# Informe de Laboratorio 07

## Tema: Proyecto en la Nube

| Nota |
|---|
| |

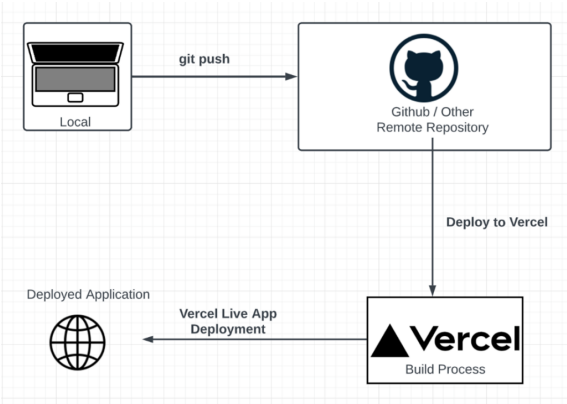
| Estudiante | Escuela | Asignatura |
|---|---|---|
| Bedregal Coaguila, Karla Miluska<br>Llaique Chullunquia, Jack Franco | Escuela Profesional de Ingeniería de Sistemas | Programación Web 2<br>Semestre: III<br>Código: 1702122 |

| Laboratorio | Tema | Duración |
|---|---|---|
| 07 | Proyecto en la Nube | 04 horas |

| Semestre académico | Fecha de inicio | Fecha de entrega |
|---|---|---|
| 2025 - A | Del 17 Junio 2025 | Al 29 Junio 2025 |

## 1. Tarea

Subir el repositorio del proyecto de Programación Web II a la nube utilizando Vercel y Supabase. Realizar pruebas de los métodos GET, POST, PUT y DELETE utilizando SoapUI.

# 2. Equipos, materiales y temas utilizados

- **Visual Studio Code**: Edición del código fuente del proyecto.
- **Overleaf (web)**: Elaboración del informe técnico en LaTeX.
- **Git y GitHub**: Control de versiones y repositorio remoto.
- **Vercel (plan gratuito)**: Despliegue de la API Django en la nube.
- **Supabase**: Base de datos PostgreSQL en la nube.
- **SoapUI**: Pruebas de endpoints GET, POST, PUT y DELETE.
- **Python 3.11**: Lenguaje usado en el backend.
- **Django + Django REST Framework**: Construcción de la API.
- **Windows 11 y Linux**: Sistemas operativos utilizados en el desarrollo.

# 3. URL de Repositorio Github

- **URL Repositorio GitHub**: https://github.com/KarlaBedregal/Cuna-API-unsa.git

# 4. Resolucion

## 4.1. Estructura del Proyecto Cuna-API-unsa

- El contenido que se entrega en este proyecto es el siguiente:

```
Cuna-API-unsa/
 |-- .gitignore
 |-- db.sqlite3
 |-- manage.py
 |-- requirements.txt
 |-- vercel.json
 |-- Apps/
    |-- Cuna/
       |-- admin.py
       |-- apps.py
       |-- models.py.deprecated
       |-- serializers.py
       |-- tests.py
       |-- urls.py
       |-- views.py
       |-- __init__.py
       |-- migrations/
       |-- models/
          |-- Announcement.py
          |-- Course.py
          |-- Grade.py
          |-- Inscription.py
          |-- Proxy.py
          |-- Student.py
          |-- Teacher.py
          |-- Workload.py
          |-- YearCourse.py
          |-- __init__.py
          |-- __pycache__/
       |-- templates/
       |-- __pycache__/
 |-- MyDjangoProject/
    |-- asgi.py
    |-- settings.py
    |-- urls.py
    |-- wsgi.py
    |-- __init__.py
    |-- __pycache__/
 |-- static/
```

## 4.2. Implementacion

### 4.2.1. Settings.py

Listing 1: Clase settings.py

```python
from pathlib import Path
import os
import dj_database_url
# Configuración CORS
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",
    "http://127.0.0.1:3000",
    "http://localhost:8000",
    "http://127.0.0.1:8000",
    "https://cuna-api-unsa-nine.vercel.app",
]
if not DEBUG:
    CORS_ALLOWED_ORIGINS.extend([
        "https://your-project-name.vercel.app",
    ])
CORS_ALLOW_ALL_ORIGINS = DEBUG
CORS_ALLOW_CREDENTIALS = True
CORS_ALLOW_METHODS = [
    'DELETE',
    'GET',
    'OPTIONS',
    'PATCH',
    'POST',
    'PUT',
]
CORS_ALLOW_HEADERS = [
    'accept',
    'accept-encoding',
    'authorization',
    'content-type',
    'dnt',
    'origin',
    'user-agent',
    'x-csrftoken',
    'x-requested-with',
]
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.2/howto/static-files/

STATIC_URL = 'static/'
# Configuración para desarrollo
if DEBUG:
    STATICFILES_DIRS = [
        BASE_DIR / 'static',
    ]
    STATIC_ROOT = BASE_DIR / 'staticfiles'
else:
    # Configuración para producción (Vercel)
    STATIC_ROOT = BASE_DIR / 'staticfiles'
    STATICFILES_DIRS = [
        BASE_DIR / 'static',
    ]
# Configuración de WhiteNoise para archivos estáticos
STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'
if os.environ.get('VERCEL_URL'):
```

```
    ALLOWED_HOSTS.append(os.environ.get('VERCEL_URL'))
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = os.environ.get('DEBUG', 'False').lower() == 'true' # ← Cambiar 'True' por 'False'
# Siempre agregar hosts de producción (más seguro)
ALLOWED_HOSTS.extend([
    '.vercel.app',
    '.now.sh',
    'cuna-api-unsa-nine.vercel.app',
])
# Application definition
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
        },
    },
    'root': {
        'handlers': ['console'],
        'level': 'INFO',
    },
    'loggers': {
        'django': {
            'handlers': ['console'],
            'level': 'INFO',
        },
        'Apps.Cuna': {
            'handlers': ['console'],
            'level': 'DEBUG',
        },
    },
}
# Database
# https://docs.djangoproject.com/en/5.2/ref/settings/databases
if os.environ.get('DATABASE_URL'):
    # Producción - Supabase PostgreSQL
    DATABASES = {
        'default': dj_database_url.parse(os.environ.get('DATABASE_URL'))
    }
    print("      Usando Supabase PostgreSQL")
else:
    # Desarrollo - SQLite local
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': BASE_DIR / 'db.sqlite3',
        }
    }
    print("      Usando SQLite local")

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    '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',
]
```

Modificaciones

- `CORS_ALLOWED_ORIGINS:` Lista de dominios específicos desde los cuales se permiten solicitudes CORS, incluyendo entornos locales y el dominio del proyecto en Vercel.
- `if not DEBUG:` Agrega el dominio de Vercel a los orígenes permitidos (CORS) cuando estás en producción.
- `CORS_ALLOW_ALL_ORIGINS = DEBUG:` Permite todas las solicitudes de origen solo cuando `DEBUG` está activado (modo desarrollo).
- `CORS_ALLOW_CREDENTIALS = True:` Permite el uso de cookies y encabezados de autenticación en las solicitudes CORS.
- `CORS_ALLOW_METHODS:` Define los métodos HTTP permitidos en solicitudes CORS (`GET`, `POST`, `PUT`, etc.).
- `CORS_ALLOW_HEADERS:` Especifica qué encabezados HTTP se permiten en solicitudes CORS.
- `if DEBUG:` En desarrollo, se configuran las rutas de archivos estáticos: `STATICFILES_DIRS` (dónde están los archivos) y `STATIC_ROOT` (dónde se recopilan).
- `STATICFILES_STORAGE:` Usa WhiteNoise para servir archivos estáticos comprimidos y con nombre único (hash) en producción.
- `ALLOWED_HOSTS:` Se añaden los dominios autorizados desde los que Django puede aceptar peticiones, incluyendo dominios genéricos de despliegue como `.vercel.app` y `.now.sh`, así como el dominio específico del proyecto: `cuna-api-unsa-nine.vercel.app`.
- `if os.environ.get('VERCEL_URL'):` Agrega el dominio generado por Vercel a `ALLOWED_HOSTS` si existe la variable de entorno.
- `DEBUG:` Activa o desactiva el modo debug según la variable de entorno. Debe estar en `False` en producción por seguridad.
- `ALLOWED_HOSTS.extend([]):` Agrega los dominios permitidos para producción, incluyendo Vercel y el dominio del proyecto.
- `LOGGING:` Configura el sistema de logs para mostrar mensajes en consola. Usa nivel `INFO` para Django y `DEBUG` para la app `Apps.Cuna`.
- `DATABASES:` Verifica si existe la variable de entorno `DATABASE_URL`. Si está definida, usa Supabase PostgreSQL en producción; de lo contrario, usa SQLite local en desarrollo.
- `MIDDLEWARE:` Lista de capas intermedias que procesan cada solicitud y respuesta. Incluye seguridad, manejo de sesiones, CORS, archivos estáticos (WhiteNoise), autenticación y protección contra ataques comunes como CSRF y clickjacking.

### 4.2.2. Wsgi.py

Listing 2: Clase wsgi.py

```python
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'MyDjangoProject.settings')

application = get_wsgi_application()
# Para Vercel
app = application
```

- `wsgi.py`: Configura la aplicación WSGI para que Django pueda ejecutarse. Se establece el módulo de configuración (`settings.py`) y se asigna la aplicación a `app`, como lo requiere Vercel para el despliegue.

### 4.2.3. Urls.py (App/Cuna)

Listing 3: Clase urls.py

```python
from django.contrib import admin
from django.urls import path, include
from rest_framework.urlpatterns import format_suffix_patterns
from django.urls import path
from .views import StudentListCreateAPIView
from . import views
from .views import student_dashboard
from rest_framework.authtoken.views import obtain_auth_token

urlpatterns = [
    path('', views.APIRootView.as_view(), name='api-root'),
    path('api/test-methods/', views.HTTPMethodsTestView.as_view(), name='test-methods'),

    path('api/initialize/', views.InitializeDataView.as_view(), name='initialize-data'),
    .....
]
```

**Modificaciones**

- `path(", APIRootView)`: Ruta raíz de la API. Devuelve una vista general de los endpoints disponibles.
- `path('api/test-methods/', HTTPMethodsTestView)`: Ruta para probar los distintos métodos HTTP (GET, POST, PUT, DELETE, etc.).
- `path('api/initialize/', InitializeDataView)`: Ruta que inicializa datos base en la aplicación, útil para pruebas o configuración inicial.

### 4.2.4. Views.py (App/Cuna)

Listing 4: Clase views.py

```python
class APIRootView(APIView):
    """Vista de bienvenida para la API"""
    permission_classes = [AllowAny]

    def get(self, request):
        return Response({
            'success': True,
            'message': 'Bienvenido a la API de CUNA UNSA',
            'version': '1.0.0',
            'endpoints': {
                'estudiantes': '/api/students/',
                'docentes': '/api/teachers/',
                'cursos': '/api/courses/',
                'cargas_academicas': '/api/workloads/',
                'inscripciones': '/api/inscriptions/',
                'notas': '/api/grades/',
                'autenticacion': {
                    'registro': '/api/auth/register/',
                    'login': '/api/auth/login/',
                    'perfil': '/api/auth/profile/',
                    'estado': '/api/auth/status/'
                },
                'usuarios': '/api/users/',
                'admin': '/admin/'
            },
            'documentation': 'https://cuna-api-unsa-nine.vercel.app/api/',
            'timestamp': datetime.now().isoformat()
        })

class InitializeDataView(APIView):
    """Vista para inicializar datos en produccin"""
    permission_classes = [AllowAny]

    def post(self, request):
        try:
            from django.contrib.auth.models import User
            from rest_framework.authtoken.models import Token
            from Apps.Cuna.models.Student import Student
            from Apps.Cuna.models.Course import Course

            results = []

            # Verificar conexión a base de datos
            from django.db import connection
            with connection.cursor() as cursor:
                cursor.execute("SELECT version()")
                db_version = cursor.fetchone()[0]
                results.append(f" Conexin BD exitosa: {db_version}")

            # Crear superusuario si no existe
            if not User.objects.filter(username='admin').exists():
                admin_user = User.objects.create_superuser(
```

```
                    username='admin',
                    email='admin@cunaunsa.com',
                    password='AdminCuna2025'
                )
                results.append(f" Admin creado: {admin_user.username}")
            else:
                results.append(" Admin ya existe")


            return Response({
                'success': True,
                'message': ' Inicializacin de datos completada',
                'results': results,
                'timestamp': datetime.now().isoformat()
            })

        except Exception as e:
            return Response({
                'success': False,
                'message': f' Error durante inicializacin: {str(e)}',
                'error_type': type(e).__name__,
                'timestamp': datetime.now().isoformat()
            }, status=status.HTTP_500_INTERNAL_SERVER_ERROR)

class HTTPMethodsTestView(APIView):
    """Vista para probar todos los mtodos HTTP en Vercel"""
    permission_classes = [AllowAny]

    def get(self, request):
        return Response({
            'method': 'GET',
            'success': True,
            'message': ' GET funciona correctamente',
            'timestamp': datetime.now().isoformat()
        })

    def post(self, request):
        return Response({
            'method': 'POST',
            'success': True,
            'message': ' POST funciona correctamente',
            'data_received': request.data,
            'timestamp': datetime.now().isoformat()
        })

    def put(self, request):
        return Response({
            'method': 'PUT',
            'success': True,
            'message': ' PUT funciona correctamente en Vercel',
            'data_received': request.data,
            'timestamp': datetime.now().isoformat()
        })

    def patch(self, request):
        return Response({
```

```
            'method': 'PATCH',
            'success': True,
            'message': ' PATCH funciona correctamente en Vercel',
            'data_received': request.data,
            'timestamp': datetime.now().isoformat()
        })

    def delete(self, request):
        return Response({
            'method': 'DELETE',
            'success': True,
            'message': ' DELETE funciona correctamente en Vercel',
            'timestamp': datetime.now().isoformat()
        })
```

### Modificaciones

- `APIRootView`: Vista principal de bienvenida de la API. Hereda de `APIView` y no requiere autenticación (`AllowAny`).
- `get(self, request)`: Método que responde a solicitudes GET mostrando un mensaje de bienvenida, la versión de la API, rutas disponibles y la hora actual.
- `Response({...})`: Devuelve un diccionario con información estructurada sobre los endpoints públicos y de autenticación de la API.
- `InitializeDataView`: Vista basada en clase (`APIView`) que permite ejecutar una inicialización de datos mediante una solicitud `POST`. Se utiliza principalmente en producción.
- `permission_classes = [AllowAny]`: La vista es pública y no requiere autenticación para ser accedida.
- `post(self, request)`: Método que ejecuta las operaciones de inicialización cuando se recibe una solicitud POST.
- `Verificación de base de datos`: Usa un cursor SQL para comprobar que la conexión a la base de datos funciona correctamente.
- `Creación de superusuario`: Si no existe un usuario con nombre `'admin'`, se crea un superusuario con credenciales predefinidas.
- `results`: Lista donde se almacenan los mensajes de éxito o advertencia que se devuelven en la respuesta.
- `try-except`: Captura cualquier error que ocurra durante la ejecución y devuelve un mensaje con el tipo de excepción y el estado HTTP 500.
- `datetime.now().isoformat()`: Se utiliza para incluir una marca de tiempo exacta en la respuesta.
- `HTTPMethodsTestView`: Vista basada en `APIView` que permite verificar el funcionamiento de todos los métodos HTTP en el entorno de Vercel.
- `permission_classes = [AllowAny]`: Permite el acceso sin autenticación, útil para pruebas libres de la API.
- `get(self, request)`: Responde a solicitudes GET indicando que el método funciona correctamente, junto con una marca de tiempo.

- `post(self, request)`: Responde a solicitudes POST con los datos recibidos y un mensaje de confirmación.
- `put(self, request)`: Verifica que el método PUT esté habilitado, devolviendo los datos enviados y una confirmación.
- `patch(self, request)`: Comprueba el funcionamiento del método PATCH, mostrando también los datos recibidos.
- `delete(self, request)`: Responde a solicitudes DELETE con un mensaje de éxito.
- `datetime.now().isoformat()`: Incluye la hora exacta de cada respuesta en formato ISO.

### 4.2.5. Vercel.json

Listing 5: Clase vercel.json

```
{
    "version": 2,
    "builds": [
        {
            "src": "MyDjangoProject/wsgi.py",
            "use": "@vercel/python",
            "config": {
                "maxLambdaSize": "15mb",
                "runtime": "python3.9"
            }
        }
    ],
    "routes": [
        {
            "src": "/static/(.*)",
            "dest": "/static/$1"
        },
        {
            "src": "/(.*)",
            "dest": "MyDjangoProject/wsgi.py",
            "methods": ["GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"]
        }
    ],
    "env": {
        "PYTHONPATH": ".",
        "DJANGO_SETTINGS_MODULE": "MyDjangoProject.settings"
    }
}
```

**Modificaciones**

- `vercel.json`: Archivo de configuración para desplegar un proyecto Django en Vercel usando Python.
- `"builds"`: Define el punto de entrada del proyecto (`wsgi.py`) y especifica que se usará el runtime `python3.9` con un límite de 15MB por Lambda.
- `routes`: Configura dos rutas:

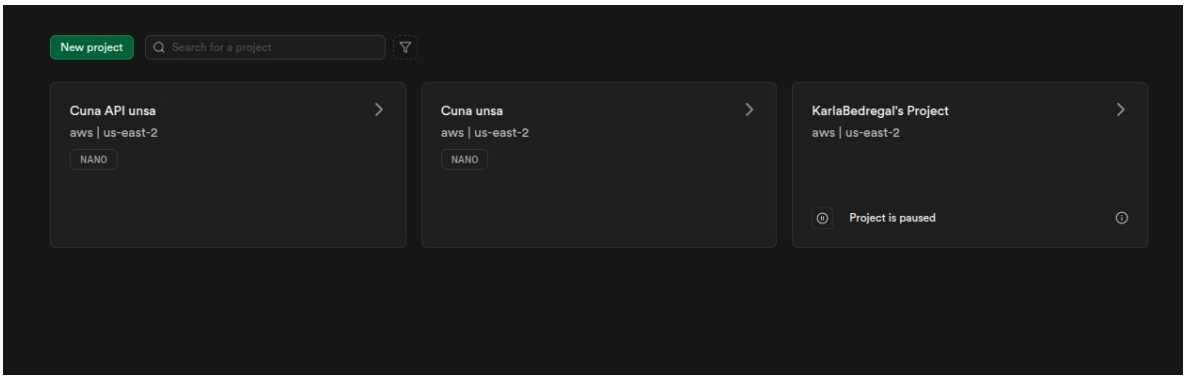
- `/static/(.*)`: Redirige archivos estáticos a la carpeta correspondiente.
- `/(.*)`: Redirige todas las demás solicitudes a `wsgi.py`, permitiendo métodos como GET, POST, PUT, DELETE, PATCH y OPTIONS.

- `.env`: Define variables de entorno necesarias para que Django funcione en Vercel, como `PYTHONPATH` y `DJANGO_SETTINGS_MODULE`.

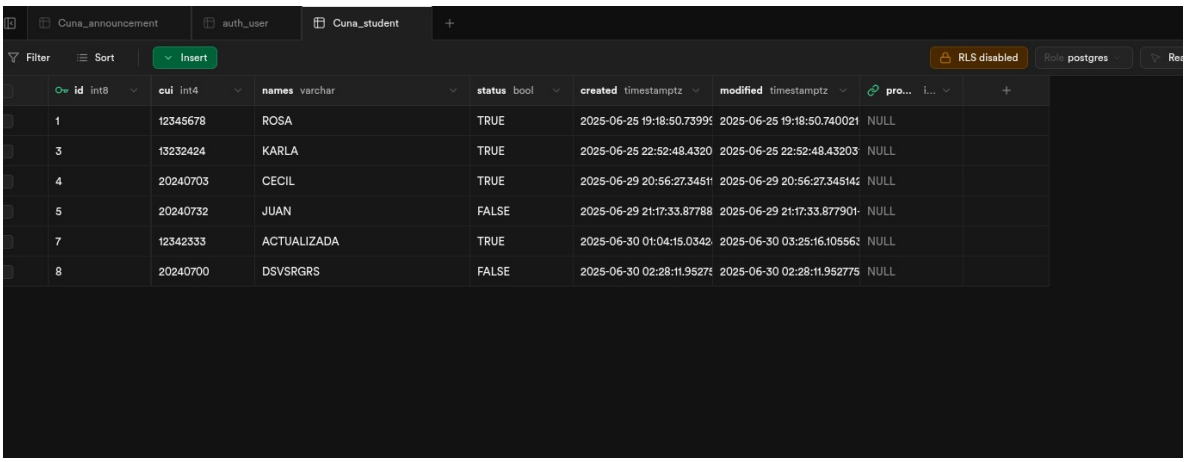
# 5. Ejecucion

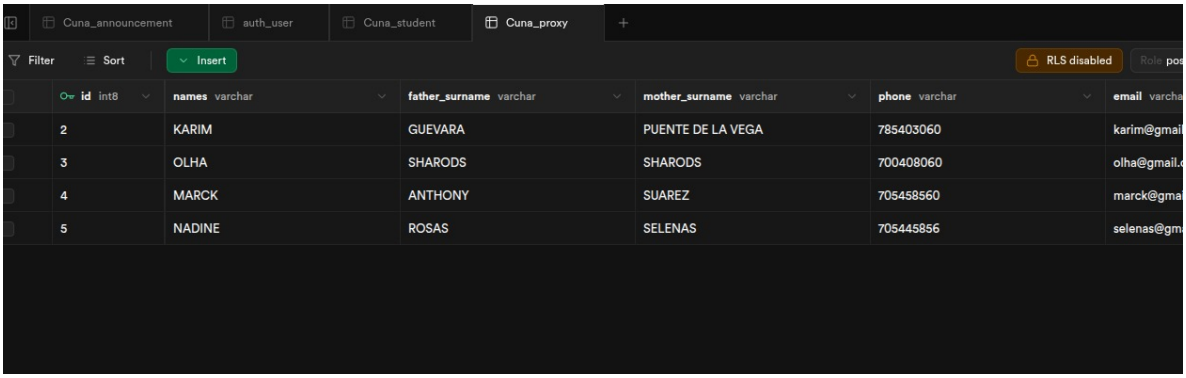
## 5.1. Supabase

- Se utiliza **Supabase** como base de datos en producción, conectándose mediante la variable de entorno `DATABASE_URL`.
- Para la autenticación, se emplea el sistema propio de usuarios de **Django**, junto con `TokenAuthentication` del **Django REST Framework**.
- Se creó un proyecto en **Supabase**, el cual proporciona una base de datos PostgreSQL gestionada en la nube.

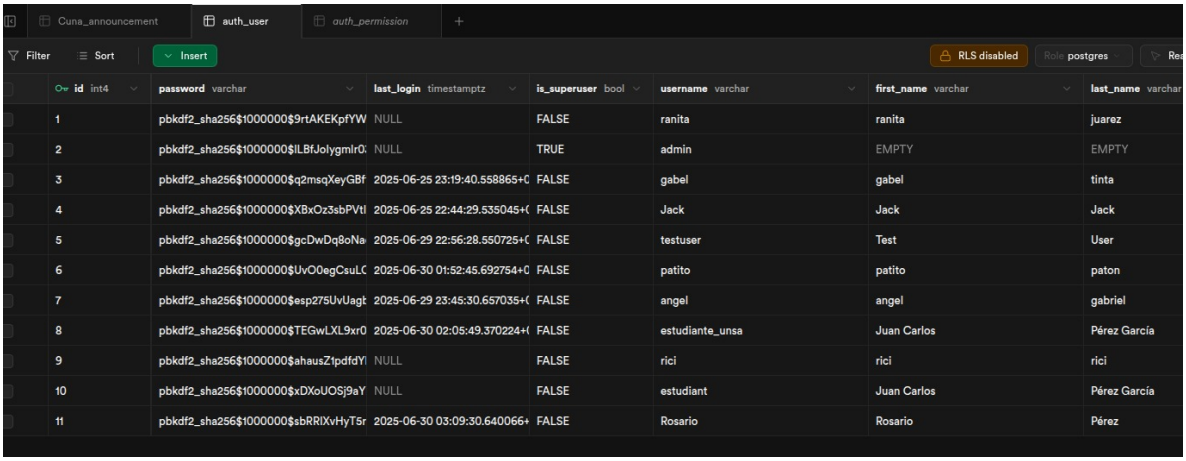



- La conexión con Supabase se realizó utilizando la variable de entorno `DATABASE_URL`, la cual se integró en el archivo `settings.py` mediante la librería `dj_database_url`.



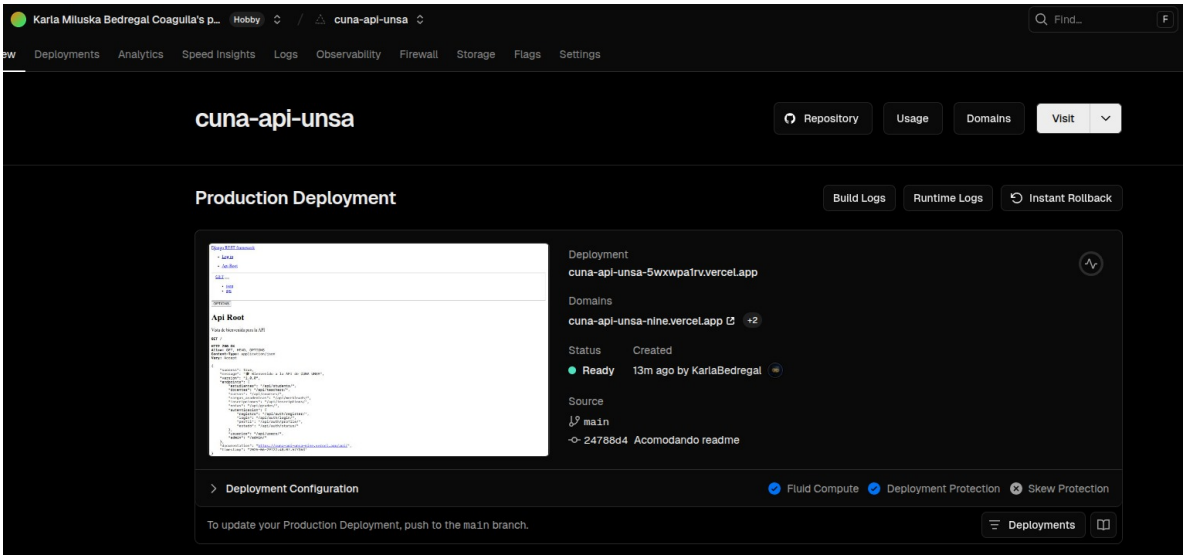



- La conexión con **Supabase** se logró correctamente y permitió visualizar las tablas creadas desde Django, como por ejemplo las tablas de `estudiantes`, `usuarios` y `apoderados`, directamente en la interfaz web de Supabase.

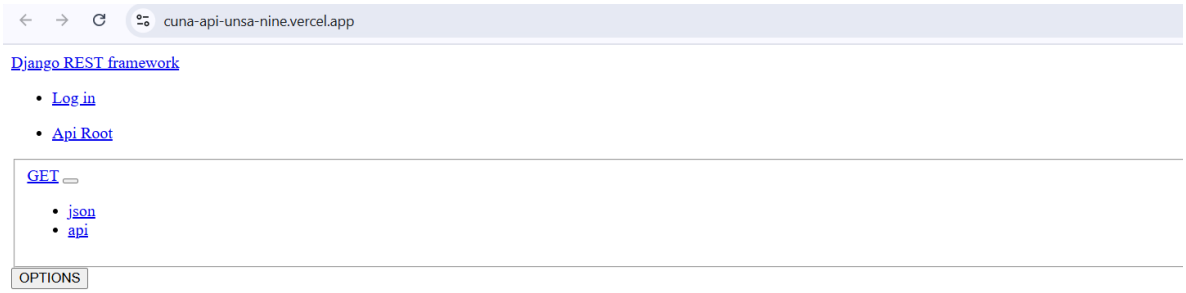



## 5.2. Vercel

- El despliegue en **Vercel** se realizó vinculando directamente el repositorio del proyecto alojado en **GitHub**, lo que permite integrar automáticamente los cambios mediante cada `push` o actualización del repositorio.




- Se configuró Vercel con las rutas necesarias, el archivo `wsgi.py` como punto de entrada, y las variables de entorno requeridas.
- Gracias a esta configuración, la API se ejecuta correctamente y es accesible desde una URL pública.




# Api Root

Vista de bienvenida para la API

**GET /**

**HTTP 200 OK**
**Allow:** GET, HEAD, OPTIONS
**Content-Type:** application/json
**Vary:** Accept


```
{
    "success": true,
    "message": "🎓 Bienvenido a la API de CUNA UNSA",
    "version": "1.0.0",
    "endpoints": {
        "estudiantes": "/api/students/",
        "docentes": "/api/teachers/",
        "cursos": "/api/courses/",
        "cargas_academicas": "/api/workloads/",
        "inscripciones": "/api/inscriptions/",
        "notas": "/api/grades/",
        "autenticacion": {
            "registro": "/api/auth/register/",
            "login": "/api/auth/login/",
            "perfil": "/api/auth/profile/",
            "estado": "/api/auth/status/"
        },
```


- Este es el enlace al video explicativo donde se muestra cómo utilizar **Vercel** del proyecto: https://youtu.be/Q7MpwKug-co

- Puedes acceder directamente a la Pagina desde el siguiente enlace público: https://cuna-api-unsa-nine.vercel.app/

# 6. Rúbricas

## 6.1. Entregable Informe

Tabla 1: Tipo de Informe

| **Informe** | |
|---|---|
| **Latex** | El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer. |

## 6.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

| | Nivel | | | |
|---|---|---|---|---|
| **Puntos** | Insatisfactorio 25 % | En Proceso 50 % | Satisfactorio 75 % | Sobresaliente 100 % |
| **2.0** | 0.5 | 1.0 | 1.5 | 2.0 |
| **4.0** | 1.0 | 2.0 | 3.0 | 4.0 |

Tabla 3: Rúbrica para contenido del Informe y demostración

| Contenido y demostración | | Puntos | Checklist | Estudiante | Profesor |
|---|---|---|---|---|---|
| **1. GitHub** | Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar. | 2 | X | 2 | |
| **2. Commits** | Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación). | 4 | X | 3 | |
| **3. Código fuente** | Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones. | 2 | X | 1 | |
| **4. Ejecución** | Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente. | 2 | X | 2 | |
| **5. Pregunta** | Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación). | 2 | X | 1 | |
| **6. Fechas** | Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos. | 2 | X | 2 | |
| **7. Ortografía** | El documento no muestra errores ortográficos. | 2 | X | 2 | |
| **8. Madurez** | El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación). | 4 | X | 4 | |
| **Total** | | 20 | | 17 | |

# 7. Referencias


- W3Schools. *Python Reference*. `https://www.w3schools.com/python/python_reference.asp`
- Python Software Foundation. *The Python Tutorial*. `https://docs.python.org/3/tutorial/`
- Real Python. *ASCII Art: Converting Images to Text*. `https://realpython.com/python-ascii-art/`
- Python Software Foundation. *Python Standard Library - Modules*. `https://docs.python.org/3/library/`
- The Python Packaging Guide. *Installing Python Modules*. `https://packaging.python.org/en/latest/tutorials/installing-packages/`
- Django Software Foundation. *Model Field Reference*. `https://docs.djangoproject.com/en/stable/ref/models/fields/`
- Django Software Foundation. *Validation in Django*. `https://docs.djangoproject.com/en/stable/ref/validators/`
- Python Software Foundation. *Python 3 Documentation*. `https://docs.python.org/3/`
- Real Python. *How to Use Django Models*. `https://realpython.com/django-models-python-web-applications/`
- Mozilla Developer Network (MDN). *Web Development with Django*. `https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django`
- Simple is Better Than Complex. *Django Best Practices*. `https://simpleisbetterthancomplex.com/`