# Paso a Paso - Trabajo Práctico

**1. Preparar el entorno**

* Instalar **Docker Desktop** (Windows/Mac) o Docker Engine (Linux).
* Clonar o descargar el repositorio del proyecto con el stack (Neo4j, MongoDB, Redis, Python+Jupyter).
* Copiar .env.example a .env sin cambios:
* cp .env.example .env
* Levantar los contenedores:
* docker compose up --build
* Verificar acceso a:
  + JupyterLab → [http://localhost:8888](http://localhost:8888/)
  + Neo4j → [http://localhost:7474](http://localhost:7474/)
  + MongoDB → mongodb://admin:admin123@localhost:27017
  + Redis → redis://:redis123@localhost:6379

**2. Cargar los datos iniciales**

* Crear colecciones en MongoDB (usuarios, destinos, hoteles, actividades, reservas).
* Insertar los datasets de ejemplo en MongoDB.
* Crear nodos y relaciones en Neo4j (Usuario, Destino, relaciones VISITÓ, AMIGO\_DE, etc.).
* Configurar claves en Redis para:
  + Usuarios conectados.
  + Reservas temporales.

**3. Implementar consultas**

* En Neo4j: usuarios que visitaron **Bariloche**.
* En Neo4j: amigos de **Juan** que compartieron destinos.
* En Neo4j: sugerencias de destinos no visitados por usuario ni amigos.
* En Neo4j + MongoDB: hoteles en destinos recomendados.
* En MongoDB: reservas pendientes o en proceso.
* En Redis: listar usuarios conectados.
* En MongoDB: destinos con precio < $100.000.
* En MongoDB: hoteles de **Jujuy**.
* En MongoDB: cantidad de hoteles de un destino elegido.
* En MongoDB: actividades de **Ushuaia** de tipo “aventura”.
* En MongoDB: cantidad de reservas confirmadas por usuario.
* En MongoDB + Python: estadísticas → destino más visitado, hotel más barato, actividad más popular.

**4. Visualizaciones con Python**

* Crear gráficos con **matplotlib**:
  + Reservas por destino.
  + Comparación de precios de hoteles.
  + Actividades más populares.

**5. Modificación de datos**

* Incrementar en **5%** precios de actividades en **Tucumán**.
* Agregar servicio **SPA** al hotel con id=1.
* Eliminar un destino elegido.
* Eliminar un usuario elegido.
* Borrar relaciones AMIGO\_DE de un usuario.

**6. Informe final**

* Explicar cómo se integraron las 3 bases de datos.
* Justificar el modelado (qué va en Neo4j, qué en MongoDB, qué en Redis).
* Agregar capturas de consultas y gráficos.
* Redactar el informe en:
  + Un PDF **o**
  + Markdown dentro del Notebook.

**7. Entrega**

* Generar archivo **ZIP** con:
  + Notebook de JupyterLab con consultas y gráficos.
  + Carga de datos y scripts.
  + Informe (Markdown o PDF).
  + Capturas de pantalla.

# 📁 Estructura del Proyecto

📦 sistema-viajes/

│

├── 🧠 notebooks/

│ ├── 01\_carga\_datos.ipynb

│ ├── 02\_consultas\_neo4j.ipynb

│ ├── 03\_consultas\_mongo.ipynb

│ ├── 04\_consultas\_redis.ipynb

│ ├── 05\_visualizaciones.ipynb

│ ├── 06\_modificaciones.ipynb

│ └── 07\_informe\_final.ipynb

│

├── 📂 data/

│ ├── usuarios.json

│ ├── destinos.json

│ ├── hoteles.json

│ ├── actividades.json

│ └── reservas.json

│

├── 🐳 docker/

│ ├── Dockerfile # Imagen de Python con Jupyter y librerías

│ ├── requirements.txt # pymongo, neo4j, redis, matplotlib, pandas, etc.

│ ├── docker-compose.yml # Levanta Neo4j, MongoDB, Redis y JupyterLab

│ └── .env # Variables de entorno (passwords, puertos)

│

├── 📜 scripts/

│ ├── load\_mongo.py # Carga inicial de colecciones Mongo

│ ├── load\_neo4j.py # Crea nodos y relaciones en Neo4j

│ ├── load\_redis.py # Configura claves y estados en Redis

│ └── test\_connections.py # Prueba conexiones a las tres bases

│

├── 📊 reportes/

│ ├── graficos/ # Carpeta con los gráficos generados (PNG o JPG)

│ ├── capturas/ # Screenshots de consultas en Neo4j, Mongo, Redis

│ └── informe.pdf # Informe final (si no se usa Markdown)

│

├── 📘 README.md # Explicación general del proyecto

├── 📋 checklist.md # Lista de tareas paso a paso (interactiva)

└── 📦 entrega\_final.zip # Archivo que subirás (generado al final)

## 📑 Descripción de cada parte

**🧠 Notebooks**

Separar el desarrollo en notebooks temáticos ayuda a mantener claridad:

1. **01\_carga\_datos.ipynb** → carga datasets en Mongo y crea nodos/relaciones en Neo4j y Redis.
2. **02\_consultas\_neo4j.ipynb** → consultas sobre usuarios, amigos, destinos y sugerencias.
3. **03\_consultas\_mongo.ipynb** → consultas de reservas, hoteles, precios, actividades.
4. **04\_consultas\_redis.ipynb** → manejo de usuarios conectados y reservas temporales.
5. **05\_visualizaciones.ipynb** → generación de gráficos con Python.
6. **06\_modificaciones.ipynb** → operaciones de actualización, inserción y eliminación.
7. **07\_informe\_final.ipynb** → redacción del informe con Markdown y conclusiones.

**📂 Data**

Archivos .json con los datasets base (usuarios, destinos, hoteles, actividades, reservas).  
Cada uno se importa en MongoDB desde Python o directamente con insert\_many().

**🐳 Docker**

Contiene todo para levantar el entorno completo:

* docker-compose.yml define los servicios Neo4j, Mongo, Redis y Jupyter.
* Dockerfile crea la imagen personalizada de Python.
* .env contiene usuarios y contraseñas (ejemplo: neo4j123, admin123, redis123).

**📜 Scripts**

Scripts Python independientes por si necesitás cargar datos o probar conexiones sin Jupyter:

* load\_mongo.py, load\_neo4j.py, load\_redis.py para inicializar las bases.
* test\_connections.py para verificar conectividad y credenciales.

**📊 Reportes**

Carpeta para guardar las salidas del trabajo:

* **/graficos/**: exportar gráficos generados (ejemplo: destinos\_mas\_visitados.png).
* **/capturas/**: incluir capturas de pantallas de las consultas y resultados.
* **informe.pdf** (opcional): versión final del informe si no usás Markdown.

**📋 Checklist**

Archivo checklist.md (en formato Markdown) con todas las tareas, así podés marcar con [x] lo completado.

**💡 Consejo**

Cuando termines todo, generá el ZIP así:

zip -r entrega\_final.zip sistema-viajes/

Y verificá que dentro estén:

* Los notebooks.
* Los datasets.
* El docker-compose y scripts.
* El informe o notebook con conclusiones.
* Las capturas y gráficos.

# Explicación de los archivos de la estructura

**🗂️ 1. Fuente de datos inicial**

* **Carpeta:** datos/\*.csv  
  Estos son los archivos base con la información cruda (por ejemplo, productos, ventas, usuarios, etc.).
* **Rol:** Son el punto de partida. Se leen desde los notebooks o scripts para cargar la información a las bases de datos.

**📘 2. Notebook de carga de datos**

* **Archivo:** notebooks/01\_carga\_datos.ipynb
  + Se encarga de **leer los CSV** y **ejecutar los scripts de carga** para alimentar las bases de datos.
  + Llama a:
    - scripts/load\_mongo.py → Carga datos en **MongoDB**.
    - scripts/load\_redis.py o scripts/load\_noi.py → Cargan estructuras auxiliares o cache en **Redis**.
  + También puede comunicarse con **Neo4j** si es necesario generar relaciones entre los datos.

**🧠 3. Scripts de carga**

* **Carpeta:** scripts/
  + load\_mongo.py → Inserta documentos o colecciones en **MongoDB**.
  + load\_redis.py → Inserta claves/valores o listas en **Redis**.
  + load\_noi.py → Puede preparar datos o transformarlos antes de la carga.

📤 Estos scripts son ejecutados por los notebooks de carga o directamente desde consola.

**🗄️ 4. Bases de datos**

* **MongoDB:** almacena datos estructurados y semiestructurados.
* **Neo4j:** guarda relaciones entre entidades (por ejemplo, “usuario compra producto”).
* **Redis:** sirve como caché o para acelerar consultas temporales.

💬 Interacciones:

* Los scripts escriben en estas bases.
* Los notebooks de análisis **leen** y **consultan** de ellas.

**📊 5. Notebooks de consultas**

* **Carpetas:**
  + notebooks/02\_consultas.ipynb
  + notebooks/03\_consultas.ipynb
* **Rol:**  
  Ejecutan **consultas** sobre MongoDB, Redis o Neo4j para analizar la información.  
  Pueden combinar resultados entre bases o transformarlos para análisis estadístico.

**🧾 6. Notebook de modificaciones**

* **Archivo:** notebooks/05\_modificaciones.ipynb
  + Permite **editar o actualizar datos** directamente en las bases (por ejemplo, corregir valores o agregar campos).
  + Usa las mismas conexiones que los notebooks de consulta, pero ejecuta comandos update o delete.

**📑 7. Generación de reporte**

* **Archivo:** reporte/informe.pdf
  + Recibe la **salida procesada** desde los notebooks de consulta.
  + Puede generarse con librerías como ReportLab o pandas-profiling.
  + Es la etapa final donde se compilan los resultados analíticos.

# Pseudocódigo del Flujo del Proyecto

INICIO DEL SISTEMA DE GESTIÓN DE VIAJES

1️⃣ CARGA DE DATOS INICIALES

----------------------------------

→ Leer archivos CSV o JSON desde /data/

- usuarios.json

- destinos.json

- hoteles.json

- actividades.json

- reservas.json

→ Ejecutar notebook: 01\_carga\_datos.ipynb

↳ Llama al script load\_mongo.py

→ Conecta a MongoDB

→ Crea colecciones: usuarios, destinos, hoteles, actividades, reservas

→ Inserta los datos

↳ Llama al script load\_neo4j.py

→ Conecta a Neo4j

→ Crea nodos: Usuario, Destino, Actividad, Hotel

→ Crea relaciones: (Usuario)-[RESERVA]->(Destino), (Destino)-[TIENE]->(Hotel)

↳ Llama al script load\_redis.py

→ Conecta a Redis

→ Crea claves temporales de usuarios activos o reservas en curso

Resultado → Las tres bases quedan inicializadas y sincronizadas.

----------------------------------

2️⃣ VALIDACIÓN DE CONEXIONES

----------------------------------

→ Ejecutar script test\_connections.py

↳ Prueba conexión con MongoDB, Neo4j y Redis

↳ Devuelve “OK” o error según respuesta

Si todo es correcto → continuar.

----------------------------------

3️⃣ CONSULTAS Y ANÁLISIS

----------------------------------

→ Ejecutar notebook: 02\_consultas\_neo4j.ipynb

↳ Ejecuta consultas Cypher (por ejemplo: destinos más visitados, redes de usuarios)

→ Ejecutar notebook: 03\_consultas\_mongo.ipynb

↳ Consultas con PyMongo (hoteles más reservados, precios promedio)

→ Ejecutar notebook: 04\_consultas\_redis.ipynb

↳ Consultas rápidas en Redis (usuarios conectados, reservas activas)

Resultado → DataFrame o resultados parciales para análisis.

----------------------------------

4️⃣ VISUALIZACIONES

----------------------------------

→ Ejecutar notebook: 05\_visualizaciones.ipynb

↳ Usa pandas + matplotlib / seaborn

↳ Genera gráficos: destinos más visitados, actividad por usuario, etc.

↳ Guarda resultados en /reportes/graficos/

----------------------------------

5️⃣ MODIFICACIONES Y ACTUALIZACIONES

----------------------------------

→ Ejecutar notebook: 06\_modificaciones.ipynb

↳ Actualiza o elimina datos:

- Cambios en reservas

- Nuevos destinos

- Actualización de precios

↳ Sincroniza cambios entre MongoDB y Neo4j

----------------------------------

6️⃣ INFORME FINAL

----------------------------------

→ Ejecutar notebook: 07\_informe\_final.ipynb

↳ Resume resultados, gráficos y conclusiones

↳ Exporta informe.pdf a /reportes/

↳ Opcionalmente genera resumen Markdown o HTML

----------------------------------

7️⃣ ENTREGA FINAL

----------------------------------

→ Verificar checklist.md (todas las tareas completas)

→ Comprimir carpeta del proyecto:

zip -r entrega\_final.zip sistema-viajes/

FIN DEL PROCESO

## 💡 Relación entre componentes

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo** | **Archivo / Carpeta** | **Interactúa con** | **Función** |
| 📁 Datos | /data/\*.json | Notebooks de carga | Fuente inicial de información |
| 🧠 Notebook | 01\_carga\_datos.ipynb | Scripts de carga | Control del flujo inicial |
| 🧾 Script | load\_mongo.py, load\_neo4j.py, load\_redis.py | Bases de datos | Inserta información |
| 🧩 Base de datos | MongoDB, Neo4j, Redis | Notebooks de consulta | Almacenan los datos |
| 📈 Notebook | 02-05\_consultas\_\*.ipynb | Bases de datos | Análisis y visualización |
| ✏️ Notebook | 06\_modificaciones.ipynb | Bases de datos | Actualiza o elimina datos |
| 📊 Reporte | 07\_informe\_final.ipynb → informe.pdf | Todos los notebooks | Documento de entrega final |

# Modelados

**🧩 1. Modelo Conceptual (Entidad–Relación)**

Este es el nivel **más abstracto**: muestra *qué entidades existen* en el sistema y *cómo se relacionan*.  
No se ve ninguna base de datos todavía, solo el dominio del problema.

**Entidades principales:**

* **Usuario**
  + Atributos: usuario\_id, nombre, email, teléfono
* **Destino**
  + Atributos: destino\_id, ciudad, país, tipo, precio\_promedio
* **Hotel**
  + Atributos: hotel\_id, nombre, ciudad, precio, calificación, servicios
* **Actividad**
  + Atributos: actividad\_id, nombre, tipo, ciudad, precio
* **Reserva**
  + Atributos: reserva\_id, fecha\_reserva, estado, precio\_total

**Relaciones:**

* **Usuario —[REALIZA]—> Reserva**
  + 1 Usuario puede tener muchas Reservas.
* **Reserva —[PERTENECE\_A]—> Destino**
  + Cada Reserva corresponde a un Destino.
* **Destino —[TIENE]—> Hotel**
  + Un Destino puede tener varios Hoteles.
* **Destino —[OFRECE]—> Actividad**
  + Un Destino puede ofrecer varias Actividades.
* **Usuario —[VISITÓ]—> Destino**
* **Usuario —[AMIGO\_DE/FAMILIAR\_DE]—> Usuario**

**📘 Representación en texto (simplificada)**

[Usuario]───(REALIZA)───[Reserva]───(PERTENECE\_A)───[Destino]

│ │

(VISITÓ) (TIENE)

│ │

[Destino]────────(OFRECE)────────[Actividad]

│

(TIENE)

│

[Hotel]

Relaciones sociales:

[Usuario]──(AMIGO\_DE / FAMILIAR\_DE)──[Usuario]

**🧱 2. Modelo Lógico (Distribución entre bases de datos)**

Aquí se define *dónde* y *cómo* se almacenan las entidades según la naturaleza de cada base.

| **Componente** | **Base de datos** | **Motivo / Uso** |
| --- | --- | --- |
| **Usuarios, Destinos, Hoteles, Actividades, Reservas** | 🟢 **MongoDB** | Ideal para almacenar documentos JSON estructurados, catálogos, precios y reservas. |
| **Relaciones entre Usuarios y Destinos (VISITÓ)** **Relaciones entre Usuarios (AMIGO\_DE, FAMILIAR\_DE)** | 🔵 **Neo4j** | Ideal para representar grafos sociales y recomendaciones. |
| **Usuarios conectados** **Reservas temporales (aún no confirmadas)** **Caché de búsquedas recientes** | 🔴 **Redis** | Base en memoria, rápida para almacenar información volátil. |

**💾 Representación visual (simplificada)**

┌───────────────┐

│ MongoDB │

│---------------│

│ usuarios │

│ destinos │

│ hoteles │

│ actividades │

│ reservas │

└──────┬────────┘

│

│ (conecta a)

▼

┌───────────────┐

│ Neo4j │

│---------------│

│ VISITÓ │

│ AMIGO\_DE │

│ FAMILIAR\_DE │

└──────┬────────┘

│

│ (usa para recomendación)

▼

┌───────────────┐

│ Redis │

│---------------│

│ sesiones │

│ reservas\_temp │

│ búsquedas │

└───────────────┘

**⚙️ 3. Nivel Técnico: Interacción entre Archivos y Programas**

Este nivel conecta los notebooks y contenedores Docker que ejecutan cada parte del sistema.

**📁 Estructura general**

proyecto\_viajes/

│

├── notebooks/

│ ├── main.ipynb # Notebook principal (consultas, gráficos y conexión a bases)

│ └── test\_connections.ipynb # Prueba de conexión a MongoDB, Neo4j y Redis

│

├── python/

│ ├── Dockerfile # Imagen de Python + JupyterLab

│ └── requirements.txt # Librerías: neo4j, pymongo, redis, matplotlib, etc.

│

├── docker-compose.yml # Orquesta los 4 servicios (Jupyter, Neo4j, Mongo, Redis)

├── .env # Credenciales (NEO4J\_PASSWORD, MONGO\_INITDB\_ROOT\_USER, etc.)

├── datasets/ # Archivos JSON o CSV de carga inicial

│ ├── usuarios.json

│ ├── destinos.json

│ ├── hoteles.json

│ └── reservas.json

└── informe.md / pdf # Explicación del sistema, consultas y gráficos

**🔄 Flujo de interacción entre archivos y servicios**

1. **Docker Compose** levanta los contenedores de:
   * Python (JupyterLab)
   * MongoDB
   * Neo4j
   * Redis
2. **JupyterLab (main.ipynb)** ejecuta el código Python que:
   * Carga los datos iniciales en MongoDB.
   * Crea nodos y relaciones en Neo4j.
   * Guarda usuarios conectados y reservas temporales en Redis.
3. **Consultas y visualizaciones** se ejecutan desde JupyterLab usando:
   * pymongo → consultas a Mongo.
   * neo4j → consultas Cypher.
   * redis → acceso rápido a sesiones o datos temporales.
4. **Gráficos** con matplotlib muestran estadísticas (ej. destinos más visitados).
5. **Informe final** combina todo en Markdown o PDF.

