



PROBLEMÁTICA

En la actualidad, muchos negocios dependen de software obsoleto con interfaces poco amigables, carencia de automatización en procesos críticos y limitaciones para sincronizar datos en tiempo real.

InventoryView DIAGRAMA DE CLASES +displayInventoryList(): void +showProductDetails(product: Product): void POSController InventoryController Inventory +startNewSale(employee: Employee): Sale POSView -products: List<Product> +searchProduct(query: str): List<Product> +updateProductStock(productId: int, quantity: int): void +addProductToSale(saleId: int, productId: int, quantity: int): void +addProduct(product: Product): void +manualStockAdjustment(productId: int, newQuantity: int): void +displayProducts(): void +removeProductFromSale(productId: int, saleId: int): void +getLowStockProducts(): List<Product> +removeProduct(product: Product): void +inputPaymentMethod(): void +applyDiscount(saleId: int, discount: float): void +findProduct(productId: int): Product +getProductsExpiringSoon(): List<Product> +displaySaleDetails(sale: Sale): void +setPaymentMethod(saleId: int, paymentMethod: str): void +getLowStockAlerts(): List<Product> +suggestRestockProducts(): List<Product> +showError(message: str): void +addNoteToSale(saleId: int, note: str): void +suggestRestock(): List<Product> +finalizeSale(saleId: int): Receipt LoginView Product -id: int Sale +displayLoginForm(-name: str -description: str -id: int -price: float -date: datetime Employee -stock: int -employee: Employee -id: int -expirationDate: datetime -products: List <Product> -name: str SecurityController quantities: List<int> +updateStock(quantity: int): void -username: str totalAmount: float +isLowStock(): boolean -passwordHash: str -paymentMethod: str +authenticateUser(username: str, password: str): boolean +isExpiringSoon(): boolean -role: str -discount: float +authenticateBiometric(employeeld: int, biometricData: str): boolean -biometricData: str +checkPermission(employeeId: int, permissionName: str): boolean -notes: str +authenticate(username: str, password: str): boolean +addProduct(product: Product, quantity: int): void +checkln(): void +checkOut(): void Receipt +removeProduct(product: Product): void +calculateTotal(): float -receiptld: int Shift +applyDiscount(discount: float): void -sale: Sale +processPayment(): boolean -generatedAt: datetime -shiftld: int +generateRéceipt(): Receipt -startTime: datetime +generateDigital(): void -endTime: datetime +printReceipt(): void -employee: Employee EmployeeController +assignEmployee(employee: Employee): void +exchangeShift(withEmployee: Employee): boolean Report Metrics FinancialMetricsController +employeeCheckIn(employeeId: int): void +employeeCheckOut(employeeld: int): void -id: int -totalSales: float +assignShift(shiftld: int, employeeld: int): void +suggestEmployeeShifts(employeeld: int): List<Shift> -date: datetime +generateFinancialDashboardMetrics(): Metrics -totalExpenses: float -sales: List<Sale> +sendReportByEmail(reportId: int, email: str): void -profit: float Schedule -expenses: List<Expense> +exchangeShifts(shiftld: int, fromEmployeeld: int, toEmployeeld: int): boolean -shifts: List<Shift> +generateSalesReportPDF(): File +generateExpenseReportExcel(): File +suggestShifts(employee: Employee): List<Shift> DashboardView EmployeeView +displayMetrics(metrics: Metrics) +updateDashboard() +displayEmployeeSchedule(employee: Employee): void Expense +showEmployeeDetails(employee: Employee): void +inputShiftChangeRequest(): Shift id: int SalesReportController ExpenseReportController -date: datetime -description: str +generateSalesReport(format: str): File +generateExpenseReport(format: str): File -amount: float +sendReportByEmail(reportId: int, email: str): void +sendReportByEmail(reportId: int, email: str): void SalesReportView ExpenseReportView +displaySalesReport(): void +displayExpenseReportView(): void

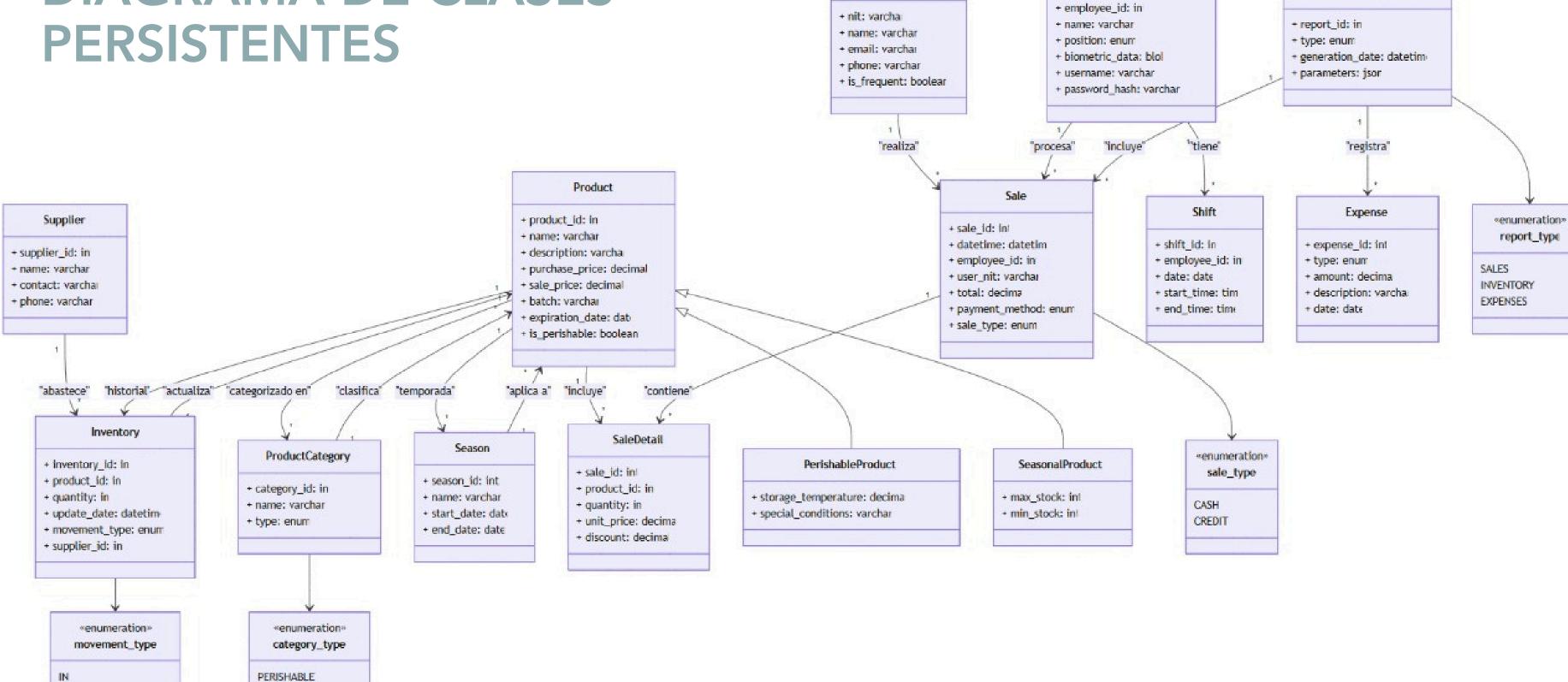
DIAGRAMA DE CLASES

SEASONAL

GENERAL

OUT

ADJUSTMENT

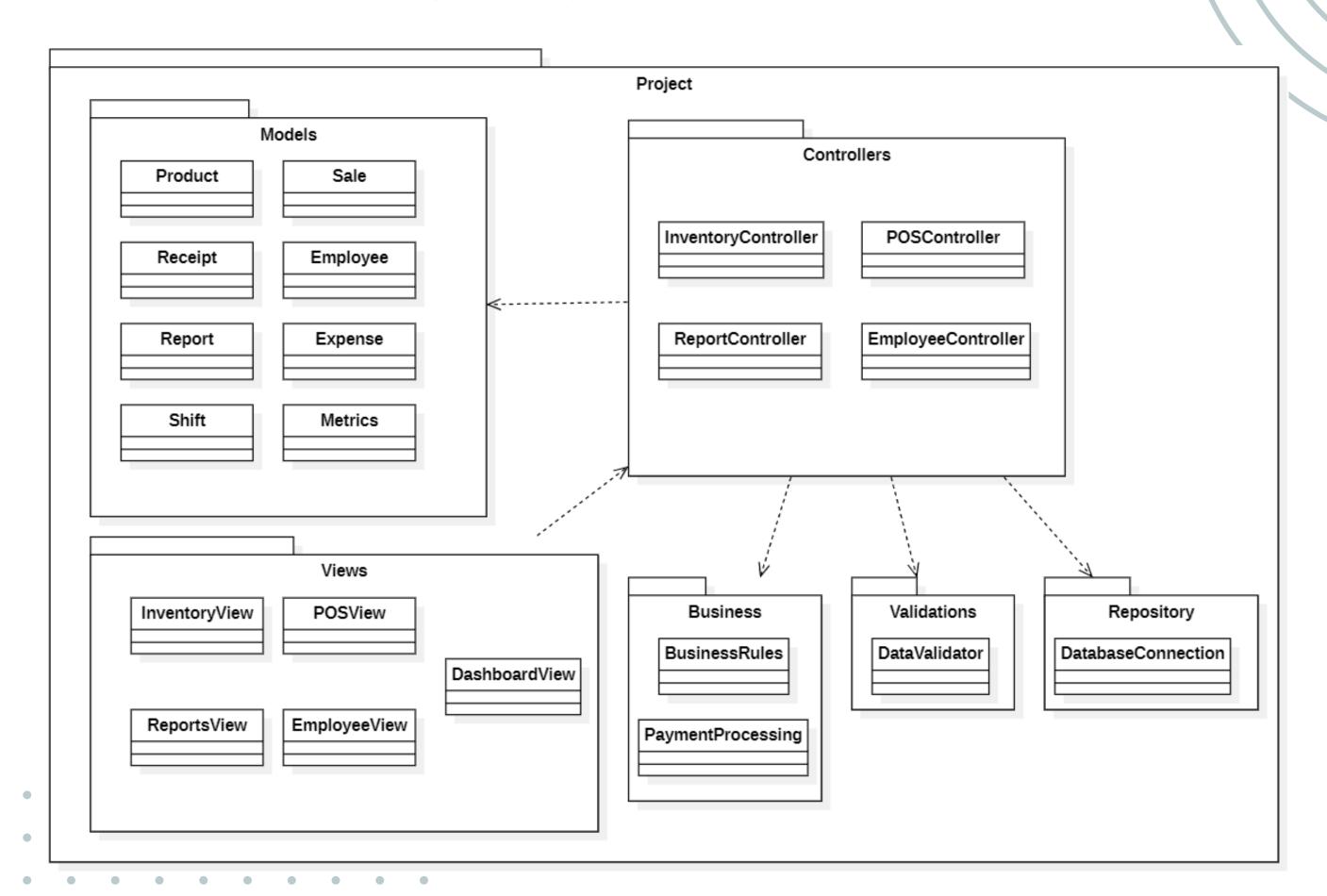


User

Employee

Report

DIAGRAMA DE PAQUETES



TECNOLOGÍAS PARA FRONTEND

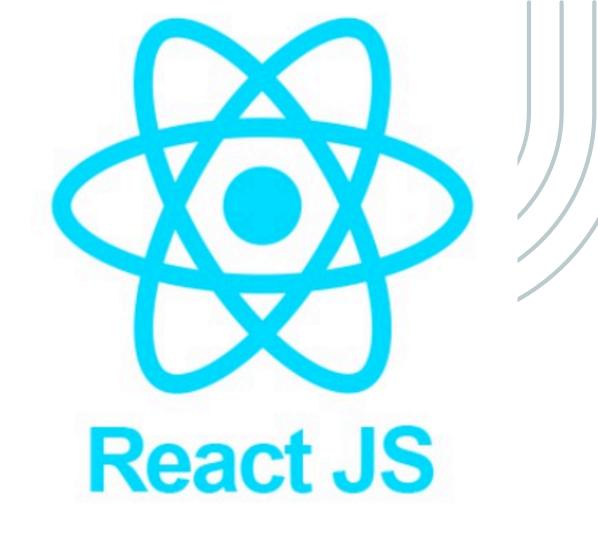
Cuadro Comparativo: ventajas y desventajas

Tecnología	Ventajas	Desventajas
React.js (JavaScript)	 Amplio ecosistema de librerías y herramientas Flexible y adaptable a diferentes arquitecturas Soporte de renderizado del lado del servidor (SSR). 	 JSX y manejo de estados que requieren un mayor dominio. Puede ser complejo para principiantes. Actualizaciones pueden generar problemas de compatibilidad
Vue.js (JavaScript)	 Fácil de aprender. Sintaxis más simple que React. Reactividad incorporada facilita la gestión de estados simples 	 Menos opciones de librerías y herramientas en comparación con React Puede ser menos adecuado para aplicaciones muy grandes y complejas.
Angular (TypeScript)	 Arquitectura bien definida y escalable Soporte oficial de Google. Buen manejo de grandes aplicaciones. 	 Mayor rigidez en la estructura puede limitar la flexibilidad Tamaño del bundle puede ser mayor en comparación con otros frameworks Proceso de actualización puede ser complejo.

.

React JS

- Se pueden crear interfaces de usuario interactivas y de alto rendimiento.
- Centrada en componentes reutilizables, facilitando la organización y escalabilidad del código.
- Su gran comunidad y recursos facilitan la resolución de problemas.



TECNOLOGÍA SELECCIONADA

TECNOLOGÍAS PARA BACKEND

Cuadro Comparativo: ventajas y desventajas

Tecnología	Lenguaje	Ventajas	Desventajas
Spring Boot	Java	 Gran ecosistema y comunidad amplia. Soporte nativo para multi-tenant con Hibernate. Seguridad robusta con Spring Security. Escalabilidad horizontal con microservicios. 	 Curva de aprendizaje pronunciada. Configuración compleja para aplicaciones grandes. Más consumo de recursos que opciones más livianas.
Django + Django Tenants	Python	 Rápido desarrollo con enfoque DRY. Módulo "Django Tenants" para manejar bases de datos compartidas o separadas. Admin integrado para gestión de inquilinos. 	 Menos eficiente para cargas extremadamente altas. No es ideal para arquitecturas de microservicios complejos.

.

.

.

Node.js + NestJS	JavaScript/TypeScript	 Rendimiento asíncrono para alta concurrencia. NestJS ofrece estructura modular similar a Spring. Flexible para distintos patrones multi-tenant (DB compartida o aislada). 	 Mayor complejidad al gestionar grandes bases de datos. Más susceptible a errores de ejecución por su naturaleza asíncrona.
Laravel	PHP	 Framework experimentado y bien documentado. Soporte para multi-tenancy con paquetes como "Tenancy for Laravel". Fácil despliegue en infraestructuras tradicionales. 	 Rendimiento comparativamente más lento que otros Frameworks. Dependencia de demasiadas librerías externas.
Ruby on Rails + Apartment	Ruby	 Rápido desarrollo con convenciones claras. Gem "Apartment" para manejar multi-tenancy. Comunidad sólida y bien documentada. 	 Rendimiento menor que opciones como Java o C#. Dificultad para escalar horizontalmente.

.

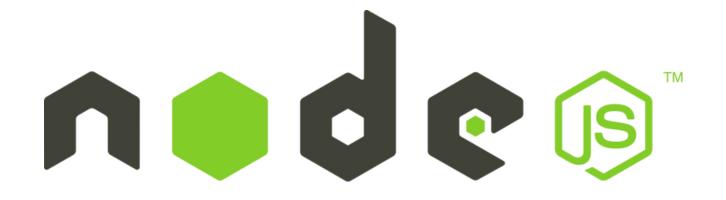
.

NODE.JS

Permite desarrollar aplicaciones eficientes y escalables, especialmente en sistemas basados en eventos y en tiempo real.

Ofreciendo un alto rendimiento, ideal para aplicaciones con muchas conexiones concurrentes, y se usa comúnmente para APIs RESTful.

Usa el mismo lenguaje que el frontend, facilitando el desarrollo.



TECNOLOGÍA SELECCIONADA

TECNOLOGÍAS PARA PERSISTENCIA DE DATOS

Cuadro Comparativo: ventajas y desventajas

TECNOLOGÍAS DE PERSISTENCIA

TECNOLOGÍA	VENTAJAS	DESVENTAJAS
PostgreSQL	-Soporte avanzado para transacciones (ACID)Escalable y maneja grandes volúmenes de datosSeguridad avanzada con autenticación basada en roles y cifrado.	-Mayor consumo de RAM y CPU en comparación con MySQL Requiere mayor conocimiento técnico para su configuración y administración.
MySQL	-Buen rendimiento en consultas de lectura. -Fácil de aprender y usar. -Amplia documentación	- Menos eficiente en transacciones complejas en comparación con PostgreSQL. - Soporte limitado para consultas analíticas avanzadas.
MariaDB	-Más rápido en ciertas operaciones que MySQL. -Mayor escalabilidad.	-Menos soporte empresarial que SQL Server y PostgreSQLPuede presentar problemas de compatibilidad en algunas aplicaciones.
SQL Server	-Buen soporte técnico y actualizaciones constantesSeguridad avanzada con cifrado de datos y control de accesoAlto rendimiento en sistemas transaccionales y analíticos.	-Licencia de pago con costos elevados. -Mayor consumo de recursos en comparación con otras opciones.

PostgreSQL

PostgreSQL fue seleccionado por su escalabilidad, seguridad y rendimiento, que el proyecto necesita.

Su capacidad para manejar grandes volúmenes de datos sin afectar el rendimiento garantiza la disponibilidad y la escalabilidad de los datos.

Ofrece cifrado y control de accesos avanzados, asegurando la protección de datos sensibles.



TECNOLOGÍA SELECCIONADA

