

LAURA OVALLE
JOSE RODRÍGUEZ
JUAN DAVID GONZÁLEZ

GRUPO 4

STACK MERN

+ TYPESCRIPT

ARQUITECTURA DE SOFTWARE

ÍNDICE

01. Definición
02. Características
03. Historia y Evolución
04. Ventajas y Desventajas
05. Casos de Uso
06. Popularidad
08. Matrices de análisis
07. Implementación

DEFINICIÓN

STACK MERN CON TYPESCRIPT

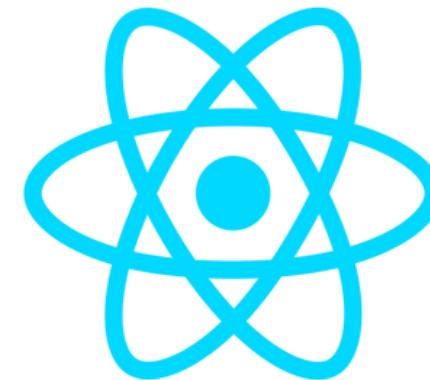
Conjunto de tecnologías modernas para desarrollo full-stack de aplicaciones web.



MongoDB
Base de datos NoSQL
orientada a documentos.



Express.js
Framework backend
minimalista sobre
Node.js.



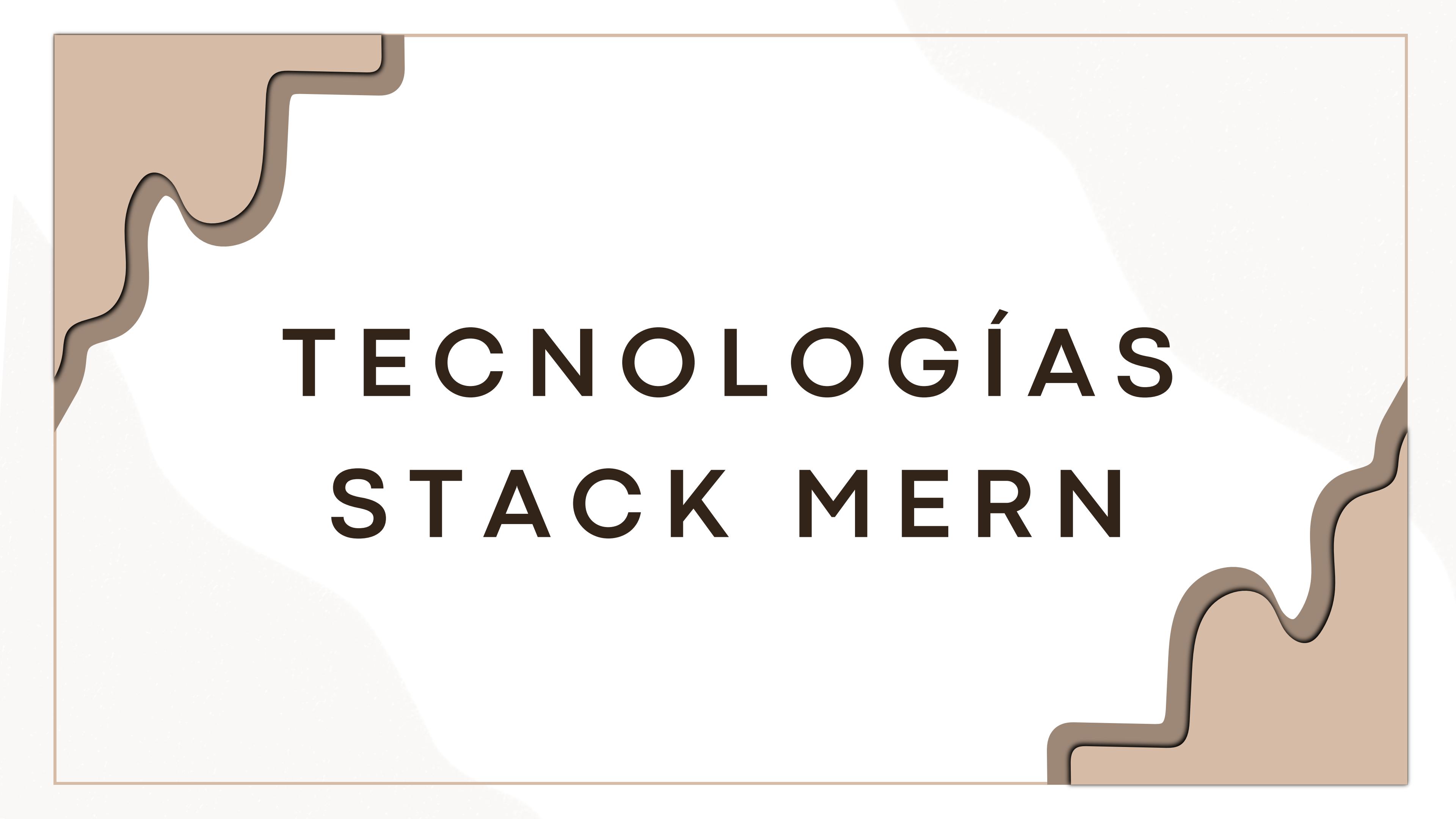
React
Biblioteca frontend
basada en
componentes.



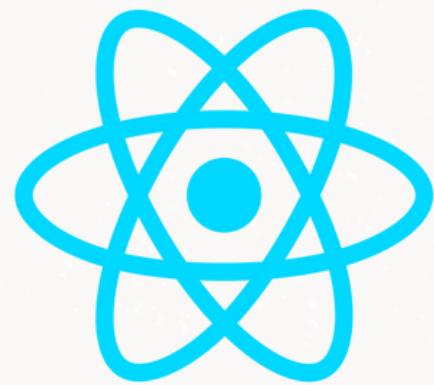
Node.js
Entorno de ejecución
JavaScript del lado del
servidor.

Uso integral de TypeScript, proporcionando tipado estático para mejorar mantenibilidad y
detección temprana de errores.

Ganó popularidad por ofrecer soluciones completas y eficientes para aplicaciones web modernas

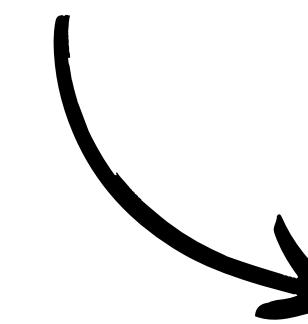


TECNOLOGÍAS STACK MERN



REACT

- Biblioteca JavaScript para interfaces de usuario.
- Componentes reutilizables.
- Enfoque declarativo y flujo de datos unidireccional.



MONGODB

- Base de datos NoSQL.
- Almacena documentos en formato BSON.
- Escalabilidad horizontal y flexibilidad en esquema.
- Ideal para grandes volúmenes de datos no estructurados.



EXPRESS.JS

- Framework minimalista para Node.js.
- Facilita creación de aplicaciones web y APIs.
- Manejo eficiente de rutas, middleware y solicitudes HTTP.



NODE.JS

- Entorno de ejecución JavaScript del lado servidor.
- Modelo de I/O no bloqueante y orientado a eventos.
- Alta escalabilidad y rendimiento en aplicaciones de red.



TYPESCRIPT

- Lenguaje desarrollado por Microsoft que extiende JavaScript.
- Tipado estático opcional.
- Facilita detección temprana de errores y mejora la mantenibilidad.





PATRONES Y ARQUITECTURAS

FRONTEND



01. Atomic Design

- Metodología para diseño de interfaces.
- Componentes organizados en átomos, moléculas, organismos, plantillas y páginas.
- Promueve reutilización y coherencia en el diseño.



02. Flux

- Patrón arquitectónico para gestionar flujo de datos.
- Flujo unidireccional basado en acciones, despachador, tiendas (stores) y vistas.
- Facilita gestión predecible del estado.

BACKEND

CA

01. Clean Architecture

- Organización del código en capas concéntricas.
- Separación clara de responsabilidades y dependencias.
- Sistemas modulares, testeables y mantenibles.
- Independencia de la lógica de negocio frente a implementación tecnológica específica.

C/S

02. Arquitectura Cliente-Servidor

- Clientes solicitan recursos o servicios.
- Servidor central responde procesando solicitudes.
- Distribución eficiente de responsabilidades.
- Mejora la escalabilidad, mantenimiento y seguridad.

CARACTERÍSTICAS

CARACTERÍSTICAS GENERALES

Robustez, escalabilidad y mantenibilidad mediante tecnologías modernas y patrones arquitectónicos claros.

Optimización del desarrollo mediante herramientas coherentes como JavaScript/TypeScript, React, Node.js y MongoDB.

Código organizado y eficiente mediante patrones Flux, Atomic Design y Clean Architecture.

Facilita evolución y mantenimiento a largo plazo mediante modularidad y desacoplamiento.

CARACTERÍSTICAS CLAVE

HOMOGENEIDAD EN EL LENGUAJE

- Un mismo lenguaje (JavaScript/TypeScript) en frontend y backend.
- Mejora colaboración, coherencia y agilidad en el desarrollo.

TIPADO ESTÁTICO CON TYPESCRIPT

- Detección temprana de errores (compilación).
- Código autodocumentado, mantenible y robusto.

FLUJO NATURAL DE DATOS CON JSON

- Integración nativa entre MongoDB, Node/Express y React.
- Simplificación de serialización/deserialización.

CARACTERÍSTICAS CLAVE

DESARROLLO BASADO EN COMPONENTES REUTILIZABLES (REACT + ATOMIC DESIGN)

- Modularidad en construcción de interfaces.
- Jerarquización en componentes: átomos, moléculas, organismos, plantillas y páginas.

GESTIÓN PREDICTIVA DEL ESTADO (FLUX)

- Flujo de datos unidireccional.
- Simplifica depuración y manejo del estado en interfaces complejas.

CARACTERÍSTICAS CLAVE

ARQUITECTURA LIMPIA Y DESACOPLADA (CLEAN ARCHITECTURE)

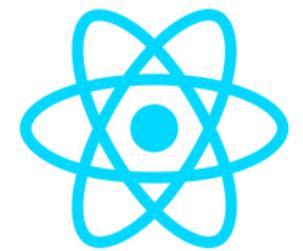
- Separación lógica en capas (entidades, casos de uso, controladores e infraestructura).
- Independencia del negocio frente a implementación tecnológica específica.
- Facilita pruebas, mantenimiento y escalabilidad.

ESCALABILIDAD Y RENDIMIENTO DEL BACKEND (NODE.JS + EXPRESS)

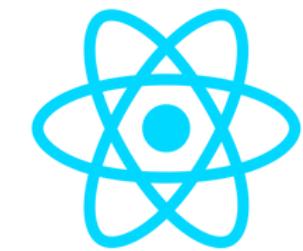
- Modelo orientado a eventos y no bloqueante.
- Ideal para aplicaciones en tiempo real (chats, dashboards).
- Express: estructura ligera y escalable para APIs RESTful.

BASE DE DATOS FLEXIBLE Y ESCALABLE (MONGODB)

- Esquemas dinámicos adaptables a cambios constantes.
- Escalabilidad horizontal adaptativa.
- Perfecto para datos semiestructurados y aplicaciones en crecimiento dinámico.



CARACTERÍSTICAS REACT



01

Componentes reutilizables

Interfaces modulares y
escalables.

02

DOM Virtual

Actualizaciones eficientes del
DOM.

03

Flujo unidireccional

Estado predecible y fácil
mantenimiento.



CARACTERÍSTICAS MONGO - DB



01

Modelo flexible de documentos
Almacenamiento en formato BSON (JSON binario)

02

Escalabilidad horizontal
Distribución eficiente entre múltiples servidores.

03

Integración natural con JSON
Comunicación eficiente con aplicaciones JavaScript.

CARACTERÍSTICAS EXPRESS.JS



01

Framework minimalista

Desarrollo rápido de aplicaciones y APIs.

02

Middleware personalizable

Gestión flexible de solicitudes HTTP.

03

Integración estrecha con Node.js

Aprovecha ventajas del entorno Node.js.



CARACTERÍSTICAS NODE.JS



01

Entorno asíncrono
Manejo eficiente de
múltiples conexiones
simultáneas.

02

Motor V8 de Chrome
Alto rendimiento en ejecución
JavaScript.

03

**Amplio ecosistema
(npm)**
Funcionalidades
adicionales fácilmente
integrables.



CARACTERÍSTICAS TYPESCRIPT



01

Tipado estático opcional
Detección anticipada de errores.

02

Facilita mantenimiento
Código más legible y refactorización sencilla.

03

Compatibilidad gradual con JavaScript
Fácil adopción progresiva en proyectos existentes.

CARACTERÍSTICAS ATOMIC DESIGN

01

Estructura jerárquica clara

Átomos, moléculas, organismos, plantillas y páginas.

02

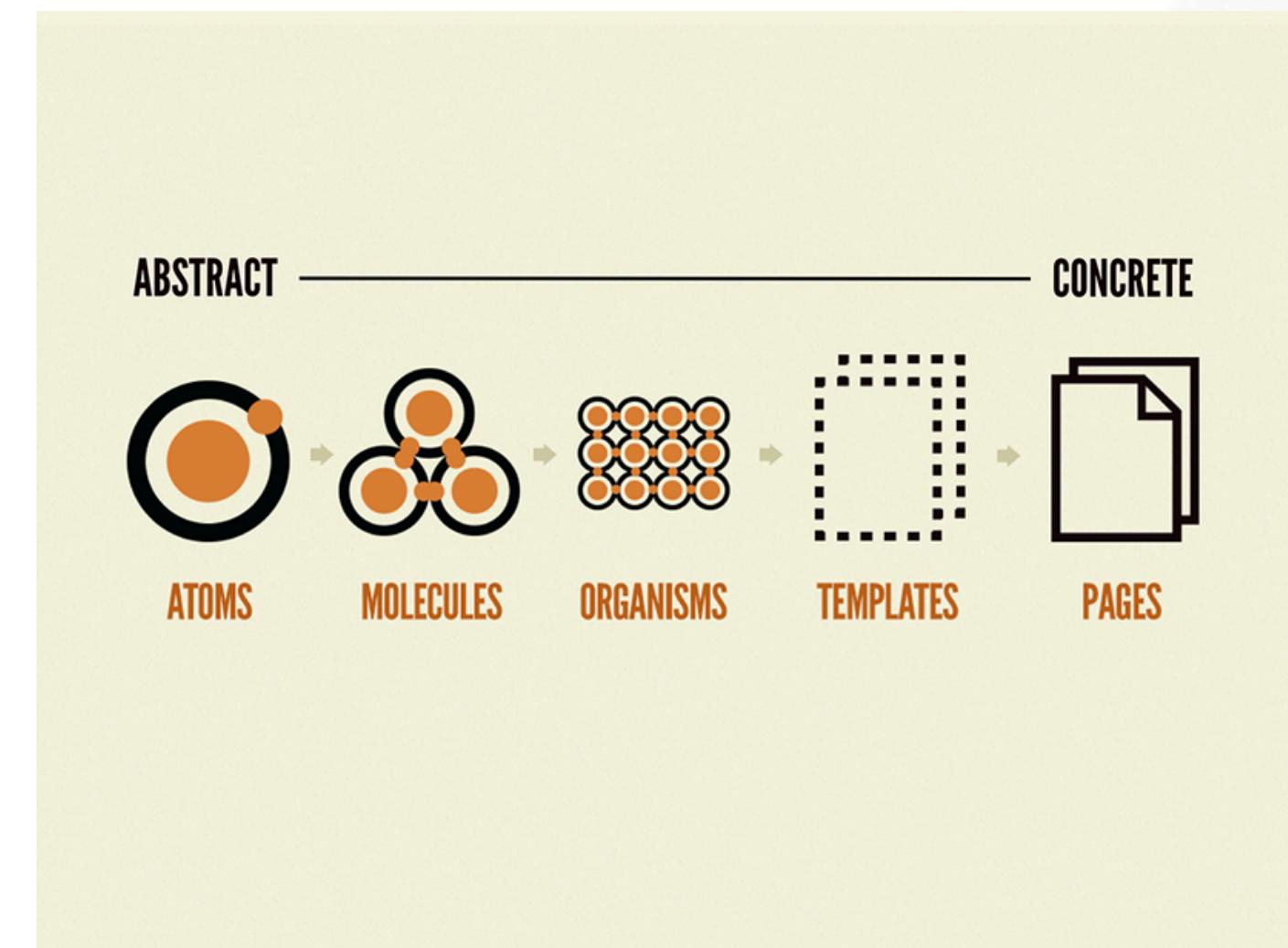
Reutilización de componentes

Interfaces consistentes y eficientes.

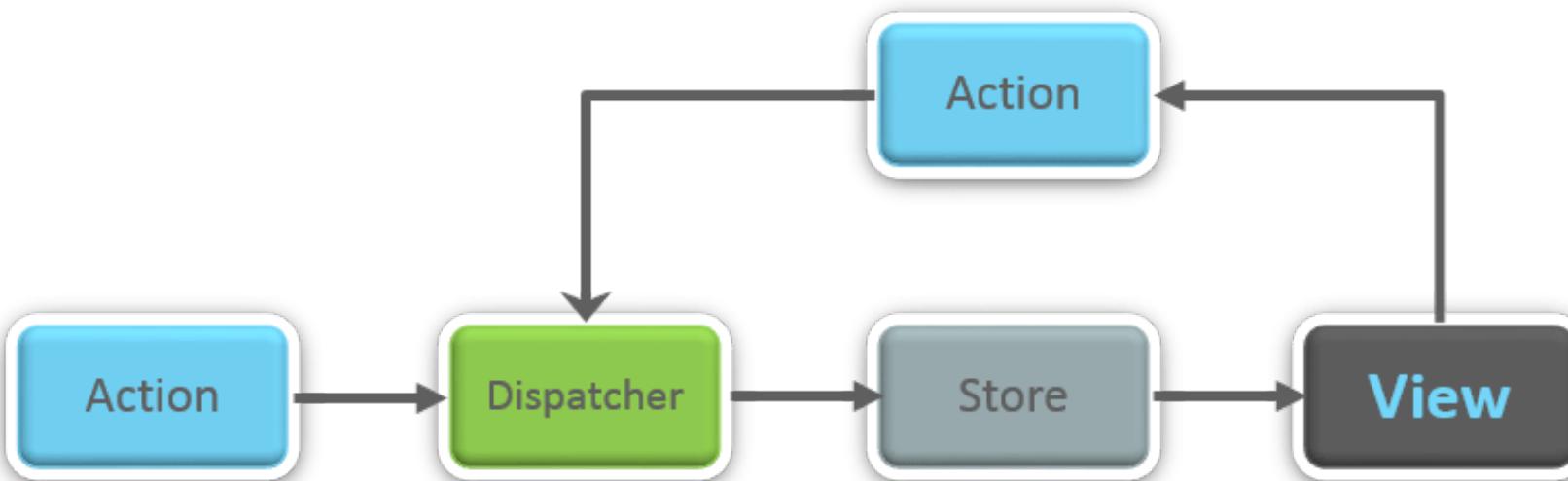
03

Diseño sistemático

Coherencia en interfaces complejas.



CARACTERÍSTICAS FLUX



01

Flujo unidireccional

Simplificación en seguimiento de cambios.

02

Componentes clave

Acciones, despachador, tiendas (stores) y vistas.

03

Estado predecible

Manejo claro en aplicaciones complejas.

CARACTERÍSTICAS CLEAN ARCHITECTURE

01

Separación de responsabilidades
Código organizado en capas independientes.

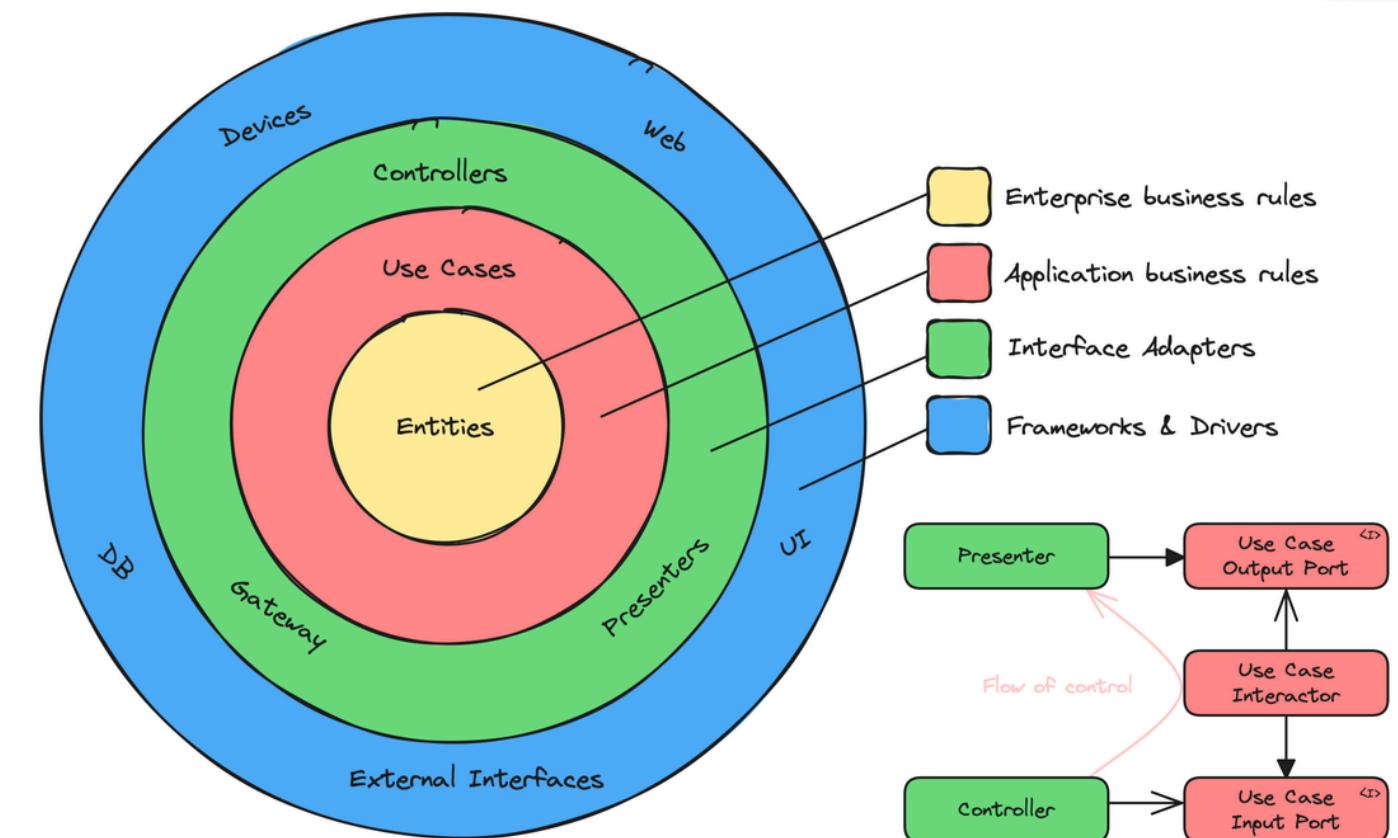
02

Independencia tecnológica
Lógica de negocio sin dependencias directas a frameworks o bases de datos.

03

Facilidad para pruebas unitarias

Capas testeables aisladamente, mejorando calidad del software.





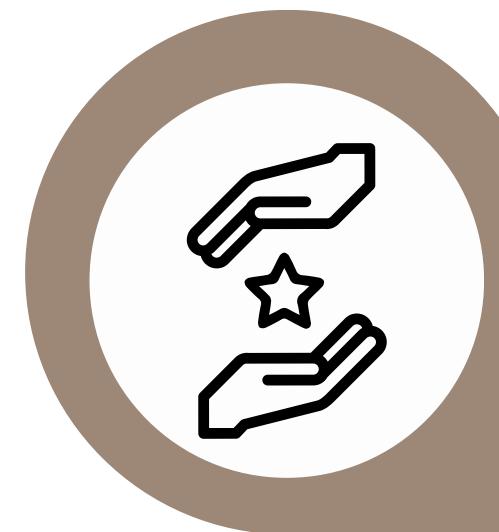
HISTORIA & EVOLUCIÓN

N O D E . J S

Creado por Ryan Dahl (2009).

Ejecución de JavaScript en servidor:

- Basado en motor V8 de Chrome.
- Modelo asíncrono y event-driven.



Revolucionó el desarrollo web:

- Un mismo lenguaje (JavaScript) en frontend y backend.
- Alta escalabilidad y rendimiento, ideal para aplicaciones en tiempo real.

MONGO - DB

Creado por 10gen
(hoy MongoDB Inc.)
en 2009.

Surgió para gestionar
datos
semiestructurados o
no estructurados.



Base de datos NoSQL
orientada a documentos
(BSON).

Características clave

- Esquemas flexibles.
- Escalabilidad horizontal sencilla.
- Integración natural con aplicaciones JavaScript.

EXPRESS.JS

Creado por TJ
Holowaychuk (2010).

Inspirado en Sinatra
(Ruby), simplificó
construcción de
aplicaciones web y
APIs.



**Manejo sencillo y
eficiente de:**

- Rutas
- Middleware
- Solicitudes HTTP

**Popularidad rápida en
el ecosistema Node.js:**

- Framework dominante
en Node.
- Clave en stacks como
MEAN y
posteriormente MERN.

REACT

Creado por
Facebook (Meta),
liberado en 2013.

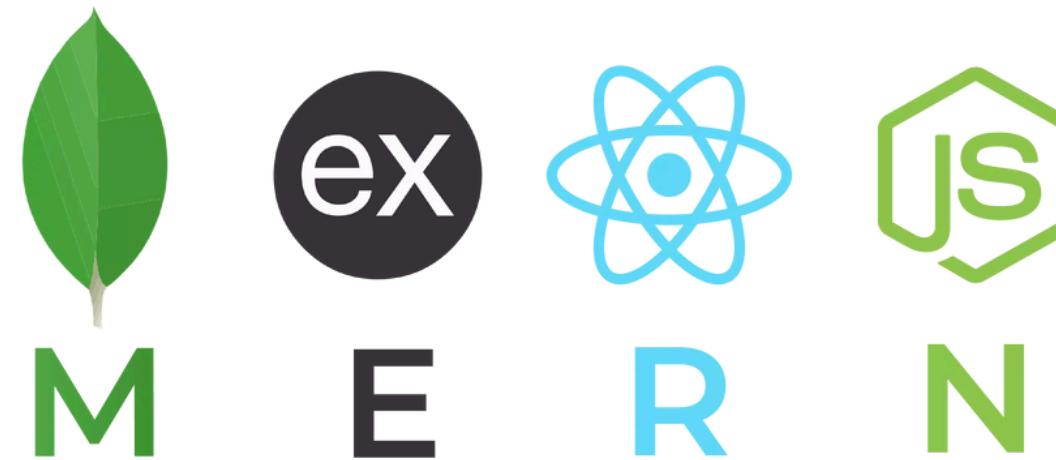
Biblioteca frontend
basada en
componentes
reutilizables.



Conceptos revolucionarios:

- Virtual DOM (optimización rendimiento)
- Enfoque declarativo (facilita gestión del estado)
- Popularidad creciente, reemplazando gradualmente a AngularJS en muchos proyectos.

CONSOLIDACIÓN DEL STACK MERN



- MERN ganó relevancia a partir de mediados de la década de 2010:
- Popular por solución unificada y basada enteramente en JavaScript.
- Razones del éxito:
- Reducción de complejidad por homogeneidad del lenguaje.
- Cohesión del código entre frontend y backend.
- Uso extendido en aplicaciones modernas:
- Empresariales, redes sociales, e-commerce, tiempo real, etc.

INCLUSIÓN DE TYPESCRIPT - EVOLUCIÓN DEL STACK MERN



- TypeScript creado por Microsoft (2012).
- Superset de JavaScript que añade tipado estático opcional.
- Aportó mejoras clave al Stack MERN:
- Detección temprana de errores (compilación).
- Facilita refactorización, mantenimiento y escalabilidad en grandes proyectos.
- Integración fluida con React, Node.js y resto del stack.

ACTUALIDAD Y FUTURO DEL STACK MERN

- MERN se mantiene vigente debido a evolución constante:
- React y MongoDB se adaptan continuamente a nuevas demandas.
- Aplicaciones más dinámicas e interactivas impulsan su popularidad:
- Chat, dashboards, notificaciones en tiempo real, aplicaciones de alta interactividad.
- Amplia adopción por parte de la comunidad:
- Consolidación como estándar en desarrollo full-stack moderno.
- Continuará influyendo en la dirección futura del desarrollo web.





VENTAJAS & DESVENTAJAS

VENTAJAS

Lenguaje unificado (JavaScript/TypeScript)

- Desarrollo full-stack con un solo lenguaje.
- Simplifica aprendizaje, incrementa productividad y mantenimiento.
- TypeScript añade robustez con tipado estático.

Desarrollo modular con React

- Componentes reutilizables para interfaces.
- Mejora mantenibilidad, modularidad y colaboración en equipos.
- Facilita interfaces dinámicas y reactivas.

VENTAJAS

Intercambio eficiente de datos (JSON/BSON)

- MongoDB compatible nativamente con JSON.
- Reduce complejidad de transformación de datos.
- Mejora rendimiento y facilita integración con otras tecnologías.

Comunidad activa y ecosistema robusto

- Amplia comunidad open-source (React, Node.js).
- Numerosas bibliotecas y recursos disponibles en GitHub y NPM.
- Constante evolución gracias a contribuciones continuas.

VENTAJAS

Flexibilidad y escalabilidad

- Sin estructura obligatoria, permite personalización (Clean Architecture, MVC).
- Node.js eficiente en aplicaciones concurrentes.
- MongoDB escalable horizontalmente (replicación, sharding).

DESVENTAJAS

Curva de aprendizaje pronunciada

- Complejidad al integrar múltiples tecnologías.
- Aprendizaje adicional con TypeScript (tipado estático, interfaces).

Falta de estructura predeterminada

- Responsabilidad en definir arquitectura del proyecto.
- Riesgo de desorganización y dificultad de mantenimiento a largo plazo.

DESVENTAJAS

Mantenimiento de múltiples tecnologías

- Ciclos independientes de actualización (MongoDB, Express, React, Node.js, TypeScript).
- Actualizaciones pueden generar incompatibilidades y esfuerzo adicional.

Limitaciones en procesamiento intensivo (Node.js)

- Naturaleza single-threaded limita procesamiento CPU intensivo.
- Requiere soluciones adicionales (workers/procesos separados).

DESVENTAJAS

Limitaciones de MongoDB frente a SQL

- Menos eficiente en consultas complejas y transacciones ACID.
- No óptimo para aplicaciones con manipulación avanzada de datos.

Ecosistema en constante evolución

- Riesgo de rápida obsolescencia (ej. React: clases → hooks).
- Requiere constante adaptación a nuevas tendencias (Koa, Fastify, Nest.js).

CASOS DE USO

EL STACK MERN ES IDEAL PARA APLICACIONES QUE REQUIEREN:

01

Desarrollo rápido y flexible.

02

Interactividad dinámica.

03

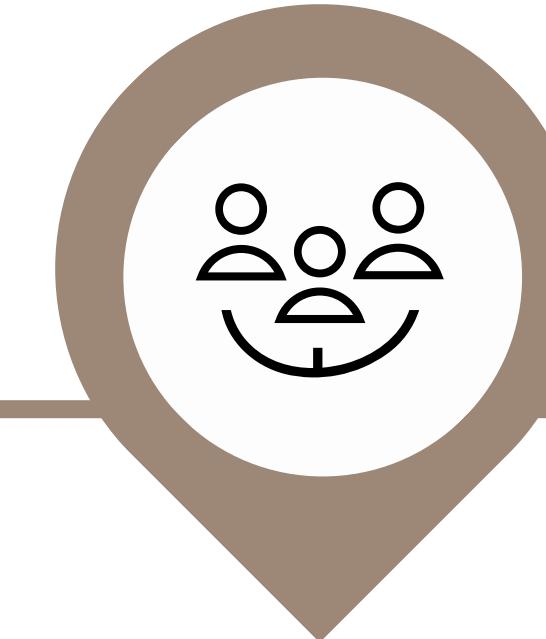
Escalabilidad moderada a alta.

04

Integración eficiente frontend-backend.

Aplicaciones CRUD Empresariales

- Ejemplo: Gestión de inventarios o pedidos.
- Manejo eficiente de grandes volúmenes de datos con MongoDB.
- Interfaces interactivas con React.

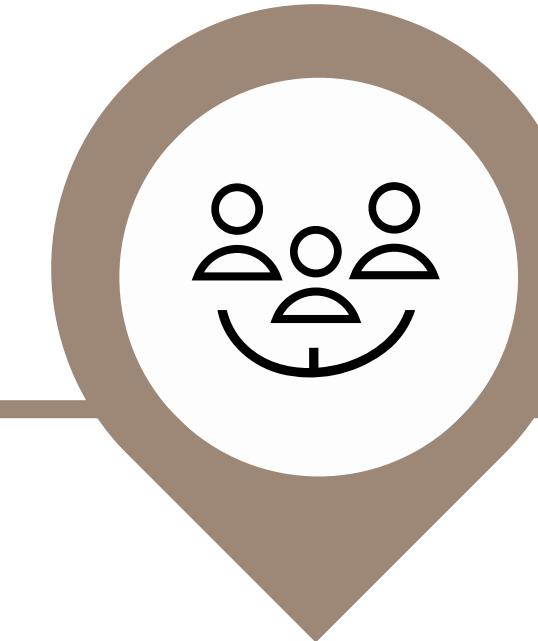


Aplicaciones con Interactividad Moderada en Tiempo Real

- Ejemplo: Chats online, tableros colaborativos.
- Actualizaciones en vivo mediante técnicas como polling o Socket.io.

Desarrollo de Prototipos y PMV (Startups)

- Implementación ágil y rápida con un único lenguaje.
- Versión funcional del producto con menor inversión inicial.



Plataformas Web Dinámicas con Contenido Generado por Usuarios

- Ejemplo: Redes sociales, foros, blogs.
- Actualización dinámica del contenido sin recargar página.
- Almacenamiento flexible de contenidos en MongoDB.

freeCodeCamp (🔥)

- Plataforma educativa de código abierto.
- Millones de usuarios activos.
- Uso de MERN completo:
 - Backend: Node.js y Express.js.
 - Frontend: React.
 - Base de datos: MongoDB.
 - Manejo eficiente de tráfico alto y datos de usuarios.



- Principal marketplace en Latinoamérica.
- Uso extensivo de React y Node.js en microservicios.
- Capacidad para manejar tráfico y transacciones elevadas.
- Escalabilidad e infraestructura robusta.



- Sector viajes global.
- Uso de MERN para recomendaciones personalizadas.
- Integración eficiente con servicios externos (hoteles, vuelos).
- React: Interfaz dinámica e instantánea.



- Industria de videojuegos.
- Dashboards interactivos con estadísticas en tiempo real.
- MongoDB: Almacenamiento eficiente de datos de usuarios.
- React: Interfaz dinámica y respuesta inmediata.



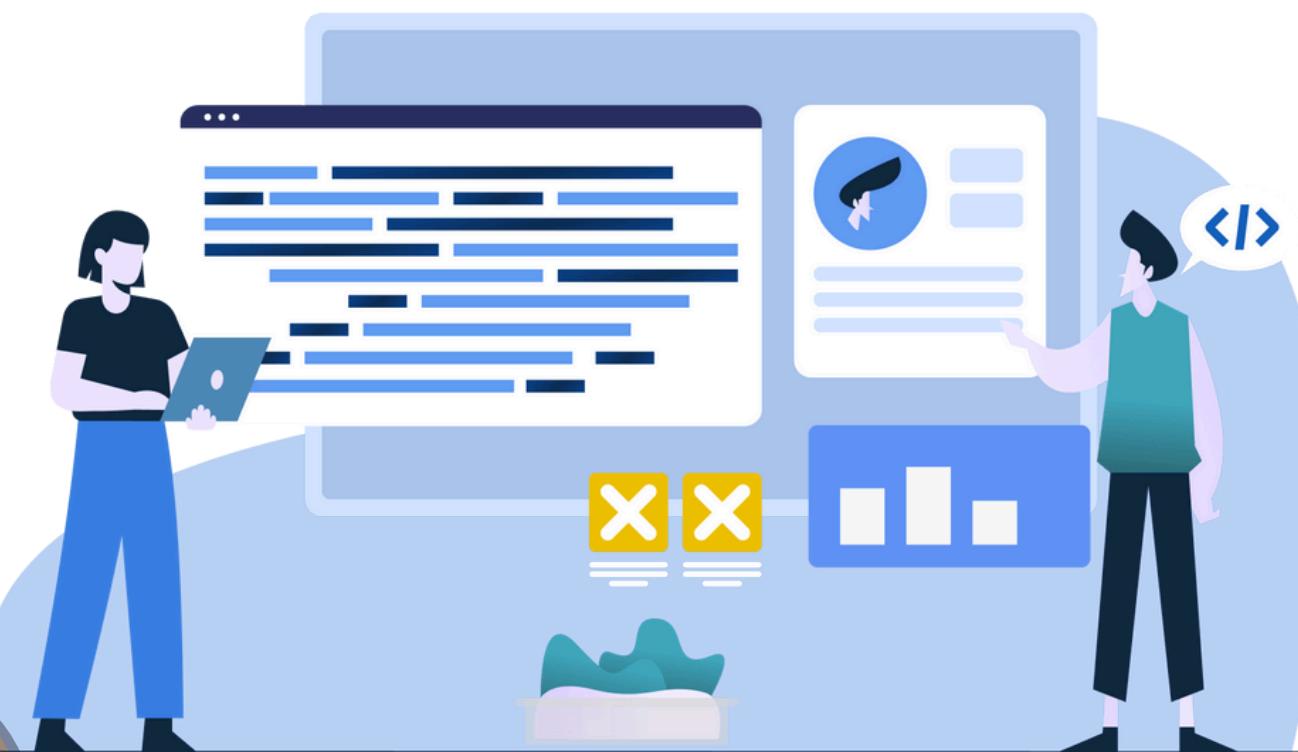
- Plataforma de criptomonedas.
- Manejo de conexiones seguras y rápidas con Node.js.
- MongoDB: Almacenamiento de grandes volúmenes de transacciones.
- React: Dashboard dinámico con precios en vivo.



- Plataforma global de subastas.
- Node.js/Express: Manejo eficiente de transacciones y solicitudes.
- MongoDB: Soporte para millones de datos transaccionales.
- React: Interacción dinámica comprador-vendedor en tiempo real.

¿QUÉ TAN COMÚN ES EL STACK MERN?

Get Development Services

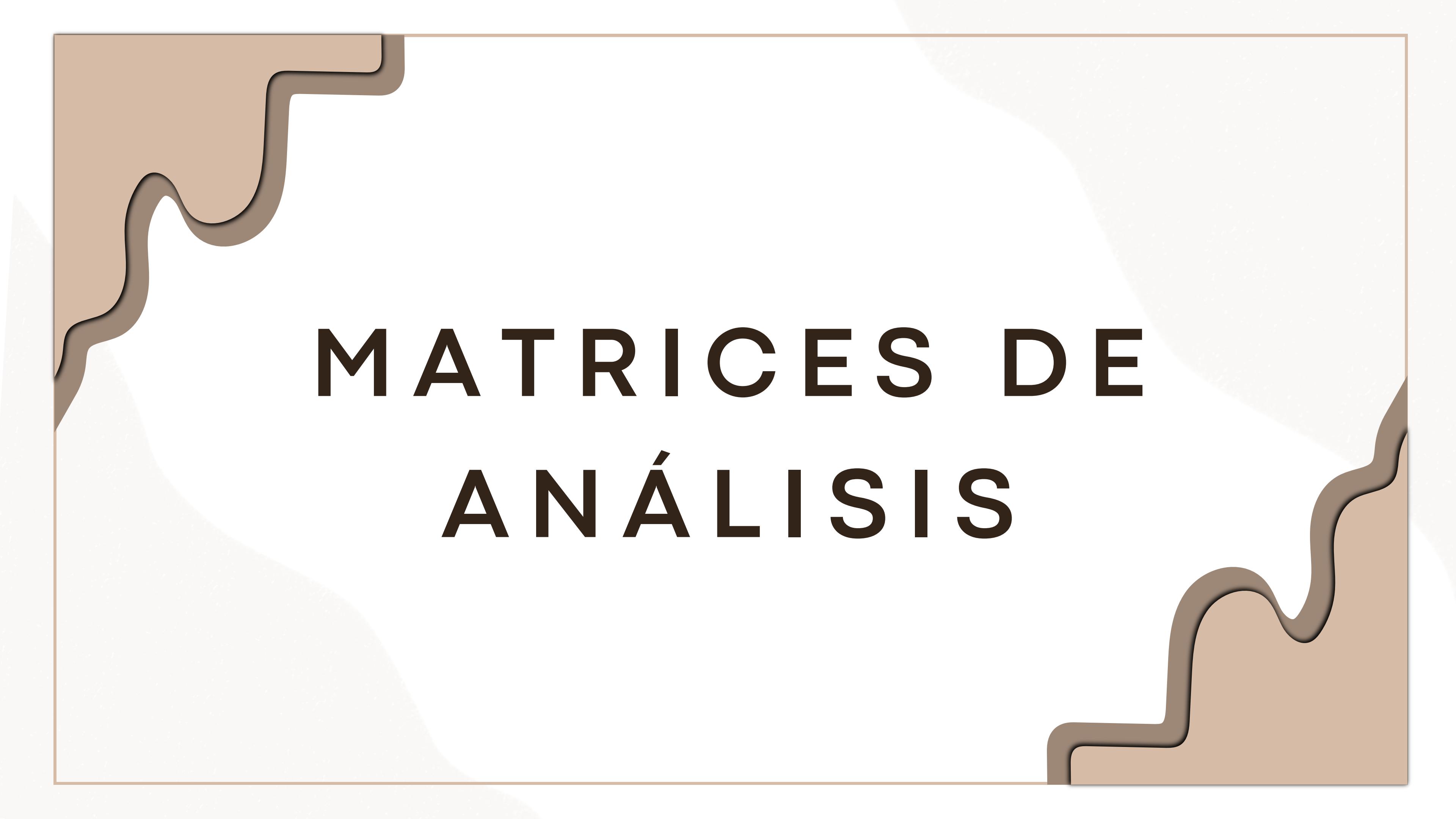


- Consolidado como uno de los stacks más populares en desarrollo web moderno.
- Desarrollo integral en JavaScript (frontend/backend).
- Alta demanda laboral en múltiples sectores de la industria.

POPULARIDAD DEL STACK MERN - INDUSTRIA Y COMUNIDAD

- Oracle destaca eficiencia y productividad del stack MERN.
- LinkedIn resalta MERN como habilidad altamente demandada para desarrolladores.
- Ofertas laborales frecuentes en plataformas como Computrabajo, LinkedIn y Glassdoor:
 - React + TypeScript (Frontend)
 - Node.js (Backend)
 - MongoDB (Base de datos)





MATRICES DE ANÁLISIS

MATRIZ DE ANÁLISIS DE SOLID

PRINCIPIO SOLID	NIVEL DE CUMPLIMIENTO	ARGUMENTACIÓN
S – Responsabilidad Única (SRP)	ALTO	Componentes y capas bien definidos en React, MERN y Clean Architecture permiten responsabilidades claras y únicas.
O – Abierto/Cerrado (OCP)	ALTO	Es fácil extender el sistema (nuevas rutas, componentes, acciones) sin modificar lo ya existente.
L – Sustitución de Liskov (LSP)	MEDIO	Puede haber problemas si las implementaciones no respetan completamente los contratos, aunque se promueven buenas prácticas.
I – Segregación de Interfaces (ISP)	ALTO	Interfaces específicas en Clean Architecture, componentes con props precisas en React y endpoints REST bien definidos.
D – Inversión de Dependencias (DIP)	ALTO	Uso de abstracciones y separación clara entre capas y servicios en Clean Architecture, MERN y arquitectura cliente-servidor.

MATRIZ DE ATRIBUTOS DE CALIDAD

ATTRIBUTO DE CALIDAD	EFFECTO	ARGUMENTO
Disponibilidad	+	REST y MongoDB permiten alta disponibilidad y escalabilidad horizontal.
Eficiencia/Rendimiento	+/-	Buen rendimiento general, pero Node puede fallar en tareas intensivas de CPU.
Escalabilidad	+	Todas las tecnologías soportan escalamiento horizontal fácilmente.
Mantenibilidad	+	TypeScript, Clean Architecture y patrones en frontend facilitan el mantenimiento.
Seguridad	+/-	Existen herramientas, pero depende de su correcta implementación.
Usabilidad	+	React y SPA ofrecen una experiencia de usuario fluida e intuitiva.
Portabilidad	+	Tecnologías multiplataforma y compatibles con Docker.
Reusabilidad	+	Atomic Design y lógica desacoplada permiten alta reutilización.

MATRIZ DE ATRIBUTOS DE CALIDAD

ATTRIBUTO DE CALIDAD	EFFECTO	ARGUMENTO
Robustez	+/-	Buenas bases, pero depende del manejo correcto de errores y rendimiento.
Capacidad de prueba	+	Arquitectura modular facilita pruebas unitarias e integración.
Flexibilidad	+	Modularidad permite adaptar o cambiar tecnologías fácilmente.
Integridad	+	Validaciones y tipado fuerte aseguran consistencia de datos.
Interoperabilidad	+	Buen soporte para integración con APIs y otros sistemas.
Fiabilidad	+	Replicación, manejo de errores y pruebas aumentan la confiabilidad.

MATRIZ DE TÁCTICAS DE DISEÑO

TÁCTICA	NODE.JS / EXPRESS	MONGO DB	REACT	REST/ JSON	CLIENTE/ SERVIDOR	EXPLICACIÓN
Optimización de consultas	Medio	Alto	N/A	Medio	Medio	MongoDB usa índices eficientes; Express necesita optimización manual.
Manejo de concurrencia	Alto	Medio	Bajo	Alto	Medio	Node.js maneja bien la concurrencia, pero puede bloquearse con CPU intensiva. REST soporta múltiples solicitudes.
Manejo de errores y fallos	Alto	Medio	Medio	Medio	Medio	Express maneja errores centralizados; MongoDB necesita reconexión; React captura errores de renderizado.
Escalabilidad	Alto	Alto	Medio	Alto	Alto	Todo el stack soporta escalado horizontal y separación de responsabilidades.
Facilidad de integración	Alto	Alto	Alto	Alto	Alto	REST/JSON facilita integración con servicios externos y capas desacopladas.

MATRIZ DE TÁCTICAS DE DISEÑO

TÁCTICA	NODE.JS / EXPRESS	MONGO DB	REACT	REST/ JSON	CLIENTE/ SERVIDOR	EXPLICACIÓN
Seguridad	Medio	Medio	Medio	Medio	Medio	Herramientas como JWT, roles en MongoDB, y escape en React requieren implementación adecuada.
Replicación	N/A	Alto	N/A	N/A	N/A	MongoDB soporta replicación nativa para alta disponibilidad.
Monitoreo	Medio	Alto	Bajo	Medio	Medio	MongoDB ofrece monitoreo avanzado, Node.js requiere herramientas externas y React depende de herramientas como Sentry.

MATRIZ DE PATRÓN DE DISEÑO

PATRÓN DE DISEÑO	APLICACIÓN EN EL STACK	RESUMEN
MVC (Model-View-Controller)	Backend: Express como controlador, lógica de negocio como modelo, y respuesta JSON como vista. Frontend: React + Flux sigue un modelo MVVM.	MVC se aplica tanto en el backend como en el frontend, manteniendo separación de responsabilidades.
Observer	Flux: las vistas se suscriben a los stores. React: renderizado reactivo al cambio de estado. Backend: uso con EventEmitter.	React y Flux siguen un modelo reactivo, donde las vistas responden a cambios de estado.
Dependency Injection / Inversión de Dependencias	Backend: Clean Architecture inyecta repositorios en los casos de uso. Frontend: React Context.	Facilita el desacoplamiento entre componentes y servicios.
Singleton	Backend: conexión MongoDB compartida, servicios únicos. Frontend: stores en Flux como instancia única.	Asegura que solo haya una instancia de ciertos servicios o recursos.
Factory	Backend: funciones factory para crear middlewares o repositorios. Frontend: inicialización de hooks.	Proporciona una forma flexible de crear objetos según el entorno o configuración.

MATRIZ DE PATRÓN DE DISEÑO

PATRÓN DE DISEÑO	APLICACIÓN EN EL STACK	RESUMEN
Composite	Frontend: React y Atomic Design modelan componentes complejos como combinación de componentes más simples.	Estructura modular y jerárquica que facilita la construcción de interfaces complejas.
Adapter	Backend: controladores y repositorios actúan como adaptadores. Frontend: capas que adaptan respuestas de APIs.	Facilita la integración entre sistemas con interfaces diferentes.
Facade	Backend: API REST simplifica el acceso a la lógica interna. Frontend: funciones que encapsulan múltiples llamadas a APIs.	Proporciona una interfaz simplificada y centralizada para el acceso al sistema.
Middleware / Chain of Responsibility	Backend: Express procesa solicitudes a través de una cadena de middlewares.	Se encarga de manejar tareas comunes como autenticación, validación y errores de manera modular.

MATRIZ DE ANÁLISIS DE MERCADO LABORAL

TECNOLOGÍA	DEMANDA (EMPRESAS)	NUEVAS OFERTAS (TENDENCIA)	SALARIO PROMEDIO MENSUAL (COP)	COMPETENCIA (TALENTO DISPONIBLE)
MongoDB	Alta	Alta	4.500.000 – 7.000.000	Media
Express/Node	Alta	Alta	6.000.000 – 9.000.000	Alta
React	Muy Alta	Muy Alta	5.000.000 – 14.000.000	Alta
Node.js	Muy Alta	Muy Alta	6.500.000 – 12.000.000	Alta
TypeScript	Alta	Muy Alta	5.000.000 – 8.000.000	Media

IMPLEMENTACIÓN

DIAGRAMA DE ALTO NIVEL

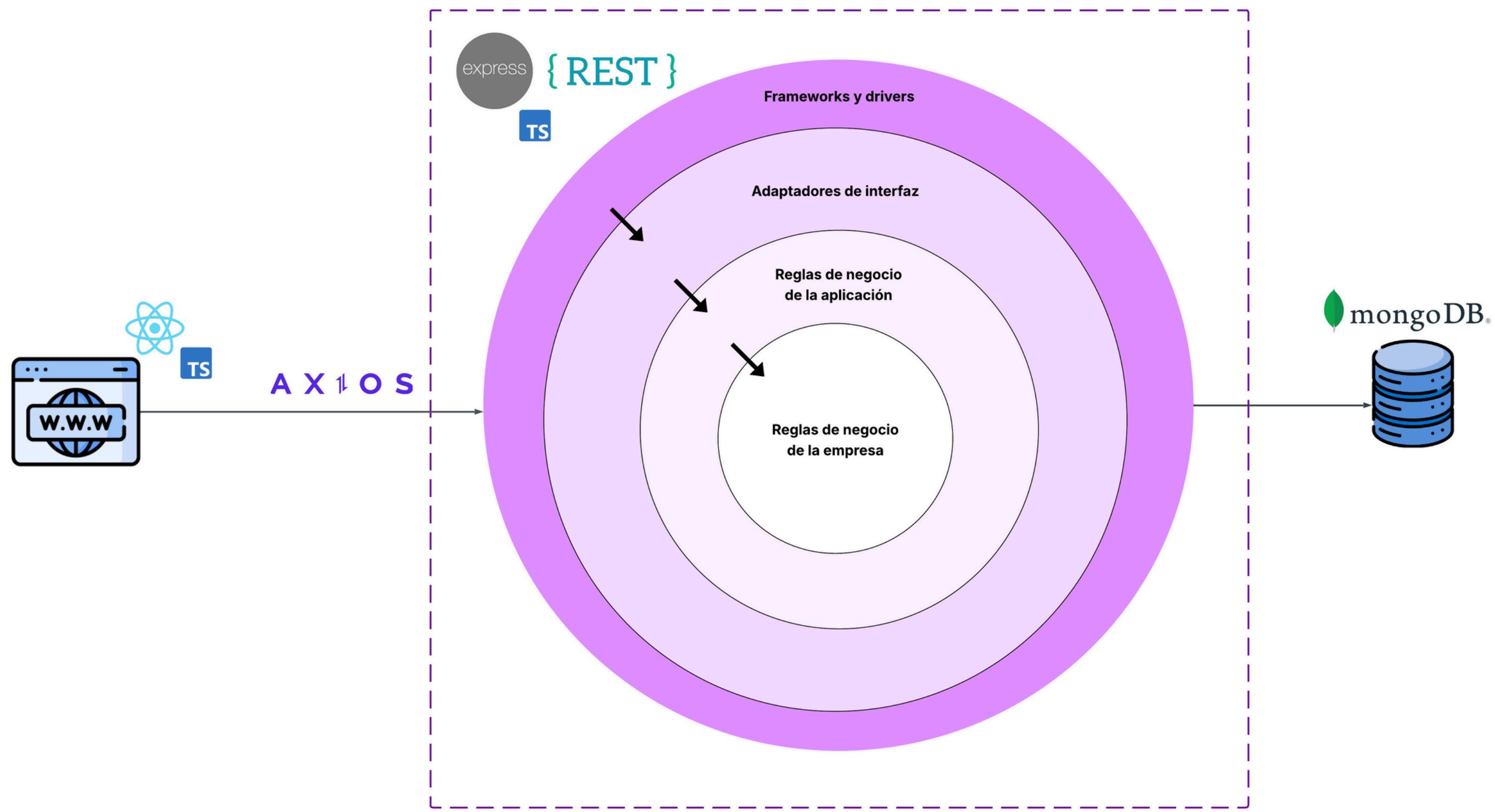


DIAGRAMA DE CONTEXTO C4

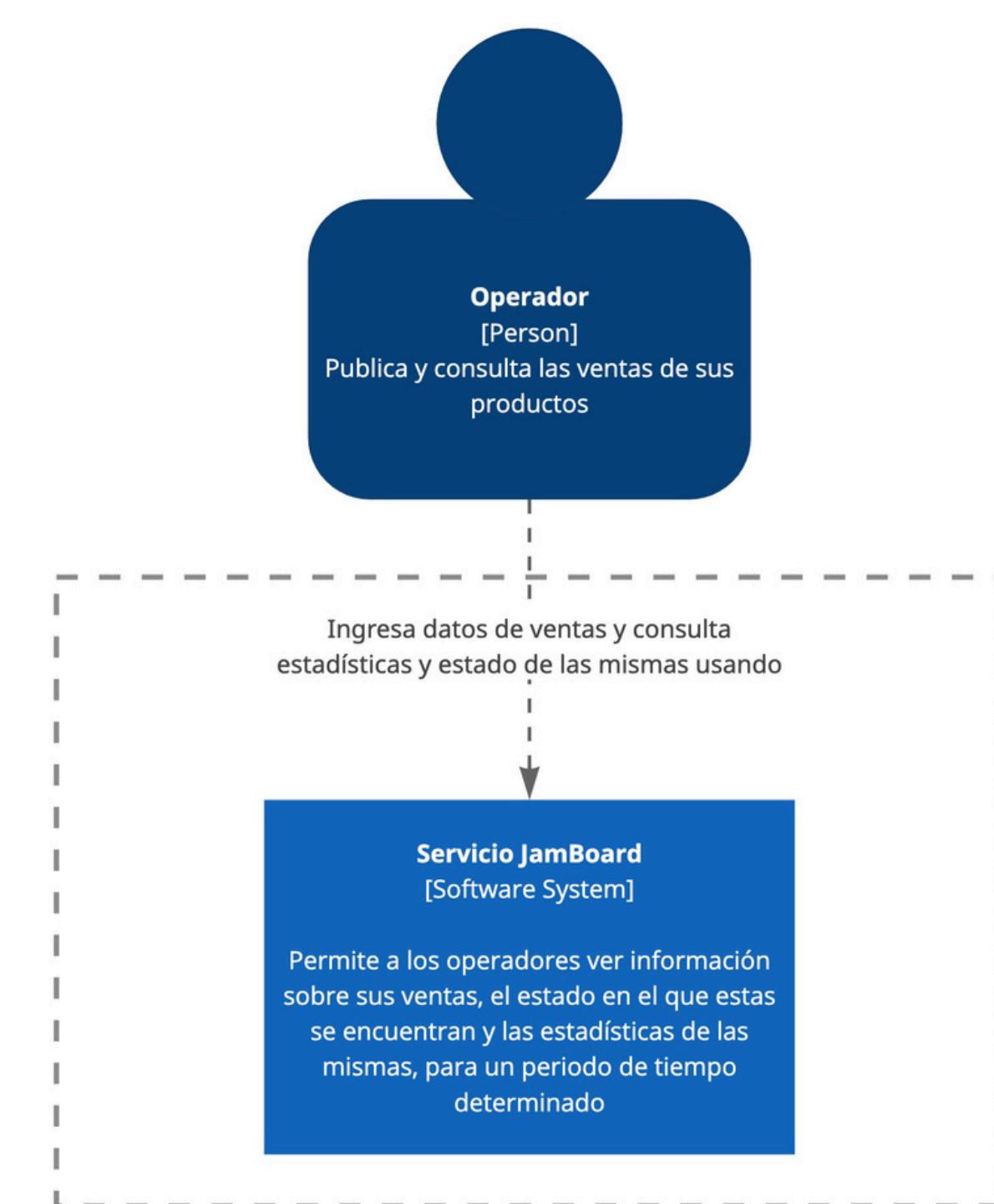


Diagrama de contexto del sistema para JamBoard

DIAGRAMA DE CONTENEDORES C4

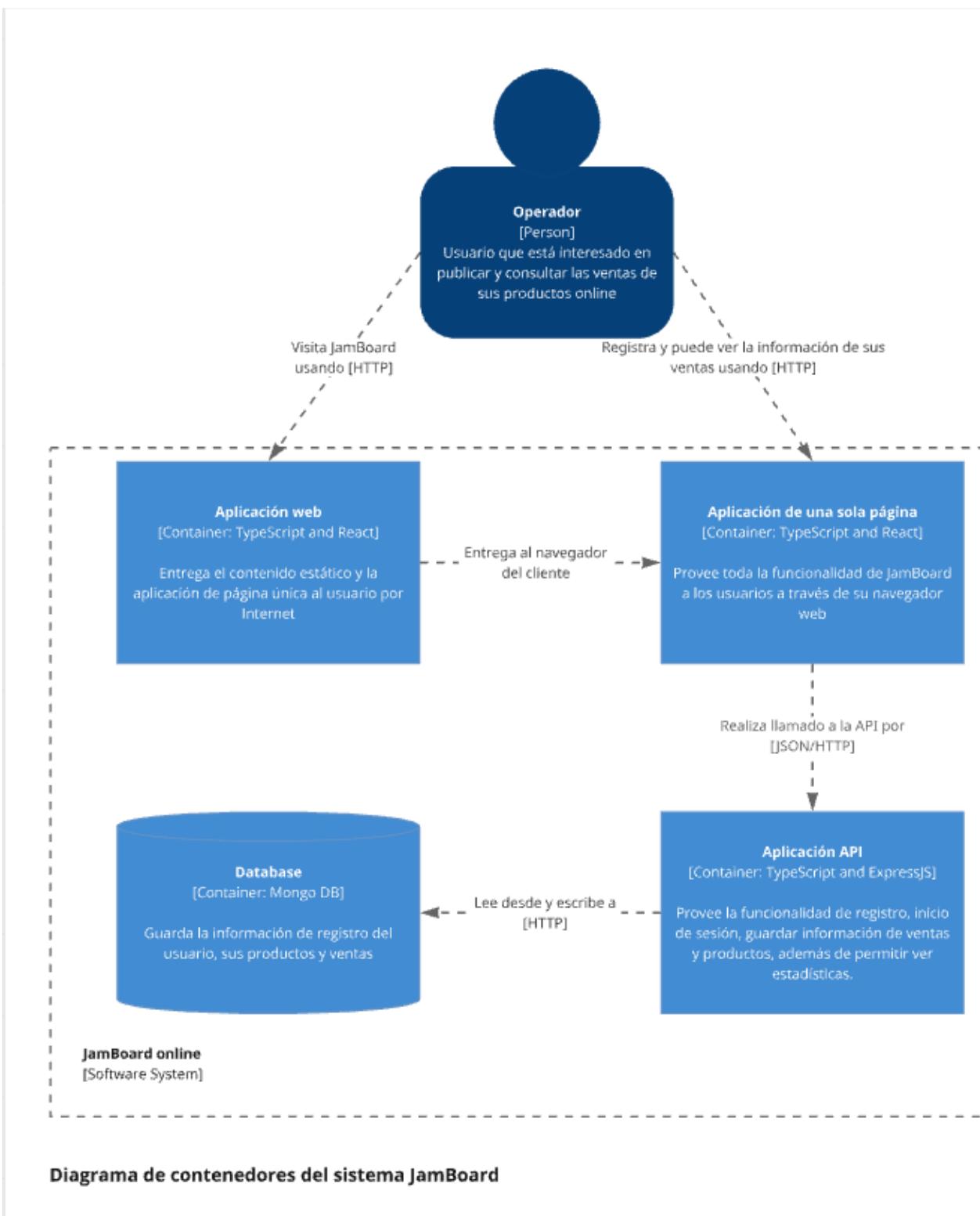


DIAGRAMA DE COMPONENTES C4

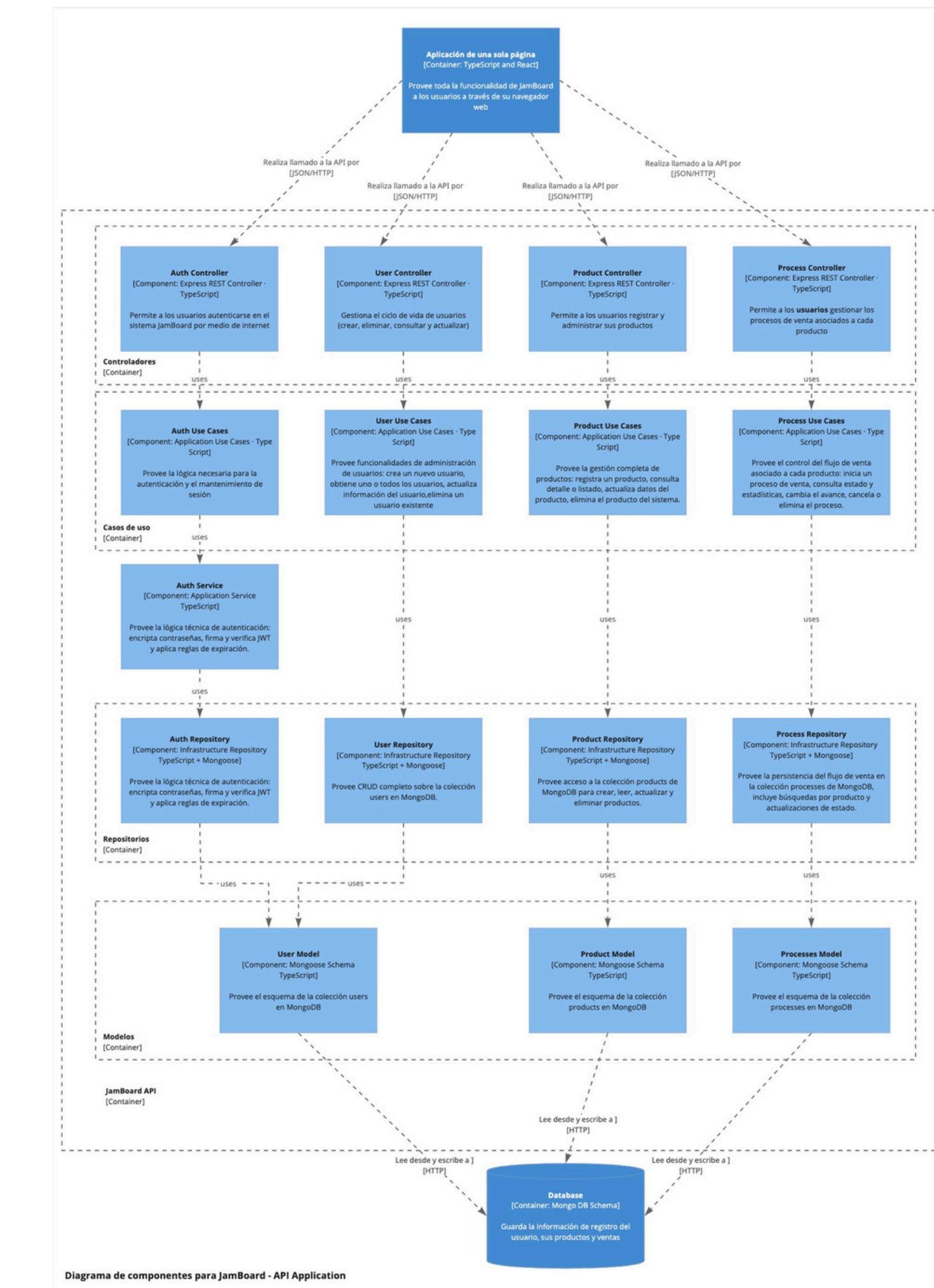


DIAGRAMA DINÁMICO C4

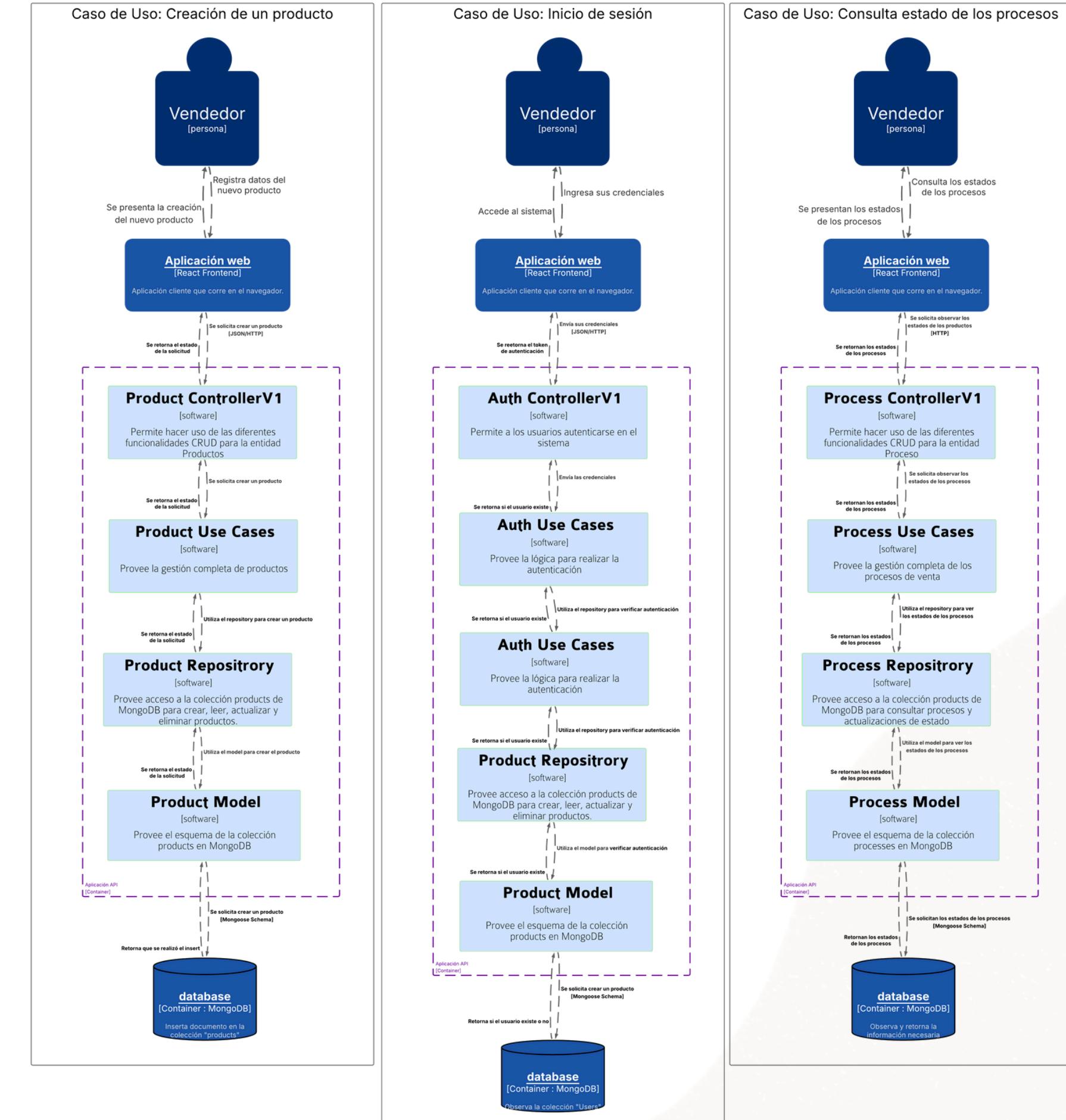


DIAGRAMA DE DESPLIEGUE C4

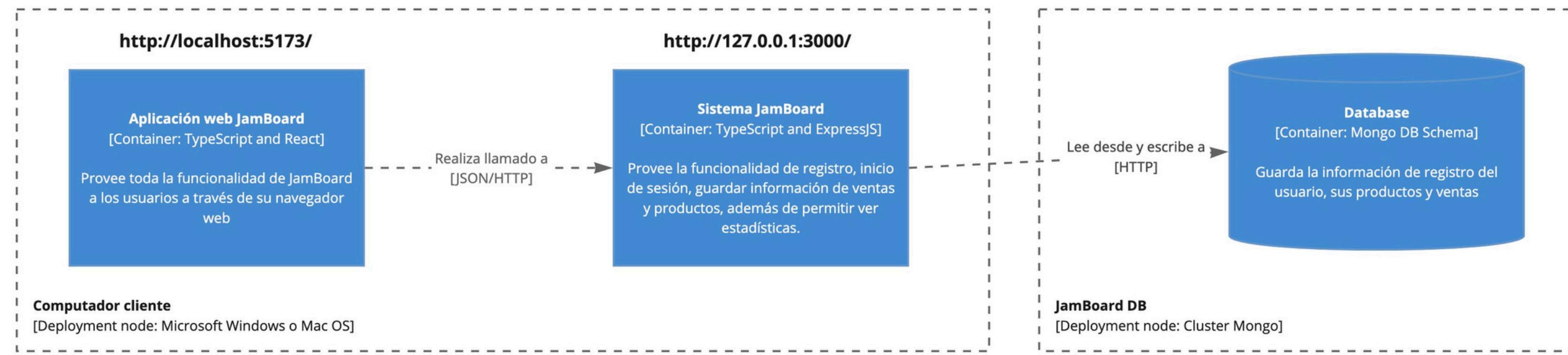
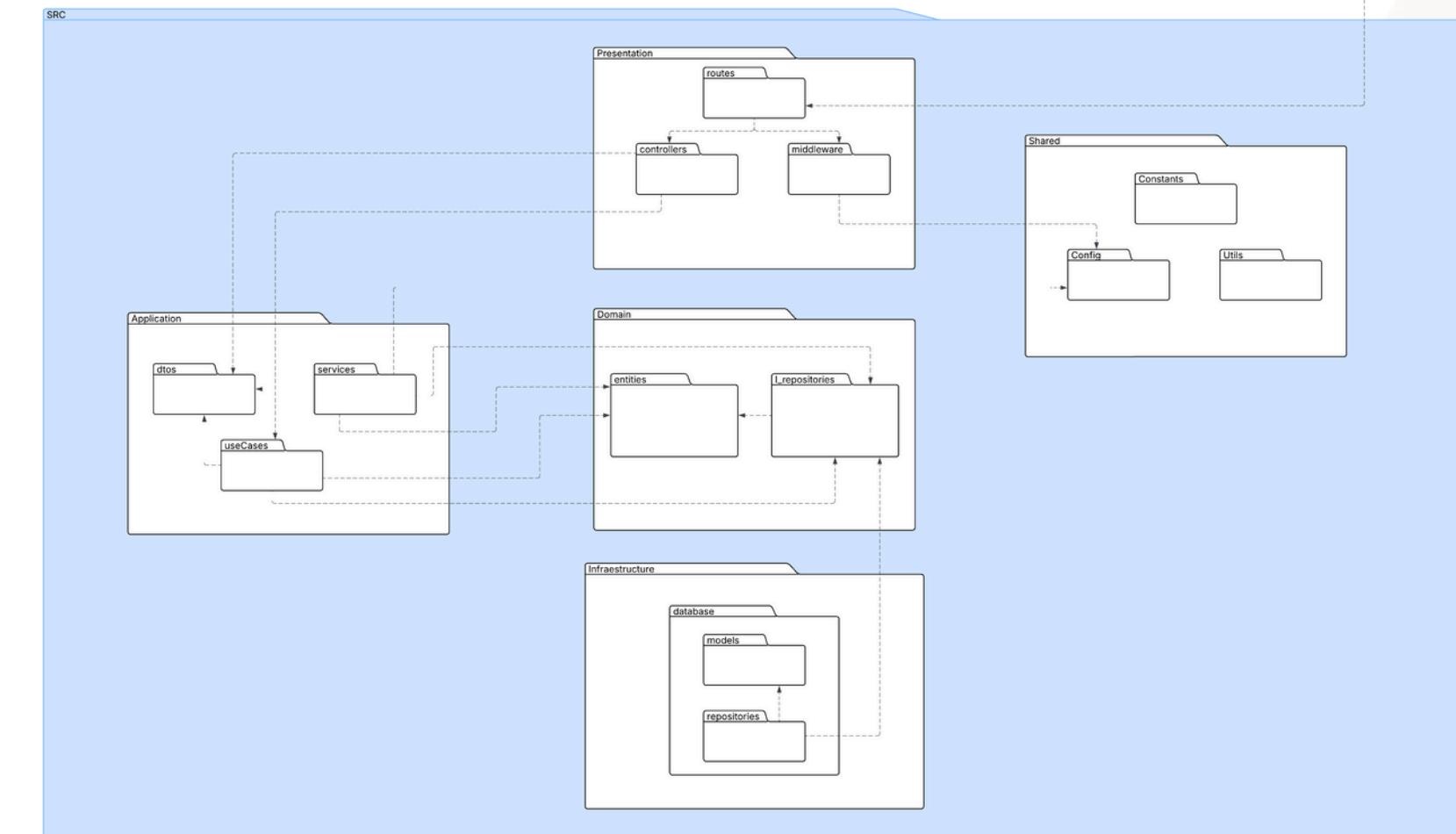
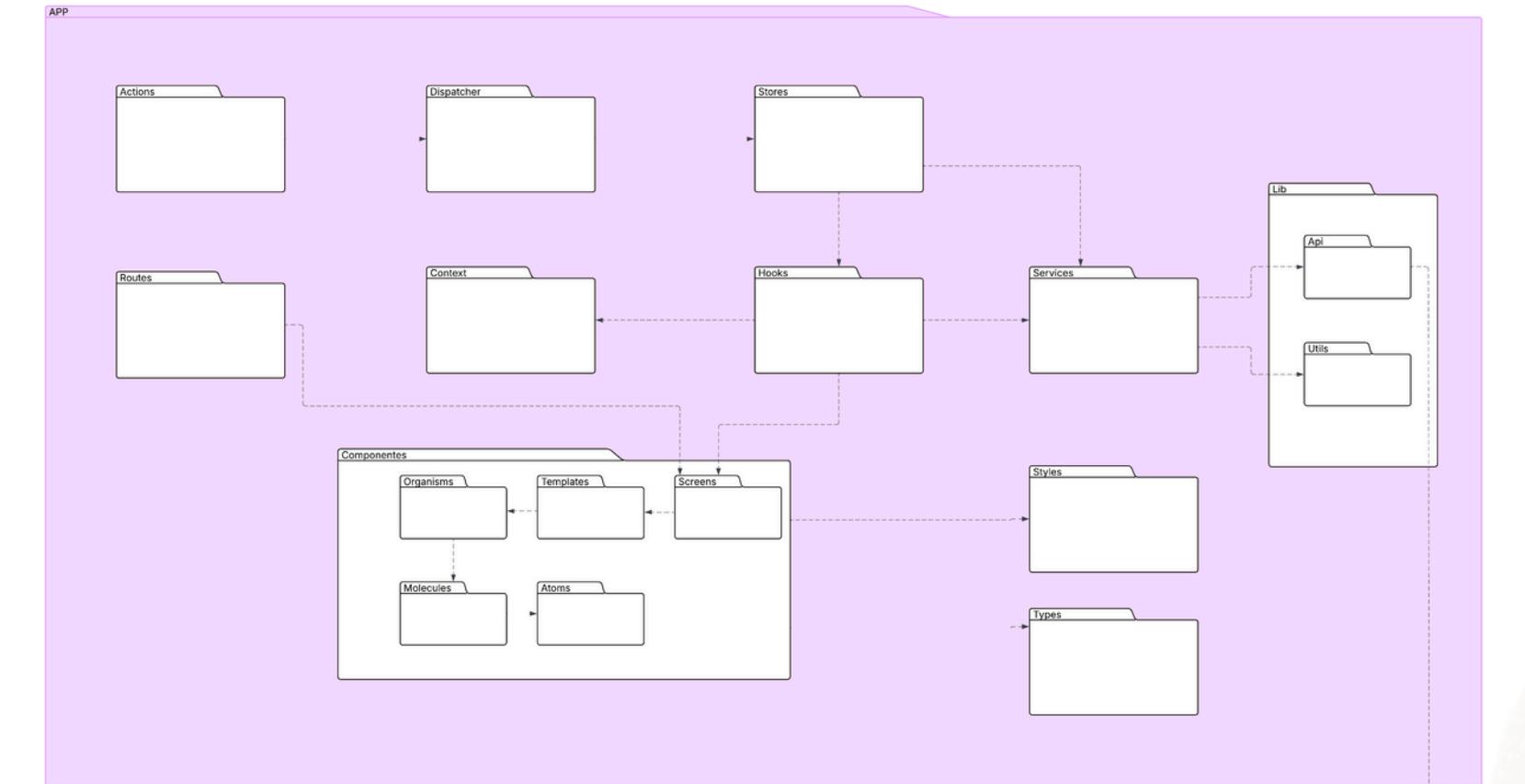
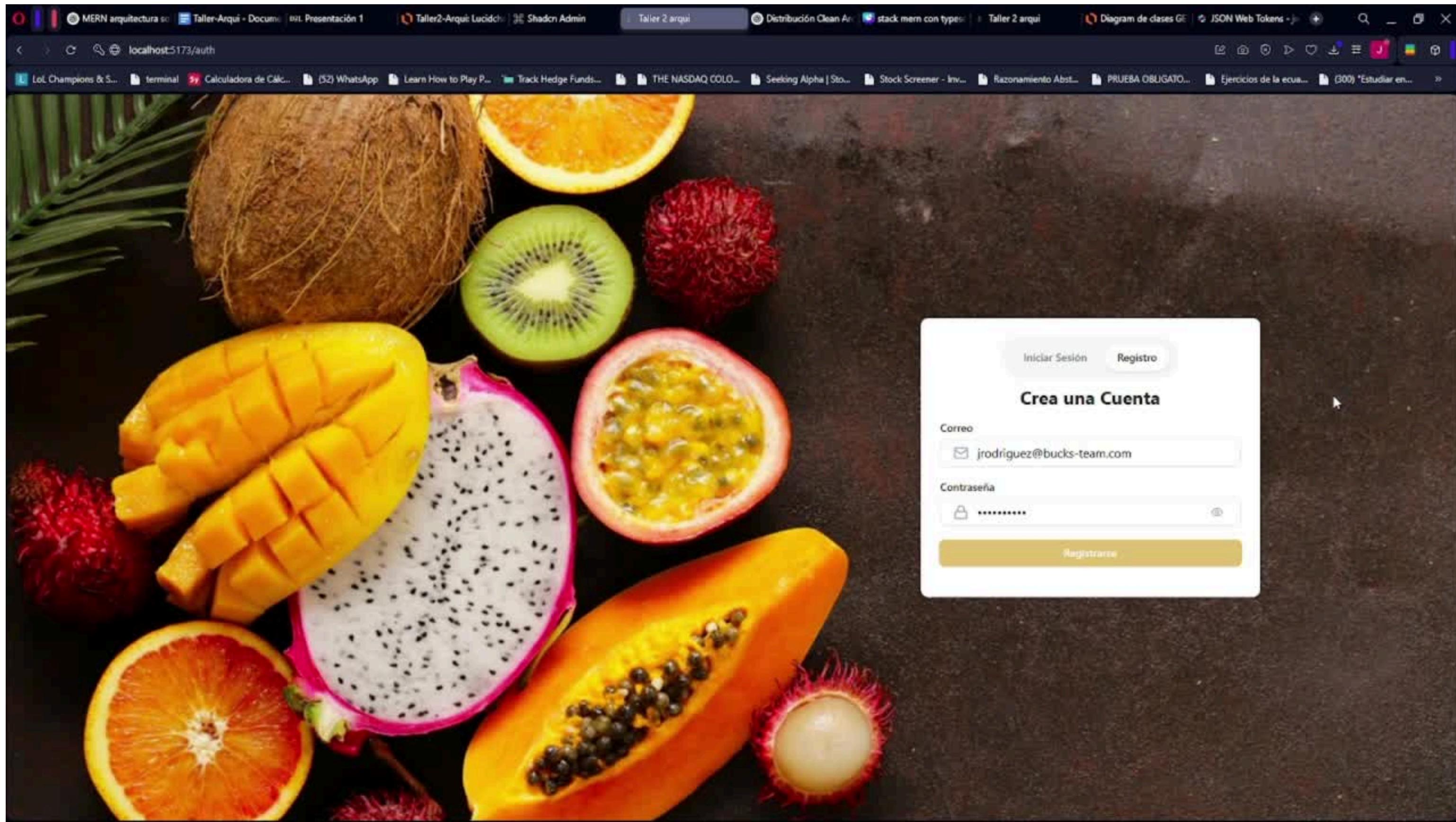


Diagrama de despliegue para JamBoard

DIAGRAMA DE PAQUETES C4



DEMO





M u c h a s
G R A C I A S