

Tramitarte

[1\) Estructura del proyecto](#)

[2\) Entidades](#)

[2.1\) User](#)

[2.2\) AVORquest](#)

[2.3\) Notification](#)

[2.4\) DownloadRequest](#)

[2.5\) Documentation](#)

[2.6\) Stage](#)

[2.7\) TranslationRequest](#)

[2.8\) TranslationTask](#)

[2.9\) Process](#)

[3\) Servicios](#)

[3.1\) UserService](#)

[3.2\) AVORquestService](#)

[3.3\) NotificationService](#)

[3.4\) DownloadRequestService](#)

[3.5\) DocumentationService](#)

[3.6\) TranslationTaskService](#)

[3.7\) ProcessService](#)

[3.8\) TesseractService](#)

[4\) Routers](#)

[4.1\) UserRouter](#)

[4.2\) NotificationRouter](#)

[4.3\) DownloadRequestRouter](#)

[4.4\) TranslationTaskRouter](#)

[4.5\) ProcessRouter](#)

[4.6\) TesseractRouter](#)

[5\) Vistas](#)

[6\) Configuración de base de datos \(local\)](#)

[7\) Configuración de imagen de Docker](#)

[8\) Diagrama de clases](#)

[9\) Requisitos de instalación](#)

1)Estructura del proyecto

- /backend/: carpeta donde se almacena todo el backend del proyecto.
 - /database/: configuración de la base de datos para su uso.
 - /Docker/: inicialización de la base de datos utilizada para backend local.
 - /entities/: ubicación de todas las entidades.
 - /enums/: ubicación de todos los enums.
 - /exceptions/: ubicación de las excepciones personalizadas.
 - /models/: ubicación de los modelos de entrada y salida de los endpoints.
 - /routers/: ubicación de todos los routers de la aplicación.
 - /services/: ubicación de todos los servicios.
 - /tesseract_data/: ubicación de los archivos de entrenamiento de Tesseract.
 - /test_files/: ubicación de todos los archivos de prueba.
 - /tests/: ubicación de todos los tests del backend.
 - /entities/: almacenamiento de todos los tests de las entidades.
 - /routers/: almacenamiento de todos los tests de los routers.
 - /services/: almacenamiento de todos los tests de los services.
- /postman/: carpeta donde se almacena el archivo JSON de los endpoints.
- /Docker/: inicialización de la base de datos utilizada para imagen de toda la app.
- /documentation/: carpeta donde se almacena la documentación del proyecto.
 - /demo/: carpeta donde se almacenan los videos de las demos.
- /test_files/: ubicación de todos los archivos de prueba.
- /frontend/: carpeta donde se almacena todo el frontend del proyecto.
 - /src/: ubicación de todas las vistas y componentes de la aplicación.
 - /assets/: ubicación de las imágenes y logos.
 - /components/: ubicación de los componentes utilizados en las vistas.
 - /AnimatedLogo/: ubicación del logo animado.
 - /cards/: ubicación de las tarjetas.
 - /inputs/: ubicación de los Inputs.
 - /modals/: ubicación de los mensajes emergentes.
 - /requesterDocumentation/: ubicación de la estructura de Inputs para cargar documentación.
 - /data/: ubicación de la información utilizada en los mapas.
 - /pages/: ubicación de las páginas de la aplicación.
 - /router/: ubicación del router.
 - /services/: ubicación de la comunicación del frontend con el backend

2)Entidades

2.1) User

Entidad que representa al usuario en la aplicación. Nombre de la tabla: "users"

Propiedades	Tipo de dato
id	Integer
username	String
name	String
surname	String
role	Enum(Role)
birthdate	Date
need_traduction	Boolean
photo	String (optional)

Funciones:

- **update_user()** recibe un objeto de tipo **UpdateUserModel** que puede incluir los campos *username*, *surname* y *name*. Cada uno de estos campos, si se proporciona un valor, se utiliza para actualizar la información correspondiente del usuario; en caso contrario, la propiedad asociada permanecerá sin modificaciones.

2.2) AVORquest

Entidad que representa al AVO del usuario. Nombre de la tabla: "avo_requests".

Propiedades	Tipo de dato
id	Integer
first_name	String
last_name	String
birth_date	Date
gender	Enum(Gender)

Funciones:

- **is_valid()** → **bool**: retorna un valor booleano que determina si los datos del AVO cumplen con los criterios de validación establecidos. Se verifica que las propiedades *first_name* y *last_name* contengan valores válidos (no nulos) y que la propiedad *birth_date* no represente una fecha futura.

2.3) Notification

Entidad que representa las notificaciones de los usuarios. Nombre de la tabla: "notifications".

Propiedades	Tipo de dato
id	Integer
user_origin_id	Integer
user_destination_id	Integer
description	String

2.4) DownloadRequest

Entidad que representa los pedidos de descarga de documentación, Esta misma será enviada a los traductores con los archivos a traducir y a los solicitantes con los archivos traducidos . Nombre de la tabla: "download_requests".

Propiedades	Tipo de dato
id	Integer
translator_id	Integer
requester_id	Integer
documentation	List[Documentation]

2.5) Documentation

Entidad que representa los archivos cargados al sistema. Nombre de la tabla: "documentation".

Propiedades	Tipo de dato
id	Integer
name	String
file_type	String
file_base64	String
process_id	Integer
download_request_id	Integer (optional)
document_type	String

La aplicación recibe documentación de diversos actores a lo largo del proceso. Para diferenciarlas, se han definido las siguientes entidades, las cuales heredan de **Documentation**:

- **AvoDocumentation**: Representa la documentación del AVO. Nombre de la tabla: *avo_documentation*.
- **UserDocumentation**: Representa la documentación del usuario. Nombre de la tabla: *user_documentation*.
- **AncestorDocumentation**: Representa la documentación de los ancestros del usuario. Nombre de la tabla: *ancestor_documentation*.
- **TranslatedDocumentation**: Representa la documentación traducida del trámite. Nombre de la tabla: *translated_documentation*.
- **AttachmentDocumentation**: Representa la documentación que requiere traducción del trámite. Nombre de la tabla: *attachment_documentation*.

2.6) Stage

Entidad que representa los etapas del trámite. Nombre de la tabla: "stages".

Propiedades	Tipo de dato
id	Integer

description	String
stage_type	String

Funciones:

- **verify_stage()** → **bool**: retorna un valor booleano y se encarga de verificar que el trámite cuenta con todos los elementos necesarios para avanzar a la siguiente etapa. Si la validación es exitosa, la etapa actual del trámite se actualiza a la siguiente; en caso contrario, se lanza una excepción.

El proceso se lleva a cabo en múltiples etapas, cada una con su respectiva validación. Para diferenciarlas, se definen las siguientes entidades que heredan de **Stage**:

- **Stage1**: Verifica que el AVO cargado contenga información válida. Nombre de la tabla: *stage1*.
- **Stage2**: Comprueba que la documentación del usuario esté completa. Nombre de la tabla: *stage2*.
- **Stage3**: Confirma la existencia de la documentación del AVO. Nombre de la tabla: *stage3*.
- **Stage4**: Valida que la cantidad de documentación de los ancestros del usuario sea la adecuada. Nombre de la tabla: *stage4*.
- **Stage5**: Determina si el trámite dispone de toda la documentación traducida necesaria. Nombre de la tabla: *stage5*.

2.7) TranslationRequest

Entidad que representa los pedidos de traducción existentes de los traductores. Nombre de la tabla: "translation_requests".

Propiedades	Tipo de dato
id	Integer
requester_id	Integer
translator_id	Integer

2.8) TranslationTask

Entidad que representa el pedido de traducción aceptado por los traductores, del cual los traductores van a tener acceso a la documentación del trámite que debe traducirse. Nombre de la tabla: "translation_tasks".

Propiedades	Tipo de dato
id	Integer
process_id	Integer
translator_id	Integer

2.9) Process

Entidad central de la aplicación, es la encargada de administrar el proceso del trámite y almacenar toda la información del mismo. Contiene un Stage, un User, un AVO Request, y varias listas con documentación. Nombre de la tabla: "processes".

Propiedades	Tipo de dato
id	Integer
code	String
stage_id	Integer
user_id	Integer
request_avo_id	Integer
stage	Stage
user	User
request_avo	AVORequest
avo_documentation	List[AvoDocumentation]
user_documentation	List[UserDocumentation]
ancestors_documentation	List[AncestorDocumentation]
translated_documentation	List[TranslatedDocumentation]
attachments_to_translate	List[AttachmentDocumentation]
ancestor_count	Integer

Funciones:

- **assign_avo_request()**: Recibe un objeto de tipo **AVORequest** y lo asigna a la propiedad **request_avo**.
- **has_translated_documentation()** → **bool**: Retorna un valor booleano que indica si la cantidad de documentación traducida coincide con la cantidad de documentación pendiente de traducción.
- **advance_stage()**: Evalúa si el trámite cumple con los requisitos necesarios para avanzar a la siguiente etapa.
- **add_avo_documentation()**: Recibe una lista de objetos de tipo **Documentation**; cada documento se transforma en **AvoDocumentation** y se inserta en la propiedad **avo_documentation**.
- **add_user_documentation()**: Recibe una lista de objetos de tipo **Documentation**; cada documento se convierte en **UserDocumentation** y se incorpora en la propiedad **user_documentation**.
- **add_ancestors_documentation()**: Recibe una lista de objetos de tipo **Documentation**; cada documento se transforma en **AncestorDocumentation** y se agrega a la propiedad **ancestors_documentation**.
- **add_attachments_to_translate()**: Recibe una lista de objetos de tipo **Documentation**; cada documento se convierte en **AttachmentDocumentation** y se inserta en la propiedad **attachments_to_translate**.
- **add_translated_documentation()**: Recibe una lista de objetos de tipo **Documentation**; cada documento se transforma en **TranslatedDocumentation** y se incorpora en la propiedad **translated_documentation**.

3)Servicios

3.1) UserService

- **find_translators()** → **List[User]**: Retorna una lista de todos los usuarios registrados en el sistema que poseen el rol "TRANSLATOR".
- **find_by_role()** → **List[User]**: Recibe un objeto de tipo **Role** para filtrar y devolver una lista de usuarios que cumplen con dicho rol.
- **find_by_id()** → **Optional[User]**: Recibe un identificador de usuario y retorna el usuario correspondiente; en caso de no encontrarlo, retorna None.
- **create()** → **User**: Recibe un objeto de tipo **CreateUserModel**, que contiene todas las propiedades de un usuario, excepto el identificador. Crea el usuario almacenando los datos proporcionados en el modelo.

- **find_by_email()** → **User**: Recibe y valida un correo electrónico, retornando el usuario asociado. Si el formato del email es incorrecto, lanza una excepción HTTP con status 400 indicando que el formato es inválido. Si no se encuentra un usuario con el correo proporcionado, lanza una excepción HTTP con status 404 indicando que el usuario no existe.
- **find_translator_by_email()** → **Optional[User]**: Recibe y valida un correo electrónico, retornando el traductor asociado. Si el formato del email es incorrecto, lanza una excepción HTTP con status 400 indicando que el formato es inválido. Si no se encuentra un traductor con ese correo, retorna None.
- **find_notifications_by_user_destination_id()** → **List[NotificationModel]**: Recibe el identificador de un usuario. Si el usuario no existe, lanza una excepción HTTP con status 404 indicando que el usuario no existe. Filtra y retorna todas las notificaciones dirigidas a ese usuario.
- **find_translation_requests()** → **List[TranslationRequest]**: Recibe el identificador de un traductor. Si el traductor no existe, lanza una excepción HTTP con status 404 indicando que el traductor no existe. Filtra y retorna todos los pedidos de traducción asignados al traductor.
- **find_translation_requests_by_requester_and_translator()** → **List[TranslationRequest]**: Recibe los identificadores del solicitante y del traductor. Si alguno de ellos no existe, lanza una excepción HTTP con status 404 indicando que el traductor o el solicitante no existe. Filtra y retorna los pedidos de traducción correspondientes a la combinación de solicitante y traductor.
- **find_request_by_requester()** → **Optional[TranslationRequest]**: Recibe el identificador del solicitante. Si el solicitante no existe, lanza una excepción HTTP con status 404 indicando que el solicitante no existe. Retorna el pedido de traducción asociado al solicitante; en caso contrario, retorna None.
- **update()** → **User**: Recibe un identificador de usuario y un objeto de tipo **UpdateUserModel**. Si el usuario no existe, lanza una excepción HTTP con status 404 indicando que el usuario no existe. Actualiza los parámetros del usuario utilizando los datos del modelo recibido y retorna el usuario actualizado.
- **delete()** → **User**: Recibe un identificador de usuario. Si el usuario no existe, lanza una excepción HTTP con status 404 indicando que el usuario no existe. Elimina el usuario y retorna el usuario eliminado.
- **_validate_email_format()**: Recibe un correo electrónico y verifica que cumpla con el patrón `^[0-9A-Za-z.-]+@[A-Za-z]+\.[a-zA-Z]+$`. Si el correo no coincide con el patrón, lanza una excepción HTTP con status 400 indicando que el formato es inválido.

3.2) AVORequestService

- **save() → AvoRequestModel**: Recibe un objeto de tipo **AVORequestModel** que contiene todos los datos del AVO, crea el AVO y lo retorna. Si los datos del AVO son inválidos, lanza una excepción HTTP con status 400 indicando que dichos datos no son válidos.
- **update() → AVORequestModel**: Recibe un objeto de tipo **UpdateAVORequestModel** con los nuevos datos del AVO. Primero, verifica la existencia del AVO a modificar; de no existir, lanza una excepción HTTP con status 404 indicando que el AVO a modificar no existe. Posteriormente, actualiza el AVO y valida sus datos. Si la validación falla, lanza una excepción HTTP con status 400 indicando que los datos del AVO son inválidos. Finalmente, retorna el AVO modificado.
- **find_avo_by_user() → Optional[AVORequestModel]**: Recibe un objeto de tipo **UserModel** y filtra el trámite que contiene el AVO correspondiente, retornándolo. En caso de no existir un AVO en el trámite, retorna None.
- **_validate_existing_request()**: Recibe un objeto de tipo **AVORequestModel** y verifica la existencia de un AVO con los datos proporcionados. Si no se encuentra, lanza una excepción HTTP con status 404 indicando que el AVO no existe.

3.3) NotificationService

- **find_all_notifications_by_user_destination() → List[Notification]**: Recibe un identificador del usuario destinatario. Si el usuario no existe, lanza una excepción HTTP con status 404 indicando que el usuario no existe. Posteriormente, filtra y retorna todas las notificaciones cuyo destinatario coincide con el identificador proporcionado.
- **generate_alert_to_translator()**: Recibe un identificador del usuario origen y otro del usuario destino. Se verifica la existencia de ambos usuarios; en caso de que alguno no exista, se lanza una excepción HTTP con status 404 indicando que el destinatario o el usuario destino no existen. Si ambos existen, se genera una alerta y se crea una solicitud de traducción para el destinatario.
- **generate_alert()**: Recibe un identificador del usuario origen, un identificador del usuario destino y la descripción de la alerta. Se verifica que ambos usuarios existan; de lo contrario, se lanza una excepción HTTP con status 404 indicando que el destinatario o el usuario destino no existen. En caso afirmativo, se genera una alerta dirigida al destinatario.

- **delete_translation_request():** Recibe el identificador de la solicitud de traducción. Si la solicitud no existe, lanza una excepción HTTP con status 404 indicando que la solicitud no existe; de lo contrario, elimina la solicitud.
- **delete_translation_request_by_requester():** Recibe el identificador del solicitante. Si el usuario no existe, lanza una excepción HTTP con status 404 indicando que el usuario no existe. Si se encuentra una solicitud de traducción asociada, la elimina y genera una alerta para notificar al traductor que el solicitante ha eliminado la solicitud.
- **delete_notification_by_id() → Notification:** Recibe el identificador de la notificación. Si la notificación no existe, lanza una excepción HTTP con status 404 indicando que la notificación no existe. En caso contrario, elimina la notificación y la retorna.

3.4) DownloadRequestService

- **create_download_request():** Recibe el identificador del solicitante, el identificador del traductor y los documentos correspondientes. Si el solicitante o el traductor no existen, lanza una excepción HTTP con status 404 indicando que alguno de ellos no se encontró. En caso contrario, crea la solicitud de descarga.
- **find_requests_by_requester() → List[DownloadRequest]:** Recibe el identificador del solicitante. Si el solicitante no existe, lanza una excepción HTTP con status 404 indicando que el solicitante no se encontró. Devuelve una lista de solicitudes de descarga filtradas según el solicitante.
- **delete_download_request_by_id() → dict:** Recibe el identificador de una solicitud de descarga. Si la solicitud no existe, lanza una excepción HTTP con status 404 indicando que la solicitud no se encontró. De lo contrario, elimina la solicitud y devuelve un mensaje confirmando que la operación se realizó con éxito.

3.5) DocumentationService

- **update():** Recibe un identificador de documento y un objeto del tipo **DocumentationUpdateModel**. Si el documento no se encuentra, lanza una excepción HTTP con status 404 indicando que el documento no existe. En caso contrario, actualiza los campos *file_type*, *name* y *file_base64* con los datos proporcionados en el modelo y persiste los cambios en la base de datos.

3.6) TranslationTaskService

- **create_translation_task()**: Recibe un identificador del solicitante y otro del traductor. Si alguno de ellos no existe, lanza una excepción HTTP con status 404 indicando que el solicitante o el traductor no se encontraron. A continuación, se obtiene el trámite asociado al usuario solicitante; si este no existe, se lanza una excepción HTTP con status 404 indicando que el trámite no existe. Finalmente, se crea una tarea de traducción utilizando el trámite y el traductor identificados.
- **find_by_translator()** → **List[TranslationTask]**: Recibe el identificador del traductor. Si el traductor no se encuentra, lanza una excepción HTTP con status 404 indicando que el traductor no existe. En caso contrario, retorna una lista de objetos **TranslationTask** filtrados por el traductor.
- **find_by_process()** → **List[TranslationTask]**: Recibe el identificador del trámite. Si el trámite no existe, lanza una excepción HTTP con status 404 indicando que el trámite no existe. De lo contrario, retorna una lista de objetos **TranslationTask** correspondientes al trámite especificado.
- **delete_by_id()**: Recibe un identificador de **TranslationTask**. Si la tarea no se encuentra, lanza una excepción HTTP con status 404 indicando que la tarea de traducción no existe. En caso contrario, elimina la tarea.

3.7) ProcessService

- **start_process()** → **ProcessModel**: Recibe un identificador de usuario. Si el usuario no existe, lanza una excepción HTTP con status 404 indicando que el usuario no se encontró. En caso contrario, crea un trámite asignándole un código aleatorio, el usuario y la primera etapa del proceso.
- **upload_avo()** → **ProcessModel**: Recibe un identificador de trámite y un objeto de tipo **AVORequestModel**. Si el trámite no existe, lanza una excepción HTTP con status 404 indicando que el trámite no se encontró. Posteriormente, asocia el AVO al trámite y avanza a la siguiente etapa.
- **upload_user_documents()** → **ProcessModel**: Recibe un identificador de trámite y una lista de objetos **DocumentationUpdateModel**. Si el trámite no existe, lanza una excepción HTTP con status 404. Cada uno de los modelos se transforma en una instancia de **UserDocumentation** y se inserta en la lista de documentación del usuario del trámite; además, se filtran los documentos de tipo PDF para agregarlos a la lista de adjuntos a traducir. Finalmente, se avanza a la siguiente etapa.
- **upload_avo_documents()** → **ProcessModel**: Recibe un identificador de trámite y una lista de objetos **DocumentationUpdateModel**. Si el trámite no existe, lanza una

excepción HTTP con status 404. Cada modelo se convierte en una instancia de **AvoDocumentation** y se inserta tanto en la lista de documentación del AVO como en la lista de adjuntos a traducir. Finalmente, se procede a la siguiente etapa.

- **upload_ancestors_documents()** → **ProcessModel**: Recibe un identificador de trámite y un objeto **AncestorDocumentationModel**, el cual contiene la cantidad de ancestros y sus respectivos documentos. Si el trámite no existe, lanza una excepción HTTP con status 404. Los modelos de documentación se transforman en instancias de **AncestorDocumentation** y se agregan tanto a la lista de documentación de ancestros del trámite como a la lista de adjuntos a traducir. Finalmente, se avanza a la siguiente etapa.
- **upload_translated_documents()** → **ProcessModel**: Recibe un identificador de trámite y una lista de objetos **DocumentationUpdateModel**. Si el trámite no existe, lanza una excepción HTTP con status 404. Cada modelo se convierte en una instancia de **TranslatedDocumentation** y se inserta en la lista de documentos traducidos del trámite. Finalmente, se avanza a la siguiente etapa.
- **get_documents()** → **List[DocumentationModel]**: Recibe un identificador de trámite. Si el trámite no existe, lanza una excepción HTTP con status 404. Retorna todos los documentos asociados al trámite.
- **delete_process()**: Recibe un identificador de trámite. Si el trámite no existe, lanza una excepción HTTP con status 404. Elimina el trámite y todos los elementos asociados al mismo.
- **find_by_user()** → **Optional[ProcessModel]**: Recibe un identificador de usuario y retorna el trámite correspondiente; en caso de no encontrar ninguno, retorna None.

3.8) TesseractService

- **recognize_image()** → **str**: Extrae texto de una imagen utilizando Tesseract. Si no se proporciona una imagen, lanza una excepción HTTP con status 400, indicando que el archivo no es una imagen válida.
- **recognize_pdf()** → **str**: Extrae texto de un archivo PDF utilizando PdfReader.
- **extract_images_from_pdf_with_ocr()** → **str**: Extrae texto de las imágenes contenidas en un archivo PDF utilizando Tesseract. Si el archivo proporcionado no es un PDF, lanza una excepción HTTP con status 400, indicando que el archivo no es un PDF válido.
- **is_dni_front()** → **bool**: Verifica si la imagen escaneada corresponde al frente de un DNI.

- **is_dni_back()** → **bool**: Verifica si la imagen escaneada corresponde al dorso de un DNI.
- **is_certificate()** → **bool**: Determina si el archivo PDF escaneado es un certificado.
- **is_birth_certificate()** → **bool**: Verifica si el archivo PDF escaneado es un certificado de nacimiento.
- **is_marriage_certificate()** → **bool**: Verifica si el archivo PDF escaneado es un certificado de casamiento.
- **is_death_certificate()** → **bool**: Verifica si el archivo PDF escaneado es un certificado de defunción.
- **is_birth_certificate_italy()** → **bool**: Verifica si el archivo PDF escaneado es un certificado de nacimiento traducido al italiano.
- **is_marriage_certificate_italy()** → **bool**: Verifica si el archivo PDF escaneado es un certificado de casamiento traducido al italiano.
- **is_death_certificate_italy()** → **bool**: Verifica si el archivo PDF escaneado es un certificado de defunción traducido al italiano.
- **_contains_phrase_pdf_death_italy()** → **bool**: Determina si el texto extraído contiene una palabra o frase clave característica de un certificado de defunción traducido al italiano.
- **_contains_phrase_pdf_marriage_italy()** → **bool**: Determina si el texto extraído contiene una palabra o frase clave característica de un certificado de matrimonio traducido al italiano.
- **_contains_phrase_pdf_birth_italy()** → **bool**: Determina si el texto extraído contiene una palabra o frase clave característica de un certificado de nacimiento traducido al italiano.
- **_contains_phrase_pdf_death()** → **bool**: Determina si el texto extraído contiene una palabra o frase clave característica de un certificado de defunción.
- **_contains_phrase_pdf_marriage()** → **bool**: Determina si el texto extraído contiene una palabra o frase clave característica de un certificado de matrimonio.
- **_contains_phrase_pdf_birth()** → **bool**: Determina si el texto extraído contiene una palabra o frase clave característica de un certificado de nacimiento.
- **_contains_phrase_birth()** → **bool**: Determina si el texto extraído contiene una palabra o frase clave relacionada con un certificado.

- **_contains_phrase_front()** → **bool**: Verifica si el texto extraído contiene una palabra o frase clave correspondiente al frente de un DNI.
- **_contains_phrase_back()** → **bool**: Verifica si el texto extraído contiene una palabra o frase clave correspondiente al dorso de un DNI.
- **_find_pattern()** → **bool**: Verifica si el texto extraído contiene un patrón típico de un DNI.

4)Routers

4.1) UserRouter

- **GET /api/user/translators**
Retorna una lista de todos los traductores registrados.
- **GET /api/user/requesters**
Retorna una lista de todos los solicitantes registrados.
- **GET /api/user/{user_destination_id}/notifications**
Retorna las notificaciones asociadas al usuario identificado por *user_destination_id*.
- **GET /api/user/{id}**
Retorna la información de un usuario específico identificado por su *id*.
- **GET /api/user/**
Retorna la información de un usuario basado en su correo electrónico.
 - **Query Parameters:** email (str): Correo electrónico del usuario.
- **GET /api/user/{translator_id}/translation-requests**
Retorna una lista de objetos **TranslationRequest** asociados al traductor identificado por *translator_id*.
- **GET /api/user/translation-requests/requester/{requester_id}/translator/{translator_id}**
Retorna una lista de objetos **TranslationRequest** filtrados por los identificadores del solicitante y del traductor.
- **GET /api/user/requests/requester/{requester_id}**
Retorna el objeto **TranslationRequest** asociado al solicitante identificado por *requester_id*.

- **GET /api/user/translator/email**
Retorna la información de un traductor basado en su correo electrónico.
 - **Query Parameters:** email (str): Correo electrónico del traductor.
- **POST /api/user/**
Crea un nuevo usuario.
 - **Body:** Objeto de tipo **CreateUserModel**.
- **PUT /api/user/{id}**
Actualiza la información de un usuario específico identificado por su *id*.
 - **Body:** Objeto de tipo **UpdateUserModel**.
- **DELETE /api/user/{id}**
Elimina el usuario identificado por su *id*.

4.2) NotificationRouter

- **GET /api/notification/{user_destination_id}**
Retorna una lista de notificaciones asociadas al usuario destino identificado por *user_destination_id*.
- **POST /api/notification/alert-translator/{origin_id}/{destination_id}**
Crea una alerta y un objeto **TranslationRequest** dirigido al traductor, utilizando los identificadores del usuario origen (*origin_id*) y del usuario destino (*destination_id*).
 - **Query Parameters:** description (str): Descripción de la alerta.
- **POST /api/notification/alert/{origin_id}/{destination_id}**
Genera una alerta dirigida al usuario destino, utilizando los identificadores del usuario origen (*origin_id*) y del usuario destino (*destination_id*).
 - **Query Parameters:** description (str): Descripción de la alerta.
- **DELETE /api/notification/alert/{notification_id}**
Elimina la notificación identificada por *notification_id*.
- **DELETE /api/notification/request/{request_id}**
Elimina el objeto **TranslationRequest** identificado por *request_id*.
- **DELETE /api/notification/request/requester/{requester_id}**
Elimina el objeto **TranslationRequest** asociado al solicitante identificado por *requester_id*.

4.3) DownloadRequestRouter

- **GET /api/download-request/requester/{requester_id}**
Retorna una lista de objetos **DownloadRequestModel** asociados al solicitante identificado por *requester_id*.
- **POST /api/download-request/requester/{requester_id}/translator/{translator_id}**
Crea un objeto **DownloadRequest** utilizando el identificador del solicitante, el identificador del traductor y los documentos a descargar.
 - **Body:** Lista de objetos **DocumentationUpdateModel**.
- **DELETE /api/download-request/{request_id}**
Elimina el objeto **DownloadRequest** identificado por *request_id*.

4.4) TranslationTaskRouter

- **GET /api/task/translator/{translator_id}**
Retorna una lista de objetos **TranslationTask** asociados al traductor identificado por *translator_id*.
- **GET /api/task/process/{process_id}**
Retorna una lista de objetos **TranslationTask** correspondientes al trámite identificado por *process_id*.
- **POST /api/task/requester/{requester_id}/translator/{translator_id}**
Crea un objeto **TranslationTask** utilizando los identificadores del solicitante y del traductor.
- **DELETE /api/task/{request_id}**
Elimina el objeto **TranslationTask** identificado por *request_id*.

4.5) ProcessRouter

- **GET /api/process/user/{user_id}**
Retorna el trámite asociado al usuario identificado por *user_id*.
- **GET /api/process/request/user/{user_id}**
Retorna el objeto **AVORequest** correspondiente al usuario identificado por *user_id*.
- **GET /api/process/documentation/{process_id}**
Retorna toda la documentación cargada en el trámite identificado por *process_id*.

- **POST /api/process/{user_id}**
Crea un nuevo trámite para el usuario identificado por *user_id*.
- **POST /api/process/upload-avo/{process_id}**
Carga la información del AVO en el trámite identificado por *process_id* y avanza a la siguiente etapa.
 - **Body:** Objeto de tipo **AVORequestModel**.
- **POST /api/process/upload/documentation/user/{user_id}**
Carga la documentación del usuario en el trámite utilizando el identificador del usuario y avanza a la siguiente etapa.
 - **Body:** Lista de objetos **DocumentationUpdateModel**.
- **POST /api/process/upload/documentation/avo/{process_id}**
Carga la documentación del AVO en el trámite identificado por *process_id* y avanza a la siguiente etapa.
 - **Body:** Lista de objetos **DocumentationUpdateModel**.
- **POST /api/process/upload/documentation/ancestors/{process_id}**
Carga la documentación de los ancestros, junto con la cantidad correspondiente, en el trámite identificado por *process_id* y avanza a la siguiente etapa.
 - **Body:** Lista de objetos **AncestorDocumentationModel**.
- **POST /api/process/upload/documentation/translated/{user_id}**
Carga la documentación traducida en el trámite utilizando el identificador del usuario y avanza a la siguiente etapa.
 - **Body:** Lista de objetos **DocumentationUpdateModel**.
- **PUT /api/process/modify/document/{document_id}**
Actualiza un documento existente identificado por *document_id*.
 - **Body:** Objeto de tipo **DocumentationUpdateModel**.
- **PUT /api/process/avo**
Actualiza la información del AVO cargado en el trámite.
 - **Body:** Objeto de tipo **UpdateAVORequestModel**.
- **DELETE /api/process/{process_id}**
Elimina el trámite identificado por *process_id*.

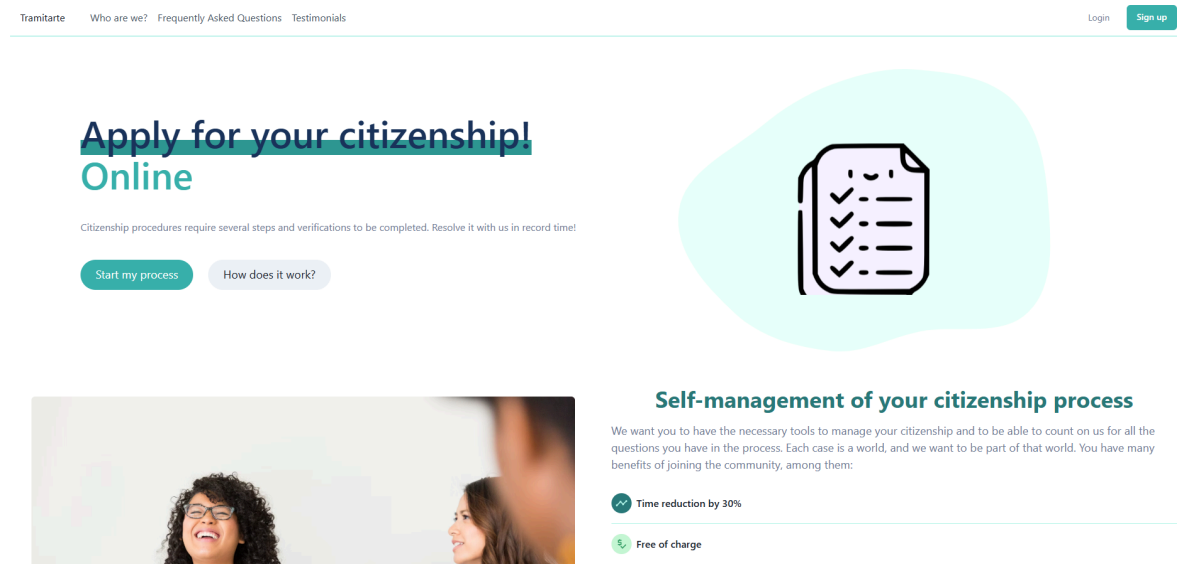
4.6) TesseractRouter

- **POST /api/ocr/image**
Extrae texto de una imagen utilizando OCR.
 - **Body:** img (form-data): Archivo de imagen.
- **POST /api/ocr/pdf**
Extrae texto de un archivo PDF utilizando OCR.
 - **Body:** file (form-data): Archivo PDF.
- **POST /api/ocr/is_dni_front**
Verifica si la imagen proporcionada corresponde al frente de un DNI.
 - **Body:** img (form-data): Archivo de imagen.
- **POST /api/ocr/is_dni_back**
Verifica si la imagen proporcionada corresponde al dorso de un DNI.
 - **Body:** img (form-data): Archivo de imagen.
- **POST /api/ocr/pdf/is_certificate**
Verifica si el PDF proporcionado corresponde a un certificado.
 - **Body:** pdf (form-data): Archivo PDF.
- **POST /api/ocr/pdf/text_from_img**
Extrae texto de las imágenes contenidas en un archivo PDF utilizando OCR.
 - **Body:** pdf (form-data): Archivo PDF.
- **POST /api/ocr/pdf/is_marriage**
Verifica si el PDF corresponde a un certificado de casamiento.
 - **Body:** pdf (form-data): Archivo PDF.
- **POST /api/ocr/pdf/is_birth**
Verifica si el PDF corresponde a un certificado de nacimiento.
 - **Body:** pdf (form-data): Archivo PDF.
- **POST /api/ocr/pdf/is_death**
Verifica si el PDF corresponde a un certificado de defunción.
 - **Body:** pdf (form-data): Archivo PDF.
- **POST /api/ocr/pdf/is_marriage_italy**
Verifica si el PDF corresponde a un certificado de casamiento traducido al italiano.
 - **Body:** pdf (form-data): Archivo PDF.
- **POST /api/ocr/pdf/is_birth_italy**
Verifica si el PDF corresponde a un certificado de nacimiento traducido al italiano.
 - **Body:** pdf (form-data): Archivo PDF.
- **POST /api/ocr/pdf/is_death_italy**
Verifica si el PDF corresponde a un certificado de defunción traducido al italiano.
 - **Body:** pdf (form-data): Archivo PDF.

5)Vistas

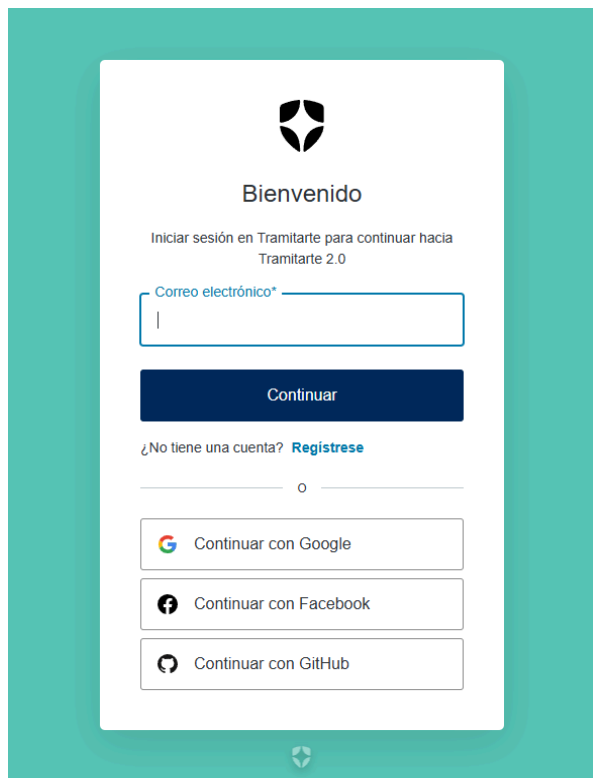
5.1) Home

Página inicial de la aplicación. Muestra un resumen sobre las ventajas y el uso del proyecto. Incluye opciones para **registrarse** o **iniciar sesión**.



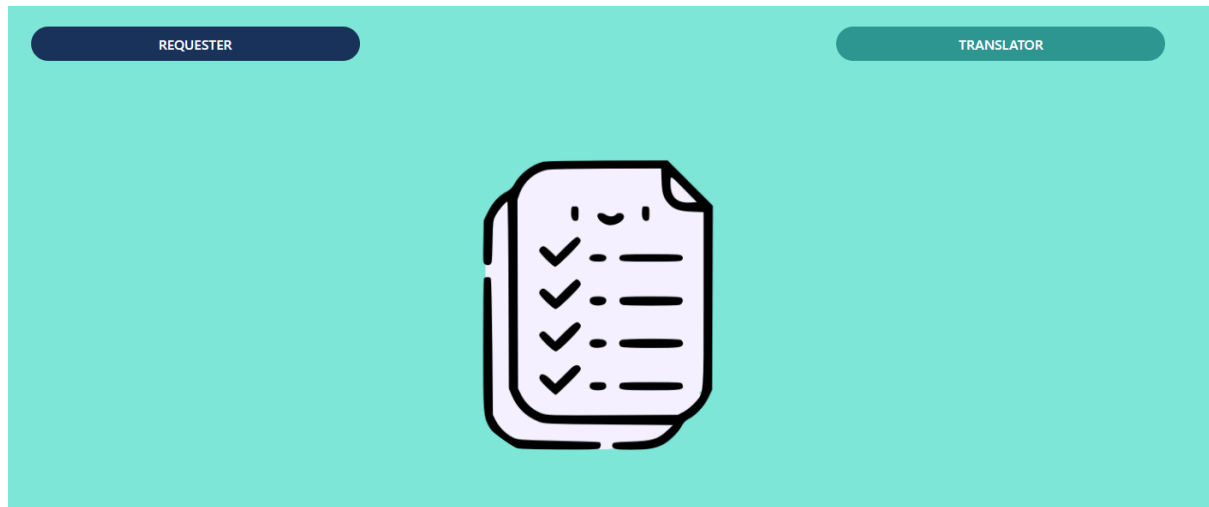
5.2) Login

Página para **iniciar sesión** o **registrarse** en la aplicación.



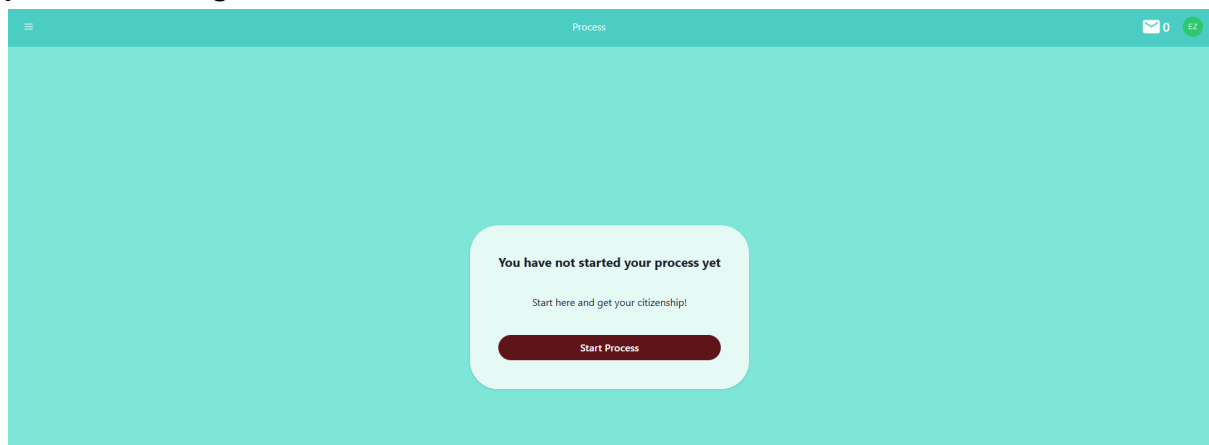
5.3) RoleElection

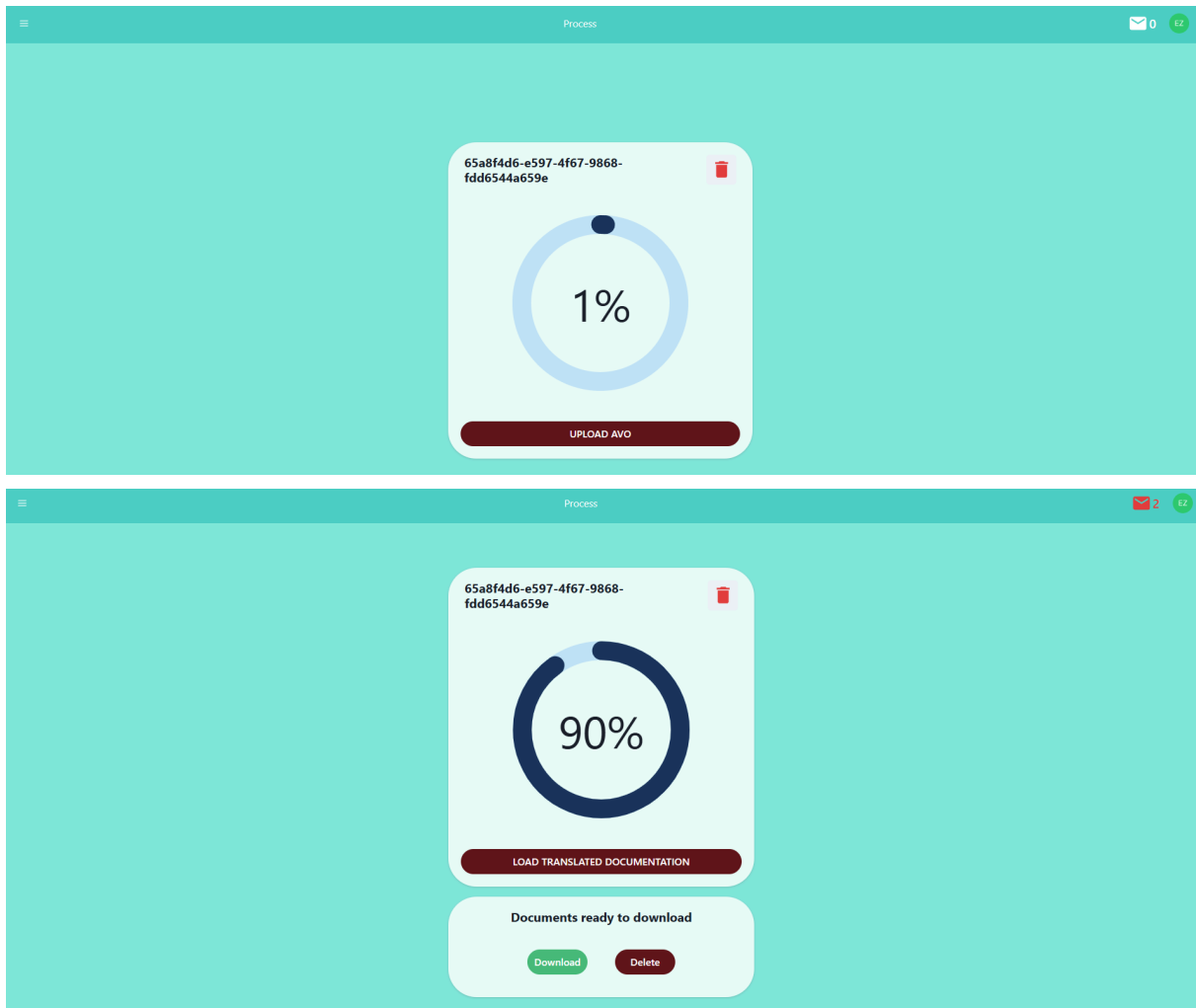
Página para elegir el rol del usuario. Permite al usuario **elegir su rol**. Una vez seleccionado, se crea el rol y se **redirige al Home** del usuario.



5.4) UserHome (Solicitante)

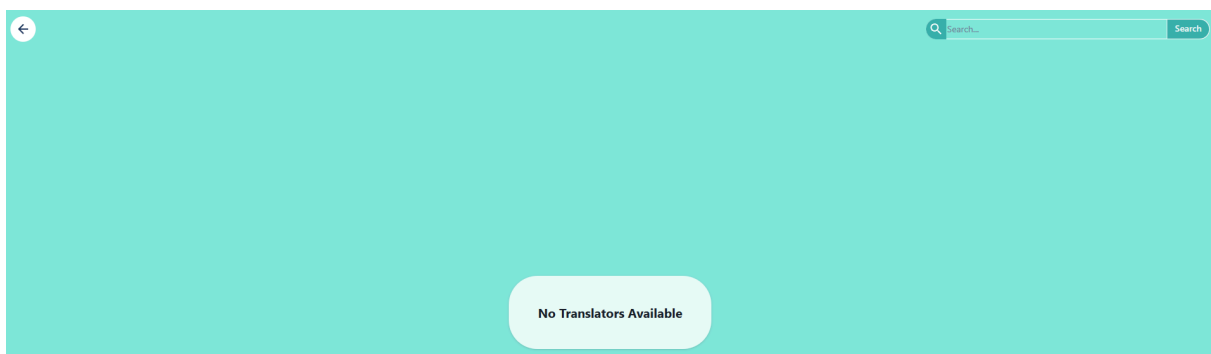
Esta página presenta la interfaz principal para los solicitantes. En el centro, se muestra una notificación destacada para **iniciar un nuevo trámite**. En los laterales, se dispone de un **menú desplegable**, una sección de **notificaciones** y el acceso al **perfil del usuario**. Además, en caso de que un traductor haya enviado documentos, se mostrará una **alerta para su descarga**.





5.5) Registered Translators

Esta página muestra a los **traductores registrados** en la aplicación. Los usuarios pueden **filtrar y buscar** traductores mediante un **buscador**, facilitando la selección según sus necesidades.





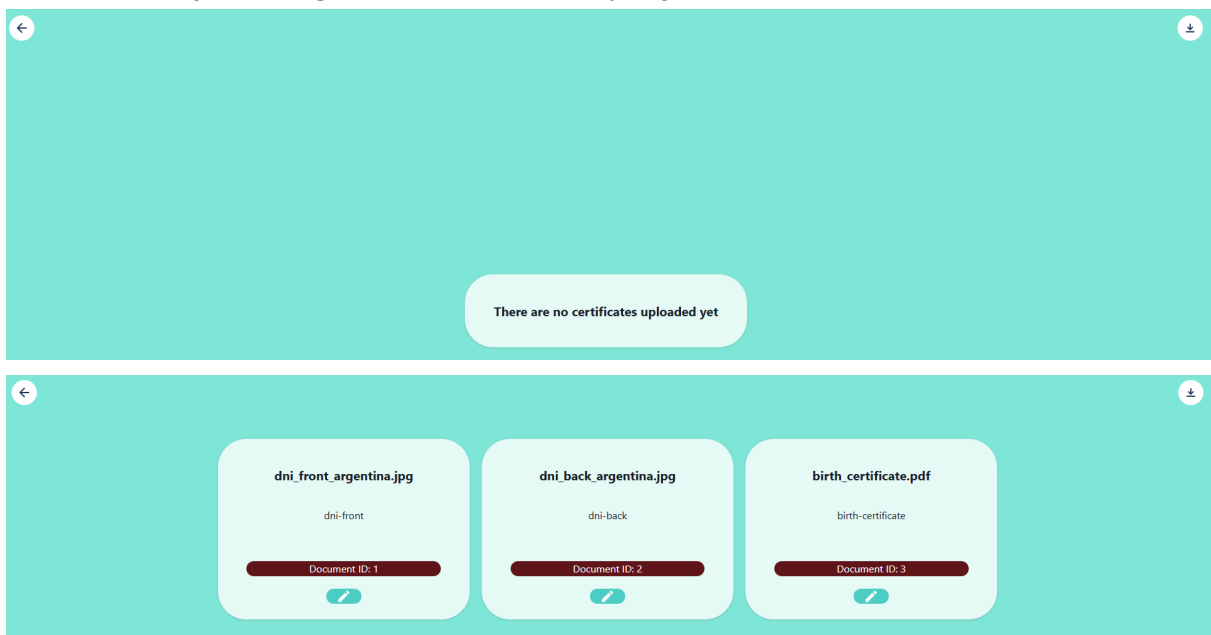
5.6) FrequentQuestions

Esta página proporciona un apartado de **preguntas frecuentes** para resolver dudas comunes. Además, incluye **mapas interactivos** con las ubicaciones de **consulados**, **Family Searches** y **centros de traducción**, facilitando el acceso a estos servicios.



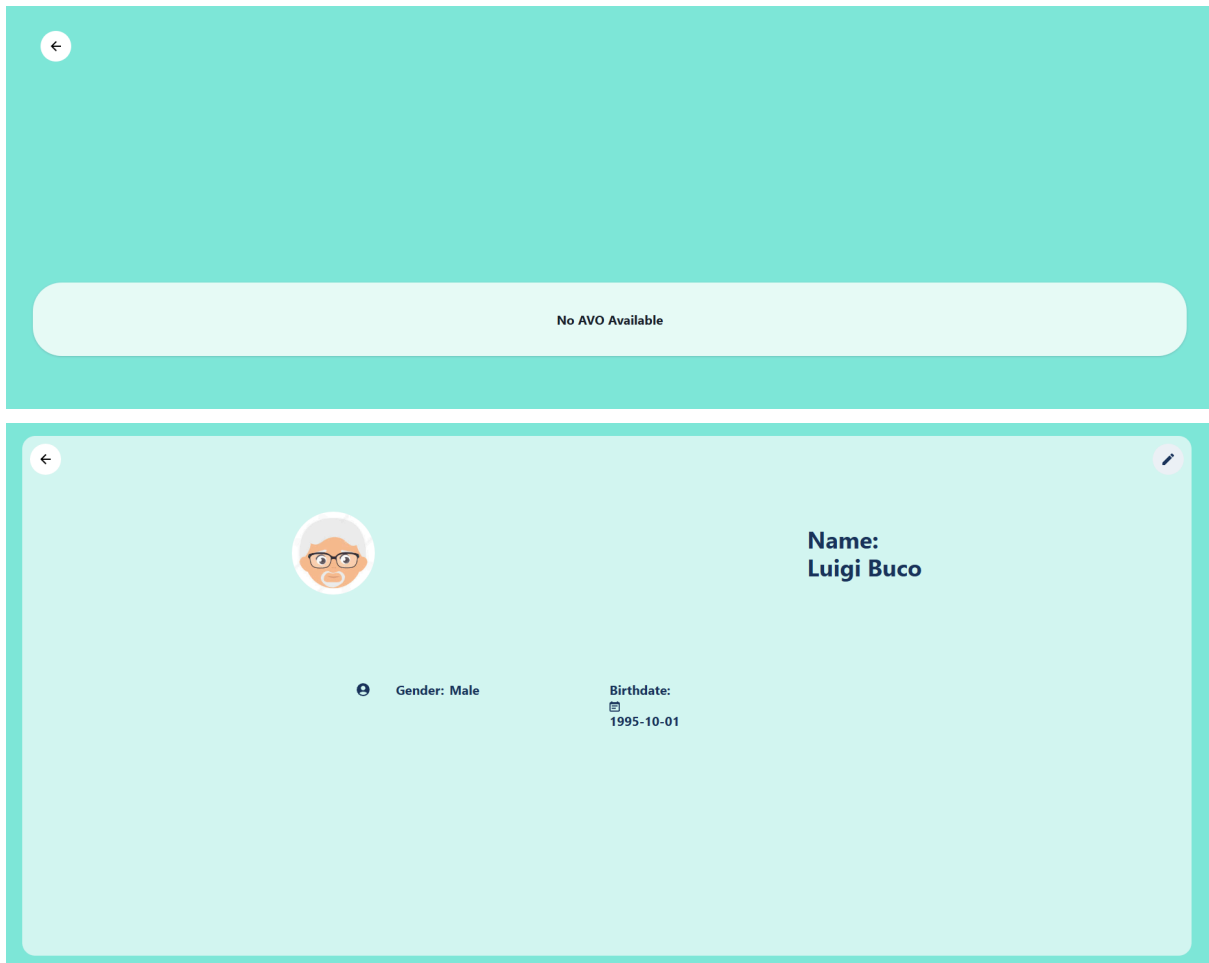
5.7) DocumentationUploaded

Esta página muestra toda la documentación cargada por el usuario, permitiendo su **visualización y descarga** de manera sencilla y organizada.



5.8) AvoProfile

Esta página muestra la información del AVO cargado, permitiendo su **visualización y edición** para asegurar la actualización de los datos según sea necesario.



5.9) AVORequest

Esta página permite al usuario **ingresar y registrar la información del AVO**, asegurando que los datos sean completados correctamente para su posterior procesamiento.

A screenshot of a web form titled 'AVO SEARCHING'. At the top, there is a dark blue header with a white back arrow on the left and a dark blue pill-shaped button containing the alphanumeric string '65a8f4d6-e597-4f67-9868-fdd6544a659e'. Below the header is a light blue box with a dark blue bar at the top that says 'AVO SEARCHING'. Inside this box, the text 'Fill in the details of your Italian ancestor who emigrated' is displayed. Below this is a dark blue bar with the text 'COMPLETE REQUEST' and a small white checkmark. The form contains several input fields: 'Name *' with a placeholder 'Name...', 'Surname *' with a placeholder 'Surname...', and 'Birthdate *' with a date range from 1 to 1995 and a calendar icon. Below the birthdate field, the text 'Your AVO's date of birth' is shown. There is also a 'Biological sex *' section with radio buttons for 'Female' (selected) and 'Male'. At the bottom of the form is a dark blue bar with the text 'UPLOAD AVO'.

5.10) PersonalDocumentation

Esta página permite al usuario **subir las imágenes de su documento de identidad y certificado de nacimiento**, asegurando que la información sea registrada correctamente para su validación y procesamiento.

A screenshot of a web form titled 'Personal Documentation'. At the top, there is a dark blue header with a white back arrow on the left and a dark blue pill-shaped button containing the alphanumeric string '65a8f4d6-e597-4f67-9868-fdd6544a659e'. Below the header is a light blue box with a dark blue bar at the top that says 'Personal Documentation'. Inside this box, there are three upload buttons: 'DNI FRONT (JPG)' with a file icon, 'DNI BACK (JPG)' with a file icon, and 'BIRTH CERTIFICATE (PDF)' with a file icon. Below these buttons is a dark blue bar with the text 'SHARE PERSONAL DOCUMENTATION'.

5.11) AvoDocuments

Esta página permite al usuario **cargar la documentación relacionada con el AVO**, garantizando que los documentos necesarios sean registrados de manera adecuada para su posterior revisión y procesamiento.

65a8f4d6-e597-4f67-9868-fdd6544a659e

AVO documentation

Is dead? ☒ Yes ☐ No

Was he in a marital relationship? ☒ Yes ☐ No

MARRIAGE CERTIFICATE (.PDF)

BIRTH CERTIFICATE (.PDF)

UPLOAD AVO DOCUMENTS

5.12) AncestorsDocumentation

Esta página permite al usuario **ingresar la documentación correspondiente a todos los ancestros**. El proceso comienza con la indicación de la cantidad de ancestros, seguido por la carga de la documentación obligatoria requerida para cada uno de ellos.

65a8f4d6-e597-4f67-9868-fdd6544a659e

How many ancestors do you have until your AVO?

**not including your AVO*

Ancestors count

LOAD NUMBER OF ANCESTORS ->

65a8f4d6-e597-4f67-9868-fdd6544a659e

Ancestors Documentation

Is dead? ☐ Si ☒ No

Was in a marital relationship?? ☐ Si ☒ No

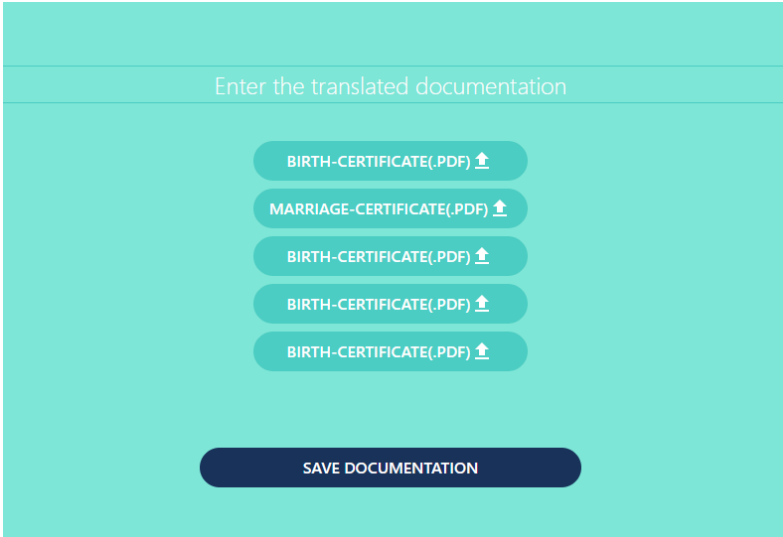
BIRTH CERTIFICATE (.PDF)

BIRTH CERTIFICATE (.PDF)

SAVE DOCUMENTATION

5.13) TranslatedDocumentation

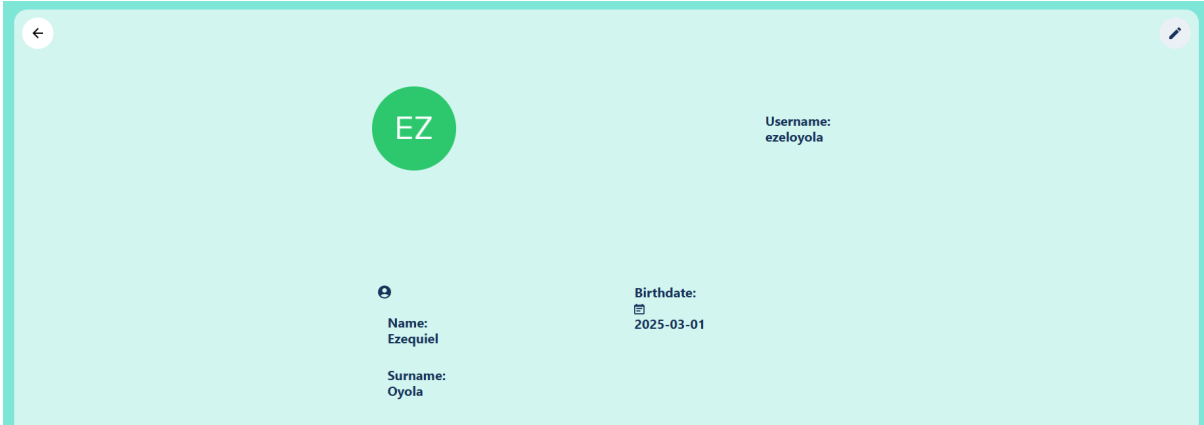
Esta página permite al usuario **ingresar la documentación traducida** asociada a su trámite.



The screenshot shows a light blue interface for uploading translated documentation. At the top, a header bar contains the text "Enter the translated documentation". Below this, there are five identical buttons stacked vertically, each labeled "BIRTH-CERTIFICATE(.PDF)" with a small upload icon. At the bottom of the interface is a dark blue button labeled "SAVE DOCUMENTATION".

5.14) UserProfile

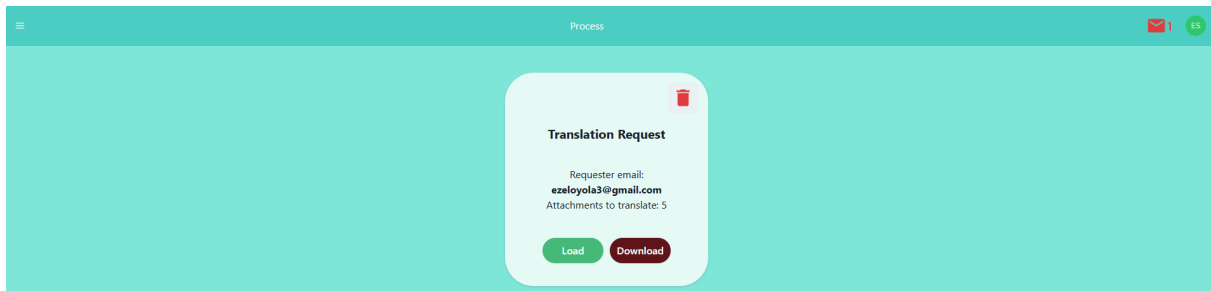
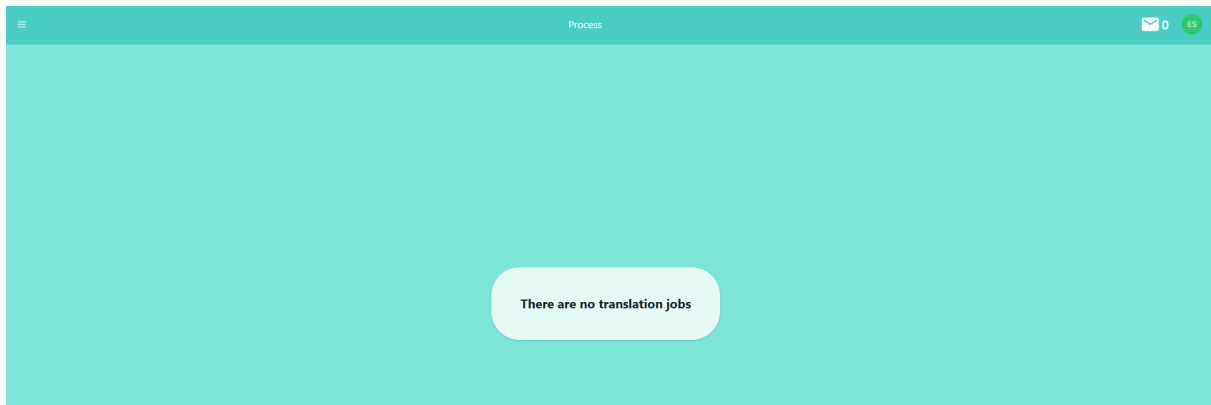
Esta página muestra la **información de perfil** del usuario, la cual puede ser **editada** según sea necesario.



The screenshot displays a user profile page with a light blue background. At the top left is a back arrow icon, and at the top right is an edit icon. The profile information is organized into two columns. The left column features a green circular avatar with the letters "EZ", followed by a user icon, and then the fields "Name: Ezequiel" and "Surname: Oyola". The right column shows the "Username: ezeloyola" and the "Birthdate: 2025-03-01" with a calendar icon.

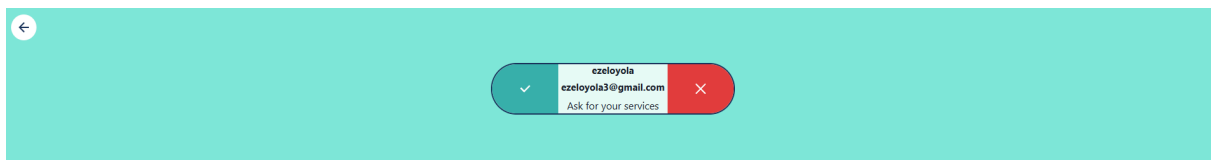
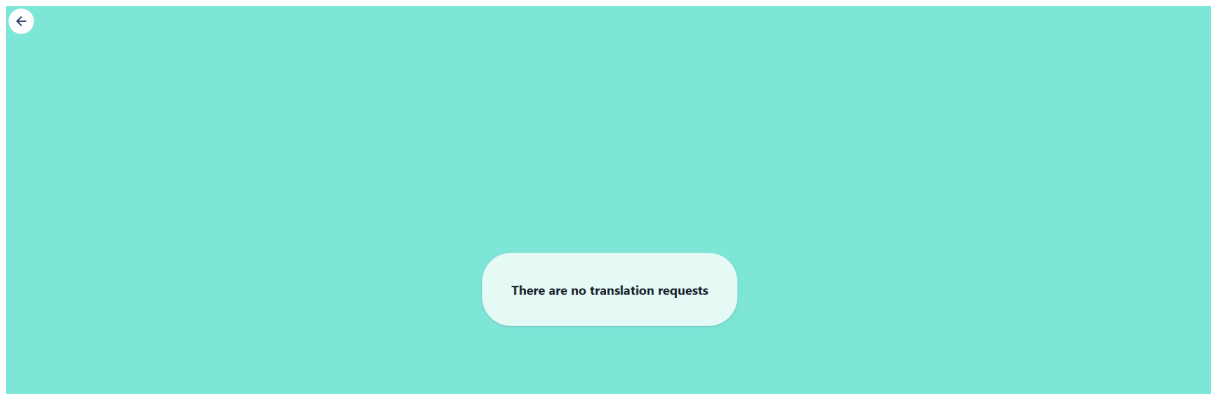
5.15) UserHome (traductor)

Esta página corresponde al **Home** del traductor, donde se muestran de manera centralizada las **tareas de traducción** asignadas.



5.16) TranslationRequests

En esta página se visualizan todas las **solicitudes de traducción** realizadas por los usuarios.



6) Configuración de base de datos (local)

La base de datos utilizada es **PostgreSQL**, versión 15-alpine. Para facilitar la visualización y gestión de la base de datos, se emplea **pgAdmin 4**.

La configuración de la base de datos se realiza a través del archivo **docker-compose.yml**, ubicado en la carpeta **backend**:

```
version: '3.8'
services:
  db:
    image: postgres:15-alpine
    container_name: tramitarte_sql
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    ports:
      - '5432:5432'
    volumes:
      - db:/var/lib/postgresql/data
      - ./Docker/init.sh:/docker-entrypoint-initdb.d/01_init.sh

  pgadmin:
    image: dpage/pgadmin4
    container_name: pgadmin4_container_tramitarte
    restart: always
    ports:
      - "5050:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@phm.edu.ar
      PGADMIN_DEFAULT_PASSWORD: admin
    volumes:
      - pgadmin-data:/var/lib/pgadmin

volumes:
  db:
  pgadmin-data:
```

Para crear la base de datos utilizada en el proyecto, se indica a Docker que ejecute el archivo **init.sh**, ubicado en la carpeta **Docker**.

```
set -e

psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname
"$POSTGRES_DB" <<-EOSQL
    CREATE DATABASE "TramitarteApp";
    CREATE USER docker WITH ENCRYPTED PASSWORD 'docker';
    GRANT ALL PRIVILEGES ON DATABASE "TramitarteApp" TO docker;
EOSQL
```

Finalmente, se configura la URL de la base de datos y la sesión correspondiente, que será utilizada por los servicios y endpoints, dentro de la carpeta **database**:

```
URL_DATABASE =
"postgresql://postgres:postgres@localhost:5432/TramitarteApp"

engine = create_engine(URL_DATABASE)

SessionLocal = sessionmaker(autoflush=False, bind=engine)

Base = declarative_base()
```

7) Configuración de imagen de Docker

La configuración de la base de datos es la misma que en el entorno local. Además, se agregan las configuraciones necesarias para el correcto funcionamiento del backend y el frontend:

```
version: '3.8'

services:
  db:
    image: postgres:15-alpine
    container_name: tramitarte_postgres
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    ports:
      - '5432:5432'
    volumes:
```

```
- db:/var/lib/postgresql/data
- ./Docker/init.sh:/docker-entrypoint-initdb.d/01_init.sh

pgadmin:
  image: dpage/pgadmin4
  container_name: pgadmin_container_tramitarte
  restart: always
  ports:
    - "5050:80"
  environment:
    PGADMIN_DEFAULT_EMAIL: admin@phm.edu.ar
    PGADMIN_DEFAULT_PASSWORD: admin
  volumes:
    - pgadmin-data:/var/lib/pgadmin

backend:
  build:
    context: ./backend
  container_name: backend_tramitarte
  ports:
    - "8000:8000"
  depends_on:
    - db
  environment:
    -
DATABASE_URL=postgresql://postgres:postgres@db:5432/TramitarteApp
  volumes:
    - ./backend:/app

frontend:
  build:
    context: ./frontend
  container_name: frontend_tramitarte
  ports:
    - "3000:3000"
  depends_on:
    - backend
  volumes:
    - ./frontend:/app

volumes:
  db:
  pgadmin-data:
```

El backend requiere que la base de datos esté en funcionamiento antes de su inicio. De la misma manera, el frontend depende de que el backend esté operativo para su correcto funcionamiento.

Para generar la imagen del backend, se utiliza la siguiente configuración en el archivo **Dockerfile**:

```
# Backend Dockerfile

# Python image
FROM python:3.9-slim

# Upgrade pip
RUN pip install --upgrade pip

# Install tesseract
RUN apt-get update && \
    apt-get install -y tesseract-ocr libtesseract-dev

# Copy the train data into the tessdata directory of tesseract
COPY ./tesseract_data/ /usr/share/tesseract-ocr/5/tessdata/

# Work directory
WORKDIR /app

# Copy dependencies file
COPY requirements.txt /app

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy app code
COPY . /app

# Expose port
EXPOSE 8000

# Execute app
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Además de instalar las librerías indicadas en el archivo **requirements.txt**, es necesario instalar los paquetes de Tesseract. Asimismo, se deben colocar en la carpeta **tessdata** los archivos de entrenamiento correspondientes a los idiomas requeridos para su correcto funcionamiento.

También se le debe cambiar el puerto el **localhost** de la configuración de la base de datos a **db** para que el backend pueda funcionar:

```
URL_DATABASE = "postgresql://postgres:postgres@db:5432/TramitarteApp"

engine = create_engine(URL_DATABASE)

SessionLocal = sessionmaker(autoflush=False, bind=engine)

Base = declarative_base()
```

Para poder generar la imagen del frontend se configura el siguiente Dockerfile:

```
# Frontend Dockerfile

# Node.js image
FROM node:16

# Work directory
WORKDIR /app

# Copy dependencies files
COPY package.json package-lock.json /app/

# Install dependencies
RUN npm install

# Copy app code
COPY . /app/

COPY .env.development /app/

# Build React app
RUN npm run build

# Expose port
EXPOSE 3000

# Execute app
CMD ["npm", "start"]
```

Su implementación es más simple que la del frontend, ya que solo requiere la instalación de todos los paquetes especificados en los archivos JSON.

8)Diagrama de clases

El diagrama de clases se encuentra almacenado como una imagen titulada "Tramitarte Class Diagram" en la carpeta **documentation**.

9)Requisitos de instalación

Backend (local)

1) Instalar las Librerías Necesarias

Para asegurar que el backend funcione correctamente, instale las librerías listadas en el archivo **requirements.txt** ejecutando el siguiente comando:

- `pip install -r requirements.txt`

2) Inicializar la Base de Datos

Una vez que las librerías estén instaladas, inicialice la base de datos configurada en el archivo **docker-compose.yml** con el siguiente comando:

- `docker-compose up`

3) Conexión a la Base de Datos desde pgAdmin

Para conectarse a la base de datos mediante pgAdmin, utilice los siguientes detalles:

- **Host name:** db
- **Usuario:** postgres
- **Contraseña:** postgres

4) Configurar el Servicio de Tesseract

Para que el servicio de Tesseract funcione correctamente descargue el archivo de entrenamiento para el idioma **italiano** (ita.traindata) desde el siguiente enlace:

- <https://github.com/tesseract-ocr/tessdata>

Copie el archivo descargado y ponlo en la carpeta **tessdata** dentro del directorio **Tesseract** de su máquina.

5) Iniciar el Backend

Finalmente, en el directorio **backend**, inicie el backend con el siguiente comando:

- `uvicorn main:app --reload`

Frontend (local)

1) Instalar los Paquetes Necesarios

Para instalar todos los paquetes requeridos para ejecutar la aplicación, ejecute el siguiente comando:

- `npm i`

2) Iniciar el Frontend

Después de instalar los paquetes, inicie el frontend dentro de la carpeta **frontend** utilizando el siguiente comando:

- `npm start`

Imagen del proyecto

1) Generar las imágenes

Para generar todas las imágenes del proyecto se debe ejecutar el siguiente comando:

- `docker-compose build`

2) Iniciar el contenedor

Después de generar todas las imágenes, debe prender el contenedor con el siguiente comando:

- `docker-compose up`