlaceholder(URL, word)));
    }
}
```

```
@Service
public class RandomWordApiServiceImpl extends External ApiService { Complexity is 3 Everything is cool!
    1 usage
    private final static String URL = "https://random-word-api.herokuapp.com/word";
    1 usage
    private final static String QUERY_NUMBER = "number";
    1 usage
    private final static String LANGUAGE = "lang";
    1 usage
    private final static String ENGLISH = "en";
    1 usage
    public RandomWordApiResponseDTO getSome(Short number) throws RequestApiError {
        return new RandomWordApiResponseDTO(requestGetWithParams(URL, getSimpleParams(number)));
    }
    1 usage
    private QueryParams getSimpleParams(Short number){
        return new QueryParams(
            newQueryParam(QUERY_NUMBER, String.format("%s", number)),
            newQueryParam(LANGUAGE, ENGLISH)
        );
    }
}
```

3.2.2.3.2 Requesters:

En la mayoría de los casos los servicios de creación de juegos no llamarán directamente a los servicios de las APIs externas. Existe antes otra capa, añadida para simplificar aún más el trabajo. La llamada capa de Requesters.

Esta capa cuenta con dos servicios, RandomRequesterImpl y RandomRequesterImpl . Como se puede ver no existe uno de diccionario y eso es por un motivo muy sencillo: ambos llaman a esa API. No sirve de nada una palabra que no tiene registro en el diccionario porque no se puede extraer información de ella y desafortunadamente tanto en la API de palabras relacionadas como en la de aleatorias muchas veces se devuelven palabras así.

Estas capas se encargan de hacer ese filtrado, entre otros, y devolver solo palabras válidas y existentes en el diccionario.

RandomRequesterImpl realiza una petición de mil palabras con la que llena una lista y devuelve la primera que encuentra en esa lista que existe en el diccionario, volviendo a hacer la petición para llenar la lista siempre que se queda sin palabras.

RelatedRequesterImpl es más complejo, además de con la lista cuenta con dos atributos un juego y una instancia del enum RelatedType. La lista se llenará una vez se le pasen estos atributos porque para hacer las peticiones a DatamuseApi hace falta

una palabra (sacada del juego) y un tipo (sacado del enum) y una vez se terminen las palabras que proporciona en vez de volver a hacer una petición lanzará una excepción indicando que se ha quedado sin información que proveer.

```
public enum RelatedType {
    3 usages
    ASSOCIATED_WITH, MEANING_LIKE, RIMES_WITH, SOUND_LIKE, SYLLABLES
}

private void search() throws RequestApiError {
    switch(relatedType){
        case ASSOCIATED_WITH -> list = datamuse ApiService.getAssociatedWith(game).getEntries();
        case MEANING_LIKE -> list = datamuse ApiService.getMeaningLike(game).getEntries();
        case RIMES_WITH -> list = datamuse ApiService.getRimesWith(game).getEntries();
        case SOUND_LIKE -> list = datamuse ApiService.getSoundLike(game).getEntries();
    }
}
```

3.2.2.3.3 Validadores:

Todo mi sistema de creación de juegos depende de una cosa muy sencilla: una palabra padre a partir de la que se obtiene información y palabras relacionadas que también tendrás datos para confundir y dificultar el proceso de reconocimiento de, por ejemplo, la definición acertada.

Esas palabras relacionadas siempre deben ser diferentes a la palabra padre y entre ellas, pero no solo eso, no deben compartir raíz.

Para este tipo de validaciones y muchas más, como la validación de que una definición no incluye ni a la palabra que define ni a sus derivados y lo mismo con la palabra padre del juego; creé las clases: Validator y WordValidator.

Destaca el método `getRoots(String word)` utilizado para extraer de una palabra un array de posibles raíces eliminando un determinado número de letras según el tamaño de la palabra.

3.2.2.3.4 Servicios de creación:

Explicadas todas las partes involucradas en el proceso de creación de juegos podemos pasar al proceso en sí. Todo se realiza a través del servicio GameCreator que contaría con los siguientes atributos:

```
private final PhoneticSimpleCreator phoneticCreator;
private final DefinitionSimpleCreator definitionCreator;
private final PhoneticSingleCreator phoneticSingleCreator;
private final DefinitionSingleCreator definitionSingleCreator;
private final SynonymCreator synonymCreator;
private final RandomRequesterImpl randomRequester;
private final DictionaryServiceImpl dictionary ApiService;
```

Y dos métodos públicos, los dos tipos de creación de juego que existen, a partir de una palabra aleatoria y de una palabra indicada.

```

public Game create(){ Complexity is 5 Everything is cool!
    Game createdWord = null;
    while(createdWord == null){
        try {
            createdWord = createFromDictionary(randomRequester.getDictionaryRegister());
        }catch(ObjectCreationError error){
            log.info(error.getMessage());
        }
    }
    return createdWord;
}

public Game createFromWord(String word) throws GameCreationError {
    try{
        return createFromDictionary(dictionary ApiService.get(word));
    }catch(ObjectCreationError | RequestApiError error){
        log.info(error.getMessage());
        throw new GameCreationError(word, error.getMessage());
    }
}

```

Como se puede observar ambos llaman al método *createFromDictionary*, que simplemente se encarga de llamar a los métodos de los distintos servicios de creación y llenar las propiedades del juego (objeto Game).

```

private Game createFromDictionary(DictionaryApiResponseDTO response) throws ObjectCreationError {
    Game game = new Game();
    game.setWord(response.getEntries().get(0).getWord());
    game.setSynonym(synonymCreator.findSynonym(response));
    game.setPhonetic(phonicCreator.find(response));
    game.setDefinition(definitionCreator.find(response));
    game.setSimilarSynonyms(synonymCreator.findSimilars(game, response));
    game.setSimilarPhonetics(phonicSingleCreator.findSimilars(response, game));
    game.setSimilarDefinitions(definitionSingleCreator.findSimilars(response, game));
    return game;
}

```

Cada una de estas peticiones lanzará una excepción que hereda de ObjectCreationError en caso de no encontrar información necesaria en algún punto o RequestApiError si sucede algún problema con las APIs, como que se han realizado demasiadas peticiones.

La creación aleatoria simplemente buscará otro registro aleatorio en el diccionario en caso de que suceda una excepción de información y volverá a intentarlo. Solo parará si el error es de petición. La creación a partir de la palabra devolverá una nueva excepción con la información del error por el que no se ha podido crear el juego.

3.2.2.3.4.1 Creador de fonética:

Este creador implementa la interfaz ISimpleCreator, que contiene los métodos definidos *find (DictionaryApiResponse)* y *findAccordingTo (DictionaryApiResponse)*. De momento nos centraremos en el primero.

```

default T find(DictionaryApiResponseDTO response) throws ObjectCreationError {
    ObjectCreationError error = null;
    for (DictionaryApiEntryDTO entry : response.getEntries()) {
        try {
            return find(entry);
        } catch (ObjectCreationError e) {
            error = e;
        }
    }
    throw error;
}

```

Y los métodos por definir *find (DictionaryApiEntry)* y *findAccordingTo (DictionaryApiEntry)* que, de nuevo, centrándonos en el *find*, el creador implementa de la siguiente manera:

```

@Override
public Phonetic find(DictionaryApiEntryDTO entry) throws PhoneticCreationError {
    return new Phonetic(findAudio(entry), findSyllables(entry));
}

```

La lógica es sencilla. Por cada respuesta de la API diccionario se recorren las entradas de las que dispone y por cada una de ellas se busca el audio y el número de sílabas de la palabra.

```

private String findAudio(DictionaryApiEntryDTO entry) throws PhoneticAudioCreationError {
    return entry.getPhonetics().stream() Stream<Phonetic>
        .map(Phonetic::getAudio) Stream<String>
        .filter(Validator::objectValid)
        .findFirst() Optional<String>
        .orElseThrow(()-> new PhoneticAudioCreationError(entry.getWord()));
}

```

```

private Short findSyllables(Word wordObject) {
    String word = wordObject.getWord();
    char[] vowels = { 'a', 'e', 'i', 'o', 'u', 'y' };
    char[] currentWord = word.toCharArray();
    Short numVowels = 0;
    boolean lastWasVowel = false;
    for (char wc : currentWord) {
        boolean foundVowel = false;
        for (char v : vowels) {
            //don't count diphthongs
            if ((v == wc) && lastWasVowel)
            {
                foundVowel = true;
                lastWasVowel = true;
                break;
            }
            else if (v == wc && !lastWasVowel)
            {
                numVowels++;
                foundVowel = true;
                lastWasVowel = true;
                break;
            }
        }
        // If full cycle and no vowel found, set lastWasVowel to false;
        if (!foundVowel)
            lastWasVowel = false;
    }
    // Remove es, it's _usually_ silent
    if (word.length() > 2 &&
        word.substring(word.length() - 2) == "es")
        numVowels--;
    // remove silent e
    else if (word.length() > 1 &&
        word.substring(word.length() - 1) == "e")
        numVowels--;
    return numVowels;
}

```

findAudio recoge todas las fonéticas de la palabra, saca de ellas el audio y filtra los que no sean strings vacíos (*Validator::objectValid*) y devuelve el primero encontrado o, en caso de que no exista, lanza una excepción.

findSyllables es más complejo, sigue un sistema de detección de sílabas diseñado por un usuario de la famosa página stack overflow con un margen de acierto del 99% inspirado en el Algoritmo de Porter.

3.2.2.3.4.2 Creador de definición:

Este servicio también hereda de la interfaz ISimpleCreator y su funcionamiento en esencia es el mismo que el del creador de fonéticas. La diferencia radica en el modo en que se encuentran las definiciones en una entrada de diccionario y por tanto en el que se procesa la búsqueda.

```
public Definition find(DictionaryApiEntryDTO entry) throws DefinitionCreationError {
    for(DictionaryApiMeaningDTO meaning: entry.getMeanings()){
        Definition definition = findDefinitionOnMeaning(entry.getWord(), meaning);
        if(definition!=null){
            return definition;
        }
    }
    throw new DefinitionCreationError(entry.getWord());
}
```

Al existir varios significados aunque en uno no se encuentre una definición válida otra puede serlo de manera que no es hasta haber recorrido todas hasta que se lanza la excepción de fallo.

```
private Definition findDefinitionOnMeaning(String word, DictionaryApiMeaningDTO meaning) { Complexity is 5 Everything is cool
    return meaning.getDefinitions().stream() Stream<DictionaryApiDefinitionDTO>
        .map(dictionaryApiDefinition -> new Definition(meaning.getPartOfSpeech(), dictionaryApiDefinition.getDefinition()))
        .filter(definition-> Validator.objectsValid(definition.getType(), definition.getText()))
        .filter(definition -> !WordValidator.wordInPhrase(word, definition.getText()))
        .findFirst() Optional<Definition>
        .orElse( other: null);
}
```

El proceso de búsqueda de definición es muy similar al de la fonética. Coge todas las definiciones con su partOfSpeech (tipo) y luego va filtrando, borra las que estén vacías (Validator.objectsValid), las que tengan su propia palabra en la definición (!WordValidator.wordInPhrase) y coge la primera que encuentre.

3.2.2.3.4.3 Creador de similares a fonética:

Aquí entra en juego otra interfaz: ISingleCreator. Dedicada a crear los registros individuales de similares y que todos los servicios que la implementen deben definir los siguientes métodos:

```
public interface ISingleCreator<T>{
    2 usages 2 implementations
    List<T> findSimilar(DictionaryApiResponseDTO dictionaryApiResponseDTO, Game game);
    2 usages 2 implementations
    void fillListWithRelateds(DictionaryApiResponseDTO dictionaryApiResponseDTO, Game game, List<T> similars, RelatedType[] relatedTypes);
    2 usages 2 implementations
    void fillListWithRelated(DictionaryApiResponseDTO dictionaryApiResponseDTO, Game game, List<T> similarDefinitions);
    2 usages 2 implementations
    void fillListWithRandom(DictionaryApiResponseDTO dictionaryApiResponseDTO, Game game, List<T> similarDefinitions);
    4 usages 2 implementations
    T findRelated(Game game) throws NoAvailableElementsError;
    4 usages 2 implementations
    T findRandom(Game game);
    4 usages 2 implementations
    * T getSingle(Game game, DictionaryApiResponseDTO dictionaryApiResponseDTO) throws ObjectCreationError;
}
```

findSimilar es el método externo al que llama el creador de juegos, es decir, el que devuelve la lista de elementos individuales similares y siempre hará lo mismo: crear una lista, llenarla llamando a *fillListWithRelateds* y *fillListWithRandom* y devolverla.

```
public List<PhoneticSingle> findSimitars(DictionaryApiResponseDTO dictionaryApiResponseDTO, Game game){
    List<PhoneticSingle> similarPhonetics = new ArrayList<>();
    fillListWithRelateds(dictionaryApiResponseDTO, game, similarPhonetics,
        new RelatedType[]{RelatedType.SOUND_LIKE, RelatedType.RIMES_WITH});
    fillListWithRandom(dictionaryApiResponseDTO, game, similarPhonetics);
    return similarPhonetics;
}
```

fillListWithRelated utiliza instancias del enum *RelatedType*, que son simples opciones para indicar al requester de palabras relacionadas el tipo de búsqueda a realizar. En este caso, primero de palabras que suenen igual y luego de palabras que riman.

Su trabajo será recorrer estos tipos de relacionado, indicarle la palabra y el tipo de búsqueda a realizar al requester de relacionadas para que realice la búsqueda y llamar a *fillListWithRelated* que recorrerá los registros de obtenidos de cada búsqueda hasta que o bien haya encontrado tres opciones (la lista de similares se ha llenado) o ya no queden palabras relacionadas.

```
public void fillListWithRelateds(DictionaryApiResponseDTO dictionaryApiResponseDTO,
    for (RelatedType relatedType: relatedTypes) {
        relatedRequester.setGameAndRelatedType(game, relatedType);
        fillListWithRelated(dictionaryApiResponseDTO, game, similars);
    }
}

public void fillListWithRelated(DictionaryApiResponseDTO dictionaryApiResponseDTO, Game game,
    Boolean emptySearch = false;
    while(similars.size()<3 && emptySearch==false) {
        try {
            PhoneticSingle phoneticSingle = findRelated(game);
            if(!WordValidator.present(phoneticSingle, similars)){
                similars.add(phoneticSingle);
            }
        } catch (NoAvailableElementsError e) {
            emptySearch = true;
        }
    }
}
```

Antes de indicar lo que hace el *findRelated* pasemos al *fillListWithRandom*. Que será muy sencillo. Buscará una palabra aleatoria con datos parecidos hasta que la lista esté llena.

```
public void fillListWithRandom(DictionaryApiResponseDTO dictionaryApiResponseDTO, Game game,
    while(similars.size()< Validator.SIMILAR_LIST_SIZE) {
        PhoneticSingle phoneticSingle = findRandom(game);
        if(!WordValidator.present(phoneticSingle, similars)){
            similars.add(phoneticSingle);
        }
    }
}
```

findRandom y *findRelated* hacen básicamente lo mismo. Buscan registros de los requesters e intentan crear similares, en este caso, fonéticas, mediante el antes mencionado método del creador simple *findAccordingTo* para intentar encontrar una opción lo suficiente difícil para entorpecer al usuario pero no imposibilitar encontrar la opción adecuada del juego.

```

    default T findAccordingTo(Game game, DictionaryApiResponseDTO response) throws ObjectCreationError {
        ObjectCreationError error = null;
        for (DictionaryApiEntryDTO entry : response.getEntries()) {
            try {
                return findAccordingTo(game, entry);
            } catch (ObjectCreationError e) {
                error = e;
            }
        }
        throw error;
    }

    public Phonetic findAccordingTo(Game game, DictionaryApiEntryDTO entry) throws PhoneticCreationError {
        Short syllables = findSyllables(entry);
        if(similarSyllables(game.getPhonetic().getSyllables(), syllables)){
            return new Phonetic(findAudio(entry), syllables);
        }
        throw new PhoneticSyllablesCreationError(entry.getWord());
    }
}

```

Para este juego se valida que la palabra encontrada tiene un número de sílabas similar a la palabra padre (como mucho una más o una menos).

3.2.2.3.4.4 Creador de similares a definición:

El funcionamiento del creador de definiciones similares es idéntico al de las fonéticas, lo único que cambian son los tipo de relacionados pasados al método *fillListWithRelated*: con significado parecido y asociadas.

```

public List<DefinitionSingle> findSimilarDefinitions(DictionaryApiResponseDTO dictionaryApiResponseDTO, Game game){
    List<DefinitionSingle> similarDefinitions = new ArrayList<>();
    fillListWithRelateds(dictionaryApiResponseDTO, game, similarDefinitions,
        new RelatedType[]{RelatedType.MEANING_LIKE, RelatedType.ASSOCIATED_WITH});
    fillListWithRandom(dictionaryApiResponseDTO, game, similarDefinitions);
    return similarDefinitions;
}

```

Y las validaciones realizadas para determinar si una definición es válida en el *findDefinitionAccordingTo* del creador simple que además de validar que la palabra que se define no se haya en la definición y que la definición no esté vacía, se asegura de que la palabra padre o cualquier otra que comparte raíz con ella tampoco sea mencionada de ningún modo; y que el tipo de palabra de ambas sea el mismo (Si la padre es un verbo que la otra también lo sea).

```

private Definition findDefinitionOnMeaningAccordingTo(Game game, String word, DictionaryApiMeaningDTO meaning) { Complexity is 0 It's time to do something...
    return meaning.getDefinitions().stream() Stream<DictionaryApiDefinitionDTO>
        .map(dictionaryApiDefinition -> new Definition(meaning.getPartOfSpeech(), dictionaryApiDefinition.getDefinition()))
        .filter(definition -> Validator.objectsValid(definition.getType(), definition.getText()))
        .filter(definition -> game.getDefinition().getType() == definition.getType())
        .filter(definition -> !WordValidator.wordInPhrase(word, definition.getText()))
        .filter(definition -> !WordValidator.wordInPhrase(game, definition.getText()))
        .findFirst() Optional<Definition>
        .orElse( other: null);
}

```

Por último, es necesario validar que las palabras no son sinónimos de la del juego porque entonces el significado también sería válido. Para ello se cogen los sinónimos como veremos en el siguiente punto ([3.2.2.3.4.5 Creador de sinónimo y similares](#)) y se verifica que la palabra a añadir no se encuentre entre ellos antes de añadirla a la lista.

```

public void fillListWithRelated(DictionaryApiResponseDTO dictionaryApiResponseDTO, Game game, List<DefinitionSingle> similars){ Complex
    Boolean emptySearch = false;
    List<String> synonyms = getSynonyms(dictionaryApiResponseDTO);
    while(similars.size()<3 && emptySearch==false) {
        try {
            DefinitionSingle definitionSingle = findRelated(game);
            if(!WordValidator.present(definitionSingle, similars) && !WordValidator.listIncludes(synonyms, definitionSingle.getWord())){
                similars.add(definitionSingle);
            }
        } catch (NoAvailableElementsError e) {
            emptySearch = true;
        }
    }
}

public void fillListWithRandom(DictionaryApiResponseDTO dictionaryApiResponseDTO, Game game, List<DefinitionSingle> similarDefinitions){ Complex
    List<String> synonyms = getSynonyms(dictionaryApiResponseDTO);
    while(similarDefinitions.size()< Validator.SIMILAR_LIST_SIZE) {
        DefinitionSingle definitionSingle = findRandom(game);
        if(!WordValidator.present(definitionSingle, similarDefinitions) && !WordValidator.listIncludes(synonyms, definitionSingle.getWord())){
            similarDefinitions.add(definitionSingle);
        }
    }
}

```

3.2.2.3.4.5 Creador de sinónimo y similares:

Al contrario que los otros creadores el de sinónimos y similares no implementa ninguna de las interfaces anteriormente mencionadas ni tampoco divide su funcionamiento en dos clases (creador de palabra y de similares).

Esto sucede por un motivo: Mientras que con la fonética y la definición basta con encontrar un registro válido, no se tienen en cuenta los que no lo sean, a la hora de trabajar con sinónimos hay que estar constantemente validando no solo que las palabras similares no comparten raíz con la padre sino tampoco con ningún sinónimo. De modo que encontré más eficiente recorrer una vez todas las entradas del diccionario de la palabra y recoger todos los sinónimos y antónimos de un registro en listas que trabajar de la misma manera que con los otros creadores, lo que llevaba a realizar constantes iteraciones sobre los mismos objetos.

Esto se lleva a cabo con el método *getRelatedWords* que devuelve un hashmap que contiene tres listas: sinónimos, antónimos y otra lista con todos los registros de ambos.

```

public Map<String, List<String>> getRelatedWords(DictionaryApiResponseDTO response){ Complexity is 4 Everything is cool!
    Map<String, List<String>> relatedWords = new HashMap<>();
    relatedWords.put(SYNONYMS, new ArrayList<>());
    relatedWords.put(ANTONYMS, new ArrayList<>());
    relatedWords.put(ALL, new ArrayList<>());
    response.getEntries().stream().forEach(entry->{
        entry.getMeanings().stream().forEach(meanings->{
            relatedWords.get(SYNONYMS).addAll(meanings.getSynonyms());
            relatedWords.get(ANTONYMS).addAll(meanings.getAntonyms());
        });
    });
    relatedWords.put(SYNONYMS, filterList(response.getEntries().get(0).getWord(), relatedWords.get(SYNONYMS)));
    relatedWords.put(ANTONYMS, filterList(response.getEntries().get(0).getWord(), relatedWords.get(ANTONYMS)));
    relatedWords.get(ALL).addAll(relatedWords.get(SYNONYMS));
    relatedWords.get(ALL).addAll(relatedWords.get(ANTONYMS));
    return relatedWords;
}

```

Explicado esto pasemos al método de obtención del sinónimo, que llama a *findSynonym* pero pasando además el map antes mencionado para ir recorriendo uno a uno los sinónimos y en caso de que la palabra exista en el diccionario devolverla.

```

public String findSynonym(DictionaryApiResponseDTO response) throws ObjectCreationError {
    return findSynonym(response.getEntries().get(0).getWord(), getRelatedWords(response));
}

public String findSynonym(String word, Map<String, List<String>> relatedWords) throws SynonymNotFoundCreationError {
    if(relatedWords.get(SYNONYMS).size()>0){
        for(String synonym: relatedWords.get(SYNONYMS)){
            try {
                dictionary ApiService.get(synonym);
                return synonym;
            } catch (RequestApiError e) {
                // Doesn't exist or ApiError
            }
        }
    }
    throw new SynonymNotFoundCreationError(word);
}

```

El método de obtención de similares aunque con parámetros distintos se hace de un modo muy parecido al de cualquier otro creador.

```

public List<String> findSimilars(Game game, DictionaryApiResponseDTO response){
    return findSimilars(game, getRelatedWords(response));
}

public List<String> findSimilars(Game game, Map<String, List<String>> relatedWords){
    List<String> similars = new ArrayList<>();
    fillListWithAntonyms(similars, relatedWords.get(ANTONYMS));
    fillListWithRelateds(game, similars, relatedWords, new RelatedType[]{RelatedType.ASSOCIATED_WITH});
    fillListWithRandom(similars, relatedWords.get(ALL));
    return similars;
}

```

En este caso se ve que antes de buscar relacionados (de tipo asociados), eso sí, se llama al método *fillListWithAntonyms*, que añade uno o dos, un número aleatorio, de antónimos de la lista de antónimos obtenida anteriormente (en caso de existir en el diccionario y no encontrarse ya en la lista).

```

public void fillListWithAntonyms(List<String> similars, List<String> antonyms){ Complexity is 6 It's time to
    Integer maxAddition = new Random().nextInt( bound: 2);
    for(String antonym: antonyms){
        try {
            dictionary ApiService.get(antonym);
            if(similars.size() < maxAddition && !WordValidator.listIncludes(similars, antonym)) {
                similars.add(antonym);
            }
        } catch (RequestApiError e) {
            // Doesn't exist or ApiError
        }
    }
}

```

3.3 Arquitectura de seguridad:

Para mantener la seguridad en la aplicación decidí usar JWT (Json Web Token) ya que actualmente es uno de los métodos de seguridad más utilizados y útiles debido a su sencillez y que permite guardar información de los usuarios para tenerla a disposición del usuario.

El proceso de obtención de tokens es sencillo: el usuario inicia sesión con unas credenciales (en el caso de *Dayctionary* email y contraseña) cuya validez spring security en conjunto con el formulario Angular se encarga de validar de la siguiente manera. Cuando se introducen unos credenciales se llama al siguiente método:

```
no usages ▲ mmontalvillo98 *
@Override
public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) throws AuthenticationException {
    UserDTO userDTO = new UserDTO();
    try {
        userDTO = new ObjectMapper().readValue(request.getReader(), UserDTO.class);
    } catch (IOException e) {}
    UsernamePasswordAuthenticationToken usernamePAT = new UsernamePasswordAuthenticationToken(
        userDTO.getMail(),
        userDTO.getPassword(),
        Collections.emptyList()
    );
    return getAuthenticationManager().authenticate(usernamePAT);
}
```

attemptAuthentication de *JwtAuthenticationFilter* crea una instancia de *UsernamePasswordAuthenticationToken* a partir de la información facilitada por el usuario, lo que, entre otras cosas como encriptar la contraseña llama a *loadUserByUsername* de *UserDetailsServiceImpl*:

Otro método que utiliza *userService.getByMailCompleteUser* (llamado así porque devuelve la información de la contraseña, algo que no se realiza en el resto de método que obtienen información del usuario) para recibir los datos reales del usuario con ese mail (en caso de que exista) y guardarla en una instancia de *UserDetails* (un híbrido entre clase de Usuario y servicio que almacena la información del usuario y cuenta con un atributo llamado *username* y otro *password* donde se guardan el mail y la contraseña respectivamente además varios métodos para saber si la cuenta está bloqueada (*isAccountNonLocked*) o directamente disponible (*isEnabled*) entre otros).

En caso de que estos datos sean correctos la api genera un token con un tiempo de validez de 86400000 milisegundos, es decir, 1 día; que guarda el id y los roles del usuario en cuestión y se lo envía en la cabecera de la respuesta.

Esta información se haya en el archivo application-local, junto a la clave secreta que protegerá el token:

```
MONTALVILLO98, 24/07/2020
# SECURITY
jwt.secret=secret
jwt.expiration=86400000
```

Esta información es accesible colocando la anotación `@Value("${jwt.nombrevariable}")` encima del atributo de un componente.

El cliente toma esta respuesta y extrae el token para guardarla en localstorage del usuario, permitiéndole de esta forma extraer e incorporar ese token a todas las siguientes llamadas que realizará.

Este usuario puede ser de dos tipos: Usuario o administrador.

Ejemplos de tokens:

Usuario:

Encoded	Decoded
PASTE A TOKEN HERE	HEADER: ALGORITHM & TOKEN TYPE
<code>eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiI2NTEwMjkwODI3ODhhODMyNGM2MTA2NTAiLCJleHAiOjE20TU10TQwMDUsInJvbGVzIjo1WTVTRVJdIn0.HMNSjRyQvwGq4WoqJnA1FajGV-JF1ib8hn5wuZzCV5ChYSWFT1RhU4XUZoqP2nsIWgY4YNoXCC-cC5SDgnCtw</code>	<code>{ "alg": "HS512" }</code>
PAYLOAD: DATA	
	<code>{ "sub": "651829182788e8324c610650", "exp": 1695594085, "roles": "[USER]" }</code>
VERIFY SIGNATURE	
	<code>HMACSHA512(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret) □ secret base64 encoded</code>

Administrador:

Encoded	Decoded
PASTE A TOKEN HERE	HEADER: ALGORITHM & TOKEN TYPE
<code>eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiI2NTEwMjkljYzVlYzAyOTIwNDAzYjZTYiLCJleHAiOjE20TU10TQyNjIsInJvbGVzIjo1WTVTRViIsIEFETUL0XSJ9.sRlhBqjYA6gIBLiZ5CfDOKiMXiG85TWCC_1jqNb5JFfLnPPxLjQ2YW-2GfPmCb96_Zeklx1Lt4eo13UMvsuWBw </code>	<code>{ "alg": "HS512" }</code>
PAYLOAD: DATA	
	<code>{ "sub": "651829cc5ec02928483b6be6", "exp": 1695594262, "roles": "[USER, ADMIN]" }</code>
VERIFY SIGNATURE	
	<code>HMACSHA512(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret) □ secret base64 encoded</code>

Los usuarios cuentan con permisos de lectura y escritura sobre respuestas y usuarios, pero ni de lectura ni escritura sobre juegos, estos solo los tienen los administradores, cosa que se maneja desde spring con el siguiente bloque de código:

```
@Bean
SecurityFilterChain filterChain(HttpSecurity http, AuthenticationManager authManager) throws Exception{ Complexity is 4 Everything is cool!
    JwtAuthenticationFilter jwtAuthenticationFilter = new JwtAuthenticationFilter();
    jwtAuthenticationFilter.setAuthenticationManager(authManager);
    jwtAuthenticationFilter.setFilterProcessesUrl("/login");

    return http
        .cors().and()
        .csrf(csrf -> csrf.disable())
        .authorizeRequests()
            .requestMatchers( ...patterns: "/games/**").hasAuthority("ADMIN")
            .requestMatchers( ...patterns: "/answers/**", "/mini/**", "/users/token/**").hasAnyAuthority( ...authorities: "ADMIN", "USER")
            .requestMatchers( ...patterns: "/**").permitAll()
        .and()
        .httpBasic()
        .sessionManagement(sess -> sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .addFilter(jwtAuthenticationFilter)
        .addFilterBefore(jwtAuthorizationFilter, UsernamePasswordAuthenticationFilter.class)
        .build();
}
```

Como se puede ver en la primera imagen solo los usuarios con rol admin pueden hacer llamadas con patrón “/games/**” (juegos) mientras que a “/answers/**”

(respuestas), “/mini/**” (minijuegos), “/users/token/**” (datos privados de usuario).

El resto de llamadas “/**” estarían permitidas para usuarios sin autenticar, lo que permitirá hacer llamadas a “/users/**” (operaciones de registro de usuarios como validación de email y creación de cuenta) y “/mails/**” (Envío de emails como los enviados cuando el usuario ha olvidado su contraseña o se registra por primera vez).

Una vez el código pierda su validez, ya sea porque llega la fecha de expiración o porque el usuario decide hacer logout el usuario será devuelto al login inmediatamente. La api cliente está preparada para no realizar ninguna acción si no hay un token válido, esto se consigue mediante los llamados “guards” (guardias) de angular, un tipo de componentes que se pueden colocar en las rutas a los componentes de angular y añadir condiciones a su acceso.

```
const routes: Routes = [
  { path: 'auth', loadChildren: () => import("./modules/auth.module").then(m => m.AuthModule),
    canActivate: [PublicGuard], canMatch: [PublicGuard]},
  {
    path: 'word', loadChildren:()=>import('./modules/word.module').then(m=>m.WordModule),
    canActivate: [AuthGuard], canMatch: [AuthGuard]
  },
  {path:'**', redirectTo:"word"}
  // { path: '404', component: ErrorPageComponent },
  // { path: '**', redirectTo: "404" }
]
```

Como se puede observar el proyecto se sirve de dos guards. PublicGuard que se encarga de dar acceso al login y el registro a los usuarios que no tienen token válido y redirigir al resto y AuthGuard, que hace exactamente lo contrario.

PublicGuard:

```
constructor(private _authService: AuthService, private router: Router){ }

private checkAuthStatus():boolean|Observable<boolean>{
  return this._authService.checkAuthentication()
  .pipe(
    tap( isAuthenticated => {
      if(isAuthenticated){
        this.router.navigate(['./word']);
      }
    }),
    map(isAuthenticated=> !isAuthenticated) // para que canActivate deje pasar
  )
}

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | Observable<boolean> {
  return this.checkAuthStatus();
}

canMatch(route: Route, segments: UrlSegment[]): boolean | Observable<boolean> {
  return this.checkAuthStatus();
}
```

AuthGuard:

```

constructor(private _authService: AuthService, private router: Router){ }

private checkAuthStatus():boolean|Observable<boolean>{
  return this._authService.checkAuthentication()
    .pipe(
      tap(isAuthenticated => {    Parameter 'isAuthenticated' implicitly has an 'any' type.
        if(!isAuthenticated){
          this.router.navigate(['./auth/login']);
        }
      })
    )
}

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | Observable<boolean> {
  return this.checkAuthStatus();
}

canMatch(route: Route, segments: UrlSegment[]): boolean | Observable<boolean> {
  return this.checkAuthStatus();
}

```

Ambos componentes son prácticamente iguales, cuentan con tres métodos, *checkAuthStatus*, *canActivate* y *canMatch*. Estos dos últimos los deben tener todos los guards y sirven para permitir acceder a la url y activarla respectivamente. El primero es el método que alberga con toda la lógica de acceso realizado gracias al método de authService *checkAuthentication* que haría lo siguiente:

```

checkAuthentication(): Observable<boolean> {
  if (!this.token || this.isTokenExpired) return of(false); //of genera observable con boolean
  return this.http.get<User>(`${environment.API_PATH}/users/token/${this.tokenId}`)
    .pipe(
      tap(user => this.user = user),    Parameter 'user' implicitly has an 'any' type.
      map(user => !!user),    Parameter 'user' implicitly has an 'any' type.
      catchError(err => {    Parameter 'err' implicitly has an 'any' type.
        localStorage.removeItem("token");
        return of(false)
      })
    )
}

```

Devuelve True solo en caso de que exista un token no expirado y se pueda realizar con éxito una petición a la api que devuelva toda la información del usuario sin ningún error de por medio.

3.4 Arquitectura de emails:

El proceso de creación y envío de emails se lleva a cabo a través de varios elementos:

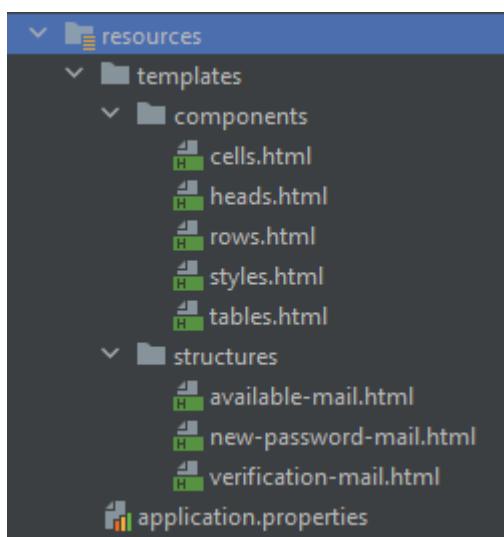
- Configuración: datos incluidos en el archivo application.properties para vincular el email a través del que se enviarán los correos de la aplicación.

Para poder realizar esta conexión es necesario activar la verificación en dos pasos de la cuenta de google. y una vez se puede acceder a ese menú ir a la última opción, “Contraseñas de aplicaciones”, donde se desplegará un menú que permitirá incluir una aplicación. Al hacerlo, esto generará una contraseña que es la que usaremos en el proyecto Spring Boot para acceder al correo y usarlo.

The screenshot shows the Google Account Security settings. On the left, a sidebar lists: Inicio, Información personal, Datos y privacidad, Seguridad (highlighted), Contactos y compartir, Páginas y suscripciones, and Información general. The main content area is titled 'Proteger tu cuenta' and shows 'Actividad de seguridad reciente' (No ha habido ninguna alerta ni actividad relacionadas con la seguridad en los últimos 28 días). Below this, 'Cómo inicias sesión en Google' lists: Verificación en dos pasos (Active desde: 24 jul), Llaves de acceso (Empiezar a usar llaves de acceso), Contraseña (Última modificación: 24 jul), Notificación de Google (1 dispositivo), Teléfono de recuperación (Añadir un número de teléfono móvil), Correo de recuperación (Verifica m.montalvillo98@gmail.com), and Códigos de verificación alternativos (10 códigos disponibles). At the bottom, it says 'Puedes añadir más opciones de inicio de sesión'. On the right, a separate window titled 'Contraseñas de aplicaciones' shows a single entry for 'Springboot': 'Creada: 24 jul; utilizada por última vez: 14:40'. Below this is a placeholder 'To create a new app specific password, type a name for it below...' with a 'App name' input field containing 'Springboot'.

```
# MAIL CONFIGURATION:
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=tfgmariomontalvilloherreuelo@gmail.com
spring.mail.password=kwldynkxrubqltd
spring.mail.properties.mail.smtp.starttls.enable=true
#spring.mail.properties.mail.smtp.starttls.required=true
spring.mail.properties.mail.smtp.auth=true
#spring.mail.properties.mail.smtp.connectiontimeout=5000
#spring.mail.properties.mail.smtp.timeout=5000
#spring.mail.properties.mail.smtp.writetimeout=5000
```

- Plantillas thymeleaf: gracias a este motor de creación de HTML puedo reutilizar las partes compartidas por los emails y cambiar solo lo necesario. Estas plantillas se almacenan en la carpeta “resources” y “templates”, en su interior, del proyecto java:



Debido al funcionamiento de los mails y lo capados que se hallan en cuanto a código encontré que la mejor manera de maquetar los mensajes creados era a través de tablas.

Aprovechando el funcionamiento de Thymeleaf, que permite la creación de bloques reutilizables de código, establecí varios archivos, cada uno relacionado con un tipo de objeto, en los que introduce todas las versiones creadas. Estos elementos serán los encontrados dentro de la carpeta “componentes”.

“cells” contiene el conjunto de plantillas de celdas a utilizar, “rows” las filas padre de estas y “tables” las tablas que contienen ambos elementos. Además, “styles.html” será el archivo que incluye la plantilla con los estilos CSS de los mensajes del mismo modo que “heads”, la cabecera.

Los elementos dentro de la carpeta “structures” serán simplemente los encargados de elegir cuáles de estos elementos se usan para cada email.

- Mail: un enum que cuenta con un valor por cada tipo de email existente (actualmente tres: de verificación, de cambio de contraseña y de juego disponible) con el título del mail, el asunto y la ruta de la plantilla de email a utilizar.

```
1 usage
VERIFICATION( action: "Verify account", subject: "Verify MYPAGE account", template: "structures/verification-mail.html"),
1 usage
PASSWORD( action: "New MY_PAGE password", subject: "Your password has been changed", template: "structures/new-password-mail.html"),
1 usage
AVAILABLE( action: "New game available", subject: "New MYPAGE game available", template: "structures/available-mail.html");
```

- MailCreatorService: encargado de a través de un enum Mail crear los emails con la información necesaria. Encadena una serie de métodos. Ejemplo de los utilizados para crear el email de verificación de email.

```
1 usage
@Override
public MimeMessage verifyUserMail(String mail, String code) throws MessagingException {
    final Context context = createContext();
    context.setVariable( name: "code", code);
    return createThymeleafMailWithMail(Mail.VERIFICATION, mail, context);
}
```

El primero es en el que crea el context (conjunto de variables que se transmitirán a la plantilla thymeleaf para que las utilice), se realiza toda la lógica independiente de ese mail (en este caso añadir el código de verificación al context) y después se llama al creador con Mail donde se pasa la instancia de enum en cuestión.

```
3 usages
private MimeMessage createThymeleafMailWithMail(Mail mail, String toMail, Context context) throws MessagingException {
    context.setVariable( name: "title", MY_PAGE);
    context.setVariable( name: "action", mail.getAction());
    return createThymeleafMailWith(toMail, mail.getSubject(), mail.getTemplate(), context);
}
```

En este paso se añade al context el título y la acción (subtítulo y asunto del mensaje) y se llama al último método con la información del enum.

```
1 usage
private MimeMessage createThymeleafMailWith(String toEmail, String subject, String template, IContext context) throws MessagingException{
    final MimeMessage mimeMessage = this.javaMailSender.createMimeMessage();
    final MimeMessageHelper message = new MimeMessageHelper(mimeMessage, multipart: false, encoding: "UTF-8");
    final String htmlContent = this.templateEngine.process(template, context);
    message.setTo(toEmail);
    message.setSubject(subject);
    message.setText(htmlContent, html: true);
    return mimeMessage;
}
```

Que crea la instancia de MimeMessage, procesa la plantilla, asigna el destinatario, el asunto y el texto y devuelve el mensaje construido.

- MailSenderService: servicio que llama al creador y envía el mensaje recibido al usuario a través de una instancia de JavaMailSender (de org.springframework.mail.javamail). Ejemplo enviando el email con el código de verificación de email:

```

@Usage
@Async
public VerifyCodeDTO sendVerifyMail(String mail){ Complexity is 4 Everything is cool!
    String code = CodeGeneratorUtil.generateCode();
    try{
        this.javaMailSender.send(mailCreatorService.verifyUserMail(mail, code));
    }catch(MessagingException e){
        throw new MailNotSendException();
    }
    return new VerifyCodeDTO(code);
}

```

3.5 Arquitectura de procesos periódicos:

```

public interface ITask {
    // @Scheduled(initialDelay = 1000, fixedRate = 1000000)
    // @Scheduled(cron = DO_VAR)
    no usages
    static final String DO_YEARLY = "@yearly";
    no usages
    static final String DO_ANNUALLY = "@annually";
    no usages
    static final String DO_MONTHLY = "@monthly";
    no usages
    static final String DO_WEEKLY = "@weekly";
    no usages
    static final String DO_DAILY = "@daily";
    1 usage
    static final String DO_MIDNIGHT = "@midnight";
    no usages
    static final String DO_HOURLY = "@hourly";
    no usages
    static final String EVERY_DAY_AT_TEN_AM = "0 0 10 * * *";
    1 usage 1 implementation
    void process();

    no usages
    default void run() { process(); }
}

```

Además de ser una api, el proyecto Spring Boot realiza diariamente algunos procesos. Cada uno de ellos implementa la interfaz ITask.

Esta interfaz contiene una serie de variables con distintos períodos en los que se pueden realizar las tareas o que sirven de guía para crear otros nuevos; y dos métodos, uno que tendrán que implementar todas las clases que implementen la interfaz, *process*, y *run*, que solamente incluye la llamada a *process*. Sirve para mantener la lógica de activación de tareas separada de la de lo que se busca realizar.

Ejemplo de proceso encargado de avisar a media noche de juego disponible a los usuarios con notificaciones activas:

```

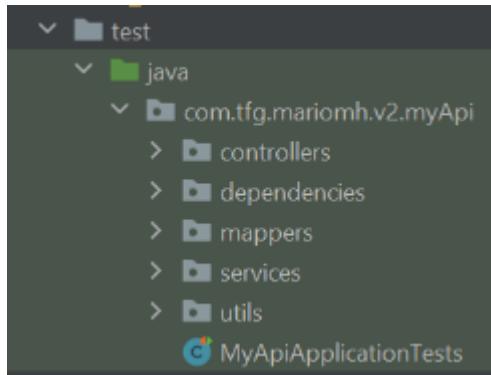
no usages
@Component
@Log
@AllArgsConstructor
public class NewGameAvailableTaskImpl implements ITask { Complexity is 4 Everything is cool!

    private final IMailSenderService mailService;
    private final GameServiceImpl gameService;

    1 usage
    @Scheduled(cron = DO_MIDNIGHT)
    public void process() { Complexity is 3 Everything is cool!
        log.info( msg: "Initializing new Game available process...");
        try {
            Game game = gameService.getDateWord(DateUtil.getTodayDate());
            mailService.sendNewGameAvailableMails();
            log.info( msg: "New game available process ended successfully");
        }catch(GameNotFoundWithDateResponseException e){
            log.info( msg: "There is no new game available today");
        }catch(MailNotSendException e){
            log.info( msg: "An error occurred during the sending of mails");
        }
    }
}

```

3.6 Arquitectura de pruebas:



Uno de mis objetivos con este proyecto era aprender a realizar pruebas completas y complejas para mis clases, métodos y código en general. Lo primero que encontré investigando es que existen métodos de prueba distintos para cada tipo de componente en Spring, por ello, cree una distribución de carpetas muy parecida a la del proyecto.

Los tests se pueden dividir en tres tipos, ordenados por su complejidad:

3.6.1 Tests estándar:

Son las pruebas normales en las que se prueba una funcionalidad básica de un método. Por ejemplo, mi generador de códigos aleatorios tiene un método que es *generar* que crea un string con un código aleatorio de diez caracteres dentro de CHARS (un String que hay en la misma clase que contiene las letras del diccionario en mayúscula y minúscula y los números del 1 al diez). Para probarlo he creado un test que sería el siguiente:

```
@RunWith(JUnit4.class)
public class CodeGeneratorUtilTest { Complexity is 3 Everything is cool!

    @Test
    public void generateCode(){
        String generatedCode = CodeGeneratorUtil.generateCode();
        Assertions.assertEquals( expected: 10, generatedCode.length());
        for(char character: generatedCode.toCharArray()){
            Assertions.assertTrue(CodeGeneratorUtil.CHARS.contains(String.valueOf(character)));
        }
    }
}
```

Valida que el código generado es de diez caracteres y que todos forman parte de CHARS.

Este tipo de tests se hacen sobre todo en las carpetas dependencies, mappers y utils.

3.6.2 Tests de componentes:

Para estos tests ya estamos trabajando con clases más complejas, del tipo de servicios o controllers que gracias a Spring se crean automáticamente y se relacionan entre sí.

Para probar toda esta funcionalidad fue necesario implementar la dependencia de Spring mockito-core, incluida en las clases test a través de la anotación @RunWith(mockitoJUnitRunner.class), que permite, resumiendo sus funcionalidades, crear instancias de esas clases como haría Spring y unirlas a un objeto padre (el que se quiere probar) dando valores a las respuestas que deberían dar en un contexto idílico, independientemente de su código real.

Es más fácil verlo en la práctica: UserServiceImplTest se encarga de probar UserServiceImpl un servicio que tiene dos atributos que son, a su vez, servicios por lo que usando mockito se crearía de esta manera:

```
5 usages
@InjectMocks
private UserServiceImpl userService;
2 usages
@Mock
private UserDao userDao;
2 usages
@Mock
private IUserMapper userMapper;
```

La anotación InjectMocks indica el objeto sobre el que se quieren hacer pruebas, cuyo código se quiere respetar, y Mock los objetos que se inyectan en él, cuyo código es irrelevante.

Ahora bien, todavía falta un paso para poder probar el servicio. No sería suficiente con crear los assertions y lanzar los métodos como haríamos en un test estándar, antes tenemos que indicar qué deben hacer (en caso de que hagan algo relevante para el servicio) los mocks inyectados cuando se usen en el código. De modo que para realizar una prueba de caso ideal de este método, por ejemplo:

```
@Override
public UserDTO getByIdInfoUser(String id){ Complexity is 3 Everything is cool!
    return userMapper.userToUserDTOWithoutPassword(userDao.findById(id).orElseThrow(
        () ->new UserNotFoundException()));
}
```

Hay antes que indicar con el método `Mockito.doReturn(objetoADevolver).when(servicioMock).nombreDelMétodoServicioMock(parametros)` a `userMapper` que cuando se llame a su `userToUserDTOWithoutPassword` y se le pase como parámetro un usuario se devuelva un `usuarioDTO` (`Mockito.doReturn(usuarioDTO).when(userMapper).userToUserDTOWithoutPassword(Mockito.any(User.class))`) y a `userDao` que cuando se haga `findById` y se le pase un long devuelva un usuario (`Mockito.doReturn(user).when(userDao).findById(Mockito.anyLong())`).

Este es un caso muy sencillo pero con esto se podría indicar que los métodos realizan todo tipo de acciones mucho más complejas, permitiendo probar todas las alternativas, como el manejo de excepciones.

3.6.3 Test de controladores:

Para estas pruebas fue necesaria otra dependencia además de Mockito-core: spring-boot-starter-web-flux, una librería que entre otros incluye clases clientes que permiten realizar peticiones lo que es muy útil para probar que los controladores de un proyecto están respondiendo a las peticiones correctas.

Usar uno de estos clientes es sencillo solo tendremos que crear en la clase test un atributo `WebTestClient` con la anotación `@Autowired`. La clase a su vez tendrá `@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)` que lanza el proyecto en un puerto aleatorio y en sus métodos de pruebas usará `@Test` de junit.jupiter.

Después usaremos el cliente para lanzar la petición a la ruta que busquemos con el método que utilice (GET, POST, PUT, DELETE...):

```
@Test
public void getByIdTest(){
    String token = TokenUtils.createToken(ID, List.of(Role.USER));
    ParameterizedTypeReference<UserDTO> typeRef = new ParameterizedTypeReference<UserDTO>() {};
    Mockito.doReturn(userDTO()).when(userService).getByIdInfoUser(Mockito.anyString());
    UserDTO responseBody = webTestClient.get()
        .uri("/users/token/{id}", ID)
        .captureOf()
        .headers(http -> http.setBearerAuth(token))
        .accept(MediaType.APPLICATION_JSON)
        .exchange()
        .expectStatus()
        .isOk()
        .expectBody(typeRef)
        .returnResult()
        .getResponseBody();
    Assertions.assertNotNull(responseBody);
    Assertions.assertEquals(ID, responseBody.getId());
}
```

En este ejemplo se ve el uso real de un mock con un `webtestclient`. Primero se establece el tipo de objeto que debería devolver la petición: un objeto de la clase `UserDTO`. Se establece que el servicio al que llama el controlador devuelve ese `UserDTO` y se lanza la petición a la uri concreta (/users/token/id).

Debería devolver un código 200, de ahí la línea “`.isOk()`”, “`.expectedBody(typeRef)`” indica el body que debería devolverse, que se extrae en las siguientes líneas para proceder a realizar las validaciones.

4 Diseño de la base de datos:

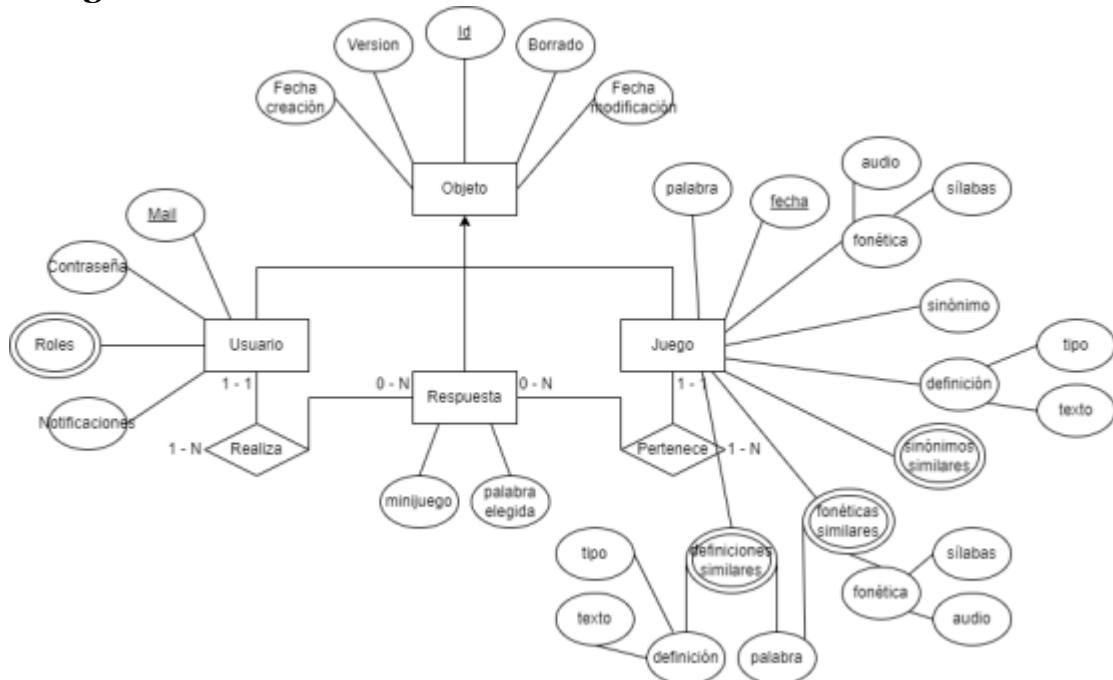
4.1 Arquitectura de la base de datos:

Decidí usar una base de datos noSQL para poder aprender su funcionamiento. Mi plan inicial era no usar relaciones pero debido a la naturaleza de mi proyecto esta parte acabó por ser imposible, era la manera más clara, rápida y fácil de implementarlo para luego llevar a cabo la funcionalidad.

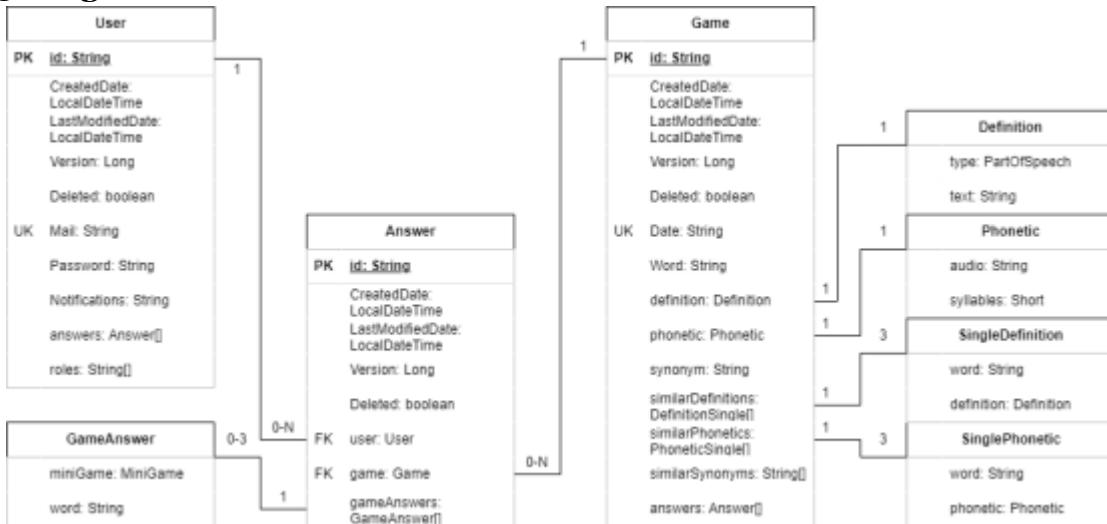
Para llevar estas relaciones al código Java me sirvo del identificador generador por MongoDB para la entidad, lo que en Spring se traduce en el uso de la anotación `@DocumentReference` sobre el atributo. Por ejemplo, una respuesta tiene un atributo de tipo usuario (esto es importante, no del tipo del id, que es String) con `@DocumentReference`, así desde la respuesta se puede acceder al usuario que la ha hecho y al juego sobre el que se hace y viceversa (más información en [4.4 Clases Java y su almacenamiento](#)).

Los usuarios y juegos cuentan con una lista de respuestas que de momento no se utiliza más que para validar que no se realizan dos respuestas al mismo juego por la misma persona pero que en un futuro podría servir para implementar muchas más funcionalidades como un sistema que muestre el porcentaje de jugadores que ha votado cada opción una vez se ha respondido un juego.

4.2 Diagrama entidad relación:



4.3 Diagrama de clases:



4.4 Clases Java y su almacenamiento:

Para poder guardar una clase Java como objeto en la base de datos Mongo es necesario que lleve la anotación `@Document`. Si queremos que se cree una tabla de ese objeto se usa `@Document("nombre_tabla")`.

BdObject (no lleva `@Document("nombre")`, ni siquiera sin nombre, pero todos los objetos que heredan de él sí):

```

public abstract class BdObject { Complexity is 6 It's time to do something...
    @Id
    protected String id;
    @Version
    protected Long version;
    protected Boolean deleted = false;
    @CreatedDate
    @Field("created_date")
    protected LocalDateTime createdDate;
    @LastModifiedDate
    @Field("last_modified_date")
    protected LocalDateTime lastModifiedDate;
    public void setVersion(Long version){ Complexity is 5 Everything is cool!
        if(this.version != null && !Objects.equals(this.version, version)){
            throw new VersionResponseException();
        }
        this.version = version;
    }
}
  
```

Esta clase sirve para dotar a los objetos de la base de datos de una serie de propiedades útiles para la lógica de negocio y mantener un historial de la aplicación:

- `CreatedDate`: fecha/hora de creación de un registro, lo que en un futuro permitiría crear sistemas adicionales.
- `LastModifiedDate`: fecha/hora de última modificación, gracias a lo que se podría conocer los usuarios más recientes.

- Versión: evita conflictos a la hora de actualizar objetos realizando una comparación. Si el objeto a modificar no comparte el mismo valor lanza una excepción.
- Deleted: para llevar a cabo un borrado lógico que permite al usuario librarse de su cuenta de la misma manera que lo haría si se borrara mientras la aplicación se aprovecha de su información.

La interfaz Word la implementan todos aquellos objetos que tienen una palabra para entre otras cosas las comparaciones puedan llevarse a cabo independientemente del objeto (muy útil para las validaciones):

```
public interface Word {
    6 implementations
    void setWord(String word);
    6 implementations
    String getWord();
}
```

Usuario:

```
public class User extends BdObject {
    private String mail;
    private String password;
    private Boolean notifications;
    private List<Role> roles;
    @JsonIgnore
    @DocumentReference(lazy = true)
    private List<Answer> answers;
}
```

```
public enum Role {
    USER, ADMIN;
}
```

Juego:

```
public class Game extends BdObject implements Word {
    @Indexed(name="word_word_game_uk", unique = true)
    private String word;
    @Indexed(name="date_word_game_uk", unique = true)
    private String date;
    private Definition definition;
    private Phonetic phonetic;
    private String synonym;
    private List<DefinitionSingle> similarDefinitions;
    private List<PhoneticSingle> similarPhonetics;
    private List<String> similarSynonyms;
    @JsonIgnore
    @DocumentReference(lazy = true)
    private List<Answer> answers;
```

```
public class Phonetic {
    private String audio;
    private Short syllables;
}

public class Definition {
    private PartOfSpeech type;
    private String text;
}
```

```
public class PhoneticSingle implements Word {
    private String word;
    private Phonetic phonetic;
}

public class DefinitionSingle implements Word {
    private String word;
    private Definition definition;
```

Respuesta:

```
public class Answer extends BdObject {
    @DocumentReference(lazy=true)
    private User user;
    @DocumentReference(lazy=true)
    private Game game;
    private List<GameAnswer> gameAnswers;
}

public class GameAnswer {
    private MiniGame miniGame;
    private String word;
}

public enum MiniGame {
    1 usage
    DEFINITION, PHONETIC, SYNONYM
}
```

MongoDB almacena los datos con formato de un archivo JSON, de modo que cada registro es un objeto de un array donde sus propiedades son los atributos de la clase Java y su valor, el valor. La única diferencia es que se añade una propiedad “_class” que indica a MongoDB la clase a la que pertenece ese objeto y su procedencia. De manera que, por ejemplo, un juego guardado sería de la siguiente manera (visionado con Mongo Compass):

```
_id: ObjectId('650b69666e5fde3dd43a65bb')
mail: "m.montalzuelo@gmail.com"
password: "123"
notifications: true
answers: Array (1)
  0: ObjectId('650b6c3e3ac01154232d395b')
version: 8
deleted: true
created_date: 2023-09-20T21:51:34.924+00:00
last_modified_date: 2023-09-20T22:03:42.573+00:00
_class: "com.tfg.mariomh.v2.myApi.models.entities.bdObjects.User"
```

Una respuesta a un juego tendría el siguiente formato:

```
_id: ObjectId('650b6c3e3ac01154232d395b')
user: ObjectId('650b69666e5fde3dd43a65bb')
game: ObjectId('64f46e62e046b127ba70eb2f')
gameAnswers: Array (2)
  0: Object
    miniGame: "SYNONYM"
    word: "light"
  1: Object
    miniGame: "PHONETIC"
    word: "light"
version: 1
deleted: false
created_date: 2023-09-20T22:03:42.490+00:00
last_modified_date: 2023-09-20T22:04:18.032+00:00
_class: "com.tfg.mariomh.v2.myApi.models.entities.bdObjects.Answer"
```

Y un juego sería de la siguiente manera con sus respectivos datos:

```
_id: ObjectId('64f46e62e046b127ba70eb2f')
word: "light"
date: "20231016"
definition: Object
  type: "NOUN"
  text: "Visible electromagnetic radiation. The human eye can typically detect ..."
phonetic: Object
  audio: "https://api.dictionaryapi.dev/media/pronunciations/en/light-uk.mp3"
  syllables: 1
  synonym: "ignite"
similarDefinitions: Array (3)
  0: Object
    word: "illumination"
    definition: Object
      type: "NOUN"
      text: "Adornment of books and manuscripts with colored illustrations. See ill."
  1: Object
  2: Object
similarPhonetics: Array (3)
  0: Object
    word: "late"
    phonetic: Object
      audio: "https://api.dictionaryapi.dev/media/pronunciations/en/late-uk.mp3"
      syllables: 2
  1: Object
  2: Object
similarSynonyms: Array (3)
  0: "bulbs"
  1: "wavelength"
  2: "cruiser"
answers: Array (3)
  0: ObjectId('650b6c3e3ac01154232d395b')
  1: ObjectId('650b72fcbb926411e708af53')
  2: ObjectId('652d7bfbb142c05a047e1826')
version: 10
deleted: false
last_modified_date: 2023-10-16T18:07:55.425+00:00
_class: "com.tfg.mariomh.v2.myApi.models.entities.Game"
```

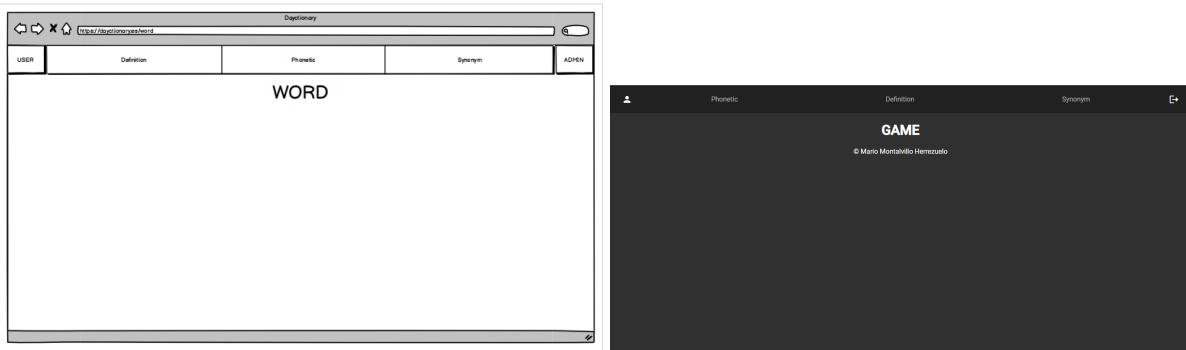
5 Interfaz:

5.1 Características generales:

5.1.1 Mock-ups:

Desde el momento en el que empecé a pensar en *Dayctionary*, la aplicación tenía un aspecto general muy concreto: Sencillo, nada recargado, poblado con elementos simples, cuadrados, de colores apagados, que no cansan la vista. En definitiva, un fondo nada relevante ni molesto que permitiera al usuario centrarse en lo importante: el aprendizaje.

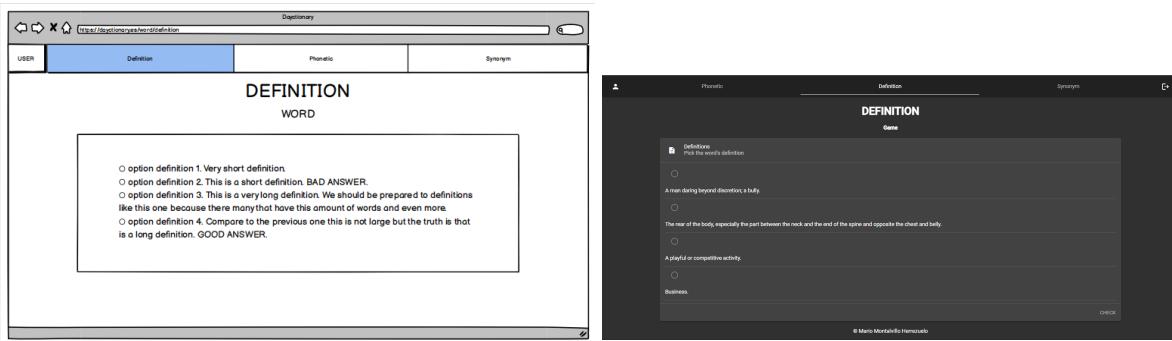
Este es el mockup que hice de mi página principal, como se puede ver muy poco a variado del diseño que acabé por realizar:



Lo mismo sucede con el menú lateral, el llamado menú de usuario:



Los juegos son quizá la parte que más se diferencia. El principal motivo es que mi primera idea fue no dejar al usuario replantearse sus preguntas, simplemente elegía la opción que quería y directamente se validaba pero en la práctica encontré ese manejo poco accesible. Con el botón “check”, el utilizado para contestar, vendrían las cabeceras con pequeñas explicaciones de los juegos con las que cuenta la página:



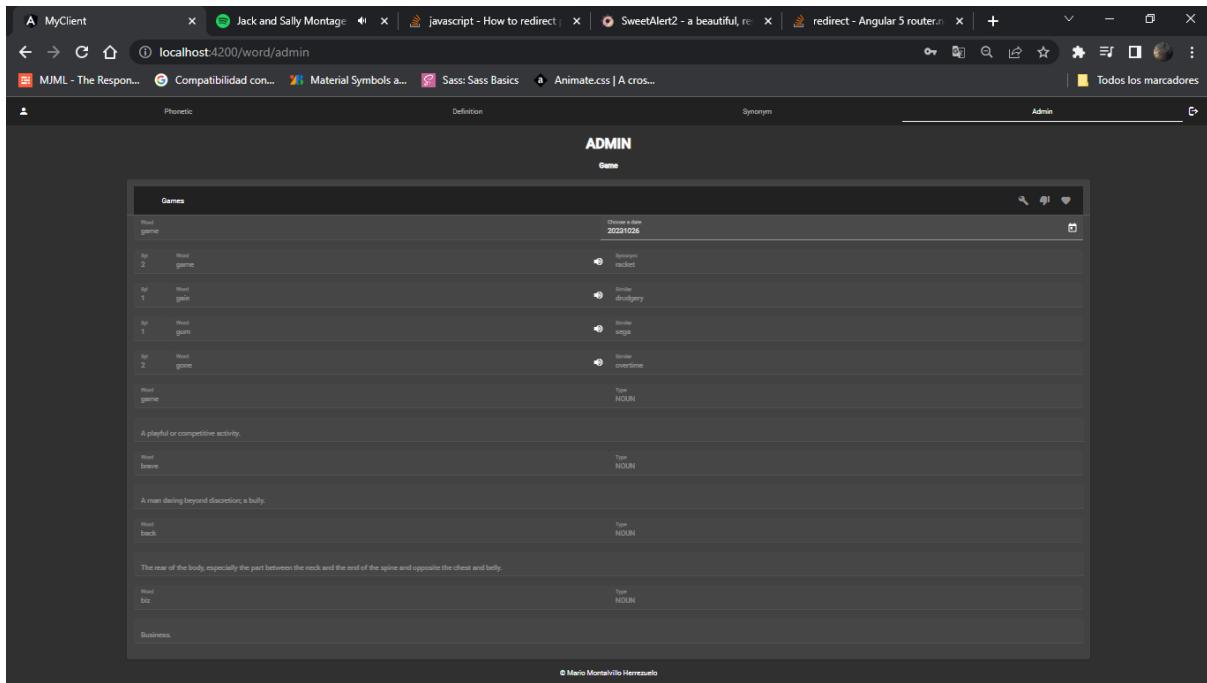
The image displays four wireframe prototypes of the Dayctionary interface, arranged in a 2x2 grid. The top row shows the 'PHONETIC' section, and the bottom row shows the 'SYNONYM' section. Each row contains a light gray prototype on the left and a dark gray prototype on the right.

- Top Left (Light Gray):** Shows a 'WORD' section with three radio button options: '(GOOD ANSWER)', '(BAD ANSWER)', and two empty circles. Each option has a speaker icon to its right.
- Top Right (Dark Gray):** Shows a 'WORD' section with four radio button options: 'Audios' (selected), 'Pick the word's audio', and three empty circles. Each option has a speaker icon to its right. A 'CHECK' button is at the bottom right.
- Bottom Left (Light Gray):** Shows a 'WORD' section with four radio button options: 'Three', 'Bad', 'Phrase', and 'Good'. Each option has a speaker icon to its right.
- Bottom Right (Dark Gray):** Shows a 'WORD' section with five radio button options: 'Related words' (selected), 'Pick the word's synonym', and four other words: 'overtime', 'racket', 'sega', and 'drudgery'. Each option has a speaker icon to its right. A 'CHECK' button is at the bottom right.

El menú de creación de los usuarios tampoco sufrió muchos cambios, una simple reordenación de la información para no tener tantos huecos en blanco:

The image shows a wireframe prototype of the Dayctionary Admin interface. The top navigation bar includes 'USER', 'Definition', 'Phonetic', 'Synonym', and 'ADMIN' (highlighted in blue). The main content area is titled 'ADMIN' and contains the following sections:

- WORD:** A text input field containing 'WORD' and a small icon.
- DEFINITIONS:** A list of four definitions, each preceded by a radio button:
 - option definition 4. Compare to the previous one this is not large but the truth is that is a long definition.
GOOD ANSWER.
 - option definition 1. Very short definition.
 - option definition 2. This is a short definition.
 - option definition 3. This is a very long definition. We should be prepared to definitions like this one because there many that have this amount of words and even more.
- PHONETIC:** A list of four items under 'Word' and three under 'Similar' (each preceded by a radio button and a speaker icon).
- SYNONYM:** A list of five items under 'GOOD' (each preceded by a radio button and a speaker icon).

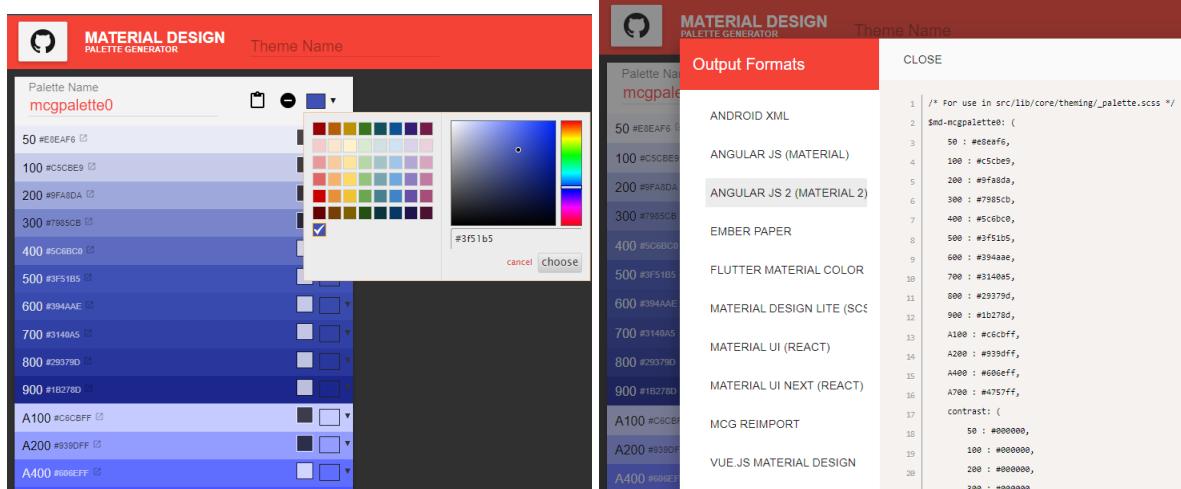


Para más información de los mockups acceder a los archivos adjuntados al respecto.

5.1.2 Tema oscuro y colores de la página:

Angular material, con SASS, permite un sistema de creación de temas y paletas de gamas de tres colores (primario (primary), destacado (accent) y aviso (warning)) muy fácil de configurar. Los colores utilizados de base son azul, rosa y rojo. Mi plan principal era integrar una gama de colores parecida pero a la hora de implementarla me di cuenta de que no quedaba del todo bien, por ese motivo acabé por optar por usar el blanco como color primario y destacado, manteniendo el rojo para los errores. Esto se hizo de la siguiente manera:

El primer paso es crear la paleta. Eso se puede hacer manualmente o, como hice yo, aprovechando alguna herramienta online. En mi caso yo usé Material Design Palette Generator (<http://mcg.mbitson.com/>), una página que genera toda una paleta de un color que el usuario introduzca y copiar el código generado de la pestaña ANGULAR JS 2 (MATERIAL 2).



```
$white: (
  50 : □#ffffff,
  100 : □#ffffff,
  200 : □#ffffff,
  300 : □#ffffff,
  400 : □#ffffff,
  500 : □#ffffff,
  600 : □#ffffff,
  700 : □#ffffff,
  800 : □#ffffff,
  900 : □#ffffff,
  A100 : □#ffffff,
  A200 : □#ffffff,
  A400 : □#ffffff,
  A700 : □#ffffff,
  contrast: (50 : □#000000,
    100 : □#000000,
    200 : □#000000,
    300 : □#000000,
    400 : □#000000,
    500 : □#000000,
    600 : □#000000,
    700 : □#000000,
    800 : □#000000,
    900 : □#000000,
    A100 : □#000000,
    A200 : □#000000,
    A400 : □#000000,
    A700 : □#000000,
  );
);
```

De esa forma yo generé mis paleta y para usarla solo hizo falta introducir la variable (\$white) en el archivo de estilos proveído por angular material (styles.scss).

```
// Define the palettes for your theme using the Material Design palettes available in palette.scss
// (imported above). For each palette, you can optionally specify a default, lighter, and darker
// hue. Available color palettes: https://material.io/design/color/
// $MyClient-primary: mat.define-palette(mat.$indigo-palette);
$MyClient-primary: mat.define-palette($white);
// $MyClient-accent: mat.define-palette(mat.$pink-palette, A200, A100, A400);
$MyClient-accent: mat.define-palette($white);

// The warn palette is optional (defaults to red).
$MyClient-warn: mat.define-palette(mat.$red-palette);

// Create the theme object. A theme consists of configurations for individual
// theming systems such as "color" or "typography".
$MyClient-theme: mat.define-dark-theme((color: (primary: $MyClient-primary,
  accent: $MyClient-accent,
  warn: $MyClient-warn,
  )));

// Include theme styles for core and each component used in your app.
// Alternatively, you can import and @include the theme mixins for each component
// that you are using.
@include mat.all-component-themes($MyClient-theme);
/* You can add global styles to this file, and also import other style files */
```

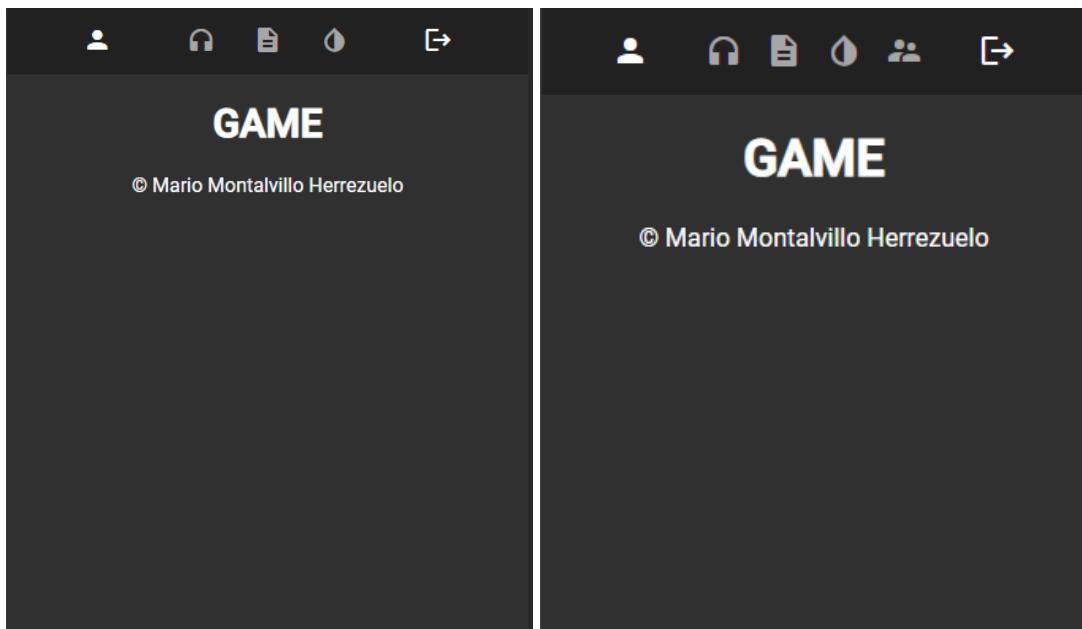
Otro punto que veía indispensable en *Dayctionary*, no solo por sus beneficios de cara a la vista y a la comodidad del usuario sino porque pensé que eso le recordaría a *Wordle*; era el tema oscuro. Para implementarlo, de nuevo, con angular material no hizo falta mucho, bastó, de hecho, con cambiar una sola palabra.

\$MyClient-theme: mat.define-light-theme(...) por
\$MyClient-theme: mat.define-dark-theme(...).

5.2 Adaptación a dispositivos móviles:

Debido a la naturaleza casual y rápida era indispensable que *Dayctionary* fuera una aplicación totalmente responsive, adaptada a todos los dispositivos. Hasta ahora, las capturas hemos estado viendo son de un terminal 1080 ya que a la hora de trabajar era más sencillo pero personalmente prefiero la vista móvil ya que es más sencilla todavía, sustituyendo muchos textos por iconos.

Esta es una captura de la página principal para usuarios y administradores:



Como se puede ver los nombres de los juegos han sido sustituidos por iconos. Esto se realiza mediante HTML y CSS:

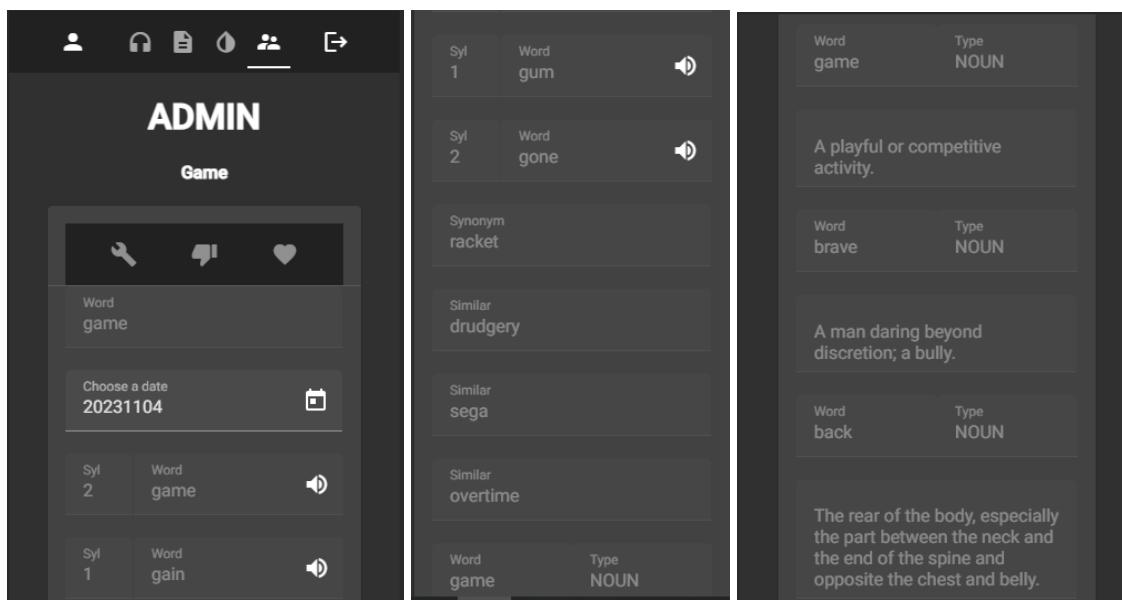
Si nos fijamos en el HTML del componente layout-menu-tab se ve que cada enlace tiene ambas cosas, el ícono (mat-icon) y el texto (span):

```
<nav mat-tab-nav-bar [tabPanel]="tabPanel">
  <a mat-tab-link *ngFor="let item of menuItems" routerLinkActive #rla="routerLinkActive" mat-tab-link
    [active]="rla.isActive" [routerLinkActiveOptions]="{{exact: true}}" [routerLink]="item.route">
    <mat-icon>{{item.icon}}</mat-icon>
    <span>{{item.title}}</span>
  </a>
```

```
.mdc-tab__text-label > mat-icon{
  display: none;
}
@media (max-width: 30em) {
  .mdc-tab__text-label > mat-icon{
    display: block;
  }
  .mdc-tab__text-label > span{
    display: none;
  }
}
```

Mediante CSS se elige cual se muestra: de base se quita el ícono (display: none) pero si el ancho de la pantalla es menor de 30 em (480 pixeles) se le vuelve a mostrar (display: block) y el que se esconde es el texto.

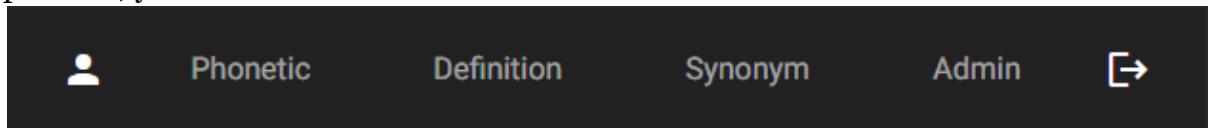
Algo muy similar se realiza en la página de administrador, donde con bajas resoluciones no se podía observar bien toda la información y tuve que en vez de esconderla cambiar su forma de fila a columna con flex.



5.3 Usabilidad:

Con lo que quería transmitir con *Dayctionary* era indispensable que fuera una página intuitiva y fácil de utilizar, donde en ningún momento el usuario se sintiera perdido o confundido.

Esto se consigue con una disposición de los elementos, como es el menú de navegación, de una manera semi-estandarizada como es la parte superior de la pantalla; y el uso de iconos conocidos como son el de usuario o cerrar sesión.



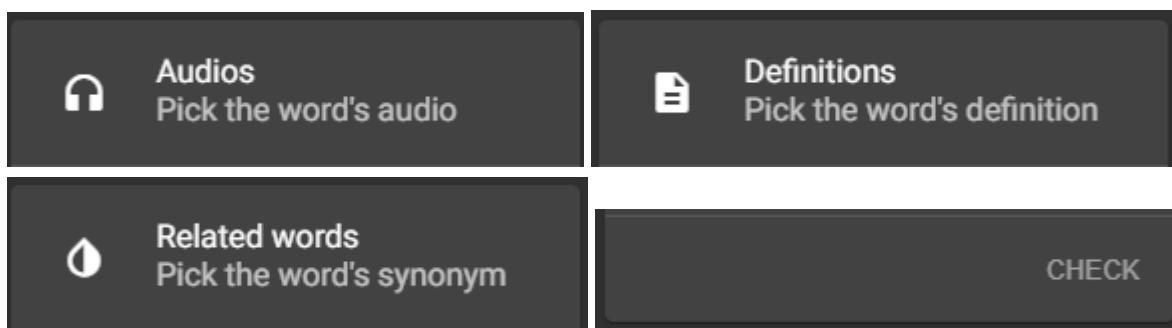
Soy consciente de que en la versión móvil esto se complica un poco debido al uso de iconos un poco más raros y que poca gente reconocerá pero es deliberado. El usuario tampoco entenderá muy bien qué es “Phonetic”, “Definition” o ninguna de las otras opciones la primera vez que entre a una página. Pero una vez dentro hará la relación. La intención con estos iconos, también visibles dentro de las páginas de manera que a cualquier usuario que haya visto algo antes de *Dayctionary* ya le resulten familiares es establecer un conjunto de elementos propios para la página.

Related words
Pick the word's synonym

- drudgery
- racket
- sega
- overtime

CHECK

Enseñando la página en una primera versión a varios conocidos, profesores y familiares me di cuenta de que *Dayctionary* y su sistema de juegos podría ser un poco confuso para las personas de entrada. Es por eso que creé una plantilla de visualización compartida, con una cabecera donde se introdujo pequeñas descripciones de uso y un pie con un botón de responder (“CHECK”):



Gracias a esta información la curva de aprendizaje de los usuarios se redujo considerablemente ya que se participa en todos los juegos de la misma manera, haciendo click en el input radio y despues en “CHECK”. Una vez un usuario juegue una vez ya sabrá jugar; lo único que tendrá que averiguar será lo que contestar, indicado en las descripciones de la cabeceras.

La única página sin información de este tipo es la de administrador por dos motivos:

- No quería que en caso de que el usuario, en caso de llegar a esa página por accidente (algo que creo imposible), supiera moverse. Nada más llegar vería los botones de creación bloqueados porque se muestra el juego del día y quizá entendería que no puede hacer nada.
- Y porque ya es una página demasiado recargada para mi gusto. Entiendo que todos los administradores recibirán el manual de usuario que se adjunta junto a esta memoria donde se explica paso a paso el uso de esa herramienta de creación.

5.4 Accesibilidad:

En cuanto a accesibilidad debido a la naturaleza de *Dayctionary*, había puntos, como el juego de la fonética era completamente imposible adaptar para todos los usuarios. Sin embargo, creo haber hecho todo lo posible aportando descripciones para todos los campos con los que el usuario puede interactuar mediante labels, manteniendo un orden natural en el salto de elemento a elemento hecho por usuarios que utilizan teclado en vez de ratón para moverse por la página y proveyendo un gama de colores (básicamente blanco y negro) que cualquier persona con cualquier grado de daltonismo puede percibir.

```
<button mat-icon-button matSuffix (click)="play($event)" (keydown.enter)="play($event)">
  <mat-icon>volume_up</mat-icon>
</button>
```

Por ejemplo, mediante la directiva “keydown.enter” se permite a los usuarios activar el botón de la fonética y escuchar un audio pulsando la tecla enter además de con el ratón (“click”).

```
<mat-form-field>
  <mat-label>Synonym</mat-label>
  <input type="text" matInput formControlName="simSyWord0">
</mat-form-field>
<mat-form-field *ngFor="let option of game?.similarSynonyms; let i = index">
  <mat-label>Similar</mat-label>
  <input type="text" matInput [formControlName]="'simSyWord'+(1+i)">
</mat-form-field>
```

Etiquetas a los campos de formulario implementadas mediante mat-label. Muestra de página de administrador, juego de sinónimo.

6. Autoevaluación y conclusiones:

6.1 Valoración del trabajo y dificultades encontradas:

Antes de empezar con *Dayctionary* mi experiencia con Spring Boot se reducía a un pequeño proyecto que realizamos en la asignatura de despliegue de aplicaciones web de segundo de desarrollo de aplicaciones web. No había instalado siquiera angular o hecho una consulta a MongoDB ni a cualquier otra base de datos no relacional. Y los proyectos más grandes que había creado habían sido un modesto diccionario de inglés que hice para la asignatura de desarrollo de clientes web y el proyecto final del primer curso de desarrollo de aplicaciones web, un cajero programado con java.

Resumiendo, no solo carecía de referencias de proyectos de la envergadura de *Dayctionary* sino de conocimientos la mayoría de las tecnologías que iba a utilizar para llevarlo a cabo.

No es de extrañar entonces que haya sido un proceso arduo y extenso, que me ha exigido un aprendizaje y evolución constantes para poder seguir adelante.

Empecé realizando los cursos mencionados en [2.3 Recursos humanos](#) para familiarizarme con los framework y tan pronto me encontré con el que sería el mayor de los problemas a los que me enfrentaría durante toda la elaboración del proyecto: la documentación obsoleta.

En angular no fue tanto problema, durante los cursos me encontré por ejemplo con el uso de formularios de plantilla (template) en vez de dinámicos (reactive), los recomendados ahora y sobre los que me acabaría informando por otros medios para utilizar. La información que encontraba podía estar desactualizada debido a su antigüedad pero funcionaba, en Spring Boot, no.

Esto se notó sobre todo en la implantación de la generación y validación de tokens JWT. Me di cuenta de que la mayoría de documentación que encontraba hacía mención a librerías que se han tachado de deprecated sino han desaparecido por completo de manera que implantar esta tecnología acabó llevando mucho trabajo.

Aprovechando que hablamos de las librerías, otro gran problema con el que me crucé en mi API fue lidiar con los conflictos generados entre ellas. Tuve que renunciar al uso de Mapstruct porque fui incapaz de arreglar unos fallos derivados de mi versión de Maven y Lombok.

En angular no tuve mucho problema en este tema, pero no quiere decir que todo fuera fácil. El uso de angular material y su componentes en ocasiones me hacía pensar que hubiera sido mejor no utilizar esa librería y hacer todo por mí mismo debido al gran número de bugs generados del modo en el que trabaja. Sin embargo, es innegable que ha sido precisamente esta librería la que más trabajo de cara al front-end y puede que a todo el proyecto me haya ahorrado, permitiéndole tener un aspecto visual muy estilizado con muy poco código.

Otro punto que todavía a día de hoy no acaba de convencerme de angular es el gran número de carpetas y archivos con los que trabaja. Un componente cuenta con mínimo dos (el archivo html y el typescript), que en la mayoría de casos son tres por el archivo de estilos, SASS en mi caso. Esto viene bien para separar funcionalidad y mantener en cada archivo lo justo y necesario pero debido probablemente a mi inexperiencia, aunque he oído quejas similares en internet y hablando con otra gente que lleva años con angular, muy pocas veces conseguía englobar toda la funcionalidad en un componente o servicio de manera que fuera prácticamente independiente de modo que cuando ocurría un error seguir el flujo entre todos esos archivos se volvía una tarea muy compleja.

Todo esto, sin embargo, son quejas menores. Lo cierto es que si tuviera que volver a desarrollar *Dayctionary* lo haría de nuevo con estos dos frameworks y muchas,

sino todas, las librerías utilizadas porque me parecen dos de los mejores framework y librerías indispensables para lo que busco.

Lo que sí que cambiaría sería el uso de MongoDB. Aunque no me arrepiento de haber usado este sistema de base de datos, al contrario, me alegro de haber podido aprender sus bases y comprobar su funcionamiento; no creo que sea el sistema más adecuado para mi aplicación debido al claro sistema de relaciones con el que cuenta.

Lo mismo pasaría con mi modo de plantearlo, porque a pesar de haber intentado crear una estructura inicial y un sistema de capas en ambos proyectos antes de empezar a programar todo acababa siempre complicándose y requería cambios y ejecuciones no planeadas de modo que tuve que repetir y rehacer mucho trabajo. Si lo volviera a hacer de nuevo dedicaría mucho más tiempo a esa primera fase de análisis ya que creo que de cara al futuro ahorraría muchos contratiempos, trabajo y quebraderos de cabeza.

También cambiaría el aspecto visual, aunque fuera un poco para dotarlo de cierta personalidad de la que creo carece por el uso de tantos elementos de librerías.

Y, sobretodo de cara a profesionalizar y comercializar el proyecto, me gustaría utilizar otras APIs para recibir la información mucho más completas y mejor planteadas que las usadas ya que mucha de la complejidad de la API Spring Boot se debe al modo de trabajar de estas y muchas veces viendo los resultados en los juegos creado no creo que haya merecido la pena el trabajo que ha llevado adaptarse a ellas.

6.2 Valoración de la herramienta o aplicación desarrollada:

Dayctionary es lo que planeaba. No esperaba crear una aplicación tan puntera, imaginativa ni rompedora como lo es *Wordle* ni muchas otras del mercado, al final idear algo así, tan sencillo e intuitivo y a la vez complejo y difícil lleva tiempo y una imaginación con la que no cuenta todo el mundo.

Quería crear el prototipo de una modesta página de aprendizaje de inglés, que en cierta manera funcionara un poco de pequeño tributo a todas esas páginas mejores en las que he pasado tantas horas estos últimos años y creo que he cumplido.

No es tan completa como me gustaría, quería añadir otros juegos, como uno de averiguar las traducciones pero no encontré APIs gratuitas que facilitaran esa información; y el juego de los sinónimos todavía a día de hoy no me convence demasiado, siento que le falta una vuelta.

El diseño no es para nada lo que imaginaba cuando pensé por primera vez en la página. En mi cabeza había planeado un aspecto visual muy concreto que se asemejaba a carpetas, post-its, subrayados y todo ese tipo de elementos tan relacionados con el estudio. No encontré sin embargo manera fácil y clara de integrarlo.

Tampoco es cien por cien fiable, el creador todavía requiere de pulido y por eso es tan necesaria esa revisión realizada antes de publicar los juegos pero creo que es un primer paso en la dirección adecuada si algún día quiero crear una página real de este tipo o, realmente, una página en general. Algo que me gustaría hacer en el futuro.

6.3 Conclusiones finales:

Desarrollar *Dayctionary* ha sido, sin duda, una de las cosas más difíciles que he hecho en mi vida pero sería injusto limitarse a decir eso sin mencionar lo mucho que he aprendido y, siendo sinceros, disfrutado.

Aunque creo que todo esto se puede resumir anunciando que ahora mismo me encuentro en la fase de análisis de la versión 3.0 de *Dayctionary* donde entre otras muchas cosas planeo separar la API en dos, una para gestionar todo el sistema de

usuarios y otra para los juegos; establecer un sistema de comodines parecidos a los del famoso programa: Quién quiere ser millonario (uno del cincuenta por ciento que reduzca las opciones de cuatro a dos, otro del público que muestre el número de usuarios que ha respondido a cada opción y otro de pista que según el juego aporte una pequeña guía).

Dudo que sea la última vez que me encuentre empezando de cero este proyecto ya que soy consciente de que aún no dispongo de los conocimientos suficientes. Sin embargo, esa es gran parte de la gracia de nuestro sector. Hay multitud de cosas por aprender, cada vez más, todo se halla en constante renovación y cambio. Solo hay que ver todos los problemas que me he encontrado de cara a documentación obsoleta o como en unos pocos años las inteligencias artificiales han pasado a ser una herramienta indispensable.

Otro de los cambios que había planeado para la versión 3.0, ya que lo menciono, era intercambiar el uso de APIs de terceros por el de una de estas tecnologías ahora que ha pasado un poco de tiempo y hay mucha más información y guías al respecto. Porque creo que podría mejorar el rendimiento de mi aplicación y sobre todo porque tengo curiosidad de entender su funcionamiento.

Porque disfruto de cada segundo que paso programando, planteando proyectos o aprendiendo sobre el tema.

7 Bibliografía:

- [Duda] Solución a CSRF() deprecated de Spring Security? | Foro Alura Latam. (s. f.). Alura Latam. <https://app.aluracursos.com/forum/topico-duda-solucion-a-csrf-deprecated-de-spring-security-213325> Consultado: 23/09/2023
- [Solved]-WebTestClient is null when autowired in a @SpringBootTest-Springboot. (s. f.). Hire Developers, Free Coding Resources for the Developer. <https://www.appsloveworld.com/springboot/100/40/webtestclient-is-null-when-autowired-in-a-springboottest> Consultado: 07/10/2023
- Agudelo, J. (2021, 14 diciembre). ¿Cómo enviar un mail con Spring Boot? - Karibublog - Medium. <https://medium.com/karibu-blog/c%C3%B3mo-enviar-un-mail-con-spring-boot-f86c2f7af678> Consultado: 11/11/2023
- Agudelo, J. (2021, 14 diciembre). ¿Cómo enviar un mail con Spring Boot? - Karibublog - Medium. <https://medium.com/karibu-blog/c%C3%B3mo-enviar-un-mail-con-spring-boot-f86c2f7af678> Consultado: 24/07/2023
- Animate.Css | a cross-browser library of CSS animations. (s. f.). <https://animate.style/> Consultado: 14/09/2023
- autho.com. (s. f.). JWT.IO. JSON Web Tokens - jwt.io. <https://jwt.io/> Consultado: 24/09/2023
- Baeldung, & Baeldung. (2020). Spring ResponseStatusException | Baeldung. <https://www.baeldung.com/spring-response-status-exception> Consultado: 05/08/2023
- Baeldung, & Baeldung. (2022). Return only specific fields for a query in spring data MongoDB | Baeldung. <https://www.baeldung.com/mongodb-return-specific-fields> Consultado: 30/07/2023
- BCrypt: Empty encoded password with Spring Security. (s. f.). Stack Overflow. <https://stackoverflow.com/questions/56559059/bcrypt-empty-encoded-password-with-spring-security> Consultado: 23/09/2023
- Boulila, I. (2022, 12 marzo). How to create a custom security expression method in Spring Security. <https://medium.com/@islamboulila/how-to-create-a-custom-security-expression-method-in-spring-security-e5b6353f062f> Consultado: 24/09/2023
- Calle, N. R. (2023). Ejemplos de testing en Spring Boot. Refactorizando. <https://refactorizando.com/ejemplos-testing-spring-boot/> Consultado: 30/08/2023
- Can I email. . . (s. f.). <https://www.caniemail.com/search/?s=margin> Consultado: 29/07/2023
- Can't connect to MonGODB Atlas from Render Web-Hosted app. (2022, 8 octubre). MongoDB Developer Community Forums. <https://www.mongodb.com/community/forums/t/cant-connect-to-mongodb-atlas-from-render-web-hosted-app/192110> Consultado: 10/11/2023
- Class has been compiled by a more recent version of the Java environment. (s. f.). Stack Overflow. <https://stackoverflow.com/questions/47457105/class-has-been-compiled-by-a-more-recent-version-of-the-java-environment> Consultado: 10/11/2023
- Cloud application hosting for developers | Render. (s. f.). Cloud Application Hosting for Developers | Render. <https://render.com/> Consultado: 10/11/2023

- Compatibilidad con CSS. (s. f.). *Google for Developers.*
<https://developers.google.com/gmail/design/css?hl=es-419> Consultado: 19/08/2023
- David. (2023, 17 marzo). *Host a Spring Boot application for free on Render.* JavaWhizz.
<https://javawhizz.com/2023/03/host-a-spring-boot-application-for-free-on-render> Consultado: 10/11/2023
- Deploying Spring Boot applications.* (s. f.).
<https://docs.spring.io/spring-boot/docs/current/reference/html/deployment.html> Consultado: 10/11/2023
- Detecting syllables in a word.* (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/405161/detecting-syllables-in-a-word> Consultado: 29/08/2023
- Ditschedev. (s. f.). *GitHub - ditschedev/Mailo: This is a client for sending emails easily from a Spring Boot REST API. It makes use of Mustache for templating and is adaptable.* GitHub. <https://github.com/ditschedev/mailo> Consultado: 31/07/2023
- Documentation for MJML - the Responsive Email Framework.* (s. f.).
<https://documentation.mjml.io/> Consultado: 31/07/2023
- Easily send beautiful and templated emails with Java (Spring Boot) – Toby Christopher.* (2020, 1 diciembre). Toby Christopher.
<https://ditsche.dev/blog/mjml-emails-with-spring-boot> Consultado: 31/07/2023
- Exception in Monitor thread while connecting to server localhost:27017 while accessing MongoDB with Java.* (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/35415308/exception-in-monitor-thread-while-connecting-to-server-localhost27017-while-acc> Consultado: 10/11/2023
- Favicon.ico generator.* (s. f.). <https://www.favicon.cc/>
- How to add token in HTTP Request Angular.* (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/63167450/how-to-add-token-in-http-request-angular> Consultado: 26/09/2023
- How to decode the JWT encoded token payload on client-side in Angular?* (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/48075688/how-to-decode-the-jwt-encoded-token-payload-on-client-side-in-angular> Consultar: 25/09/2023
- How to send JWT Token as Authorization Header in Angular 6.* (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/52468071/how-to-send-jwt-token-as-authentication-header-in-angular-6> Consultado: 26/09/2023
- JavaMail API - Sending an HTML email.* (s. f.).
https://www.tutorialspoint.com/javamail_api/javamail_api_send_html_in_email.htm Consultado: 25/07/2023
- Java Spring Security - User.withDefaultPasswordEncoder() is deprecated?* (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/49847791/java-spring-security-user-withdefaultpasswordencoder-is-deprecated> Consultado: 23/09/2023
- Kounang, F. (2021, 16 diciembre). How to deploy Angular App to Netlify (Simple and Clean). Medium.
<https://medium.com/weekly-webtips/how-to-deploy-angular-app-to-netlify-simple-and-clean-48faba1f314> Consultado: 10/11/2023
- Lahoti, A. (2021, 16 diciembre). Send email with thymeleaf template in spring boot. CodingNConcepts.

- <https://codingnconcepts.com/spring-boot/send-email-with-thymeleaf-template/>
 Consultado: 25/07/2023
- Mapping JPA entities into DTOs in Spring Boot using MapStruct.* (s. f.). Autho - Blog.
<https://autho.com/blog/how-to-automatically-map-jpa-entities-into-dtos-in-spring-boot-using-mapstruct/> Consultado: 29/07/2023
- MapStruct + Lombok together not compiling: unknown property in result type.* (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/47676369/mapstruct-and-lombok-not-working-together> Consultado: 26/08/2023
- Material Design Theme & Palette Color generator.* (s. f.).
<http://mcg.mbitson.com/#!mcgpalette0=%23fofofo&mcgpalette1=%23afeodo&mcgpalette2=%23dbc4eo&mcgpalette3=%23b9c2fo&mcgpalette4=%23000000&mcgpalette5=%235694f&mcgpalette6=%23be7394&themename=mcgtheme>
 Consultado: 30/09/2023
- Material symbols and icons - Google Fonts.* (s. f.). Google Fonts.
<https://fonts.google.com/icons> Consultado: 29/09/2023
- Method Security :: Spring Security.* (s. f.).
<https://docs.spring.io/spring-security/reference/6.0-SNAPSHOT/servlet/authorization/method-security.html> Consultado: 24/09/2023
- Method Security :: Spring Security.* (s. f.-b.).
<https://docs.spring.io/spring-security/reference/servlet/authorization/method-security.html> Consultado: 24/09/2023
- Minh, N. H. (s. f.). *Spring Security JWT role-based authorization tutorial.*
<https://www.codejava.net/frameworks/spring-boot/spring-security-jwt-role-based-authorization> Consultado: 24/09/2023
- MJML - the Responsive Email Framework.* (s. f.). <https://mjml.io/> Consultado: 31/07/2023
- Mocking JWT token in @SpringBootTest with WebTestClient.* (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/66278129/mock-jwt-token-in-springboot-with-webtestclient> Consultado: 08/10/2023
- Moment.js | Home.* (s. f.). <https://momentjs.com/> Consultado: 13/09/2023
- MongoDB. (s. f.). *MongoDB Atlas | Plataforma de datos multicloud para desarrolladores | MongoDB.* <https://www.mongodb.com/es/atlas> Consultado: 10/11/2023
- Muskan.Lama. (2023). Step by step guide : Sending emails in Spring Boot | TO THE NEW blog. TO THE NEW BLOG.
<https://www.tothenew.com/blog/step-by-step-guide-sending-emails-in-spring-boot/> Consultado: 24/07/2023
- Paraschiv, E., & Paraschiv, E. (2022). Error handling for REST with Spring | Baeldung.
<https://www.baeldung.com/exception-handling-for-rest-with-spring> Consultado: 05/08/2023
- Riecks, P. (2021). Spring WebTestClient for efficient REST API testing. *rieckpil.*
<https://rieckpil.de/spring-webtestclient-for-efficient-testing-of-your-rest-api/>
 Consultado: 07/10/2023
- SaSS: Syntactically awesome style sheets.* (s. f.). <https://sass-lang.com/> Consultado: 10/10/2023
- Scribbr. (2022, 31 agosto). *Formato con el generador de Scribbr.*
<https://www.scribbr.es/citar/> Consultado: diariamente.

- Sending email in spring with Thymeleaf - thymeleaf.* (s. f.). <https://www.thymeleaf.org/doc/articles/springmail.html> Consultado: 25/07/2023
- Set which fields are returned — MongoDB Compass.* (s. f.). <https://www.mongodb.com/docs/compass/master/query/project/> Consultado: 30/07/2023
- Singh, R. (2022, 18 febrero). *Spring 5 WebClient and WebTestClient tutorial with examples.* CalliCoder. <https://www.callicoder.com/spring-5-reactive-webclient-webtestclient-examples/> Consultado: 07/10/2023
- Spring Data MongoDB annotation @CreatedDate isn't working, when ID is assigned manually.* (s. f.). Stack Overflow. <https://stackoverflow.com/questions/35584271/spring-data-mongodb-annotation-createddate-isnt-working-when-id-is-assigned-m> Consultado: 19/09/2023
- Spring Data MongoDB - Relation Modelling.* (2021, 29 noviembre). Spring Data MongoDB - Relation Modelling. <https://spring.io/blog/2021/11/29/spring-data-mongodb-relation-modelling> Consultado: 21/09/2023
- Stemmer, implementing the porter stemming algorithm.* (s. f.). Gist. <https://gist.github.com/lclakmal/667d8ecb620a0cce7d3dedae80a2c013> Consultado: 29/08/2023
- SweetAlert2.* (s. f.). <https://sweetalert2.github.io/> Consultado: 13/09/2023
- Testing method security.* (s. f.). <https://docs.spring.io/spring-security/site/docs/4.2.x/reference/html/test-method.html> Consultado: 24/09/2023
- Todo TIC Academy. (2022, 20 octubre). *Protege tu API usando JWT y Spring Security (2022)* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=p-Odh3MZJc> Consultado: 23/09/2023
- Todo TIC Academy. (2022b, diciembre 22). *Autenticación JWT con Spring Boot 3 y Angular 15* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=yOxxA3DpgnU> Consultado: 25/09/2023
- Tomcat exception cannot call SendError() after the response has been committed?* (s. f.). Stack Overflow. <https://stackoverflow.com/questions/20813496/tomcat-exception-cannot-call-senderror-after-the-response-has-been-committed> Consultado: 24/09/2023
- Using @DocumentReference for relations in Spring Boot MongoDB | Bootify.io.* (2022, 23 noviembre). <https://bootify.io/mongodb/document-reference-in-spring-boot-mongodb.html> Consultado: 21/09/2023
- Vatana. (2023, 9 junio). *How to send email in SpringBoot | SpringBoot Tutorial.* DEV Community. <https://dev.to/vatana7/how-to-send-email-in-springboot-springboot-tutorial-4jjm> Consultado: 11/11/2023
- WARNING in budgets, maximum exceeded for initial.* (s. f.-b). Stack Overflow. <https://stackoverflow.com/questions/53995948/warning-in-budgets-maximum-exceeded-for-initial> Consultado: 10/11/2023