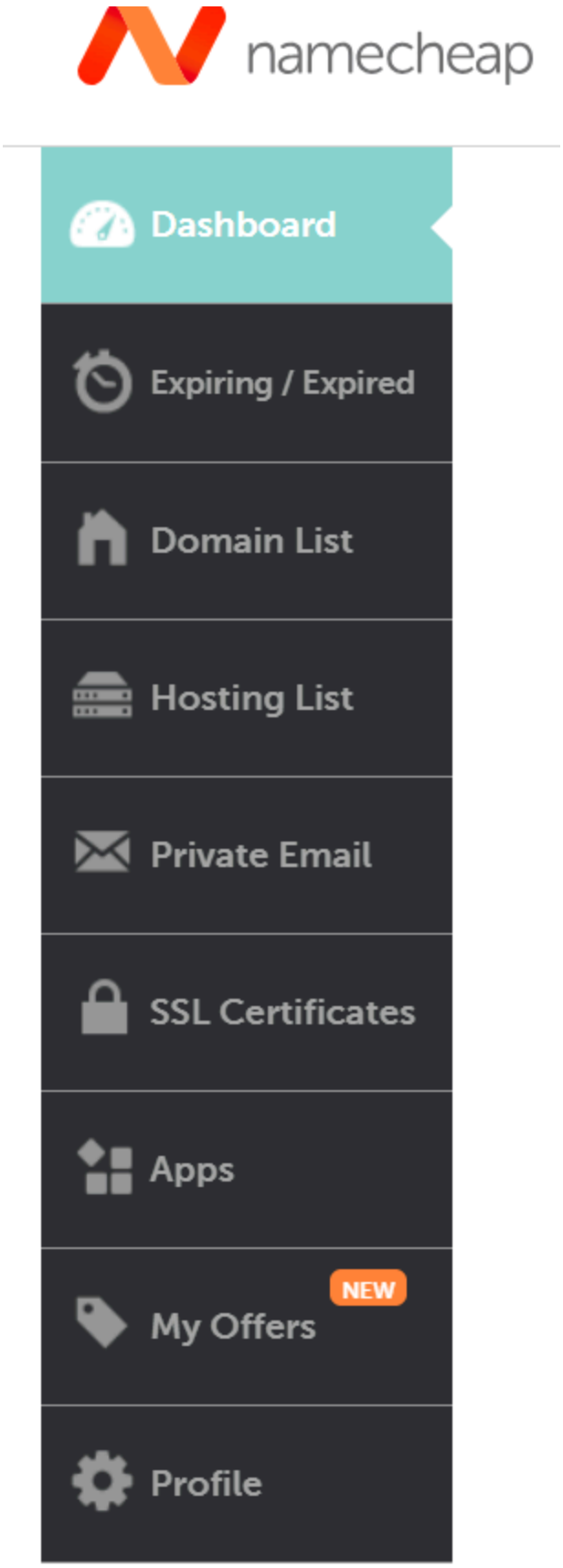
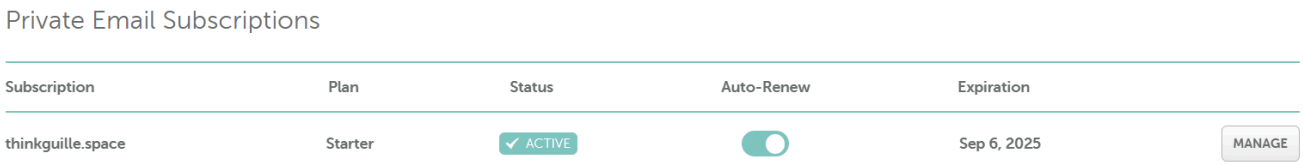
Dentro de namecheap tendremos una opción de agregar a nuestro dominio un servicio de correos, este será fácil de configurar y de usar dentro de nuestras aplicaciones web.

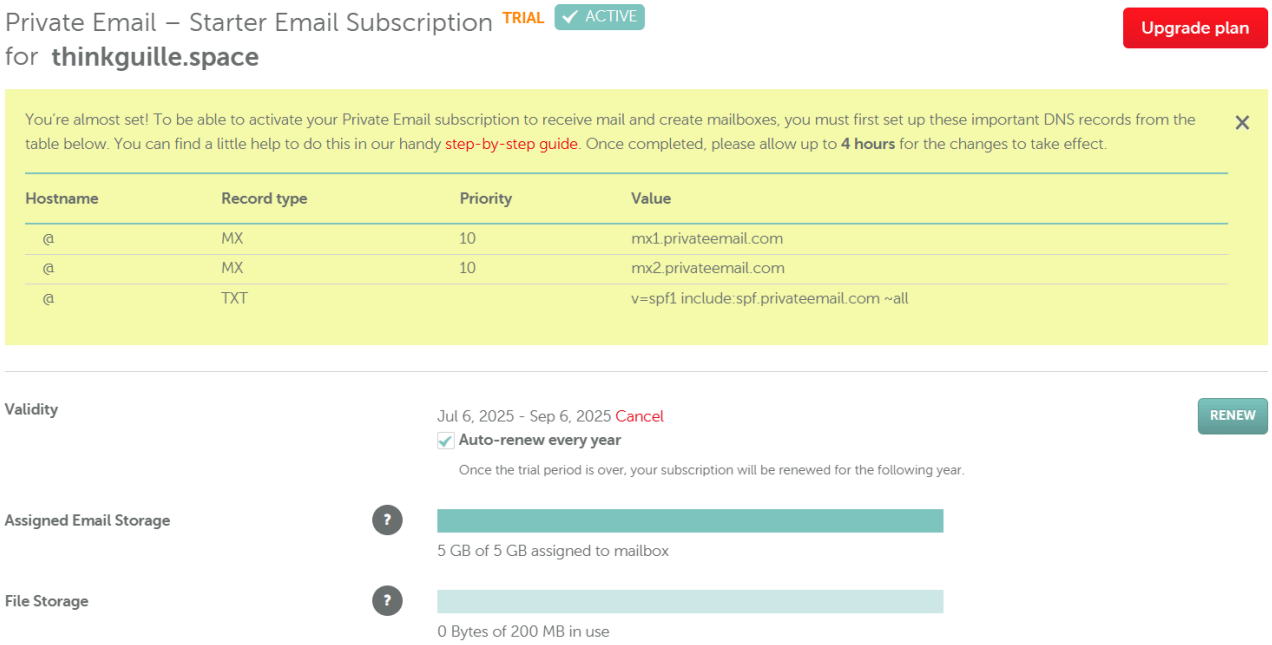


Cuándo nosotros estemos dentro de **NameCheap** veremos que tendremos de lado del lateral izquierdo una serie de opciones con los cuáles podemos personalizar o configurar nuestro dominio.

En el apartado que nos enfocaremos será el de Private Email:



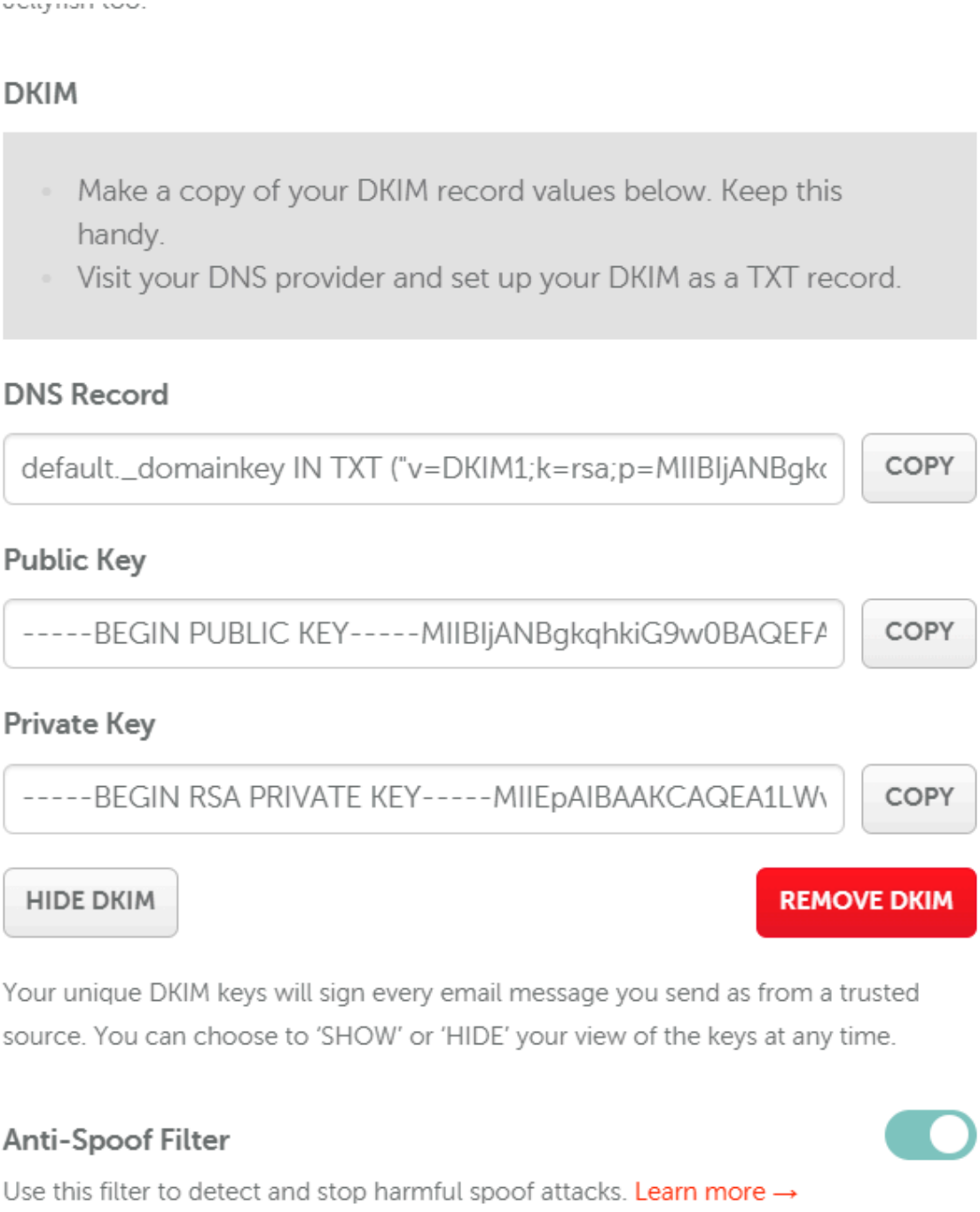
Esto será el lugar dónde nosotros podremos ver nuestras subscripciones activas, en caso de que seamos nuevos usando este servicio veremos que nos ofrece un plan de prueba gratuito, una vez que nosotros "compremos" este plan veremos que la interfaz cambia



Aquí nos dirá nuestro plan y nos dirá que es importante tener tanto un mailbox configurado, cómo agregar a nuestro entregador de dominio nuestra configuración esencial para usar el protocolo de correos.

Cloudflare

En mi caso manejé cloudflare, por lo que de manera automática este servicio jaló la configuración básica de mi subscripción y la indexó dentro de mi zona (entorno), por lo que lo único que debí meter de manera manual es una llave de dominio y una llave DKIM, esta es accesible si nosotros vamos a la parte inferior de nuestro plan de correos:



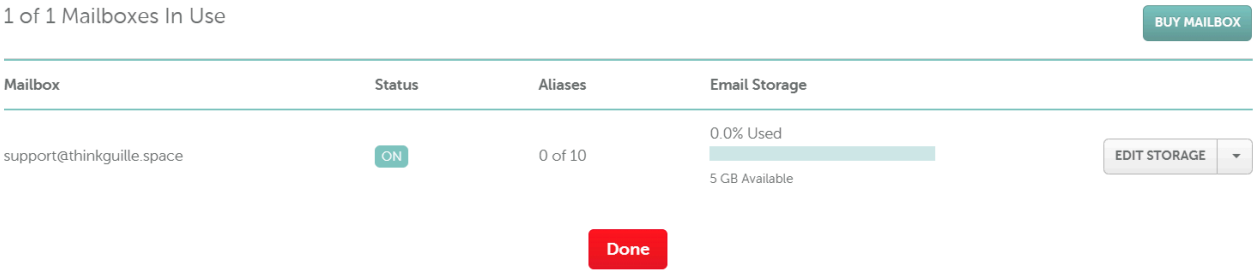
Esa llave la pondremos como archivos TXT dentro de nuestro perfil de Cloudflare

Tipo	Nombre	Contenido	Estado de proxy	TTL	Acciones
A	thinkguille.space	161.35.56.53	Solo DNS	Automático	Editar
A	www	161.35.56.53	Solo DNS	Automático	Editar
TXT	default._domainkey	"v=DKIM1; k=rsa; p=MII...	Solo DNS	Automático	Editar
TXT	_dmarc	"v=DMARC1; p=quaranti...	Solo DNS	Automático	Editar

Bastará con poner estas dos llaves, para que reconozca el tipo MX para los correos.

Servicio de MailBox

Al momento que nosotros escogemos nuestro paquete va a pedir que registremos el correo y la contraseña de nuestra cuenta, esa por default será "NoReplay@nuestrodominio", una vez que terminemos de configurar esta cuenta podremos acceder a ella.

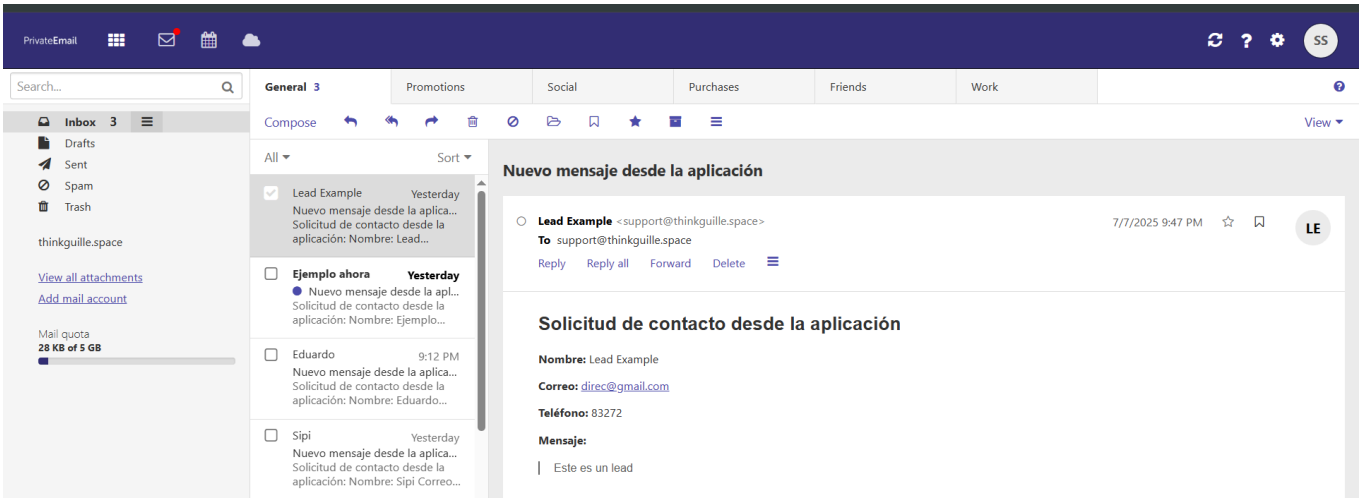


Podemos tener la cantidad que nosotros deseemos, siempre y cuándo paguemos por ellas, así como si queremos que tengan más o menos capacidad de disco para los mensajes, o que habilitemos la opción de envío desde el servicio, irán alterando el precio que pagaremos por este mismo.

Liga de acceso: https://privateemail.com

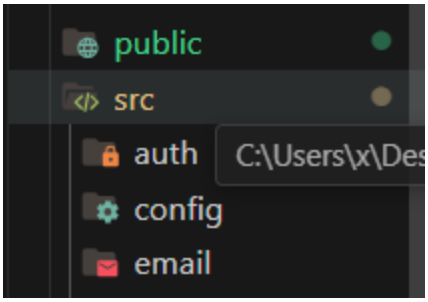
Aquí pondremos las credenciales que configuramos al momento de crear nuestro servicio de correo.

Una vez que accedemos podremos ver todos nuestros correos.

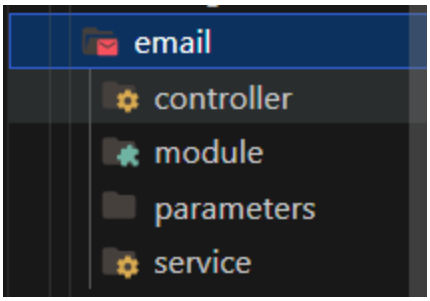


Configurando el back

Para consumir este servicio dentro de nuestra aplicación podemos manejarlo de la manera directa o estructurada, al ser un servicio de consumo externo, nosotros podemos jugar un poco con la manera en que hacemos que este cargue en nuestro servidor.



Yo decidí ponerlo dentro de la carpeta "email", no es obligatorio, pero es parte de un estándar el dividir estas secciones según su funcionalidad.



Aquí entra la parte de la adaptabilidad a nuestras necesidades, ya que podemos trabajar el modelado en capas o trabajar toda la lógica en nuestro controlador.

Herramientas adicionales

He decidido que podemos aprovechar la conexión STP que nos genera por defecto el servicio (cuentas institucionales), para aprovechar las ventajas que nos ofrece **NodeMailer**

Con esta dependencia nosotros solamente deberemos crear nuestro constructor y generar así un entorno dirigido.

Aquí es importante que prestemos atención a la documentación de esta librería, ya que depende de nuestro servicio será el puerto que deberemos activar y si ponemos o no el modo seguro

Link de la documentación: https://www.npmjs.com/package/nodemailer

---

Template.ts

```
import { CreateFormDto } from "src/formulario/dto/formulario.dto";

export function createEmailParameters(data: CreateFormDto) {

    const { nombre_completo, correo, telefono, mensaje } = data;



    const plainText = `

Solicitud de contacto desde la aplicación:



Nombre: ${nombre_completo}
```

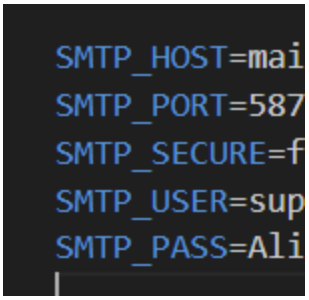
```
Correo: ${correo}

Teléfono: ${telefono}



Mensaje:

${mensaje}

    `.trim();



    const html = `

        <html>

            <body style="font-family: Arial, sans-serif; color: #333;">

                <h2>Solicitud de contacto desde la aplicación</h2>

                <p><strong>Nombre:</strong> ${nombre_completo}</p>

                <p><strong>Correo:</strong> ${correo}</p>

                <p><strong>Teléfono:</strong> ${telefono}</p>

                <p><strong>Mensaje:</strong></p>

                <blockquote style="background:#f9f9f9; padding:10px; border-
left:5px solid #ccc;">

                    ${mensaje}

                </blockquote>

            </body>
```

```
        </html>

    `;



    return {

        email: 'support@thinkguille.space',

        name: nombre_completo,

        message: plainText,

        html: html,

        title: 'Nuevo mensaje desde la aplicación',

    };

}
```

En este apartado, solamente vamos a manejar la estructura de nuestro correo, cómo queremos pintar este en nuestras notificaciones y el email al que irá dirigido, este será el que nosotros configuramos en nuestra caja de correos.

Protección

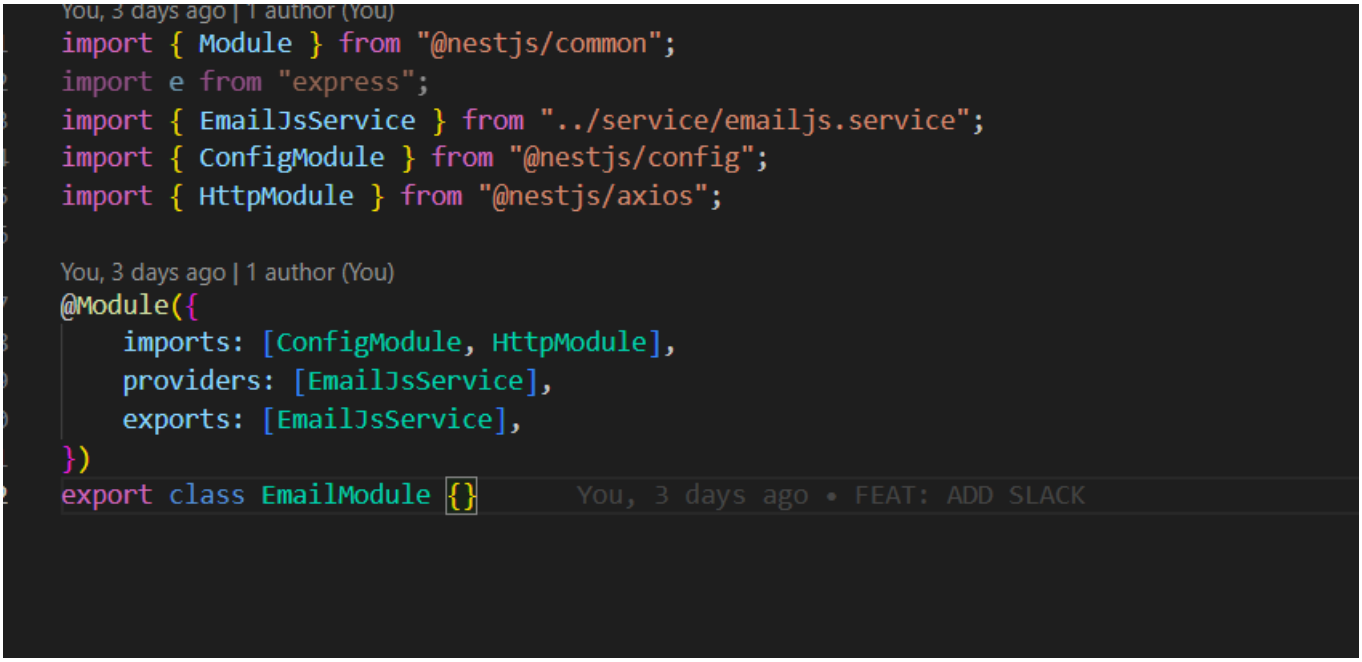
Aunque en nuestro constructor pudimos ver los parametros, por estándar de seguridad es recomendable que migremos esa configuración al .env en caso de que queramos deployar en producción:



Configuraciones adicionales

En mi caso mi servidor es ssh, por lo que si queremos que nuestras peticiones salgan de manera correcta, debemos especificar en nuestro archivo

Email.Module.ts



Que esperamos una configuración HTTP y funciones Axios adicionales

Imports

Vamos a declarar tanto módulos como servicios en la cofiguración de

App.module.ts

```
@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
      load: [recaptchaConfig, slackConfig, emailConfig],
    }),      eduardo87DR, 2 weeks ago • fix:validation and key …

    TypeOrmModule.forRoot({
      type: 'sqlite',
      database: process.env.DATABASE_PATH || './db.sqlite',
      entities: [User, Lead],
      synchronize: true, // Solo en desarrollo
    }),

    FormularioModule,
    RecaptchaModule,
    AuthModule,
    EmailModule
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

De esta manera haremos que se pueda acceder manera global a todas las configuraciones que nosotros hayamos definido.

Uso

Por último bastará con que declaremos nuestra variable y definamos en que momento esperamos que se dispare nuestro servicio para mandar nuestro correo de la app a la caja de correos.

Formulario.controller.ts

```
    await this.slackService.sendMessageToSlack(data);
    await this.emailService.sendEmail(data);
```