Universidad Nacional de Lanús

DESARROLLO DE SOFTWARE EN SISTEMAS DISTRIBUIDOS

Web Services: SOAP/REST

Docentes:

- ➢ Ing Diego Andrés Azcurra
- ➢ Lic. Marcos Amaro

Integrantes:

- ➢ Lucas Nahuel Ambesi
- ➢ Dario Fabian Casais

Enunciado

Agregar a la web de recetas, las siguientes funcionalidades, las cuales deben estar divididas en módulos independientes comunicados mediante web services SOAP/REST.

Requerimientos

• **Recetario:** (similar a listas de reproducción) los usuarios registrados en la web de recetas podrán crear o borrar sus propios recetarios, los cuales contendrán las recetas que escojan. Los recetarios de cada usuario se identifican por su nombre.

a. Se requiere modificar la pantalla de detalle de receta para incluir la opción de “Agregar a recetario”, al clickear en dicha opción, el usuario debe elegir a qué recetario incluir dicha receta.
b. Debe existir una pantalla para que el usuario pueda ver, crear y borrar sus recetarios.

• **Denuncias:** los usuarios podrán denunciar recetas (no propias) para que un moderador decida qué hacer con ellas:

a. Agregar el rol de moderador, quien tendrá acceso a un panel donde se verá el listado de recetas denunciadas. Tendrá la opción de Ignorar denuncia o Eliminar receta.
b. Cuando se ignora o elimina la receta, se marcará la denuncia como Resuelta y no volverá a aparecer en el listado de recetas denunciadas.
c. Modificar la pantalla de detalle de receta para agregar la opción de denunciar. d. Cuando un usuario denuncia una receta, debe elegir un motivo de denuncia: Contenido inapropiado, Ingredientes prohibidos, Peligroso para la salud.

• **Correo interno:** los usuarios podrán enviarse mensajes privados entre sí.

a. Para esto deberá crearse la pantalla que contará con un buscador para elegir al destinatario, una caja de texto para escribir el asunto, el mensaje en sí y el botón para enviarlo.

b. Agregar la vista para consultar los mensajes recibidos y poder contestarlos. Cada mensaje es del tipo “asunto-mensaje-respuesta”. Es decir, por cada asunto puede haber solo un mensaje y una respuesta.

• **Carga Masiva:** un usuario podrá hacer una carga masiva de recetas mediante un archivo CSV. Los datos cargados se guardarán como un “Borrador” ya que las recetas cargadas de esta forma no están completas. El archivo csv se compone de los siguientes campos: título;descripción;categoría;tiempo de preparación.

a. Mediante la pantalla de Borradores, se listarán los borradores creados. Se podrán ir completando parcialmente o modificar los datos del borrador.
b. Una vez que se completen todos los datos restantes: fotos, pasos, ingredientes, se habilitará la opción de Guardar como receta, persistiendo esos datos en la entidad de Receta y eliminándola de los Borradores.

Se pide:
➢ Dos de los módulos hechos con SOAP y dos con REST.
➢ La documentación de las API REST se deben hacer con Swagger.
➢ Los módulos deben ser independientes (un proyecto para el servidor SOAP y otro para REST).

NORMAS DE ENTREGA

El trabajo entregado deberá contener un documento incluyendo:

➢ La estrategia de resolución del trabajo práctico. Es un texto descriptivo de cómo se estructuró la aplicación, todo aquello que consideren significativo para explicar la resolución del trabajo: diagrama del modelo de datos, arquitectura del sistema, etc.
➢ El código fuente, DE PROPIA AUTORÍA, del proyecto subido a un repositorio público de Github.
➢ Un enlace al video narrando las pruebas realizadas.
➢ Integrantes del grupo y las tareas realizadas por cada uno.
➢ Este trabajo práctico contará con una defensa, en fecha a definir.

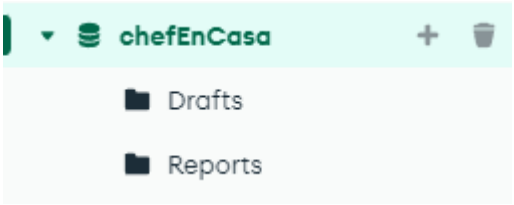
El incumplimiento de cualquiera de las normas de entrega implica la desaprobación del trabajo práctico.

Resolución

1. Estrategia de resolución

Para esta entrega y basándonos en los requisitos se crearon dos nuevos proyectos.

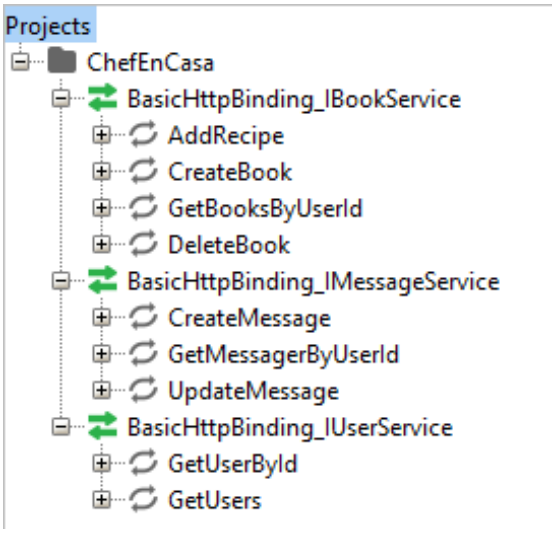
- **Proyecto rest-net:** este proyecto basado en **.NET** expone una api rest documentada con swagger para las funcionalidades de **“Carga masiva”** y **“Denuncias”**.
  Al ser dos funcionalidades poco dependientes del resto del sistema nos pareció interesante integrar una base de datos noSQL MongoDB al sistema para tener mayor distribución, está bd tiene dos colecciones “Drafts” y “Reports”.



  Esta API interactúa directamente con el front-end React mediante http rest.

- **Proyecto soap-net:** este proyecto basado en **.NET** expone una api soap para las funcionalidades de **“Recetario”** y **“Correo interno”**.
  La API se conecta con la base de datos relacional MySQL utilizada en los trabajos anteriores de manera directa.

  Se crearon tres servicios:
    - UserService
    - BookService
    - MessageService

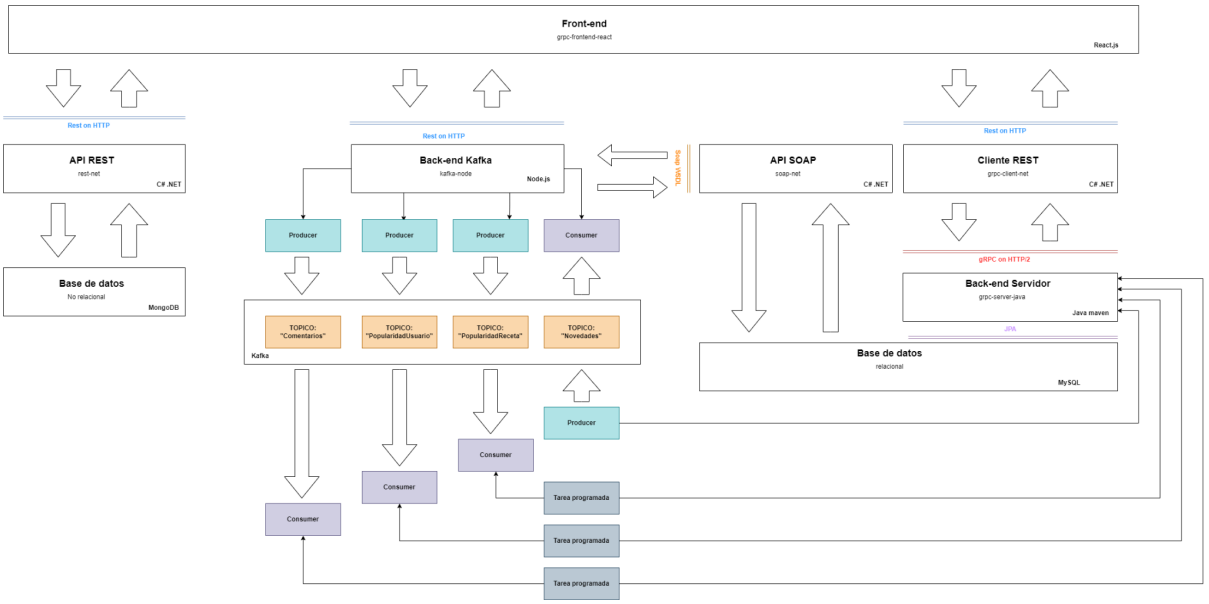


Decidimos utilizar el proyecto “kafka-node” como middleware para la comunicación con con front-end React, este proyecto desarrollado en node nos permite tener un mayor control en la manipulación de datos en la interacción con los servicios soap mediante la librería *“soap”* y simplificar la lógica en el lado del cliente dándole escalabilidad al sistema. El es el encargado de procesar las peticiones desde el frontend mediantes rest (json) y enviarlas a la API soap en formato xml, para luego transformar la respuesta recibida en xml a json para retornarla al front-end nuevamente.

Se puede hallar el código en el siguiente repositorio:

[lucasambesi/8627-1-TM-UNLa]

A continuación se mostrará la arquitectura de resolución planteada:

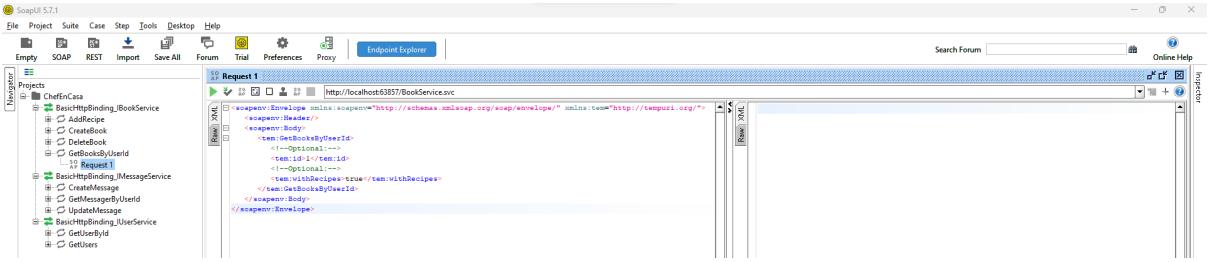


Ver en: [Arquitectura general.drawio]

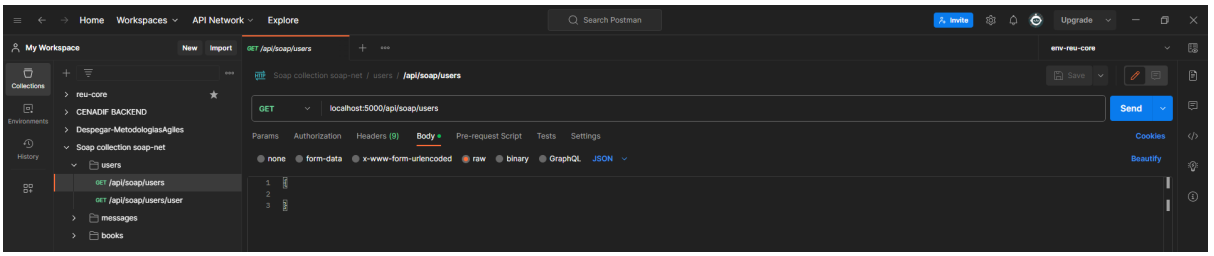
2. Herramientas utilizadas

Para este trabajo incorporamos nuevas herramientas al ambiente además de las ya utilizadas para los trabajos anteriores. Estas son:

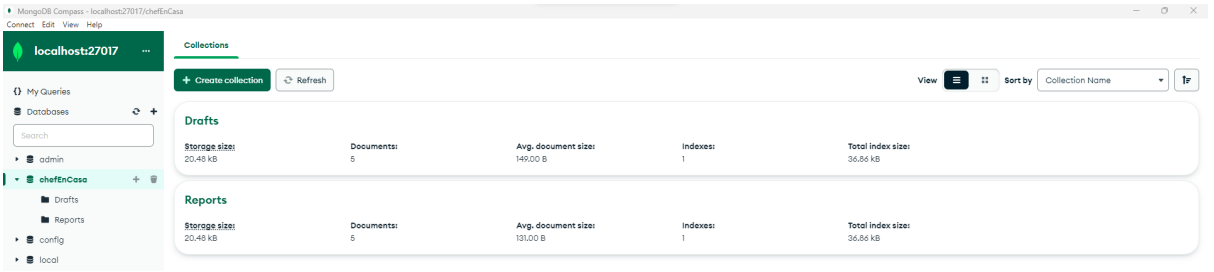
- Soap UI: Herramienta que utilizamos para realizar las pruebas de los servicios creados en el proyecto soap-net antes de integrarlos con el proyecto “kafka-node”.



- Postman: aunque esta herramienta ya fue utilizada en tps anteriores nos parece importante destacar ya que fue útil para realizar las pruebas de la api rest expuesta en el proyecto “kafka-node” el cual comunica con el proyecto soap-net antes de integrarlos con el frontend.



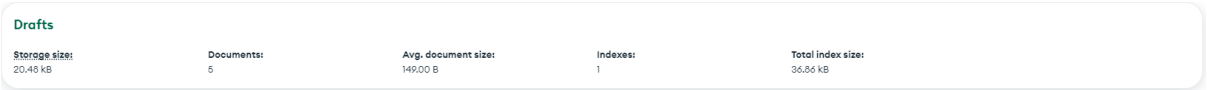
- MongoDB Compass: Herramienta que utilizamos como interfaz para poder acceder a la base de datos de MongoDB y tener facilidad de interacción con las colecciones y datos.



3. Funcionalidades

3.1 Carga masiva

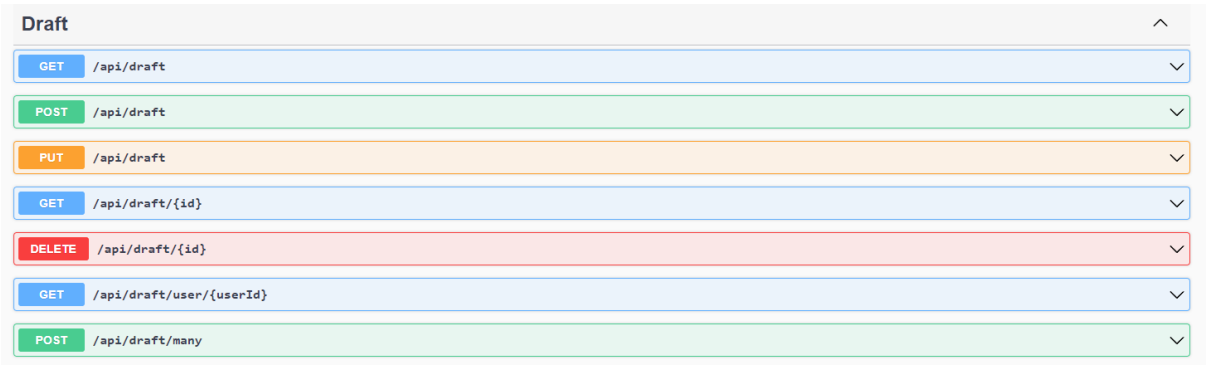
Para el desarrollo de esta funcionalidad fue necesario la creación de nuevas colección en la base de datos MongoDB para la persistencia de los borradores llamada “Drafts”.



Donde un ejemplo de documento es siguiente:

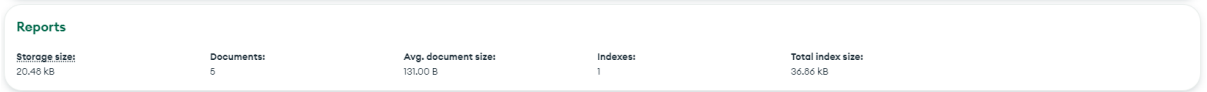
```
_id: ObjectId('65346f863552e5cfa205f803')
DraftId: null
UserId: "1"
Title: "prueba test6"
Description: "descripcion prueba"
Category: "4"
PreparationTime: "60"
```

También fue necesario la creación de los siguientes controllers (endpoints) en la api del proyecto rest-net:

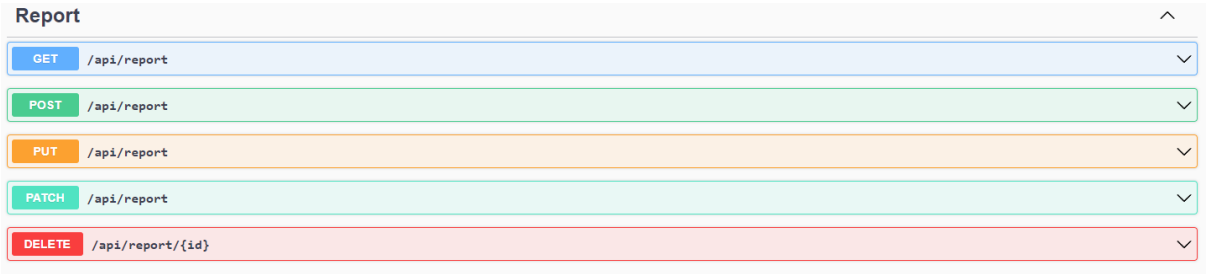


3.2 Denuncias

Para el desarrollo de esta funcionalidad fue necesario la creación de nuevas colección en la base de datos MongoDB para la persistencia de los borradores llamada “Reports”.

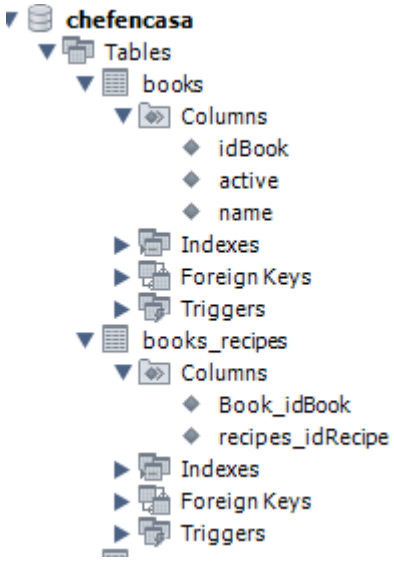


También fue necesario la creación de los siguientes controllers (endpoints) en la api del proyecto rest-net:

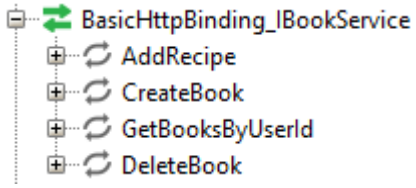


3.3 Recetario

Para el desarrollo de esta funcionalidad fue necesario la creación de nuevas tablas en la base de datos MySQL para la persistencia de los recetarios y la relación con sus recetas, estas son “books” y “books_recipes”:



También fue necesario la creación de un nuevo Servicio “BookService” en la API SOAP:



en donde se crearon los contratos:

- **AddRecipe:** en él se recibe un “idRecipe” y “idBook” para poder insertar una nueva receta en un recetario.
  Retorna un boolean que refleja el éxito de la operación.

```
<soapenv:Envelope xmlns:soapenv="http://
   <soapenv:Header/>
   <soapenv:Body>
      <tem:AddRecipe>
         <!--Optional:-->
         <tem:idRecipe>6</tem:idRecipe>
         <!--Optional:-->
         <tem:idBook>1</tem:idBook>
      </tem:AddRecipe>
   </soapenv:Body>
</soapenv:Envelope>
```

- **CreateBook:** en él se recibe un “name” y “idUser” para poder insertar un nuevo Recetario.
  Retorna un boolean que refleja el éxito de la operación.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmls
   <soapenv:Header/>
   <soapenv:Body>
      <tem:CreateBook>
         <!--Optional:-->
         <tem:name>Prueba de recetario</tem:name>
         <!--Optional:-->
         <tem:idUser>1</tem:idUser>
      </tem:CreateBook>
   </soapenv:Body>
</soapenv:Envelope>
```

- **DeleteBook:** en él se recibe un “idBook” para poder eliminar de manera lógica un Recetario.
  Retorna un boolean que refleja el éxito de la operación.

```
<soapenv:Envelope xmlns:soapenv="http://sch
   <soapenv:Header/>
   <soapenv:Body>
      <tem:DeleteBook>
         <!--Optional:-->
         <tem:idBook>?</tem:idBook>
      </tem:DeleteBook>
   </soapenv:Body>
</soapenv:Envelope>
```

- **GetBooksByUserId:** en él se recibe un “id” que pertenece al id del usuario y “withRecipes” para incorporar las recetas del recetario en la respuesta.
  Retorna una lista de los recetarios activos pertenecientes al usuario y las recetas que tiene cada recetario si “withRecipes” es indicado como “true”.

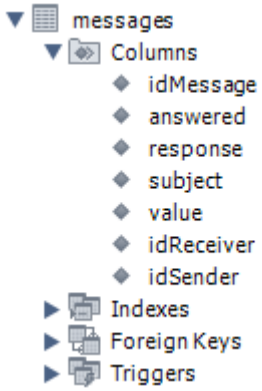
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xml
   <soapenv:Header/>
   <soapenv:Body>
      <tem:GetBooksByUserId>
         <!--Optional:-->
         <tem:id>1</tem:id>
         <!--Optional:-->
         <tem:withRecipes>true</tem:withRecipes>
      </tem:GetBooksByUserId>
   </soapenv:Body>
</soapenv:Envelope>
```

Dentro del proyecto “kafka-node” se expusieron endpoints, que a su vez redireccionan a callbacks que comunican con la api soap, para ser consumidos desde el frontend react:

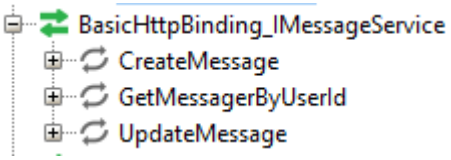
```
router.get('/api/soap/books/user', bookCallbacks.GetBooksByUserId);
router.post('/api/soap/books/recipe', bookCallbacks.AddRecipe);
router.post('/api/soap/books', bookCallbacks.CreateBook);
router.delete('/api/soap/books', bookCallbacks.DeleteBook);
```

3.4 Correo interno

Para el desarrollo de esta funcionalidad fue necesario la creación de una nueva tabla en la base de datos MySQL para la persistencia de los mensajes, esta es “messages”:



También fue necesario la creación de un nuevo Servicio “MessageService” en la API SOAP:



en donde se crearon los contratos:

- **CreateMessage:** en él se recibe un “IdReceiver”, “IdSender”, “Subject” y “Value” para poder insertar un nuevo mensaje.

Retorna un boolean que refleja el éxito de la operación.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns
   <soapenv:Header/>
   <soapenv:Body>
      <tem:CreateMessage>
         <!--Optional:-->
         <tem:request>
            <!--Optional:-->
            <ns:IdReceiver>?</ns:IdReceiver>
            <!--Optional:-->
            <ns:IdSender>?</ns:IdSender>
            <!--Optional:-->
            <ns:Subject>?</ns:Subject>
            <!--Optional:-->
            <ns:Value>?</ns:Value>
         </tem:request>
      </tem:CreateMessage>
   </soapenv:Body>
</soapenv:Envelope>
```

- **GetMessagerByUserId:** en él se recibe un “id” que corresponde al id de un usuario.
  Retorna un listado de mensajes filtrados por el id del usuario.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns
   <soapenv:Header/>
   <soapenv:Body>
      <tem:GetMessagerByUserId>
         <!--Optional:-->
         <tem:id>?</tem:id>
      </tem:GetMessagerByUserId>
   </soapenv:Body>
</soapenv:Envelope>
```

- **UpdateMessage:** en él se recibe un “IdMessage” y “Response” para poder agregar una respuesta a un mensaje.
  Retorna un boolean que refleja el éxito de la operación.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns
   <soapenv:Header/>
   <soapenv:Body>
      <tem:UpdateMessage>
         <!--Optional:-->
         <tem:request>
            <!--Optional:-->
            <ns:IdMessage>?</ns:IdMessage>
            <!--Optional:-->
            <ns:Response>?</ns:Response>
         </tem:request>
      </tem:UpdateMessage>
   </soapenv:Body>
</soapenv:Envelope>
```

Dentro del proyecto “kafka-node” se expusieron endpoints, que a su vez redireccionan a callbacks que comunican con la api soap, para ser consumidos desde el frontend react:

```
router.get('/api/soap/messages/user', messageCallbacks.GetMessagesByUserId);
router.post('/api/soap/messages', messageCallbacks.CreateMessage);
router.put('/api/soap/messages', messageCallbacks.UpdateMessage);
```

4. **Pruebas y demostración**

Para este punto decidimos grabar un video mostrando las nuevas funcionalidades. Compartimos el link:

TP N°3 WS SOAP/REST Grupo i

5. **Participación:**

Para desarrollar estas funcionalidades decidimos dividir el trabajo entre todos los integrantes de la forma más equitativa según nuestro punto de vista, para luego unirlo todo y realizar las conexiones.

- **Lucas Ambesi:** Proyecto SOAP e integración con middleware, conexión con la base de datos MySQL, integración de proyectos con frontend.
- **Darios Casais:** Proyecto rest-net, integración MongoDB con rest-net, documentación con Swagger.