Flask-MongoDB-CRUD Documentation

Juan Felipe Agudelo Sanchez
Jhonatan Stiven Albarracin Agudelo
Anderson Yamid Arciniegas Tarapues

Presentado a:
Felipe Gutiérrez Isaza
fgutierrez@utp.edu.co

Sistemas Distribuidos
Ingeniería de Sistemas y Computación
Universidad Tecnológica de Pereira

2024

Contents

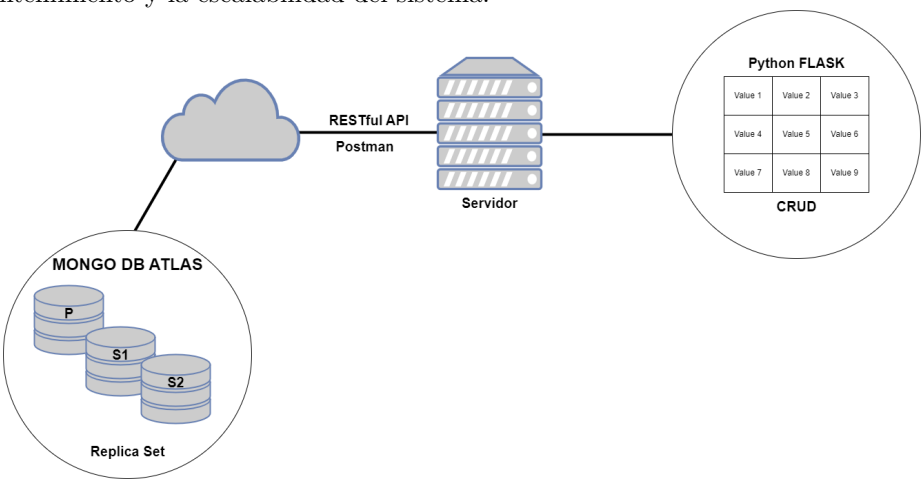
1 Marco Teórico

1.1 Introducción

El proyecto es una plataforma de gestión de datos distribuidos desarrollada como parte de la asignatura de sistemas distribuidos. Utiliza tecnologías modernas como Flask para el desarrollo del backend y MongoDB Atlas como la base de datos en la nube. Esta plataforma está diseñada para ofrecer operaciones CRUD (Crear, Leer, Actualizar, Eliminar) de manera eficiente a través de una API RESTful, permitiendo a los usuarios gestionar datos de forma escalable y altamente disponible.

1.2 Arquitectura

El backend está estructurado según un patrón de arquitectura cliente/servidor, utilizando Python y el framework Flask. Esta arquitectura permite una separación clara entre la lógica de presentación y la lógica de negocio, facilitando el mantenimiento y la escalabilidad del sistema.



1.3 Configuración de la base de datos

La base de datos del proyecto se encuentra en MongoDB Atlas, una solución de base de datos en la nube que ofrece escalabilidad automática y alta disponibilidad. Para garantizar la integridad y disponibilidad de los datos, así como la capacidad de escalar horizontalmente según la demanda, se ha implementado una configuración que emplea conjuntos de réplicas.

La configuración consta de un conjunto de réplicas en el que cada instancia de MongoDB Atlas actúa como una réplica de la base de datos principal. Esto asegura la disponibilidad continua de los datos y proporciona redundancia en caso de fallo de alguno de los servidores.

Además, se realiza una replicación de datos entre las instancias del conjunto de réplicas para mantener copias exactas de los datos en cada una de ellas. De esta manera, se garantiza la coherencia de los datos y se minimiza el riesgo de pérdida de información en caso de fallo de algún servidor.

Esta configuración redundante y escalable asegura la integridad y la disponibilidad de los datos en todo momento, proporcionando un entorno confiable para la gestión de datos distribuidos en la nube.

1.4 Características principales

- **Gestión de datos distribuidos**: Este proyecto proporciona una plataforma para gestionar datos distribuidos en una red de servidores en la nube.
- **Operaciones CRUD**: La plataforma permite realizar operaciones de Crear, Leer, Actualizar y Eliminar datos de manera eficiente a través de una API RESTful.
- **Escalabilidad**: Este proyecto está diseñado para escalar horizontalmente, lo que permite manejar grandes volúmenes de datos y aumentar la capacidad de forma dinámica según las necesidades del sistema.
- **Alta disponibilidad**: Utilizando MongoDB Atlas como base de datos en la nube, este proyecto garantiza la disponibilidad y la redundancia de los datos para minimizar el tiempo de inactividad y garantizar la integridad de los datos.

1.5 Tecnologías Utilizadas

Para el desarrollo del proyecto, se han seleccionado las siguientes tecnologías:

- **Flask**: Como framework ligero y flexible para el desarrollo del backend de la aplicación, Flask proporciona una base sólida para implementar la API RESTful del proyecto de manera eficiente. Flask es un "micro" framework escrito en Python concebido para facilitar el desarrollo de aplicaciones bajo el patrón M.V.C.

  Flask como micro framework se refiere a la existencia de herramientas para crear una aplicación funcional con pocas lineas de codigo pero al encontrarse con una aplicación la cual presenta oportunidades de escalabilidad, esta debe hacer llamado a extensiones o plugins que se pueden instalar con Flask para permitirle manejar este tipo de escalabilidad.

1.5.1 ¿Por qué se selecciona Flask para el desarrollo?

- **Micro framework**: permitiendo la creación de un app funciona de manera ágil y rápida

- **Servicio web de desarrollo integrado**: No requiere de infraestructura con servidor web para probar las aplicaciones, simplemente se corre un servidor en paralelo y se compara en tiempo real los cambios sobre el código que se escribe
- **Depurador y soporte integrado para pruebas unitarias**: Si existen errores en el código que se construye, Flask permite la depuración de cada error y se puede visualizar los valores que obtienen las variables. De igual manera permitiendo las pruebas unitarias.
- **Sirve para construir servicios web (APIs REST) o aplicaciones de contenido estatico**

1.5.2 Estructura de aplicacion minima funcional en Flask

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

1.5.3 ¿Cómo funciona Flask?

1. **Primero se realiza una importación de la clase Flask. Una aplicación WSGI será la instancia de la clase.**
2. **Se crea una instancia de la clase. El primer argumento es el nombre del módulo o paquete. `__name__` es siempre la elección debido a que funciona como un atajo conveniente en la mayoría de los casos. Esto es requerido ya que de esta manera Flask sabe donde buscar los recursos ya sean archivos estáticos o plantillas.**
3. **Se utiliza el decorador `route()` para que así Flask entienda qué URL debe utilizar la función.**
4. **La función retorna el mensaje que queremos mostrar en el navegador del usuario. El contenido por defecto es HTML, así que se retorna una variable tipo String renderizada por el navegador.**

- **MongoDB Atlas**: Como base de datos en la nube, MongoDB Atlas ofrece una solución escalable y altamente disponible para almacenar y gestionar los datos distribuidos del proyecto.

- **RESTful API**: La API del proyecto sigue los principios de REST (Representational State Transfer), lo que permite una comunicación eficiente entre clientes y servidores a través de operaciones estándar como GET, POST, PUT y DELETE. Esto facilita la interoperabilidad y la escalabilidad del sistema.

- **MongoDB Replica Sets**: MongoDB replica sets es una herramienta fundamental en la protección de registros. Mediante comandos se hace una copia exacta de los conjuntos de datos y se distribuye entre varios nodos de diferentes servidores. En la estructura Replica sets se manejan dos tipos de nodos:

  - **Primario**: El nodo primario se le conoce como nodo maestro. Es el punto de partida en donde se crean los MongoDB Replica Sets. Tiene permisos de lectura y escritura.
  - **Secundario**: Los nodos secundarios son copias exactas del nodo maestro. Su uso se limita a almacenar los datos copiados y no tienen permisos de escritura ni de lectura.
    - **Un nodo secundario asume el papel de maestro sólo cuando falla el nodo primario.**
    - **Se recomienda crear un número impar de nodos secundarios.**

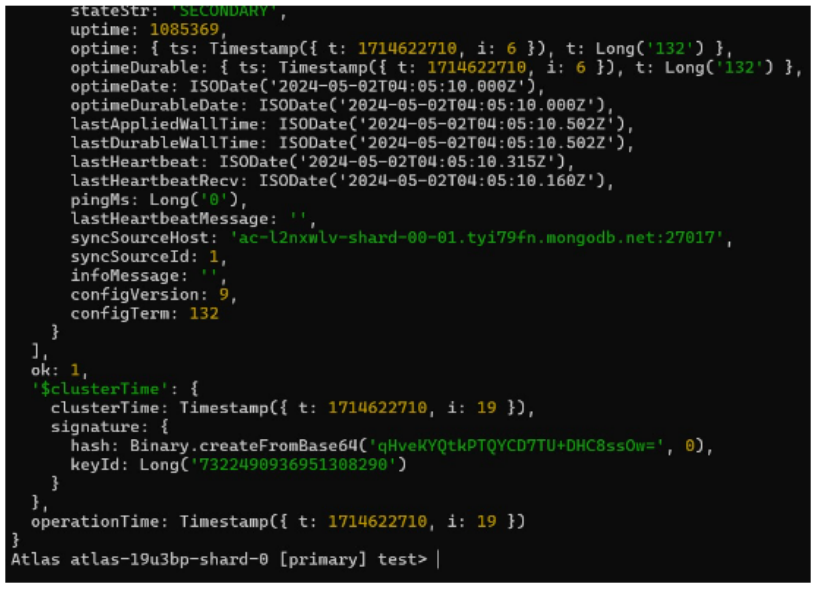
1.5.4 ¿Cómo funciona el replica set?

- **Mongodb replicaset consiste en un nodo primario y varios secundarios que se crean a partir de copias exactas del maestro. Se recomienda un mínimo de tres copias. El flujo de transmisión de datos es única y exclusivamente desde el nodo primario a los nodos secundarios.**
- **Si el nodo primario falla debido a un error, fallo de sistema o debido a trabajos de mantenimiento necesarios, un nodo secundario pasa a ocupar la posición del maestro inmediatamente y garantiza que los datos estén disponibles durante el fallo. Si el nodo defectuoso se puede reparar, este se puede reincorporar a la red como nodo secundario, a su vez por medio del comando rs.status() podemos observar cuales son los nodos que se encuentran al momento de iniciar la conexion con mongosDB Atlas**

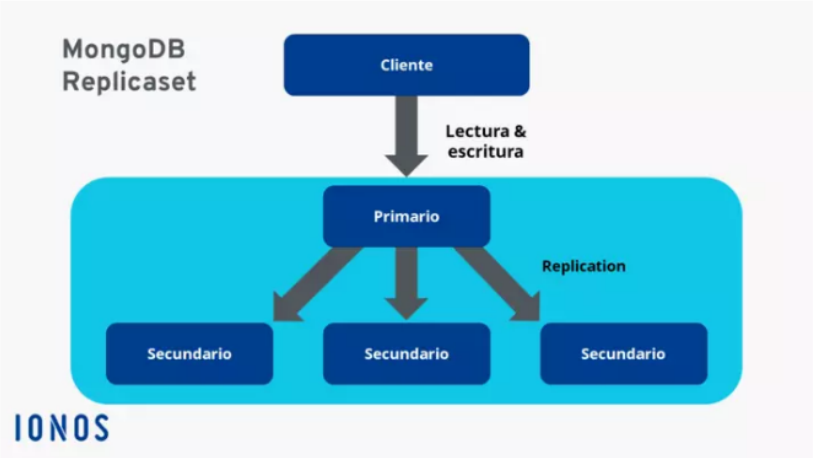
```
C:\Users\Anderson>mongosh "mongodb+srv://greatstackdev:12345@cluster0.tyi79fn.mongodb.net"
Current Mongosh Log ID: 663310e7576648e55646b798
Connecting to:          mongodb+srv://<credentials>@cluster0.tyi79fn.mongodb.net/?appName=mongo
Using MongoDB:          7.0.8
Using Mongosh:          2.2.5

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

Atlas atlas-19u3bp-shard-0 [primary] test> rs.status()
{
  set: 'atlas-19u3bp-shard-0',
  date: ISODate('2024-05-02T04:05:10.508Z'),
  myState: 1,
  term: Long('132'),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1714622710, i: 19 }), t: Long('132') },
    lastCommittedWallTime: ISODate('2024-05-02T04:05:10.502Z'),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1714622710, i: 19 }), t: Long('132') },
    appliedOpTime: { ts: Timestamp({ t: 1714622710, i: 19 }), t: Long('132') },
    durableOpTime: { ts: Timestamp({ t: 1714622710, i: 19 }), t: Long('132') },
    lastAppliedWallTime: ISODate('2024-05-02T04:05:10.502Z'),
    lastDurableWallTime: ISODate('2024-05-02T04:05:10.502Z')
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1714622706, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'stepUpRequestSkipDryRun',
    lastElectionDate: ISODate('2024-04-19T14:28:36.747Z'),
    electionTerm: Long('132'),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1713536916, i: 9 }), t: Long('131') },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1713536916, i: 9 }), t: Long('131') },
    numVotesNeeded: 2,
    priorityAtElection: 7,
```

```
    priorityAtElection: 7,
    electionTimeoutMillis: Long('5000'),
    priorPrimaryMemberId: 2,
    numCatchUpOps: Long('0'),
    newTermStartDate: ISODate('2024-04-19T14:28:36.812Z'),
    wMajorityWriteAvailabilityDate: ISODate('2024-04-19T14:28:36.927Z')
  },
  members: [
    {
      _id: 0,
      name: 'ac-l2nxwlv-shard-00-00.tyi79fn.mongodb.net:27017',
      health: 1,
      state: 2,
      stateStr: 'SECONDARY',
      uptime: 1085929,
      optime: { ts: Timestamp({ t: 1714622709, i: 23 }), t: Long('132') },
      optimeDurable: { ts: Timestamp({ t: 1714622709, i: 23 }), t: Long('132') },
      optimeDate: ISODate('2024-05-02T04:05:09.000Z'),
      optimeDurableDate: ISODate('2024-05-02T04:05:09.000Z'),
      lastAppliedWallTime: ISODate('2024-05-02T04:05:10.502Z'),
      lastDurableWallTime: ISODate('2024-05-02T04:05:10.502Z'),
      lastHeartbeat: ISODate('2024-05-02T04:05:09.857Z'),
      lastHeartbeatRecv: ISODate('2024-05-02T04:05:10.256Z'),
      pingMs: Long('0'),
      lastHeartbeatMessage: '',
      syncSourceHost: 'ac-l2nxwlv-shard-00-01.tyi79fn.mongodb.net:27017',
      syncSourceId: 1,
      infoMessage: '',
      configVersion: 9,
      configTerm: 132
    },
    {
      _id: 1,
      name: 'ac-l2nxwlv-shard-00-01.tyi79fn.mongodb.net:27017',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 1085971,
      optime: { ts: Timestamp({ t: 1714622710, i: 19 }), t: Long('132') },
      optimeDate: ISODate('2024-05-02T04:05:10.000Z'),
```

```
    stateStr: 'PRIMARY',
    uptime: 1085971,
    optime: { ts: Timestamp({ t: 1714622710, i: 19 }), t: Long('132') },
    optimeDate: ISODate('2024-05-02T04:05:10.000Z'),
    lastAppliedWallTime: ISODate('2024-05-02T04:05:10.502Z'),
    lastDurableWallTime: ISODate('2024-05-02T04:05:10.502Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1713536916, i: 10 }),
    electionDate: ISODate('2024-04-19T14:28:36.000Z'),
    configVersion: 9,
    configTerm: 132,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 2,
    name: 'ac-l2nxwlv-shard-00-02.tyi79fn.mongodb.net:27017',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 1085369,
    optime: { ts: Timestamp({ t: 1714622710, i: 6 }), t: Long('132') },
    optimeDurable: { ts: Timestamp({ t: 1714622710, i: 6 }), t: Long('132') },
    optimeDate: ISODate('2024-05-02T04:05:10.000Z'),
    optimeDurableDate: ISODate('2024-05-02T04:05:10.000Z'),
    lastAppliedWallTime: ISODate('2024-05-02T04:05:10.502Z'),
    lastDurableWallTime: ISODate('2024-05-02T04:05:10.502Z'),
    lastHeartbeat: ISODate('2024-05-02T04:05:10.315Z'),
    lastHeartbeatRecv: ISODate('2024-05-02T04:05:10.160Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: 'ac-l2nxwlv-shard-00-01.tyi79fn.mongodb.net:27017',
    syncSourceId: 1,
    infoMessage: '',
    configVersion: 9,
    configTerm: 132
  }
```



```
      stateStr: 'SECONDARY',
      uptime: 1085369,
      optime: { ts: Timestamp({ t: 1714622710, i: 6 }), t: Long('132') },
      optimeDurable: { ts: Timestamp({ t: 1714622710, i: 6 }), t: Long('132') },
      optimeDate: ISODate('2024-05-02T04:05:10.000Z'),
      optimeDurableDate: ISODate('2024-05-02T04:05:10.000Z'),
      lastAppliedWallTime: ISODate('2024-05-02T04:05:10.502Z'),
      lastDurableWallTime: ISODate('2024-05-02T04:05:10.502Z'),
      lastHeartbeat: ISODate('2024-05-02T04:05:10.315Z'),
      lastHeartbeatRecv: ISODate('2024-05-02T04:05:10.160Z'),
      pingMs: Long('0'),
      lastHeartbeatMessage: '',
      syncSourceHost: 'ac-l2nxwlv-shard-00-01.tyi79fn.mongodb.net:27017',
      syncSourceId: 1,
      infoMessage: '',
      configVersion: 9,
      configTerm: 132
    }
  ],
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1714622710, i: 19 }),
    signature: {
      hash: Binary.createFromBase64('qHveKYQtkPTQYCD7TU+DHC8ssOw=', 0),
      keyId: Long('7322490936951308290')
    }
  },
  operationTime: Timestamp({ t: 1714622710, i: 19 })
}
Atlas atlas-19u3bp-shard-0 [primary] test>
```



1.6 ¿Cómo crear una instancia de un replica sets en MongoDB Atlas?

– **La sintaxis básica de los comandos MongoDB -replSet funciona de la siguiente manera**

```
mongod --port "PORT" --dbpath "IHR_DB_DATA_PATH" --replSet "NOMBRE_DEL_REPLICASET"
```

– **Para crear replica sets en MongoDb, se debe realizar lo siguiente**

1. Cerrar los servidores de MongoDB activos
2. Introducir el comando

```
mongod --port 32014 --dbpath "D:\set up\mongodb\data" --replSet Ejemplo
```

3. Se realiza la conexión a dicha instancia
4. Utilizando el comando rs.initiate() para iniciar un nuevo MongoDb Replicaset
5. Luego se usa el comando rs.config(), permitiendo configurar el replicaset, mientras que el comando rs.status() permite comprobar el estado.
6. Se hace uso del comando rs.add() para añadir miembros a el MongoDB replicaset. La sintaxis de este comando sería la siguiente,

```
rs.add(HOST_NAME:PORT)
```

Dockerfile: Cuando se habla de contenedores en el ámbito tecnológico, se refiere al concepto de contenedores como en la cotidianidad pero que, en lugar de guardar objetos físicos tangibles, guarda una pila o stack (Quiere decir un conjunto de aplicaciones) que unidos entre sí nos sirve para crear una imagen y posteriormente se utiliza para un determinado propósito.

- **¿Por qué utilizar un Dockerfile?**: Un dockerfile es un archivo de texto plano que contiene una serie de instrucciones necesarias para crear una imagen que, posteriormente, se convertirá en una sola aplicación utilizada para un determinado propósito.
- **Como hacer un dockerfile**: Se muestra dockerfile mediante el cual construiremos una imagen con base sistemas operativo Ubuntu 14:04 las cuales tiene las siguientes tareas.
  - Instalacion y actualizacion de paquetes del sistema operativo.
  - Establecer variable de entorno llamada Debian-Frontend.
  - Instalación de git.

```
# Descarga la imagen de Ubuntu 14.04
FROM ubuntu:14.04

# Actualiza la imagen base de Ubuntu 14.04
RUN apt-get update

# Definir ambiente de entorno
ENV DEBIAN_FRONTEND noninteractive

# Instalar Git
RUN apt-get -qqy install git
```

FROM: indica la imagen base sobre la que se construirá la aplicación dentro del contenedor

RUN: permite ejecutar comandos en el contenedor por ejemplo, instalar paquetes o librerias(apt-get, etc.)

ENV: Establece variables de entorno para el contenedor, en este caso la variable de entorno es DEBIAN-FRONTEND, el cual nos permite instalar archivos .deb sin tener que interactuar entre ellos.

Este archivo de docker construye una imagen basada en Nginx,

Con el comando docker build se construye la imagen siguiendo cada instrucción escrita en el dockerfile. El docker daemon o servicio de docker ejecuta las instrucciones del dockerfile y nos muestra el proceso por pantalla y el resultado final será la imagen que queremos construir

2 Marco Lógico

2.1 Objetivo General

Desarrollar una plataforma de gestión de datos distribuidos que permita a los usuarios realizar operaciones CRUD de manera eficiente a través de una API RESTful, utilizando tecnologías modernas como Flask y MongoDB Atlas.

2.2 Objetivos Específicos

1. **Diseño e Implementación de la Arquitectura Cliente/Servidor**
   - Definir la estructura y la comunicación entre los componentes del sistema, utilizando un patrón de arquitectura cliente/servidor.
   - Implementar el backend utilizando Python y Flask, asegurando una separación clara entre la lógica de presentación y la lógica de negocio.
2. **Configuración y Gestión de la Base de Datos en MongoDB Atlas**
   - Crear un cluster en MongoDB Atlas denominado "Cluster()", con un usuario llamado "greatstackdev" y contraseña "12345", agregando la base de datos "mydatabase" para almacenar la tabla de ítems.
   - Establecer las colecciones necesarias en la base de datos "mydatabase" para almacenar los datos de manera no relacional.
   - Utilizar pymongo y MongoClient para generar la conexión con la base de datos MongoDB Atlas desde Flask, verificando la conexión antes de continuar con las operaciones CRUD.
3. **Desarrollo de una API RESTful**
   - Implementar endpoints en Flask para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los datos almacenados en la base de datos MongoDB Atlas.
   - Utilizar las bibliotecas jsonify y request para manejar la comunicación con los clientes y obtener respuestas JSON de manera eficiente.
   - Definir las rutas y los métodos (GET, POST, PUT, DELETE) para cada endpoint de la API, asegurando una comunicación uniforme y predecible.
4. **Verificación y Puesta en Marcha del Sistema**
   - Verificar la correcta conexión entre el backend desarrollado en Flask y la base de datos MongoDB Atlas, asegurando que todas las funciones de la API operen según lo esperado.
   - Iniciar el servidor web de desarrollo de Flask utilizando app.run(debug=True), permitiendo el acceso a la API a través de la dirección proporcionada.

2.3 Metodología

El desarrollo del proyecto se llevará a cabo siguiendo una metodología ágil, basada en iteraciones cortas y entregas incrementales. Se utilizará el marco de trabajo Scrum, con reuniones diarias de seguimiento y retrospectivas periódicas para evaluar el progreso y realizar ajustes según sea necesario. La participación activa de los stakeholders será fundamental para garantizar la alineación con los objetivos del proyecto y la satisfacción del cliente.

2.4 Cronograma

El proyecto se dividirá en varias etapas, cada una con sus propios objetivos y entregables. Se establecerán hitos importantes y se asignan plazos realistas para cada tarea. El cronograma se ajustará según las necesidades y los cambios en los requisitos del proyecto, garantizando una planificación flexible y adaptable.

3 Bibliografia

References

[1] Rodríguez, X. (2019, julio 24). Qué es DockerFile. Openwebinars.net. https://openwebinars.net/blog/que-es-dockerfile/

[2] MongoDB Replicaset: copia y respaldo de datos. (2023, February 8). IONOS Digital Guide; IONOS. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/mongodb-replicaset/

[3] (N.d.). Amazon.com. Retrieved May 2, 2024, from https://aws.amazon.com/es/what-is/restful-api/

[4] Muñoz, J. D. (2017, November 17). Qué es Flask. Openwebinars.net. https://openwebinars.net/blog/que-es-flask/

[5] ¿Qué es Flask Python? Un breve tutorial sobre este microframework. (2023, March 1). IONOS Digital Guide; IONOS. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/flask/

[6] Replication. (n.d.). Mongodb.com. Retrieved May 2, 2024, from https://www.mongodb.com/docs/manual/replication/