Taller 2

Integrantes

Julián Andrés Salamanca Tellez - 1841654

Juan Pablo Salgado Salas - 1843123

Andrés Felipe Giron Perez - 1842504

Kevin David Rodríguez Belalcazar - 1841109

Asignatura

Desarrollo de software II

Docente

Leidy Johanna Carvajal

Facultad de ingeniería, Universidad del Valle

Santiago de Cali, Valle del Cauca

30 de Junio del 2022

Tabla de contenido

Introducción **3**

Diseño de Software **4**
Modelo de arquitectura de software 4
ORM del proyecto 4

Patrones de construcción de software **5**
The Hooks pattern (“Observer”) 5
Conditional Rendering pattern 6
Presentational and Container Component Pattern 7

Calidad de software **8**
Atributos de calidad 8
Métricas 9
Estándares de calidad 10

Referencias **12**

Índice de figuras

Diagrama de arquitectura de software Pág 4
Diagrama Hooks pattern (observer) Pág 5
Diagrama Conditional Rendering Pattern Pág 6
Diagrama presentational and container component pattern Pág 7

Lista de definiciones, acrónimos y abreviaturas

1. MVC: Modelo vista controlador
2. ORM: Object-relational mapping
3. ISO: International Organization For Standardization

Introducción

En este documento se presentan algunas de las especificaciones que hacen parte del desarrollo de la aplicación web “IPS Salud en Casa”, de las cuales expondrán aspectos de su diseño (incluyendo un modelo práctico sobre su arquitectura); los patrones de construcción que permitirán un desarrollo más ligero y reutilizable; y por último se mencionarán algunos de nuestros estándares de calidad que podrán ser observados en el producto final, por los cuales permitirán destacar nuestro producto respecto a otros del mismo ámbito .

Diseño de Software

En este proyecto se presenta un estilo arquitectónico o patrón llamado *MVC* , en el cual el usuario presentará sus peticiones al modelo de datos por medio de un intermediario llamado controlador, este se encargará de comunicarse con la información almacenada en el sistema, además de actualizar la vista, es decir, el apartado que el usuario podrá observar en su dispositivo.
Por medio de esta separación de sectores, la labor de cada uno de nuestros desarrolladores se verá afectada positivamente, facilitando así la codificación y detección de mejoras y errores. Además de ello, este patrón de arquitectura de software nos facilitará nuestro proceso en la fase de pruebas ya que podemos probar el correcto funcionamiento del sistema por cada uno de los sectores que hacen parte del *MVC*.

Modelo de arquitectura de software

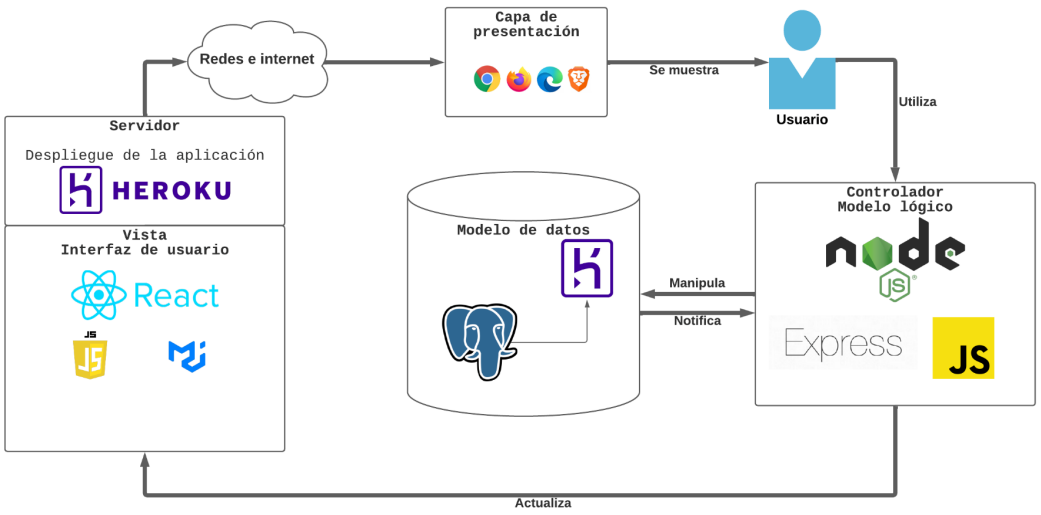

Figura 1. Diagrama de arquitectura de software

ORM del proyecto

El mapeador de objeto-relacional u ORM por sus siglas en inglés, está siendo utilizado en nuestra aplicación para abstraer las estructuras del modelo de datos relacional, a partir de esto los integrantes del desarrollo tienen a su disposición una herramienta que facilita la consulta y modificación de la información almacenada.
Queremos destacar que uno de nuestros propósitos como equipo de trabajo, es el uso del ORM como método principal para la manipulación de los datos entregados por el usuario. Además de lo anterior, después de múltiples investigaciones y pruebas, escogimos la aplicación de mapeo Prisma como nuestra herramienta principal de abstracción de estructuras de modelos de datos.

Patrones de construcción de software

The Hooks pattern (“Observer”)

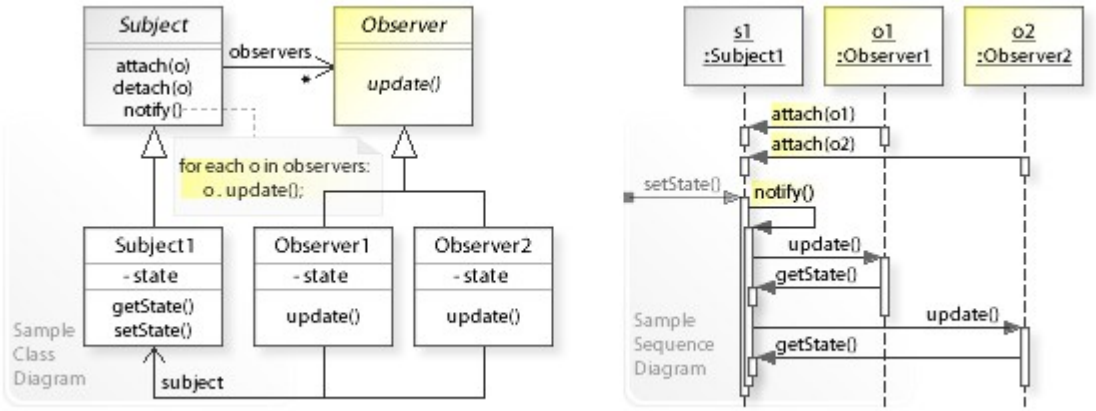

Figura 2. Diagrama Hooks pattern (observer)

El patrón de diseño observer consiste en tener una herramienta que nos permita la vigilancia constante de un componente o variable para reaccionar en cuanto cambie, para nuestro caso en concreto se usan los hooks. En el código de ejemplo se presenta la típica situación de formulario, el usuario puede escoger el tipo de especialidad del médico y en cuanto esta cambia el componente actualiza las opciones de médicos que tiene el usuario a solo aquellas que coinciden con la especialidad, cada vez que cambie la lista de médicos, se selecciona como opción el primer médico, este cambio se ve reflejado en la interfaz. En resumen, este patrón de diseño es muy útil para componentes o variables que dependen del estado de otros componentes o variables

```
const [medicos, setMedicos] = useState({})
const [medicoSeleccionado, setMedicoSeleccionado] = useState({})

[...]

useEffect(() => {
  if (JSON.stringify(medicos) !== '{}') {
    if (medicos.length !== 0) {
      setMedicoSeleccionado(medicos[0].id_trabajador)
    } else {
      setMedicoSeleccionado('0')
    }
  }
}, [medicos])
```

Conditional Rendering pattern

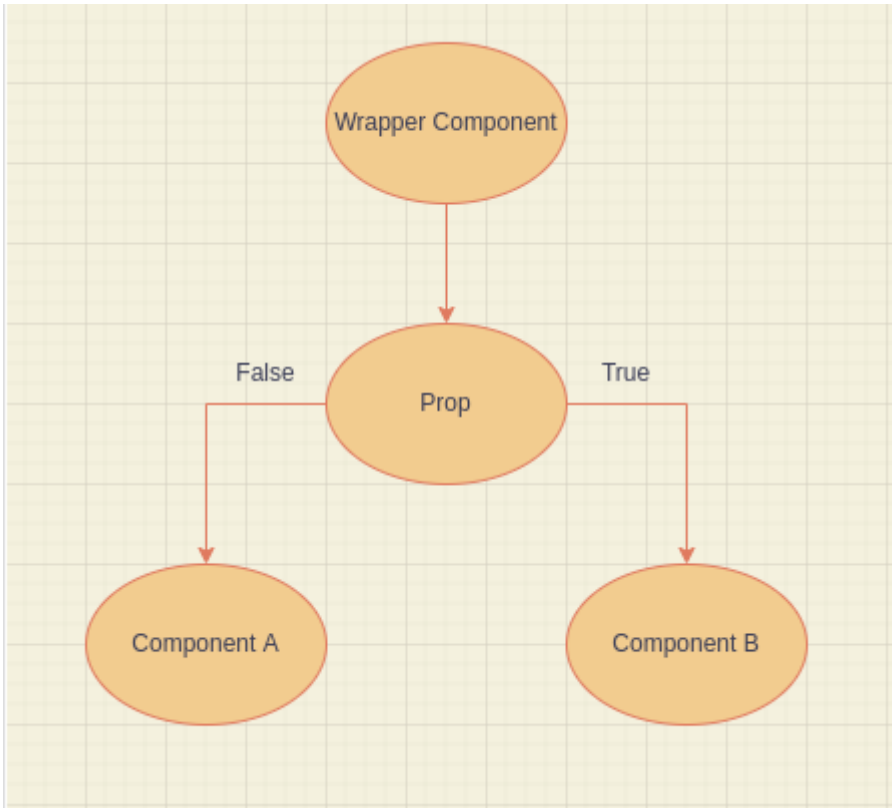

Figura 3. Diagrama Conditional Rendering Pattern

Este patrón de diseño consiste en la revisión de una condición lógica antes del renderizado para decidir qué componente es menester aparecer (las modificaciones de un componente se entienden como otro componente distinto y no como modificaciones). En el ejemplo que presentamos a continuación se muestra cómo este tipo de patrón es muy útil cuando se requiere traer información de la base de datos justo en el instante de aparición del componente, puesto que así es más fácil controlar el curso de acción de la aplicación en caso de que ocurra un error al traer información o si es necesario desplegar la información de manera distinta dependiendo de lo que se requiera.

```
 const [admin, setAdmin] = useState({})

[...] //some code between

 useEffect(() => {
   getInfoAdmin(user.sub, setAdmin)
 }, [])

[...] //some code between

 if (JSON.stringify(admin) === '{}') {
   return (
```

```
      <Box sx={{ width: '100%' }}>
        <LinearProgress />
      </Box>
    )
  } else {
    return (
       [...] // code of the react component
       )
}
```

Presentational and Container Component Pattern

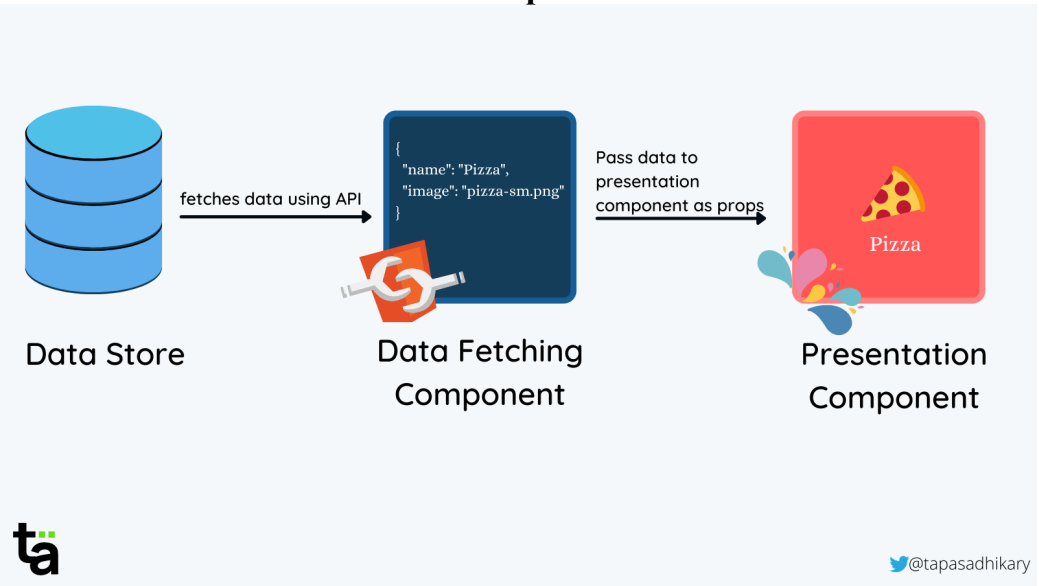

Figura 4. Diagrama presentational and container component pattern

La estrategia que sigue este patrón de diseño en particular es la de separar las partes del componente que corresponden a la lógica de las que corresponden a la presentación de la interfaz al usuario, lo cual deja de manera explícita que en la sección que compete a la presentación no esta permitido ningún tipo de lógica implícita. En el código a continuación se ejemplifica que antes del return existen una serie de componentes que tienen una lógica embebida (como el código es muy largo hemos decidido omitir y solo dejar en claro que ahí se encuentra la lógica) luego estos componentes se acoplan al componente de presentación. Este patrón de diseño es muy útil, en ocasiones parecidas al código que usamos aquí de ejemplo, es decir para la presentación de datos en tablas, ya que operar la lógica en medio del componente MUIDataTable (o cualquiera parecido que sirviese la misma función) haría menos comprensible el comportamiento del código, por lo cual futuras modificaciones supondrán una relectura completa del código y no solo de la sección que se requiera modificar.

```
export default function TableMedicos (props) {
     [Container components]
 return (
     <Box sx = {{ mt: 8, ml: 1, mr: 1 }}>
         <MUIDataTable
           title={'Medicos'}
           data={props.info}
           columns={columns}
           options={options}
         />
     </Box>
 )
}
```

Calidad de software

Atributos de calidad

1. **Seguridad**
   Este atributo permite describir el grado de protección, integridad y/o confidencialidad de los datos ante una posible pérdida o uso malintencionado en el sistema.

   En el proyecto, el sistema de acceso a la aplicación se encuentra regulado por la plataforma de gestión auth0, por lo tanto, las contraseñas de los correos registrados no estarán comprometidas en el caso de anomalías durante la ejecución de la aplicación.

2. **Usabilidad**
   Describe qué tan inteligible e intuitivo es el sistema para los usuarios y qué tan protegido está frente a errores que pueda cometer el usuario mientras aprende el uso de la aplicación.

   En el proyecto se tiene en cuenta que los procesos de registro y formularios tienen que ser en lo posible cortos y muy específicos con la información a ingresar. No obstante, se han implementado validaciones tanto en el back como en el front para proteger al sistema de datos erróneos que puedan ser ingresados por el usuario.

Además, para facilitar el aprendizaje del uso de la aplicación, se hizo el mayor esfuerzo en minimizar el número de campos a ingresar en los formularios implementados

3. **Adecuación funcional**

   Explica la completez funcional del proyecto en la cual se llegan a cubrir todos los requerimientos funcionales propuestos para el mismo.

4. **Modularidad**

   Describe la capacidad del sistema para modificar un componente y que tenga un impacto mínimo en los demás.

   En el proyecto, se trabajó con diferentes módulos para clasificar a los usuarios de aplicación y en estos se usaron diferentes componentes para las distintas funcionalidades de cada rol, de tal forma que a la hora de realizar un cambio en ellos, sólo afectará a dicha acción de ese módulo en específico y el impacto no será alto.

Métricas

Tabla 1. Métricas

Atributo	**Criterio**	**Métrica**
Seguridad	Protección de cuentas en caso de robo o anomalías.	Uso de al menos una plataforma de gestión de acceso externa.
Usabilidad	Protección frente al ingreso de datos erróneos por parte del usuario	Uso de al menos una alerta en cada formulario.
Usabilidad	Cantidad de acciones que debe realizar un paciente para agendar una cita	5 Acciones.
Funcionalidad	Completitud funcional de todos los requerimientos funcionales programados	Satisfacción de los Acceptance Criteria de cada historia de usuario.
Modularidad	Capacidad de realizar cambios con un impacto mínimo	Uso de 1 list-item para cada módulo.

Estándares de calidad

ISO ISO/IEC 9126

Esta norma nos permitirá evaluar nuestro producto en términos de las características de calidad y sus métricas asociadas. En el desarrollo de software podemos considerar la calidad en términos del proceso de desarrollo del producto y de su usabilidad. La calidad de usabilidad hace referencia a la perspectiva de calidad que tiene el usuario referente al producto de software cuando este es usado en el entorno para el que fue diseñado.

Según esta norma las características de calidad de software son:

• **Funcionalidad:** determina la capacidad del software de responder en términos de lo que el usuario necesita. Para esto se evaluará en nuestro proyecto el cumplimiento funcional, la conformidad y seguridad de acceso.

• **Confiabilidad:** depende de aspectos como la capacidad y facilidad de recuperación, la mitigación de fallos y la cantidad de tiempo que el software está disponible para su uso. La confiabilidad en nuestro caso está ligada a 3 componentes principales de nuestro proyecto. Primero, tenemos nuestra base de datos montada en heroku. Segundo, tenemos el backend de nuestra aplicación que interactúa con dicha base de datos. Por último, está el frontend de la aplicación que interactúa con el backend. Para poder asegurar la confiabilidad de nuestro producto tendríamos que asegurarnos que los servicios de servidores que utilicemos sean buenos.

• **Usabilidad:** mide qué tan intuitivo es el software, el manejo que el usuario le da al sistema y si este presenta menús sencillos, lectura de textos ágil, cuenta con funciones de forma clara. Para asegurarnos de cumplir con esto pensamos en la posibilidad de probar nuestro software con personas que no hayan participado en su desarrollo.

• **Eficiencia:** analiza y mide cómo el software hace uso de los recursos del sistema. Con las herramientas de desarrollo que ofrecen los navegadores modernos tenemos la posibilidad de ver el rendimiento de nuestra aplicación y determinar en que áreas deberías hacer optimizaciones para disminuir los tiempos de respuesta y uso de recursos en la medida de los posible

• **Portabilidad:** la facilidad con que el software puede ser llevado de un entorno a otro. Debido a que nuestro software es una aplicación web podemos asegurar que funcionará en cualquier computador capaz de ejecutar un navegador de internet moderno.

ISO 14598

El estándar es usado para la evaluación del producto software. Esta evaluación se desarrolla en las siguientes etapas:

Visión General: se establece un resumen de las otras cinco etapas y explica la relación entre la evaluación del producto software y el modelo de calidad. Para iniciar este proceso nuestro equipo debería seguir los siguientes pasos:

1. Establecer los requerimientos de evaluación
2. Especificar la evaluación
3. Planear la evaluación
4. Ejecutar la evaluación

Planificación y Gestión: en el momento de planificar la evaluación del proyecto se deberán seguir los siguientes pasos:

1. Definición de objetivos
2. Identificación de la tecnología a utilizar para la evaluación
3. Asignación de responsabilidades

Proceso de desarrolladores: proceso evaluación que se hace paralelo al desarrollo del producto con el objetivo de mejorarlo utilizando indicadores como su organización, planeamiento, especificación, diseño y montaje.

Proceso de comparadores: proceso de evaluación en función de los compradores del producto.

Proceso evaluadores: este proceso es utilizado por organizaciones externas encargadas de evaluar. La evaluación busca ser imparcial y objetiva.

Módulo evaluación: Especifica las mediciones y criterios sobre los atributos de calidad que se definieron anteriormente para la evaluación del producto.

Referencias

- Eagles, L. (2022, March 2). *React component design patterns for 2022*. LogRocket Blog. Retrieved June 30, 2022, from React component design patterns for 2022 - LogRocket Blog
- UXpin. (n.d.). *The Best React Design Patterns You Should Know About 1*. UXPin. Retrieved June 30, 2022, from The Best React Design Patterns You Should Know About
- *Normas y Estándares de calidad para elNormas y Estándares de calidad para el desarrollo de Software*. (n.d.). Normas y Estándares de calidad para el desarrollo de Software – Fernando Arciniega. Retrieved June 30, 2022, from http://fcaenlinea.unam.mx/anexos/1728/Unidad_2/u2_act2_1.pdf
- *Todo sobre la ISO/IEC 9126: 2001 y su importancia en las empresas | Verity*. (n.d.). Verity Consulting. Retrieved June 30, 2022, from https://www.verity.cl/blog/que-es-norma-iso-iec-9126-2001
- *Norma ISO/IEC 14598*. (n.d.). EcuRed. Retrieved June 30, 2022, from https://www.ecured.cu/Norma_ISO/IEC_14598