



Proyecto Final Programación

PASSWORD MANAGER

Joan Pastor Vicéns
a 23 de Noviembre de 2024



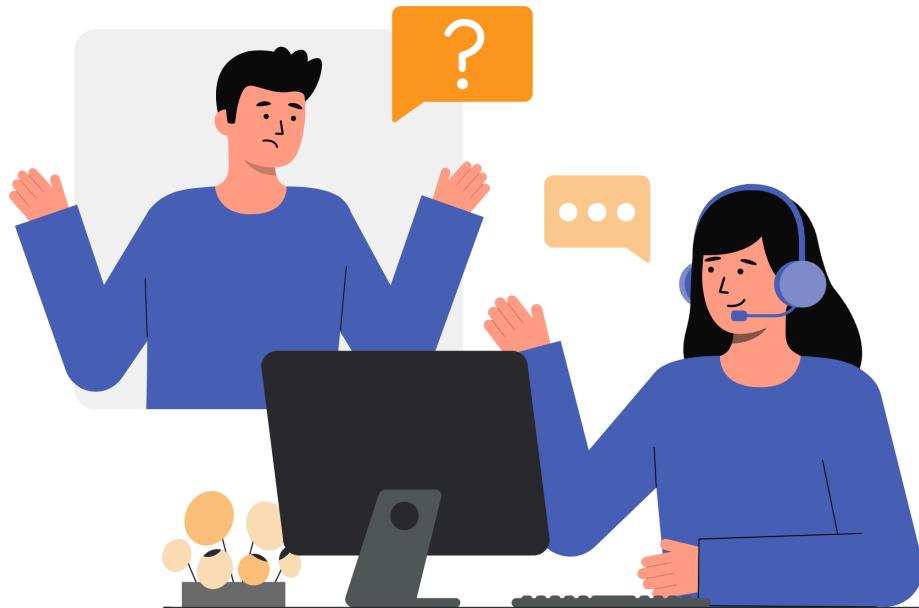
Índice

Presentación general del proyecto ¿Qué quiero hacer?	2
Gestor de contraseñas	3
Objetivos y alcance del proyecto	4
Organización del directorio principal	5
Programación orientada a objetos	8
Manejo de una base de datos	20
Autenticación de usuarios	27
Encriptación y desencriptación	29
Manejo del correo electrónico	35
Validaciones	44
Inyección de Datos	52
Testeo de archivos	56
Interfaz gráfica	59
Stack tecnológico principal	65
Stack tecnológico secundario	66
Alternativas evaluadas	68
Modelo de datos	72
Creación de usuario	76
Inicio de sesión	77
Reseteo de contraseña	78
Adición de elementos	80
Carga de elementos	81
Principales casos de uso	82
Creación de usuario	82
Inicio de sesión	84
Reseteo de contraseña	86
Adición, edición y eliminación de elementos	91
Envío de correo electrónico	97
Eliminación de la cuenta	99
Generadores	102
Manual de instalación	104
Conclusiones	112
Evolutivos del proyecto	113
Bibliografía	116
Recursos	119

Presentación general del proyecto | ¿Qué quiero hacer?

Tras un año académico altamente enfocado en la utilización y aprendizaje del lenguaje **Python**, me siento capaz de realizar la siguiente propuesta: Un gestor de contraseñas.

Dicho proyecto me ayudará a sentar las bases de la programación aprendida a lo largo del año, enfocándose en la utilización, preparación y manejo de bases de datos, trabajar con conceptos nuevos como la autenticación de usuario, la encriptación de información y la creación de un entorno gráfico agradable para cualquier usuario medio.



Gestor de contraseñas

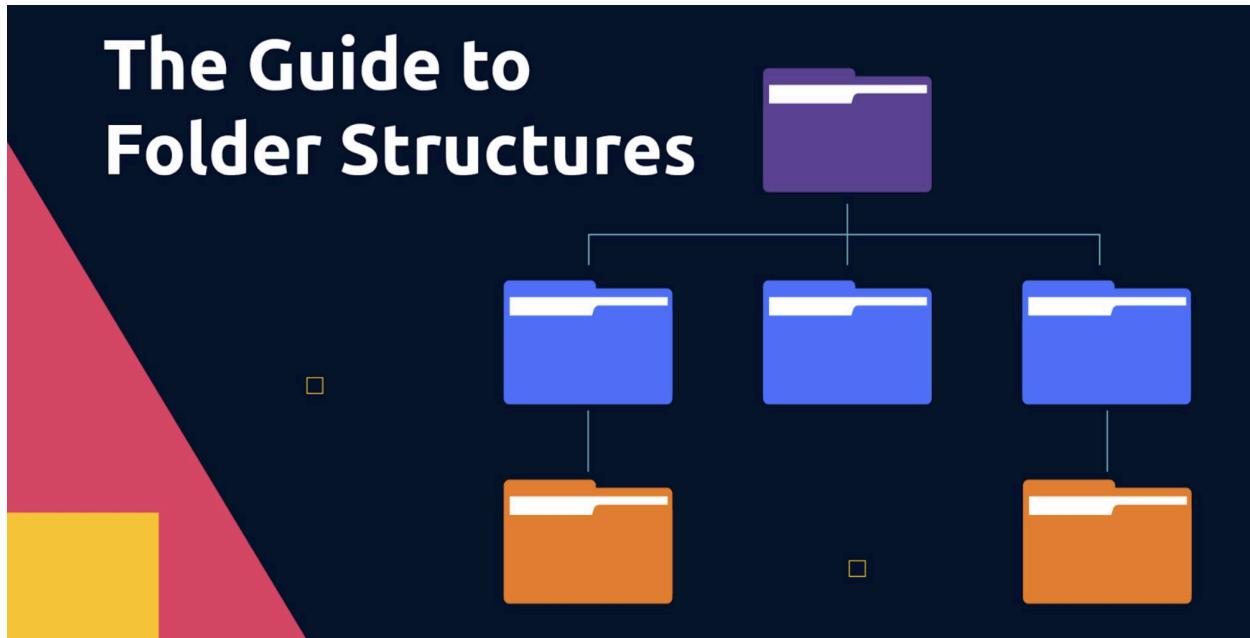
El mundo actual ha avanzado lo suficientemente rápido como para que más del 90% de las acciones se realicen a través de la computadora siendo la creación de credenciales lo más importante para ello. Aunque el aumento de la ciberseguridad ha proporcionado maneras sencillas de almacenar dichas credenciales, siguen siendo multinacionales privadas las que manejan dichos datos. Las políticas de privacidad implementadas suelen ser unos documentos largos y poco intuitivos para el usuario medio, haciendo que éste las acepte sin tan siquiera terminar de leerlas y obviando, por tanto, qué ocurre con sus datos.

Es por ello que mi propuesta es la realización de una aplicación de escritorio que sea capaz de almacenar contraseñas o información importante de forma cifrada, ya sea el acceso a cualquier tipo de cuenta web, tarjetas de crédito o notas seguras.



Objetivos y alcance del proyecto

- **Organización de un proyecto real:** Al ser un proyecto de gran envergadura, deberemos mantener la carpeta principal organizada para asegurar una correcta escalabilidad del proyecto.
- **Programación Orientada a Objetos:** Utilizaremos la programación orientada a objetos para delimitar de forma abstracta los elementos con los que vamos a trabajar.
- **Manejo de una base de datos:** Dicha aplicación deberá disponer de una base de datos que almacene correctamente los elementos introducidos por el usuario.
- **Autenticación de Usuarios:** Como la principal funcionalidad de la aplicación es almacenar tarjetas bancarias, usuarios, contraseñas y notas seguras, deberá disponer de autenticación para poder mostrar los elementos de cada usuario registrado.
- **Envío de Correo Electrónico:** Funcionalidad basada en el reseteo de contraseña para cuando el usuario olvide sus credenciales.
- **Encriptación y desencriptación de información:** Al almacenar información sensible, deberemos implementar un sistema de encriptación y desencriptación de información.
- **Diseño de una interfaz gráfica:** Por último y no menos importante, deberemos implementar una interfaz gráfica moderna y agradable para el usuario medio y fácil de entender.



Organización del directorio principal

La estructuración de un proyecto se entiende como la organización de carpetas y subcarpetas necesarias para las diferentes funcionalidades que dispondrá una aplicación. Dicho enfoque es fundamental para la escalabilidad del mismo pero claro, **¿cómo sabemos dónde almacenar cada elemento?** [Arquitecturas*](#) como la *Hexagonal*, la *Espagueti*, por Capas, *DDD* (*Domain Driven Design*) entre otras suelen ser las más usadas. Su elección recae simplemente en cómo se va a plantear nuestro proyecto.

Como en nuestro caso no es un proyecto muy grande, hemos optado por la arquitectura por capas, que se encarga de dividir las funcionalidades del programa de la siguiente manera:

1. Presentación (directorio: *interface*)

En esta capa se incluyen todos los elementos de la interfaz visual (como sus páginas, formularios, *widgets*, fuentes o imágenes) y sus respectivos controladores (elementos de control del usuario como botones o barras de navegación y menús).

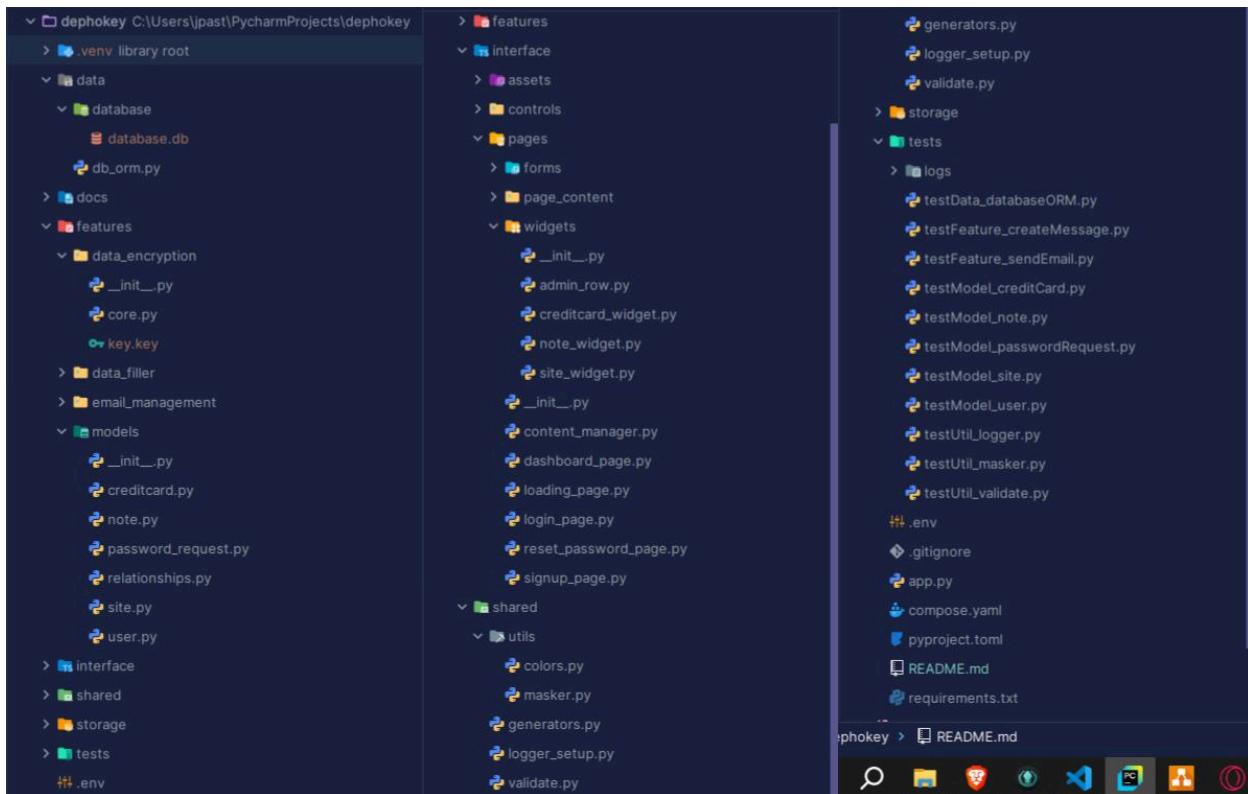
2. Lógica de negocio (directorio: *features*)

Este directorio contiene todos los modelos que se encarguen de la creación de objetos (como las clases de cada uno de los elementos a almacenar en el base de datos) y los servicios o herramientas (en este caso la herramienta de encriptación de los elementos y el servicio de reseteo de la contraseña).

3. Acceso a los datos (directorio: *data*)

Aquí se colocan todos los elementos que se encargan del manejo de la base de datos y la propia base de datos. Al estar separada de todo lo demás, facilita la migración de dicha base de datos a otro tipo de lenguaje.

Teniendo en cuenta estas [reglas*](#), podemos observar que el directorio principal desglosado mostraría su contenido de la siguiente forma:

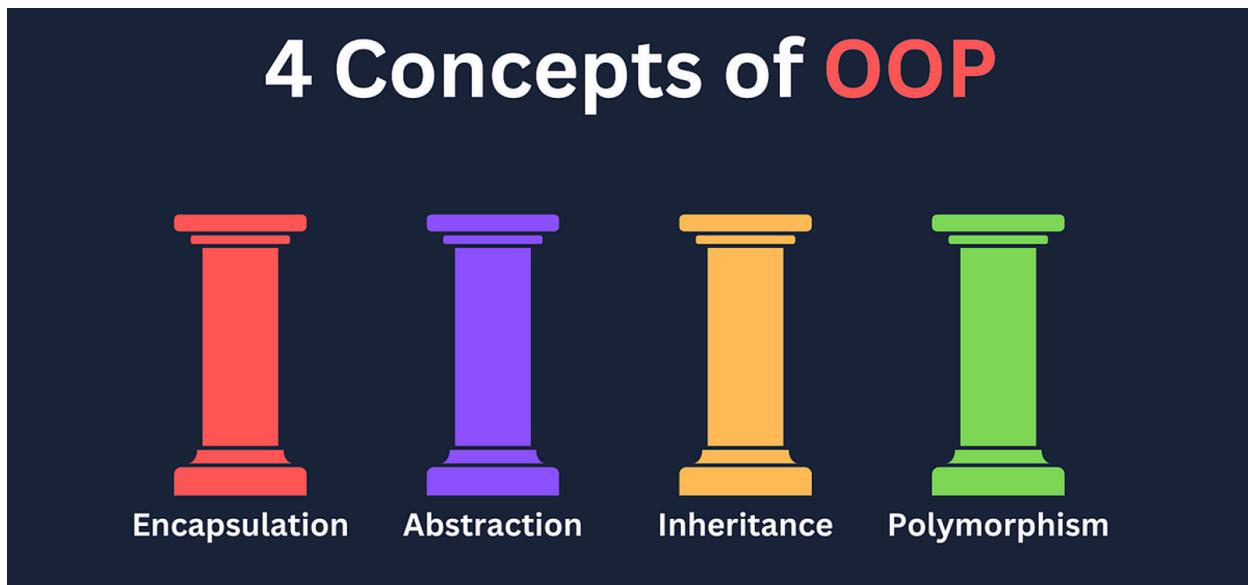


También crearemos un entorno virtual para controlar todas las librerías necesarias en un entorno seguro, una carpeta para realizar los tests necesarios para cerciorarnos de que no hay fallos en nuestro código si realizamos algún cambio y trataremos de mantener el directorio [organizado*](#) al ir añadiendo más archivos.

Programación orientada a objetos

¿Qué es la programación orientada a objetos? Este [paradigma*](#) de programación está basado en el concepto de clases y objetos. Esta forma de concebir el código facilita la estructuración del diseño de software en pequeños bloques de código haciéndolos reutilizables. Sin embargo, la POO necesita estar formada por cuatro pilares fundamentales como son la encapsulación, la abstracción, la herencia y el polimorfismo.

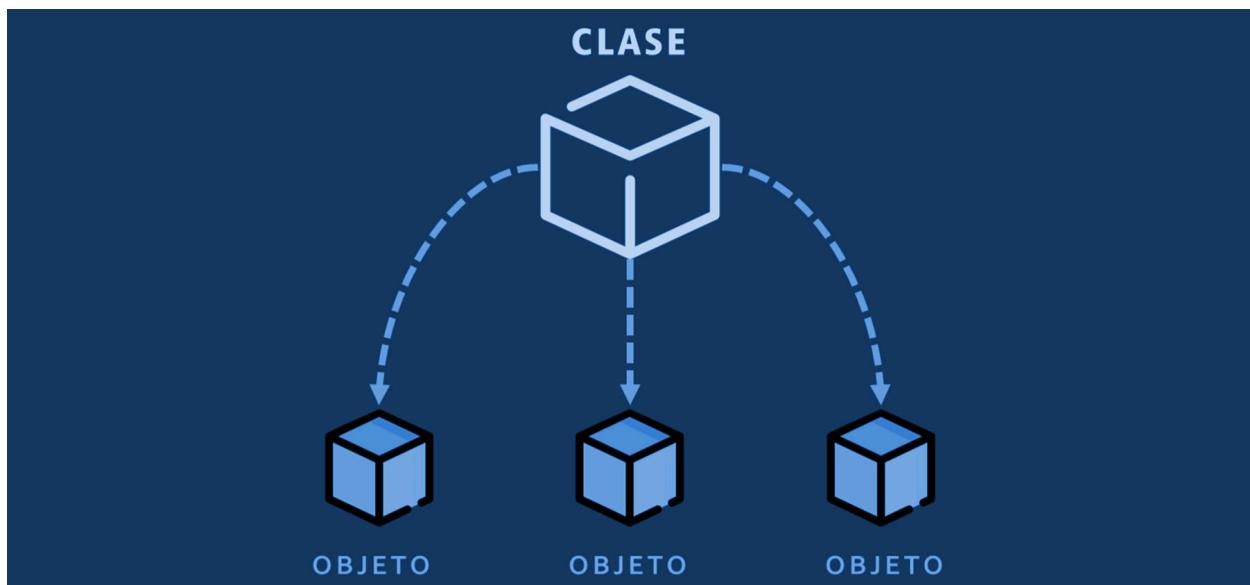
Esto nos ayuda muchísimo en sistemas grandes, ya que en vez de pensar en funciones, pensamos en las relaciones o interacciones de los diferentes componentes del sistema.



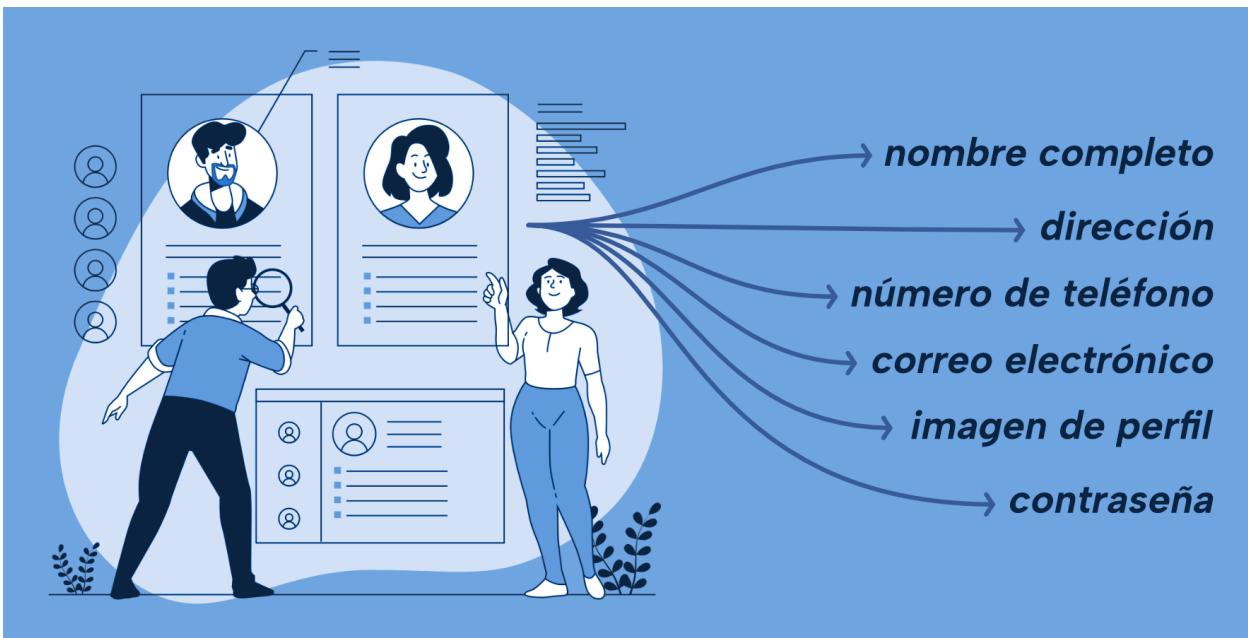
- **Encapsulación:** Esta característica nos asegura que cualquier clase u objeto puede tener elementos públicos (cualquiera fuera de la clase puede verlos, modificarlos o eliminarlos), elementos protegidos (cuyo contenido no puede ser modificado por elementos externos a la clase pero sí se pueden heredar) y elementos privados (cuyo contenido sólo puede ser llamado por la propia clase).

- **Abstracción:** La abstracción en la programación orientada a objetos se realiza simplificando los elementos de un objeto real. Esta funcionalidad permite la utilización de métodos y atributos sólo en el contexto necesario del programa, simplificando la creación del código.
- **Herencia:** La herencia es la capacidad que tiene el código de poder utilizar los métodos o atributos de una superclase desde su subclase. Esta funcionalidad permite la reutilización de código y puede extender las funcionalidades de la misma.
- **Polimorfismo:** Por último, esta característica permite la sobreescritura de métodos o atributos haciendo que la subclase cambie su comportamiento según se desee, facilitando la reutilización de código.

Es por ello que debemos pensar en cómo serían los elementos a utilizar de manera abstracta y funcional, dotándolos de una serie de atributos y funcionalidades necesarios según su comportamiento.



Creación del usuario



Para la creación de un usuario hay que tener en cuenta qué elementos pueden ir asociados a él. En este caso es necesario dotar a dicho usuario de un ID único cuya funcionalidad será ser la clave foránea de los demás elementos. Se le añade un rol dentro de los dos necesarios como Administrador o Cliente. También debe tener un nombre completo, un correo electrónico válido y una contraseña para que el usuario pueda acceder a sus datos. Por último, incluimos una fecha de creación para saber en qué momento se designó este nuevo usuario.

En la clase `User`, concretamente en su inicializador, podemos ver como los elementos se van asociando a los atributos especificados de la instancia:



```
user.py

# Initializer
def __init__(self, fullname: str, email: str, password: str, user_role: UserRole = UserRole.CLIENT) -> None:
    super().__init__()

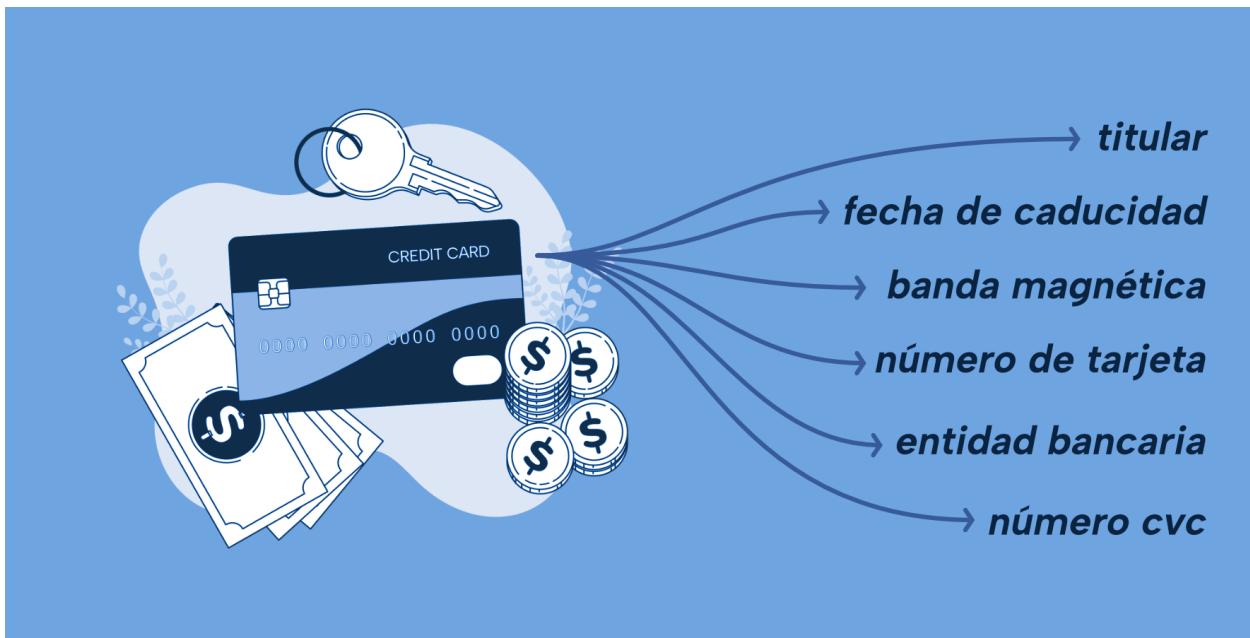
    self.id: str = GenerateID.short_id()
    self.role: UserRole = user_role
    self.fullname: str = fullname
    self.email: str = email
    self.hashed_password: str = sha256(password.encode()).hexdigest()
    self.created: datetime = datetime.today()

    # Logs new user
    log.info(f"Instancia de USER creada por {repr(mask_email(self.email))}.")


```

La función `__init__()` se encargará de recibir dichos parámetros, asociarlos a sus respectivos atributos y, finalmente, instanciar el objeto. ¡Usuario creado!

Creación de la tarjeta de crédito



Ahora probemos con algo más tangible. Cuando hablamos de una tarjeta de crédito es más fácil saber qué elementos debería de tener dicha clase. Aunque en la imagen anterior se pueden ver la mayoría de atributos necesarios, para simplificar la situación obviaremos unos cuantos.

También debemos añadirle tres atributos más que no se pueden ver, como su propio ID, el ID de usuario y una variable booleana que indique si está caducada o no. Para concluir con esta clase, podemos afirmar que la tarjeta debe estar estrictamente formada por un ID único para identificarla, un ID del usuario que la ha creado (su clave foránea), el nombre del titular de la tarjeta, su número, el código CVC / CVV, la fecha de validez y su fecha de creación. Además,

podemos agregar un alias para la tarjeta para ayudar al usuario a identificarla. Por tanto, la clase `CreditCard` quedaría tal que así:



```
creditcard.py

# Initializer
def __init__(self,
             cardholder: str,
             number: str,
             cvc: str,
             valid_until: datetime,
             user: User,
             alias: str | None = None
) -> None:
    super().__init__()

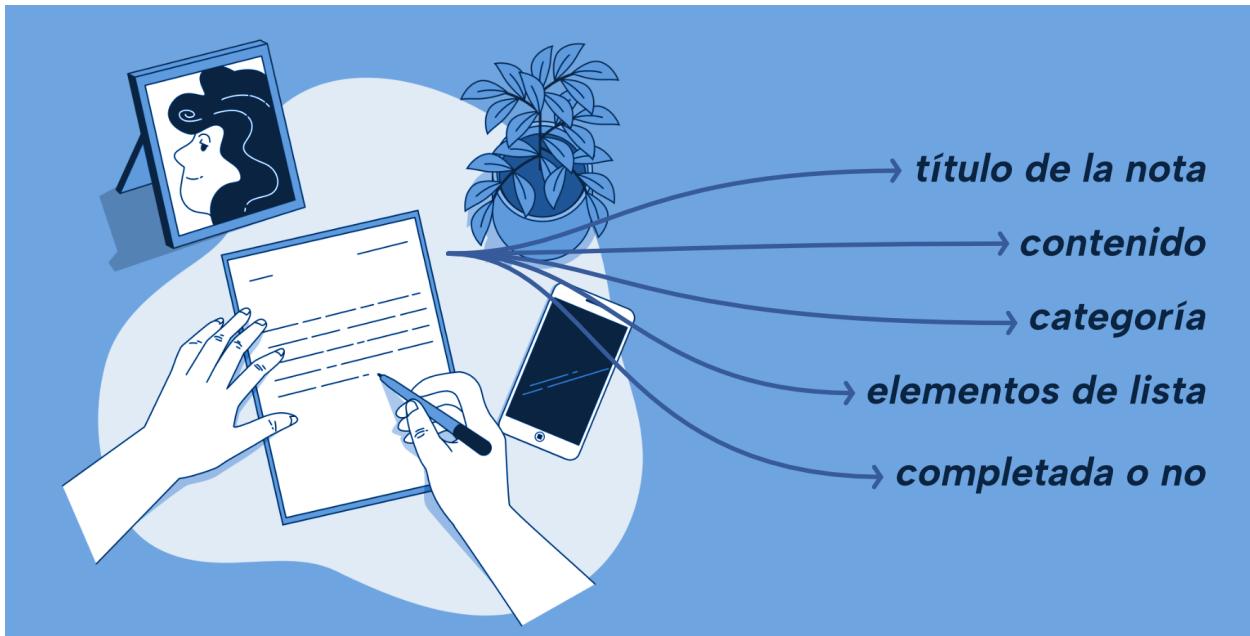
    self.id: str = GenerateID.short_id()
    self.cardholder: str = cardholder
    self.encrypted_number: str = encrypt_data(number)
    self.encrypted_cvc: str = encrypt_data(cvc)
    self.valid_until: datetime = valid_until
    self.expired: bool = False
    self.alias: str | None = alias
    self.user: User = user
    self.created: datetime = datetime.today()

    self.update_expired()

# Logs new creditcard
log.info(f"Instancia de CREDITCARD creada por {repr(mask_email(self.user.email))}.")
```

Tan solo haría falta enviar cada uno de esos datos a sus respectivos atributos y nuestro *builder* se encargará de instanciarlo. ¡Tarjeta de crédito creada!

Creación de la nota segura



Este objeto está formado por dos conceptos un tanto abstractos. En nuestro caso tan sólo necesitamos que dicha instancia almacene un título y un contenido. Se podrían añadir más atributos como sus tipos (notas de seguridad como códigos externos, de documentos oficiales o acreditaciones), pero por ahora nos quedaremos con lo básico.

Para crear [Note](#) utilizamos los mismos atributos que en las clases anteriores: Su propio ID y el ID de usuario. Por otro lado, sus atributos principales son el título que permitirá dotarla de un elemento identificativo para el propio usuario y un contenido que almacena su información encriptada. También hay que incluir una fecha de creación.

Así, podemos terminar con que nuestro objeto de clase `Note` se puede representar de la manera en que se muestra a continuación:



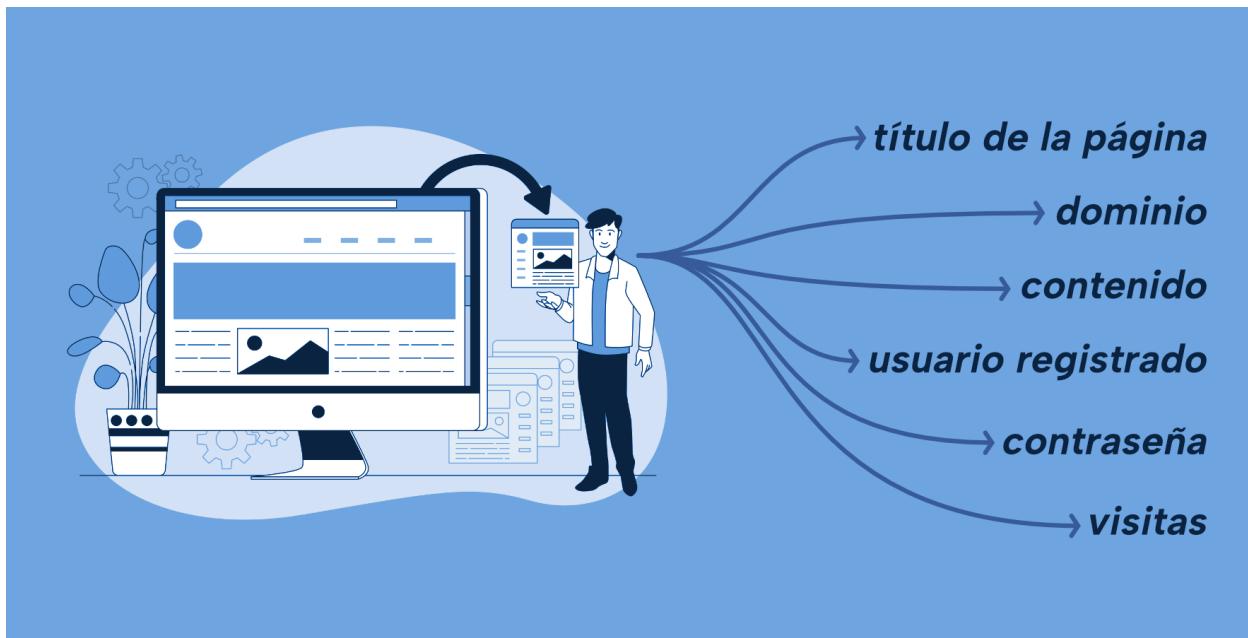
```
● ● ● note.py

# Initializer
def __init__(self, content: str, user: User, title: str | None = None) -> None:
    super().__init__()

    self.id: str = GenerateID.short_id()
    self.title: str | None = title
    self.encrypted_content: str = encrypt_data(content)
    self.user: User = user
    self.created: datetime = datetime.today()

# Logs new note
log.info(f"Instancia de NOTE creada por {repr(mask_email(self.user.email))}.")
```

Creación del sitio web



Para almacenar los datos de una página web hay que tener en cuenta su abstracción y pensar qué elementos necesita el usuario, pues están formadas de muchas características. En nuestro caso, el usuario sólo quiere almacenar sus credenciales así que no hay muchos datos que almacenar.

Como se puede observar, a parte de necesitar todos los atributos anteriores que tenemos en común (como la ID única, la ID de usuario, y la fecha de creación), nuestro objeto de tipo [Site](#) dispone de una dirección web válida y de las credenciales del usuario (el nombre de usuario y la contraseña con la que se ha registrado en dicha web). Podemos colocar también un nombre o

título para ayudar al usuario a encontrarla más fácilmente pero no es estrictamente necesario. Así que podemos concluir que nuestro inicializador quedará así:

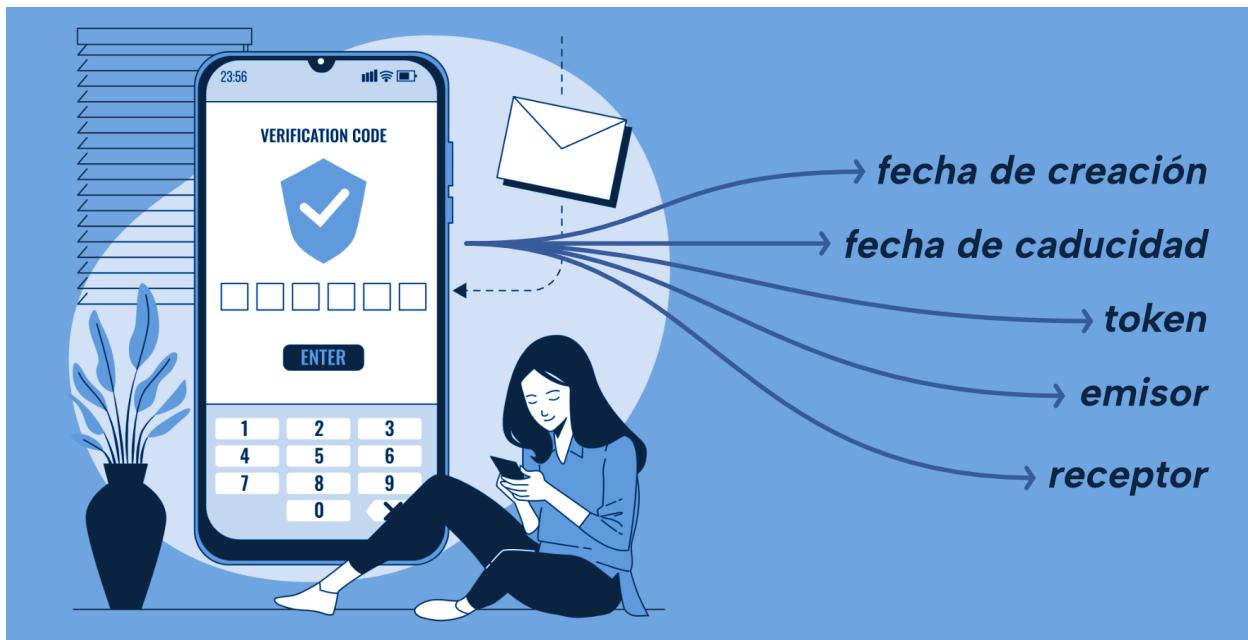
```
● ● ● site.py

# Initializer
def __init__( self,
              address: str,
              username: str,
              password: str,
              user: User,
              name: str | None = None
) -> None:
    super().__init__()

    self.id: str = GenerateID.short_id()
    self.name: str | None = name
    self.address: str = address
    self.username: str = username
    self.encrypted_password: str = encrypt_data(password)
    self.user: User = user
    self.created: datetime = datetime.today()

    # Logs new note
    log.info(f"Instancia de SITE creada por {repr(mask_email(self.user.email))}.")
```

¡Sitio web creado con éxito!

Creación de la petición de cambio de contraseña

¿Qué es esta clase? Este es el aspecto más abstracto y del cual no poseemos un ejemplo tangible. Nuestra [PasswordRequest](#) es una clase que se encarga de almacenar códigos de seguridad encriptados que se envían a los usuarios para poder actualizar sus credenciales cuando no consiguen acceder al programa.

Al ser una clase que almacena una solicitud solo es necesario incluir un ID identificativo, el propio código de seguridad encriptado, el ID de usuario al que está asociado, su fecha de creación y una fecha de caducidad. Es importante tener en cuenta que dicha fecha de caducidad debe almacenar también horas, minutos y segundos (por ello se utiliza *datetime* y no

date) porque según el tiempo almacenado, podemos implementar una comparación de datos que confirme si dicho *token* ha expirado. Por tanto, nuestra clase se muestra así:



```
# Initializer
def __init__(self, user: User) -> None:
    super().__init__()

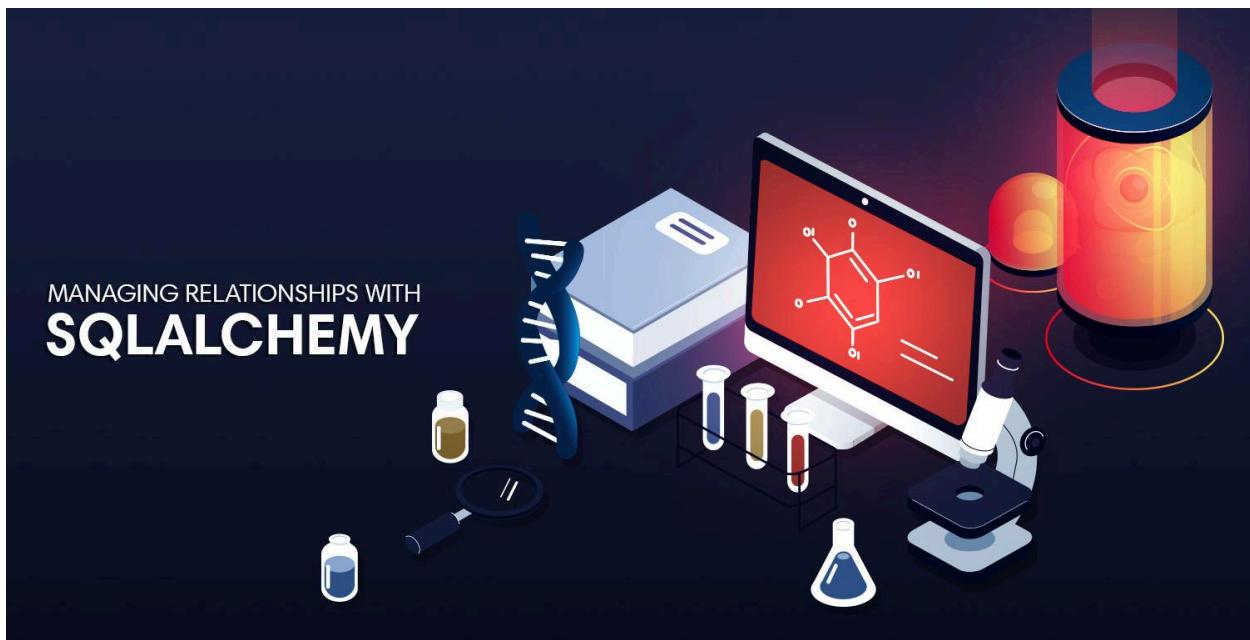
    self.id: str = GenerateID.short_id()
    self.encrypted_code: str = encrypt_data(GenerateToken.generate())
    self.user: User = user
    self.created: datetime = datetime.today()
    self.expires_at = datetime.now() + timedelta(minutes=5)

    # Logs new note
    log.info(f"Instancia de PASSWORD REQUEST creada por {repr(mask_email(self.user.email))}.")
```

Todas las clases anteriores, aunque no se vea explícitamente, tienen un atributo que las asocia a la clase `User` el cual dispone de tres atributos que no se requieren en el inicializador por ser una clase mapeada, pero que son igual de importantes: son cuatro listas de elementos a ser `notes` (formada por elementos de tipo `Note`), `creditcards` (formada por elementos de tipo `CreditCard`), `sites` (formada por elementos de tipo `Site`) y `password_change_requests` (formada por elementos de tipo `PasswordRequest`).

Manejo de una base de datos

¿Qué es un ORM? Tal y como su nombre indica, un **ORM*** es un *Object Relational Mapping* (o Mapeo de Objetos Relacionales). Su principal misión es facilitar la comunicación entre un programador y la base de datos, ya que él mismo se encarga de adaptar las consultas en lenguaje **SQL** al lenguaje que está utilizando el programador. Estos objetos tendrán atributos y métodos que ayudarán a realizar la creación, lectura, modificación y eliminación (*CRUD* en inglés) de los mismos. Ahí es donde entra **SQLAlchemy**.



SQLAlchemy* es nuestro puente entre **Python** y nuestra base de datos **SQL**. Como nos proporciona una capa de abstracción entre ambos lenguajes, nos facilita la realización de consultas sin tener que cambiar parte del código. Además, si en un futuro decidimos cambiar de lenguaje, nos permite una migración de base de datos limpia y sin grandes esfuerzos.

Para poder manejar el intercambio de datos con nuestra base de datos. Primero debemos configurar la clase `Base`. Ésta se encarga de mapear cada una de las clases hijas para que el *ORM* sepa qué objetos deben introducirse en la *database* y qué métodos son accesibles.



```
# Create database settings
DB_PATH = Path(__file__).parent.joinpath("database")
DB_FILENAME = "database.db"
DB_TEST_PATH = Path(":memory:")

class Base(DeclarativeBase): new*
    """This class transform attributes to mapped data and link them to database table."""
    pass

class DatabaseManager: 2 usages ↗ Joan
    """
    Official documentation: https://docs.sqlalchemy.org/en/14/core/engines.html
    This class allows to handle a database with different settings.
    """

    def __init__(self, db_path: Path, db_filename: str | None = None, language: str = "sqlite") -> None:
        """
        Initialize a DatabaseManager instance.
        :param db_path: Path | Database's directory
        :param db_filename: str | The file's name
        :param language: str | Database language (sqlite by default)
        :return: None
        """

        # Create main attributes
        self.db_path = db_path
        self.db_filename = "" if db_filename is None else db_filename

        # Update database URL
        self.DB_URL: str = f"{language}:///{{self.db_path.joinpath(self.db_filename)}}"

        # Create database attributes
        self.engine: Engine | None = None
```

Después, y mediante la clase `DatabaseManager`, se asegura de que la carpeta donde

permanece la base de datos existe y no si no la encuentra, crea el directorio. Se prepara la base de datos y por último, se instancia el objeto `session` para poder interactuar con los datos.

Implementación en el modelo principal

```
user.py

class UserRole(Enum):  __Joan
    CLIENT = "client"
    ADMIN = "admin"

class User(Base):  __Joan
    """
    This class creates a new user instance. It needs an UserID, a Role, a CompleteName, a Username, a Password
    and a CreatedDate.
    """

    # Table settings
    __tablename__ = "user"

    # Column settings
    id: Mapped[str] = mapped_column(String(15), primary_key=True)
    role: Mapped[UserRole]
    fullname: Mapped[str] = mapped_column(String(250))
    email: Mapped[str] = mapped_column(String(250))
    hashed_password: Mapped[str] = mapped_column(String(100))
    created: Mapped[datetime]
```

Anteriormente vimos que en nuestros modelos faltaban datos. Bien, aquí es donde podemos ver cómo se configuran las clases para que el *ORM* pueda trabajar con ellas. Primero debemos configurar los elementos de la clase `User`, pues es nuestra clase principal y a la que se asocian las otras clases.

Primero se configura el nombre de la tabla que guarda esa clase de elementos (en este caso, "user") y después creamos unos atributos de clase que son de tipo *mapped*. Esta opción le indica al *ORM* que debe crear columnas en la tabla dada con los nombres de dichos atributos (`id`, `role`, `fullname`, `email`, `hashed_password` y `created`). Podemos ver el resultado en la siguiente imagen:

id	role	fullname	email	hashed_password	created
69t1bduxoegm3b0	ADMIN	Sergio Administrativo Muñoz	admin.24@gmail.com	60fe74406e7f353ed979f350f2...	2025-03-23 18:53:50.506813
gfhqblyfgbmnmvv	CLIENT	Manuel Tester García	client.24@gmail.com	e6d1c38f0d9d46c5512533d49...	2025-03-23 18:53:50.524812

Ahora bien, hay un problema muy importante y que puede desestabilizar la creación completa del proyecto: la [importación circular*](#). Este problema se presenta cuando una clase depende de otra que depende de la primera y esto ocurre ya que, en nuestro caso, se utiliza el tipado. En este ejemplo, los modelos secundarios dependen de la clase `User`, mientras que ésta depende de cada una de las otras clases restantes para configurar las relaciones.

Una forma de evitar esta importación circular es separar el tipo de relación (uno a muchos) del propio archivo principal (`user.py`) y añadirla después de la creación de las clases. Por ello utilizamos la forma en que **Python** maneja los paquetes. Mediante el archivo `__init__.py` que maneja la creación de las clases configuramos la creación de las relaciones en la clase `User`.

```
● ● ● __init__.py
from .creditcard import CreditCard
from .note import Note
from .password_request import PasswordRequest
from .relationships import user_relationships
from .site import Site

user_relationships()

__all__ = [
    "CreditCard",
    "Note",
    "PasswordRequest",
    "Site",
]
```

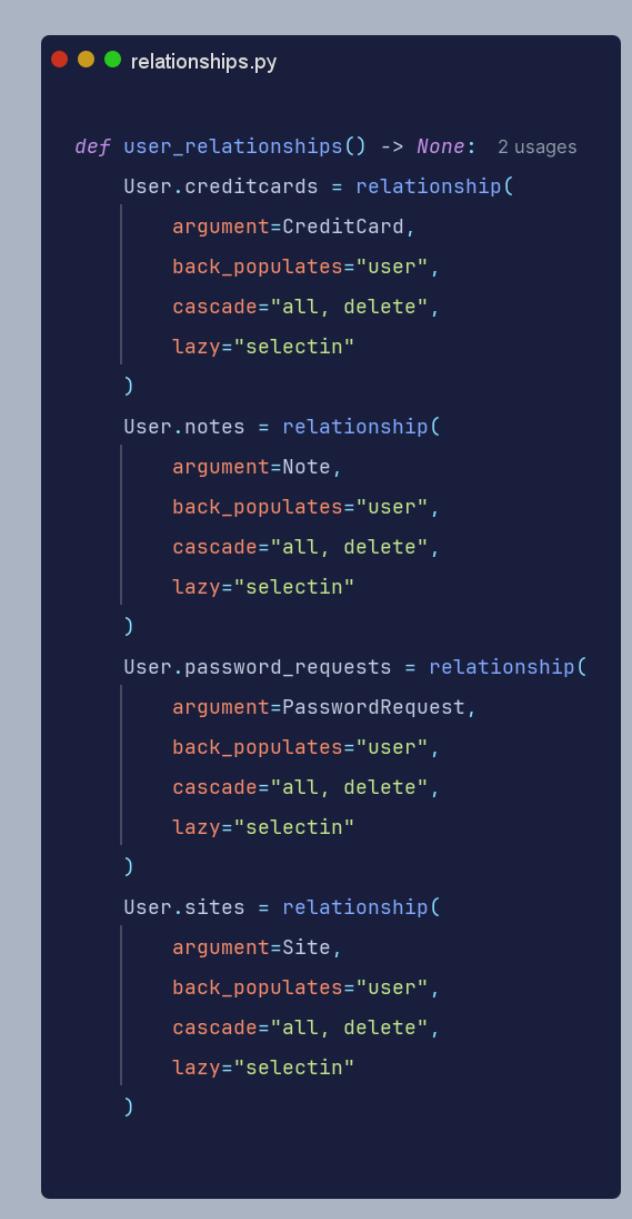
Como se ve en la imagen, el archivo `__init__.py` carga primero las clases principales (creándolas) y luego llama a la función `user_relationships()` cuya finalidad es implementar dichas relaciones. Por tanto hemos conseguido que se implementen las relaciones una vez finalice la creación de cada clase y no se lance un error de importación.

En la imagen de la derecha podemos ver cómo se implementa la relación de la tabla `user` con las demás tablas.

Estos atributos son de tipo lista lo que da a entender que desde la propia instancia de la clase `User`, podemos acceder a todos sus elementos tal cual se accede a cualquier lista en `Python`. Es el *ORM* el que se encarga de realizar la petición y traducir a `SQL` nuestras acciones.

Estas cuatro relaciones apuntan a cada una de las clases mediante el nombre utilizado en ellas. Es importante que dicho nombre que se emplee en la clase `User` sea el mismo utilizado en la `relationship()` de la clase asociada en caso contrario no se podrán comunicar entre ellas.

También damos un valor `"all, delete"` a `cascade` pues esta opción permite la eliminación en cascada de los datos al eliminar el usuario correspondiente.



```

def user_relationships() -> None: 2 usages
    User.creditcards = relationship(
        argument=CreditCard,
        back_populates="user",
        cascade="all, delete",
        lazy="selectin"
    )
    User.notes = relationship(
        argument=Note,
        back_populates="user",
        cascade="all, delete",
        lazy="selectin"
    )
    User.password_requests = relationship(
        argument=PasswordRequest,
        back_populates="user",
        cascade="all, delete",
        lazy="selectin"
    )
    User.sites = relationship(
        argument=Site,
        back_populates="user",
        cascade="all, delete",
        lazy="selectin"
    )

```

Implementación en los modelos secundarios



The screenshot shows a code editor window with a dark theme. At the top, there's a file icon with three colored dots (red, yellow, green) followed by the filename "site.py". The code itself is written in Python and uses SQLAlchemy's declarative base. It defines a class named "Site" which inherits from "Base". The class has several attributes and relationships defined. The code includes comments explaining the purpose of each part, such as creating a new site instance and defining table settings like the primary key and foreign keys.

```
site.py

class Site(Base): __joan

    """
    This class creates a new site instance. It needs an ID, a UserID, a Name, an Address, a UserName,
    a Password and a CreatedDate.
    """

    # Table settings
    __tablename__: str = "site"

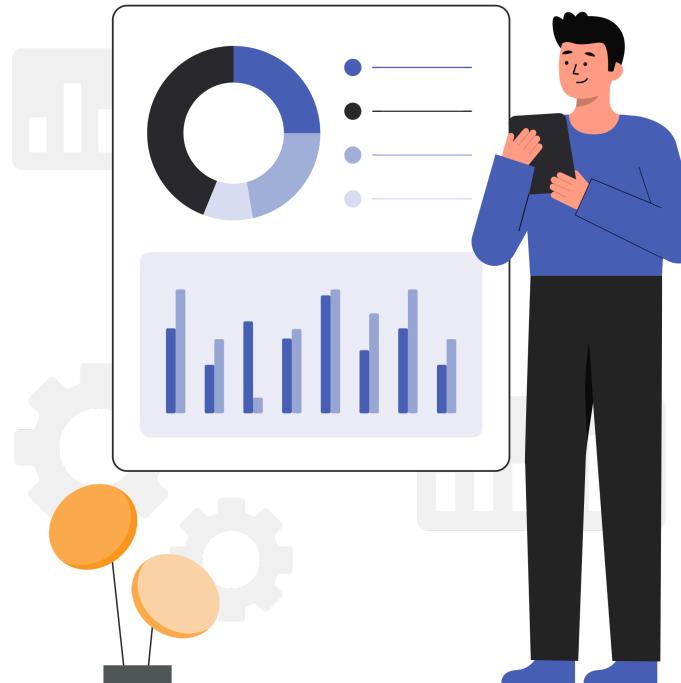
    # Column settings
    id: Mapped[str] = mapped_column(String(15), primary_key=True)
    user_id: Mapped[str] = mapped_column(
        String(15),
        ForeignKey(column="user.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )
    name: Mapped[str | None] = mapped_column(String(250))
    address: Mapped[str] = mapped_column(String(4100))
    username: Mapped[str] = mapped_column(String(250))
    encrypted_password: Mapped[str] = mapped_column(String(4100))
    created: Mapped[datetime]

    # Relationship settings
    user: Mapped[User] = relationship(
        argument="User",
        back_populates="sites"
    )
```

Aunque se trate solo de la clase `Site`, la imagen es para ver el contexto. La configuración es la misma para todas las clases que deben ir asociadas a `User`. Podemos observar que se le da un nombre a la tabla (`"site"`) y se añaden los atributos que se deben mapear para la creación de las columnas. Sin embargo, en el apartado *Relationship settings* podemos comprobar varias cosas:

La primera es que el atributo de clase `user` tiene el mismo nombre que se utiliza en el `back_populates` de la clase `User`, mientras que en esta `relationship()` se denomina `sites` (como en `User`). Esto hace que ambas tablas se apunten entre sí, permitiendo su comunicación.

La segunda es que `ForeignKey()` se encarga de configurar la clave foránea del objeto. En el siguiente ejemplo queremos que almacene el ID de usuario una vez se cree una instancia de clase. También permite la configuración de la acción que realizará el *ORM* cuando el usuario se elimine. La palabra clave "`CASCADE`" permite la eliminación del objeto cuando se elimine el usuario haciendo que todos los sitios asociados a la ID de éste sean borrados.



Autenticación de usuarios

Antes de nada, debemos entender el significado de la autenticación. **¿Qué es realmente? ¿En qué se basa y qué factores son necesarios?**

La [autenticación*](#) en sí es el proceso de confirmar que algo (en este caso, un usuario) tiene permisos para acceder a cierta información (sus datos almacenados). Se divide en dos apartados, el probador y el verificador. La parte que se identifica como probador suele ser el usuario que quiere acceder mientras que el verificador es el sistema que protege el acceso.

Como cualquier base de datos es vulnerable de ser atacada y por tanto, sus datos pueden estar comprometidos, la forma ideal de almacenar los datos sensibles es mediante el [hash*](#). Esta funcionalidad implica transformar la contraseña dada en una cadena alfanumérica usando algoritmos especializados y almacenar el resultado en la base de datos. Cuando el usuario quiere acceder, simplemente se comparan los *hashes* proporcionados (el almacenado, y el resultado de volver a hashear la contraseña introducida por el usuario).



Como se puede ver, parece una codificación similar a encriptar la contraseña pero, sin embargo, el *hash* es unidireccional. No se pueden devolver los datos a su contenido original y es por ello

que deberemos implementar una herramienta que permite el reseteo de la contraseña para que el usuario pueda volver a acceder al programa cuando no recuerde o pierda sus credenciales.



```
user.py
def __init__(self, fullname: str, email: str, password: str, user_role: UserRole = UserRole.CLIENT) -> None:
    super().__init__()

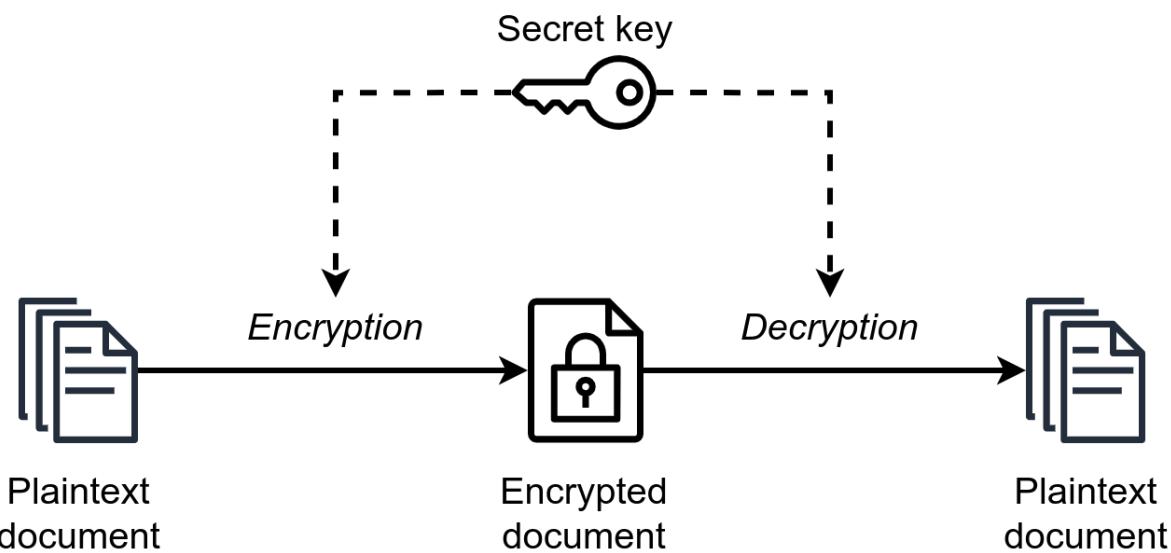
    self.id: str = GenerateID.short_id()
    self.role: UserRole = user_role
    self.fullname: str = fullname
    self.email: str = email
    self.hashed_password: str = sha256(password.encode()).hexdigest()
    self.created: datetime = datetime.today()
```

Para ahondar más en detalle, veamos cómo se construye nuestro usuario: Podemos ver que la tabla `User` almacena el atributo `hashed_password` directamente, asegurando que la base de datos desconozca la contraseña original. Mediante `sha256` (uno de los algoritmos que se utilizan para el *hash*), tan sólo almacenamos una cadena de caracteres aleatorios que representan dicha contraseña y que podemos comprobar en la imagen de abajo.

id	role	fullname	email	hashed_password	created
69t1bdximoegm3b0	ADMIN	Sergio Administrativo Muñoz	admin.24@gmail.com	60fe74406e7f353ed979f350f2...	2025-03-23 18:53:50.506813
gfhqblyfgbmnmvv	CLIENT	Manuel Tester García	client.24@gmail.com	e6d1c38f0d9d46c5512533d49...	2025-03-23 18:53:50.524812

Encriptación y desencriptación

El mundo de la protección de nuestros datos nos brinda soluciones como el *hashing* (codifica los datos en una línea de caracteres de un tamaño dado y que no se puede descifrar), el *salting* (que se realiza antes del *hashing* e implica la adición de información aleatoria a los datos a cifrar) o la [criptografía*](#) (cuya funcionalidad se basa en el encriptado y desencriptado mediante una clave dada).



Nuestro proyecto tiene en cuenta el almacenaje de contraseñas en la base de datos. Esta funcionalidad debe realizarse mediante un *hash* por seguridad, sin embargo, cuando el usuario almacena en la base de datos la contraseña de un sitio web, debe poder acceder a dicho dato. Por tanto, realizar un *hash* queda descartado ya que no se puede deshacer. Es por ello que empleamos una sencilla encriptación simétrica.

Esta clase de encriptación se basa en utilizar la misma clave secreta para cifrar y descifrar la información y, al no estar conectada a servidores o necesitar el envío de datos cifrados al exterior, perfectamente puede mantenerse segura dentro del propio equipo del usuario.

Creación de la clave de encriptación

Para asegurarnos de que la encriptación funcione con normalidad, debemos crear una clave segura y luego almacenarla en un archivo ***.key**. Esta funcionalidad (que podemos observar en la imagen siguiente) se almacena en el archivo **init_.py** para que sea la primera acción que se realice al importar el paquete de encriptación.



```
● ● ● __init__.py

# Key settings
__KEY_PATH = Path(__file__).parent
__KEY_FILE = "key.key"

def __create_key() -> None: 1 usage 2 Joan
    new_key = Fernet.generate_key()

    try:
        with open(__KEY_PATH.joinpath(__KEY_FILE).__str__(), "xb") as key_file:
            key_file.write(new_key)
        log.info(f"Archivo {repr(__KEY_FILE)} creado!")
    except PermissionError as permission:
        log.error(
            f"{type(permission).__name__} | "
            f"No se disponen de permisos para la creación del archivo {repr(__KEY_FILE)}: {permission}"
        )
    except FileExistsError as already_exists:
        log.error(
            f"{type(already_exists).__name__} | El archivo {repr(__KEY_FILE)} ya existe: {already_exists}"
        )
    except Exception as unknown:
        log.error(
            f"{type(unknown).__name__} | "
            f"Un error inesperado a ocurrido al tratar de crear el archivo {repr(__KEY_FILE)}: {unknown}"
        )

# Check if key already exists:
if __KEY_FILE not in os.listdir(__KEY_PATH):
    log.info(f"No se ha encontrado {repr(__KEY_FILE)}. Se procede a su creación.")
    __create_key()
```

El incluir esta parte del código en el archivo `__init__.py` no es una casualidad. Esta clase de fichero informa a **Python** de que el directorio en el que se encuentra en realidad es un módulo y siempre debe ejecutar este archivo. Así nos aseguramos de que compruebe si existe el archivo `key.key` y si no lo encuentra, que lo cree.

Función '`load_key()`'



```
core.py

def __load_key() -> bytes | None: 2 usages ↗ Joan

    # Prepare key for encrypt
    try:
        with open(f"{KEY_PATH}/key.key", "rb") as key_file:
            key = key_file.read()
        return key

    except FileNotFoundError as not_found:
        log.error(f"{type(not_found).__name__} | No se ha encontrado el archivo 'key': {not_found}")
    except Exception as unknown:
        log.error(
            f"{type(unknown).__name__} | Un error inesperado a ocurrido al tratar de abrir el archivo 'key': {unknown}"
        )
```

La función `__load_key()` se encuentra en el archivo `core.py` de nuestro módulo de encriptación. Su principal trabajo es cargar en memoria la clave almacenada y cuenta con su propio manejo de excepciones para evitar posibles fallas al momento de abrir el archivo. La doble barra baja del principio de su nombre le indica a **Python** que se trata de una función privada y no se puede llamar desde fuera del propio fichero.

Como podemos observar, en esta función podemos retornar dos valores: el primero es un valor de tipo *bytes*, que en este caso sería la clave leída. En el supuesto caso de que no la pueda leer, por defecto retorna *None*. Más adelante veremos qué hacer con dichos datos.

Función 'encrypt_data()'

```
core.py
def encrypt_data(new_data: str) -> str: 20 usages  ↵ Joan
    data_coded = new_data.encode(encoding="utf-8", errors="replace") # 'Replace' changes character by official '◊'
    key = __load_key()

    # Prepare data_encryption module
    try:
        fernet = Fernet(key)

        # Encrypt data
        encrypted_data = fernet.encrypt(data_coded)
        return encrypted_data.decode()

    except TypeError as current_type:
        log.error(f"{type(current_type).__name__} | No se ha podido encriptar el archivo: {current_type}")
        return data_coded.decode()
    except Exception as unknown:
        log.error(
            f"{type(unknown).__name__} | Un error inesperado ha ocurrido al tratar de encriptar el archivo: {unknown}"
        )
        return data_coded.decode()
```

La función `encrypt_data()` se encarga de recibir los datos que queremos encriptar. Es importante que dichos datos sean de tipo *string* para poder castearlos más adelante. Una vez recibidos, se codifican a datos de tipo *bytes*. Este paso es importante, pues la encriptación sólo funciona con este tipo de datos.

Después, dentro del bloque `try – except`, se intentan encriptar los datos. Si la lectura de la `key` son `bytes`, se puede encriptar correctamente. En cambio, si es `None`, entonces vuelve a codificar los datos, se devuelve el `string` original y se crea un registro donde se explica el error acontecido mediante nuestro `logger`.

Función '`decrypt_data()`'



```
core.py

def decrypt_data(load_data: str) -> str: __Joan
    data_coded = load_data.encode()
    key = __load_key()

    # Prepare data_encryption module
    try:
        fernet = Fernet(key)

        # Decrypt data
        decrypted_data = fernet.decrypt(data_coded)
        return decrypted_data.decode(encoding="utf-8", errors="replace")

    except TypeError as current_type:
        log.error(f"{type(current_type).__name__} | No se ha podido desencriptar el archivo: {current_type}")
        return data_coded.decode()
    except Exception as unknown:
        log.error(
            f"{type(unknown).__name__} | "
            f"Un error inesperado ha ocurrido al tratar de desencriptar el archivo: {unknown}"
        )
        return data_coded.decode()
```

La funcionalidad `decrypt_data()` prácticamente dispone de la misma configuración que la anterior pero en sentido inverso. Recibe como argumento un `string` cifrado (que viene directo de

la base de datos), carga la clave en memoria y si no puede hacerlo, simplemente devuelve los datos originales.

Para terminar con la sección cabe resaltar que, al ser una funcionalidad que se encarga de abrir, leer y cerrar archivos, además de tratar de cambiar su contenido, debemos implementar el bloque *try – except* para un correcto manejo de archivos y evitar que nuestro programa se bloquee. Si algún fallo ocurre durante la encriptación, tan solo almacena los datos tal y como han sido introducidos por el usuario y deja constancia del fallo en nuestro registro.

ADVERTENCIA: Esta funcionalidad es un claro fallo de seguridad, pues almacena la información sensible sin cifrar en la base de datos. Es por eso que se recomienda no almacenar credenciales verídicas en el programa. Tan sólo es una aplicación diseñada para el testeo de las habilidades aprendidas.

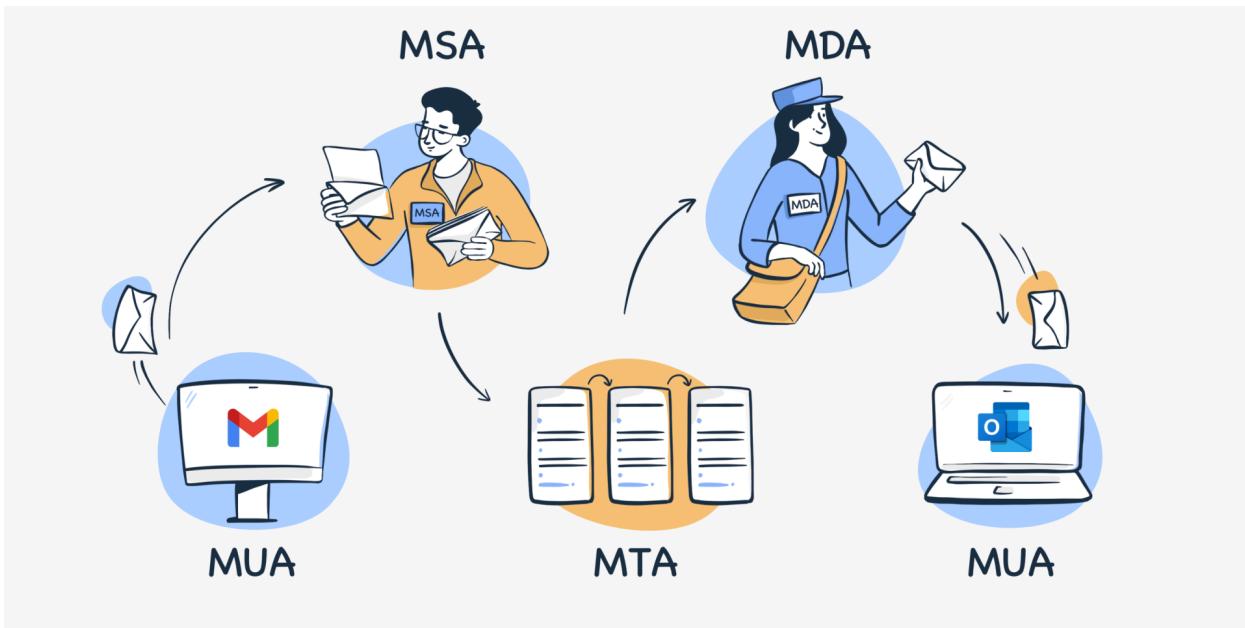
Manejo del correo electrónico

Para entender cómo manejar el envío de correos electrónicos, primero debemos entender cómo funciona el [correo electrónico*](#). **¿Cómo está formado?** La dirección de correo se crea a partir de una parte local (como nombre de usuario o apodo), el símbolo @ y el dominio que indica dónde se aloja la cuenta de correo electrónico.

Una vez tenemos la dirección, necesitamos de un protocolo para poder enviar y recibir correos. Para ello, utilizaremos el protocolo [SMTP*](#) (o Protocolo Simple de Transferencia de Correo). Este lenguaje de intercambio de mensajes electrónicos entre ordenadores u otros dispositivos trabaja como si de un cartero se tratase:

1. El servidor del correo del remitente inicia una conexión con el protocolo *SMTP*.
2. Dicho protocolo se encarga de que el mensaje disponga de un emisor y un receptor válidos. Después, mediante el sistema de nombres de dominio o *DNS*, traduce el nombre de ambos dominios a una dirección *IP*.
3. Ahora se encarga de buscar un servidor de intercambio de correo electrónico asociado al nombre del dominio del destinatario. Si existe, se manda a dicho servidor.
4. El correo electrónico queda almacenado en el servidor del destinatario y puede acceder a él mediante dos protocolos de oficina de correo: El *POP3*, que descarga el correo en el dispositivo desde el que accede el usuario y lo elimina del servidor o el *IMAP*, que deja el correo en el servidor y permite al usuario acceder a él desde cualquier dispositivo.

Ahora que ya sabemos qué hay detrás del envío y recepción de un correo electrónico, podemos empezar a configurar nuestro servicio en ambos sentidos (del usuario final a la aplicación y viceversa).



Ante todo hay que dar constancia de que no vamos a trabajar ni con servidores ni con correos reales, sino que vamos a simularlo. Por tanto debemos de utilizar una herramienta que intercepte los mensajes enviados para que no interfieran en el correcto funcionamiento de la aplicación. Es por ello que, mediante *docker*, crearemos un servidor cuya principal herramienta es interceptar esos correos enviados.

Para ello, simplemente debemos configurar un archivo ***.yaml** que incorpore una imagen de *mailcatcher* y que cree una página web con la que podemos ver qué correos han llegado a la dirección **localhost:1080**.

```
● ● ● compose.yaml
services:
  mailcatcher:
    image: schickling/mailcatcher
    ports:
      - "1025:1025" # SMTP port
      - "1080:1080" # Web interface port
    restart: unless-stopped
```

A la derecha podemos ver la captura del archivo ***.yaml**. En él disponemos del nombre de la imagen (desde dónde se descargó) y dos puertos abiertos (el 1025, que se encarga

de la salida y recepción de los mensajes y el 1080, que crea una interfaz gráfica para su revisión).

Creación del mensaje



The screenshot shows a code editor window with a dark theme. The file is named 'create_message.py'. The code defines a class 'CreateMessage' that inherits from 'MIMEMultipart'. It has an __init__ method that takes parameters for style, send_from, send_to, subject, token, name, and content. It then calls super().__init__(). The code then sets attributes like path_img, From, To, and Subject based on the input parameters. It also initializes self.style, self.name, self.token, and self.img. Finally, it calls __update_appearance().

```
class CreateMessage(MIMEMultipart):    7 usages  ↗ Joan
    def __init__( ↗ Joan
        self,
        style: MessageStyle,
        send_from: str | None = None,
        send_to: str | None = None,
        subject: str | None = None,
        token: str | None = None,
        name: str | None = None,
        content: str | None = None
    ) -> None:
        super().__init__()

        # General attributes
        self.path_img = Path(__file__).parent.joinpath("templates/assets/logotype.png")
        self["From"] = send_from if send_from else "dephokey.team@gmail.com"
        self["To"] = send_to if send_to else "dephokey.team@gmail.com"
        self["Subject"] = subject if subject else "¡Tu token para Dephokey está listo!"

        # Message attributes
        self.style = style
        self.name = name
        self.token = token
        self.img = self.__load_image()
        self.with_text = content
        self.with_html = None

        self.__update_appearance()
```

Para la gestión del correo electrónico de nuestra aplicación, decidimos realizarlo con la creación de dos clases. La primera se encarga de crear el mensaje con un estilo determinado y la segunda de enviarlo. Actualmente nuestro mensaje tiene dos estilos: el primero es el diseño

estandarizado para enviar un *token* al usuario y que así pueda cambiar su contraseña y el segundo es el mensaje que pueda mandar el usuario a nuestro servidor para cualquier consulta.

La clase `CreateMessage` se encarga de aceptar un emisor o receptor según convenga, un estilo que permite enfocarse en el diseño de su cuerpo y en la carga del logo de la empresa que veremos más adelante, además de otros atributos ya no tan necesarios.

Función '`__update_appearance()`'



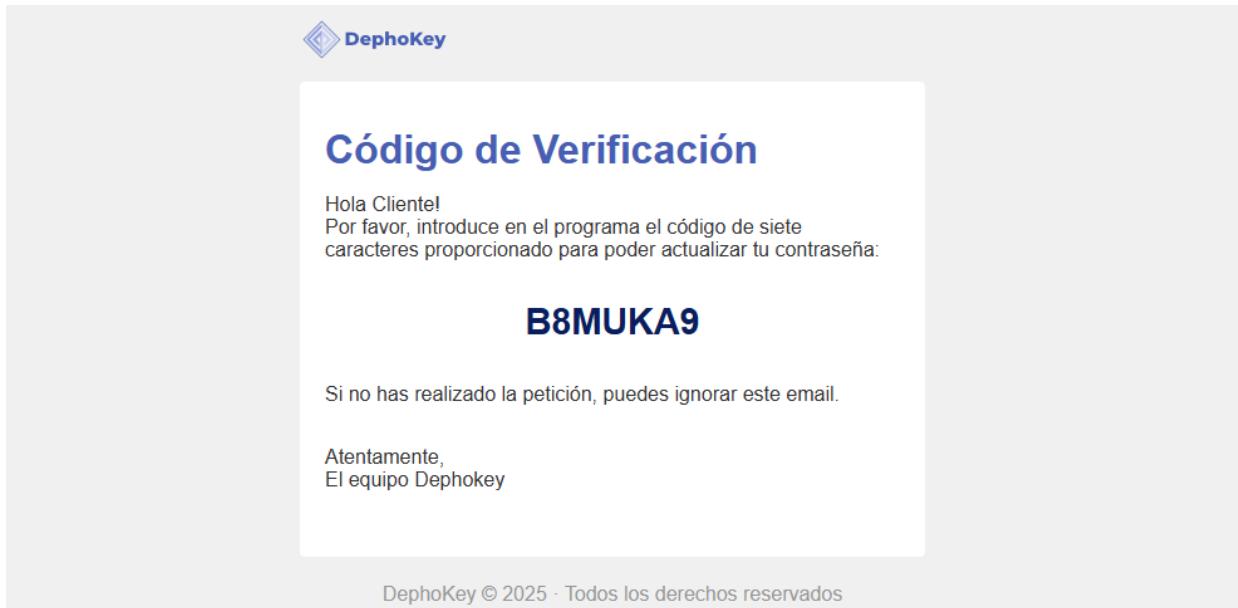
```
create_message.py

def __update_appearance(self) -> None:  # Joan
    match self.style:
        case MessageStyle.RESET:
            self.with_text = (
                f"Hello {self.name}!\nPlease, enter the seven character code "
                f"provided below to update your password:\n\n{self.token}\n\nIf you have not "
                f"made the request, you can ignore this email.\n\nBest regards,\nThe Dephockey team"
            )
            self.with_html = self.__load_template(
                template_name="reset_password.html",
                title=self.name,
                message=self.token
            )

        case MessageStyle.QUERY:
            self.with_html = self.__load_template(
                template_name="user_request.html",
                title=self["subject"],
                message=self.with_text
            )
```

Esta función privada sólo puede ser llamada desde la propia clase y es la que se encarga de actualizar los elementos del mensaje según el estilo dado. En el caso de que el estilo sea

recover, la función creará el mensaje estándar con el *token* necesario. Sin embargo, si el estilo es *feedback*, el estilo añade los datos introducidos directamente por el usuario ya que se envían las dudas o consultas realizadas por el mismo.



Ejemplo de *feedback* del usuario:



Función '__load_image()'



```
create_message.py

def __load_image(self) -> MIMEBase | None: 1 usage  ↵ Joan
    try:
        with open(self.path_img, "rb") as img:
            logo = MIMEBase(_maintype="logotype", _subtype="png")
            logo.set_payload(img.read())

        # Encode image
        encoders.encode_base64(logo)

        # Assign headers
        logo.add_header(_name="Content-ID", _value "<logotype.png>")
        logo.add_header(_name="Content-Disposition", _value="inline", filename="logotype.png")
        return logo

    except PermissionError as not_allowed:
        log.error(f"{type(not_allowed).__name__} | No se tienen permisos para leer el archivo: {not_allowed}.")
        return None
    except FileNotFoundError as not_found:
        log.error(f"{type(not_found).__name__} | No se ha encontrado el logotipo para el email: {not_found}.")
        return None
```

Esta función también es privada y su finalidad consiste en cargar la imagen del logotipo dada, codificarla en [base64*](#), asignarle un *Content-ID* y devolverla. En el caso de que haya algún fallo, la función retorna `None`. **¿Por qué la codificación en base64 y la asignación de un Content-ID?** Bien, estas decisiones se han tomado por una simple razón: durante la creación del *HTML*, no se pueden incluir imágenes en el correo (salvo que sea un adjunto) así que la imagen no puede cargar. Simplemente mostraría el error de imagen faltante. Sin embargo, y gracias a este método, podemos incrustar la imagen perfectamente.

Función '__load_template()'



```
create_message.py

@staticmethod 2 usages Joan
def __load_template(template_name: str, title: str, message: str) -> str | None:
    # Loads file directory
    env = Environment(loader=FileSystemLoader(f"{MAIN_PATH}/templates"))

    try:
        new_template = env.get_template(template_name)
        return new_template.render(mail_title=title, mail_message=message)

    except TemplateNotFound as not_template:
        log.error(f"{type(not_template).__name__} | No se ha encontrado la plantilla HTML: {not_template}")
    except Exception as unknown:
        log.error(
            f"{type(unknown).__name__} | Un error inesperado ha ocurrido al procesar la plantilla HTML: {unknown}"
        )
```

La finalidad de esta función es cargar el nombre de la [plantilla*](#) dada y renderizarla con los datos introducidos. Esto nos permite crear mensajes únicos utilizando el mismo diseño, sin tener que adaptarlo con cada creación de mensaje.

Se le incluye el decorador [@staticmethod](#) porque es un método que no necesita de la instancia de clase para funcionar y se puede llamar directamente, haciéndolo útil al realizar tareas de forma aislada a la creación del objeto.

Función 'create()'



```
create_message.py

def create(self) -> "CreateMessage": # Joan *
    if self.img:
        self.attach(self.img)

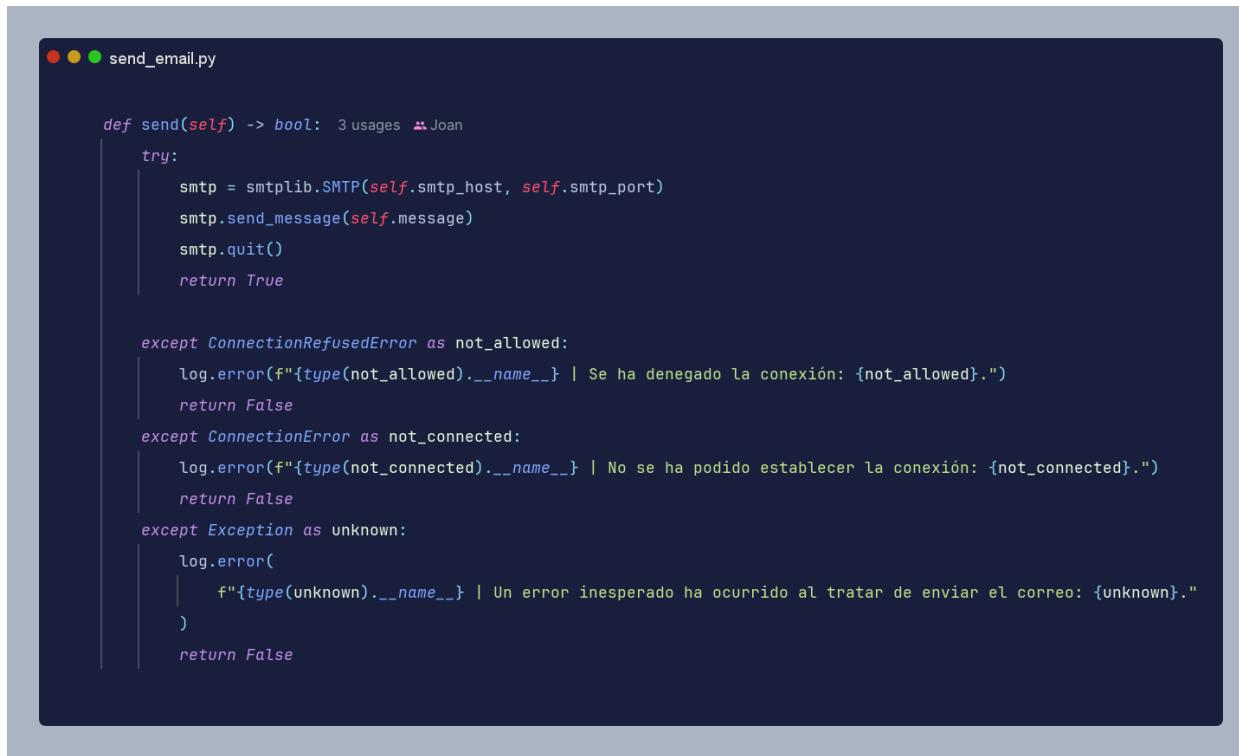
    if self.with_text:
        text_part = MIMEText(self.with_text, _subtype: "plain")
        self.attach(text_part)

    if self.with_html:
        html_part = MIMEText(self.with_html, _subtype: "html")
        self.attach(html_part)

    return self
```

Para finalizar, esta función se encarga de comprobar cada uno de los elementos necesarios para la correcta creación del mensaje. Si su existencia es correcta, entonces se encarga de adjuntar cada apartado al propio objeto `CreateMessage` y de devolverlo para poder enviarlo de manera correcta.

Envío del correo electrónico



```
send_email.py

def send(self) -> bool: 3 usages ↵ Joan
    try:
        smtp = smtplib.SMTP(self.smtp_host, self.smtp_port)
        smtp.send_message(self.message)
        smtp.quit()
        return True

    except ConnectionRefusedError as not_allowed:
        log.error(f"{type(not_allowed).__name__} | Se ha denegado la conexión: {not_allowed}.")
        return False
    except ConnectionError as not_connected:
        log.error(f"{type(not_connected).__name__} | No se ha podido establecer la conexión: {not_connected}.")
        return False
    except Exception as unknown:
        log.error(
            f"{type(unknown).__name__} | Un error inesperado ha ocurrido al tratar de enviar el correo: {unknown}."
        )
        return False
```

Disponemos de una clase llamada `SenEmail` que se encarga del envío de los correos. Sus principales atributos son la configuración del servidor (un *host* y un *port*) y un mensaje a enviar (en nuestro caso, la instancia devuelta por la clase `CreateMessage`). Una vez tenemos todos los atributos necesarios, podemos enviar el mensaje mediante su método `send()`.

Este se encarga de gestionar el envío de un correo. Para ello configuramos el *host* como `localhost` y el puerto `SMTP` que debe ser el mismo que el del archivo `*.yaml` (en nuestro caso `1025`). Después se encarga de realizar el envío. Se le ha incluido el retorno de un valor booleano para poder informar al usuario de que el correo se ha enviado correctamente.

Validaciones

La **validación*** de datos es un concepto muy importante a nivel de programación. Su misión es detener la ejecución del código antes de que pueda producirse una catástrofe pero, en realidad, **¿qué son y para qué sirven?**

Simplemente son eso, validaciones. Se aseguran de que los datos introducidos por el usuario son correctos y afortunadamente hay tres importantes razones para comprobar dichos datos:

- **Queremos obtener los datos en el formato correcto:** Muchas veces necesitamos que el formato de los datos esté adecuado a una forma específica (como los datos referentes a fechas), que no estén vacíos o que sean los que pide nuestra aplicación, para no provocar un fallo interno.
- **Queremos proteger al usuario:** Obligando al usuario a introducir contraseñas con un número de longitud definido o cumpliendo requisitos a nivel de caracteres, podemos asegurar que las credenciales almacenadas sean lo suficientemente seguras.
- **Queremos proteger nuestra aplicación:** Aunque no lo parezca, hay usuarios malintencionados que podrían dañar nuestra aplicación introduciendo datos erróneos.

Hay que tener en cuenta que las validaciones son de dos niveles. La de primer nivel se encuentra en el lado cliente (son los formularios que él puede llenar) y la de segundo nivel que se encuentran en el lado del servidor (cuyos datos se envían para ser comprobados).

En nuestro caso nos planteamos utilizar ambas. En primer nivel utilizaremos las herramientas proporcionadas por el *framework* para limitar la longitud del texto que se puede introducir y filtrar el tipo de caracteres. En la siguiente imagen podemos ver su configuración.

El `max_length` define la cantidad máxima de caracteres que puede aceptar el campo de texto y el `input_filter` sólo permite la introducción de caracteres numéricos. Por tanto el usuario sólo podrá introducir un máximo de 19 caracteres numéricos (pero cuidado, este filtro garantiza que sean números y que no sobrepasen la cantidad especificada, pero no garantiza que el usuario introduzca 18 caracteres o menos, haciendo que nuestra aplicación falle. Por eso es importante incluir las validaciones de segundo nivel).

```
creditcard_form.py
self.cc_number = CustomTextField(
    hint_text="Añade el número de tarjeta",
    can_reveal_password=True,
    max_length=19,
    password=True,
    input_filter=ft.NumbersOnlyInputFilter(),
    on_change=self._update_field_inputs
)
```

Para dicho nivel crearemos una clase llamada `Validate`. Aquí incluiremos una serie de **expresiones regulares*** que se encargarán de analizar los *inputs* del usuario y devolverán un valor *booleano*.

```
validate.py
class Validate: ✎ Joan
"""
This class helps to validate any element with different methods using regular expressions.
Other formulas can be implemented in the future.
"""


```

Correo electrónico



A screenshot of a code editor window titled "validate.py". The code defines a static method "is_valid_email" that takes a string "email" and returns a boolean value. The docstring specifies that the email must have a local-part and domain separated by an "@" sign, with parameters for email input and return type. The code uses regular expressions to parse the email into local_name, domain, and extension components, then concatenates them into a pattern and uses re.match to validate the input.

```
@staticmethod 9 usages  ↗ Joan
def is_valid_email(email: str) -> bool:
    """
    Email must have local-part and domain separated by an @ sign.
    :param email: str | email user input
    :return: boolean | True if value is valid else False
    """

    local_name = r"^\w+\.?\w+"
    domain = r"@+\w+\."
    extension = r"+\w{2,3}$"

    # Creates pattern
    pattern = local_name + domain + extension
    return bool(re.match(pattern, email))
```

El código que está asociado a `pattern` es una expresión regular. Esta se encarga de validar el nombre de usuario (que puede incluir punto o no), seguido del arroba. Añade un nombre de dominio, un punto y finaliza con su extensión de dos o tres caracteres. Si el texto dado no cumple estos requisitos dará por fallida la validación. Es importante destacar que esta validación no comprueba si la dirección de correo electrónico existe, tan solo comprueba que esté escrita cumpliendo dichos requisitos.

Contraseña



```
validate.py

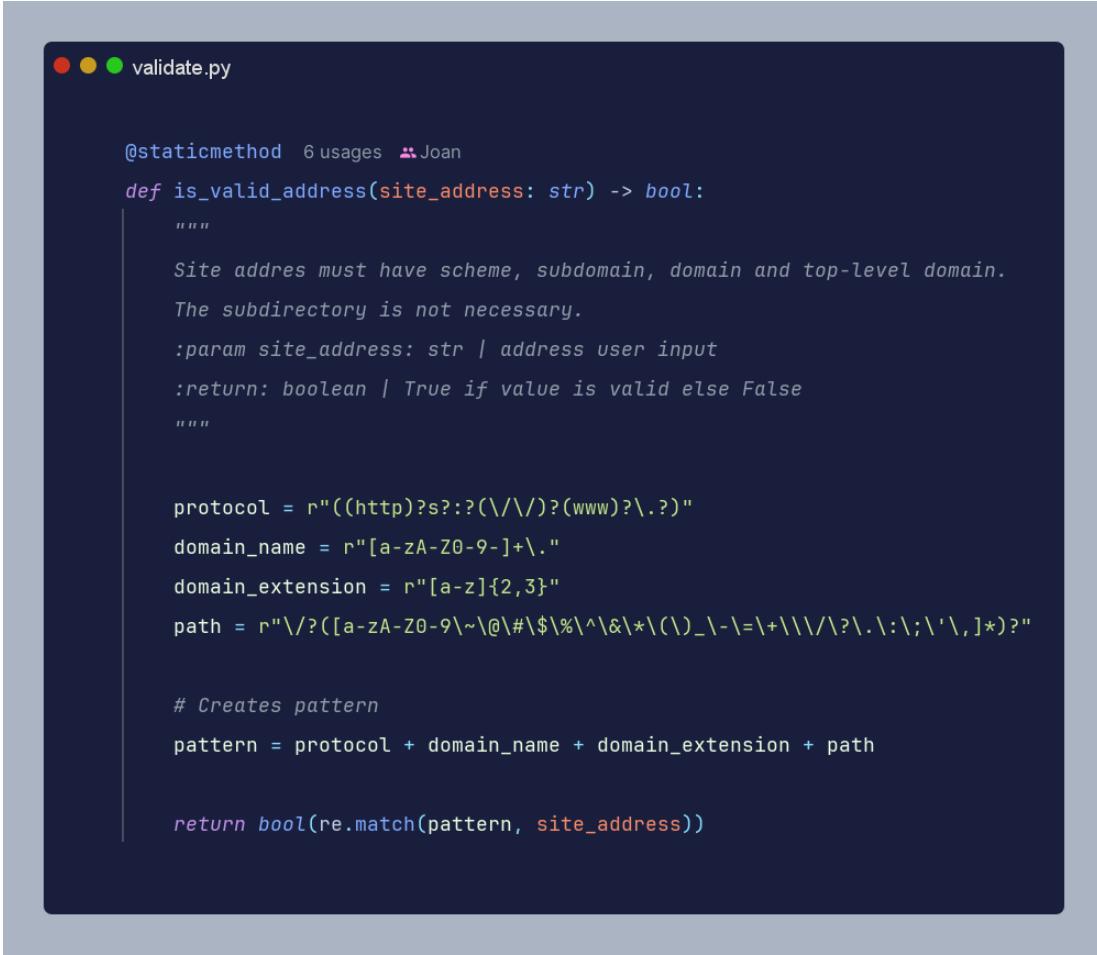
@staticmethod 8 usages ↵ Joan
def is_valid_password(password: str) -> bool:
    """
    Password must have at least one uppercase, one lowercase and one number.
    :param password: str | password user input
    :return: boolean | True if value is valid else False
    """

    has_digit = r"^(?=.*\d)"
    has_lowercase = r"^(?=.*[a-z])"
    has_uppercase = r"^(?=.*[A-Z])"
    min_length = r".{8,}$"

    # Creates pattern
    pattern = has_digit + has_lowercase + has_uppercase + min_length
    return bool(re.match(pattern, password))
```

Esta secuencia se encargará de comprobar que la contraseña dada cumpla con los siguientes requisitos: Debe disponer de, al menos, un dígito, una letra minúscula y una letra mayúscula y su longitud no puede ser inferior a ocho caracteres. Esta expresión regular proporciona un nivel básico de seguridad para contraseñas ya que no incluye caracteres especiales. Aún así, se recomienda utilizar librerías especializadas ya que tienen en cuenta muchísimos más factores de seguridad.

Dirección web



```
validate.py

@staticmethod 6 usages ↵ Joan
def is_valid_address(site_address: str) -> bool:
    """
    Site address must have scheme, subdomain, domain and top-level domain.
    The subdirectory is not necessary.
    :param site_address: str | address user input
    :return: boolean | True if value is valid else False
    """

    protocol = r"((http)?s?:?(\//)?(www)?\.\?)"
    domain_name = r"[a-zA-Z0-9-]+\."
    domain_extension = r"[a-z]{2,3}"
    path = r"\/?([a-zA-Z0-9\~@\#\$\%\^\&\*\(\)\_\-\.=\+\\\\/\?\.\:\;\\\'\,]*)"

    # Creates pattern
    pattern = protocol + domain_name + domain_extension + path

    return bool(re.match(pattern, site_address))
```

En esta ocasión la expresión regular se encarga de verificar que la *URL* introducida por el usuario es correcta. Para ello, verifica si se incluye *http://* o *https://* de forma opcional, igual que el prefijo *www*. Luego verifica que el nombre del dominio esté formado por caracteres alfabéticos (ya sea mayúsculas o minúsculas o dígitos) y guiones cortos. Seguidamente se

asegura que haya un punto y una extensión de dominio de entre dos y tres caracteres válidos. Por último comprueba de forma opcional la ruta hacia el recurso.

Sin embargo, esta verificación no garantiza que la *URL* exista además de que puede contener caracteres unicode, elemento que esta expresión regular no contempla y, por tanto, daría un falso negativo.

Fecha de caducidad



The screenshot shows a code editor window with a dark theme. At the top, there are three colored circular icons (red, yellow, green) followed by the file name "validate.py". The code itself is as follows:

```
@staticmethod 6 usages ↗ Joan
def is_valid_date(new_date: str) -> bool:
    """
    Date must have the next format: 'mm/yy'.
    :param new_date: str | date user input
    :return: boolean | True if value is valid else False
    """

    month = r"^(0[1-9]|1[0-2])"
    sep = r"/"
    year = r"([0-9][0-9])$"

    # Creates pattern
    pattern = month + sep + year
    return bool(re.match(pattern, new_date))
```

Esta es la expresión regular más sencilla. Su principal función es verificar que el formato de la fecha introducida es de *mm/aa*. Sin embargo, no garantiza que la fecha exista ya que no tiene

en cuenta ni los días de cada mes ni los años bisiestos. Además, como el año comprende valores desde el 00 al 99, no validará si éste último es importante en un contexto específico (puede que sea 2099 o 1999).

Número de tarjeta de crédito



```
validate.py

@staticmethod 8 usages  m Joan
def is_valid_creditcard_number(creditcard_number: str) -> bool:
    """
    Creditcard number must be tested by The Luhn Formula.
    :param creditcard_number: str | creditcard number input
    :return: boolean | True if value is valid else False
    """
    if len(creditcard_number) < 16 or len(creditcard_number) > 19:
        return False

    try:
        # Cast text to number
        list_numbers = [int(number) for number in creditcard_number]

    except ValueError as error_message:
        log.error(f"{type(error_message).__name__} | "
                  f"No se han podido castear los datos a número entero: Datos introducidos "
                  f"{repr(mask_number(creditcard_number))} | {error_message}")
        return False

    else:
        # Drop control digit
        control_digit = list_numbers.pop(-1)

        # Reverse list
        list_numbers.reverse()

        # Duplicate odd index number
        list_numbers = [number * 2 if index % 2 == 0 else number for index, number in enumerate(list_numbers)]

        # Subtract 9 to numbers over 9:
        list_numbers = [number - 9 if number > 9 else number for number in list_numbers]

        # Add control digit again
        list_numbers.append(control_digit)

        # Add numbers
        total = sum(list_numbers)
        return total % 10 == 0
```

Esta validación es más compleja. Como podemos ver en la imagen anterior, primero validamos que la longitud del *input* del usuario sea de entre 16 y 19 caracteres, que según el ISO / IEC 7812, es la longitud de un número de [tarjeta de crédito*](#). Después, y dentro de un bloque *try – except*, se castean cada uno de los caracteres a *integer* porque necesitamos realizar una serie de cálculos con ellos más adelante.

Para finalizar, una vez tengamos todos y cada uno de los números casteados, procederemos a aplicar el [algoritmo de Luhn*](#), que se encarga de verificar que el resto del cálculo entre 10 sea igual a cero. Si es así, el número de tarjeta será válido, de lo contrario no.

Inyección de Datos

En esta sección vamos a ver la creación de una función que se encargue de inyectar elementos a nuestra base de datos para comprobar su correcto funcionamiento.

```
● ● ● filler_settings.py

def fill_with_users() -> None: 2 usages ↵ Joan
    # Add admin user automatically
    if not session.query(User).filter(User.email == "admin.24@gmail.com").first():
        log.info("Usuario ADMIN no encontrado. Se procede a crearlo...")
        __create_admin_account()

    # Add client user automatically
    if not session.query(User).filter(User.email == "client.24@gmail.com").first():
        log.info("Usuario CLIENT no encontrado. Se procede a crearlo...")
        __create_client_account()
```

Este fichero se encarga de comprobar si en la base de datos actual se encuentran estos dos usuarios. Si no los encuentra, procede a llamar a las funciones pertinentes para crearlos y mantiene un registro de qué está realizando.



The screenshot shows a code editor window with a dark theme. At the top, there are three colored circular icons (red, yellow, green) followed by the file name "filler_settings.py". The code itself is as follows:

```
● ● ● filler_settings.py

def __create_client_account() -> None: 1 usage  ↵ Joan
    client = User(
        fullname="Manuel Tester García",
        email="client.24@gmail.com",
        password="Client1234",
        user_role=UserRole.CLIENT
    )
    session.add(client)
    session.commit()

    # Add client data examples
    _fill_with_data(client)
    log.info("¡Datos de prueba creados exitosamente!")
```

Como ejemplo utilizaremos la función `__create_client_account()`. Esta función crea una instancia de la clase `User` con los atributos ya especificados. Después la añade a la base de datos, guarda los cambios y por último llama a la última función incluida en el fichero.



```
● ● ● filler_settings.py

def __fill_with_data(user: User) -> None: 1 usage ⌘ Joan
    log.info(f"Añadiendo elementos de prueba al usuario {user.email}...")

    some_sites = []
    for _ in range(10):
        some_sites.append(
            Site(
                address=FAKE.url(),
                username=user.email,
                password=FAKE.password(length=18, special_chars=False),
                user=user,
                name=FAKE.domain_name().capitalize()
            )
        )
    session.add_all(some_sites)
```

La función `__fill_with_data()` se encarga de inicializar la clase `Faker`, cuya función es devolver diferentes datos. Gracias a esta herramienta, podemos añadir automáticamente varias instancias a un usuario dado y guardarlas en la base de datos para comprobar que todo funciona como es debido.

id	user_id	name	address	username	encrypted_password	created
s6jlld04xngfwfr	gfhqblyfgbmmvv	Rojas.com	https://www.despacho.net/	client.24@gmail.com	gAAAAABn4Equ2DbEQ5sFbFC8kqK0rpJ...	2025-03-23 18:53:50.555813
ccuzf7ccoa0az7w	gfhqblyfgbmmvv	Nelida.es	http://consultoria.org/	client.24@gmail.com	gAAAAABn4EquYaOSRcYhkOm-ysspoU...	2025-03-23 18:53:50.556811
hjuewm827fp45u6	gfhqblyfgbmmvv	Mineria.org	https://hierro.es/	client.24@gmail.com	gAAAAABn4Equ4_daRzh2hSC6vxXT7rZ0...	2025-03-23 18:53:50.557812
d46sn1ia0i6l26s	gfhqblyfgbmmvv	Barrio.com	https://hotel.com/	client.24@gmail.com	gAAAAABn4EquQ-ZIUsxZngOfFm2LR4U...	2025-03-23 18:53:50.557812
txlx3ahymx72v2k	gfhqblyfgbmmvv	Elba.com	http://www.manuelita.net/	client.24@gmail.com	gAAAAABn4Equ9lqSgiwroP0W9ElcGD16...	2025-03-23 18:53:50.558813
hhzf62xhncagzzi	gfhqblyfgbmmvv	Restauracion.com	http://jover.com/	client.24@gmail.com	gAAAAABn4EquWm2Sp6DH4eCIUneTYE...	2025-03-23 18:53:50.558813
h1n325l7wn7yj1u	gfhqblyfgbmmvv	Promociones.es	https://paloma.com/	client.24@gmail.com	gAAAAABn4EqupZXip8_cEnlxse1BTvOLE...	2025-03-23 18:53:50.559812
f4f0vitqlq44bau	gfhqblyfgbmmvv	Mineria.com	https://www.estavez.com/	client.24@gmail.com	gAAAAABn4EquHN5HdtJf_z4klBkn5n_8...	2025-03-23 18:53:50.559812
sjj36vwa0o42lcv	gfhqblyfgbmmvv	Instalaciones.es	http://alamo.com/	client.24@gmail.com	gAAAAABn4EquLoqSoGKEmpW3WWRYX...	2025-03-23 18:53:50.560813
gsqohtg9uxyps9t	gfhqblyfgbmmvv	Supermercados.org	http://www.inversiones.es/	client.24@gmail.com	gAAAAABn4Equdv1A12oQv8ucd12mYgw...	2025-03-23 18:53:50.560813

Al abrir la base de datos después de realizar un *commit*, vemos que los datos se han añadido perfectamente, que los datos encriptados sí lo están y que el `user_id` es el mismo para todos los elementos. ¡Todo un éxito!

Este es el primer de los ejemplos añadidos a la función, ya que ésta se encarga de llenar todas y cada una de las listas que tiene el usuario. Como ya no disponemos de una base de datos vacía, podemos empezar a trabajar en la interfaz gráfica y comprobar que las funcionalidades de añadir, leer, editar y eliminar funcionan correctamente.

Testeo de archivos

Este, de seguro, es el apartado más importante de un [proyecto*](#). Todo programador debe de realizar una serie de comprobaciones o tests antes de pasar a la siguiente fase de un proyecto. Estos tests se encargan de que, cualquier cambio realizado en los archivos, no pueda echar a perder horas de trabajo. En nuestro caso y como apenas estamos empezando, lo principal es aprender su funcionamiento y cómo nos puede beneficiar.

Para ello, utilizamos la biblioteca [unittest*](#), que viene preinstalada junto con [Python](#). Esta librería nos ayuda con la creación de tests unitarios que se encargan de comprobar que los datos son correctos. No vamos a exemplificar todos los tests realizados, pero sí nos basamos en el ejemplo de test unitario para [User](#).



The screenshot shows a code editor window with a dark theme. The file is named `testModel_user.py`. The code defines a class `UserBuilder` with three methods: `__init__`, `with_role`, and `build`. The `__init__` method initializes a user with a specific name, email, and password. The `with_role` method allows changing the user's role. The `build` method returns the built user object.

```
testModel_user.py

class UserBuilder: 2 usages ↵ Joan
    def __init__(self) -> None: ↵ Joan
        """Helps to create a User instance."""
        self.__user = User(
            fullname="UserTest Name",
            email="user.email@example.com",
            password="User_1234"
        )

    def with_role(self, new_role: UserRole) -> "UserBuilder":
        self.__user.role = new_role
        return self

    def build(self) -> User: ↵ Joan
        return self.__user
```

Primero nos encargamos de crear una clase constructora de usuarios. Su principal función es preparar instancias de usuario según necesitemos. Hay que recalcar que se ha empleado el encadenamiento de métodos que permite concatenar acciones. En este ejemplo, si el tester necesita cambiar el rol del usuario, simplemente lo hará a la vez que crea el objeto.



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there are three small circular icons (red, yellow, green) followed by the file name "testModel_user.py". The code itself is written in Python and defines a class named "TestUser" which inherits from "unittest.TestCase". It contains two methods: "setUp" and "tearDown". The "setUp" method logs an info message, creates a new user instance using "UserBuilder", sets its hashed password to "User_1234", and logs another info message indicating the user instances are ready for testing. The "tearDown" method deletes the user object from memory.

```
class TestUser(unittest.TestCase):    # Joan
    def setUp(self) -> None:        # Joan
        log.info("Preparing USER instance...")
        # Create new instance
        self.user = UserBuilder().with_role(UserRole.ADMIN).build()
        self.hashed_password = sha256(b"User_1234").hexdigest()

        log.info("USER instances ready for test...")

    def tearDown(self) -> None:        # Joan
        del self.user
```

Ahora bien, **¿cómo funciona?** Cuando se crea un test, se divide en tres secciones. La primera o `setUp()` se encarga de la creación de los elementos necesarios para cada test (o los más genéricos). La segunda es `tearDown()`, que se ejecutará cada vez que un test termine. En la configuración que podemos ver arriba, lo único que realizará será la eliminación del objeto de la memoria. Y la tercera sección son los tests que se quieran realizar.

Idealmente, cada test debería de verificar sólo una cosa a la vez para saber a ciencia cierta en qué ha fallado. En el ejemplo de arriba podemos ver que el primer test que realiza es comprobar si existe algo (no necesariamente un objeto de tipo usuario porque eso podría formar parte de

otro test). Si en este caso el objeto es tipo `None` entonces el test lanzará un `AssertionError`, haciendo que el test no haya pasado. Esta acción dará pie al usuario de que algo ha ido mal y le dará tiempo de revisar y arreglar el código en su defecto.

Abajo se pueden ver más ejemplos donde se comprueba que la instancia de usuario tiene un elemento contraseña, que es de tipo `string` y que debe estar *hasheada*.

```
● ● ● testModel_user.py

def test_userPasswordExists(self) -> None:  ↵Joan
    self.assertIsNotNone(
        obj=self.user.hashed_password,
        msg="User MUST HAVE password."
    )
    log.info(">>> Confirm if USER has PASSWORD...    OK")

def test_userPasswordType(self) -> None:  ↵Joan
    self.assertIsInstance(
        obj=self.user.hashed_password,
        cls=str,
        msg="Password MUST BE string type."
    )
    log.info(">>> Confirm if USER PASSWORD is instance of STRING...    OK")

def test_userPasswordHashed(self):  ↵Joan
    self.assertEqual(
        first=self.user.hashed_password,
        second=self.hashed_password,
        msg="Password MUST BE hashed."
    )
    log.info(">>> Confirm if PASSWORD is HASHED...    OK")
```

Interfaz gráfica

La **interfaz gráfica*** es el intermediario que hay entre el usuario y la máquina. Está formada por elementos visuales que representan conceptos y que ayudan al usuario a comunicarse con ella traduciendo sus acciones a un lenguaje que la máquina pueda entender. Dentro de sus variantes, la que vamos a manejar en este proyecto es la interfaz gráfica de escritorio o interfaz orientada al usuario, cuyos principales elementos son botones, ventanas emergentes y barras de búsqueda.

Esta interfaz se encarga de manejar las acciones del usuario de forma cíclica: primero mostrando información y dejando que el usuario modifique elementos; luego traduciendo sus acciones a lenguaje máquina, obteniendo los resultados de dichas acciones; y finalmente volviéndolos a mostrar al usuario ya actualizados.

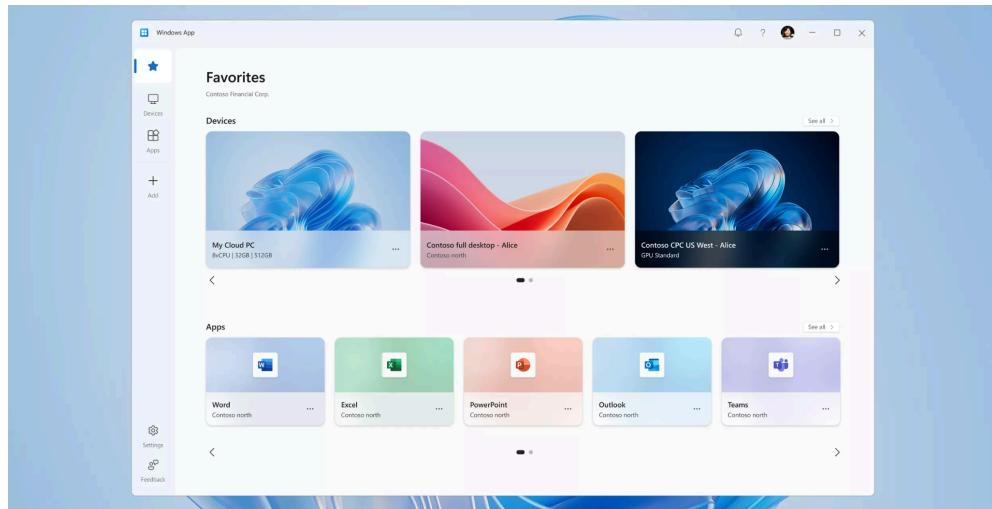
Para poder poner en perspectiva nuestra **aplicación¹**, la vamos a diseccionar en diferentes capas que irán desde las más genéricas a las más específicas, explicando sus diferentes funcionalidades de forma general.



¹ Ver el documento “GUI-diagram” anexado al final del proyecto para revisar diseños y comportamientos.

Páginas principales

Las páginas principales se encargan de mostrar las diferentes secciones de nuestra aplicación. Son los *endpoints* que, al navegar por nuestra aplicación, muestran diferentes datos. Un ejemplo claro podría ser la página *home*, que puede mostrar un resumen de las últimas modificaciones realizadas por el usuario en la última sesión.



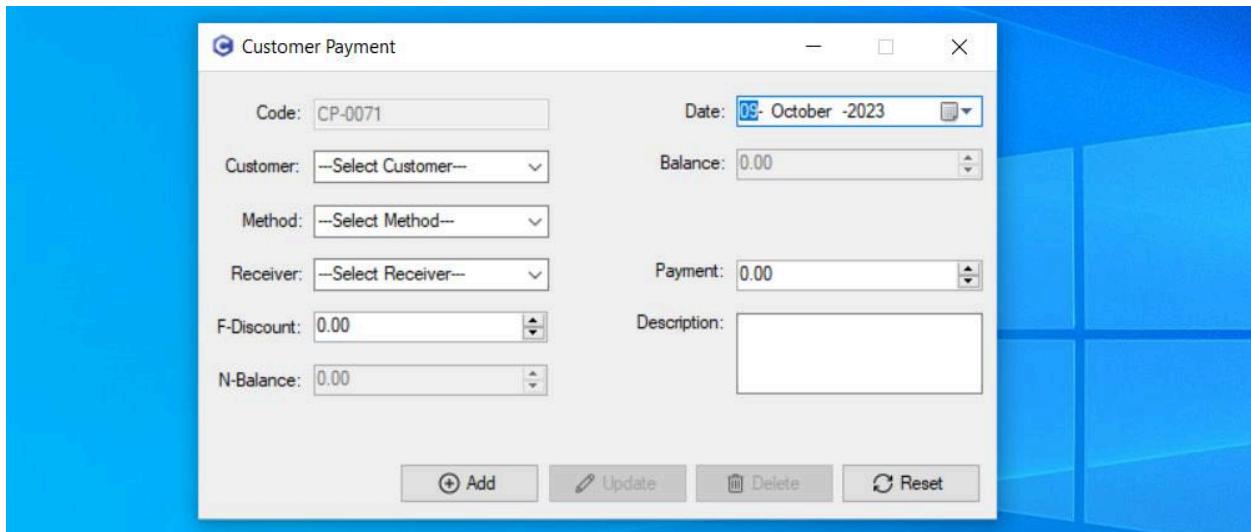
Para manejar la carga de datos dentro de las diferentes páginas de navegación creamos un archivo que maneje la construcción de dichas páginas y, si ya están creadas, que las actualice. Así podemos ahorrar recursos y mejorar la fluidez de nuestra aplicación.

```
content_manager.py
def __update_appearance(self, user_input: str | None = None) -> None:
    match self.style:
        case ContentStyle.ABOUT:
            if not isinstance(self.about_pg, AboutPage):
                self.about_pg = AboutPage(self.page, self.snackbar, self.confirm_changes)
                self.canvas.controls = [self.about_pg]
                log.info("Página 'ABOUT' creada.")

            self.about_pg.update_content()
            self.canvas.controls = [self.about_pg]
            log.info("Página 'ABOUT' actualizada.")
```

Formularios

Los **formularios*** son un grupo de campos de texto, botones o pestañas que permiten la recogida de datos de manera rápida y ordenada. Esos datos introducidos por el usuario son con los que trabajará nuestra aplicación.



Ésta dispone de tres formularios prácticamente iguales, sólo cambia su contenido así que, mediante la programación orientada a objetos, podemos crear un formulario base y que éste cambie su contenido cuando sea necesario.

Los elementos que se repiten a lo largo del formulario son el botón de cerrar ventana, el botón de envío de las respuestas, el botón que cancela los datos introducidos y algún elemento estético que se puede emplear en los formularios, como son su color, su tamaño o su forma.

En la siguiente imagen podemos ver la configuración del formulario base:

```
● ● ● base_form.py

class BaseForm(ft.AlertDialog): 16 usages ↗ Joan
    """Creates a basic form with some attributes by default like form size, main buttons and their behavior."""
    def __init__(self) -> None: ↗ Joan
        super().__init__()

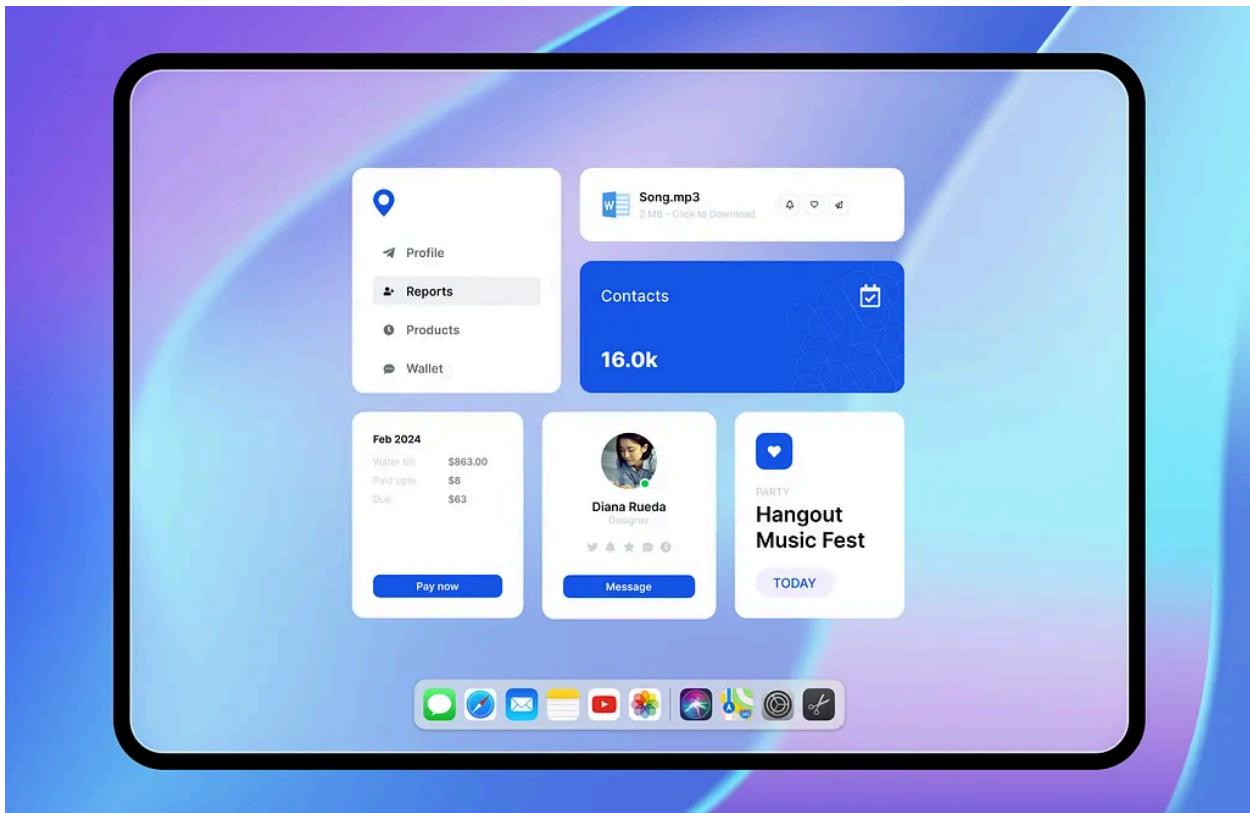
        # General attributes
        self.fields = []
        self.submit_button = CustomElevatedButton(
            name="Aceptar",
            style=ButtonStyle.DEFAULT,
            disabled=True
        )
        self.cancel_button = CustomElevatedButton(
            name="Cancelar",
            style=ButtonStyle.CANCEL
        )
        self.span = ft.TextSpan(
            text="*",
            style=ft.TextStyle(font_family="AlbertSansB", color=dangerTextColor)
        )
        self.close_button = ft.IconButton(
            icon=ft(Icons.CLOSE_ROUNDED,
            icon_color=neutral80,
            on_click=lambda _: self.page.close(self),
            highlight_color=neutral40,
            hover_color=neutral20
        )

        # Form general settings
        self.modal = True
        self.actions = [self.cancel_button, self.submit_button]

        # Form general design
        self.shape = ft.RoundedRectangleBorder(4)
        self.bgcolor = neutral00
        self.content = ft.Container(width=550, height=378)
```

Widgets

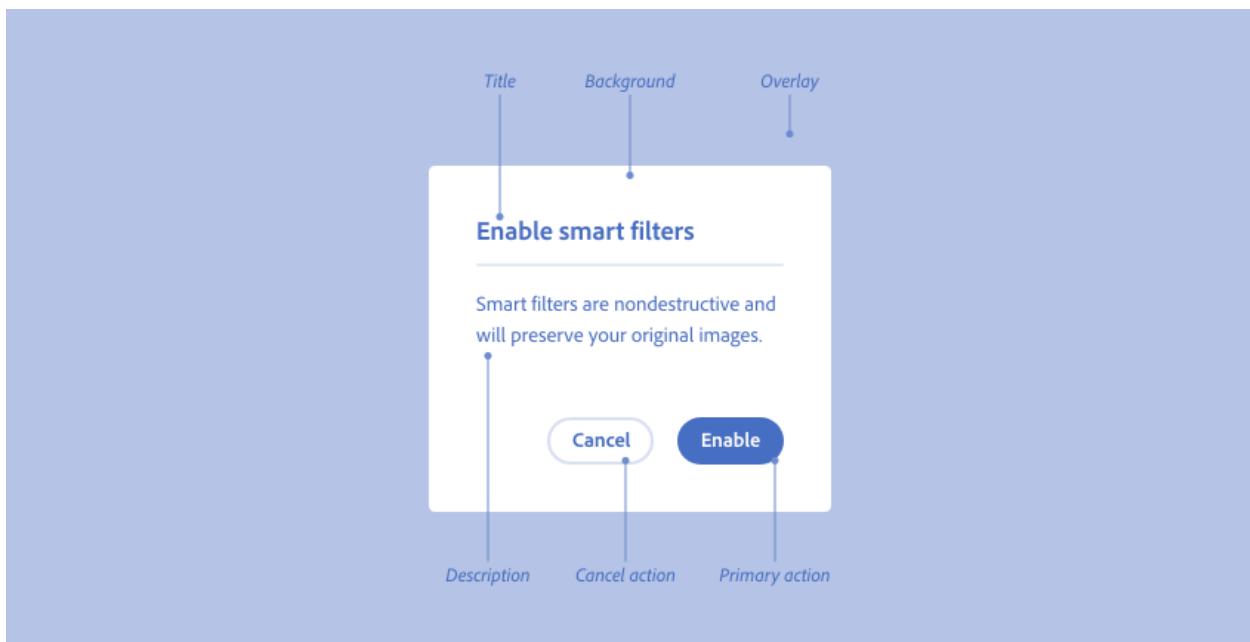
Los *widgets** son elementos de interfaz gráfica que se caracterizan por mostrar determinada información y mejoran la funcionalidad de la aplicación al permitir la interacción del usuario de manera eficiente y fluida. Nuestros *widgets* principales se basan en la representación de una tarjeta informativa para enumerar los atributos de una página web, una tarjeta de crédito y una nota adhesiva.



Cuadros de Diálogo

Los [cuadros de diálogo*](#) o ventanas emergentes se utilizan para la comunicación sencilla entre el usuario y el programa. Están formados por una pregunta cuya respuesta puede ser afirmativa o negativa y según la respuesta realizan una acción. Hay de dos tipos: modales y no modales y la diferencia es que en los cuadros de diálogo modales el usuario está obligado a responder antes de continuar.

En nuestra aplicación hemos elegido los cuadros de diálogo modales ya que son necesarios para la confirmación del usuario cuando éste decida eliminar un ítem de la base de datos o, incluso, su propia cuenta.



Stack tecnológico principal

Python*: Como lenguaje principal de programación. Todos los archivos básicos del programa emplearán dicho lenguaje en su estructuración.

Git*: Como herramienta de control de versiones.

HTML*: Como lenguaje secundario utilizado en el diseño del mensaje enviado para la funcionalidad de recuperación de contraseña.

CSS*: Para dotar de estilo al mensaje *html* enviado.

SQLite*: Como lenguaje para consultas estructuradas.

Flet*: Framework que permitirá el diseño de la interfaz de usuario.

Docker*: Permite crear, montar y administrar contenedores virtuales aislados para el uso compartido de las aplicaciones sin utilizar dependencias o la creación de servidores virtuales mediante imágenes de forma estandarizada.

Cryptography*: Herramienta que permitirá una encriptación simétrica para almacenar los datos de forma segura.

Stack tecnológico secundario

Pathlib*: Librería que permite la manipulación de los directorios del sistema como si fueran objetos. Usada para gestionar directorios.

Jinja2*: Librería que permite la creación de documentos personalizados *HTML* desde una plantilla dada.

OS*: Librería que permite realizar modificaciones en el sistema. Usada para gestionar archivos.

SQLAlchemy*: Como herramienta de mapeo que permitirá manipular los datos que queremos introducir en la base de datos de forma parecida a los objetos en la *POO*.

Hashlib*: Permite la encriptación unidireccional de las contraseñas de usuario.

Logging*: Manejador de registros de la aplicación. Se encargará de almacenar información de diferentes eventos en un archivo principal que se enviará al creador del sistema para analizar los datos.

Unittest*: Se utilizará para la creación de clases que realicen el testeo de los modelos planteados.

Smtplib*: Se utilizará configurar los protocolos de envío y recepción de correo electrónico.

Email*: Se emplea para el manejo y creación de correos electrónicos que se puedan enviar mediante el protocolo *SMTP* (Protocolo Simple de Transferencia de Correo).

Datetime*: Creará registros temporales de los objetos almacenados en la base de datos, útil para su gestión temporal.

Nanoid*: Generará números de identificación para cada uno de los elementos con un tamaño determinado. Aunque la cantidad de caracteres sea limitada, con la configuración actual se prevé una colisión en aproximadamente 800 años (generando mil códigos por segundo).

Asyncio*: Esta librería permitirá el seguimiento de la sesión iniciada en segundo plano. Cuando detecte que la sesión ha expirado, volverá el programa a su estado inicial.

Time*: Esta librería permitirá el pausado de la aplicación entre funcionalidades.

Enum*: Librería que permite la creación de constantes enumeradas. Se utilizará para la creación de roles de usuario y estilos de formulario.

Faker*: Como generador de elementos falsos pero válidos (muy útil para llenar datos de forma rápida y eficaz).

VirtualEnv*: Como herramienta para crear el entorno virtual principal.

PyCharm*: Como IDE principal donde programar el software.

Lunacy*: Framework parecido a *Figma* que permite el prototipado del diseño de interfaces, permitiendo la creación de botones, páginas, enlaces y comportamientos.

DB Browser*: Aplicación que permite la lectura, modificación y eliminación de elementos en una base de datos.

Alternativas evaluadas

1. SQL vs. no SQL

Existen varios [tipos de bases de datos*](#) en la actualidad y cada uno está diseñado para diferentes propósitos y necesidades. Así que vamos a dividirlos en dos tipos genéricos: relacionales y no relacionales.

El lenguaje que vamos a implementar para la base de datos en esta ocasión se trata del lenguaje **SQL** (*Structured Query Language* por sus siglas en inglés). Dicho lenguaje nos permite la creación de tablas relacionales que nos facilita una ordenación de los datos de forma más legible y sencilla.

Pero, **¿por qué no utilizar bases de datos no relacionales?** Bien, para responder a esa pregunta, deberemos analizar las ventajas y desventajas de cada una:

Las ventajas de las bases de datos **SQL**:

- **Tienen un lenguaje estandarizado:** Permite una facilidad de aprendizaje bastante elevada y su portabilidad.
- **Integridad de los datos:** Suelen implementar mecanismos para garantizar la integridad de sus datos como claves foráneas y primarias.
- **Soporte:** Al ser un tipo de base de datos con una larga trayectoria en el mercado, la documentación y el soporte es fácil de encontrar.
- **A menor cantidad de datos, mayor robustez:** Cuando las bases de datos son pequeñas, la rapidez y robustez del lenguaje proporciona ventajas ante otros tipos de bases de datos.

Sin embargo, por contra podemos encontrar las siguientes características:

- **Tiene rigidez en el esquema:** Deben tener un esquema predefinido y cambiar algún elemento en el futuro puede acarrear problemas.
- **Escalabilidad limitada:** A mayor cantidad de datos, mayores deberán ser las características del hardware para alimentar el sistema.

Para las bases de datos no relacionales podemos decir que:

- **Flexibilidad en el esquema:** No requieren un esquema predefinido lo que permite adición de elementos faltantes.
- **Variedad de modelos de datos:** Se pueden trabajar diferentes modelos de datos como documentos, pares de clave-valor, grafos, columnas...
- **Escalabilidad sencilla:** Su escalabilidad se puede configurar de forma horizontal, permitiendo el almacenaje en diferentes servidores sin tener que mejorar las características del hardware.
- **Permiten un gran manejo de datos:** Cuando las bases de datos almacenan muchos elementos, pueden afrontar dicha carga.

Pero sus desventajas son las siguientes:

- **No tiene un lenguaje estandarizado:** Característica que dificulta el aprendizaje del lenguaje y su portabilidad.
- **Menor consistencia:** Muchas bases de datos sacrifican la consistencia de los datos a favor del rendimiento.
- **Novedad:** Al ser relativamente más nuevas a las bases de datos **SQL**, pueden tener menos herramientas, soporte o documentación disponible.

En conclusión: Una base de datos **SQL** es idónea para nuestro proyecto pues la nuestra será relativamente pequeña y no necesita tanta escalabilidad al almacenarse en el ordenador del propio usuario.

2. Flet vs. Tkinter

Antes de nada, ¿qué es un *framework*?

En este caso, un *framework** de interfaz gráfica es un entorno de trabajo que nos permitirá la unión y creación de diferentes elementos visuales del proyecto de manera que el usuario pueda interactuar con ellos. Actualmente hay diferentes tipos de *frameworks* enfocados a distintos tipos de programación, abriendo un abanico bastante imponente en cuanto a selección se refiere. Sin embargo, al final elegimos entre dos opciones: *TKinter*, un *framework* que viene incluido en **Python** y no necesita de instalación o *Flet*, un *framework* externo que sí requiere de instalación.

Bien, las ventajas de *Tkinter* que podemos observar son las siguientes:

- **Viene incluida en el propio lenguaje Python:** No es necesaria una instalación previa del *framework*.
- **Es multiplataforma:** En cuanto a sistema operativo se refiere, puede funcionar tanto en *Windows* como en *Linux* o *Macintosh*.
- **Robusto:** Es un *framework* capaz de crear aplicaciones de escritorio robustas.

Sin embargo, sus limitaciones son:

- **Dispone de un diseño anticuado:** Aunque se pueda actualizar el tema o el color, se basa en el diseño antiguo de los primeros sistemas *windows*.
- **La documentación:** Su documentación no está organizada ni actualizada, haciendo que la búsqueda de información sea exasperante.
- **Carece de funcionalidades avanzadas:** No se pueden incluir animaciones ni transiciones entre páginas, haciendo una experiencia de usuario aburrida y monótona.

Por otro lado, tenemos el *framework Flet*, cuyas ventajas son:

- **Desarrollo rápido:** Simplifica la creación de páginas y diseños por permitir una creación de dichos elementos parecidos a nivel estructural a *HTML*.
- **Es 100% multiplataforma:** Además de poder emplear dicha biblioteca en los tres entornos de escritorio más comunes, también permite la integración en desarrollo web y móvil.
- **Su documentación:** La documentación está actualizada y bien organizada, haciendo que la resolución de dudas sea fácil y sencilla. Además cuenta con una comunidad muy activa.

Sin embargo, también dispone de sus limitaciones:

- **Es relativamente joven:** *Flet* surgió hace poco, haciendo que pueda tener herramientas todavía no disponibles.
- **Su sintaxis:** Aunque sea más sencillo de manejar, *Flet* necesita que aprendas su estructuración para saber cómo proceder.
- **Su dependencia de Flutter:** Al estar basado en dicho lenguaje, hereda sus limitaciones.

Ahora bien, sabiendo todas las características anteriores y teniendo en cuenta el acabado final que vamos a darle al programa, podemos resumir en que, en cuestión de diseño y facilidad resolutiva, el claro ganador en este caso es *Flet*. Aunque *TKinter* pueda presumir de robustez y madurez en el mercado, *Flet* termina cumpliendo nuestras expectativas a nivel visual.

En conclusión: La facilidad de uso que nos ha brindado *Flet* en comparación con *TKinter* y su curva de aprendizaje ha superado nuestras expectativas y la apariencia atractiva del *framework*, que es relativamente parecida a la estructuración *HTML*, hace más sencilla la transición hacia su uso.

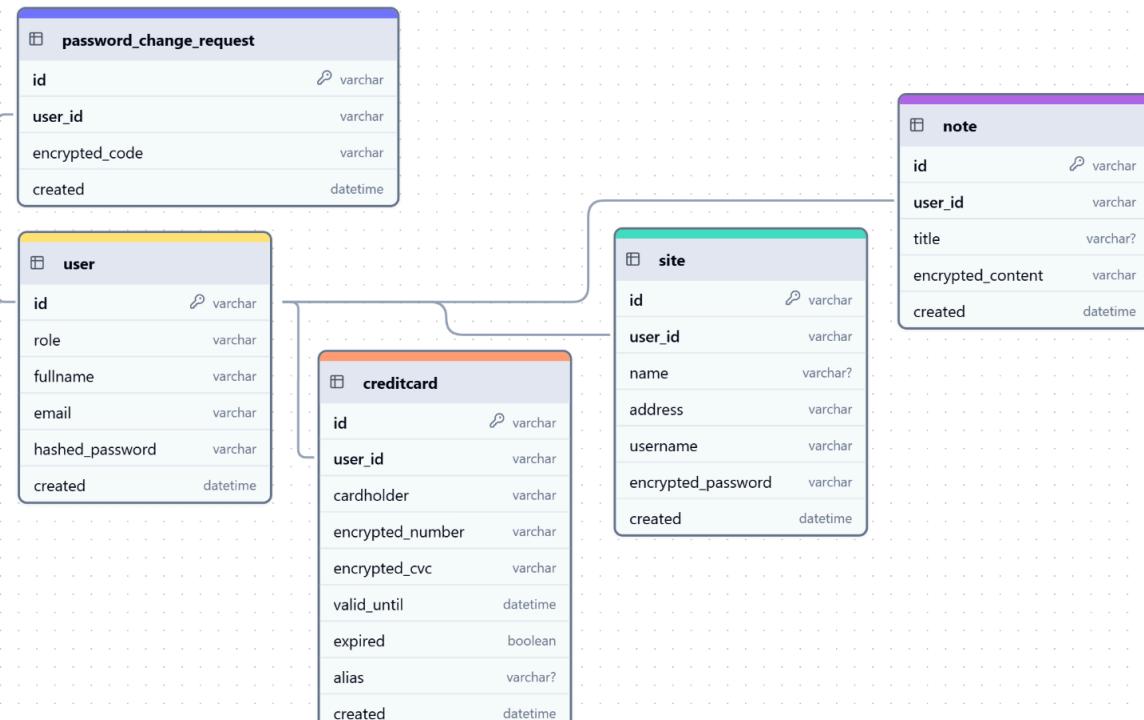
Modelo de datos

Toda sección comienza con una pregunta: **¿qué es el modelado de datos?** El **modelado de datos*** es, en esencia, la representación gráfica del diseño de conexiones y limitaciones que deberá implementarse en una base de datos. También incluye cómo se deben manipular los datos y qué funciones desempeña cada parte, sin embargo, **¿cómo lo podemos implementar en nuestra aplicación?** Para ello, debemos desglosar cada apartado en diferentes secciones:

- **Entidades:** Las entidades representan los elementos principales o, en todo caso, las tablas. Sabemos que nuestra aplicación deberá almacenar diferentes datos y sabemos que principalmente son usuarios, tarjetas de crédito, sitios web, notas seguras y peticiones de cambio de contraseña.
- **Atributos:** Los atributos varían según el tipo de elemento del que forman parte. Los usuarios, por ejemplo, disponen de un nombre y un correo electrónico mientras que una tarjeta de crédito dispone de un número y de un código cvc. En algunos casos puede coincidir el nombre de atributo, pero no el elemento al que hace referencia.
- **Relaciones:** Las relaciones ayudan a comprender la conexión que hay entre las entidades. En nuestro caso, el usuario está relacionado con cada entidad y, sin embargo, las otras entidades no se relacionan entre sí. Es decir, el usuario tiene acceso a las demás entidades, pero las notas no pueden acceder a los sitios web.
- **Identificadores:** Los identificadores son, en esencia, una serie de números y letras aleatorios que se asocian a una entidad y le otorga una clave única para diferencia del resto. En nuestro caso utilizaremos la clave primaria para dotar de una identidad única a cada elemento y de una clave foránea para asociarlos entre sí.
- **Restricciones:** Las restricciones en la base de datos son fundamentales ya que debemos mantener a raya la inserción de datos. Una cantidad masiva y grande de elementos podría ralentizarla de forma que termine colapsando.

- **Tipos de datos:** Los tipos de datos almacenados en nuestra base de datos se caracterizan por ser todos de tipo texto o *string* y otras son datos temporales o de tipo *datetime*. Aunque existe el tipo *text*, no es necesario pues esa opción sólo se usa cuando el texto que se almacena es muy grande.
- **Esquema:** El esquema se encarga de describir de forma lógica toda la base de datos. Se encarga de definir sus restricciones y relaciones.

Ahora que tenemos los puntos clave explicados y hemos podido ahondar lo suficiente, podemos prestar atención al siguiente diagrama:

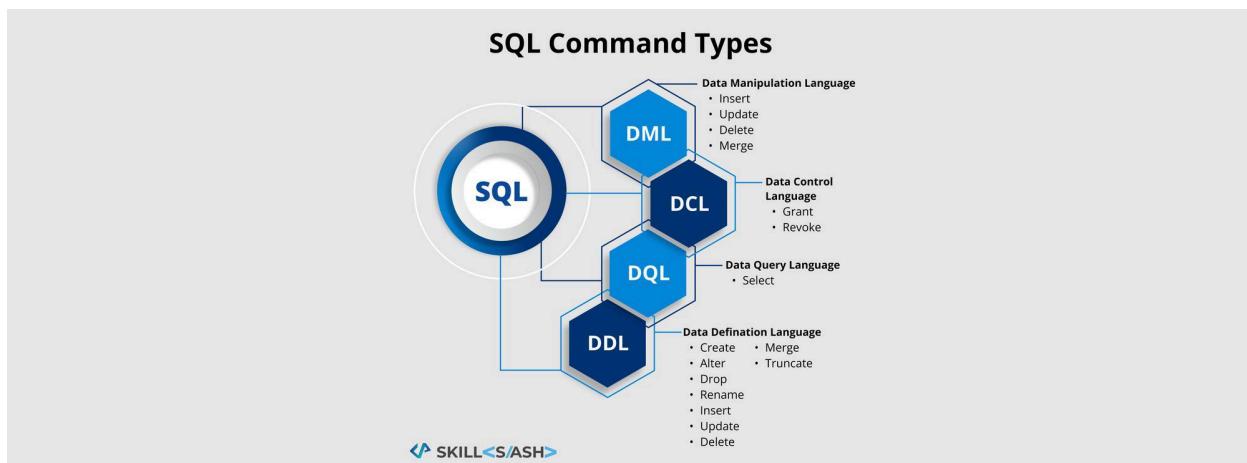


Esta imagen representa las entidades que necesitamos almacenar en la **base de datos**² y nos ayuda a visualizar más el concepto. Como podemos comprobar, el usuario utiliza su ID como clave foránea en todas las demás tablas, ayudando a representar su relación. Sin el usuario, no pueden existir las demás entidades.

Eso es una condición importante a tener en cuenta pues nuestra base de datos es la que se encargará de eliminar todas las entidades asociadas a él como son su cuenta, sus tarjetas, sitios web y demás. Esta configuración se puede observar al principio del documento, en la sección **Manejo de una Base de datos** donde podemos observar cómo se ha implementado en la relación entre clases.

Los atributos de cada entidad están correctamente asociados a los elementos que necesitamos almacenar, los cuales mayormente se clasifican como *varchar*, cuya traducción equivale a cadena variable de caracteres (mayormente son cadenas de texto, no almacenamos números) y unos pocos son de tipo *datetime* y *boolean*. Algunos elementos finalizan con un interrogante ya que para **SQL**, son elementos que pueden almacenar *null* (sin datos).

Respecto a los identificadores de identidad podemos ver que la única entidad que dispone de uno es el usuario. Todas las demás disponen de dos: el suyo propio que hace referencia a sí misma, y el `user_id`, que hace referencia al usuario al que pertenece.



² Ver el documento "Database Settings" anexado al final del proyecto para más información.

¿Cómo funciona en realidad la petición de eliminación del usuario a nivel interno? Bien, pues es algo muy curioso. Aunque se configure como una sola acción, realmente lo que estamos haciendo es que el *ORM* se encargue de realizar cinco peticiones de eliminación: Para la eliminación del usuario realizará una petición SQL como la siguiente:

```
DELETE FROM user WHERE id = 'ID_DEL_USUARIO';
```

Y para la eliminación de los otros elementos realizará las siguientes peticiones encadenadas una tras otra:

```
DELETE FROM site WHERE user_id = 'ID_DEL_USUARIO';
DELETE FROM creditcard WHERE user_id = 'ID_DEL_USUARIO';
DELETE FROM note WHERE user_id = 'ID_DEL_USUARIO';
DELETE FROM password_change_request WHERE user_id = 'ID_DEL_USUARIO';
```

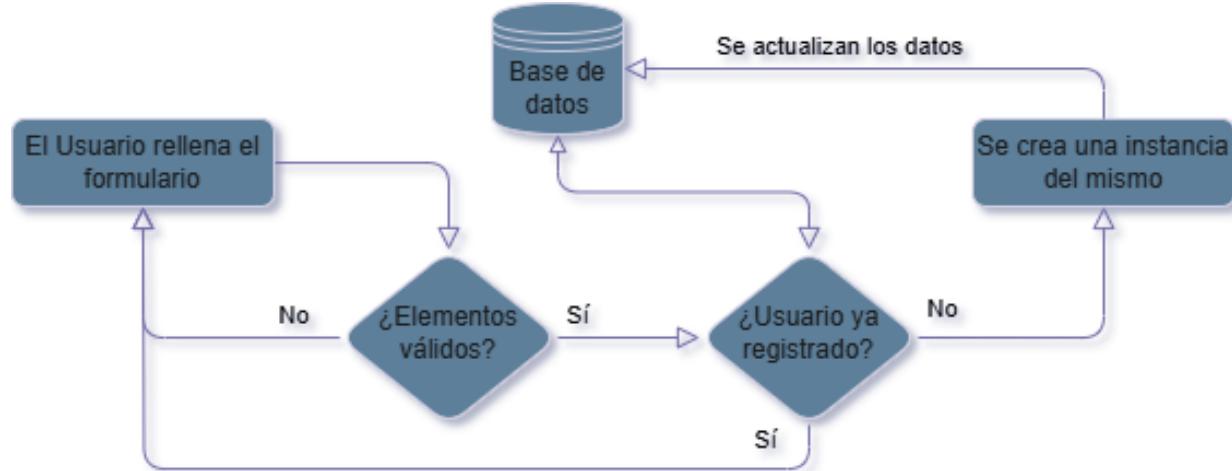
Finalmente, guardará los cambios con el comando:

```
COMMIT;
```

Cabe destacar que todas las transacciones realizadas con la base de datos son atómicas (uno de los principios de *ACID*, que son atomicidad, consistencia, aislamiento y durabilidad) y significa que una acción o se realiza o no. Es por eso que, hasta que no se confirma que todas las peticiones se han realizado correctamente no se procede a guardar los cambios.

Ahora bien, **¿cómo se comunican entre todas las partes?** A continuación hemos incluido una serie de mapas conceptuales de flujo de trabajo que representan la forma interna que utilizan ambos elementos (el *front-end* o nuestra aplicación y el *back-end* o la base de datos) para su funcionamiento:

Creación de usuario



Aunque la creación de usuario se realice de la siguiente manera, el proceso interno del lenguaje **SQL** es distinto:

El usuario rellena el formulario > Se validan los datos introducidos > Se realiza una petición a la base de datos para comprobar que dicho usuario no existe > Se crea una instancia del usuario > Se realiza un commit para guardar la base de datos.

Primero se busca el usuario:

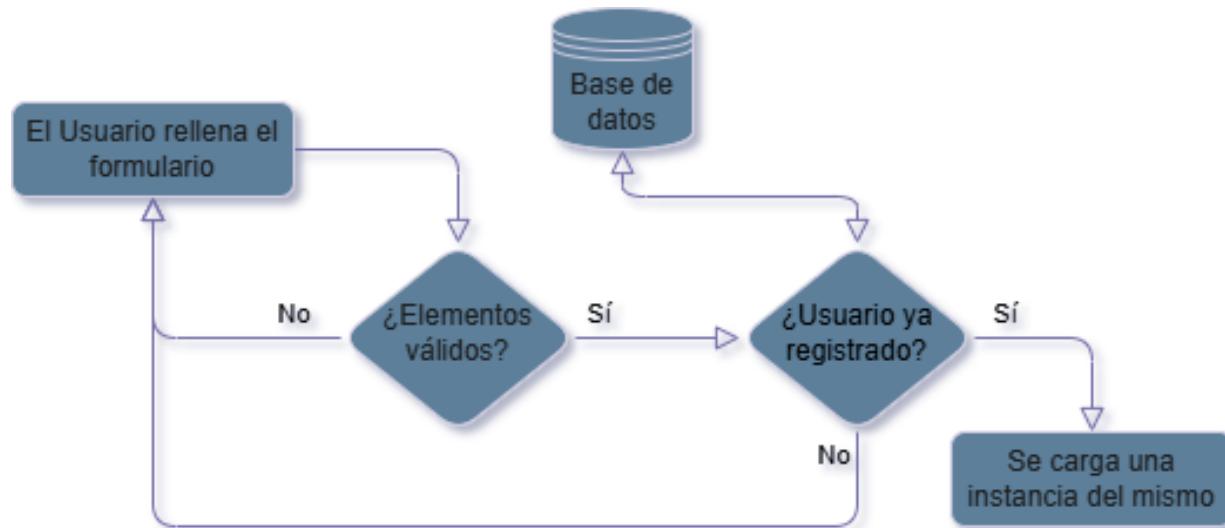
```
SELECT * FROM user WHERE email = 'email_input';
```

Después se crea la instancia en memoria:

```
INSERT INTO user (id, role, fullname, email, hashed_password, created)
VALUES ('ID_GENERADO', 'client', 'name_input', 'email_input',
'hashed_password_input', 'fecha_actual');
```

Al realizar la petición de guardado es cuando se almacenan los datos en la base de datos.

Inicio de sesión



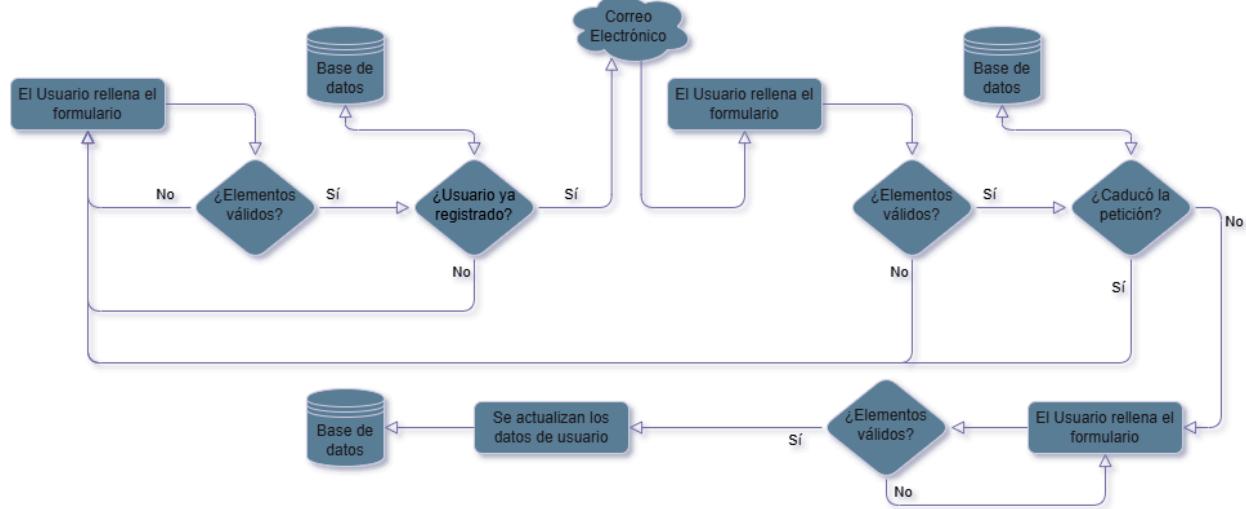
Para el inicio de sesión se utiliza más lógica en la aplicación que en la consulta, pues la petición a la base de datos solo devuelve el usuario si éste se encuentra almacenado:

El usuario rellena los campos > Se validan los datos introducidos > Se realiza una petición a la base de datos para comprobar si dicho usuario existe > Se comprueba que la contraseña introducida es la misma que la almacenada > Se carga la instancia de usuario y se almacena en memoria.

En este flujo de trabajo, la petición será la misma que la anterior, y simplemente se almacenará el resultado en memoria. Ahora ya tendremos la sesión iniciada:

```
SELECT * FROM user WHERE email = 'email_input';
```

Reseteo de contraseña



El reseteo de la contraseña si el usuario olvida sus credenciales es, hasta ahora, la funcionalidad más compleja. Hay que tener en cuenta el tiempo de caducidad del token y la actualización de los datos en la base de datos, así que su flujo principal será el siguiente:

Se rellena el formulario > Se validan los datos > Se comprueba si el usuario existe en la base de datos > Se envía un mensaje por correo electrónico facilitando un código > El usuario rellena los campos con dicho código > Se validan los datos > Se comprueba que no haya caducado la petición de reseteo > Se permite crear una nueva contraseña > Se validan los datos > Se actualizan los datos de Usuario > Se realiza un 'commit' para guardar la base de datos nuevamente.

A partir de ahí, las peticiones se realizarán del siguiente modo:

```
SELECT * FROM user WHERE email = 'email';
```

Esta petición nos asegura que el usuario existe y nos permite trabajar con él en caso de que sea cierto.

Al confirmar la petición, debemos actualizar la base de datos con la nueva instancia:

```
INSERT INTO password_change_request (id, user_id, encrypted_code, created)
VALUES ('ID GENERADO', 'USER_ID', 'CODIGO_ENcriptado', 'FECHA_ACTUAL');
COMMIT;
```

Esta situación almacena en la base de datos la nueva petición junto con el *token* ya encriptado.

Una vez el usuario obtiene el código en su correo, procederá a introducirlo en el formulario, donde se comprobará su existencia y caducidad. Para ello deberemos realizar una petición a la base de datos para que nos muestre el último código pedido por el usuario:

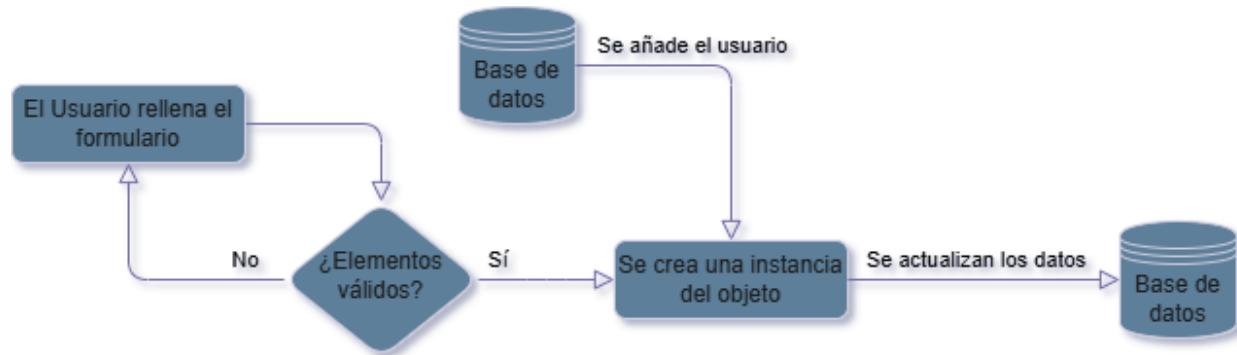
```
SELECT * FROM password_change_request WHERE user_id = 'USER_ID'
ORDER BY created DESC LIMIT 1;
```

Si todos los datos son válidos, se procederá a permitir al usuario a actualizar su contraseña y a cambiar los datos almacenados:

```
UPDATE user SET hashed_password = 'NUEVO_HASH_DE CONTRASEÑA'
WHERE id = 'USER_ID';
COMMIT;
```

Y aquí finaliza la actualización de las credenciales después de su solicitud.

Adición de elementos



La agregación de nuevos elementos se realizará de la siguiente forma en general, ya sea una nueva nota, una nueva tarjeta de crédito o un nuevo sitio web:

Se rellena el formulario > Se validan los datos > Se crea una nueva instancia del elemento a agregar > Se añade la instancia de Usuario almacenada en memoria > Se realiza un 'commit' para guardar la base de datos nuevamente.

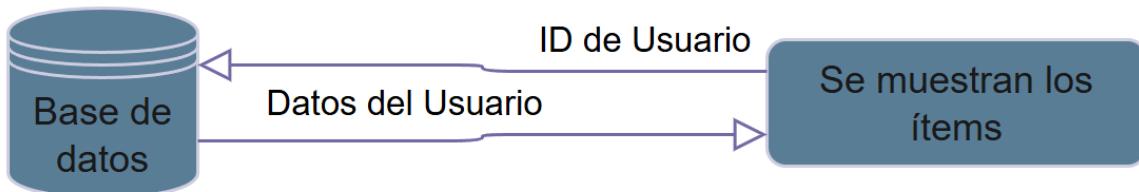
Por tanto, y sólo como ejemplo, mostramos la petición en el caso de nuestra clase `Site`:

```

INSERT INTO site (id, user_id, name, address, username, encrypted_password,
created) VALUES ('ID_GENERADO', 'USER_ID', 'new_name', 'new_address',
'new_username', 'encrypted_password', 'fecha_actual');
COMMIT;
  
```

Esta petición crea una instancia del objeto mapeado con los siguientes datos y sólo se almacenan en la base de datos una vez se ejecuta el *commit*.

Carga de elementos



Hasta ahora, todas las peticiones realizadas se han hecho cuando el usuario lo requería. Sin embargo, esta es la única petición que se realiza de forma automática cada vez que el usuario realiza una modificación. Esto ocurre porque se encarga de actualizar los elementos que hay en memoria para poder mostrar los datos en la interfaz:

Se realiza una petición a la base de datos filtrando los datos según el ID de usuario > Se muestran dichos datos.

Sin embargo, hay una opción en la aplicación que permite la búsqueda de elementos según qué palabra. Actualmente se ha realizado muy básica, pero el concepto se puede ver claramente:

```

SELECT * FROM site WHERE user_id = 'USER_ID' AND name LIKE '%USER_INPUT%';
SELECT * FROM creditcard WHERE user_id = 'USER_ID' AND alias LIKE
'%USER_INPUT%';
SELECT * FROM note WHERE user_id = 'USER_ID' AND title LIKE '%USER_INPUT%';
  
```

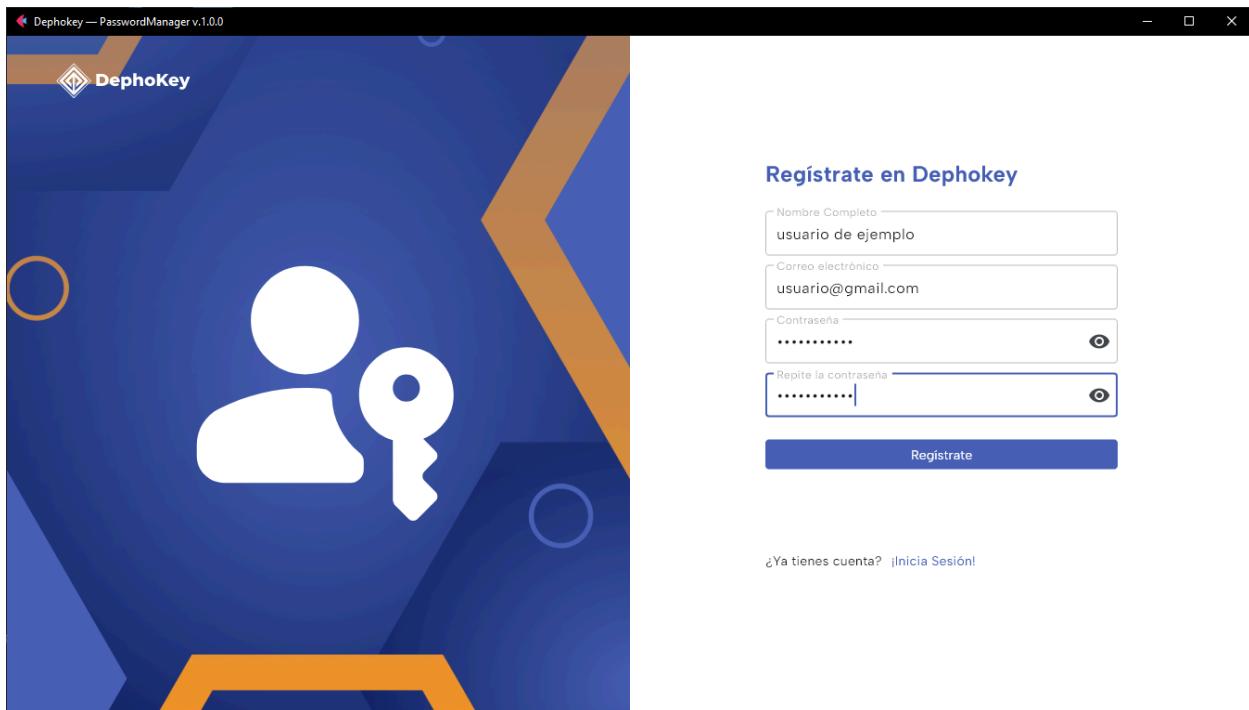
Esta solicitud se encarga de mostrar los diferentes elementos cuyo atributo coincide parcialmente (por eso se emplean los símbolos de porcentaje) con el *input* del usuario.

Principales casos de uso

Esta sección se encarga de explicar las funcionalidades que puede desempeñar nuestra aplicación.

Creación de usuario

Para crear una nueva cuenta, el usuario debe navegar a la página de registro y, una vez allí, rellena el formulario y lo envía.



Si el formulario no presenta ningún error, la aplicación redirige al usuario a la página de inicio de sesión, informando al mismo de que la cuenta ha sido creada con éxito.

Si abrimos la base de datos con **DB Browser**, podemos comprobar que el usuario `usuario@gmail.com` se ha agregado con éxito.

The screenshot shows the DB Browser for SQLite interface. The title bar indicates it's connected to a database at C:\Users\jpast\PycharmProjects\dephockey\data\database\database.db. The main window displays a table named 'user' with the following data:

	id	role	fullname	email	hashed_password	crea
1	w7f7mohf4ov15by	ADMIN	Sergio Administrativo Muñoz	admin.24@gmail.com	60fe74406e7f353ed979f350f2fbb6a2e86...	2025-03-24 16:24:40
2	bt9br4h0f2j15x0	CLIENT	Manuel Tester Garcia	client.24@gmail.com	e6d1c38f0d9d46c5512533d49f90888312a...	2025-03-24 16:24:40
3	tbqn18qwlebmcdc	CLIENT	Usuario De Ejemplo	usuario@gmail.com	8a82a5f6fc722db8a57e25b508f5a008ac6...	2025-03-25 10:44:20

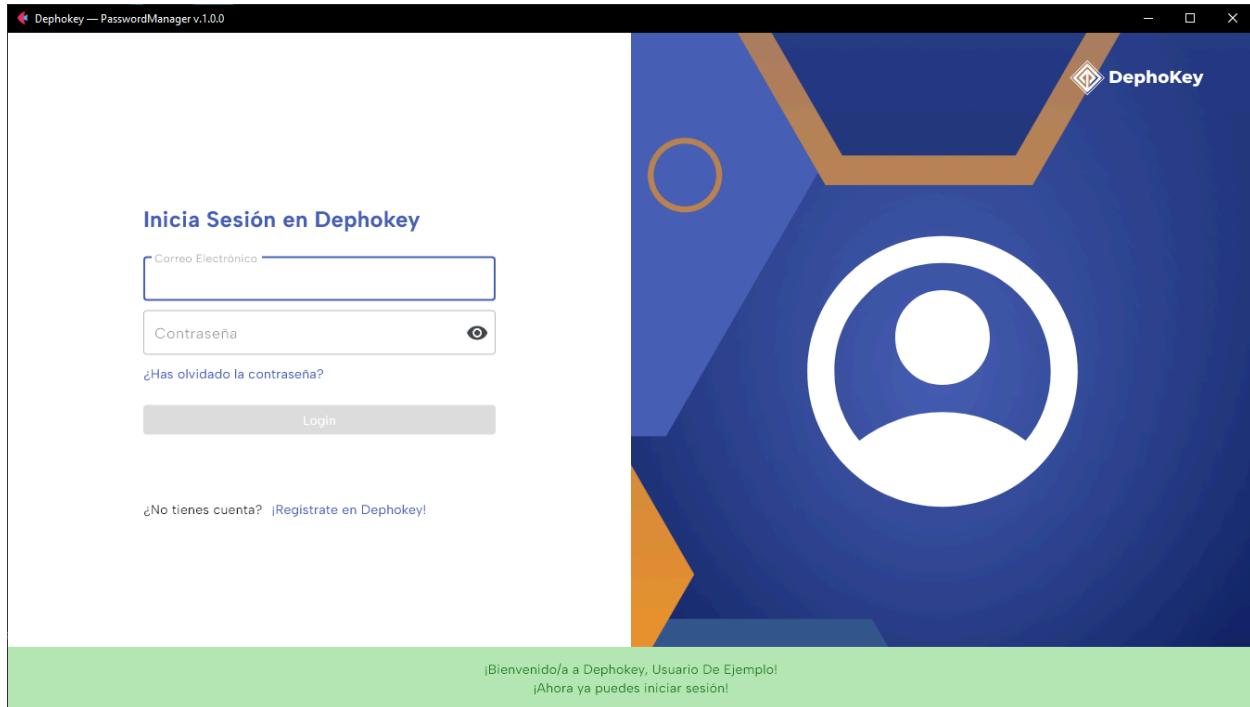
The right side of the interface shows the 'Editando fila=1, columna=6' (Editing row 1, column 6) dialog, which is currently set to 'Texto / Numérico; Tamaño: 1 carácter' (Text / Number; Size: 1 character). The 'Nombre' field is empty.

También confirmamos que el usuario tiene un ID correcto, un rol especificado, nombre completo y correo electrónico y lo más importante, una contraseña encriptada.

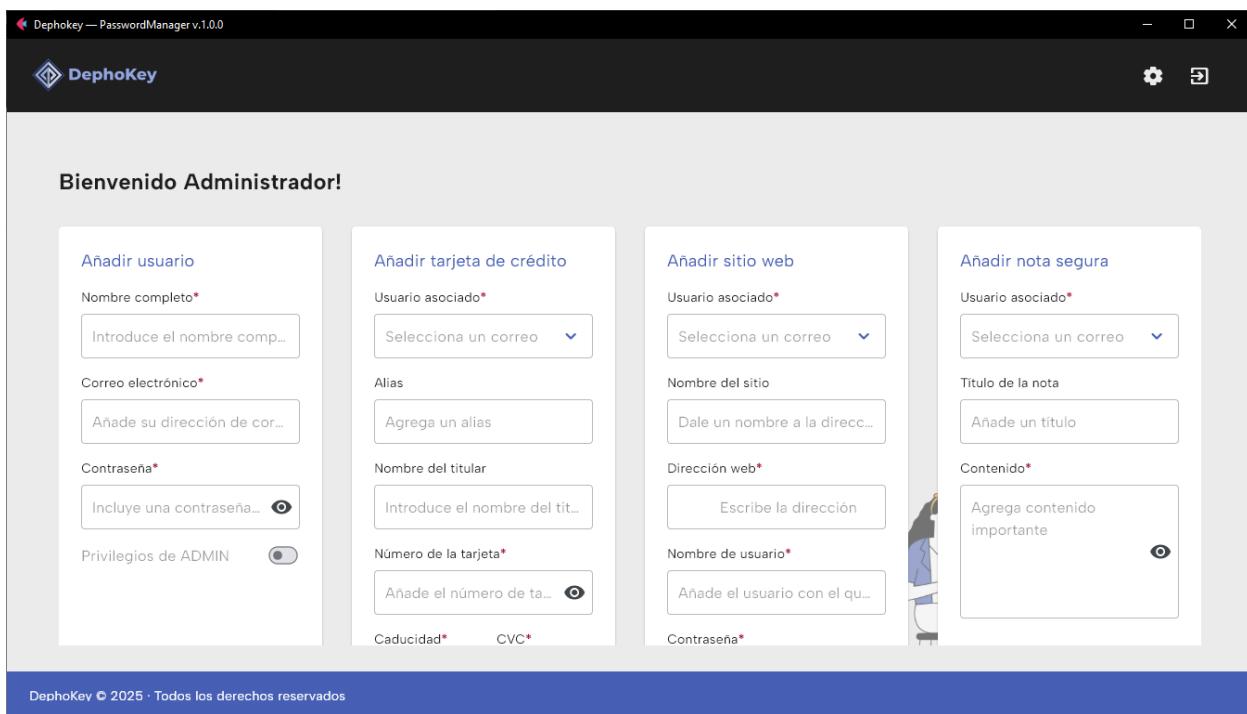
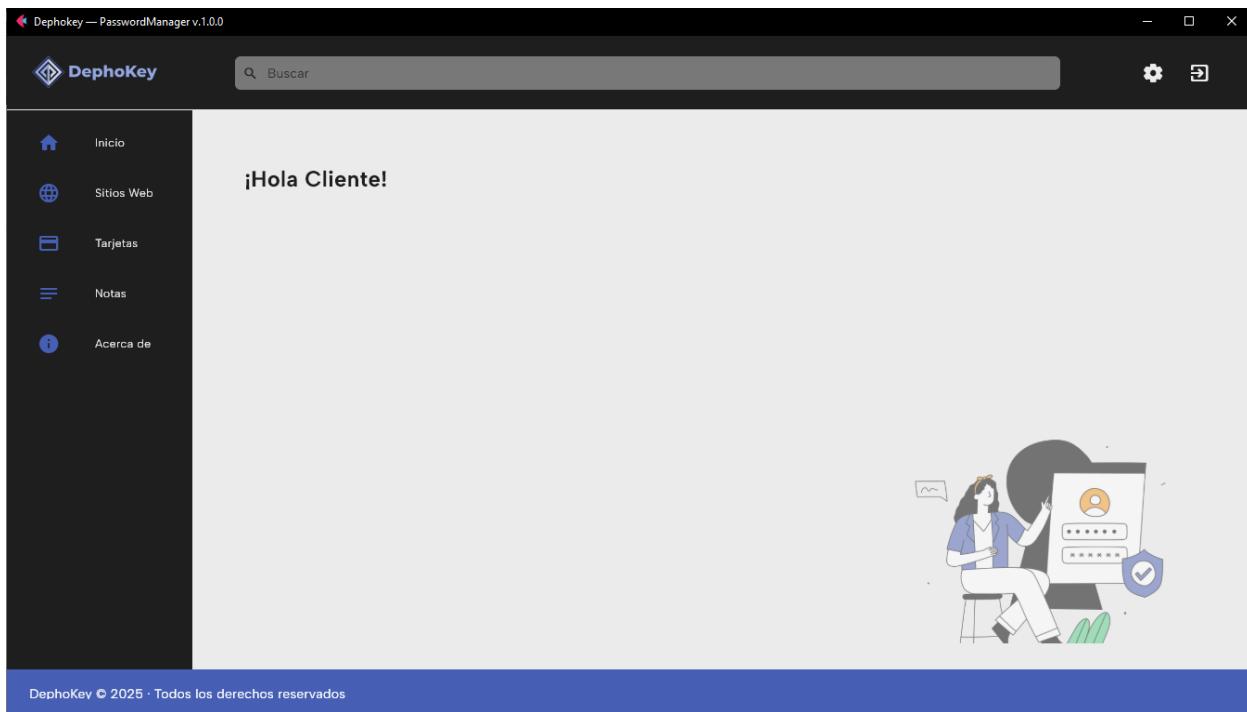
Después de realizar estas comprobaciones, podemos volver a nuestra aplicación y verificamos que se ha redirigido a la página de inicio de sesión.

Inicio de sesión

Para iniciar sesión en la aplicación el usuario debe llenar los campos de texto de correo electrónico y contraseña. Una vez listo, pulsa sobre el botón de **Iniciar sesión** o la tecla **enter**.

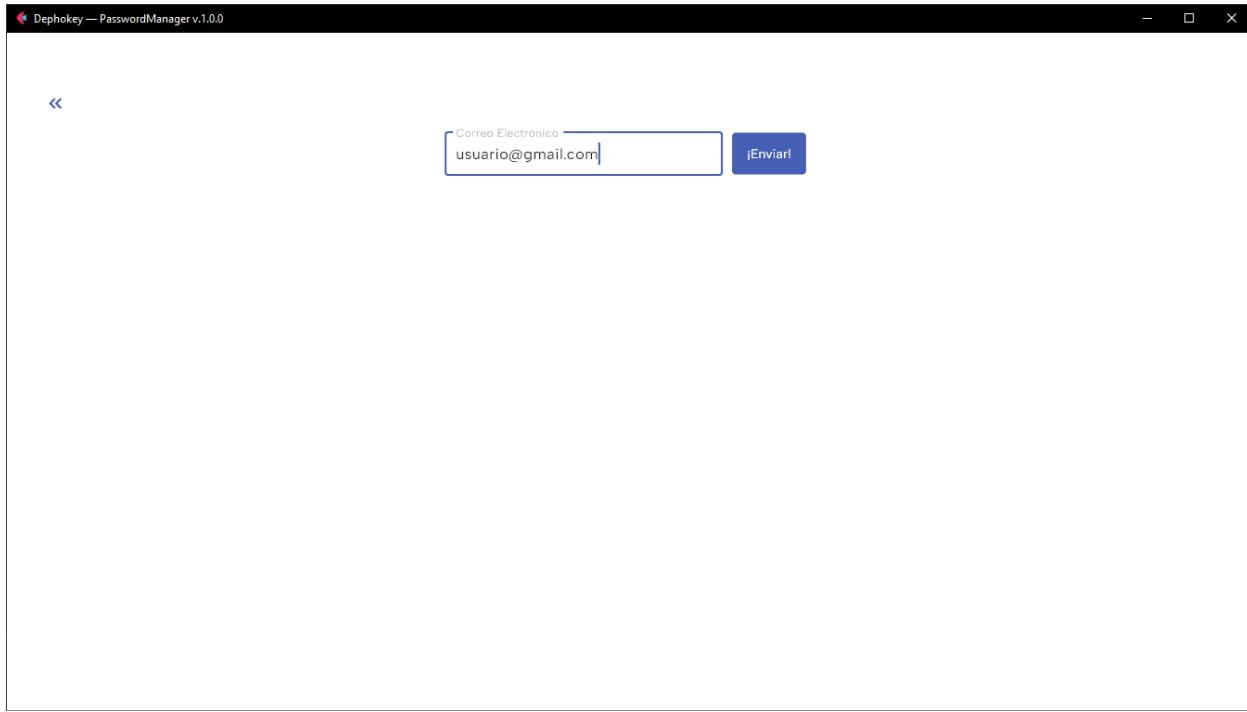


Si la cuenta existe, la aplicación redirige al usuario a la página *home* donde puede ver el saludo inicial. Dicha página varía su contenido según el rol del usuario, seccionando el flujo de trabajo según se es *CLIENTE* o *ADMINISTRADOR*.

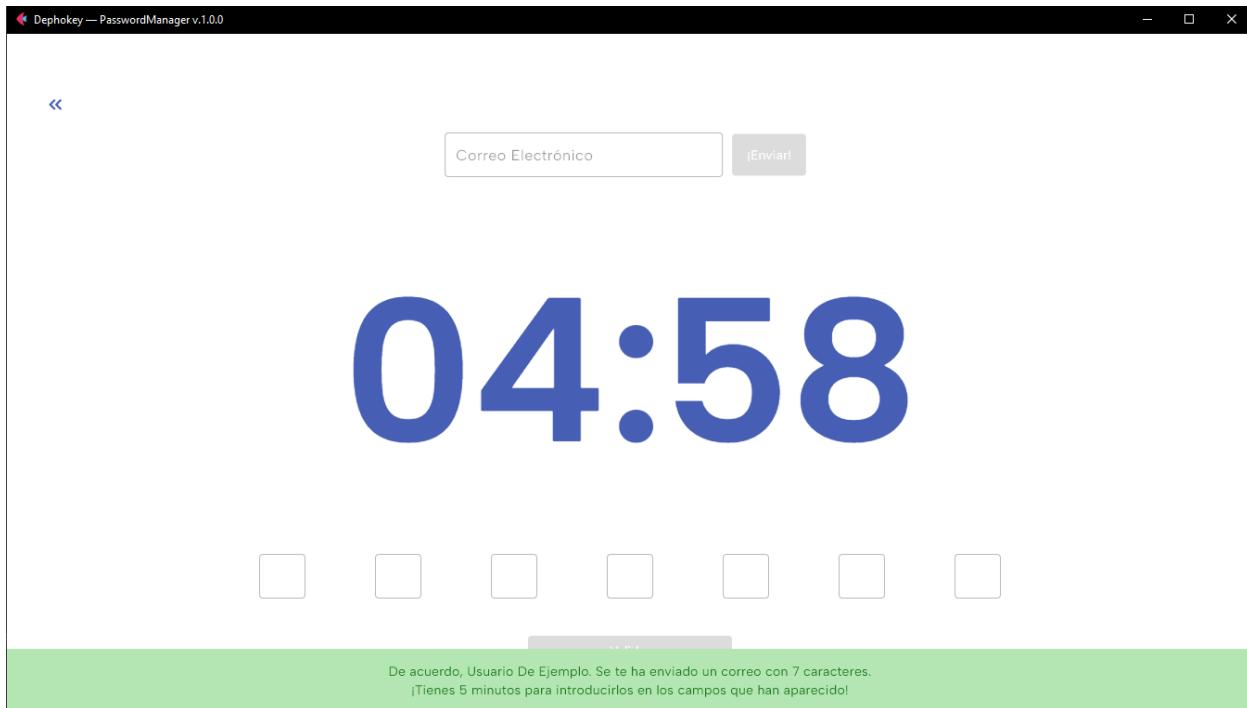


Reseteo de contraseña

Si el usuario olvida sus credenciales puede acceder a la página de recuperación de contraseña, donde debe introducir el correo electrónico que registró en la aplicación.



Si el correo existe y es válido, el contenido de la página se actualiza y podemos ver una cuenta regresiva y un mensaje informativo que dice que el correo de recuperación ha sido enviado. Si no es posible, se muestra un mensaje informando al usuario de que el correo no se ha podido enviar correctamente.



Si abrimos nuestra base de datos podemos confirmar también que se ha creado una nueva instancia de **ResetPassword** y que sus datos se han guardado correctamente: un ID único, el ID del usuario asociado a dicha petición, la fecha de creación del mismo y hasta qué hora es válida. Cabe resaltar que el código también es almacenado de forma encriptada tal y como se especifica en la sección de **Encriptación y Desencriptación**.

Una vez recibido el email con el *token* generado, el usuario tan solo debe introducirlo de forma correcta rellenando los campos que se muestran en la aplicación. Si el *token* es válido permite la creación de una nueva contraseña o informa al usuario de que no es posible en caso contrario.

DB Browser for SQLite - C:\Users\jpast\PycharmProjects\dephokey\data\database\database.db

Archivo Editar Ver Herramientas Ayuda

Nueva base de datos Abrir base de datos Guardar cambios Revertir cambios Deshacer Abrir proyecto Guardar proyecto Anexar base de datos Cerrar base de datos

Estructura Hoja de datos Editar pruebas Ejecutar SQL

Table: password_change_request

id	user_id	encrypted_code	created	expires_at
1	Onf9zpu0atwcrfa	bt9br4h0f2j15x0	gAAAAABn4aOYuSShyra3YsYbt335LLwmjYp...	2025-03-24 19:25:28.713504 2025-03-24 19:30:28.713504
2	6ieazv3oagkwk5j	tbqn18qwlkmcdc	gAAAAABn4op6afffxS20DrzKicsQnpLtc...	2025-03-25 11:50:34.083116 2025-03-25 11:55:34.083115

Modo: Texto

1

Editando fila=1, columna=0
Tipo: Texto / Numérico; Tamaño: 1 carácter

Aplicar

Remoto Selecciona una identidad Cargar al servidor

DBHub.io Local Base de datos actual

Nombre

Último cambio: 2025-03-25 11:55:34.083115

Historial de SQL Gráfica Esquema Remoto

Ir a: 1

Crea un nuevo archivo de base de datos

Último cambio: 2025-03-25 11:55:34.083115

MailCatcher (1)

localhost:1080

MailCatcher

From: <dephokey.team@gmail.com> size=9888 To: <usuario@gmail.com> Subject: ¡Tu token para Dephokey está listo! Received: Tuesday, 25 Mar 2025 11:54:24 AM

Received: Tuesday, 25 Mar 2025 11:54:24 AM
From: <dephokey.team@gmail.com> size=9888
To: <usuario@gmail.com>
Subject: ¡Tu token para Dephokey está listo!
Attachments: [logotype.png](#)

HTML Plain Text Source Download

 DephoKey

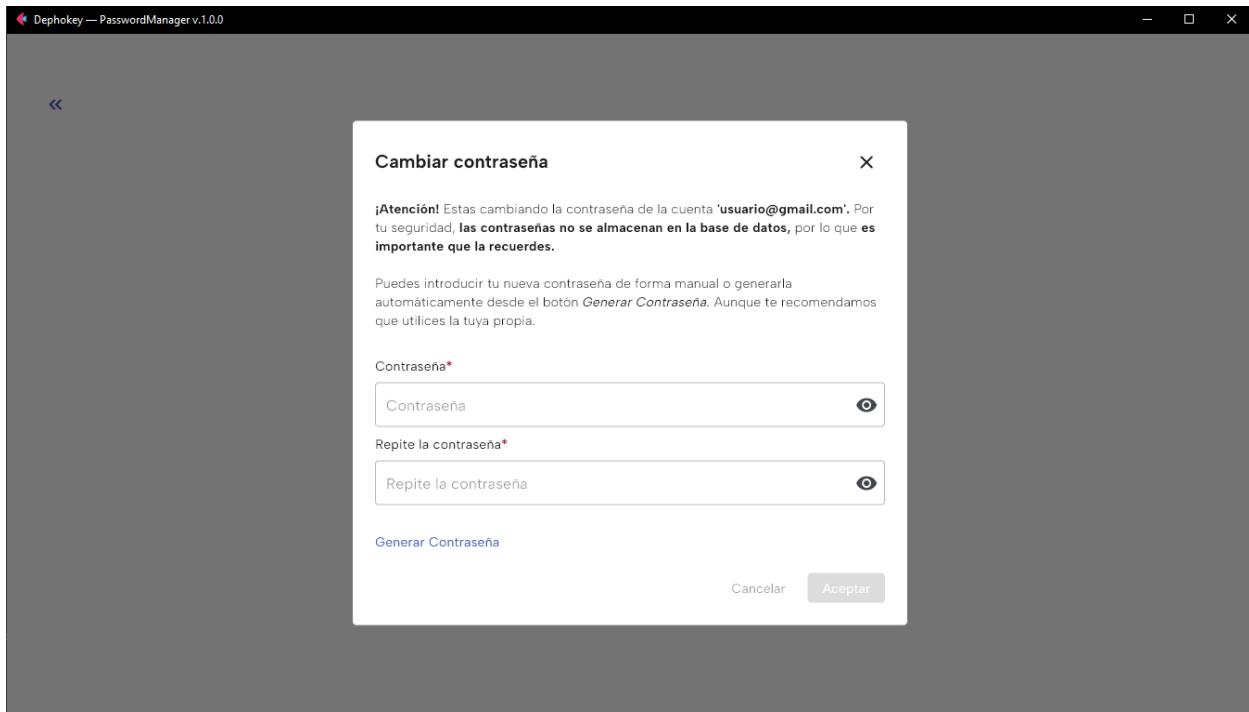
Código de Verificación

Hola Usuario!
Por favor, introduce en el programa el código de siete caracteres proporcionado para poder actualizar tu contraseña.

NPDPAJD

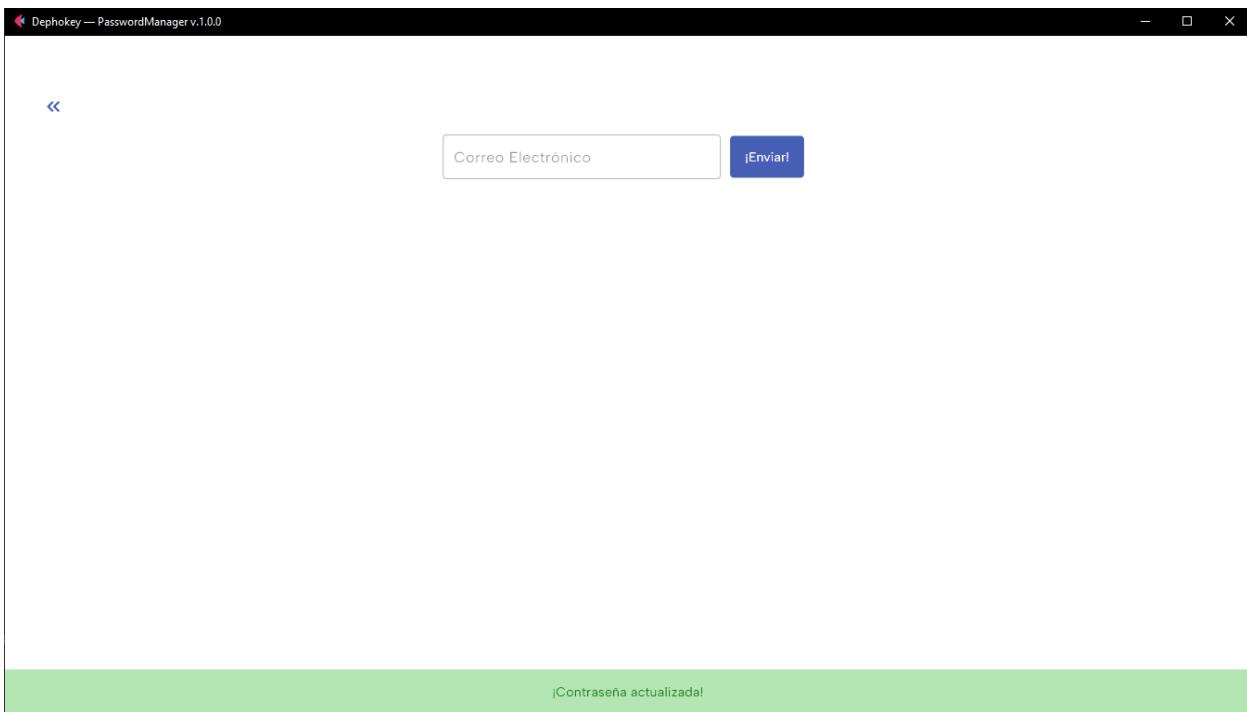
Si no has realizado la petición, puedes ignorar este email.

Atentamente,
El equipo Dephokey



Si el *token* es válido y se encuentra dentro del período de tiempo acordado, se abre un formulario para actualizar la contraseña del usuario solicitante.

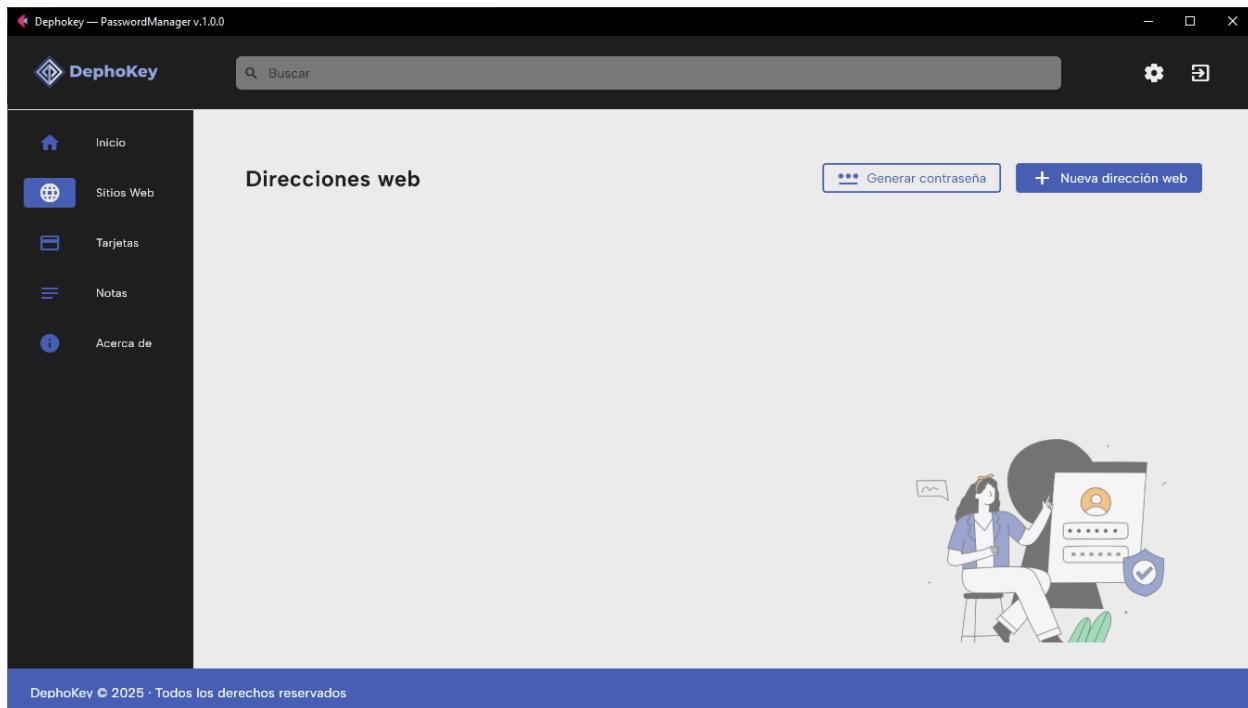
Cuando la contraseña ha sido actualizada dentro de los estándares establecidos, el formulario se cierra automáticamente, la página se resetea a su estado inicial y se muestra un mensaje informando al usuario de que la contraseña ha sido actualizada de manera exitosa.



Ahora el usuario puede volver a la página de inicio de sesión haciendo *click* en el botón situado en la parte superior izquierda.

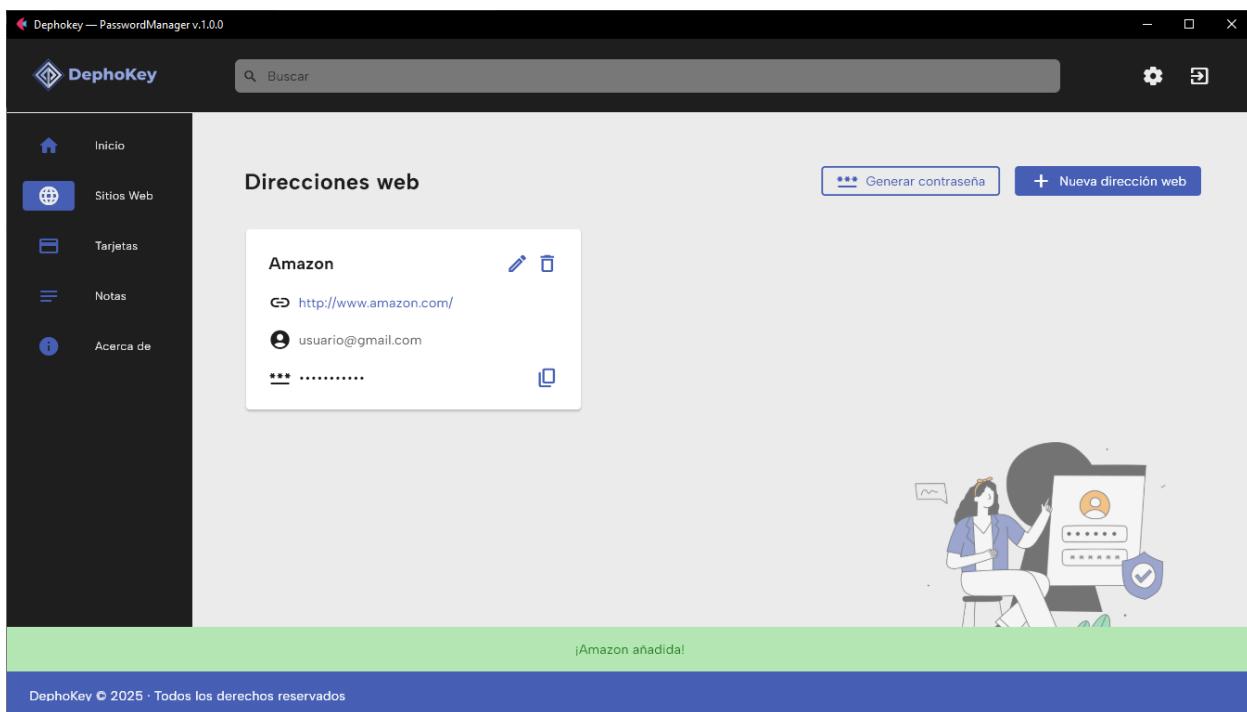
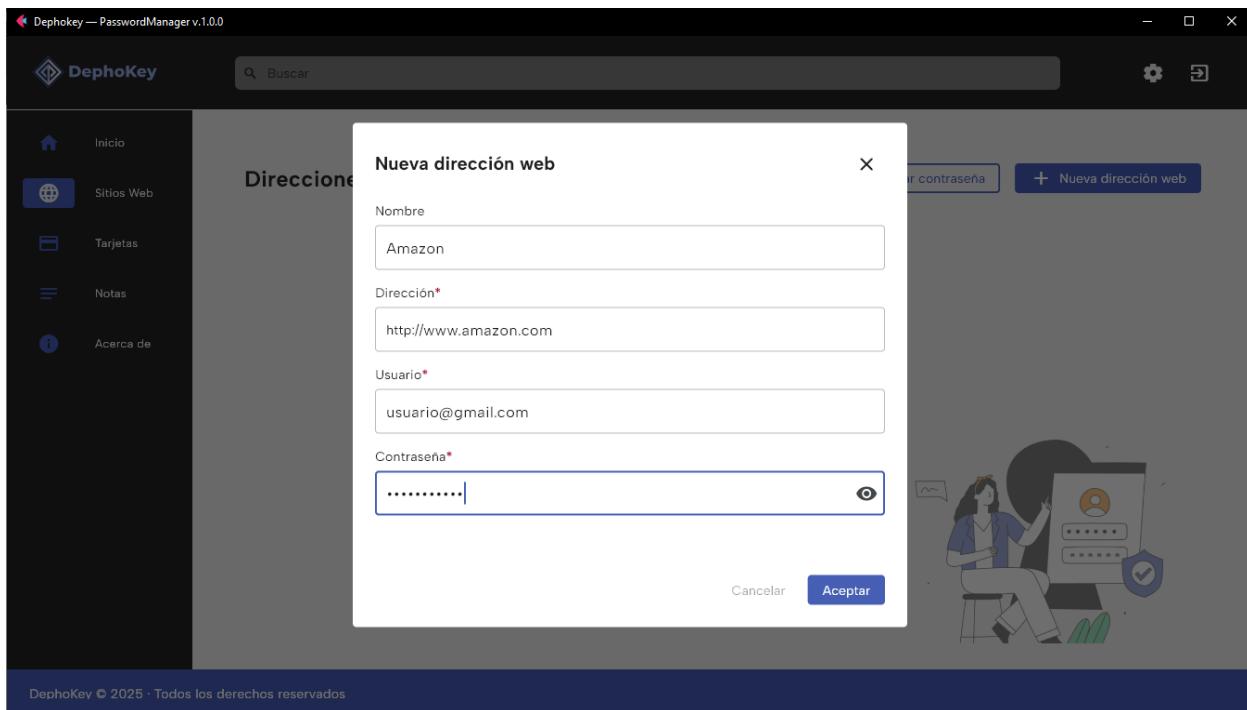
Adición, edición y eliminación de elementos

Para la adición de elementos, el usuario tan solo debe ir a la sección de la página donde se quiere agregar información. En el siguiente ejemplo vamos a ver cómo el usuario agrega un sitio web nuevo. Para ello se desplaza a la página de sitios web y pulsa el botón de **Nueva dirección web**.

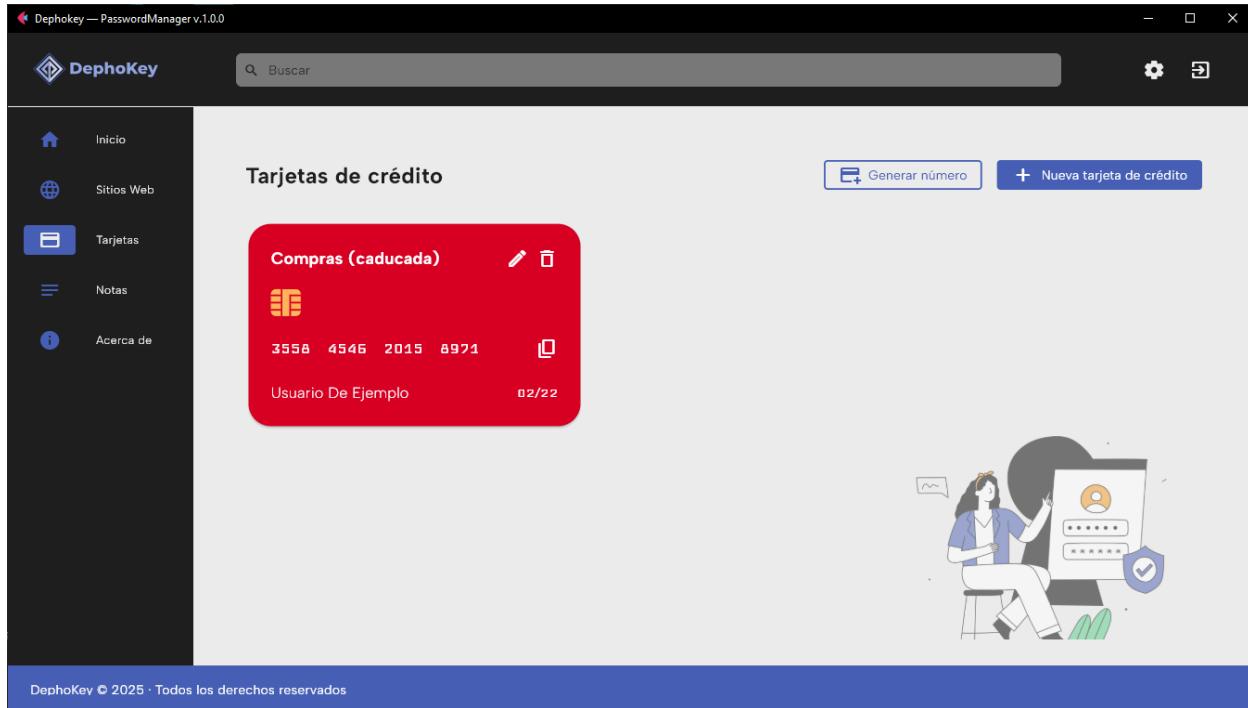


Se abre un formulario que se rellena para guardar la información en la base de datos. Una vez se pulse el botón **Aceptar** y, mientras el formulario tenga los campos marcados con un asterisco, él mismo se cierra y actualiza la página. Se informa al usuario con una barra de texto.

4	bp5r0zcwz9rjdti bt9br4h0f2j15x0 Soluciones.com	https://suministros.com/	client.24@gmail.com gAAAAABn4ZxPV_C5qrw_AH	
5	ctt2pycw3ndsfjm bt9br4h0f2j15x0 Pareja.es	https://banco.org/	client.24@gmail.com gAAAAABn4ZxP6oXCrGar-	
6	9cegettkk0zemd bt9br4h0f2j15x0 Servicios.com	http://www.castillo.org/	client.24@gmail.com gAAAAABn4ZxP8ZQv_V-2M	
7	duylnkafb3doqfm bt9br4h0f2j15x0 Construcion.es	http://promociones.org/	client.24@gmail.com gAAAAABn42xPXY0ntkl2IT	
8	jpzq2zt8uzq8ulm bt9br4h0f2j15x0 Gras.es	http://juan.com/	client.24@gmail.com gAAAAABn4ZxPyzIwlw6aA	
9	jc8000kbjecjaazv bt9br4h0f2j15x0 Restauracion.es	http://hermanos.com/	client.24@gmail.com gAAAAABn4ZxPDUR2eTYuKx	
10	naolc8thduzzuiy bt9br4h0f2j15x0 Hnos.es	https://mendizabal.es/	client.24@gmail.com gAAAAABn4ZxPXP1bSQ2Q1H	
11	g5cc9bzlimixc5s tbqn18qw1ebmcde Amazon	http://www.amazon.com/	usuario@gmail.com gAAAAABn4r9bknaid9d92D	



Para la edición de elementos, el usuario tan solo tiene que ir a la sección de la aplicación donde tenga algún elemento que modificar. El *widget* que brinda la información del mismo dispone de un botón en su parte superior derecha con el icono de un lápiz. Debe pulsar sobre él.



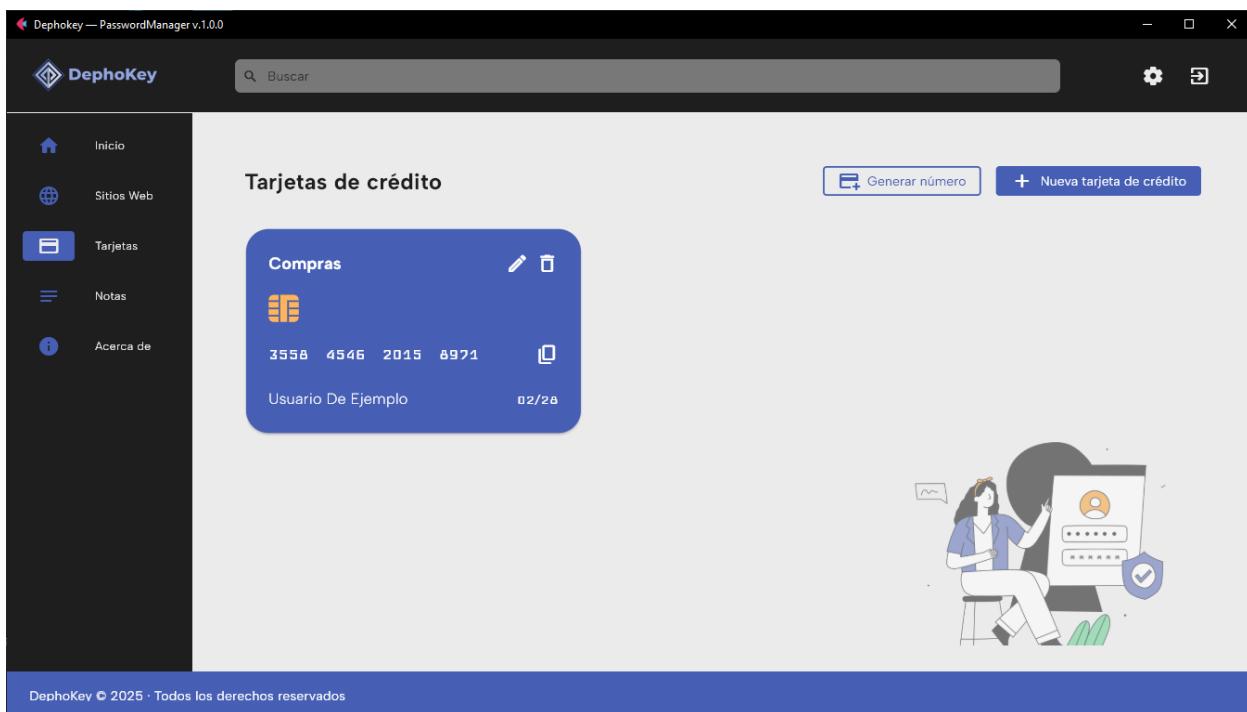
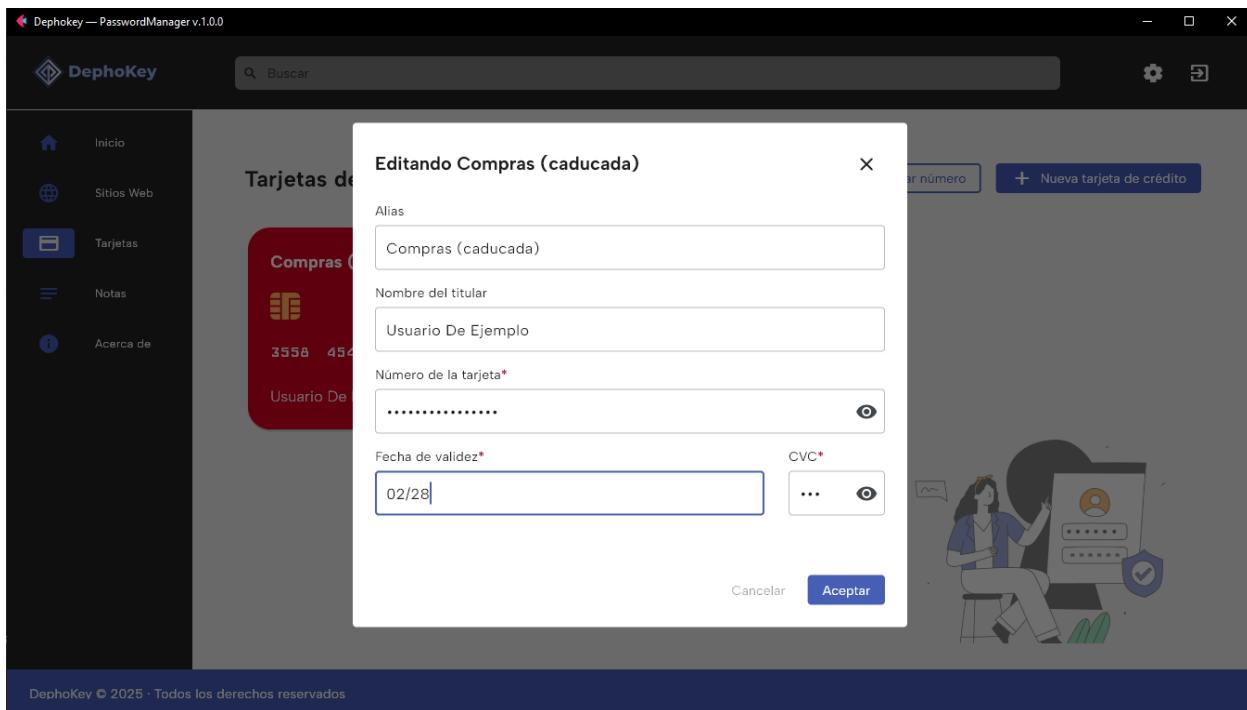
Esta tarjeta de crédito aparece caducada porque la fecha de expiración está mal puesta. Para ello el usuario debe modificar sólo ese campo y pulsar en el botón **Aceptar** del formulario abierto.

Como podemos ver, la tarjeta ha cambiado su estado a no caducada, ha actualizado su color y los datos que hemos modificado se ven reflejados en la base de datos.

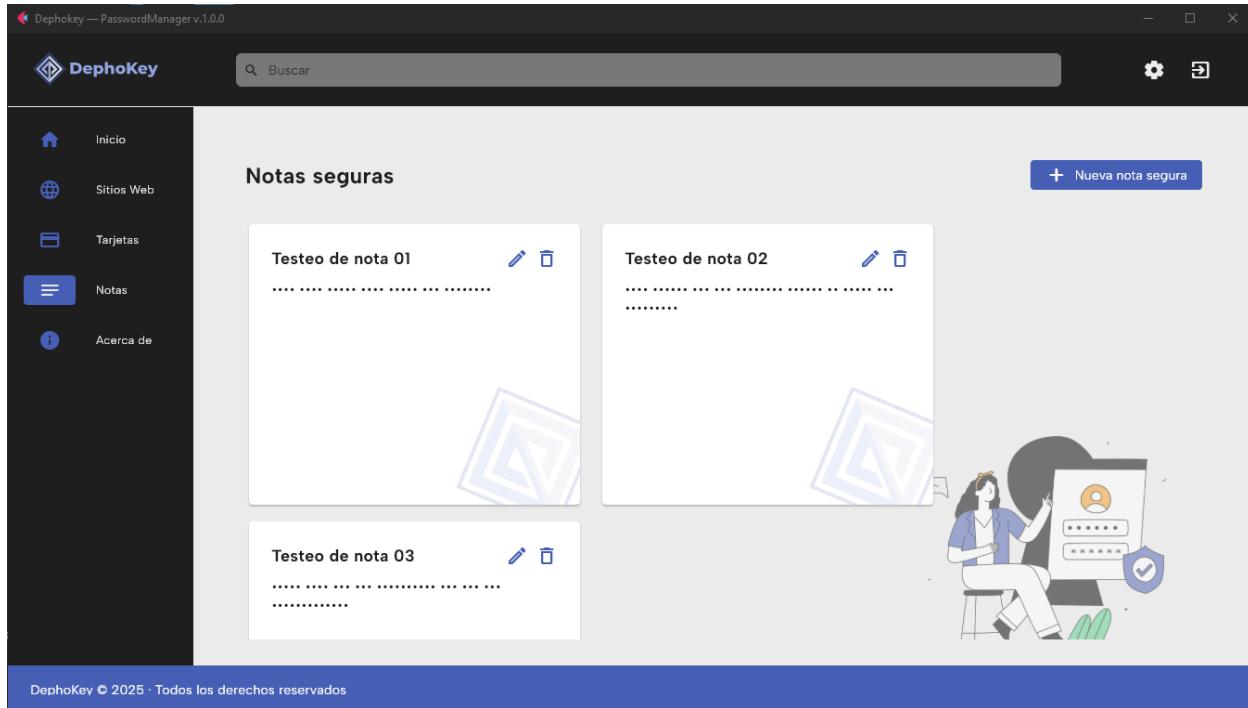
4	Lz...	gAAAAABn4ZxPgdiSvFDhMQTowYQpvVG6oVq...	2022-07-07 18:08:19.000000	1 Possimus (caducada)	2025-03-24 18:54:23.647557	
5	gAAAAABn4ZxPowGQ4HP1h_PI3CM1qFfHkqD...		2022-05-21 21:24:02.000000	1 A (caducada)	2025-03-24 18:54:23.647557	
6	Un...	gAAAAABn4ZxPRA2PQoFekcbaFlGFgcPbFCG...	2024-07-13 13:21:54.000000	1 Deserunt (caducada)	2025-03-24 18:54:23.649556	
7	gAAAAABn4ZxPtKy7igCteMVEjDuSIJ-...		2028-05-08 12:39:21.000000	0 Iusto	2025-03-24 18:54:23.649556	
8	Rg...	gAAAAABn4ZxPW08VG2tBxejV6j17ct2Npof...	2027-09-16 00:38:01.000000	0 Aperiam	2025-03-24 18:54:23.649556	
9	f1...	gAAAAABn4ZxPmnX-lAIAsGaVylf14wIJwtl...	2021-06-26 00:53:45.000000	1 Dolorem (caducada)	2025-03-24 18:54:23.650556	
10	yz...	gAAAAABn4ZxPP8b9oeb_TIpnoCwx7aOdIzs...	2028-04-16 15:52:15.000000	0 Aliquid	2025-03-24 18:54:23.651556	
11	UL...	gAAAAABn4r7cuhjDNtLeXQ9Y00Q7jLSUIBn...	2028-02-01 00:00:00.000000	0 Compras	2025-03-25 12:57:43.580939	

Overlaid on the table is a modal dialog box with the following content:

- Text: "Editando fila=1; columna=1"
 - Type: Texto / Numérico; Tamaño: 15 caracteres
 - Buttons: Aplicar, Remoto, Cargar al servidor
- Text: "Idioma: Seleccione una idioma r" with a dropdown menu.
- Text: "DBHub.io Local Base de datos actual" with a dropdown menu.
- Text: "Nombre" with a text input field.



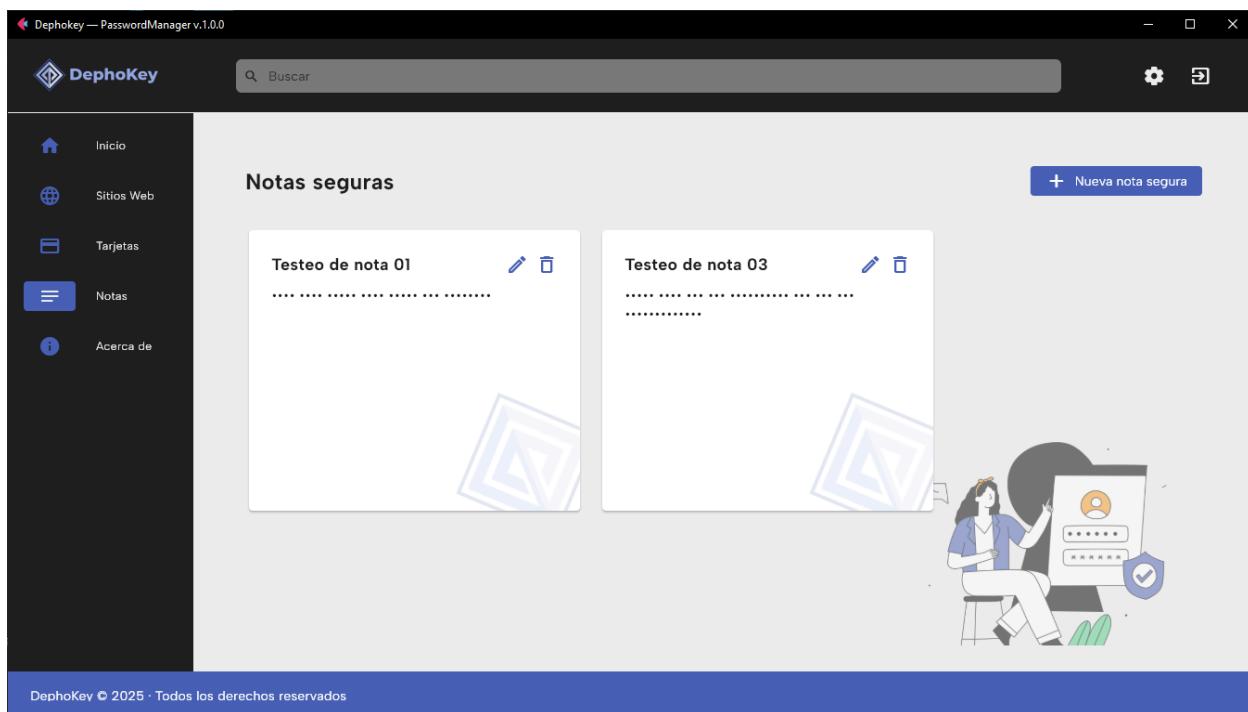
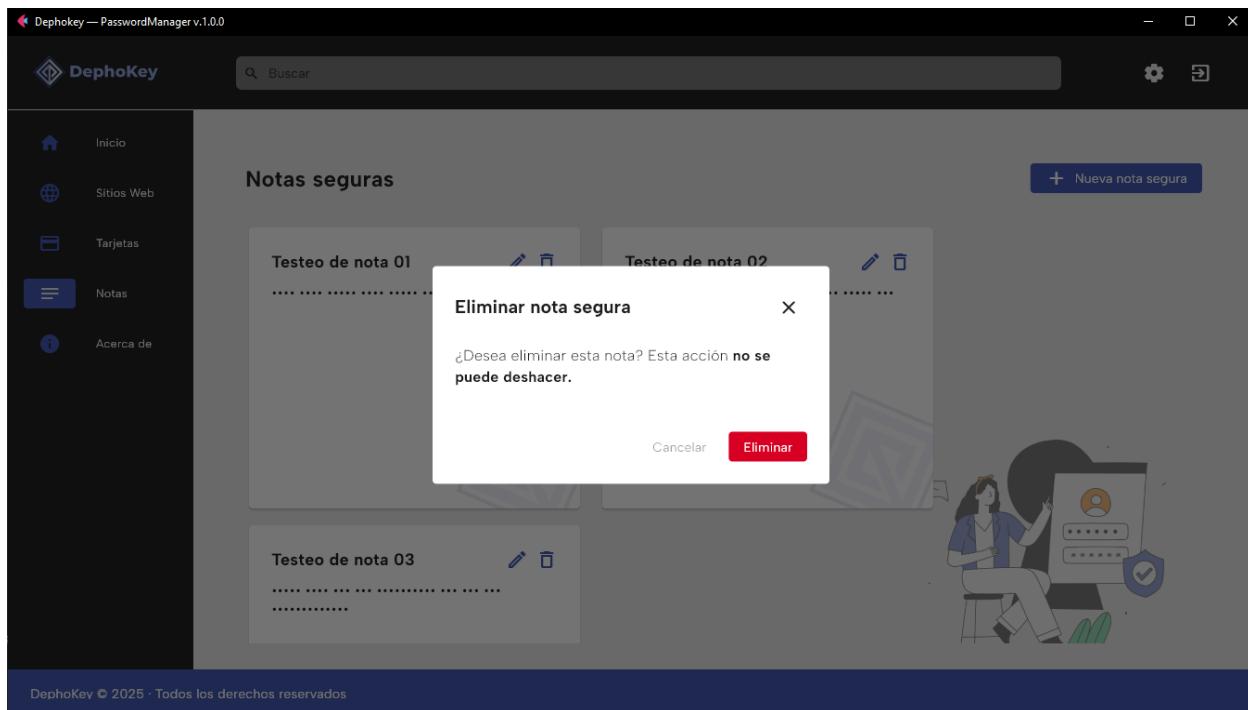
Para la eliminación de un ítem, el usuario debe navegar hasta la sección del ítem que desea eliminar. Como antes, tan solo debe buscar el botón diseñado para eliminar el mismo desde el propio *widget* (en el ejemplo, un ícono con forma de papelera) y debe pulsar sobre él.



Se abre una ventana emergente que pregunta si quiere eliminar el elemento. Con tan sólo aceptar, la aplicación se encarga de actualizar tanto la base de datos como la interfaz gráfica.

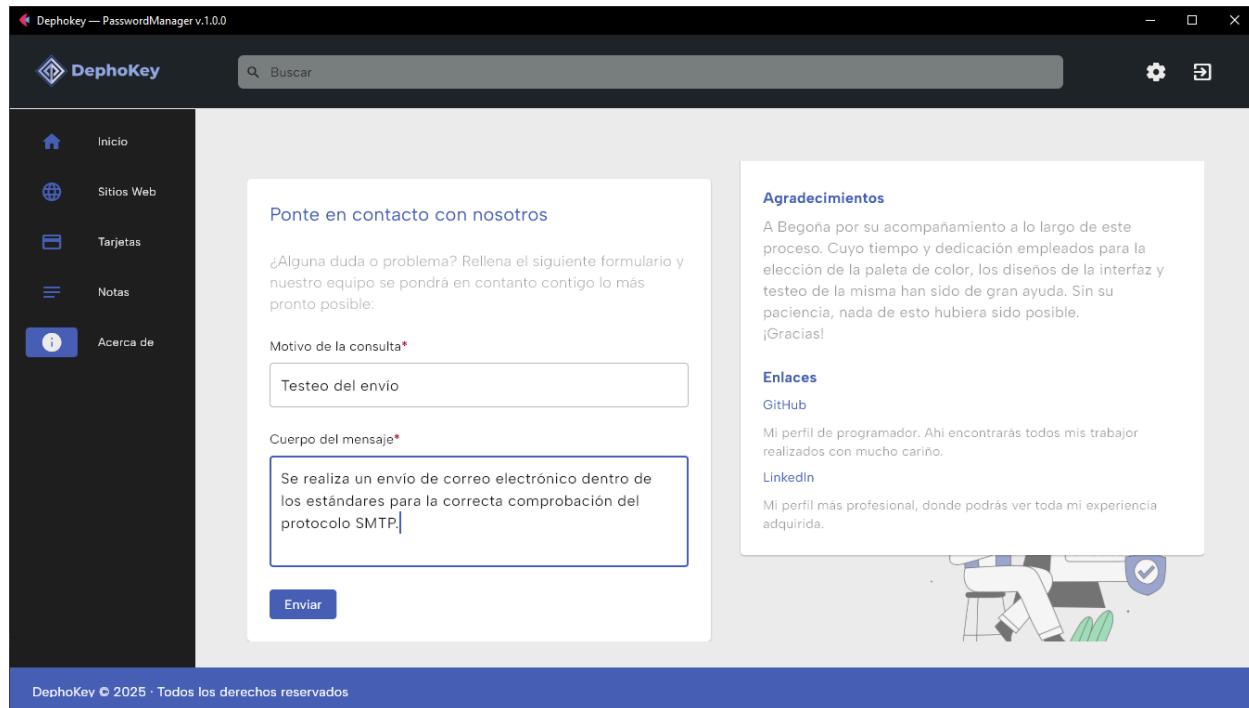
The screenshot shows a database table with 12 rows of note data and a modal dialog for deletion confirmation. The table columns are labeled with truncated column names. The modal dialog has fields for 'Nombre' (Name) and 'Apellido' (Last Name), with 'Apellido' currently set to 'Último'. The modal also includes a 'Cancelar' (Cancel) button and a 'Borrar' (Delete) button.

	Y66H0JQV1E059v...	WU7NLT7HVL6jLwAV	VVUE	VAAAAMM0000000000...	666J-UU-47	2025-03-24 18:54:23.654556
6	Y66H0JQV1E059v...	WU7NLT7HVL6jLwAV	VVUE	VAAAAMM0000000000...	666J-UU-47	2025-03-24 18:54:23.654556
7	v12K3rnsewgnhgk	bt9br4h0f2j15x0	Nisi	gAAAAABn4ZxPefTQIS41NCX_EUt0-fymc1T2...	2025-03-24 18:54:23.654556	
8	ggdfqetbyj6t2k5	bt9br4h0f2j15x0	Sed	gAAAAABn4ZxPxHusBRU0FFejRR2syb6007H...	2025-03-24 18:54:23.654556	
9	yv61uqisqy4q2y	bt9br4h0f2j15x0	Pariatur	gAAAAABn4ZxRkszImcz5wcQwU6TUud6H8FQ...	2025-03-24 18:54:23.655557	
10	rt1z2n6ji18bo1	bt9br4h0f2j15x0	Beatae	gAAAAABn4ZxPBal_FTRZw2YegT51DQEwD01...	2025-03-24 18:54:23.655557	
11	2z49ee0f9dt8xnz	tbqn18qwlebmcdc	Testeo de nota 01	gAAAAABn4qgrlzNdhjQMo5GHIC29-8okpI...	2025-03-25 13:57:15.417614	
12	3d514ml7u5fnm4i	tbqn18qwlebmcdc	Testeo de nota 03	gAAAAABn4qhacMEexQmsPiWHW18J6frQOXq...	2025-03-25 13:58:02.976869	



Envío de correo electrónico

Para enviar un correo electrónico el usuario puede navegar a la página de **acerca de** y llenar el formulario. Una vez rellene los campos (y siempre y cuando tenga **Docker** funcionando en segundo plano), puede enviar el mensaje.



Si abrimos el navegador web y nos dirigimos a **localhost:1080**, podemos encontrar la interfaz de **MailCatcher**, una aplicación que intercepta los correos enviados.

Allí podemos encontrar tanto el correo enviado con todos los datos como los correos enviados para la restauración de la contraseña.

Ponte en contacto con nosotros

¿Alguna duda o problema? Rellena el siguiente formulario y nuestro equipo se pondrá en contacto contigo lo más pronto posible:

Motivo de la consulta*

Cuerpo del mensaje*

¡Gracias por tus comentarios!
Trabajaremos lo más rápido posible para hacerte llegar una respuesta.

Agradecimientos

A Begoña por su acompañamiento a lo largo de este proceso. Cuyo tiempo y dedicación empleados para la elección de la paleta de color, los diseños de la interfaz y testeo de la misma han sido de gran ayuda. Sin su paciencia, nada de esto hubiera sido posible.
¡Gracias!

Enlaces

[GitHub](#)
Mi perfil de programador. Allí encontrarás todos mis trabajos realizados con mucho cariño.

[LinkedIn](#)
Mi perfil más profesional, donde podrás ver toda mi experiencia adquirida.

DephoKey © 2025 · Todos los derechos reservados

MailCatcher

From: <usuario@gmail.com> size=9192 To: <dephokey.team@gmail.com> Subject: Testeo del envío Received: Tuesday, 25 Mar 2025 3:55:11 PM

Received: Tuesday, 25 Mar 2025 3:55:11 PM
From: <usuario@gmail.com> size=9192
To: <dephokey.team@gmail.com>
Subject: Testeo del envío
Attachments: logtype.png

HTML Plain Text Source Download

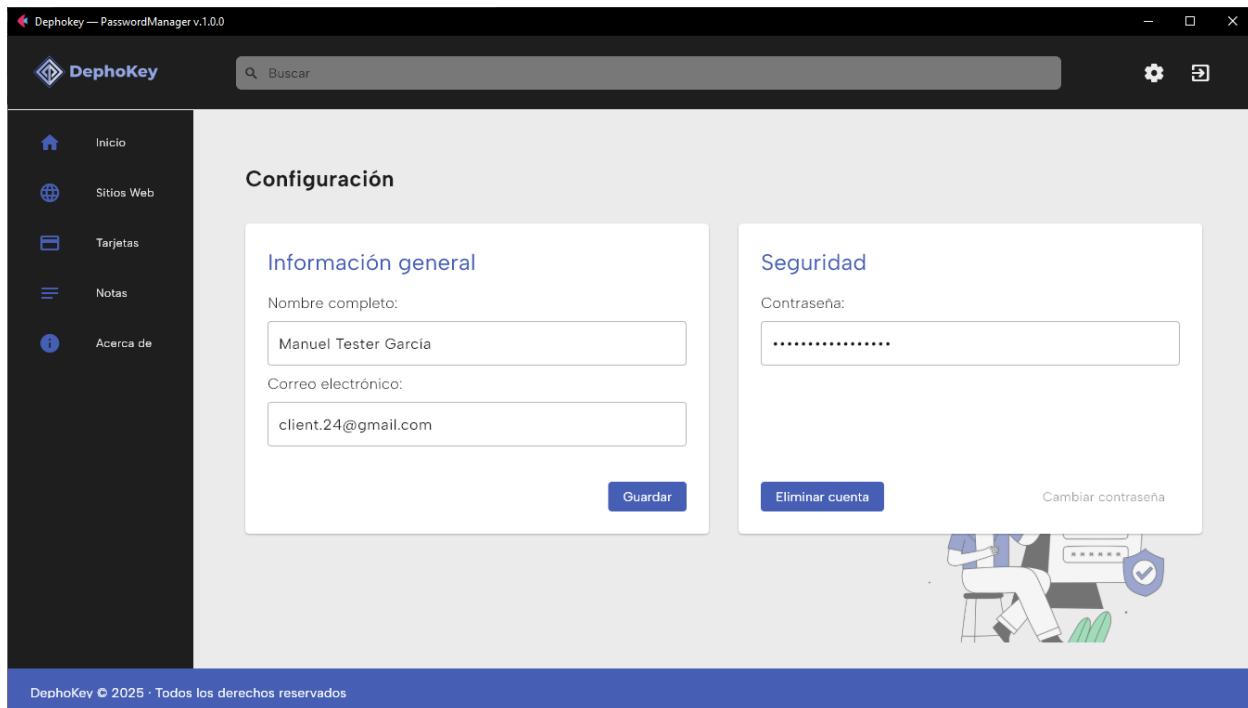
Testeo del envío

Se realiza un envío de correo electrónico dentro de los estándares para la correcta comprobación del protocolo SMTP.

DephoKey © 2025 · Todos los derechos reservados

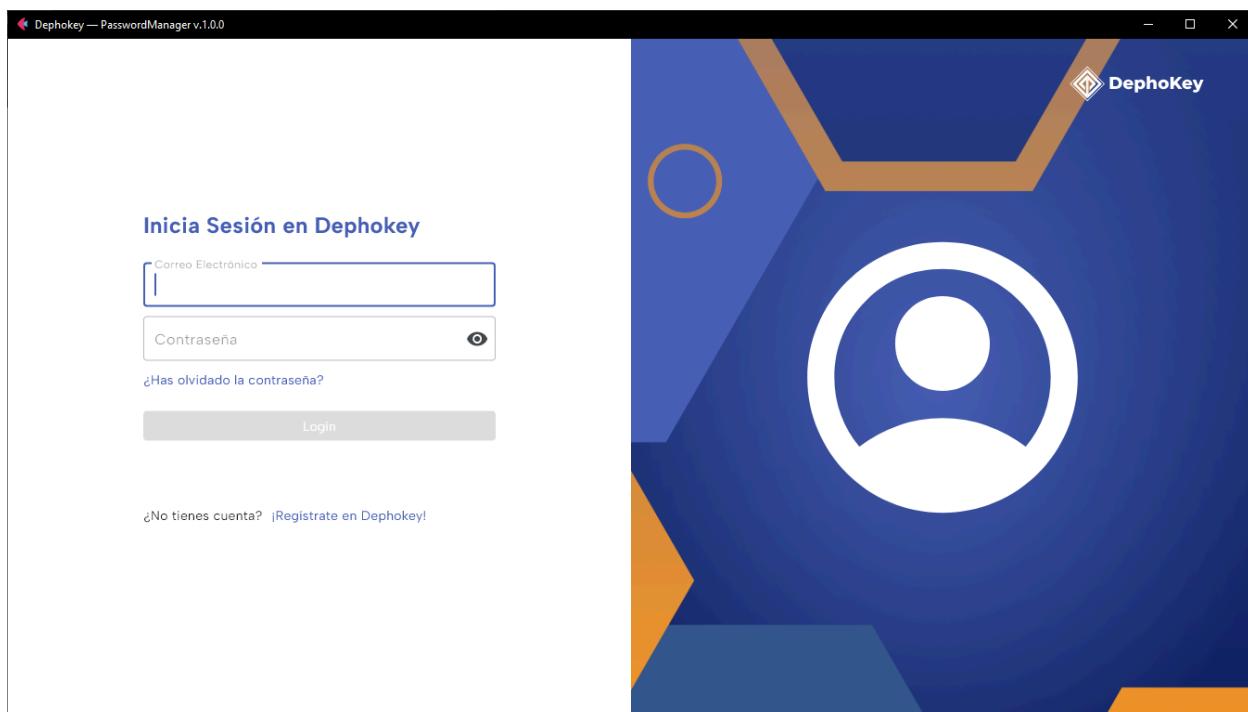
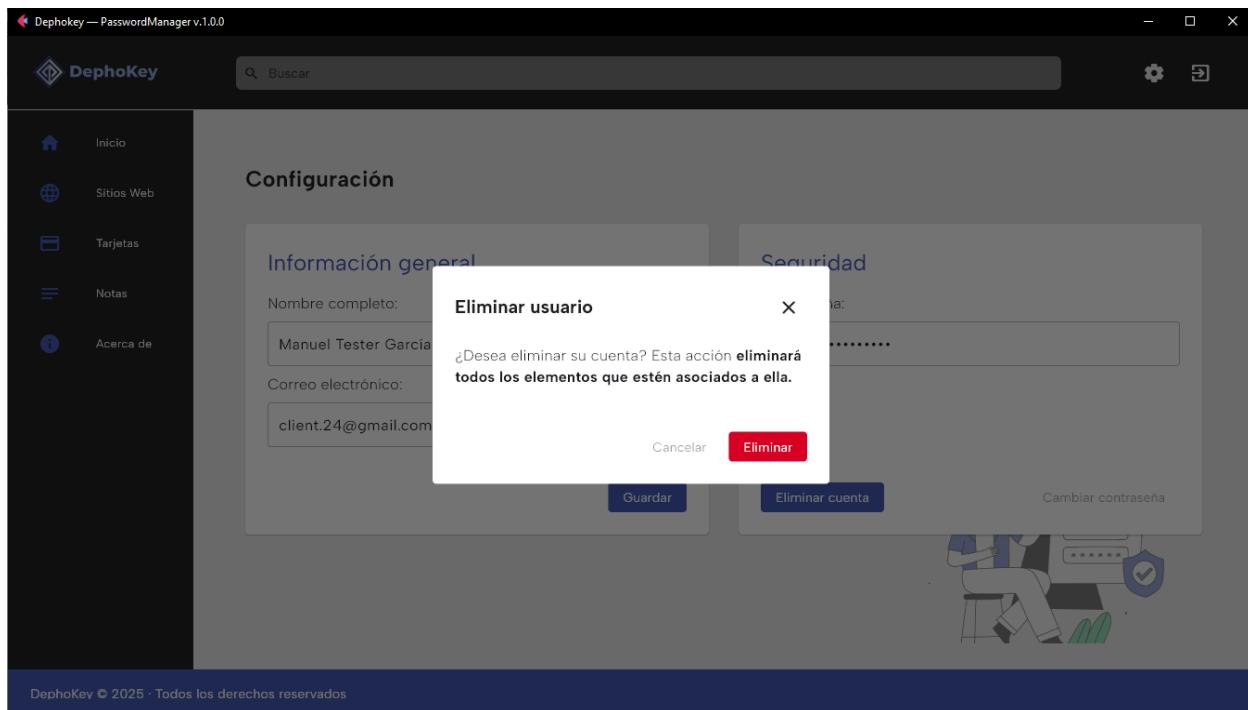
Eliminación de la cuenta

Para la eliminación de la cuenta, el usuario puede dirigirse a la ventana de opciones de la aplicación, que se encuentra pulsando el segundo ícono en la barra superior. Una vez cambie el contenido de la página y en el apartado **Seguridad**, el usuario encuentra un botón que dice **Eliminar cuenta**.



Este botón abre una ventana emergente que pregunta al mismo si desea eliminar la cuenta con todo lo que ello conlleva.

Una vez acepte la acción, la aplicación elimina la sesión actual, todos los datos asociados a la cuenta y cierra la conexión con la base de datos, redirigiendo al usuario a la página de inicio. El testeo ha sido realizado con la cuenta de ejemplo `client.24@gmail.com` que almacena diferentes tipos de datos.



Ahora podemos comprobar que la base de datos se ha actualizado correctamente y ha eliminado todos los datos asociados a la cuenta de `client.24@gmail.com`.

Tabla 'User':

id	role	fullname	email	hashed_password	created	updated
1 w7f7mohf4ovl5by	ADMIN	Sergio Administrador Muñoz	admin.24@gmail.com	60fe74406e7f353ed975f350f2fbb6a2e86...	2025-03-24 14:45:11	2025-03-24 14:45:11
2 tbqni8qwlebmcdc	CLIENT	Usuario De Ejemplo	usuario@gmail.com	8a82a5ff6fc722db8a57e25b508f5a008ac6...	2025-03-25 10:58:02	2025-03-25 10:58:02

Tabla 'Site':

id	user_id	name	address	username	encrypted_password
1 95cc9bzlimlxcs5	tbqni8qwlebmcdc	Amazon	http://www.amazon.com/	usuario@gmail.com	gAAAAABn4r9bknaid9d92Dm...

Tabla 'Creditcard':

id	user_id	cardholder	encrypted_number	encrypted_cvv
1 uvd0yhbbgllku4	tbqni8qwlebmcdc	Usuario De Ejemplo	gAAAAABn4r7cyU2HWLy3uBef7fejHIutCUL...	gAAAAABn4r7cuhyDNtLeXQ...

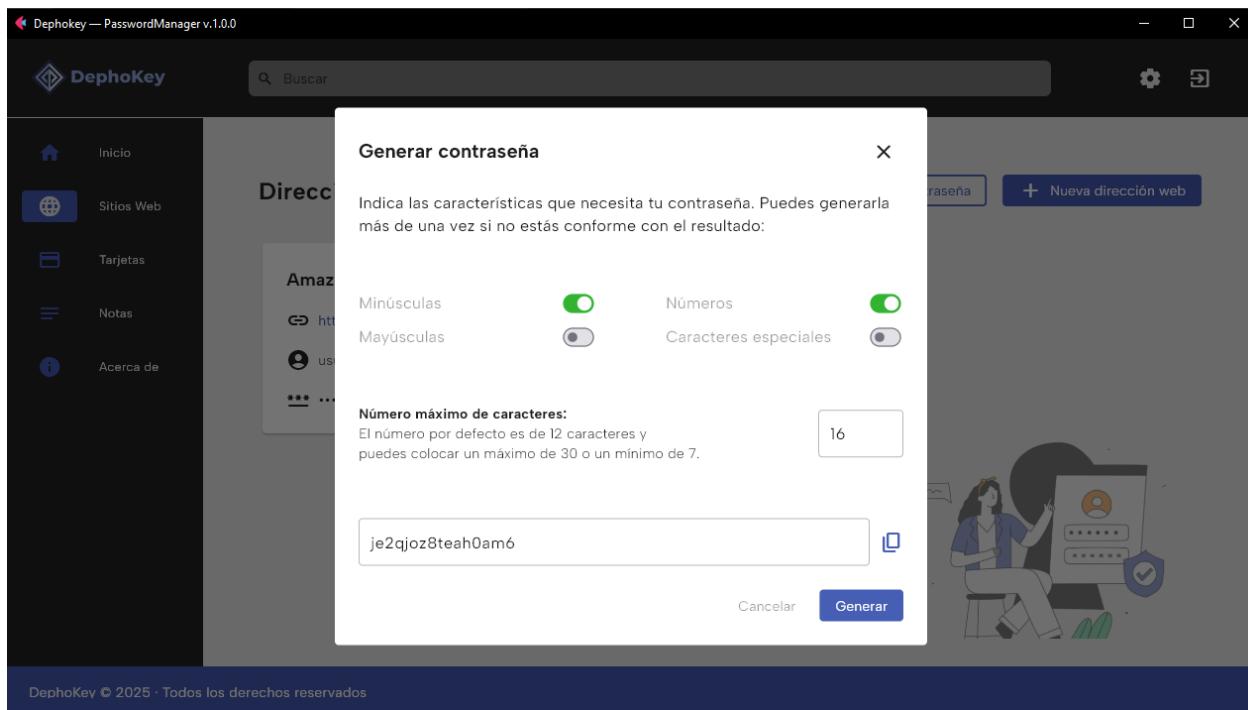
Tabla 'Note':

id	user_id	title	encrypted_content	created
1 z249es0f9dt8xnxz	tbqni8qwlebmcdc	Testeo de nota 01	gAAAAABn4gr1zNdhjQM0SGH1IC29-8okpT...	2025-03-25 13:57:15.417614
2 3d514ml7u5fnm4i	tbqni8qwlebmcdc	Testeo de nota 03	gAAAAABn4qhaMEExQmsPIWHW16J6frQOXq...	2025-03-25 13:58:02.976869

Generadores

Los generadores que hemos introducido son herramientas que puede utilizar el usuario para ayudar a completar diferentes campos. De momento se han implementado dos pero no se cierran las puertas a añadir más herramientas en un futuro.

El generador de contraseñas:

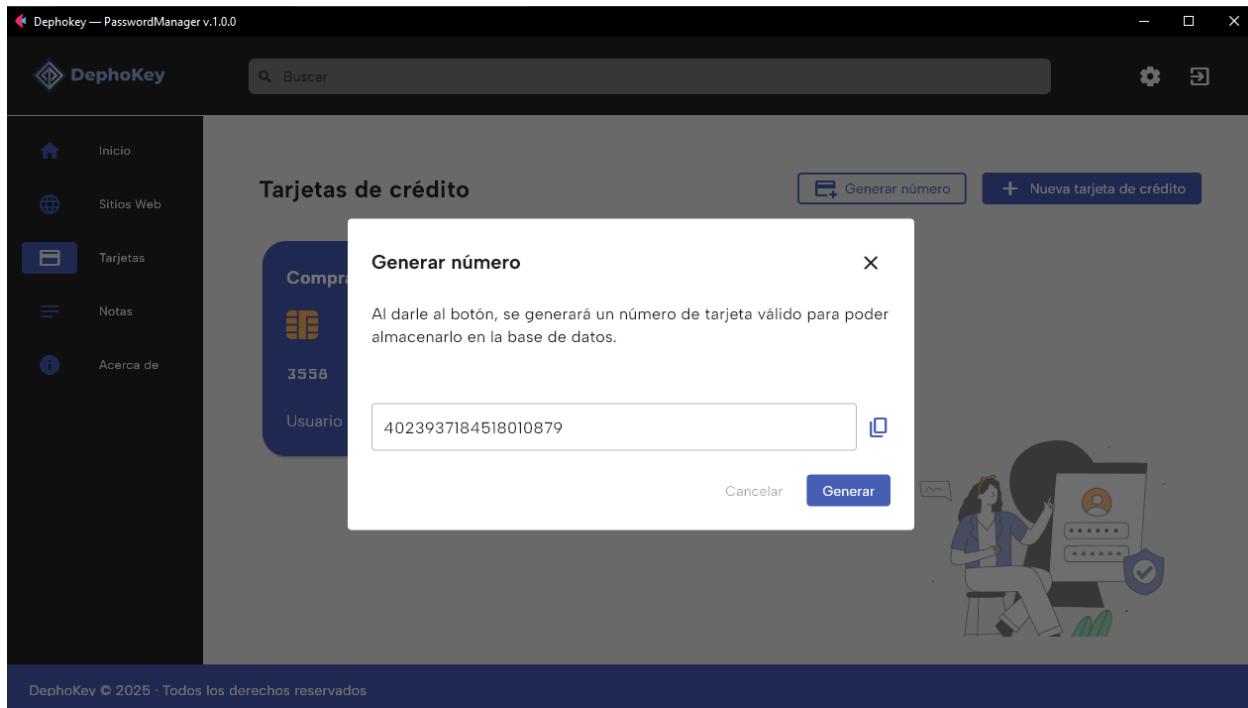


El generador permite al usuario generar contraseñas aleatorias dentro de unos parámetros elegidos por el mismo, ya sea añadiendo letras en minúscula, mayúscula, números o caracteres especiales y puede configurar la longitud de las mismas.

Con tan solo pulsar el botón **Generar**, una nueva contraseña aparecerá en el campo de texto sustituyendo la anterior si el resultado no le satisface. Cuando encuentre una de su agrado,

puede pulsar el botón de copiado y automáticamente se añadirá dicha contraseña a su portapapeles.

El generador de números de tarjeta de crédito:



Este generador permite al usuario generar números de tarjeta de crédito válidos que hayan pasado el algoritmo de Luhn, creando números de tarjeta (no reales) que puede introducir al tratar de guardar una nueva en la base de datos.

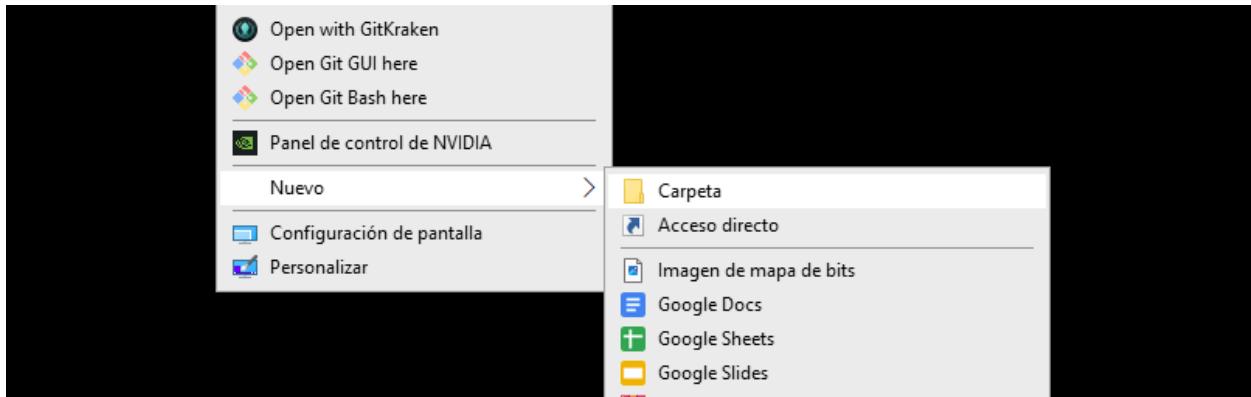
Esta herramienta permite ayudar al usuario a añadir datos falsos en la aplicación, ya que por su seguridad, es una aplicación que no cumple con los [estándares de seguridad PCI DSS*](#) y no se recomienda almacenar datos verídicos.

Manual de instalación

Antes de empezar con la instalación debemos tener en cuenta que necesitamos tener instalados **Python v3.11.5** o superior, **Git v2.45.1** o superior y, de forma opcional, la última versión de **Docker v28.0.1** (en nuestro caso). Una vez realizadas dichas comprobaciones, cabe añadir que todo el tutorial presente se encuentra realizado desde el *PowerShell* de Windows.

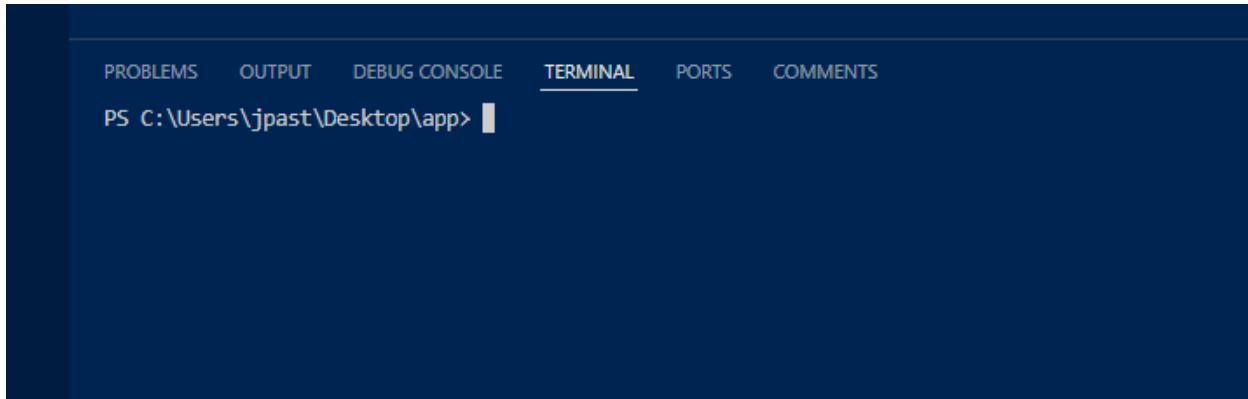
Para la instalación y testeo de la aplicación deberemos seguir los siguientes pasos:

- **Crear la carpeta principal:** Crea una carpeta donde quieras almacenar todos los archivos necesarios para el correcto funcionamiento de la aplicación, en nuestro caso la llamaremos **app**:



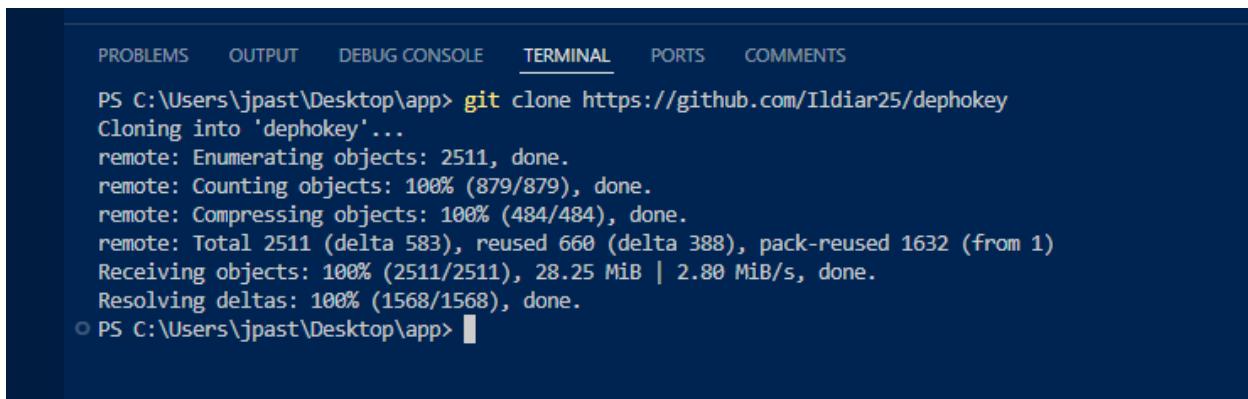
- **Seleccionar IDE:** Abrimos nuestro *IDE* favorito y abrimos la carpeta que hemos creado. Comprobamos en el terminal que nos encontramos dentro de dicha carpeta. Si no es

así, nos moveremos mediante los comandos `cd <folder>` y `cd..` (adelante y atrás, respectivamente):



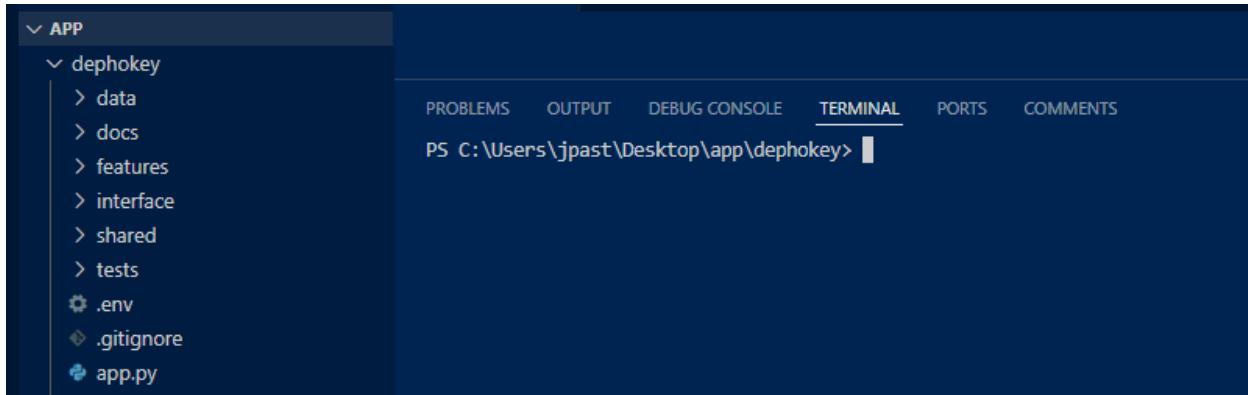
A screenshot of a terminal window. At the top, there are tabs labeled PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), PORTS, and COMMENTS. Below the tabs, the command prompt is shown as "PS C:\Users\jpast\Desktop\app>". The rest of the window is blank.

- **Clonamos el repositorio:** En la propia terminal, introducimos el comando `git clone https://github.com/Ildiar25/dephokey`. Esta función descarga una copia del repositorio en la carpeta donde nos encontramos.



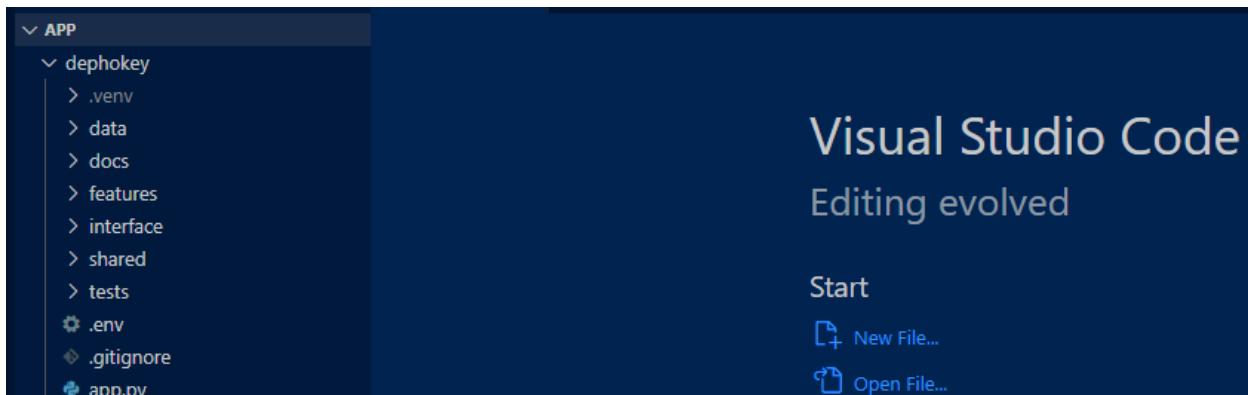
A screenshot of a terminal window. At the top, there are tabs labeled PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), PORTS, and COMMENTS. Below the tabs, the command prompt is shown as "PS C:\Users\jpast\Desktop\app>". The terminal then displays the output of the `git clone` command:
git clone https://github.com/Ildiar25/dephokey
Cloning into 'dephokey'...
remote: Enumerating objects: 2511, done.
remote: Counting objects: 100% (879/879), done.
remote: Compressing objects: 100% (484/484), done.
remote: Total 2511 (delta 583), reused 660 (delta 388), pack-reused 1632 (from 1)
Receiving objects: 100% (2511/2511), 28.25 MiB | 2.80 MiB/s, done.
Resolving deltas: 100% (1568/1568), done.
PS C:\Users\jpast\Desktop\app>

- **Accedemos a la carpeta *dephokey*:** una vez terminada la descarga, veremos que se ha creado una carpeta llamada ***dephokey***, mediante los comandos anteriores deberemos acceder a ella y podremos ver su contenido en el lateral de nuestro *IDE*.



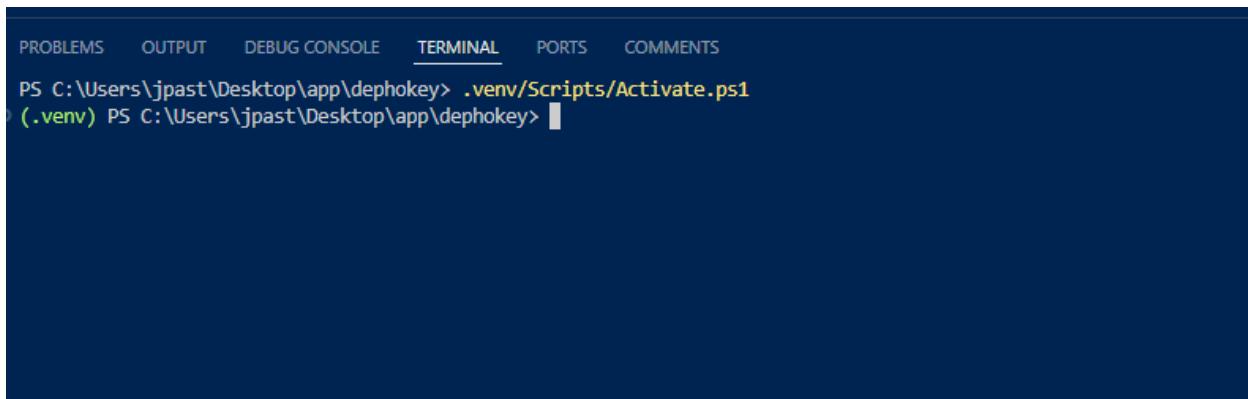
¡Debemos estar atentos de que nuestro terminal termine en *dephokey*, porque ahora debemos trabajar en ese directorio y los siguientes pasos pueden dar problemas!

- **Entorno virtual:** Para no interferir con las librerías que ya se tengan instaladas, deberemos crear un entorno virtual para nuestra aplicación. Para ello, utilizaremos el comando `python -m venv .venv`:



Esta acción habrá creado una carpeta llamada **.venv** (el punto es para indicarle al sistema operativo que la carpeta está oculta). Sin embargo ahora debemos activar dicho entorno.

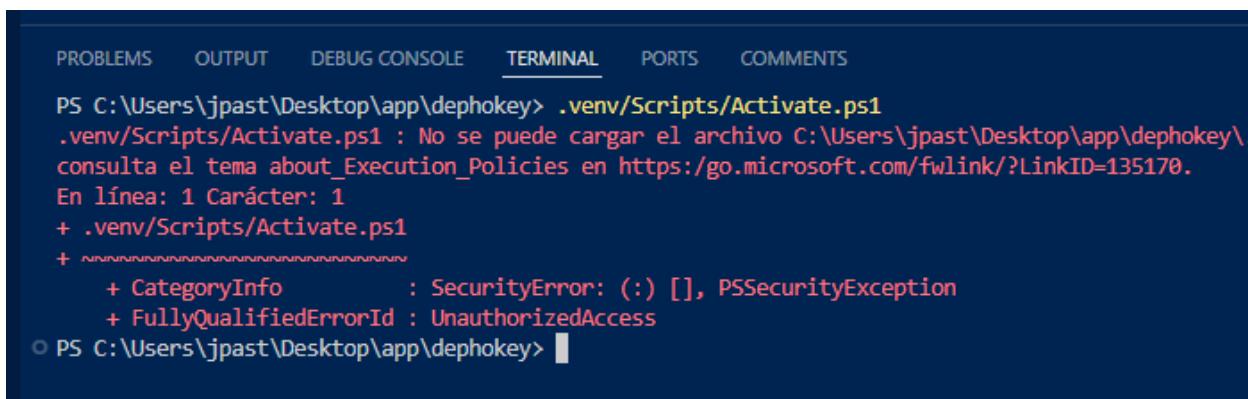
- **Activación del entorno:** Con tan solo introducir `.venv\Scripts\Activate.ps1` debería de activarse. Lo podemos saber si delante de nuestra ruta de acceso encontramos `(venv)`.



A screenshot of a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS  
PS C:\Users\jpast\Desktop\app\dephokey> .venv\Scripts\Activate.ps1  
> (.venv) PS C:\Users\jpast\Desktop\app\dephokey>
```

ATENCIÓN: *Este paso puede dar el siguiente error:*



A screenshot of a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS  
PS C:\Users\jpast\Desktop\app\dephokey> .venv\Scripts\Activate.ps1  
.venv\Scripts\Activate.ps1 : No se puede cargar el archivo C:\Users\jpast\Desktop\app\dephokey\.consultas el tema about_Execution_Policies en https://go.microsoft.com/fwlink/?LinkID=135170.  
En línea: 1 Carácter: 1  
+ .venv\Scripts\Activate.ps1  
+ ~~~~~  
    + CategoryInfo          : SecurityError: () [], PSSecurityException  
    + FullyQualifiedErrorId : UnauthorizedAccess  
○ PS C:\Users\jpast\Desktop\app\dephokey>
```

Esto significa que no se tienen permisos suficientes para poder realizar la ejecución del *script*. Es por ello que la solución en este caso sería cerrar el programa, abrir en modo administrador el *PowerShell* de *Windows* e introducir el siguiente comando:

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Unrestricted
```

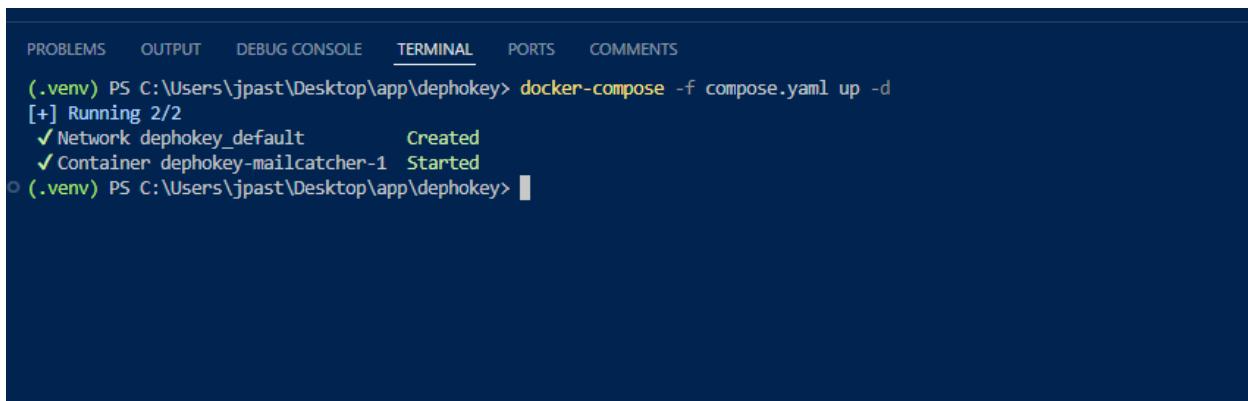
Cuando nos pregunte, simplemente respondemos yes y ya podremos abrir el programa y realizar el paso anterior.

- **Instalación de las dependencias:** Este paso es importante, pues se encargará de leer nuestro archivo ***requirements.txt*** y de instalar las librerías necesarias para el funcionamiento de nuestra app. Introducimos **pip install -r requirements.txt** y esperamos a que finalice.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
Using cached anyio-4.8.0-py3-none-any.whl (96 kB)
Using cached idna-3.10-py3-none-any.whl (70 kB)
Using cached certifi-2025.1.31-py3-none-any.whl (166 kB)
Using cached pycparser-2.22-py3-none-any.whl (117 kB)
Using cached h11-0.14.0-py3-none-any.whl (58 kB)
Using cached sniffio-1.3.1-py3-none-any.whl (10 kB)
Installing collected packages: nanoid, typing-extensions, sniffio, six, python-dotenv, pycparser, oauthlib, MarkupSafe, idna, h11, gree
er, cryptography, flet
Successfully installed Faker-35.1.0 Jinja2-3.1.6 MarkupSafe-3.0.2 SQLAlchemy-2.0.39 anyio-4.8.0 certifi-2025.1.31 cffi-1.17.1 cryptogra
-2.0.0 oauthlib-3.2.2 pycparser-2.22 python-dateutil-2.9.0.post0 python-dotenv-1.0.1 repath-0.9.0 six-1.17.0 sniffio-1.3.1 typing-exten
[notice] A new release of pip is available: 23.2.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
↳ (.venv) PS C:\Users\jpast\Desktop\app\dephokey>
```

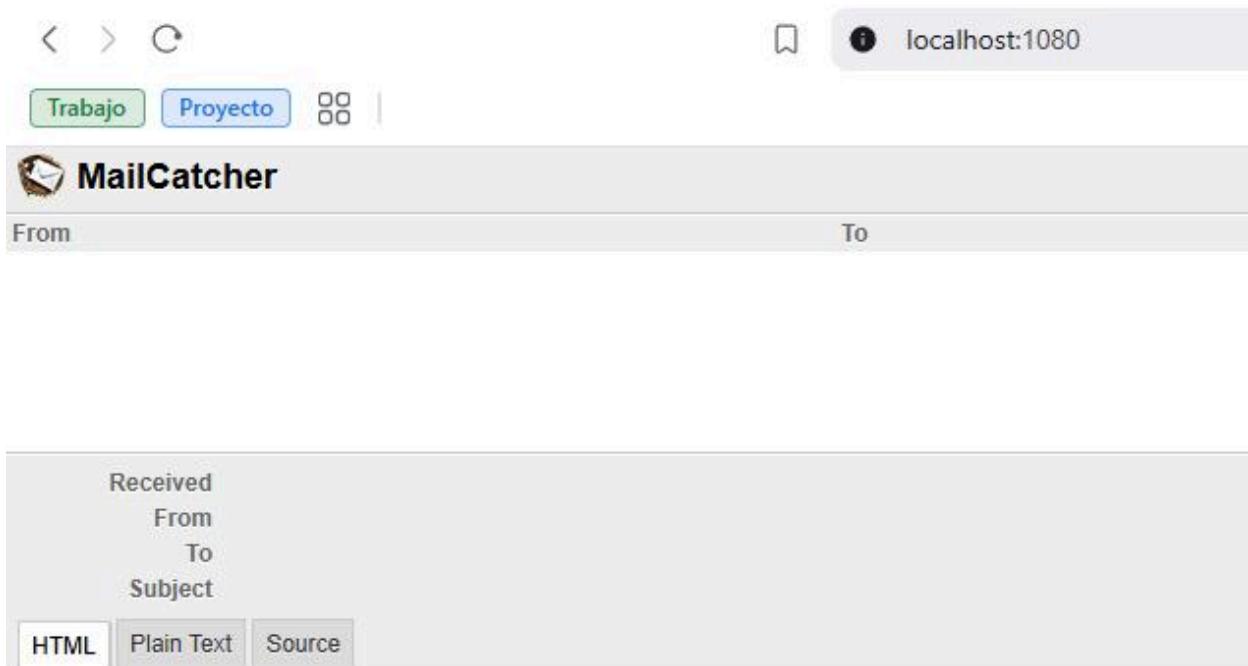
- **Preparar el servidor email:** Este paso no es crucial, pues nuestra aplicación puede funcionar perfectamente sin poder enviar correos electrónicos. Simplemente no se podrían enviar ni los tokens de recuperación de cuenta ni contactar con nuestro equipo.

Sin embargo, vamos a activarlo escribiendo `docker-compose -f compose.yaml up -d` en la terminal:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
(.venv) PS C:\Users\jpast\Desktop\app\dephokey> docker-compose -f compose.yaml up -d
[+] Running 2/2
  ✓ Network dephokey_default          Created
  ✓ Container dephokey-mailcatcher-1   Started
○ (.venv) PS C:\Users\jpast\Desktop\app\dephokey>
```

Esta acción abrirá los puertos **1025** (donde se enviarán los correos electrónicos) y **1080** (donde se cargará la interfaz de *MailCatcher*, una aplicación para capturar los correos enviados por la aplicación). Puedes acceder a ella escribiendo `localhost:1080` en tu navegador



- **Realizar el testeo de los servicios:** Este paso no es obligatorio, pero sí recomendable. Permite saber si la aplicación tiene errores antes de abrirla, por tanto hay que ejecutar `python -m unittest discover tests`. Esto realizará una serie de tests predeterminados que comprobarán el funcionamiento de cada elemento.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
WHERE test_table.name = ?
2025-03-13 20:17:19,155 INFO sqlalchemy.engine.Engine [generated in 0.00049s] ('Test01',)
2025-03-13 20:17:19,155 INFO sqlalchemy.engine.Engine ROLLBACK
[20:17:19] |   WARNING | Module: [C:\Users\jpast\Desktop\app\dephokey\tests\logs\test_results.log] | Esto es un mensaje en modo ERROR
[20:17:19] | CRITICAL | Module: [C:\Users\jpast\Desktop\app\dephokey\tests\logs\test_results.log] | Esto es un mensaje en modo CRITICA
[20:17:19] |   WARNING | Module: [C:\Users\jpast\Desktop\app\dephokey\tests\logs\app.log] | Esto es un mensaje en modo WARNING
[20:17:19] |   ERROR | Module: [C:\Users\jpast\Desktop\app\dephokey\tests\logs\app.log] | Esto es un mensaje en modo ERROR
[20:17:19] | CRITICAL | Module: [C:\Users\jpast\Desktop\app\dephokey\tests\logs\app.log] | Esto es un mensaje en modo CRITICAL
-----
Ran 112 tests in 0.446s
OK
○ (.venv) PS C:\Users\jpast\Desktop\app\dephokey>

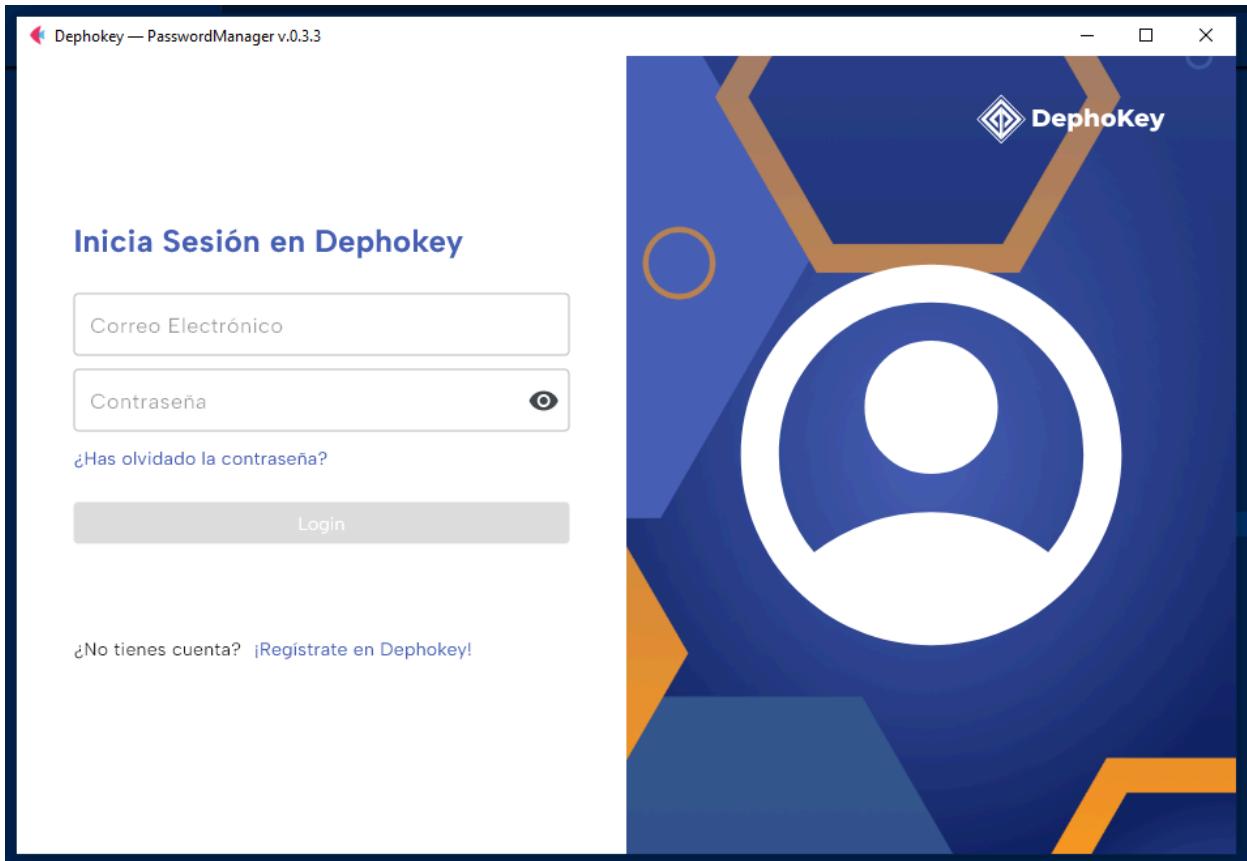
```

- **Abrir la aplicación:** Y finalmente, para abrir la aplicación tan solo deberemos ingresar el comando `flet run app.py`. Realizará unas comprobaciones (puede demorar un rato) y se abrirá la página principal de la aplicación:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
(.venv) PS C:\Users\jpast\Desktop\app\dephokey> flet run app.py
Installing flet-cli 0.25.2 package...OK
Installing flet-desktop 0.25.2 package...OK
[ ]

```



- **Cerrar el servidor:** Una vez terminemos de utilizar la aplicación tan solo quedará cerrar el servidor. Para ello, introducimos `docker-compose -f compose.yaml down` en el terminal y listo. ¡Gracias por utilizar Dephokey — PasswordManager!

A screenshot of a terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), PORTS, and COMMENTS. The terminal shows the command `docker-compose -f compose.yaml down` being run in a virtual environment named ".venv" located at `C:\Users\jpast\Desktop\app\dephokey`. The output indicates that 2/2 containers are running and are being removed, specifically the `dephokey-mailcatcher-1` container and the `dephokey_default` network.

Conclusiones

Conclusiones generales del proyecto, aprendizajes y experiencias adquiridas gracias a Dephokey ya que, tras todos estos meses tratando de cumplir con los objetivos, puedo afirmar que estoy orgulloso del producto final.

La aplicación cumple con todos y cada uno de los requisitos planteados al principio del mismo ya que se ha mantenido organizado correctamente; dispone de una amplia programación orientada a objetos; el manejo de la base de datos funciona como es debido; se mantiene una encriptación correcta de los elementos de la misma y se almacenan las credenciales principales de forma segura; el envío de los correos electrónicos es funcional (aunque optativo, pues no interfiere con el funcionamiento general) y el acabado a nivel visual mantiene la misma estética en todos sus ámbitos.

Los mayores obstáculos que se presentaron fueron los errores de importación circular. Fue un quebradero de cabeza orquestado por la implementación del *ORM* que, al tratar de tipar como es debido los atributos de las clases, generaban dicho fallo. Al final pude solucionar el problema separando la forma en que se crea la clase principal mediante su archivo de inicialización.

Este viaje me ha sido muy útil para aprender cómo funcionan las jerarquías de una aplicación, pudiendo entender desde dentro, el funcionamiento y comunicación de los diferentes elementos, ya sean páginas, botones o la simple visualización de la interfaz. También amplió mi enfoque al momento de tratar de implementar la ciberseguridad, haciéndome entender que no todo se puede dar por sentado.

Por último, debo añadir que lo que más he disfrutado ha sido el planificar todas las funcionalidades (aunque al final, resulta que habían muchas más que no había tenido en cuenta), aprender nuevos conceptos como son el testeo de cualquier elemento programado, el cómo se comunican las herencias, en cómo funciona de verdad la programación orientada a objetos y, sobre todo, en la investigación y documentación de cada elemento.

Evolutivos del proyecto

En esta sección encontramos los puntos a mejorar o realizar en nuestra aplicación, ordenados desde sus capas más inferiores hasta el exterior.

- **Creación del modelo Login:** Implementar una clase base de tipo **Login** que se encargue de verificar si el usuario está registrado, permita el inicio de sesión y la opción de *remember me* que ayude recordar el usuario en el equipo. A partir de ella, heredará otras clases con diferentes métodos de inicio de sesión, como **Google**, **Facebook** o **Github**. Lo importante será implementar el uso de las *API*.
- **Creación del modelo Password:** Este modelo se encarga de la creación de una clase tipo **Password** cuya funcionalidad es verificar el tipo de contraseña. Podrá asignarle un nivel de seguridad según los criterios especificados (baja, media o alta), saber si la contraseña se ha utilizado en diferentes lugares (permite el control de su repetición) y deberá tener una fecha de creación para poder avisar al usuario al sobrepasar la fecha límite (es hora de cambiar la contraseña por su seguridad).
- **Mejoras en la clase User:** Se debe añadir una fecha de **último inicio de sesión**; esto ayudará a verificar si la cuenta está en uso y si rebasa el plazo máximo se podrá eliminar sin la acción del usuario. Incluir un **límite de intentos** de inicio de sesión; permite bloquear la cuenta y delimitar su uso sólo al administrador. Añadir un atributo que almacene el **tipo de login** empleado; así el programa sabe qué credenciales debe utilizar. Implementar un atributo que se encargue de **almacenar la clave** de encriptación del usuario; esta acción permite separar los elementos agregados del usuario de los demás clientes. Y por último, implementar la **verificación de correo** electrónico; esta funcionalidad permite saber si el correo es válido y si pertenece al usuario que lo ha proporcionado.

- **Mejoras de la clase Site:** Añadir **fecha de modificación**; permitirá ordenar los elementos del usuario por fecha de modificación y no por creación.
- **Mejoras de la clase CreditCard:** Añadir **fecha de modificación**; permitirá ordenar los elementos del usuario por fecha de modificación y no por creación. Incluir el **tipo de tarjeta**; así el usuario puede decidir si la tarjeta es de débito o de crédito.
- **Mejoras de la clase Note:** Añadir **fecha de modificación**; permitirá ordenar los elementos del usuario por fecha de modificación y no por creación.
- **Mejoras en la clase PasswordRequest:** Implementar un atributo que se encargue de almacenar la **fecha de expiración** del token; así tan sólo será necesario realizar una comparación.
- **Mejoras de la Base de Datos:** Que la aplicación almacene la base de datos en las **carpetas del propio sistema** utilizando los permisos adecuados. Crear roles con diferentes limitaciones para cuando se manipule la base de datos.
- **Mejoras en la seguridad:** Crear una **clave maestra** que se almacene en el *keyring* del sistema; esta funcionalidad permite encriptar cada clave de usuario (que se utiliza en la encriptación y desencriptación de sus elementos), añadiendo una capa extra de seguridad. **Verificar** la existencia del usuario durante la navegación; permite cerrar la página principal si la sesión no se encuentra disponible. **Reducir el tiempo** de expiración de la sesión; al detectar la inactividad del usuario en la aplicación, esta debería cerrar la sesión automáticamente. **Petición de contraseña** al realizar cambios graves; cuando el

usuario necesite cambiar expresamente la contraseña de la cuenta o decida eliminarla, se debe exigir la confirmación de cambios en el sistema mediante el ingreso de la contraseña principal.

- **Mejoras en la interfaz gráfica:** Que el administrador pueda **cambiar fechas** de creación y/o modificación según convenga. **Visualizar el nivel** de seguridad de las contraseñas introducidas por el usuario en tiempo real. Enviar un **correo de bienvenida** para verificar el correo del usuario. Añadir un **registro de eventos** al correo enviado por el usuario en el caso de que el usuario decida ponerse en contacto con los responsables de la aplicación; el registro será único por cada acceso a la aplicación. Que se puedan **visualizar los datos sensibles** de toda la aplicación con un solo botón.
- **Implementación de diferentes temas:** Actualizar la página de configuración de la aplicación dejando que el usuario pueda **cambiar el tema** de la aplicación a su gusto, mejorando la experiencia de usuario. Junto con el tema también se puede implementar el **cambio de idioma** de la aplicación, ya que actualmente disfrutamos de la internacionalización.

Y hasta aquí quedarían las posibles mejoras de la aplicación. Después de todo, todos estos cambios quedarían registrados en una nueva versión de la misma y ya sólo haría falta actualizarla.

Bibliografía

1. Inesdi Business TechSchool, 2024. *Arquitectura de software: 5 patrones principales* [online]. INESDI. Disponible en: [Inesdi | Web Development | Arquitectura de software: definición y tipos principales](#) [28-Noviembre-2024]
2. Carlos Hernando, 2018. *Estructurar un proyecto Python* [online]. Problema, Solución, Herramienta. Disponible en: [CHernando | Blog | Cómo estructurar tu proyecto](#) [28-Noviembre-2024]
3. Xukyo, 2024. *Buenas prácticas para crear un proyecto Python* [online]. AranaCorp. Disponible en: [AranaCorp | Tutoriales | Buenas Prácticas para crear un proyecto Python](#) [28-Noviembre-2024]
4. Míriam Martínez, 2020. *¿Qué es la Programación Orientada a Objetos?* [online]. Profile Software Services. Disponible en: [Profile | Blog | ¿Qué es la Programación Orientada a Objetos?](#) [28-Noviembre-2024]
5. EDTeam, 2022. *¿Qué es un ORM?* [online]. EDTeam. Disponible en: [EDTeam | Blog | ¿Qué es un ORM?](#) [03-Diciembre-2024]
6. Carlos Vecina, 2023. *¿Qué es SQLAlchemy?* [online]. TypeThePipe. Disponible en: [TypeThePipe | Python | Posts | ¿Qué es SQLAlchemy? Prueba la v2.0 para acceder a bases de datos desde Python](#) [03-Diciembre-2024]
7. Rajiv Chandra, 2023. *Python, Importación Circular: Métodos para evitarla* [online]. Kanaries. Disponible en: [Kanaries | Topics | Python | Importación circular: Métodos para evitarla](#) [23-Marzo-2025]
8. Andrew Hughes & Rich Keith, 2024. *Encryption vs. Hashing vs. Salting - What's the difference?* [online]. Ping Identity. Disponible en: [Ping Identity | Blog | Encryption vs. Hashing vs. Salting - What's the Difference?](#) [10-Enero-2025]
9. Matthew Kosinski, 2024. *¿Qué es la autenticación?* [online]. IBM. Disponible en: [IBM | Think | Temas | Autenticación | ¿Qué es la autenticación?](#) [10-Enero-2025]
10. Ronald Rivest, 2023. *¿Qué es el hash de contraseñas?* [online]. Dashlane. Disponible en: [Dashlane | Blog | ¿Qué es el hash de contraseñas?](#) [10-Enero-2025]

11. Nonick, 2004. *Criptografía Simétrica* [online]. Wikipedia. Disponible en: [Wikipedia | Criptografía Simétrica](#) [10-Enero-2025]
12. Cloudflare, 2025. *¿Qué es el correo electrónico? | Definición de correo electrónico* [online]. Cloudflare. Disponible en: [Cloudflare | Centro de Aprendizaje | Más información sobre la seguridad del correo electrónico | ¿Qué es la seguridad del correo electrónico? | ¿Qué es el correo electrónico?](#) [12-Marzo-2025]
13. Webmaestro, 2001. *Simple Mail Transfer Protocol* [online]. Wikipedia. Disponible en: [Wikipedia | Simple Mail Transfer Protocol](#) [12-Marzo-2025]
14. Antxon Pous, 2020. *Distintos métodos para incluir imágenes en tus campañas de email marketing* [online]. Newsletter Soft. Disponible en: [Newsletter Soft | Blog | Distintos métodos para incluir imágenes en tus campañas de email marketing](#) [12-Marzo-2025]
15. Salman Ravoof, 2022. *Guía para principiantes sobre cómo crear y enviar un correo HTML* [online]. Kinsta. Disponible en: [Kinsta | Blog | Guía para principiantes sobre cómo crear y enviar un correo HTML](#) [28-Febrero-2025]
16. Sil Zubikarai, 2022. *Validación de datos: Cómo verificar la entrada del usuario en formularios HTML con código JavaScript de ejemplo* [online]. FreeCodeCamp. Disponible en: [FreeCodeCamp | News | Validación de datos: Cómo verificar la entrada del usuario en formularios HTML con código JavaScript de ejemplo](#) [27-Febrero-2025]
17. Speedy González, 2003. *Expresión Regular* [online]. Wikipedia. Disponible en: [Wikipedia | Expresión Regular](#) [27-Febrero-2025]
18. Melog Ithamalfonso, 2015. *Número de tarjeta bancaria* [online]. Wikipedia. Disponible en: [Wikipedia | Número de Tarjeta Bancaria](#) [12-Diciembre-2024]
19. Quertymith, 2012. *Algoritmo de Luhn* [online]. Wikipedia. Disponible en: [Wikipedia | Algoritmo de Luhn](#) [12-Diciembre-2024]
20. Takuya Murata, 2003. *Test-driven Development* [online]. Wikipedia. Disponible en: [Wikipedia | Test-Driven Development](#) [27-Febrero-2025]
21. Alberto Casero, 2024. *Las 3 leyes de TDD de Robert C. Martin* [online]. KeepCoding. Disponible en: [KeepCoding | Blog | Las 3 leyes de TDD de Robert C. Martin](#) [27-Febrero-2025]

22. Tuatara Agencia Boutique, 2023. *Lo que debes saber sobre la interfaz gráfica (GUI)* [online]. Tuatara Agencia Boutique. Disponible en: [Tuatara | Blog | Software | Interfaz Gráfica | Lo que debes saber sobre la Interfaz Gráfica \(GUI\)](#) [24-Marzo-2025]
23. Pablo Londoño, 2023. *Qué es un formulario, para qué sirve y características* [online]. Hubspot. Disponible en: [Hubspot | Inicio | Website | Qué es un formulario, para qué sirve y características](#) [24-Marzo-2025]
24. Mariusz Bal, 2011. *What is a widget* [online]. Localo. Disponible en: [Localo | Resources | Dictionary | Widget](#) [24-Marzo-2025]
25. César Salza, 2021. *¿Qué es un cuadro de diálogo?* [online]. El Grupo Informático. Disponible en: [El Grupo Informático | Tutoriales | Curiosidades | ¿Qué es un cuadro de diálogo?](#) [24-Marzo-2025]
26. Coursera Staff, 2025. *SQL vs. NoSQL: The differences explained & when to use each* [online]. Coursera. Disponible en: [Coursera | Data | Data Science | SQL vs. NoSQL: The differences explained & when to use each](#) [28-Febrero-2025]
27. UNIR Formación Profesional, 2022. *Framework: Qué es, para qué sirve y algunos ejemplos* [online]. UNIR FP. Disponible en: [UNIR FP | Revista | Ingeniería y Tecnología | Framework: Qué es, para qué sirve y algunos ejemplos](#) [28-Febrero-2025]
28. Cristina Ortega, 2023. *Modelo de datos: Qué es, tipos, técnicas y mejores prácticas* [online]. QuestionPro. Disponible en: [QuestionPro | Inicio | Investigación de mercado | Modelo de datos: Qué es, tipos, técnicas y mejores prácticas](#) [12-Marzo-2025]
29. Stripe Inc, 2025. *Guía para el cumplimiento de la normativa PCI DSS* [online]. Stripe Inc. Disponible en: [Stripe | Guías | Recursos para Productos | Guía para el cumplimiento de la normativa PCI DSS](#) [25-Marzo-2025]

Recursos

1. Illustration Kit, s.f. *Haloo Illustrations* [imágenes]. [Haloo Illustrations](#)
2. Chart DB, s.f. *Database Diagramming* [utilidades]. [Database Diagramming](#)
3. UNICODE Symbols, s.f. *Símbolos Unicode* [utilidades]. [Símbolos Unicode](#)
4. Reg EXR, s.f. *Regular Expressions* [utilidades]. [Regular Expressions](#)
5. Draw IO, s.f. *Online Diagram Software* [utilidades]. [Online Diagram Software](#)
6. Live Agent, s.f. *Plain Text Email Templates* [herramientas]. [Plain Text Email Templates](#)
7. Mail Meteor, s.f. *HTML Email Templates* [herramientas]. [Email Templates](#)
8. Super Designer, s.f. *Backgrounds* [utilidades]. [Create Backgrounds](#)
9. Nano ID, s.f. *NanoID Collision Calculator* [herramientas]. [Collision Calculator](#)
10. Gemini, s.f. *Consulta de la lógica en programación* [herramientas]. [Gemini 2.0](#)
11. Google Fonts, s.f. *Familias tipográficas Albert Sans y Roboto* [utilidades]. [Google Fonts](#)
12. Google Material Symbols, s.f. *Material Symbols* [utilidades]. [Google Material Symbols](#)
13. Freepik, s.f. *Iconografía* [utilidades]. [Freepik](#)
14. Pixabay, s.f. *Sound FX by Universfield* [utilidades]. [Pixabay Sound Effects](#)
15. Flet, s.f. *Buscador de iconos* [herramientas]. [Flet Icon Browser](#)

Diagrama de flujo (páginas principales)

En esta sección se puede observar la navegación que realiza el usuario por diferentes páginas. Esta sección se considera la capa superior de la aplicación puesto que el flujo presentado no permite el acceso a las capas inferiores sin las credenciales necesarias.

Una vez se realiza el inicio de sesión el flujo se bifurca teniendo en cuenta el rol establecido del usuario que ingresa.

Punto de inicio

Una vez confirmada la dirección de correo, la página se refresca, mostrando una cuenta atrás en tiempo real que monitoriza la caducidad del token.



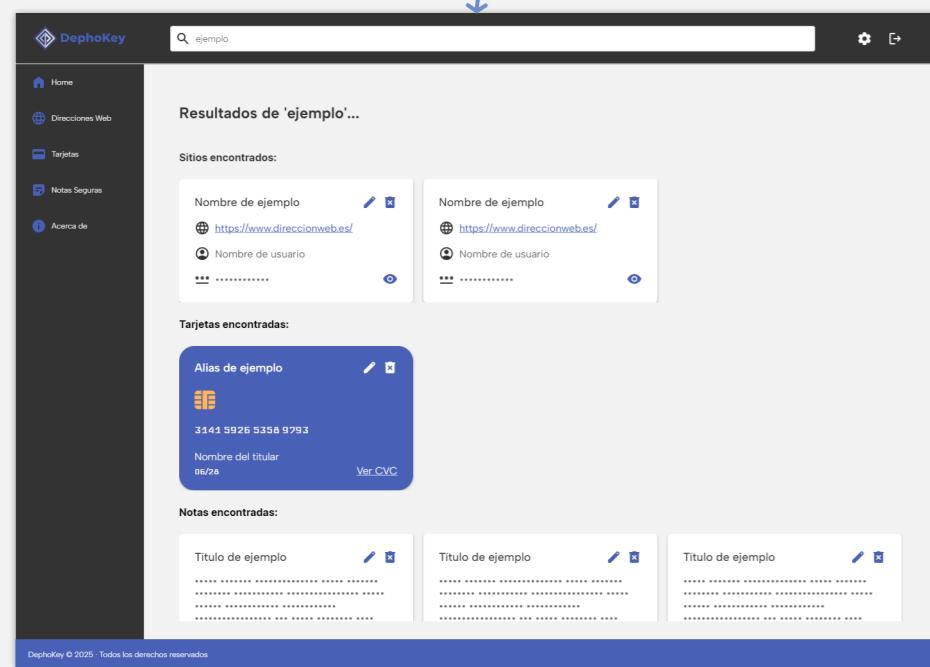
La página de administración no dispone de la barra de navegación lateral ya que no la necesita para ver los elementos almacenados en la base de datos.

Diagrama de flujo (páginas secundarias)

En la siguiente sección se puede observar la navegación que realiza el usuario a través de la barra lateral. Esta sección se considera la capa intermedia de la aplicación puesto que el flujo presentado no permite la eliminación de otros usuarios ni elementos que no estén asociados al mismo.

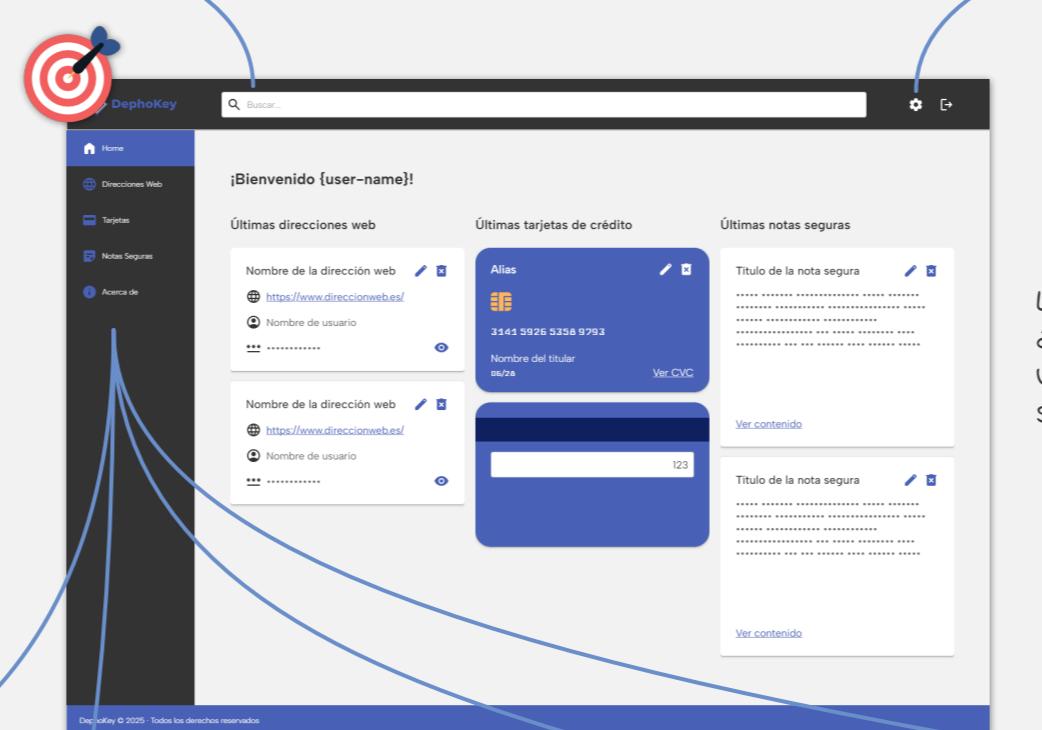
Una vez se realiza el inicio de sesión el usuario puede navegar entre los elementos asociados a su ID.

Punto de inicio

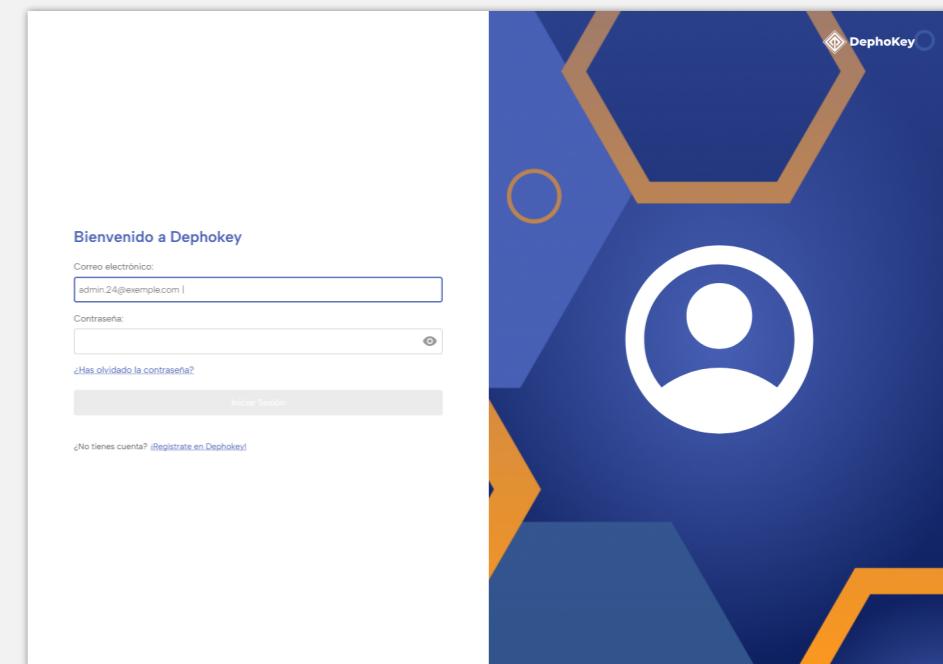


Los resultados de la barra de búsqueda se muestran en una página diferente.

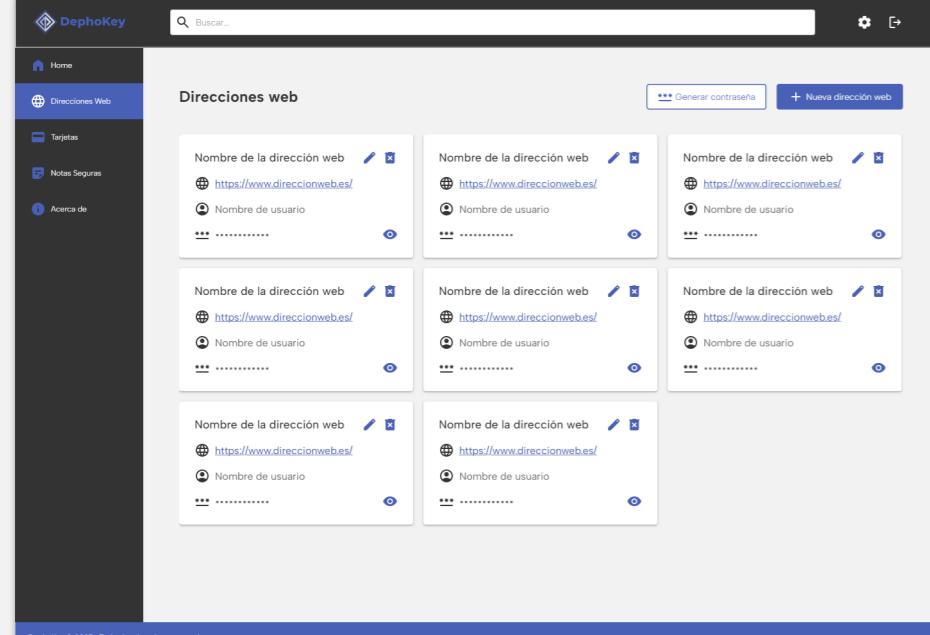
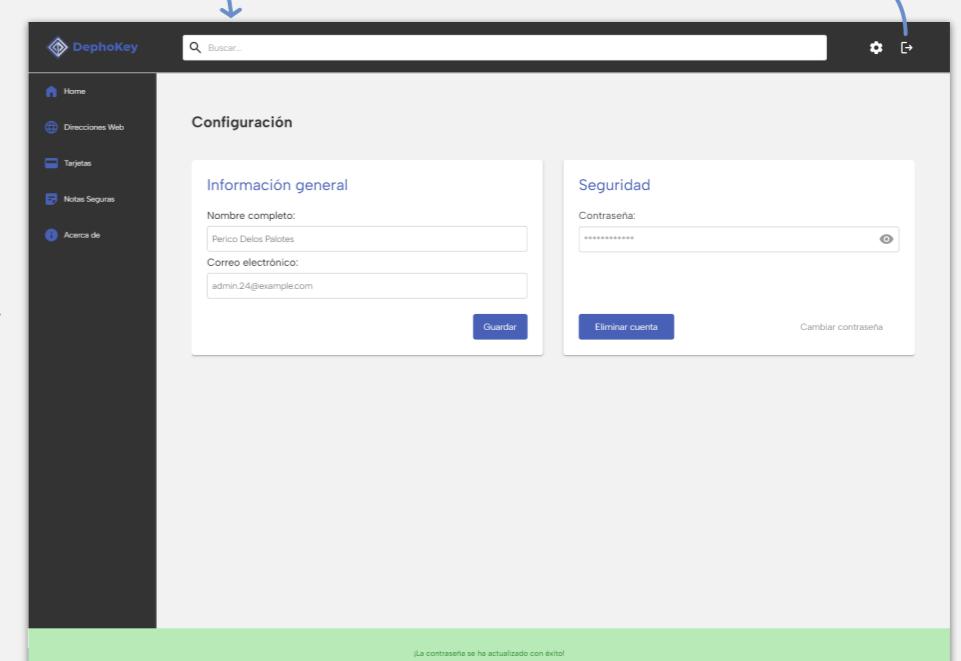
Sus resultados provienen de buscar palabras clave en atributos específicos de los elementos.



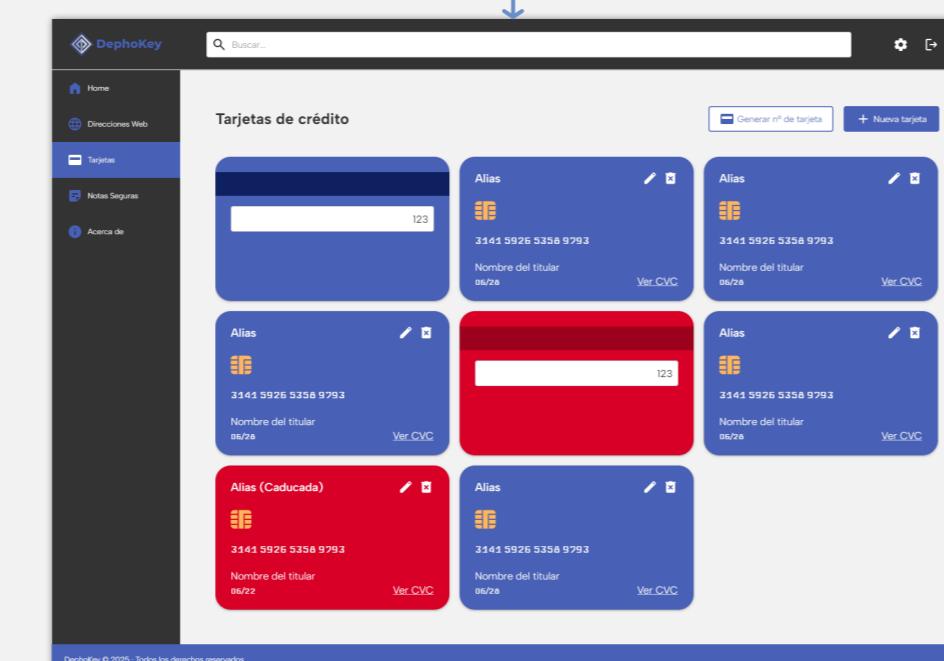
La página de inicio de la aplicación muestra los últimos elementos añadidos a la base de datos.



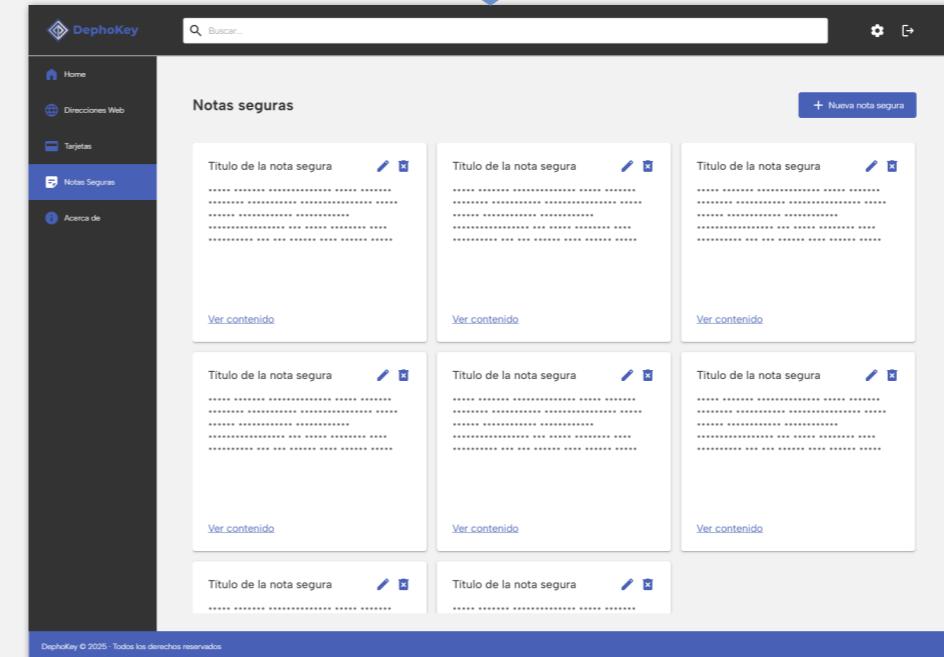
De vuelta a la página de inicio de sesión una vez el usuario la cierre.



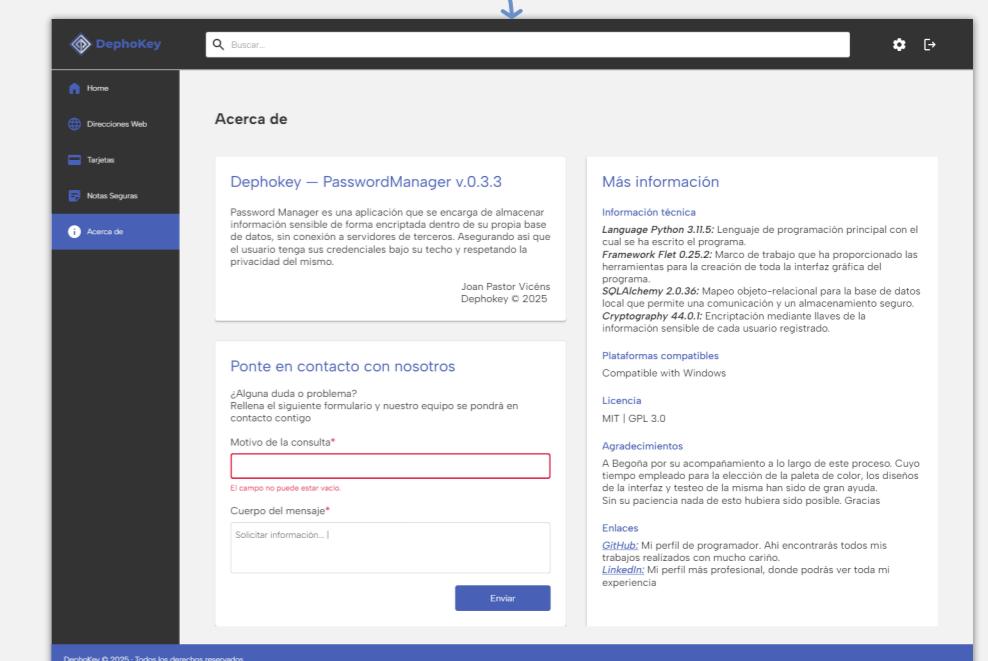
Sitios se encarga de cargar los widgets que muestran la información de cada página web agregada por el usuario.



Tarjetas muestra en pantalla todos y cada uno de los widgets que el usuario ha creado al añadir tarjetas. El color del mismo se actualiza según la caducidad de la tarjeta



Notas muestra en pantalla los widgets pertenecientes a la clase Note.



Acerca de muestra información de la aplicación además de incluir un pequeño formulario para ponerte en contacto con el equipo de la aplicación.

Diagrama de flujo (formularios principales)

En esta sección podemos analizar la apertura de formularios que realiza el usuario cuando pulsa los botones principales de cada página. Esto permite la agregación de elementos a la base de datos al llenar dichos formularios.

También permite modificar los datos del usuario o la eliminación de la propia cuenta.

The screenshot shows the 'Direcciones web' (Web Addresses) section of the application. It lists several saved web addresses with their URLs and user names. A modal window titled 'Generar contraseña' (Generate Password) is open, allowing the user to specify character types (Mayúsculas, Minúsculas, Números, Caracteres especiales) and password length (Número mínimo de caracteres, Número máximo de caracteres). A generated password ('tpq2mU8Zz3U') is displayed in the input field.

Con el formulario de creación de contraseñas, el usuario puede elegir de qué elementos se forma cada contraseña.

The screenshot shows the 'Notas seguras' (Secure Notes) section. It displays a grid of notes with titles like 'Título de la nota segura'. A detailed view of a note is shown on the right, containing fields for 'Título de la nota segura', 'Contenido', and 'Ver contenido' (View content).

The screenshot shows the 'Nueva nota segura' (New Secure Note) modal. It has fields for 'Título (opcional)' (Optional title) and 'Contenido' (Content), with a large text area for the note's content.

Para la creación de las notas, el campo de texto que puede llenar el usuario es lo suficientemente amplio para poder escribir con soltura.

The screenshot shows the 'Direcciones web' section again. A modal window titled 'Nueva dirección web' (New Web Address) is open, prompting the user to enter a name for the web, its URL, the user's name, and a password.

Para agregar una nueva dirección web tan solo es necesario llenar el formulario con los datos correspondientes.

The screenshot shows the 'Tarjetas de crédito' (Credit Cards) section. It lists several credit cards with their last four digits and cardholder names. A modal window titled 'Generar número' (Generate Number) is open, showing a generated card number ('4907034455020445').

La generación de números de tarjeta válidos no es necesaria pero sí valiosa para el testeo correcto de la aplicación.

The screenshot shows the 'Configuración' (Configuration) section. It includes 'Información general' (General Information) with fields for 'Nombre completo' (Full Name) and 'Correo electrónico' (Email), and a 'Seguridad' (Security) section with a password field and a 'Cambiar contraseña' (Change Password) button.

The screenshot shows the 'Generar Contraseña' (Generate Password) modal, which allows the user to enter a new password and confirm it.

Apoyándose en la reutilización del diseño se puede emplear el mismo formulario para diferentes fines como la actualización de la contraseña por diferentes motivos.

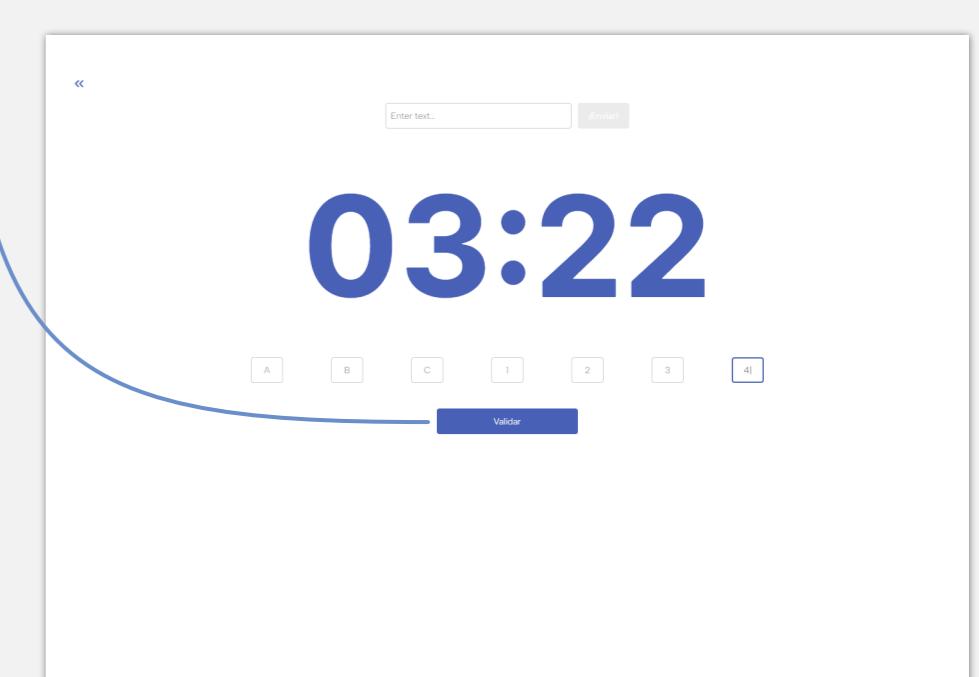
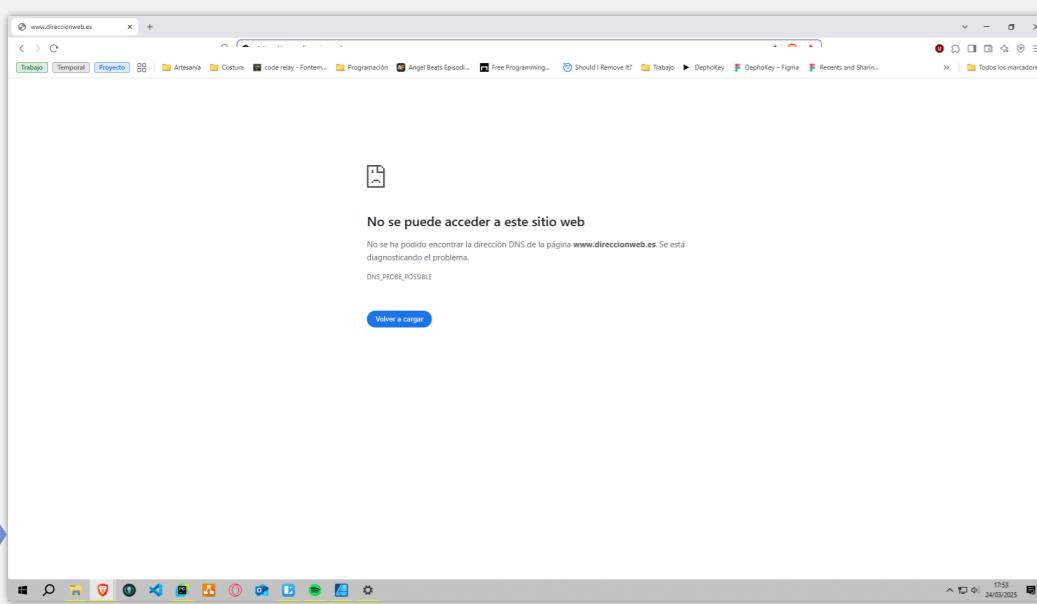


Diagrama de flujo (widgets)

Funcionamiento y apertura de formularios desde los propios widgets creados. En esta sección se detalla el comportamiento de los mismos y cómo se comunican.

También se incluyen comportamientos externos a la aplicación como la navegación directa a páginas web.



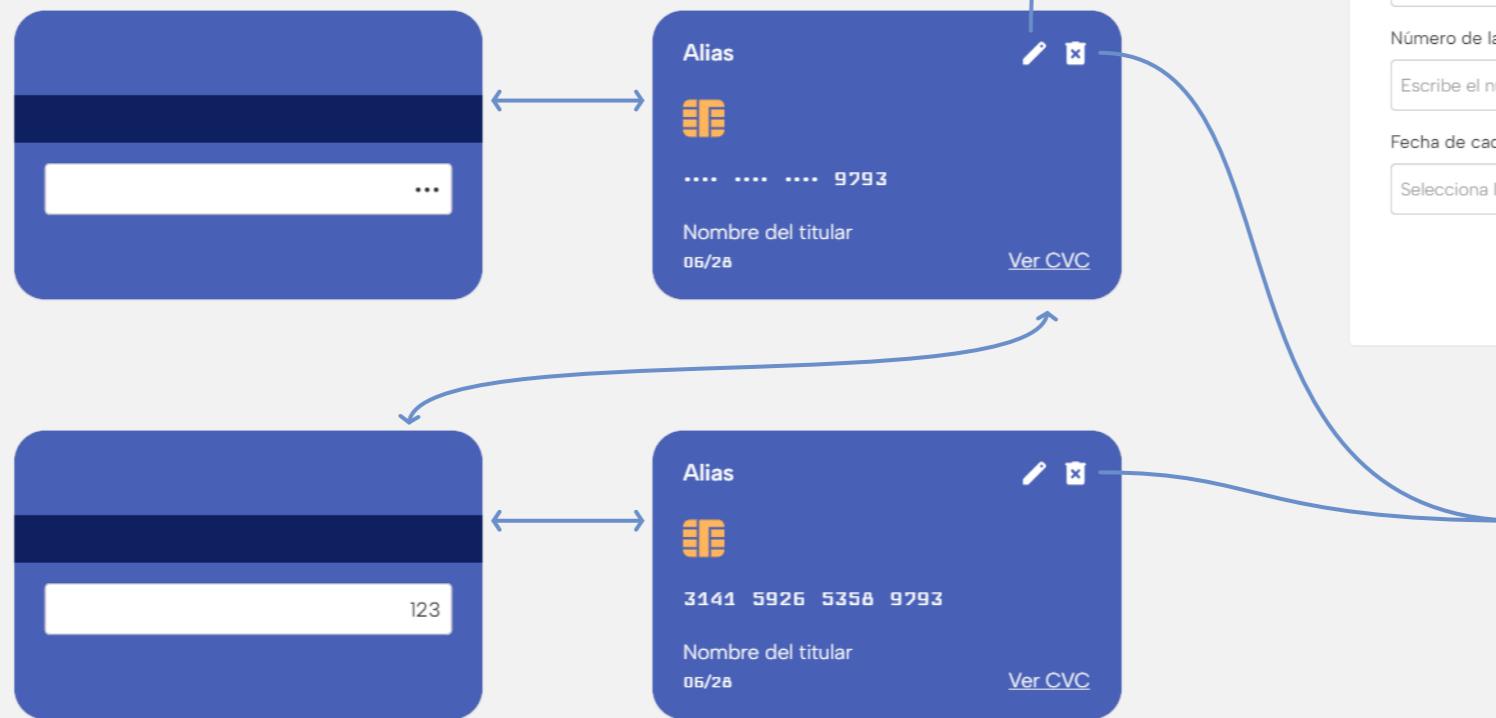
El widget diseñado para mostrar los datos dispone de un lanzador que al pulsar sobre la dirección web, ésta abre el navegador y viaja directamente hasta allí.

Nuevo widget

Nombre de la dirección web
https://www.direccionweb.es/
Nombre de usuario
***
Contraseña
xCPY6cXDzZK

Al pulsar sobre el icono con forma ocular, el widget se actualizará mostrando el contenido enmascarado.

El comportamiento del widget de las tarjetas de crédito recae en que pueda voltearse para ver los datos situados en la parte de atrás.



En cambio, para ver la información enmascarada tan sólo hará falta pasar el cursor por encima.

Nueva dirección web

Nombre (opcional)
Escribe un nombre para la web
Dirección web
Escribe la dirección web
Nombre de usuario
Escribe el nombre de tu usuario
Contraseña
Escribe tu contraseña

Eliminar {item}

¿Estás seguro de querer eliminar este registro?

Para editar la información asociada al widget sólo habrá que pulsar el botón del lápiz situado en la parte superior derecha del mismo. Esta acción abrirá el formulario de edición.

El widget para las notas está basado en el diseño de una nota adhesiva. Simplemente se le añade el título y el texto enmascarado.

Si se desea ver el contenido de la misma, tan solo deberá pulsar el botón 'ver contenido' situado en la parte inferior izquierda.

Nueva tarjeta de crédito

Alias (opcional)
Escribe un alias para la tarjeta
Nombre del titular (opcional)
Escribe el nombre del titular
Número de la tarjeta
Escribe el número de la tarjeta
Fecha de caducidad
Selecciona la fecha de expiración CVC
Introduce el CVC

Eliminar {item}

¿Estás seguro de querer eliminar este registro?

El propio widget tiene acceso a los formularios de edición y eliminación del ítem que muestra. Esta opción facilita al usuario entender qué está haciendo.

Nueva nota segura

Titulo (opcional)
Escribe un título para la nota
Contenido
Escribe el contenido de la nota

Eliminar {item}

¿Estás seguro de querer eliminar este registro?

Este widget también facilita la edición y eliminación del mismo mediante los botones situados en la parte superior derecha.

Tablas (5)

Nombre	Tipo	Esquema
creditcard		<pre>CREATE TABLE creditcard (id VARCHAR(15) NOT NULL, user_id VARCHAR(15) NOT NULL, cardholder VARCHAR(250) NOT NULL, encrypted_number VARCHAR(4100) NOT NULL, encrypted_cvc VARCHAR(4100) NOT NULL, valid_until DATETIME NOT NULL, expired BOOLEAN NOT NULL, alias VARCHAR(250), created DATETIME NOT NULL, PRIMARY KEY (id), FOREIGN KEY(user_id) REFERENCES user(id) ON DELETE CASCADE)</pre>
id	VARCHAR(15)	"id" VARCHAR(15) NOT NULL
user_id	VARCHAR(15)	"user_id" VARCHAR(15) NOT NULL
cardholder	VARCHAR(250)	"cardholder" VARCHAR(250) NOT NULL
encrypted_number	VARCHAR(4100)	"encrypted_number" VARCHAR(4100) NOT NULL
encrypted_cvc	VARCHAR(4100)	"encrypted_cvc" VARCHAR(4100) NOT NULL
valid_until	DATETIME	"valid_until" DATETIME NOT NULL
expired	BOOLEAN	"expired" BOOLEAN NOT NULL
alias	VARCHAR(250)	"alias" VARCHAR(250)
created	DATETIME	"created" DATETIME NOT NULL
note		<pre>CREATE TABLE note (id VARCHAR(15) NOT NULL, user_id VARCHAR(15) NOT NULL, title VARCHAR(250), encrypted_content VARCHAR(4100) NOT NULL, created DATETIME NOT NULL, PRIMARY KEY(id), FOREIGN KEY (user_id) REFERENCES user(id) ON DELETE CASCADE)</pre>
id	VARCHAR(15)	"id" VARCHAR(15) NOT NULL
user_id	VARCHAR(15)	"user_id" VARCHAR(15) NOT NULL
title	VARCHAR(250)	"title" VARCHAR(250)
encrypted_content	VARCHAR(4100)	"encrypted_content" VARCHAR(4100) NOT NULL
created	DATETIME	"created" DATETIME NOT NULL
password_change_request		<pre>CREATE TABLE password_change_request (id VARCHAR(15) NOT NULL, user_id VARCHAR(15) NOT NULL, encrypted_code VARCHAR(4100) NOT NULL, created DATETIME NOT NULL, expires_at DATETIME NOT NULL, PRIMARY KEY(id), FOREIGN KEY(user_id) REFERENCES user (id) ON DELETE CASCADE)</pre>
id	VARCHAR(15)	"id" VARCHAR(15) NOT NULL
user_id	VARCHAR(15)	"user_id" VARCHAR(15) NOT NULL
encrypted_code	VARCHAR(4100)	"encrypted_code" VARCHAR(4100) NOT NULL
created	DATETIME	"created" DATETIME NOT NULL
expires_at	DATETIME	"expires_at" DATETIME NOT NULL
site		<pre>CREATE TABLE site (id VARCHAR(15) NOT NULL, user_id VARCHAR(15) NOT NULL, name VARCHAR(250), address VARCHAR(4100) NOT NULL, username VARCHAR(250) NOT NULL, encrypted_password VARCHAR(4100) NOT NULL, created DATETIME NOT NULL, PRIMARY KEY(id), FOREIGN KEY(user_id) REFERENCES user(id) ON DELETE CASCADE)</pre>
id	VARCHAR(15)	"id" VARCHAR(15) NOT NULL
user_id	VARCHAR(15)	"user_id" VARCHAR(15) NOT NULL
name	VARCHAR(250)	"name" VARCHAR(250)
address	VARCHAR(4100)	"address" VARCHAR(4100) NOT NULL
username	VARCHAR(250)	"username" VARCHAR(250) NOT NULL
encrypted_password	VARCHAR(4100)	"encrypted_password" VARCHAR(4100) NOT NULL
created	DATETIME	"created" DATETIME NOT NULL
user		<pre>CREATE TABLE user (id VARCHAR(15) NOT NULL, role VARCHAR(6) NOT NULL, fullname VARCHAR(250) NOT NULL, email VARCHAR(250) NOT NULL, hashed_password VARCHAR (100) NOT NULL, created DATETIME NOT NULL, PRIMARY KEY (id))</pre>

id	VARCHAR(15)	"id" VARCHAR(15) NOT NULL
role	VARCHAR(6)	"role" VARCHAR(6) NOT NULL
fullname	VARCHAR(250)	"fullname" VARCHAR(250) NOT NULL
email	VARCHAR(250)	"email" VARCHAR(250) NOT NULL
hashed_password	VARCHAR(100)	"hashed_password" VARCHAR(100) NOT NULL
created	DATETIME	"created" DATETIME NOT NULL

Índices (4)

Nombre	Tipo	Esquema
ix_creditcard_user_id		CREATE INDEX ix_creditcard_user_id ON creditcard (user_id) "user_id"
user_id		
ix_note_user_id		CREATE INDEX ix_note_user_id ON note (user_id) "user_id"
user_id		
ix_password_change_request_user_id		CREATE INDEX ix_password_change_request_user_id ON password_change_request (user_id) "user_id"
user_id		
ix_site_user_id		CREATE INDEX ix_site_user_id ON site (user_id) "user_id"
user_id		

Vistas (0)

Nombre	Tipo	Esquema

Disparadores (0)

Nombre	Tipo	Esquema

