Software Design Document (SDD) Template (summarized from IEEE STD 1016)

Software design is a process by which the software requirements are translated into a representation of software components, interfaces, and data necessary for the implementation phase.

The SDD shows how the software system will be structured to satisfy the requirements.

It is the primary reference for code development and, therefore, it must contain all the information required by a programmer to write code.

The SDD is performed in two stages. The first is a ***preliminary design*** in which the overall system architecture and data architecture is defined. In the second stage, i.e. the ***detailed design*** stage, more detailed data structures are defined and algorithms are developed for the defined architecture.

This template is an annotated outline for a software design document adapted from the IEEE Recommended Practice for Software Design Descriptions.

# (Equipo 2)

**(My DevBot)**

# Software Design Document

Name (s):

Luis Ángel González Romo | A01235962

Aaron Inzunza Inzunza | A01114528

Alexander Alexeev Bogaychuk | A01740751

Ramón Esaú Gómez González | A00832787

Raúl Kenyu Medina Takada | A01236331

Lab Section: Workstation:

Date: (06/13/2024)

**TABLE OF CONTENTS**

1. **INTRODUCTION**
   1. Purpose
   2. Scope
   3. Overview
   4. Definitions and Acronyms
2. **SYSTEM OVERVIEW**
3. **SYSTEM ARCHITECTURE**
   1. Architectural Design
   2. Decomposition Description
   3. Design Rationale
4. **DATA DESIGN**
   1. Data Description
   2. Data Dictionary
5. **COMPONENT DESIGN**
6. **HUMAN INTERFACE DESIGN**
   1. Overview of User Interface
   2. Screen Images
7. **REQUIREMENTS MATRIX**
   1. Modo 1 (Usuario)
   2. Modo 2 (Manager)
   3. Modo 3 (Desarrollador)
8. **INTRODUCTION**
   1. **Purpose**

El propósito de este Documento de Diseño de Software (DDS) es esbozar las especificaciones de diseño y la arquitectura del proyecto de software de un Chatbot alojado en Oracle Cloud. Sirve como una guía integral para el equipo del proyecto, proporcionando información detallada sobre las decisiones de diseño, metodologías y tecnologías utilizadas a lo largo del ciclo de vida del proyecto. La audiencia principal de este documento incluye miembros del proyecto, instructores y otras partes interesadas involucradas en el proyecto.

## Scope

El software desarrollado para este proyecto tiene como objetivo principal mejorar la productividad y visibilidad del equipo de desarrollo mediante la automatización de tareas y la implementación de un ChatBot como servicio de comunicación y seguimiento.

Objetivos:

-Automatizar tareas repetitivas del equipo de desarrollo para aumentar la eficiencia y liberar tiempo para actividades de mayor valor.

-Proporcionar visibilidad a través de un ChatBot, permitiendo al manager y a cada miembro del equipo conocer el estado de las actividades y tareas en tiempo real.

-Mejorar la colaboración y la comunicación dentro del equipo mediante una plataforma centralizada y accesible.

Beneficios:

-Aumento de la productividad del equipo de desarrollo al reducir el tiempo dedicado a tareas administrativas y repetitivas.

-Mejora en la visibilidad y seguimiento de las actividades del equipo, lo que facilita la identificación de posibles cuellos de botella y la toma de decisiones informadas.

-Mayor colaboración y comunicación entre los miembros del equipo, lo que promueve un ambiente de trabajo más cohesionado y eficiente.

## Overview

Este Documento de Diseño de Software (SDD, por sus siglas en inglés) describe las especificaciones de diseño y la arquitectura del proyecto Chatbot alojado en Oracle Cloud. Sirve como una guía integral para el equipo del proyecto, detallando las decisiones de diseño, metodologías y tecnologías utilizadas a lo largo del ciclo de vida del proyecto. La audiencia principal incluye a los miembros del proyecto, instructores y otros stakeholders como Oracle. El software tiene como objetivo mejorar la productividad y visibilidad del equipo mediante la automatización de tareas e implementación de un Chatbot para la comunicación y seguimiento. El documento cubre la arquitectura del sistema, descomposición de subsistemas, justificación del diseño, diseño de datos, funcionalidad de los componentes e interacción del usuario para asegurar un proceso de desarrollo cohesivo y eficiente.

**1.4 Definitions and Acronyms**

* DevOps: La relación de trabajo colaborativo entre Desarrollo y Operaciones de TI.
* ChatBot: Un programa informático diseñado para simular conversaciones con usuarios humanos, especialmente a través de Internet.
* DDS: Documento de Diseño de Software.
* API: Interfaz de Programación de Aplicaciones.
* CI/CD: Integración Continua/Despliegue Continuo.

1. **SYSTEM OVERVIEW**

El proyecto se centra en el desarrollo de un sistema de mejora de productividad y visibilidad para el equipo de desarrollo de software de Oracle. Con el fin de lograr este objetivo, se implementará un ChatBot como servicio de comunicación y seguimiento, junto con la automatización de tareas clave del proceso de desarrollo.

**Funcionalidad**

Gestión de Tareas: Los miembros del equipo pueden utilizar el ChatBot para asignar y gestionar tareas. El ChatBot puede crear, actualizar y cerrar tareas en función de los comandos recibidos de los usuarios.

Consulta de Información: Los usuarios pueden realizar consultas al ChatBot para obtener información sobre el estado de los proyectos, detalles de las tareas, plazos de entrega, recursos disponibles, entre otros.

Interfaz de Usuario Intuitiva: El ChatBot proporcionará una interfaz de usuario intuitiva y fácil de usar, permitiendo a los miembros del equipo interactuar con él de manera natural utilizando comandos de voz o texto.

**Contexto**

En el dinámico entorno tecnológico actual, la calidad y la velocidad de entrega del software son factores críticos para el éxito empresarial. Los principios de DevOps, que enfatizan la colaboración entre desarrollo y operaciones, han emergido como una estrategia clave para alcanzar estos objetivos. Sin embargo, los equipos de desarrollo aún enfrentan desafíos relacionados con la productividad y la visibilidad de las actividades, lo que puede afectar negativamente la eficiencia y la calidad del software entregado.

**Diseño**

El diseño del sistema se basa en los principios de DevOps, promoviendo la colaboración entre el equipo de desarrollo y las operaciones de TI para lograr un flujo de trabajo rápido y confiable. Se implementará una arquitectura modular y escalable que permita la fácil integración de nuevas funcionalidades y la adaptación a los cambios en los requisitos del proyecto.

**Antecedentes**

El equipo de desarrollo de software de Oracle ha reconocido la necesidad de mejorar la productividad en un 20% y de aumentar la visibilidad de las actividades de cada miembro del equipo. Actualmente, las tareas rutinarias y repetitivas consumen una parte significativa del tiempo de los desarrolladores, lo que reduce su capacidad para enfocarse en actividades de mayor valor añadido. Además, la falta de visibilidad en tiempo real sobre el progreso de las tareas dificulta la gestión efectiva del proyecto, limitando la capacidad del manager para tomar decisiones informadas y oportunas.

1. **SYSTEM ARCHITECTURE**

## Architectural Design

En esta sección se detalla el diseño de arquitectura del sistema de gestión de tareas desplegado en Oracle Cloud Infrastructure, utilizando Kubernetes para la orquestación de contenedores y servicios variados para la gestión de tareas e interacción con los usuarios. A continuación se describen los subsistemas de alto nivel, sus responsabilidades y la colaboración entre ellos para lograr la funcionalidad deseada.

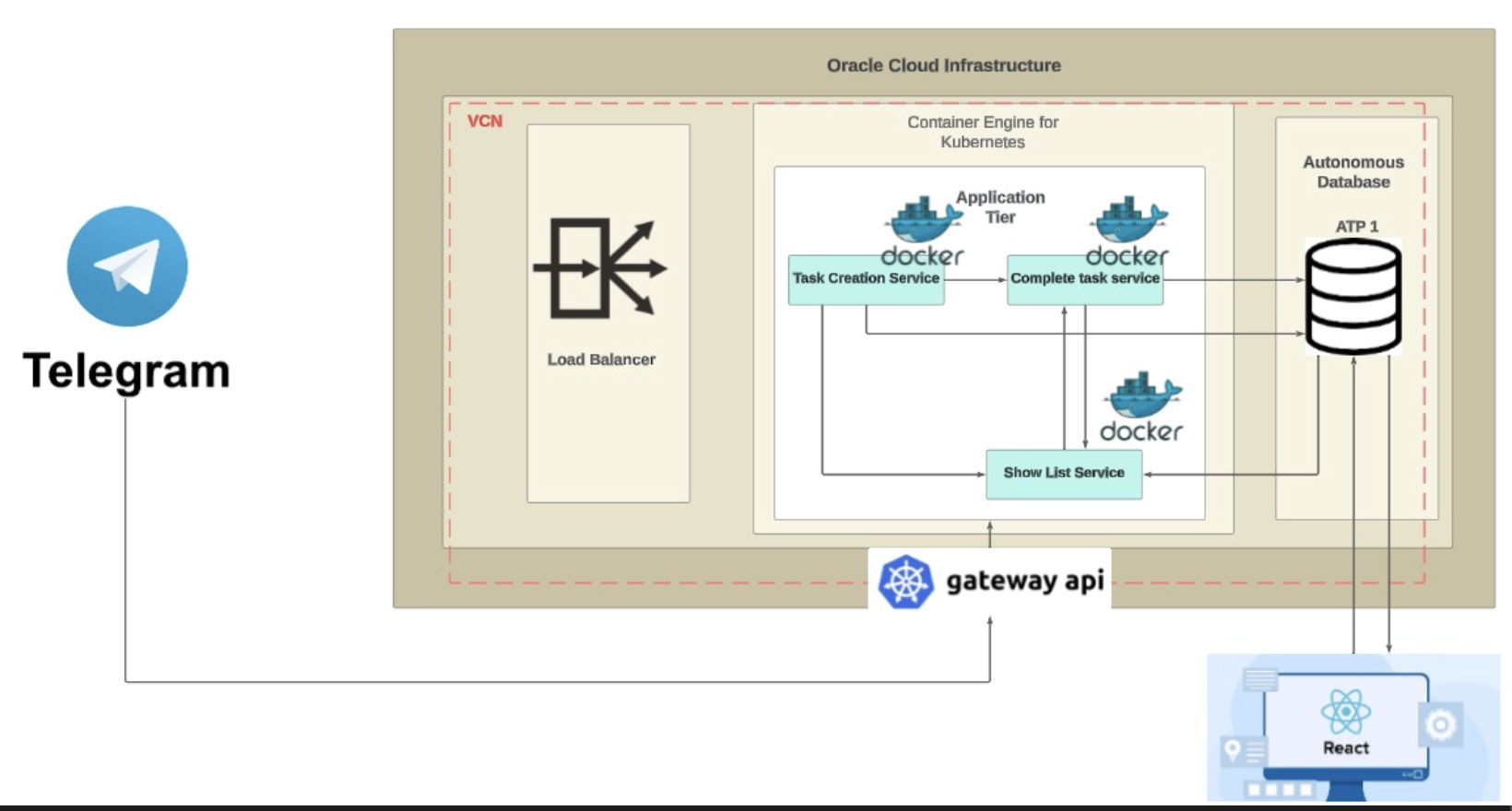
**Subsistemas de Alto Nivel:**

1. **Interfaz de Bot de Telegram:**
   * **Rol:** Proporciona una interfaz de usuario para la gestión de tareas a través de Telegram.
   * **Responsabilidades:** Permite a los usuarios crear, completar y ver tareas mediante mensajes en Telegram.
2. **Balanceador de Carga:**
   * **Rol:** Distribuye el tráfico entrante para asegurar un equilibrio de cargas entre los servicios.
   * **Responsabilidades:** Garantiza alta disponibilidad y confiabilidad al enrutar las solicitudes a los servicios backend adecuados.
3. **API Gateway:**
   * **Rol:** Actúa como un punto de entrada central para todas las solicitudes de los clientes.
   * **Responsabilidades:** Enruta las solicitudes desde Telegram y la aplicación React a los microservicios correspondientes dentro del clúster de Kubernetes.
4. **Capa de Aplicación (Microservicios):**
   * **Rol:** Contiene la lógica de negocio para la gestión de tareas.
   * **Responsabilidades:**
     + **Servicio de Creación de Tareas:** Gestiona la creación de nuevas tareas.
     + **Servicio de Completar Tareas:** Administra la finalización de tareas.
     + **Servicio de Mostrar Lista:** Recupera y muestra la lista de tareas.
5. **Motor de Contenedores para Kubernetes (OKE):**
   * **Rol:** Orquesta aplicaciones en contenedores.
   * **Responsabilidades:** Gestiona la implementación, escalado y operación de los contenedores de aplicaciones.
6. **Base de Datos Autónoma (ATP 1):**
   * **Rol:** Almacena datos persistentes para la aplicación.
   * **Responsabilidades:** Gestiona los datos de las tareas, incluyendo creación, estado de finalización y listas de tareas.
7. **Aplicación React:**
   * **Rol:** Proporciona una interfaz de usuario basada en web para la gestión de tareas.
   * **Responsabilidades:** Permite a los usuarios interactuar con el sistema a través de un navegador web, ofreciendo funcionalidad similar a la interfaz de Telegram.

### Colaboración entre Subsistemas:

1. **Interacción del Usuario:**
   * Los usuarios interactúan con el sistema a través de Telegram o la aplicación web React.
   * Ambas interfaces envían solicitudes al **API Gateway**.
2. **API Gateway:**
   * El **API Gateway** enruta las solicitudes entrantes a los microservicios adecuados dentro de la **Capa de Aplicación** según el tipo de solicitud (por ejemplo, crear tarea, completar tarea, mostrar lista de tareas).
3. **Servicios de la Capa de Aplicación:**
   * **Servicio de Creación de Tareas:** Recibe solicitudes para crear nuevas tareas, procesa los datos y almacena la nueva tarea en la **Base de Datos Autónoma**.
   * **Servicio de Completar Tareas:** Actualiza el estado de las tareas como completadas en la **Base de Datos Autónoma**.
   * **Servicio de Mostrar Lista:** Recupera la lista de tareas de la **Base de Datos Autónoma** y la devuelve al **API Gateway**, que luego envía los datos a la interfaz solicitante (Telegram o React).
4. **Balanceador de Carga:**
   * Asegura que las solicitudes se distribuyan eficientemente entre los diferentes microservicios, evitando que algún servicio se convierta en un cuello de botella.

**Diagrama**



## Decomposition Description

#### Descripción Funcional:

Para la descripción funcional, utilizaremos Diagramas de Flujo de Datos (DFD) y Diagramas de Descomposición Estructural para ilustrar la arquitectura y la interacción entre los subsistemas.

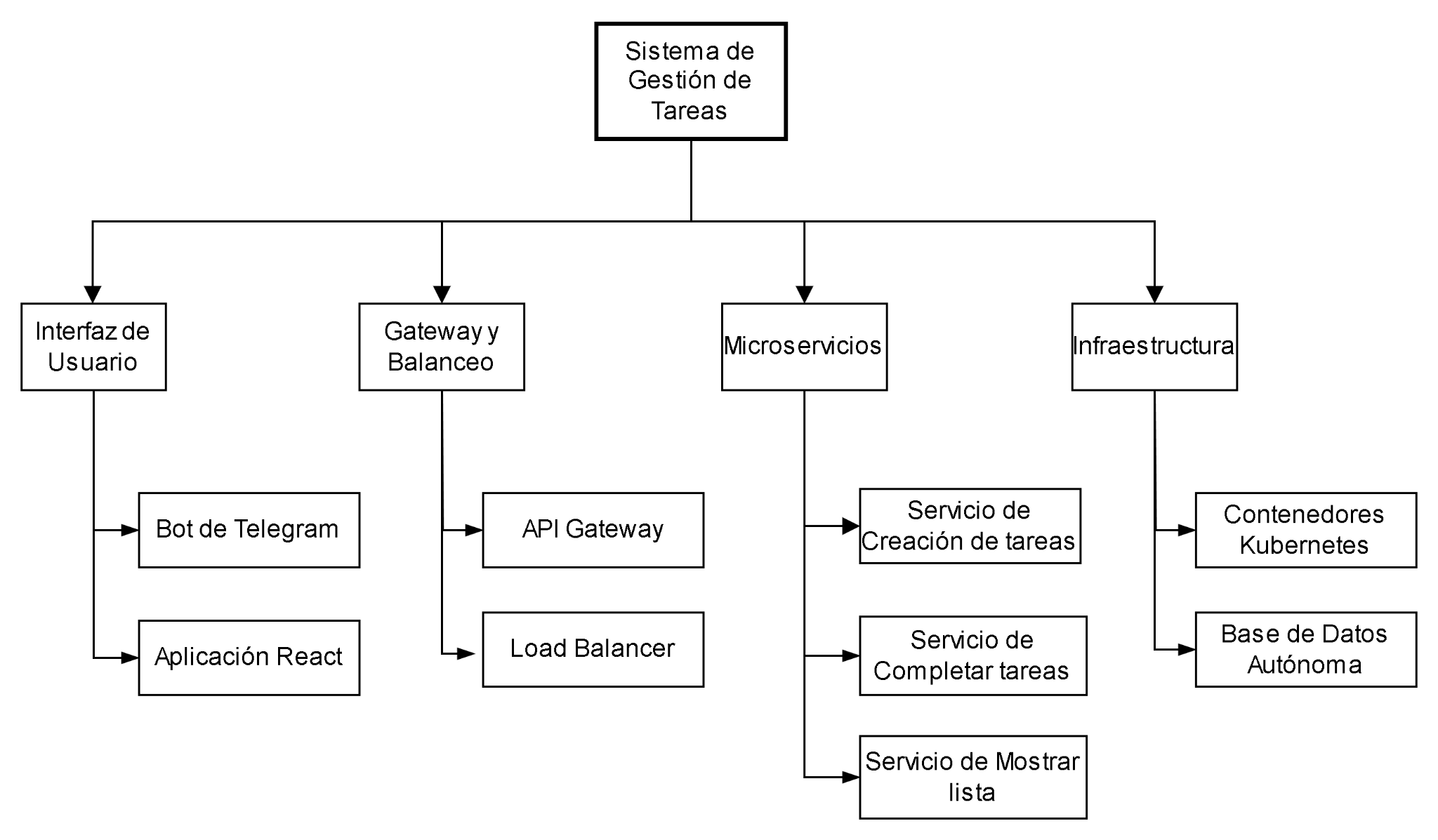
##### Diagrama de Flujo de Datos (DFD) de Alto Nivel:

##### 

##### Descripción Funcional de los Subsistemas:

1. **Usuario:**
   * Interactúa con el sistema a través de Telegram o una aplicación web React.
2. **Telegram Bot:**
   * Recibe y envía mensajes a los usuarios para interactuar con el sistema.
3. **API Gateway:**
   * Enruta las solicitudes de los usuarios a los microservicios correspondientes.
4. **Servicios de Microservicios:**
   * **Task Creation Service:** Gestiona la creación de nuevas tareas.
   * **Complete Task Service:** Administra la finalización de tareas.
   * **Show List Service:** Recupera y muestra la lista de tareas.
5. **Load Balancer:**
   * Distribuye el tráfico entre los microservicios para evitar cuellos de botella.
6. **Contenedores Kubernetes:**
   * Orquesta los contenedores de aplicaciones, asegurando su despliegue y escalabilidad.
7. **Base de Datos Autónoma:**
   * Almacena los datos de las tareas de manera persistente.

#### Diagrama de Descomposición Estructural:



## Design Rationale

La arquitectura seleccionada es una arquitectura basada en microservicios desplegada en Oracle Cloud Infrastructure, utilizando Kubernetes para la orquestación de contenedores. Esta decisión se tomó después de considerar varios factores críticos y realizar algunos compromisos.

**Factores Críticos Considerados:**

1. **Escalabilidad:**
   * La arquitectura basada en microservicios permite escalar cada componente de forma independiente según la demanda. Kubernetes facilita el escalado automático de los contenedores, asegurando que el sistema pueda manejar picos de carga eficientemente.
2. **Mantenimiento:**
   * La descomposición en microservicios hace que cada componente sea más fácil de mantener y actualizar sin afectar al sistema completo. Esto mejora la capacidad de realizar cambios y mejoras continuas.
3. **Resiliencia:**
   * La separación de funciones en diferentes microservicios reduce el riesgo de fallos sistémicos. Si un servicio falla, otros pueden continuar funcionando, aumentando la resiliencia general del sistema.
4. **Flexibilidad:**
   * La arquitectura modular permite la integración de nuevas funciones y servicios con facilidad, adaptándose a las necesidades cambiantes del negocio.

##### Compromisos Realizados:

1. **Complejidad Operacional:**
   * La gestión de múltiples microservicios y su orquestación puede ser compleja. Sin embargo, esta complejidad se mitiga mediante el uso de Kubernetes, que automatiza gran parte del proceso de despliegue y escalado.
2. **Coste Inicial:**
   * La configuración inicial y el despliegue de una arquitectura basada en microservicios puede ser más costoso en términos de tiempo y recursos. No obstante, los beneficios a largo plazo en términos de mantenimiento y escalabilidad justifican esta inversión inicial.

##### Arquitecturas Alternativas Consideradas:

1. **Arquitectura Monolítica:**
   * Una arquitectura monolítica podría haber sido más sencilla de implementar inicialmente. Sin embargo, la falta de escalabilidad y la dificultad para mantener y actualizar componentes individuales la hacen menos adecuada para un sistema que se espera que crezca y evolucione con el tiempo.
2. **Arquitectura de Microservicios sin Kubernetes:**
   * Otra opción podría haber sido usar microservicios sin la capa de orquestación de Kubernetes. Aunque esto simplificaría la infraestructura inicial, la falta de capacidades de escalado automático y gestión de contenedores habría limitado la capacidad del sistema para manejar cargas variables de manera eficiente.
3. **DATA DESIGN**

## Data Description

## En esta sección se explica cómo el dominio de la información del sistema se transforma en estructuras de datos. Se describen las principales entidades de datos o del sistema y cómo se almacenan, procesan y organizan. También se enumeran las bases de datos o elementos de almacenamiento de datos utilizados.

#### Almacenamiento y Organización de Datos

## Estructura de Almacenamiento:

## Utiliza una base de datos relacional autónoma para el almacenamiento persistente de datos.

## La base de datos está optimizada para tareas de gestión y consulta de datos.

## Principales Entidades de Datos:

## Tareas: Información sobre las tareas, incluyendo detalles como el ID de la tarea, descripción, estado (completada/no completada), y la fecha de creación.

## Usuarios: Información sobre los usuarios, incluyendo el ID de usuario, nombre, y detalles de contacto.

## Historial de Tareas: Registro de las acciones realizadas sobre las tareas, como creación, actualización y eliminación.

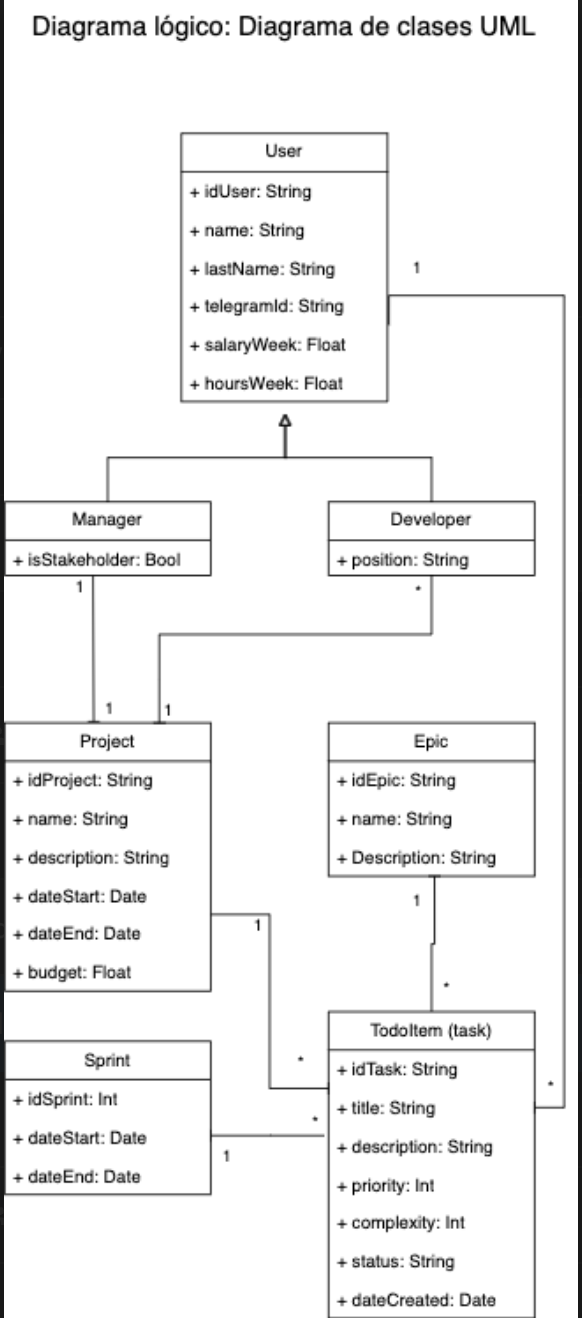
## Data Dictionary

A continuación se presenta un diccionario de datos basado en el diagrama de clases UML.

#### Entidades y Atributos

1. **User**
   * **idUser:** Tipo: String. Descripción: Identificador único para cada usuario.
   * **name:** Tipo: String. Descripción: Nombre del usuario.
   * **lastName:** Tipo: String. Descripción: Apellido del usuario.
   * **telegramId:** Tipo: String. Descripción: Identificador de usuario en Telegram.
   * **salaryWeek:** Tipo: Float. Descripción: Salario semanal del usuario.
   * **hoursWeek:** Tipo: Float. Descripción: Horas trabajadas a la semana por el usuario.
2. **Manager (hereda de User)**
   * **isStakeholder:** Tipo: Boolean. Descripción: Indica si el manager es un stakeholder del proyecto.
3. **Developer (hereda de User)**
   * **position:** Tipo: String. Descripción: Posición o rol del desarrollador en el equipo.
4. **Project**
   * **idProject:** Tipo: String. Descripción: Identificador único para cada proyecto.
   * **name:** Tipo: String. Descripción: Nombre del proyecto.
   * **description:** Tipo: String. Descripción: Descripción detallada del proyecto.
   * **dateStart:** Tipo: Date. Descripción: Fecha de inicio del proyecto.
   * **dateEnd:** Tipo: Date. Descripción: Fecha de finalización del proyecto.
   * **budget:** Tipo: Float. Descripción: Presupuesto asignado al proyecto.
5. **Epic**
   * **idEpic:** Tipo: String. Descripción: Identificador único para cada épica.
   * **name:** Tipo: String. Descripción: Nombre de la épica.
   * **description:** Tipo: String. Descripción: Descripción detallada de la épica.
6. **Sprint**
   * **idSprint:** Tipo: Int. Descripción: Identificador único para cada sprint.
   * **dateStart:** Tipo: Date. Descripción: Fecha de inicio del sprint.
   * **dateEnd:** Tipo: Date. Descripción: Fecha de finalización del sprint.
7. **TodoItem (task)**
   * **idTask:** Tipo: String. Descripción: Identificador único para cada tarea.
   * **title:** Tipo: String. Descripción: Título de la tarea.
   * **description:** Tipo: String. Descripción: Descripción detallada de la tarea.
   * **priority:** Tipo: Int. Descripción: Prioridad de la tarea.
   * **complexity:** Tipo: Int. Descripción: Complejidad de la tarea.
   * **status:** Tipo: String. Descripción: Estado actual de la tarea.
   * **dateCreated:** Tipo: Date. Descripción: Fecha de creación de la tarea.

**Diagrama**



1. **COMPONENT DESIGN**

En esta sección se examina de manera más sistemática lo que hace cada componente. Se proporcionará un resumen de los algoritmos para cada función listada anteriormente en lenguaje de descripción procedural (PDL) o pseudocódigo.

**Servicio de Creación de Tareas:**

| *//Uno de los múltiples casos que puede recibir el bot es:*  *Si bot recibe comando /addTask*  *Arroja mensaje de solicitud de Título*  *Marcar estado de usuario como: Creando Task*  *//Al tener la el estado de Creando Task, todos los mensajes entrarán en el siguiente Switch Case:*  *Si usuario está Creando Task*  *Si usuario está en estado 1:*  *Asigna Titulo a DummyTask de Usuario*  *Arroja mensaje de solicitud de descripción*  *Cambia estado a 1*  *Si usuario está en estado 2:*  *Asigna Descripción a DummyTask de Usuario*  *Arroja mensaje de solicitud de Prioridad*  *Cambia estado a 2*  *Si usuario está en estado 3:*  *Asigna Complejidad a DummyTask de Usuario*  *Arroja mensaje de solicitud de Prioridad*  *Cambia estado a 4*  *Si usuario está en estado 4:*  *Asigna Prioridad a DummyTask de Usuario*  *Actualiza Task en base de datos con DummyTask*  *Reinicia estados*  *Vacía DummyTask*  *Arroja mensaje de Task Creada* |
| --- |

**Servicio de Completar Tareas:**

| *Si bot recibe Carácter de Done + id de Task*  *Bot obtiene el item de Task*  *Bot cambia el atributo done a True*  *Bot actualiza la base de datos con la copia modificada de Task* |
| --- |

**Servicio de Mostrar Lista:**

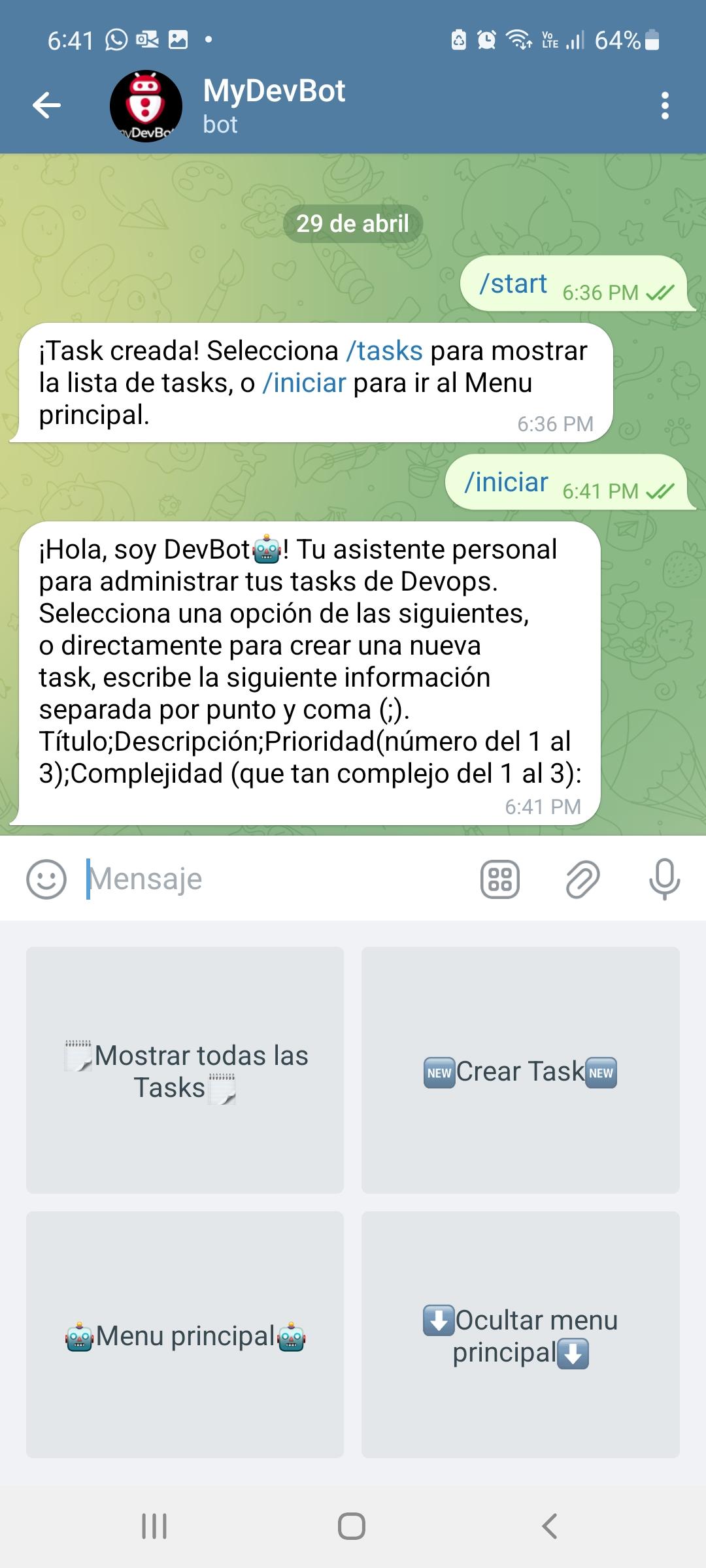
| *Si bot recibe /mostrarTasks*  *bot obtiene todas las tasks de la base de datos*  *se filtran las tasks con el id del usuario*  *se filtran las tasks en progreso*  *se coloca cada task en una keyboard row*  *se despliegan las task en progreso en la zona del teclado*  *se obtienen las tasks restantes del usuario*  *se coloca cada task en una keyboard row*  *se despliegan las task terminadas en la zona del teclado* |
| --- |

1. **HUMAN INTERFACE DESIGN**

## Overview of User Interface

El sistema permite a los usuarios gestionar tareas y proyectos de manera eficiente a través de Telegram y una aplicación web desarrollada en React. Los usuarios pueden registrarse, iniciar sesión y crear tareas utilizando comandos en Telegram o formularios en la aplicación web, recibiendo confirmaciones y mensajes de retroalimentación en ambas interfaces. Los managers pueden crear proyectos, sprints y épicas, así como asignar tareas a estos elementos desde la aplicación web, con confirmaciones visibles en tiempo real. Las listas de tareas y estados se pueden consultar tanto en Telegram como en la aplicación web, asegurando una accesibilidad rápida y una gestión visual detallada. Este diseño integral facilita una interacción fluida y efectiva para la gestión de tareas y proyectos.

## Screen Images

1. **REQUIREMENTS MATRIX**

**7.1 Modo 1 (usuario)**

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF101** | El usuario se puede comunicar con el chatbot mediante lenguaje escrito | Chat Interface |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF102** | El usuario puede ingresar sus claves de acceso para mostrar su rol asignado e interactuar con el chatbot de la manera correspondiente | Authentication Module |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF103** | Al iniciar el chatbot, el sistema mostrará una interfaz gráfica con botones en el área del teclado con diferentes opciones según su rol | Chat Interface, Role Management |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF104** | El usuario podrá ocultar y mostrar la interfaz de botones al presionar un botón | Chat Interface |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF105** | Mientras el usuario no esté en la pantalla principal de la interfaz gráfica, el chatbot debe mostrar un botón “Volver a pantalla principal” que lo devuelva al inicio | Chat Interface |

**7.2 Modo 2 (manager)**

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF301** | El manager debe poder solicitar al chatbot todas las tareas activas de todo su equipo de desarrollo de mediante el campo de texto y botón de interfaz | Team Management Module, Chat Interface |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF302** | El manager debe poder solicitar al chatbot mostrarle todas las tareas activas de un miembro específico de su equipo de desarrollo mediante el campo de texto y botón de interfaz | Team Management Module, Chat Interface |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF303** | El manager debería poder ver una lista de todas las tareas terminadas por el equipo mediante el campo de texto y botón de interfaz | Team Management Module, Chat Interface |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF304** | El manager debería poder ver una lista de todas las tareas terminadas por un miembro específico del equipo de desarrollo mediante el campo de texto y botón de interfaz | Team Management Module, Chat Interface |

**7.3 Modo 3 (desarrollador)**

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF201** | El desarrollador debe poder agregar una tarea ToDo al escribir sobre el campo de texto | ToDo Management Module |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF202** | Al presionar el botón “Agregar ToDo” dentro de la interfaz de botones, el bot le dará la indicación escrita de ingresar el ToDo mediante el campo de texto | ToDo Management Module |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF203** | Al presionar el botón “Listar ToDo’s”, el bot le debe mostrar al desarrollador una lista interactiva de todos los ToDo’s en el área de la interfaz de botones | ToDo Management Module |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF204** | El desarrollador debería poder solicitar al chatbot una lista filtrada de sus ToDo’s terminados | ToDo Management Module |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF205** | El desarrollador debería poder solicitar al chatbot cambiar la fecha de entrega de un ToDo | ToDo Management Module |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF206** | El desarrollador debería poder solicitar al chatbot cambiar la descripción de un ToDo | ToDo Management Module |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF207** | El desarrollador puede solicitar al chatbot cambiar el nombre de un ToDo | ToDo Management Module |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF208** | El desarrollador debería poder cambiar el nivel de prioridad de un ToDo | ToDo Management Module |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF209** | Al mostrar la lista de ToDo’s, el chatbot deberá mostrar la opción de marcarlo como “Completado” en los ToDo’s activos | ToDo Management Module |

| **ID** | **Requerimiento** | **Component** |
| --- | --- | --- |
| **RF210** | Al mostrar la lista de ToDo’s, el chatbot deberá mostrar las opción de “Deshacer” el “Completado” y “Eliminar” en los ToDo´s inactivos | ToDo Management Module |