Guia Practica clase 11 - Cohorte 2025 - Programación lineal entera

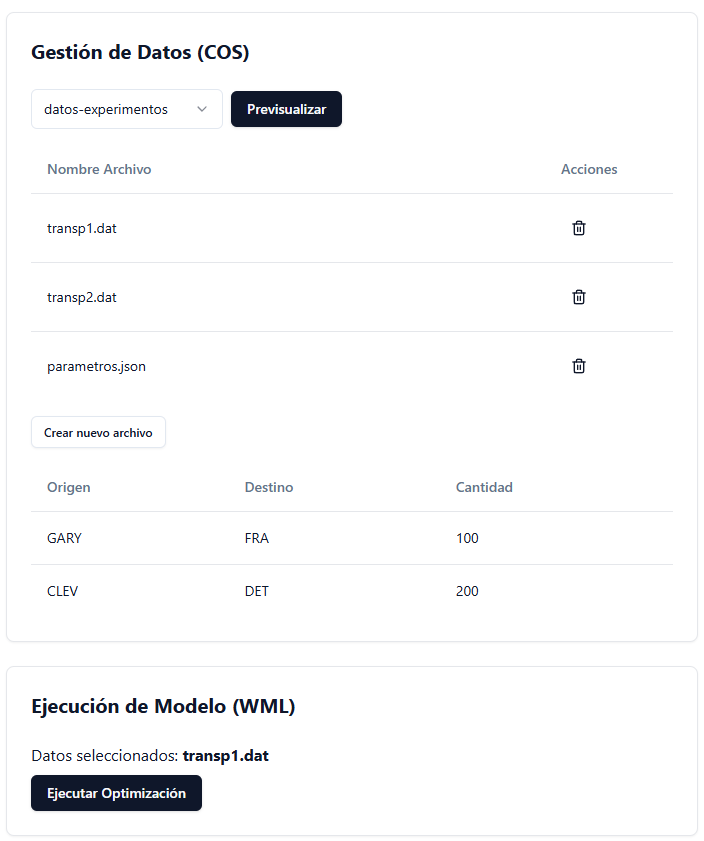
# 1. Analice y documente el código.

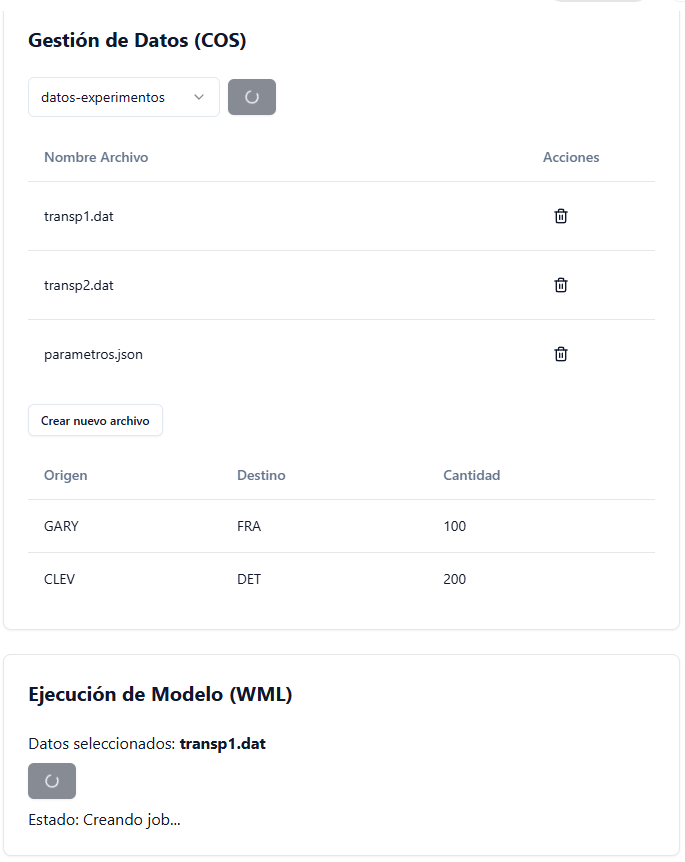
| // -------------------------------------------------------------------------- // Licensed Materials - Property of IBM // 5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 // Copyright IBM Corporation 1998, 2022. All Rights Reserved. // --------------------------------------------------------------------------  // Conjunto de ciudades {string} Cities = ...;  // Conjunto de productos a transportar {string} Products = ...;  // Capacidad máxima de envío entre cada par (o,d) float Capacity = ...;  // Oferta disponible de cada producto en cada ciudad origen float Supply[Products][Cities] = ...;  // Demanda requerida de cada producto en cada ciudad destino float Demand[Products][Cities] = ...;  // Verificación: para cada producto, oferta total == demanda total. De lo contrario, el problema no sería balanceado. assert  forall(p in Products)  sum(o in Cities) Supply[p][o]  == sum(d in Cities) Demand[p][d];  // Costo unitario de transportar producto p desde o hasta d float Cost[Products][Cities][Cities] = ...;  // Variable de decisión: cantidad transportada de p desde o hasta d (>= 0) dvar float+ Trans[Products][Cities][Cities];  // ------------------------- Función Objetivo ------------------------- minimize  // Minimizar el costo total de transporte sobre todos p, o, d  sum(p in Products, o in Cities, d in Cities)  Cost[p][o][d] \* Trans[p][o][d];  // --------------------------- Restricciones --------------------------- subject to {   // 1) Satisfacer la oferta en cada origen: lo enviado == disponible  forall(p in Products, o in Cities)  ctSupply:  sum(d in Cities)  Trans[p][o][d]  == Supply[p][o];   // 2) Satisfacer la demanda en cada destino: lo recibido == requerido  forall(p in Products, d in Cities)  ctDemand:  sum(o in Cities)  Trans[p][o][d]  == Demand[p][d];   // 3) No superar capacidad logística entre cada par (origen→destino)  forall(o in Cities, d in Cities)  ctCapacity:  sum(p in Products)  Trans[p][o][d]  <= Capacity; }  // --------------------- Generación de Salida ---------------------  // Tipo de tupla para resultados no nulos tuple solutionT {  string Products; // nombre del producto  string City1; // ciudad origen  string City2; // ciudad destino  float Trans; // cantidad transportada }  // Conjunto con solo los transportes donde Trans != 0 {solutionT} solution = {  < p, c1, c2, Trans[p][c1][c2] >  | p in Products, c1 in Cities, c2 in Cities  : Trans[p][c1][c2] != 0 };  // Mostrar en consola la matriz completa y la lista de transportes activos execute DISPLAY {  writeln("trans = ", Trans);  writeln("solution = ", solution); } |
| --- |

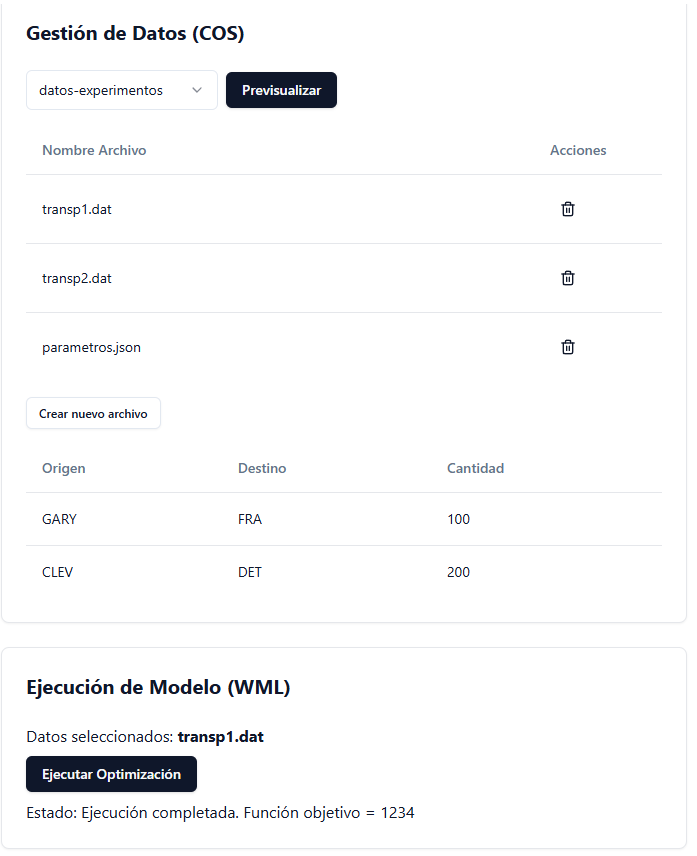
# 2. Realice las transformaciones necesarias para ejecutar el proyecto en ibm watson studio.

Utilizamos a modo de ejemplo para la implementación la versión 4 del proyecto de ejemplo.

# 3. Diseñe, a modo prototipo, un sistema cuyo principal objetivo es ejecutar el modelo con diferentes juego de datos. La aplicación debe utilizar la API de IBM Cloud Object Storage (COS API), la API de IBM Watson Machine Learning. Con COS API se accede a los datos que usa el modelo para modificar, eliminar , crear, entre otras cosas. Mientras que con la API de IBM watson machine learning , nos permite acceder al espacio de despliegue y crear y ejecutar trabajos que utilicen los datos de COS.







**Gestión de datos (COS)**

* Selección de bucket y listado de archivos
* Previsualización de contenidos
* Creación y eliminación de objetos

**Ejecución del modelo (WML)**

* Selección de los datos
* Botón para ejecutar el job
* Monitorización básica del estado y resultados

# 4. Diseñe las distintas interacciones de la aplicación teniendo en cuenta que cliente llama a funciones en el servidor, y que estas funciones en el servidor tienen por objeto llamar a funcionalidades de las API o endpoints.

**Diseñando interacciones de API**

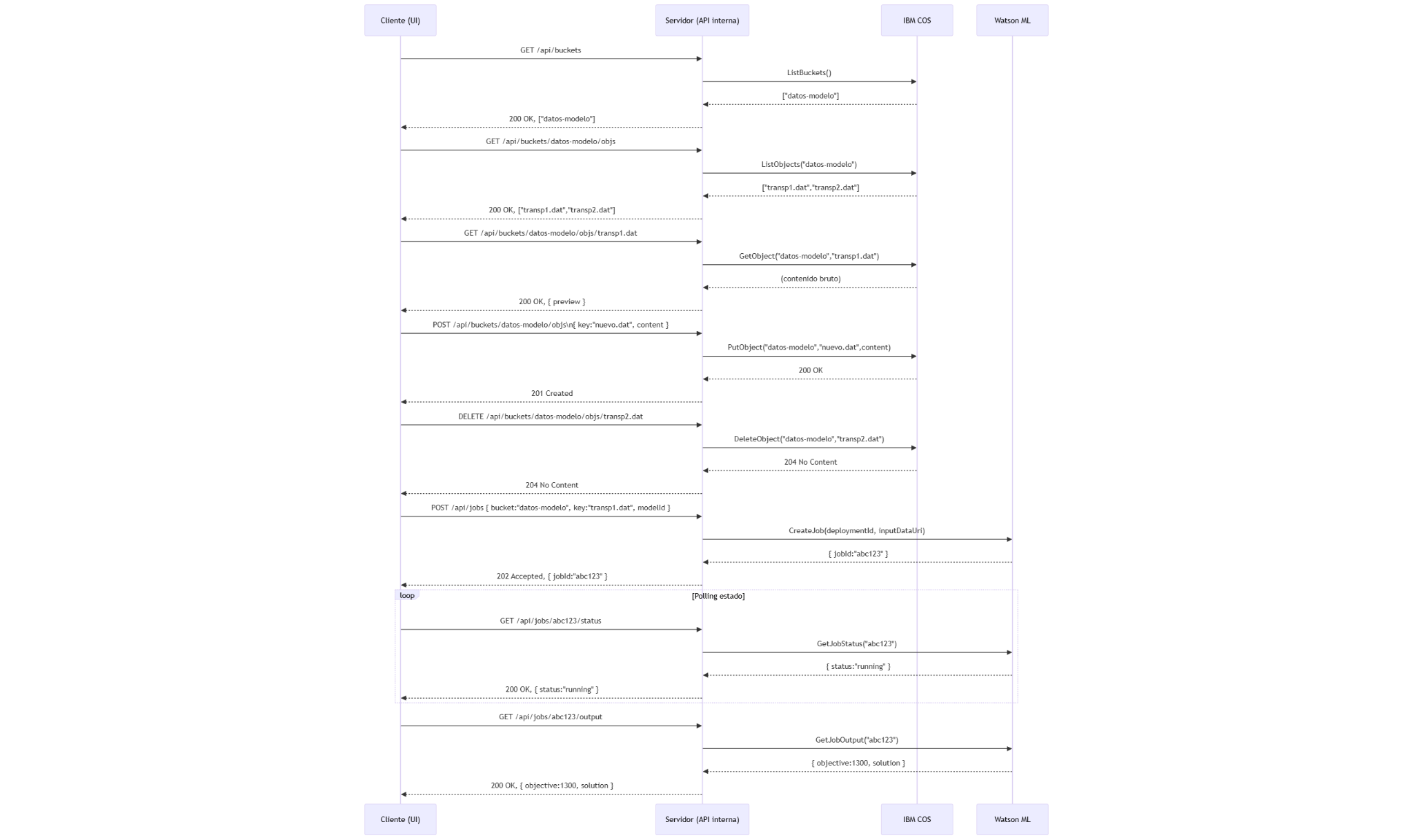
**1. Gestión de buckets y objetos (COS)**

| **N°** | **Cliente → Servidor** | **Servidor → IBM COS API** | **Descripción** |
| --- | --- | --- | --- |
| 1.1 | **GET** /api/buckets | ListBuckets() | Obtiene la lista de buckets disponibles. |
| 1.2 | **GET** /api/buckets/{bucket}/objs | ListObjects(bucket) | Lista nombres de archivos en el bucket. |
| 1.3 | **GET** /api/buckets/{bucket}/objs/{key} | GetObject(bucket,key) | Descarga el contenido real del archivo para vista previa. |
| 1.4 | **POST** /api/buckets/{bucket}/objsBody: { key, content } | PutObject(bucket,key,content) | Crea o reemplaza un archivo en el bucket. |
| 1.5 | **DELETE** /api/buckets/{bucket}/objs/{key} | DeleteObject(bucket,key) | Elimina un archivo del bucket. |

**2. Ejecución de optimización (WML)**

| **N°** | **Cliente → Servidor** | **Servidor → IBM WML API** | **Descripción** |
| --- | --- | --- | --- |
| 2.1 | **POST** /api/jobsBody: { bucket, key, modelId, params… } | CreateJob(deploymentId, inputDataUri) | Crea y lanza un nuevo job en Watson ML. |
| 2.2 | **GET** /api/jobs/{jobId}/status | GetJobStatus(jobId) | Consulta estado (running, succeeded, failed). |
| 2.3 | **GET** /api/jobs/{jobId}/output | GetJobOutput(jobId) | Recupera resultados (e.g. función objetivo y sol). |
| 2.4 | **DELETE** /api/jobs/{jobId} | (opcional) DeleteJob(jobId) | Cancela o limpia el trabajo. |

**3. Flujo de datos completo**



**Aclaraciones**

* **Autenticación**: Cada llamada a COS y WML debe incluir credenciales/tokens.
* **Errores**: El servidor traduce errores de API externas a códigos HTTP estándar.
* **Extensibilidad**: Puedes añadir endpoints para editar parámetros, descargar logs, etc.

# 5. En base a su diseño preliminar decida qué tecnologías y frameworks utilizar, teniendo en cuenta la integración con las API de terceros y la seguridad.

Para el cliente se empleará **Vue.js** (por su reactividad y ecosistema de componentes) y en el servidor **Laravel** (por su robustez, manejo integrado de seguridad, autenticación y facilidad de consumo de APIs REST como COS y WML).

# 6. Sin llegar a codificar, establezca un plan de trabajo para cada integrante del equipo que permita desarrollar el sistema , incluya en estas tareas el tiempo estimado.

**Semana 1**:

* **Miembro A**: Configurar proyecto, diseñar UI, implementar páginas de gestión de datos
* **Miembro B**: Configurar backend, desarrollar endpoints
* Seguir con integración y pruebas, documentación final.

**Plan de trabajo (1 semana, entorno local, 2 personas)**

### **Integrante A – Frontend (Vue.js)**

| **Día** | **Tarea** | **Est. horas** |
| --- | --- | --- |
| Lunes | • Crear proyecto Vue y estructura de rutas/componente base | 3 h |
|  | • Maquetar pantalla “Buckets & objetos” (lista y botones CRUD) | 3 h |
| Martes | • Consumir endpoint GET /api/buckets y GET /api/buckets/{b}/objs | 4 h |
|  | • Previsualización de contenido (tabla) y manejo de estados | 2 h |
| Miércoles | • Maquetar pantalla “Ejecución de modelo” (formulario y status) | 2 h |
|  | • Integrar llamadas POST /api/jobs y polling de status | 4 h |
| Jueves | • Validaciones de formulario, manejo de errores y loaders | 3 h |
|  | • Estilizado final y ajustes responsivos | 3 h |
| Viernes | • Escribir manual breve de uso (README/UI) y preparar demo | 4 h |

**Total Integrante A:** 30 horas

### **Integrante B – Backend (Laravel)**

| **Día** | **Tarea** | **Est. horas** |
| --- | --- | --- |
| Lunes | • Crear proyecto Laravel, configurar .env y rutas básicas | 2 h |
|  | • Implementar IBM COS y configurar credenciales locales | 2 h |
|  | • Definir controladores y servicios para COS | 2 h |
| Martes | • Endpoint GET /api/buckets → ListBuckets() | 2 h |
|  | • Endpoint GET /api/buckets/{b}/objs → ListObjects(b) | 2 h |
|  | • Endpoint GET /api/buckets/{b}/objs/{k} → GetObject(b,k) | 2 h |
| Miércoles | • Endpoint POST /api/buckets/{b}/objs → PutObject(b,key,content) | 3 h |
|  | • Endpoint DELETE /api/buckets/{b}/objs/{k} → DeleteObject(b,k) | 2 h |
| Jueves | • Implementar Watson ML y configurar cliente | 2 h |
|  | • Endpoint POST /api/jobs → CreateJob(deploymentId, inputUri) | 2 h |
|  | • Endpoint GET /api/jobs/{id}/status → GetJobStatus(id) | 2 h |
| Viernes | • Endpoint GET /api/jobs/{id}/output → GetJobOutput(id) | 2 h |
|  | • Pruebas end-to-end de todos los endpoints y ajustes finales | 4 h |
|  | • Documentar API (README/backend) y preparar demo | 4 h |

**Total Integrante B:** 30 horas