Tecnológico de Monterrey

Sprint 1 (Design & Architecture) (David)

Jesus Enrique Bañales Lopez | A01642425

Luis Fernando Cuevas Arroyo | A01647254

Carlos Tellez Bermudez | A01637089

Aaron Hernandez Jimenez | A01642529

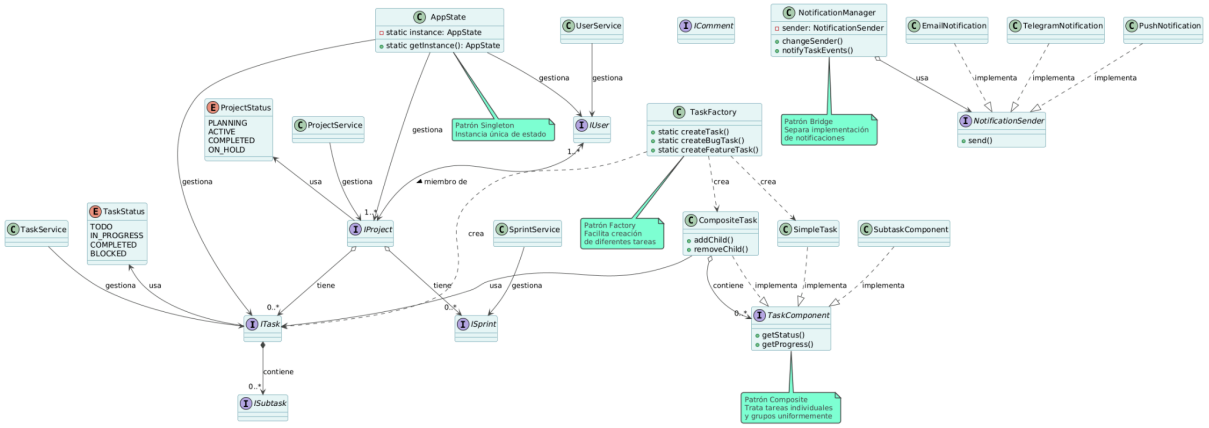
Diego Villanueva Terrazas | A01568601

Aram Barsegyan | A01642781

Planeación de sistemas de software

3 abr 2025

UML Diagram:



1. **Implementar un sistema modular y escalable para la gestión de tareas:**
    - El sistema se ha diseñado con componentes independientes que se comunican a través de interfaces bien definidas.
    - La organización del código en módulos (interfaces, modelos, patrones, servicios, componentes) facilita su mantenimiento y extensión.
    - La arquitectura permite añadir nuevas funcionalidades sin afectar al código existente.
    - Los servicios están separados de la lógica de negocio y de la interfaz de usuario.
2. **Aplicar principios SOLID a través del uso de patrones de diseño:**
    - **S (Responsabilidad Única)**: Cada clase tiene una única responsabilidad (por ejemplo, TaskFactory solo se encarga de crear tareas, NotificationManager solo de gestionar notificaciones).
    - **O (Abierto/Cerrado)**: El código está abierto para extensión pero cerrado para modificación, permitiendo añadir nuevos tipos de notificaciones o tareas sin modificar el código existente.
    - **L (Sustitución de Liskov)**: Las clases derivadas (como SimpleTask y CompositeTask) pueden sustituir a sus clases base sin alterar el comportamiento.
    - **I (Segregación de Interfaces)**: Las interfaces están divididas en unidades más pequeñas y específicas (por ejemplo, TaskComponent, NotificationSender).
    - **D (Inversión de Dependencias)**: El código depende de abstracciones (interfaces) en lugar de implementaciones concretas.
3. **Demostrar el uso de patrones estructurales y creacionales en un entorno práctico:**
    - **Patrón Bridge (estructural)**: Usado para separar la abstracción de notificaciones (NotificationManager) de su implementación (EmailNotification, TelegramNotification, etc.).

- **Patrón Factory (creacional)**: Implementado en TaskFactory para encapsular la lógica de creación de tareas y subtareas.
- **Patrón Singleton (creacional)**: Aplicado en AppState para mantener una única instancia compartida del estado de la aplicación.
- **Patrón Composite (estructural)**: Utilizado para tratar tareas individuales y grupos de tareas (con subtareas) de manera uniforme a través de la interfaz TaskComponent.

El proyecto implementa todos estos patrones y principios en un contexto práctico de un sistema de gestión de tareas, con funcionalidades completas para gestionar proyectos, tareas, sprints, usuarios y más. La implementación es completamente funcional y podría utilizarse como base para una aplicación real con pequeñas modificaciones.