Presentación 1 - MERN

Pontificia Universidad Javeriana
Arquitectura de Software

Integrantes:
Laura Ovalle
José Manuel Rodríguez
Juan David González

Bogotá D.C
21 de abril 2025

DEFINICIÓN

El stack MERN con TypeScript es un conjunto de tecnologías modernas para el desarrollo full-stack de aplicaciones web, compuesto por MongoDB (base de datos NoSQL orientada a documentos), Express.js (framework backend minimalista sobre Node.js), React (biblioteca frontend basada en componentes) y Node.js (entorno de ejecución JavaScript del lado del servidor). En esta implementación, se utiliza el lenguaje de programación TypeScript que se incorpora en cada capa del stack, lo que permite tipado estático, mejora la mantenibilidad del código y facilita la detección de errores.

Este stack se caracteriza por su coherencia tecnológica (JavaScript/TypeScript en toda la arquitectura), su compatibilidad natural con JSON y su capacidad para construir aplicaciones escalables, modulares y fácilmente testeables. Además, se basa en patrones de arquitectura modernos como Clean Architecture en el backend y Atomic Design junto con la arquitectura Flux en el frontend, lo que refuerza la separación de responsabilidades y la reutilización de componentes.

A continuación se presentan las diferentes definiciones de las tecnologías del stack MERN y de los patrones y arquitecturas utilizadas para el desarrollo de esta actividad:

Tecnologías del Stack MERN

React es una biblioteca de JavaScript desarrollada por Facebook para construir interfaces de usuario basadas en componentes reutilizables. Permite crear aplicaciones web rápidas y escalables mediante un enfoque declarativo y unidireccional en el flujo de datos [1]

MongoDB es una base de datos NoSQL orientada a documentos que almacena datos en formato BSON (una extensión de JSON). Ofrece escalabilidad horizontal, flexibilidad en el esquema y es adecuada para manejar grandes volúmenes de datos no estructurados [2].

Express.js es un framework web minimalista para Node.js que facilita la creación de aplicaciones web y APIs. Proporciona una estructura robusta para manejar rutas, middleware y solicitudes HTTP de manera eficiente [3].

Node.js es un entorno de ejecución de JavaScript del lado del servidor, basado en el motor V8 de Chrome. Permite desarrollar aplicaciones de red escalables y de alto rendimiento mediante un modelo de I/O no bloqueante y orientado a eventos [3].

TypeScript es un lenguaje de programación desarrollado por Microsoft que extiende JavaScript añadiendo tipado estático opcional. Facilita la detección temprana de errores y mejora la mantenibilidad del código en aplicaciones de gran escala [4].

Patrones y arquitecturas

Atomic Design es una metodología de diseño de interfaces que descompone la UI en cinco niveles jerárquicos: átomos, moléculas, organismos, plantillas y páginas. Este enfoque promueve la reutilización de componentes y la coherencia en el diseño [5].

Flux es un patrón arquitectónico propuesto por Facebook para gestionar el flujo de datos en aplicaciones web. Se basa en un flujo unidireccional de datos y se compone de acciones, un despachador, tiendas (stores) y vistas, lo que facilita la gestión predecible del estado de la aplicación [6].

Clean Architecture es un enfoque de diseño de software que organiza el código en capas concéntricas, separando las responsabilidades y dependencias. Su objetivo es crear sistemas modulares, testables y mantenibles, donde la lógica de negocio se mantiene independiente de detalles de implementación como bases de datos o interfaces de usuario [7].

La arquitectura Cliente-Servidor es un modelo de diseño en el que los clientes (como aplicaciones web o móviles) solicitan servicios o recursos, y un servidor central los proporciona, procesando las solicitudes y enviando las respuestas. Esta separación permite distribuir responsabilidades y facilita la escalabilidad, mantenimiento y seguridad del sistema.

CARACTERÍSTICAS

El uso de tecnologías modernas y patrones de arquitectura bien definidos permite construir aplicaciones robustas, escalables y fáciles de mantener. Este enfoque no solo optimiza el desarrollo al ofrecer herramientas como JavaScript/TypeScript, React, Node.js y MongoDB, sino que también organiza el código de manera eficiente mediante patrones como Flux, Atomic Design y Clean Architecture. A continuación, se exploran las características clave que ofrecen estas tecnologías, su integración entre capas y cómo contribuyen a crear sistemas con un flujo de datos predecible, interfaces modulares y una arquitectura desacoplada, lo que facilita la evolución y el mantenimiento de las aplicaciones a largo plazo.

Características claves de utilizar estas tecnologías, patrones y arquitectura

1. Homogeneidad en el lenguaje

El uso de JavaScript/TypeScript en todo el stack permite un desarrollo full-stack unificado, lo que facilita la colaboración entre frontend y backend al emplear un mismo lenguaje. Esta homogeneidad reduce la curva de aprendizaje, mejora la coherencia del código y agiliza el proceso de desarrollo.

2. Tipado estático con TypeScript

El uso de TypeScript mejora la mantenibilidad y escalabilidad de las aplicaciones al ofrecer un sistema de tipado estático que detecta errores en el tiempo de compilación. Además, facilita la autodocumentación del código y fortalece el trabajo en equipo al prevenir errores comunes de tipado y mal uso de datos, promoviendo un desarrollo más robusto y eficiente.

3. Flujo natural de datos con JSON

MongoDB, Node/Express y React manejan de forma nativa estructuras tipo JSON, lo que elimina la necesidad de transformar datos entre el frontend, backend y la base de datos. Esta compatibilidad

simplifica la lógica de serialización y deserialización, agilizando el flujo de información y reduciendo la complejidad del desarrollo.

4. Desarrollo basado en componentes reutilizables (React + Atomic Design)

React facilita la reutilización de componentes y promueve un enfoque declarativo en la construcción de interfaces. Al integrarse con metodologías como Atomic Design, permite organizar el diseño desde elementos básicos hasta estructuras complejas, lo que favorece un desarrollo modular, mantenible y escalable.

5. Gestión de estado predecible (Flux)

El patrón Flux establece un flujo unidireccional de datos, lo que facilita la depuración de errores y proporciona un control más claro sobre los cambios en el estado de la aplicación. Este enfoque resulta especialmente útil en aplicaciones con flujos de datos complejos, como dashboards o formularios con múltiples pasos.

6. Arquitectura limpia y desacoplada (Clean Architecture)

Este enfoque divide la lógica de negocio en capas bien definidas, como entidades, casos de uso, controladores e infraestructura, lo que aísla la lógica central del negocio de los frameworks, bases de datos y servicios externos. Esto mejora la facilidad de hacer pruebas, la mantenibilidad y escalabilidad de la aplicación, además de permitir reemplazar tecnologías sin afectar el dominio central del sistema.

7. Escalabilidad y rendimiento del backend (Node.js + Express)

Node.js ofrece un modelo event-driven y non-blocking, lo que lo hace ideal para aplicaciones en tiempo real, como chats, dashboards y notificaciones. Por su parte, Express proporciona una estructura flexible y ligera para construir APIs RESTful de manera rápida, siendo escalable tanto horizontal como verticalmente.

8. Base de datos flexible y escalable (MongoDB)

No requiere esquemas rígidos, lo que facilita realizar cambios en la estructura de los datos durante el desarrollo. Es ideal para aplicaciones que manejan datos semiestructurados o en constante evolución, y ofrece compatibilidad con la nube, siendo fácil de escalar horizontalmente para adaptarse a necesidades de crecimiento dinámico.

React

- Componentes reutilizables: Permite construir interfaces de usuario mediante componentes modulares y reutilizables.
- DOM virtual: Optimiza el rendimiento al minimizar las actualizaciones reales del DOM.
- Flujo de datos unidireccional: Facilita la gestión predecible del estado de la aplicación.

MongoDB

- Modelo de documentos flexible: Almacena datos en formato BSON (Binary JSON), permitiendo esquemas dinámicos.
- Escalabilidad horizontal: Diseñada para distribuir datos a través de múltiples servidores.
- Integración natural con JSON: Facilita la comunicación con aplicaciones JavaScript.

Express.js

- Framework minimalista: Proporciona una estructura ligera para desarrollar aplicaciones web y APIs.
- Middleware personalizable: Permite gestionar solicitudes HTTP de manera flexible.
- Integración con Node.js: Aprovecha las capacidades de Node.js para construir servidores eficientes.

Node.js

- Entorno de ejecución asíncrono: Utiliza un modelo de I/O no bloqueante para manejar múltiples conexiones simultáneamente.
- Basado en el motor V8 de Chrome: Ofrece un rendimiento rápido en la ejecución de código JavaScript.
- Amplio ecosistema de módulos: Facilita la incorporación de funcionalidades adicionales mediante npm.

TypeScript

- Tipado estático opcional: Ayuda a detectar errores en tiempo de compilación.
- Mejora la mantenibilidad: Facilita la comprensión y refactorización del código.
- Compatibilidad con JavaScript: Permite una adopción gradual en proyectos existentes.

Atomic Design

- Estructura jerárquica: Divide la interfaz en cinco niveles: átomos, moléculas, organismos, plantillas y páginas.
- Reutilización de componentes: Promueve la creación de elementos reutilizables y consistentes.
- Facilita el diseño sistemático: Ayuda a mantener la coherencia en interfaces complejas.

Flux

- Flujo de datos unidireccional: Los datos fluyen en una sola dirección, lo que simplifica el seguimiento de cambios.
- Componentes principales: Incluye acciones, despachador, tiendas (stores) y vistas.
- Gestión predecible del estado: Facilita el manejo del estado en aplicaciones complejas.

Clean Architecture

- Separación de responsabilidades: Organiza el código en capas independientes (entidades, casos de uso, interfaces, frameworks).
- Independencia tecnológica: Permite que la lógica de negocio no dependa de detalles de implementación como bases de datos o interfaces de usuario.
- Facilita las pruebas: Cada capa puede ser testeada de forma aislada, mejorando la calidad del software.

HISTORIA Y EVOLUCIÓN

El stack MERN es una combinación de tecnologías que ha ganado popularidad debido a su capacidad para ofrecer soluciones completas y eficientes para el desarrollo de aplicaciones web modernas. La estructura MERN está compuesta por cuatro componentes clave: MongoDB, Express.js, React y Node.js. Cada uno de estos componentes tiene una historia propia que ha contribuido al desarrollo y consolidación de este stack como una de las opciones preferidas en la creación de aplicaciones web.

Node.js, creado por Ryan Dahl en 2009, permitió ejecutar JavaScript en el lado del servidor, lo que marcó un antes y un después en el desarrollo web. Basado en el motor V8 de Chrome, Node.js introdujo un modelo de ejecución asíncrono y event-driven que permitió manejar múltiples conexiones simultáneamente de manera eficiente, ideal para aplicaciones en tiempo real [9]. Antes de Node.js, JavaScript era principalmente un lenguaje utilizado en el frontend, pero la capacidad de ejecutarlo en el servidor abrió nuevas posibilidades para los desarrolladores, quienes ya no necesitaban aprender lenguajes diferentes para trabajar en el frontend y backend. Node.js se integró rápidamente con otras tecnologías, como MongoDB y Express, creando un entorno de desarrollo de alta escalabilidad y rendimiento.

MongoDB nació en 2009, desarrollado por la empresa 10gen (actualmente MongoDB Inc.). Su creación respondió a la necesidad de contar con una base de datos más flexible y escalable que las tradicionales bases de datos relacionales, que no podían manejar fácilmente los datos no estructurados o semiestructurados que emergen en las aplicaciones web modernas [10]. MongoDB adoptó un modelo NoSQL orientado a documentos, usando BSON (Binary JSON), lo que le permitió almacenar datos en un formato que se alinea perfectamente con el uso de JSON en JavaScript. Esta base de datos se destacó por su flexibilidad en cuanto a la estructura de los datos y su escalabilidad horizontal, lo que la hacía ideal para aplicaciones en tiempo real y de gran volumen de datos. Su integración natural con aplicaciones JavaScript permitió un flujo de datos sin necesidad de realizar complejas transformaciones entre capas.

En 2010, Express.js fue creado por TJ Holowaychuk como un framework minimalista para Node.js. Inspirado por el framework Sinatra de Ruby, Express fue diseñado para simplificar la construcción de aplicaciones web y APIs, proporcionando una capa ligera para el manejo de rutas, middleware y solicitudes HTTP. Su llegada fue un hito importante en la evolución de Node.js, ya que permitió a los desarrolladores crear servidores y manejar peticiones de manera eficiente y sencilla. En poco tiempo, Express se consolidó como el framework de facto para Node.js, y su popularidad creció con la adopción de stacks como MEAN (MongoDB, Express, Angular, Node) y, más tarde, MERN. Este stack ofrecía una solución coherente y flexible para aplicaciones web modernas, permitiendo una integración fluida entre el backend y el frontend.

React, desarrollado por Facebook (hoy Meta) y lanzado como código abierto en 2013, cambió por completo la manera en que se construían las interfaces de usuario en aplicaciones web. Esta biblioteca frontend se centró en la creación de interfaces basadas en componentes reutilizables, lo que facilitó la creación de aplicaciones más modulares y mantenibles. React introdujo conceptos innovadores, como el Virtual DOM, que optimiza las actualizaciones de la vista al minimizar la interacción directa con el DOM real, mejorando el rendimiento de las aplicaciones. Además, su enfoque declarativo permitió a los desarrolladores describir cómo debería verse la interfaz en función del estado de la aplicación, simplificando la gestión del estado y mejorando la experiencia de desarrollo.

La popularidad de React creció rápidamente, y su adopción masiva comenzó a reemplazar a AngularJS en muchos proyectos. Este cambio de paradigma en la construcción de interfaces, sumado al poder de MongoDB, Express y Node.js en el backend, dio lugar al acrónimo MERN. A partir de la segunda mitad de la década de 2010, MERN se consolidó como una de las combinaciones más populares para el desarrollo de aplicaciones web full-stack, ofreciendo una solución unificada basada completamente en JavaScript.

Aunque TypeScript no forma parte del acrónimo MERN original, su adopción ha sido fundamental en la evolución del stack. Desarrollado por Microsoft y lanzado en 2012, TypeScript es un superset de JavaScript que agrega tipado estático opcional y otras características avanzadas. Esta capacidad de añadir tipado estático ha resultado crucial para mejorar la calidad del código, especialmente en aplicaciones de gran escala. TypeScript permite a los desarrolladores detectar errores en tiempo de compilación, lo que reduce la cantidad de bugs en tiempo de ejecución y facilita la refactorización y mantenimiento del código. Su integración con React y Node.js ha hecho que muchos proyectos en MERN adopten TypeScript para obtener una mayor robustez, especialmente cuando se trabaja en equipos grandes.

A medida que las tecnologías en el stack MERN han madurado, se ha consolidado como una solución poderosa para el desarrollo de aplicaciones web modernas. La principal ventaja de MERN radica en su uso completo de JavaScript, lo que permite a los desarrolladores trabajar con un solo lenguaje tanto en el frontend como en el backend, reduciendo la complejidad y mejorando la cohesión del código. Con la inclusión de TypeScript, este stack ha ganado aún más en robustez, escalabilidad y mantenibilidad.

Además, las tecnologías que componen MERN, como React y MongoDB, siguen evolucionando para adaptarse a nuevas demandas y desafíos del desarrollo web. La tendencia hacia aplicaciones más interactivas, dinámicas y en tiempo real sigue impulsando la popularidad de este stack, y su uso sigue siendo relevante para el desarrollo de aplicaciones empresariales, aplicaciones de redes sociales, e-commerce y mucho más.

Desde sus orígenes con MongoDB en 2009 hasta la consolidación de MERN en la década de 2010, estas tecnologías han sido adoptadas ampliamente por la comunidad de desarrollo, y su evolución continúa marcando el rumbo del futuro del desarrollo full-stack.

VENTAJAS Y DESVENTAJAS

El stack MERN (MongoDB, Express.js, React y Node.js) es una opción poderosa y flexible para el desarrollo de aplicaciones web modernas, que ha ganado una notable popularidad debido a su

capacidad para unificar el desarrollo de frontend y backend utilizando JavaScript o TypeScript. Este enfoque proporciona numerosas ventajas, pero también implica ciertos desafíos que deben ser considerados al tomar una decisión sobre su adopción en un proyecto. A continuación, se presentan las principales ventajas y desventajas de utilizar el stack MERN, sus tecnologías y los patrones arquitectónicos que lo acompañan.

Una de las principales ventajas del stack MERN es la posibilidad de usar un solo lenguaje a lo largo de todo el proceso de desarrollo. Con JavaScript o TypeScript, los desarrolladores pueden trabajar tanto en el frontend como en el backend, así como en la base de datos, que utiliza un formato similar a JSON (BSON) en MongoDB. Esta unificación del lenguaje elimina la necesidad de cambiar de contexto y de aprender diferentes lenguajes para cada capa del stack. Esto no solo simplifica el proceso de desarrollo, sino que también mejora la productividad y facilita el mantenimiento del código, al tener una única base de conocimiento para todo el proyecto. Además, el uso de TypeScript aporta una ventaja adicional con su tipado estático, lo que mejora la calidad del código y reduce los errores en tiempo de ejecución, especialmente cuando el proyecto crece en complejidad.

Otra ventaja clave de MERN es su enfoque en el desarrollo modular del frontend, gracias a React. React permite la creación de interfaces de usuario a través de componentes reutilizables, lo que facilita la construcción de aplicaciones complejas sin tener que duplicar código. Este enfoque modular no solo acelera el desarrollo de la interfaz, sino que también mejora la mantenibilidad del software a largo plazo. Los componentes pueden ser organizados y gestionados en herramientas como Storybook, lo que permite su reutilización eficiente y su documentación adecuada. Esto hace que el trabajo en equipo sea más efectivo, ya que los desarrolladores pueden compartir y modificar componentes sin interferir con otras partes del proyecto. Además, React facilita la creación de interfaces dinámicas y reactivas, lo que mejora la experiencia del usuario.

Una característica importante de MERN es el intercambio eficiente de datos en formato JSON. MongoDB, como base de datos NoSQL, almacena los datos en un formato BSON (Binary JSON), que es perfectamente compatible con las APIs RESTful que utilizan JSON para el intercambio de datos. Este alineamiento natural entre el formato de almacenamiento de datos y el formato de intercambio entre cliente y servidor reduce la complejidad del proceso de desarrollo, ya que no es necesario realizar costosas conversiones de formato. La compatibilidad nativa con JSON también mejora el rendimiento general de la aplicación, ya que se evitan las transformaciones de datos que pueden generar cuellos de botella. Además, el uso de JSON hace que la integración con otros sistemas y servicios sea más sencilla, dado que JSON es un formato ampliamente adoptado en la industria.

El stack MERN también se beneficia de una comunidad activa y un ecosistema robusto. Cada una de las tecnologías que lo componen es open-source y cuenta con una amplia base de usuarios y desarrolladores que contribuyen al desarrollo de nuevas funcionalidades, así como a la creación de bibliotecas y herramientas de terceros. Por ejemplo, React y Node.js son dos de los proyectos más populares en plataformas como GitHub, lo que asegura que siempre haya una gran cantidad de recursos disponibles, como tutoriales, documentación, librerías y paquetes. El ecosistema de NPM (Node Package Manager) es un factor clave en este sentido, ya que ofrece una vasta cantidad de herramientas que los desarrolladores pueden integrar fácilmente en sus aplicaciones, lo que agiliza el desarrollo y evita reinventar soluciones ya existentes. Además, el hecho de que estas tecnologías sean de código abierto significa que los desarrolladores tienen acceso al código fuente y pueden modificarlo o contribuir al proyecto, lo que fomenta una colaboración continua y la evolución de las herramientas.

Una ventaja adicional del stack MERN es su flexibilidad y escalabilidad. A diferencia de otros stacks más opinados, como Ruby on Rails o MEAN, MERN no impone una estructura predeterminada de proyecto. Esto permite a los desarrolladores elegir la arquitectura que mejor se adapte a sus necesidades, como Clean Architecture o el patrón MVC (Modelo-Vista-Controlador). Esta libertad arquitectónica es ideal para proyectos que requieren una alta personalización, ya que los desarrolladores pueden organizar el código de la manera más eficiente posible según el caso de uso. Además, Node.js, que se basa en un modelo asíncrono y event-driven, es extremadamente eficiente para manejar aplicaciones con altas demandas de concurrencia. Su capacidad para gestionar múltiples conexiones simultáneas sin bloquear el hilo principal lo convierte en una opción ideal para aplicaciones web de alto rendimiento. MongoDB también es altamente escalable, ya que permite la replicación y el sharding, lo que facilita la expansión horizontal de la base de datos cuando es necesario manejar grandes volúmenes de datos.

Sin embargo, a pesar de sus numerosas ventajas, el stack MERN también presenta ciertos desafíos. Uno de los principales es la curva de aprendizaje pronunciada. Para un desarrollador que no esté familiarizado con estas tecnologías, aprender cómo interactúan entre sí MongoDB, Express.js, React y Node.js puede ser una tarea abrumadora. Además, el uso de TypeScript agrega una capa adicional de complejidad, ya que requiere comprender conceptos como tipado estático, interfaces y clases genéricas. Aunque JavaScript es un lenguaje ampliamente conocido, su uso en diferentes capas del stack MERN requiere que el desarrollador se familiarice con una variedad de herramientas y patrones que pueden resultar complejos, especialmente si se es nuevo en el desarrollo web full-stack.

Otra desventaja significativa de MERN es la falta de una estructura predeterminada en el proyecto. Mientras que otros frameworks, como Ruby on Rails o Angular, imponen una organización clara del código y del flujo de trabajo, MERN deja a los desarrolladores la responsabilidad de definir la estructura del proyecto. Esto puede ser beneficioso en términos de flexibilidad, pero también puede llevar a problemas de mantenimiento a largo plazo si no se siguen buenas prácticas desde el principio. Sin una estructura bien definida, los proyectos MERN pueden volverse difíciles de gestionar, especialmente cuando crecen en tamaño y complejidad. Para mitigar este problema, es recomendable adoptar patrones arquitectónicos establecidos, como Clean Architecture, y asegurarse de que todos los miembros del equipo sigan las mismas convenciones.

El mantenimiento de múltiples partes móviles es otro desafío asociado con el uso de MERN. Al estar compuesto por varias tecnologías, cada una de ellas puede tener su propio ciclo de actualizaciones y mejoras. Esto significa que los desarrolladores deben estar constantemente al tanto de las nuevas versiones de MongoDB, Express, React, Node.js y TypeScript. Además, las actualizaciones en una de estas tecnologías pueden introducir cambios incompatibles con otras partes del stack, lo que requiere tiempo y esfuerzo para ajustar el código y asegurarse de que todo funcione correctamente. Este desafío es particularmente relevante en proyectos grandes, donde el impacto de las actualizaciones puede ser significativo.

Además, aunque Node.js es extremadamente eficiente para aplicaciones basadas en I/O, no es la mejor opción para tareas que requieren un procesamiento intensivo en la CPU. Su naturaleza de un solo hilo (single-threaded) significa que puede haber limitaciones en cuanto a la capacidad de realizar cálculos pesados o procesamiento paralelo. Para estos casos, se pueden implementar soluciones adicionales, como usar workers o procesos separados, pero esto puede agregar complejidad y requerir más recursos. Además, MongoDB, al ser una base de datos NoSQL, es muy flexible, pero puede no ser tan eficiente como una base de datos SQL optimizada para realizar consultas complejas y

transacciones ACID, lo que puede ser un inconveniente en aplicaciones que requieren una manipulación de datos avanzada.

Finalmente, el ecosistema de JavaScript está en constante evolución, lo que puede ser tanto una ventaja como una desventaja. Si bien la innovación continua puede generar nuevas funcionalidades y mejoras, también puede hacer que las prácticas anteriores queden obsoletas rápidamente. En el frontend, por ejemplo, React ha cambiado de clases a hooks en los últimos años, lo que ha generado la necesidad de adaptarse a nuevas formas de escribir el código. De manera similar, en el backend, nuevas tecnologías como Koa, Fastify y Nest.js han comenzado a competir con Express, lo que obliga a los desarrolladores a estar al tanto de las últimas tendencias y a evaluar constantemente nuevas herramientas.

CASOS DE USO (SITUACIONES Y/O PROBLEMAS DONDE SE PUEDEN APLICAR)

El stack MERN es ideal para aplicaciones web que requieren desarrollo rápido, interactividad dinámica y escalabilidad. Su uso unificado de JavaScript a lo largo de todas las capas facilita la integración entre frontend y backend, lo que lo convierte en una opción eficiente para varios tipos de proyectos.

Uno de los casos más comunes para MERN es en aplicaciones CRUD de negocio, como sistemas de gestión de productos o pedidos para tiendas en línea. Gracias a React en el frontend y MongoDB en el backend, es posible crear interfaces interactivas y administrar grandes volúmenes de datos de manera eficiente, ofreciendo una experiencia de usuario fluida y bien integrada.

MERN también se adapta bien a aplicaciones que requieren interactividad en tiempo real moderada, como chats en línea o tableros colaborativos. Aunque no es ideal para aplicaciones de tiempo real intensivo, MERN puede manejar actualizaciones en vivo utilizando técnicas como polling o bibliotecas como Socket.io, lo que permite una comunicación eficiente entre el cliente y el servidor.

Además, el stack es muy popular en el desarrollo de prototipos y PMV (Producto Mínimo Viable) para startups, gracias a su rapidez de implementación. Al utilizar un solo lenguaje en todo el stack, los equipos pueden centrarse en construir versiones funcionales del producto sin complicaciones adicionales, permitiendo un lanzamiento ágil y económico.

Otro caso de uso importante es en plataformas web con contenido dinámico generado por los usuarios, como redes sociales, foros o blogs. MERN permite actualizar contenido en tiempo real sin necesidad de recargar la página, lo que mejora la experiencia del usuario. MongoDB almacena eficientemente los datos, mientras que React y Node.js se encargan de las interacciones en el frontend y el backend, respectivamente.

CASOS DE APLICACIÓN (EJEMPLOS Y CASOS DE ÉXITO EN LA INDUSTRIA)

El stack MERN ha ganado popularidad en diversos sectores debido a su capacidad para ofrecer soluciones rápidas, escalables e interactivas. Muchas empresas, desde startups hasta grandes corporaciones, han adoptado este conjunto de tecnologías para construir aplicaciones robustas y de

alto rendimiento. A continuación, se presentan algunos casos de éxito que destacan el uso de MERN en la industria.

FreeCodeCamp

FreeCodeCamp, una de las plataformas de aprendizaje de programación más populares, es un ejemplo claro de la efectividad de MERN en una aplicación de código abierto a gran escala. Utiliza Node.js y Express en el backend para manejar la lógica de servidor, mientras que React potencia su frontend, ofreciendo una interfaz de usuario dinámica y fluida. Además, MongoDB almacena la información de los usuarios, lo que facilita el manejo de grandes volúmenes de datos. Con millones de usuarios activos, FreeCodeCamp ha demostrado que MERN puede escalar para satisfacer las demandas de plataformas educativas de alto tráfico.

MercadoLibre

MercadoLibre, el principal marketplace en Latinoamérica, ha integrado varias tecnologías de la pila MERN en su plataforma. Aunque no todo su stack está compuesto por MERN, la empresa hace uso extensivo de Node.js en microservicios y React en su interfaz de usuario. Esta adopción parcial del stack MERN ha permitido a MercadoLibre mantener una infraestructura eficiente y escalable, capaz de manejar grandes volúmenes de transacciones y tráfico, lo que es esencial en el sector del comercio electrónico.

Expedia

Expedia, gigante global del sector de viajes, ha utilizado el stack MERN para personalizar las recomendaciones de viaje en su plataforma. MongoDB se emplea para gestionar las preferencias de los usuarios, mientras que Node.js y Express facilitan la integración en tiempo real de servicios externos, como ofertas de vuelos y hoteles. React, por su parte, se encarga de la interfaz de usuario interactiva, lo que permite a los usuarios obtener información actualizada instantáneamente. Esta combinación ha mejorado significativamente la experiencia del usuario, proporcionando un servicio de viajes más personalizado y dinámico.

Sega

Sega, una de las compañías líderes en la industria de los videojuegos, ha adoptado MERN para desarrollar dashboards interactivos que permiten a los jugadores ver estadísticas en tiempo real. MongoDB almacena los datos del juego y las estadísticas de los usuarios, mientras que Node.js y Express gestionan las APIs que proveen acceso a esta información en tiempo real. React ofrece una interfaz de usuario altamente interactiva, permitiendo a los jugadores acceder a su desempeño sin interrupciones. Este uso de MERN demuestra su efectividad en entornos de alto rendimiento, como los videojuegos, donde la latencia y la capacidad de respuesta son críticas.

Coinbase

Coinbase, una de las plataformas más grandes de intercambio de criptomonedas, utiliza MERN en varios componentes de su infraestructura. Node.js y Express permiten mantener conexiones seguras y de baja latencia con los mercados de criptomonedas, mientras que MongoDB se encarga de almacenar grandes volúmenes de transacciones. React se utiliza en su panel de usuario para mostrar precios en

vivo y otros datos cruciales. Este uso de MERN destaca la capacidad del stack para manejar transacciones rápidas y seguras, lo cual es fundamental en el ámbito de las finanzas digitales.

eBay

eBay ha integrado tecnologías MERN en su plataforma para mejorar la experiencia de subastas y la gestión de listados en tiempo real. Node.js y Express proporcionan una API ágil para manejar las solicitudes de los usuarios, mientras que MongoDB soporta millones de transacciones de datos de productos. React se utiliza para crear una interfaz de usuario dinámica que permite a los compradores y vendedores interactuar sin interrupciones. El uso de MERN en eBay es un ejemplo de cómo este stack puede ser implementado en plataformas de alto volumen de usuarios y transacciones, garantizando rapidez y eficiencia.

¿QUÉ TAN COMÚN ES EL STACK DESIGNADO?

El stack MERN, compuesto por MongoDB, Express.js, React y Node.js, se ha consolidado como una de las combinaciones tecnológicas más populares y ampliamente adoptadas en el desarrollo web moderno. Su enfoque integral basado en JavaScript, tanto en el frontend como en el backend, ha facilitado su adopción por parte de desarrolladores y empresas que buscan eficiencia y coherencia en sus proyectos.

La popularidad del stack MERN se refleja en su capacidad para construir aplicaciones web sólidas y escalables. Según Oracle, esta pila tecnológica permite a los desarrolladores crear aplicaciones web robustas utilizando un único lenguaje de programación en todas las capas del desarrollo, lo que simplifica el proceso y mejora la productividad [11].

Además, la comunidad de desarrolladores ha reconocido la relevancia del stack MERN en la industria. En plataformas como LinkedIn, se destaca que el stack MERN ha ganado una inmensa popularidad en los últimos años debido a su eficiencia y versatilidad, convirtiéndose en una de las habilidades más demandadas para los desarrolladores [12].

Asimismo, una revisión exhaustiva en diversas plataformas de empleo como Computrabajo, LinkedIn y Glassdoor evidencia que el stack MERN continúa siendo altamente demandado en el mercado laboral actual. En particular, el desarrollo frontend con React y el uso de TypeScript se destacan como habilidades clave requeridas por numerosas empresas. De igual manera, Node.js en el backend mantiene una sólida presencia en ofertas laborales, lo que confirma la relevancia y aplicabilidad del stack MERN en múltiples sectores de la industria del desarrollo de software.

MATRIZ DE ANÁLISIS DE PRINCIPIOS SOLID

Principio SOLID	Nivel de Cumplimiento	Argumentación

S – Principio de Responsabilidad Única (SRP)	ALTO	Tecnologías como React (en el stack MERN) y patrones como Atomic Design fomentan la separación de responsabilidades al descomponer la UI en componentes reutilizables. Cada componente o módulo tiene una única responsabilidad, alineándose con este principio. En Clean Architecture, la separación entre capas refuerza esta práctica, permitiendo que cada clase o módulo cumpla una única función clara.
O – Principio de Abierto/Cerrado (OCP)	ALTO	Clean Architecture y la arquitectura cliente-servidor permiten extender funcionalidades sin modificar el núcleo del sistema. En MERN, es común agregar nuevas rutas o controladores (Express), o nuevos componentes (React) sin alterar la lógica existente. Flux también permite introducir nuevas acciones y stores sin afectar las vistas actuales.
L – Principio de Sustitución de Liskov (LSP)	MEDIO	Aunque se promueve el uso de interfaces (puertos en Clean Architecture y componentes en React), su cumplimiento puede verse limitado si las implementaciones no respetan completamente los contratos definidos. En algunos casos, un componente o servicio puede comportarse de manera inesperada al ser reemplazado, especialmente cuando no se realiza una validación exhaustiva de tipos, incluso con TypeScript.
I – Principio de Segregación de Interfaces (ISP)	ALTO	Este principio se cumple especialmente bien en Clean Architecture, donde los puertos definen interfaces específicas y granulares que los adaptadores implementan. En React, los componentes reciben solo las props necesarias, y en REST, los endpoints están diseñados para servir propósitos específicos, reduciendo dependencias innecesarias.
D – Principio de Inversión de Dependencias (DIP)	ALTO	En Clean Architecture, el dominio depende de abstracciones y no de detalles concretos. MERN permite aplicar este principio mediante la inyección de dependencias en Express o servicios en React. Flux también lo respeta al separar el flujo de datos en capas independientes. La arquitectura cliente-servidor y REST cumplen este principio al desacoplar cliente y backend a través de contratos API bien definidos.

El uso combinado del stack MERN con TypeScript, junto con arquitecturas como Clean Architecture, cliente-servidor, Atomic Design y Flux, favorece significativamente la aplicación de los principios SOLID. Gracias a la modularidad y separación de responsabilidades en todas las capas del sistema,

principios como SRP e ISP se cumplen de manera destacada. La estructura de componentes en React, el uso de puertos e interfaces en el backend y la organización por capas en Clean Architecture contribuyen a un diseño claro y enfocado. Aunque el principio de sustitución de Liskov (LSP) puede no aplicarse rigurosamente en todos los casos —por ejemplo, en componentes UI altamente personalizados—, este impacto se reduce mediante contratos bien definidos. En general, la inversión de dependencias y la orientación modular permiten mantener un código extensible, mantenible y alineado con buenas prácticas de diseño de software.

MATRIZ DE ANÁLISIS DE ATRIBUTOS DE CALIDAD

Atributo de calidad	Efecto	Argumentación
Disponibilidad	+	La arquitectura sin estado (RESTful), que no almacena datos de sesión en el servidor, en Node/Express y la replicación de MongoDB facilitan la alta disponibilidad. El backend puede escalarse horizontalmente y React distribuye carga al cliente.
Eficiencia/Rendimiento	+/-	Node.js es eficiente en operaciones de entrada/salida, MongoDB optimiza con índices y React usa Virtual DOM. Sin embargo, el modelo single-thread de Node puede afectar el rendimiento en operaciones intensivas en CPU como procesamiento de imágenes o cifrado.
Escalabilidad	+	Todas las tecnologías permiten escalabilidad horizontal: Node.js detrás de balanceadores, MongoDB con sharding y React procesando del lado del cliente.
Mantenibilidad	+	TypeScript y Clean Architecture promueven modularidad y separación de responsabilidades. Flux y Atomic Design estructuran el frontend, facilitando el mantenimiento.
Seguridad	+/-	La seguridad depende de la implementación. Existen herramientas y prácticas recomendadas como helmet, JWT, CORS, cifrado en MongoDB y validaciones en Express, pero no se aplican automáticamente.
Usabilidad	+	React permite crear interfaces ricas e intuitivas. Atomic Design asegura consistencia visual. Las SPA mejoran la experiencia del usuario con actualizaciones rápidas.
Portabilidad	+	El stack está basado en tecnologías web estándar. Node, React y MongoDB son multiplataforma y compatibles con contenedores Docker.

Reusabilidad	+	Atomic Design promueve componentes reutilizables en la UI. En el backend, la lógica de negocio es independiente del framework. TypeScript facilita compartir tipos entre proyectos.
Robustez	+/-	TypeScript, validaciones y manejo de errores en Express aportan a la robustez. Sin embargo, se requiere aplicar buenas prácticas como no bloquear el event loop, optimizar consultas y manejar adecuadamente las promesas.
Capacidad de prueba	+	Clean Architecture separa lógica de dependencias, lo que permite pruebas unitarias y de integración más sencillas. En frontend, la estructura modular facilita el testeo.
Flexibilidad	+	La arquitectura modular permite cambiar tecnologías o adaptarse a nuevos requerimientos fácilmente. Por ejemplo, se puede reemplazar MongoDB u otras capas sin afectar la lógica central.
Integridad	+	Gracias a la validación de datos en múltiples capas (Express, lógica de negocio, MongoDB) y al uso de tipos en TypeScript, se puede mantener integridad entre módulos.
Interoperabilidad	+	Express permite integración sencilla con APIs externas. MongoDB puede conectarse con distintos clientes. React consume APIs REST fácilmente. Clean Architecture facilita esta integración mediante adaptadores bien definidos.
Fiabilidad	+	Las capacidades de replicación de MongoDB, el manejo centralizado de errores en Express y las pruebas unitarias favorecen la fiabilidad del sistema.

La tabla presenta un análisis cualitativo de diversos atributos de calidad del software aplicados a un sistema desarrollado con el stack tecnológico MERN (MongoDB, Express, React y Node.js), complementado con principios de Clean Architecture, Flux y Atomic Design. Cada fila evalúa un atributo de calidad clave, indicando su efecto global (positivo, negativo o mixto) y proporcionando una argumentación técnica que justifica dicha valoración.

En general, se destaca que este stack favorece atributos como la disponibilidad, escalabilidad, mantenibilidad, portabilidad, usabilidad, reusabilidad, flexibilidad, capacidad de prueba, integridad, interoperabilidad y fiabilidad, gracias a sus componentes desacoplados, uso de herramientas modernas, soporte de contenedores y estructuras bien definidas en el frontend y backend.

Sin embargo, se observan efectos mixtos en eficiencia/rendimiento, seguridad y robustez, los cuales dependen fuertemente de la implementación específica. Por ejemplo, aunque Node.js es eficiente en I/O, su modelo single-threaded puede ser limitante en tareas intensivas en CPU, y las prácticas de seguridad no vienen aplicadas por defecto.

MATRIZ DE TÁCTICAS DE DISEÑO

La siguiente tabla analiza cómo las principales tecnologías del stack MERN (MongoDB, Express, React, Node.js) y el protocolo REST/JSON, dentro de una arquitectura cliente-servidor, soportan diversas tácticas de diseño arquitectónico. Se asigna un nivel de soporte (Alto, Medio, Bajo, N/A) a cada tecnología según su contribución a las propiedades de calidad del sistema distribuido.

Táctica	Node.js / Express	Mongo DB	React	REST/ JSON	Cliente/ Servidor	Comentario clave
Optimización de consultas	Medio	Alto	N/A	Medio	Medio	MongoDB permite consultas eficientes con índices; Express necesita buenas prácticas para optimizar.
Manejo de concurrencia	Alto	Medio	Bajo	Alto	Medio	Node.js usa un event loop eficiente, pero puede bloquearse con tareas intensivas en CPU. REST permite múltiples solicitudes paralelas sin estado.
Manejo de errores y fallos	Alto	Medio	Medio	Medio	Medio	Express facilita el manejo centralizado de errores; MongoDB replica datos pero requiere manejo de reconexión; React puede capturar errores de renderizado.
Escalabilidad	Alto	Alto	Medio	Alto	Alto	Todo el stack permite escalado horizontal mediante balanceadores, sharding y separación de responsabilidades.
Facilidad de integración	Alto	Alto	Alto	Alto	Alto	REST/JSON y Clean Architecture permiten integración sencilla con servicios externos y capas desacopladas.

Seguridad	Medio	Medio	Medio	Medio	Medio	Hay buenas herramientas disponibles (helmet, JWT, roles en MongoDB, escaping en React), pero requieren implementación cuidadosa.
Replicación	N/A	Alto	N/A	N/A	N/A	MongoDB soporta replicación nativa para alta disponibilidad. Las demás capas no requieren replicación por diseño sin estado.
Monitoreo	Medio	Alto	Bajo	Medio	Medio	MongoDB (especialmente en Atlas) ofrece monitoreo avanzado; Node.js requiere integración de APMs; React necesita herramientas externas como Sentry.

Esta matriz proporciona una visión integral de cómo cada componente tecnológico del stack MERN contribuye a las tácticas arquitectónicas clave para sistemas distribuidos modernos:

- Optimización de consultas es especialmente fuerte en MongoDB, gracias a su sistema de índices, agregaciones y proyecciones. Express puede contribuir si se usan buenas prácticas y caching, mientras que REST permite optimizar la transferencia de datos mediante filtros, paginación y reducción de payloads.

- Manejo de concurrencia está bien soportado por Node.js debido a su modelo asíncrono no bloqueante. REST/JSON al ser stateless facilita la concurrencia. MongoDB maneja concurrencia a nivel de documento. React es limitado en este aspecto, ya que corre sobre un hilo único del navegador.

- Manejo de errores y fallos es robusto en el backend, donde Express permite control centralizado de errores y reinicio de procesos con herramientas como PM2. MongoDB responde bien ante fallas a través de réplicas. En el frontend, React permite usar *error boundaries* para capturar fallos de renderizado. REST proporciona códigos estándar (400, 500, etc.) para notificar errores, pero el manejo activo depende del cliente.

- Escalabilidad es una de las grandes fortalezas del stack MERN: Express y Node.js permiten balanceo horizontal; MongoDB soporta sharding; y React distribuye la carga al ejecutarse en el cliente. REST, al no mantener estado, también permite escalar fácilmente el backend.

- Facilidad de integración es alta en todas las capas, debido al uso de estándares como JSON y HTTP, la modularidad del stack y la abundancia de librerías compatibles. Esto permite construir arquitecturas desacopladas y servicios que se integran fácilmente entre sí.

- Seguridad tiene soporte medio por defecto. Cada tecnología incluye herramientas (helmet, autenticación con JWT, validación de inputs, CORS, etc.), pero su efectividad depende de una implementación explícita y rigurosa.

- Replicación se gestiona principalmente a nivel de base de datos. MongoDB ofrece replicación nativa de datos para garantizar alta disponibilidad. Las otras capas no manejan replicación directa por su diseño stateless.

- Monitoreo es sólido en MongoDB (especialmente si se usa Atlas) y aceptable en el backend si se integran herramientas como New Relic, Grafana o PM2. El frontend requiere servicios externos para monitorear errores y rendimiento en el cliente.

MATRIZ DE ANÁLISIS DE PATRONES

Patrón de Diseño	Aplicación en el Stack
MVC (Model-View-Controller)	Backend: Express como controlador, lógica de negocio como modelo, y respuesta JSON como vista. En frontend: React + Flux sigue un modelo inspirado en MVVM.
Observer	Flux: las vistas se suscriben a los stores. React: renderizado reactivo al cambio de estado. Backend: uso puntual con Eventemitter.
Dependency Injection / Inversión de Dependencias	Backend: casos de uso reciben repositorios inyectados (Clean Architecture). Frontend: React Context puede verse como una forma ligera de inyección.
Singleton	Backend: conexión MongoDB compartida, servicios únicos gracias al sistema de módulos de Node. Frontend: stores en Flux como instancia única.
Factory	Backend: funciones factory para construir middlewares o repositorios según entorno. Frontend: inicialización de hooks o utilidades configurables.
Composite	Frontend: React y Atomic Design modelan componentes complejos como combinación de otros simples (jerarquía de átomos, moléculas, organismos).

Adapter	Backend: controladores y repositorios funcionan como adaptadores (entrada/salida) en Clean Architecture. Frontend: capas que adaptan respuestas de APIs.
Facade	Backend: API REST actúa como fachada simplificada del sistema interno. Frontend: funciones que encapsulan múltiples llamadas a APIs.
Middleware / Chain of Responsibility	Backend: Express procesa requests a través de una cadena de middlewares. Se aplican para autenticación, validaciones, errores, etc.

A lo largo del desarrollo del sistema basado en el stack MERN, se aplicaron diversos patrones de diseño con el objetivo de promover una arquitectura modular, desacoplada y mantenible. Aunque React no implementa el patrón Modelo–Vista–Controlador (MVC) de forma estricta, el backend en Express presenta una organización funcionalmente equivalente: los controladores gestionan las solicitudes HTTP (Controller), la lógica de negocio se encapsula en los casos de uso o servicios (Model), y las respuestas en formato JSON representan la capa de presentación (View). En el cliente, el patrón Flux sigue principios derivados del Modelo–Vista–Presentador (MVP), con vistas que se suscriben a almacenes de estado (stores) y reaccionan ante cambios.

El patrón Observer se evidencia en la gestión del estado tanto en React como en Flux, donde los componentes se actualizan automáticamente en respuesta a modificaciones del estado. En el servidor, se hace uso puntual de mecanismos de observación a través de EventEmitter para notificaciones internas desacopladas. El principio de Inversión de Dependencias se aplica principalmente en el backend, siguiendo los lineamientos de la Arquitectura Limpia: los casos de uso reciben sus dependencias externas (como los repositorios) mediante inyección explícita, lo que facilita las pruebas unitarias y reduce el acoplamiento. En el frontend, el uso de contextos en React permite emular este principio, proporcionando dependencias de forma jerárquica a los componentes.

El patrón Singleton se encuentra implementado en elementos como la conexión única a la base de datos MongoDB, compartida entre los distintos repositorios, y en los stores de estado de React, que se instancian una sola vez a lo largo del ciclo de vida de la aplicación. El patrón Factory se aplica mediante funciones que generan instancias configuradas de componentes, como middlewares o repositorios, en función del entorno de ejecución. El patrón Composite es inherente al enfoque de Atomic Design adoptado en la construcción de la interfaz de usuario, donde componentes complejos se conforman por la composición jerárquica de elementos más simples.

Asimismo, el patrón Adapter se implementa en múltiples capas: los controladores del backend actúan como adaptadores de entrada que traducen solicitudes HTTP a llamadas de casos de uso, mientras que los repositorios actúan como adaptadores de salida, traduciendo operaciones abstractas en consultas concretas a MongoDB. En el cliente, se utilizan adaptadores para transformar la estructura de las respuestas de la API a un formato adecuado para los stores. El patrón Facade se refleja en la API REST del backend, que expone una interfaz simplificada para operaciones complejas del sistema. Finalmente, el patrón Middleware, correspondiente a la cadena de responsabilidad, se emplea en el backend mediante el pipeline de middlewares de Express, permitiendo procesar solicitudes en etapas, con funciones como autenticación, validación y manejo de errores.

MATRIZ DE ANÁLISIS DE MERCADO LABORAL

Tecnología	Demanda (Empresas)	Nuevas Ofertas (Tendencia)	Salario Promedio Mensual (COP)	Competencia (Talento Disponible)
MongoDB	Alta	Alta	7.000.000 – 10.000.000	Media
Express/Node	Alta	Alta	6.000.000 – 9.000.000	Alta
React	Muy Alta	Muy Alta	3.000.000 – 9.000.000	Alta
Node.js	Muy Alta	Muy Alta	7.000.000 – 11.000.000	Alta
TypeScript	Alta	Muy Alta	4.000.000 – 5.500.000	Media

Demanda y Tendencias de Ofertas: Las tecnologías del stack MERN continúan siendo altamente demandadas en el mercado laboral colombiano. React y Node.js lideran en cuanto a nuevas ofertas, reflejando su adopción generalizada en el desarrollo web moderno. TypeScript ha experimentado un crecimiento notable en nuevas ofertas, evidenciando su creciente adopción en proyectos que buscan mayor robustez y mantenibilidad.

Salario Promedio: Los desarrolladores con experiencia en el stack MERN disfrutan de salarios competitivos. Por ejemplo, un desarrollador React puede ganar entre 3.000.000 y 9.000.000 COP mensuales, dependiendo de la experiencia y la empresa contratante. Los desarrolladores Node.js tienen un rango salarial similar, con promedios alrededor de 7.000.000 a 11.000.000 COP mensuales. En el caso de TypeScript, los salarios oscilan entre 4.000.000 y 5.500.000 COP mensuales, variando según la experiencia y la ubicación.

Competencia (Talento Disponible): La popularidad del stack MERN ha llevado a un aumento en la cantidad de desarrolladores que lo dominan, especialmente en niveles junior. Sin embargo, existe una menor disponibilidad de profesionales con experiencia sólida y habilidades avanzadas, lo que representa una oportunidad para aquellos que buscan destacarse en el mercado laboral.

DIAGRAMA ALTO NIVEL

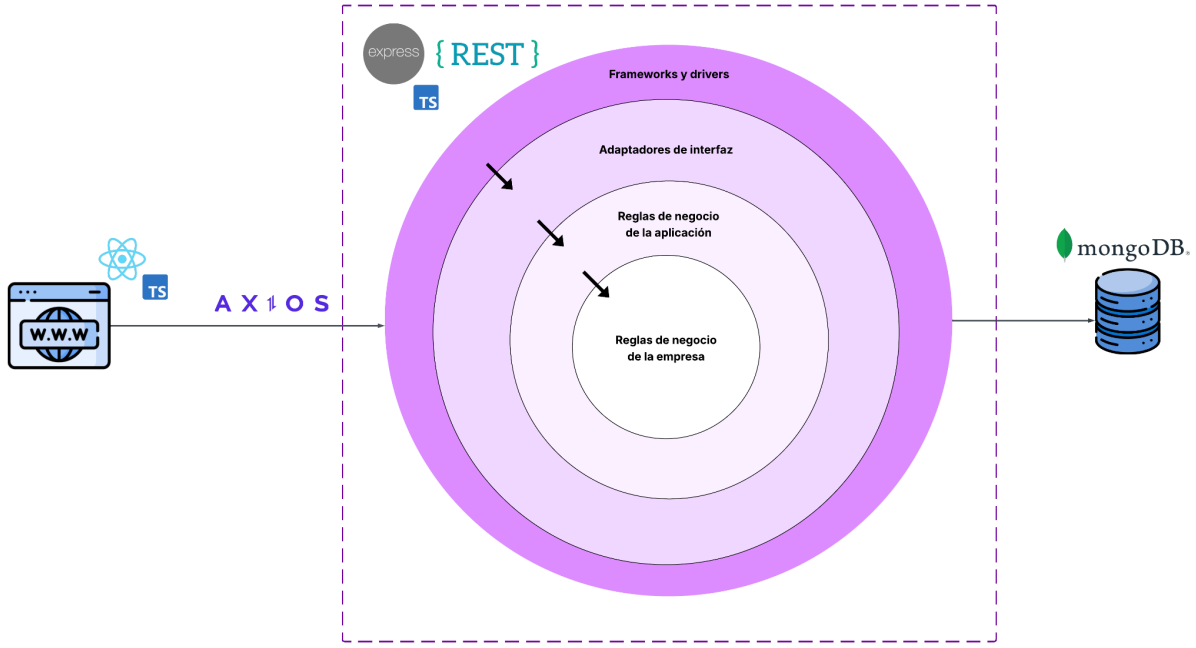

Imagen 1. Diagrama de alto nivel de la implementación realizada.

Este diagrama representa una arquitectura de software basada en el enfoque de Clean Architecture, la cual organiza la aplicación en capas concéntricas con el objetivo de separar las responsabilidades, reducir acoplamientos y aumentar la mantenibilidad del sistema. En el centro se encuentran las reglas de negocio de la empresa, que encapsulan la lógica más estable e independiente del sistema. Esta capa está rodeada por los casos de uso o lógica de aplicación, los cuales definen cómo se deben ejecutar los procesos del sistema en función de las reglas empresariales.

En una capa intermedia se encuentran los adaptadores de interfaz, responsables de traducir la información entre el dominio y los elementos externos como bases de datos o interfaces de usuario. Finalmente, en la capa más externa se ubican los frameworks y drivers, donde residen tecnologías como Express (para el backend en Node.js con TypeScript), MongoDB como base de datos, y React (también en TypeScript) como framework del frontend.

La comunicación entre el frontend y el backend se realiza a través de peticiones HTTP utilizando la librería Axios, siguiendo un enfoque REST. La arquitectura garantiza una estructura desacoplada, facilitando pruebas unitarias, cambios tecnológicos y la evolución del sistema a largo plazo sin afectar su núcleo de negocio.

DIAGRAMA DE CONTEXTO C4

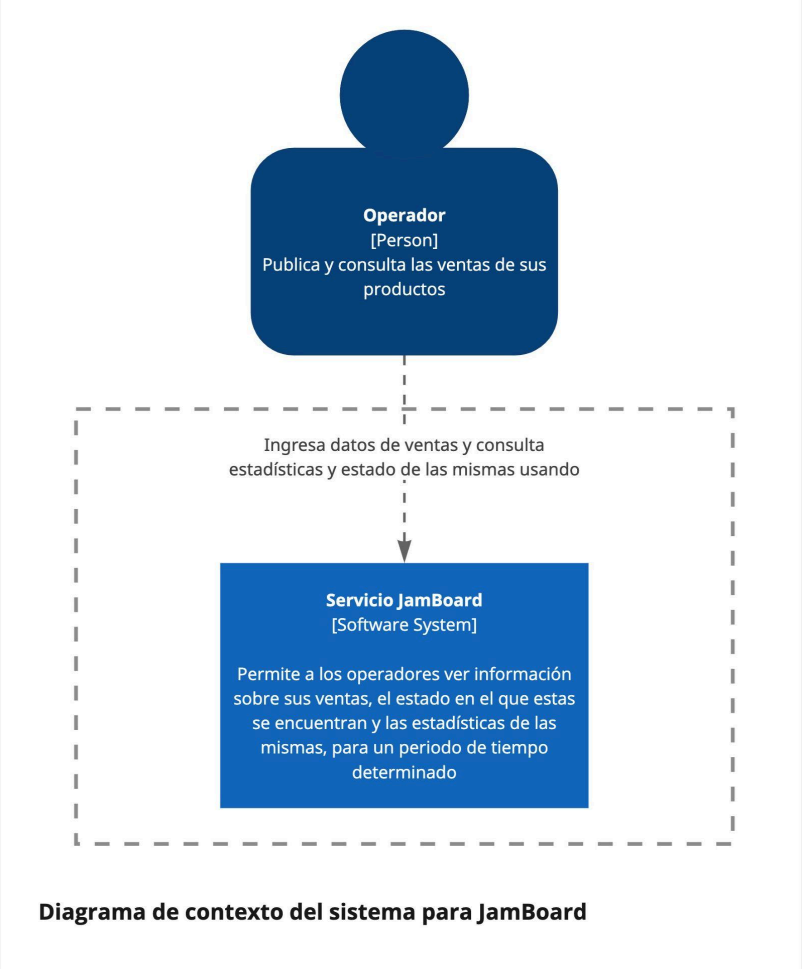

Imagen 2. Diagrama de contexto C4.

El diagrama de contexto, elaborado bajo el enfoque C4, describe a alto nivel la interacción principal entre los usuarios y el sistema JamBoard. Su propósito es mostrar de forma clara quién usa el sistema, qué funcionalidades se ofrecen y cómo fluye la información entre los actores involucrados. El sistema en cuestión se denomina Servicio JamBoard y se clasifica como un sistema de software. Este servicio actúa como intermediario entre el usuario y la información sobre las ventas, permitiendo la consulta del estado actual de las mismas, así como el análisis de estadísticas asociadas a un periodo de tiempo determinado.

DIAGRAMA DE CONTENEDORES C4

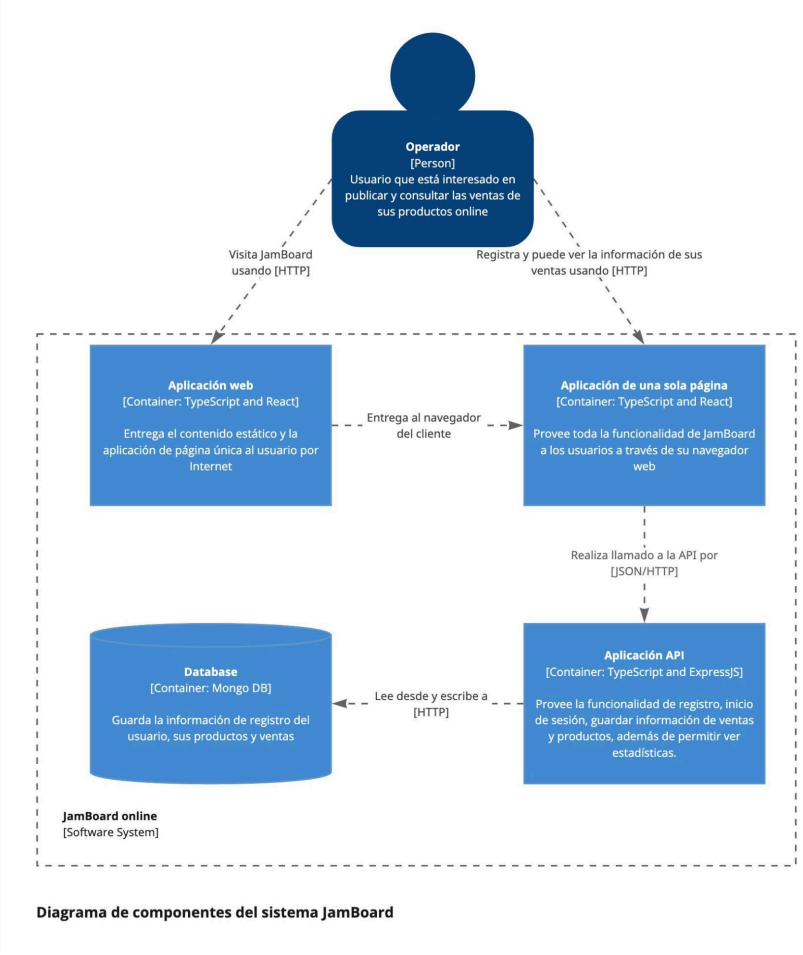

Imagen 3. Diagrama de contenedores C4.

El diagrama de contenedores del sistema JamBoard detalla la estructura principal del sistema, identificando los componentes clave y cómo interactúan entre sí para cumplir con las funcionalidades esperadas por el usuario. El actor principal es el usuario, que publica y consulta ventas de productos en línea mediante un navegador web. Su interacción con el sistema se da a través del protocolo HTTP.

El sistema JamBoard se compone de los siguientes contenedores:

- Aplicación Web (TypeScript + React): entrega el contenido estático y la SPA (Single Page Application) al navegador del operador.
- Aplicación de una sola página (SPA) (TypeScript + React): permite al operador acceder a toda la funcionalidad de JamBoard desde el navegador, realizando llamadas HTTP/JSON a la API.
- API Backend (TypeScript + ExpressJS): gestiona el registro e inicio de sesión, el almacenamiento de productos y ventas, y el cálculo de estadísticas. Recibe las solicitudes de la SPA y opera sobre la base de datos.
- Base de datos (MongoDB): almacena de forma persistente la información del usuario, productos y ventas. Es accedida únicamente por la API.

El flujo general consiste en que el operador accede a la aplicación web, esta entrega la SPA, y desde allí se realizan solicitudes a la API, que interactúa con la base de datos para consultar o registrar información.

DIAGRAMA DE COMPONENTES C4

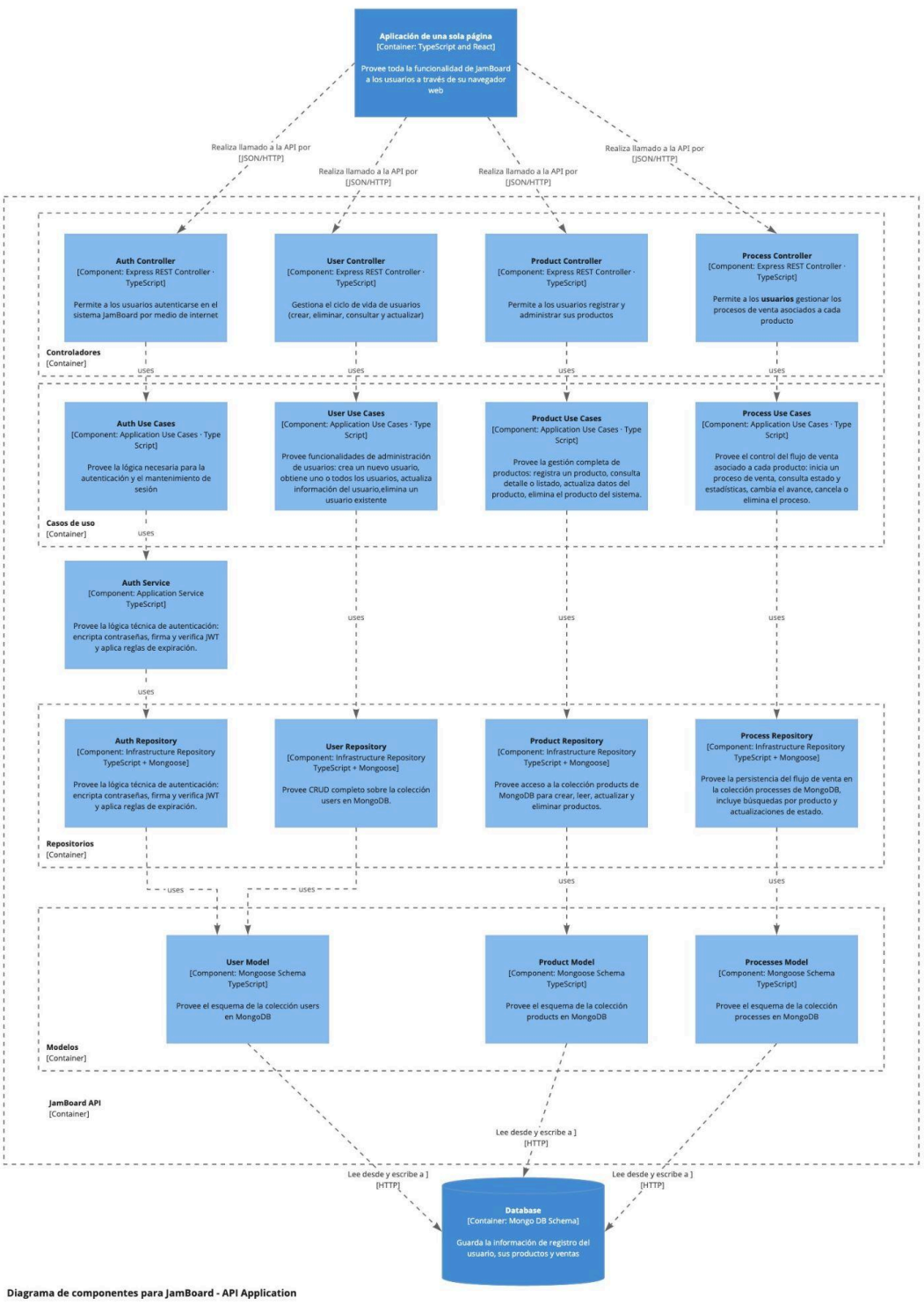

Imagen 4. Diagrama de componentes C4.

Este diagrama muestra los componentes internos que conforman la API de JamBoard, detallando la estructura lógica y su interacción con la base de datos.

Desde el frontend (una SPA en TypeScript y React), se realizan llamadas HTTP/JSON hacia distintos controladores REST en la API, los cuales están organizados según funcionalidades específicas:

- Controladores (Controllers): gestionan las solicitudes del frontend para autenticación, usuarios, productos y procesos de venta.
- Casos de uso (Use Cases): encapsulan la lógica de negocio, permitiendo realizar operaciones como autenticarse, registrar usuarios o administrar productos y procesos.
- Servicios (Services): contienen la lógica técnica crítica, como la encriptación, validación JWT y reglas de sesión.
- Repositorios: gestionan el acceso a la base de datos, implementando operaciones CRUD y búsquedas avanzadas.
- Modelos: definen los esquemas de las colecciones MongoDB (usuarios, productos y procesos).

Finalmente, todos estos componentes interactúan con una base de datos MongoDB, que persiste la información del sistema de manera estructurada según los modelos definidos.

DIAGRAMA DINÁMICO C4

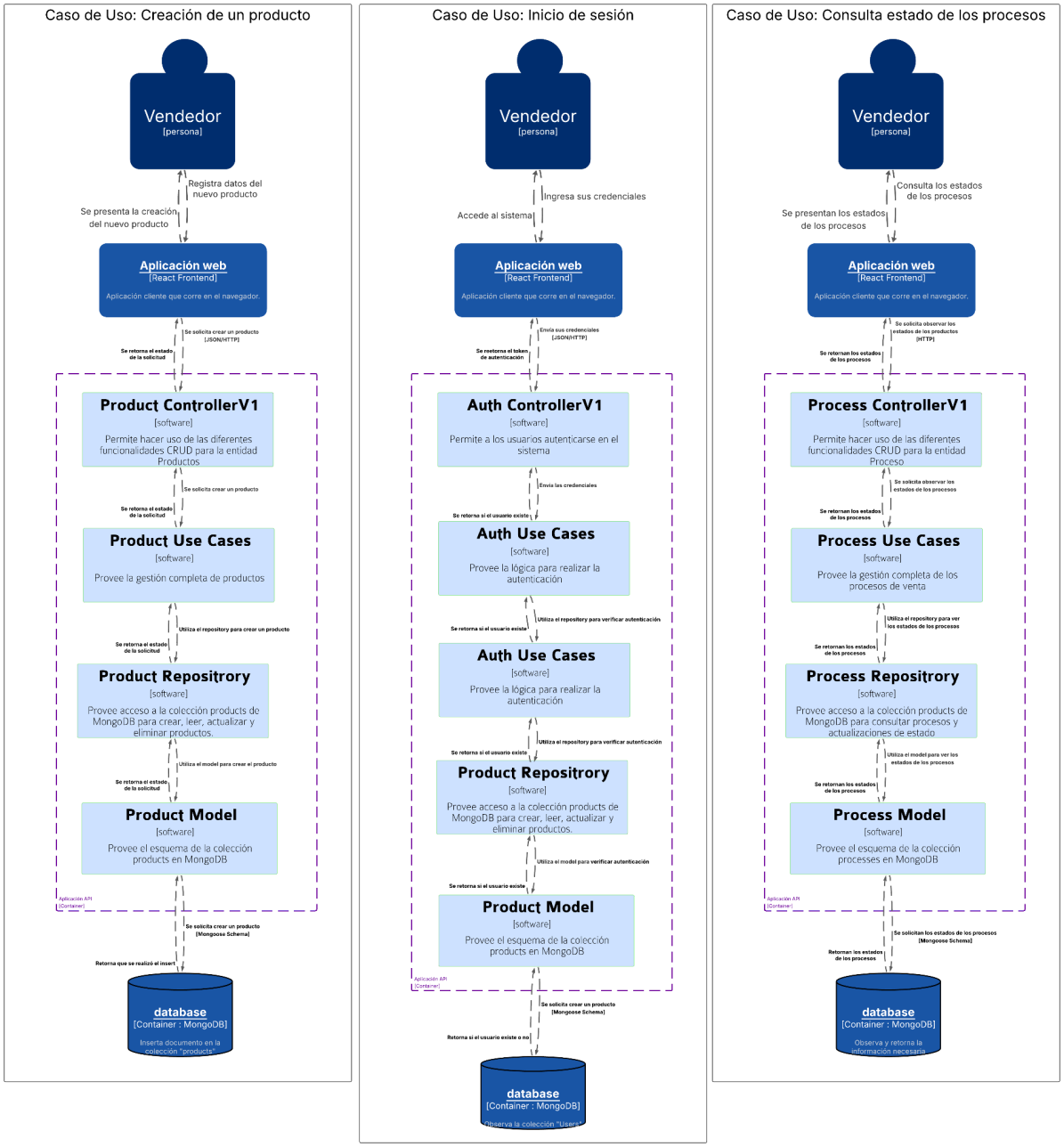

Imagen 5. Diagramas Dinámicos C4.

En la Imagen 4, se presentan 3 casos de uso para los cuales se realizaron distintos diagramas dinámicos. Para el caso de uso de inicio de sesión, el usuario accede al sistema ingresando sus credenciales a través de la misma aplicación web React. La solicitud es manejada por el Auth Controller, el cual dirige la autenticación al conjunto de Auth Use Cases. Estos contienen la lógica específica para verificar las credenciales y generar tokens de sesión si es necesario. La capa de casos de uso interactúa con el Product Repository para realizar validaciones adicionales si se requiere acceder a datos del vendedor relacionados con productos. Este repositorio consulta el esquema de productos definido en Product Model, accediendo a la base de datos MongoDB correspondiente.

De forma similar, el caso de uso de creación de producto, muestra el flujo de interacción para la creación de un nuevo producto por parte del vendedor. El proceso inicia cuando el usuario ingresa los

datos del producto en la aplicación web construida con React. La solicitud pasa al Product Controller V1, que expone las operaciones necesarias para crear productos. Luego, la lógica de negocio se gestiona en la capa Product Use Cases, que orquesta la operación y delega al Product Repository el acceso a la base de datos. Este repositorio interactúa con la colección products de MongoDB, definida por el Product Model, que representa el esquema del producto. Finalmente, los datos se almacenan en el contenedor de base de datos MongoDB.

Finalmente, en el caso de uso de consulta de estado de procesos, el vendedor consulta el estado de sus procesos de venta mediante la aplicación web. La solicitud es dirigida al Process Controller V1, que habilita las operaciones CRUD necesarias sobre la entidad de procesos. Luego, la lógica se gestiona en Process Use Cases, los cuales coordinan la operación con el Process Repository, responsable del acceso a la colección process en la base de datos MongoDB. El esquema de estos datos está definido por el Process Model. Finalmente, los datos se obtienen del contenedor database, que representa la base de datos específica para esta funcionalidad.

DIAGRAMA DE DESPLIEGUE C4

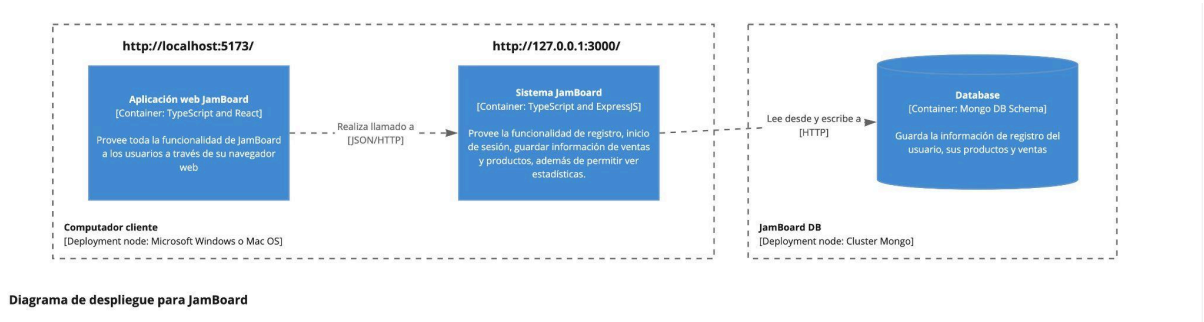

Imagen 6. Diagrama de despliegue C4.

El despliegue de JamBoard se basa en una arquitectura distribuida sencilla con tres nodos principales:

1. Cliente (navegador web)
    - Componente: Aplicación Web JamBoard
    - Tecnología: TypeScript + React
    - Funcionalidad: Interfaz de usuario que permite interactuar con el sistema desde el navegador.
2. Servidor Backend (API)
    - Componente: Sistema JamBoard
    - Tecnología: TypeScript + ExpressJS
    - Funcionalidad: Gestiona el registro e inicio de sesión, manejo de productos, procesos de venta y estadísticas.
3. Base de Datos
    - Componente: MongoDB
    - Tecnología: Mongo DB Schema (conectado por Mongoose)
    - Funcionalidad: Almacena la información de usuarios, productos y ventas.

Las interacciones se realizan a través de llamadas HTTP usando JSON, desde el cliente hacia la API, y desde la API hacia la base de datos.

DIAGRAMA PAQUETES

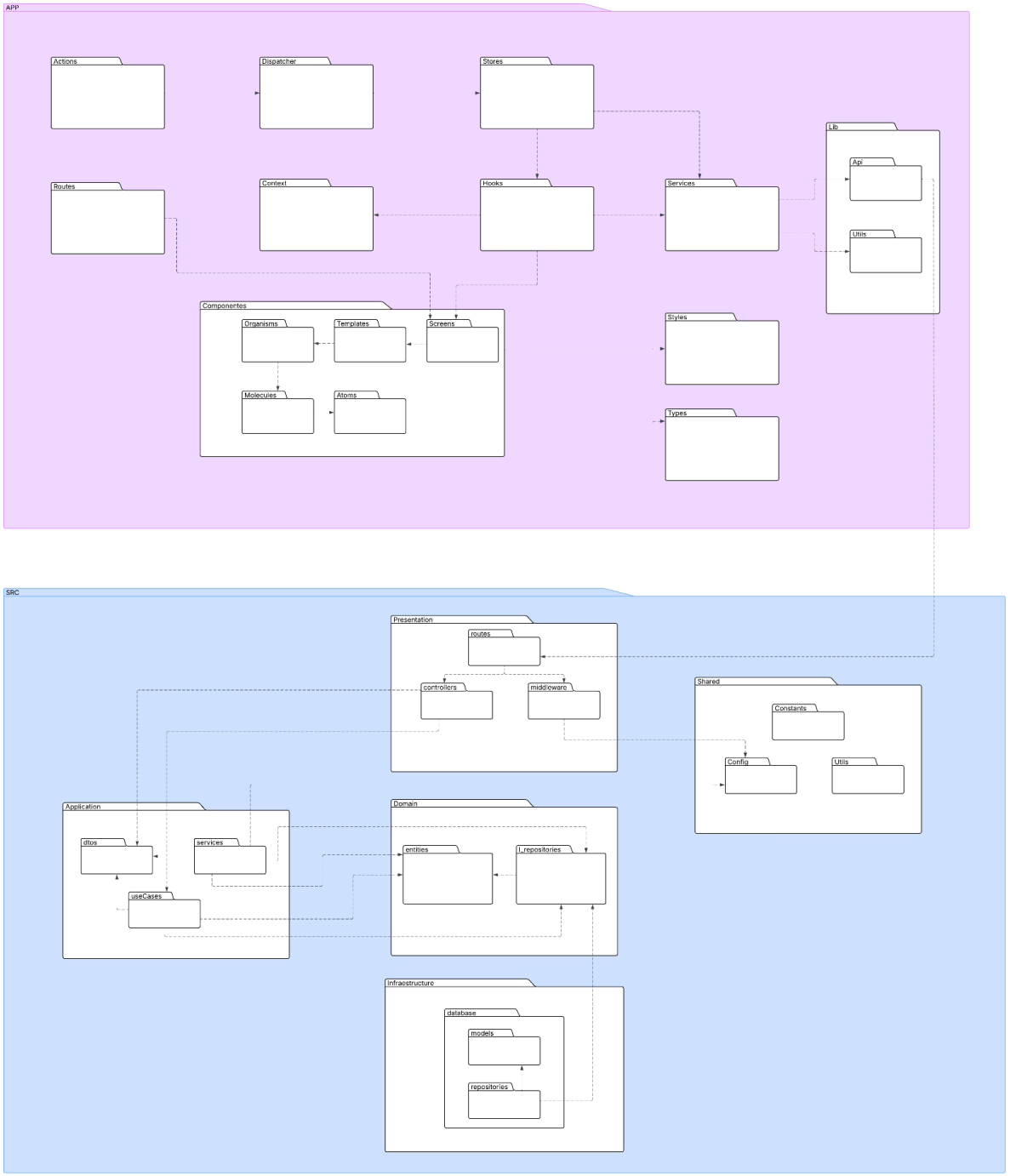

Imagen 7. Diagrama de paquetes.

En el caso del sistema JamBoard, el diagrama de paquetes refleja claramente dos contextos principales: **el frontend**, desarrollado en React y TypeScript, y **el backend**, construido con Node.js y

Express bajo el enfoque de Clean Architecture. En la sección correspondiente a la aplicación web (frontend), se emplea una organización basada en Atomic Design, donde los paquetes se dividen en niveles de abstracción como atoms, molecules y organisms, junto con otros módulos funcionales como pages, routes, services y context. Esta estructuración permite un desarrollo escalable, reutilizable y con un alto nivel de cohesión entre componentes de la interfaz de usuario.

Por otro lado, el backend se encuentra estructurado bajo los principios de Clean Architecture, donde los paquetes se agrupan en capas bien definidas: domain, application, infrastructure y presentation. La capa domain contiene las entidades del negocio y las interfaces (contratos) que definen el comportamiento deseado; application agrupa los casos de uso y servicios que implementan la lógica de negocio; infrastructure contiene las implementaciones concretas, como adaptadores para bases de datos; y presentation está encargada de exponer las rutas y controladores que se comunican con el cliente. Esta separación estricta de responsabilidades facilita el mantenimiento del sistema, promueve el desacoplamiento y mejora la capacidad de prueba de cada componente.

CÓDIGO FUENTE EN REPOSITORIO PÚBLICO GITHUB

Código fuente: https://github.com/Kose117/JAMBOARD-ArquiTaller

REFERENCIAS

[1] https://es.react.dev/
[2] https://www.mongodb.com/es/company/what-is-mongodb
[3] https://expressjs.com/
[4] https://www.typescriptlang.org/
[5] https://platzi.com/clases/2484-react-webpack-sass/42217-que-es-atomic-design/
[6] https://java-design-patterns.com/patterns/flux/
[7] https://medium.com/@rudrakshnanavaty/clean-architecture-7c1b3b4cb181
[8] https://openwebinars.net/blog/mern-stack-que-es-y-que-ventajas-ofrece/
[9] https://www.luisllamas.es/historia-de-nodejs/
[10] https://www.mongodb.com/es/company
[11] https://www.oracle.com/es/database/mern-stack/
[12] https://www.linkedin.com/pulse/future-mern-stack-developer-2024-anusha-srivastava-f2umf/
[13] https://zipdo.co/mern-stack-developer-salary-statistics/
[13]https://dev.to/raji_moshood_ee3a4c2638f6/the-demand-for-mern-stack-developers-how-to-stand-out-in-the-job-market-4k0j
[14] https://futuristiccodingacademy.com/mern-stack-developer-salary/
[15] https://www.ambitionbox.com/profile/mern-stack-developer-salary/internet-industry
[16] https://www.softcrayons.com/blog/global-prospects-for-mern-stack-developers