



Alexandre Ribeiro
Guilherme Amorim
Matilde Teixeira
Rafael Ferreira
Rodrigo Graça

**Relatório Técnico: *DiseaseCard* - Sistema de
Visualização e Agregação de Doenças Genéticas
Raras**

**Technical Report: *DiseaseCard* - Visualization and
Data Aggregation System for Rare Genetic Diseases**



Alexandre Ribeiro
Guilherme Amorim
Matilde Teixeira
Rafael Ferreira
Rodrigo Graça

**Relatório Técnico: *DiseaseCard* - Sistema de
Visualização e Agregação de Doenças Genéticas
Raras**

**Technical Report: *DiseaseCard* - Visualization and
Data Aggregation System for Rare Genetic Diseases**

“Lack of knowledge - that is the problem.”

— W. Edwards Deming



**Alexandre Ribeiro
Guilherme Amorim
Matilde Teixeira
Rafael Ferreira
Rodrigo Graça**

**Relatório Técnico: *DiseaseCard* - Sistema de
Visualização e Agregação de Doenças Genéticas
Raras**

**Technical Report: *DiseaseCard* - Visualization and
Data Aggregation System for Rare Genetic Diseases**

Documento apresentada em Engenharia Informática da Universidade de Aveiro para cumprimento dos requisitos necessários à conclusão da unidade curricular Projeto em Informática realizada sob a orientação científica do Doutor (João Rafael Almeida), Professor associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor (José Luís Oliveira), Professor catedrático do Departamento de Eletrónica Telecomunicações e Informática da Universidade de Aveiro.

Dedicamos este trabalho ao nosso orientador, pela paciência e orientação dados ao nosso grupo.

**agradecimentos /
acknowledgements**

Agradecemos toda a ajuda a todos os nossos colegas. A todos os que nos conseguiram ajudar em qualquer uma das fase do nosso projeto: quer na recolha de requisitos, ou nos testes de usabilidade. Agradecemos ainda a paciência e orientação dadas pelo professor regente da cadeira de Projeto em Informática, Prof. Doutor Osvaldo Pacheco e pelo aluno de Mestrado em Engenharia Informática Daniel Ferreira.

Por último, mas não menos importante, queremos agradecer aos nossos três orientadores: Prof. Doutor João Almeida, pela paciência ao gerir este projeto, ao Prof. Doutor José Luís Oliveira e ao investigador Tiago Almeida.

Palavras Chave

Doenças Raras, Integração de Dados, Base de Dados Doenças Raras, Agregação de Dados

Resumo

O contexto atual da saúde e da investigação médica exige constante inovação e avanços tecnológicos e, para atender a essas necessidades, são essenciais sistemas capazes de se adaptar aos avanços tecnológicos e à crescente disponibilidade de dados. *Diseasecard*, desenvolvido na Universidade de Aveiro, e publicado em 2003, teve como objetivo agregar conhecimentos genéticos e médicos, a partir de informações espalhadas por várias fontes on-line e fornecer análises. Com a necessidade de novas atualizações, devido à evolução contínua da visualização de dados e das tecnologias, surgiu a oportunidade de desenvolver um sistema mais eficiente do que o existente. Portanto, neste projeto propõe-se o desenvolvimento de uma nova solução para substituir o *Diseasecard*.

Keywords

Rare Diseases, Data Integration, Rare Disease Database, Data Aggregation

Abstract

The current context of healthcare and medical research demands constant innovation and technological advancements and, to meet these needs, responsive systems capable of adapting to technological advancements and the increasing availability of data are essential. *Diseasecard*, developed at the University of Aveiro, and published in 2003, aimed to aggregate genetic and medical information scattered across various online sources and provide analysis. With the need of updates in the system, due to the continuous evolution of data visualization and technologies, the opportunity to develop a more efficient system has arisen. Therefore, in this project, it is proposed the development of a new software in order to replace *Diseasecard*.

Conteúdo

Conteúdo	i
Lista de Figuras	v
Lista de Tabelas	vii
1 Introduction	1
1.1 Motivation	2
1.1.1 <i>DiseaseCard</i> Problems	3
1.1.2 Improvements on <i>Diseasecard</i> to elaborate on next topic	3
1.2 Objectives	3
1.3 Outline	4
2 Context and State of Art	5
2.1 Existent Databases	5
2.1.1 OMIM	5
2.1.2 PubMed	5
2.1.3 HGNC	5
2.1.4 UniProt	6
2.1.5 Ensembl	6
2.1.6 GenBank	6
2.2 Existent Tools	6
2.2.1 Orphanet	6
2.2.2 Malacards	7
2.2.3 DO	7
2.2.4 GARD	7
2.2.5 NORD	7
2.2.6 NCBI	8
2.3 Previous <i>DiseaseCard</i> Version	8
2.4 Summary	8

2.5	Comparison Matrix	9
3	System Analysis Requirements	11
3.1	Requirements Gathering	11
3.2	Functional Requirements	11
3.2.1	Comprehensive Disease Information per Source	11
3.2.2	Flexible and Accessible Search Capability for Diseases	11
3.2.3	Versatile Source Oriented and Article Oriented Search	12
3.2.4	Daily Information Updates	12
3.2.5	Disable or Enable Article and Source for Maintenance Oversight for Administrator	12
3.2.6	Real-Time System Information for Administrator	12
3.2.7	Accessible CRUD Methods for Administrator	12
3.2.8	Graphical System Representation for Administrator	12
3.2.9	Secure Administrative Authentication for Administrator	13
3.3	Actors	13
3.3.1	User	13
3.3.2	Admin	13
3.4	Personas	13
3.4.1	User	13
3.4.2	Main Admin	14
3.4.3	Medical Admin	14
3.5	Use Cases	14
3.6	Non-functional Requirements	15
3.6.1	Modular Architecture for Feature Expansion	15
3.6.2	Ease of Deployment	16
3.6.3	Accessibility (All users including colourblind)	16
3.6.4	Cross-Platform Accessibility (Mobile and Web)	16
3.6.5	Rapid Information Retrieval	16
3.6.6	Low Latency	16
3.6.7	Scalability for Growing Disease Base	16
3.7	Visual Identity	17
4	Domain Model, Architecture and Deployment Diagram	19
4.1	Domain Model	19
4.1.1	Disease Family	20
4.1.2	Disease	20
4.1.3	Source	20
4.1.4	Article	20

4.1.5	Admin	20
4.1.6	Feedback	20
4.2	System Architecture	21
4.2.1	Primary Architecture	21
4.2.2	Secondary Architecture	21
4.2.3	Third Architecture	22
4.3	API Documentation	23
4.3.1	Admin	23
4.3.2	Authentication	24
4.3.3	Disease Family	24
4.3.4	Disease	24
4.3.5	Source	24
4.3.6	Article	24
4.3.7	Feedback	25
4.4	Web Scraping	25
4.4.1	Initial Method	25
4.4.2	Improved Method	27
4.4.3	Temporary Method	28
4.5	Aggregation Algorithm	29
4.6	Data Security	30
4.6.1	Role Based Access Control	30
4.7	Deployment Architecture	30
5	Results	33
5.1	Diseasecard	33
5.1.1	Homepage	33
5.1.2	Browse Page	34
5.1.3	Disease Family Page	35
5.1.4	Article Page	37
5.1.5	Feedback Option	37
5.2	<i>DiseaseCard</i> Admin	41
5.2.1	DashBoard	41
5.2.2	Manage Data	42
5.2.3	Monitorize Endpoints	42
5.2.4	Manage Medical Admins	43
5.2.5	Medical Admin View	43
5.3	<i>DiseaseCard</i> App	44
5.3.1	Disease Search	44

5.3.2	Disease Info	44
5.3.3	Article View	45
5.4	Summary	45
6	Analysis of Results and Future Works	47
6.1	Project Summary	47
6.2	Features and Benefits of the Product	47
6.3	Limitations of the Product	48
6.3.1	Sources	48
6.4	Potential Future Improvements	48
6.4.1	Client-Side	48
6.4.2	Admin-Side	49
6.4.3	Back-end Related	49
	Referências	51

Lista de Figuras

3.1	Emily Watson - User	13
3.2	Charles Sterling - Main Admin	14
3.3	George Stevens - Medical Admin	14
3.4	Use Cases Diagram	15
4.1	Domain Model Diagram	19
4.2	Enterprise Architecture	22
4.3	Admin Controller (1)	23
4.4	Admin Controller (2)	23
4.5	Authentication Controller	24
4.6	Disease Family Controller	24
4.7	Disease Controller	24
4.8	Source Controller	24
4.9	Article Controller	25
4.10	Feedback Controller	25
4.11	Outdated Scraping Process	27
4.12	Scraping Process	28
4.13	Algorithm Sequence Diagram	30
4.14	Deployment Diagram	31
5.1	Homepage	34
5.2	Browse Page	34
5.3	Bubble Graph	35
5.4	Hypertree Graph	36
5.5	Accordion List	36
5.6	Article Page	37
5.7	Disease Feedback	38
5.8	Disease Family Name Not Correct	38
5.9	Disease in Wrong Family	39
5.10	Article Feedback	40

5.11	Article in Wrong Disease	40
5.12	Empty Article	41
5.13	DashBoard Admin	41
5.14	Admin Feedback Acceptance/Refusal Page	42
5.15	Admin Endpoint Monitoring Page	42
5.16	Medical Admins Management Page	43
5.17	Medical Admin View	43
5.18	Disease Search	44
5.19	Disease Info	45
5.20	Article View	45

Lista de Tabelas

2.1 Comparison Matrix 10

Introduction

*"Knowing but lacking the power to express it is no better than never having any ideas."—
Pericles*

A rare disease is a health condition that affects a small number of people compared with other prevalent diseases in the general population [1]. Data regarding this condition can vary throughout different environments, such as Europe or the United States. As a matter of fact, in the European Union, it affected no more than one person in 2000 [2]. In contrast, in the United States, it affects 25 to 30 million adults (approximately 1 in 11 people) [3].

When viewing the rare genetic disease data, the results are even lower, with 1 in 50 individuals in the European general population being affected [4][5]. Even though the numbers are low, upon further investigation, these conditions not only affect the patient's physical and mental health but also have repercussions on their economic well-being due to the late and lack of information known by professionals when diagnosing, a consequence of the scarce number of cases [6]. This diagnosis can often be wrong, derived from the reasons shown above, with 50 % of patients not even receiving a diagnosis [7] [8].

Some initiatives have been started towards funding and improving investigations [9]. One initiative is IRDiRC¹, that count with currently more than sixty members from across the globe, whose goal is to spread awareness towards these cases and to foster collaborative research on these topics [10] [11].

As a possible solution to the problem, several platforms were developed, such as the National Organisation for Rare Disorders (NORD), OMIM² and Malacards, which are widely used to search for disease information. Each contains a well-defined list of diseases and large amounts of data for each one. This data includes genes, proteins, symptoms, enzymes, and other related information. Although these services portray information from many diseases, the information is scarce and spread across various pages.

¹International Rare Disease Research Consortium (<https://irdirc.org/>)

²OMIM: Online Mendelian Inheritance in Man- a database that catalogues information about genetic rare diseases.

These platforms, which will be thoroughly analyzed in the following chapter, often need a more user-friendly experience and must be updated and prepared for the current technological landscape. In this search to unify knowledge and offer a better understanding to the user, *DiseaseCard* emerged. This service was first developed by the bioinformatics department in IEETA ³ in 2003, but it suffered some modifications over the years. However, over the last five years, no improvements have been made.

1.1 MOTIVATION

The previous version of *DiseaseCard* was a platform for the visualization of data available across various pages/services that presented information for rare diseases. It counted with a client-side page for searches and an admin dashboard to monitor endpoints and their related data [12].

For this to be possible, it utilized parsers for every service, using an RDF ⁴ triplet structure. It used these frameworks to access COEUS, a semantic web framework to map SPARQL query results, directly to one specific ontology or more. To obtain the information from the different ontologies present, such as Omim, CSV and XML, and to give access to those other sources and articles, the parsers were used. [13]. Its architecture was based on this framework, using three databases: MySQL, Redis and SoLR. The first to store the triplet information for the different ontologies, Redis as a cache mechanism to facilitate searching and SoLR to improve the fetching time for the present data.

It also counted with several backend core components, such as a Data Collector, Data Connector, Browser Component, Cashier's, Index, Controller and AlertBox.

Firstly, the Data Collector, was used to ensure a correct connection to the RDF model that *DiseaseCard* implements. The Data Connector modeled the set of entities that perform requests to the API. This acted as a bridge between the requests and the possible operations that can be performed. A Browser component whose aim was to store the various diseases' names alphabetically in cache and map it to the disease's name.

A Cashier's whose purpose was to associate the OMIM with a list of tuples containing various identifiers and names of the sources containing the OMIM. An Index component that aggregates disease names with the sources to guarantee performance. A Controller, who has two separate modules, communicates with the service through a Web Socket to allow access to a REST API. The last component was an AlertBox, which presented information about the endpoints, namely when they failed.

One part worth mentioning is that this website presented a way to update the websites it cached information from by modifying the parser in the DCadmin interface. It also counted with the use of Redux, a framework that allows the creation of reusable components and implements large web applications, allowing a refresh-free data update.

³Institute of Electronics and Informatics Engineering of Aveiro (<https://www.ieeta.pt/>)

⁴Resource Description Framework

1.1.1 *DiseaseCard* Problems

This platform presented some problems. The first aspect is its usability. In the graph view, the principal and leaf nodes were presented by numbers instead of names, which professionals can perceive wrongfully.

Another problem happens when searching for a determinate article. The previous version of *DiseaseCard* showed an I frame of the article on the source's website. Due to the constant changes in these websites, much information is now not accessible through the same links, which shows an error to the user.

Another issue was the dispersed disease data: when the user searched for a determinate disease, it showed every appearance of the respective characters in every article. For example, when searching for diabetes, every single appearance of diabetes appeared even in cardiac arrest articles.

By doing this, the user could not perceive what information the disease searched had associated with and its related diseases. Beyond this, the above-said diseases were not aggregated by a family, presenting their information separately and not providing a central concept for the disease. Therefore, the data was broadly spread, causing errors in common usage.

On top of all, our goal in improving this platform was to use modern frameworks and technologies to build a maintainable architecture open to possible modifications. With this in mind, we utilized different frameworks and abandoned utterly the use of RDFs, SPARQL, and COEUS, whose usage was done since the start of the said project, aiming to simplify the possible changes this project may be open to in the near future.

1.1.2 Improvements on *Diseasecard* to elaborate on next topic

With that said, the primary purpose of our work is to make the platform more accessible to the user by using standard nomenclature and aggregating the diseases into a disease family.

The creation of families was relevant due to the spread of disease data of related diseases. With this, the user can explore the different diseases in a family and explore its various sources and articles. The user can now search while knowing exactly which article or source they want to visit by title. By doing this, it provides the user with a better experience.

1.2 OBJECTIVES

The aim of this project is to improve the live view of the articles and information the website presents about a specific disease. With this in mind, we want to get the article content by using web scrappers to gather the source's HTML and then display it on our platform, not using I-Frame. This approach will make the website much cleaner, accessible, disease-oriented, and updated. The website will also not have so many articles that cannot be accessed due to the ever-changing platforms from which the I-frame was gathering the information.

Another relevant aspect is the abandonment of previously used technologies, such as a triplet structure of RDFs, SPARQL queries, and COEUS, the semantic framework used

to aggregate that. By not implementing that, we bring modernity to this platform and make it more accessible to future modifications, given that it is more easily understood and interchangeable.

To innovate, we aim to build a mobile app. The idea of mobile development surged. This was due to the generality of the population that uses their cell phone more often to access information instead of a website.

With this change, we hope to provide information in a broader format and be the first from its state of the art. It is important to note that given its smaller display area, it will only present a small portion of data. The About Me page states that the complete data can be accessed through the website.

In sum, the main purpose of this document is to address these issues:

- Implement a solution with the information presented concisely and properly for every user to be capable of accessing it
- Build a completely different *Diseasecard* from scratch using newer technologies
- Create an accessible way for admin usage and easy monitoring of the system
- Create a different form of access for the general user through a mobile app

1.3 OUTLINE

This document contains five more chapters. The second one aims to delve into the state of the art, exhibiting the existing platforms and portraying an insight into what should be improved. In this chapter, we will analyze the databases and services available in the market.

In Chapter 3, we will present the requirement gathering, functional requirements, actors used to portray the service's use, use cases for these actors, and nonfunctional requirements. In Chapter 4, the Domain Model will be addressed and its various versions, the choice of Architecture and its different setbacks, and the Deployment Diagram.

In Chapter 5, we will show the results and a brief summary. In Chapter 6, we will discuss the system obtained and possible improvements in the future.

Context and State of Art

This chapter will explore the databases and tools that provide access to different disease information. This is essential to understanding the possible obstacles we must overcome to innovate and make the necessary changes.

2.1 EXISTENT DATABASES

2.1.1 OMIM

OMIM, short for Online Mendelian Inheritance in Man, is a comprehensive and authoritative collection of human genes and genetic phenotypes that is freely available and updated daily, published in 1985 as an effort of the NCBI to bring the MIM book editions to an online format. It has both an option to be downloaded or accessed through an API.

Unlike primary data databases, OMIM synthesizes and summarizes new and essential information based on the literature review. It is authored and edited by Dr Ada Hamosh at the McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins School of Medicine.

2.1.2 PubMed

They were published a decade after OMIM as a new NCBI project to provide an interface to the MEDLINE database. PubMed Central is a full-text biomedical and life sciences journal literature archive at the U.S. National Institutes of Health's National Library of Medicine (NIH/NLM). It was redesigned for the last time in 2020, improving its last update in 2009.

2.1.3 HGNC

The HGNC is responsible for approving unique symbols and names for human loci, including protein-coding genes, ncRNA genes, and pseudogenes, to allow unambiguous scientific communication.

The rise of problems with nomenclature in human genetics was recognized in the early 1960s, and in 1979, full guidelines for human gene nomenclature were presented at the Edinburgh Genome Meeting.

One problem is the compromise between convenience and simplicity for everyday human use and the need for an adequate definition of the concepts involved. Starting in 1996, HGNC stores all approved symbols in the HGNC database, ensuring that each gene is unique. They are currently storing a total of forty-three thousand symbols.

2.1.4 UniProt

UniProt, or The Universal Protein Resource, is a comprehensive protein sequence and annotation data resource. It controls the following databases: UniProtKB (UniProt Knowledgebase), UniProt Reference Clusters (UniRef), and UniProt Archive (UniParc).

Over 100 people are actively involved in the project, which is still actively maintained as a collaboration between the European Bioinformatics Institute, the Swiss Institute of Bioinformatics, and the Protein Information Resource.

2.1.5 Ensembl

The Ensembl project, which started in 1999, gained life as the need for automatic genome annotation became increasingly urgent.

The project also added the possibility of integrating these annotations with other available biological data and making all this publicly available via the web. By 2020, the project supported over fifty thousand genomes across its two websites (Ensemble and Ensemble Genomes).

2.1.6 GenBank

GenBank is the NIH genetic sequence database. There, we can find a collection of all publicly available DNA sequences. GenBank has new releases every two months with detailed information about the release and notification of upcoming changes to GenBank.

Since most data comes directly from individual submissions, including researchers, some content may have been copyrighted by their submitters.

2.2 EXISTENT TOOLS

2.2.1 Orphanet

Orphanet provides reference information and knowledge to all actors in the rare disease ecosystem. In particular, It has created and maintained the rare disease nomenclature, which serves as a common language for medical professionals, researchers, and decision-makers worldwide.

With an acceptably good interface, Orphanet allows us to perform a wide range of tasks, the most obvious being searching and reading information about a disease. It also allows us to perform an advanced search, such as searching for a specific OMIM or gene symbol.

The autocomplete functionality is implemented but only sometimes works as desired. They also have functionality that allows searching for other information, like news and treatments.

2.2.2 Malacards

MalaCards is an integrated searchable database of human maladies and their annotations, modelled on the architecture and richness of the GeneCards databases of human genes.

Each card contains detailed information about a specific disease, including its aliases, classifications, anatomical context, drugs and therapeutics, expression, genes, genetic tests, pathways, publications, related diseases, sources, summaries, symptoms and phenotypes, and variations. It has 22811 disease entries, consolidated from 75 sources.

One aspect worth mentioning is Malacards' need for better UIX. The front page could be more or appealing to the human eye. They also support all basic operations, although advanced search and auto-complete need to be revised.

2.2.3 DO

The Disease Ontology(D.O.) has been developed as a standardized ontology for human disease.

Its purpose is to provide the biomedical community with a consistent, reusable, and sustainable description of human disease terms, phenotype characteristics, and related medical vocabulary disease concepts through collaborative efforts of biomedical researchers coordinated by the University of Maryland School of Medicine.

D.O.'s main objective is to provide an open-source ontology as a genomic resource for integrating biomedical data associated with human disease, disease features, and mechanisms. It also serves as a reference framework for multiscale biomedical data integration and analysis, strengthening the disease information ecosystem.

Disease Ontology is a flawed system, even with a good home page. Getting to the search bar could be more intuitive as it is hidden behind the Disease Ontology button in the menu. Once there, we find an outdated interface with unexpected behaviour. The lack of auto-complete and a basic disease summary also discourages further use.

2.2.4 GARD

GARD is an NIH program that provides free access to reliable, easy-to-understand information about genetic and rare diseases.

Genetic and Rare Diseases (GARD) emphasize translational science, a field dedicated to turning laboratory, clinic, and community observations into innovations to improve public health.

GARD's website is well-maintained and filled with numerous functionalities. Nonetheless, it lacks essential tools, like advanced search or related diseases.

2.2.5 NORD

NORD, or National Organization for Rare Diseases, is a non-profit organization that studies rare diseases. It has a dedicated database for this purpose. NORD aims to improve individuals' lives by increasing care, studies, and coverage of rare diseases through various initiatives and projects.

The website U.I. could be improved, where it makes the customer experience hard but offers selection by disease (that often does not work) and for selection in the disease report. This report also provides an option to print the report as a whole. Results are given in different languages and shown randomly in the search process.

2.2.6 NCBI

NCBI, short for National Center of Biotechnology Information, has contributed to the NIH mission of 'uncovering new knowledge' by pioneering on many fronts.

Understanding modern molecular biology, from an alphabet of four letters representing chemical subunits to unravelling these into new "words and phrases," is a central focus of molecular biology. The staggering volume of molecular data and its cryptic and subtle patterns have required computerized databases and analysis tools.

Therefore, the NCBI has accepted the challenge of finding new approaches to dealing with the volume and complexity of data and providing researchers with better access to better tools to advance understanding of our genetic legacy and its role in health and disease.

NCBI's webpage has many search options and provides results from different databases regarding genes, literature, proteins, clinical, and pubchem. However, how the information is presented to the user is awful and confusing. It does not have autocomplete and has rather complicated and unnecessary search mechanisms, such as an advanced search builder.

2.3 PREVIOUS *DISEASECARD* VERSION

The previous *DiseaseCard* version was a platform aggregating disease information from various sources. The website used a live-view feature to display source information. However, this feature is not available for most articles due to the constant updates and changes to the website.

Another aspect worth mentioning was the website's accessibility. When searching for a disease, the results were from every appearance within the platform. Furthermore, when viewing a disease, the way its corresponding articles were presented induced confusion because they were presented as numbers rather than titles. The platform also had the option to search all the diseases and see the associated OMIM and concepts.

2.4 SUMMARY

In sum, a wide range of databases and tools for this type of service have emerged with the evolution of technology and data processing. They represent a crucial role in searching, discovering, and comprehending multiple rare genetic diseases through their access to multiple sources, related articles, and other related information.

These platforms offer information, from genetic sequence to literature, like OMIM, PubMed, GenBank, or even specific disease information, that facilitates research and collaboration between health professionals and researchers. Within this range, we can identify problems such as usability issues that make general use decay and lack of actualization issues that produce the same effect.

2.5 COMPARISON MATRIX

The following matrix illustrates the many differences between the abovementioned databases and tools. Our evaluation was based on two main factors: functional and technical.

Within functional, we analyzed seven different aspects: if there is a disease summary for every disease presented (**Disease Summary**), the capability and efficiency of the search mechanism (**Search**), the capability of search selecting advanced parameters such as different filters (**Advanced Search**) if a functional auto-complete system is provided to help the user in the search for diseases (**Auto-complete**) if there were disease categories, indicating different ranges of the disease (**Disease Categories**), and presents information of other diseases that were related with (**Related Diseases**). Some diseases' symptoms were present in some services (**Symptoms**).

Regarding technical aspects, we analyzed three fundamental elements: **Number of sources**, **number of diseases**, **number of articles**, the degree of usage each user interface had (**User Interface**) and whether they had or not a mobile app (**Mobile**).

In the following table, we present the results of this gathering using the following symbols:

- ✓ - this symbol represents that the feature is implemented
- ! - this symbol represents that the feature is implemented but not functional
- X - this symbol represents that the feature is not implemented at all

System Analysis Requirements

Gathering functional actors, use cases and non-functional requirements.

3.1 REQUIREMENTS GATHERING

To build a better service that will be used by many people, we recurred to requirements gathering to perceive the principal pains our desired user could have and the features required to address them. By understanding their challenges and desires, we aimed to craft a platform that not only addresses their needs but exceeds their expectations.

3.2 FUNCTIONAL REQUIREMENTS

Examining the process to extrapolate requisites presented in the previous chapter highlights the essential practical standards that the system must have. These criteria play a vital role in specifying the changes needed to transform *Diseasecard* into a more user-friendly and efficient platform to support greater public use.

3.2.1 Comprehensive Disease Information per Source

As said before, one of *Diseasecard*'s goals is to aggregate different sources with information about the same disease. By implementing this, the final user will be able to access all information about the disease. This is done by navigating through the different sources presented, allowing a more enjoyable and accessible experience by getting all the information online.

3.2.2 Flexible and Accessible Search Capability for Diseases

One of *Diseasecard*'s problems is the search mechanism. How it is implemented now shows all sources with a piece of information that has been searched. The result is an aggregation of all appearances of the disease name, only showing different appearances of the word, part of the word search, or even other sources of the same information.

By using the website the way it was, the user was left disoriented about which source it had accessed. Therefore, we aim to centralize the information about the disease in one search option, allowing the user to have a more accessible use.

3.2.3 Versatile Source Oriented and Article Oriented Search

The previous version of *Diseasecard* already possessed this feature; however, it was no longer accessible to users because of the complicated terminology used. This resulted in users getting lost in the course of searches. Our objective is to give this information more transparently, ensuring that anyone can easily access and use it.

3.2.4 Daily Information Updates

One of the main problems in the previous version of *Diseasecard* is outdated information, with some articles missing and others altered since the last update. As a result, we will update it regularly to address the issue, ensuring missing articles are introduced and any alterations are promptly meditated, thereby preserving consistency.

3.2.5 Disable or Enable Article and Source for Maintenance Oversight for Administrator

Monitoring the different endpoints is essential for providing a fully functional service. By observing user feedback and related information we can infer if a website has been hacked or if the articles it presents start showing wrongful data. By having this feature, the admin can disable a determinate article or source.

3.2.6 Real-Time System Information for Administrator

Performing real-time assessments on outside endpoints is essential to ensure the accuracy of the cited content material. Engaging in continuous verifications, we acquire up-to-date data about the various endpoints, contributing to maintaining the service's integrity. This ongoing monitoring strategy is vital for maintaining the system's accuracy and reliability.

3.2.7 Accessible CRUD Methods for Administrator

As a health service-related platform, accuracy and reliability are crucial, and even so, administrators should be able to manage data effectively. This consists of the ability to delete faulty records, update resources or articles when errors arise, retrieve data from the platform, and add new information, sources or articles as needed. Moreover, administrators can manipulate the device, enabling them to actively manage and ensure the website's content's accuracy and quality.

3.2.8 Graphical System Representation for Administrator

To provide a comprehensive system assessment, we will implement a dashboard allowing entry to the above usages. This dashboard will function as a centralized hub, empowering administrators to efficiently control and screen diverse factors of the platform in an accessible interface.

3.2.9 Secure Administrative Authentication for Administrator

For a safer dashboard, we plan to implement admin authentication. By implementing this approach we secure sensitive data from general usage, ensuring security.

3.3 ACTORS

To analyze the usage of our service we will define actors and personas, for a more authentic approach.

3.3.1 User

The primary user is a healthcare professional who wants to use *Diseasecard* to search for disease information.

3.3.2 Admin

The admin has access to the administrative platform and maybe, or maybe not, someone from the health area since there are two types of administrator: a **medical admin**, who has permission to verify and accept changes related to medical data; a **main admin**, who apart of the medical permissions, can, as well, monitor and manage endpoints. For example, he may turn off an article endpoint or all the endpoints from some data source.

3.4 PERSONAS

3.4.1 User

Emily Watson is 23 years old and studies Medicine at Oxford University. She wants to use *Diseasecard* because of her research work about Diabetes, so she pretends to read many articles about the many types of Diabetes.



Figura 3.1: Emily Watson - User

3.4.2 Main Admin

Charles Sterling is 46 years old, has a master's degree in Software Engineering and is the head of *Diseasecard*. He desires to access the administrative platform to add the most recently hired "medical admin", George Stevens, to the system.



Figura 3.2: Charles Sterling - Main Admin

3.4.3 Medical Admin

George Stevens is 67 years old. He has been a medic his whole life, and now that he is retired, he was offered a part-time position to manage *Diseasecard* diseases data. He wants to use the platform to check the feedback proposed by the users and accept the ones he agrees with.



Figura 3.3: George Stevens - Medical Admin

3.5 USE CASES

So, the generic user uses the system to search for information about a disease. He may do that by searching directly for the disease name in the homepage search bar or browsing through all diseases available, applying filters on the browse page. After these, the user accesses the information about the disease and has three ways to visualize the data: in a graph, tree, or list. In one of these components, the diseases belonging to the disease family chosen are presented to the user, and many articles from many data sources for each disease are presented that the user can open and read.

About the admin, technically, there are two different types of admin: the **main** one, which is the only person who can access all administrative functionalities, and the **medical admin**, which can be assigned to more than one person, to have access to filtered functions of the admin. At first, we did not make use cases like this, but after some very well-posed questions in one of the class milestones, we decided to change it to have a system with two different admins, as explained above.

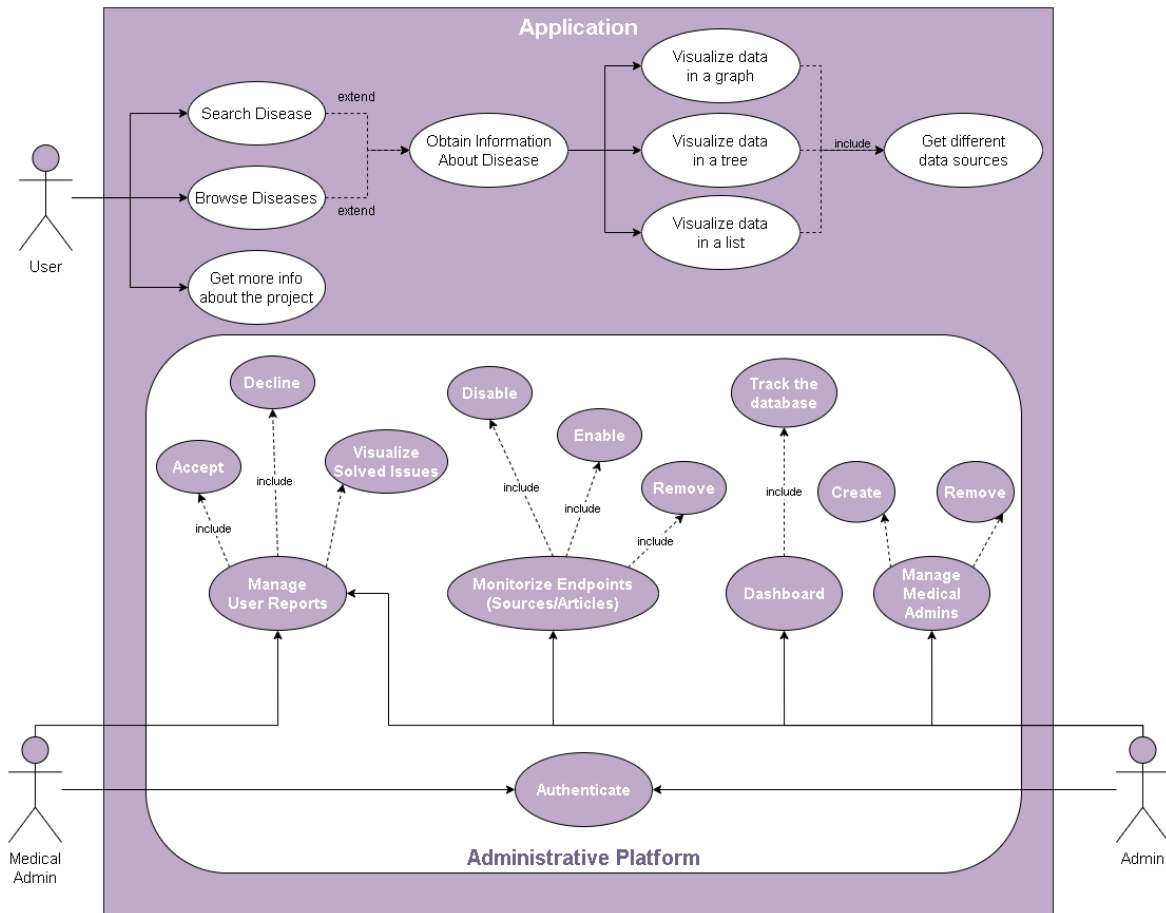


Figura 3.4: Use Cases Diagram

First, the admin needs to authenticate; when this happens, the system filters the possible actions by his role. The **Main Admin** has access to a dashboard to see the number of Families, Diseases, Sources and Articles that exist in the database; a section to monitor sources and articles endpoints (including disable, enable again or remove them); a segment to create or remove medical admins. The next topic is the one that all the administrators can access, and so, the only one that a medical admin can see on the platform. All user reports are received and can be accepted or denied in the Manage Data section. The admin can also visualize previously solved issues.

3.6 NON-FUNCTIONAL REQUIREMENTS

Concerning the previous requirements shown above, it is essential to also categorize the non-functional requirements to make the system more usable and user-friendly. This type of approach ensures a targeted method, thinking about elements that include scalability, accessibility, reliability, and modularity, thus improving the previous service model.

3.6.1 Modular Architecture for Feature Expansion

Implementing a modular structure is essential, promoting greater independence between modules, minimizing coupling, ensuring easy preservation, and enabling scalability. This

approach complements current system efficiency and provides a stable path for future project development.

3.6.2 Ease of Deployment

Efficiency, lowering downtime, and effortless deployment are prioritized to decrease complexity and guarantee an easy transition from development to practical use. This schema complements overall performance and enables seamless integration into practical usage scenarios.

3.6.3 Accessibility (All users including colourblind)

The platform will be designed to be universally accessible, ensuring that all people of various backgrounds and ages, including people with colourblindness, can effectively research data. This is a significant improvement over the previous version of the platform, which did not cater to colourblind users. Additionally, incorporating colorblind-friendly design practices aligns with broader accessibility goals, making the platform extra inclusive and user-friendly for a wider target audience.

3.6.4 Cross-Platform Accessibility (Mobile and Web)

The previous platform counted only with a web platform. To ensure greater use by professionals and students, we will maintain the web version and develop a mobile app to make the search experience more appealing and ensure its use is everywhere.

3.6.5 Rapid Information Retrieval

As an aggregating system, information access is crucial, as it greatly influences user engagement. Therefore, by allowing rapid information retrieval, we will encourage regular usage. This improvement will not only promote user satisfaction but also contribute to the global effectiveness of the platform by providing a seamless and compelling experience for all users.

3.6.6 Low Latency

Regarding the aspects discussed above, today's world is a factor to consider. As users demand even faster services and give up on those with more latency than usual, we have to take measures for this not to affect *DiseaseCard*. To be widely and commonly used as our endeavour, *DiseaseCard* must have low latency, allowing for a more pleasant experience.

3.6.7 Scalability for Growing Disease Base

Information is generated and uploaded every second, so the scalability of our database is of utmost significance. This ensures that users can always get accurate and up-to-date information. Adopting a scalable database not only meets current demands but also prepares our system to evolve easily to the ever-changing data landscape, thereby ensuring a reliable and responsive user experience.

3.7 VISUAL IDENTITY

Concerning these requisites, we modelled our platform to be accessible to all. To ensure this, we tested each colour used in the website with specific Color Blindness Websites to see if all coloured colours were used. This need surged because Color Blindness is a rare genetic disease. Where even though its percentage is relatively low, we want to ensure that information is available to all users. The other sections were inspired by the *DiseaseCard's* previous versions.

Domain Model, Architecture and Deployment Diagram

In this chapter, we will explore the building blocks of our platform and examine the decisions made during the project's elaboration phase.

4.1 DOMAIN MODEL

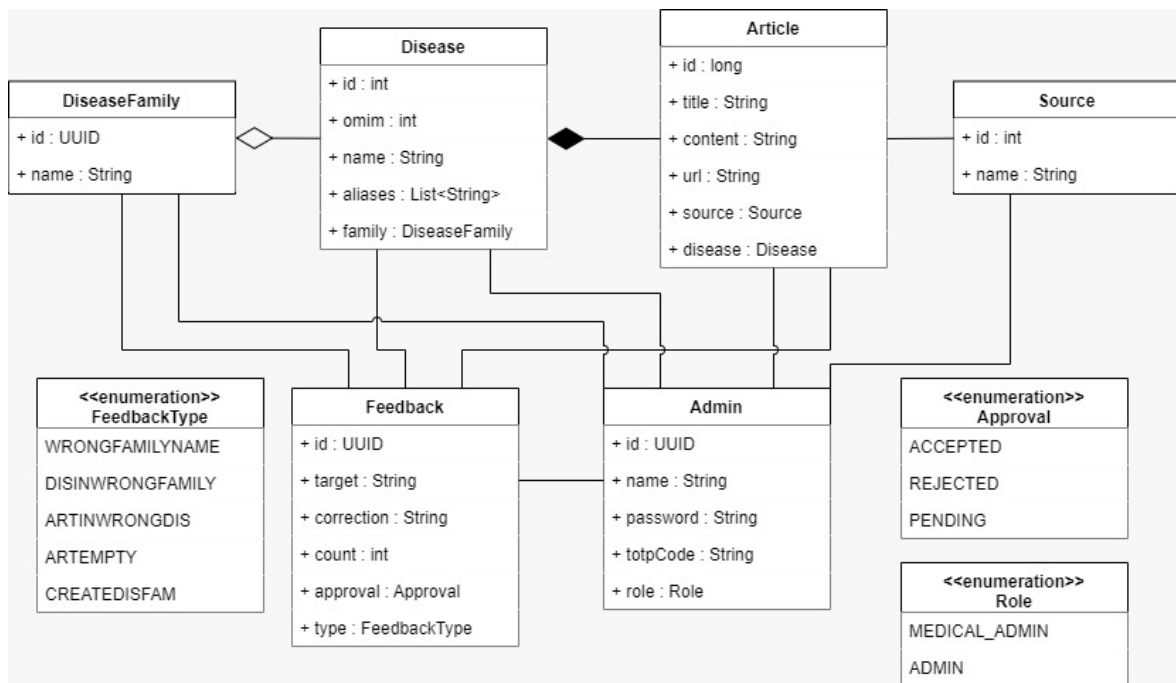


Figura 4.1: Domain Model Diagram

4.1.1 Disease Family

The *Disease Family* class allows us to group diseases under a common name, simplifying search queries and allowing users to find what they are looking for more quickly. Not every *Disease* belongs to a higher encompassing family, so there are *Diseases* that have no connection to any *Disease Family*.

4.1.2 Disease

The *Disease* class is the most essential part of our platform because it serves as the cornerstone for all other classes. The identifier we chose is the **OMIM**, referring to the Online Mendelian Inheritance in Man, because it is the most common way to identify rare genetic disorders, so much so that we can use it to gather relevant information from other sources quickly.

4.1.3 Source

The *Source* class, similar to the *Disease Family*, serves to group articles by their respective source. We defined it as a different entity and not an attribute of *Article* to monitor and manage the various sources in case they become unavailable or compromised.

4.1.4 Article

The *Article* class stores information that we gather from sources outside OMIM. Each Article is uniquely associated with a single disease and contains the HTML code selected from their respective source.

4.1.5 Admin

Unlike regular users, admins can access parts of our service that allow for maintenance. There are two roles: system admin and medical admin. We store the login information for each, which may include a TOTP token.

4.1.6 Feedback

Users can submit feedback whenever they spot an error regarding articles or diseases. There are different types of feedback, and for each of them, the content stored in *target* and *correction* will be different. **WRONGFAMILYNAME** means that a disease family does not have the correct name, so the *target* will be the current name, and *correction* will be requested update. **DISINWRONGFAMILY** indicates that the disease named in *target* is not in the right family and should instead belong to the family in *correction*. **CREATEDISFAM** means that the disease named in *target* should be in a family named *correction* that does not exist. **ARTINWRONGDIS**, much like the previous type, means that the Article in *target* belongs to the disease in *correction*. Lastly, **ARTISEMPTY** serves to report empty articles. Feedback can be rejected or approved by medical admins.

4.2 SYSTEM ARCHITECTURE

It is essential to consider that during the development of this project, our initial architecture suffered massive changes, which will be documented in this section.

4.2.1 Primary Architecture

In the first phase of our project, we designed an architecture aligned with the previous choices made in the previous version of DiseaseCard.

However, we used for the front-end NextJs, a ReactJs framework, to add innovation to the project. We opted for Neo4j for the back-end to keep the disease, article, and source information in graph format. The main reason for using this framework was the graph manipulation inherent to the framework, which simplified the process of querying different articles, sources, or diseases.

We wanted to utilize Redis for monitoring purposes because it is a fast engine derived from its memory management. This monitoring was meant to be done manually for each endpoint.

It also included a proper disease ontology, using RDFs, SPARQL queries, and the COEUS semantic framework to access and connect the different disease information. For the backend core component, we opted to use Spring Boot using Java due to our familiarity with this tool.

4.2.2 Secondary Architecture

In the second phase, we realized this during the previous project's presentation during classes. As a result of multiple research concerning RDF triplet structure and the inherent semantic framework, we decided to completely change our architecture to simplify the approach for our team and for possible changes the future may present for future developers.

The main change in this architecture was the use of web-scrapers. By doing this, we can access a large portion of information quickly and steadily. The scrapers access each page from each trusted source and collect the necessary information (title, doi, and abstract).

To conduct that, we intended to save this information in an elastic search database with all the information available for the Neo4j database to structure the information into a graph.

We intended to use Redis as a message broker to process the information. After some suggestions made by the regent teachers, we opted to discharge this due to the inefficiency of the service.

We opted to use Elastic Search due to the possible endpoint monitoring option, which was well thought out due to the monitoring feature we wanted to implement. The use of elastic search was also justifiable because of the way we wanted to implement it due to the possible scalability, the document-oriented search mechanism, and the fast search mechanism.

The intended way for the process to work will be described in a sequence diagram below. The process would start when the web scraper was initiated; they would recollect the necessary information from each source and then put it in the elastic search database. For the various queries the database needed to do, the neo4j would store them to provide faster results and then show them to the user.

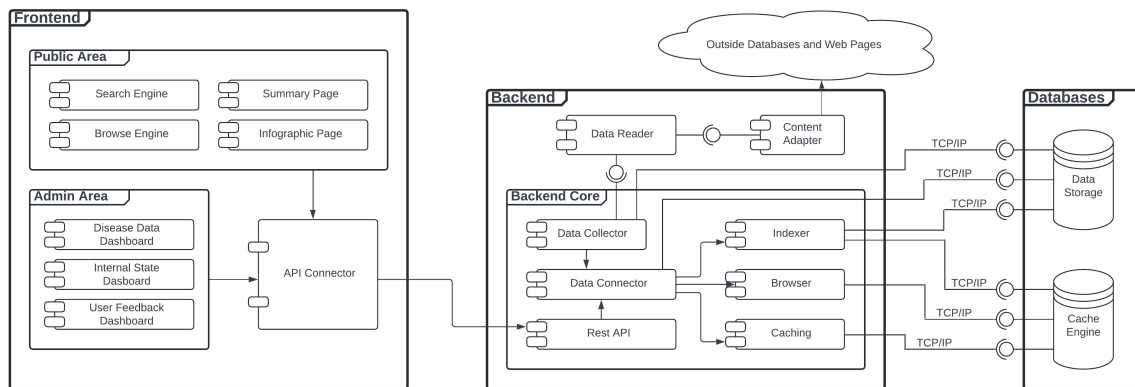


Figura 4.2: Enterprise Architecture

4.2.3 Third Architecture

In our third and final version, we fixed some issues essential to the application's overall structure.

It is important to note that the final version of our architecture comes from the problems we faced and raised from mentors and teachers, as well as continued research to provide the best system and be open to modifications in the future and possible improvements.

Based on a suggestion from our advisor, we decided to change our database structure due to the interconnected tables and data. Instead of using Elasticsearch and Neo4j, we switched to PostgreSQL, as recommended by our advisor. This may not seem as an obvious choice at first, but once we realise that the graphs are in fact tree-structured, then we can see that a relational model has a much better application than a graphs database, as there are no cycles and no multiple relations between nodes.

The choice between PostgreSQL and elastic search was derived from various factors, such as the facility of use of each database, the aggregation techniques to use, and PostgreSQL being primarily used as a primary data source, where PostgreSQL was proven useful only in retrieving fast results.

Given these considerations, the team decided to use Elastic Search only if the query time became extensive due to PostgreSQL's robustness.

For the scrappers, we utilized a plug-in architecture to simplify the addition of a new source and recollecting as much information as possible from different sources. We changed the form we were supposed to use to do this by running the scrappers all at once and providing a user-friendly GUI to access and disable those scrappers momentarily or entirely.

In a later phase, while testing, we discovered that PostgreSQL could not support more than 255 characters, so we implemented MongoDB tactically only in articles. This document is connected to others, source and disease, through the corresponding IDs, thus being able to store all information.

4.3 API DOCUMENTATION

The system contains a robust API that handles all requests regarding data processing, management and user logins. This documentation was done using SwaggerUI.

4.3.1 Admin

Admin		Operations pertaining to administrative operations	^
PUT	/api/admin/enableSource/{name}	enable source	▼
PUT	/api/admin/enableArticle/{id}	enable article	▼
PUT	/api/admin/editMedicalAdminName/{id}	Edit medical admin name	▼
PUT	/api/admin/disableSource/{name}	Disable source	▼
POST	/api/admin/testAggregate	Test the aggregation algorithm using external data	▼
POST	/api/admin/createMedicalAdmin	Create medical admin	▼
PATCH	/api/admin/changeDiseaseToOtherFamily/{omim}	Change disease to other family	▼
PATCH	/api/admin/changeArticleToOtherDisease/{id}	Change article to other disease	▼
PATCH	/api/admin/alterFamilyname/{id}	Alter family name	▼
GET	/api/admin/test	Test	▼
GET	/api/admin/testMedicalAdmin	Test medical admin	▼
GET	/api/admin/testAdmin	Test admin	▼
GET	/api/admin/safeAggregate	Safely test the aggregation algorithm on current data	▼
GET	/api/admin/role/{name}	Get admin role	▼
GET	/api/admin/numbSrc	Get number of sources	▼

Figura 4.3: Admin Controller (1)

GET	/api/admin/numbFamilies	Get number of families	▼
GET	/api/admin/numbDiseases	Get number of diseases	▼
GET	/api/admin/numbArticles	Get number of articles	▼
GET	/api/admin/medicalAdmins	Get medical admins	▼
GET	/api/admin/disabledSources	Get disabled sources	▼
GET	/api/admin/disabledArticles	Get disabled articles	▼
GET	/api/admin/clearFamilies	Clear families	▼
GET	/api/admin/aggregateDiseasesByFamily	Aggregate diseases by family, using an aggregation algorithm	▼
GET	/api/admin/activeSources	Get active sources	▼
DELETE	/api/admin/{id}	Delete an admin	▼
DELETE	/api/admin/deleteSource/{name}	Delete source	▼
DELETE	/api/admin/deleteMedicalAdmin/{id}	Delete medical admin	▼
DELETE	/api/admin/deleteArticle/{id}	Delete article	▼

Figura 4.4: Admin Controller (2)

4.3.2 Authentication

Authentication		Operations pertaining to authentication	^
POST	/api/auth/validate	Validate	∨
POST	/api/auth/signup	Signup	∨
POST	/api/auth/login	Login	∨

Figura 4.5: Authentication Controller

4.3.3 Disease Family

Disease Family		Operations pertaining to disease families	^
GET	/api/family/{name}	Get family by name	∨
GET	/api/family/substring/{name}	Get family that start with the inserted letters	∨
GET	/api/family/startsWithLetter/{letter}	Get family whose first letter matches the inserted letter	∨
GET	/api/family/refreshView	Refresh materialized view	∨
GET	/api/family/graph/{name}	Get graph info	∨
GET	/api/family/getArticleCountPerFamily/{name}	Get article count per family	∨
GET	/api/family/allFamilies	Get all families	∨

Figura 4.6: Disease Family Controller

4.3.4 Disease

Disease		Operations pertaining to diseases	^
GET	/api/disease/{omim}	Get disease by omim	∨
GET	/api/disease/all	Get all diseases	∨

Figura 4.7: Disease Controller

4.3.5 Source

Source		Operation pertaining to data sources	^
GET	/api/source/specific/{name}	Get source by name	∨
GET	/api/source/all	Get all sources	∨

Figura 4.8: Source Controller

4.3.6 Article

Article		Operations pertaining to articles	^
GET	/api/articles/{id}	Get article by ID	∨
GET	/api/articles/source/{sourceName}	Get articles by source	∨
GET	/api/articles/all	Get all articles	∨

Figura 4.9: Article Controller

4.3.7 Feedback

Feedback		Operations pertaining to feedback	^
PUT	/api/feedback/reject/{id}	Reject feedback	∨
PUT	/api/feedback/approve/{id}	Approve feedback	∨
POST	/api/feedback/save	Save feedback	∨
GET	/api/feedback/type/{type}	Get feedback by type	∨
GET	/api/feedback/structured	Get structured feedback	∨
GET	/api/feedback/solved	Get solved feedback	∨
GET	/api/feedback/id/{id}	Get feedback by id	∨
GET	/api/feedback/all	Get all feedback	∨
DELETE	/api/feedback/delete/{id}	Delete feedback	∨

Figura 4.10: Feedback Controller

4.4 WEB SCRAPING

4.4.1 Initial Method

Our data-scraping process is built on three fundamental premises that form the foundation of our method.

- **Accessing OMIM Data:** The Online Mendelian Inheritance in Man (OMIM) database is an invaluable resource for genetic information. OMIM provides a file called `mim2gene.txt` (available *here*) which contains a comprehensive list of OMIM numbers. These numbers indicate whether the entries are related to diseases. This file serves as the starting point for our scrapping efforts.
- **Converting OMIM Numbers:** Once we have the OMIM numbers, the next step is to translate them into actual disease names and their respective acronyms. The OMIM database allows us to make this conversion. This step is crucial because it transforms numerical data into meaningful medical terminology that can be used in further searches.
- **Gathering Additional Information:** With the disease names in hand, we can now contact other websites for more detailed information. The primary sources we use for this additional data are PubMed and Malacards. PubMed is a free search engine accessing primarily the MEDLINE database of references and abstracts on life sciences and biomedical topics (available *here*). Malacards is an integrated database of human

maladies and their annotations (available *here*). These sites provide many articles and summaries that can give us deeper insights into each disease.

These three steps create the basic flow of our initial scrapping method. Here is how it works in practice:

- **Step 1:** We start by downloading the '*mim2gene.txt*' file from OMIM. This file gives us all the OMIM numbers and indicates whether each entry is a disease. Entries related to moved, removed, or gene-specific information are ignored at this stage because we focus solely on diseases.
- **Step 2:** Using the OMIM numbers identified as diseases, we convert these numbers into their corresponding disease names and acronyms. This conversion is done through the OMIM database itself.
- **Step 3:** With a list of disease names, we perform search requests on PubMed and Malacards. These searches return a list of valid articles related to each disease. From these lists, we can request the full text of the articles, focusing mainly on the main body, summary, abstract, and other relevant sections.

By following these steps, we efficiently gather and compile detailed information about various diseases, leveraging multiple trusted sources to ensure the accuracy and depth of the data. This method allows us to build a comprehensive database that can be used for further research and analysis.

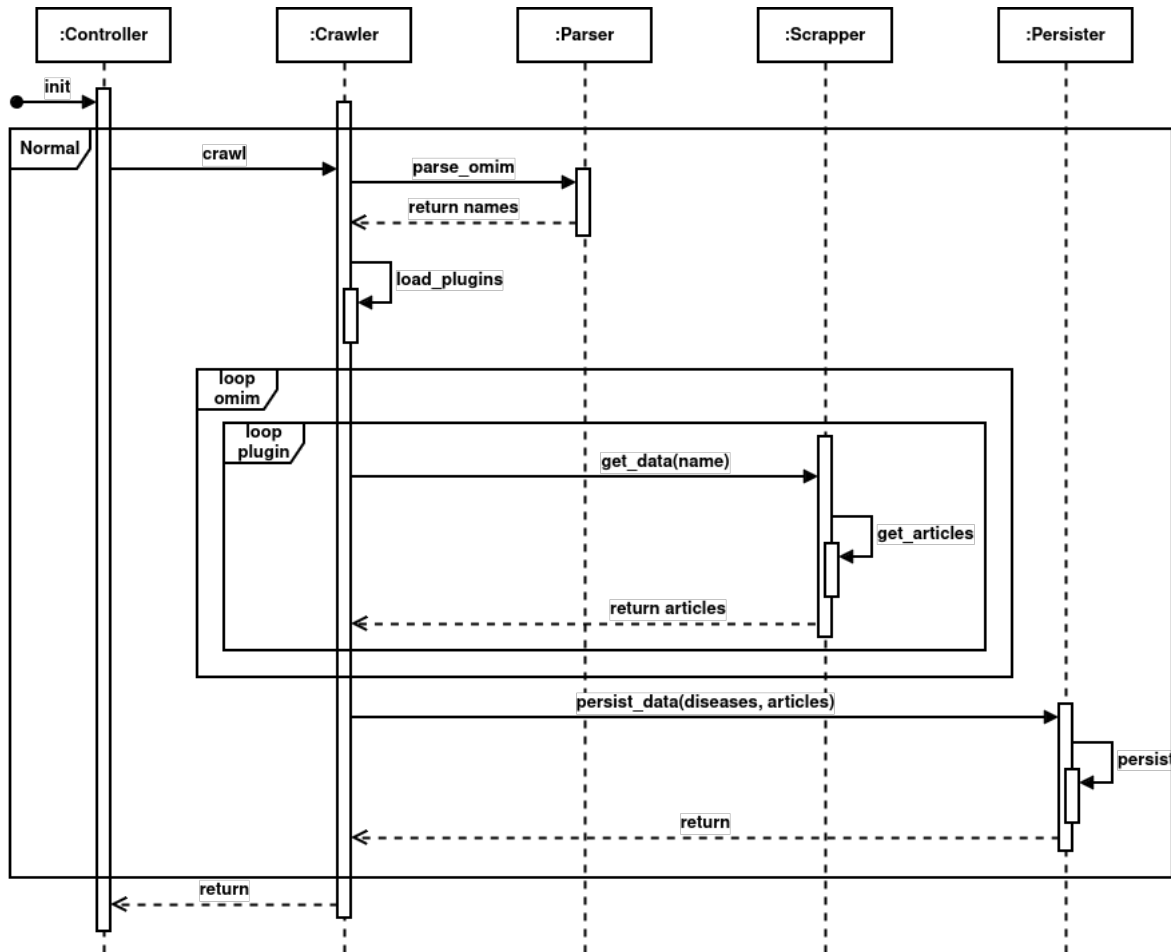


Figura 4.11: Outdated Scraping Process

4.4.2 Improved Method

After implementing our initial scraping method, we received a request to modify our approach to utilize a plugin architecture. The goal was to make our system more flexible and adaptable. With this new approach, an administrator could easily add support for scraping new websites by submitting a JSON file. This JSON file would define the scraping parameters and rules specific to the new website. Our system would then convert this JSON file into a Python class, which would be dynamically loaded and used to perform the scraping. This modular architecture allows for easy updates and the addition of new sources without requiring changes to the existing codebase. This flexibility ensures that our system can quickly adapt to new requirements and integrate additional data sources with minimal effort.

We used this new method to address an already identified problem. The initial separation of parsing, crawling, and data persistence into three distinct phases proved to be inefficient. Specifically, we faced issues with getting IP blocked during the parsing phase. The time spent on data persistence could have been used more effectively to avoid these bans. Therefore, we decided to intertwine all three processes so that they occurred in a more integrated manner rather than sequentially. This interleaving of tasks allowed us to process data more smoothly and reduce the risk of IP blocks. Additionally, this approach made it possible to divide the

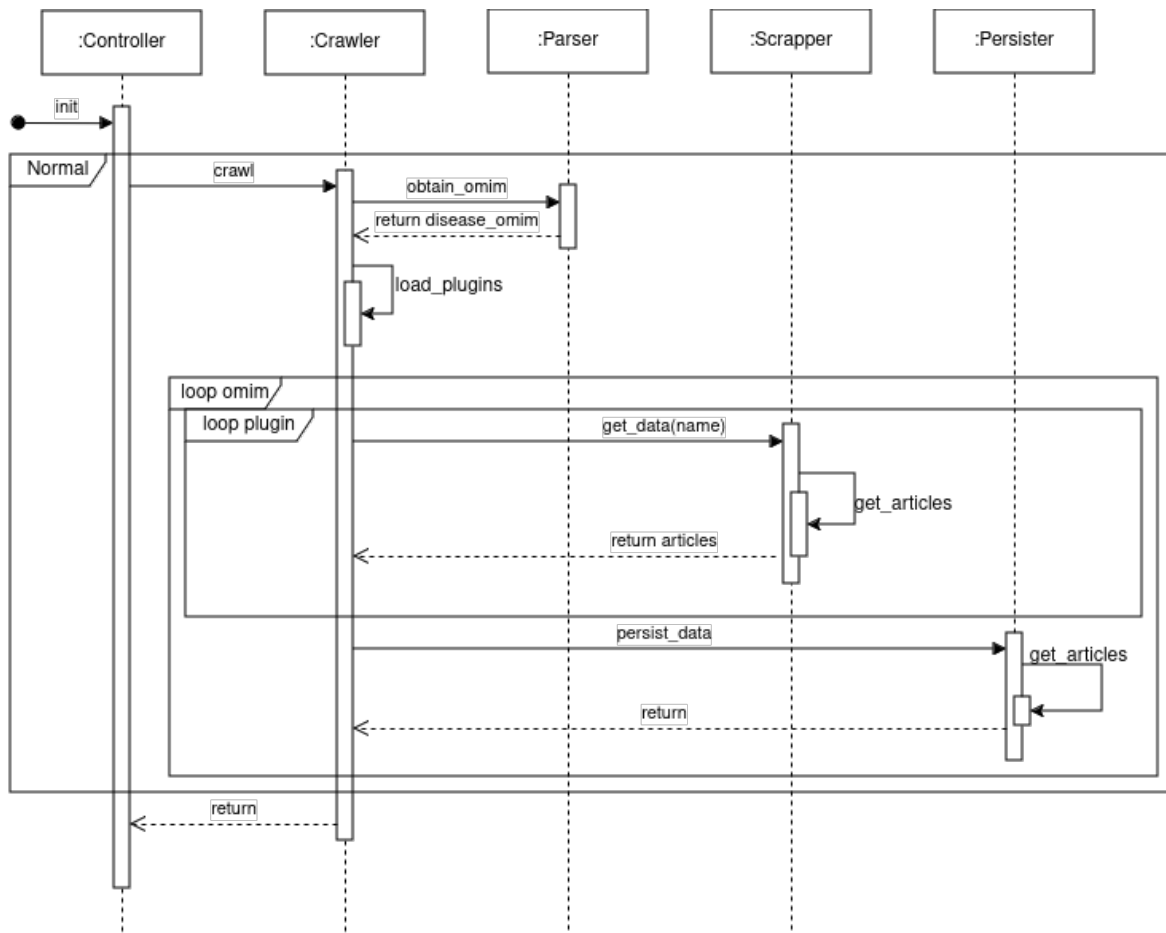


Figura 4.12: Scraping Process

collection tasks across different machines more easily. However, this was not the primary aim of our change. Since at the deployment machine we were limited to scraping from a single IP address, our main bottleneck was not the data processing but the rate at which we could work with our data sources. By combining the processes, we optimized our workflow and made better use of the available time to avoid detection and maintain continuous operation.

4.4.3 Temporary Method

In addition to improving our system, we also had to develop a temporary method to address some technical challenges we encountered. One significant issue was the need to fine-tune our scraping times. Initially, our scraping frequency was too aggressive, resulting in bans from several websites. This forced us to adopt a more controlled and methodical approach to scraping.

To address these challenges, we had to incorporate multiprocessing into one of our plugins. This allowed us to significantly reduce the time required for that specific plugin by around 40%. However, for other plugins, we had to implement processing sleep functions to slow down our requests and avoid being banned.

Overall, we initially estimated that the entire scraping process would require about 110 hours of continuous work, roughly equivalent to one workweek. Thanks to the improvements

we made, particularly the time saved by the optimised plugin, we managed to complete the task in approximately 64 hours. This matched our expectation of saving around 40% of the time on that one plugin, which was responsible for about 85% of the total workload.

These adjustments allowed us to balance efficiency with the need to avoid detection and bans, ensuring that our scraping activities could continue without interruption. The temporary method, while more rigid, proved effective in managing these challenges and helped us maintain our overall project timeline.

4.5 AGGREGATION ALGORITHM

Some of the thousands of rare genetic diseases can be grouped into families. To facilitate this aggregation, we developed an algorithm that takes all the diseases and creates disease families, naming them in the process.

The method we chose for determining which diseases belong together was based on the disease acronym. Around 70% of the diseases in OMIM have an associated acronym, which consists of an abbreviation of the disease that distinguishes genetic variations. Take, for example, MODY (Maturity-Onset Diabetes of the Young).

This disease has a noticeable genetic heterogeneity, meaning many genetic variations affect and give nature to other diseases. However, in most cases, because the mutations do not alter the disease characteristics much, the new disease maintains the same name but is attributed to a number. As such, these diseases take on the acronyms with an added numeral, which in the case of MODY results in MODY1, MODY2, MODY3, etc, all the way up to MODY14.

Based on this naming convention, our algorithm takes the acronym of each disease, strips it from all numerals and constructs the family name by matching each letter of the acronym to a word of the disease name. It then groups all diseases with the same family name and discards any family with less than two members.

This algorithm is good enough for our goals, grouping around 50% of all diseases into hundreds of families, but it has some flaws. The main one is that the estimated 30% of diseases that don't have an acronym are not aggregated.

In the following diagram, the symbol refers to the acronym.

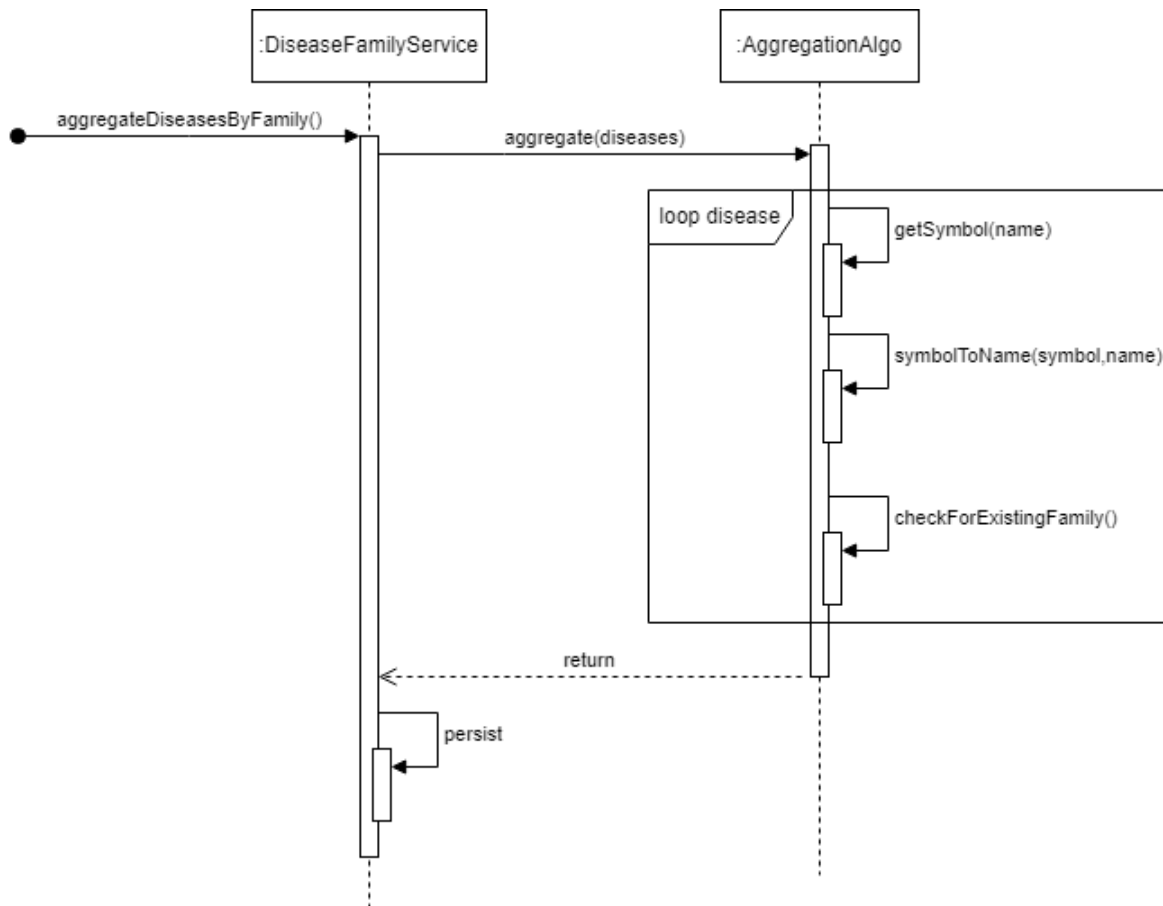


Figure 4.13: Algorithm Sequence Diagram

4.6 DATA SECURITY

For this project, the main web page can be viewed by anyone without needing a signup and login, but that is different for the Admin. Because an Admin and Medical Admin deal with sensitive information that, if wrongly accepted or denied, could have adverse effects, we implemented JWT tokens. Only the admin can access the admin area by logging in. The passwords are encrypted using the BCrypt Encryption algorithm. By using JWT tokens, the tokens are signed and randomized, ensuring more security.

4.6.1 Role Based Access Control

Throughout the project's development and with the different milestones and questions, there surged a need to add a Medical Admin. As explained above, in the different actors of the system, a medical admin can only do certain operations, unlike the Admin, who has full autonomy in the system. For that to be possible, in the storage of credentials, it is associated with a role so that it can be specified which endpoints which user can access.

4.7 DEPLOYMENT ARCHITECTURE

The tool we used to make the deployment of *DiseaseCard* was Docker, more specifically docker-compose, which allows for the containerization of each component of the platform.

Docker, which is massively used to run applications, runs using containers, a packaged format that stores all the code and dependencies, allowing it to run smoothly and quickly across different computing environments.

Using a docker-compose, the user can define and use multi-container applications and connect them into a network, as in this project, or wait for one to start the other.

The diagram is represented below. Note that for the deployment, we opted to separate the frontend from the backend and the databases to provide a modular architecture. This architecture is easier for deployment, easier to understand, and allows us to decouple the used containers. By ensuring this, we are fulfilling one of the non-functional requirements.

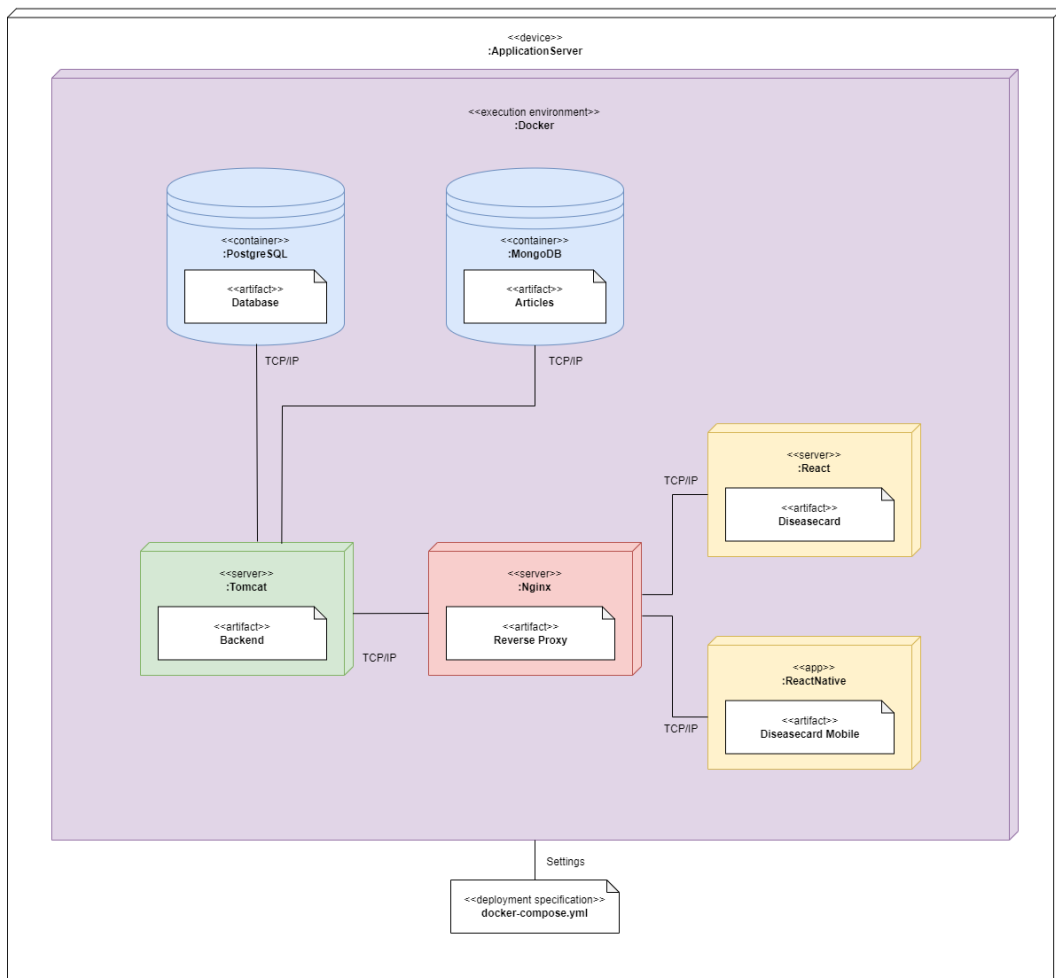


Figura 4.14: Deployment Diagram

Results

Throughout the previous chapters, we explored different requirements and details to make the product ideal for use. We have established personas and use cases, defined technologies to use in the product to make the requirements work, and the different ways the different personas could use the product.

5.1 DISEASECARD

On the main website of Diseasecard, we can access three different pages: Homepage, Browse Page and Disease Page.

5.1.1 Homepage

The homepage is the entry point to the platform, and it is where the user can search for all the disease families and diseases that are not aggregated. That component uses an auto-complete system to allow the user to navigate to the disease family page intended. At the top, we can find a navbar that we can use to access other pages, such as the browse page.



Welcome to Diseasecard!

Try the system by entering the name of a disease in the search field above or navigating through our Diseases Tab.

Figura 5.1: Homepage

5.1.2 Browse Page

The Browse page allows users to access and explore the alphabetically ordered list of all disease families on the platform. This page consists of a horizontal component that allows the user to select the chosen letter and a data table that displays the requested data. This table contains three columns: the first with the name of the disease, the second with the number of diseases of the respective family and the third containing the total number of articles in the family. Also, in the second column, if there is only one disease, it is not aggregated to any family. If the user clicks on a table row, as explained in the previous section, he will be redirected to the respective disease family page.

Families started with letter: **G** Filter by Name...

Name	Diseases	Articles
Gaba-Transaminase Deficiency	1	18
Gabriele-De Vries Syndrome	1	14
Galactorrhea	1	17
Galactosemia	4	44
Galactosialidosis	1	20
Gallbladder, Agensis of	1	20
Gallbladder Disease	4	45
Galloway-Mowat Syndrome	10	68

Figura 5.2: Browse Page

5.1.3 Disease Family Page

When the URL parameters redirect to the disease family page, the name is passed in. Then, the API is requested to return an object organized hierarchically: at the top, we have the name of the family (if there is one), followed by the diseases in that group. Each disease can have information from many data sources; for each source, there is a list of articles. The data can be displayed in three different strategies: "Bubble Graph,Hypertree Graph,"and "Accordion List."

Bubble Graph

The amCharts library was used for the bubble graph implementation. This component has great functionality that lets the user see more details about the specifics of the desired disease, removing irrelevant ones.

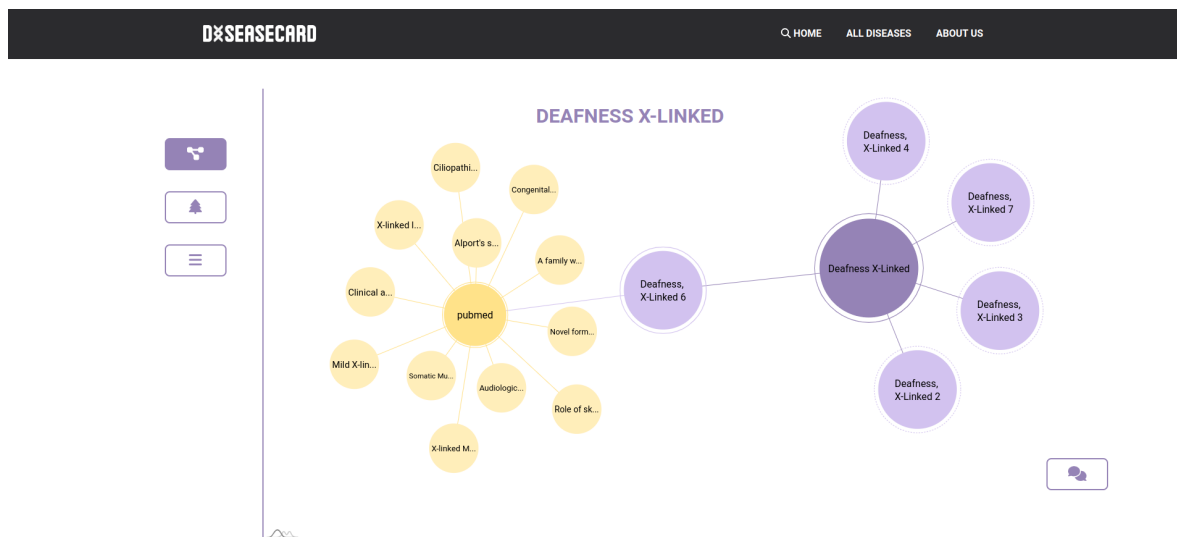


Figura 5.3: Bubble Graph

Hypertree Graph

It was implemented a hypertree graph with *jit hypertree* from Nicolas Garcia Belmonte. This element allows the user to zoom in on a specific disease or source to see more detail.

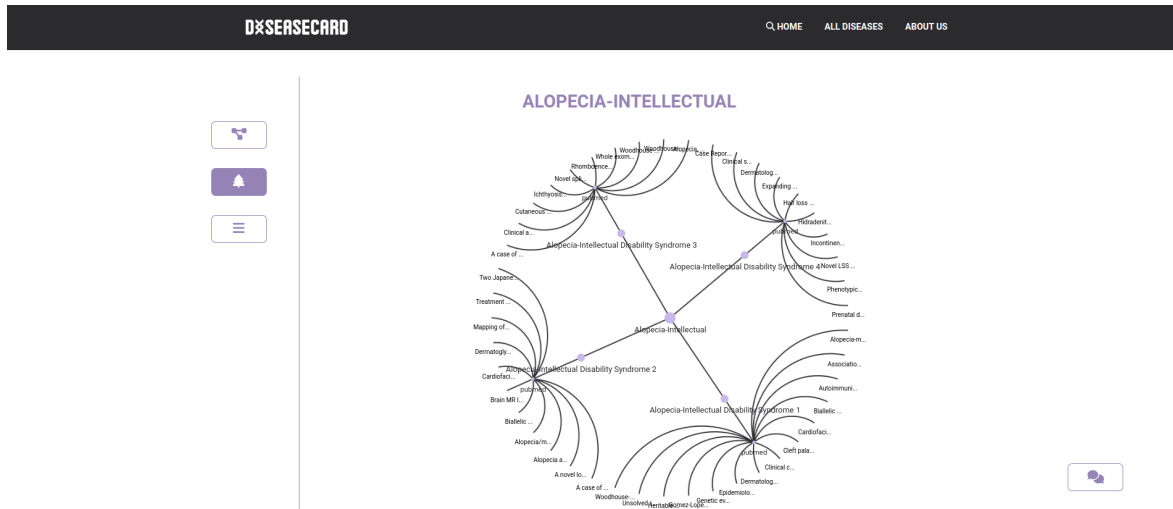


Figure 5.4: Hypertree Graph

Accordion List

To demonstrate the data in a list, we used the Accordion *Bootstrap* component to represent the different hierarchical levels. For more significant amounts of data, this is the most effective visualization method.

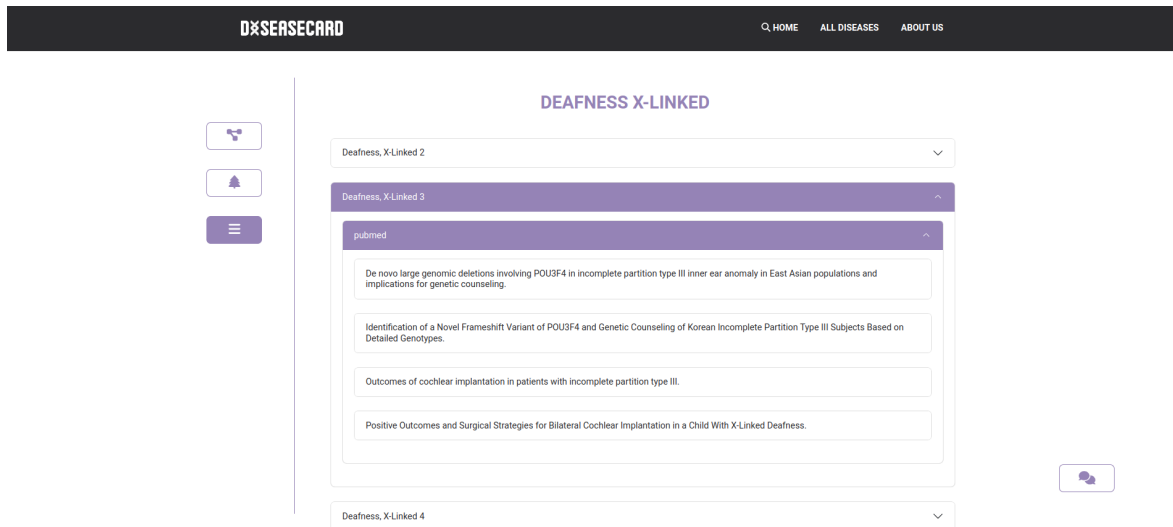


Figure 5.5: Accordion List

5.1.4 Article Page

When the user selects an article, a new component is showed. Beyond the title and content, the user can also check the last update of the article, see what is its source and navigate to the respective url.

The screenshot shows a web interface for 'DISEASECARD'. The top navigation bar includes 'HOME', 'ALL DISEASES', and 'ABOUT'. The article title is 'DEAFNESS, AMINOGLYCOSIDE-INDUCED' with a sub-title 'PARP-1-MODULATED AIF TRANSLOCATION IS INVOLVED IN STREPTOMYCIN-INDUCED COCHLEAR HAIR CELL DEATH.' and a last update date of '2024-05-25 20:51:14'. The abstract text describes the study on SM-induced cochlear hair cell death and the role of PARP-1. A 'Source | pubmed' link and an 'Article URL' are provided at the bottom. A feedback icon is visible in the bottom right corner of the article content area.

Figura 5.6: Article Page

5.1.5 Feedback Option

If a user finds information wrongly placed or wrongly associated, they can report it. The reports will later be analyzed by a medical admin and admin, and then they will be denied or accepted.

Disease Feedback

The user can report if a family name is incorrect or a disease is in the wrong family.

Family Name Not Correct Feedback

The user only needs to write the name the family should have, and then it will be visible to the admins.

Disease in Wrong Family

The user only needs to write the name of the family that a disease should have or suggest the creation of a new disease, and then it will be visible to the admins.

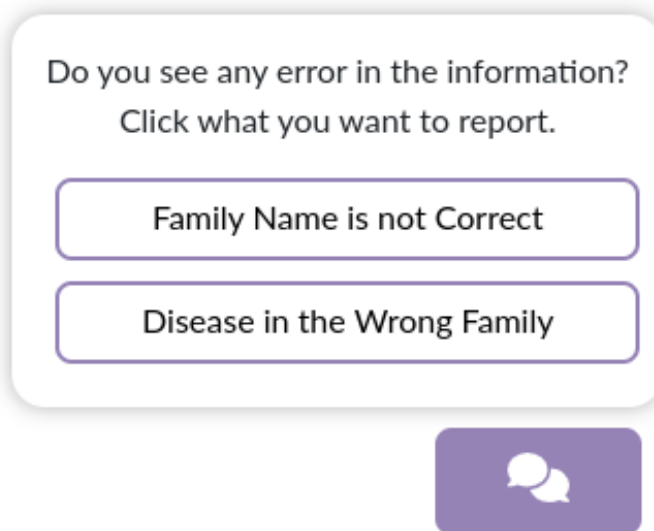


Figura 5.7: Disease Feedback

FEEDBACK - FAMILY NAME IS NOT CORRECT



As you may know, all the info about the Diseases is not from our responsibility, therefore, we cannot change it and if you notice something wrong with them you should inform the responsible of the information you read.

What we are actually responsible and able to change is the Family of the Diseases or create new ones, so if you notice that a Disease Family has an Incorrect Name or a Disease is in the Wrong Family.

We can as well change Articles location if you notice that they are in the wrong Disease. In the cases the Article is just Empty we can remove it.

You can help us by filling the forms of the problem you found.

Fill with the Disease Family that has a Incorrect Name!

3-hydroxy-3-...

Fill with the Name you think the Family should have!

Name...

Thank you!



Figura 5.8: Disease Family Name Not Correct

FEEDBACK - DISEASE IN THE WRONG FAMILY



As you may know, all the info about the Diseases is not from our responsibility, therefore, we cannot change it and if you notice something wrong with them you should inform the responsables of the information you read.

What we are actually responsible and able to change is the Family of the Diseases or create new ones, so if you notice that a Disease Family has an Incorrect Name or a Disease is in the Wrong Family.

We can as well change Articles location if you notice that they are in the wrong Disease. In the cases the Article is just Empty we can remove it.

You can help us by filling the forms of the problem you found.

Fill with the Disease that is in the Wrong Family!

3-hydroxy-3-Methylglutaryl-Coa Lyase Deficiency | H

Fill with the Correct Family of the Disease above!

Search...

Don't find the disease family pretended? Click [here](#) to suggest a new one!

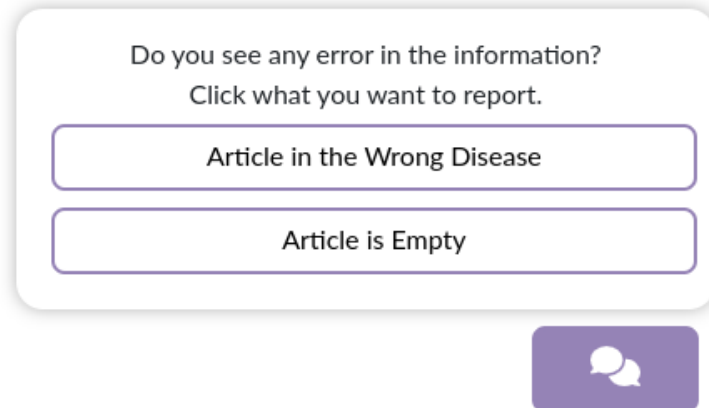
Thank you!



Figura 5.9: Disease in Wrong Family

Article Feedback

The user can report if an article is placed on the wrong disease or if the article is empty.



Do you see any error in the information?
Click what you want to report.

Article in the Wrong Disease

Article is Empty




Figura 5.10: Article Feedback

Article in Wrong Disease The user can report if the article is wrongly associated with a disease by fulfilling the name of the correct disease name.



FEEDBACK - ARTICLE IN THE WRONG DISEASE

As you may know, all the info about the Diseases is not from our responsibility, therefore, we cannot change it and if you notice something wrong with them you should inform the responsables of the information you read.

What we are actually responsible and able to change is the Family of the Diseases or create new ones, so if you notice that a Disease Family has an Incorrect Name or a Disease is in the Wrong Family.

We can as well change Articles location if you notice that they are in the wrong Disease. In the cases the Article is just Empty we can remove it.

You can help us by filling the forms of the problem you found.

Article selected automatically.

3_hydroxy_3_methylglutaryl_coa_lyase_deficiency

Fill with the Correct Disease of the Article above!

Search...

Thank you! 

Figura 5.11: Article in Wrong Disease

Empty Article

The user can report if an article does not have any information.

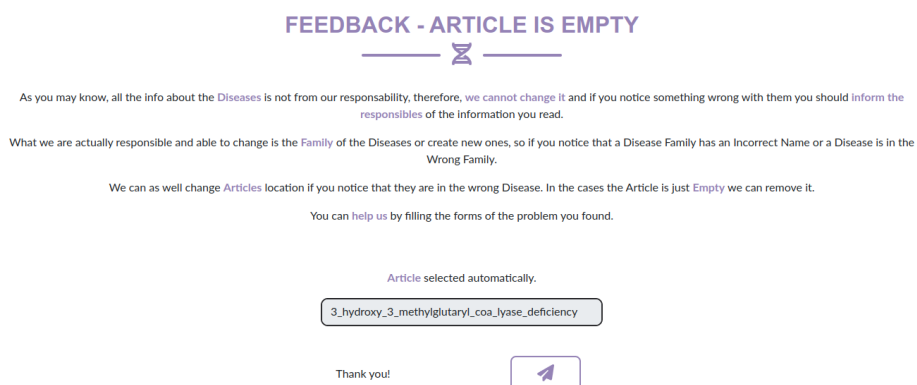


Figura 5.12: Empty Article

5.2 DISEASECARD ADMIN

5.2.1 DashBoard

In the dashboard section, the admin can visualize the database number of disease families, diseases, sources, and articles.

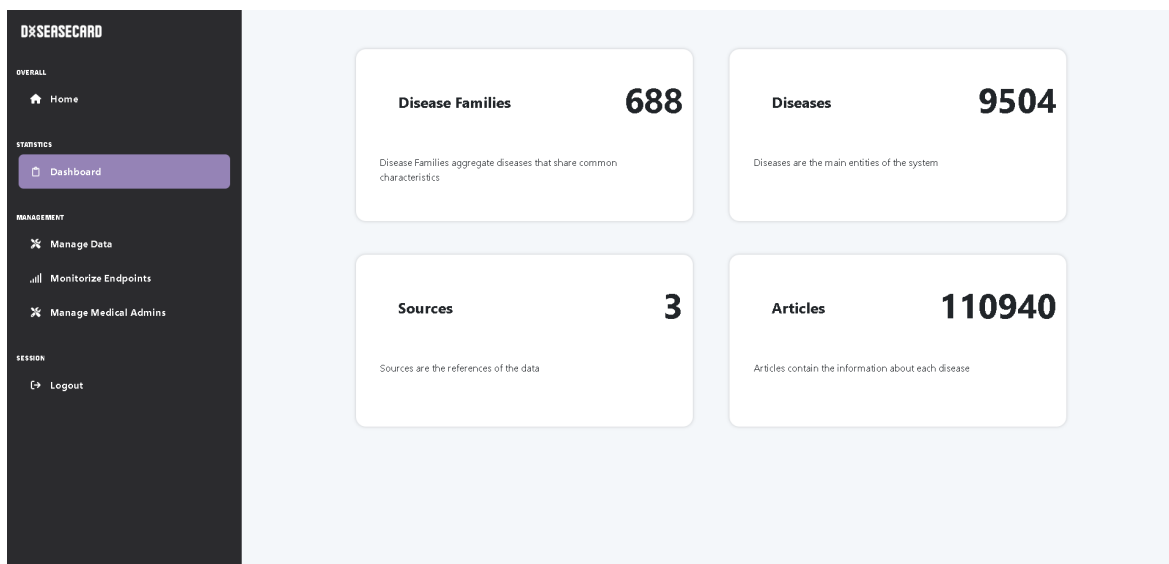


Figura 5.13: DashBoard Admin

5.2.2 Manage Data

On this tab, both the admin and the medical admin can check the feedback provided by the users and accept or deny it. It is also possible to visualize the feedback history.

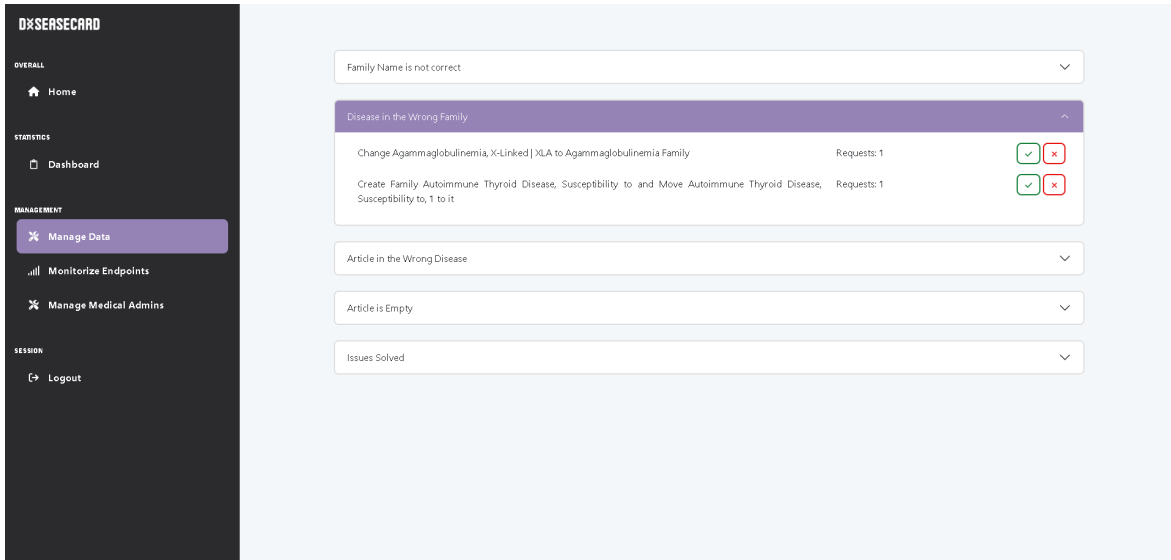


Figura 5.14: Admin Feedback Acceptance/Refusal Page

5.2.3 Monitorize Endpoints

This tab is where the admin can monitorize the articles and source endpoints. It is possible to see all the sources presented in the system and disable, enable or delete them. The same occurs with the articles, although in this case, it just shows the disabled articles (the empty ones) and enables or deletes them.

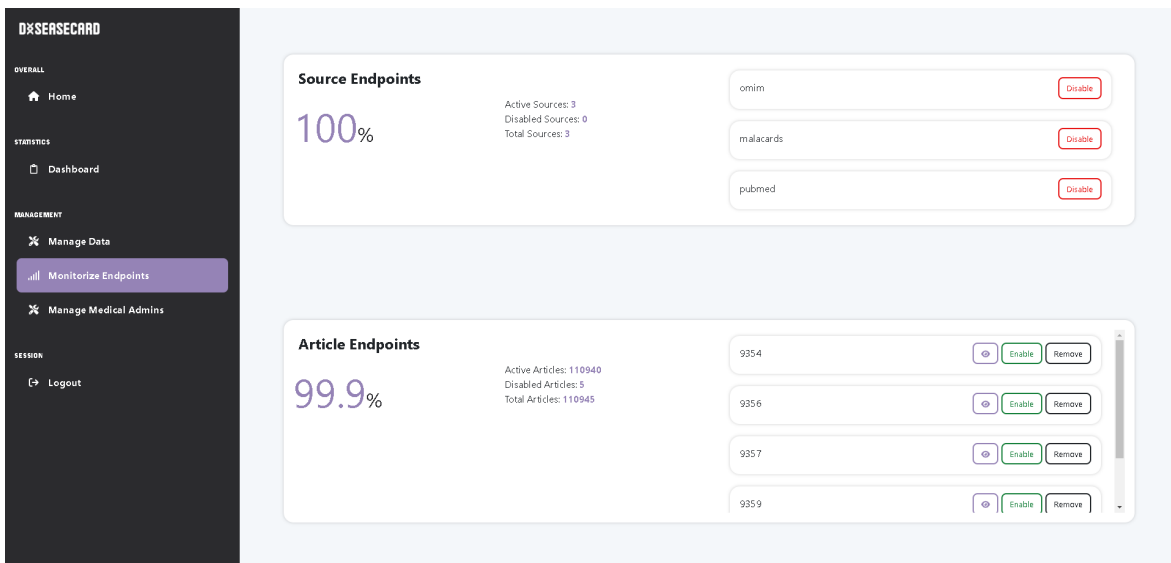


Figura 5.15: Admin Endpoint Monitoring Page

5.2.4 Manage Medical Admins

In this section, the main admin can create or delete accesses to the platform by medical admins.

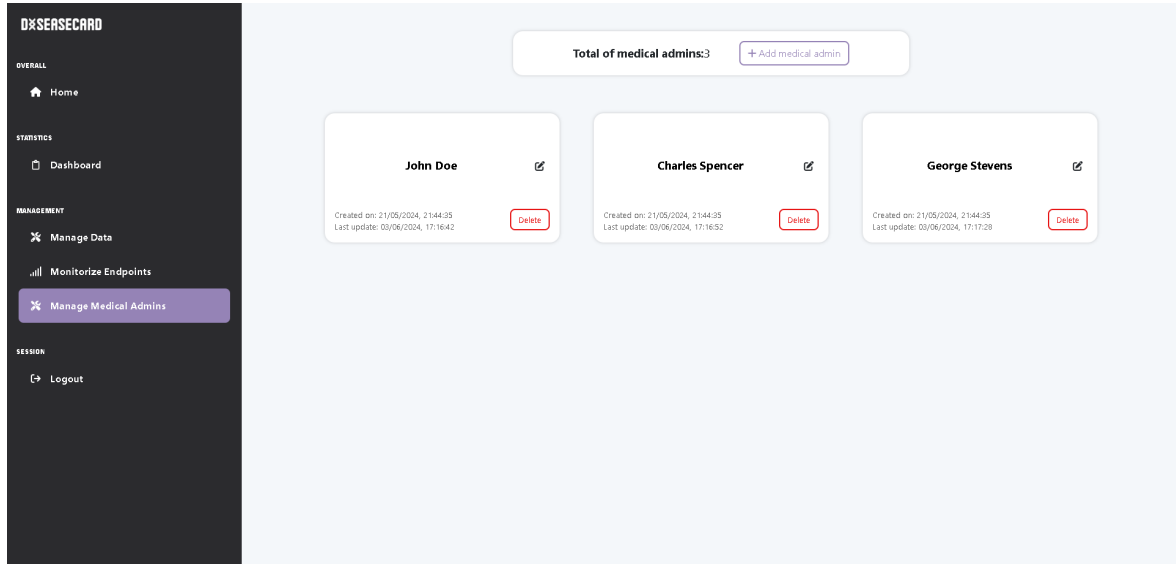


Figure 5.16: Medical Admins Management Page

5.2.5 Medical Admin View

When a Medical Admin log in the system, due to *RBAC*, he only can access to the **Manage Data** tab. This was made based on that a medical admin has no technical knowledge but instead has medic knowledge that a engineer doesn't. So a medical admin is good for the purpose of confirming users reports related to diseases.

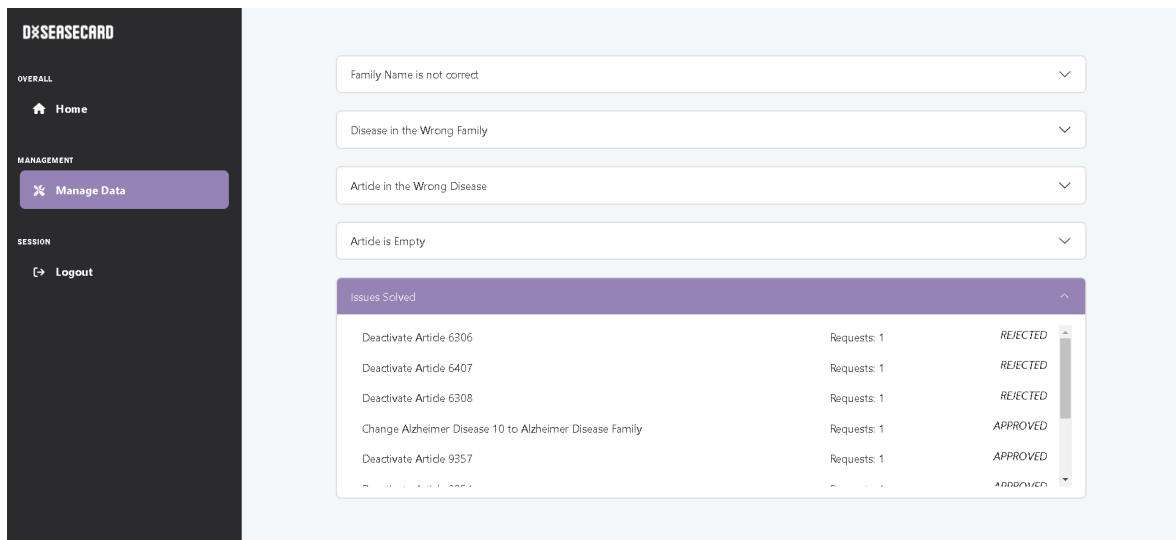


Figure 5.17: Medical Admin View

5.3 DISEASECARD APP

The mobile app is very similar to the main website, concerning the user workflow, since that the two systems only differ in relation to the admin platform, as this is not present on the app.

5.3.1 Disease Search

Just like in the website, as soon as the user enter the app, he will be facing a search bar, that will allow him to search through all the data that exists in the website too, all the disease families and diseases that are not aggregated. The search bar includes an auto-complete feature to help the user to easily find the disease family he is looking for.

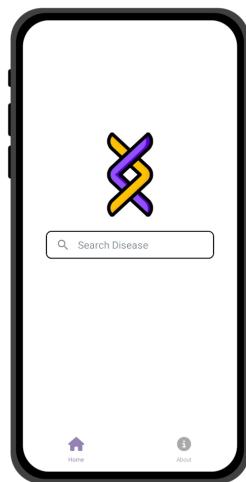


Figura 5.18: Disease Search

5.3.2 Disease Info

When the user chooses the disease family he wants to see, he will be redirected to correspondent page containing the same family name as a title. At the same time the app will do a API request to return the object which in the app, contrary to the website, will only be possible to see in one view, a Accordion List similar to the website one. The differences in those are were made so to the interface would be more user-friendly and effective. The Accordion is not the same in the aspect that the user only have to press the disease once and all the sources and articles will pop up instead of opening the data child by child.

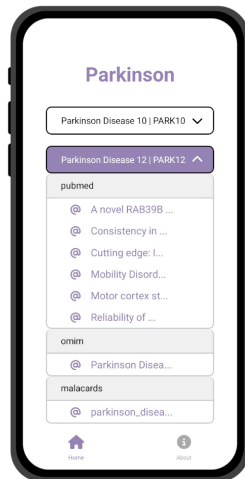


Figura 5.19: Disease Info

5.3.3 Article View

Then, after choosing which article to read, the API will once again be requested to return the data the user asked for. This time, it will be a request to get the article by its ID. After the request is completed, the article will be presented to the user in a very simple format, with just the text from the original source displayed in a linear font and size. In addition to the content, the user will also receive the source name, the original article URL, and the last updated date of the article. This information will be shown at the end of each article.

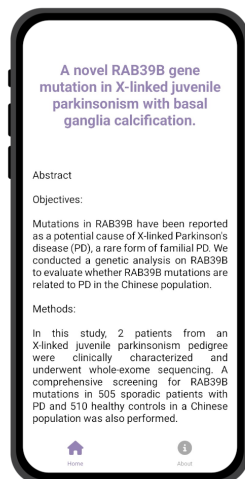


Figura 5.20: Article View

5.4 SUMMARY

In this chapter, we detailed the functionalities and design of the *Diseasecard* platform, both on the website and the mobile app. We explored the key pages, including the Homepage, Browse Page, Disease Family Page, and Article Page, describing their purpose and user interactions.

Additionally, we discussed the Feedback Option, enabling users to report inaccuracies, and the *Diseasecard* Admin interface, where administrators manage feedback and monitor endpoints. The mobile app mirrors the website's functionality, focusing on user-friendly navigation. The chapter provides a comprehensive overview of how *Diseasecard* facilitates disease information access and management for both users and administrators.

Analysis of Results and Future Works

6.1 PROJECT SUMMARY

In sum, this project allowed us to take a deep dive into the world of genetics and diseases, especially rare genetic diseases. It's astounding how sparse disease information is spread around and how difficult it is to locate information about one specific disease.

By implementing *DiseaseCard*, the user can access different articles from different sources and observe different families and diseases associated with that family.

If information is placed incorrectly, we count on a feedback option to alert the admins that that piece of information is wrong. This information can only be added to the website if the medical admin and admin accept it, with an option to deny it.

On the general page, a user can search for diseases from the first letter and text using the main page. On the admin page, the admin can add medical admins, turn articles and sources on and off, and monitor endpoint information.

6.2 FEATURES AND BENEFITS OF THE PRODUCT

As a product, *DiseaseCard* allows for consultation of articles and scientific papers related to a specific disease by searching for the disease family and then being redirected to the graphs page. The user may observe the different diseases in the family, the websites with papers on particular ones, and the actual papers themselves. The hyper-tree view, in contrast, allows for an overview of how different entities in the family are interconnected. For example, one may see how some diseases or sources have more papers than others. Last, the list view allows a more direct search of a specific article.

DiseaseCard also allows users to browse an all-disease list for users unsure of which disease they wish to select, following the same path as the regular user. We have an administrative platform incorporated into the website for an administrator. There, an admin may perform

activities like monitoring endpoints, where he can see which sources and articles are activated or deactivated. He may also perform actions like creating a Medical Administrator entity. This is the last entity with whom the administrator shares some actions.

These are the general dashboard, where both entities may consult the total number of families, diseases, sources, or articles; and data management, where they may, according to user feedback, modify data, whether this is disabling articles, changing articles from one disease to another, or even changing entire diseases or families. They may also create or edit disease families, having access to previous changes.

6.3 LIMITATIONS OF THE PRODUCT

6.3.1 Sources

The product must still show limitations due to high demand and complexity.

One of the major extra goals that was ultimately not achieved was allowing the administrator to know about the most accessed endpoints, mainly disease families and articles. This would allow for the eventual identification of articles with defects, whether they are not properly displayed or the information is simply wrong.

At this point, you come to another feature that was scrapped from the final product. That information would allow the administrator to run the web crawling process to re-fill the databases. Unfortunately, an IP block from our leading site forced us to abdicate from this method.

Another goal of this project was to gather articles and information from as many sources as possible to maximize the value created by our product. However, as it stands, *DiseaseCard* can only crawl in 3 sources, with most articles coming from PubMed.

The main reasons for this are the heterogeneous nature of the websites, the insane amount of actual data, and the impossibility of applying multiprocessing due to recurring IP banning/restricting at most websites.

6.4 POTENTIAL FUTURE IMPROVEMENTS

We would have liked to work on these features and improvements if we continued to sustain and develop the project.

6.4.1 Client-Side

Related Diseases

While a primitive version of this feature is already implemented in the system through disease family aggregation, the next step would be to gather more information about each disease and create a better network capable of relating diseases based on genetics, symptoms, geographic data, and more.

iOS Support

One step in scaling the product would be to introduce iOS support, complementing the already-in-place Android support.

Chat Bot - AI support

One of the ultimate and additional features was implementing a chat using AI to authenticate the search process. In there, a user could say keywords like the desired disease or possible symptoms, and the AI would help narrow it down to a single disease family.

6.4.2 Admin-Side

Endpoints Monitoring

As a system handling disease data that is highly used by professionals, it's important that the displayed information is, in fact, correct.

Our human review system allows us to tag on as wrongly displayed. Considering this, we would first and foremost build a proper monitoring dashboard that would allow the administrator to have complete information regarding disease families, diseases, and article accesses. This would effectively help decide the seriousness of reports, as ten reports out of 100 visits carry a different weight than ten reports out of 20 visits, and also which ones should be human-reviewed to ensure proper information display.

Security - 2FA

In a system where logged users have high responsibility, it is crucial to ensure that attackers cannot usurp users' accounts. For this, it would be of utmost priority to introduce a two-factor authentication method, like a time-based one-time password system.

6.4.3 Back-end Related

Database Refresh

We developed a web scraping system capable of not only gathering information and articles from sources but also creating the disease entities by searching through OMIM.

Referências

- [1] T. Richter, S. Nestler-Parr, R. Babela et al., «Rare disease terminology and definitions-A systematic global review: Report of the ISPOR rare disease special interest group,» en, *Value Health*, vol. 18, n.º 6, pp. 906–914, set. de 2015.
- [2] European Commission, *Rare Diseases*, https://research-and-innovation.ec.europa.eu/research-area/health/rare-diseases_en, Accessed: March 25, 2024, 2024.
- [3] J. K. Stoller, «The challenge of rare diseases,» en, *Chest*, vol. 153, n.º 6, pp. 1309–1314, jun. de 2018.
- [4] Orphanet, *Prevalence of Rare Diseases by Alphabetical List*, https://www.orpha.net/pdfs/orphacom/cahiers/docs/GB/Prevalence_of_rare_diseases_by_alphabetical_list.pdf, Accessed: March 25, 2024, 2024.
- [5] K. M. Boycott, A. Rath, J. X. Chong et al., «International cooperation to enable the diagnosis of all rare genetic diseases,» en, *Am. J. Hum. Genet.*, vol. 100, n.º 5, pp. 695–705, mai. de 2017.
- [6] C. C. Y. Chung, Hong Kong Genome Project, A. T. W. Chu e B. H. Y. Chung, «Rare disease emerging as a global public health priority,» en, *Front. Public Health*, vol. 10, p. 1028545, out. de 2022.
- [7] J. S. Mattick, M. Dinger, N. Schonrock e M. Cowley, «Whole genome sequencing provides better diagnostic yield and future value than whole exome sequencing,» en, *Med. J. Aust.*, vol. 209, n.º 5, pp. 197–199, set. de 2018.
- [8] J. Lord e D. Baralle, «Splicing in the diagnosis of rare disease: Advances and challenges,» en, *Front. Genet.*, vol. 12, p. 689892, jul. de 2021.
- [9] P. Lopes e J. L. Oliveira, «An innovative portal for rare genetic diseases research: the semantic Diseasecard,» en, *J. Biomed. Inform.*, vol. 46, n.º 6, pp. 1108–1115, dez. de 2013.
- [10] International Rare Disease Research Consortium, *International Rare Diseases Research Consortium Policies Guidelines*, <https://irdirc.org/wp-content/uploads/2020/05/IRDIRC-Policies-and-Guidelines-May-2020.pdf>, Accessed: April 1, 2024, 2020.
- [11] European Commission, *Support for international rare disease research to serve the IRDiRC objectives*, <https://cordis.europa.eu/project/id/305207/fr>, Accessed: April 1, 2024, 2016.
- [12] F. M. Sequeira, «Recuperação e visualização de informação de Doenças Raras,» tese de mestrado, Universidade de Aveiro, 2022.
- [13] P. Lopes e J. L. Oliveira, «COEUS: “semantic web in a box” for biomedical applications,» en, *J. Biomed. Semantics*, vol. 3, n.º 1, p. 11, dez. de 2012.

