



**El futuro digital
es de todos**

MinTIC



Ciclo 4a:

Desarrollo de aplicaciones web



**Misión
TIC 2022**

VERSIÓN 1.0

**Unidad de educación
continua y permanente
Facultad de Ingeniería**



Unidad Camilo Torres
Calle 44 # 45-67
Bloque B5 piso 1



(57) + 316 5000
uec.ftbog@unaleduco

Actividad Práctica

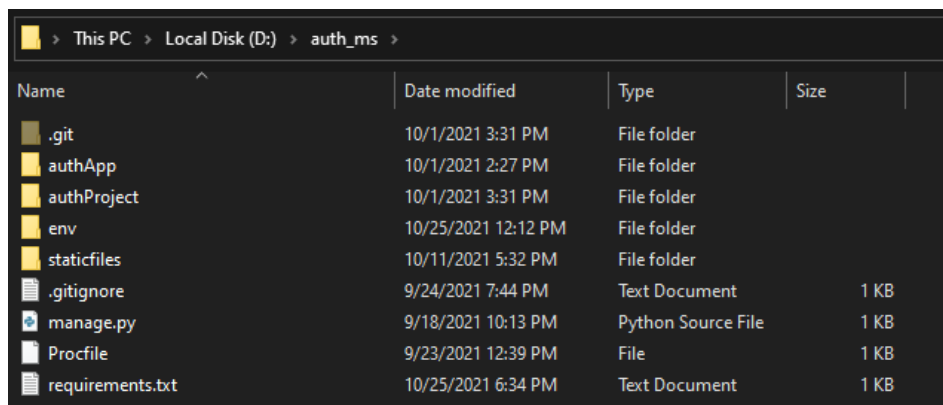
(Microservicio AuthMS)

Para el desarrollo del microservicio *auth_ms* se utilizará el componente lógico del sistema de software planteado en el ciclo 3: *bank_be*. Por esta razón, el propósito de esta guía es realizar los cambios y ajustes necesarios para adaptar el proyecto *bank_be* a la arquitectura de microservicios indicada en sesiones anteriores.

Nota: de ser necesario, en el material de la clase se encuentra un archivo llamado *C4a.AP.01. bank_be.rar*, con el componente desarrollado en Ciclo 3.

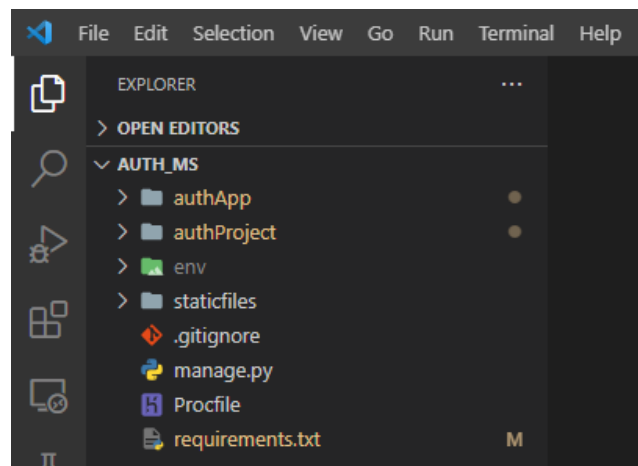
Preparación del entorno de edición del proyecto

Para editar el proyecto es necesario abrir la carpeta que contiene el proyecto *bank_be* en Visual Studio Code. Sin embargo, el nombre de dicha carpeta (*bank_be*) no describe el propósito que tendrá este componente dentro de la nueva arquitectura. Por lo cual, antes de abrirla se cambiará su nombre por uno que permita identificar su rol dentro de la arquitectura, en este caso: *auth_ms*. Este cambio solo afecta el nombre de la carpeta, por lo cual, la estructura del proyecto debe permanecer igual:

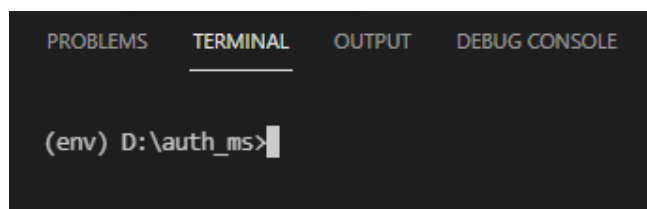


Name	Date modified	Type	Size
.git	10/1/2021 3:31 PM	File folder	
authApp	10/1/2021 2:27 PM	File folder	
authProject	10/1/2021 3:31 PM	File folder	
env	10/25/2021 12:12 PM	File folder	
staticfiles	10/11/2021 5:32 PM	File folder	
.gitignore	9/24/2021 7:44 PM	Text Document	1 KB
manage.py	9/18/2021 10:13 PM	Python Source File	1 KB
Procfile	9/23/2021 12:39 PM	File	1 KB
requirements.txt	10/25/2021 6:34 PM	Text Document	1 KB

Una vez se ha nombrado adecuadamente la carpeta contenedora del proyecto, esta se debe abrir en Visual Studio Code:



Posteriormente, se debe abrir una nueva terminal e iniciar el entorno virtual de Python (`env\Scripts\activate` en Windows o `source env/bin/activate` en Linux/MacOS):



Cambiar las configuraciones del proyecto

Dentro del proyecto se configuraron los headers CORS para que el usuario pudiera acceder a los servicios del componente lógico desde un navegador web. Sin embargo, dicha configuración no será necesaria en la arquitectura de microservicios, pues el único componente al que el usuario tendrá acceso desde el navegador web será al API Gateway. Por esta razón, se eliminará la configuración de los headers CORS de la siguiente forma:

Primero se debe eliminar el paquete `django-cors-headers` de la lista de paquetes que se encuentra en el archivo `requirements.txt`. Una vez hecho esto, el código dentro del archivo debe ser el siguiente:

```
django==3.2.7
djangorestframework==3.12.4
djangorestframework-simplejwt==4.8.0
psycopg2==2.9.1
gunicorn==20.1.0
django-heroku==0.3.1
```

Luego se deben eliminar las configuraciones de CORS en el archivo `authProject/settings.py`. Para ello, se deben eliminar las variables `ALLOWED_HOSTS` y `CORS_ALLOW_ALL_ORIGINS`, y el valor `'corsheaders'` de la lista `INSTALLED_APPS`. Al hacerlo la lista debe quedar de la siguiente forma:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'rest_framework',  
    'authApp',  
]
```

Por último, se debe eliminar el valor `'corsheaders.middleware.CorsMiddleware'` de la lista `MIDDLEWARE`. Al hacerlo la lista debe quedar de la siguiente forma:

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]
```

Adicionalmente, es necesario modificar una configuración del paquete `Simple JWT` para evitar inconvenientes con la validación de tokens. Para ello, en la lista `SIMPLE_JWT` se debe cambiar el valor del atributo `BLACKLIST_AFTER_ROTATION` por `False`. Al hacerlo la lista debe quedar de la siguiente forma:

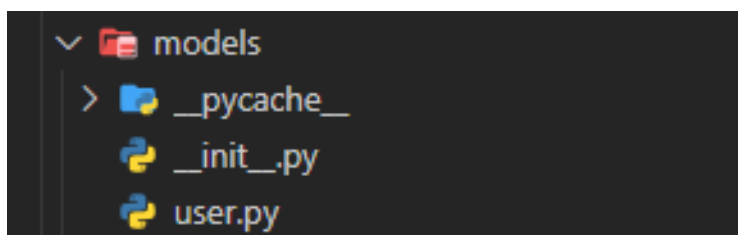
```
SIMPLE_JWT = {  
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=5),  
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),  
    'ROTATE_REFRESH_TOKENS': False,  
    'BLACKLIST_AFTER_ROTATION': False,  
    'UPDATE_LAST_LOGIN': False,  
  
    'ALGORITHM': 'HS256',  
    'USER_ID_FIELD': 'id',
```

```
'USER_ID_CLAIM': 'user_id',  
}
```

Eliminar el modelo Account

Como se mencionó en sesiones anteriores, el propósito del microservicio [auth_ms](#) tiene que ver únicamente con el registro y autenticación de un usuario. Por ello, el microservicio que se encargue de las funcionalidades asociadas con el modelo [Account](#) será uno diferente a [auth_ms](#) y, por ende, se debe eliminar este modelo del microservicio. Para eliminar el modelo [Account](#) y todo lo relacionado con este se deben seguir los siguientes pasos:

Inicialmente se debe eliminar el archivo [authApp/models/account.py](#), al hacerlo la estructura de la carpeta [authApp/models](#) será la siguiente:



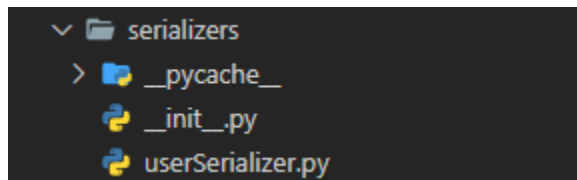
Luego, se debe eliminar el registro de la clase [Account](#) dentro del archivo [authApp/models/__init__.py](#). Una vez se ha hecho esto, el código de este archivo debe ser el siguiente:

```
from .user import User
```

Posteriormente, debido a que se eliminó uno de los modelos que Django estaba utilizando, se debe eliminar también su registro en el archivo [authApp/admin.py](#). Una vez se ha hecho esto, el código de este archivo debe ser el siguiente:

```
from django.contrib import admin  
from .models.user import User  
  
admin.site.register(User)
```

Ya que ahora el modelo [Account](#) no existe en el microservicio, es necesario eliminar todo lo que hacía referencia a este. En este caso, este modelo se utilizaba en los archivos [accountSerializer.py](#) y [userSerializer.py](#). En el primer caso, se debe eliminar el archivo [authApp/serializers/accountSerializer.py](#), de tal forma que la estructura de la carpeta [authApp/serializers](#) sea la siguiente:



Además, se debe eliminar el registro de la clase `AccountSerializer` dentro del archivo `authApp/serializers/__init__.py`. Una vez se ha hecho esto, el código de este archivo debe ser el siguiente:

```
from .userSerializer import UserSerializer
```

Finalmente, para eliminar la referencia del modelo `Account` en el archivo `userSerializer.py` se debe reemplazar el código del archivo `authApp/serializers/userSerializer.py` por uno en el que no se relacione un registro del modelo `User` con un elemento del modelo `Account`. Para ello, se reemplaza por el siguiente código:

```
from rest_framework import serializers
from authApp.models.user import User

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'password', 'name', 'email']
```

Se debe notar que se eliminó tanto el campo `account` del modelo `User`, como el método para la creación de un registro del modelo `User`, pues ya no es necesario crear una cuenta para asociarla con el usuario creado.

Adición y modificación de vistas

Debido al propósito que tiene el microservicio dentro del sistema, se deben realizar algunos cambios a las funcionalidades que este expone, lo cual a su vez implica modificar algunas vistas del microservicio.

La vista `UserCreateView` no se debe modificar, pues el registro de nuevos usuarios sigue siendo una de las funcionalidades de este microservicio. Sin embargo, la vista `UserDetailView` si se debe modificar, pues ya no es necesario verificar que el token de la petición, le pertenece al usuario cuya información se solicita. Esto se debe a que, como se verá en próximas sesiones, el API Gateway se encargará de esta operación. Por

esta razón, se debe eliminar la petición GET de la vista [UserDetailView](#). Al hacerlo, en código del archivo [authApp/views/userDetailView.py](#) debe ser similar al siguiente:

```
from rest_framework import generics

from authApp.models.user import User
from authApp.serializers.userSerializer import UserSerializer

class UserDetailView(generics.RetrieveAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
```

Por otro lado, debido a que será el API Gateway el componente que se encargue de verificar que un usuario se encuentra autenticado al hacer una petición HTTP, es necesario crear una vista que le permita a este verificar si un token es válido o no, y conocer el id del usuario al que le pertenece dicho token. Para esto, el paquete [Simple JWT](#) dispone de la vista [TokenVerifyView](#) ya implementada, sin embargo, esta únicamente indica si el token es válido o no. Por ello, para satisfacer las necesidades del API Gateway, es necesario sobrescribir el método POST de esta vista para retornar el [id](#) del usuario en caso de que el token sea válido.

Para hacerlo, se debe crear el archivo [authApp/views/verifyTokenView.py](#) y dentro de este se debe utilizar el siguiente código:

```
from django.conf import settings
from rest_framework import status
from rest_framework.response import Response
from rest_framework_simplejwt.views import TokenVerifyView
from rest_framework_simplejwt.backends import TokenBackend
from rest_framework_simplejwt.exceptions import InvalidToken, TokenError
from rest_framework_simplejwt.serializers import TokenVerifySerializer

class VerifyTokenView(TokenVerifyView):

    def post(self, request, *args, **kwargs):
        serializer = TokenVerifySerializer(data=request.data)
        tokenBackend = TokenBackend(algorithm=settings.SIMPLE_JWT['ALGORITHM'])

        try:
            serializer.is_valid(raise_exception=True)
            token_data = tokenBackend.decode(request.data['token'], verify=False)
            serializer.validated_data['UserId'] = token_data['user_id']

        except TokenError as e:
```



```
raise InvalidToken(e.args[0])

return Response(serializer.validated_data, status=status.HTTP_200_OK)
```

En este código se utilizan los métodos y Serializers dispuestos por [Simple JWT](#) tanto para verificar la validez del token, como para obtener su id asociado, y se retorna una respuesta HTTP con el [id](#) del usuario solo si el token es válido.

Una vez se ha creado la vista, al igual que se ha hecho con las demás, esta se debe registrar en el archivo [__init__.py](#) de la carpeta en la que se encuentra. Al hacerlo, el código del archivo [authApp/views/__init__.py](#), debe ser similar a lo siguiente:

```
from .userCreateView import UserCreateView
from .userDetailView import UserDetailView
from .verifyTokenView import VerifyTokenView
```

Además, también se debe asociar la nueva vista con una url del microservicio, la cual será consumida por el API Gateway. Una vez se ha hecho esto, el código del archivo [authProject/urls.py](#) debe ser similar a lo siguiente:

```
from django.urls import path
from rest_framework_simplejwt.views import (TokenObtainPairView, TokenRefreshView)
from authApp import views

urlpatterns = [
    path('login/', TokenObtainPairView.as_view()),
    path('refresh/', TokenRefreshView.as_view()),
    path('verifyToken/', views.VerifyTokenView.as_view()),
    path('user/', views.UserCreateView.as_view()),
    path('user/<int:pk>', views.UserDetailView.as_view()),
]
```

Reinicio de la base de datos

Una vez se han realizado todas las modificaciones necesarias al microservicio [auth_ms](#) es necesario reiniciar la base de datos, esto para que no haya problemas de inconsistencia con los modelos de los datos. Para ello, se debe dirigir a la consola de Heroku, y se debe abrir el proyecto donde se encuentra desplegada la base de datos del microservicio:

Personal New

Filter apps and pipelines

- mission-tic-bank-be
- mission-tic-bank-db**
- mission-tic-bank-fe

Luego se debe dirigir a la pestaña **Resources**:

mission-tic-bank-db Open app More

Overview **Resources** Activity Access Settings

Get a complete visualization of your app in a team-based continuous delivery environment with [Heroku Pipelines](#). Hide Create a Heroku Pipeline

Installed add-ons **\$0.00/month** [Configure Add-ons](#)

- Heroku Postgres Hobby Dev postgresql-regular-05207

Dyno formation **\$0.00/month** [Configure Dynos](#)

This app is using **free** dynos

web gunicorn authProject.wsgi **ON**

Latest activity [All Activity](#)

- missiontic2022@hotmail.com: Deployed 1d293d70 Sep 23 at 8:01 PM · v5
- missiontic2022@hotmail.com: Build succeeded Sep 23 at 8:00 PM · [View build log](#)
- missiontic2022@hotmail.com: @ref:postgresql-regular-05207 completed provisioning, setting DATABASE_URL. Sep 15 at 9:37 PM · v4

Allí, se debe acceder a la consola de la base de datos, pulsando el botón **Heroku Postgres**:

mission-tic-bank-db ☆ Open app More ↕

Overview Resources Deploy Metrics Activity Access Settings

Free Dynos Change Dyno Type

web gunicorn authProject.wsgi 🔴 \$0.00 ✎

Add-ons Find more add-ons

🔍 Quickly add add-ons from Elements

📦 Heroku Postgres 🔴 Attached as DATABASE ↕ Hobby Dev Free ↕

Estimated Monthly Cost \$0.00

En la consola de debe dirigir a la pestaña [Settings](#):

Databases > postgresql-regular-05207

SERVICE heroku-postgresql PLAN hobby-dev BILLING APP mission-tic-bank-db

Overview Durability Settings 🔴

HEALTH


🟢 Available

PRIMARY Yes VERSION 13.4 CREATED a month ago MAINTENANCE Unsupported ⓘ

ROLLBACK Unsupported ⓘ

Allí se debe pulsar el botón [Reset Database](#):

Datastores > postgresql-regular-05207

SERVICE heroku-postgresql PLAN hobby-dev BILLING APP  mission-tic-bank-db

Overview Durability **Settings** Dataclips

ADMINISTRATION

Database Credentials View Credentials...

Get credentials for manual connections to this database.

Reset Database Reset Database...

Reset the database to its originally-provisioned state, deleting all data inside it.

Destroy Database Destroy Database...

Destroys the database and all of the data inside it.

Una vez pulsado, se solicita ingresar el nombre de la base de datos. Cuando se haga, se pulsa el botón [Reset Database](#):

Reset Database Cancel

Reset the database to its originally-provisioned state, deleting all data inside it.

This action will **permanently destroy all of your data**.

To confirm, please type the name of your app (`mission-tic-bank-db`).

Reset Database

Migraciones de los modelos

Una vez se ha reiniciado la base de datos, es necesario hacer las *migrations*. Para ello, se deben utilizar el comando *python manage.py makemigrations* seguido del comando *python manage.py migrate*:

```
(env) D:\auth_ms>python manage.py makemigrations
Migrations for 'authApp':
  authApp\migrations\0002_delete_account.py
  - Delete model Account
```

```
(env) D:\auth_ms>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, authApp, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0001_initial... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying authApp.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying authApp.0002_delete_account... OK
  Applying sessions.0001_initial... OK
```

Finalmente, para asegurarse que los cambios al proyecto se realizaron adecuadamente se debe ejecutar el servidor del microservicio utilizando el comando *python manage.py runserver*:

```
(env) D:\auth_ms>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
October 25, 2021 - 18:59:05
Django version 3.2.7, using settings 'authProject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
█
```

Se debe recordar que para detener la ejecución del servidor se pulsán las teclas *CTRL+C* al tiempo.

Nota: de ser necesario, en el material de la clase se encuentra un archivo llamado *C4a.AP.01. auth_ms.rar*, con el desarrollo del componente realizado en esta guía.