

## Assignment 02.02: Report

This report outlines the development of a secure web application designed to manage medical records, adhering to stringent security measures and integrating comprehensive observability through OpenTelemetry. The application, developed using a C# web API with a React frontend and SQLServer for data management, incorporates advanced security features such as authentication, access control mechanisms, and data masking to protect client information. Through the implementation of OpenTelemetry, the application achieves a high level of observability, enabling efficient monitoring, tracing, and logging of system operations.

### Application Architecture

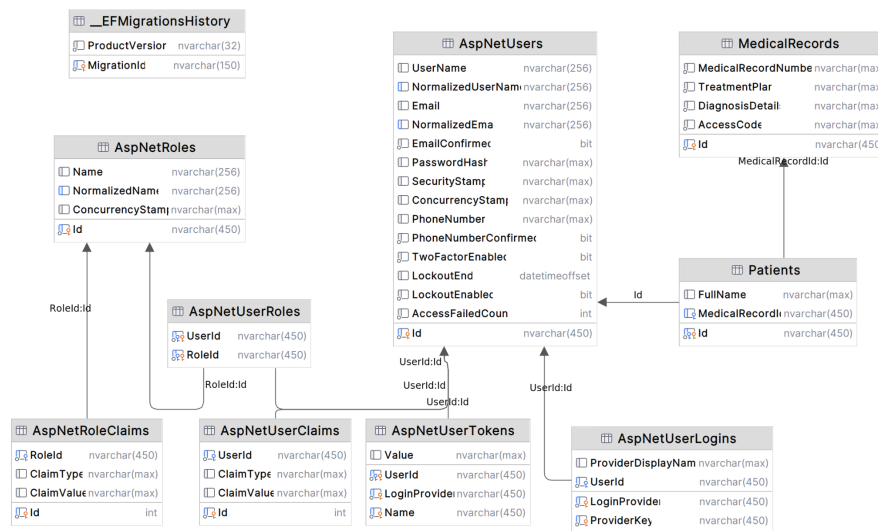
For the web application development, the technologies used were designated in the assignment: React for the presentation layer, .NET 8 webapi for the business layer and SQLServer for the Data Persistence Layer - the communication between the business and data persistence layers is addressed by the Entity Framework.

### API Structure

The API implementation was divided between controllers, repositories and models to employ segregation of responsibilities. There are three models ApplicationUser, Patient and MedicalRecord that generate the main tables of the database schema. The Patient model represents the main entity of the application which is composed of MedicalRecord to store all the medical data of the patient and ApplicationUser is extended from IdentityUser. This final model is necessary to not inject custom fields into the ASP.NET tables.

The AuthRepository and PatientRepository are classes that handle data operations related to authentication and patient data respectively. They are part of the Repository layer in the application, which is responsible for abstracting away the details of data access.

For the board part of the API the Controller classes will handle the authentication and authorization to gather the user role and information



# Data Masking

The data masking is handled at the database level with two database users without login, helpdesk and patient, with read access to the database. The sensible fields are masked using dynamic data masking showing only a partial string of the first two characters of the field. To grant the desired level of access to the data. Furthermore, to display data two stored procedures were created which receive as an argument the role of the user, which is identified using the information inside the JWT token from the login.

```
ALTER TABLE dbo.AspNetUsers
ALTER COLUMN PhoneNumber ADD MASKED WITH (FUNCTION =
'partial(2,"xxx",0)');
ALTER TABLE dbo.MedicalRecords
ALTER COLUMN TreatmentPlan ADD MASKED WITH (FUNCTION =
'partial(2,"xxx",0)');

ALTER TABLE dbo.MedicalRecords
ALTER COLUMN DiagnosisDetails ADD MASKED WITH (FUNCTION =
'partial(2,"xxx",0)');

CREATE USER helpdesk WITHOUT LOGIN;

CREATE USER patient WITHOUT LOGIN;

ALTER ROLE db_datareader ADD MEMBER helpdesk;

ALTER ROLE db_datareader ADD MEMBER patient;

GRANT UNMASK ON dbo.MedicalRecords(DiagnosisDetails) TO patient;

GRANT UNMASK ON dbo.AspNetUsers(PhoneNumber) TO patient;

GRANT UNMASK ON dbo.MedicalRecords(TreatmentPlan) TO patient;
```

Fig.2 - SQL to generate the database users and grant database permissions

```
CREATE OR ALTER PROCEDURE dbo.GetUserData
    @role NVARCHAR(50)
AS
BEGIN
    IF @role = 'patient'
    BEGIN
        EXECUTE AS USER
            = 'patient';
    END
    ELSE
    BEGIN
        EXECUTE AS USER
            = 'helpdesk';
    END

    SELECT
        [dbo].[Patients].Id,
        [dbo].[Patients].FullName,
        [dbo].[AspNetUsers].Email,
        [dbo].[AspNetUsers].[PhoneNumber],
        [dbo].[MedicalRecords].DiagnosisDetails,
        [dbo].[MedicalRecords].MedicalRecordNumber,
        [dbo].[MedicalRecords].TreatmentPlan
    FROM [dbo].[Patients] INNER JOIN [dbo].[MedicalRecords] ON
        [dbo].[Patients].MedicalRecordId = [dbo].[MedicalRecords].Id
    INNER JOIN [dbo].[AspNetUsers] ON
        [dbo].[Patients].[Id] = [dbo].[AspNetUsers].[Id]
    REVERT;
END;
GO
```

Fig.3 - Stored Procedure to get the user's data

## Access Code and Access Control Mechanisms

The platform must allow the helpdesk to access and edit the data from the user to help them. For simplicity's sake, the logic behind the Access code was simplified by letting the user define their access token in the sign and then the access code is hashed using one-way hashing to protect it.

To protect the different endpoints from the API the access is restricted by the role of the user that is trying to hit, giving Unauthorised code if the user does not have the required permissions. Furthermore, on the repository level, the restriction by role is also applied by checking the role inside the JWT.

The presentation layer also restricted access to some pages using the role in the JWT token redirecting to the homepage if the user is trying to access a restricted page such as a patient trying to access the list of patients.

```
[HttpGet("{id}")]
[Authorize(Roles = "helpdesk")]
0 references
public async Task<IActionResult> Get(string id)
{
    _logger.LogInformation($"Get method called with id: {id}");
    var response = await _patientRepository.GetPatient(User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.Role)?.Value, id);
    return Ok(response);
}
```

Fig.4 - Example of an endpoint that only helpdesk users can access

## Open Telemetry Integration

The integration with Open Telemetry was made by tweaking the Open Telemetry demo exposing the OpenTelemetry Collector ports so that the API could connect to it. Moreover, some directives were added to the Program.cs that will be presented next. By integrating with the OpenTelemetry it became possible to track and monitor useful data from the API. To get more custom information from the API logging was also enabled in the OpenTelemetry as is possible in Fig.4. With the API connected to the OpenTelemetry a Grafana board is created with the information related to the different metrics, logs and traces of it. To get a more detailed view of the traces with Jaeger.

```
Action<ResourceBuilder> appResourceBuilder =  
    resource => resource  
        .AddService("Patient Inc. API")  
        .AddDetector(new ContainerResourceDetector())  
        .AddDetector(new HostDetector());  
  
builder.Services.AddOpenTelemetry()  
    .ConfigureResource(appResourceBuilder)  
    .WithTracing(tracerBuilder => tracerBuilder  
        .AddRedisInstrumentation(  
            options => options.SetVerboseDatabaseStatements = true)  
        .AddAspNetCoreInstrumentation()  
        .AddHttpClientInstrumentation()  
        .AddOtlpExporter())  
    .WithMetrics(meterBuilder => meterBuilder  
        .AddProcessInstrumentation()  
        .AddRuntimeInstrumentation()  
        .AddAspNetCoreInstrumentation()  
        .AddOtlpExporter());
```

Fig.5 - Code added to the Program.cs to connect to OpenTelemetry

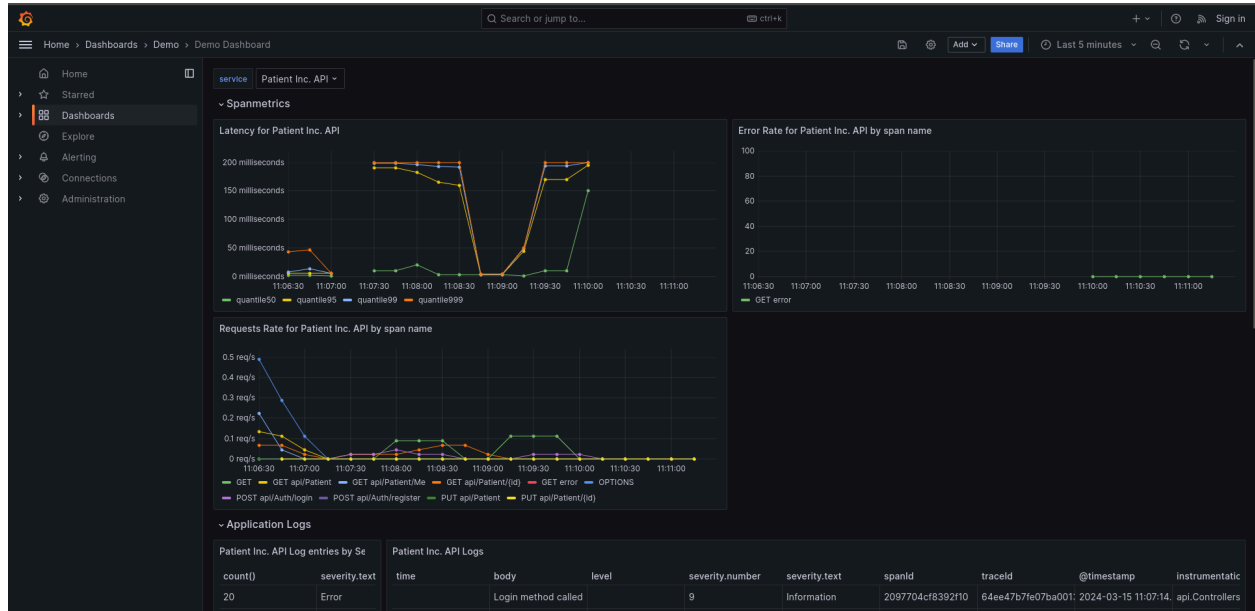


Fig.6 - Grafana Dashboard displaying the different hit metrics of the API endpoints

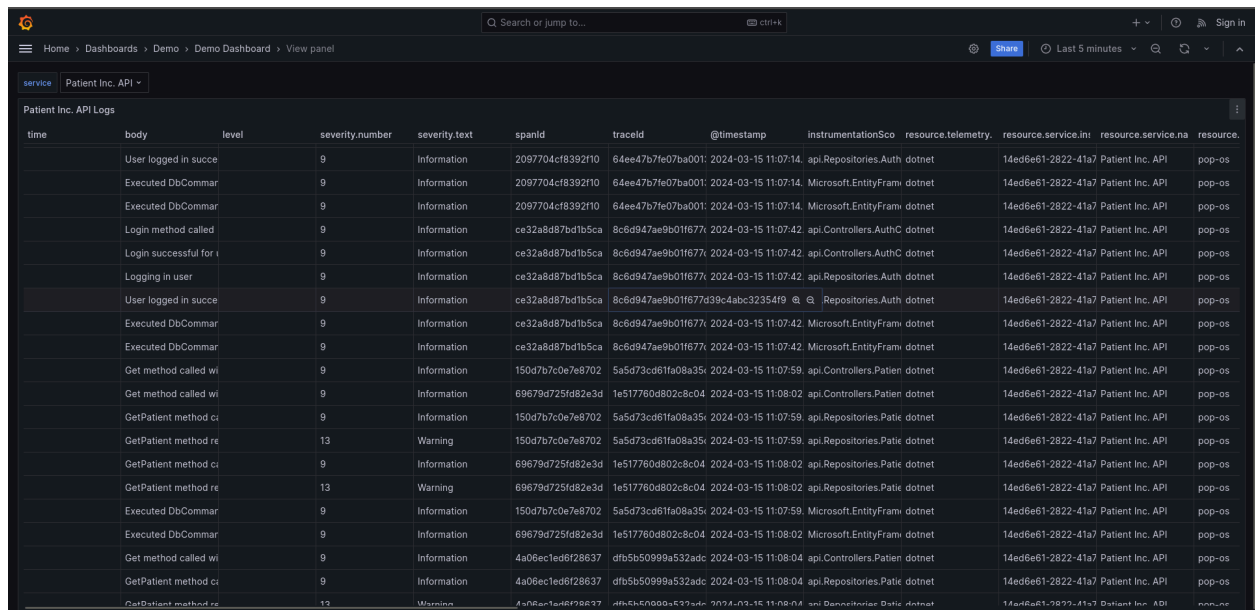


Fig.7 - Grafana Board of the logs provided by the application



Fig. 8 - Another Board of Grafana displaying the latency of API endpoints

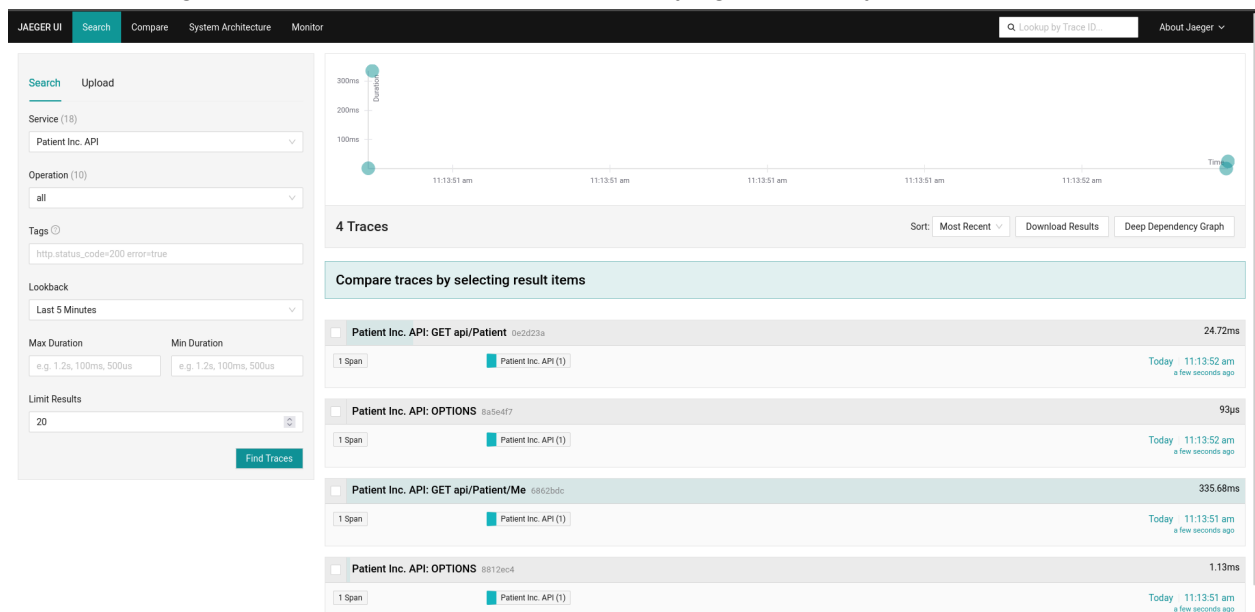


Fig. 9 - Trace Results of the last 5min on Jaeger

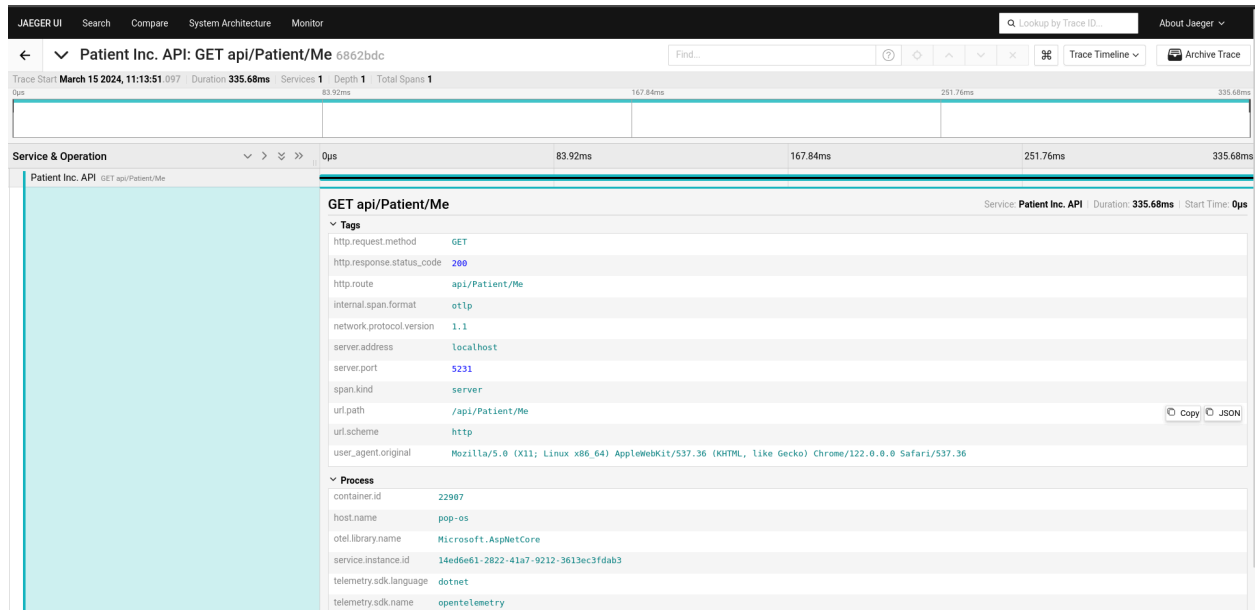


Fig. 10 - A detailed view of a trace of the endpoint api/Patient/Me

## Presentation Layer

The frontend page was thought to be simple and easy to show the use cases of the assignment straightforwardly. The main pages of the application are the home page to display the data unmasked; the helpdesk user list where all the users are displayed with their data masked; the forms to edit the data with or without access code and the sign in and sign up pages.

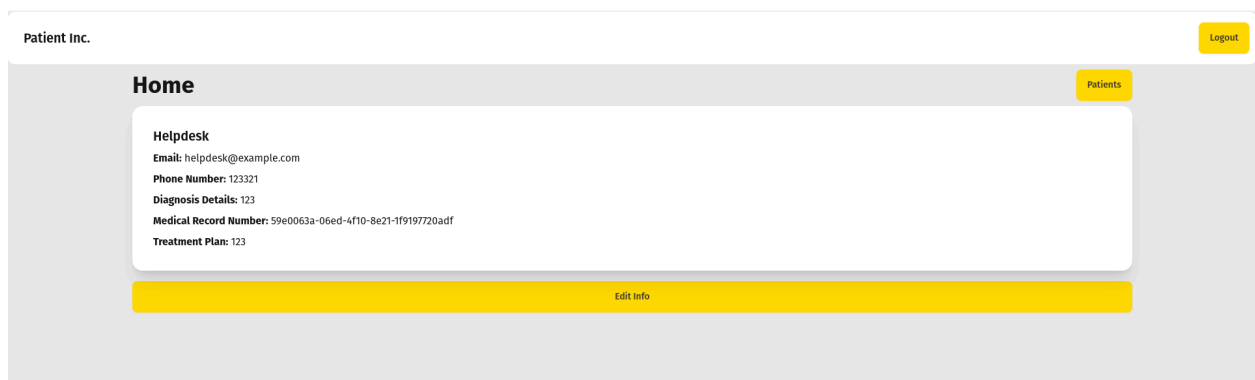


Fig. 11 - Homepage displaying user's personal data

Patient Inc. Logout

### Home

Editing Data

Fullname  
Vicente Barros

Email  
vicente@ua.pt

Phone Number  
911111111

Diagnosis Details  
teste

Medical Record Number  
6b25ead7-a080-4501-9caf-d5ec29055b1f

Treatment Plan  
Teste

Submit

Cancel

Fig. 12 - Form to change the user's data

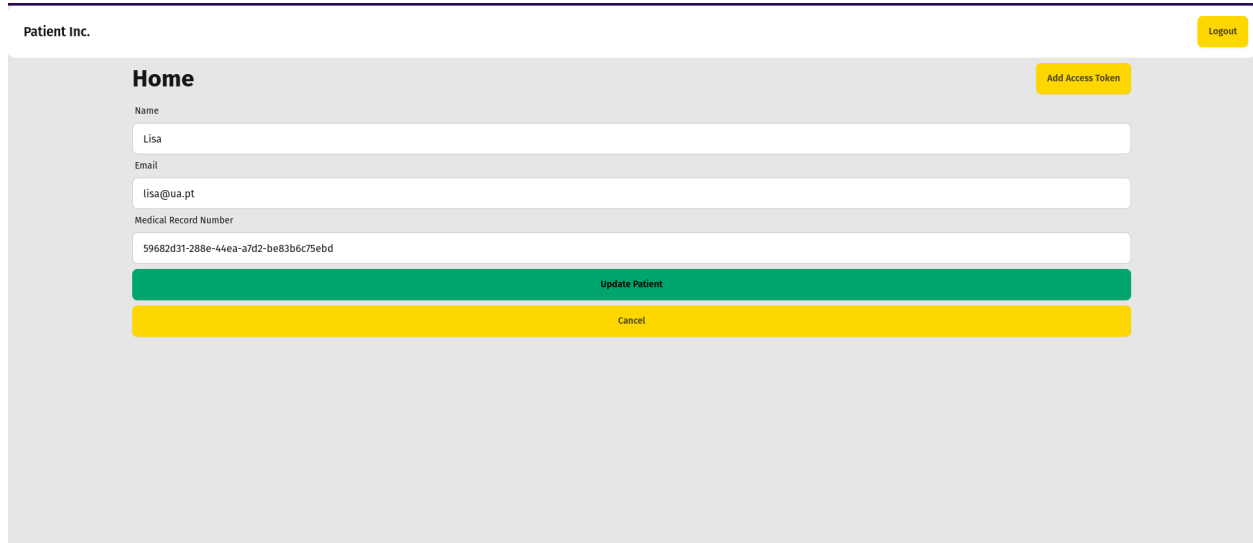
Patient Inc. Home Logout

### Home

| Full Name      | Phone Number | Diagnosis Details | Medical Number                       | Treatment Plan | Edit                      |
|----------------|--------------|-------------------|--------------------------------------|----------------|---------------------------|
| Lisa           | 92xxxx       | texxxx            | 59682d31-288e-44ea-a7d2-be83b6c75ebd | texxxx         | <span>Edit Patient</span> |
| Vicente Barros | 91xxxx       | texxxx            | 6b25ead7-a080-4501-9caf-d5ec29055b1f | Texxxx         | <span>Edit Patient</span> |
| Helpdesk       | 12xxxx       | 12xxxx            | 59e063a-06ed-4f10-8e21-1f9197720adf  | 12xxxx         | <span>Edit Patient</span> |
| Bob            | 93xxxx       | texxxx            | 4e7523d0-74dd-411a-bc78-c9244123e9d3 | 0xxxx          | <span>Edit Patient</span> |

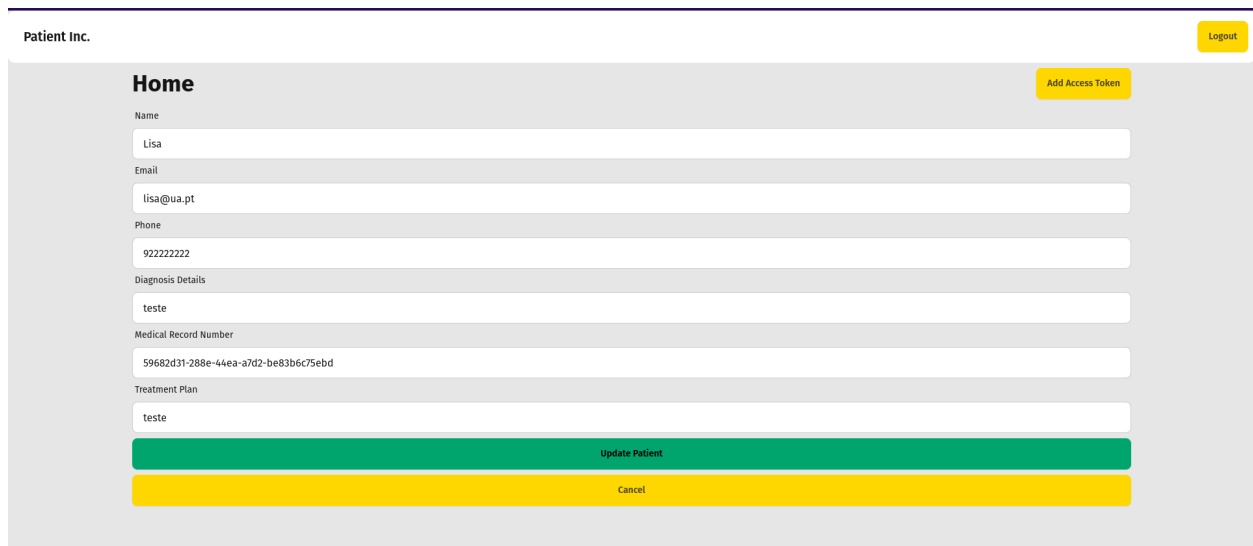
Fig. 13 - Helpdesk view of all the users in the application with their data masked





The screenshot shows a web interface for 'Patient Inc.' with a 'Logout' button in the top right. The main section is titled 'Home' and contains a form for editing a user's data. The form fields are: Name (Lisa), Email (lisa@ua.pt), and Medical Record Number (59682d31-288e-44ea-a7d2-be83b6c75ebd). Below the fields are two buttons: 'Update Patient' (green) and 'Cancel' (yellow). An 'Add Access Token' button is located in the top right corner of the form area.

Fig. 14 - Helpdesk form to edit the user's data without access token



The screenshot shows a web interface for 'Patient Inc.' with a 'Logout' button in the top right. The main section is titled 'Home' and contains a form for editing a user's data. The form fields are: Name (Lisa), Email (lisa@ua.pt), Phone (922222222), Diagnosis Details (teste), Medical Record Number (59682d31-288e-44ea-a7d2-be83b6c75ebd), and Treatment Plan (teste). Below the fields are two buttons: 'Update Patient' (green) and 'Cancel' (yellow). An 'Add Access Token' button is located in the top right corner of the form area.

Fig. 15 - Helpdesk form to edit the user's data with the access token

## Conclusion

Throughout this project, a deep and nuanced understanding of the intricacies involved in developing secure and observable web applications was cultivated. The practical application of dynamic data masking and access control mechanisms within a healthcare context illuminated the critical importance of safeguarding sensitive information, while also ensuring that such security measures do not impede the accessibility and functionality of the system for authorized users. The integration of OpenTelemetry provided a comprehensive insight into the significance of observability within modern software development, highlighting how metrics, traces, and logs can be leveraged to enhance system performance, reliability, and security.