Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)

Selección y Justificación de Tecnologías para el proyecto Pos-Ya

Este documento detalla y argumenta acerca de la selección de tecnologías, frameworks, bibliotecas y herramientas que se pretenden utilizar para el desarrollo de la app Pos-Ya, considerando sus ventajas, sus desventajas, su relación con los objetivos del curso y, por su puesto, su alineación con las capacidades de los integrantes del equipo 🦔.

Lenguaje de Programación y Frameworks Principales

Lenguaje: JavaScript, versión ES6+.
Frameworks: React.js para Frontend, y tanto Node.js cómo Express para backend.

Ventajas:

- Permite el desarrollo full-stack con un solo lenguaje, facilitando la integración, la colaboración y el mantenimiento del proyecto a lo largo del semestre.
- Posee una gran comunidad, abundante documentación, recursos y soporte en foros; esto es fundamental en caso de dudas o dificultades en alguna implementación específica.
- React ofrece componentes reutilizables, virtual DOM y una UI reactiva y moderna, lo cual es ideal para interfaces escalables y, además, acaudaladas en colores y diseño.
- Node.js es bastante eficiente para aplicaciones I/O intensivas, APIs REST y tiempo real; así mismo, posee un ecosistema de paquetes que otras tecnologías muy diverso.
- El ecosistema de desarrollo resulta ser compatible con herramientas modernas, tales como Vite, Tailwind, Electron, ESLint, entre otras.
- Se facilita considerablemente la integración con Electron para aplicaciones de escritorio multiplataforma.
- Node.js y Express facilitan la creación de una aplicación monolítica, integrando el backend, la lógica del negocio, el acceso a los datos y la base de datos en un solo lugar.

Desventajas:

- El rendimiento de los cálculos intensivos disminuye si lo comparamos con Java o C#.
- JavaScript ofrece una gran flexibilidad, la cual, puede llevar a errores si no se aplican buenas prácticas y herramientas de linting.
- El tipado dinámico puede dificultar -en algunos casos- el mantenimiento de proyectos muy grandes. No obstante, esto puede mitigarse con el uso de TypeScript.

¿Por qué no otro lenguaje de programación u otra tecnología 🤔?

Desde luego, antes de decidirnos por JavaScript, React y Node.js, analizamos otras alternativas ampliamente usadas en la industria para el desarrollo de sistemas de tipo POS. Dichas alternativas fueron:

- **Java/Spring Boot:** es un estándar en el desarrollo de aplicaciones empresariales robustas y seguras. Su ecosistema es muy maduro, con soporte para arquitecturas complejas, integración con bases de datos y despliegue en la nube. Muchos miembros del equipo han tenido contacto académico o profesional con Java, lo que lo hacía una opción viable.
- **Python/Django:** Python es conocido por su sintaxis sencilla y su rapidez para prototipar; Django, en particular, permite construir aplicaciones web completas en poco tiempo, con un enfoque en la productividad y la seguridad. Es una tecnología popular en startups y proyectos académicos, y algunos integrantes del equipo han trabajado con Python en cursos previos.
- **C#/ASP.NET:** C# y el framework ASP.NET son ampliamente utilizados en el sector empresarial, especialmente en entornos Windows; Ofrecen herramientas avanzadas, buen rendimiento y soporte para aplicaciones de escritorio y web. Su integración con herramientas de Microsoft y su robustez lo hacían atractivo para proyectos de mayor escala o con necesidades específicas de integración.

Para cada alternativa identificamos claras ventajas y desventajas:

	Java/Spring Boot	**Python/Django**	**C#/ASP.NET**
Ventajas	● Es muy robusto y seguro, estándar en sistemas empresariales. ● Posee un tipado fuerte y orientación a objetos, facilita el mantenimiento en proyectos grandes. ● Tiene un excelente soporte para bases de datos, transacciones y arquitecturas complejas. ● Ostenta un amplio soporte para despliegue en la nube y microservicios.	● Posee una sintaxis sencilla, lo cual propicia el desarrollo rápido. ● Tiene una gran comunidad y una abundante documentación. ● Ideal para prototipos, scripts y aplicaciones web sencillas. ● Tiene un buen soporte para bases de datos y ofrece seguridad.	● Ostenta una gran potencia en entornos empresariales y corporativos. ● Se caracteriza por un buen rendimiento y herramientas de desarrollo avanzadas. ● Posee una integración nativa con Windows y el ecosistema Microsoft. ● Tiene soporte para aplicaciones web y de escritorio.
Desventajas	● La curva de aprendizaje resulta ser alta, por lo que requiere mayor experiencia previa. ● Es menos ágil para prototipado rápido y cambios frecuentes. ● No posee tanta flexibilidad para interfaces de usuario modernas y multiplataforma.	● No es tan eficiente en interfaces altamente interactivas o SPA. ● Posee una menor integración nativa con Electron y apps de escritorio. ● No resulta ser tan adecuado para aplicaciones con necesidades gráficas avanzadas.	● A grandes rasgos, resulta menos flexible para multiplataforma. ● Posee una comunidad open source menor en frontend. ● El licenciamiento y la complejidad es mayor en algunos escenarios.

Por esto elegimos JavaScript, Node.js y React.js 😉

El equipo tiene experiencia previa en JavaScript y React, esto -indudablemente- resulta ser muy beneficioso, pues permite una mayor productividad, calidad y velocidad de entrega; la arquitectura basada en componentes facilita el mantenimiento y la escalabilidad, alineándose con los objetivos del curso de construir una solución moderna, eficiente, monolítica y facil de evolucionar. Adicionalmente, la comunidad y el soporte aseguran que cualquier dificultad pueda ser resuelta de manera efectiva y rápida.

Base de Datos Relacional Seleccionada

Tecnología: SQLite.

Ventajas:

- Esta tecnología es ligera, fácil de instalar y de administrar. Resulta ser ideal para aplicaciones de escritorio, prototipos y despliegues rápidos.
- No se requiere de un servidor dedicado, lo que simplifica el despliegue, reduce costos y elimina dependencias externas.
- Es compatible con Electron y Node.js, permitiendo integración directa y persistencia local.
- Soporta SQL estándar y es ampliamente utilizada en aplicaciones comerciales y móviles.

Desventajas:

- No es muy óptima para aplicaciones de alta concurrencia o grandes volúmenes de datos.
- Es menos adecuada para entornos distribuidos o multiusuario a gran escala.

¿Por qué no otra tecnología para la base de datos 🤔?

Como es de esperarse, antes de decidirnos por SQLite, analizamos otras alternativas ampliamente usadas en diferentes industrias del desarrollo de aplicaciones comerciales. Dichas alternativas fueron:

MySQL/PostgreSQL

Ventajas:

- Se caracterizan por ser bastante robustas y escalables, además de ampliamente usadas en la industria.
- Poseen un soporte avanzado para transacciones, integridad y concurrencia.
- Gozan de una gran comunidad, documentación y soporte multiplataforma.
- Son compatibles con la mayoría de lenguajes y frameworks modernos.

Desventajas:

- Requieren configuración y administración de un servidor dedicado.
- Poseen mayor complejidad operativa y de mantenimiento.

- Consumen más recursos y pueden ser excesivas para proyectos pequeños o de escritorio.

SQL Server

Ventajas:

- Es potente y seguro, estándar en entornos empresariales y corporativos.
- Tiene una excelente integración con herramientas Microsoft y Windows.
- Posee soporte avanzado para análisis, replicación y administración.

Desventajas:

- Posee licenciamiento y costos asociados en la mayoría de escenarios.
- Tiene una complejidad mayor en instalación y administración.
- Es menos portable para aplicaciones de escritorio multiplataforma.

Por esto elegimos SQLite 😉

SQLite es suficiente para la escala y necesidades del proyecto, permitiendo un desarrollo ágil y sin sobrecarga de administración, alineado con los recursos y objetivos del equipo; su integración directa con Node.js y Electron permite una experiencia de usuario fluida y sin dependencias externas. Además, SQLite destaca por su portabilidad y bajo consumo de recursos, lo que permite ejecutar la aplicación en casi cualquier equipo sin requerir instalaciones adicionales ni configuraciones complejas. Esto facilita el uso en entornos académicos o de prototipo, donde la simplicidad y la rapidez de despliegue son esenciales.

Bibliotecas y Herramientas Complementarias

Dependencias principales:

- **@headlessui/react:** provee componentes accesibles y sin estilos para construir modales, menús y listas desplegables, mejorando la accesibilidad y experiencia de usuario.
- **cors:** permite el manejo seguro de peticiones entre frontend y backend durante el desarrollo y despliegue.
- **express:** otorga framework minimalista y robusto para construir APIs REST en Node.js, ampliamente adoptado y documentado.
- **jspdf y jspdf-autotable:** permite la generación de reportes PDF y tablas desde el frontend, permitiendo exportar información relevante para el usuario.
- **react**: se trata de la biblioteca principal para construir la interfaz de usuario, basada en componentes y virtual DOM.
- **react-dom:** permite renderizar componentes React en el DOM del navegador o en Electron.
- **react-icons:** provee una amplia variedad de íconos vectoriales modernos y personalizables.
- **sqlite3:** es el driver oficial para interactuar con bases de datos SQLite desde Node.js, asegurando persistencia local eficiente.

Dependencias de desarrollo:

- **@eslint/js, eslint, eslint-plugin-react-hooks, eslint-plugin-react-refresh, globals:** son herramientas de linting y análisis estático que se encargan de mantener la calidad, evitar errores comunes y mantener un código consistente.
- **@types/react, @types/react-dom:** ofrece tipado para desarrollo con TypeScript o para mejorar la experiencia en editores y herramientas de análisis.
- **@vitejs/plugin-react, vite:** Vite es un bundler y servidor de desarrollo ultrarrápido, optimizado para React y hot reload.
- **autoprefixer, postcss, tailwindcss:** permiten escribir CSS moderno, responsivo y compatible con todos los navegadores, usando utilidades y clases de Tailwind.
- **concurrently:** permite ejecutar múltiples scripts (backend, frontend, Electron) en paralelo durante el desarrollo, mejorando la productividad.
- **electron:** es un framework para empaquetar la aplicación como escritorio multiplataforma (Windows, Mac, Linux), integrando Node.js y frontend web.
- **wait-on:** sincroniza el arranque de procesos, asegurando que Electron solo inicie cuando el frontend esté listo.

Cada dependencia fue seleccionada para cubrir una necesidad específica del proyecto: accesibilidad, experiencia de usuario, generación de reportes, desarrollo ágil, calidad de código, compatibilidad multiplataforma y despliegue sencillo. El uso de ESLint y Prettier (configurados en el proyecto) asegura que el código siga estándares modernos y sea fácil de mantener por cualquier miembro del equipo.