Informe Proyecto 1

Felipe Sosa Patiño
fsosap@eafit.edu.co

Manuel Alejandro Gutierrez
magutierr@eafit.edu.co

Santiago Gil Zapata
sgilz@eafit.edu.co

March 2021

1 Introducción

El middleware orientado a mensajes es una aplicación que nos permite gestionar los mensajes enviados entre sistemas distribuidos, estos sistemas se comunican mediante una arquitectura cliente-servidor. MOM proporciona una capa que le permite a los clientes hacer una gestión correcta de sus mensajes mediante diferentes estructuras, sean canales o colas, estas estructuras si bien se comportan diferente y tienen un propósito único, las dos buscan gestionar ciertos mensajes que nacen de una aplicación productor y parten a una aplicación consumidor. En el siguiente informe se presenta la documentación realizada durante el proceso de desarrollo de nuestro servidor MOM, el cuál fue desarrollado en el framework de *PHP Laravel*, mientras que la aplicación productor y consumidor fueron desarrolladas en *Java* y *Python* respectivamente.

2 Lista de tablas

2.1 Reglas de seguridad para el servidor del middleware

Tipo	Protocolo	Rango de puertos	Origen
SSH	TCP	80	0.0.0.0/0
HTTP	TCP	22	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

Table 1: Reglas de entrada

Tipo	Protocolo	Rango de puertos	Origen
Todo el tráfico	Todos	Todos	0.0.0.0/0

Table 2: Reglas de salida

2.2 Rutas asociadas a la gestión de usuarios

Ruta	Método	Paramétros
/api/register	POST	name, email, password
/api/login	POST	email, password
/api/user-info	GET	-

Table 3: Rutas que permiten gestionar los usuarios

2.3 Rutas asociadas a la gestión de colas

Ruta	Método	Paramétros
/api/queue/create	POST	queue
/api/queue/list	GET	-
/api/queue/push	PUT	queue, message
/api/queue/pull	GET	queue
/api/queue/delete	DELETE	queue

Table 4: Rutas que permiten gestionar las colas

2.4 Rutas asociadas a la gestión de canales

Ruta	Método	Paramétros
/api/channel/channel	POST	channel
/api/channel/list	GET	-
/api/channel/push	PUT	channel, message
/api/channel/subscribe	POST	channel
/api/channel/getMessages	GET	channel
/api/channel/delete	DELETE	channel

Table 5: Rutas que permiten gestionar los canales

3 Lista de figuras

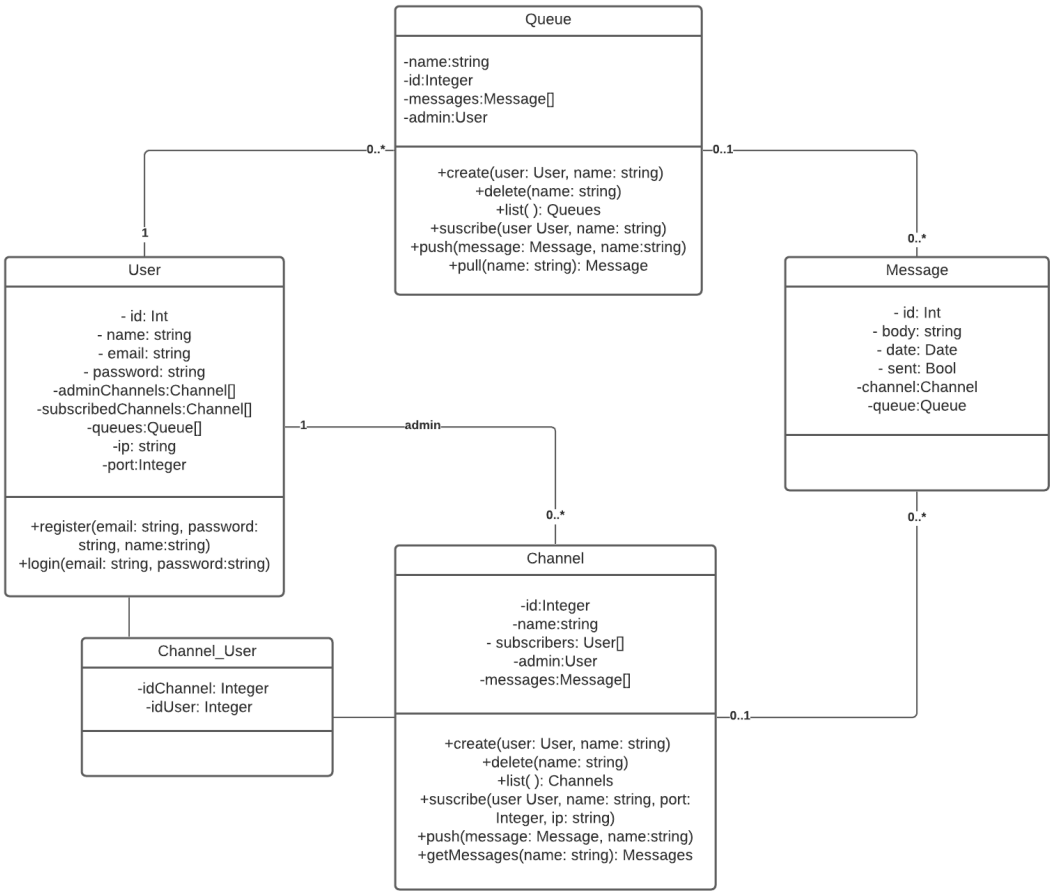

Figure 1: Diagrama de clases servidor MOM

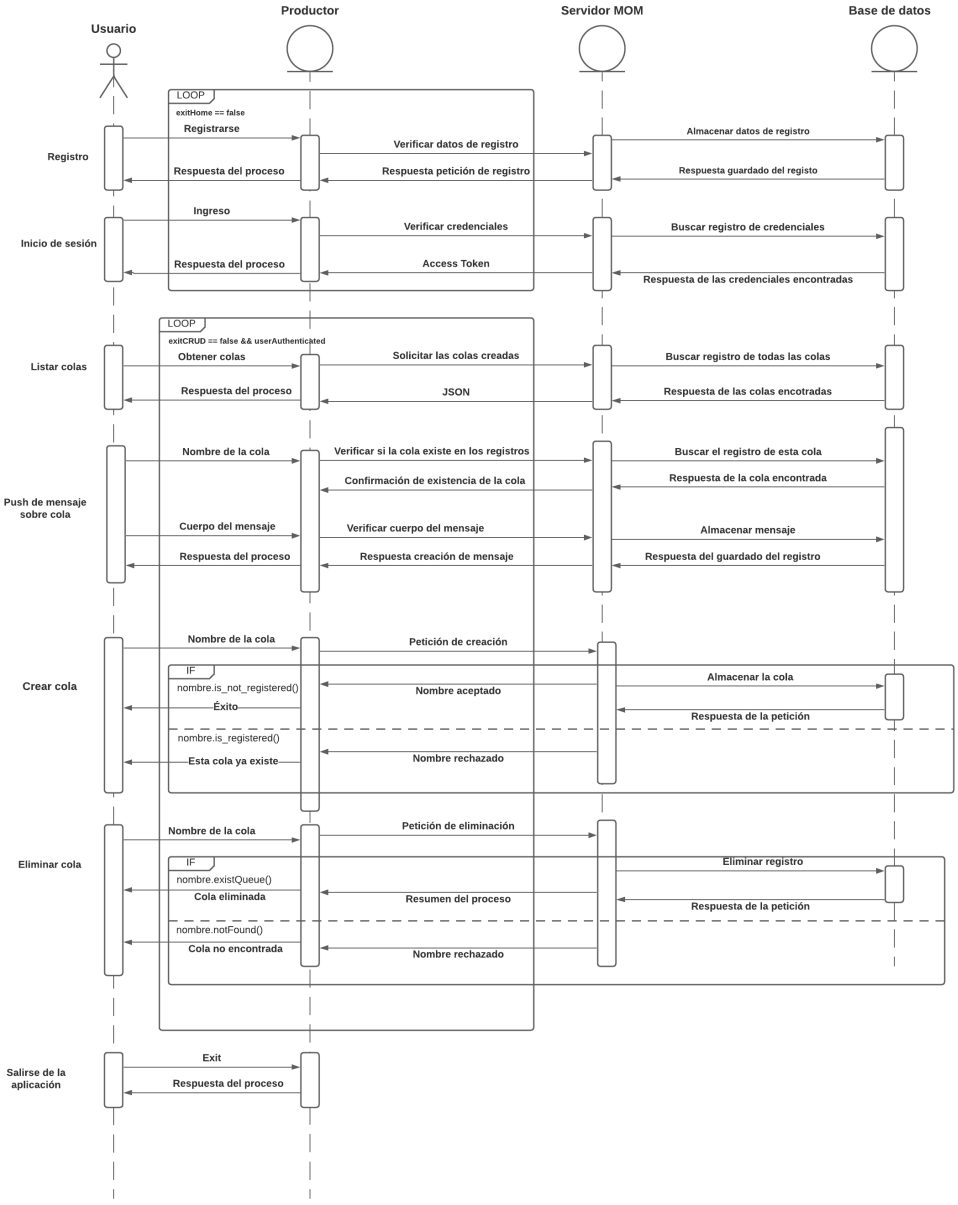

Figure 2: Diagrama de secuencias aplicación productor

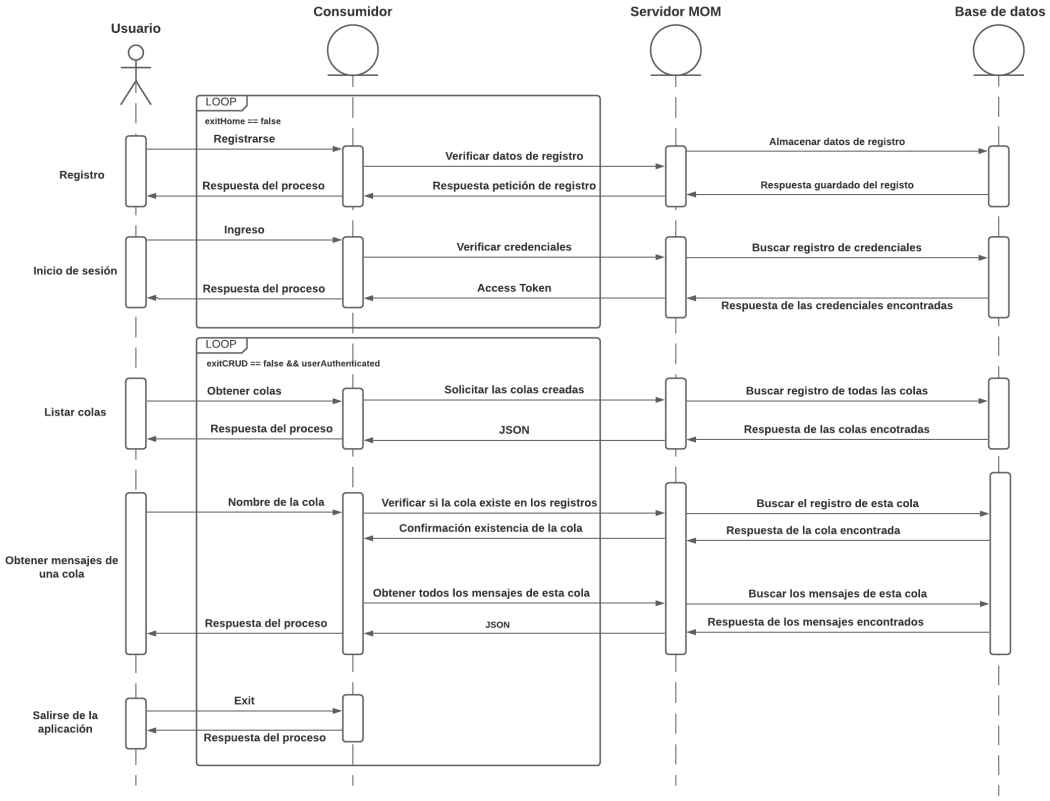

Figure 3: Diagrama de secuencias aplicación consumidor

```
PS D:\Universidad\7mo\Topicos-Telematica\consumer-middleware> & C:/Python38/python.exe
eware/Main.py
----------------Welcome to Consumer APP----------------
You can select one of these options:
1. Login
2. Sign in
3. Exit
-------------------------------------------------------
2
Please insert your e-mail: test@test.edu.co
Please insert your password: estoesuntest
Please insert your name: test
```

Figure 4: Primera interacción aplicación consumidor

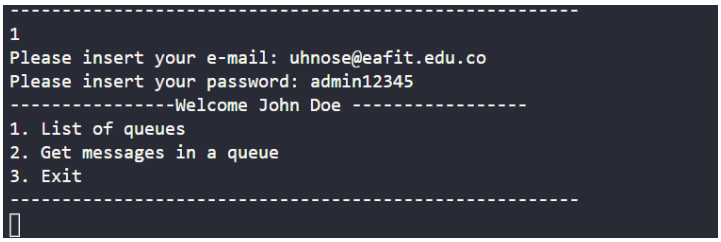

Figure 5: Segunda interacción aplicación consumidor

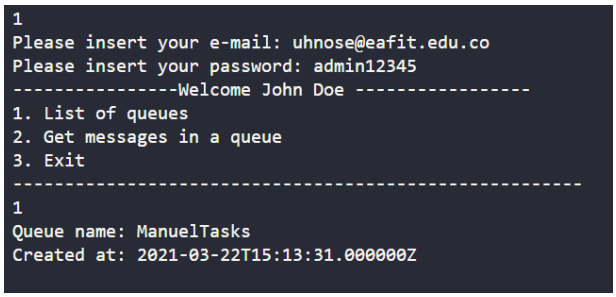

Figure 6: Tercera interacción aplicación consumidor

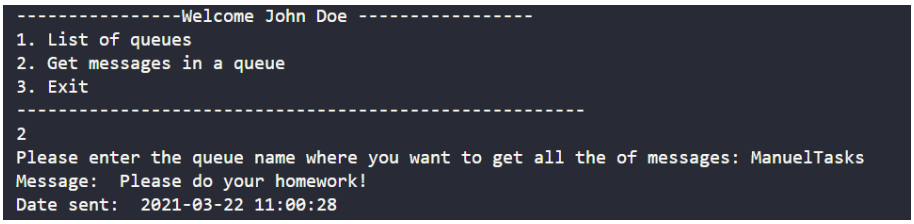

Figure 7: Cuarta interacción aplicación consumidor

4 Diseño

Para el diseño del MOM se tuvieron en cuenta las siguientes consideraciones:

1. La implementación del MOM se hizo dirigida al paradigma de Programación Orientada a Objetos (POO) donde las clases necesarias fueron: *User*, *Queue*, *Channel* y *Message*. Además de una clase intermedia para la conección entre múltiples Usuarios con múltiples canales. En la Figura 1 se puede ver su diagrama correspondiente.

2. Para las aplicaciones productora y consumidora, se crearon diagramas de secuencia con el fin de ilustrar cómo se realiza la interacción entre estas y el MOM. En las figuras 2 y 3 se puede apreciar esta interacción.

3. Para el desarrollo del MOM se utilizó PHP, específicamente el framework *Laravel*, el cual facilita el manejo de usuarios, rutas, manejo de bases de datos seguras, etc. En el caso de la aplicación productor se implementó en Java y la aplicación comsumidor se implementó en Python.

4. La autenticación de los usuarios se realizó utilizando el módulo *Sanctum* que provee Laravel para que cada usuario posea un *access token*. Esto se explica mejor en la Sección 5.1.1.

5. Se garantiza que sólo los dueños de sus canales y colas puedan eliminarlos. De igual modo, cuando esto sucede, se elminan en cascada sus registros correspondientes (los mensajes existentes en un canal o una cola).

6. Los servicios del MOM están expuestos como una Rest API y sus correspondientes métodos y rutas están descritos en las secciones 5.1.2 y 5.1.3.

7. El mecanismo de reepción de mensajes se definión en modo PULL dado que es más complejo entablar conexiónes persistentes en el tiempo por medio de HTTP.

8. La persistencia de datos se realizó utlizando el módulo *Eloquent* de *Laravel*, el cual es un ORM que facilita el manejo de bases de datos. Las especificaciones de la base de datos están descritas en la Sección 5.1.4.

9. Para la tolerancia a fallos en los mensajes, funcionarían bien una sincronización entre varias réplicas que tengan el mismo modelo entidad relación. En cuanto al servicio expuesto, tener varias réplicas de docker ejecutando la aplicación en diferentes instancias de una red privada sería ideal. En este caso, es importante pensar en un balanceador de cargas que esté distribuyendo estas peticiones entre los servidores disponibles.

5 Desarrollo

5.1 Middleware

5.1.1 Autenticación

Para realizar una comunicación segura entre la aplicación proveedor y el servidor MOM y una comunicación segura entre el servidor MOM y la aplicación consumidor se utiliza un token de autenticación que permite identificar al usuario que trata de acceder a los servicios de la API. Este token es creado en el servidor MOM usando una utilidad de laravel llamada sanctum que sirve para crear multiples llaves que le permitan al usuario tener acceso a diferentes servicios, pero en este caso, después de logeado el usuario, se le asigna un único token que le permite desde la aplicación proveedor hacer la creación y la eliminación de colas y realizar push de mensajes. En cambio, este token permite desde la aplicación consumidor hacer pull de los mensajes. Para hacer uso de *Sacntum*,

desde la clase User se hace un llamado al módulo *HasToken* que provee la biblioteca. De esta forma, cada que el usuario se autentique, se generará un nuevo Token perteneciente al usuario. La aplicación productor y consumidor reciben este token al hacer la petición de registro o de login. Esto permite que el usuario recientemente registrado no tenga que identificarse para poder acceder a los servicios y se agilice el proceso para empezar a hacer uso de los servicios del MOM. Este token es incluido en las credenciales enviadas en cada petición http que tenga que ver con acceso a la información de la base de datos del MOM, como solicitar la información del usuario logeado, y en las peticiones que interactúan con las colas y los mensajes en ellas, como hacer el envío de un mensaje sobre una cola.

Existen varias formas de realizar el manejo de credenciales de usuarios en laravel, dentro de las cuales destacan passport y sanctum. Fue elegido sanctum pues fue pensado como un paquete de autenticación para aplicaciónes single page. Esto implica que sanctum es una herramienta liviana y más simple a la hora de ser implementada y encaja con las necesidades del sistema propuesto para la presente práctica. Para disponer de sanctum, fue descargado y configurado el paquete. Además, se alteró la configuración del modelo del usuario permitiendo que utilizara access tokens para identificarse. Después de esta configuración inicial fue implementado el controlador de autenticación conectado con las rutas API de login, de registro y de obtener la información del usuario logeado. Los metodos de login y de registro, ubicados en dicho controlador, validan el formato de la información y que no falten datos obligatorios para posteriormente, cuando sea validada la información ingresada, se cree un access token para el usuario que permita tener acceso a las funcionalidades del MOM. Este token tiene validez dentro del sistema por 1 hora, lo que permitiría usar el mismo token y no tener que volver a logearse dentro de dicho plazo. Finalmente, fueron desarrolladas las funcionalidades de la API y dentro de las rutas se especifica cuales de ellas necesitan autenticación para ser utilizadas.

5.1.2 Métodos

Las reglas de entrada para el servidor en el que se encuentra alojado el middleware desarrollado se pueden apreciar en la Tabla 1 donde se permite una conexión ssh para hacer las configuraciones necesarias a través de control remoto con el fin de desplegar el código y hacerlo disponible a través de un contenedor de docker. Además, el servidor cuenta con conexiones HTTP y HTTPS para recibir peticiones REST de forma segura, peticiones enviadas por la app consumidor y la app proveedor y permite trafico de información libre para poder enviar la respuesta a las aplicaciones.

Para la gestión de usuarios, vemos en la Tabla 3 que tanto el registro como el login son peticiones tipo POST gracias a que se crea un access token al realizar alguna de las 2 operaciones. Nótese que se debe ingresar el email y la contraseña en ambas, pero para el registro se solicita un username.

Despues de realizadas alguna de las 2 operaciones anteriores se obtiene un access token, el cual es necesario enviarlo dentro del header de las peticiones

posteriores como las que vemos en la Tabla 4 y en la Tabla 5 que corresponden a las rutas que permiten gestionar las colas y los canales. Esto permite, hacer la creación(método POST), eliminación(método DELETE) y lista(método GET) de las colas y los canales existentes. Para que funcionen las colas y los canales como medio de comunicación entre proveedor y consumidor se hace necesario tener mensajes para transmitir. En el caso de las colas, los mensajes se pueden enviar realizando la operación push, que utiliza el método PUT pues realiza un cambio en la cola a donde va dirigido el mensaje y los mensajes se reciben haciendo la operación de pull que es tipo GET, pues simplemente trae el resultado de la consulta. Por el otro lado, el canal al cual se le envían mensajes a través de un método push de tipo PUT, tiene la particularidad de tener un método llamado subscribe que significa que para que un usuario reciba datos de dicho canal debe suscribirse a dicho canal. Esto se hace a través de una petición tipo POST porque se crea la suscripción. Despues de suscrito el usuario puede traer los mensajes usando la ruta getMessage usando una petición GET tal como se hizo pull en las colas.

5.1.3 Rutas

En las Tabla 3, Tabla 4, Tabla 5 se muestran las rutas que puede gestionar el middleware, en las rutas anteriores se podrá darle la gestión a las canales y las colas. Es importante aclarar que para poder utilizar estas rutas el usuario debió haber estado registrado y logueado previamente, ya que para tener acceso a estas rutas cualquier aplicación sea Productor o Consumidor deberá obtener su *Access Token*, que le permitirá identificarse ante el servidor.

5.1.4 Base de datos

Para garantizar la persistencia de la aplicación, utilizamos el motor de bases de datos *MariaDB* debido a que tiene una gran integración con *Eloquent*, el ORM de *Laravel* que permite mapear los modelos de nuestra aplicación hacia sus tablas correspondientes en base de datos. Además garantizar que el acceso a base de datos pase por una capa intermedia de seguridad, de este modo se evitan los ataques de SQL injection. El despliegue de la base de datos se hizo en una instancia de EC2, y es explicada en el literal 3 de la siguiente sección.

5.2 Despliegue

Para el despliegue de nuestro MOM, utilizamos los servicios de AWS. A continuación, los pasos para la adecuación del entorno:

1. Creamos una VPC donde recidirá la subred en la que vamos a desplegar la aplicación. De allí, creamos una subred de dirección *172.16.0.0/16* para identificar localmente las instancias de la red. A esta VPC se le agregó un *Internet Gateway* y un *Route table* para garantizar el flujo de paquetes desde y hacia las instancias de la red.

2. Lanzamos una nueva instancia EC2 con un sistema operativo *Amazon Linux* (AMI 2) dentro de subred la manejada por la VPC que creamos en el paso anterior. Para proteger el tráfico paquetes desde Internet hacia la intancia, configuramos un *Security group* (SG) que sólo permitiera el tráfico de entrada a través de SSH, HTTP y HTTPS. Así, se podrá sólo acceder al MOM desplegado y desde la consola para términos de administración de la instancia. Las reglas agregadas al SG se pueden ver en las tablas 1 y 2.

3. Una vez con la instancia corriendo, instalamos el servidor de base de datos de MariaDB:

```
sudo yum update -y
sudo yum install -y mysql
sudo systemctl start mysql
sudo systemctl enable mysql
sudo mysql_secure_installation
...
mysql -u root -p

    > CREATE DATABASE middleware;
    > CREATE USER gzsantiago22@'%' IDENTIFIED BY '201810091010';
    > GRANT ALL privileges ON middleware.* TO 'gzsantiago22'@'%'
      IDENTIFIED BY '201810091010';
```

4. Decidimos aislar la ejecución de nuestra aplicación. Para ellos instalamos *docker*:

```
sudo yum install -y docker
sudo systemctl start docker
sudo systemctl enable docker
```

5. Para la correcta instalación de las dependincias del proyecto, en el *Dockerfile* incluimos lo siguiente:

```
FROM php:7.3-apache-stretch
RUN apt-get update -y && apt-get install -y \
    openssl zip unzip git
RUN docker-php-ext-install pdo_mysql
RUN curl -sS https://getcomposer.org/installer | \
    php -- --install-dir=/usr/local/bin --filename=composer
RUN sed -ri -e 's!/var/www/html!/var/www/html/public!g' \
    /etc/apache2/sites-available/*.conf
RUN sed -ri -e 's!/var/www/!/var/www/html/public!g' \
```

```
    /etc/apache2/apache2.conf /etc/apache2/conf-available/*.conf
COPY . /var/www/html
COPY ./public/.htaccess /var/www/html/.htaccess
WORKDIR /var/www/html
RUN composer install \
    --ignore-platform-reqs \
    --no-interaction \
    --no-plugins \
    --no-scripts \
    --prefer-dist
RUN php artisan key:generate
RUN php artisan migrate
RUN chmod -R 777 storage
RUN a2enmod rewrite
RUN service apache2 restart
```

6. Creamos la imagen donde se instalarán los paquetes necesarios para la app:

```
docker image build -t middleware-image .
```

7. Finalmente montamos nuestro contenedor de docker donde recidirá el MOM. Note que se hace un mapeo del puerto 80 de nuestra máquina al mismo dentro del contenedor para que la comunicación desde internet se redirija al contenedor:

```
docker container run -d --name middleware-container \
    -p 80:80 middleware-image
```

6 Productor

La aplicación productor tiene como objetivo permitir al usuario enviar mensajes a una cola. Esta aplicación hace parte de un sistema gestor de tareas para procesamiento distribuído. Para cumplir con esta tarea la aplicación también permite realizar la gestión de colas, facilitando la creación, eliminación y listando las colas existentes. Todas estas tareas necesitan que el usuario esté identificado por el servidor MOM, por lo que la aplicación cuenta con un menú inicial donde se le permite hacer login o hacer registro. Después de que alguna de las operaciones anteriores sea exitosa, se accede al menú que permite crear, eliminar , listar colas y enviar tareas a una cola específica. Estas tareas tienen un formáto definido, en el que se especifica el número de la tarea enviada, el nombre e email del usuario que la envió y una descripción de la tarea. La aplicación permite navegar en el menú de opciones regresando al menú principal cuando se desee y va dando información sobre el estado de la solicitud enviada.

6.1 Ejecución del programa productor

Para poder ejecutar correctamente la aplicación productor se deberá seguir los siguientes pasos:

1. Descargar el repositorio que encontrará en *GitHub* utilizando el siguiente comando en la terminal.

```
git clone https://github.com/fsosap/producer.git
```

2. Ingresar al directorio principal del proyecto usando el comando

```
cd producer
```

3. Este cliente fue realizado en java version "1.8.0-281" JDK 8. Para ejecutar este programa debe de tener instalado java, preferiblemente en dicha versión

4. Ejecute el programa usando el siguiente comando

```
java -jar provider.jar
```

6.2 Interacción de la aplicación productor con el middleware

Para realizar las peticiones al middleware se utilizó la librería *HttpURLConnection* del recurso java.net. Todas las peticiones se hacen mediante el protocolo de aplicación *HTTP*. Véase el diagrama de secuencia en la Figura 2. De dicho diagrama destacamos lo siguiente:

- Es necesario hacer un registro exitoso para poder obtener el token de acceso a los servicios, esto permite que en el primer ingreso(cuando se hizo el registro) permita el acceso a los servicios del MOM. a partir del segundo ingreso se hace el proceso de login para tener dicho acceso. Esta operación realiza una petición tipo *POST* a la ruta

```
http://54.164.144.28/api/register?name=<yourusername>
&email=<your@email.com>&password=<yourpassword>
```

```
http://54.164.144.28/api/login?email=<your@email.com>
&password=<yourpassword>
```

- Después de crear el usuario y tener acceso a la plataforma, el flujo más común sería crear una cola lo que se haría usando también una petición tipo *POST* a la ruta

  ```
  http://54.164.144.28/api/queue/create?name=<queuename>
  ```

- Puedes confirmar la creación de esta cola haciendo una petición tipo *GET*

  ```
  http://54.164.144.28/api/queue/list
  ```

- Lo siguiente dentro del flujo normal de la app sería hacer push de una tarea en dicha cola creada o en una de las que se encuentre en la lista. Esto se hace a través de un método tipo *PUT* a la ruta

  ```
  http://54.164.144.28/api/queue/push?queue=<targetqueue>
  &body=<taskmessage>
  ```

- Finalmente, si se desea y se ha comprobado que se hizo pull desde la aplicación consumidor, se puede hacer la eliminación de dicha cola a través de un método tipo *DELETE* a la ruta

  ```
  http://54.164.144.28/api/queue/delete?name=<queuename>
  ```

7 Consumidor

La aplicación consumidor tiene como objetivo permitirle a un usuario recolectar mensajes de una cola. Para poder realizar esta tarea la aplicación le permite al usuario registrarse y loguearse, para poder crear un usuario se debe ingresar su nombre, contraseña y correo electrónico. Finalmente la aplicación le permitirá al usuario poder listas las colas para poder traer el mensaje de la cola que desee.

7.1 Ejecución del programa consumidor

Para poder ejecutar correctamente la aplicación consumidor usted deberá seguir los siguientes pasos:

1. Descargar el repositorio en *GitHub* desde la terminal de comandos usted puede digitar el siguiente comando

   ```
   git clone https://github.com/ManuelG28/consumer-middleware.git
   ```

2. Una vez se haya bajado el repositorio, se debe asegurar que esté en el directorio principal del proyecto, para ello debe digitar el siguiente comando

```
cd consumer-middleware
```

3. Este cliente fue realizado en *Python 3.8.x*, debe asegurarse que cuenta con esta versión del lenguaje o con una superior.

4. Además, deberá instalar una biblioteca que se utilizó durante el desarrollo del cliente. Esta biblioteca se llama *requests* la cual nos permitirá realizar peticiones HTTP de una manera sencilla, para poder instalarla debes digitar el siguiente comando

   ```
   python -m pip install requests
   ```

5. Finalmente podrá ejecutar el proyecto con el siguiente comando

   ```
   python Main.py
   ```

7.2 Interacción de la aplicación consumidor con el middleware

Para realizar las peticiones al middleware, se utilizó la librería *requests* de python, todas las peticiones que se hacen son mediante *HTTP* el flujo de la aplicación se puede observar en la Figura 3, los siguientes puntos se explicará detalladamente como interactúa la aplicación consumidor con el servidor.

- El paso iniciar es crear un usuario si aún no lo ha hecho para poder interactuar con todas las funcionalidades que tiene el servidor, para esto deberá seleccionar la segunda opción al iniciar la aplicación, esto se puede observar en la Figura 4. Esta interacción realizará un método *POST* a la ruta

  ```
  http://54.164.144.28/api/register?
  name=test&email=test@test.edu.co&password=estoesuntest
  ```

- Una vez este registrado, podrás iniciar sesión en el servidor y comenzar a realizar peticiones a las colas, para poder realizar esto deberás escoger la opción 1 como se observa en la Figura 5. Esta interacción realizará un método *POST* a la ruta

  ```
  http://54.164.144.28/api/login?
  email=uhnose@eafit.edu.co&password=admin12345
  ```

- Una vez este autenticado, podrá listar las colas que actualmente tiene el servidor creado como se observa en la Figura 6. Esta interacción realizará un método *GET* a la ruta

`http://54.164.144.28/api/queue/list`

- Cuando sepa cuales, y cuántas colas han sido creadas, podrá traer los mensajes de la cola que usted desee, este paso se puede observar en la Figura 7. Esta interacción realizará un método *GET* a la ruta

  `http://54.164.144.28/api/queue/pull?queue=ManuelTasks`