
# Integração de Sistemas de Informação

**Autores:**

Diogo Bernardo **21144**
João Ribeiro **23795**
Edgar Baptista **16447**

**Professor**:

Luís Ferreira

*Submission of the project to the course*
*Engenharia de Sistemas Informáticos*

*ESI*

December 31, 2023

# *Synopsis*

**IPCA**

Escola superior de Tecnologia

**Engenharia de Sistemas Informáticos**

**Integração de Sistemas de Informação**

by

Diogo Bernardo **21144**
João Ribeiro **23795**
Edgar Baptista **16447**

# Contents

# List of Figures

**Chapter 1**

# Introduction

This project for the course unit ISI, proposed by Luís Ferreira, 2023, has a focus on exploring and developing processess of interoperability between systems, based on web services. It is intended a library of new services (SOAP, RESTful), complemented with the reutilization of existing external services.

It is also intended:
Develop, test and document a set of new services to enrich an API;
Develop a client application that demonstrates the applicability of this API.

The general objectives of this project are the following:

1. Consolidate ISI concepts using web services;
2. Design system integration architectures, using interoperability APIs;
3. Explore tools that support the development of web services;
4. Explore new Technologies, Frameworks or Paradigms for implementing web services (SOAP and RESTful);
5. Enhance the experience in application development;
6. Assimilate content from the course unit ISI.

Chapter 2

# Context

## 2.1 Project Theme

We plan to build a service that allows a user, through a mobile application, the creation of his ideal home or preview changes that he may want to do in his housing space. The user can upload his own photos, or generate and edit an image of a house interior or exterior with the help of an A.I model.
Our target audience are those who wish to see their own ideas materialize into reality, those who wish for inspiration, amateurs or professionals, or any individual who just wishes to experiment with random creations for fun. This project will be a subscription based A.I.service with multiple plans that allow for the creation and editing of images related to housing zones. It will be controlled by a simple interface that allows the user to write text prompts to edit and generate images. It will be possible to use images provided from the user's gallery or a photo directly from the camera, everything created or edited will be saved in an image history directly in the app.
To use the text based prompts, English is the only language supported but this may be expanded in the future. As far as the time required per edit or generation, it may vary depending on the complexity of the prompt or the server load, but it should take between 20sec and 20min per use. Every image generated or edited will be able to receive a rating and, if set to public, other users can also rate them. The best rated images will be highlighted on the front page.

## 2.2 Problems to Solve

List of possible Challenges:

- **Expression of needs:** Our users may find it difficult to express what they want and obtaining unwanted/useless results.
- **Optimization of the app:** Our integration with the A.I. model may be inefficient and thus resulting in a slow and clunky mobile application.
- **Complex interface:** A complex and overcrowded interface may result in a negative and difficult experience for the users, resulting in a loss of value for both parties, the user and our company.

## 2.3 Project Objectives

The main objective of this project is to create a mobile application that integrates an already existing A.I. model, allowing for our users and clients the ability to generate and edit custom images of housing spaces. More specifically, the project aims to reach the following objectives:

1. **Integration of an existing model:** Integrate, in an effective manner, the A.I. model with a mobile application.
2. **Interface:** Build an easy to use interface that allows the users to express their needs as well as receive and show images in a simple manner.
3. **Optimization for mobile devices:** Guarantee that the application runs and works smoothly in the majority of mobile devices.

Chapter 3

# Architecture of the Developed Solution

The Housify system has been designed to facilitate seamless communication between clients and various backend services.

- **WAN Communication:** Clients can simultaneously interact with the Housify system over the WAN using Housify's mobile application, using TCP and UDP protocols, ensuring efficient and reliable data transmission.
- **Main Router (pfSense) and Firewall:** At the network's edge, the system employs pfSense as the Main Router and Firewall to provide robust network security and management. This component handles IP filtering and integrates seamlessly with Cloudflare solutions for enhanced security and performance.
- **Main Reverse Proxy (nginx):** The Main Reverse Proxy, powered by nginx, acts as an intermediary between clients and the backend services. It is responsible for terminating Transport Layer Security (TLS/SSL), supporting multiple HTTP versions (HTTP/1.1, HTTP/2, HTTP/3), and facilitating both UDP and TCP communication.
- **Load Balancer (nginx):** Situated behind the Main Reverse Proxy, the Load Balancer efficiently distributes network requests based on predefined criteria. It ensures optimal resource allocation by balancing the traffic load across available servers.
- **Web Servers:** Web servers, designed for low computational requirements, are responsible for establishing and maintaining client communication. These servers employ the ASGI (Asynchronous Server Gateway Interface) using Uvicorn and FastAPI, facilitating both RESTful API and WebSocket communication. They manage user data and service usage history efficiently.
- **AI Inference Load Balancer:** To cater to AI-driven services, an AI Inference Load Balancer optimizes the allocation of inference requests to AI servers. This component helps ensure quick response times and efficient resource utilization.
- **AI Servers:** The AI servers are organized in clusters and dedicated to performing AI inferences based on client requests. They utilize a microservices architecture and communicate with web servers through an internal API. The servers are equipped with FastAPI, Uvicorn, and TortoiseORM for seamless integration and efficient data management.
- **Diffusers and LammaCPP:** These components play a crucial role in the AI inference process. Diffusers and LammaCPP are used for handling inference requests and processing AI-related tasks efficiently.

- **Databases and File Storage:** Data storage is a critical aspect of the Housify system. This section encompasses user management databases, AI services databases, and a storage server for images and logs. Redis is utilized as an in-memory data store for enhancing data retrieval speed and efficiency.
- **AI Models:** AI models are efficiently stored in a super fast file server to be used for inference tasks. The models are stored and managed to meet the technical demands of the system.

Housify system's technical architecture is designed with a focus on security, scalability, and efficiency. It employs a combination of networking components, web servers, load balancers, AI inference capabilities, and data management systems to ensure seamless communication and AI services for clients. This architecture is well-suited for modern applications requiring real-time communication and AI-driven functionalities.

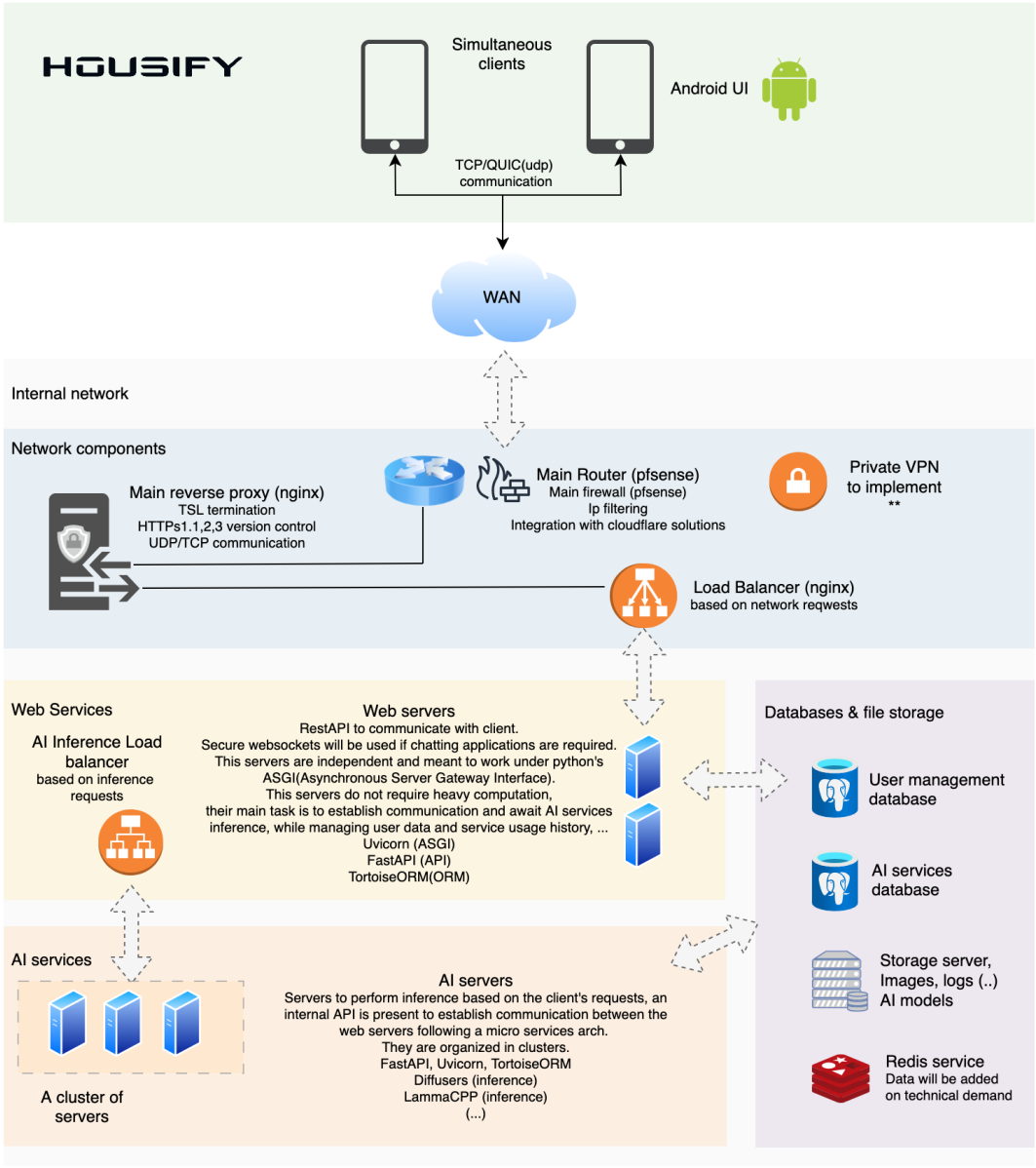


FIGURE 3.1: Figure of the Technical Architecture of the Developed Solution

Chapter 4

# Housify System

## Housify Project

In this section, we outline the different systems implemented to construct our solution, following a microservices architecture. This encompasses independent projects, primarily web services (SOAP, AI services), and a web server with business rules. By adopting this approach, our system harnesses the advantages of having distinct projects, keeping scalability and maintainability in mind.

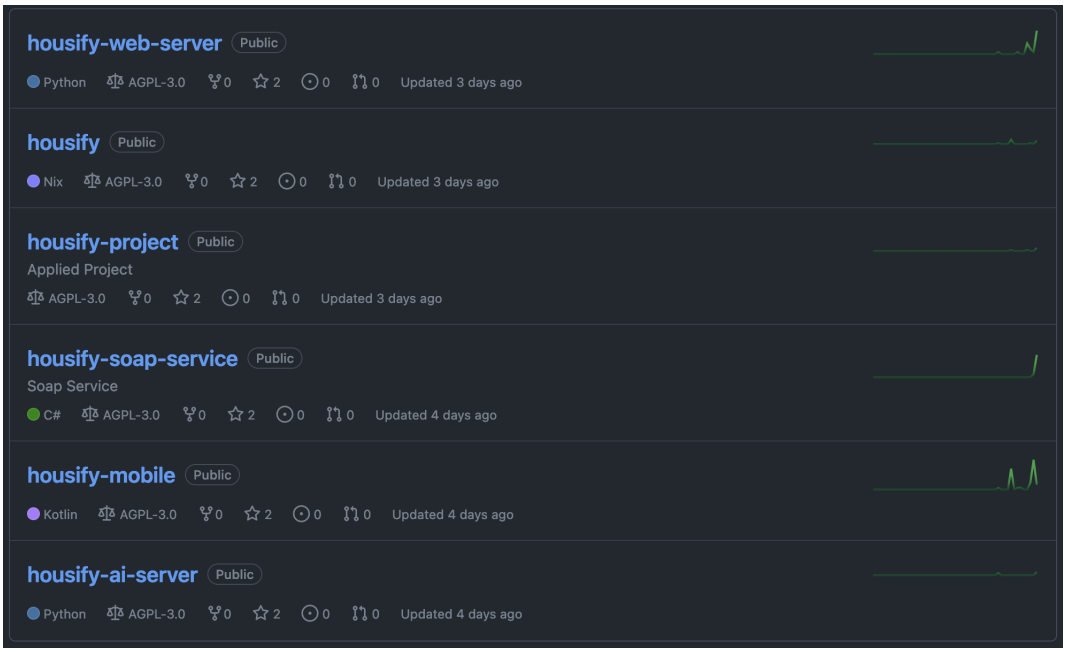


FIGURE 4.1: Github project

## 4.1 Web Server Project

This project provides the core services for user management and business rules. The subsystem within this project is responsible for managing all user-related tasks and appropriately calling underlying services that the user may need, such as AI services or SOAP services

## 4.2 AI server Project

This project serves as the brain of the system. Within this service, we process requests for image processing and analysis inference.

## 4.3 Mobile Project

This project represents the user interface accessible to the client through a mobile application, currently limited to Android. This application enables clients to integrate device capabilities (e.g., camera) to interact with our system, making image processing tasks in the housing context more intuitive and user-friendly.

## 4.4 SOAP service Project

This project represents a service designed with a different architecture in mind, specifically meant to operate under the SOAP protocol. The service is not directly accessible to the user. Instead, the client communicates with the REST service, and the web server handling the tasks will then invoke the SOAP service if required. This service serves as an example of financial cryptocurrency conversion (e.g., BTC to USD) and relies on the Binance API as an external web service.

## 4.5 Housify Project

In this project we have our legal documents, all the planning, business rules, SCRUM, report, etc.

## 4.6 Housify

In this project, we have numerous configuration files for development. The Nix language was utilized as our servers run on Linux with NixOS 23.11. Nix enables declarative builds and deployments. Additionally, an email server was hosted with the help of mail-docker.

Chapter 5

# Development

## 5.1 https://housify.geniadynamics.org/

All of our services are accesabes through housify.geniadynamics.org.


FIGURE 5.1: Page image of our domain

## 5.2 Documentation of the Developed Solution

### 5.2.1 Open API

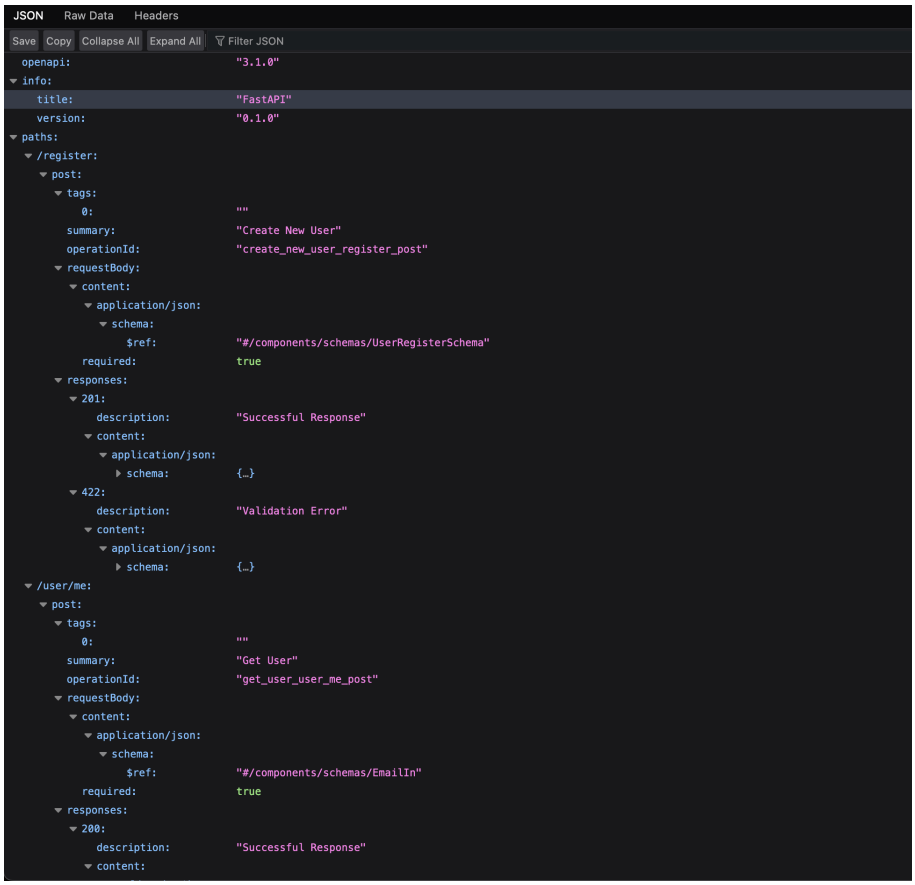


FIGURE 5.2: Page image of Open API

### 5.2.2 Swagger UI

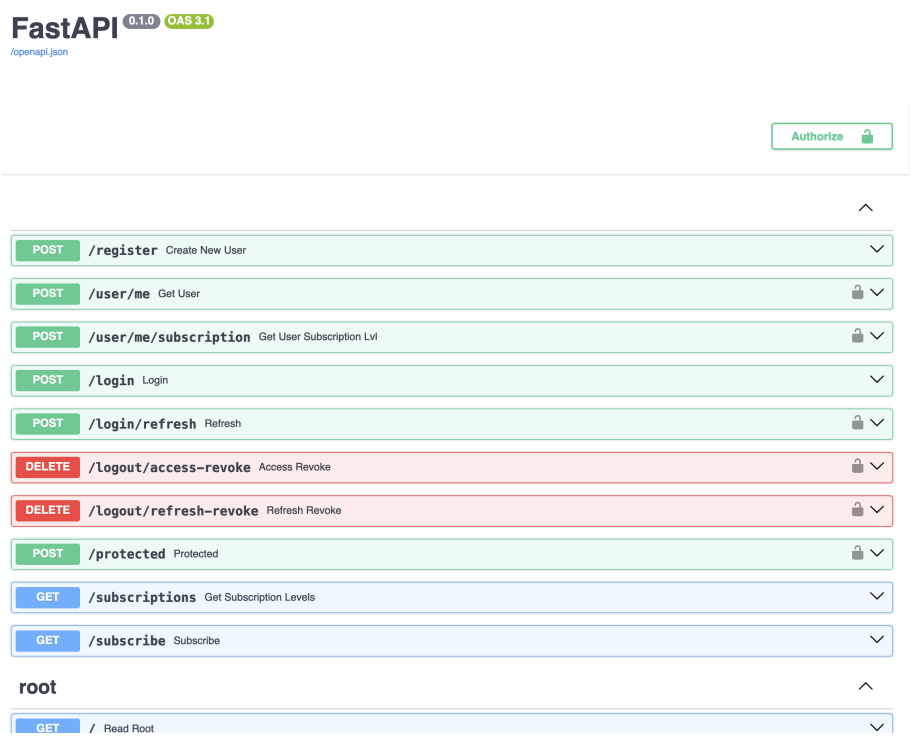


FIGURE 5.3: Page image of Swagger UI

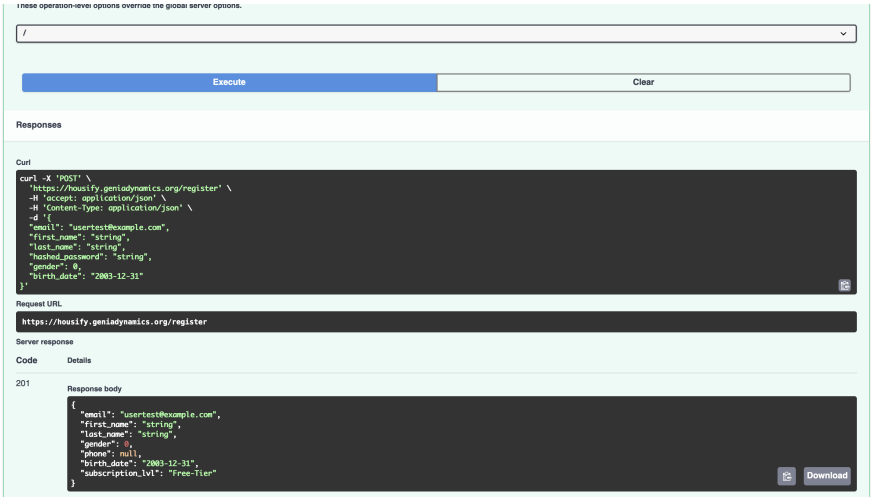


FIGURE 5.4: Page image of Swagger register

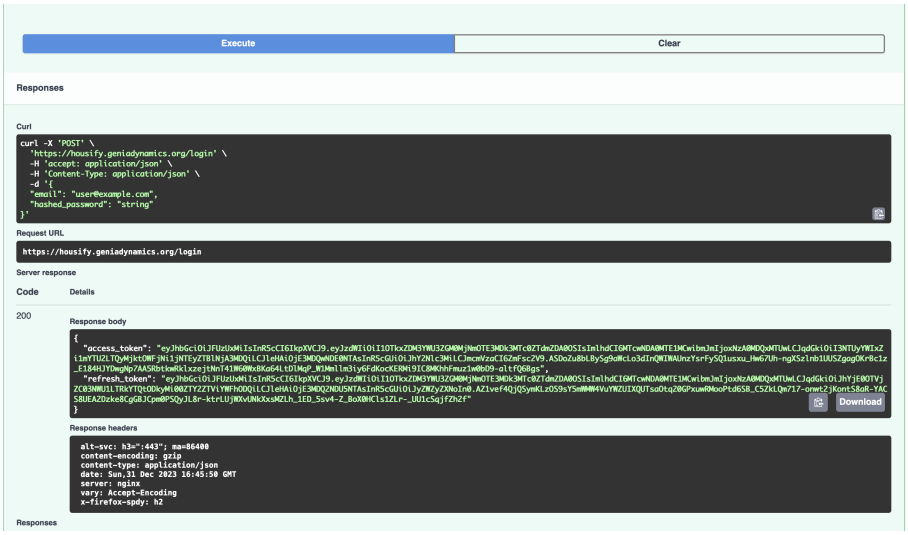


Figure 5.5: Page image of Swagger login

### 5.2.3 Redocly

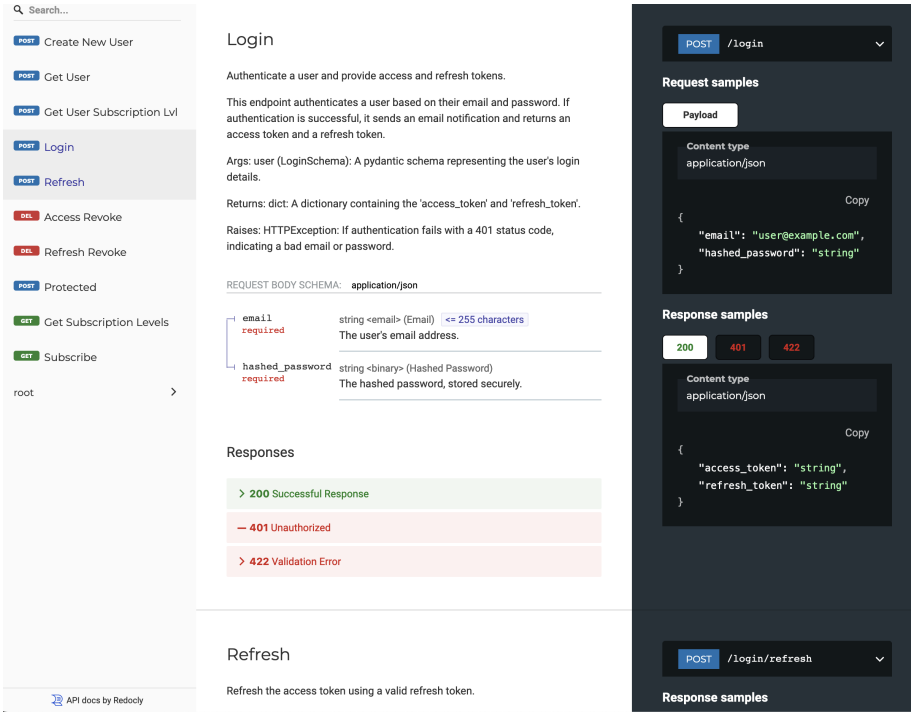


Figure 5.6: Page image of Redocly login

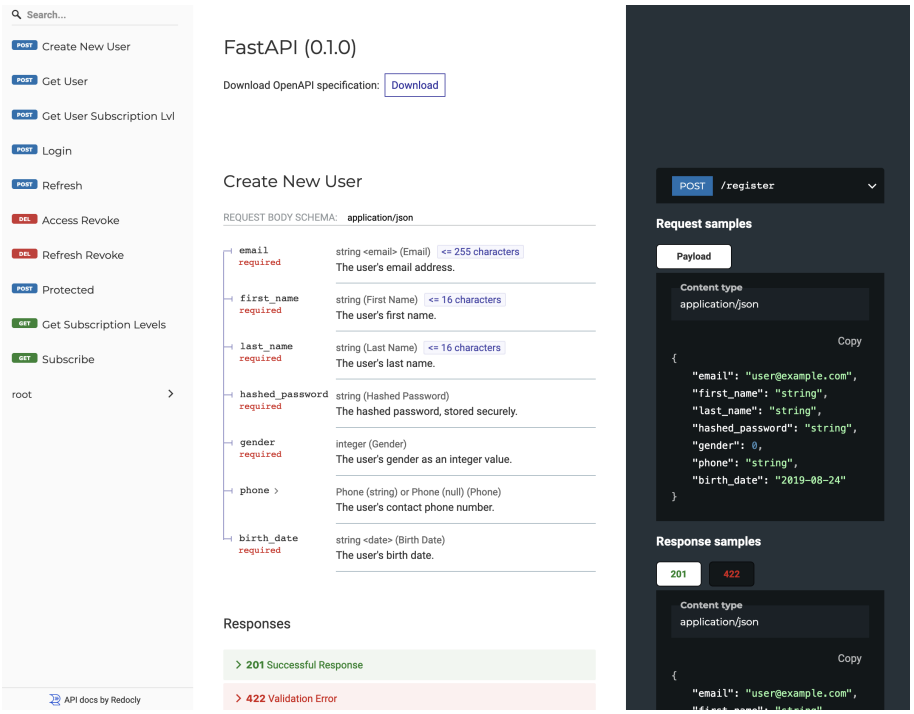


FIGURE 5.7: Page image of Redocly register

### 5.2.4 Code Documentation

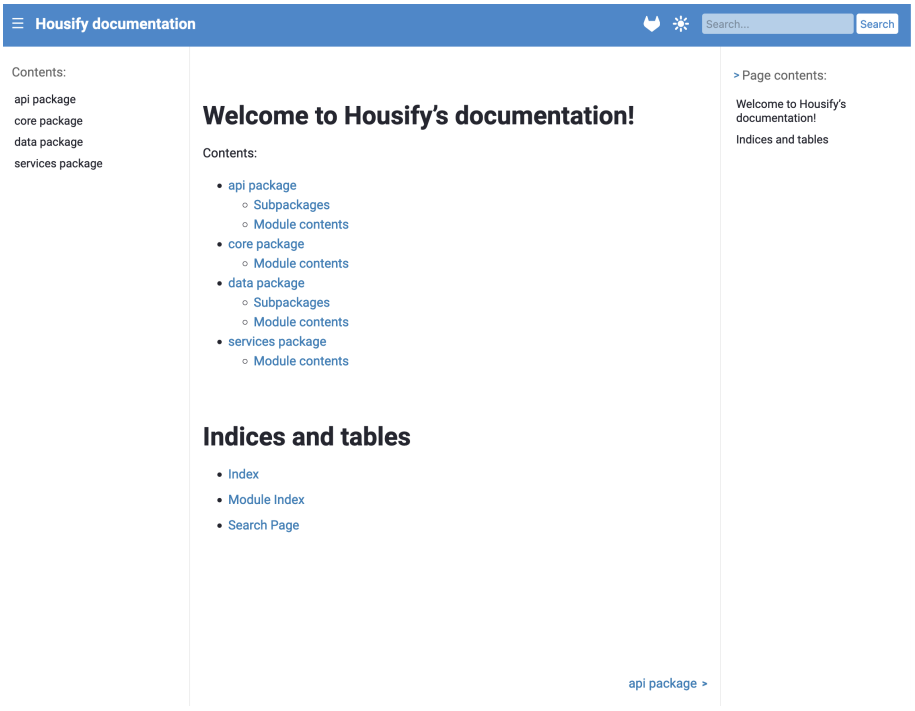


FIGURE 5.8: Page image of Code Documentation (part1)

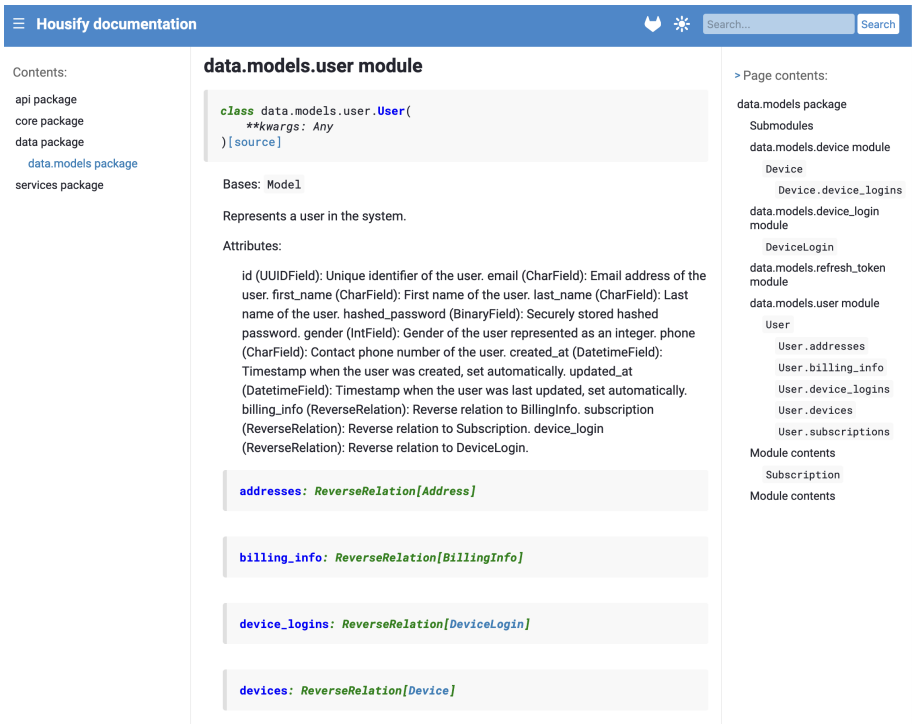


FIGURE 5.9: Page image of Code Documentation (part2)

## 5.3 Email Server

Example of an email sent when a customer logs into the application.

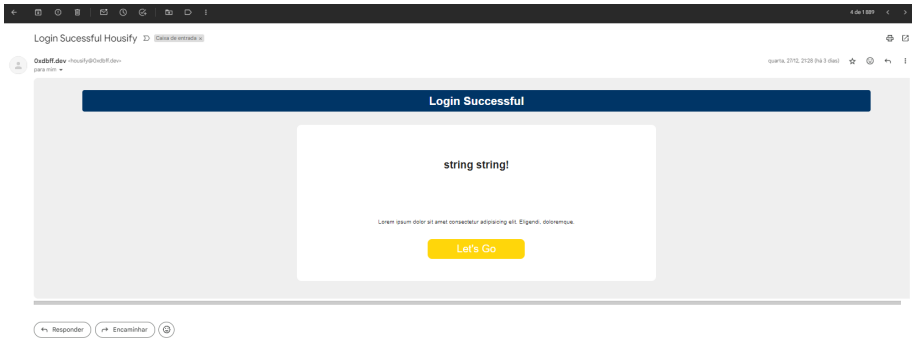


FIGURE 5.10: Figure of the example email

## 5.4 SOAP Service

### 5.4.1 What is SOAP

The *Simple Object Access Protocol* n.d. (SOAP) is a way to pass information between applications in an XML format.

SOAP messages are transmitted from the sending application to the receiving application, typically over an HTTP session. The actual SOAP message is made up of the Envelope element, which contains a Body element and an optional Header element.

- **Envelope.** This mandatory element is the root of the SOAP message, identifying the transmitted XML as being a SOAP packet. An envelope contains a body section and an optional header section.
- **Header.** This optional element provides an extension mechanism indicating processing information for the message. For example, if the operation using the message requires security credentials, those credentials should be part of the envelope header.
- **Body.** This element contains the message payload, the raw data being transmitted between the sending and receiving applications. The body itself may consist of multiple child elements, with an XML schema typically defining the structure of this data.

### 5.4.2 Interface

```
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Threading.Tasks;

/// <summary>
/// Service contract for retrieving cryptocurrency rates.
/// </summary>
[ServiceContract]
public interface ICryptoRateService
{
    /// <summary>
    /// Retrieves the rate of a specific cryptocurrency.
    /// </summary>
    /// <param name="symbol">The symbol of the desired cryptocurrency.</param>
    /// <returns>A Task representing the asynchronous operation that returns a
    CryptoRateResponse object.</returns>
    [OperationContract]
    Task<CryptoRateResponse> GetCryptoRate(string symbol);

    /// <summary>
    /// Retrieves all cryptocurrency rates.
    /// </summary>
    /// <returns>A Task representing the asynchronous operation that returns a list
    of CryptoRateResponse objects.</returns>
    [OperationContract]
    Task<List<CryptoRateResponse>> GetAllCryptoRates();
}

/// <summary>
/// Represents the response containing cryptocurrency rate information.
/// </summary>
[DataContract(Namespace = "http://localhost:5000")]
public class CryptoRateResponse
{
```

```
    /// <summary>
    /// Gets or sets the symbol of the cryptocurrency.
    /// </summary>
    [DataMember]
    public string Symbol { get; set; } = "";

    /// <summary>
    /// Gets or sets the price of the cryptocurrency.
    /// </summary>
    [DataMember]
    public decimal Price { get; set; }
}
```

### 5.4.3 Implementation

```
/// <summary>
/// Service class for retrieving cryptocurrency rates from the Binance API and
    interacting with a database.
/// </summary>
public class CryptoRateService : ICryptoRateService
{
    // Base URL for the Binance API to obtain cryptocurrency rate information.
    private const string BinanceApiBaseUrl = "https://api.binance.com/api/v3/ticker/
    price?symbol=";

    /// <summary>
    /// Retrieves the rate of a specific cryptocurrency from the Binance API and
    stores it in the database.
    /// </summary>
    /// <param name="symbol">The symbol of the desired cryptocurrency.</param>
    /// <returns>A CryptoRateResponse object containing the symbol and price of the
    cryptocurrency.</returns>
    public async Task<CryptoRateResponse> GetCryptoRate(string symbol)
    {
        // Creates an instance of HttpClient to make the call to the Binance API.
        using (var httpClient = new HttpClient())
        {
            // Gets the API response in JSON format.
            var response = await httpClient.GetStringAsync(BinanceApiBaseUrl +
    symbol);

            // Deserializes the JSON response into a dynamic object.
            var data = JsonConvert.DeserializeObject<dynamic>(response);

            // Inserts the information into the database.
            await Data.DataBase.CmdExecuteNonQueryAsync(
                $"INSERT INTO public.binance_data (symbol, price) " +
                $"VALUES ('{symbol}', {data.price})"
            );

            // Returns a CryptoRateResponse object with the symbol and price.
            return new CryptoRateResponse
            {
                Symbol = symbol,
                Price = data.price
            };
```

```
        }
    }

    /// <summary>
    /// Retrieves all cryptocurrency rates stored in the database.
    /// </summary>
    /// <returns>A list of CryptoRateResponse objects containing symbols and prices
    of all stored cryptocurrencies.</returns>
    public async Task<List<CryptoRateResponse>> GetAllCryptoRates()
    {
        var cryptoRates = new List<CryptoRateResponse>();

        // Executes the query to obtain all rates from the database.
        var queryResult = await Data.DataBase.CmdExecuteQueryAsync(
            $"SELECT symbol, price FROM public.binance_data");

        // If there are query results, iterates over each row and adds it to the
    list.
        if (queryResult != null)
        {
            foreach (var row in queryResult)
            {
                var symbol = row[0]?.ToString() ?? string.Empty;
                var price = row[1] != null ? Convert.ToDecimal(row[1]) : 0;

                cryptoRates.Add(new CryptoRateResponse
                {
                    Symbol = symbol,
                    Price = price
                });
            }
        }

        // Returns the list of cryptocurrency rates.
        return cryptoRates;
    }
}
```

### 5.4.4 SOAP in context

In the context of this practical work, the development of SOAP services was undertaken to provide a layer of interoperability between systems. These services were designed to offer specific functionalities, ensuring effective and standardized communication. The implementation of SOAP services involved the clear definition of available operations, the structure of XML messages, and appropriate data handling.

## 5.5 Database Operations

### 5.5.1 Non Query

```
/// <summary>
```

```
///     Initialize a database connection as a postgres user, and
///     Execute a command that returns no queries, only the number of
///     rows altered.
///     Connections are disposed when they are no longer needed.
/// </summary>
/// <param name="sql">sql commands. </param>
/// <returns> The number of rows altered. </returns>
public static async Task<int> CmdExecuteNonQueryAsync(string? @sql)
{
    try
    {
        // Open a connection that will live through the execution of this method's
        // stack frame.
        var dataSourceBuilder = new NpgsqlDataSourceBuilder(ConnString);
        await using var dataSource = dataSourceBuilder.Build();

        // Execute command(s) in the dbms and await results.
        await using var cmd = dataSource.CreateCommand(sql);
        return (await cmd.ExecuteNonQueryAsync());
    }
    catch (NpgsqlException e)
    {
        Log.Error(e);
        return default;
    }
}
```

### 5.5.2 Query

```
/// <summary>
///     Initialize a database connection as a postgres user, and
///     Execute a command that returns data. Every line is a value in a list,
///     while every column is a Dictionary with key value pairs.
///     Connections are disposed when they are no longer needed.
/// </summary>
/// <param name="sql">sql commands.</param>
/// <returns> Data from the query. </returns>
public static async Task<List<Dictionary<int, object?>>?>
CmdExecuteQueryAsync(string? @sql)
{
    try
    {
        // Open a connection that will live through the execution of this method's
        // stack frame.
        var dataSourceBuilder = new NpgsqlDataSourceBuilder(ConnString);
        await using var dataSource = dataSourceBuilder.Build();

        await using var cmd = dataSource.CreateCommand(sql);
        await using var reader = await cmd.ExecuteReaderAsync();

        // A List of Dictionary with key value pairs, to hold return values
        // from a query.
        var values = new List<Dictionary<int, object?>>();

        while (await reader.ReadAsync())
        {
```

```
            // A generic Dictionary to hold all the columns from the Table.
            var columns = new Dictionary<int, object?>();

            foreach (var currentField in Enumerable.Range(0, reader.FieldCount))
                columns.Add(currentField, reader.GetValue(currentField));

            values.Add(columns);
        }

        return values;
    }
    catch (NpgsqlException e)
    {
        Log.Error(e);
        return default;
    }
}
```

### 5.5.3 Initialize DataBase

```
/// <summary>
///     Initialize a database connection and ensure there are no connection
///     isues. Also validate Data model.
///     Create database and tables on Failure, also load backups if there are
///     any.
/// </summary>
/// <returns> An awaitable Task </returns>
public static async Task Init()
{
    try
    {
        // Test if there is a database with {DbName}
        var queryReturn = await AdminCmdExecuteQueryAsync<string>(
            $"SELECT datname FROM pg_catalog.pg_database " +
            $"WHERE lower(datname) = lower('{DbName}')");

        if (queryReturn == DbName.ToLower())
        {

            await EnsureDataBaseTables();

            return;
        }

        throw new NpgsqlException($"Database {DbName} does not exist");
    }
    catch (NpgsqlException e)
    {
        Log.Error(e);
        Log.Warn(
            $"Proceeding without database '{DbName}', no values are present...");
        Log.Warn($"Creating a new one with name '{DbName}' as user {User}.");

        // Create Database.
        await CreateDatabaseAsync();
```

```
        // Create Tables
        await EnsureDataBaseTables();

        // Try Loading database backups if there are any.
        Log.Warn($"Attempting to load DataBase backups");
        //! TODO
    }
    catch (Exception e)
    {
        Log.Error(e);
        Log.Error("Cannot proceed, exiting program with exit code 1");

        // Exit the executable with an error code.
        Environment.Exit(1);
    }
}
```

## 5.6 RESTful Services

### 5.6.1 What is REST

*REST, or REpresentational State Transfer* n.d., is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other. REST-compliant systems, often called RESTful systems, are characterized by how they are stateless and separate the concerns of client and server.

### 5.6.2 REST in Context

RESTful web services are built around the transfer of representations of resources. These resources are manipulated using a standard set of operations, often corresponding to HTTP methods.

### 5.6.3 GET

The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.

```
@router.get("/subscriptions", response_model=list[SubscriptionLvlSchema])
async def get_subscription_levels():
    """
    Get a list of subscription levels.

    Returns:
        List[SubscriptionLvlSchema]: A list of subscription levels.

    Example:
```

```
        ```python
        response = test_client.get("/subscriptions")
        assert response.status_code == 200

        subscription_levels = response.json()
        assert len(subscription_levels) > 0
        ```
    """
    return await SubscriptionLvl.all()
```

### 5.6.4 POST

POST is used to send data to a server to create or update a resource. The data sent to the server with POST is stored in the request body of the HTTP request.

```
@router.post(
    "/user/me",
    response_model=UserSchema,
    dependencies=[Depends(ui_auth_rule)]
)
async def get_user(data: EmailIn, Authorize: AuthJWT = Depends()):
    """
    Get user information based on the provided email address.

    Args:
        data (EmailIn): The email address of the user to retrieve.
        Authorize (AuthJWT): The JWT authorization dependency.

    Raises:
        HTTPException: If the user is not found, returns a 404 status code with a "
    User not found" detail.

    Returns:
        UserSchema: The user information.

    Example:
        ```python
        from your_project.models import User
        from your_project.schemas import EmailIn

        email_data = EmailIn(email="user@example.com")

        response = test_client.post("/user/me", json=email_data)
        assert response.status_code == 200
        assert response.json()["email"] == "user@example.com"
        ```
    """
    await Authorize.jwt_required()
    user = await User.get(email=data.email)
    if user is None:
        raise HTTPException(status_code=404, detail="User not found")
    return user
```

### 5.6.5 PUT

PUT is used to send data to a server to create or replace a resource. The difference between POST and PUT is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result.

### 5.6.6 PATCH

PATCH is used for modifying resources. The PATCH request only needs to contain the changes to the resource, not the complete resource. This is more efficient and secure than PUT which requires sending the entire updated entity.

### 5.6.7 DELETE

DELETE is used to delete a resource identified by a URI.

```
    @router.delete("/logout/access-revoke", dependencies=[Depends(ui_auth_rule)])
async def access_revoke(
    Authorize: AuthJWT = Depends(), redis_client=Depends(get_redis_client)
):
    """
    Revoke an access token.

    This endpoint revokes the current user's access token, adding it to a denylist
    in Redis.

    Returns:
        dict: A dictionary with a message indicating that the access token has been
    revoked.

    Raises:
        HTTPException: If the access token is invalid, with a 401 status code.
    """

    await Authorize.jwt_required()
    raw_jwt = await Authorize.get_raw_jwt()
    if raw_jwt is None:
        return {"detail": "Invalid token"}, 401

    jti = raw_jwt["jti"]
    redis_client.sadd("denylist", jti)
    return {"detail": "Access token has been revoke"}
```

### 5.6.8 Using REST in our system

Understanding and correctly implementing various HTTP methods such as GET, POST, PUT, PATCH, and DELETE is crucial for the development of RESTful web services. Each method has its specific role and guidelines for use, contributing to the overall robustness and functionality of web services. In Housify system REST is

used to establish almost every communication between the user and the services we provide.

## 5.7 External Web Services

To enhance the system's functionality, external web services were integrated. The integration of these external services expanded the capabilities of the application, providing relevant and up-to-date information from external sources. The SOAP system fetches information from the binance API to get real time data of crypto currencie conversion.

```
curl -X GET "https://api.binance.com/api/v3/ticker/price?symbol=BNBBTC"
{"symbol":"BNBBTC","price":"0.00747100"}
```

## 5.8 Tests Performed

A comprehensive set of tests was developed to validate the functionality and robustness of the API. These tests covered various scenarios, including different operations, data manipulation, and error handling. The specifications of these tests contributed to ensuring the quality and reliability of the API, guaranteeing appropriate responses to various situations.

### 5.8.1 main_test.py

```python
import pytest
from httpx import AsyncClient
from core.housify_service import app
from api.router import setup_router


@pytest.fixture
async def client():
    await setup_router(app)
    async with AsyncClient(app=app, base_url="http://test") as ac:
        yield ac


@pytest.mark.anyio
async def test_root(client):
    response = await client.get("/")
    assert response.status_code == 200
    assert response.json() == {"message": "Welcome to Housify API"}
```

### 5.8.2 test_register.py

```python
import pytest
from httpx import AsyncClient
from core.housify_service import app
from api.router import setup_router
from datetime import date


@pytest.fixture
async def client():
    await setup_router(app)
    async with AsyncClient(app=app, base_url="http://test") as ac:
        yield ac


@pytest.mark.anyio
async def test_create_new_user(client):
    test_user_data = {
        "email": "testuser@example.com",
        "first_name": "Test",
        "last_name": "User",
        "hashed_password": "hashed_test_password",
        "gender": 1,
        # "phone": "+1234567890",
        "birth_date": str(date(2000, 1, 1)),
    }
    response = await client.post("/register", json=test_user_data)
    response_data = response.json()

    assert response.status_code == 201
    assert response_data["email"] == test_user_data["email"]
    assert response_data["first_name"] == test_user_data["first_name"]
    assert response_data["last_name"] == test_user_data["last_name"]
    assert response_data["gender"] == test_user_data["gender"]
    assert response_data["phone"] == test_user_data["phone"]
    assert response_data["birth_date"] == test_user_data["birth_date"]
```

## 5.9 Security rules

Several security rules were implemented to comply with data protection regulations:

1. The client's password undergoes double hashing with SHA512, both on the client and server. The server incorporates an internal UUID unknown to the client, ensuring that clients with the same password receive different hashes.
2. TLS v1.3 is employed and preferred to establish communication in the reverse proxy (nginx).
3. We obtain TLS certificates from Let's Encrypt to ensure secure communication.
4. JSON Web Tokens (JWT) are utilized for authentication over HTTPS, employing asymmetric keys (ECDSA512) for both access and refresh tokens to balance server load and enhance security. Additionally, a Redis service is employed with token blacklisting to deny access to potential threats.

### 5.9.1 Json Web Token Implementation

```
from async_fastapi_jwt_auth import AuthJWT
from datetime import timedelta
import os
from fastapi import HTTPException
from services.crud.user import get_id_with_email

ALGORITHM = "ES512"
key_path = os.getenv("ECDSA_KEY_PATH")


async def setup():
    """
    Set up JWT authentication by loading ECDSA keys and configuring AuthJWT.

    This function loads the ECDSA private and public keys from files and sets
    up the JWT authentication configuration for AuthJWT.

    Raises:
        FileNotFoundError: If the key files are not found.
    """

    def load_ecdsa_key(file_path: str):
        with open(file_path, mode="r") as file:
            return file.read()

    def load():
        return (
            load_ecdsa_key(key_path if key_path else "" + "ec_private.pem"),
            load_ecdsa_key(key_path if key_path else "" + "ec_public.pem"),
        )

    private_key, public_key = load()

    config = {
        "authjwt_algorithm": ALGORITHM,
        "authjwt_public_key": public_key,
        "authjwt_private_key": private_key,
        "authjwt_access_token_expires": timedelta(minutes=5),
        "authjwt_refresh_token_expires": timedelta(days=7),
    }

    @AuthJWT.load_config
    def get_config():
        return [(key, value) for key, value in config.items()]


async def is_token_in_denylist(jti, redis_client):
    """
    Check if a token's JTI (JSON Token Identifier) is in the denylist.

    Args:
        jti (str): The JTI of the token.
        redis_client: The Redis client for interacting with the denylist.

    Returns:
```

```
        bool: True if the token's JTI is in the denylist, False otherwise.
    """
    return redis_client.sismember("denylist", jti)


async def validate_access_token(Authorize, redis_client, email: str | None =
                                None, validate_sub_with_internal_id=True):
    """
    Validate an access token and check if it's in the denylist.

    Args:
        Authorize (AuthJWT): The AuthJWT instance for token validation.
        redis_client: The Redis client for interacting with the denylist.

    Raises:
        HTTPException: If the token is invalid or revoked, with a 401 status code.
    """
    await Authorize.jwt_required()

    raw_jwt = await Authorize.get_raw_jwt()
    if raw_jwt is None:
        raise HTTPException(status_code=401, detail="Invalid token")

    jti = raw_jwt["jti"]
    if await is_token_in_denylist(jti, redis_client):
        raise HTTPException(status_code=401, detail="Token has been revoked")

    if validate_sub_with_internal_id:
        if not email:
            raise ValueError("Email has to be provided")
        if (await Authorize.get_jwt_subject()) != await get_id_with_email(email):
            raise HTTPException(status_code=401, detail="Invalid Token")


async def validate_refresh_token(Authorize, redis_client, email: str | None =
                                None, validate_sub_with_internal_id=True):
    """
    Validate a refresh token and check if it's in the denylist.

    Args:
        Authorize (AuthJWT): The AuthJWT instance for token validation.
        redis_client: The Redis client for interacting with the denylist.

    Raises:
        HTTPException: If the token is invalid or revoked, with a 401 status code.
    """
    await Authorize.jwt_refresh_token_required()

    raw_jwt = await Authorize.get_raw_jwt()
    if raw_jwt is None:
        raise HTTPException(status_code=401, detail="Invalid token")

    jti = raw_jwt["jti"]
    if await is_token_in_denylist(jti, redis_client):
        raise HTTPException(status_code=401, detail="Token has been revoked")

    if validate_sub_with_internal_id:
```

```
        if not email:
            raise ValueError("Email has to be provided")
        if (await Authorize.get_jwt_subject()) != await get_id_with_email(email):
            raise HTTPException(status_code=401, detail="Invalid Token")
```

## 5.10 Crud

### 5.10.1 subscription_lvl.py

```
from data.models.subscription_lvl import SubscriptionLvl


async def create_default_subscription_lvl():
    default_levels = {
        "Free-Tier": {
            "price": 0.00,
            "upload_size_limit": 100,  # MB
            "storage_limit": 10,  # GB
            "its": 2.5,
            "api_key_limit": 1,
            "requests_hour": 6,
            "watermark": True,
        },
        "Pro-Tier": {
            "price": 9.99,
            "upload_size_limit": 1000,  # MB
            "storage_limit": 100,  # GB
            "its": 5.0,
            "api_key_limit": 10,
            "requests_hour": 60,
            "watermark": False,
        },
    }

    for description, attributes in default_levels.items():
        if not await SubscriptionLvl.filter(description=description).exists():
            subscription_lvl = SubscriptionLvl(description=description, **attributes
    )
            await subscription_lvl.save()
            print(f"Default {description} subscription level created")
```

### 5.10.2 user.py

```
from data.schemas.user import UserRegisterSchema
from data.models.user import User
from services.utils.hash import hash_combined_passwd
import uuid


async def create_user(
    user_data: UserRegisterSchema,
) -> User:
```

```
    assigned_id = uuid.uuid4()
    hashed_password = hash_combined_passwd(user_data.hashed_password, assigned_id)

    user = await User.create(
        id=assigned_id,
        email=user_data.email,
        first_name=user_data.first_name,
        last_name=user_data.last_name,
        hashed_password=hashed_password,
        gender=user_data.gender,
        phone=user_data.phone,
        birth_date=user_data.birth_date,
        subscription_lvl="Free-Tier",
    )

    return user


async def get_id_with_email(email: str) -> str:
    return (await User.get(email=email)).id.hex
```

**Chapter 6**

# Deployment

Due to the significant computation cost of AI inference, all of our services are self-hosted on our servers.

1. Cloudflare was used as a DNS resolver for 'housify.geniadynamics,' directing it to our IPv4 address.
2. All web services accessible to the client are appropriately hosted and documented. They are served by Nginx acting as a reverse proxy and hosted by Uvicorn.
3. Nginx also manages the HTTPS protocol, redirecting port 80 to 443. It prioritizes HTTP/3 while also enabling the usage of other versions.
4. The email server was self-hosted, allowing us to have control over its configuration and ensure the security and privacy of our email communication.
5. PfSense served as our primary router, overseeing both the internal network and WAN. Additionally, it functioned as a robust and enterprise-ready firewall, ensuring the security and integrity of our network infrastructure.

Given the substantial amount of information we store about the client and their associated usage, enabling cloud services for the client becomes a necessary step.

**Chapter 7**

# Conclusion

The development of this practical work on Information Systems Integration was a valuable opportunity to consolidate fundamental knowledge in the field, explore relevant technologies and frameworks, and apply service-oriented development practices. By focusing on interoperability between systems using web services, especially SOAP and RESTful, it was possible to achieve specific objectives aimed at creating a service library, detailed API documentation, and the development of a client application.

In summary, this work allowed for the consolidation of theoretical and practical knowledge, providing a comprehensive experience in the development of interoperable information systems based on web services. The application of the concepts learned not only strengthened the understanding of system integration but also prepared for future challenges in the field of Computer Science.

# Bibliography

Luís Ferreira, Óscar Ribeiro (2023). *project for the course unit ISI*. URL: https://elearning2.ipca.pt/2324/pluginfile.php/698457/mod_resource/content/1/ESI-ISI%202023-24%20-%20TP2%20-%20enunciado.pdf.

*REST, or REpresentational State Transfer* (n.d.). Accessed: 2023. URL: https://www.codecademy.com/article/what-is-rest.

*Simple Object Access Protocol* (n.d.). Accessed: 2023. URL: https://www.ibm.com/docs/en/sc-and-ds/8.1.0?topic=services-web-service-protocol-stack.