



# django



@SENAComunica

[www.sena.edu.co](http://www.sena.edu.co)

# Integración Django - React

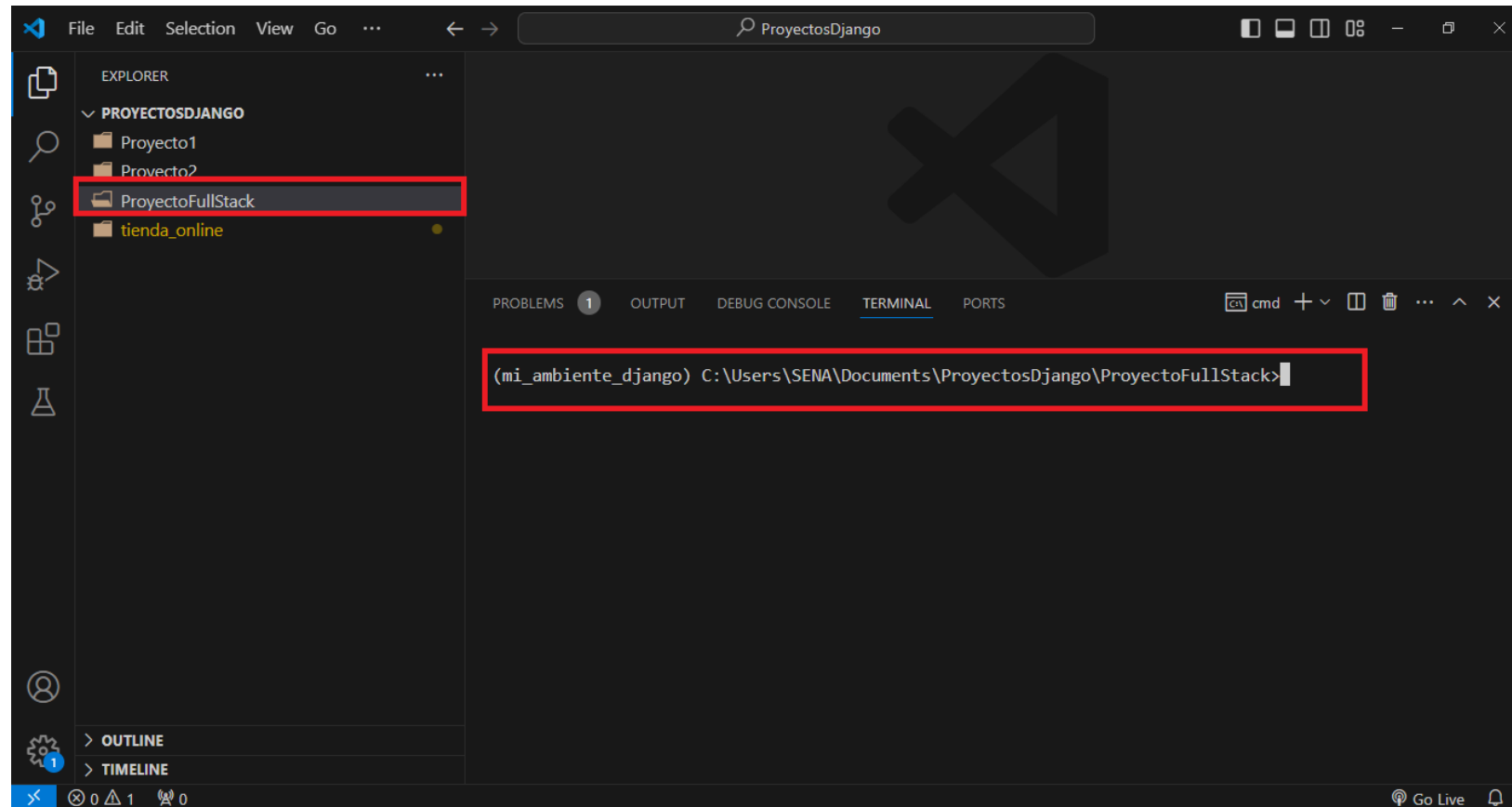


# BackEnd en Django



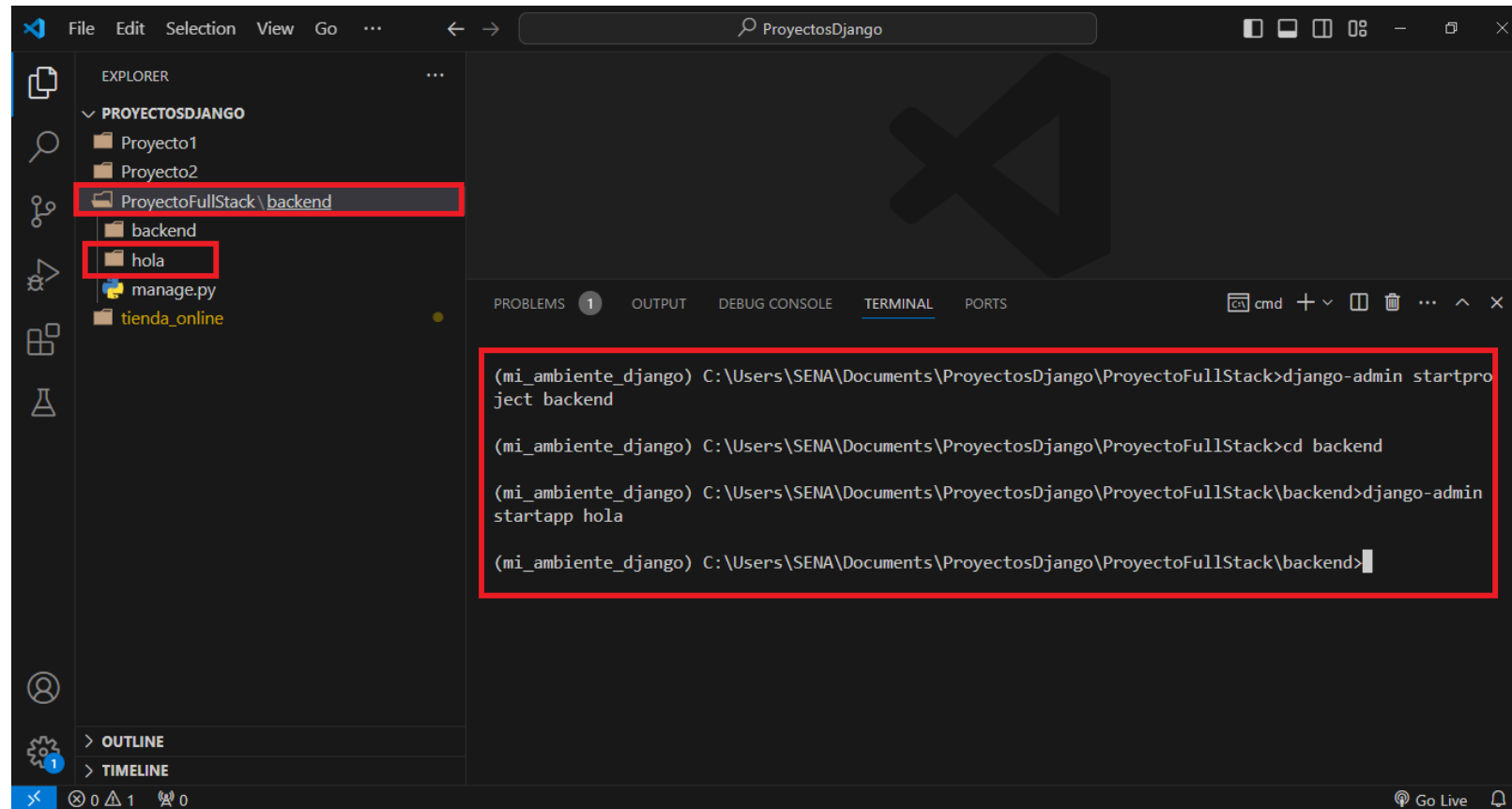
# Crear una Nueva Carpeta de Proyecto

Cree una nueva carpeta de proyecto llamada **ProyectoFullStack** e ingresar a esta carpeta y al ambiente virtual



# Crear Proyecto y App

Cree un nuevo proyecto llamado **backend**, luego ingrese a la carpeta del proyecto y cree una app llamada hola



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left. The file tree shows a project structure with folders 'Proyecto1', 'Proyecto2', and 'ProyectoFullStack'. Inside 'ProyectoFullStack', there are subfolders 'backend' and 'hola', and files 'manage.py' and 'tienda\_online'. The 'backend' and 'hola' folders are highlighted with red boxes. The Terminal panel at the bottom shows the following commands and their output:

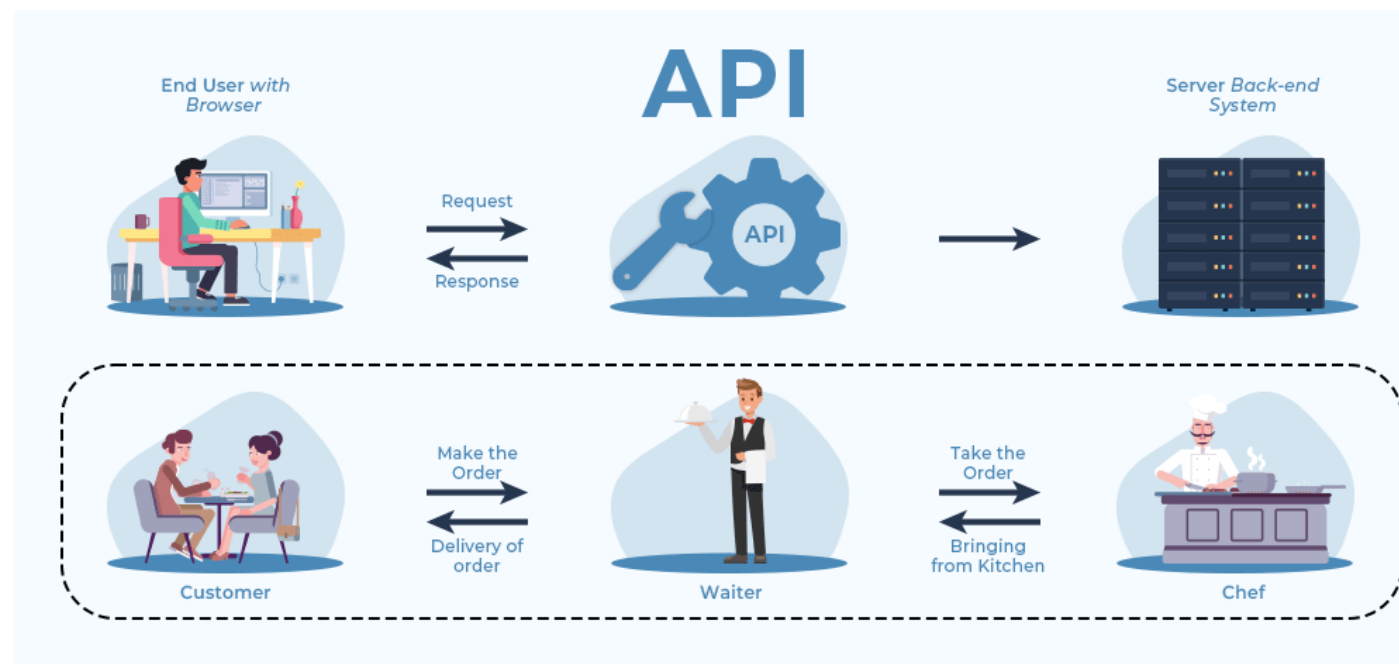
```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack>django-admin startproject backend  
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack>cd backend  
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack\backend>django-admin startapp hola  
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack\backend>
```

# Conceptos previos



# API

Una API (Interfaz de Programación de Aplicaciones, por sus siglas en inglés) es un conjunto de definiciones y protocolos que permiten que diferentes aplicaciones se comuniquen entre sí. Actúa como un intermediario que permite a los desarrolladores interactuar con servicios externos o sistemas sin tener que conocer los detalles internos de cómo están implementados.



# API Conceptos Claves

- El cliente inicia las solicitudes a través del URI (Identificador uniforme de recursos) de la API.
- La API realiza una llamada al servidor después de recibir la solicitud.
- Luego el servidor envía la respuesta a la API con la información
- Finalmente, la API transfiere los datos al cliente.



## 1. Interfaz:

- Las APIs actúan como una interfaz entre diferentes sistemas de software.
- Facilitan la integración y la comunicación.

## 2. Solicitudes y Respuestas:

- Las APIs funcionan a través de solicitudes HTTP (GET, POST, PUT, DELETE).
- Las solicitudes se envían a un servidor que procesa y devuelve una respuesta.

## 3. Endpoints:

- Puntos de acceso a la API donde se pueden realizar operaciones específicas.
- Cada endpoint representa una ruta que maneja una función particular.

## 4. Formatos de Datos:

- APIs suelen intercambiar datos en formatos como JSON o XML.
- JSON es el formato más común debido a su simplicidad y legibilidad.



# EndPoints

En el contexto del desarrollo de aplicaciones web y APIs, un endpoint es una URL específica a la que se puede enviar una solicitud para interactuar con el servidor. Los endpoints definen las rutas a través de las cuales los clientes (como aplicaciones web, móviles u otros servicios) pueden acceder a los recursos y funcionalidades expuestos por el backend alojado en un servidor. Cada endpoint corresponde a una función o método específico en el backend que maneja una solicitud HTTP.

## Características de un Endpoint

URL: La dirección donde el servidor escucha las solicitudes. Por ejemplo, <http://localhost:8000/api/users/>.

Método HTTP: El tipo de solicitud que se envía al servidor, como GET, POST, PUT, DELETE, etc. Cada método tiene un propósito específico: GET: Solicitar datos. POST: Enviar datos para crear un nuevo recurso. PUT: Enviar datos para actualizar un recurso existente. DELETE: Eliminar un recurso.

# Django Rest Framework

Django REST Framework (DRF) es una biblioteca para construir APIs web en Django. Proporciona herramientas y funcionalidades que facilitan la creación de APIs RESTful, lo que permite que las aplicaciones web se comuniquen con otras aplicaciones o servicios a través de HTTP.

## Características:

- **Serialización:** DRF facilita la conversión de datos complejos, como consultas de bases de datos, en formatos de datos nativos de Python que se pueden renderizar fácilmente en JSON o XML.
- **Autenticación y Permisos:** Incluye soporte para autenticación y autorización, permitiendo la creación de sistemas seguros.



# Django Rest Framework

- Vistas y Endpoints: Ofrece vistas genéricas que simplifican la creación de endpoints de API.
- Navegación de API: Proporciona una interfaz de usuario web para explorar y probar la API.
- Validación de Datos: Maneja la validación de datos de entrada, asegurando que los datos recibidos sean correctos.



# Django Rest Framework Decoradores

Los decoradores en Django REST Framework se utilizan para modificar o extender el comportamiento de las vistas basadas en funciones. Aquí están algunos de los más importantes:

1. `@api_view`: Este decorador convierte una función regular de Django en una vista que puede manejar solicitudes de API. Permite especificar los métodos HTTP permitidos.
2. `@permission_classes`: Este decorador se utiliza para especificar las clases de permisos que deben aplicarse a una vista. Los permisos determinan si una solicitud tiene acceso permitido a la vista.

# CORS (Cross-Origin Resource Sharing)

## intercambio de recursos entre orígenes

CORS es una característica de seguridad implementada por los navegadores web para controlar el acceso a recursos en un origen (dominio) diferente. Cuando su frontend (React) y backend (Django) están en dominios diferentes, el navegador puede bloquear las solicitudes por razones de seguridad.

### ¿Qué es CORS?

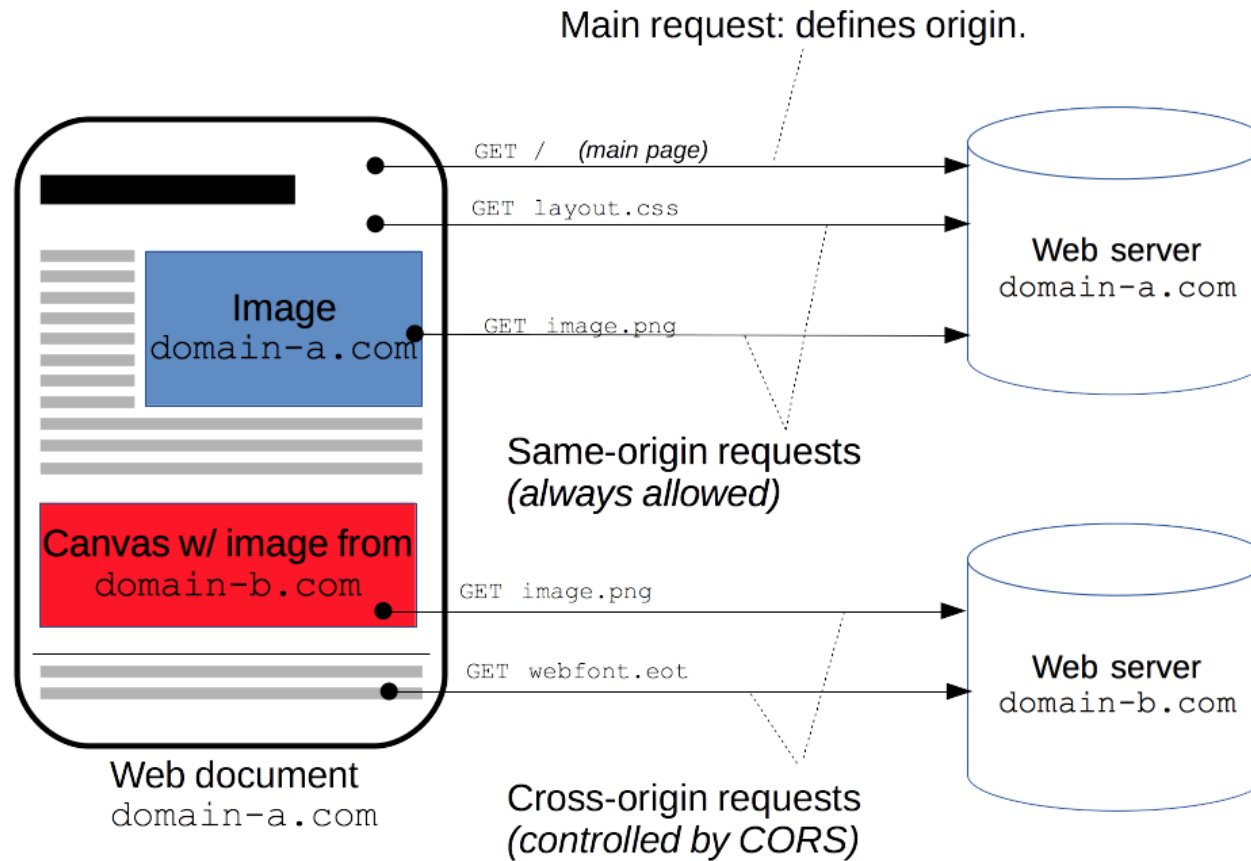
El intercambio de recursos entre orígenes (CORS) es un mecanismo que permite o restringe las aplicaciones web que se ejecutan en un origen para realizar solicitudes de recursos de un origen diferente.

### ¿Por qué es importante CORS?

Las aplicaciones web modernas suelen implicar solicitudes entre diferentes dominios. CORS garantiza que estas solicitudes de origen cruzado solo se permitan si el servidor lo permite explícitamente. Esta es una característica de seguridad crucial para evitar el acceso no autorizado a los recursos.

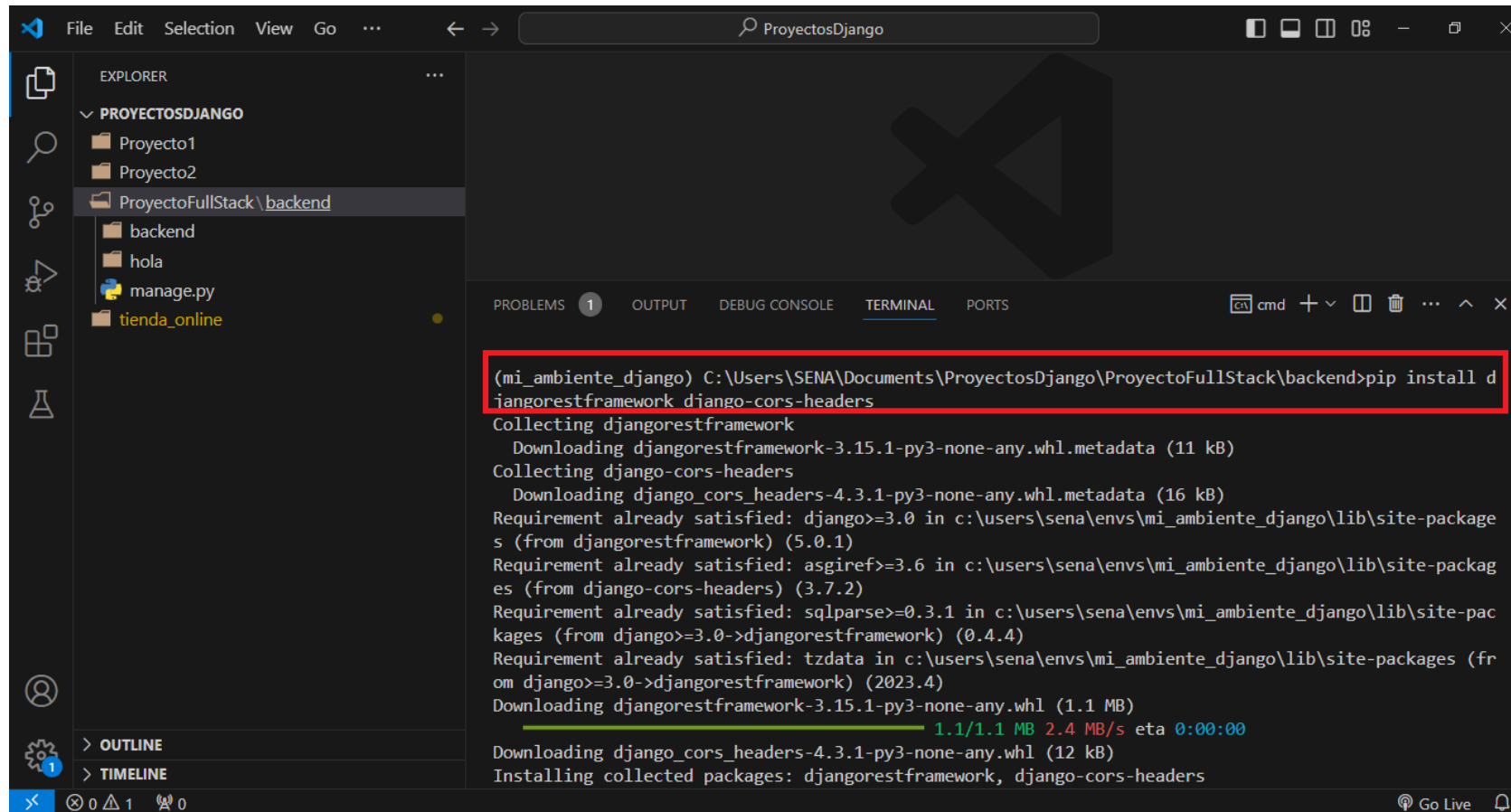
# CORS (Cross-Origin Resource Sharing)

intercambio de recursos entre orígenes



# Instalar los paquetes (App) necesarios

Una vez creados el proyecto y la aplicación, necesitamos instalar algunos paquetes. Para Django , necesitamos instalar Django Rest Framework y Django Cors Headers . Realiza un **pip list** para verificar que se instalaron los paquetes requeridos



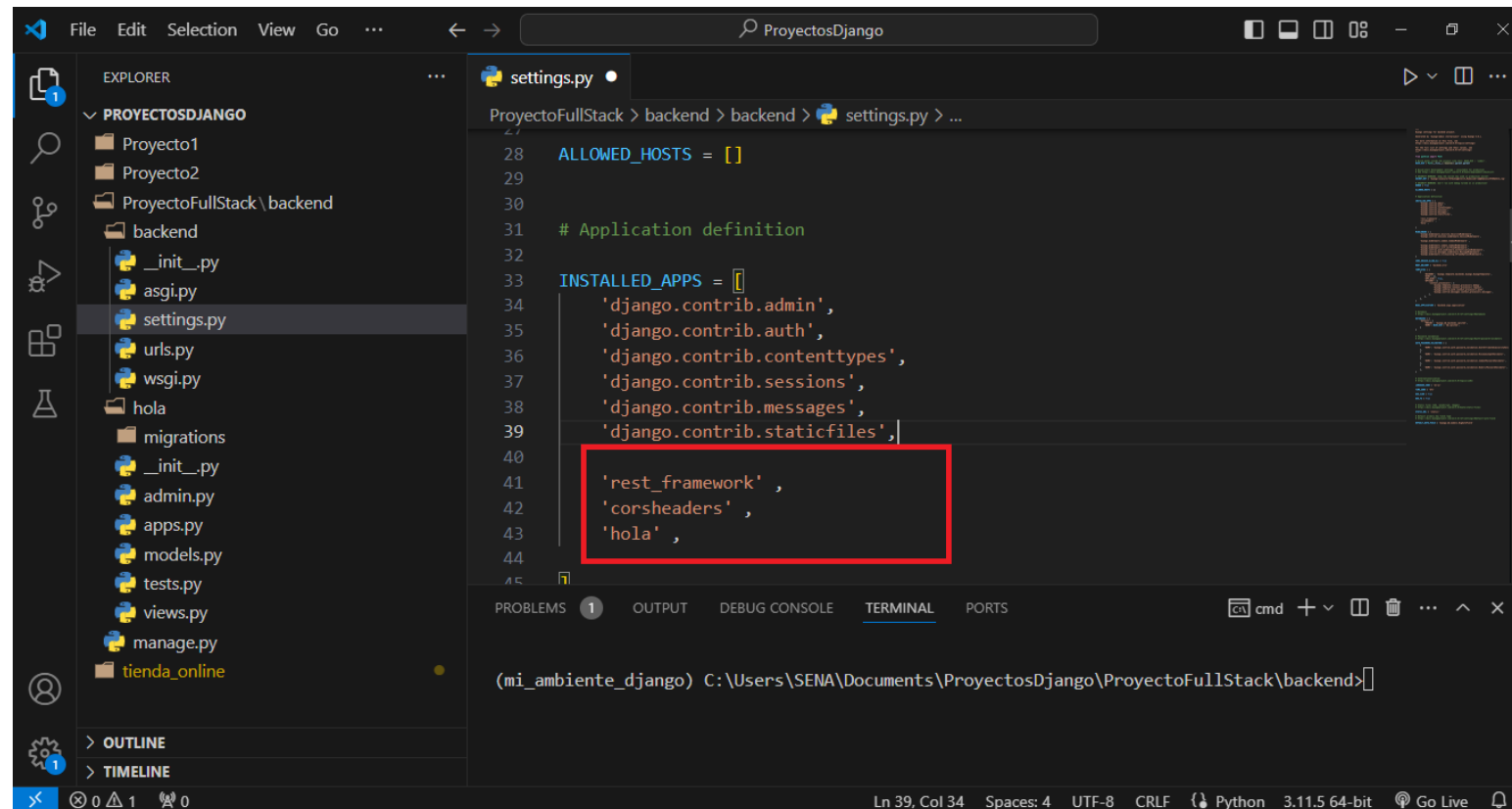
```

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack\backend>pip install d
jangorestframework django-cors-headers
Collecting djangorestframework
  Downloading djangorestframework-3.15.1-py3-none-any.whl.metadata (11 kB)
Collecting django-cors-headers
  Downloading django_cors_headers-4.3.1-py3-none-any.whl.metadata (16 kB)
Requirement already satisfied: django>=3.0 in c:\users\sena\envs\mi_ambiente_django\lib\site-package
s (from djangorestframework) (5.0.1)
Requirement already satisfied: asgiref>=3.6 in c:\users\sena\envs\mi_ambiente_django\lib\site-packag
es (from django-cors-headers) (3.7.2)
Requirement already satisfied: sqlparse>=0.3.1 in c:\users\sena\envs\mi_ambiente_django\lib\site-pac
kages (from django>=3.0->djangorestframework) (0.4.4)
Requirement already satisfied: tzdata in c:\users\sena\envs\mi_ambiente_django\lib\site-packages (fr
om django>=3.0->djangorestframework) (2023.4)
Downloading djangorestframework-3.15.1-py3-none-any.whl (1.1 MB)
1.1/1.1 MB 2.4 MB/s eta 0:00:00
Downloading django_cors_headers-4.3.1-py3-none-any.whl (12 kB)
Installing collected packages: djangorestframework, django-cors-headers
  
```

Package	Version
-----	-----
asgiref	3.7.2
Django	5.0.1
django-cors-headers	4.3.1
djangorestframework	3.15.1
pip	24.0
psycpg2	2.9.9
setuptools	69.0.2
sqlparse	0.4.4
tzdata	2023.4
wheel	0.42.0

# Registrar los paquetes (App) instalados y la app creada

A continuación, abra el archivo settings.py dentro del directorio de proyecto **backend** y agregue las siguientes líneas de código:



```
28 ALLOWED_HOSTS = []
29
30
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40
41     'rest_framework',
42     'corsheaders',
43     'hola',
44 ]
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

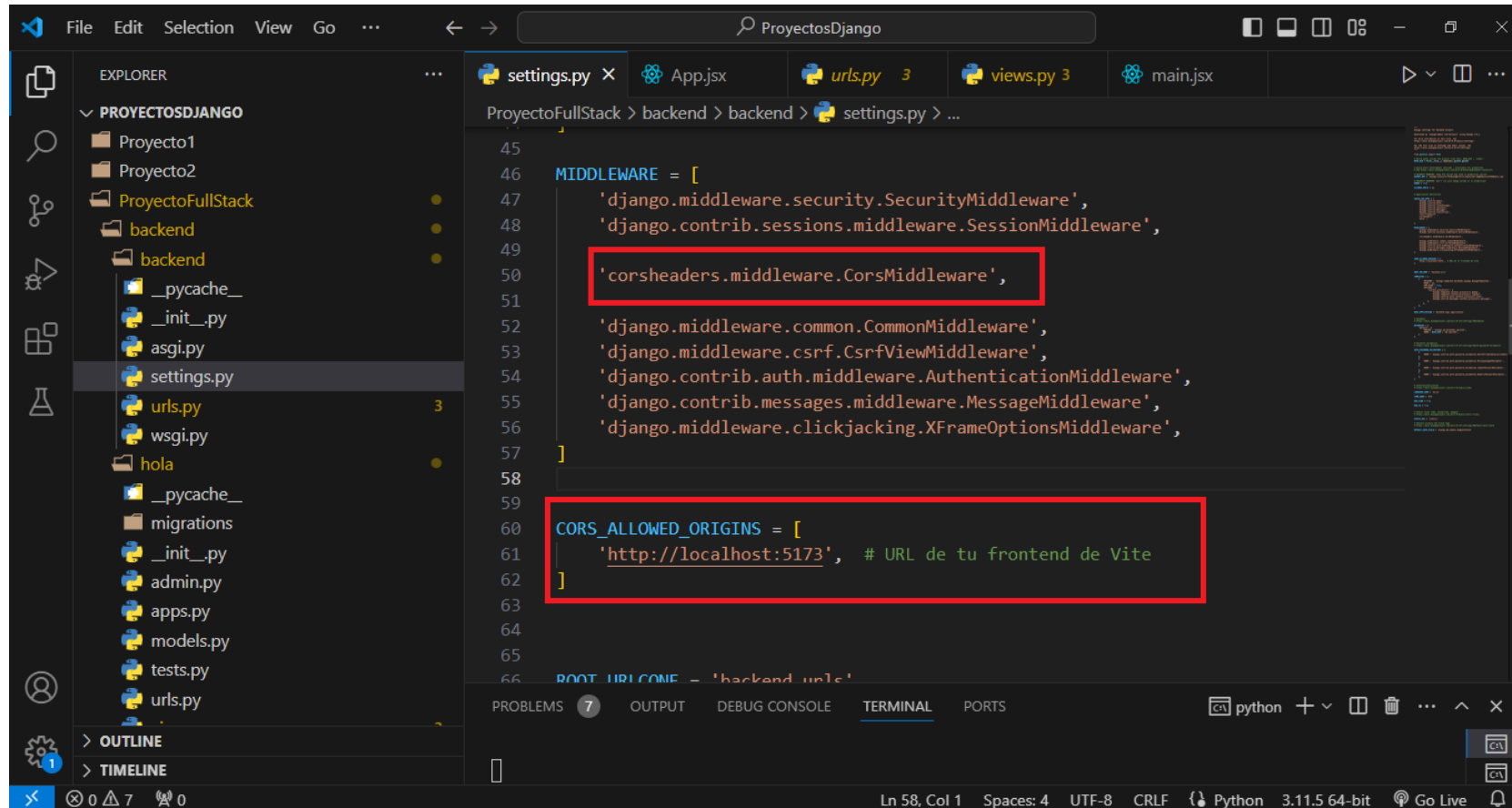
(mi\_ambiente\_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack\backend>

Ln 39, Col 34 Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit Go Live



# Registrar los paquetes (App) instalados y la app creada

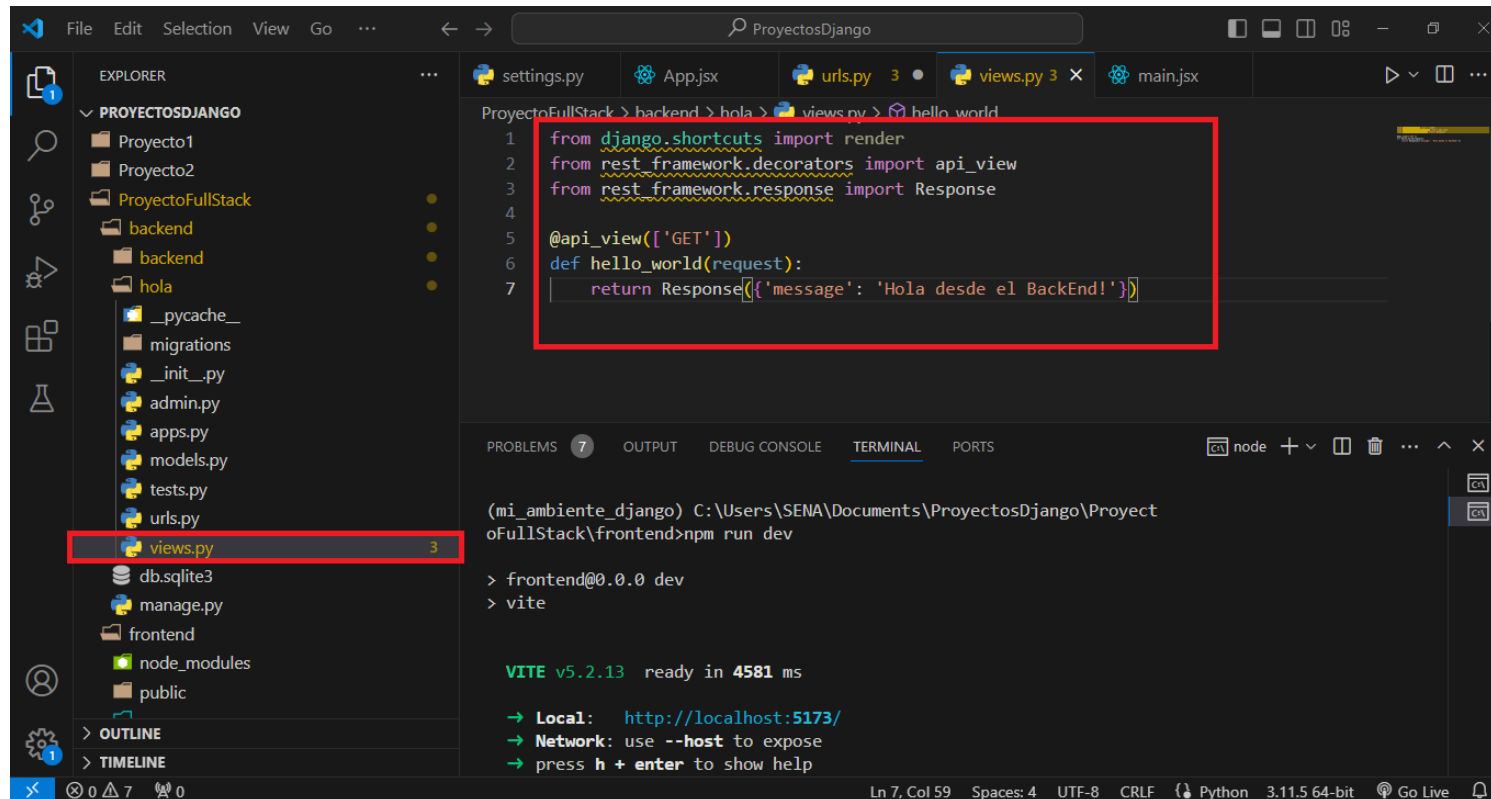
A continuación, abra el archivo `settings.py` dentro del directorio de proyecto **backend** y agregue las siguientes líneas de código:



```
45
46 MIDDLEWARE = [
47     'django.middleware.security.SecurityMiddleware',
48     'django.contrib.sessions.middleware.SessionMiddleware',
49     'corsheaders.middleware.CorsMiddleware',
50
51     'django.middleware.common.CommonMiddleware',
52     'django.middleware.csrf.CsrfViewMiddleware',
53     'django.contrib.auth.middleware.AuthenticationMiddleware',
54     'django.contrib.messages.middleware.MessageMiddleware',
55     'django.middleware.clickjacking.XFrameOptionsMiddleware',
56 ]
57
58
59
60 CORS_ALLOWED_ORIGINS = [
61     'http://localhost:5173', # URL de tu frontend de Vite
62 ]
63
64
65
66 ROOT_URLCONF = 'backend.urls'
```

# Crear los EndPoints de la API

Creamos la vista `hello_world` en `views.py` de la aplicación `hola`



The screenshot shows the Visual Studio Code interface with a Django project named 'ProyectosDjango'. The Explorer panel on the left shows the project structure, with the 'views.py' file in the 'hola' app highlighted. The main editor shows the content of 'views.py', which defines a new API endpoint named 'hello\_world'. The code is as follows:

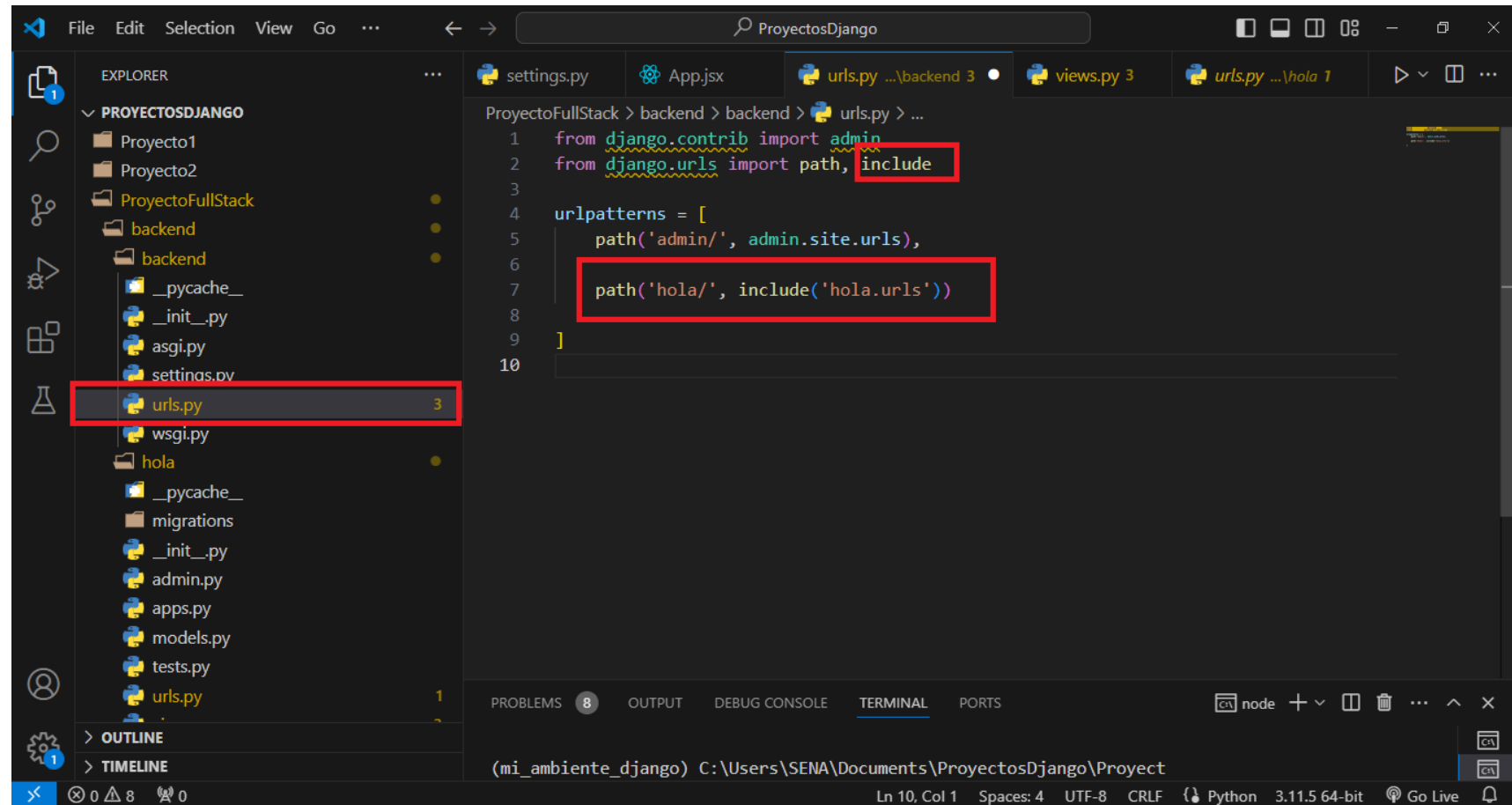
```
1 from django.shortcuts import render
2 from rest_framework.decorators import api_view
3 from rest_framework.response import Response
4
5 @api_view(['GET'])
6 def hello_world(request):
7     return Response({'message': 'Hola desde el BackEnd!'})
```

The bottom panel shows the terminal output, indicating that the frontend is running on port 5173 and the Django backend is running on port 5173.

Este código define un nuevo EndPoint de la API que devuelve una respuesta JSON con el mensaje "Hola desde el backend".

# Crear la URLS

Creamos la url en **urls.py** del proyecto **backend**



The screenshot shows the Visual Studio Code interface with the Django project 'ProyectosDjango' open. The Explorer panel on the left shows the project structure, with the 'urls.py' file in the 'backend' directory highlighted. The main editor shows the content of 'urls.py', which includes imports for 'admin' and 'include' from 'django.urls', and a list of URL patterns. The 'include' function is used to include the 'hola.urls' module.

```

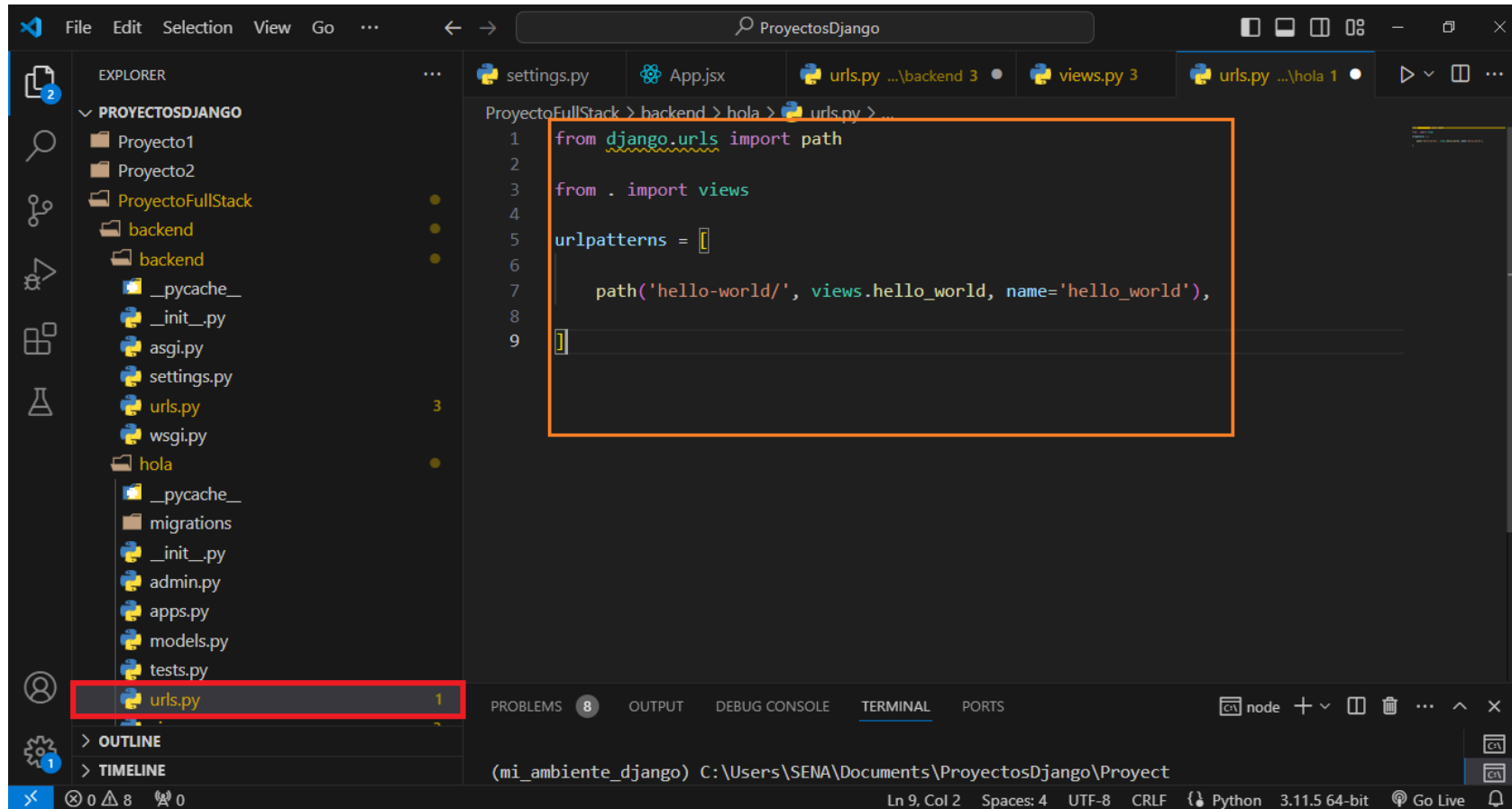
1 from django.contrib import admin
2 from django.urls import path, include
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path('hola/', include('hola.urls'))
7 ]
8
9
10

```

The status bar at the bottom indicates the current file is 'urls.py' at line 10, column 1, with 4 spaces, UTF-8 encoding, and CRLF line endings. The Python version is 3.11.5 64-bit.

# Crear la URLS

Creamos el archivo `urls.py` en la plicación **hola** del proyecto **backend**



The screenshot shows the Visual Studio Code interface with the following details:

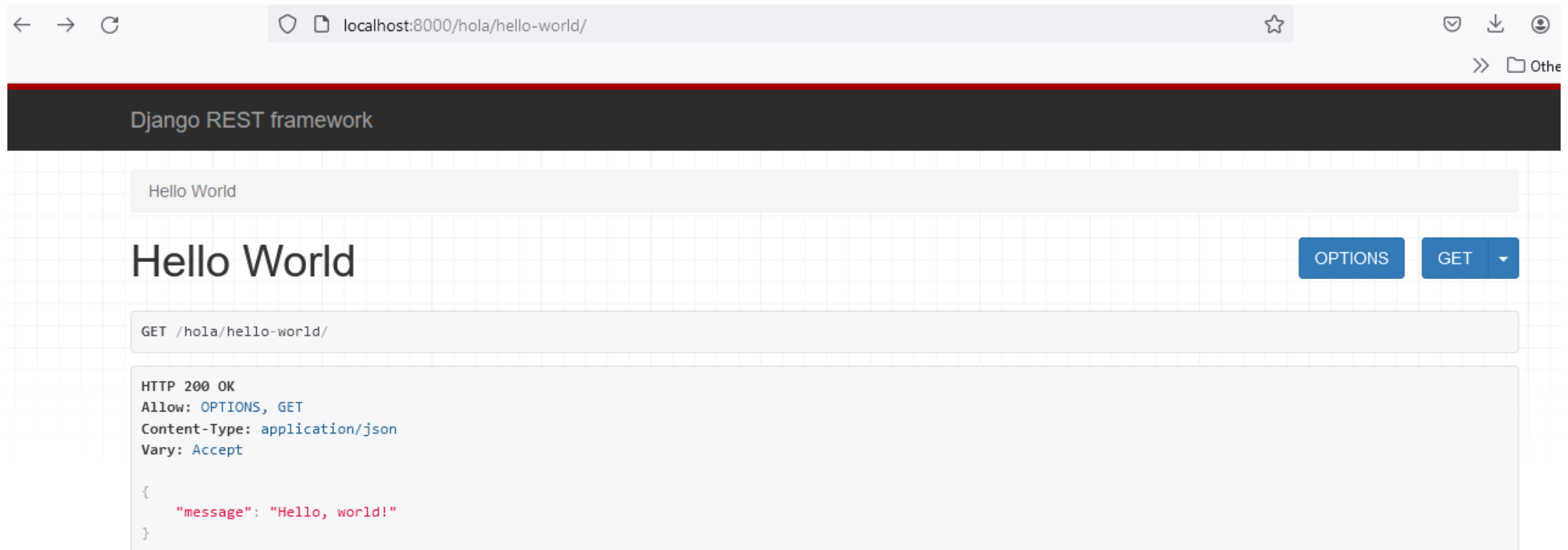
- Explorer Panel:** Displays the project structure. The file `urls.py` under the `hola` app is highlighted with a red box and a '1' icon.
- Editor Panel:** Shows the content of `urls.py` with an orange box highlighting the code:
 

```
1 from django.urls import path
2
3 from . import views
4
5 urlpatterns = []
6
7     path('hello-world/', views.hello_world, name='hello_world'),
8
9 
```
- Terminal Panel:** Shows the command prompt with the path `(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\Proyect`.
- Status Bar:** Indicates the file is `Ln 9, Col 2` with `Spaces: 4`, `UTF-8`, and `CRLF` line endings. It also shows the Python version `3.11.5 64-bit`.

# Probamos la API EndPoints en Django

Ejecutamos el servidor Django en Mozilla

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack\backend>python manage.py runserver
```



localhost:8000/hola/hello-world/

Django REST framework

Hello World

OPTIONS GET

GET /hola/hello-world/

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

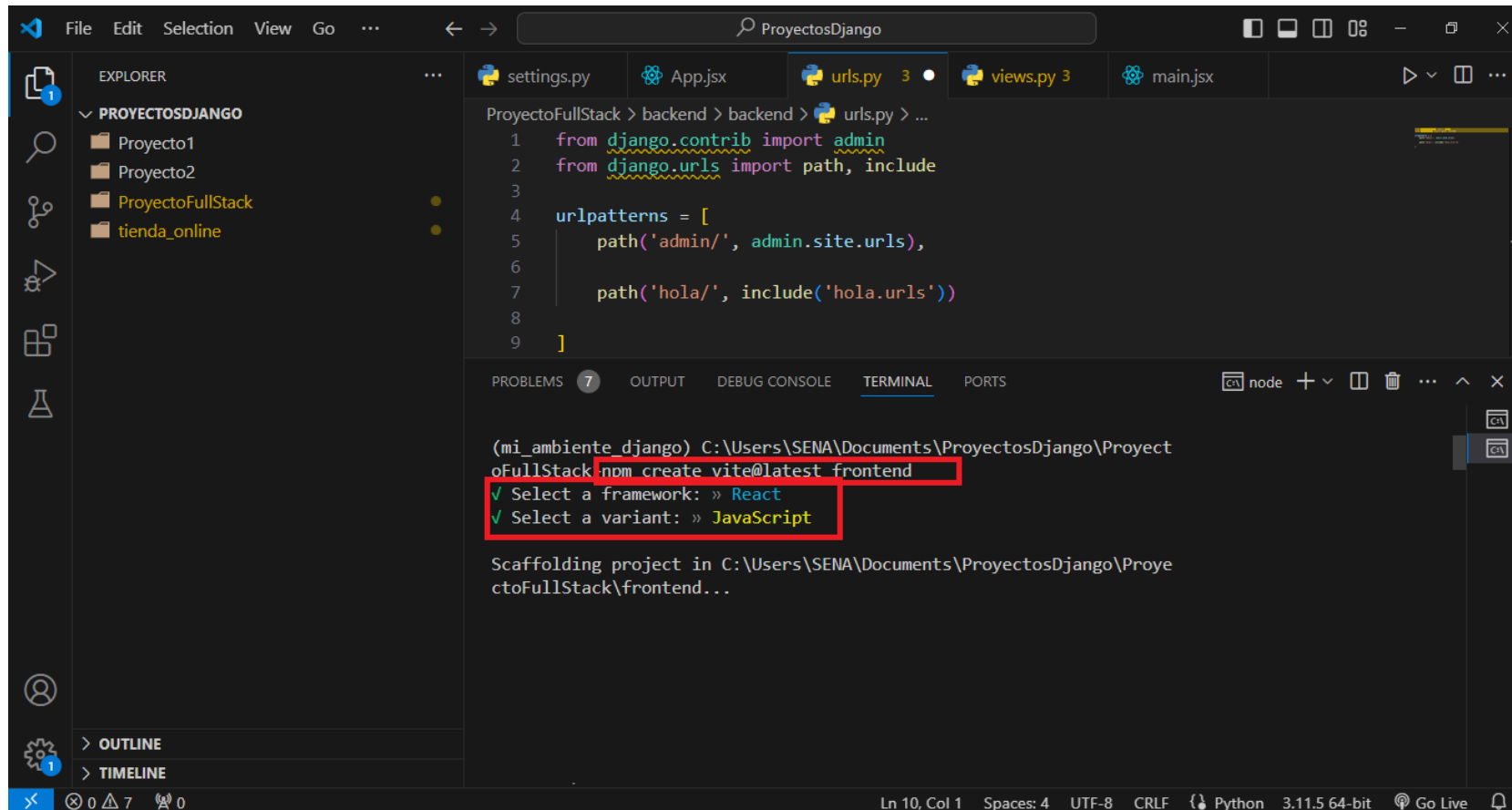
```
{
  "message": "Hello, world!"
}
```

# FrontEnd en React



# Crear Proyecto React

Abrir una nueva terminal en Visual Studio Code y activar el ambiente virtual. Cree un nuevo proyecto llamado **frontend**, dentro de la carpeta **ProyectoFullStack**



The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** A folder named `PROYECTOSDJANGO` is expanded, showing subfolders `Proyecto1`, `Proyecto2`, `ProyectoFullStack`, and `tienda_online`.
- Editor:** The file `urls.py` is open, showing Django URL configuration code:
 

```
1 from django.contrib import admin
2 from django.urls import path, include
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6
7     path('hola/', include('hola.urls'))
8
9 ]
```
- Terminal:** A terminal window is open at the bottom with the following commands and output:
 

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack> npm create vite@latest frontend
✓ Select a framework: » React
✓ Select a variant: » JavaScript

Scaffolding project in C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack\frontend...
```

# Conceptos previos





# Axios

Axios es una popular librería de JavaScript que se utiliza para realizar solicitudes HTTP desde el navegador y Node.js. Está basada en promesas, lo que facilita el manejo de operaciones asíncronas.

## Características Principales

- Basada en Promesas: Axios utiliza promesas, lo que facilita el manejo de las solicitudes y respuestas asíncronas. Esto permite usar `then` y `catch` para manejar resultados y errores respectivamente.
- Manejo de Errores: Proporciona una forma clara y sencilla de manejar errores en las solicitudes HTTP.
- Automático Manejo de JSON: Las solicitudes y respuestas en formato JSON se manejan automáticamente, simplificando el trabajo con APIs RESTful.

The Axios logo, with the letters "A", "X", "I", "O", and "S" in a stylized, blue, sans-serif font. The "I" is slightly smaller and positioned between the "X" and "O".

# Axios

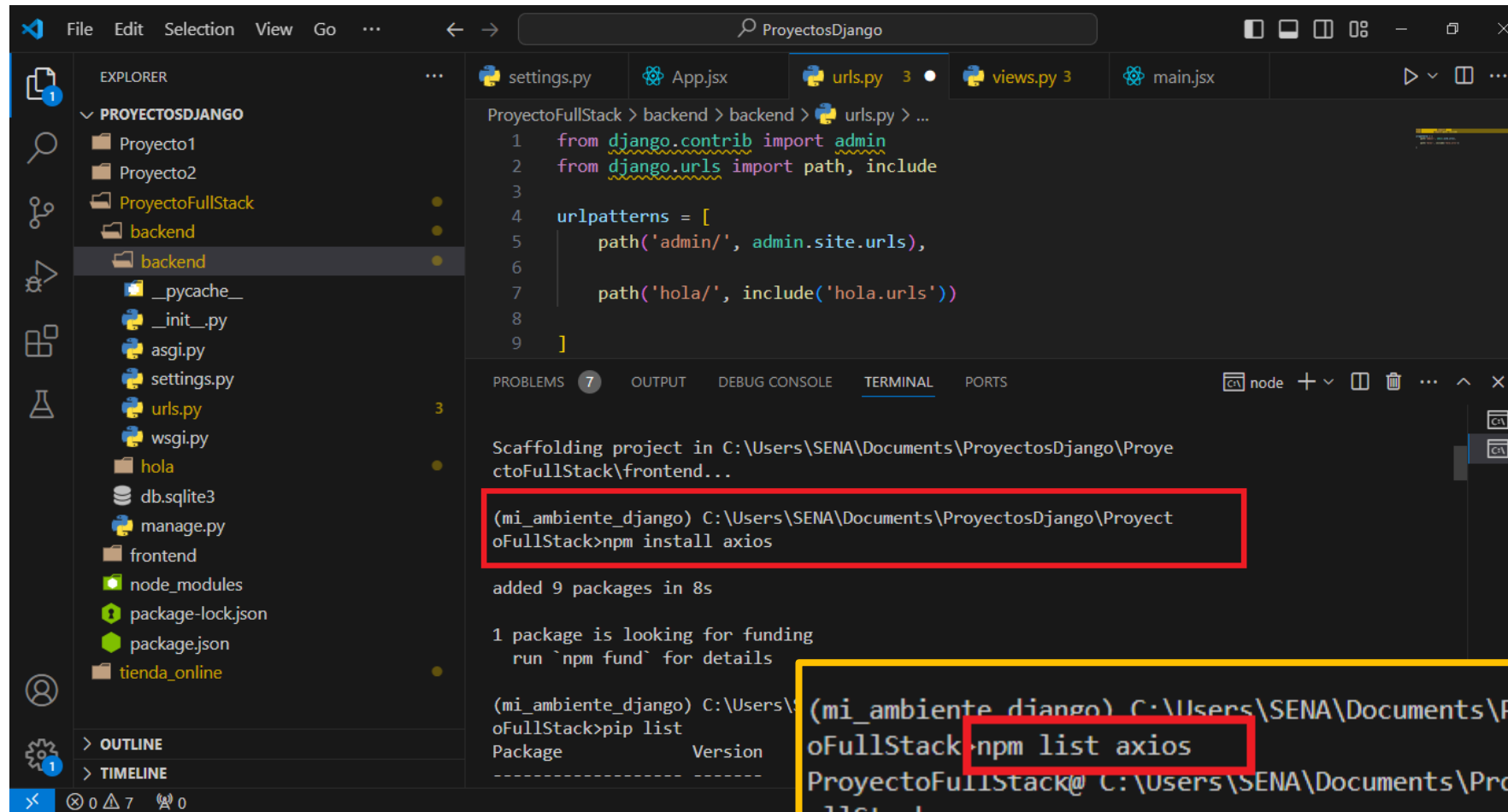
## Ejemplo de Solicitud GET

```
axios.get('https://api.example.com/data')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error fetching data:', error);
  });
```

## Ejemplo de Solicitud POST

```
axios.post('https://api.example.com/data', {
  key1: 'value1',
  key2: 'value2'
})
  .then(response => {
    console.log('Data saved:', response.data);
  })
```

# Instalar paquete Axios



The screenshot shows the Visual Studio Code interface with a Django project named 'ProyectosDjango'. The Explorer panel on the left shows the project structure, including a 'backend' directory. The main editor displays the 'urls.py' file in the 'backend' directory, which contains Django URL patterns. The Terminal panel at the bottom shows the command prompt for the 'mi\_ambiente\_django' environment. The command 'npm install axios' has been executed, and the output shows that 9 packages were added in 8 seconds. A red box highlights the command and its output in the terminal. A blue arrow points from the terminal output to a separate box showing the result of the 'npm list axios' command.

```

ProjectoFullStack > backend > backend > urls.py > ...
1  from django.contrib import admin
2  from django.urls import path, include
3
4  urlpatterns = [
5      path('admin/', admin.site.urls),
6
7      path('hola/', include('hola.urls'))
8
9  ]

```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Scaffolding project in C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack\frontend...

(mi\_ambiente\_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack>npm install axios

added 9 packages in 8s

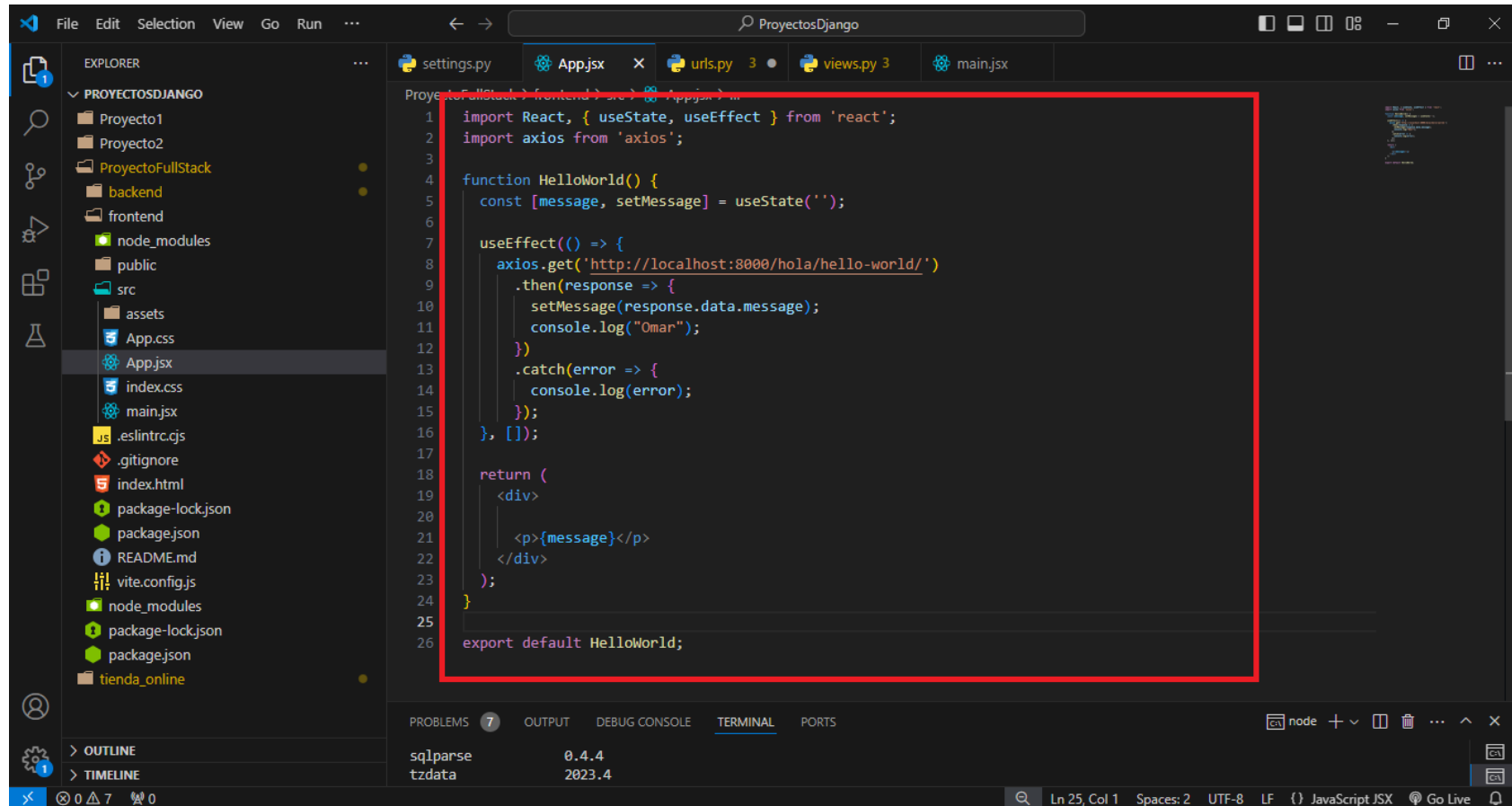
1 package is looking for funding  
run `npm fund` for details

(mi\_ambiente\_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack>pip list

Package	Version
axios	1.7.2

# Crear un componente de React para realizar solicitudes HTTP

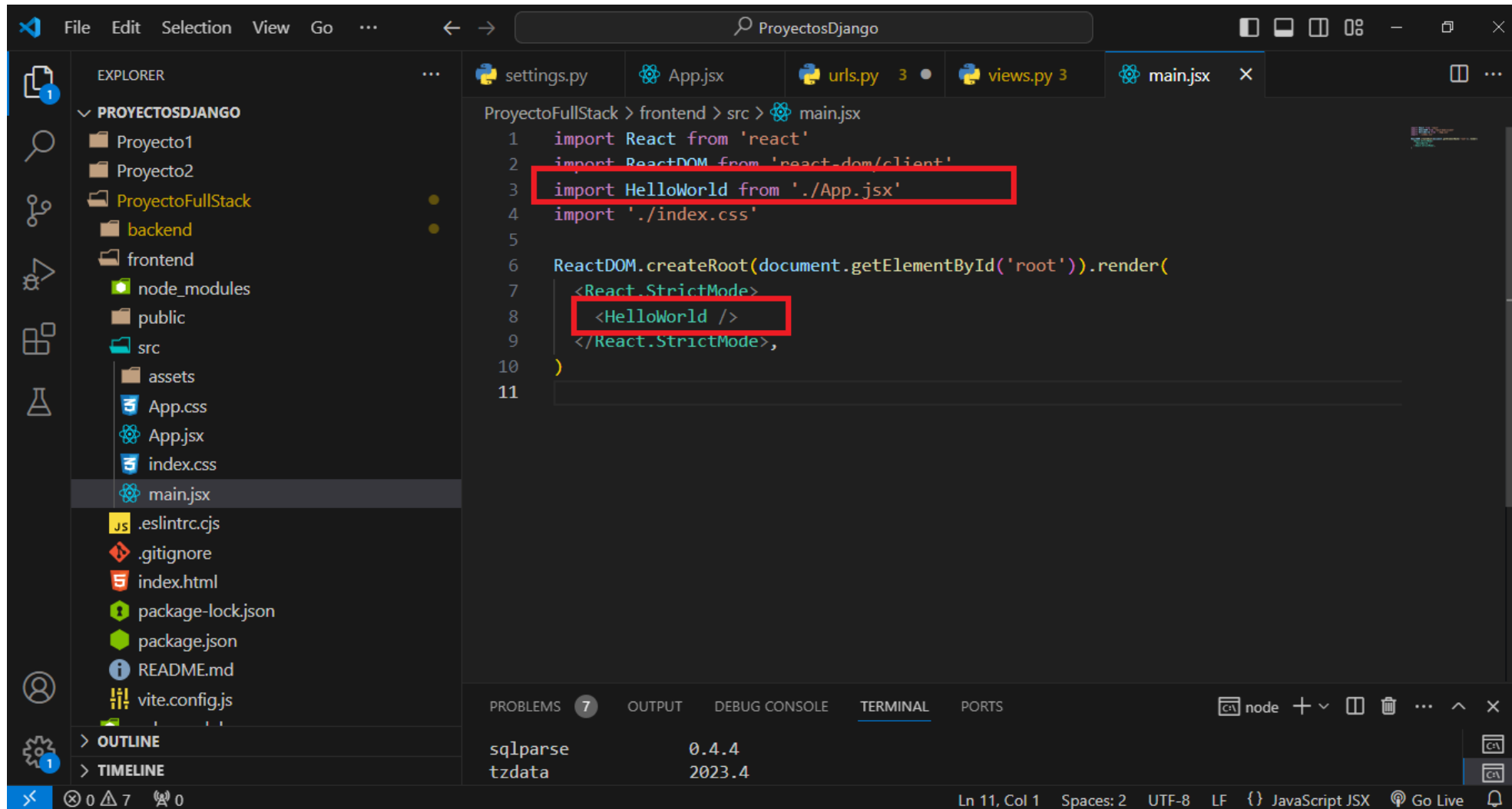
A continuación, necesitamos crear un componente React que realice solicitudes HTTP a los puntos finales de la API . Usaremos Axios para realizar las solicitudes HTTP . Una vez que recibimos los datos JSON , podemos mostrarlos en la página web usando componentes de React.



The screenshot shows a Visual Studio Code editor window with a project named 'ProyectosDjango'. The Explorer sidebar on the left shows the file structure, with 'App.jsx' selected under the 'frontend' directory. The main editor area displays the code for 'App.jsx', which is highlighted with a red rectangle. The code imports React, useState, and useEffect from 'react', and axios from 'axios'. It defines a 'HelloWorld' function that uses useState to manage a 'message' state and useEffect to perform an axios GET request to 'http://localhost:8000/hola/hello-world/'. The request success handler sets the message state and logs 'Omar'. The error handler logs the error. The component returns a JSX element with a paragraph tag displaying the message. The file is exported as 'HelloWorld'.

```
1 import React, { useState, useEffect } from 'react';
2 import axios from 'axios';
3
4 function HelloWorld() {
5   const [message, setMessage] = useState('');
6
7   useEffect(() => {
8     axios.get('http://localhost:8000/hola/hello-world/')
9       .then(response => {
10         setMessage(response.data.message);
11         console.log("Omar");
12       })
13       .catch(error => {
14         console.log(error);
15       });
16   }, []);
17
18   return (
19     <div>
20       <p>{message}</p>
21     </div>
22   );
23
24 export default HelloWorld;
```

# Registrar componente en el main.jsx



The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor in the center. The file explorer shows the project structure for 'ProyectosDjango', with the 'main.jsx' file selected under the 'src' directory. The code editor displays the content of 'main.jsx', which includes imports for React, ReactDOM, HelloWorld, and index.css. The 'HelloWorld' component is imported from './App.jsx' and is used within the ReactDOM.createRoot function. The 'HelloWorld' component is highlighted with a red box, and the import statement is also highlighted with a red box.

```

1 import React from 'react'
2 import ReactDOM from 'react-dom/client'
3 import HelloWorld from './App.jsx'
4 import './index.css'
5
6 ReactDOM.createRoot(document.getElementById('root')).render(
7   <React.StrictMode>
8     <HelloWorld />
9   </React.StrictMode>,
10 )
11

```

The status bar at the bottom indicates the current file is 'main.jsx' at line 11, column 1, with 2 spaces, UTF-8 encoding, and LF line endings. The language is set to JavaScript JSX.

# Ejecutar servidor React

Ejecutamos el servidor react. Entramos a la carpeta del **frontend**

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack>cd frontend
```

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack\frontend>npm install
```

```
npm WARN deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
```

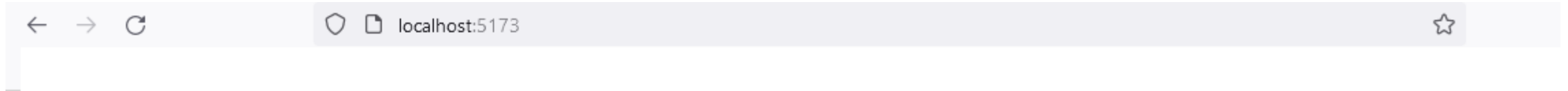
```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\ProyectoFullStack\frontend>npm run dev
```

```
> frontend@0.0.0 dev
> vite
```

```
VITE v5.2.13 ready in 4581 ms
```

```
→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

# Ejecución servidor React



Hola desde el BackEnd!



**G R A C I A S**

Línea de atención al ciudadano: 01 8000 910270  
Línea de atención al empresario: 01 8000 910682



[www.sena.edu.co](http://www.sena.edu.co)