**Documentación Arquitectónica Exhaustiva: Sistema de Gestión de Tareas (sgestiontask2)**

**1. Contexto del Proyecto y Stack Tecnológico de Alto Rendimiento**

El proyecto sgestiontask2 es una aplicación de gestión de tareas construida sobre un *stack* moderno, priorizando el rendimiento, la experiencia del desarrollador (DX) y la escalabilidad. La arquitectura está firmemente cimentada en el paradigma **declarativo** y tipado estático.

|  |  |  |
| --- | --- | --- |
| Capa | Tecnología | Rol Principal y Justificación Técnica |
| Meta-Framework | **Nuxt 3 (Vue 3)** | Sirve como el *framework* Full-Stack, proveyendo características cruciales como **Auto-Imports**, **Enrutamiento Basado en Archivos** (pages/), y **SSR/SSG** (aunque actualmente se ejecuta como SPA de desarrollo, la arquitectura permite fácil migración a SSR). |
| UI/UX | **Vuetify 3** | Implementación de Material Design, proporcionando un vasto catálogo de componentes reactivos y accesibles. Su configuración se maneja mediante un *plugin* personalizado, desacoplándolo del sistema de módulos de Nuxt. |
| Tipado | **TypeScript (TS)** | Fundamental para la **Integridad de Datos**. El tipado estático previene errores de *runtime* (TypeError: Cannot read properties of undefined) y mejora significativamente la refactorización y la colaboración. |
| Gestión de Estado | **Pinia** | El sistema de gestión de estado recomendado por el equipo de Vue. Ofrece una API minimalista, tipado inherente con TS y una arquitectura basada en módulos (stores) de fácil comprensión. |
| Estilo | **CSS y Variables CSS** | Utilización de *Custom Properties* (--spacing-md) en assets/css/variables.css para centralizar constantes de diseño no dependientes del sistema de *theming* de Vuetify. |

**2. Arquitectura Lógica: Principios y Patrones**

El diseño del Front-End sigue un estricto principio de **Separación de Preocupaciones (SoC)** y el patrón **Composición** (gracias al Composition API de Vue 3).

**2.1. Desacoplamiento de Componentes y Lógica (Composables)**

La lógica de negocio y el acceso a datos están estrictamente separados de la capa de presentación:

* **useTasks.ts (Capa de Acceso a Datos - DAL):** Este *composable* actúa como un *middleware* entre la aplicación Vue y la API externa.
  + **Encapsulamiento de la API:** Oculta los detalles de la comunicación HTTP (useFetch), la autenticación (token), y el formato específico de la API (ej., Content-Type: application/x-www-form-urlencoded).
  + **Gestión de Autenticación (Token):** El *composable* es el responsable de adjuntar el token de usuario necesario para la API (USER\_TOKEN). **Actualmente, este token está codificado (hardcoded) directamente en el archivo useTasks.ts** así mismo la clave (USER\_TOKEN) **es inyectada dinámicamente** desde un archivo de configuración (.env o nuxt.config.ts) mediante el *runtime config* de Nuxt. Esta refactorización es crucial para la seguridad y la portabilidad del código.
  + **Funcionalidad Pura:** Implementa funciones *puras* de CRUD (getOne, list, create, update, remove) que pueden ser llamadas desde cualquier lugar, manteniendo el código DRY (Don't Repeat Yourself).

**2.2. Flujo de Datos (Pinia Store)**

El estado global de la aplicación se gestiona mediante la Pinia Store, ejemplificada en taskStore.ts.

* **Estado (State):** Contiene la lista de tareas, los parámetros de filtro/búsqueda y el estado de carga/error.
* **Acciones (Actions):** Las acciones son los únicos lugares autorizados para modificar el estado (mediante mutaciones directas) e invocar la lógica del negocio. Por ejemplo, taskStore.fetchTasks() llama a useTasks.list().
* **Gestión del Listado:** La taskStore es responsable de mantener la lista de tareas actualizada, permitiendo que pages/index.vue y components/TaskCard.vue reaccionen automáticamente a los cambios.

**2.3. Estructura de Componentes (Atomic Design. Similitud)**

Se seleccionó una estructura de componentes con similitudes al Atomic Design para el proyecto porque es un patrón de diseño que promueve la modularidad, la reutilización y la escalabilidad, cualidades esenciales en un framework moderno como Vue/Nuxt.

|  |  |  |  |
| --- | --- | --- | --- |
| Categoría | Ejemplo | Función | Principio Aplicado |
| Moléculas (molecules/) | Pagination.vue, SearchBar.vue | Elementos de UI con lógica propia y un único propósito. | Reutilización y Modularidad. |
| Organismos (organisms/) | TaskCard.vue, TaskForm.vue | Bloques de UI complejos, combinando moléculas y lógica local. | Componentización y Encapsulamiento de la Presentación. |
| Páginas (pages/) | index.vue, tasks/[id].vue | Orquestan la composición de organismos y acceden a stores/composables. | Separation of Concerns (Orquestación vs. Lógica). |

**3. Manejo de Datos y Correcciones de Integración (TypeScript)**

La fase de integración con la API requirió una serie de decisiones críticas para garantizar la solidez de los tipos.

**3.1. Estandarización de Propiedades de la Tarea**

* **Problema:** La API utiliza el *string* '1' o '0' en el campo is\_completed, mientras que el código de Vue/TS espera un **booleano** (completed: true/false).
* **Solución (Mapeo en onMounted):** En pages/tasks/[id].vue, la respuesta de la API (apiResponse) es inmediatamente mapeada al objeto task.value de la siguiente manera

*task*.*value* *=* {

*...data*,

      is\_completed: *isCompletedValue*,

      date: *data*.*due\_date* *||* *data*.*date* *||* ''

    }

* **Lógica de Renderizado Corregida:** El *computed* property isCompleted se simplificó a **computed(() => !!task.value?.completed)**. Esto obliga al *template* y a Vuetify a depender del valor booleano mapeado, resolviendo los errores de visualización del v-chip.

**4. Gestión del Estilo y Theming (Vuetify)**

La personalización visual se logra manteniendo una clara separación entre los estilos del *framework* (Vuetify) y los estilos de la aplicación.

* **plugins/vuetify.ts:**
  + Define el tema custom (myDarkTheme) y lo establece como **defaultTheme**.
  + La clave es sobrescribir los colores semánticos de Vuetify (primary, success, warning) para que todos los componentes heredados se adapten automáticamente al esquema de color verde deseado.
* **Utilidades de Diseño:** El *template* utiliza clases de utilidad de Vuetify (ej., d-flex, text-h5, mb-4, ml-2) para un diseño responsivo y consistente, siguiendo los lineamientos de Material Design sin escribir CSS de bajo nivel.