Escuela Politécnica Superior de Linares

Grado en Ingeniería de Telemática

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

DESARROLLO DE UNA APLICACIÓN WEB PARA LA GESTIÓN DE INVENTARIO DE LOS LABORATORIOS DOCENTES Y DE LA GESTIÓN DEL PRÉSTAMO DE EQUIPOS.

Alumno: Ramón Elbal Ruiz

Tutor: Juan Carlos Cuevas Martínez

Depto.: Ingeniería de Telecomunicación

Junio, 2023

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

Trabajo Fin de Grado

DESARROLLO DE UNA APLICACIÓN WEB PARA LA GESTIÓN DE INVENTARIO DE LOS LABORATORIOS DOCENTES Y DE LA GESTIÓN DEL PRÉSTAMO DE EQUIPOS.

Alumno: Ramón Elbal Ruiz

Tutor: Juan Carlos Cuevas Martínez

Depto.: Ingeniería de Telecomunicación

ELBAL RUIZ RAMON - 26509331Z

Firmado digitalmente por ELBAL RUIZ RAMON - 26509331Z
Fecha: 2023.06.20 14:12:14 +02'00'

Firma del autor

CUEVAS MARTINEZ JUAN CARLOS - 26038239P

Firmado digitalmente por CUEVAS MARTINEZ JUAN CARLOS - 26038239P
Fecha: 2023.06.20 16:20:54 +02'00'

Firma del tutor

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En el siguiente punto se detalla el contexto y desarrollo del trabajo realizado, junto a la estructura de éste.

1.1 Justificación, contexto y descripción del TFG

El desarrollo de aplicaciones web es una destreza que, incluso hoy en día, es demandada en múltiples sectores. La prevalencia de este tipo de aplicaciones se debe en gran medida a su facilidad de acceso desde cualquier dispositivo electrónico con capacidad de conectarse a Internet mediante un navegador. Dichas aplicaciones pueden diseñarse para una diversa gama de entornos de trabajo y utilidades.

Otra destreza cada vez más demandada es el manejo eficiente de bases de datos, que cada vez contienen más y más datos y requieren actualizarse con frecuencia. Para lo cual se suelen desarrollar interfaces que permitan al usuario acceder de manera intuitiva al contenido de dichas bases de datos.

El presente trabajo fin de grado (TFG) tiene como objetivo el desarrollo de una aplicación web que permita realizar el control de inventario de los equipos disponibles en los laboratorios docentes de la universidad. Además, incluirá la posibilidad de realizar la trazabilidad del préstamo, por un tiempo definido, de cualquiera de estos equipos, ya sea a profesorado o estudiantes.

La aplicación tendrá una vista de administrador para la gestión de usuarios, inventario y control de los préstamos, y una vista para los usuarios que les permita ver los equipos que tienen a préstamo y el plazo, así como la posibilidad de solicitar la renovación del equipo.

La aplicación deberá avisar tanto al administrador, como a los que tienen equipos prestados, de préstamos que estén a punto de caducar o que hayan caducado. Este aviso deberá ser, además de en la aplicación, por correo electrónico, y deberá enviar a los usuarios a una vista de la aplicación en la que puedan decidir qué hacer con el préstamo. También se notificará por medio de un correo del estado de las solicitudes de renovación

1.2 Objetivos

Los objetivos principales del presente trabajo fin de grado son los siguientes:

- Realizar el control de inventario de los equipos disponibles en los laboratorios docentes de la universidad. Además, incluirá la posibilidad de realizar la trazabilidad del préstamo, por un tiempo definido, de cualquiera de estos equipos, ya sea a profesorado o estudiantes.
- La aplicación tendrá una vista de administrador para la gestión de usuarios, inventario y control de los préstamos, y una vista para los usuarios que tengan algún

equipo prestado puedan ver los equipos que tienen a préstamo y el plazo, así como parar solicitar la renovación del equipo.

- La aplicación deberá avisar tanto al administrador, como a los que tienen equipos prestados de que ven prestatarios. Este aviso deberá ser, además de en la aplicación, por correo electrónico.

Para lograr la funcionalidad antes expuesta, se han determinado una serie de objetivos secundarios para el desarrollo:

- Análisis de la tarea y determinación de las interacciones que ha de poder tener cada tipo de usuario con la aplicación, así como la concepción de las vistas de la interfaz.
- Estudio de las dependencias necesarias para realizar las tareas.
- Desarrollo del servidor (*back-end*) y conexión con la base de datos.
- Implementación de los servicios y comprobación del correcto funcionamiento de éstos.
- Desarrollo de la aplicación cliente (*front-end*) y conexión con el servidor.
- Implementación de la interacción con el servidor.
- Adecuar el aspecto de las vistas al de la Universidad de Jaén.
- Generación de la versión final (*release)* de la aplicación cliente para integrarla con el servidor para obtener una aplicación completa.
- Revisión del funcionamiento correcto de la aplicación
- Elaboración de la memoria del proyecto.

2 ESTADO DEL ARTE

Actualmente, el lenguaje más utilizado en entornos de desarrollo web es JavaScript, debido a su facilidad de mantenimiento y compatibilidad con aplicaciones de desarrollo de servidores como Node.js, así como con el formato *JavaScript Object Notation* (JSON), empleado para el intercambio de datos entre sistemas.

Otra utilidad de JavaScript es su compatibilidad con programación basada en eventos, la cual aporta mucha facilidad a la hora de desarrollar entornos interactivos y contenido dinámico. También es compatible con multitud de librerías o *frameworks*, código preconstruido que facilita el desarrollo de nuevas aplicaciones. De entre las más importantes en la actualidad se encuentran React.js, Node.js, Angular y Express (1).

En cuanto a aplicaciones web del mismo ámbito que la nuestra, no hay ejemplo más cercano que el portal Petrus, diseñado por D. Pedro Aguilar Aguilar para la universidad de Jaén y que se sigue usando hoy en día. Esta plataforma tiene las siguientes funcionalidades a destacar:

- Permitir a los usuarios, entre ellos el profesorado, solicitar servicios (por ejemplo, mantenimiento, calibración, control de *stock,* etc…) que serán atendidos por los técnicos de laboratorio, que tendrán su propia vista.
- Gestionar el *stock* o inventario de los laboratorios, con funciones para añadir, buscar o eliminar artículos de la base de datos. Los artículos se pueden añadir uno por uno o varios a la vez mediante la importación de un archivo CSV.
- Controlar el mantenimiento de los equipos.

Para realizar las siguientes funciones, se han usado una variedad de lenguajes, incluyendo lenguajes para el manejo de bases de datos como MySQL y PHP, lenguajes para el diseño y estilado de páginas web como HTML y CSS y lenguajes de programación como JavaScript (el mismo que se utiliza en la aplicación web de este TFG), entre otros.

Donde más destacan este tipo de aplicaciones de gestión, sin embargo, es en el ámbito comercial para gestionar negocios. La gestión de inventario, sin embargo, no es sino una parte del conjunto de herramientas de gestión que conforman los sistemas de planificación de recursos empresariales o ERP. Estas plataformas permiten a las empresas controlar mediante una interfaz clara e intuitiva no solo qué productos tienen en inventario, sino también la cadena de suministro, el almacén, el estado de los pedidos y las finanzas de la empresa, además de permitir la generación de informes financieros. Uno de los ERP más importantes actualmente es Oracle NetSuite (2).

Otras aplicaciones de vital importancia en el desarrollo de aplicaciones web son aquellas que permiten el diseño y lanzamiento de plataformas de contenido web personales a los usuarios con menos conocimiento de programación. Un gran ejemplo es WordPress.

WordPress es una solución gratuita de código abierto (*open-source*) que permite crear prácticamente cualquier tipo de plataforma web (blogs, tiendas online, etc…) y es una de las más populares del mundo, siendo utilizada por un 43.1% de todas las páginas webs actualmente (3).

3 MATERIALES Y MÉTODOS

En esta sección, se detallan los materiales empleados para el desarrollo de la aplicación y cómo se han utilizado para lograr los objetivos. También se explicará cómo está compuesto el TFG, los componentes desarrollados y su función, así como los servicios REST que se han implementado.

3.1 Esquema del sistema

A continuación, se mostrará un esquema que ejemplifica el funcionamiento de la aplicación web y su conexión con otros sistemas.

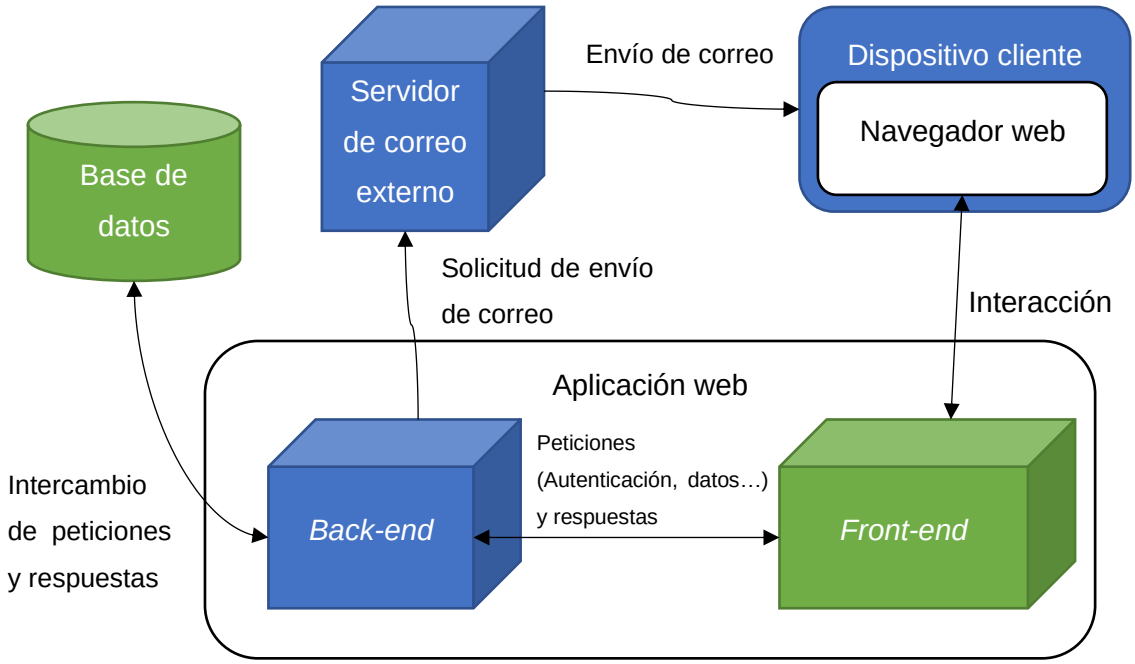

Ilustración 1: Esquema de las interacciones entre sistemas de la aplicación

Como se puede observar en el esquema, el cliente accederá a la aplicación web desde un dispositivo con conexión a Internet utilizando un navegador web. La aplicación cliente (*front-end*) provee al usuario de una interfaz mediante la cual puede interactuar con la aplicación y, en función de las interacciones que se produzcan, mandará peticiones al servidor (*back-end).* El servidor conecta con la base de datos para o bien actualizar la información dentro de esta, o para proveer a la aplicación cliente de la información que le ha solicitado mediante una petición REST. También conectará con un servidor de correo externo para solicitar el envío de correos generados por la aplicación.

Las tecnologías y aplicaciones empleadas en el desarrollo del TFG se comentarán en los siguientes apartados.

3.2 Lenguaje de programación: JavaScript

JavaScript es el lenguaje de programación más empleado en el ámbito del desarrollo web. Es un lenguaje orientado a objetos que puede usarse tanto en la parte de servidor para definir la lógica de la aplicación, como en la interfaz web para desarrollar una serie de vistas intuitivas y dinámicas de manera síncrona y asíncrona. Utilizándolo en ambos extremos, nos ahorramos conflictos derivados de traducir la lógica de un lenguaje a otro. También tiene soporte de bibliotecas y dependencias que dan mayor funcionalidad al programa y que facilitan la implementación de funciones. Esto incluye no solo las dependencias oficiales de uso extendido como React.js o Node.js, de las cuales hablaremos más adelante, sino también dependencias más pequeñas que añaden métodos más específicos.

3.3 Software externo al desarrollo de la aplicación

Esta categoría engloba los programas y plataformas alejados en mayor o menor medida de la aplicación en sí que se han utilizado para facilitar el desarrollo de la aplicación.

3.3.1 Visual Studio Code

Visual Studio Code (4) de Microsoft es el *Integrated Development Environment* (IDE) más utilizado del mundo para desarrollo web. Como cualquier IDE, es un entorno que permite el desarrollo de aplicaciones de manera eficiente. No solo tiene una interfaz intuitiva y personalizable, sino que soporta múltiples lenguajes de programación, así como complementos que facilitan el trabajo al programador y soporte nativo de repositorios en línea.

3.3.2 GitHub

Github (5) es el repositorio web más popular del mundo. En él, se puede alojar archivos y actualizarlos conforme el desarrollo del proyecto progresa. Así se puede tener un seguimiento de los cambios que se van haciendo a los archivos que conforman el proyecto y compartirlo con otros usuarios.

3.3.3 Postman

Postman (6) es una plataforma API que permite, entre muchas otras funciones, la prueba del funcionamiento de servidores en una etapa de desarrollo temprana en la que todavía no se ha desarrollado la aplicación cliente.

3.3.4 Zoho Mail

Zoho Mail (7) es un servicio de correos orientado al ámbito comercial que permite el automatizado de envíos de correo incluso desde aplicaciones de terceros como la nuestra. La razón por la que se ha utilizado este servicio en lugar de un servicio de correo más estandarizado como Gmail es porque la mayoría, incluyendo Gmail, están pensados

para uso personal y por tanto es más complicado configurar el servidor para que envíe correos desde estos servicios.

3.4 Base de datos

La base de datos se utilizará para almacenar la información de usuarios, artículos, préstamos y solicitudes de renovación pendientes. Se accederá a dicha información desde el servidor para responder a las peticiones enviadas por la aplicación cliente.

3.4.1 MongoDB

MongoDB (8) es una plataforma utilizada para la creación y gestión de bases de datos NoSQL escalables. Uno de sus principales atractivos es la compatibilidad con el formato JSON, utilizado ampliamente para almacenar objetos con atributos definidos. También dispone de una aplicación llamada MongoDB Compass, la cual permite al desarrollador interactuar directamente con la base de datos mediante una interfaz clara e intuitiva.

3.5 Back-end

El *back-end* se refiere a la parte de la aplicación que conforma el servidor, es decir, la parte que soporta la lógica de negocio sobre la cual la aplicación web se sostiene. Para su desarrollo, se ha empleado el siguiente material:

3.5.1 Node.js

Node.js (9) es una librería de código abierto basada en JavaScript que permite, de manera rápida e intuitiva, el despliegue de un servicio web fácilmente escalable y con buen rendimiento, proporcionando la funcionalidad necesaria para ello. Se pueden agregar funciones adicionales al servidor mediante la instalación de paquetes gracias a su sistema de gestión de paquetes *npm* desde una terminal de comandos. Todas estas cualidades lo hacen la librería web más utilizada en el mundo (1).

3.5.2 Mongoose

De entre las dependencias que se han añadido al servidor, una de las más importantes para acceder a la base de datos es Mongoose (10). Mongoose permite la generación de esquemas que definen un formato para los objetos que se añaden a una base de datos en MongoDB, cosa que no se puede hacer directamente en la base de datos. También permite definir modelos, constructores con métodos asíncronos para buscar, añadir, modificar o eliminar objetos de la base de datos que sigan el formato definido por los esquemas, haciendo que manejar la base de datos sea mucho más sencillo.

3.5.3 Express

Express es la librería para API REST más popular de Node.js. Con ella, se puede:

- Definir manejadores de peticiones REST para diferentes rutas en forma de URLs.

- Integración con motores de renderización de "vistas" para generar respuestas mediante la introducción de datos en plantillas.
- Establecer ajustes de aplicaciones web como qué puerto hay que usar para conectarse y la localización de las plantillas que se utilizan para renderizar la respuesta.
- Añadir procesamiento de peticiones *middleware* adicional en cualquier punto dentro de la tubería de manejo de la petición.

En este proyecto, se ha utilizado para indicar cómo debe responder el servidor a diferentes peticiones.

3.5.4 Axios

Axios es un cliente HTTP basado en promesas para Node.js. Se ha utilizado para realizar comprobaciones dentro del propio servidor que utilicen peticiones REST a métodos ya implementados.

3.5.5 Nodemailer

Nodemailer (11) es la dependencia más utilizada para la generación y envío automático de correos electrónicos desde servidores de dominios de confianza. Para su uso requiere autenticación a uno de estos servidores. En el contexto de la aplicación, se utiliza para generar los correos electrónicos que se han de mandar a los correos que nos indica la base de datos.

3.5.6 JSON Web Token

JSON Web Token o JWT (12) es un estándar de creación de tokens de ingreso o de sesión utilizados principalmente para autenticación de usuarios y verificación de sus privilegios. En el contexto de este proyecto, se proporcionarán a cada usuario una vez se hayan autenticado. Dicho token se comprueba conforme se utiliza la aplicación para confirmar la validez de la sesión.

3.5.7 Node-cron

Node-cron (13) es una dependencia que implementa un programador de tareas para que éstas se ejecuten en instantes de tiempo determinados. En nuestra aplicación se utiliza para que los correos se manden a diario.

3.6 Front-end

La aplicación cliente o *front-end* es la parte más gráfica de la aplicación. Su función consiste en proveer de una interfaz que permita a los usuarios interactuar con la parte lógica de la aplicación y con la base de datos.

3.6.1 React

React (14) es una biblioteca para aplicaciones cliente basada en JavaScript utilizada para diseñar interfaces de usuario interactivas. Está basada en componentes con estados que se pueden incluir dentro de otros para generar vistas de manera modular. Los componentes resultantes se pueden expresar con JSX, una sintaxis de JavaScript muy similar a XML, mezclada con elementos HTML, lo que hace su diseño e implementación bastante intuitivo.

Estas cualidades hacen que React sea uno de las bibliotecas más populares para el diseño de páginas web incluso hoy en día, siendo utilizada por multitud de empresas y desarrolladores en el mundo (1).

3.6.1.1 ¿Cómo se crea una aplicación web de React?

Al principio se puede pensar que construir la aplicación web para poder conectarse y tener un entorno de desarrollo requiere mucho trabajo. Pero la realidad es que, con un solo comando y mínima configuración previa, podemos empezar con una base muy sólida y centrarnos en diseñar los componentes y enrutarlos. Dicho comando es 'npx create-react-app <nombre de la app>'.

Esto nos genera un fichero con lo necesario para que, una vez ejecutemos el comando '*npm start*' en esa carpeta, nos inicie la aplicación web en el puerto 3000 en un entorno de desarrollo, donde mientras esa aplicación se esté ejecutando, podremos ver los cambios que hagamos en los archivos sin tener que apagar la aplicación. Además, no tenemos que estar completamente aislados del servidor. Se puede diseñar la aplicación para que se esté conectando con la aplicación servidor y así comprobamos que la aplicación funcione. Esto requiere que configuremos el CORS (*Cross-Origin Resource Sharing*) en el servidor para que la política del navegador SOP (*Same Origin Policy*) no bloquee la conexión entre aplicación cliente y servidor al ser de diferentes dominios.

Una vez completada la aplicación, ejecutaremos el comando 'npm run build'. Este comando genera en una carpeta llamada *'build'* una serie de archivos que constituyen la aplicación web en una versión de lanzamiento o *'release'*. Estos archivos se copiarán en la carpeta /public del servidor. Así, la aplicación cliente y servidor forman parte de la misma ejecución y cuando se ejecute el servidor, al acceder a la dirección en la que se ejecuta desde un navegador, se mostrarán las vistas definidas en la aplicación cliente y se podrá interactuar con la página.

3.6.1.2 Material UI

De entre las librerías a las que da acceso React, una de las que más se han usado en el desarrollo de la interfaz es *Material UI* (15). Consisten en una serie de componentes preconstruidos aplicando los principios del estándar *Material Design* de Google. Esto nos

permite crear una página web que se asemeje lo máximo posible en diseño y manejo a una página web convencional. También nos ahorra tener que definir exhaustivamente la mayor parte de la funcionalidad de los componentes ya que viene integrada en los mismos y es fácilmente modificable para adaptarla al comportamiento esperado del componente o al diseño que queremos buscar en la página. Además, como hemos dicho antes, React es de naturaleza modular, por lo que implementar estos componentes una vez importada la dependencia es simple. Adicionalmente, podemos definir una paleta de colores de la página compartida entre los componentes para que sea consistente y sea más sencillo aplicarlo a los que se añadan. También se incluyen plantillas de diseños de plataformas web.

3.6.2 Fetch

Fetch es una API para JavaScript que permite enviar peticiones asíncronas a un servidor. En esta aplicación, la utilizamos para cualquier situación en la que necesitemos formular una petición para obtener información de la base de datos o aplicar la funcionalidad que definimos anteriormente en el servidor (identificar usuarios, realizar reservas, borrar entradas de la base de datos, etc…).

3.6.3 Bibliotecas varias para añadir funciones adicionales

Como se ha mencionado anteriormente, para el proyecto se han utilizado una serie de dependencias de código abierto que han permitido implementar pequeñas funcionalidades de manera rápida, eficiente y sin muchos problemas de compatibilidad, ahorrando una cantidad considerable de tiempo de desarrollo. Estas dependencias se pueden encontrar en la página oficial de *npm* (13) y se instalan desde la consola de comandos. Las dependencias son:

- *React-edit-text*: Dependencia utilizada para añadir campos de texto editable
- *React-medium-image-zoom*: Dependencia utilizada para hacer que las imágenes sean ampliables
- *React-tag-input*: Dependencia utilizada para incluir un sistema de filtrado por etiquetas.

3.7 Explicación del TFG y sus componentes

En esta categoría se explicará cómo está hecho el TFG y los componentes implementados durante su desarrollo. También se explicarán los servicios REST que se han implementado en la aplicación.

3.7.1 Base de datos

La base de datos está desplegada por medio de MongoDB y se ha utilizado MongoDB Compass como interfaz para verificarla e interactuar. Su función, como se ha

elaborado anteriormente, es almacenar la información de usuarios, artículos, préstamos y solicitudes de renovación en formato JSON.

La URI usada para acceder a la base de datos durante todo el desarrollo ha sido mongodb://localhost:27017/. La base de datos se llama *test* y contiene cuatro colecciones, una por cada tipo de elemento, cada una con un formato que ha sido definido desde el servidor por medio de Mongoose al que MongoDB aplica un atributo de tipo *ObjectID* único llamado _id que se utiliza para diferenciarlo de otros elementos. Las colecciones definidas son las siguientes:

- objetos: Contiene los artículos que conforman el inventario. Los artículos no requieren de otros elementos de la base de datos para generarse. El administrador añade artículos desde la aplicación por medio de la vista de artículos a través de un formulario que genera una petición REST al servicio correspondiente. En la misma vista, los artículos se pueden eliminar mediante un botón, lo que activa el servicio REST correspondiente. Además, cuando se guardan los cambios hechos al texto editable que forma la presentación de los artículos, se modifica el objeto correspondiente en la base de datos.

  Formato:

```
{
    nombre: String,
    descripcion: String,
    ubicacion: String,
    observaciones: String,
    imagen: String, //para ser más exactos, es el directorio relativo de la imagen
    self_link: {
        type: String,
        required: true,
        default: function () {
            return `/objetos/${this._id}`;
        }
    }
}
```

- usuarios: Contiene los usuarios, tanto administradores como clientes. No requieren de otros elementos de la base de datos para generarse. Un usuario nuevo se puede registrar mediante el formulario de registro al entrar en la aplicación o por medio de la vista de usuarios del administrador. Los administradores pueden eliminar usuarios desde esa vista al hacer clic en el botón correspondiente. El texto que forma la representación de los usuarios en la aplicación es editable pero solo se

actualizará en la base de datos si el nuevo texto cumple con las exigencias de formato del campo de texto (si las hay).

Formato:

```
{
    user: String,
    nombre: String,
    apellidos: String,
    email: String,
    telefono: String,
    password: String,
    esadmin: {
        type: Boolean,
        required: true,
        default: false
    },
    self_link: {
        type: String,
        required: true,
        default: function () {
            return `/usuarios/${this._id}`;
        }
    }
}
```

- prestamos: Contiene los préstamos de artículos vigentes. Requieren de un artículo y un usuario existentes. Un usuario puede ser titular de múltiples préstamos simultáneos pero un artículo solo puede ser prestado a un usuario a la vez. Además, la fecha de fin no puede ser anterior a la fecha en la que se hizo el préstamo.

  Los clientes pueden realizar reservas de artículos desde su vista mediante un formulario. Los administradores pueden, de manera análoga, asignar reservas a otro usuario.

  Los préstamos se pueden cancelar en cualquier momento por su usuario respectivo o por un administrador, lo que lleva a su borrado de la base de datos. Cuando un usuario es eliminado de la base de datos, también se eliminan sus préstamos vigentes.

  Las observaciones se pueden modificar por un administrador al ser texto editable. La fecha de fin solo se puede modificar mediante una solicitud de renovación aceptada.

Formato:

```
{
    object_id: {
        type: String,
        required: true
    },
    user_id: {
        type: String,
        required: true
    },
    observaciones: String,
    fecha_inicio: Date,
    fecha_fin: {
        type: Date,
        required: true
    },
    self_link: {
        type: String,
        required: true,
        default: function () {
            return `/prestamos/${this._id}`;
        }
    },
    object_link: {
        type: String,
        required: true,
        default: function () {
            return `/objetos/${this.object_id}`;
        }
    },
    user_link: {
        type: String,
        required: true,
        default: function () {
            return `/usuarios/${this.user_id}`;
        }
    }
```

}

- solicitudes: Contiene las solicitudes de renovación de préstamos. Requiere de un préstamo, que a su vez requiere de un usuario y un artículo.

  Un cliente puede realizar una solicitud de renovación mediante el formulario correspondiente.

  Esta solicitud puede ser aceptada o rechazada por un administrador. En ambos casos, la solicitud se elimina de la base de datos. También se eliminará de la base de datos si se cancela el préstamo asociado.

  Una solicitud de préstamo no se puede modificar.

  Formato:

```
{
    prestamo_id: {
        type: String,
        required: true
    },
    object_id: {
        type: String,
        required: true
    },
    user_id: {
        type: String,
        required: true
    },
    motivo: {
        type: String,
        required: true
    },
    fecha_renovacion: {
        type: Date,
        required: true
    },
    self_link: {
        type: String,
        required: true,
        default: function () {
            return `/solicitudes/${this._id}`;
        }
```

```
    },
    user_link: {
        type: String,
        required: true,
        default: function () {
            return `/usuarios/${this.user_id}`;
        }
    },
    object_link: {
        type: String,
        required: true,
        default: function () {
            return `/objetos/${this.object_id}`;
        }
    },
    prestamo_link: {
        type: String,
        required: true,
        default: function () {
            return `/prestamos/${this.prestamo_id}`;
        }
    }

}
```

La interacción con la base de datos se realiza principalmente por medio del servidor a través de los métodos que proporciona Mongoose para el manejo de colecciones.

3.7.2 Back-end

El *back-end* es la parte que gestiona la aplicación que se ejecuta en el lado del servidor. Se despliega mediante un servidor en Node.js y realiza las siguientes funciones:

- Conectar con la base de datos utilizando los métodos de *Mongoose* para facilitar el control y modificación de ésta.
- Conectar con la aplicación cliente para responder a sus peticiones REST y hacerle llegar información de la base de datos
- Conectar con el servidor de correo para enviar mensajes de correo electrónico de manera automática cuando sea necesario.
- Generar y autenticar los JWT que necesitan los usuarios para utilizar los servicios de la aplicación.

Esta es la parte de la aplicación que se ha desarrollado primero, utilizando *Postman* como cliente para simular el envío y respuesta de peticiones. Los archivos que lo conforman son los siguientes:

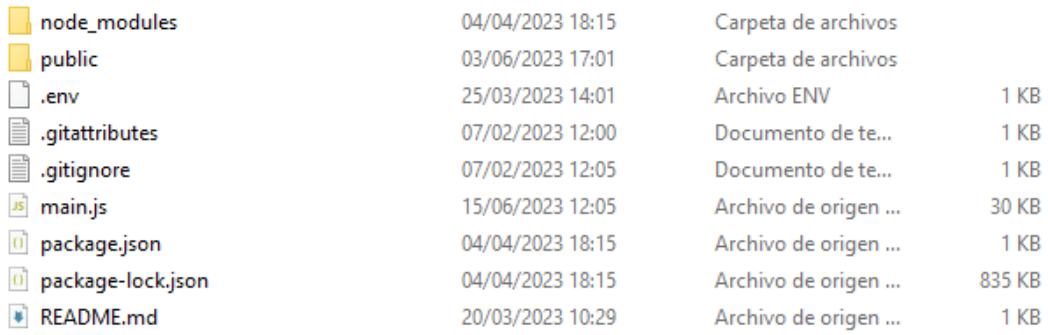

Ilustración 2: Carpeta donde se encuentra el back-end

- Node_modules: Carpeta que almacena todos los módulos que se han instalado por medio de npm. Node.js accede a ellos cuando se importan a un fichero.
- public: Carpeta que contiene los elementos necesarios para la parte cliente de la aplicación, como los archivos HTML y las imágenes. Durante el proceso de desarrollo, solo se han copiado los archivos de la versión final de la aplicación cliente a esta carpeta.
- .env: Archivo donde se definen credenciales en formato clave-valor para servicios. Se ha guardado el secreto del JSON Web Token, el e-mail y contraseña que se usan para enviar los correos (avisoalmacentfguja@zohomail.eu) y el puerto que ocupa la aplicación. No se guardan en repositorios online como Github.
- main.js: El archivo principal del servidor y donde se ha definido todo el código, incluyendo los servicios REST y el código que permite el despliegue de la aplicación.
- package.json: Indica todos los archivos necesarios para la correcta ejecución de la aplicación, incluyendo archivos, scripts y dependencias.
- package-lock.json: Archivo generado automáticamente cada vez que se modifica package.json que representa estados anteriores de ese archivo.
- README.md: Descripción del repositorio de GitHub.

3.7.3 Front-end

La aplicación cliente o *front-end* tiene como función proporcionar una interfaz clara e intuitiva a los usuarios que les permita interactuar con la aplicación y acceder a los servicios de ésta. Para su desarrollo, se ha generado una aplicación React con entorno de desarrollo separado del servidor mediante los procesos indicados anteriormente. Dentro de la aplicación, se han desarrollado varios componentes, que se pueden dividir entre:

- Vistas: React permite definir componentes con notación similar a la de HTML (el estándar en desarrollo web) desde JavaScript utilizando la sintaxis JSX. Estos componentes se pueden exportar a otros ficheros para combinarlos y así generar vistas de la aplicación. Los componentes también pueden pasarse valores JavaScript de padre a hijo utilizando lo que se conoce como *props.* Esta cualidad se ha empleado para definir la vista de administrador y de usuario para el mismo servicio en el mismo archivo, reduciendo considerablemente la cantidad de archivos necesarios y la codificación redundante que hay que hacer. También se han utilizado componentes importados de plantillas MUI para minimizar el tiempo invertido en el estilado de los componentes.
- Funciones: Al igual que muchos otros lenguajes de programación, se pueden definir funciones o métodos en archivos aparte e importarlos en las vistas donde sea necesario. Esto se ha hecho sobre todo para las funciones que envían peticiones REST al servidor.

Los archivos que conforman la aplicación cliente son:

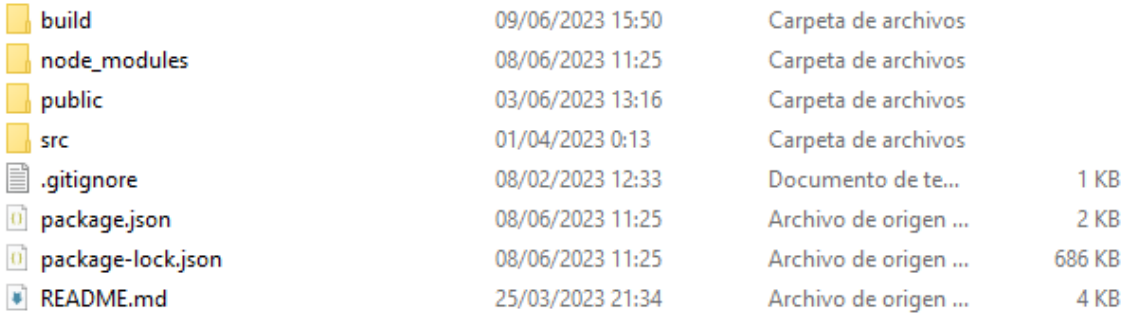

Ilustración 3: Imagen de la carpeta donde se ha desarrollado el front-end

- build: Carpeta que contiene la versión *release* del *front-end*, conteniendo los manifiestos, elementos y código empleado en un aspecto más comprimido y listo para su uso. Su contenido se ha copiado a la carpeta public del servidor para obtener una aplicación completa.
- node_modules: Exactamente la misma función que en el servidor.
- public: Carpeta que contiene el archivo index.html que se lee para tener la vista en el navegador y es donde se renderizan los elementos de React, así como otros elementos, pero no ha sido necesario modificarlos en la aplicación.
- src: Carpeta que contiene los archivos JavaScript en los que definimos los componentes y constantes que afectan a las vistas.
- package.json, package-lock.json y README.md: Mismas funciones que las descritas en el apartado del servidor.

Dentro de la carpeta *src,* encontramos lo siguiente:

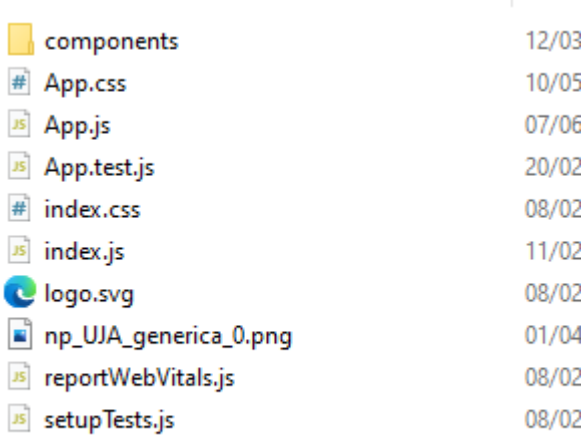

Ilustración 4: Carpeta src del front-end

- components: Carpeta donde se encuentran la mayoría de los archivos donde se han definido componentes, funciones o constantes.
- App.css: Archivo CSS donde se encuentran los estilos relevantes para el archivo App.js.
- App.js: Componente en el cual se ha definido un *div* que contiene un componente *Routes* que aloja muchos componentes *Route*. Cada uno de estos componentes renderiza un componente determinado cuando se accede a una ruta determinada. Por ejemplo, uno de los *Route* está configurado de tal forma que si se accede a la URL [servidor]/login, se renderice el componente *SignIn*. Este archivo es lo que permite que tengamos vistas diferentes para diferentes URLs dentro de la misma aplicación.
- index.css: Archivo de estilos de index.js
- Index.js: Archivo que renderiza el contenido de React definido en App en el archivo index.html. El componente App está envuelto en otro llamado *BrowserRouter* que le hace pasar la URL para alternar entre vistas, el cual está envuelto en otro llamado *React.StrictMode*, que establece el modo estricto de JavaScript que permite realizar mayores comprobaciones de errores en un entorno de desarrollo.
- reportWebVitals.js: Archivo que monitoriza las estadísticas de funcionamiento de la web para leerse mediante un módulo llamado *web-vitals.*
- setupTests.js: Archivo que se usa para generar un entorno de pruebas en caso de que utilicemos una API.

A continuación, explicaremos los componentes que se han definido para cada vista y su función:

3.7.3.1 Vista inicial

URL relativa: /

Archivo local: /src/components/inicio/Inicio.js

Para esta vista, se ha definido una imagen, una nota al pie (ambos usando tags de HTML) y dos botones usando el componente *Botonnormal* (que es el componente *Button* de MUI con un estilo definido) que llevan a la pantalla de inicio de sesión y de registro respectivamente.

3.7.3.2 Vista de inicio de sesión

URL relativa: /login

Archivo local: /src/components/login/SignIn.js

Estados:

- *open*: Booleano usado para definir el estado de apertura del *snackbar.*
- *status*: Estado usado para definir el mensaje de error

Componentes implementados:

- *Avatar*: Componente MUI que define un recuadro para imágenes de avatar
- *Botonnormal:* Definido en /src/components/vistas/Constantes.js. Componente *Button* de MUI, es decir un botón, estilado.
- *Box*: Componente MUI que envuelve otros componentes principalmente para necesidades de estilado
- *Container*: Componente MUI que centra contenido horizontalmente.
- *Copyright*: Componente definido en el archivo /src/componentes/vistas/Constantes.js. Es un pie de copyright de MUI.
- *CssBaseline*: Componente MUI similar a *normalize.css*. Este componente normaliza el estilado de los demás componentes. Para ser más exactos, elimina los márgenes y aplica la paleta de colores por defecto de *Material Design* al fondo.
- *Grid*: Componente MUI que define una cuadrícula que se adapta a múltiples tamaños de vista.
- *Link*: Componente MUI que define un texto con enlace.
- LockOutLinedIcon: Icono MUI de un candado.
- *Snackbar*: Componente MUI que define una notificación emergente de carácter temporal similar a las de la aplicación Google Keep. En esta vista se usa para informar de errores.
- *TextField*: Componente MUI que define un campo de texto para formularios.
- *ThemeProvider*: Utiliza el tema definido mediante el método createTheme() para aplicar su estilo a los componentes hijos.
- *Typography*: Componente MUI para definir recuadros de texto con estilo y tamaño consistente.

Métodos utilizados:

- *DoLogin(user, password)*: Definido en el archivo /components/login/DoLogin.js. Recibe las credenciales introducidas en el formulario de inicio de sesión para enviarlas mediante un POST /login al servidor en el cuerpo de texto del mensaje. Luego recoge el token de la cabecera de la respuesta.

Relación con otras vistas:

- El botón Iniciar Sesión lleva a la vista general de usuario o administrador según el tipo de usuario si el inicio de sesión es exitoso.
- El texto debajo del botón Iniciar Sesión lleva a la vista de registro.

3.7.3.3 Vista de registro

URL relativa: /register

Archivo local: /src/components/signup/SignUp.js

Estados:

- *open*
- *status*

Componentes implementados:

- *Avatar*
- *Box*
- *Botonnormal*
- *Container*
- *Copyright*
- *CssBaseline*
- *Grid*
- *Link*
- *LockOutLinedIcon*
- *Snackbar*
- *TextField*
- *ThemeProvider*
- *Typography*

Métodos utilizados:

- *DoRegister(user, password, nombre, apellidos, email, tlf):* Definido en el archivo /components/signup/DoRegister.js. Recibe las credenciales introducidas en el formulario de registro para enviarlas mediante un POST /registrarusuario al servidor en el cuerpo de texto del mensaje. Luego recoge el token de la cabecera de la respuesta.

Relación con otras vistas:

- El botón Registrarse lleva a la vista general de usuario si el registro es exitoso.

- El texto debajo del botón Registrarse lleva a la vista de registro.

3.7.3.4 Vista general de usuario

URL relativa: /user/

Archivo local: /src/components/vistas/Usuarioview.js

Estados:

- *objects*: Contiene los artículos que hay en la base de datos.
- *open*: Booleano que controla si la barra lateral está abierta o no
- *prestamos*: Contiene los préstamos vigentes del usuario actual
- *renovaciones:* Contiene el número de solicitudes de renovación vigentes del usuario actual
- *usuario:* Contiene el usuario de la sesión actual.

Componentes definidos:

- *AppBar*: MuiAppBar estilada en /src/components/vistas/Constantes.js. Componente MUI que renderiza una barra en la parte superior de la página con información contextual (título de pantalla, sesión, navegación, etc…)
- *Box*
- *Container*
- *Copyright*
- *CssBaseline*
- *Divider*: Componente MUI que establece una división entre elementos.
- *Drawer:* Componente MUI que define una barra lateral que se controla mediante un estado y cuyo estilo está definido en el archivo /components/vistas/Constantes.js
- *Grid*
- *MenuIcon:* Icono de menú del MUI.
- *Item:* Elementos hijo del componente Grid que pueden ocupar diferentes tamaños de la cuadrícula en función del tamaño de la pantalla. Albergan otros componentes.
- *IconButton*: Icono interactivo (en este caso, tiene el icono MenuIcon) que, en el contexto de la aplicación, abre o cierra la barra lateral.
- *List*: Componente MUI que define un listado de texto e imágenes.
- *ListDrawer*: Componente definido en /src/components/vistas/ListDrawer.js. Define el contenido de la lista en el componente *Drawer*. Cada una de las entradas de esta lista se compone de un icono y texto que, al hacer clic en ellas, llevan a su respectiva vista. Está controlado por dos *props*:
    a) *admin*: Booleano que depende de si el usuario actual es administrador o no. Alterna entre enlaces a las vistas de cada rol y, si está activo, muestra el enlace a la vista de usuarios.

    b) *numrenovaciones*: Número de renovaciones existente para indicar el número de la notificación visible en el icono de la entrada de renovaciones
- *LogoutIcon*: Icono que cuando se le hace clic, el usuario cierra su sesión y es redirigido a la pantalla de inicio de sesión.
- *ObjectList*: Componente definido en /src/components/listas/Listaobjetos.js. Define una tabla resumen que muestre los artículos mediante el componente *Table* de MUI. Su contenido se controla mediante dos props:
    a) admin: Booleano que depende de si el usuario es administrador o no. Controla el texto con enlace debajo de la tabla
    b) objectslist: Lista de artículos que la tabla debe mostrar
- *Paper*: Componente MUI que actúa como un papel de fondo.
- *PrestamosList:* Componente definido en /src/components/listas/Listaprestamos.js. Define una tabla resumen que muestre los préstamos mediante el componente *Table* de MUI. Su contenido se controla mediante dos props:
    a) admin: Booleano que depende de si el usuario es administrador o no. Controla el texto con enlace debajo de la tabla y si la tabla ha de mostrar o no el recipiente del préstamo.
    b) prestamoslist: Lista de préstamos que la tabla debe mostrar
- *ThemeProvider*
- *Toolbar*: Subcomponente de AppBar que contiene los objetos de esta.
- *Typography*
- *Zoom:* Componente definido por la dependencia *react-medium-image-zoom.* Cuando se envuelve una imagen con él, se puede hacer clic a la imagen para ampliarla y que ocupe toda la pantalla, similar al famoso portal público de artículos Medium.

Métodos utilizados:

- getObjetos(prestado, id): Definido en /src/components/getters/getobjetos.js. Prepara una petición GET al servidor para obtener una lista de artículos. La URL a la que se hace depende de las opciones que se introduzcan:
    a) Si se pone prestado en *false*, se hace GET a /objetossinprestar/ y /objetosprestados/ y se concatenan los resultados.
    b) Si no se introduce un id y se pone prestado en *true*, se hace un GET /objetos/.
    c) Si se introduce un id y se pone prestado en *true,* se hace un GET /objetos/[id]
- getPrestamos(id, criterio): Definido en /src/components/getters/getprestamos.js. Prepara un GET al servidor para obtener una lista de préstamos. La URL a la que se hace depende de las opciones que se introduzcan:

a) Si no se introduce un id, se hace un GET a /prestamos/.
b) Si se introduce un id, dependiendo de si la variable 'criterio' vale "prestamo" o "usuario", se hace un GET a /prestamos/[id] o /prestamosdeusuario/[id].

- getRenovaciones(id, criterio): Definido en /src/components/getters/getrenovaciones.js. Prepara un GET al servidor para obtener una lista de renovaciones con o sin diferentes criterios. La URL a la que se hace depende de las opciones que se introduzcan:
  a) Si no se introduce un valor de id y de criterio, se hace un GET a /renovaciones/.
  b) Si se introduce un id, dependiendo de si la variable 'criterio' vale "prestamo", "usuario" o ninguno de los dos, se hace un GET a /solicitudesdeprestamo/[id], /solicitudesdeusuario/[id] o /solicitudes/[id] respectivamente.
- getUsuarios(id): Definido en /src/components/getters/getusuarios.js. Prepara un GET /usuarios/ al servidor para obtener una lista de usuarios. Si además se introduce un id, prepara uno a GET /usuarios/[id] en su lugar para obtener el usuario de ese id.
- addImagesPrestamo(data): Definido en /src/components/listas/addImagesPrestamo.js. Función que obtiene el nombre y la imagen a mostrar del artículo de cada préstamo y lo adjunta a su préstamo correspondiente.
- validarToken(token): Definido en /src/components/tokens/validartoken.js. Prepara un GET /usuario/validatetoken con el token en la cabecera para autenticarlo.
- logout(): Definido en /src/components/login/logout.js. Elimina el token del almacenamiento local del navegador, cerrando la sesión actual.

Relación con otras vistas:

- El icono de *logout* en la esquina superior derecha devuelve a la pantalla de inicio de sesión.
- El texto *Ver más* debajo de cada tabla lleva a la vista de los elementos de su respectiva tabla.
- Cada elemento del listado del menú lateral lleva a su vista correspondiente (la entrada de Artículos a la vista de artículos, y así sucesivamente.

3.7.3.5 Vista general de administrador

URL relativa: /admin/

Archivo local: /src/components/vistas/Adminview.js

Estados:

- open: Booleano que controla si la barra lateral está abierta o no

- usuario: Contiene el usuario de la sesión actual.
- *objects*: Contiene los artículos que hay en la base de datos.
- prestamos: Contiene los préstamos vigentes del usuario actual
- renovaciones: Contiene el número de solicitudes de renovación vigentes del usuario actual

Componentes definidos:

- *AppBar*
- *Box*
- *Container*
- *Copyright*
- *CssBaseline*
- *Divider*
- *Drawer*
- *Grid*
- *IconButton*
- *List*
- *ListDrawer*
- *LogoutIcon*
- *ObjectList*
- *Paper*
- *PrestamosList*
- *ThemeProvider*
- *Toolbar*
- *Typography*
- *UserList:* Definido en /src/components/vistas/ListaUsuarios.js. Define una tabla resumen que muestre los préstamos mediante el componente MUI. Su contenido se controla mediante un *prop (*users), que contiene la lista de usuarios.
- *Zoom*

Métodos utilizados:

- getObjetos(prestado, id)
- getPrestamos(id, criterio)
- getRenovaciones(id, criterio)
- getUsuarios(id)
- addImagesPrestamo(data)
- validarToken(token)
- logout()

Relación con otras vistas:

- El icono de *logout* en la esquina superior derecha devuelve a la pantalla de inicio de sesión.
- El texto *Ver más* debajo de cada tabla lleva a la vista de los elementos de su respectiva tabla.
- Cada elemento del listado del menú lateral lleva a su vista correspondiente (la entrada de Artículos a la vista de artículos, y así sucesivamente).

3.7.3.6 Vista de datos de usuarios

URL relativa: /admin/users

Archivo local: /src/components/vistas/AdminUsers.js

Props:

- *admin:* Siempre está true. Se utiliza para alternar entre vistas de administrador y no (no muy relevante en esta vista, pero sí en las demás)

Estados:

- *adduser*: Booleano que controla la apertura del formulario para añadir nuevo usuario
- *needupdate*: Estado que controla las actualizaciones del renderizado de la vista
- *open*
- *renovaciones*
- *state:* En este estado se guardan los datos que se quieren mostrar en función de la vista, en este caso, los usuarios.
- *tags*: Array que contiene las etiquetas del sistema de filtrado.
- *usuario*

Componentes definidos:

- *AppBar*
- *Box*
- *ChevronLeftIcon*
- *Collapse:* Componente MUI que implementa un efecto de transición controlado por un estado en el que el objeto se expande/despliega desde uno de sus lados.
- *Container*
- *CssBaseline*
- *Divider*
- *Drawer*
- *EditText:* Componente definido por la dependencia *react-edit-text.* Consiste en un cuadro de texto editable si se hace clic en éste. Al pulsar Enter, los cambios hechos al campo se actualizan en la base de datos (en el caso de la vista de datos de usuarios, se realiza una comprobación de formato primero). Pulsar ESC o hacer clic fuera del campo de texto no guarda los cambios realizados.

- *FormControl:* Componente MUI que permite asociar formularios con texto de ayuda, etiquetas, indicadores de error, etc…
- *Formnewuser:* Componente definido dentro del mismo archivo que el resto de la vista. Consiste en un formulario para introducir un nuevo usuario.
- *Grid*
- *IconButton*
- *Item*
- *List*
- *ListDrawer*
- *LocalizationProvider:* Componente que adapta la fecha introducida en los formularios a la localización indicada para solucionar conflictos causados por diferencia horaria.
- *LogoutIcon*
- *MenuIcon*
- *ReactTags:* Componente definido por la dependencia *react-tag-input.* Consiste en un cuadro de texto en el que el usuario puede introducir términos de búsqueda. Cuando se pulsa Enter, el contenido del texto se vuelve una etiqueta arrastrable o eliminable.
- *TextField*
- *ThemeProvider*
- *Toolbar*
- *Typography*

Métodos utilizados:

- *DoRegister(user, password, nombre, apellidos, email, tlf)*
- *deleteusuarios(id):* Definido en /src/componentes/deleteusuarios/deleteusuarios.js. Prepara un DELETE /usuarios/[id] para borrar el usuario al que pertenece la id.
- *Filtrarlista(database,tags):* Definido en /src/componentes/vistas/FilterList.js. Si hay etiquetas existentes, recorre la información a mostrar para quedarse solo con las entradas que tengan al menos una coincidencia de texto independientemente del atributo con todas las etiquetas. Devuelve los resultados de filtrado.
- *getRenovaciones(id,criterio)*
- *getUsuarios(id)*
- *logout()*
- *ModificarUsuario(id, name, value):* Definido en /src/components/posters/ModificarUsuario.js. Prepara un POST /modificarusuario al servidor utilizando el id del usuario objeto de modificación y el par nombre-valor del atributo a modificar.

- *validarToken(token)*

Relación con otras vistas:

- El icono de *logout* en la esquina superior derecha devuelve a la pantalla de inicio de sesión.
- Cada elemento del listado del menú lateral lleva a su vista correspondiente (la entrada de Artículos a la vista de artículos, y así sucesivamente).

3.7.3.7 Vista de datos de artículos

URL relativa: /admin/objetos y /user/objetos (ambas llevan al mismo componente)

Archivo local: /src/components/vistas/AdminObjects.js

Props:

- *admin:* Booleano que es true o false dependiendo de la URL que se utilice. Se utiliza para alternar entre vistas de administrador y de usuario definidas en el mismo archivo.

Estados:

- *additem:* Booleano que controla la apertura del formulario para añadir nuevo usuario
- *needform*: Estado que, junto con el atributo *viewform* añadido sobre cada elemento renderizado, controla la apertura y cierre de los formularios individuales de cada componente *Container* donde se muestran elementos de la base de datos.
- *needupdate*
- *open*
- *renovaciones*
- *state:* En este estado se guardan los datos que se quieren mostrar en función de la vista, en este caso, los artículos.
- *tags*
- *usuario*

Componentes definidos:

- *AppBar*
- *Box*
- *Button*
- *ChevronLeftIcon*
- *Collapse*
- *Container*
- *CssBaseline*
- *DatePicker:* Componente MUI que define un campo de formulario para introducir una fecha mediante un selector con calendario incorporado.
- *DeleteIcon:* Icono MUI.

- *Divider*
- *Drawer*
- *EditText*
- *EditTextarea:* Componente definido en la dependencia *react-edit-text.* Funcionalmente similar a *Edittext* pero el campo de texto editable es multilínea y hacer Shift+Enter genera una nueva línea.
- *Formnewobject:* Componente definido en el mismo archivo que consiste un formulario para introducir un nuevo objeto.
- *Grid*
- *IconButton*
- *InputLabel:* Etiqueta que designa el uso de un campo dentro del formulario
- *Item*
- *List*
- *ListDrawer*
- *LogoutIcon*
- *MenuIcon*
- *MenuItem:* Componente hijo de un componente *Select* que designa una de las opciones de dicho componente.
- *ReactTags:* Componente definido por la dependencia *react-tag-input.* Consiste en un cuadro de texto en el que el usuario puede introducir términos de búsqueda. Cuando se pulsa Enter, el contenido del texto se vuelve una etiqueta arrastrable o eliminable.
- *Select:* Campo para formulario en el que se da a elegir una de entre múltiples opciones discretas mediante un menú desplegable. Las opciones se designan mediante componentes *MenuItem* hijos.
- *TextField*
- *ThemeProvider*
- *Toolbar*
- *Typography*
- *Zoom*

Métodos utilizados:

- *addObjeto(nombre,descripcion,ubicación,observaciones,imagen):* Definido en /src/components/posters/AddObjeto.js. Prepara un POST /registrarobjeto con la información de artículo introducida en el formulario para añadir. El cuerpo de la solicitud incluye los datos textuales introducidos y la imagen introducida en formato de cadena de caracteres. Devuelve el texto de la respuesta.

- *deleteobjetos(id):* Definido por /src/components/deleters/deleteobjetos.js. Prepara un DELETE /objetos/[id] para borrar el artículo al que corresponde ese id. Devuelve el texto de la respuesta.
- *Filtrarlista(database,tags)*
- *getObjetos(id,prestado)*
- *getRenovaciones(id,criterio)*
- *getUsuarios(id)*
- *logout()*
- *ModificarObjeto(id, name, value):* Definido en /src/components/posters/ModificarObjeto.js. Prepara un POST /modificararticulo al servidor utilizando el id del artículo objeto de modificación y el par nombre-valor del atributo a modificar. Devuelve el mensaje de estado
- *Reservarobjeto(user_id,object_id,fecha_fin,observaciones):* Definido en /src/components/posters/Reservarobjeto.js. Prepara un POST /registrarprestamo en el cual se introduce en el cuerpo de la solicitud los datos necesarios para añadir un préstamo de un artículo a la base de datos. Devuelve el mensaje de estado.
- *validarToken(token)*

Relación con otras vistas:

- El icono de *logout* en la esquina superior derecha devuelve a la pantalla de inicio de sesión.
- Cada elemento del listado del menú lateral lleva a su vista correspondiente (la entrada de Artículos a la vista de artículos, y así sucesivamente).

3.7.3.8 Vista de datos de préstamo

URL relativa: /admin/prestamos y /user/prestamos (ambas llevan al mismo componente)

Archivo local: /src/components/vistas/AdminPrestamos.js

Props:

- *admin:* Booleano que es true o false dependiendo de la URL que se utilice. Se utiliza para alternar entre vistas de administrador y de usuario definidas en el mismo archivo.

Parámetros de búsqueda:

- id: En este parámetro opcional, se puede pasar una id de préstamo para que el programa solo renderice ese préstamo (utilizado para la funcionalidad del botón “Ver préstamo”)
- notificacion: Si vale “true”, se alterna a la vista que se muestra al hacer clic en el enlace de una notificación de correo. Si vale cualquier otra cosa o nada, se deja la vista como estaba

Estados:

- *needform*
- *needupdate*
- *open*
- *renovaciones*
- *state:* En este estado se guardan los datos que se quieren mostrar en función de la vista, en este caso, los préstamos.
- *tags*
- *usuario*

Componentes definidos:

- *AppBar*
- *Box*
- *Button*
- *ChevronLeftIcon*
- *Collapse*
- *Container*
- *CssBaseline*
- *DatePicker*
- *Divider*
- *Drawer*
- *EditText*
- *EditTextarea*
- *FormularioRenovacion:* Componente definido en el mismo archivo que la vista. Formulario mediante el cual el cliente puede solicitar una renovación del préstamo.
- *Grid*
- *IconButton*
- *InputLabel*
- *Item*
- *List*
- *ListDrawer*
- *LogoutIcon*
- *MenuIcon*
- *MenuItem*
- *ReactTags*
- *Select*
- *TextField*
- *ThemeProvider*
- *Toolbar*

- *Toolbar*
- *Typography*
- *Zoom*

Métodos utilizados:

- *addImagesPrestamo(data)*
- *addObjeto(nombre,descripcion,ubicación,observaciones,imagen)*
- *deleteprestamos(id):* Definido por /src/components/deleters/deleteprestamos.js. Prepara un DELETE /objetos/[id] para borrar el artículo al que corresponde ese id. Devuelve el texto de la respuesta.
- *Filtrarlista(database,tags)*
- *getPrestamos(id,criterio)*
- *getRenovaciones(id,criterio)*
- *getUsuarios(id)*
- *logout()*
- *ModificarPrestamos(id, name, value):* Definido en /src/components/posters/ModificarPrestamos.js. Prepara un POST /modificarprestamo al servidor utilizando el id del préstamo objeto de modificación y el par nombre-valor del atributo a modificar. Devuelve el mensaje de estado.
- *solicitarrenovacion(prestamo,motivo,fecha_renovacion):* Definido en /src/components/posters/Solicitarrenovacion.js. Prepara un POST /solicitarrenovacion al servidor con los datos necesarios para solicitar una renovación de un préstamo. Devuelve el mensaje de estado.
- *validarToken(token)*

Relación con otras vistas:

- El icono de *logout* y el de *login* en la esquina superior derecha devuelven a la pantalla de inicio de sesión.
- Cada elemento del listado del menú lateral lleva a su vista correspondiente (la entrada de Artículos a la vista de artículos, y así sucesivamente).

3.7.3.9 Vista de datos de renovaciones

URL relativa: /admin/renovaciones/ y /user/renovaciones/ (ambas llevan al mismo componente)

Archivo local: /src/components/vistas/AdminRenovaciones.js

Props:

- *admin:* Booleano que es true o false dependiendo de la URL que se utilice. Se utiliza para alternar entre vistas de administrador y de usuario definidas en el mismo archivo.

Estados:

- *needupdate*
- *open*
- *renovaciones*
- *state:* En este estado se guardan los datos que se quieren mostrar en función de la vista, en este caso, las solicitudes de renovación.
- *usuario*

Componentes definidos:

- *AppBar*
- *Box*
- *Button*
- *ChevronLeftIcon*
- *Container*
- *CssBaseline*
- *Divider*
- *Drawer*
- *Grid*
- *IconButton*
- *Item*
- *List*
- *ListDrawer*
- *LogoutIcon*
- *MenuIcon*
- *ThemeProvider*
- *Toolbar*
- *Typography*
- *Zoom*

Métodos utilizados:

- *adddataRenovaciones(data):* Definido por /src/components/listas/adddatarenovaciones.js. Añade datos adicionales a las renovaciones procedentes del usuario, artículo y préstamo involucrado. Estos son el nombre del usuario involucrado, la fecha de final del préstamo y la imagen del artículo prestado.
- *deleterenovaciones(id):* Definido por /src/components/deleters/deleterenovaciones.js. Prepara un DELETE /renovaciones/[id] para borrar la solicitud de renovación denegada a la que corresponde ese id. Devuelve el texto de la respuesta.

- *DoRenovation(id,fecha_renovacion):* Definido por /src/components/listas/DoRenovation.js. Prepara un POST /renovar donde se hace pasar el id de la solicitud y la nueva fecha de renovación para renovar el préstamo correspondiente.
- *Filtrarlista(database,tags)*
- *getRenovaciones(id,criterio)*
- *getUsuarios(id)*
- *logout()*
- *validarToken(token)*

Relación con otras vistas:

- El icono de *logout* en la esquina superior derecha devuelve a la pantalla de inicio de sesión.
- Cada elemento del listado del menú lateral lleva a su vista correspondiente (la entrada de Artículos a la vista de artículos, y así sucesivamente).

3.7.4 Servicios REST

Los servicios REST hacen de intermediarios en una arquitectura cliente-servidor, como la que se forma entre nuestro *front-end* (cliente) y *back-end* (servidor). Por tanto, son especialmente útiles para el traspaso de datos utilizando el formato JSON, que es el mismo que utiliza nuestra base de datos. A continuación, se explican las funcionalidades que se ofrecen por medio de servicios REST con el siguiente formato:

Servicio (Método y URL):

Datos que enviar (si no se especifica dónde, se asume que es en el cuerpo de la solicitud).

Funcionamiento.

Datos devueltos (en caso de que se haga con éxito)

- **Servicio POST /login:**
  - Datos que enviar:

    {
    “user”: String,
    “password”: String
    }
  - Funcionamiento: El servidor comprueba que las credenciales son válidas y, en el caso de que lo sean, genera y envía un JSON Web Token (generado usando la fecha actual y el id de usuario y firmada con una clave de 32 caracteres) que el usuario utilizará para validar su sesión.

    - Datos devueltos: Token JWT del usuario en la cabecera JWT-Token de la respuesta
- **Servicio GET /usuario/validateToken**
    - Datos que enviar: Cabecera tokenHeaderKey con la JWT a identificar
    - Funcionamiento: El servidor verifica si el JWT enviado es válido usando el secreto de 32 caracteres definido en el archivo .env (el mismo secreto usado para crearla). Si el token es válido, se envía el id del usuario correspondiente, que el cliente usará para verificar si el usuario es administrador o no.
    - Datos devueltos: ID de usuario (_id) en el cuerpo de la respuesta como cadena de caracteres
- **Servicio POST /registrarusuario**
    - Datos que enviar:

      {
      "user": String,
      "nombre": String,
      "apellidos": String,
      "email": String,
      "telefono": String,
      "password": String,
      "esadmin": false
      }
    - Funcionamiento: El servidor comprueba que el usuario no existe ya en la base de datos y si el usuario no es administrador (ya que este proceso es para meter clientes). Si ambas son ciertas, se genera la JWT como en el proceso de login y se añade el usuario en la base de datos
    - Datos devueltos: Token JWT del usuario en la cabecera *JWT-token* de la respuesta
- **Servicio POST /registrarobjeto**
    - Datos que enviar:

      {
      "nombre": nombre,
      "descripcion": String,
      "ubicacion": String,
      "observaciones": String,
      "imagen": String que contiene la imagen en formato de caracteres
      }

- Funcionamiento: La información de la imagen se extrae y se genera un archivo con nombre pseudoaleatorio y la misma extensión que contiene esa información. Esa imagen se llama cuando se tenga que mostrar el objeto.
- Datos devueltos: Mensaje de estado

- **Servicio POST /registrarprestamo**
  - Datos que enviar:

    {
    "user_id": String,
    "object_id": String
    "fecha_inicio": Date
    "fecha_fin": Date
    “observaciones": String
    }
  - Funcionamiento: Se comprueba si el artículo y usuario involucrados existen en la base de datos. Después, se comprueba si el artículo está siendo prestado. Por último, se verifica que la fecha de final del préstamo es posterior a la fecha en la que se hizo. Si todas las comprobaciones son correctas, el préstamo se añade a la base de datos.
  - Datos devueltos: Mensaje de estado
- **Servicio POST /solicitarrenovacion**
  - Datos que enviar:

    {
    "prestamo_id": prestamo._id,
    "user_id": prestamo.user_id,
    "object_id": prestamo.object_id,
    "motivo": String,
    "fecha_renovacion": Date
    }
  - Funcionamiento: Se añade la solicitud de renovación a la base de datos
  - Datos devueltos: Mensaje de estado
- **Servicio POST /renovar**
  - Datos que enviar:

    {
    “id_solicitud”: String,
    "fecha_renovacion": Date
    }

    - Funcionamiento: El servidor modifica la fecha del préstamo afectado, borra la solicitud y envía un correo al cliente que confirme que se ha renovado el préstamo.
    - Datos devueltos: Mensaje de estado
- **Servicio DELETE /renovaciones/:id**
    - Datos que enviar: El id de la renovación como parámetro en la URL
    - Funcionamiento: El servidor borra la solicitud y envía un correo al cliente que confirme que se ha denegado el préstamo.
    - Datos devueltos: Mensaje de estado
- **Servicio DELETE /objetos/:id**
    - Datos que enviar: El id del artículo como parámetro en la URL
    - Funcionamiento: El servidor comprueba si el artículo está siendo prestado a un usuario y solo lo borra si no es el caso.
    - Datos devueltos: Mensaje de estado
- **Servicio DELETE /usuarios/:id**
    - Datos que enviar: El id del usuario como parámetro en la URL
    - Funcionamiento: El servidor borra, además del usuario indicado, los préstamos pendientes de ese usuario.
    - Datos devueltos: Mensaje de estado
- **Servicio DELETE /prestamos/:id**
    - Datos que enviar: El id del usuario como parámetro en la URL
    - Funcionamiento: El servidor borra, además del préstamo indicado, las solicitudes de renovación pendientes.
    - Datos devueltos: Mensaje de estado
- **Servicio GET /usuarios/**
    - Datos que enviar: Nada
    - Funcionamiento: El servidor recoge todos los usuarios de la base de datos
    - Datos devueltos: Array JSON de todos los usuarios
- **Servicio GET /usuarios/:id**
    - Datos que enviar: El id del usuario como parámetro en la URL
    - Funcionamiento: El servidor busca el usuario al que corresponde ese id
    - Datos devueltos: El usuario al cual corresponde ese id
- **Servicio GET /objetos/**
    - Datos que enviar: Nada
    - Funcionamiento: El servidor devuelve todos los artículos de la base de datos
    - Datos devueltos: Array JSON de todos los artículos

- **Servicio GET /objetos/:id**
    - Datos que enviar: El id del artículo como parámetro en la URL
    - Funcionamiento: El servidor busca el artículo al que corresponde ese id
    - Datos devueltos: El artículo al cual corresponde ese id
- **Servicio GET /objetossinprestar/**
    - Datos que enviar: Nada
    - Funcionamiento: El servidor busca todos los artículos que no están siendo prestados
    - Datos devueltos: Array JSON con los artículos que cumplen la condición
- **Servicio GET /objetosprestados/**
    - Datos que enviar: Nada
    - Funcionamiento: El servidor busca todos los artículos que están siendo prestados
    - Datos devueltos: Array JSON con los artículos que cumplen la condición
- **Servicio GET /prestamos/**
    - Datos que enviar: Nada
    - Funcionamiento: El servidor devuelve todos los préstamos en la base de datos
    - Datos devueltos: Array JSON de todos los préstamos
- **Servicio GET /prestamos/:id**
    - Datos que enviar: El id del préstamo como parámetro en la URL
    - Funcionamiento: El servidor busca el préstamo al que corresponde ese id
    - Datos devueltos: El préstamo al cual corresponde ese id
- **Servicio GET /prestamosdeusuario/:id**
    - Datos que enviar: El id de un usuario como parámetro en la URL
    - Funcionamiento: El servidor busca en la base de datos todos los préstamos realizados por el usuario al que corresponde ese id
    - Datos devueltos: Un array JSON con todos los préstamos realizados por ese usuario
- **Servicio GET /solicitudes/:id**
    - Datos que enviar: El id de una solicitud de renovación como parámetro en la URL
    - Funcionamiento: El servidor busca la solicitud de renovación al que corresponde ese id
    - Datos devueltos: La solicitud de renovación a la que corresponde ese id en formato JSON

- **Servicio GET /solicitudesdeprestamo/:id**
    - Datos que enviar: La id de un préstamo como parámetro en la URL
    - Funcionamiento: El servidor busca las solicitudes de renovación del préstamo al que corresponde ese id
    - Datos devueltos: Array JSON con las solicitudes de renovación que cumplan esa condición
- **Servicio GET /solicitudesdeusuario/:id**
    - Datos que enviar: La id de un usuario como parámetro en la URL
    - Funcionamiento: El servidor busca las solicitudes de renovación realizadas por el usuario al que corresponde ese id
    - Datos devueltos: Array JSON con las solicitudes de renovación que cumplan esa condición
- **Servicio GET ***
    - Datos que enviar: URL a la que se quiere acceder
    - Funcionamiento: Este servicio permite acceder a las vistas del *front-end.* Si no estuviera, nos sale un error CANNOT GET.
    - Datos devueltos: El archivo index.html dentro de la carpeta public de la aplicación
- **Servicio POST /modificararticulo**
    - Datos que enviar:

      {
      "_id": id del artículo,
      "[atributo a cambiar]": [nuevo valor]
      }
    - Funcionamiento: El servidor actualiza el artículo correspondiente con el nuevo valor del atributo indicado.
    - Datos devueltos: Mensaje de estado
- **Servicio POST /modificarprestamo**
    - Datos que enviar:

      {
      "_id": id del préstamo,
      "[atributo a cambiar]": [nuevo valor]
      }
    - Funcionamiento: El servidor actualiza el préstamo correspondiente con el nuevo valor del atributo indicado.
    - Datos devueltos: Mensaje de estado

- **Servicio POST /modificarusuario**
    - Datos que enviar:

      {

      “_id”: id del usuario,

      “[atributo a cambiar]”: [nuevo valor]

      }
    - Funcionamiento: El servidor actualiza el usuario correspondiente con el nuevo valor del atributo indicado.
    - Datos devueltos: Mensaje de estado
- **Servicio POST /enviarcorreos**
    - Datos que enviar: Nada
    - Funcionamiento: El servidor fuerza el envío del correo de notificación de préstamos a 1 semana de expirar. Este servicio se usó principalmente para probar la función de envío de correos.
    - Datos devueltos: Mensaje de estado
- **Servicio POST /notificarcaducación**
    - Datos que enviar: Nada
    - Funcionamiento: El servidor fuerza el envío del correo de notificación de préstamos expirados. Este servicio se usó principalmente para probar la función de envío de correos
    - Datos devueltos: Mensaje de estado

4 RESULTADOS

Para comprobar el correcto funcionamiento de la aplicación, se han hecho una serie de pruebas que se comentarán en los siguientes apartados.

4.1 Comprobación del manejo correcto a nivel de usuario

En este apartado se encuentran las pruebas directamente relacionadas con las funciones principales de la aplicación en manos de un usuario cliente que maneje con éxito la aplicación.

4.1.1 Registro de un usuario nuevo

Primero, vamos a comprobar el proceso de creación de un nuevo usuario. Una vez desplegado el sistema, se puede registrar de dos maneras; o bien el usuario se registra desde la pantalla de registro o el administrador crea el perfil desde su vista. Ambas utilizan el mismo formulario así que comprobaremos el primer registro. Nos conectamos a la aplicación desplegada conforme el anexo I y nos registramos. Para el correo, utilizaremos un correo desechable:

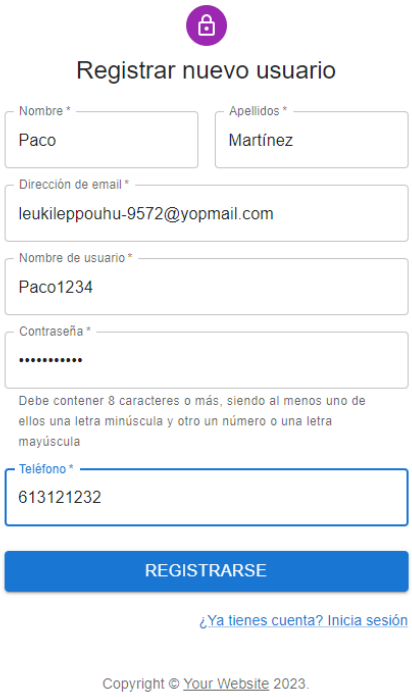

Ilustración 5: Formulario de registro

Una vez le damos a registrarse, nos devuelve con éxito a la vista de cliente esperada:

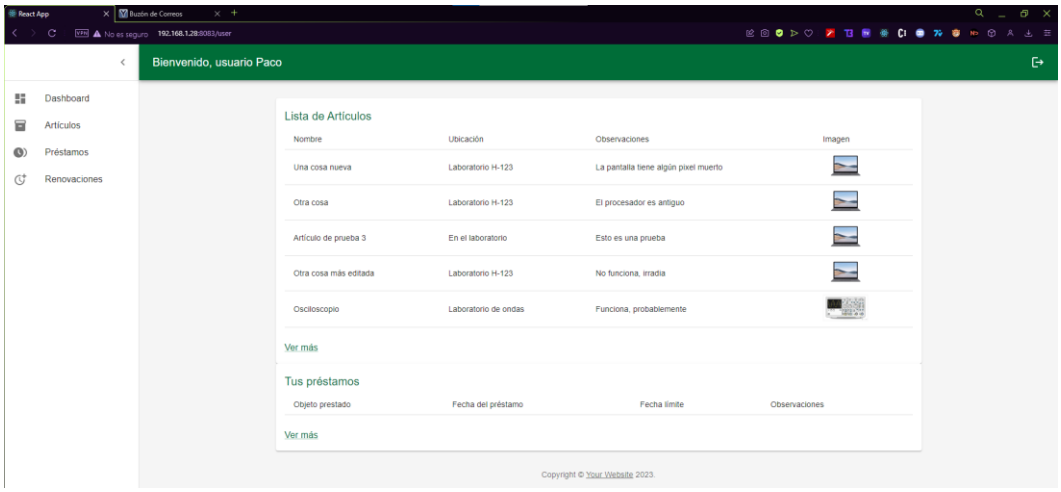

Ilustración 6: Vista general de usuario tras iniciar sesión

En inspeccionar elemento, podemos observar que el usuario tiene su JWT token en el almacenamiento local (no confundir con una *cookie*):

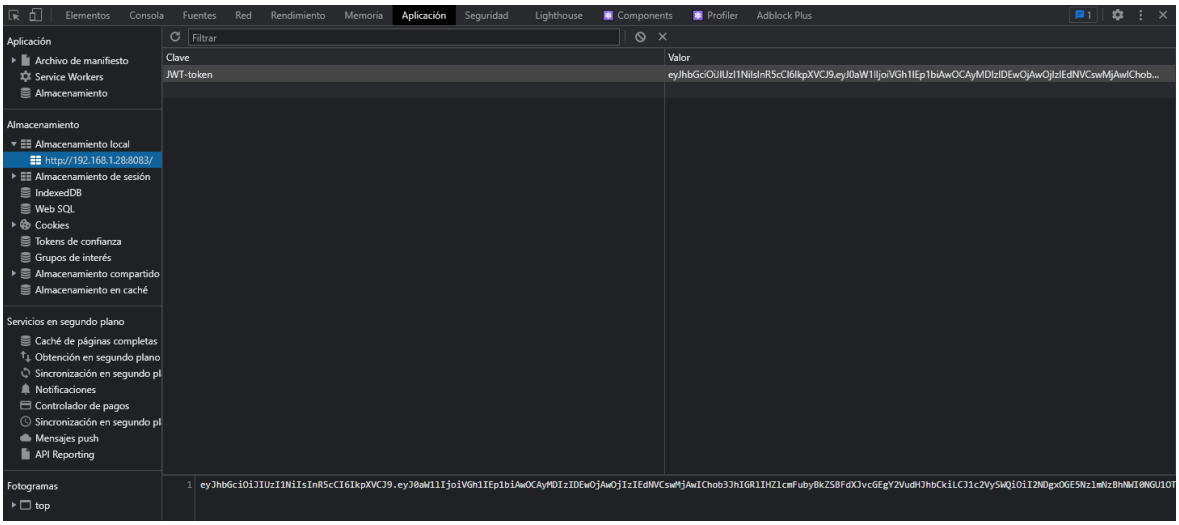

Ilustración 7: Menú de Inspeccionar elemento para ver que el token se guarda en el almacenamiento

Desde la vista de usuarios, el administrador también puede comprobar que el usuario se ha registrado con éxito.

Ilustración 8: Vista de usuarios del administrador utilizando otro dispositivo

Se aprovecha esta prueba para comprobar además que el sistema funciona con naturalidad incluso cuando no hay información que mostrar. Como podemos observar, los campos en los que se muestran las reservas funcionan con normalidad incluso cuando nuestro usuario no ha hecho ninguna:

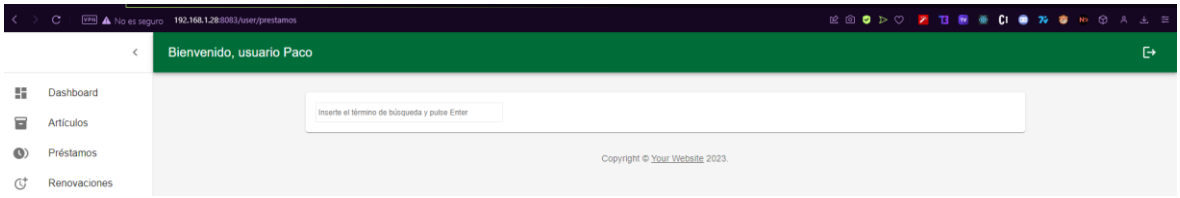

Ilustración 9: Vista de renovaciones del nuevo usuario

4.1.2 Realizar un préstamo de un artículo

Ahora el usuario quiere reservar un artículo. Para ello, se irá a la pestaña de artículos.

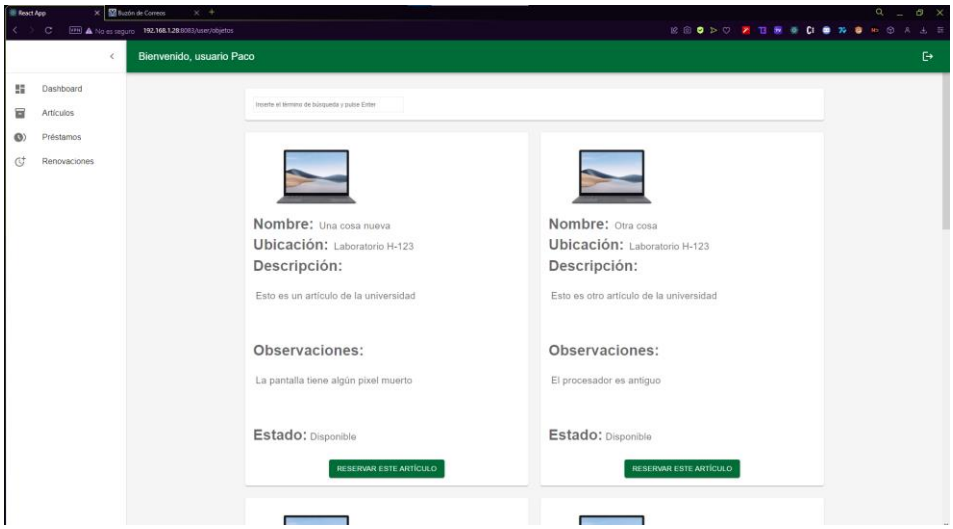

Ilustración 10: Vista de artículos de la aplicación web

Supongamos que quiere un osciloscopio, así que lo buscará en la barra de búsqueda:

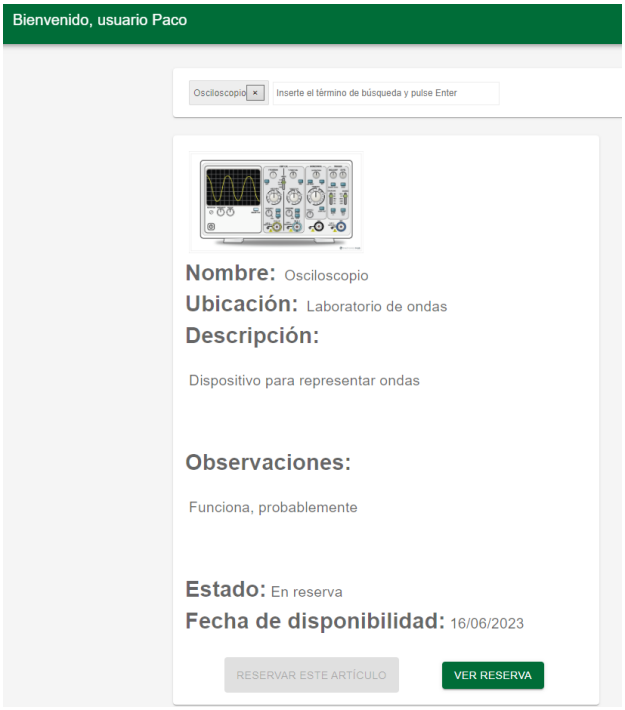

Ilustración 11: Demostración del funcionamiento de la barra de búsqueda

Desafortunadamente, el objeto ya está siendo prestado a otro usuario, como muestra la vista. Si quiere ver la reserva, al no ser suya, no tendrá más información importante.

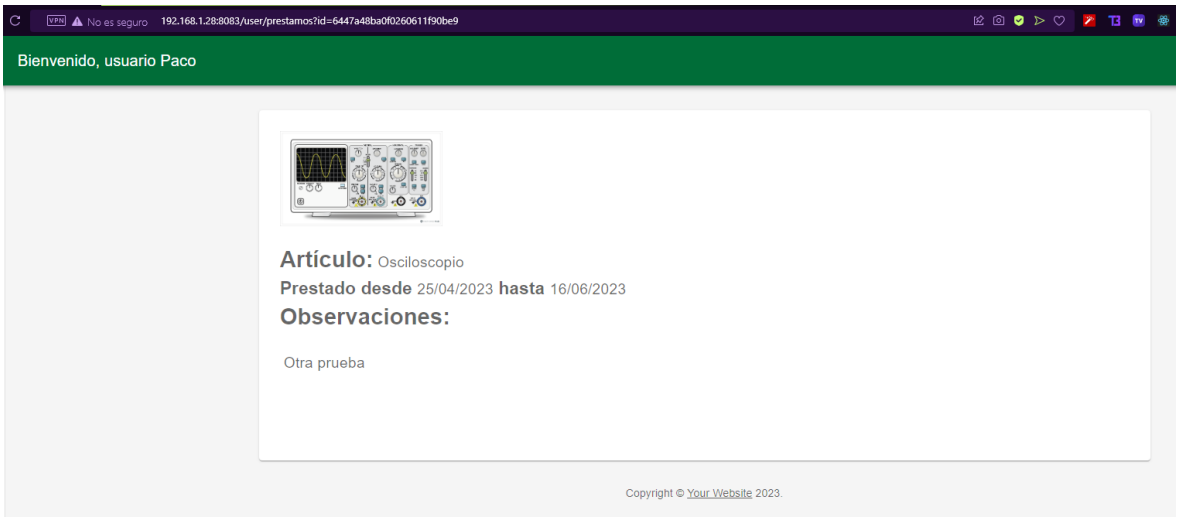

Ilustración 12: Datos de la reserva de otro usuario

Así que al final va a reservar otro objeto. Al hacer clic en el botón “Reservar este artículo” en cualquier objeto disponible, le saldrá un formulario donde tiene que poner la fecha hasta la que lo quiere reservar y observaciones respecto al estado del objeto en el momento de la reserva.

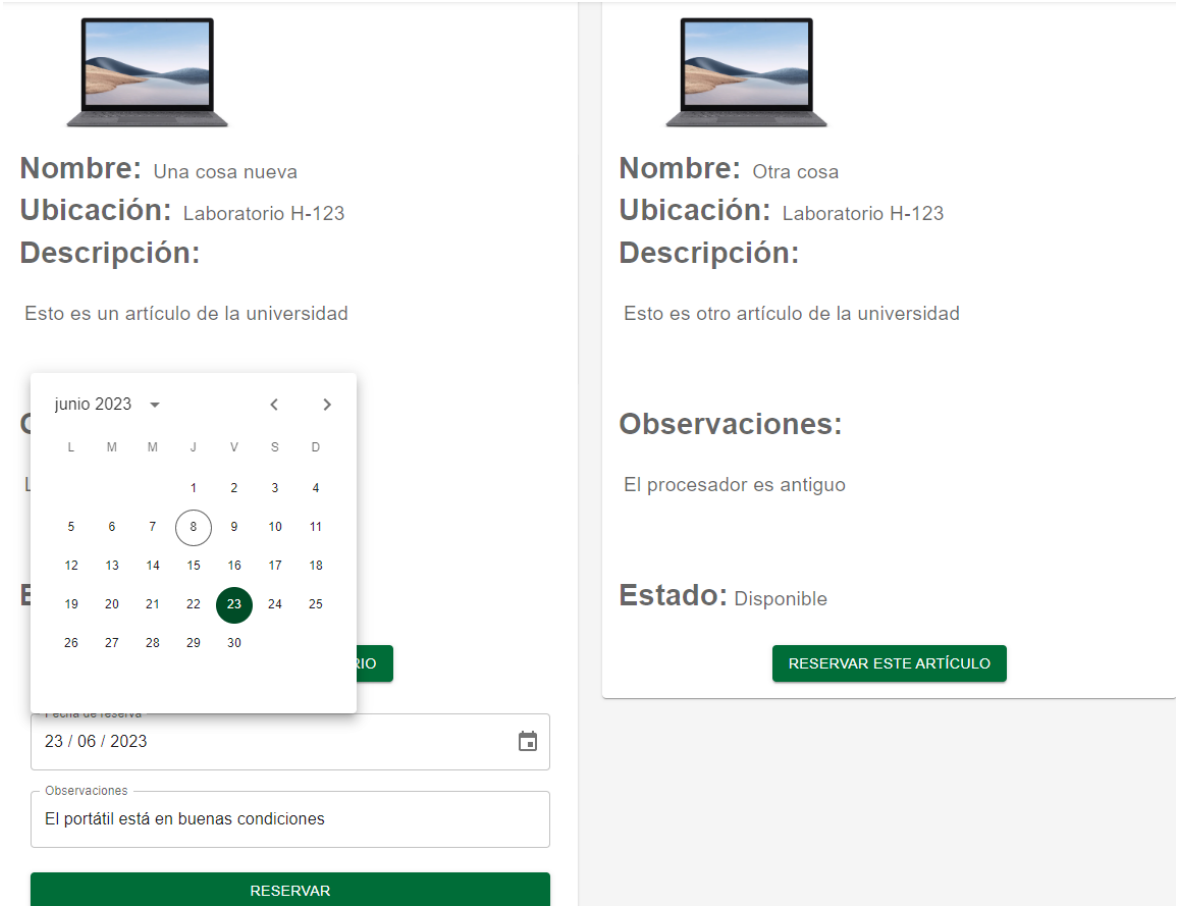

Ilustración 13: Formulario de reserva de artículo

La alerta confirma al usuario que el préstamo se ha hecho con éxito:

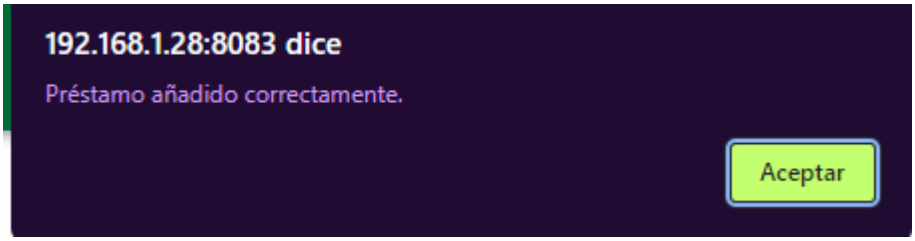

Ilustración 14: Alerta de confirmación de la reserva

Inmediatamente, se actualiza la vista y el artículo aparece en estado de reserva:

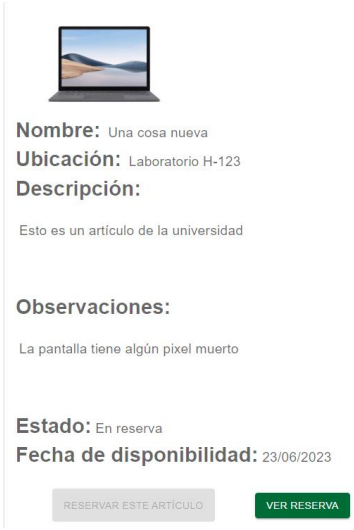

Ilustración 15: Artículo en estado de reserva

El usuario puede acceder al estado del préstamo desde el botón “Ver Reserva” debajo del artículo o desde la vista de préstamos:

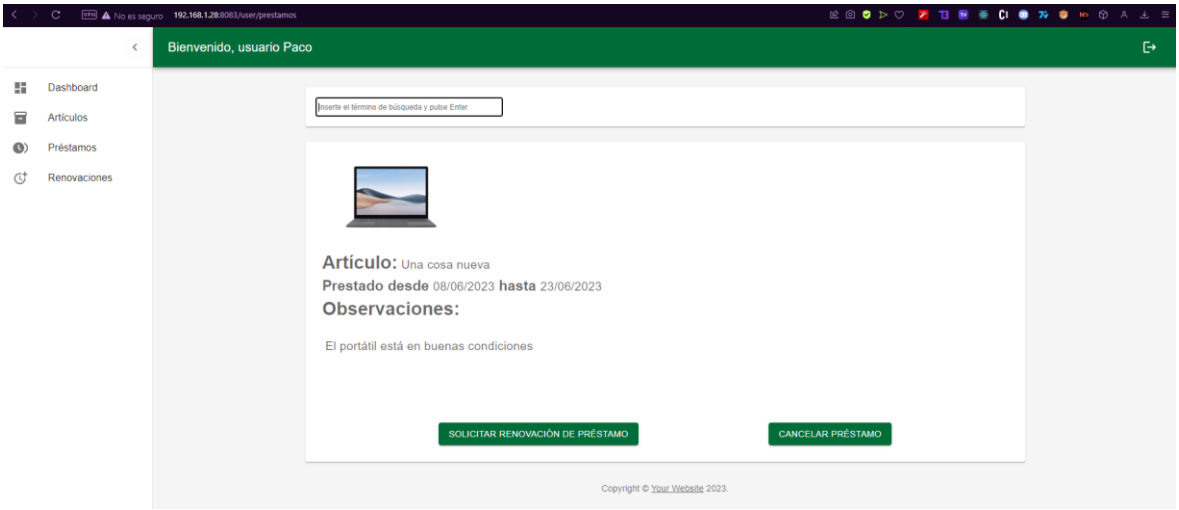

Ilustración 16: Vista de un préstamo específico del usuario de la sesión actual

Si quiere cancelarlo, puede hacer clic en Cancelar préstamo, donde se le preguntará para confirmarlo:

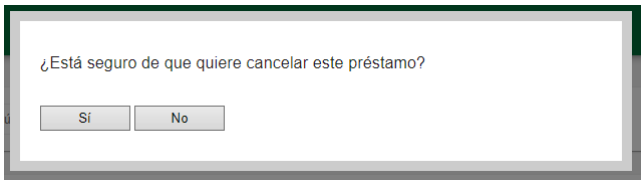

Ilustración 17: Alerta personalizada en la que el usuario confirma si quiere cancelar ese préstamo

Si dice que sí, se le confirmará el borrado del préstamo y se actualizará la vista acorde:

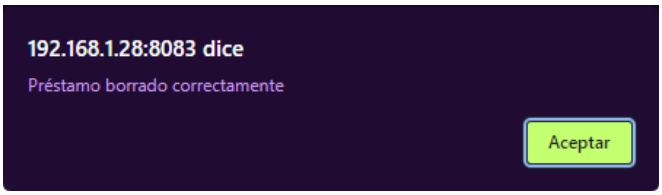

Ilustración 18: Alerta de confirmación de que el préstamo se ha borrado

4.1.3 Solicitar una renovación

Supongamos que en su lugar el usuario quiere solicitar una renovación de su préstamo. Para ello, le tendrá que dar al botón de solicitar renovación, donde se abrirá un formulario en el que tiene que introducir el motivo de renovación y la nueva fecha de final del préstamo.

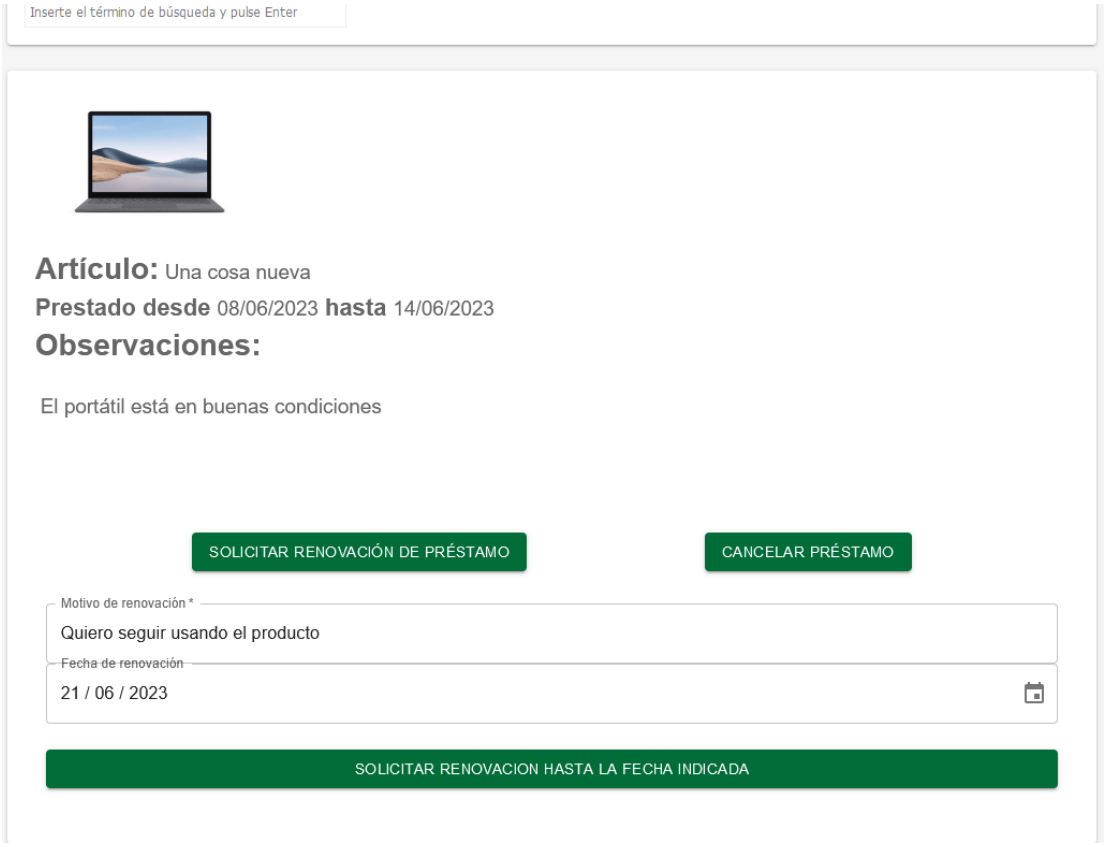

Ilustración 19: Formulario de solicitud de renovación

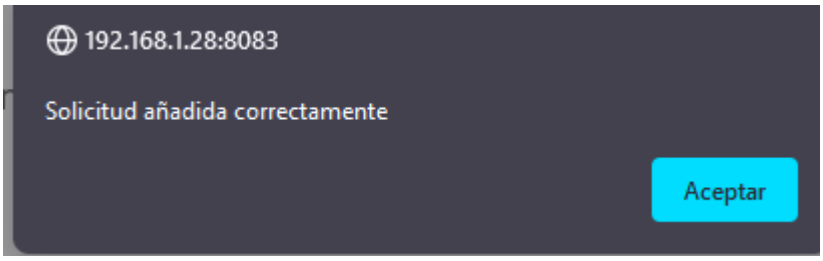

Ilustración 20: Alerta de confirmación de solicitud

El usuario puede, en cualquier momento, observar sus solicitudes por verificar en la vista de renovaciones:

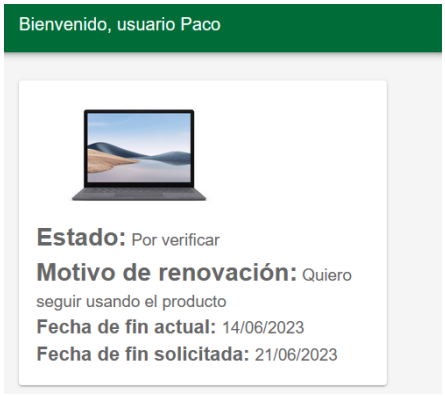

Ilustración 21: Vista de renovaciones del usuario

Al administrador se le notifica dentro de la aplicación de que hay solicitudes de renovación pendientes:

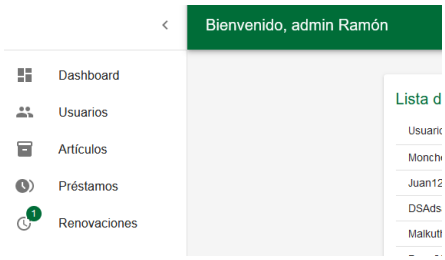

Ilustración 22: Imagen de la notificación de renovación pendiente dentro de la aplicación

En la vista de renovaciones, el administrador podrá elegir si renovar la solicitud o no.

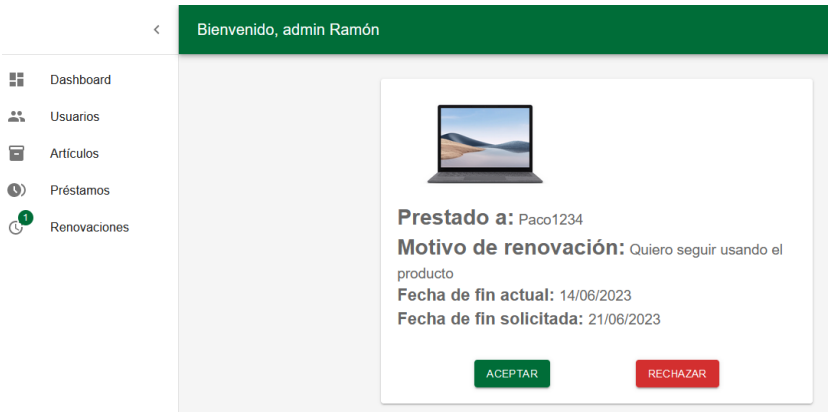

Ilustración 23: Vista de renovaciones del administrador

En ambos casos, se enviará un correo que informe del estado de la petición al usuario. Si, por ejemplo, ha sido aceptada:

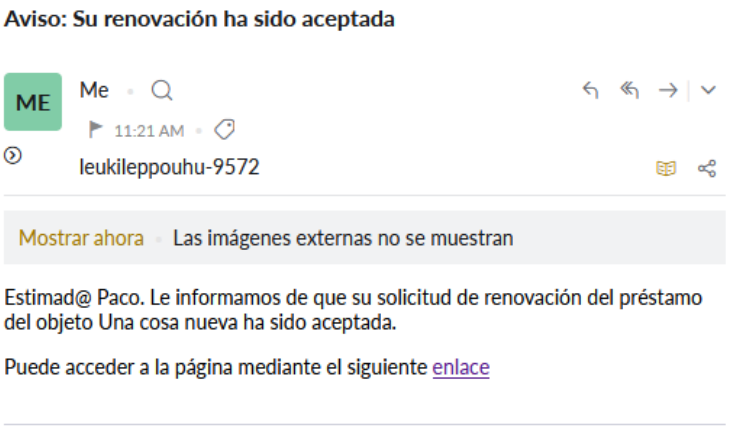

Ilustración 24: Correo en el que se notifica de la resolución de la solicitud de renovación

Y como se puede observar, el objeto tiene la nueva fecha de renovación:

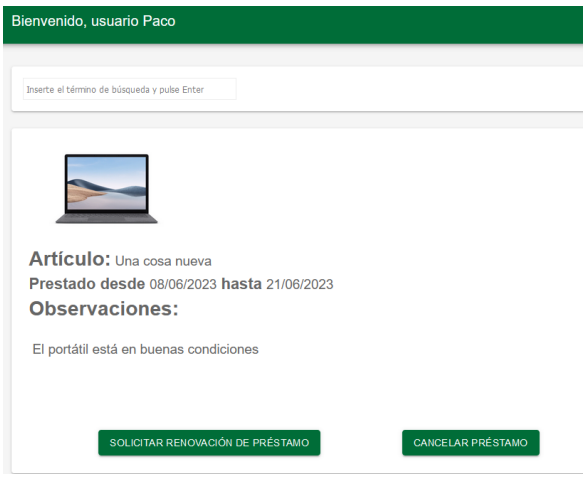

Ilustración 25: Reserva de artículo con la fecha renovada

4.2 Comprobación de mensajes

La aplicación usa el servicio de correo para notificar a sus usuarios de ciertos eventos. Estos eventos son:

- Cuando un préstamo está a una semana o menos de caducar
- Cuando un préstamo ha caducado
- Cuando se ha respondido a una solicitud de renovación

La aplicación hace comprobaciones cada mediodía para enviar los dos primeros tipos de correos, pero se puede forzar por medio de una petición, cosa que se ha utilizado durante el periodo de pruebas.

Dicho correo tiene un enlace que permite al usuario acceder a la aplicación. En los dos primeros tipos de correo, este enlace lleva a una vista especial, mientras que, en el último tipo, envía a la página principal de la aplicación. A continuación, se observa la vista especial:

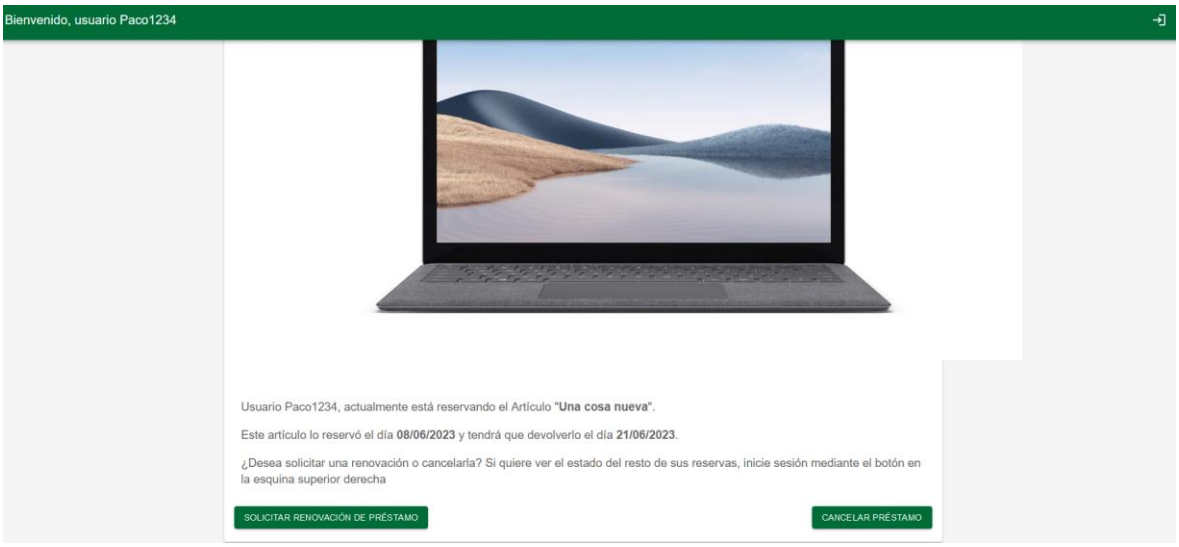

Ilustración 26: Vista de la aplicación cuando se accede desde el enlace de una notificación de préstamo

Durante el proceso de pruebas, se ha confirmado el envío de mensajes de los tres tipos:

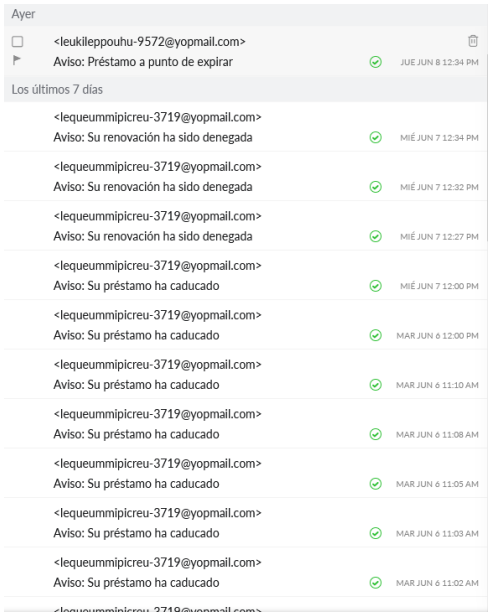

Ilustración 27: Bandeja de mensajes de correo enviados desde la aplicación

4.3 Comprobación del manejo correcto a nivel de administrador

También tenemos que comprobar el uso de las herramientas que utilizará el administrador para monitorizar la aplicación. Para evitar redundancia, no hablaremos de la funcionalidad ya cubierta en el anterior apartado o cuyo procedimiento es exactamente el mismo (el proceso de inicio de sesión, por ejemplo, es igual).

4.3.1 Añadir nuevos elementos a la base de datos

El método de añadir nuevos usuarios es por medio de un formulario exactamente igual al del cliente:

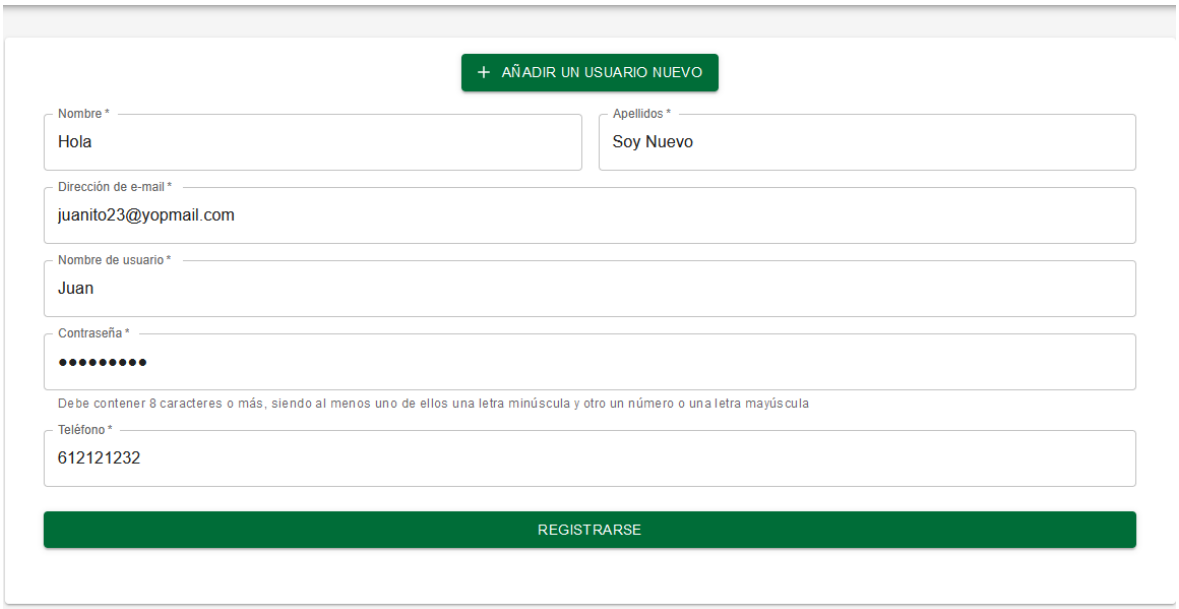

Ilustración 28: Formulario de inscripción de un nuevo usuario desde la vista de administrador

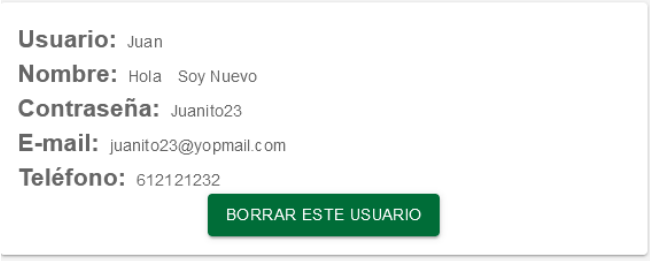

Ilustración 29: Usuario registrado

Para añadir nuevos artículos, se utiliza un formulario especial en la vista de artículos:

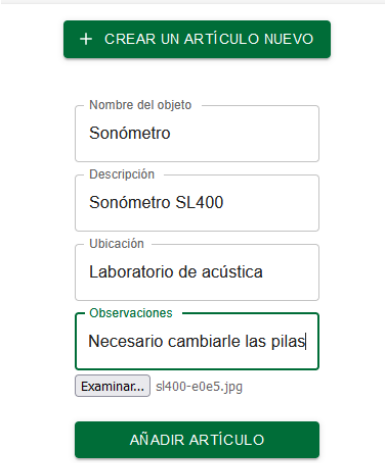

Ilustración 30: Formulario para añadir un nuevo artículo

El campo de archivo acepta cualquier archivo de imagen. Con una modificación al tamaño máximo de la petición (por defecto es 100kb), se puede enviar la imagen con éxito. La aplicación nos indica que el archivo se ha introducido correctamente:

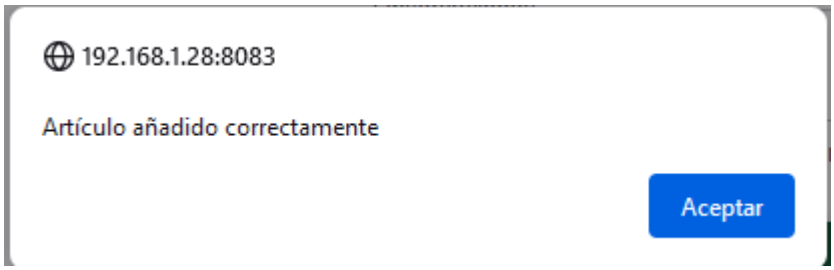

Ilustración 31: Alerta que confirma que el artículo se ha añadido con éxito

Ilustración 32: Artículo añadido recientemente

Si queremos reservar un artículo para otro usuario, tenemos un formulario para ello en la vista de artículos de administrador:

Ilustración 33: Formulario de administrador en el que se reserva un artículo

4.3.2 *Editar contenido*

Casi todos los campos de texto donde se muestra información de artículos o usuarios son editables y el contenido se actualiza debidamente.

4.4 Comprobación de uso indebido o escenarios poco convencionales

Si bien es intuitivo comprobar el uso debido de la aplicación, ninguna aplicación se utilizará perfectamente en todos los casos. Es por tanto importante verificar que, incluso cuando se producen errores, la aplicación funciona correctamente y es capaz de lidiar con esos errores de manejo sin colapsar.

4.4.1 *Introducción de información equivocada*

El error más común es introducir información equivocada o que no cumple con el formato esperado. A continuación, se mostrarán ejemplos de puntos donde esto puede ocurrir y cómo reacciona el sistema:

4.4.1.1 *Errores en inicio de sesión y registro*

Si se insertan credenciales erróneas en el inicio de sesión, la aplicación devuelve la siguiente notificación emergente o *snackbar* en la esquina inferior izquierda:

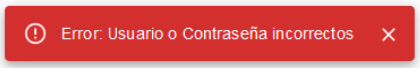

Ilustración 34: Snackbar que aparece cuando se introducen credenciales incorrectas

En el registro, todos los campos son requeridos, y se tienen las siguientes restricciones:

- El campo de *e-mail* sigue el formato estándar para correos electrónicos así que el formato que se observa es:
  [cadena de caracteres y números]@[cadena de caracteres y números].[al menos 2 caracteres]
  Si no lo es, se avisa al usuario de esta manera:

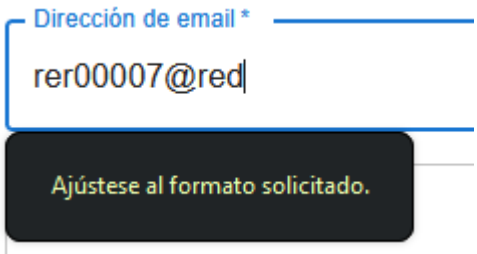

Ilustración 35: Aviso de formato incorrecto

- La contraseña tiene el siguiente formato: Debe contener 8 caracteres o más, siendo al menos uno de ellos una letra minúscula y otro un número o una letra mayúscula. Si no se sigue ese formato, se avisa al usuario de la misma forma

- El nombre de usuario no debe coincidir con ninguno ya existente en la base de datos ya que ha de ser único para cada usuario. Si eso ocurre, el *snackbar* avisa de ello:

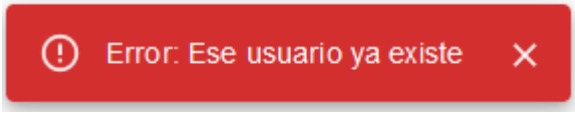

Ilustración 36: Snackbar que aparece cuando el usuario ya existe

- El campo de teléfono se compone siempre de 9 números. Si no es así, se avisa al usuario.

Si el formulario no cumple con las restricciones de formato impuestas, no se envía (excepto en el caso del nombre de usuario porque su comprobación se hace en el *back-end*).

4.4.1.2 Errores al editar texto

Las comprobaciones de formato se hacen incluso cuando se editan usuarios para evitar que se produzcan errores de formato. Los artículos no tienen restricciones respeto al contenido así que el texto no requiere revisión a la hora de editar.

4.4.2 Acceso a páginas no acordes con su rol

Cuando se carga una vista después de registrarse y mientras se actualiza tras hacer ciertas acciones (por ejemplo, un cambio en la base de datos), se comprueba el *token* que se generó en el momento del inicio de sesión. Ese *token* se generó a partir del id de usuario, por lo que descifrándolo en el *back-end* nos permite averiguar si ese usuario es administrador o no.

En el caso de que se acceda a una vista de administrador siendo usuario o viceversa, se alertará al usuario y se redireccionará a la pestaña de inicio de sesión para autenticarse. Esto también ocurre cuando el token expira. Esto es un ejemplo de lo que se ve si un usuario cliente intenta acceder a la base de datos de usuarios antes de ser redirigido a la pantalla:

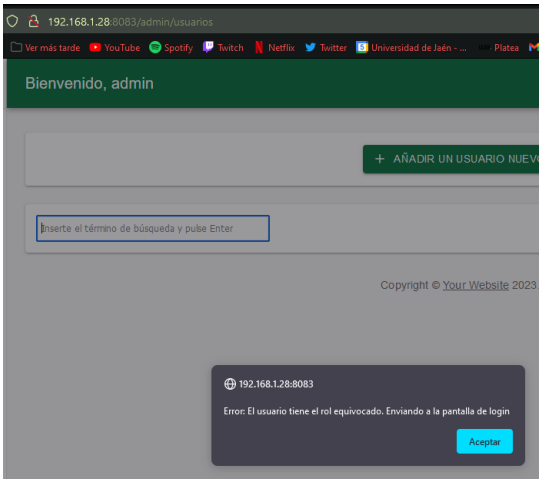

Ilustración 37: Aviso de conflicto de rol

4.4.3 *Renderizado sin elementos en la BD*

Es importante averiguar qué se mostraría si la base de datos está vacía, ya que eso ocurrirá cuando se despliegue la aplicación desde cero. Para averiguar esto, se ha borrado temporalmente los artículos de la base de datos y se ha ejecutado la vista relevante. Estos son los resultados:

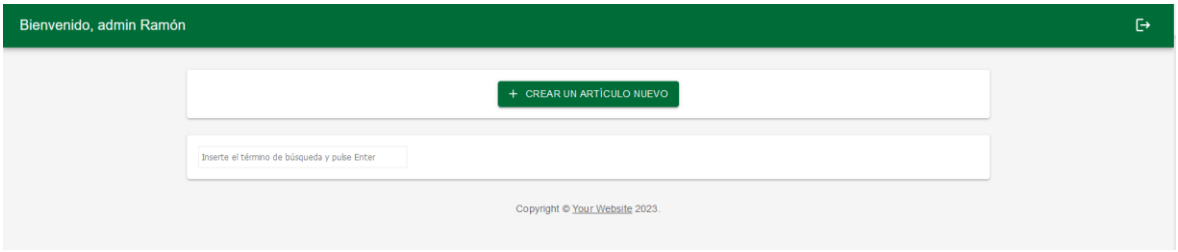

Ilustración 38: Vista de artículos sin artículos

Como se puede observar, la vista funciona aún sin artículos que mostrar, cosa que también se puede observar cuando el filtro no devuelve resultados. Lo mismo se ha podido comprobar en las pruebas de usuario cuando no había renovaciones.

4.4.4 *Introducción de imagen de artículo con nombre igual a otra ya introducida*

Una posible preocupación es qué pasaría si una imagen de artículo tiene el mismo nombre que otra ya introducida antes, ya que eso causaría conflictos a la hora de acceder a las imágenes. Para evitar esto, el programa renombra la imagen que se va a introducir a un número basado en la fecha de introducción de la imagen. Así, se evita la posibilidad de que haya dos imágenes con el mismo nombre (la imagen prueba.png se usó como elemento provisional o *placeholder* de imagen de artículos antes de implementar este sistema):

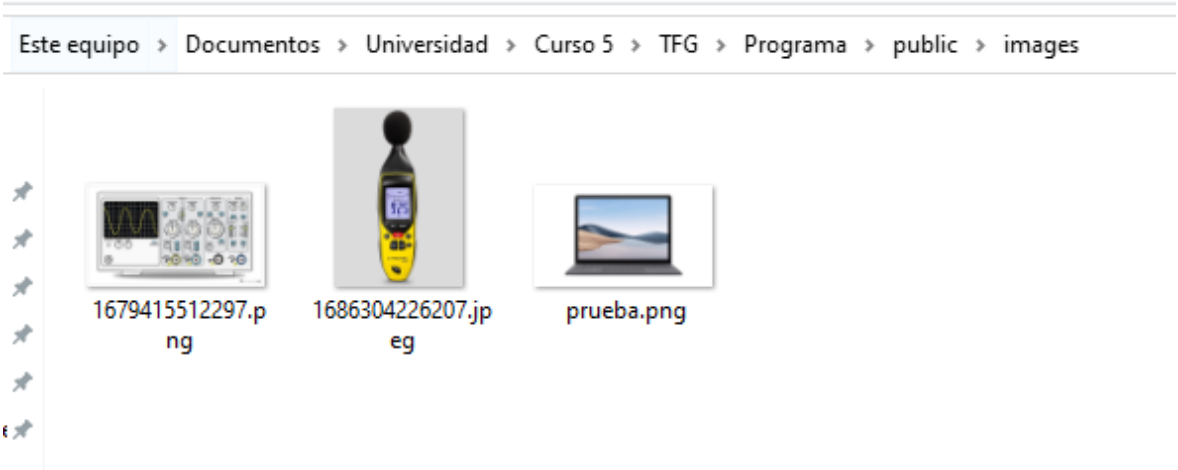

Ilustración 39: Archivos de imágenes de artículos guardados en la aplicación.

5 CONCLUSIONES Y LÍNEAS FUTURAS DE DESARROLLO

En este apartado, se determinará si el trabajo presentado cumple con los objetivos propuestos y se propondrán mejoras o nuevas funcionalidades que el autor considere.

Para empezar, se considera que el trabajo satisface con éxito los objetivos propuestos en el TFG. La aplicación permite la monitorización de préstamos de artículos realizados por usuarios de esta y las solicitudes de renovación de estos, siendo los cuatro tipos de datos mencionados almacenados en una base de datos común, existe una separación entre la vista de usuario cliente y usuario administrador que se ve reflejada en la funcionalidad a la que tiene acceso cada tipo de usuario y se notifica tanto desde la propia aplicación como por correo electrónico del estado de préstamos y solicitudes de renovación.

Además de los objetivos mencionados anteriormente, se ha propuesto como meta complementaria la implementación de funciones de calidad de vida (*Quality of Life*) que faciliten el manejo de la aplicación o la hagan más accesible. Entre estas funciones, se encuentran la capacidad de editar el texto perteneciente a las propiedades de los artículos de manera directa, la capacidad de hacer clic en las imágenes para ampliarlas y la implementación de un sistema de búsqueda por filtros.

Sin embargo, la aplicación no está exenta de vías de mejora. Para empezar, si bien la aplicación es funcional, se puede argumentar que no aprovecha al máximo el potencial de React para ser lo más eficiente posible. La codificación, si bien es capaz de ejecutar las funciones solicitadas con éxito, hay situaciones en las cuales no se aprovecha el carácter modular de React al máximo, y ciertos elementos no son declarados una sola vez e importados donde sean necesarios, sino que están incrustados directamente en el código o *hard-coded* en múltiples vistas que comparten ese elemento. En algunos casos se ha tomado la decisión de optimizar el código, como en la implementación de una paleta de colores universal en todo el documento declarada una sola vez e importada en cada vista en lugar de sobrescribir los colores directamente en cada elemento. En otros, sin embargo, no se ha realizado esta labor de optimización, ya que se consideraba una tarea demasiado ardua requiriendo un posible replanteamiento de gran parte de la aplicación y había tareas que tenían mayor prioridad como la implementación de las funcionalidades.

En algunas vistas, se ha tenido problemas para mostrar el contenido sin salirse del formato previsto. Esto es especialmente notable al ver la aplicación web en una pantalla de móvil. La aplicación web está diseñada principalmente para ser vista en un ordenador, con un esfuerzo relativamente menor puesto en la vista de móvil salvo ciertos puntos. Esto

deja un margen de mejora bastante notable en el proceso de adecuación de la imagen a otras plataformas.

Adicionalmente, el diseño de la pantalla inicial contrasta demasiado con el resto de la página.

Respecto a funcionalidades futuras que se podrían implementar serían:

- Un sistema de verificación de e-mail para verificar la cuenta antes de permitir usarla.
- Un sistema de recuperación/restablecimiento de contraseña.
- Un método mediante el cual se puedan insertar cuentas de administrador desde la aplicación sin tener que hacerlo de manera directa.
- La capacidad de importar múltiples artículos desde un archivo escrito siguiendo un formato específico, como puede ser un archivo CSV.
- Que la aplicación tenga un nombre de dominio asignado en lugar de tener que usar la dirección IP del dispositivo que ejecuta la aplicación.
- Añadir la opción de alternar entre el tema actual y uno más oscuro para la interfaz

6 ANEXOS

6.1 Anexo I: Despliegue

Este anexo describirá la información necesaria para desplegar la aplicación web.

Para el despliegue de la aplicación se requieren los siguientes programas adicionales instalados en el dispositivo:

- La última versión de Node.js, cuya instalación se puede realizar descargándose el instalador desde la página oficial.
- La última versión de MongoDB. Se recomienda instalar MongoDB Compass para facilitar el manejo de la base de datos mediante una interfaz

Una vez realizado, se utilizará la consola de comandos para iniciar la aplicación. Para ello, nos vamos al directorio donde esté alojada la aplicación e introducimos el comando *node main.js*.

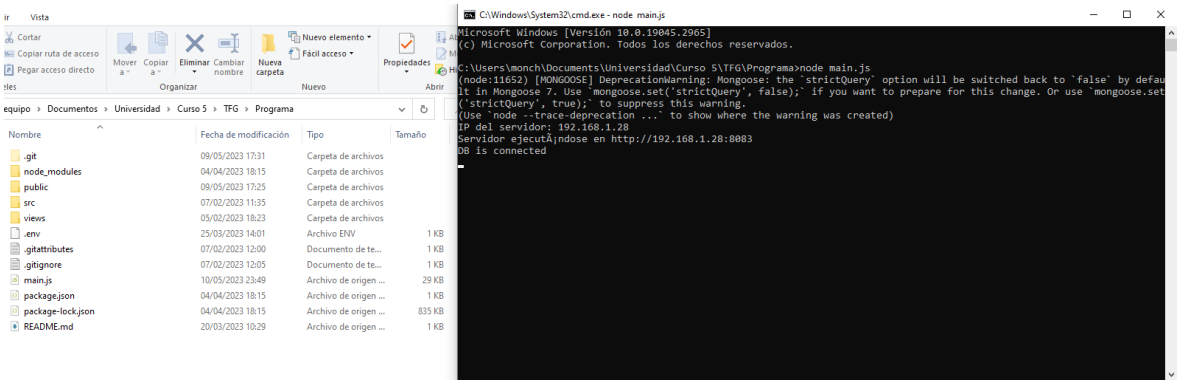

Ilustración 40: Imagen de la aplicación web siendo ejecutada

Esto abrirá el puerto 8083 para permitir a cualquier dispositivo conectarse a la aplicación. Si introducimos la dirección indicada en un navegador web, el resultado debería ser el siguiente:

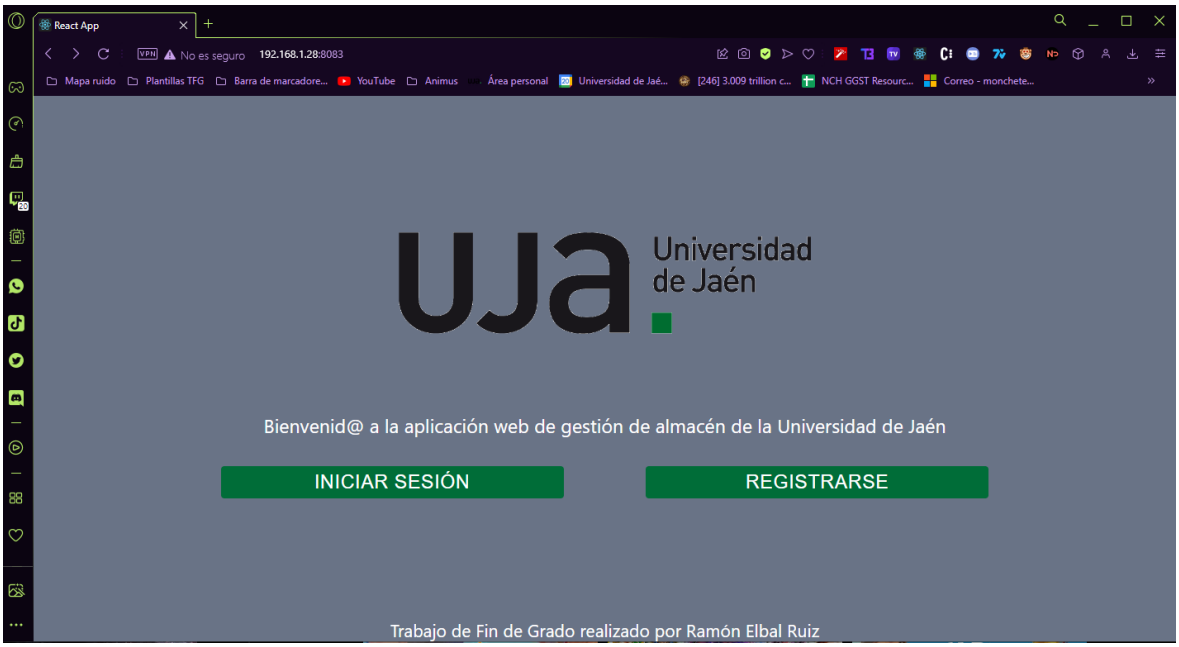

Ilustración 41: Página principal de la aplicación web

6.2 Anexo II: Manual de usuario

Este anexo tiene como función guiar al usuario, ya sea cliente o administrador, en la navegación por la aplicación web, así como solucionar posibles problemas que puedan surgir en el manejo de información.

6.2.1 Inicio de sesión/Registro

Las cuentas de usuario en esta aplicación son independientes de las cuentas de la universidad de Jaén. Cualquier usuario nuevo que quiera utilizar el servicio debe registrarse haciendo clic en el botón homónimo de la vista inicial. Se pedirán los siguientes campos (todos ellos obligatorios):

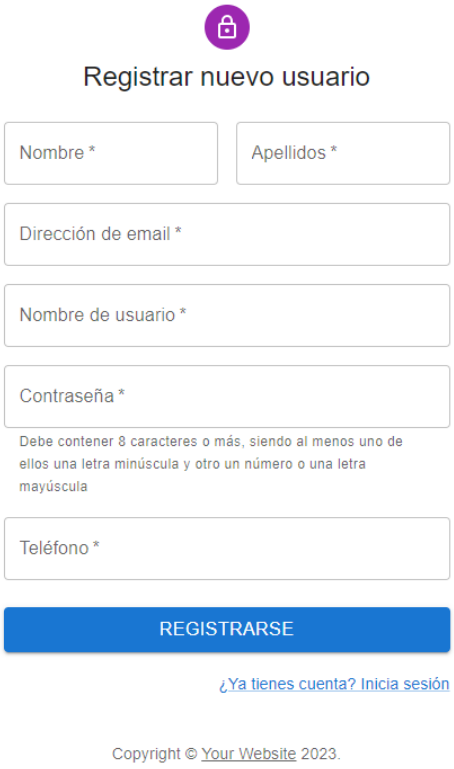

Ilustración 42: Formulario de registro de usuario de la página web

El nombre de usuario y la contraseña serán utilizados para iniciar sesión. Se recomienda que la dirección de email sea válida para que el sistema pueda enviar notificaciones. Todos los usuarios creados por este método serán usuarios cliente. Para crear usuarios de rol administrador, se ha de introducir directamente en la base de datos.

Una vez iniciada la sesión, se procederá a la vista general, donde se puede observar los datos más relevantes para el usuario según su rol. En cada vista, se realizará una autenticación de usuario. En el caso de que la autenticación no se lleve a cabo con éxito, el usuario será redirigido a la pantalla de inicio de sesión.

6.2.2 Para administradores

6.2.2.1 Vista general para administradores

A continuación, se muestra la vista general de administrador, a la que se accederá una vez realizado el inicio de sesión:

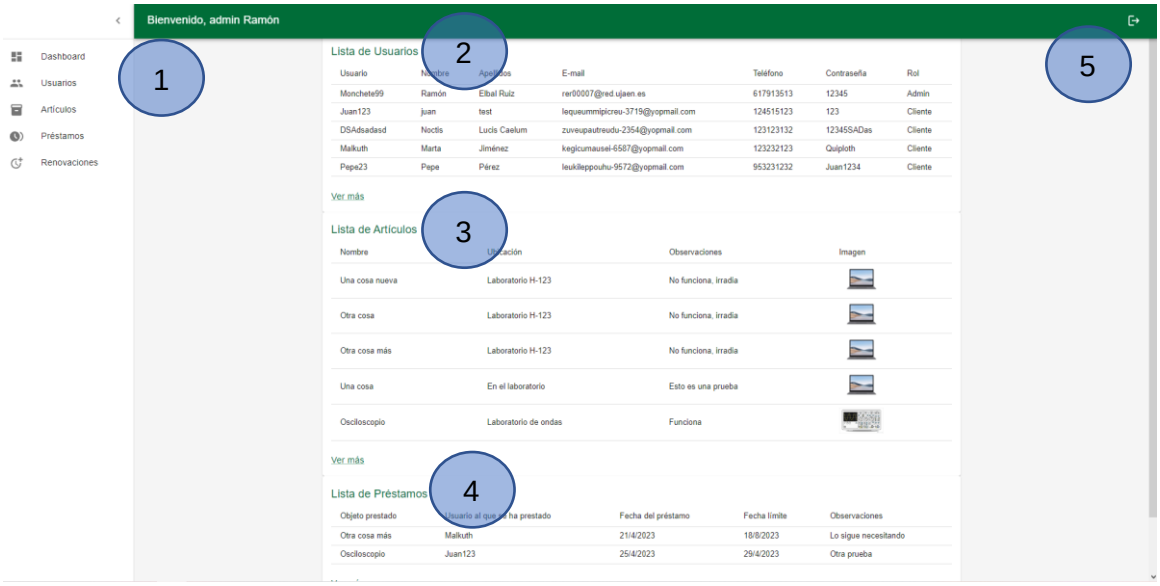

Ilustración 43: Vista general de usuario administrador

1. Barra lateral con enlaces a vistas para cada elemento. Este objeto se encuentra en casi todas las vistas de la aplicación. Cuando haya renovaciones por verificar, el icono de renovaciones lo mostrará.
2. Lista de todos los usuarios con sus datos
3. Lista de artículos con sus datos
4. Lista de préstamos realizados
5. Botón de cierre de sesión

6.2.2.2 Vista de usuarios

En esta vista exclusiva para administradores se pueden añadir, eliminar o modificar los usuarios de la aplicación web.

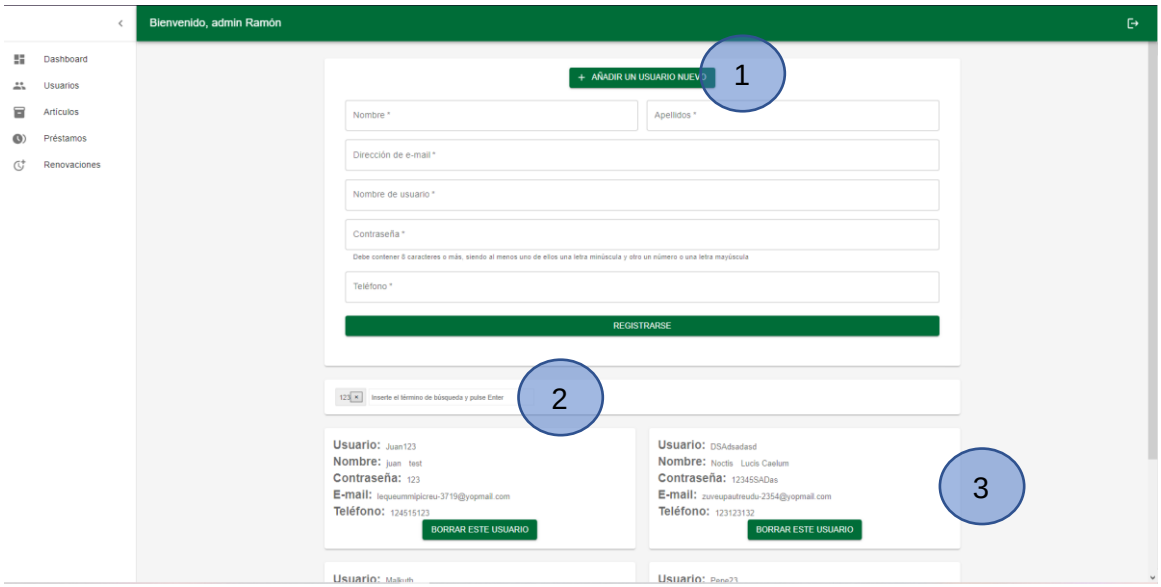

Ilustración 44: Vista en la que se observan los usuarios registrados.

Para añadir un usuario, se utiliza el formulario desplegable (1), que tiene el mismo formato y restricciones que el de la pantalla de registro.

Esta vista y las demás cuenta con un sistema de filtrado por términos de búsqueda (2). Una vez generada, solo se mostrarán elementos que contengan la secuencia de caracteres en dicha etiqueta. Para eliminarla, se ha de hacer clic en la X en la etiqueta.

Los datos de cada usuario aparecen en su respectivo contenedor (3). Si queremos editar la información, tan solo tenemos que hacer clic en el campo de texto que queremos editar, añadir la nueva información y hacer clic fuera del campo de texto. Para eliminar el usuario, tan solo tenemos que hacer clic en el botón del contenedor correspondiente.

6.2.3 Para usuarios

6.2.3.1 Vista general para usuarios

A continuación, se muestra la vista general de usuario, a la que se accederá una vez realizado el inicio de sesión:

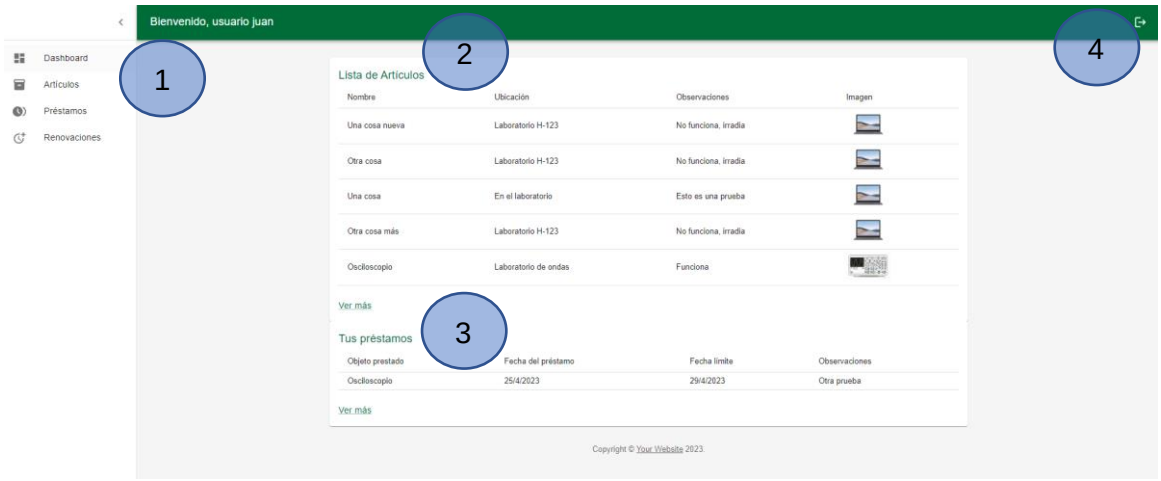

Ilustración 45: Vista general de usuario cliente

1. Barra lateral con enlaces a vistas para cada elemento accesible para el usuario. Este objeto se encuentra en casi todas las vistas de la aplicación. Cuando haya renovaciones por verificar, el icono de renovaciones lo mostrará.
2. Lista de artículos con sus datos
3. Lista de préstamos realizados por el usuario de la sesión
4. Botón de cierre de sesión. Devuelve a la pantalla de inicio de sesión. También se encuentra en casi todas las vistas de la aplicación.

6.2.4 Para usuarios y administradores

6.2.4.1 Vista de artículos

En esta vista, se tiene acceso a los artículos disponibles. Esta vista está compartida entre usuario cliente y usuario administrador, pero con diferentes funciones para cada rol.

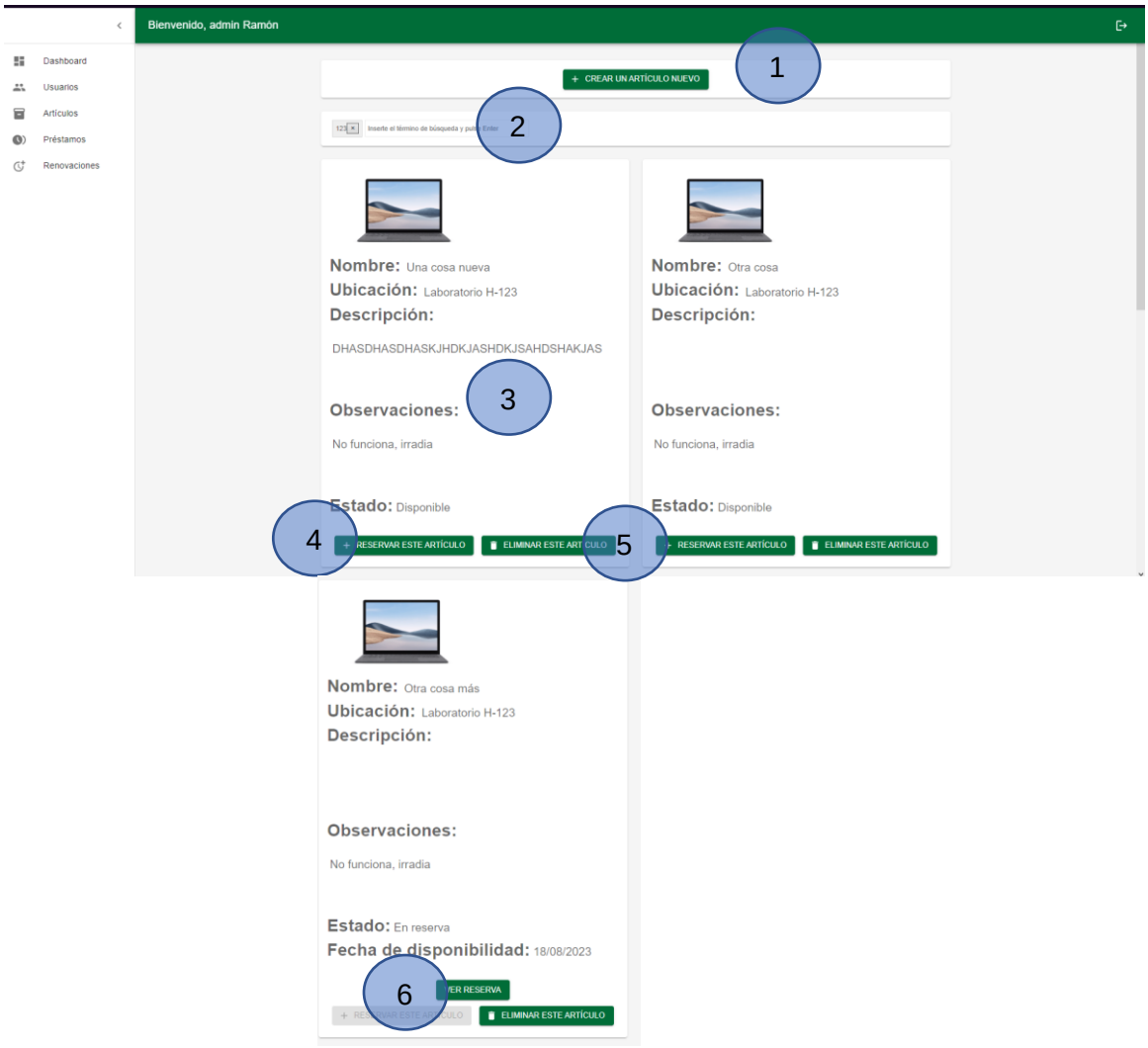

Ilustración 46: Vista de artículos

1) Formulario para añadir nuevos artículos (solo administradores). Este formulario requiere:
    - El nombre del artículo
    - La descripción del artículo (opcional)
    - Observaciones respecto al artículo
    - Archivo de imagen del artículo
2) Filtro de términos de búsqueda común a todas las vistas relevantes. Cada término de búsqueda aparece en forma de etiqueta y se puede eliminar haciendo clic en la X.
3) Contenedor donde se muestran los datos de cada artículo y si están siendo prestados o no. Los administradores pueden editar el texto directamente para modificar los artículos.

4) Botón de reserva de artículo. Se puede introducir una fecha de reserva y el motivo de la reserva. Los administradores pueden asignar la reserva a cualquier usuario. Solo se puede hacer una reserva por artículo.
5) Botón para eliminar el artículo (solo para administradores)
6) Botón para ver los detalles de la reserva existente. Se enviará a una vista donde solo se encuentra ese préstamo y solo se permitirá solicitar una renovación o cancelarlo si es el uno de ese usuario de la sesión o el usuario es un administrador.

6.2.4.2 Vista de préstamos

En esta vista, se encuentran los préstamos de artículos vigentes. Los administradores verán todos los préstamos, pero los clientes solo pueden ver los suyos propios. Cualquier préstamo que esté a 1 semana o menos de caducar (o ya haya caducado) saldrá con un fondo amarillo. A continuación, se muestran las vistas de usuario administrador y de usuario cliente:

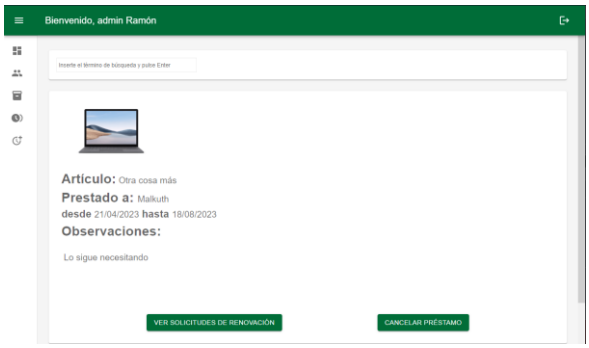

Ilustración 47: Vista de artículos de administrador

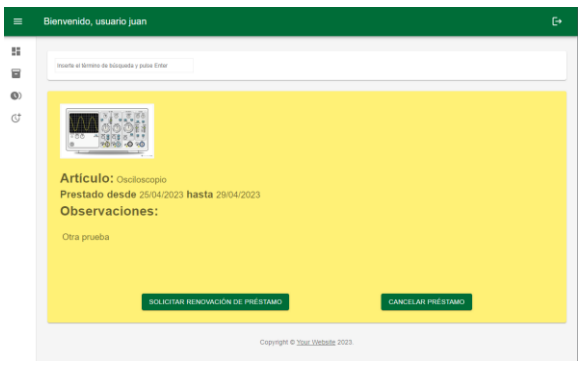

Ilustración 48: Vista de artículos de cliente

Por medio de esta vista, los administradores pueden editar las observaciones de los préstamos, ver las solicitudes de renovación de ese préstamo y cancelarlos. Los clientes pueden solicitar una renovación del préstamo o cancelar sus préstamos. En la solicitud de renovación, se tiene que añadir el motivo de la renovación, así como la nueva fecha límite.

6.2.4.3 *Vista de renovaciones*

En esta vista, se pueden observar las solicitudes de renovación de préstamo vigentes. Los usuarios solo pueden leer las solicitudes que han hecho y tienen pendientes. Los administradores pueden ver las solicitudes de todos los usuarios y elegir si aceptarlas para renovar el préstamo o rechazarlas. Una vez la renovación se haya hecho, se mandará un correo al usuario notificando del resultado con un enlace a la aplicación web.

Ilustración 49: Vista de renovaciones de usuario administrador.

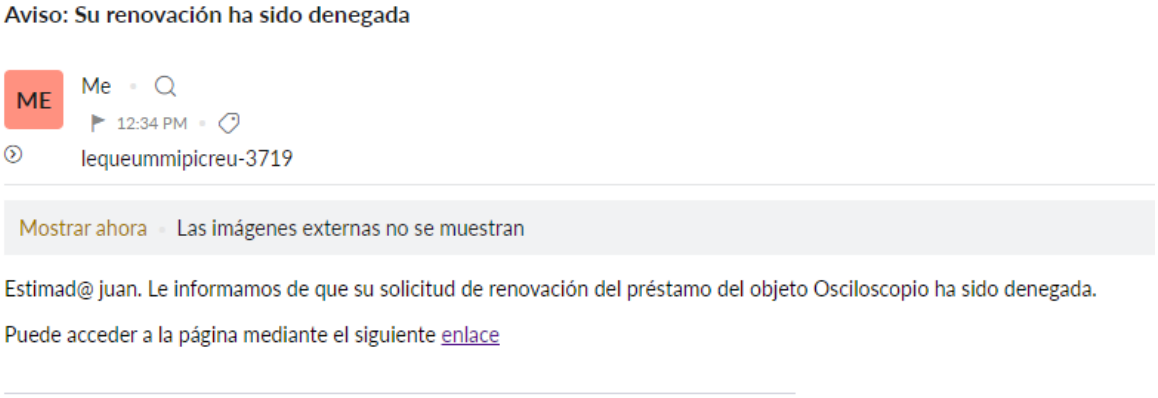

Ilustración 50: Ejemplo de correo de aviso del estado de la solicitud

6.2.4.4 Notificaciones de correo automáticas

La aplicación enviará un correo automáticamente cada mediodía cuando un préstamo está a punto de caducar o ya ha caducado.

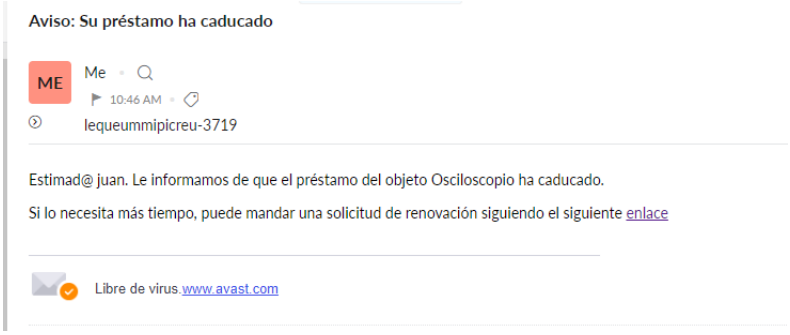

Ilustración 51: Ejemplo de correo automático de aviso

El enlace llevará al usuario a una vista de la aplicación donde se le informará del estado de su préstamo y se le dará la opción de o bien solicitar la renovación del préstamo o cancelarlo. Arriba a la derecha, hay un botón que permite al usuario autenticarse para acceder al resto de la aplicación:

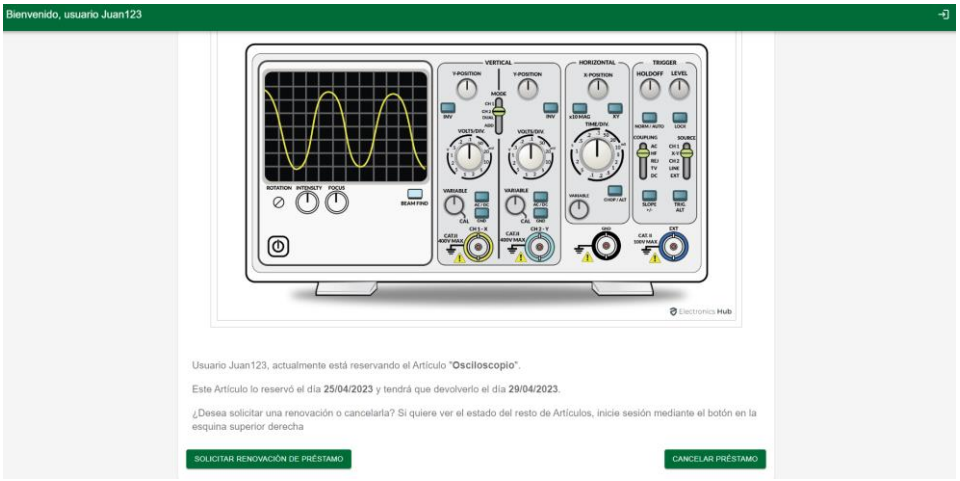

Ilustración 52: Vista en la que se indica al usuario el estado de su reserva

6.2.5 Solución de problemas

6.2.5.1 La aplicación indica que ha habido un error en el proceso de comprobación de token

La sesión ha caducado o se ha entrado a una vista no permitida y hace falta volver a iniciar sesión. También puede ocurrir si se cambia de vista mientras otra vista está comprobando el token demasiado rápido.

6.2.5.2 Hay un artículo duplicado en la vista, uno aparece reservado y otro sin reservar

Un error gráfico a la hora de mostrar los artículos. Intentar reservar el objeto no afecta a la base de datos, pero borrarlo sí. Recargar la página soluciona este error.

7 REFERENCIAS

1. **Stack Overflow.** Stack Overflow Developer Survey 2022. [En línea] 2022. [Citado el: 24 de Mayo de 2023.] https://survey.stackoverflow.co/2022/#section-most-popular-technologies-programming-scripting-and-markup-languages.

2. **Oracle.** Business Software, Business Management Software | NetSuite. [En línea] 2023. [Citado el: 13 de Junio de 2023.] https://www.netsuite.com/portal/home.shtml.

3. **W3Techs.** Usage statistics and market share of WordPress. [En línea] 2023. [Citado el: 24 de Mayo de 2023.] https://w3techs.com/technologies/details/cm-wordpress.

4. **Microsoft.** Visual Studio Code . Code Editing Redefined. [En línea] 2023. https://code.visualstudio.com/.

5. **Github Inc.** Github: Let's build from here. [En línea] 2023. https://github.com.

6. **Postman Inc.** Postman API Platform | Sign Up for Free. [En línea] 2023. [Citado el: 2 de Abril de 2023.] https://www.postman.com.

7. **Zoho Corporation Pvt. Ltd.** Email Hosting | Secure Business Email for your organization - Zoho Mail. [En línea] 2023. [Citado el: 25 de Marzo de 2023.] https://www.zoho.com/mail/.

8. **MongoDB Inc.** MongoDB: The Developer Data Platform | MongoDB. [En línea] 2023. [Citado el: 7 de Febrero de 2023.] https://www.mongodb.com/.

9. **OpenJS Foundation.** Node.js. [En línea] 2023. [Citado el: 2 de Abril de 2023.] https://nodejs.org/en.

10. **MIT.** Mongoose ODM v7.2.2. [En línea] 2023. [Citado el: 14 de Marzo de 2023.] https://mongoosejs.com.

11. **Reinman, Andris.** Nodemailer :: Nodemailer. [En línea] 2023. [Citado el: 25 de Marzo de 2023.] https://nodemailer.com/about/.

12. **IETF.** JSON Web Tokens - jwt.io. [En línea] 2023. [Citado el: 25 de Marzo de 2023.] https://jwt.io.

13. **npm, Inc.** npm. [En línea] 2023. [Citado el: 21 de Marzo de 2023.] https://www.npmjs.com.

14. **Meta.** React. [En línea] 2023. [Citado el: 31 de Marzo de 2023.] https://react.dev.

15. **Material UI SAS.** MUI: The React component library you always wanted. [En línea] 2023. [Citado el: 12 de Marzo de 2023.] https://mui.com.