UNIVERSIDAD SAN PABLO - CEU

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA DE SISTEMAS DE INFORMACIÓN



TRABAJO FIN DE GRADO

Diseño e Implementación de una

aplicación RESTful para la disminución del abandono en el primer año universitario

Design and Implementation of a

RESTful application to reduce dropout in the first year of university



Junio 2022

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Datos del alumno   |  | | --- | | Nombre: DAVID RECIO ARNÉS |   Datos del Trabajo   |  | | --- | | TÍTULO DEL PROYECTO:  Diseño e Implementación de una aplicación RESTful  para la disminución del abandono en el primer año universitario. |   Tribunal calificador   |  |  | | --- | --- | | Presidente: | Fdo.: |  |  |  | | --- | --- | | Secretario: | Fdo.: |  |  |  | | --- | --- | | Vocal: | Fdo.: |  |  | | --- | | Reunido este tribunal el \_\_\_ /Junio/2022, acuerda otorgar al Trabajo Fin de Grado presentado por D. David Recio Arnés la calificación de \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |

Resumen

En la actualidad, los jóvenes que están en el proceso de acceder a la universidad se enfrentan a una problemática que es la elección de su grado o carrera universitaria, puesto que normalmente se basan únicamente en su vocación. Muchos de ellos se equivocan en sus elecciones, lo cual se ve reflejado en el abandono del primer curso del grado, siendo este punto la motivación central para la realización de este TFG, pues mediante la creación de un Servicio Web RESTful especializado se garantiza al estudiante la orientación adecuada para la elección de su grado, a través de formularios estandarizados con bases psicológicas, psicotécnicas y pedagógicas, asegurando una mayor continuidad y evitando así el abandono universitario. El Servicio Web RESTful realizado en este TFG da un servicio óptimo mediante dos formularios; uno dónde se orienta al estudiante sobre los grados universitarios (ingeniería, ciencias sociales, artes y otras ciencias) y otro formulario que mide la capacidad de concentración para ver cuánto esfuerzo le puede suponer elegir la carrera que desea. Por último, se toman las notas del usuario, y se da así una respuesta final mediante sugerencias en la facilidad o no del grado que desea.

Palabras Clave

Estudiante, Servicios Web, test estandarizados, abandono universitario, elección, estudios, apoyo.

Abstract

Currently, young people who are in the process of accessing university face a problem that is the choice of their degree or university career, because they are normally based only on their vocation. Many of them make mistakes in their choices, which is reflected in the abandonment of the first year of the degree, this point being the central motivation for the completion of this TFG, since by creating a specialized RESTful Web Service the student is guaranteed adequate guidance for choosing your degree, through standardized forms with psychological, psychotechnical and pedagogical bases, ensuring greater continuity and thus avoiding university dropout. The RESTful Web Service carried out in this TFG provides an optimal service through two forms; one where the student is oriented on university degrees (engineering, social sciences, arts and other sciences) and another form that measures the ability to concentrate to see how much effort it may take to choose the career you want. Finally, the user's notes are taken, and a final answer is given by means of suggestions at the ease or not of the degree you want.

Keywords

Student, Web Services, standardized tests, university dropout, election, studies, support.

Índice de contenidos

Contenido

[1 Introducción 1](#_Toc136166136)

[1.1 Objetivos 2](#_Toc136166137)

[2 Gestión del proyecto 2](#_Toc136166138)

[2.1 Modelo de ciclo de vida 2](#_Toc136166139)

[2.2 Papeles desempeñados en el proyecto 4](#_Toc136166140)

[2.2.1 Roles del tutor 4](#_Toc136166141)

[2.2.2 Roles del estudiante 4](#_Toc136166142)

[2.3 Planificación 4](#_Toc136166143)

[2.4 Presupuesto 5](#_Toc136166144)

[2.4.1 Costes en la fase de desarrollo 5](#_Toc136166145)

[2.4.2 Costes en la fase de producción 5](#_Toc136166146)

[2.5 Ejecución 6](#_Toc136166147)

[3 Estado del arte 7](#_Toc136166148)

[3.1 ¿Qué son los Servicios Web y cómo funcionan? 7](#_Toc136166149)

[3.1.1 Servicios Web tradicionales 8](#_Toc136166150)

[3.1.2 Servicios RESTful (estilo arquitectónico REST) 8](#_Toc136166151)

[3.1.3 Modelo de madurez de Richardson 10](#_Toc136166152)

[4 Análisis 11](#_Toc136166153)

[4.1 Análisis de dominio 11](#_Toc136166154)

[4.2 Especificación de requisitos 17](#_Toc136166155)

[4.3 Análisis de seguridad 18](#_Toc136166156)

[5 Diseño 18](#_Toc136166157)

[5.1 Arquitectura del Sistema 19](#_Toc136166158)

[Para poder cubrir el análisis del capítulo 4, se ha elegido una arquitectura que se divide en dos partes, la primera parte es el Backend donde se desarrolla la parte funcional de la API y la base de datos, y la segunda parte un ligero Frontend donde se encuentra una interfaz amigable que consume los servicios de la API de forma amigable. 19](#_Toc136166159)

[5.2 Diseño de subsistema backend 20](#_Toc136166160)

[5.2.1 Diseño de los Srevicios RESTful 20](#_Toc136166161)

[5.2.2 Diseño de la base de datos 23](#_Toc136166162)

[5.3 Diseño del subsistema frontend 24](#_Toc136166163)

[6 Implementación 24](#_Toc136166164)

[6.1 Implementación de backend 25](#_Toc136166165)

[6.1.1 Estructura e implementación de la lógica de negocio con Play Framework 25](#_Toc136166166)

[6.1.2 Estructura e implementación de la BBDD 25](#_Toc136166167)

[6.2 Referencia al repositorio de software 25](#_Toc136166168)

[6.3 Manuales 25](#_Toc136166169)

[7 Pruebas y validación 25](#_Toc136166170)

[8 Conclusiones y líneas futuras 25](#_Toc136166171)

[Bibliografía 26](#_Toc136166172)

[Anexos 29](#_Toc136166173)

[1. Glosario de definiciones 29](#_Toc136166174)

[1.1. JSON 29](#_Toc136166175)

[1.2. Token Bearer 30](#_Toc136166176)

[1.3. Docker 30](#_Toc136166177)

[2. Definición de los recursos de la API 30](#_Toc136166178)

[2. Métodos HTTP de los recursos 33](#_Toc136166179)

Índice de ilustraciones

[Ilustración 1. Metodología en Cascada 3](#_Toc135865803)

[Ilustración 2. Planificación realizada 8](file:///C:\Users\david\Documents\GitHub\tfg_Contenedor\Doc\david-recio-memoria.docx#_Toc135865804)

[Ilustración 3. Servicio Web 9](#_Toc135865805)

[Ilustración 4. Servicio Web Tradicional 10](#_Toc135865806)

[Ilustración 5. Diagrama de una estructura REST 11](file:///C:\Users\david\Documents\GitHub\tfg_Contenedor\Doc\david-recio-memoria.docx#_Toc135865807)

[Ilustración 6. Niveles de madurez de los Servicios Web REST 12](#_Toc135865808)

Índice de tablas

[Tabla 1. Coste de la luz por horas trabajadas…………………………………………………….…7](#_Toc106842494)

Tabla 2. Tasas de abandono el primer año universitario…………………………………….24

Tabla 3. Evaluación CHASIDE……………………………………………………………………………..25

Tabla 4. Análisis de seguridad…………………………………………………………………………….32

Tabla 5. Exposición del recurso: /usuarios ……………………………………………….……….39

Tabla 6. Exposición del recurso: /usuarios/:id…………………………………………………...41

Tabla 7. Exposición del recurso: /usuarios/:idUsuarios/notas/:id………………..…….42

Tabla 8. Exposición del recurso: /usuarios/:idUsuarios/formularios/:tipo………...44

# Introducción

Desde hace algunos años los problemas más importantes que se encuentran las universidades durante el primer año universitario son la tasa de abandono y el fracaso académico. Para evitar este fracaso, hay que centrarse en el estudio de los factores que determinan el éxito o fracaso de un estudiante, tales como: factores comportamentales (hábitos de estudio), factores afectivos (nivel de satisfacción), y factores motivacionales (internos y externos).

Por su parte, el estudiante al iniciar la universidad se encuentra con una serie de dificultades. En primer lugar, la elección de la titulación, para ello existen unas jornadas que facilitan los colegios el último año donde visitan distintas universidades y carreras, a modo de orientación. Para la elección de la carrera también es fundamental conocer la vocación y las aptitudes del estudiante; la vocación tiene carácter intrínseco que no puede evaluarse de la misma forma que las aptitudes, en cambio, las aptitudes deberían alinearse con la carrera que seleccionara el alumno para que obtuviera un mejor rendimiento. Para ello, pueden realizarse unos formularios estandarizados, que ofrecen unos resultados que sirven a modo de recomendación para elegir mejor una titulación.

Una vez que el estudiante conoce sus aptitudes, puede determinar qué asignaturas requieren de una mayor o menor concentración de estudio, y así poder realizar una mejor planificación de su tiempo. Esto es fundamental, dado que muchos estudiantes tienen que abordar toda la carga de trabajo que conllevan unos estudios universitarios, y una planificación que hasta ese momento de sus vidas no han tenido que hacer. Por último, no hay que olvidar que otra de las dificultades que se encuentra el estudiante es establecer relaciones entre compañeros, que le servirán en un futuro para facilitar el trabajo en equipo.

## Objetivos

Para abordar las dificultades que se encuentra el estudiante antes de comenzar su primer año de universidad y durante el desarrollo mismo (mencionadas en el apartado anterior), se va a crear un Servicio Web que asesora y acompaña al estudiante mediante recomendaciones durante ese período. Para ello se han establecido los siguientes objetivos:

1. El sistema será capaz de realizar una valoración de las aptitudes del estudiante, y de su concentración mediante el análisis de los resultados de unos formularios estandarizados, para realizar recomendaciones sobre la elección de la titulación.
2. El sistema será capaz de realizar una planificación de tiempos de estudio, mediante recomendaciones de los datos obtenidos anteriormente.
3. El sistema ofrecerá una parte de aplicación para operar con el mediante una interfaz
4. El sistema por parte de la API tendrá una interfaz en la que expondrá los recursos que ofrece, además esta interfaz permitirá probarlos individualmente

# Gestión del proyecto

## Modelo de ciclo de vida

En este caso la metodología escogida fue la metodología en cascada ya que es la que más se ajusta a este TFG, dado que los requerimientos son fijos y el trabajo avanza en forma lineal hacia el final[8].

La versión original fue presentada por Royce en 1970, aunque son más conocidos los trabajos realizados por Boehm en 1981, Sommerville en 1985y Sigwart y col. en 1990. Esta metodología se basa en la evolución del producto a través de una secuencia de fases de forma lineal mediante iteraciones del estado anterior.

Diagrama

Descripción generada automáticamente con confianza media

Ilustración 1. Metodología en Cascada

Esta metodología comienza con la especificación de los requerimientos por parte del cliente (comunicación con el cliente) y avanza a través de planificación, modelado, construcción y despliegue, para concluir con el mantenimiento del software.

1. **Comunicación**. En esta etapa el analista se reúne con el cliente escuchando sus necesidades y, tras estas reuniones, genera el SRD (*Documento de Especificación de Requisitos)* que contiene la especificación completa de lo que debe hacer el sistema sin detalles técnicos. Dicho documento debe estar consensuado con el cliente para delimitar el alcance del proyecto.
2. **Planeación o planificación**. En esta etapa se realiza una planificación de los recursos y se estiman los tiempos para el desarrollo de cada una de las etapas.
3. **Modelado**. En esta etapa se realiza un análisis de los requisitos que darán como resultado el diseño del sistema y del programa:
   1. **Diseño del sistema.** Se descompone y organiza el sistema en partes separadas, generando el SDD (*Descripción del Diseño del Software*), que contiene la descripción de la estructura del sistema y la funcionalidad de sus partes, así como la manera en que se combinan unas con otras.
   2. **Diseño del programa.** Se desarrollan los algoritmos necesarios para satisfacer los requerimientos del cliente, además del estudio necesario para saber qué herramientas son requeridas para la etapa de codificación.
4. **Construcción.** Se implementa el código del programa para que realice las funcionalidades detalladas en los algoritmos, y después se realizan un conjunto de pruebas y corrección de errores con el objetivo de revisar el cumplimiento de lo acordado con el cliente.
5. **Despliegue del software.** Se trata de la ejecución del sistema, donde el cliente revisa y valida si se han cubierto todas sus necesidades. Una vez revisadas, se realizan las correcciones oportunas para solucionar las necesidades no cubiertas por parte del cliente.

## Papeles desempeñados en el proyecto

Según la naturaleza del proyecto, nos encontramos 2 entidades, siendo éstas el tutor del trabajo fin de grado (TFG) y el estudiante.

### Roles del tutor

El tutor ha realizado dos roles. Por un lado, el rol de director del proyecto, ya que ha participado en la planificación y definición de objetivos; por otro lado, de analista de requisitos, ya que ayudó a establecer los requisitos de la aplicación.

### Roles del estudiante

El alumno ha ejercido cuatro roles. En primer lugar, el rol de cliente, puesto que propuso la idea de la aplicación. En segundo lugar, de analista de requisitos dado que estableció los requisitos de la aplicación. En tercer lugar, de desarrollador, ya que diseñó y escribió el código. Finalmente, de *tester*, ya que realizó las pruebas necesarias para validar el correcto funcionamiento de la aplicación.

## Planificación

En este apartado se muestran, mediante un diagrama de GANTT, los tiempos estimados que se dedican a las tareas, antes y durante el desarrollo del programa, para extraer una visión más amplia del recorrido.

Imagen que contiene Gráfico

Descripción generada automáticamente

Ilustración 2. Planificación estimada

Como se puede observar en la ilustración anterior[8], la parte que más tiempo ocupa del desarrollo del programa es la creación de las bases del proyecto, a saber: el desarrollo de la idea del proyecto, análisis del problema y los objetivos que se van a abordar en el proyecto. Esto se debe a la relevancia que tienen, ya que condicionarán el desarrollo de éste al ser dependiente uno del otro.

## Presupuesto

En este apartado hay que diferenciar entre dos tipos de costes, según la fase del desarrollo del programa en que nos encontremos: costes en la fase de desarrollo y costes en la fase de producción.

### Costes en la fase de desarrollo

1. Luz (kW/h). Teniendo en cuenta que durante los días hábiles de destinan dos horas/día al TFG, los días de fin de semana cuatro horas/día, y los días festivos no se incluyen (puesto que no se trabaja en el TFG). El precio medio de la luz actualmente en el mercado regulado es de 0,21846 euros/kWh. Por tanto, tras los cálculos, supone 73,4 euros/336 horas totales destinadas al desarrollo del programa[8].



Tabla 1. Coste de la luz por horas trabajadas

1. Equipo Acer Predator g3620 con i7-3770, 16GB RAM, 2TB. Tiene un precio en MediaMarkt de 849 euros IVA incluido.
2. Licencia Microsoft Windows 10 Home 64 Bits OEM. Tiene un precio en PC Componentes de 127,07 euros.
3. Licencias de Software (Sublimetext, Intellijide). Gratis al ser estudiante.

### Costes en la fase de producción

1. Amazon WebService:
   1. 50 TB/mes 🡪 0,77 euros por GB.
   2. t2.medium 🡪 0,93 euros por hora el API.
   3. Totales 🡪 670,37 euros/mes.
2. Salario de los empleados: desarrollador, dedicado al mantenimiento y experto técnico en casos puntuales (externo) 🡪 19.000 euros por el desarrollador y 2200 euros brutos/mes por el experto.

## Ejecución

Tabla

Descripción generada automáticamenteEn este apartado se muestran, mediante un diagrama de GANTT, los tiempos estimados que se dedican a las tareas una vez terminadas todas las etapas del proyecto.

Ilustración 2. Planificación realizada

En comparación del mostrado en el apartado 2.3, en la ilustración anterior se puede observar un aumento del tiempo en la extracción de los objetivos de las ideas del proyecto, en el análisis del dominio y en el estado del arte, esto se debe a la falta de bases en la rama de la psicología, necesarias para el proyecto, factor que se vio reflejado también en el diseño de los servicios.

# Estado del arte

En este capítulo se expondrá el contexto y definición de los servicios Web RESTful, las tecnologías utilizadas y una comparativa entre las aplicaciones ya existentes.

## ¿Qué son los Servicios Web y cómo funcionan?

Los Servicios Web son un medio estandarizado para permitir la interacción máquina a máquina a través de una red (cliente, servidor). Está diseñado de forma modular para realizar una serie de tareas concretas, estos pueden buscarse a través de la red e invocarse en consecuencia, proporcionando un servicio concreto al cliente que realizó la petición[4].

Diagrama

Descripción generada automáticamente

Ilustración . Servicio Web

En el diagrama anterior se puede ver el funcionamiento de un Servicio Web.

Primero, la aplicación del cliente realizaría una serie de peticiones al Servidor Web donde se encuentran alojados los servicios. Estas peticiones se realizan a través de lo que se conoce como llamadas a procedimientos remotos. Las llamadas a procedimientos remotos son métodos alojados en el Servicio Web correspondiente.

Por ejemplo, Amazon ofrece un servicio web que proporciona los precios de los productos vendidos en línea a través de amazon.com. La aplicación del cliente( la interfaz de usuario de la página web de Amazon) podría estar en un lenguaje completamente diferente al Servidor Web y aun así se pueden comunicar ya que son independientes.

Para diseñar un Servicio Web hay que tener en cuenta que el principal componente son los datos que se transfieren entre el cliente y el servidor, para transferirlos se hace uso de lenguajes de marcado como XML (Extensible markup language) ya que proporciona una plataforma común para que las aplicaciones desarrolladas en varios lenguajes de programación se comuniquen entre sí.

En la actualidad, existen dos estilos para la construcción de Servicios Web:

* Servicios basados en Arquitectura Orientada a Servicios (SOA, Servicios Web tradicionales, basados en protocolo WS-\*)
* Servicios basados en una Arquitectura Orientada a Recursos (Servicios Web RESTful).

### Servicios Web tradicionales

Los Servicios Web tradicionales que se basan en la pila de protocolos de WS-\*,

Utilizan SOAP*(Simple Object Access Protocol)*,que se basa en la transferencia de datos XML como mensajes SOAP[4].

Diagrama

Descripción generada automáticamente

Ilustración . Servicio Web Tradicional

Tal como muestra la imagen, el mensaje SOAP necesita un elemento raíz (<Envelope>, siendo este el primer elemento del documento XML) y dentro de este se divide el contenido en dos partes, por un lado, la cabecera, que indica a

qué cliente debe enviarse y, por otro lado, el cuerpo contendrá el mensaje propiamente dicho.

### Servicios RESTful (estilo arquitectónico REST)

El estilo arquitectónico REST (*Representational State Transfer*) lo definió Roy Fielding como “una arquitectura de software para sistemas hipermedia distribuidos”. REST se fundamenta en el sistema cliente - servidor, en el que el cliente ingresa en los servicios a través de un puerto (socket), usando el protocolo HTTP como fuente de comunicación de los mensajes.

Diagrama

Descripción generada automáticamente

Ilustración 5. Diagrama de una estructura REST

Las restricciones que impone el estilo arquitectónico REST para poder seguirlo son:

1. **Recurso: Identificación, estado:** Los recursos son la abstracción principal en la que se basa REST, los cuales deben ser únicos e identificables para ello, en el caso de la web, se utilizan las URIs. Las URIs se encargan de exponer al recurso, es decir , actúan como si fueran el nombre y la dirección del recurso, para que sea accesible e identificable entre los distintos recursos. Las URIs al actuar como nombre, no pueden incluir acciones ya que sería filtrar la información del recurso que lo expones en vez de identificarlo claramente .
2. **Representación de un recurso:**  Son los datos y metadatos del recurso que poseen toda la información de este. En el caso de la Web, se pueden solicitar en formato HTML (utilizado para el navegador y por ende seria consumido por el usuario) o en JSON[**1.1]** (orientado a que se ha consumido por máquinas).
3. **Hipermedia:** Es una de las restricciones más importante del estilo arquitectónico de REST es *HATEOAS*(*Hypermedia as the Engine of the Application State*), La cual dice que mediante el uso de un hiperenlace (URI), se obtenga la representación de un recurso, que, a su vez, contiene URIs a los diferentes recursos con los que se puede interactuar.
4. **Comunicación:** REST establece una comunicación entre cliente-servidor, dicha comunicación es síncrona, donde el cliente es el encargado de iniciar la comunicación mediante solicitudes (que contienen toda la información necesaria para que el servidor pueda procesarla) a los recursos del servidor, procesará cada solicitud y le devolverá al cliente la respuesta por cada una de ellas.
5. **Interfaz homogénea:** Todos los recursos deben seguir un estándar para que sea más fácil su manipulación, por ello, en el caso de la Web, se utilizan los verbos que proporciona el protocolo HTTP, donde cada uno de ellos nos permiten realizar diferentes cambios en el estado del recurso.

Para una representación del recurso se utiliza el verbo (GET), para la creación de este se usa (POST), para actualizar la información del recurso se usa (PUT), para borrarle se usa (DELETE).

Para resumir, un servicio RESTful es un estilo de arquitectura de software que se utiliza para diseñar servicios web basado en un conjunto de principios y restricciones que permiten a los sistemas comunicarse entre sí a través de HTTP de manera eficiente, escalable y sin estado[5].

### Modelo de madurez de Richardson

Para poder evaluar y clasificar la implementación de los servicios RESTful Leonard Richardson en su artículo titulado "Maturity Model for REST Web Services" ofrece una diferenciación en niveles[6].

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Ilustración 6. Niveles de madurez de los Servicios Web REST

#### **Nivel 0**

Este nivel corresponde a los Servicios Web tradicionales, siendo el punto de partida en el uso de HTTP como sistema de transporte de las interacciones de forma remota, pero sin usar ningún tipo de mecanismo Web. Fundamentalmente, aquí se usa HTTP como un canal para trasmitir las interacciones entre los propios mecanismos, normalmente basados en RIP (*Remote Procedure Invocation*).

#### **Nivel 1**

Es la primera etapa para llegar a “the Glory of Rest” y tiene como objetivo la identificación de los recursos a través de una URI, permitiendo lanzar peticiones a "recursos” (en REST, se llama así la información con la que se interactúa, sin importar en el formato en la que esté) individuales. En vez de usar un único punto de entrada, llega a secciones o documentos del sitio Web usando las distintas URIs.

#### **Nivel 2**

En este nivel los servicios utilizan todos los métodos que ofrece HTTP, siguiendo de forma rigurosa el estándar creado por los desarrolladores REST donde se acordó: GET (accede a los datos de un recurso), POST (creando el recurso), PUT (modifica un recurso) y DELETE (elimina un recurso).

Además, están los códigos de estado para poder saber la situación de la solución, y los tipos de contenidos que especifican el formato o formatos que sigue el recurso.

#### **Nivel 3**

Es el último nivel, donde se habla del término “*HATEOAS*”, según el cual, tras al realizar una petición, la misma respuesta nos ofrece la información necesaria para comprender cómo utilizar el recurso. Para poder llegar a ese punto es necesario que los enlaces de los recursos presenten un “tipado” que le sea fácil de entender al usuario, cuya la respuesta ofrece informacion adicional como enlaces a otros recursos ampliando las interacciones con estos.

# Análisis

Investigación, análisis y requerimientos psicológicos del problema para la creación de la API.

## Análisis de dominio

La etapa universitaria es una de las experiencias más enriquecedoras de la vida de una persona, no sólo a nivel de formación en vista a un futuro laboral, sino también de crecimiento personal (madurez, independencia, etcétera). Muchos alumnos ingresan el primer año, pero su número se reduce considerablemente en el segundo año de carrera. Esto se debe al abandono universitario tras el primer año cursado.

Escala de tiempo

Descripción generada automáticamente

Ilustración 7. Población de matriculados en universidades.

Según la ilustración anterior, la tasa neta de escolarización en la educación universitaria sobre la población comprendida entre 18 y 24 años, se puede ver un interés creciente en el estudio de carreras universitarias, sin embargo, si nos fijamos en el año escolar 2016-17 se produce un leve descenso y estancamiento en el interés de los estudiantes, a pesar de ser más numerosa la franja de edad. Este dato se puede relacionar con la tabla que se mostrará a continuación

Tabla

Descripción generada automáticamente

Tabla 2. Tasas de abandono en el primer año universitario.

Como se puede observar, tanto el porcentaje de abandono como el porcentaje de cambio de estudios sigue una progresión creciente. Con relación al punto anterior, ese descenso del interés universitario se puede provocar por el aumento de la frustración o desinterés de esta, dando lugar a tan altas tasas de abandono o de cambio de estudios.

Tal y cómo se muestra en las dos tablas anteriores, la tasa de abandono en el primer año es un problema real. Esto se debe a muchos factores, como la falta de motivación, de tiempo, de planificación, etcétera. Del conjunto de causas principales este proyecto se centrará en el caso donde el estudiante no reúne las aptitudes, el grado de concentración, la atención/retención e impulsividad necesarios para lograr obtener la titulación, ya que generan una frustración o desmotivación que le inducen a tomar la decisión de abandonar los estudios superiores, lo que podría solventarse con una buena elección académica antes de comenzar el primer año ofreciendo a los estudiantes la posibilidad de realizar una serie de test estandarizados sobre sus aptitudes, motivaciones, planificación, concentración, etcétera, que le sirvan como recomendaciones para elegir mejor la titulación que deberían estudiar.

Hay muchos tipos de formularios utilizados en el estudio psicológico entorno al estudiante, pero dado el alcance del proyecto, solo se realizarán dos tipos de formularios estandarizados: test de aptitudes y test de concentración; que serán informatizados.

El “*Test de Orientación Vocacional CHASIDE*” de Holland Ríase es un formulario muy utilizado para evaluar las aptitudes que se basa en el psicoanálisis vocacional, y permite tomar una decisión según las aptitudes y los intereses del estudiante. Se trata de contestar a preguntas sencillas con Sí/No, donde a las respuestas afirmativas se le asigna 1 punto y las negativas 0 puntos, para después contabilizar todos los puntos mediante la tabla de valores que se presenta a continuación[1].

Imagen que contiene gabinete, reloj, diferente, pantalla

Descripción generada automáticamente

Tabla 3. Evaluación CHASIDE

El “*Test de Toulouse”* de E. Toulouse y H. Piéron es un formulario muy utilizado para evaluar las aptitudes perceptivas y atencionales. Consiste en localizar una serie de figuras en un conjunto extenso de figuras similares, con el objetivo de medir la cantidad de aciertos, errores y omisiones. Una vez recogidos los datos, se pueden obtener:

1. El Índice Global de Atención y Percepción (IGAP), constituye una medida de la capacidad perceptiva y atencional de los evaluados.
2. El Cociente de Concentración (CC), mide la capacidad de concentración que tiene el usuario.
3. El Índice de Control de la Impulsividad (ICI), informa sobre el nivel de impulsividad que tiene el usuario a encontrar las figuras.

Para la elaboración del proyecto se analizarán tanto los dos índices anteriores como el cociente de concentración[2]:

**IGAP= ACIERTOS – (ERRORES + OMISIONES)**

**CC = A – E / A + O**

**ICI = ACIERTOS – ERRORES / RESPUESTAS X 100**

Donde: A es acierto, E es error y O es omisión.

Patrón de fondo

Descripción generada automáticamente

Ilustración 13. Test de Toulouse

## Especificación de requisitos

En este apartado se detallarán los diferentes tipos de requisitos que cubrirá la aplicación, que son: funcionalidades del sistema, rendimiento, capacidad, seguridad, interoperabilidad con otros sistemas, protección de datos, requisitos sobre entorno tecnológico y de comunicaciones, requisitos funciones y no funcionales:

* **Funcionalidades del sistema**

1. El sistema permitirá registrarse mediante un usuario y contraseña.
2. Debe medir las aptitudes mediante un formulario estandarizado.
3. Debe medir la concentración con un formulario estandarizado.
4. Establecerá relaciones entre los resultados de los formularios estandarizados (test de aptitudes, de concentración, etcétera) para dar consejos en la planificación.
5. Debe tener un servicio donde se muestren las recomendaciones acerca de las elecciones del estudiante en cuanto a los estudios.
6. Debe tener un servicio para mostrar las materias cursadas el primer año.
7. Debe implementarse como un servicio web
8. Los formularios deben estar estandarizados y con una base probada para aumentar su probabilidad de éxito.

* **Capacidad y eficiencia**

1. Debe soportar por defecto como numero de 25, 125, 500 peticiones simultáneas por 1, 60, 360 segundos respectivamente con un tiempo de respuesta máximo de 4 segundos.

* **Interoperabilidad con otros sistemas**
  1. Debe ser accesible desde cualquier dispositivo (tablets, móviles, otras aplicaciones, etcétera).
  2. Debe ser ajena a como estén formados los sistemas que acceden a dichos recursos.
  3. Debe usar JSON[**1.1]** como formato para enviar y transcribir los datos.
* **Protección de datos**
  1. Deben estar las id de los usuarios en formato Guid.
  2. Deberán trasmitirse los datos a aquellas personas autorizadas.
  3. Deberán guardarse las contraseñas cifradas.
* **Requisitos sobre entorno tecnológico y de comunicaciones**

1. Debe ser implementado como Servicio Web.
2. Debe utilizar los métodos HTTP para interactuar con los demás recursos.
3. Debe haber una interfaz operable por el usuario.

## Análisis de seguridad

En este análisis se buscará cubrir lo posible tratándose en el contexto que se realiza(plazo de tiempo, recursos y alcance del trabajo). Para ello primero se definirá las dimensiones que abarcará:

* **Integridad**:   
  La facilidad que un tercero no autorizado corrompa la información. LA forma que el aplicativo tenga integridad se usa el Login previo para que en todo momento el usuario se encuentra identificado durante la sesión
* **Trazabilidad**:

Capacidad de descubrir qué persona ha accedido a un sistema o ha corrompido la información. Mediante el uso de los logs, se puede saber en todo momento quién es que hizo los cambios y cuáles fueron sus cambios

* **Autenticidad**:

Ser capaz de saber si la información no fuera la producida en el origen, es decir, que se pueda suplantar. Para saber el origen se pide en todo momento la autorización de acceso (Token Bearer[1.2]), que es proporcionada en el momento que entra en la aplicación

# Diseño

Para desarrollar la lógica de la aplicación, primero se ha comenzado con un diseño de los Servicios RESTful que en los siguientes subcapítulos se abordara más en detalle.

Posteriormente se ha realizado el modelo de la base de datos de manera que recoja los datos necesarios para el funcionamiento de la aplicación.

Y por último se ha diseñado una interfaz amigable para la interacción con el usuario y la máquina.

## Arquitectura del Sistema

## Para poder cubrir el análisis del capítulo 4, se ha elegido una arquitectura que se divide en dos partes, la primera parte es el Backend donde se desarrolla la parte funcional de la API y la base de datos, y la segunda parte un Frontend donde se encuentra una interfaz amigable que consume los servicios de la API de forma amigable.

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

Ilustración . Arquitectura del proyecto

Como se puede ver en la Ilustración 7. Las partes estarán albergadas en dockers[1.3] individuales para posibilitar la instalación correcta en los diferentes ordenadores y haciendo una clara distinción de las partes en 3 grupos.

* Interfaz

La interfaz está compuesta por una aplicación web, que es con la que interactúa el usuario, esta se hará con el protocolo https

* Lógica

Está compuesta por el servidor web(API RESTful) que almacena los servicios que consumirá la interfaz mediante las peticiones, que a su vez v generan logs, la comunicación entre la interfaz y la API estará basada en el envío de mensajes JSON[**1.1]**, en los que en la cabecera se encontrará el token beare.

* Almacenamiento

Respecto al almacenamiento, será realizado en una base de dados SQLServer que proporcionará la persistencia de los datos generados por el uso de la interfaz y de la lógica de la API.

## Diseño de subsistema backend

Esta parte de la arquitectura abarca tanto el servidor web como la base de datos, albergando la lógica de la aplicación y el almacenamiento de los datos.

La API RESTful esta desarrollada en .Net Core, esta recibirá las peticiones HTTP sobre los diferentes recursos que se encuentran publicados e interactuará con la base de datos para la gestión de la información.

### Diseño de los Srevicios RESTful

Para poder crear los recursos primero se han definido los recursos y los atributos, obtenidos tras el análisis del realizado en el **Análisis de dominio** se han estructurado las URIs para poder exponer cada recurso. Todo ello queda representado en el anexo

#### **Definición de los recursos**

Los recursos expuestos a continuación tienen la función de dotar toda la lógica necesaria a la aplicación siendo estos:

* + **Usuario**: permite eliminar a un usuario en concreto, acceder a su información y actualizar la contraseña u otros datos
  + **Usuarios**: permite crear un usuario o listar a todos los usuarios
  + **Formulario**: permite obtener todas las preguntas del formulario, actualizar, crear las respuestas del formulario de un usuario en concreto y calcular los resultados obtenidos del formulario y usuario en concreto.
  + **Formularios**: permite obtener todas las respuestas de un usuario en concreto.
  + Asignatura: permite obtener la información, borrar, actualizar y asignar una asignatura de un usuario.
  + **Asignaturas**: permite añadir una asignatura, obtener todas asignaturas o si se proporciona el usuario, obtener todas las que le corresponden.

#### **Atributos de los recursos**

En este apartado se detallará a cada recurso definido anteriormente, explicando sus atributos y características más importantes.

##### Usuario

La aplicación permite crear una cuenta a los alumnos de que se encuentren en primer año de carrera, interesados en obtener recomendaciones de estudio.

Los atributos que caracterizan al usuario son:

* Nombre: hace referencia al nombre del alumno
* Pass: contraseña del alumno
* Intereses de las áreas
  + Administrativas\_Contables\_Int
  + Humanisticas\_Sociales\_Int
  + Artisticas\_Int
  + Medicina\_CsSalud\_Int
  + Ingenieria\_Computacion\_Int
  + DefensaSeguridad\_Int
* Aptitudes de las áreas
  + Administrativas\_Contables\_Apt
  + Humanisticas\_Sociales\_Apt
  + Artisticas\_Apt
  + Medicina\_CsSalud\_Apt
  + Ingenieria\_Computacion\_Apt
  + DefensaSeguridad\_Apt
* CC: Cociente de Concentración
* IGAP: Índice Global de Atención y Percepción
* ICI: Índice de Control de la Impulsividad

##### Formulario

* Conjunto de preguntas
* Conjunto de respuestas

##### Asignatura

Es el elemento que del cual se realizaran las recomendaciones según el tiempo de estudio, la nota y los resultados de los formularios de CHASIDE y Toulouse.

Los atributos que caracterizan a la asignatura son:

* Nombre: nombre de la asignatura
  + Área: área de conocimiento de la asignatura
  + Introducidos por el usuario
  + Nota: puntuación obtenida tras el examen
  + TiempoEstudio: horas semanales dedicadas al estudio de la asignatura.
* Generados tras los datos introducidos y los resultados de los formularios
* TiempoRecomendado: horas semanales recomendadas al estudio de la asignatura.
* Riesgo: dificultad que supone aprobar la asignatura

#### **Patrones de las URIs**

Gracias a la definición de las URIs se pueden mandar peticiones HTTP determinadas por los métodos y los recursos, un ejemplo sería el caso del recurso formulario

**/api/formularios/{idFormulario} ->** Solo acepta el método HTTP GET. Obteniendo todas las preguntas del formulario pedido

**/api/usuario/{idUsuario}formularios ->** Solo acepta el método HTTP GET. Obteniendo todas las respuestas de todos los formularios del usuario.

**/api/usuario/{idUsuario}formularios/{idFormulario}->**Los métodos HTTP que acepta son GET, PUT, DELETE. Actuando sobre un formulario determinado de un usuario determinado

**~~Tablas de recursos~~**

### Diseño de la base de datos

Interfaz de usuario gráfica

Descripción generada automáticamentePara dar soporte a los datos recibidos de la API, se ha procedido a la creación de tablas expuestas a continuación a través del diagrama entidad-relación

Ilustración . Diagrama entidad relación

Para entender el diagrama de la Ilustración 8 conviene saber el significado de los elementos y las relaciones entre las tablas que aparecen.

* La llave significa clave primaria
* En las relaciones, la parte que tiene la llave significa que de ahí proviene la clave primaria.

En este caso las cuatro tablas principales son PreguntaFormularios, Usuarios, Asignaturas y AreasConocimientos. Las otras tablas son creadas por la normalización de la base de datos.

No se almacenan las URIs de los recursos, ya que son obtenidas de forma dinámica y transmitidas en la API. Solo el caso de PreguntaFormularios almacena la URL de las imágenes subidas a un repositorio online, evitando así incrustarlas en la base de datos.

Las funciones de las tablas son las siguientes:

* Usuarios: Almacena toda la información necesaria para la identificación y para guardar la información del resultado de los formularios.
* PreguntasFormularios: Almacena las preguntas de los formularios CHASIDE y Tolulouse diferenciados por el tipo.
* RespuestaFormulario: Asocia las preguntas al usuario, además de almacenar las respuestas que proporcionó este último.
* Asignaturas: Almacena el nombre de la asignatura.
* AsignaturasUsuarios: relaciona al usuario con la asignatura, además de almacenar los tiempos de estudio, la nota y el riesgo.
* AreasConocimiento: Almacena las áreas de conocimiento de cada asignatura.
* AsignaturasAreas:Relaciona cada asignatura con su área de conocimiento .

## Diseño del subsistema frontend

Es el responsable de mostrar los datos de manera sencilla y fácil para la interacción con el usuario, de tal forma que el alumno pueda registrarse , añadirse asignaturas, realizar los formularios y así obtener recomendaciones.

Todo esto es posible gracias a la aplicación desarrollada en .Net Core, que en su interior permite el uso en sus vistas de los lenguajes de HTML, dotándole de la estructura visual, CSS que le proporciona la apariencia al documento HTML y el JavaScript que le aporta el dinamismo y es el que llama en muchos casos a los métodos que le proporciona Core.

Por último, en este proyecto se ha hecho uso de una plantilla de software libre llamada **sb-admin2**[11]**.** Para dar un aspecto más depurado y sencillo de entender.

# Implementación

En este apartado se recogerá todos los datos necesarios para la implementación de la API y de la interfaz gráfica, además se detallarán los componentes, estructura y funciones del backend y del Frontend.

## Implementación de Backend

Para la implementación de los Servicios RESTful se ha utilizado .Net Core. La estructura de la aplicación se encuentra dividida en 3:

* Controllers: Clases encargadas de atender las diferentes peticiones HTTP y donde se encuentran las rutas de los recursos expuestos.
* Entities: Son las entidades del modelo de la aplicación, cuya función es de obtener y generar en formato JSON[1.1] los datos necesarios de las peticiones.
* Services: Son las clases destinadas a aislar las llamadas a la BBDD

A la vez que se realizaba la implementación de los servicios, se realizaba la implementación de la BD en SQL Server para dar persistencia a los datos procedentes de la API.

### Estructura e implementación de la lógica de negocio con .Net Core

A continuación, se definirá la estructura e implementación realizada en .Net Core. Para poder entender mejor la estructura, durante todo este subcapítulo se hará uso como ejemplo del recurso Asignatura.

#### Controller

Dentro de las clases *controllers* se encargan de establecer las URIs para cada servicio siguiendo los patrones de las URIs [5.2.1.3] exponiéndolos y pudiendo así hacer la lógica tras cada petición, a su vez, también llama a los Entities correspondientes para tratar los datos recibidos, obtener un resultado y mandar una respuesta acorde al resultado.

#### Entities

En el paquete *Model* están almacenadas todas las clases que sirven de recursos de la API. Utilizadas tanto para recibir los datos de la llamada, como para la concordancia de la base de datos o como respuesta ante la petición HTTP.

La nomenclatura seguida es dividirlos en paquetes por recurso y dentro de estos, las diferentes subclases de la principal. Un ejemplo sería en el paquete Asignatura, se encuentra la clase Asignatura y la clase AsignaturaGet, donde en el primer caso sería la clase que responde al esquema de la base de datos, y en el segundo caso, hace correspondencia a la respuesta GET de un caso concreto del recurso asignatura.

#### Services

Dentro del paquete BBDD se encuentran las clases que realizan la conexión a la base de datos. Para realizarlo estas clases extienden del DbContext que lleva asociado la cadena de conexión necesaria para conectar con la tabla de la base de datos correspondiente.

Una vez conectado correctamente, el objeto creado a partir de esta clase realizará el mismo comportamiento que haría la IDE de SQL Server, pudiendo realizar querys sobre las mismas según al método que llames.

Para finalizar se mostrará a continuación un fragmento del código del recurso Asignatura desde el controlador, en el que se referenciará cada elemento expuesto en este subcapítulo

[HttpGet]

[Route("usuarios/{idUsuario}/asignaturas/{idAsignatura}")]

public async Task<ActionResult> GetAsignaturasUsuario(Guid idUsuario,Guid idAsignatura)

{

AsignaturaUsuario nota = await asignaturaUsuarioBBDD.AsignaturasUsuarios.FindAsync(idAsignatura, idUsuario);

AsignaturaUsuarioGet asignaturaUsuarioGet = new() {

UrlAsignatura = new Uri(Request.GetEncodedUrl().Split("usuarios")[0] + "asignaturas/" + idAsignatura),

TiempoEstudio=nota.TiempoEstudio,

TiempoRecomendado=nota.TiempoRecomendado,

Riesgo=nota.Riesgo,

Nota=nota.Nota,

};

return Ok(asignaturaUsuarioGet);

}

En este caso se puede observar que el método utilizado para acceder a esta lógica es mediante el uso de GET, usando la URI indicada en el apartado “Route”.

Para poder realizar la lógica en este caso se sirve de una *Entitie* **AsignaturaUsuario** que es el objeto obtenido tras la traducción de los datos que genera el *Service* **asignaturaUsuarioBBDD** , a continuación, se crea un objeto de la clase **AsignaturaUsuarioGet** el cual es la *Entitie* requerida en la respuesta del servicio.

**~~Para finalizar se ha realizado un diagrama UML ubicado en el anexo para ofrecer una visión global de las clases de la API RESTful~~**

### Estructura e implementación de la BBDD

Gracias al diagrama E/R realizado en el diseño de la BD en el punto **5.2.2**, se ha realizado su implementación en el servidor SQL gracias a la interfaz gráfica SQL Server Management Studio Management Studio 19.

Por último, se han creado unos scripts sql que permiten inicializar la base de datos, con los datos necesarios para el correcto funcionamiento del proyecto. La ubicación de dichos ficheros se encuentra en el directorio del api, dentro de la carpeta SQL.

## Implementación de Frontend

Para el desarrollo de este proyecto se ha decidido crear una aplicación web en .Net Core con el objetivo de crear una interfaz amigable para el usuario, con la que pueda trabajar e interactuar con la API. Para el desarrollo de la interfaz, se realizo bajo la estructura MVC:

* Vistas: Las vistas están en formato .chtml con lo que permiten el uso de JS, HTML, CSS y Core dentro de las mismas, facilitando el uso de plantillas HTML.
* Modelo: La interfaz hace uso de las mismas clases de tipo *Entities* que se encuentran en la API, ya que serán usadas para mandar la información en un formato estandarizado facilitando así la serealizacion y deserialización de los archivos JSON intercambiados entre la API y la aplicación.
* Controlador: Este es el encargado de atender a las acciones o métodos que requiere la vista y devolver un resultado tanto en forma de petición a la API, como un cambio de página o un evento en la vista.

## Referencia al repositorio de software

El repositorio donde se encuentra el trabajo de fin de grado es:

<https://github.com/davidRecio/tfg_Contenedor>

## Manuales

En este proyecto tenemos dos manuales diferentes, por un lado, el manual de instalación del proyecto y por otro lado el manual de uso de la aplicación, este último se puede encontrar dentro de la misma aplicación en el apartado de manual de ayuda.

En el manual de instalación se detalla los pasos que se ha de seguir para lanzar las instalaciones de los Dockers hasta el parado de estos y los requisitos necesarios para la correcta instalación.

Respecto al manual de uso, estará acompañado de imágenes para su facilitar su comprensión, además de un apartado de pruebas extraídas del capítulo 7 que veremos más adelante.

Ambos documentos se pueden encontrar también en la carpeta ./Doc/Manuales dentro del directorio raíz del proyecto.

# Pruebas y validación

Tras la realización de la parte final del proyecto se han realizado pruebas individuales haciendo uso de swaguer sobre cada recurso involucrado en las peticiones que realiza la aplicación a la API, permitiendo a su vez, comprobar la correcta conexión a la base de datos.

Una vez comprobado lo anterior, se ha procedido a realizar la validación de la aplicación mediante se realizan todos los pasos relacionados al alumno, desde su registro hasta el cierre de sesión, pudiendo realizar los formularios, asociarle asignaturas y obteniendo recomendaciones de estas durante el camino.

Inicialmente el alumno se registra en la aplicación, una vez registrado podrá acceder a una interfaz que le proporcionará acceso a un Manual del Usuario y al acceso a los diferentes formularios, tras completarlos según las instrucciones que en ellos aparecen, el alumno podrá acceder a la pantalla de asignaturas, en la cual podrá signarse una serie de asignaturas y en ellas guardar las calificaciones y el tiempo dedicado a cada asignatura, en el momento que lo haga estas generaran una recomendación acorde a los datos recogidos en la aplicación, por último, si el alumno quiere ver el resultado de los formularios podrá acceder al perfil del usuario. Tras terminar de usar la aplicación, podrá cerrar sesión saliendo así de la misma.

**Pasos para realizar la validación del correcto funcionamiento descrito en el anterior escenario:**

1. El alumno realiza el registro en la aplicación haciendo uso de la interfaz insertando su usuario y contraseña.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

En el momento que el alumno pulsa sobre el botón la aplicación realiza una petición POST al recurso /api/usuarios enviándole el JSON que se muestra a continuación.

Texto

Descripción generada automáticamente

Desembocando en el guardado del usuario en la base de datos

1. Una vez realizado el registro, el usuario podrá acceder a la pagina de inicio.

# Conclusiones y líneas futuras

# Bibliografía

1. Martha Báez. Holland Ríase. Test de orientación vocacional CHASIDE (2007). Academia Edu. Consultado el 3 de Mayo de 2023. <https://www.academia.edu/10367175/Test_De_Orientaci%C3%B3n_Vocacional_Chaside>
2. E. Toulouse y H. Piéron. TEA Ediciones (1978, 2004, 2013). TP-R. Toulouse-Piéron-Revisado, prueba perceptiva y de atención. Consultado el 3 de Mayo de 2023. <https://web.teaediciones.com/Ejemplos/Extracto_libro_TP-R.pdf>
3. DR. Winston W. Royce (1970) Managin the development of large software systems. Consultado el 5 de Mayo de 2023. <https://www.praxisframework.org/files/royce1970.pdf>
4. Guru99. What are Web Services? Architecture, Types, Example [https://www.guru99.com/Web-service-architecture.html](https://www.guru99.com/Web-service-architecture.html%20) (Último acceso: mayo 2023).
5. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Tesis de doctorado. University of California, Irvine. Disponible en: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
6. Fowler, M. (2010, March 18). Richardson Maturity Model.<https://martinfowler.com/articles/richardsonMaturityModel.html>. (último acceso mayo 2023)
7. Instinto Programado.com ¿Qué es el modelo de madurez de Richardson? <https://www.instintoprogramador.com.mx/2021/01/que-es-el-modelo-de-madurez-de.html> (último accesomayo 2023)

Roger S. Pressman, Ph.D.University of Connecticut. Ingeniería del software, Un enfoque práctico , 33–35. Consultado el 19 de Mayo de 2023. <http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF>

1. json.org. Introducing JSON <https://www.json.org/json-en.html>(Último acceso: mayo 2023).
2. Auth0 docs. JSON Web Tokens. [https://auth0.com/docs/secure/tokens/json-web-tokens#benefits-of-jwts](https://auth0.com/docs/secure/tokens/json-web-tokens%23benefits-of-jwts) (Último acceso: mayo 2023).
3. Docker. Docker makes development efficient and predictable. <https://www.docker.com/> (Último acceso: mayo 2023).
4. Startbootstrap. SB Admin 2. <https://startbootstrap.com/theme/sb-admin-2>. (Último acceso: mayo 2023).

# Anexos

## Glosario de definiciones

## JSON

Un archivo JSON (JavaScript Object Notation, Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos y que además, es fácil de leer y escribir para los humanos[8].

Un archivo JSON consiste en una colección de pares clave-valor, donde la clave es una cadena de texto y el valor puede ser de diferentes tipos de datos, como una cadena, un número, un booleano, un objeto JSON anidado o una matriz de valores. Estos pares clave-valor se separan por comas y se encierran entre llaves {} para representar un objeto JSON, o entre corchetes [] para representar una matriz JSON.

Texto

Descripción generada automáticamente

## Token Bearer

JSON web token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autocontenida de transmitir información de forma segura entre las partes como un objeto JSON. Gracias a su tamaño se puede enviar a través de una URL, un parámetro POST o dentro de una cabecera HTTP, conteniendo toda la información necesaria sobre una entidad[9].

## Docker

Docker es una plataforma que permite empaquetar, distribuir y ejecutar las aplicaciones en contenedores, aislándolos en un entorno adecuado e independiente del usuario final haciéndolo consistente y portátil[10].

## Definición de los recursos de la API

|  |  |  |  |
| --- | --- | --- | --- |
| **Recurso** | **URI** | **Funcionalidad** | **Métodos soportados** |
| Usuario | /usuarios | Representa el conjunto de Usuarios | POST, GET |
| Notas | /notas | Representa el conjunto de notas | GET, DELETE |
| Usuarios/id | /usuarios/:id | Representa un usuario determinado | GET, DELETE, PATCH |
| Usuarios/idUsuarios/notas/id | /usuarios/:idUsuarios/notas/:id | Representa una nota determinada de un usuario determinado | GET, DELETE, PUT |
| Usuarios/idUsuarios/notas | /usuarios/:idUsuarios/notas | Representa el conjunto de notas de un usuario determinado | POST,GET, DELETE |
| Usuarios/idUsuarios/formularios | /usuarios/:idUsuarios/formularios | Representa un conjunto de formularios de un usuario determinado | DELETE |
| Usuarios/idUsuarios/formularios | Usuarios/idUsuarios/formularios /:id | Representa un formulario de un usuario determinado | GET , PUT |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | URI | Utilidad | Representación | Código Respuesta |
| POST | /usuarios | Añade los usuarios | JSON | 200-OK |
| 400-Bad Request |
| 500-Internal Server Error |
| GET | /usuarios | Obtiene todos los usuarios | JSON | 200-OK |
| 500-Internal Server Error |
| PUT | /usuarios | - | - | 404- Not Found |
| DELETE | /usuarios | - | - | 404-Not Found |
| PATCH | /usuarios | - | - | 404-Not Found |

## Métodos HTTP de los recursos

Recurso /notas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | URI | Utilidad | Representación | Código Respuesta |
| POST | /notas | - | - | 404-Not Found |
| GET | /notas | Obtiene todas las notas | JSON | 200-OK |
| 500-Internal Server Error |
| PUT | /notas | - | - | 404-Not Found |
| DELETE | /notas | Borra todas las notas | JSON | 200-OK |
| 400-Bad Request |
| 500-Internal Server Error |
| PATCH | /notas | - | - | 404-Not Found |

Recurso /usuarios/idUsuarios/notas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Método | URI | Utilidad | Representación | Código Respuesta |
| POST | /usuarios/:idUsuarios/notas | Iañade una nueva nota a un usuario | JSON | 200-OK |
| 400-Bad Request |
| 500-Internal Server Error |
| GET | /usuarios/:idUsuarios/notas | Obtiene todas las notas de un usuario en concreto | JSON | 200-OK |
| 500-Internal Server Error |
| PUT | /usuarios/:idUsuarios/notas | - | - | 404-Not Found |
| DELETE | /usuarios/:idUsuarios/notas | Borra todas las notas de un usuario en concreto | - | 200-OK |
| 500-Internal Server Error |
| PATCH | /usuarios/:idUsuarios/notas | - | - | 404-Not Found |