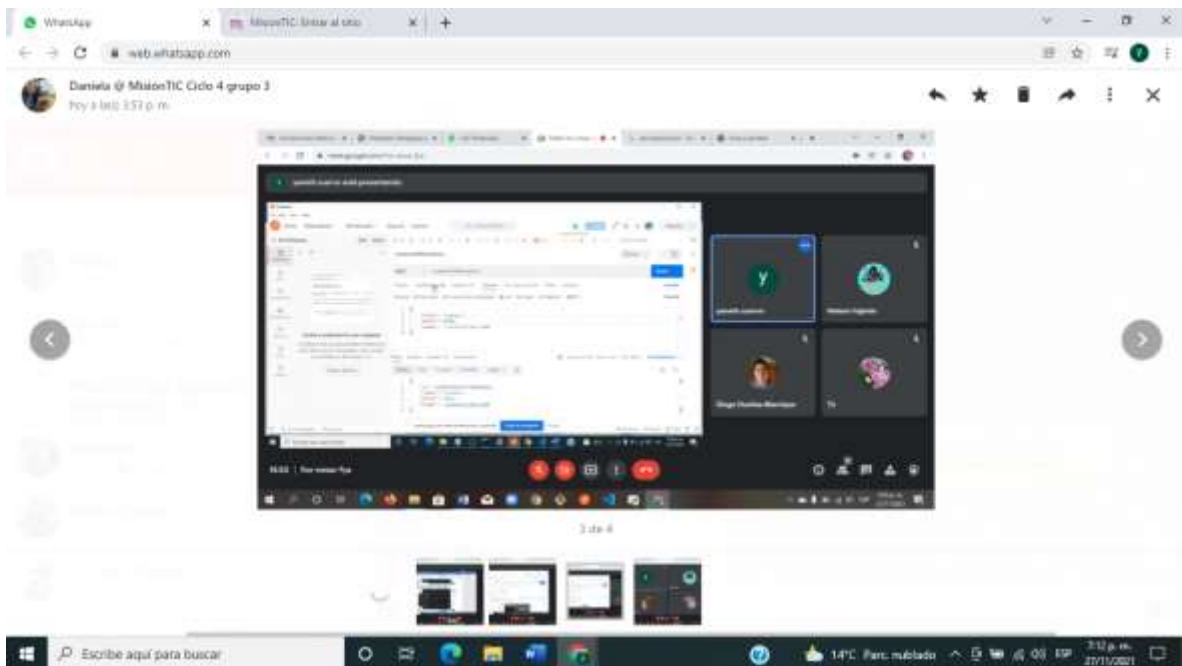


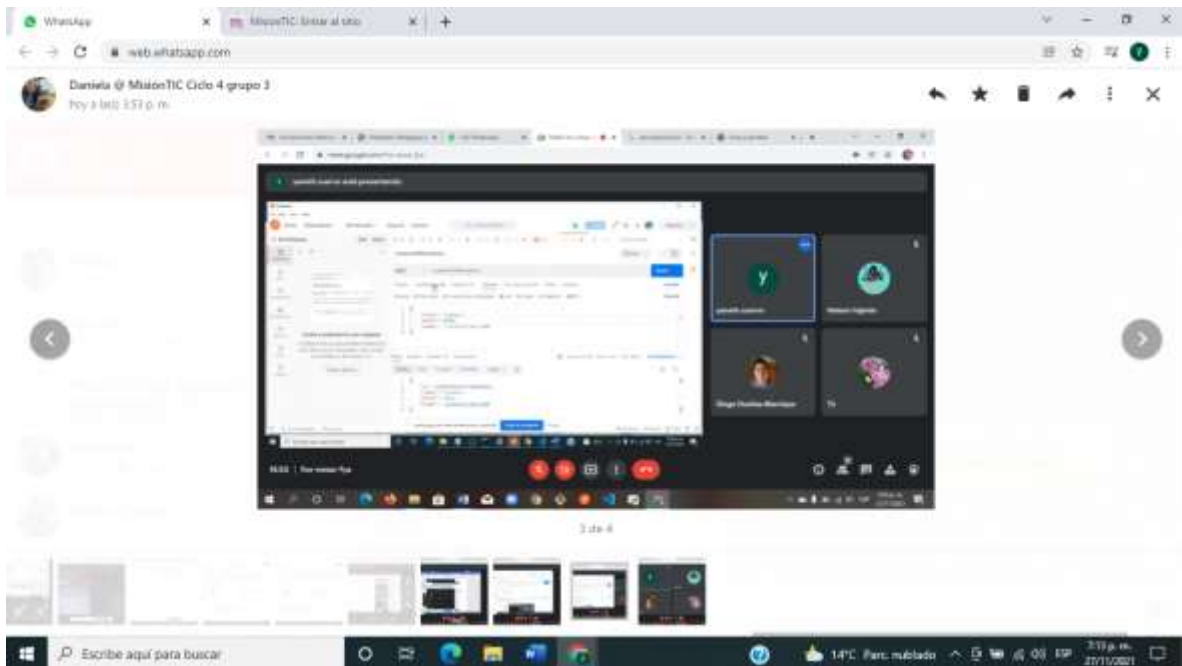
## SPRINT 3. PROYECTO TIENDA VIRTUAL MASCOTA INTEGRACION DE SERVICIOS – BACKEND

### Integrantes Grupo 12. Equipo 3 LOS TIC DE LOS NODEJS

Nombre	Cedula	Rol	Nivel de Participación
CONTRERAS NICOLAS CAMILO	1193112644	Administrador de Configuración	Alto
DUEÑAS MANRIQUE DIEGO FERNANDO	1026283999	Diseñador UI	Alto
FAJARDO ENRIQUEZ ZULY DANIELA	1086135083	Diseñador de Software	Alto
NELSON GERARDO FAJARDO PATARROYO	80155325	Lider del Equipo	Alto
YANETH MILENA CUERVO BARAHONA	40047928	Tester	Alto

1. Subir las evidencias de las reuniones diarias (pantallazo de las reuniones)





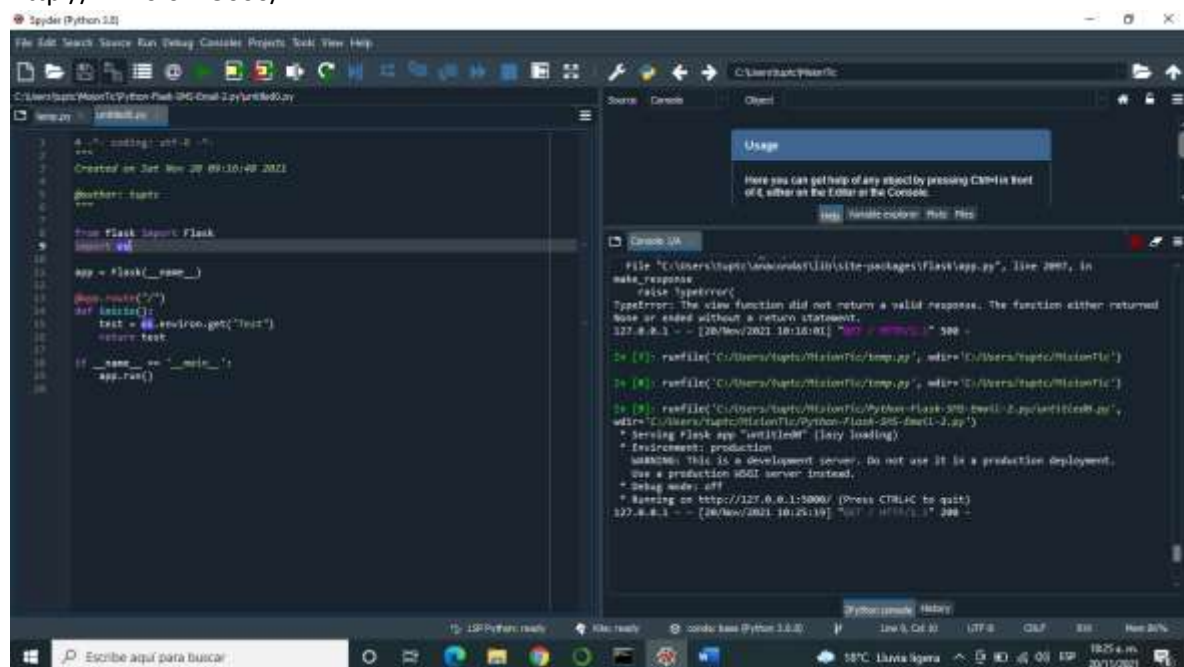
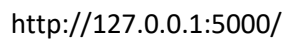
2. Pantallazo del tablero Kanban con las tareas asignadas y realizadas.



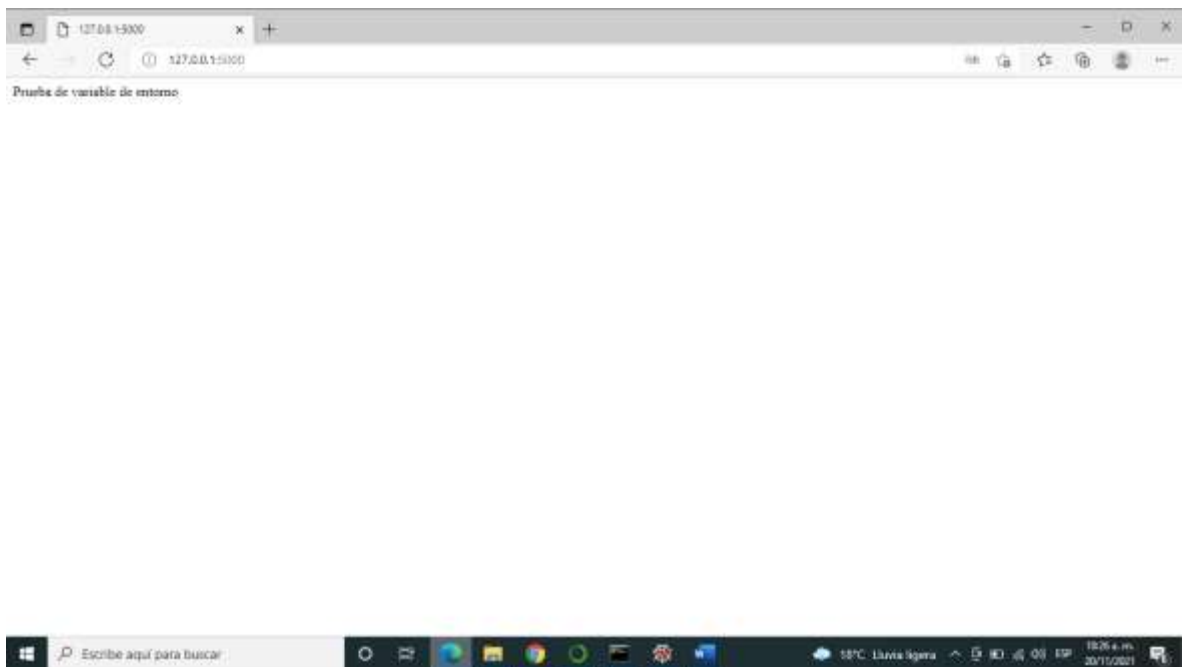
3. Envío de mensajes y correos electrónicos mediante el uso de herramientas como Twilio y Sendgrid aplicando al proyecto de Tienda Virtual Mascota.

Paso a paso.

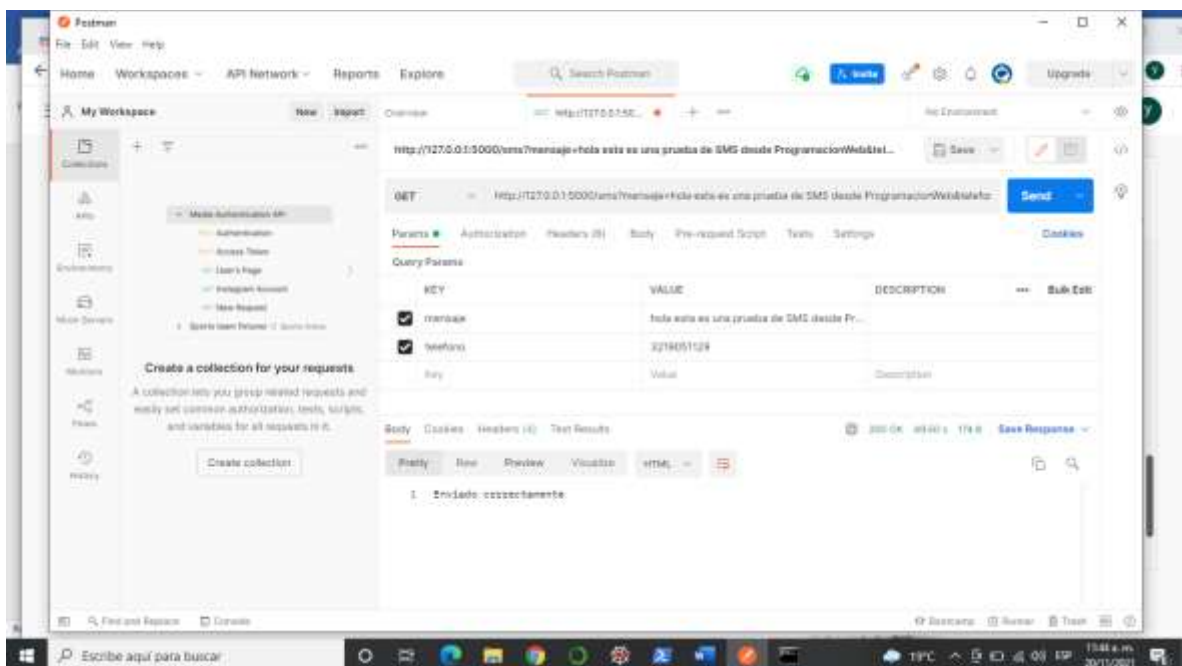
Ingresamos al entorno Spider bajo Python en donde ingresamos las diferentes variables de entorno y le asignamos el valor que lo extraemos de twillio y del sendgrid



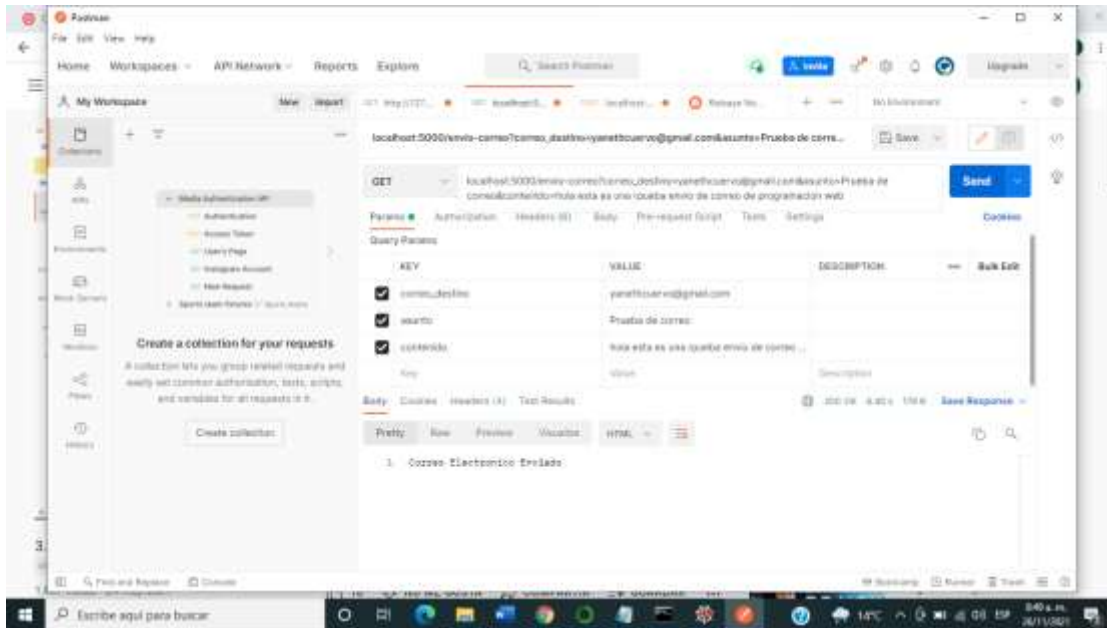
Y validamos la conexión con el puerto generado <http://127.0.0.1:5000/>



Validamos por medio de la herramienta postman en envío del mensaje utilizando el puerto localhost:3000

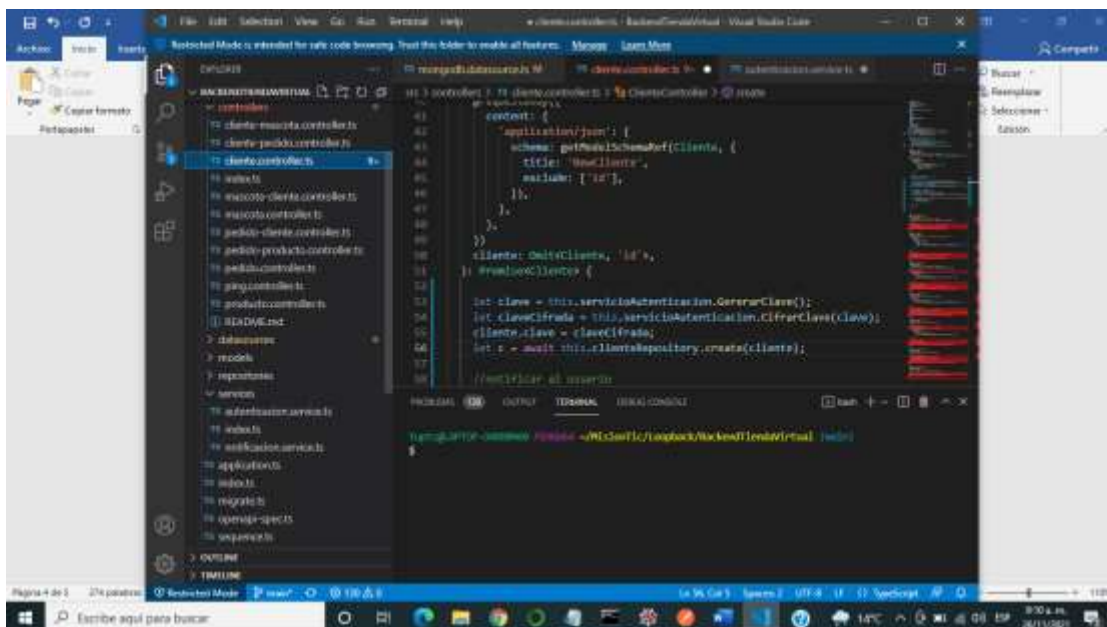


Validamos el envío del mensaje mediante el aplicativo postman mediante el puerto Localhost:5000/



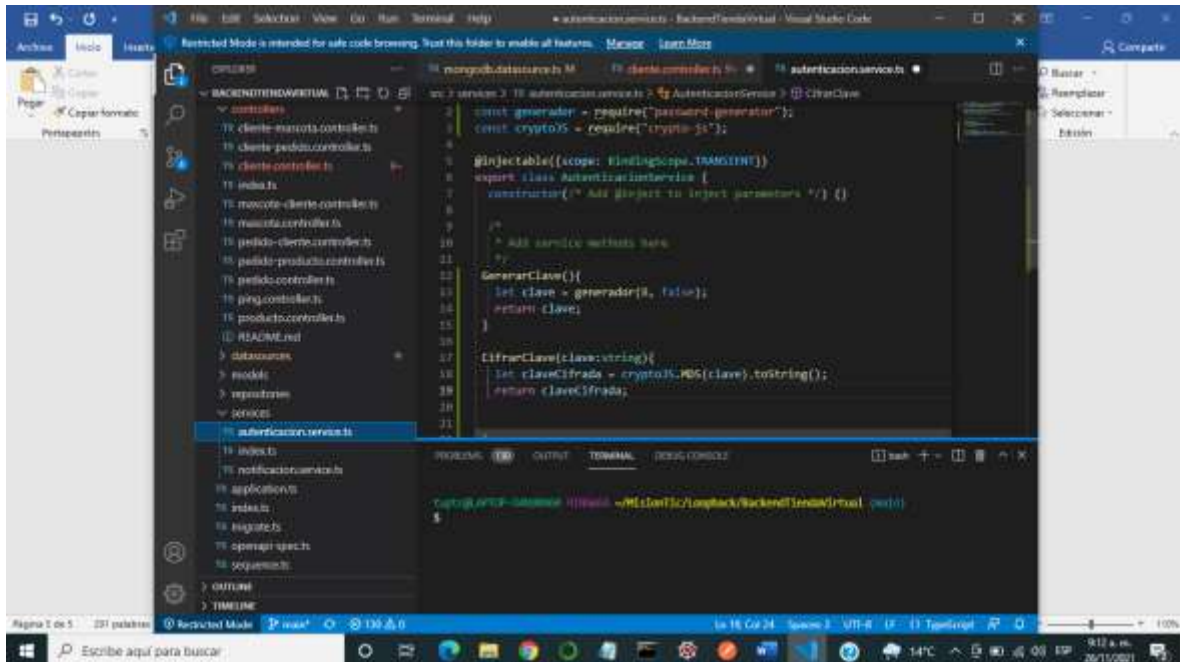
Ahora agregamos funcionalidad de notificación vía correo electrónico de una usuario y contraseña que pertenece a una persona que se registra en la plataforma utilizando el servicio de correo electrónico y mensaje de texto. Utilizando la integración de Loopback con Flask

Dentro del controlador cliente dentro de la función post modificamos la respuesta mediante un retorno

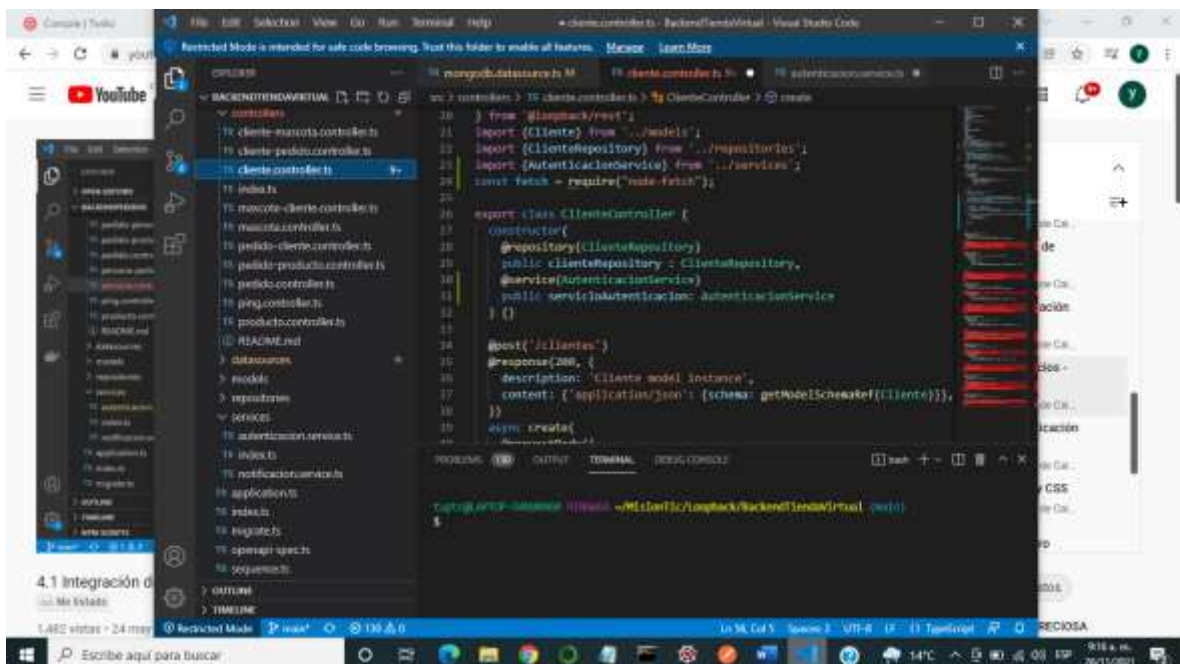




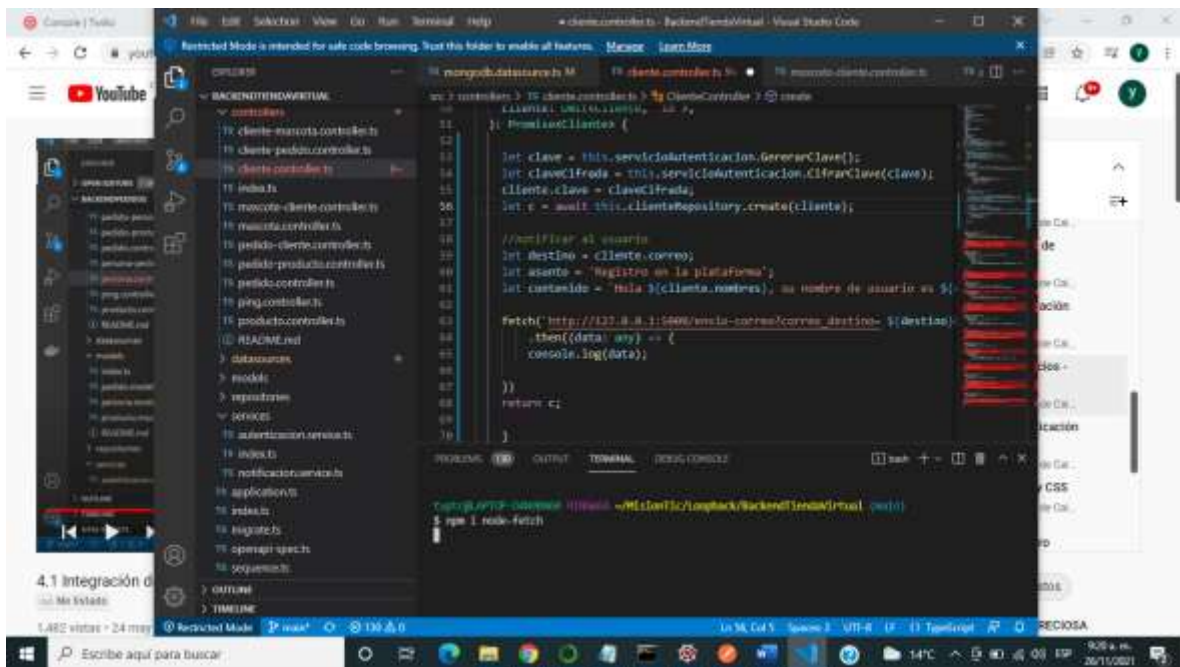
Modificamos el archivo de servicios donde tenemos autenticación y notificación. En autenticación determinamos el generar y cifrar de clave. De esta manera ya tenemos dos métodos que utilizaremos en el controller cliente



Importamos el servicio de autenticación



Ahora consumimos la aplicación de sms y email de Python instalamos un paquete el cual nos permitas hacer un llamado asíncrono con aplicaciones externas mediante la función **npm i node-fetch**

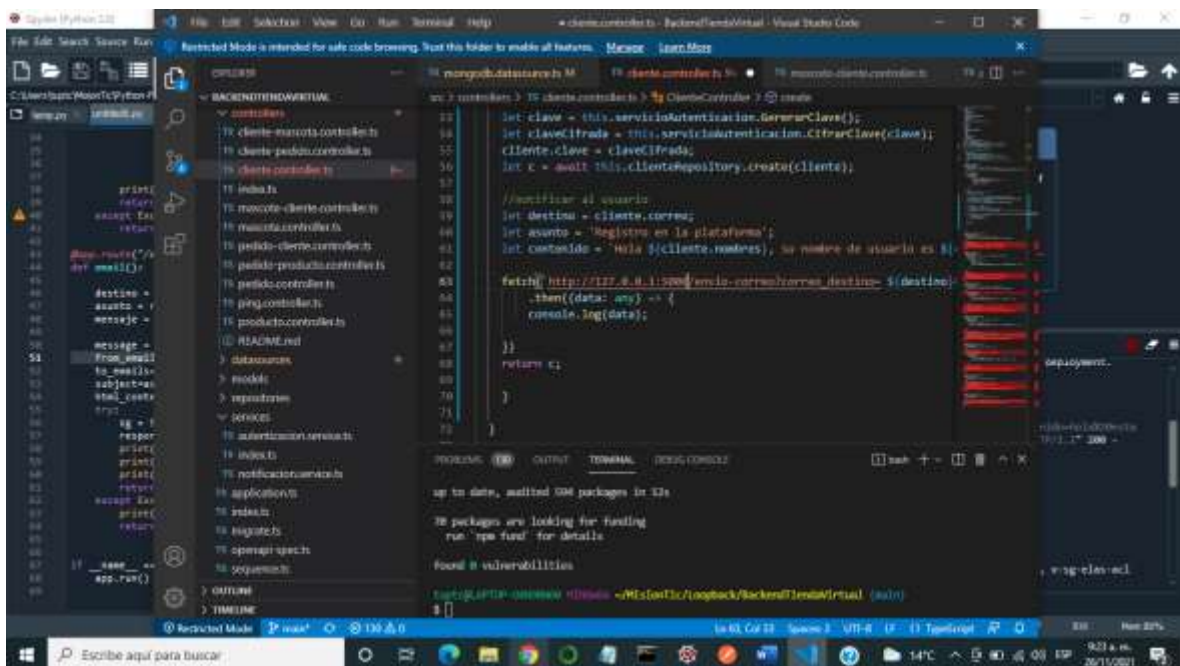


The screenshot shows the Visual Studio Code editor with a project named 'BackendLindaVirtual'. The file explorer on the left shows the project structure, including controllers, models, and services. The main editor displays the 'ClienteController.js' file. The code in this file includes a method 'crearCliente' that uses 'node-fetch' to send an email. The terminal at the bottom shows the command 'npm i node-fetch' being executed.

```
11 // crear cliente
12 crearCliente = async (req, res) => {
13   const { nombre, correo, password } = req.body;
14   if (!nombre || !correo || !password) {
15     return res.status(400).json({ error: 'Todos los campos son obligatorios' });
16   }
17   // Validar correo electrónico
18   const regex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
19   if (!regex.test(correo)) {
20     return res.status(400).json({ error: 'Correo electrónico no válido' });
21   }
22   // Validar contraseña
23   const regexPassword = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*^&])[8-10]$/;
24   if (!regexPassword.test(password)) {
25     return res.status(400).json({ error: 'Contraseña no válida' });
26   }
27   // Verificar si el correo ya existe
28   const clienteExistente = await this.clientRepository.findOne({ correo });
29   if (clienteExistente) {
30     return res.status(400).json({ error: 'El correo electrónico ya está registrado' });
31   }
32   // Crear cliente
33   const cliente = new this.clientModel({ nombre, correo, password });
34   await cliente.save();
35   // Enviar correo de bienvenida
36   const correoDestino = cliente.correo;
37   const asunto = 'Bienvenido a la plataforma';
38   const contenido = `Hola ${cliente.nombre}, su nombre de usuario es ${cliente.nombre}`;
39   fetch('http://127.0.0.1:5000/email-correo?correo=destino=${correoDestino}&asunto=${asunto}&contenido=${encodeURIComponent(contenido)}`)
40     .then((data) => {
41       console.log(data);
42     })
43     .catch((error) => {
44       console.log(error);
45     });
46   return res.status(201).json({ message: 'Cliente creado exitosamente' });
47 }
```

```
npm i node-fetch
```

Ahora definimos el envío de correo mediante el uso de variables que nos permita dicho proceso

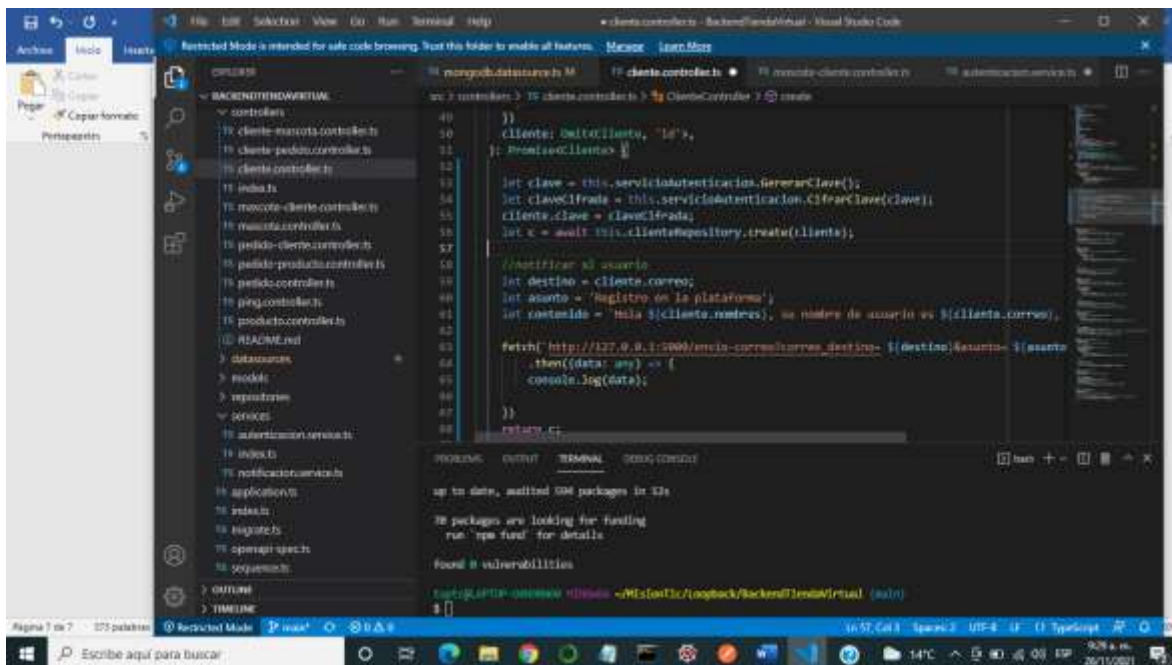


The screenshot shows the Visual Studio Code editor with the same project. The file explorer on the left shows the project structure. The main editor displays the 'ClienteController.js' file. The code in this file includes a method 'crearCliente' that uses 'node-fetch' to send an email. The terminal at the bottom shows the command 'npm i node-fetch' being executed.

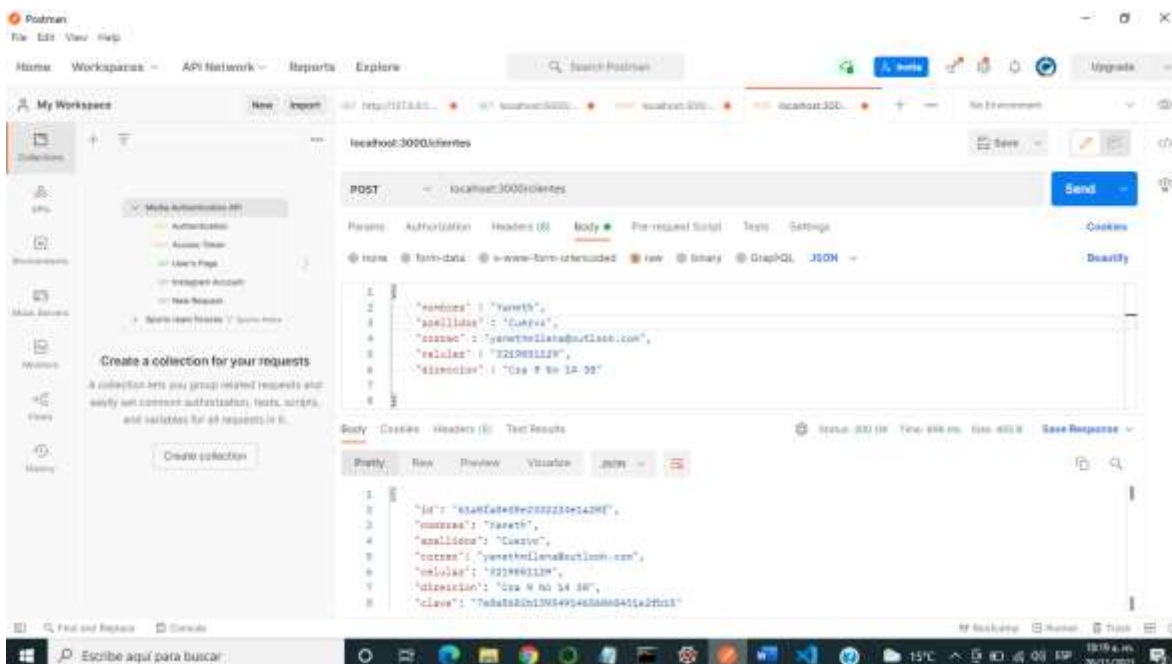
```
11 // crear cliente
12 crearCliente = async (req, res) => {
13   const { nombre, correo, password } = req.body;
14   if (!nombre || !correo || !password) {
15     return res.status(400).json({ error: 'Todos los campos son obligatorios' });
16   }
17   // Validar correo electrónico
18   const regex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
19   if (!regex.test(correo)) {
20     return res.status(400).json({ error: 'Correo electrónico no válido' });
21   }
22   // Validar contraseña
23   const regexPassword = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*^&])[8-10]$/;
24   if (!regexPassword.test(password)) {
25     return res.status(400).json({ error: 'Contraseña no válida' });
26   }
27   // Verificar si el correo ya existe
28   const clienteExistente = await this.clientRepository.findOne({ correo });
29   if (clienteExistente) {
30     return res.status(400).json({ error: 'El correo electrónico ya está registrado' });
31   }
32   // Crear cliente
33   const cliente = new this.clientModel({ nombre, correo, password });
34   await cliente.save();
35   // Enviar correo de bienvenida
36   const correoDestino = cliente.correo;
37   const asunto = 'Bienvenido a la plataforma';
38   const contenido = `Hola ${cliente.nombre}, su nombre de usuario es ${cliente.nombre}`;
39   fetch('http://127.0.0.1:5000/email-correo?correo=destino=${correoDestino}&asunto=${asunto}&contenido=${encodeURIComponent(contenido)}`)
40     .then((data) => {
41       console.log(data);
42     })
43     .catch((error) => {
44       console.log(error);
45     });
46   return res.status(201).json({ message: 'Cliente creado exitosamente' });
47 }
```

```
npm i node-fetch
```

De esta manera configuramos en envió del correo electrónico mediante este método de almacenamiento y notificación

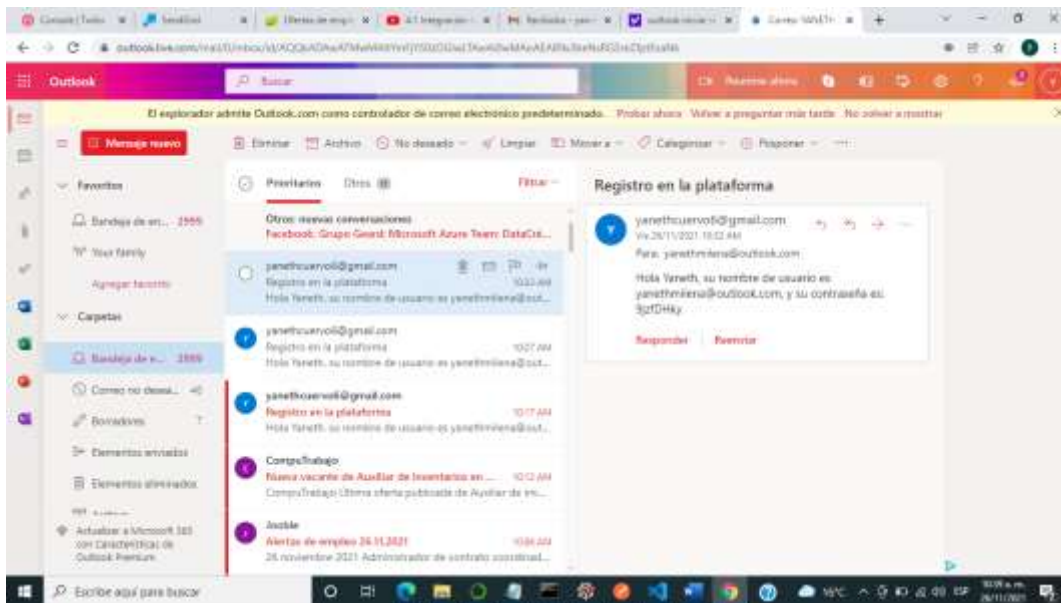


Validamos en el postman donde podemos ver que se genera una clave aleatoriamente



Validamos en el correo que se haya enviado el mensaje correctamente

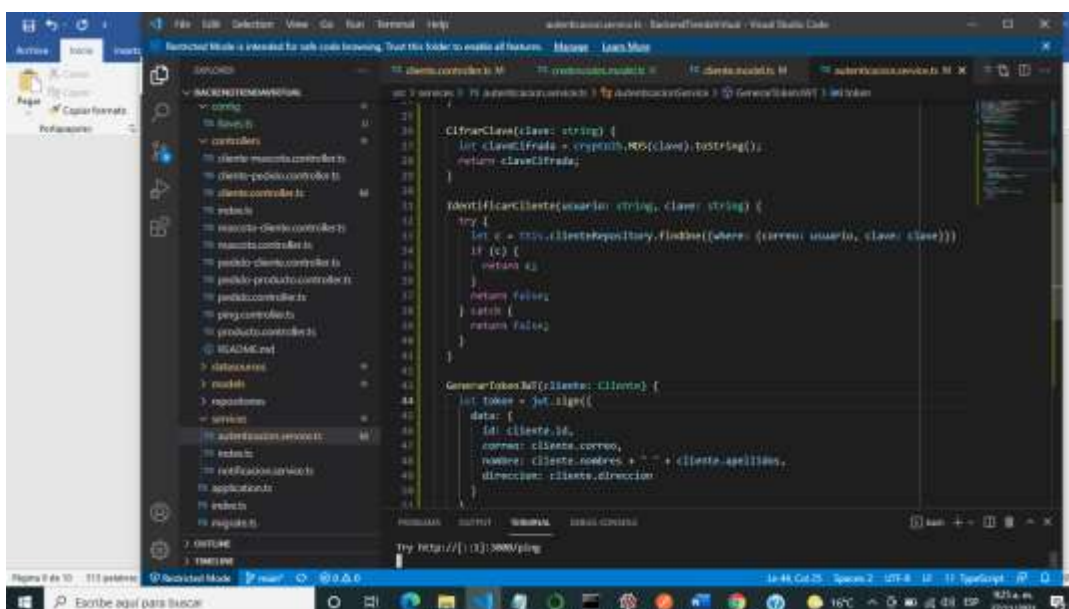


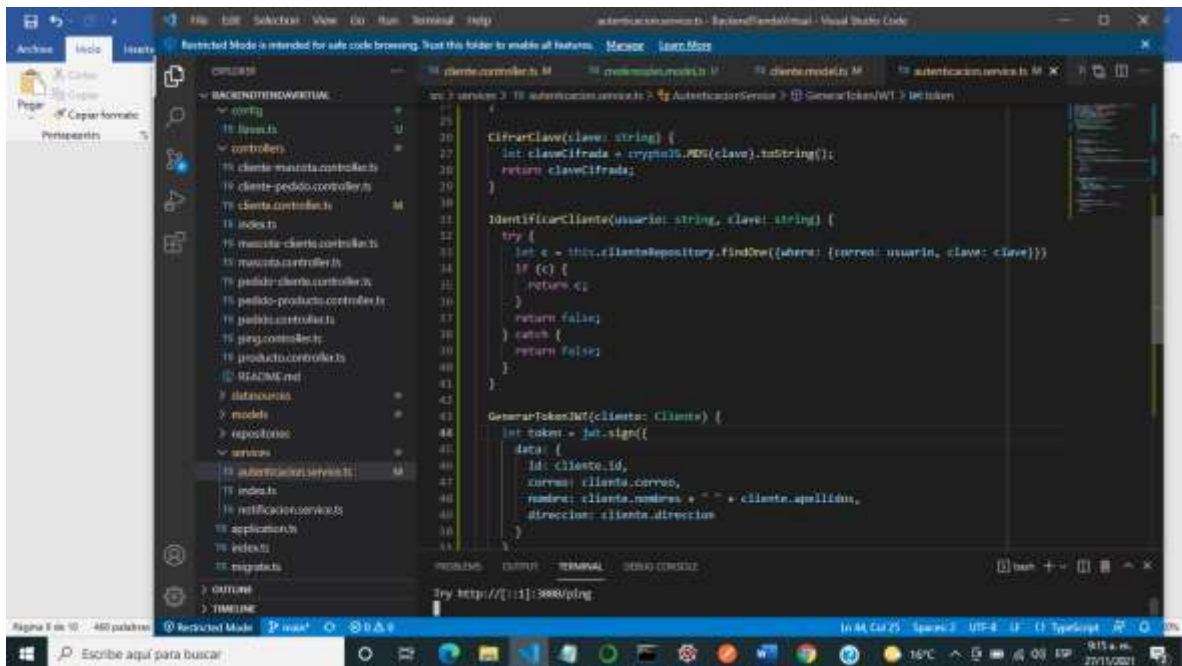


Ahora vamos agregar un componente importante con es el de la seguridad el cual nos permitirá que solo las personas autorizadas puedas hacer uso de las funciones según los permisos. El cual se hace a través de Loopback Autenticacion con la validación de un Token el cual nos permitirá el acceso a la funcionalidades.

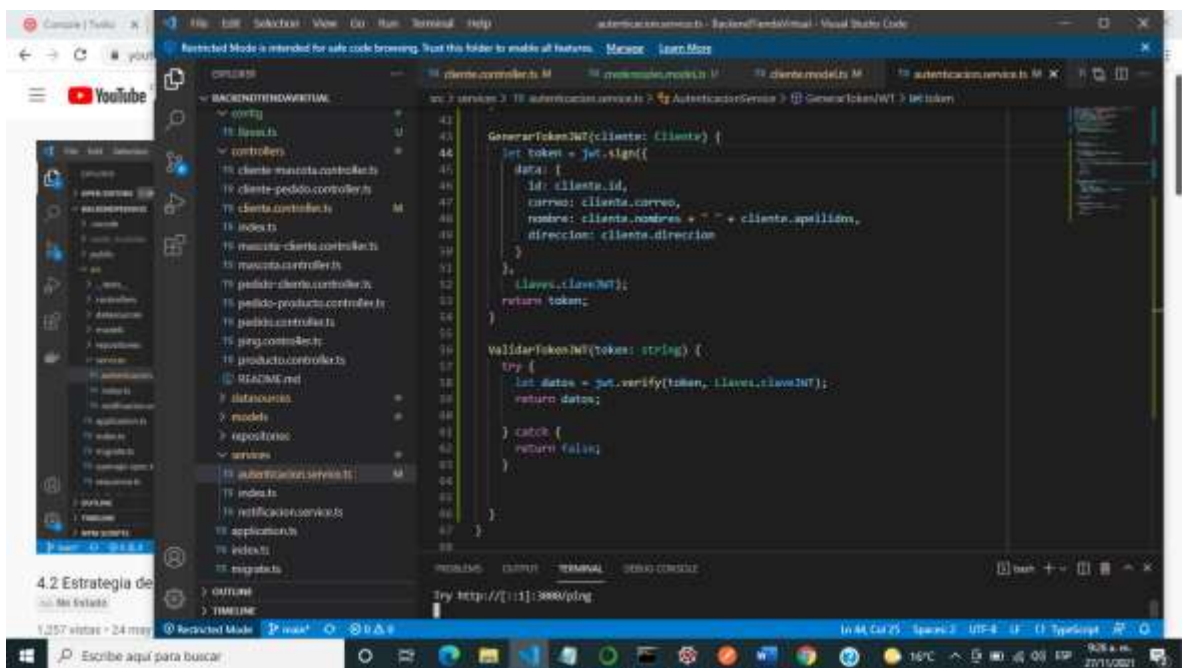
Vamos a ver como se autentica un usuario en el sistema a través de un token el cual nos permitirá el acceso a las funciones.

Modificamos el servicio de autenticación donde generamos la claves donde debemos identificar una persona mediante el usuario y la clave





Ahora realizamos la asignación de un token a una persona mediante la función `GenerarToken` igualmente validamos dicha acción mediante `ValidarToken`



Para la construcción de un token para lo cual necesitaremos instalar un paquete **npm i jsonwebtoken**

The screenshot shows the Visual Studio Code interface with the 'BackendTendovirtual' project open. The left sidebar displays the file explorer with the following structure:

- src
  - BACKENDTENDOVIRTUAL
    - controllers
      - cliente-virtual.controllers.js
      - cliente-producto.controllers.js
      - cliente.controller.js
      - index.js
      - masacota-cliente.controllers.js
      - masacota.controller.js
      - pedido-cliente.controllers.js
      - pedido-producto.controllers.js
      - pedido.controllers.js
      - ping.controllers.js
      - producto.controllers.js
    - README.md
    - database.js
    - models
    - repositories
    - services
      - administracion.servicios.js
      - index.js
      - notificacion.servicios.js
      - aplicaciones
      - index.js
      - ingresos

The main editor area shows the 'index.js' file with the following code:

```

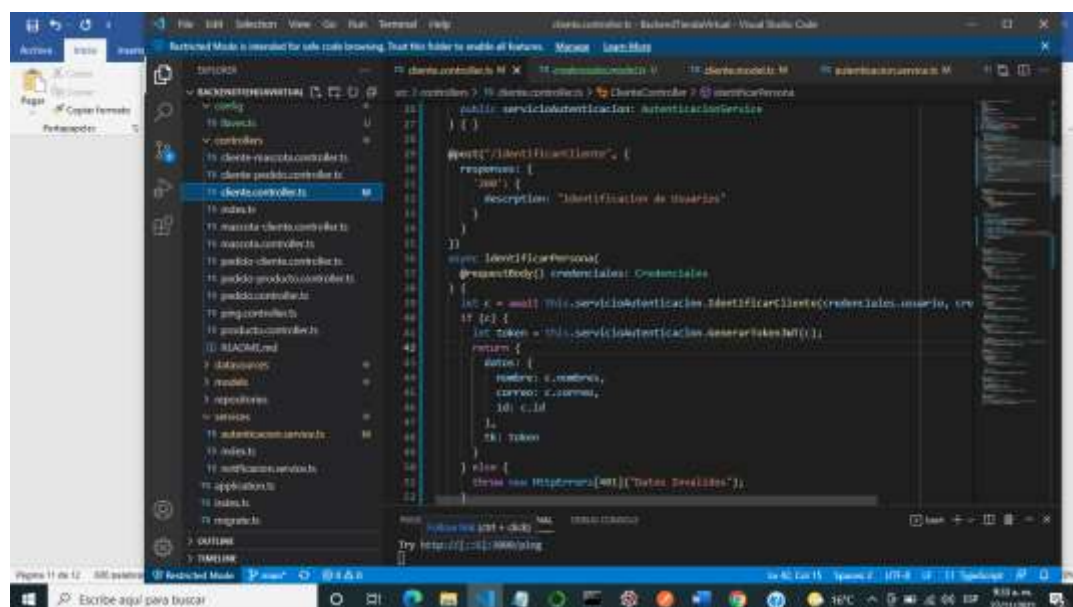
import express from 'express';
import { httpServer } from './httpServer';
import { httpServerSocket } from './httpServerSocket';

const app = express();

export default app;
  
```

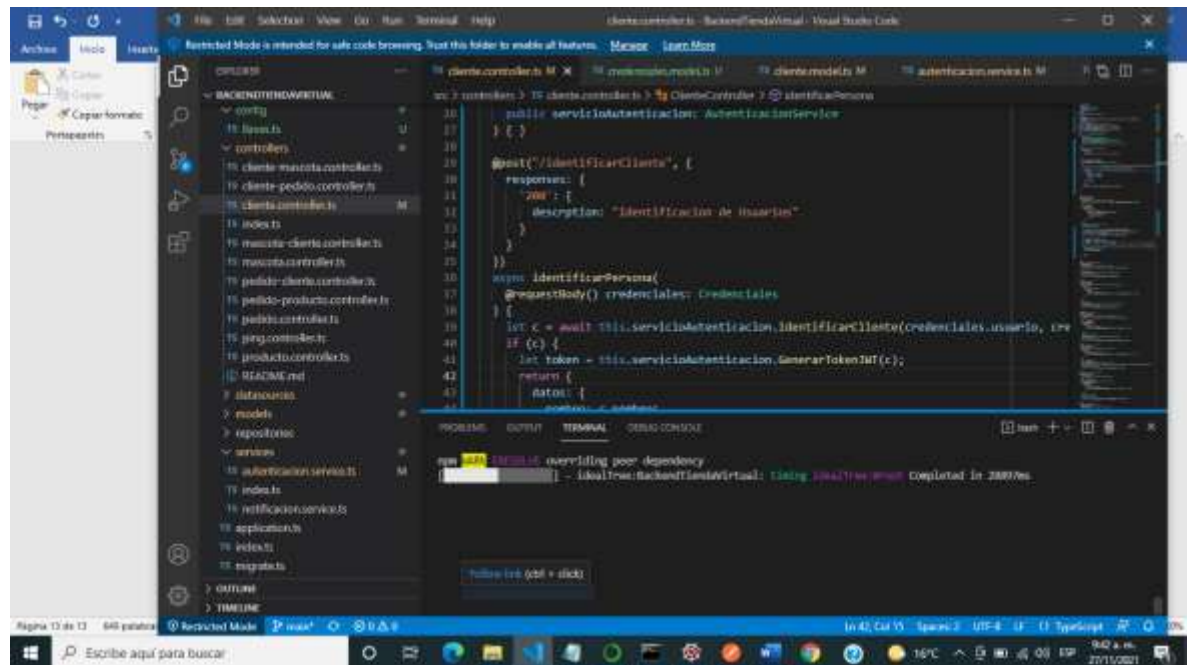
The status bar at the bottom indicates the file is 'index.js' in the 'src' directory, with a file icon and a search icon.

Ahora vamos a utilizar los métodos que se crearon el servicio autenticación mediante la creación de un método dentro del cliente



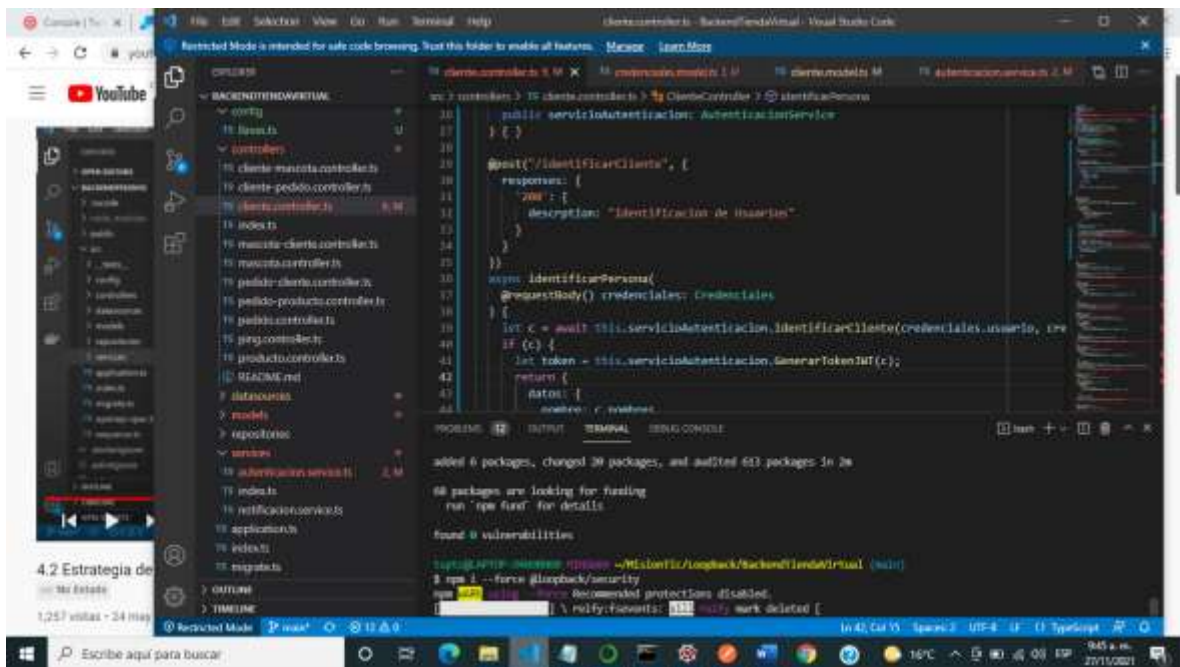
The screenshot shows the Postman application interface. At the top, there's a header bar with the Postman logo and menu options: File, Edit, View, Help. Below this is a navigation bar with tabs for Home, Workspaces, API Network, Reports, Explore, and a search bar. The left sidebar contains icons for My Workspace, Collections, Environments, Mock Servers, Monitors, and History. The main area is divided into two sections. The top section is for creating a new request, with a dropdown for 'local:localhost:3000/loginClients' and a 'Send' button. The bottom section shows the request details, including the method (POST), URL, headers, body, and a 'Send' button. The body is a JSON object with 'username' and 'password' fields. The bottom of the window shows a Windows taskbar with various icons and the system clock.

Instalamos los dos paquetes requeridos para el funcionamiento del loopback authentication **npm i --force @loopback/authentication** el cual nos proveeera la captura de solicitudes y la validación de las mismas

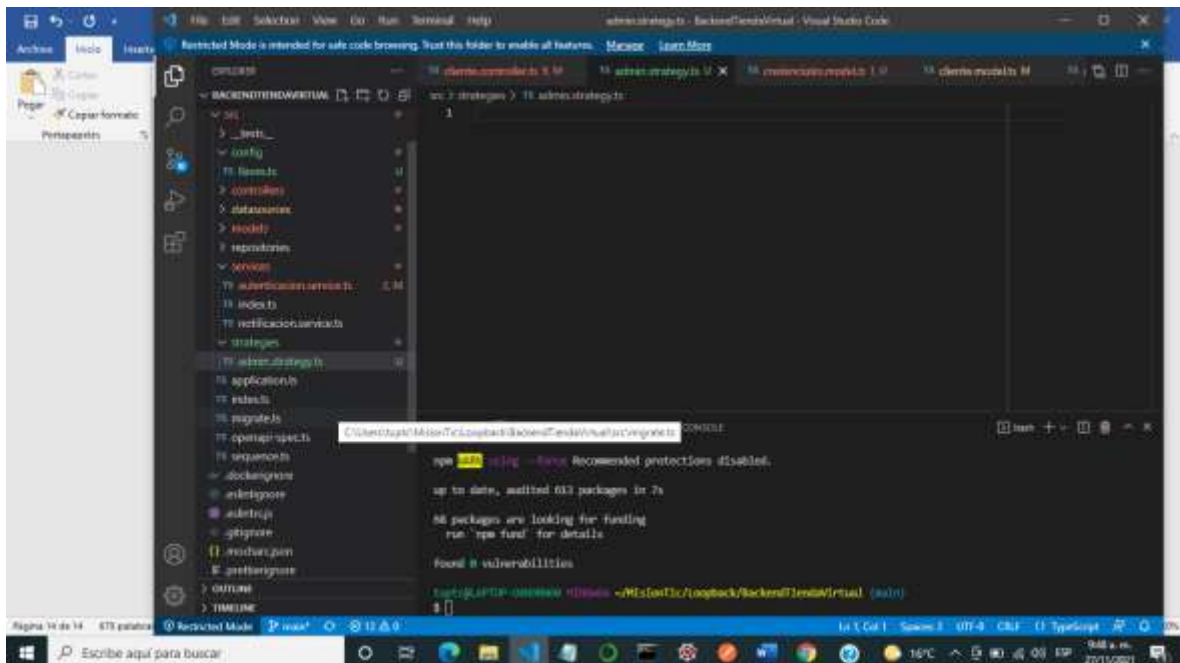




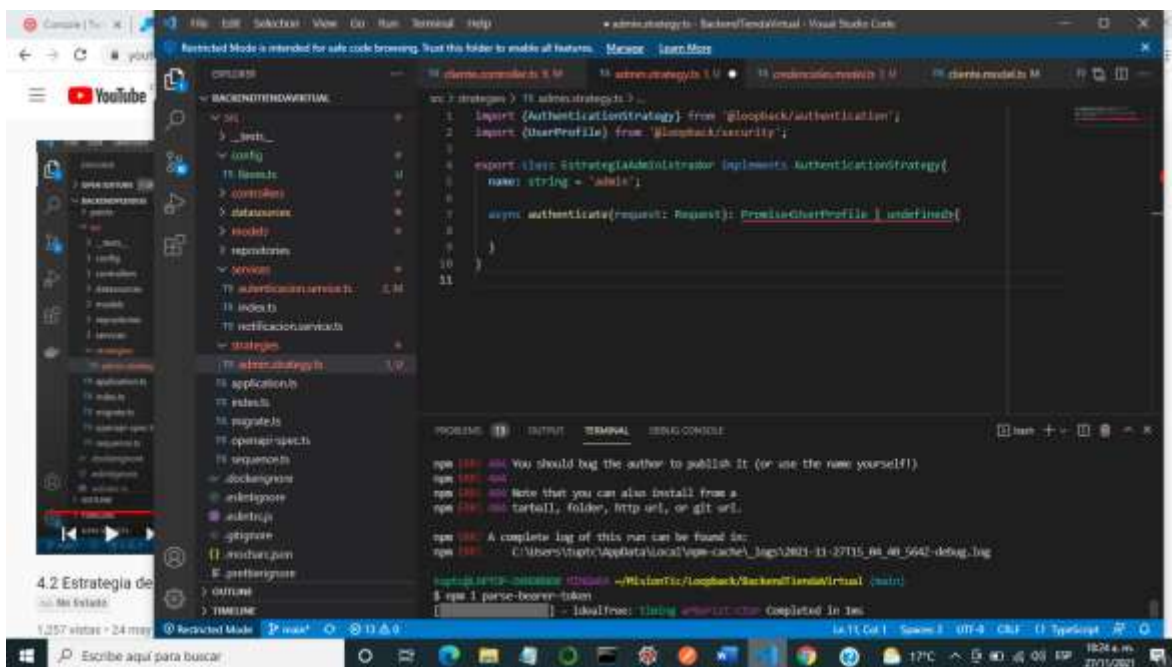
Ahora vamos a instalar otro paquete `npm i --force @loopback/security` este nos permite tener acceso al perfil del usuario



Creamos una nueva carpeta estrategias y dentro de esta creamos una fila `admin.strategy.ts` en el cual crearemos la estrategia para implementar un usuario administrador o usuario valido



Instalamos el paquete **npm i parse-bearer-token**

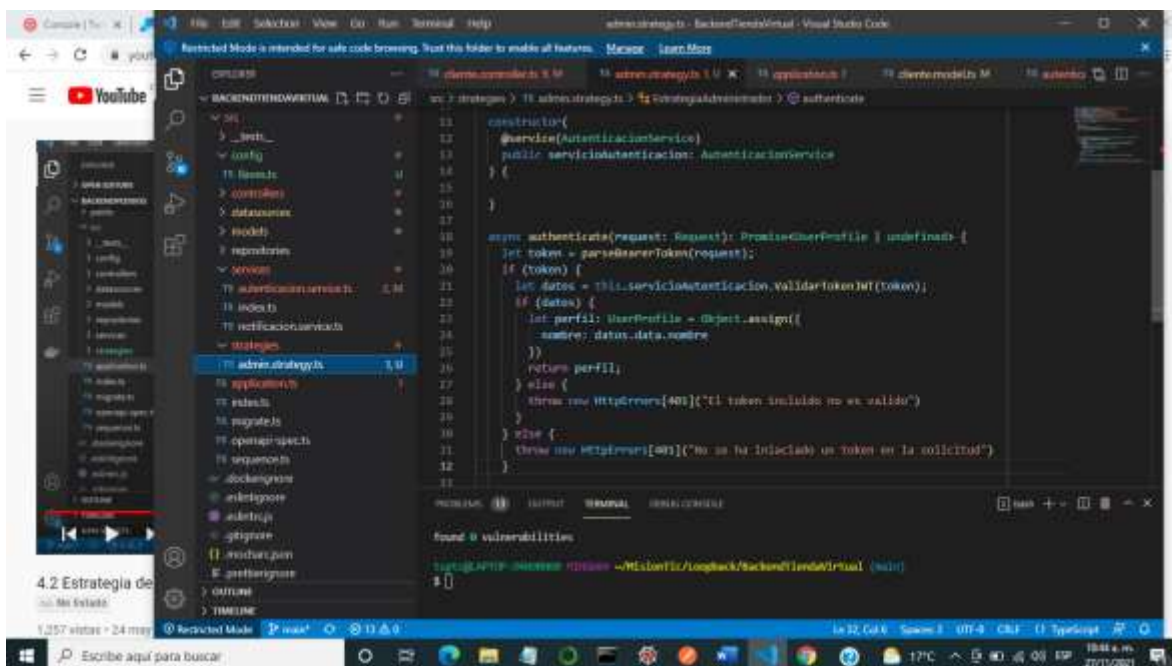


The screenshot shows a Visual Studio Code editor with the following content:

- File Explorer:** Shows the project structure with folders like `src`, `config`, `controllers`, `datastores`, `models`, `repositories`, `services`, `authentication`, `notification`, `strategies`, `application`, `index`, `register`, `openapi-specs`, `sequences`, `dockerignore`, `eslintignore`, `eslint`, `gitignore`, `nodemon.json`, and `package.json`.
- Source Explorer:** Shows the file `src/strategies/11 adminstrador.ts` selected.
- Code Editor:** Contains the following TypeScript code:

```
1 import { AuthenticationStrategy } from '@loopback/authentication';
2 import { UserProfile } from '@loopback/security';
3
4 export class StrategyAdminstrador implements AuthenticationStrategy {
5   name: string = 'admin';
6
7   async authenticate(request: Request): Promise<UserProfile> | undefined {
8     // Implementation of the authenticate method
9   }
10 }
```
- Terminal:** Shows the command `npm i parse-bearer-token` being executed, with output indicating the package was installed successfully.

Ya se genero el método authentication que ejecuta el método administrador. Recordemos que es necesario validar el rol

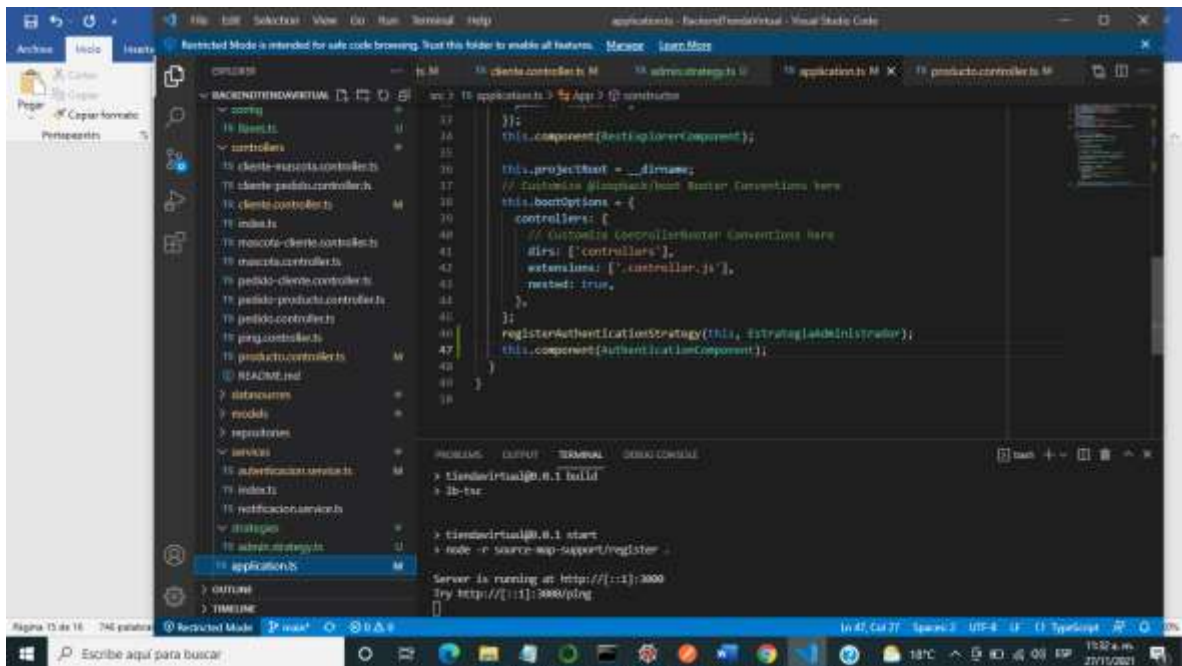


The screenshot shows the same Visual Studio Code editor with the following content:

- File Explorer:** Same as the previous screenshot.
- Source Explorer:** Shows the file `src/strategies/11 adminstrador.ts` selected.
- Code Editor:** Contains the following TypeScript code:

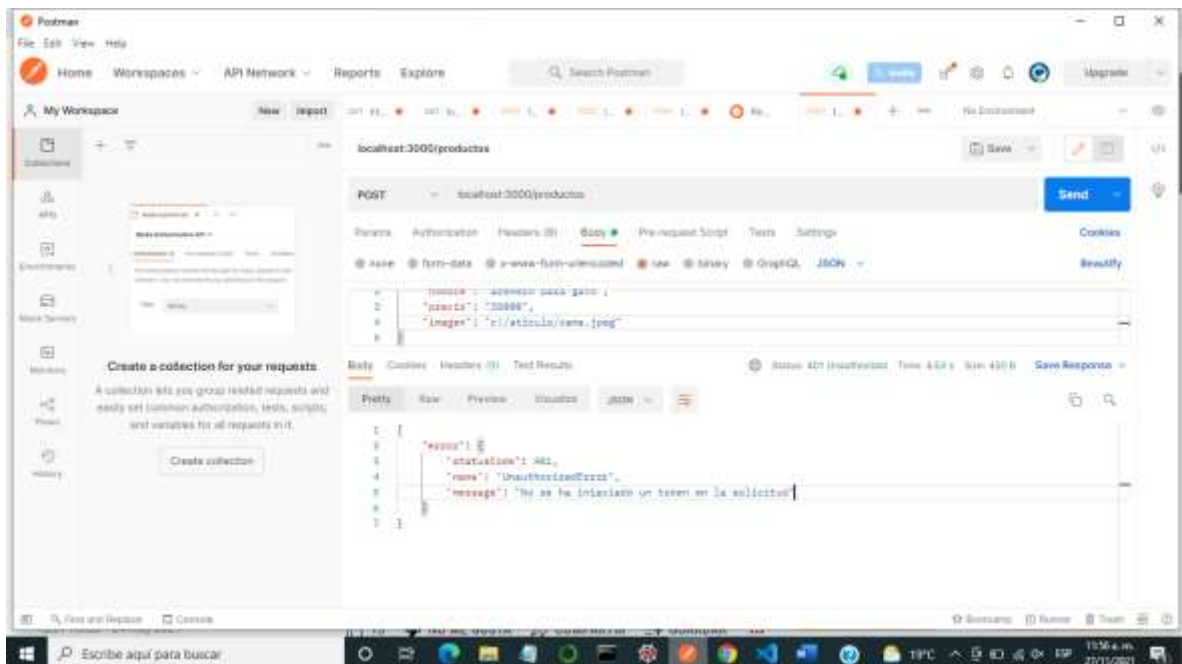
```
11 constructor() {
12   @service(AutenticacionService)
13   public servicioAutenticacion: AutenticacionService;
14 }
15
16 async authenticate(request: Request): Promise<UserProfile> | undefined {
17   let token = parseBearerToken(request);
18   if (token) {
19     let datos = this.servicioAutenticacion.validarTokenJWT(token);
20     if (datos) {
21       let perfil: UserProfile = Object.assign(
22         {},
23         { nombre: datos.nombre }
24       );
25       return perfil;
26     } else {
27       throw new HttpErrors(401)('El token incluido no es valido');
28     }
29   } else {
30     throw new HttpErrors(401)('No se ha incluido un token en la solicitud');
31   }
32 }
```
- Terminal:** Shows the command `npm i parse-bearer-token` being executed, with output indicating the package was installed successfully.

Ahora vamos incluir nuestras estrategias en aplicacion.ts después de tener las estrategias implementadas donde podamos validar el rol de los datos.



Configuramos esta vez la creación del producto, la cual es necesario que una persona este registrada

Validamos el funcionamiento de la aplicación



Mediante el manejo del token podemos proteger todos los métodos de la clase ejemplo cliente, producto. De la misma forma se puede o no proteger una clase, es decir asignarle a un método de la clase la asignación o no de un token.

