Diseño de una arquitectura de referencia para aplicaciones frontend, desarrollo de un template de CI/CD e investigación sobre herramientas para el monitoreo de errores

Cristian Camilo Mendoza Mancera

Informe de práctica para optar al título de Ingeniero de Sistemas

Asesor
Daniel Esteban Yepes Palacio, MSc. En Ingeniería.

Universidad de Antioquia
Facultad de Ingeniería
Departamento Ingeniería de Sistemas
Medellín, Antioquia, Colombia
2023

Dedicatoria

A mi madre y a mi padre por todo el apoyo que me han brindado a lo largo de todos estos años de esfuerzo y dedicación a la universidad.

Agradecimientos

A mis asesores Frank Castrillón y Daniel Yepes por todo el conocimiento compartido durante este proyecto de prácticas. A la Universidad de Antioquia por formarme profesionalmente en esta carrera.

Tabla de contenido

LISTA DE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

CI/CD	Continuous Integration and Continuous Delivery
E2E	End to End
AWS	Amazon Web Services
JS	Javascript
TS	Typescript
HOC	High Order Component
NPM	Node Package Manager

RESUMEN

En este artículo se describe las actividades llevadas a cabo durante el transcurso de un semestre de industria en el cual se contempló el diseño de una arquitectura de referencia para aplicaciones frontend, el desarrollo de un template para realizar un proceso de integración continua y despliegue continuo(*CI/CD*) y finalmente una investigación sobre herramientas para el monitoreo de errores en aplicaciones basadas en frameworks de *JavaScript*. Todo esto tiene como objetivo crear una serie de insumos que sean utilizados por todos los equipos de desarrollo dentro de la compañía en futuros proyectos frontend.

Palabras clave **— Arquitectura de referencia, Arquitectura Front End, *CI/CD*, Herramientas de monitoreo de errores**

ABSTRACT

This article describes the activities carried out during an industry semester in which the design of a reference architecture for frontend applications was contemplated, the development of a template to carry out a continuous integration and continuous deployment (*CI/CD*) process and finally a research on tools for monitoring errors in applications based on *JavaScript* frameworks. All this is aimed at creating a series of inputs that are used by all development teams within the company in future frontend projects.

Keywords **— Reference architecture, Front End Architecture, *CI/CD*, Error monitoring tools.**

I. INTRODUCCIÓN

Este documento tiene como objetivo principal el de presentar los resultados de la práctica académica que se desarrolló durante los últimos 5 meses en la empresa de software Experimentality S.A.S, empresa en la cual surgió la necesidad de contar con una arquitectura de software para aplicaciones frontend que fuera general o de referencia, que sirviera como base para los equipos de desarrollo dentro de la empresa para el diseño de las arquitecturas dentro de los diferentes proyectos de la empresa teniendo en cuenta el *stack* dentro de la misma, es decir, en este caso solo nos enfocaremos en los *frameworks* de *React* y *Angular*; en este sentido, la idea principal es la de crear una arquitectura y trasladarla a una estructura de carpetas que permita tener una buena escalabilidad y mantenibilidad en los proyectos de frontend además de establecer patrones de diseño y buenas prácticas a implementar en el desarrollo de las aplicaciones. Por otra parte, dentro de este marco de práctica también se propuso la creación de un template para un pipeline de *CI/CD* en el cual se contempla realizar un proceso de aseguramiento de la calidad de software a través de la implementación de un *análisis estático del código* mediante *Sonar* además de la ejecución de *pruebas unitarias* y *pruebas E2E* y en cuanto al despliegue de la aplicación en el proceso del *pipeline* se realizó sobre la nube de *AWS* con sus servicios de *S3* y *cloudfront*; finalmente como última propuesta se realizó una investigación sobre algunas herramientas para la implementación de monitoreo de errores en las aplicaciones frontend, investigación en la cual el objetivo principal es el de plasmar de forma general las diferentes características o funcionalidades de cada una de las herramientas así como de realizar un cuadro comparativo entre las mismas. Por otra parte, en esta práctica se siguió metodología secuencial o metodología en cascada en la que se definieron 3 bloques principales para el desarrollo de los temas definidos anteriormente de la práctica; como resultado de estos 3 bloques se tuvo el diseño de la arquitectura y la creación de un documento de arquitectura en el cual se explica a detalle cada aspecto de la misma, un template de *CI/CD* utilizando la herramienta de *Github Actions* y finalmente un documento en el cual se analizan 3 herramientas populares para el monitoreo de errores en aplicaciones frontend.

II. OBJETIVOS

A. Objetivo general

Diseñar una arquitectura de referencia para aplicaciones front-end, en la cual se contemplen atributos de *escalabilidad*, *mantenibilidad*, procesos de *CI/CD*, prácticas de *análisis estático de código*, *pruebas unitarias*, *pruebas de integración*, *pruebas E2E* e investigación de plataformas de monitoreo y logueo de errores en aplicaciones frontend.

B. Objetivos específicos

- Diseñar una arquitectura de referencia para aplicaciones frontend.
- Diseñar y desarrollar templates para procesos de *CI/CD*.
- Revisar prácticas de *análisis estático de código*, *pruebas unitarias*, *pruebas de integración* y *pruebas E2E*.
- Investigar y detallar las ventajas y desventajas sobre las plataformas más usadas para el *monitoreo y log de errores* en aplicaciones frontend.

III. MARCO TEÓRICO

En los proyectos de software es importante tomar decisiones adecuadas desde las etapas de tempranas del desarrollo, debido a que estas son las que marcan el rumbo de los proyectos y realizar cambios sobre dichas decisiones durante el desarrollo de los proyectos puede llevar a sobrecostos en tiempo y dinero, en este aspecto tenemos que una de las decisiones importantes a tener en cuenta en dichas etapas es la de definir una arquitectura que permita tener una alta escalabilidad y mantenibilidad en el proyecto y al mismo tiempo que se adapte correctamente a todos los requisitos funcionales y no funcionales del sistema. En este momento es cuando una arquitectura de referencia puede ser de ayuda para los desarrolladores ya que esta "proporciona un vocabulario común, diseños reutilizables y las mejores prácticas de la industria que se utilizan como base para arquitecturas más concretas"[1]. Es importante destacar que estas arquitecturas sirven como una guía para que los desarrolladores dentro de un proyecto de software diseñen e implementen una arquitectura que se adapte correctamente a las necesidades y requerimientos del proyecto, todo esto al realizar conversaciones entre los miembros del equipos y/o llegar a acuerdos en conjunto para adaptar los conceptos establecidos en la arquitectura de referencia con la finalidad de que se logre el objetivo de mantenibilidad y escalabilidad durante el desarrollo del proyecto. Por otra parte, este recurso provee la posibilidad de que todos los desarrolladores del proyecto tengan las mismas definiciones en cuanto a las responsabilidades de cada uno de los componentes de software del sistema, así como define una guia para la codificación que permite que cualquier miembro del equipo tenga la capacidad de modificar y/o mantener cualquier parte del código de forma sencilla y rápida. Adicionalmente, otro aspecto importante a tener en cuenta dentro de los procesos de desarrollo actuales, son los constantes cambios tanto en entornos de producción como de desarrollo y pruebas, lo cual supone el reto de lograr que estos cambios no introduzcan problemas y/o errores en los sistemas desarrollados, en este sentido, es importante definir de forma acertada los procesos de calidad a llevar a cabo dentro de los proyectos teniendo como meta lograr que el software que se desarrolló tenga un porcentaje de errores muy bajo. Es por esto que dentro de este proyecto de prácticas se buscará la implementación de un proceso de *CI/CD*, el cual "introduce la automatización continua y el monitoreo continuo a lo largo del ciclo de vida de las aplicaciones, desde las fases de integración y prueba hasta la entrega y el despliegue"[2]; esto con la finalidad de asegurar que el código que se integre en las diferentes

etapas de desarrollo y prueba cuente con un proceso de pruebas y de *análisis estático de código* que puedan asegurar la calidad de los desarrollos a lo largo del proyecto, adicionalmente se tendrá en cuenta todo el proceso para realizar las *entregas continuas* durante el transcurso del proyecto con el propósito de satisfacer las necesidades de los clientes. En este caso, el implementar estos procesos nos trae diferentes beneficios tales como "solución de errores más rápidos, infraestructura eficiente, mejor calidad de código, colaboración y comunicación, etc"[3].
A pesar de realizar todo el proceso de *CI/CD* para asegurar la *calidad del código* y de tener una arquitectura que permita tener una alta escalabilidad y mantenibilidad, ningún software es inmune a fallos de algún tipo durante el tiempo de operación en entornos de producción a pesar de contar con procesos de desarrollo y calidad muy maduros, por lo que tener un sistema de *monitoreo de errores* adecuado que permita hacer seguimiento al comportamiento del sistema puede aportar mucho valor y ser muy beneficioso para identificar las razones detrás dichos errores para que los equipos de mantenimiento puedan llegar a tener una solución rápida y oportuna para que estos errores esten presentes en la aplicación en producción durante poco tiempo y por ende los clientes no se vean afectados. Teniendo en cuenta esto, la idea en este proyecto es la de realizar una investigación de diferentes herramientas para realizar *monitoreo de errores* en aplicaciones frontend y construir un documento en el cual se describa de forma general cada una de las herramientas con sus funcionalidades más importantes para llevar a cabo dicho proceso de monitoreo y finalmente realizar un cuadro comparativo para que los equipos de desarrollo puedan decidir con mayor facilidad cuál herramienta elegir dependiendo de las necesidades y presupuesto que existan en cada uno de los proyectos.

IV. METODOLOGÍA

La ejecución de este proyecto se realizó siguiendo una metodología secuencial o en cascada que consiste en un "modelo estático el cual aborda el desarrollo de sistemas en una forma lineal y secuencial, completando una actividad antes que otra"[4]. En este aspecto para este proyecto se establecieron 3 etapas o actividades en las cuales la intención era la de cubrir todos los objetivos planteados en la sección anterior, esto a través de realizar tareas de investigación, diseño, documentación y desarrollo dentro de las diferentes actividades que serán especificadas a continuación:

1. **Investigación**: En esta primera actividad se realizó una investigación sobre las diferentes arquitecturas, patrones de diseño, anti-patrones de diseño , buenas prácticas del lenguaje principal para el desarrollo frontend(*JS/TS*), etc. Esto con la finalidad de conocer todos los aspectos relevantes a la hora de tomar decisiones respecto al diseño de la arquitectura de software.
2. **Diseño de la arquitectura**: Teniendo en cuenta todo el conocimiento que se obtuvo durante la ejecución de la actividad anterior, en esta actividad se ejecutó todo el proceso de diseño de la arquitectura, en la cual se buscó definir la estructura de carpetas y las responsabilidades de cada una estas así como la interacción entre los componentes principales de la arquitectura, adicionalmente se documentó todo este proceso a través de un documento de arquitectura en el cual se plasmó el diseño final dando una explicación detallada de cada una de las carpetas dentro de la arquitectura además de realizar una serie de recomendaciones a nivel de mejores prácticas para los frameworks que se tuvieron en cuenta en las etapas iniciales del proyecto(Angular y React).
3. **Desarrollo del proceso de CI/CD y log de errores**: En este última actividad se desarrolló un *template de CI/CD* desarrollado con *Github Actions* con el objetivo de realizar un proceso de aseguramiento de la calidad del código dentro de la compañía además de un proceso de despliegue continuo para realizar entregas tempranas dentro de los diferentes proyectos de la compañía. En un primer lugar, para realizar el aseguramiento de la calidad del código se realizaron validaciones a nivel de *pruebas unitarias* y *pruebas E2E* además de realizar un proceso de análisis estático de código a través de la herramienta de *Sonar Cloud* y finalmente en cuanto al despliegue de la

aplicación se realizó un proceso de construcción o *build* de la aplicación y subida de todos los archivos a un *S3* de *AWS* el cual se encuentra conectado a una distribución de *cloudfront* para la cual el último paso dentro del template es la de realizar una *invalidación de caché* de dicha distribución. Finalmente, dentro de esta actividad se realizó un proceso de investigación y documentación de diferentes herramientas para el *monitoreo de errores* en aplicaciones frontend, esta investigación se realizó sobre 3 herramientas de las que se documentó sobre las principales funcionalidades que ofrecen, los planes de precios y finalmente se realizó un pequeño cuadro comparativo con dichas herramientas, esto con el objetivo de que los equipos de desarrollo tengan un pequeño insumo que les evite realizar este proceso de investigación desde cero.

V. RESULTADOS

Como se mencionó anteriormente, está práctica académica consta de tres partes que se pueden considerar cada una como algo independiente la una de la otra, es por esto que en esta sección se hará una división para explicar cada uno de los objetivos de la práctica y los resultados obtenidos dentro de cada uno. En este aspecto, en esta sección se explicará en orden de ejecución los resultados obtenidos en el desarrollo de los objetivos de está práctica iniciando con el diseño de la arquitectura, siguiendo con el tema del aseguramiento de la calidad del código a través del *pipeline de CI/CD* y finalizando con la investigación realizada sobre las herramientas para el *monitoreo de errores* en aplicaciones frontend.

Arquitectura

El primer paso dentro de la práctica fue el de diseñar una arquitectura de referencia para aplicaciones frontend, arquitectura la cual estuviera enfocada en los *frameworks* principales de la empresa(*React y Angular*), en este sentido, se diseñaron dos arquitecturas muy similares entre sí pero con algunas diferencias que van ligadas a temas específicos de cada uno de los frameworks, sin embargo, los patrones de diseño y prácticas a implementar dentro de las arquitecturas son muy similares. En otro orden de ideas, cabe mencionar que para el diseño de estas arquitecturas se tuvo en cuenta realizar un diseño modular en el cual se diera foco a los diferentes módulos de las aplicaciones a desarrollar y seguir un sistema de diseño como lo es *atomic design*[5] para la reutilización de componentes visuales además de los aspectos más importantes como la escalabilidad, mantenibilidad, etc.

Teniendo como objetivo el de dar a conocer la arquitectura, en un principio se crearon dos diagramas de componentes con la finalidad de mostrar de forma gráfica los diferentes componentes de la arquitectura así como las interacciones entre estos. A continuación se tiene el diagrama de paquetes que muestra la arquitectura que se diseñó para React:

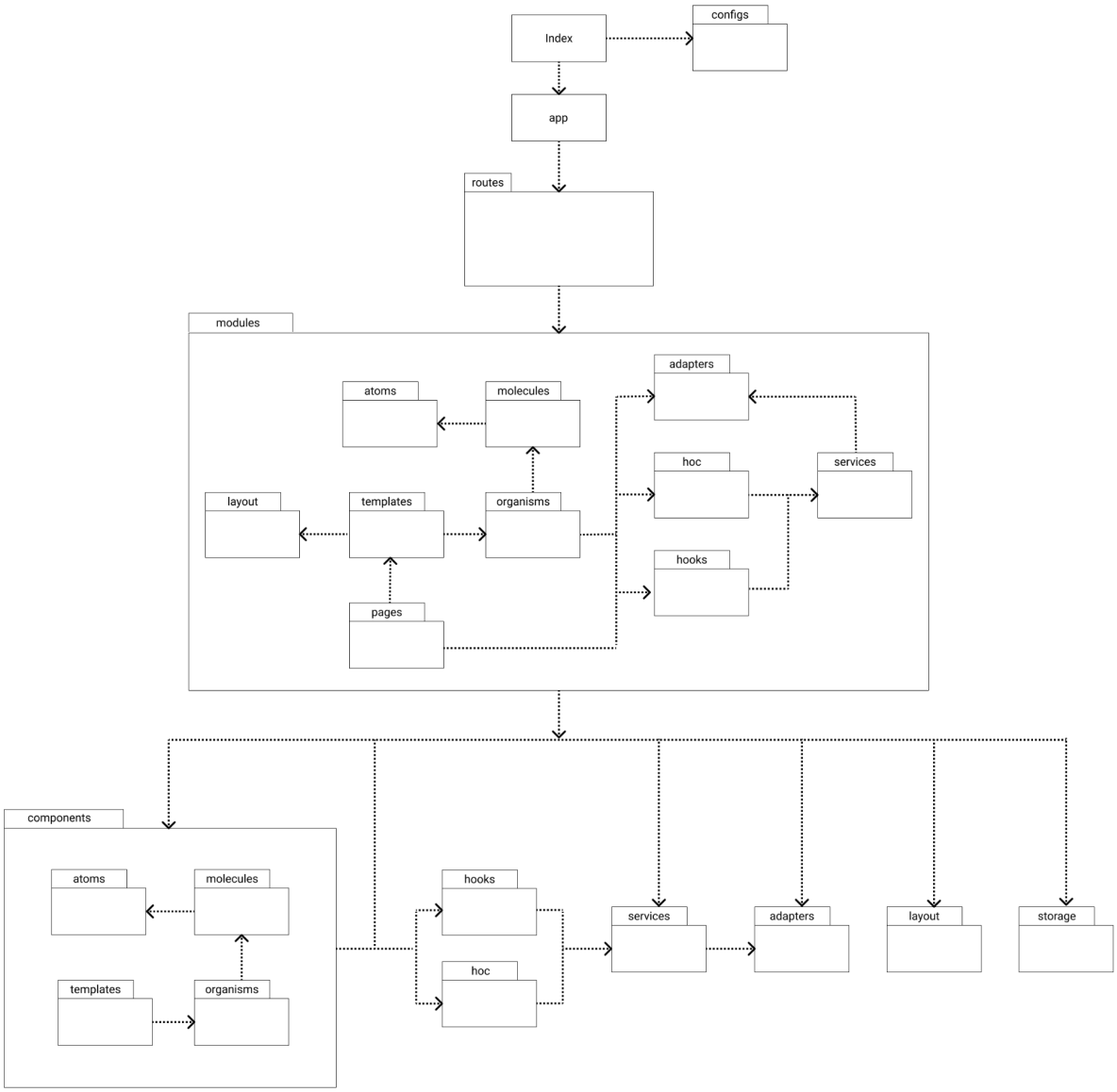

Figura 1. Diagrama de paquetes(React). Fuente elaboración propia.

En este primer caso tenemos el diagrama de paquetes para la arquitectura de React, en la cual se tienen los elementos específicos para este framework(*hooks* y los *high order components*(hoc)), los cuales nos sirven para encapsular lógica de negocio en nuestra aplicación con el propósito de separar dicha lógica de los componentes visuales. Por otra parte, en el siguiente diagrama se mostrará el diagrama de paquetes para la arquitectura de Angular la cual varía en algunos elementos como se mencionó anteriormente:

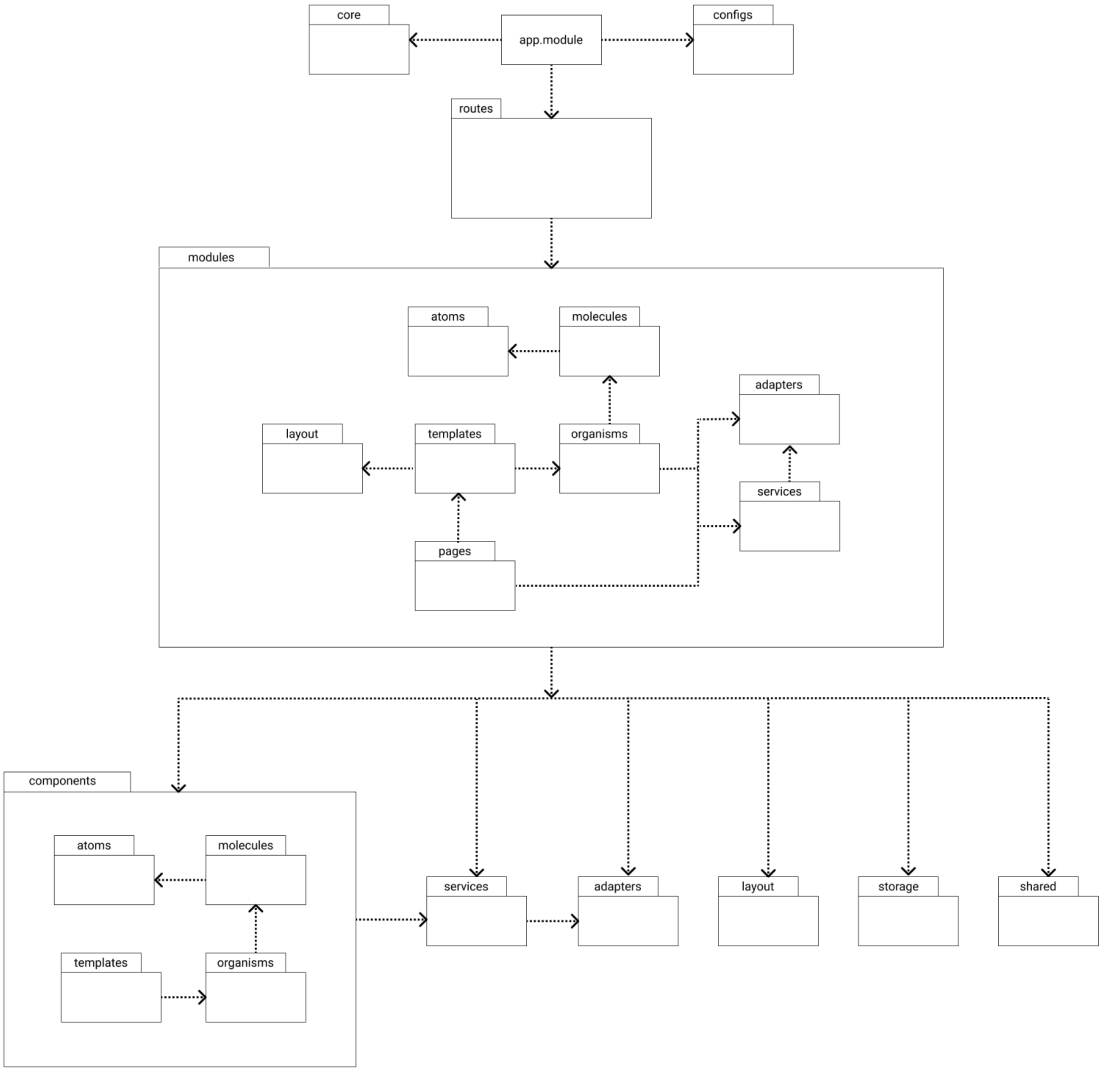

Figura 2. Diagrama de paquetes(Angular). Fuente elaboración propia.

Como se puede observar en las figuras 1 y 2 y como se mencionó anteriormente, las arquitecturas tienen como objetivo el crear la estructura de carpetas alrededor de los módulos o features de la aplicación a desarrollar así mismo como implementar o desarrollar los componentes de la aplicación siguiendo lo estipulado en el sistema de diseño de *atomic design*. Por otra parte, para dar más claridad de cada uno de los elementos que contiene las arquitecturas, a continuación se dará una descripción general de cada uno de estos:

src: En esta carpeta se tendrá todo el código fuente de nuestra aplicación. En este nivel tendremos algunas carpetas que contendrán elementos(components, adapters, hooks, hoc y services) que son transversales a toda la aplicación, esto quiere decir que son elementos que son utilizables por varios módulos o elementos dentro de la aplicación.

adapters o mappers: La idea principal de los adapters o mappers es la de tener una capa que nos permita mapear todos los datos que provengan y/o que se envíen a sistemas externos a nuestra aplicación o storage dentro la misma. En este caso lo que se quiere lograr con esto es mapear estos datos a objetos de negocio de nuestra aplicación que sean requeridos por los componentes que desarrollemos.

components: En esta carpeta se encuentran los componentes que sean más reutilizables a lo largo de toda la aplicación. La idea con estos componentes es que encapsulan funcionalidades comunes que necesitamos a lo largo del proyecto, sin embargo, estos componentes no se encargaran de gestionar lógica de negocio dentro del contexto de nuestra aplicación. En esta arquitectura se propone que estos componentes se construyan siguiendo los lineamientos de atomic design[n].

components/atoms: En este caso la idea es tener componentes pequeños que encapsulan una pequeña funcionalidad que sea reutilizable entre varios componentes.

components/molecules: En esta carpeta la idea principal es crear componentes un poco más complejos al utilizar dos o más atoms. Teniendo esto en mente, lo que nos permite esto es ir creando componentes que cada vez tengan más funcionalidades.

components/organisms: La idea en este carpeta es similar a la idea que se tiene en la carpeta de molecules, en este caso lo que se quiere lograr es construir componentes al juntar dos o más molecules o atoms con el propósito nuevamente de crear componentes que tengan más funcionalidades.

components/templates: La idea en esta carpeta es la de construir layouts comunes en diferentes módulos de nuestra aplicación; en este caso lo que se quiere lograr a este nivel es construir a través de la composición de

diferentes organisms, molecules o atoms dichos layouts. Acá es necesario crear toda la estructura a nivel de HTML y CSS(o framework/librería de CSS) ya sea a través de grid, flex, margins, paddings, etc.

configs: En esta carpeta se colocarán todos los archivos de configuración de nuestra aplicación ya sea de librerías, entornos de desarrollo, servicios externos, etc.

__mocks__: En esta carpeta se contendrá todo el código para realizar mocks de servicios, datos y/o peticiones para realizar las pruebas correspondientes de los elementos de nuestra aplicación.

hooks(React): En esta carpeta la idea es colocar los *custom hooks* desarrollados que sean utilizados por varios componentes o módulos de nuestra aplicación, es decir, custom hooks que sean transversales y que su funcionalidad no esté enfocada en la lógica de negocio de un componente o módulo en específico.

layout: La idea de esta carpeta es tener todos aquellos componentes de nuestra aplicación que sean parte del layout general, es decir, componentes como headers, barras de navegación, footers, etc.

hoc(React): La idea en esta carpeta es similar a la carpeta de hooks y es que en esta carpeta colocaremos todos aquellos *high order components* que sean muy reutilizables por varios módulos a lo largo de la aplicación.

models: En esta carpeta se contendrán los tipos y/o interfaces(modelos de negocio) de nuestra aplicación que sean más transversales a la misma.

services: En este carpeta la idea es colocar todo el código necesario para las peticiones HTTP hacia APIs o sistemas externos, esto con el objetivo de encapsular todo el código para la gestión de estas peticiones, cómo por ejemplo, gestionar los casos de errores o transformar los datos enviados y/o recibidos en las peticiones a través del uso de los adapter o mappers definidos anteriormente. Por otra parte, en Angular es posible que se encapsule lógica de negocio que no tenga que ver con peticiones HTTP dentro de algunos servicios y de esta forma quitar de esta responsabilidad a los componentes y a su vez que esta sea reutilizable.

storage(opcional): La idea principal aquí es que tengamos todo el código relacionado a nuestra librería o gestor de estado global que implementemos, debido a que existen varias posibles librerías, es necesario que se decida dentro del equipo cuál es la opción que mejor conviene teniendo en cuenta los requerimientos del proyecto y que en base a esto se discuta la mejor forma de realizar la implementación de dicha librería a nivel de arquitectura.

core(Angular): En esta carpeta la idea es tener el módulo core de Angular, en el cual se incluyan todos aquellos servicios que sean reutilizables a lo largo de varios módulos de la aplicación y que a su vez estos tienen la característica especial de ser singletons dentro de la misma.

shared(Angular): En esta carpeta la idea es tener el módulo de shared de Angular, en el cuál se incluyan componentes, directivas y/o guards que sean reutilizables a lo largo de varios módulos.

utils: En este carpeta la idea es almacenar funciones, constantes y/o validaciones que sean reutilizables a lo largo de la aplicación. En este caso se deja abierta la estructura interna de esta carpeta dependiendo de las necesidades que surjan dentro del proyecto.

routes: En esta carpeta la idea es encapsular todo el código necesario para realizar el enrutamiento de nuestra aplicación, esto con el objetivo de tener la facilidad de intercambiar en caso de ser necesario la implementación de librería de forma sencilla, ya sea que se quiera implementar una nueva librería o se quiera actualizar la versión de la misma. Por otra parte, en esta carpeta es donde implementamos todo el código necesario para realizar lazy loading, esto en caso de que según las características del proyecto, sea beneficioso implementarlo.

modules: En esta carpeta se contendrá el código más específico a cada uno de los módulos que se tenga dentro del contexto de negocio de la aplicación a desarrollar. En este caso la idea es tener una estructura similar a la que tenemos a nivel general de nuestra aplicación, sin embargo, teniendo en cuenta que la idea principal es tener todo el código enfocado en el desarrollo de cada uno de los módulos individualmente.

modules/module_name/adapters: La idea acá es principalmente la misma que tendremos a nivel general, pero como ya se mencionó anteriormente será principalmente aplicada a los componentes de nuestro módulo, por lo que lo que haremos en este caso será adaptar o mapear los datos que provengan de una API, sistema externo o storage a nuestros componentes de cada módulo en específico.

modules/module_name/atoms: En este caso la idea es la misma que la que tenemos en la carpeta de components, aquí tendríamos componentes que encapsulan una pequeña funcionalidad que sea reutilizable entre varios componentes del módulo.

modules/module_name/molecules: En esta carpeta la idea principal es crear componentes un poco más complejos al utilizar dos o más atoms.

modules/module_name/organisms: En este carpeta la idea es construir componentes más complejos al integrar varios componentes de molecules o atoms.

modules/module_name/templates: La idea en esta carpeta es la de construir los layouts principales de nuestro módulo, esto siguiendo la misma idea que tenemos en la carpeta de components.

modules/module_name/pages: En este carpeta se tendrá las diferentes páginas o vistas principales del módulo, en este sentido lo que se hará será crear instancias específicas de los templates que hemos creado anteriormente.

modules/module_name/hooks(React): En este caso tendremos la misma idea que los hooks generales, pero enfocados en encapsular la lógica de negocio del módulo.

modules/module_name/models: En esta carpeta la idea principal es la de tener todos los objetos o modelos de negocio de nuestro módulo modelados en esta carpeta.

modules/module_name/hoc(React): La idea sería tener los HOC que encapsulan lógica de negocio de un módulo específico teniendo como objetivo que estos sean reutilizables por varios componentes dentro del módulo.

modules/module_name/services: En este caso la intención es la misma que a nivel general, es decir, tener todo el código para consumir servicios(y/o encapsular lógica de negocio en Angular) ya sean de APIs o sistemas externos, pero en este caso tener esto a nivel de módulo, con la finalidad de quitar está responsabilidad a los componentes.

modules/module_name/layout(opcional): En esta carpeta tendremos los componentes de layout que sean específicos a un módulo, sin embargo, esta carpeta es opcional y solo será necesaria en casos en los que elementos de layout cambien entre módulos.

modules/module_name/__mocks__: En esta carpeta se contendrá todo el código para realizar mocks de servicios, datos y/o peticiones para realizar las pruebas correspondientes de los elementos de un módulo en específico.

Template de CI/CD

En esta parte la idea principal era la de desarrollar un template para implementar un proceso de *continuous integrations(CI)* y *continuous delivery(CD)* en aplicaciones de frontend en el que se contempló realizar una verificación de la *calidad del código* que se integrará en el repositorio remoto por medio de una validación de *pruebas unitarias*, *pruebas E2E* y de realizar un *análisis estático del código* utilizando la plataforma *Sonar Cloud,* plataforma en el cual se evalúan las métricas de complejidad ciclomática, repetición de código, cobertura de código, etc. Una vez se completan todos los pasos dentro del *proceso de CI*, se inicia con el proceso para desplegar la aplicación a la nube de *AWS* mediante el servicio de *S3* que a su vez está conectado con una *distribución de cloudfront*.

Como resultado de este desarrollo se tiene un *template* de Github Actions que se divide en dos secciones principales y que se muestran en las figuras 3 y 4, en estas secciones tenemos la configuración del *template* y del ambiente de la máquina en la cual se ejecuta el *pipeline* (figura

3), el *proceso de CI* y finalmente el *proceso de CD* (figura 4). A continuación se dará una explicación general de cada una de las secciones del *template* descritas:

En un primer lugar y como se puede ver en la figura 3 se tiene la configuración inicial del *template*, configuración en la cual, se especifican los eventos que ocurren sobre el *repositorio* que dispara la ejecución del *pipeline*, en este caso se configura que se escuchen los eventos de push y pull request sobre la rama master del *repositorio remoto* en *Github*; una vez se dispare un evento sobre el *repositorio*, el *pipeline* ejecutarán los *jobs* que se desarrollen en el *template*.

```yaml
name: CI/CD React

on:
  push:
    branches: ['master']
  pull_request:
    branches: ['master']
```

Figura 3. Configuración del ambiente de Github Actions. Fuente elaboración propia.

Como se puede observar en la figura 4, en el template desarrollado se definió un solo job que ejecutará todos los procesos de *CI/CD* además de la clonación del repositorio, configuración del *CLI* de *AWS* y del ambiente de *NodeJS* para la ejecución de las pruebas tanto unitarias como E2E y del build de la aplicación. Teniendo en cuenta esto, el primer paso dentro del job es definir el sistema operativo en el cual se ejecutarán los scripts del *pipeline* a través de la propiedad *runs-on* en el que se define que se ejecute sobre la versión más reciente de *Ubuntu*, después de esto se definen una serie de variables de ambiente para el *CLI* de *AWS* en el que se define unas credenciales para el acceso a los servicios de *AWS* y la región en la cual están dichos servicios, el siguiente paso es definir la versión de *NodeJS* que se va a usar dentro del *pipeline* y por último tenemos todos los pasos o "*steps*" que se desarrollaron para este *job*, en estos se tiene que en primer lugar, clonar el código fuente desde el *repositorio de Github* a la máquina que ejecuta los *scripts*, después se instalan las dependencias del proyecto a través de *NPM* al configurar el ambiente de *NodeJS* con la versión y ejecutar el script que instala dichas dependencias, una vez se tienen instaladas las dependencias se inicia con el proceso de *CI* que consiste en ejecutar las *pruebas unitarias* y las *pruebas E2E*, una vez se complete este proceso se inicia el *analisis estatico de códig*o con *Sonar* al integrar el pipeline con la herramienta de *Sonar Cloud* y una vez

se completa este análisis se hace una verificación sobre el cumplimiento de todos los requisitos de calidad definidos en *Sonar*; con este ultimo paso se termina el *proceso de CI* e iniciamos con *el proceso de CD* en el que el primer paso es realizar el *build* de la aplicación utilizando el *script* de *NPM*, inmediatamente a continuación se realiza el proceso de subir los archivos generados en el proceso de *build* a un *S3* mediante el *CLI* de *AWS* y definiendo que los permisos de lectura de los archivos como publicos, finalmente se realiza una invalidación de la *cache* de la distribución de *cloudfront* asociada al *S3*, esto nuevamente mediante el CLI de AWS.

```yaml
jobs:
  CI_CD:
    runs-on: ubuntu-latest
    env:
      AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
      AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      AWS_DEFAULT_REGION: ${{ secrets.AWS_DEFAULT_REGION }}

    strategy:
      matrix:
        node-version: [16.x]
        # See supported Node.js release schedule at https://nodejs.org/en/about/releases/

    steps:
      - name: Clone repository
        uses: actions/checkout@v3
        with:
          fetch-depth: 0
      - name: Setup Node.js ${{ matrix.node-version }}
        uses: actions/setup-node@v3
        with:
          node-version: ${{ matrix.node-version }}
          cache: 'npm'
      - name: Install dependencies
        run: npm install --save
      - name: Run unit tests
        run: npm run test
      - name: Sonar
        uses: sonarsource/sonarcloud-github-action@master
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
      - name: SonarQube Quality Gate check
        id: sonarqube-quality-gate-check
        uses: sonarsource/sonarqube-quality-gate-action@master
        env:
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
      - name: Run E2E test
        run: npm run test:e2e
      - name: Build
        run: npm run build
      - name: Uploading build files to S3 bucket
        run: aws s3 sync ./${{secrets.BUILD_FOLDER}} ${{ secrets.S3_BUCKET }} --acl public-read
      - name: Invalidating AWS cloudfront cache
        run: aws cloudfront create-invalidation --distribution-id ${{ secrets.CLOUDFRONT_DISTRIBUTION_ID }} --paths "/*"
```

Figura 4. Proceso de CI/CD. Fuente elaboración propia.

Investigación sobre las herramientas para *monitoreo de errores*

Como parte final de la práctica se hizo una investigación sobre 3 herramientas para el *monitoreo de errores* en aplicaciones frontend, esto con el objetivo de construir un documento en el que se plasmara una descripción general de cada una de las herramientas con sus respectivas funcionalidades destinadas al *monitoreo de errores*, a partir de esta investigación se realizó un cuadro comparativo(figura 5) con las diferentes funcionalidades que ofrece cada herramienta. Esta parte de la práctica se hizo con la finalidad de que los equipos de desarrollo de la compañía tuvieran un insumo que diera el contexto de las diferentes opciones para el monitoreo de errores y de esta forma se simplifique un poco la decisión sobre cuál herramienta elegir dependiendo de los requerimientos de cada proyecto.

	Sentry IO	BUGSNAG	RAYGUN
Breadcrumbs	✔	✔	✔
Datos sobre el entorno de ejecución y peticiones HTTP	✔	✔	✔
Stack trace	✔	✔	✔
Dashboards	✔	✔	✔
Metricas de estabilización		✔	
Monitoreo de perfomance			✔
Integración con repositorios de código y herramientas de gestión	✔	✔	✔
Alarma de errores	✔	✔	✔

Figura 5. Cuadro comparativo herramientas de monitoreo de errores. Fuente elaboración propia.

Finalmente, como se puede observar en la figura 5 no existen grandes diferencias entre las diferentes herramientas por lo que la decisión sobre qué herramienta elegir está supeditada a las funcionalidades que cada herramienta ofrezca adicionales al monitoreo de errores como funcionalidades para el monitoreo de performance, métricas de estabilización, etc. Adicionalmente los planes de cada herramienta en cuanto a precios y funcionalidades incluidas en cada uno de estos, también puede ser un factor diferenciador que se debe tener en cuenta dependiendo de los presupuesto de cada proyecto.

VI. ANÁLISIS

Como se puede observar en la sección anterior, esta práctica estuvo enfocada en diseñar, desarrollar y/o documentar algunos aspectos importantes dentro del desarrollo de aplicaciones frontend teniendo como objetivo principal el de servir como ayuda para los equipos de desarrollo dentro de la compañía en los diferentes proyectos de la misma, sin embargo, estos no pretenden ser la palabra final si no que lo que se busca es que sirvan como una base para que los equipos de desarrollo tomen las decisiones finales a la hora de realizar una implementación más concreta teniendo en cuenta los requerimientos y características de cada uno de los proyectos dentro de la compañía. En este sentido, también se insta a que la compañía implemente o desarrolle procesos internos para la continua revisión y mejoramiento de cada uno de los insumos desarrollados dentro del marco de esta práctica.

Por otra parte y entrando más en detalle a los resultados obtenidos, en cuanto al diseño de la arquitectura se tiene que se hizo un trabajo basado en la experiencia y sobre los conocimientos adquiridos durante el proceso de investigación realizado, aspectos que dieron como resultado una arquitectura que implementa conceptos para lograr ese objetivo principal de mantenibilidad y escalabilidad, sin embargo, es una arquitectura que aún no se ha probado en un proyecto real, por lo que aún no se tiene certeza sobre si en efecto logrará brindar esas características cruciales para un proyecto de software, por lo que es necesario que esta arquitectura sea probada internamente dentro de la compañía en un proyecto pequeño o mediano que no tenga tanto impacto en los clientes, esto con el objetivo de realizar una evaluación general sobre la arquitectura y los aspectos que se quieren lograr con esta, a partir de esto analizar los resultados obtenidos y decidir si es necesario realizar modificaciones realizando iteraciones sobre esta o si por el contrario se considera que la arquitectura cumple a cabalidad con brindar esos elementos de mantenibilidad y escalabilidad dentro de los proyectos de software. Otro punto es el tema del pipeline de *CI/CD*

desarrollado y es que en este caso la idea que se tiene en cuanto a una mejora que es posible plantear, es la creación o desarrollo de un proceso para el aprovisionamiento de infraestructura a través de herramientas de *infrastructure as code* como pueden ser **Terraform**, **Cloudformation**, etc. Esto con la finalidad de complementar y mejorar el proceso de **CI/CD** que fue desarrollado para que todo el proceso de integración y despliegue sea más completo y robusto. Por último respecto al tema de la herramientas para el monitoreo de errores es necesario que los equipos de desarrollo decidan con base a este insumo cuál herramienta es mejor dependiendo de las características y requerimientos generales de cada uno de los proyectos dentro de la empresa e incluso iterar este documento en base a experiencias con alguna de las herramientas o incluir en el análisis más herramientas que no se tuvieron en cuenta durante el desarrollo de esta práctica.

VII. CONCLUSIONES

En el desarrollo de aplicaciones es importante contar con varios elementos que aporten a la calidad de los procesos de desarrollo desde diferentes puntos o áreas de software, en este sentido, se puede considerar que la arquitectura de software, la calidad de código y monitoreo de errores son elementos que tienen una gran relevancia en el logro de la calidad de los proyectos de software y esto a lo largo de diferentes etapas dentro del ciclo de vida de estos, desde la planeación inicial, el desarrollo y finalmente el funcionamiento del mismo. Teniendo en cuenta esto, el desarrollo de los objetivos estuvo enfocado en estos tres ítems mencionados anteriormente, realizando esfuerzos y logrando el diseño de una arquitectura de software, el desarrollo de un pipeline de **CI/CD** y finalmente la documentación de algunas herramientas para el monitoreo de errores en aplicaciones frontend, todo esto con la finalidad de aportar al crecimiento de la madurez de los procesos de desarrollo dentro de la compañía en la cual se realizó esta práctica académica, en este aspecto, la idea es dejar una base para que al interior de la compañía se generan espacios de conversación y aprendizaje sobre estos resultados entregados con la idea de realizar mejoras, iteraciones o que nazcan iniciativas similares para llevar a cabo este proceso a otros campos dentro de la empresa, como lo pueden ser backend, analitica de datos e e-commerce.

Como se mencionó anteriormente, en este caso se considera que la propuesta sobre el diseño de una arquitectura de referencia puede brindar una base bastante buena en cuanto a buenas prácticas, mantenibilidad y escalabilidad de los desarrollo de software dentro de la compañía, sin embargo, es necesario realizar pruebas sobre la misma con el objetivo de refinar y construir una arquitectura mucho más robusta la cual pueda ser utilizada en los proyectos más importantes de la empresa. Por otra parte y siguiendo con el crecimiento en los procesos de calidad en la empresa, los temas planteados sobre el pipeline de *CI/CD* se tiene que el desarrollo de *pruebas unitarias*(e incluso pruebas de integración) y *E2E* y el análisis estático de código puede ser muy beneficioso para brindar confianza en los clientes y en el equipo de desarrollo en cuanto a la calidad de los desarrollos realizados y al correcto cumplimiento de los criterios de aceptación dentro de los proyectos e incluso el tema de monitoreo de errores puede ayudar significativamente a estos aspectos de generar confianza en los clientes de la empresa, debido a que ayuda a los equipos de desarrollo o a los equipos de mantenimiento a identificar y solucionar

errores en las aplicaciones de forma rápida y oportuna disminuyendo el impacto negativo en los procesos de los clientes.

REFERENCIAS

[1] L. I. X. GmbH, “Reference architecture - the definitive guide,” LeanIX. [Online]. Available at: https://www.leanix.net/en/wiki/ea/reference-architecture. [Accessed: 07-Nov-2022]

[2] Hat. 2022. What is CI/CD?. [online] Available at: https://www.redhat.com/en/topics/devops/what-is-ci-cd [Accessed: 09-Nov-2022]

[3] JetBrains. 2022. What are the benefits of CI/CD?. [online] Available at: https://www.jetbrains.com/teamcity/ci-cd-guide/benefits-of-ci-cd/ [Accessed: 09-Nov-2022].

[4] Adenowo, Adetokunbo & Adenowo, Basirat. (2020). Software Engineering Methodologies: A Review of the Waterfall Model and Object- Oriented Approach. International Journal of Scientific and Engineering Research. 4. 427-434.

[5] B. Frost, “Atomic Design methodology,” Atomic Design by Brad Frost, 2016. [Online]. Available: https://atomicdesign.bradfrost.com/chapter-2/. [Accessed: 11-Nov-2022].