**Aplicaciones Interactivas**

**TRABAJO PRÁCTICO OBLIGATORIO**

**“PapisFelices”**

**Integrantes del Grupo 1:**

Brea Emanuel *– LU* 1127545

**Profesores:**

*Sarasa Maria Paula*

*Malio Tomas Horacio*

*Ginabreda Santiago Jorge*

**2° Cuatrimestre 2021**



**UNIVERSIDAD ARGENTINA DE LA EMPRESA**

**FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS**

Contenido

[Requerimientos Generales 2](#_Toc87811098)

[Requerimientos funcionales 3](#_Toc87811099)

[Requerimientos no funcionales 6](#_Toc87811100)

[Diagrama de la arquitectura 6](#_Toc87811101)

[Manejo de dependencias 9](#_Toc87811102)

[Archivo de configuración 9](#_Toc87811103)

[Modelo de la base de datos 10](#_Toc87811104)

[API 12](#_Toc87811105)

[JWT y Autenticación 13](#_Toc87811106)

[Olvido de password y reseteo 13](#_Toc87811107)

[Instalación 17](#_Toc87811108)

# Requerimientos Generales

* Realizar una aplicación web que permitirá gestionar controles pediátricos por parte de los padres.
* El target de las aplicaciones es padres de niños que requieren registrar controles pediátricos en formato digital.
* La aplicación debe ser responsive.
* La aplicación deberá desarrollarse utilizando los siguientes lenguajes y librerías obligatorias: HTML/CSS, React, JavaScript y NodeJS
* La base de datos ha utilizar es opción del grupo pudiendo escogerse entre SQL (MySql, SqlServer, Postgres) y NO SQL (MongoDB)

## Requerimientos funcionales

RF01: El sistema debe permitir a los usuarios registrarse en la aplicación ingresando como datos obligatorios: nombre y apellido, DNI, mail, número de teléfono y contraseña.

RF02: El sistema debe verificar que el DNI ingresado no se encuentre previamente registrado.

RF03: El sistema debe verificar que la contraseña elegida cumpla con los siguientes parámetros:

* Longitud mayor o igual a 8 caracteres
* Al menos una letra mayúscula

RF04: El sistema debe mostrar por pantalla un cartel indicando que el registro fue exitoso y redirigir a la pantalla de log in.

RF05: El sistema debe solicitar email y contraseña para ingresar a la aplicación.

RF06: El sistema debe mostrar una opción para aquellos usuarios que se olvidaron la contraseña.

RF07: El proceso de recuperar la contraseña debe ser por medio de un código enviado al mail. Para esto el usuario debe ingresar el mail y el dni con el que se registró.

RF08: El sistema debe mostrar una pantalla de inicio (‘landing page') en la cual se muestren:

* Botones de Ingreso y Registro
* Nombre de la institución y breve descripción
* Funcionalidades de la aplicación
* Links a artículos informativos de relevancia
* Un pie de pagina (‘footer’) con datos de contacto de la institución (teléfono, mail, redes sociales)

RF09: El sistema debe permitir que, una vez ingresado los usuarios, puedan navegar por el menú principal que consiste en:

* Control pediátrico
* Vacunas
* Percentiles
* Mi perfil
* Mis hijos
* Salir

RF10: El sistema debe mostrar el nombre del usuario que se encuentra logueado en la aplicación.

RF11: El sistema debe permitir en la pantalla de Control pediátrico ingresar los siguientes datos, para cada hijo: fecha, peso, altura, diámetro cabeza, observaciones, medicamentos recetados (medicamento, dosis, periodo), estudios médicos a realizar y sus resultados.

RF12: El sistema debe permitir modificar o eliminar controles existentes.

RF13: El sistema debe mostrar una alerta al querer eliminar un control, pidiendo la confirmación del usuario.

RF14: El sistema debe permitir filtrar y ordenar por columnas.

RF15: El sistema debe permitir en el menú de Vacunas ingresar los siguientes daos para cada hijo: Fecha, Vacuna, Lugar de aplicación.

RF16: El sistema debe mostrar el calendario de vacunación en el menú de Vacunas. Dicha imagen debe agrandarse al hacer un click sobre ella, y también debe poder descargarse.

RF17: El sistema debe guardar los datos ingresados en el menú de Control pediátrico y Vacunas solo cuando el usuario seleccione el botón Guardar. Si se retira de la pantalla, estos cambios se deben descartar.

RF18: El sistema debe mostrar en el menú de Percentiles dos imágenes de las tablas de estatura y de peso con respecto a la edad, para ambos sexos. Las tablas deben cambiar cuando el usuario seleccione el sexo correspondiente.

RF19: El sistema debe mostrar en el menú de Percentiles una pequeña tabla mostrando el peso, la altura y la fecha del control para cada hijo. Debe mostrarse un menú desplegable donde el usuario seleccionara al hijo, y en base a eso mostrar los datos en la tabla.

RF20: El sistema debe mostrar en el menú Mi Perfil los siguientes datos:

* Nombre
* Apellido
* Email (no editable)
* DNI (no editable)
* Teléfono

RF21: El sistema debe permitir cambiar la contraseña en el menú Mi Perfil. Para eso deberá ingresar una nueva contraseña, y repetirla. La misma debe cumplir los requisitos de RF03.

RF22: El sistema debe mostrar un cartel verde de ‘success’ cada vez que se guarden datos de manera exitosa. En caso de haber un error, debe mostrar un cartel rojo indicando un mensaje descriptivo del error.

RF23: El sistema debe mostrar en el menú Mis Hijos los datos de cada hijo el usuario registrado. Esto se debe mostrar como pestañas, donde el usuario puede ir cambiando y viendo los datos de cada hijo.

RF24: El sistema debe permitir en el menú Mis Hijos agregar un nuevo hijo. Para eso deberá ingresar su nombre, fecha de nacimiento, grupo sanguíneo, alergias y enfermedades.

RF25: El sistema debe mostrar un menú desplegable con los grupos sanguíneos posibles.

RF26: El sistema debe permitir ingresar múltiples alergias o enfermedades, cada una en un campo de texto distinto.

RF27: El sistema debe permitir eliminar un hijo del sistema, y debe mostrar un cartel de advertencia que el usuario debe confirmar para completar la operación. El realizarse, se deben borrar los controles de dicho hijo, y las vacunas ingresadas.

RF28: El sistema debe permitir salir de la aplicación al hacer click en el menú Salir, y debe redirigir a la pantalla de inicio (‘landing page’)

## Requerimientos no funcionales

RN01: La aplicación web debe ser responsive.

RN02: La aplicación debe estar escrita en React, HTML, CSS para el front end, y Node para el backend.

RN03: La aplicación debe utilizar una base de datos SQL o No SQL.

RN04: La aplicación debe tener un color azul ( #4F46E5) como principal, y blanco de fondo.

RN05: La aplicación debe ser rápida, y la comunicación con la base de datos debe ser eficiente.

RN06: El backend debe implementar una API con distintos endpoints que serán llamados por el front end.

RN07: Se debe utilizar un token JWT para la autenticación de rutas privadas en la API.

# Diagrama de la arquitectura

Para implementar la solución se decidió utilizar principalmente las siguientes tecnologías:

Front end:

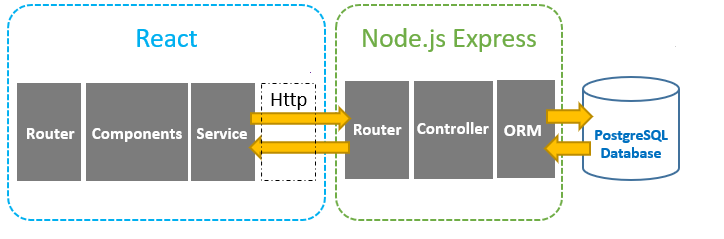
* React v.17
* Material UI v.4

Backend:

* Node v.14
* Express v.4

Base de datos:

* PostgreSQL v.12
* Node-postgres (pg) v.8

Diagrama de ejemplo:

En el front end, se organizó el código en distintas carpetas:

Imagen que contiene Diagrama

Descripción generada automáticamente

Los componentes privados son aquellos que requieren estar logueado, mientras que los públicos no.

La carpeta routes contiene la relación entre las rutas y los componentes a renderizar.

La carpeta images contiene todas las imágenes usadas en la aplicación, en formato .png, .jpg.

En código del backend se encuentra dentro de la carpeta /server, a la misma altura de /src.

Imagen que contiene Diagrama

Descripción generada automáticamente

En la carpeta auth se encuentra los relacionado al manejo del token JWT y su validación.

La carpeta bin contiene el archivo www.js, creado por express. Contiene información sobre el puerto sobre el cual va a correr el server, entre otros.

La carpeta controllers contiene todos los controladores que se encargan de ejecutar las funciones relacionadas a distintas rutas. En ellos se realizan validaciones sobre los parámetros de los endpoints, y se llama al modelo para ejecutar distintas operaciones, y al final se devuelve una respuesta del siguiente formato JSON:

{

    "success": "true",

    "message": "Usuario registrado correctamente",

    "data": {}

}

Todas las respuestas de la API tienen esos 3 campos, además del status code (200, 400, 500, etc).

En la carpeta dao se encuentran las clases dao que realizan queries sobre la base de datos. Todas utilizar la misma pool que mantiene la conexión a la base. Para esto se usó la librería [Node-postgres](https://node-postgres.com/) (pg).

En la carpeta model se encuentran las clases que representan al modelo, a las tablas de la base de datos, como User, Vacuna, Children, etc. Cada clase tiene asociado un dao.

En la carpeta routes se detallan los endpoints de la API, y a que función de los controladores están relacionadas. Ademas se detalla que tipo de llamada REST es (POST, GET, PUT, etc).

# Manejo de dependencias

Como la aplicación requiere de muchas otras librerías para funcionar, esto se detalla en el archivo package.json.

En el mismo se indica el nombre de la aplicación, la versión, las dependencias y sus versiones, como también comandos para correr la aplicación.

# Archivo de configuración

Con el objetivo de no escribir variables de configuración en el código, se utiliza un archivo .env , en el cual se detallan los valores de dichas constantes, por ejemplo, aquello relacionado a la conexión a la base de datos, o al envío de mails, o el secret key para JWT.

# Modelo de la base de datos

Se decidió utilizar una base de datos relacional PostgresSQL.

En base a los requerimientos, se diseño un modelo para la base, que consiste en las siguientes tablas:

Texto

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

La tabla principal es users, donde se guardan los usuarios registrados en la aplicación.

Los usuarios tienen hijos, y estos se guardan en la tabla children.

A su vez, cada hijo tiene controles y estos se guardan en la tabla control pediatrico.

También tienen vacunas, y se guardan en la tabla vacunas.

Por otro lado, los hijos pueden tener datos adicionales, como alergias o enfermedades. Esto se guarda en children\_data.

Por último, la tabla código se usa para guardar el código temporal generado al solicitar un reseteo de la contraseña, y es usado para validar que el usuario es efectivamente el que solicito el cambio.

# API

Al implementar el backend, se implementaron distintas funciones, cada una responsable de responder un endpoint distinto. Las rutas se separaron de acuerdo con el modelo en:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Es decir, si la operación es sobre la tabla vacunas, el endpoint está definido en el archivo vacunas.js, el cual llamara al controlador vacunasController, el cual se comunicara con el modelo Vacuna, y a su vez con el dao vacunas, para terminar realizando la operación sobre la tabla vacunas.

Endpoint -> routes -> validación -> controller -> modelo -> dao

Y luego el dao responde al modelo, el cual responde al controller, y este termina devolviendo la respuesta en la API.

Para probar los endpoints, se utilizó el programa Postman, al igual que la documentación de la API.

Link a la documentación de la API online:

[API2021 (getpostman.com)](https://documenter.getpostman.com/view/14469653/UVC8DSEB)

# JWT y Autenticación

Luego de que el usuario se loguea en la aplicación, se genera un token a partir del id del usuario y del secret key, definido en el back end y se lo envía al front end.

Este token JWT lo usa el front end para realizar llamadas a endpoints privados, pasándolo en el header ‘Authorization’. De esta manera, el back end sabe que el usuario está autorizado a realizar la operación.

El proceso de validación se encuentra en el archivo server/auth/auth.js

Para más información sobre JWT, ver [JSON Web Tokens - jwt.io](https://jwt.io/)

# Olvido de password y reseteo

En caso de que el usuario no recuerde la contraseña, deberá seguir los siguientes pasos:



Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Si los datos son correctos, se enviará un mail con un código aleatorio al mail del usuario.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ahora ingresa el código recibido en el siguiente campo, con la nueva contraseña.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Para el envío de mails, se utilizó la librería [Nodemailer](https://nodemailer.com/about/) .

Es necesario configurar la dirección de correo de la cuenta que va a enviar el mail. Se puede configurar en el archivo de configuración .env

# Instalación

1. Descargar el código fuente
2. Correr ‘npm install’
3. Descargar postgresSQL v12 o superior ([PostgreSQL: Downloads](https://www.postgresql.org/download/))
4. Instalar siguiendo los pasos recomendados.
5. Verificar que las credenciales seleccionadas coincidan con las del archivo .env

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Captura de pantalla con letras y números

Descripción generada automáticamente

1. Correr el archivo dump (postgres\_localhost-dump.sql)

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Verificar que las tablas fueron creadas

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

Descripción generada automáticamente

1. Correr ‘npm start’

El ultimo comando va a iniciar en simultaneo el server con node en el puerto 9000, y el front end en el puerto 3000.

Si solo se quiere correr el front end, correr ‘npm start-front’

Si se quiere correr el back end, correr ‘npm start-server’

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente