Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2011

José Manuel Santos Melo

OralCard: Sistema de Informação Web para a Saúde Oral

OralCard: Web Information System for Oral Health

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2011

José Manuel Santos Melo

OralCard: Web Information System for Oral Health

OralCard: Sistema de Informação Web para a Saúde Oral

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática (M.I.E.C.T.), realizada sob a orientação científica do Professor Doutor José Luís Guimarães Oliveira, Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho à minha família.

o júri

presidente	Professor Doutor Armando José Formoso de Pinho Professor Associado com Agregação do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
vogais	Professor Doutor António Manuel de Jesus Pereira Professor Coordenador do Departamento de Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria
	Professor Doutor José Luís Guimarães Oliveira Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos

Um obrigado especial ao Professor José Luís Oliveira pela minha integração no grupo de trabalho de Bioinformática, no Instituto de Engenharia Electrónica e Telemática de Aveiro (IEETA).
Um obrigado especial ao Pedro Lopes e ao Joel P. Arrais, que me ajudaram com grande dedicação no desenvolvimento deste trabalho.
Um obrigado ao Nuno Rosa pela ajuda na parte biológica e científica do trabalho.
Este projecto não estaria concluído sem reconhecer a ajuda prestada pelos meus pais e irmãs, não só na dissertação como em todo o percurso académico na Universidade de Aveiro.

palavras-chave

Sistema informação web, Saúde oral, Integração de dados, doenças, proteínas, ontologias

resumo

Os sistemas de informação na web assumem-se cada vez mais como um recurso indispensável para os que estudam as ciências biomédicas. Uma das áreas de estudo destas ciências incide na cavidade oral e nas proteínas que nela residem.

Existem variadas plataformas online que permitem a pesquisa de dados específicos a microorganismos e a proteínas associadas, mas estes dados são genéricos e não são desenhados para casos de estudo específicos.

Este trabalho tem como objectivo desenvolver uma estratégia e um protótipo para o armazenamento de informação relacionada com a cavidade oral, visando a sua utilização em investigação. Uma preocupação diferenciadora prende-se com o objectivo de integrar dados obtidos experimentalmente com referências existentes na web e estudadas por outras entidades.

O protótipo desenvolvido permite aos investigadores na área das ciências biomédicas, sem conhecimentos específicos em bases de dados, pesquisar proteínas, doenças e genes, e integrar novos resultados de ensaios na base de dados existente.

keywords

Web information system, Oral health, Data integration, Diseases, Proteins, Gene Ontologies

abstract

Information systems on the web are becoming important resources for those studying biomedical sciences. One area of study of these sciences focuses on the oral cavity and on proteins that reside in it.
Several online platforms provide specific knowledge on multiple microorganisms and associated proteins, but these are generic and are not designed for specific case studies.
This work aims to develop a strategy and a prototype for the storage of information related to the oral cavity, aiming their use in research. It will integrate data collected from experimental results with existing references on the web and explored by other entities.
The prototype allows researchers in the biomedical sciences, without particular expertise in databases, searching for proteins, genes and diseases, and integrating new test results in the existing database.

Contents

List of Acronyms v
List of Figures ix
List of Tables xi
1. Introduction 1
1.1 Motivation and Context 1
1.2 Objectives 2
1.3 Dissertation structure 3
2. Background Concepts and State of the Art 5
2.1 The emergence of data integration 5
2.1.1 Database Abstraction Layer 5
2.1.2 Object-relational Mapping 8
2.2 Software Architectural Patterns 10
2.2.1 Model-View-Controller 11
2.3 Software Development Methodology 13
2.3.1 Rapid Application Development 13
2.4 Web Application Frameworks (Full-stack) 14
2.4.1 Ruby on Rails 15
2.4.2 Symfony – Open Source PHP Web Framework 16
2.4.3 Liferay – Open Source Enterprise Portal 18
2.4.4 Molgenis 19
2.4.5 Conclusions 20

2.5 Presentation Frameworks for Web Applications (Frontend) 22
2.5.1 Stripes 23
2.5.2 Google Web Toolkit 24
2.5.3 ZK – Open Source AJAX 25
2.5.4 Conclusions 27
3. Context of the Problem 29
3.1 Data Integration 29
3.1.1 Data Warehousing 30
3.1.2 Data Integration Alternatives 32
3.1.3 Conclusions 35
3.2 The Oral Cavity 35
3.2.1 Proteins 37
3.2.2 Diseases 38
3.2.3 Pathways 39
3.2.4 Gene Ontology 40
3.3 Summary 41
4. Work Implementation 43
4.1 Backend Development 43
4.2 Database Architecture 46
4.3 Importing Data 49
4.4 Frontend Development 55
4.5 Summary 61
5. Results 63
5.1 Importing Data 63
5.2 Frontend 65
5.2.1 The Home Page 65
5.2.2 Protein Search and Details 67
5.2.3 Disease Search and Details 75
5.2.1 Direct access using URL Bindings 77
5.3 Summary 78

6. Conclusions and Future Work 79
7. Appendix 83
  7.1 Appendix A - CD 83
8. References 85

List of Acronyms

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BRENDA	Braunschweig Enzyme Database
CFML	Coldfusion Markup Language
DAO	Data Access Object
DBMS	Database Management System
DMD	Duchenne Muscular Dystrophy
DOCX	Microsoft Word File Format
DOM	Document Object Model
DW	Data Warehouse
EBI	European Bioinformatics Institute
EC	Enzyme Commission
EDA	Enterprise Data Access
EJB	Enterprise JavaBeans
EMBL	The European Molecular Biology Laboratory
EMP	Embden-Meyerhof Pathway
ESB	Enterprise Service Bus
ETL	Extract, Transform, Load
GB	Gigabyte
GO	Gene Ontology
GUI	Graphic User Interface
GWAS	Genome-Wide Association Studies

GWT	Google Web Toolkit
HGNC	HUGO Gene Nomenclature Committee
HMP	Human Microbiome Project
HOMD	Human Oral Microbiome Database
HTML	Hypertext Markup Language
IEETA	Instituto de Engenharia, Electrónica e Telemática de Aveiro
IIS	Internet Information Services
JDBC	Java Database Connectivity
JMS	Java Message Service
JPA	Java Persistence Architecture
JSON	JavaScript Object Notation
JSP	JavaServer Pages
KEGG	Kyoto Encyclopedia of Genes and Genomes
MMP8	Matrix Metallopeptidase 8
MOLGENIS	Molecular Genetics Information System
MPW	The Metabolic Pathways Database
MVC	Model View Controller
NCBI	National Center for Biotechnology Information
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
OMIM	Online Mendelian Inheritance in Man
ORM	Object-relational Mapping
PDB	Protein Data Bank
PharmGKB	The Pharmacogenomics Knowledge Base
PHP	Hypertext Pre Processor
QTL	Quantitative Trait Locus
RAD	Rapid Application Development
RAM	Random Access Memory
RCSB	Research Collaboratory for Structural Bioinformatics
REST	Representational State Transfer
RMI	Remote Method Invocation
SMART	Simple Modular Architecture Research Tool

SOAP	Simple Object Access Protocol
SQL	Structured Query Language
StAX	Streaming API for XML
TDS	Transaction Data Stores
UCP-PV	Universidade Católica Portuguesa – Pólo de Viseu
UML	Unified Modeling Language
UniProt	The Universal Protein Resource
UniProtKBAC	The UniProt Knowledgebase Accession Code
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	The World Wide Web Consortium
WAF	Web Application Frameworks
WSDL	Web Service Definition Language
XHTML	Extensible Hypertext Markup Language
XLSX	Microsoft Excel File Format
XML	Extensible Markup Language
XUL	XML User Interface Language
ZK	ZKoss
ZUML	ZK User Interface Markup Language

List of Figures

Figure 1: Model of a centralized database architecture 6

Figure 2: Model of a client-server architecture... 7

Figure 3: The 3-tier architecture 8

Figure 4: ORM in use in one of many apps using a database 10

Figure 5: The Model-View-Controller Architecture.. 12

Figure 6: Validation configuration using Stripes 23

Figure 7: Central Aspects of the GWT framework 25

Figure 8: The ZK loader, the ZK AU engine, and the ZK client engine 26

Figure 9: Data Warehousing Architecture 31

Figure 10: Molgenis Architecture 44

Figure 11: Organism Description Structure 44

Figure 12: Generated Web Interface using Molgenis 45

Figure 13: The Identifiable entity described in XML 47

Figure 14: The Organism entity described in XML 47

Figure 15: Class diagram for Molgenis generator 48

Figure 16: Excerpt of a UniProt XML file 51

Figure 17: Registration of researched UniProts on XLSX file (client version) ... 52

Figure 18: PDB XML file for Structure Id 1A85 53

Figure 19: Excerpt of XLSX file containing other organisms 54

Figure 20: Example of a query to the UniProt REST web service 54

Figure 21: OralCard Frontend Architecture 56

Figure 22: REST Query on a specified protein 57

Figure 23: Using Stripersist for reading from database 59

Figure 24: Using the @*Repository* annotation in the ProteinDaoImpl 60
Figure 25: Using the @*SpringBean* annotation in the *BaseActionBean* 60
Figure 26: Importing proteins without multithreading.. 64
Figure 27: Importing proteins using multithreading 64
Figure 28: The OralCard home page.. 66
Figure 29: Suggestions in the home page while searching 66
Figure 30: Specifying the organism entity while searching................................. 67
Figure 31: The protein search web page 68
Figure 32: Protein Details 68
Figure 33: Phanter frame... 69
Figure 34: List of PDBs related to the protein 70
Figure 35: PDB's sequence image... 70
Figure 36: PDB Structure and link for its explorer 71
Figure 37: The SMART tool showing the current protein................................... 71
Figure 38: The STRING protein interaction tool 72
Figure 39: The PharmGKB frame... 73
Figure 40: The BRENDA frame 73
Figure 41: The References tab 74
Figure 42: The Source relations tab 75
Figure 43: The disease search web page 76
Figure 44: The disease details web page with related proteins............................ 76
Figure 45: A KEGG pathway related to the selected disease 77

List of Tables

Table 1: Comparison of Web Application Frameworks 22

Table 2: Comparison between frontend development frameworks 28

1. Introduction

1.1 Motivation and Context

The growing quantity of information available online is carried out in an increasingly accelerated way. In order to be processed, this increased amount of data requires a constant development of computer applications that must adapt to increasingly complex requirements, particularly those related to integration of heterogeneous data and composition of distributed services.

Oral health is an area of research where these problems are particularly relevant. Being a very specific area of study, researchers are faced with many problems in obtaining clinically relevant information concerning the oral cavity using an easy and transparent way. This information must be stored and managed using tools that should provide the user with functionalities to retrieve, store, and search this data. These tools are designated by databases or more correctly, by database management systems (DBMS) [1].

Usually, databases for molecular biology are centered either on a specific organism, such as SGD for Saccharomyces, or on specific research topic, such as STRING for protein-protein interaction. In addition, databases like Entrez or Uniprot play a major role as hubs of biomolecular information, storing data from multiple topics and several organisms.

Despite this effort to create long lasting hubs of biomedical data has been very successful, one should not ignore the major contribute that many more specific databases provide to the actual state of science. They are of special interest for small communities that share common research interests. Examples include the DMD database, specialized on Duchenne Muscular Dystrophy, or the HEART specialized on heart diseases.

Aware of the redundancy of features shared by many of those databases, and of the lack of technical expertise from the curators, several frameworks have been proposed to ease the task of deploying new databases. Examples include LOVD [2], specialized on annotating locus specific databases, GMODWeb [3] for organism specific databases, or Molgenis [4] that allows deploying more generic biomedical databases. Despite the validity of those frameworks, there is none focused on simulating the behavior of a single human organ or set of adjacent organs. This need to partition the data of the “whole” human system is relevant because, on the one hand, it reduces the time and resources involved in searching, processing and curating information, and on the other, it facilitates the use of algorithms to retrieve biologically meaningful results.

Following this need emerged, in partnership with the pole of the Catholic University of Viseu, the OralCard project.

OralCard will be a fundamental resource for salivary diagnostic processes of protein biomarker studies of health and disease based on the analysis of saliva samples. With this information system, clinical samples from patients’ saliva may be better analysed contributing to improved diagnostic methods and to the development of more effective therapies.

1.2 Objectives

The project’s main objective is the development of a web information system, focusing on oral health but useful for both researchers and dentists. A comprehensive integrated resource of the saliva proteins, currently missing in the field of oral biology, would enable researchers to understand the basic constituents, diversity, and variability of the salivary proteome, allowing the definition and characterization of the human oral physiome. This will be achieved at two levels:

- For the application developer, a proprietary database and a set of tools to retrieve biomolecular information from the major platforms, like NCBI and UniProt;
- For the end-user, a Web portal (OralCard) directed to non-expert users, like researchers and dentists, with a set of tools for searching and filtering data from the database, and the possibility to add new information to it.

Through OralCard Web Portal, users will be able to perform their queries and search among a list of provided results. For each entity (proteins, diseases, gene ontologies), users will be able to consult and analyze a list of dependencies and information retrieved from other major databases.

To demonstrate the usefulness of this project we also present its application in the oral cavity research domain. A platform designed to integrate protein data related to this field will be implemented. This will include salivary proteins obtained in proteomic studies by different research groups, as well as proteins potentially produced and excreted by microorganisms assigned to the oral cavity. The ultimate goal is to present a tool for the community that contains accurate, manually curated and updated data regarding the oral cavity, to enable interactions, categorization and exploration.

OralCard Web Portal should offer a user-friendly interface. Ideally, the portal should take advantage of the latest technologies to ignore the concept of static pages, to give the users a fast search engine and ready to use at any time, and provide a very intuitive interface.

1.3 Dissertation structure

This dissertation is divided into the following chapters, excluding this one:

Chapter 2 – Background Concepts and State of the Art, presents all the tools and frameworks that were considered and studied before choosing the best to proceed with the project itself;

Chapter 3 – Context of the Problem, discusses the requirements for the platform, alternatives for data integration, and the reason why a specific solution was chosen instead of others. This section presents the motive and importance of implementing the presented solution to the oral cavity, as the scientific and biological topics needed to understand the current problem;

Chapter 4 – Work implementation, divided into three main topics, presents the database architecture, the backend architecture, the frontend architecture, the decisions made concerning to the backend implementation, as well the decisions and implementations made concerning to the final frontend created to tackle the problems discussed earlier;

Chapter 5 – Results, presents the measures made while importing biomolecular data into the proprietary database and some screenshots of the final web portal;

Chapter 6 – Conclusions and Future Work, presents the review of the most important aspects during the development of this work, discusses the results achieved and presents some of the lessons learned as well as possible future work within this area of research.

2. Background Concepts and State of the Art

2.1 The emergence of data integration

The term *data integration* took place when databases began to be used everywhere. One particular use case is related to the biological databases, where information increases every day from new achievements in research. With the arrival of high-throughput methods, the field of biology is increasingly faced with the problem of storing, indexing and retrieving the innumerable data types and sources available. Also, the rise of the Internet has meant that data is accessible over the web, although in differing formats and semantics.

The goal of data integration is to incorporate these web-enabled biological resources that reside in multiple and different sources, and assemble it in a unified way. Examples are relational databases, ontologies and XML repositories [5].

There is not a universal approach to data integration, and many of the techniques that are being used by IT expertise are still evolving.

Data integration focuses mainly on databases, which are an organized collection of data. Just like a file system, a database is an organizational structure for data, making it easy to search, to access and to manipulate.

2.1.1 Database Abstraction Layer

There are several ways to establish a database. Three types of architecture will be referred: the centralized database architecture, the client-server architecture and the 3-tier architecture.

The centralized database architecture (Figure 1) is a design that was implemented in previous systems, which ran on mainframe computers to provide processing power for all the most common functionalities, such as application, user interfaces and DBMS functionalities. Most users accessed these functionalities using a computer terminal that did not have almost any processing power. In order to concentrate all the treating in one side, this was the kind of architecture used [1].

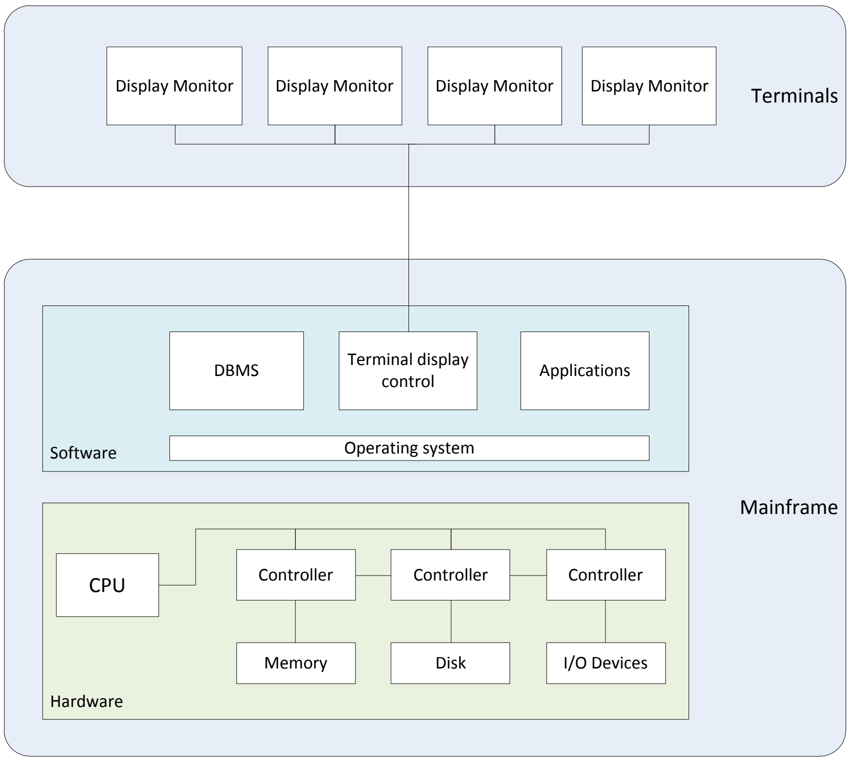


Figure 1: Model of a centralized database architecture

Over time, users replaced their machines with computers that had some processing capability, generating a new type of architecture: the client-server architecture (Figure 2). This design is built on a framework that contains multiple computers that are connected using a network. In this type of planning, the client is the entity that can provide the user interface and some local processing; the server is the entity that is responsible for providing services to the client, which in this particular case refers to a database layer access [1].

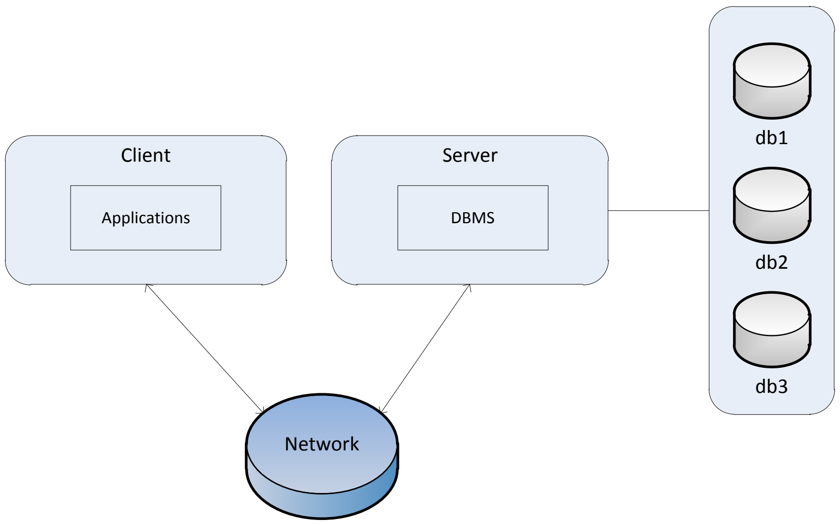


Figure 2: Model of a client-server architecture

This architecture is currently being used by most of the online systems, but there is another type of design that has been expanding in the last years and over the time may replace the client-server model: the 3-tier architecture (Figure 3). In this type of design, functionalities are even more isolated than in the previous model. The core design shows that the two main entities (client and server) cannot connect directly. A middleware layer is used to do this job. The main purpose is the client being able to send requests in the form of middleware processes requests, which are routed to the server and interpreted. A response from the server goes the same way, which is then filtered and the formatted data is finally sent to the client [1].

According to Heerschop [1], “this is the model that better fits the growth of the new generation Internet and the shift from static to dynamic pages”. In this model, the client corresponds to a web browser, just like Google Chrome or Mozilla Firefox, and the server is a DBMS implementation. The middleware layer usually is a web server like Apache Tomcat or Internet Information Services (IIS) from Microsoft.

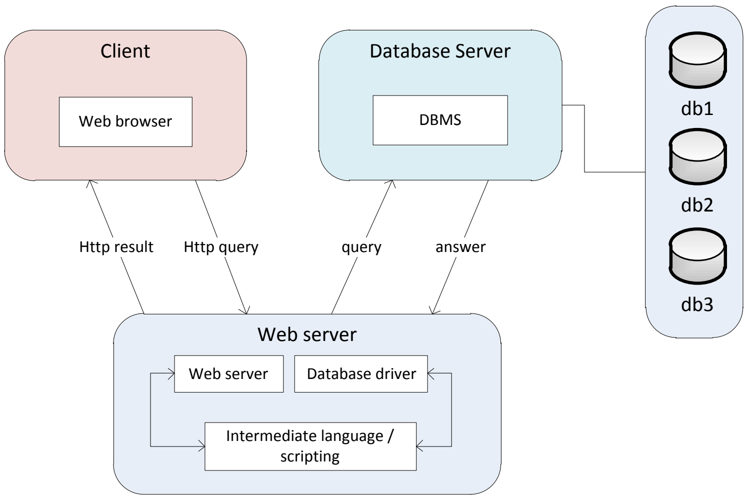


Figure 3: The 3-tier architecture

The use of dynamic content over the Internet is currently increasing, along with the number of pages viewed in the Web. Web servers are being used more often, not only for providing information to their staffs, but for administration purposes too. In resume, the increasing amount of data that is being stored and accessed is a reality, and this functionalities are possible due to network communications [1].

Most of the current online database architectures are not based in centralized proprietary systems, but are based in the last two referred models: client server or 3-tier models. This means that there is the need to work on methods necessary to access information from the DBMS. All database vendors, like MySQL or Microsoft SQL Server, have been developing their own methods for accessing data from these management systems, including property languages and APIs used to provide connectivity, such as JDBC (Java Database Connectivity) or ODBC (Open Database Connectivity) [1]. These plugins make possible the access from a web application to a database system. Web developers only have to understand how this connection to the DBMS can be done, concentrating most of the efforts on improving web application functionalities. This leads to a new concept, the *object-relational mapping*.

2.1.2 Object-relational Mapping

Object-relational mapping (ORM) can be mentioned has the join of two concepts: the object-oriented software development and relational databases. Object-oriented development is the key to develop flexible and scalable software systems because it ensures “high productivity, improved reusability, better quality and enhanced maintainability”. Relational databases are used by object-oriented applications because they provide easiness on persistent storage needs. It is simple and features solid mathematical basics and great performance on transaction processing. This relationship comes by mapping classes in the object model to tables in the relational model. It might seem that there is a direct relationship between the object model and the tables of the data model, but using further analysis, the issues involved in this process can be more complex [6].

An ORM is made up of objects that provide access to information contained in a database, and keep business rules within themselves. A clear advantage of using this approach of object-relation abstraction layer is preventing the programmer from using syntax that is specific to a database. Using this layer, the programmer can see his calls from the model objects to SQL queries that are customized for the database currently being used by the application. The automation of this task becomes an advantage, in such way that changing to another database system becomes a trivial task [7].

Thomas, D. [8], gives a nice example of ORM libraries mapping database tables to classes. If a database contains a table “orders”, the application will have a class named “Order”, in which its rows matches to objects of the class. One instance of “order” will be represented as an object of class “Order”. Inside the program, several attributes will be used to get and set each table column. For instance, the “Order” object can have methods to get and set the amount attribute, the sales tax and others.

According to Hart, A.M. [9], a developer can radically reduce the time taken to prototype an application using the Hibernate approach. Hibernate includes the recent superiority mentioned in the ORM tool, such as transparent persistence, dirty checking, efficient strategies for fetching data and a two-level cache. “Development with Hibernate can proceed in a top-down, bottom-up, middle-out or meet-in-the-middle fashion”.

Programmers usually prefer to work with persistence data detained in application objects instead of using direct SQL syntax for accessing information in the database, even if this means to work around the mismatch between data in tables and object state. This is possible because ORM systems resolve this mismatch, being responsible for transporting data to and from a relational database to the right objects, using object-relational (O/R) mappings (Figure 4). Entity Java Beans (EJB) currently uses this approach, using Hibernate 3.0, known has Java Persistence Architecture (JPA) [9].

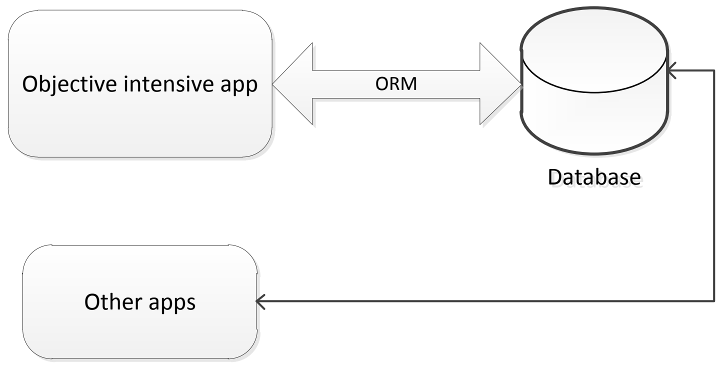


Figure 4: ORM in use in one of many apps using a database

The ORM approach makes possible the programmer to think in terms of entities and their relationships, because it is responsible of controlling these relationships at runtime. It keeps track of each update made in the object instance and converts it to the necessary SQL raw syntax to make the same update in the database at commit time (using the insert, update and delete statements) [10].

2.2 Software Architectural Patterns

Architects and managers usually find themselves unable to evaluate the economic impact of the architectural decisions they make. In order to help the optimization of the software systems “value”, software engineering items were created, attending the business goals of an enterprise. But the value and quality consequences of architectural patterns are difficult to predict, concerning to uncertainty and change. And this is especially important because “these decisions affect meeting an organization’s cost, resource, time to market and quality goals” [11].

Software architectural patterns are intended to support the creation of designs that meet quality attribute requirements. This concept has a broader scope than the concept of a software design pattern, since it addresses numerous issues in software engineering (computer hardware limitations, high availability and business risk) [11].

Examples of architectural patterns are ETL (Data Extraction Transformation & Loading), Transaction Data Stores (TDS/OLTP), Dimensional Data Modelling and Peer-to-peer. Some architectural patterns have been implemented within software frameworks, like the Model-view-controller (MVC).

Because most of the frameworks that will be explored in this section are MVC based, it is necessary to understand in detail what this pattern is all about.

2.2.1 Model-View-Controller

The Model-view-controller (MVC) is a pattern used to describe the requirements necessary to develop a graphic user interface (GUI) application using an object-oriented programming language. It was originated with Smalltalk-80, but has been used for developing GUI applications in several programming languages [12].

Smalltalk-80 and the MVC paradigm were born for the Dynabook project, “a portable personal information management tool envisioned by Alan Kay” [13].

The main idea for the MVC model is that the specifics of an application should not be confused into the user interface concerns, being two distinct aspects. The user interface is best factored into two components: presentation and interaction [13].

The MVC requires the programmer to use three types of objects to develop their own applications: models, views and controllers. Models contain all the system information, along with all the methods necessary to access and manipulate this information in the backend (like a DBMS), representing the principal application domain and encapsulating implementation details of the application’s configuration and behavior. Views are responsible of presenting the accessed information to the user and represent the way in which the model is presented to the user. Views should manage a region of the display and make sure this area is consistent with the state of the model. Finally, controllers are used to process user input data and update the current model and view appropriately [12].

Usually each view/controller pair is connected with a single model, but a model may have multiple view/controller pairs. When a model is modified, its associated view/controller pairs should be changed to reproduce this modification. This is possible due to a message sent using broadcasting to all the dependents by the model [13].

There are two types of MVC: the functional MVC and the object-oriented MVC. As it was referred earlier, in the first model a program is structured as a model (domain-specific aspects), view (abstract user) and controller (command loops). In contrast to object-oriented MVC, a controller calls its view to get user input and to display results, being active and residing of a number of recursive functions, and it calls its model to do a domain-specific work [14].

The MVC paradigm indicates a standard cycle of interaction, beginning with a user executing an input action. This action will result in the controller notifying that the model has been changed. In the case of the view, it can update its display if needed. The controller can update its method of interaction, according to the new model state. In some cases, the model can broadcast enough information so that its dependents do not need to ask for further information about the new state. This reduces the number of requests/responses going around the system after the state change [13].

The MVC abstraction paradigm can be represented as follows (Figure 5):

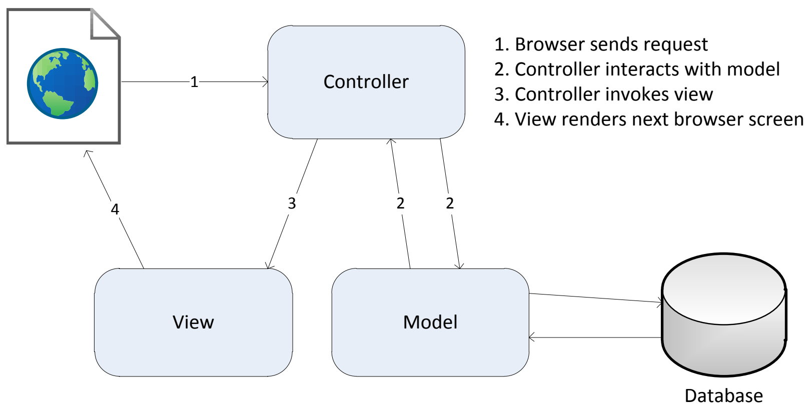

Controller
1. Browser sends request
2. Controller interacts with model
3. Controller invokes view
4. View renders next browser screen
1
2
2
3
4
View
Model
Database

Figure 5: The Model-View-Controller Architecture

The first step consists on a request originated by the client (web browser), which usually consists on a user input action. This request is sent to the controller which will interact with the model, questioning it about its state. The model will broadcast its state to the controller and to the view using specific messages. Finally the controller will invoke the respective view that will render the next client data (to be presented in the browser screen) [8].

Thomas, D. [8] refers that MVC was originally intended for usual GUI applications, “where developers found the separation for concerns led to far less coupling, which in turn made the code easier to write and maintain”.

2.3 Software Development Methodology

A software development methodology in software engineering is a framework that is used to structure, plan, and control the process of developing an information system. It is important to understand the basic concepts of a software methodology in order to evaluate a process and framework that assesses methodologies. This will enable the reader to put the process and framework that is under investigation in a better context, and aid in the understanding of the presented results [15].

There are several methodologies available, like the waterfall development, prototyping, incremental development and spiral development. In this particular case study, we are interested in a methodology that would make the development and delivery of a high quality system faster and, if possible, with no costs. This leads to the key objective of rapid application development.

2.3.1 Rapid Application Development

According to Coleman, G. [16], "the term Rapid Application Development or RAD is taken to projects based around tight timescales, which use prototyping and combine high-level development tools and techniques".

This term was first used by James Martin in the 1990s to "distinguish the methodology from the traditional waterfall model for systems development" [17].

There is no general definition for RAD, but it can be defined in two ways: first as a procedure aimed at suggesting certain stages in software development, like iterative models of software construction; and second as a group of tools that makes object development faster, design GUIs and reusable code for client/server applications [17].

Agarwal quotes Martin, when he writes the four fundamental aspects of fast development: tools, methodology, people and management. These tools include the aptitude for "planning, data and process modeling, code generation, and testing and debugging". A RAD methodology includes two kinds of stages: three-stage and four-stage cycles. The four-stage cycle consists in requirements planning, user design, construction and cutover, while in the three-stage, the first two features are merged in one only activity. In a RAD-type development, the requirements planning and user design can take up to 30% of the total effort [17].

RAD supporters say that this technique really improves productivity, reducing delivery time and "gaining high usage because of the extent of user involvement in the development" [16].

Comelan, G. [16] suggests that RAD projects can fail because three aspects are not accurately addressed: choosing the right team, management, and customer support of the project and the methodologies used. Nevertheless, RAD approach has some advantages like the ease of implementation, improved user satisfaction and the short-time to market [16].

RAD becomes attractive in these days because the world is lined by deadlines and demanding users. The faster a software project is concluded and its capacity to support new business can make the difference in the competitive advantage. "The popular press extols its virtues with adjectives like *evolutionary, iterative, interactive*, and *dynamic*, emphasizing the delivery rate increases facilitated by RAD, which range from 25% to 1 000%" [17].

A complete set of requirements needs to be written before an application starts to be developed, including UML diagrams and documentation, according to the usual software engineering life cycle. The lack of versatility of programming languages and the general speed of development are responsible for this need, joining the client's rationality and their difficulty of changing their minds regularly [7].

This way, the primary positive aspects of RAD are promoting strong collaborative atmosphere and dynamic gathering of requirements, and the business owner actively participates in prototyping, writing test cases and performing unit testing. The primary negative aspects concern to the dependence on strong cohesive teams and individual commitment to the project, and decision making relies on the feature functionality team and a communal decision-making process with lesser degree of centralized project management and engineering authority.

2.4 Web Application Frameworks (Full-stack)

Nowadays, web development is one important activity in the world of software development, with an extensive array of tools available for developers.

A web application framework is intended for implementing a web server, capable of handling multiple states and being designed to process a received event from among a group of pre-determinable events to toggle from one state to another. Each state has associated with it one or more model objects for providing the server with business logic and access to persistent data. An application framework should also include a context object class in order to create objects containing data relating to each state; a context object class providing for an entry method for execution upon entry state; and an exit method for execution upon exit of the state [18].

Next, we present several full stack frameworks, its pros and cons, and the reasons on how the final choice was made.

2.4.1 Ruby on Rails

According to the official website, Rails is a web application development framework written using the Ruby language, designed to make easier programming web applications, "by making assumptions about every developer needs to get started" [19]. Ruby on Rails adopts several principles. These are:

- DRY – "Don't Repeat Yourself" – indicates that it is not a good practice to write the same code several times while developing a single web application. Rails can reduce the amount of duplicated code, providing the developer one place (according to the MVC architecture) so he can write his code;
- Convention Over Configuration – means that Rails makes assumptions about what the user wants to do and the way he is about to do it, replacing the need to write several configuration files. "It allows writing a Rails application using less code than a typical Java web application uses in XML configuration" [8, 19].

Thomas, D. [8], suggests that every Rails web application is developed using the MVC convention. There are other tools that follow the MVC architecture, such as Tapestry and Struts. In contrast, the author proposes that "Rails takes it further by generating a working application that has place for each chunk of code, and all chunks interact in a standard way". In addition, Rails makes easy testing a web application by creating test stubs every time the developer adds a new functionality.

Rails includes built in support for AJAX and RESTful interfaces integration and it's easy to deploy several releases of an application to several servers using a single command tool (with the possibility of rolling back) [8].

This framework adopts the four values expressed in the manifesto for agile software development:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following plan [8, 20].

With Rails "there are not heavy toolsets, no complex configurations, and no elaborate processes". What the developer creates is reflected at the same time in what the customer sees, being "an intrinsically interactive process" [8].

In contrast, a web application designed with Rails is hard to debug, and its development philosophy relies in general assumptions, and after several functionalities implemented, it's hard to remember what is going on in the background.

Rails also requires plugins and libraries to be all up to date and deployed along with the web application, and when the app or server crashes, the dedicated web server doesn't automatically restart.

Despite its advantages, Ruby is a slow dynamic language. The new version is doing to improve this, but statically-typed languages, such as Java or C#, is considerable faster.

2.4.2 Symfony – Open Source PHP Web Framework

The Symfony framework is a full-stack model-view-controller (MVC) free framework that makes it easy to develop websites in a faster way. Mainly it consists of a library of consistent classes written in PHP. It also provides a group of good practices which are projected to help the user to program maintainable and secure websites [21].

Symfony is packed with some features that consist on separating business rules, server logic, and presentation rules of a web application. It contains tools and classes designated to reduce the amount of time spent developing a complex web application. Using this, common tasks are easily automated and the same piece of code can be used in several applications at the same time. This way, the developer does not need to rewrite code for mutual tasks every time a new application is built [7].

Symfony is completely written in PHP 5, being tested in several real-world projects (Dailymotion[1], Askeet[2] and Delicious), and is being used in many high-demand e-business web platforms. Symfony is compatible with the most common DBMS: MySQL, PostgreSQL, Oracle, and Microsoft SQL Server [7].

Symfony has the advantage of being easy to start a new complex project (with previously planned objects and database schema). Being bundled with improved security methods to prevent attacks and code injection, the developer does not have to worry with security specifications.

At the moment, this framework is fully compatible with *Propel* and *Doctrine* object-relational mappings (ORM) [22].

According to Zaninotto, F. [7], Symfony was built with the main purpose of fulfilling the following requirements:

- Easy to be installed and configured in the supported platforms;
- Independent of any DBMS;
- Follows the premise “convention over configuration”;
- Compatible with the best web development practices and design patters;
- Ready for enterprises (use of existing policies and architectures);
- Easy to extend, using vendor libraries.

Several tasks related with web applications development become automated when using Symfony. Mainly:

- It is possible to use templates and layouts written by HTML designers, not having to worry about how the framework works;
- Cache management takes few bandwidth and server load;
- Automated pagination, sorting and filtering;
- Easy to implement AJAX interactions [7].

Finally, Symfony provides several development environments, including many tools that can automate several common software engineering tasks. These are:

- Code-generation tools and simple back-end administration;
- Built-in tools to make easy test-driven development;
- Logging features [7].

[1] http://www.dailymotion.com
[2] http://www.askeet.com/

Symfony has the following drawbacks:

- There is a learning curve. At first everything seems unnecessarily complex, such has the directory structure;
- Despite being easy to start up and create the complex hierarchy of a project, it is hard to customize individual parts;
- Documentation exists, but as a big project, the developer needs to filter lots of it while learning.

2.4.3 Liferay – Open Source Enterprise Portal

Liferay is an open source framework that is designed for web portals. It can include several features into a portal, like blogs, wikis, discussion forums, and document management applications, by offering a runtime environment for hosting Java-based portal applications, known as "portlets".

A web portal provides an entry point to distributed information in the web, offering a unified way to access it. According to Sarang, P. [23], and as described in the Java Portlet Specifications (JSR 286), "a portal is a web-based application that commonly provides personalization, authentication, [and] content aggregation from different sources and hosts the presentation layer of information systems". It is a user-customizable web site that serve as a gateway to several contents collected from several sources [23].

Liferay Portal provides a framework for creating any type of portals, like personal, academic, regional, government, or sports web portals. Web applications developed with this framework can be deployed in several web servers, because it conforms rigorously with the standards [23].

Referring Sarang, P. [23], "a portlet is an application that delivers content to the user (…) [which] includes one or more portlets, and a portlet container manages the portlets".

Liferay has several features that are popular among the developer community, for instance:

- Built with Java;

- Based on tested components. It uses Apache ServiceMix, Hibernate, Java J2EE/JEE, JGroups, jQuery, PHP, Spring, FreeMarker, and others.
- Use of standards to communicate with other software. It makes it easy to send information from the framework to other systems. For example, AJAX, OpenSearch and Open platform with support for web services (JSON, REST, RMI), and others [24].

Many known vendors provide their own tools for creating these type of portals, and servers to host portals. Oracle WebLogic Portal, IBM WebSphere Portal Server, Glassfish Web Space Server, and Microsoft SharePoint Server can be mentioned. Liferay framework is a portal server among open source technologies [23].

2.4.4 Molgenis

Molgenis (Molecular Genetics Information System) “is a tool for rapid prototyping of data portals for life science projects”. It was developed by Morris Swertz to answer the need for a suitable structure to arrange and manage data from the Genomics research laboratories and research workflow. The amount of data produced by high-throughput experiments is increasing at large steps, so there was the need for a tool for processing this kind of heavy information [25].

Molgenis is known for providing bioinformaticians an easy way to model biological data structures and simple user interfaces. By writing two simple XML files, Molgenis converts it to a ready-to-use web application, including database, user interfaces, exchange formats, REST/JSON, SOAP and RDF. Every time Molgenis is generated, it automatically generates SQL raw instructions that design a specific database, and script analysis tools in R or HTML code that would take lots of time and effort to write by hand. This way it is easy to find errors, and any change in Molgenis code is visible to every instance just by doing a re-generation [25].

Swertz [25] reports that Molgenis toolkit was successfully evaluated for the rapid prototyping of many types of biomedical applications, such as GWAS[3], QTL, proteomics and biobanking. It makes easy to upload and exchange data and technical documentation by featuring a tab-delimited file format. Existing databases can be easily

[3] http://gwas.nih.gov/

converted to a standard Molgenis model, using the “ExtractModel” procedure included in the package.

With Molgenis we can create a customizable XML model file with several entities, each one corresponding to a table in the database backend, with different fields. With a reusable framework and generators, we can automate common patterns and reuse in family of projects. New features can be added once and therefore automatically added. This becomes part of five main solutions introduced by Molgenis framework:

- Use of automatic code generation;
- Customized data model based on standards;
- Storage of datasets as black boxes, instead of decomposing them in database tables;
- Loose linking to other programs for improved flexibility;
- Low-maintenance web-based user interface [4].

According to Swertz [4], Molgenis was developed and tuned up because of the needs of genomic researchers, requiring a low budget and being a completely customizable framework. This flexibility and simplicity was accomplished by making Molgenis responsible for data management only. File decomposition and processing tasks are attributed to dedicated software, ensuring that Molgenis does not need to be upgraded because of new versions of data formats and tools.

Specifying all the entities using just XML definitely reduces the time spent by the researcher on designing directly on a SQL platform all the database, which is a way to focus more on other aspects, such as the final user interface or collecting useful data.

2.4.5 Conclusions

Initially, it was necessary to do a review on the frameworks that offer the full-stack features. In addition to the four that have been mentioned before, there are others based on C++, Python, Perl, ASP.NET and CFML (ColdFusion Markup Language). The reason why the respective frameworks were chosen concerns itself primarily by previously acquired knowledge in the languages of Java and PHP. Concerning to Ruby on Rails, the amount of functionality could justify an investment in time learning the Ruby language.

Symphony includes a built-in authentication system, and has more of a sense of structuring applications than Rails does. This means that it is easier to move pieces of functionality between Symfony applications without having the need of plugins used by Rails. On the other side, there are more configuration files, being possible to change settings at many levels (whole of Symfony, per application or per module). For instance, validation can be carried out in configuration files rather than in model classes, which makes it easier to share validation code around but it increases the number of files to manage.

The architecture in Symfony is based on the MVC pattern, with a front-controller, very similar to Rails. Configuration in Symfony works in a similar way to ActiveRecord and is straightforward.

AJAX support in Symfony is built-in (using Prototype), and it has the same concept of helpers as Rails, meaning we can build interfaces in a similar way to Rails interfaces.

Liferay, as an enterprise portal, is designed for integrating information, people and processes across organizational boundaries. The main advantage of this framework over the previous ones is that no programming skills are required for basic website installation and administration.

Molgenis is designed for rapid generation of bioinformatics web portals, and uses Java as the programming language. Comparing this framework with the previous ones, it does not add anything new. For this project, and for the first step, we needed a framework that could generate a database, methods to update it in the simplest way possible, and a simple frontend to check the results and run simple search queries. This is the main reason why Molgenis was chosen instead of other framework. The Molgenis framework API is a useful tool regarding on updating the database, and has the simplest and most functional frontend out of the box.

Table 1 resumes the most relevant features and makes a comparison between those frameworks: languages, AJAX support, MVC approach, ORM and some other features like the possibility of migrating from databases and security options.

Table 1: Comparison of Web Application Frameworks

Framework	**Language**	**Ajax**	**MVC framework**	**MVC Push/ Pull**	**ORM**	**Testing framework**	**DB migration framework**	**Security**
Symphony	PHP	Prototype, UJS and PJS	Yes	Push	Propel, Doctrine	Yes	Yes, with plugin	Yes, with plugin
Ruby on Rails	Ruby	Prototype, jQuery	Yes (Active Record and Action Pack)	Push	Active Record	Yes	Yes	Yes, with plugin
Liferay	Java	Alloy UI	Yes	Push	Hibernate	Yes	Yes	Yes
Molgenis	Java	No	Yes	Push	Hibernate	Yes	Yes	Yes, with plugin

2.5 Presentation Frameworks for Web Applications (Frontend)

In web application development, there are several ways to achieve results. The developer can choose between normal servlets and JSP, which gives high flexibility and good control over functionalities. While using a framework, certain features are not available or not in the way the developer wants to be, leading to a lot of small configurations around it. Frameworks are written in a generic way and try to cover many situations, extending the code and, overall, reducing performance.

On the other side, if the web application consists in a complex project, the developer will end up developing his own framework. This will lead to a diversity of functionalities known only by the developer, and it will become hard for other collaborators to understand and improve code. And if these collaborators change over the time, spreading knowledge over the new team will have costs.

This is the reason why we chose to develop the OralCard web application using a frontend framework. It will reduce the amount of time on complex configurations, repetitive code, and because it has good documentation, other contributors can later grab the application code and improve it.

We will refer three presentation frameworks: Stripes, Google Web Toolkit, and ZK.

2.5.1 Stripes

Stripes is a framework that makes developing Java web applications easier because it eliminates much of the configuration that traditionally has been associated with Java web development [26]. For instance, if the developer wants to perform forms validation, there is the need to simply add the correspondent annotation in the action class, and leave it to Stripes (Figure 6).

```
@Validate (required=true, minValue=21, maxValue=100)
int temperature;
```

Figure 6: Validation configuration using Stripes

Tim Fennell, creator of the Stripes framework, took advantage of the new features of Java 5 and removed the need for extensive configuration over XML files. The only XML file needed is the *web.xml* that is used to start any Java web application. Using this framework, web development becomes easier because the developer does not have to concern about shaping his code to the framework restrictions. Stripes is designed to adjust to the programmer's code [27].

Raw Java web development has several disadvantages, such as many repetitive low-level tasks. Stripes automatically takes care of this, so the programmer can concentrate his efforts on thinking about the applications features instead of "writing clear, concise, readable, and maintainable code" [27].

Stripes framework is a MVC framework, mostly presented in the controller and view parts. "It interacts with the programmer model leaving it independent of anything Stripes-specific". The relevant benefit is that data can be transported between the programmer model and the controller/view without having to configure any complex feature in some configurations files [27].

Stripes lets the developer choose how the model will be mapped to a database, and several other frameworks can be used along with it. It's intended to focus just on the web part of a web application development. Each request is converted into an *action* which is responsible of running a Java method that will do the designated job, returning a result. This result will be interpreted and Stripes provides the appropriate response for the web browser [27].

Daoud, F. [27], presents a quick summary on Stripes:

- **Smart Binding**: Stripes makes easy the relationship between URLs, parameters and events from HTTP to Java methods and classes. This way, the code becomes clear to the developer, becoming easy to make an establishment between classes, methods and properties;
- **Autoloading**: This means that the developer can add, remove and rename his classes without having to worry about specifying it in configuration files;
- **Validation**: It's easy to validate user input fields with Stripes (Figure 6);
- **Exception handling**: It is possible to design error specific pages which will be shown when something goes wrong with the application;
- **Interceptors**: Before providing a response, Stripes can run interceptors, small pieces of code developed by the programmer to specific tasks, becoming easy to alter the data flow. As an example, a user should be logged on if he wants to realize a desired action. The method to login can be run before showing the desired web page;
- **AJAX Integration**: According to the natural request-response nature of Stripes, it becomes easy to integrate AJAX functionalities into the web application;
- **Testing**: Stripes comes with a bundle of tools created to help the developer to create automated tests.

2.5.2 Google Web Toolkit

Google Web Toolkit (GWT) is a set of development tools, programming utilities, and widgets, written using Java language, which allow creating rich web applications. The main feature that defines GWT is that it makes possible to write the browser-side of the application using Java instead of JavaScript [28].

According to Hanson, R. [28], the need to write Internet applications using Java instead of JavaScript comes from the increasing size and complexity of Internet applications, which are becoming richer every day. Nevertheless, writing raw JavaScript is still possible using GWT.

GWT provides several tools for server communication, like wrappers that are related with AJAX development and a set of classes to support JSON message format. These tools make it easy to integrate with frameworks such as JSF, Struts, and EJB, but also to cooperate with services hosted in servers. JUnit testing framework is also supported with this toolkit, and it even is bundled with a special hosted-mode browser that helps the development and debugging of an application in Java without the need to deploy code to the server [28].

Hanson, R. [28], provides a representation of the central aspects of GWT (Figure 7). It’s visible the tools that are tied up to the compiler, and the Java libraries that all together make up the GWT API.

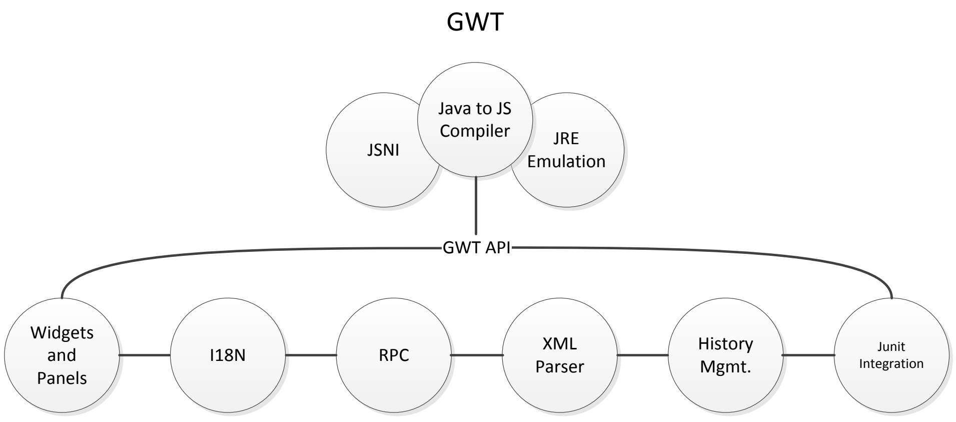


Figure 7: Central Aspects of the GWT framework

2.5.3 ZK – Open Source AJAX

ZK framework is a method to incorporate AJAX into web pages. The entire collection is based on JavaScript and AJAX. It is an event-driven, component-based framework that enables rich user interfaces for web applications. It includes an AJAX-based event-driven engine, a rich set of XML User Interface Language (XUL) and XHTML components, and a markup language designated by ZK User Interface Markup Language (ZUML). Unlike several others AJAX frameworks, ZK does not require the developer to have strong knowledge on JavaScript programming, because it generates all the necessary JavaScript code. It just requires some knowledge on HTML [29, 30].

In ZK, the developer does not need to handle with raw AJAX code. According to Chen, H. [29], ZK simplifies the development of rich AJAX web applications because:

- The event-driven engine carries the intuitive desktop programming model to web developing;
- The XUL and XHTML components empower web applications by using off-the-shelf building blocks;
- The ZUML markup language makes the design of rich user interfaces easy.

The AJAX-based mechanism of ZK can be described as follows (Figure 8) [29]:

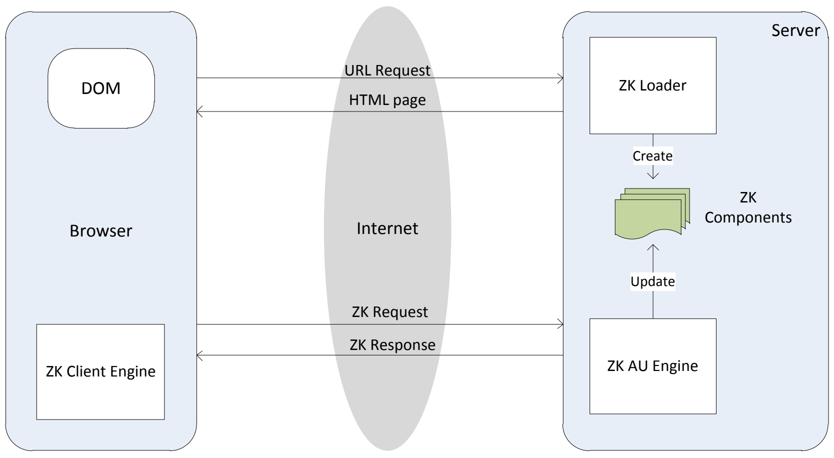


Figure 8: The ZK loader, the ZK AU engine, and the ZK client engine

The ZK Loader receives an incoming URL request from the browser side and generates the corresponding HTML page, including HTML, CSS, and JavaScript code, and ZK components at the server side. Then, it sends the generated HTML page to the client and to the ZK Client Engine. This engine is present on the client side, and its job is to monitor incoming JavaScript events queued in the browser. If any of these events are triggered, the ZK Client Engine will send those to the ZK AU Engine. The AU Engine is responsible for receiving the last AJAX requests, updating ZK Components and sending back an AJAX response to the client. Finally, the ZK Client Engine receives this response and updates the corresponding content in the browser DOM tree [29].

It includes several advantages like injection of code on the JSP page using tags, use the components in the form of Java classes, and a combination of these two. With this framework, it is not needed any backend framework, and all of the most known middleware is compatible, like JDBC, Hibernate, EJBs and JMS [29, 30].

2.5.4 Conclusions

There are countless frameworks available in the market in order to develop web applications. We reviewed Stripes, GWT and ZK because these are ones of the most used in the Java language fragment.

ZK main advantage is the facility on integrating JavaScript and AJAX into the application. It handles all the intermediate procedures needed to establish HTTP communications between requests and responses, reducing the amount of raw code the developer needs to input. In one way, its features are attractive in such way the developer can design the “top design” of the application using a graphic editor, and in a few moments the developer has a mock-up ready to use with an eye-catching frontend.

GWT is based on widgets, and its main advantage is the development of a frontend user interface by using raw Java code. It transforms it into ready-to-use, browser side JavaScript code.

Because ZK and GWT are mainly based on widgets and tools, we decided that Stripes is the most suitable framework for our requirements. It is not so focused in the final visual result of the application, but it assists the integration of useful features concerning to the browser navigation and the interaction with the backend. Its main advantage is its simplicity, and it does not create endless configuration files and code, like the others frameworks.

Table 2 presents a simple comparison between those frameworks, concerning mainly on the MVC approach and the ORM tools.

Table 2: Comparison between frontend development frameworks

Framework	**Language**	**Ajax**	**MVC framework**	**MVC Push/Pull**	**ORM**	**Testing framework**	**Security**
Stripes	Java	Yes	Yes	Push	JPA, Hibernate	Yes	Yes, with framework extension
Google Web Toolkit	Java	Yes	-	-	JPA with RequestFactory	Yes, with jUnit, jsUnit, Selenium	-
ZK	Java, ZUML	jQuery	Yes	Push and Pull	Any J2EE ORM framework	jUnit, ZTL	Spring Security

3. Context of the Problem

Despite the recent boom of data stored by main biomolecular databases, the output of many studies is stored in small, domain-specific databases. These databases play a crucial role by enabling a faster exchange of research breakthroughs among communities. Due to the evident lack of resources from those databases expert curators, it is of major importance to have an easy and efficient method for prototyping and developing new software.

In this chapter, we will present the chosen method for data integration, and the main motives why it was selected. The oral cavity was the main case study for this project, and we will describe why it is of major importance of including our solution to this particular situation.

3.1 Data Integration

The web is a universal repository of information where there is an excellent opportunity to exploit the integration of online biological resources for knowledge discovery. A major challenge is to support the actual flow of information among the sources and services on the web and their interconnection with legacy systems that are designed to operate with traditional relational databases [31].

Data centralization becomes a key to deploying strategic enterprise applications. Operational data stores, data warehouses, data marts, mash ups, and other analytic and operational applications require a greater degree of data sharing than ever before. Satisfying the information demands of these secondary-use business applications becomes a primary objective, and that means moving data from the original sources to the target business data systems.

3.1.1 Data Warehousing

A data warehouse (DW) provides information for analytical processing, decision-making and data mining tools. A DW collects data from multiple diverse operational source systems (OLTP – Online Transaction Processing) and stores summarized integrated business data in a central repository used by analytical applications (OLAP – Online Analytical Processing) [32].

The common process for obtaining decision-making information is based on using OLAP tools. These tools have their data source based on the DW data area, in which records are updated by ETL (Extraction, Transformation and Loading) tools. The ETL processes are responsible for identifying and extracting the relevant data from the OLTP source systems, customizing and integrating this data into a common format, cleaning the data and conforming it into an adequate integrated format for updating the data area of the DW, and lastly, loading the final formatted data into its database [32].

The demand for updated data in data warehouses has always been something that is really needed. Data warehouse refreshment is normally performed in an offline mode, meaning that while processes for updating the database are executed, OLAP users and applications cannot access any data. These activities usually take place in a predetermined loading time windows, to avoid overloading the operational OLTP source systems with the extra workload of this workflow [32].

Active Data Warehousing is a new tendency where DWs are refreshed as many times as possible, due to the high demands of users for new data. Nowadays, IT managers are facing crucial challenges deciding whether to build real-time data warehouse instead of a conventional one and whether their existing data warehouse is going out of style and needs to be converted into a real-time data warehouse to remain competitive [32].

Using the data in a warehouse can help the user focus on relationships and help to understand more about the environment that the business operates in. It increases the consistency of the data and allows it to be checked several times to determine how relevant it is. Because most data warehouses are integrated, we can pull data from many different areas of business.

The following illustration (Figure 9) demonstrates a typical data warehousing architecture [33].

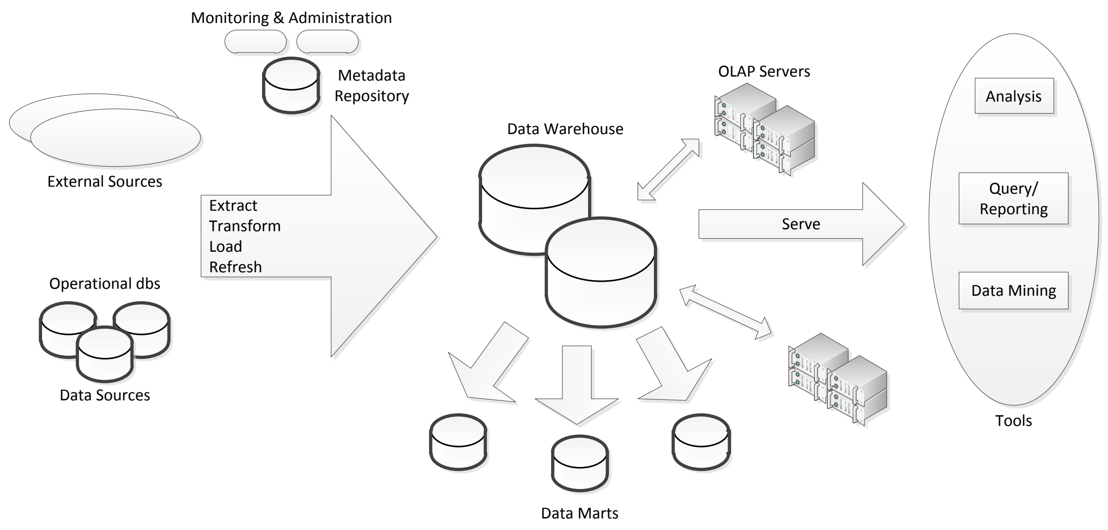


Figure 9: Data Warehousing Architecture

A DW includes tools for "extracting data from multiple operational databases and external sources; for cleaning, transforming and integrating this data; for loading data into the data warehouse; and for periodically refreshing the warehouse to reflect updates at the sources and to purge data from the warehouse". In addition to the main warehouse, there may be several departmental data marts. Data in the warehouse is stored and managed by one or more warehouse servers, which present multidimensional views of data to a variety of frontend tools: query tools, report writers, analysis tools, and data mining tools. Finally, there is a repository for managing metadata, and tools for monitoring and administering the warehousing system [33].

DW systems use a variety of data extraction and cleaning tools, and load and refresh utilities for filling warehouses. This extraction from other sources is usually implemented via gateways and standard interfaces (such as Information Builders EDA/SQL, ODBC, Oracle Open Connect) [33].

It is necessary that the data in the warehouse is accurate, since a DW is used for decision-making. However, as large amounts of data are involved from many sources, there is a high probability of errors and irregularities in the data. This is why it is crucial to use tools that help to detect data irregularities, correcting them [33].

After extracting, cleaning and transforming, data must be loaded into the warehouse. Typically, batch load utilities are used for this purpose. In addition to fulfilling the warehouse, "a load utility must allow the system administrator to monitor status, to cancel, suspend and resume a load, and to restart after failure with no loss of data integrity" [33].

Because sequential loads can take a very long time, pipelined and partitioned parallelism are typically exploited. Doing a full load has the advantage that it can be treated as a long batch transaction that builds up a new database [33].

However, adopting a data warehouse model means taking the time consuming to create and to keep operating. The user might also have a problem with current systems being incompatible with some data. Another concern is about security, especially if data is accessible over an open network.

For data integration systems that rely on information that changes frequently, a data warehouse approach is not ideal. One way on addressing this issue is to design systems that pull data directly from individual data sources. Since there is no centralized database dedicated to analysing, categorizing and integrating the data in preparation for user queries, those responsibilities are given to other system's components.

3.1.2 Data Integration Alternatives

As the needs of downstream consumers have become more sophisticated, different approaches to data integration have evolved. The more traditional *Extract, Transform, Load* (ETL) approach which takes data from its sources to a staging area in which data sets are manipulated and transformed into a target representation. An alternate approach is *data virtualization*, in which the data remains stored at the source and a conceptual view is materialized on demand.

Data Virtualization

According to Weng, L. [34], a *data virtualization* describes an abstract view of data, and a *data service* implements the mechanism to access and process data through the *data virtualization*.

Lans, R. [35] defines it as the "process of offering data consumers a data access interface that hides the technical aspects of stored data, such as location, storage structure, API, access language, and storage technology".

Data virtualization provides an abstraction layer that can be used for data access by data consumers, in a consistent way. A data consumer can be any application retrieving or manipulating data, such as reporting or data entry application. This abstraction layer hides all the technical aspects of data storage. The applications do not have to know the details where all the data is physically located, where the database servers run, what programming languages are being used [35].

Data virtualization can be implemented in several different ways. Here are some of them:

- Using a **federation server**, multiple data sources can be made to look as a single one. The applications will see a unique data source, while in reality the data is stored in multiple databases;
- An **Enterprise Service Bus** (ESB) can be used to develop a layer of services that allow access to data. The applications using these services will not know where the data is being stored, what the original source interface is and how the storage structure is. They will only see interfaces, like SOAP or REST. The ESB is the abstraction layer;
- Using the **cloud** for placing data stores. To access a data store, the applications will see the cloud API, having no information about the physical location of it. Technical aspects of how the data is stored and managed are transparent;
- A **proprietary software-based abstraction layer** can be developed by organizations, hiding the technical aspects of the data store [35].

Using data virtualization and data services, compact and specialized data formats can be hidden from the applications analysing grid-based datasets. On the other side, supporting data virtualization can require significant effort. For each dataset layout and abstract view that is desired, a set of data services need to be developed. Another challenge is about the fact that the design and implementation of an efficient data virtualization and data services require interaction of two complementary entities. The first one is the scientist or researcher, who has good knowledge of the applications, datasets, and their format, but has less knowledge on databases and data services implementations. The second entity is the database developer who is expert in the tools and techniques for efficient database and data services implementations, but has low know-how of the specific application [34].

As opposed to the traditional approach of extracting data from multiple sources and temporarily storing those data sets at a staging area, data virtualization allows the source data sets to remain in their original locations. Data virtualization introduces abstraction layers over a variety of native data sources and, as a by-product, provides relational views without requiring that data be extracted from its source. This approach to abstraction enables the establishment of reusable data services, and the data abstraction layers typically deployed within a data virtualization environment allow for the presentation of a standardized a logical representation of enterprise data concepts, thereby allowing many different downstream data consumers to see a view of the data that is both structurally and semantically consistent [36].

Data Federation

Lans, R. [35] purposes *data federation* has "a form of data virtualization where the data stored in an heterogeneous set of autonomous data stores is made accessible to data consumers as one integrated data store by using on-demand data integration".

This definition is based on the next five concepts:

- **Data virtualization**: data federation is a form of data virtualization. Not all forms of data virtualization suggest data federation, but data federation always effects in data virtualization;
- **Heterogeneous set of data stores**: An application using data federation should be able to access different types of database servers and files with various formats. It should be able to integrate data from all selected data sources and offer features for transforming the data. It should allow the applications and tools to access data using different APIs and languages;
- **Autonomous data stores**: Data stores accessed by data federation should be completely autonomous, being able to operate independently;
- **One integrated data store**: Regardless of how and where the data is stored, data federation should be presented as one combined data set. This involves transformation, cleaning and enrichment of data;
- **On-demand integration**: This means that with data federation, integration takes place on the fly, and not in batch. When the user asks for data, only then data is accessed and integrated. Data is not stored in an integrated way, but remains in its original location and format [35].

3.1.3 Conclusions

For the OralCard project, we have chosen a *hybrid* solution for data integration that combines the best features of the two main solutions: data warehouse and data virtualization.

We will implement a federation server to store identifications from several entities, which will point or indicate the respective data sources. By knowing these ids stored in our database, the final application can then fetch (live) all the necessary data concerning to a protein, disease, gene ontologies, and others.

On the other side, our server will act also as a data warehouse. It will include tools to extract, clean and transform data from external sources, and store this information. This information can be descriptions or data related to the circumstances in which a protein was researched.

Our system will then consist on a DW with tools to retrieve information related to the given list of proteins, and it will work as a federation server to store ids and links that contain detailed information.

3.2 The Oral Cavity

To access the validity of this project, the subject of the oral cavity is used as a subject. This is relevant because it consists of a complex (eco) system where a variety of proteins from numerous origins are present. As a result, being able to estimate the impact of the interactions among those proteins is crucial to understand the underlying disease mechanisms and hopefully to develop new treatment methods.

Saliva is the watery and usually frothy substance produced in the oral cavity of humans and most other animals. It is a unique clear fluid, composed of a complex mixture of electrolytes, proteins, and represented by enzymes, immunoglobulins and other antimicrobial factors, such as mucosal glycoproteins, traces of albumin and some polypeptides and oligopeptides, of importance for oral health [37].

Whole saliva is secreted mainly from three pairs of major salivary glands: the parotid, the submandibular, and the sublingual glands. Approximately 90% of total salivary volume results from the activity of these three pairs of glands, with the bulk of the remainder from minor salivary glands located at various oral mucosal sites [38].

Whole saliva also contains proteins from gingival crevicular fluid, oral mucosa and oral microbiota. The various components of saliva from these sources, together with the plasma proteins that appear in saliva, define the physiological behaviour of the oral cavity, the oral physiome.

Saliva is an ideal translational research tool and diagnostic medium and is being used in novel ways to provide molecular biomarkers for a variety of oral conditions, such as oral cancer [39, 40], dental caries [41] and periodontitis [41, 42] , as well as systemic disorders such as breast cancer [43], Sjögren's syndrome [44], diabetes mellitus [45], cystic fibrosis [46] and diffuse systemic sclerosis [47]. The ability to analyze saliva to monitor health and disease is a highly desirable goal for oral health promotion and research [48, 49]. The most important advantage in collecting saliva is that it is obtained in a non-invasive way and is of easy access.

Over the past thirty years, there have been many efforts to determine and identify the main salivary proteins and peptides. Nevertheless, the fluctuating nature of saliva from different individuals, huge dynamic protein concentration ranges and the protein detection limits of most proteomic techniques have made the saliva proteome elusive to define [50]. Even when used for healthy individual saliva, with multi-dimensional separations and advanced bioinformatics search software tools, proteins identified in different saliva proteomics experiments are often inconsistent with each other except for the most abundant proteins. To overcome the poor coverage, potential bias, and complementary nature of each experimental measurement of the human saliva proteome, it is necessary for biomedical researchers to collect and evaluate all reliable publicly-available saliva protein data sets generated from different analytical and computational platforms for healthy individuals as well as in disease conditions.

A comprehensive integrated resource of the saliva proteins would provide a high amount of comparative power for interpreting proteomics profile changes in patient's saliva, and may supplement or compensate for limitations and biases associated with the set of controls for a given study. It would also improve the ability for finding protein biomarkers that are known to occur in healthy human saliva, for instance where a protein is differentially expressed in a patient sample related to the quantities observed in the study control.

OralCard will have as a vital component an integrated database, by compiling all of the existing experimental data performed on healthy individual samples as well as in several oral and systemic diseases. It will include a collection of microbial proteins expected to be present in saliva due to their presence in the genomes of the oral microbiota [51, 52] and a subset of microbial proteins determined experimentally [53].

In the next sections, there are presented some terms and definitions regarding to the biological and scientific part of this work. The goal is to inform the general concepts behind the entities that are being used in the exploitation of knowledge.

3.2.1 Proteins

The word *protein* is derived from the Greek word *prôtos*, meaning *primary* or *first rank of importance*.

Proteins form the very basis of life. They regulate a variety of activities in all known organisms, from duplication of the genetic code to transporting oxygen, and are responsible for regulating the cellular machinery and consequently, the phenotype of an organism. Proteins accomplish their task by three-dimensional tertiary and quaternary interactions between various substrates such as DNA and RNA, and other proteins [54].

According to Lau, J. Y. [54], knowing the structure of the protein, we can probe for its function and possibly apply the new knowledge to various genome projects, such as mapping the functions of proteins in metabolic pathways for whole genomes and presuming evolutionary relationships.

There are around 20,000 to 25,000 genes in the human genome, and they code for as many as 100,000 proteins, which are made up of 20 amino acids [54].

Amino acids polymerize at the carboxylic acid group of one amino acid to the amino group of the next to form a peptide. A protein is a extensive polypeptide chain. The chemical properties of each amino acid and its unique sequence of the peptide chain are responsible for giving the protein its exclusive function and structure. Taking out one amino acid or moving it from the protein sequence can be detrimental to its structure, and in the same way its biological meaning [54].

UniProt

The Universal Protein Resource (UniProt) [55] provides the scientific community with a single, centralized, authoritative resource for protein sequences and functional formation. It was formed by uniting the Swiss-Prot, TrEMBL [56] and PIR protein database activities [57], and it produces three layers of proteins sequence databases.

These are the UniProt Archive (UniParc), the UniProt Knowledgebase (UniProt), and the UniProt Reference (UniRef) databases.

The UniProt Knowledgebase is a comprehensive, fully classified, richly and accurately annotated protein sequence knowledgebase with extensive cross-references. This core consists of two sections: the UniProt/Swiss-Prot, with fully, manually curated entries; and UniProt/TrEMBL, enriched with automated classification and annotation.

The UniParc is designed to capture all available protein sequence data, not just from the aforementioned databases, but also from sources such as Ensembl [58], the International Protein Index (IPI) [59], RefSeq [60], FlyBase [61] and WormBase [62], making it highly comprehensive.

The UniProt databases can be accessed online[4] or downloaded in several formats[5].

The Protein Data Bank

The Protein Data Bank (PDB)[6] is the single worldwide archive of structural data of biological macromolecules. Its depositors have varying expertise in the techniques of X-ray crystal structure determination, NMR, cryoelectron microscopy and theoretical modelling. PDB users consist on a diverse group of researchers in biology, chemistry and computer scientists, educators, and students at all levels [63].

The data increase in this subject claim for new ways to collect, organize and distribute data. Nowadays, the management of the PDB is responsibility of the Research Collaboratory for Structural Bioinformatics (RCSB). Its goal is to create a resource based on the most modern technology that simplifies the use and analysis of structural data, and this creates an enabling resource for biological research [63].

3.2.2 Diseases

According to the definition provided by *Biology Online*, a disease is “an abnormal condition of an organism which interrupts the normal bodily functions that often leads to feeling of pain and weakness, and usually associated with symptoms and signs”. It is a pathologic condition in which “the normal functioning of an organism or

[4] http://www.uniprot.org
[5] ftp://ftp.uniprot.org/pub
[6] http://www.rcsb.org/pdb

body is impaired or disrupted resulting in extreme pain, dysfunction, distress, or death" [64].

In order to establish a connection between a protein and several diseases, UniProt stores in each entry a set of external ids for the diseases database, the Online Mendelian Inheritance in Man (OMIM).

OMIM

Online Mendelian Inheritance in Man (OMIM) is a comprehensive, authoritative and timely knowledgebase of human genes and genetic disorders compiled to support human genetics research and education and the practice of clinical genetics. OMIM[7] is distributed by the NCBI in the web and is integrated with the Entrez suit of databases. Each OMIM entry has a full-text summary of a genetically determined phenotype and/or gene, and has numerous links to other genetic databases such as protein sequence, PubMed references, and GeneTests [65].

An OMIM entry consists in an assigned six-digit number whose first digit indicates whether its inheritance is autosomal, X-linked, Y-linked or mitochondrial. In addition, OMIM entries are categorized by whether they contain information on genes, phenotypes or both. This is denoted by the symbol that precedes an OMIM number ("*". "#", "+", "%" or "^") [65].

In 2004, OMIM had about 10,000 entries describing genes with known sequence and about 5,700 entries describing phenotypes. A log allows a quick check of the latest additions and changes to OMIM. About 70 entries are created and 600 updates each month [65].

3.2.3 Pathways

Pathways are "consecutive reaction steps, which are either biochemical transformations or sequences of signalling events, such as signal transduction" [66].

Metabolism was the first functional level in human biology studied experimentally and consequently was the source for the first database of biochemical reactions and pathways. These databases include EMP/MPW, BRENDA[8], ERGO[9], and KEGG[10] [66].

[7] http://www.ncbi.nlm.nih.gov/omim
[8] http://www.brenda-enzymes.info/

KEGG

According to Kanehisa, M. [67] KEGG (Kyoto Encyclopedia of Genes and Genomes) "is a knowledge base for systematic analysis of gene functions, linking genomic information with higher order functional information".

KEGG is an effort to link genomic information with higher order functional information by computerizing current knowledge on cellular processes and by standardizing annotations. Its functional assignment is a process of linking a set of genes in the genome with a network of interacting molecules on the cell, such as a pathway or a complex, representing a higher order biological function [67].

The genomic information is stored in the GENES database, which consists on a group of gene catalogues for all the completely sequenced genomes and some partial genomes with annotation of gene functions. It uses the PATHWAY database in order to store the higher order functional information, which contains graphical representations of cellular processes, such as metabolism and cell cycle. KEGG provides Java graphics tools for browsing genome maps, comparing genome maps, and some computational tools for sequence comparison, graph comparison and path computation [67].

3.2.4 Gene Ontology

Genomic sequencing specifies that a considerable amount of the genes specifying the core biological functions are shared by all eukaryotes. Knowledge of the biological role of such shared proteins in one organism can often be transferred to other organisms. The goal for the Gene Ontology Consortium is "to create a controlled vocabulary that can be applied to all eukaryotes even as knowledge of gene and protein roles in cells is accumulating and changing" [68].

The Gene Ontology (GO)[11] provides several attributes and classifications that cover many domains of molecular and cellular biology and are available worldwide for use in the annotation of genes, gene products and sequences [69].

According to Harris, M. A. [69], "The GO database integrates the vocabularies and contributed annotations and provides full access to this information in several formats".

[9] https://ergo.integratedgenomics.com/
[10] http://www.genome.jp/kegg
[11] http://www.geneontology.org

3.3 Summary

In this chapter we presented the concept of data integration and some solutions to implement it. These are data warehousing, data virtualization, and data federation. Data warehousing collects data from multiple operational source systems, and stores summarized data in a central repository. Data virtualization can be used in several ways, such as using a federation server or taking advantage of the cloud. Data federation is a form of data virtualization that uses on-demand data integration.

We also presented the solution chosen for the OralCard project, and it consists in a mixture of the best features between data warehouse and data virtualization.

Finally, in order to understand the goal for this assignment, we provided general information concerning to the oral cavity and the main entities that are related to it. These are the proteins, diseases, pathways, and gene ontologies. There are several online tools that provide information regarding to these objects: UniProt, Protein Data Bank, OMIM database from NCBI, and KEGG.

In the next chapter we will present the solutions and technical aspects chosen for the design and implementation of the OralCard web application.

4. Work Implementation

This chapter is intended to describe all the technical aspects of the work implementation made to empower the OralCard web application. The first part will present the technical aspects on deploying the backend, and its architecture. In the second part it will be described the database class diagram with all the entities necessary to the OralCard application. In the third we will present the steps taken in order to import data to the OralCard application. In the fourth and final part will present the topics on developing the frontend for this project.

4.1 Backend Development

For the backend we chose the Molgenis framework for generating all the necessary tools and features needed to start fulfilling our database and to rapidly view this data in an easy way.

As shown in the Molgenis architecture (Figure 10), to start using this framework we need to have preinstalled a MySQL server to store its database, an Apache Tomcat web server to be able to deploy web services and a simple user interface, and the Java Development Kit so the framework can generate all the necessary SQL and HTML code.

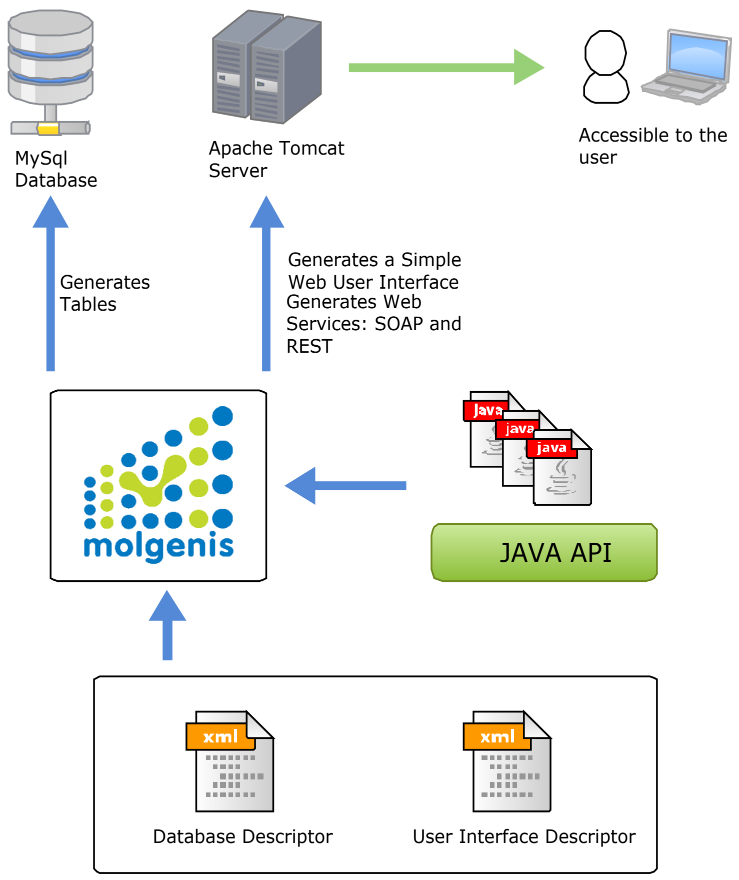


Figure 10: Molgenis Architecture

Molgenis requires the user to specify the database main structure in a XML file. This file is denominated *molgenis_db.xml*, and it contains the information needed to create MySQL tables with attributes. As an example, Figure 11 shows the necessary code to create a table designated *Organism* with the attributes *TaxId*, *Name* and *ShortName*. It will also be accessible through a generated *id*.

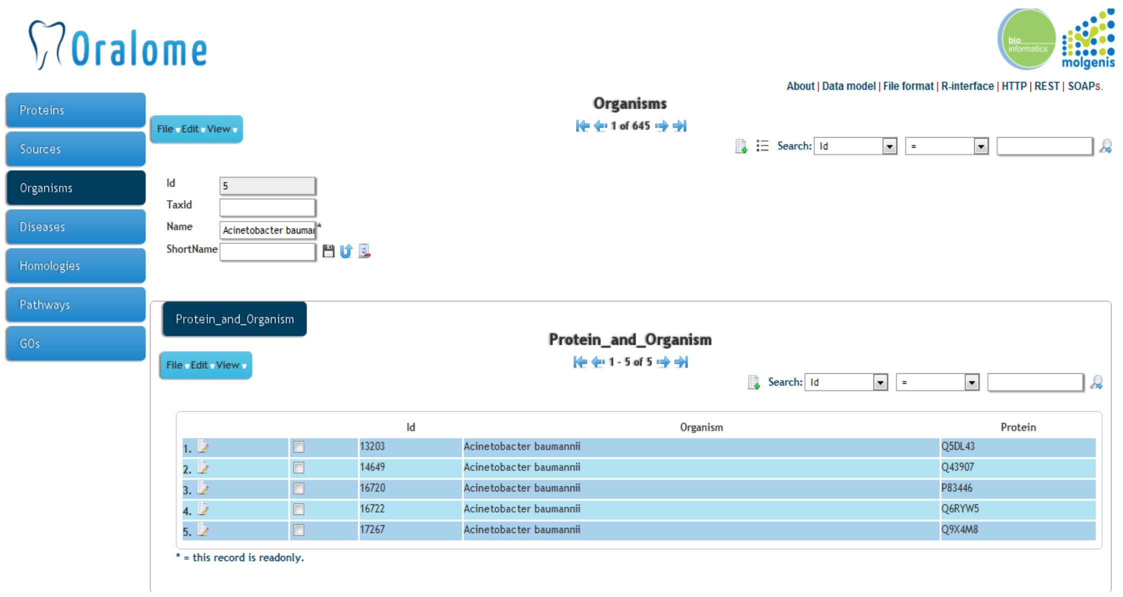
```
<entity name="Organism" implements="Identifiable">
	<description>Relevant organism information.</description>
	<field name="TaxId" type="int" unique="true" nillable="true"
		description="Taxonomic identifier of the organism." />
	<field name="Name" type="string" unique="true" description="Organism
full name." />
	<field name="ShortName" type="string" nillable="true"
		description="Organism short name." />
</entity>
```

Figure 11: Organism Description Structure

The developer can define how the final user interface will be structured by defining the “guidelines” in another XML file, the *molgenis_ui.xml*. This will decide which tables will be available through the web interface and which relationships between tables will be visible to the user. The developer can also introduce some plugins, like email or command line (not implemented in this particular case study).

By providing all the necessary requisites, the next step consists on generating all the necessary code. By running a single file (*MolgenisGenerate.java*), Molgenis generates SQL code, the java model, a web application, and web services almost ready to use. The next step consists in deploying the SQL instructions in the MySQL server by running the file *MolgenisUpdateDatabase.java*. This will create all the tables and relationships between them in which we specified in the previous XML file. The final and optional step is to deploy the generated web application in the Tomcat server. The user can now insert, update, delete and search any data through the web portal.

Figure 12 shows an example of the interface. In the left side there are the links for each table (*Organisms, Proteins, and Diseases*). There is a simple form to input new data and simple search functionality. In the lower end, it is presented the relationships between the selected tables (this example shows the existing proteins for the selected organism). This simple user interface is available at the following link: http://bioinformatics.ua.pt/oralome.



Id	5
TaxId	
Name	Acinetobacter bauma*
ShortName	

Protein_and_Organism

		Id	Organism	Protein
1.	☐	13203	Acinetobacter baumannii	Q5DL43
2.	☐	14649	Acinetobacter baumannii	Q43907
3.	☐	16720	Acinetobacter baumannii	P83446
4.	☐	16722	Acinetobacter baumannii	Q6RYW5
5.	☐	17267	Acinetobacter baumannii	Q9X4M8

* = this record is readonly.

Figure 12: Generated Web Interface using Molgenis

The next step consists on developing tools and wrappers to retrieve information regarding the entities previously mentioned in the class diagrams.

4.2 Database Architecture

In order to gather all the necessary data for the application, we selected the attributes that have the most importance for each entity. The system will have six main entities:

- *Organism*: It is the upper entity. Each organism can have one or more proteins associated;
- *Protein*: It is the main entity, the central aspect in the system's core. A protein usually is associated to one only organism;
- *Disease*: It is a single entity with no dependency. A disease can be related to more than a protein;
- *Pathway*: It is a single entity with no dependency. This table is used to retrieve the associated KEGG pathways related to a protein;
- *Homology*: This table is used to store multiple external information related to a protein, like the ids related to *RefSeq*, *InterPro, Gene3D*, *PFam*, *ProSite* and *Panther*;
- *GO*: Abbreviation for Gene Ontology, it stores the ids related to the protein (ex. *GO:0005576*);
- *PDB*: Stores the Protein Data Bank structure id related to the protein and some general information related to it.

Molgenis can link a table to another by one attribute. For instance, the table *Protein* can have a reference to an *Organism* by invoking its id (a protein is linked to an only organism). But if we want to relate a *Disease* to a *Protein* (a *Protein* can have multiple diseases) we need to manually create linkable tables for this purpose. We will call it the *DiseaseProtein* table. The same logic will be taken for the remaining entities.

We created a table called *SourceProtein* where relations can be created between a source and a protein. A protein can be retrieved from the following sources: parotid, parotid exosome, SM/SL, whole saliva, crevicular fluid, mucosa, tongue, and plasma.

We also take note about the regulation in which a protein was found (increase, decrease or none of the previous), and the condition (health or disease). If a source entry has a disease condition, we include the OMIM of the equivalent. Finally, the evidences are related to the PubMed id which refers the selected protein.

Molgenis can create an automatic id for each table. In order to achieve this, we created an entity *Identifiable*, which the framework will translate into a Java interface.

```
<entity name="Identifiable" abstract="true">
      <description>For modeling purposes only (denoted by
abstract='true', this entity defines id field centrally.
      </description>
      <field name="Id" type="autoid" description="autogenerated id
number (autoid)" />
</entity>
```

Figure 13: The Identifiable entity described in XML

After creating this interface, we implemented it in the other classes (Figure 14).

```
<entity name="Organism" implements="Identifiable">
      <description>Relevant organism information.</description>
      <field name="TaxId" type="int" unique="true" nillable="true"
            description="Taxonomic identifier of the organism." />
      <field name="Name" type="string" unique="true"
description="Organism full name." />
      <field name="ShortName" type="string" nillable="true"
description="Organism short name." />
</entity>
```

Figure 14: The Organism entity described in XML

Next we present the class diagram designed for the Molgenis generator (Figure 15. Each entity has its own specific attributes. For instance, the *Protein* class as attributes for its name, UniprotKBAC, and aminoacid sequence, and attributes for ids used to open specific views in several external services, such as PharmGKB and BRENDA.

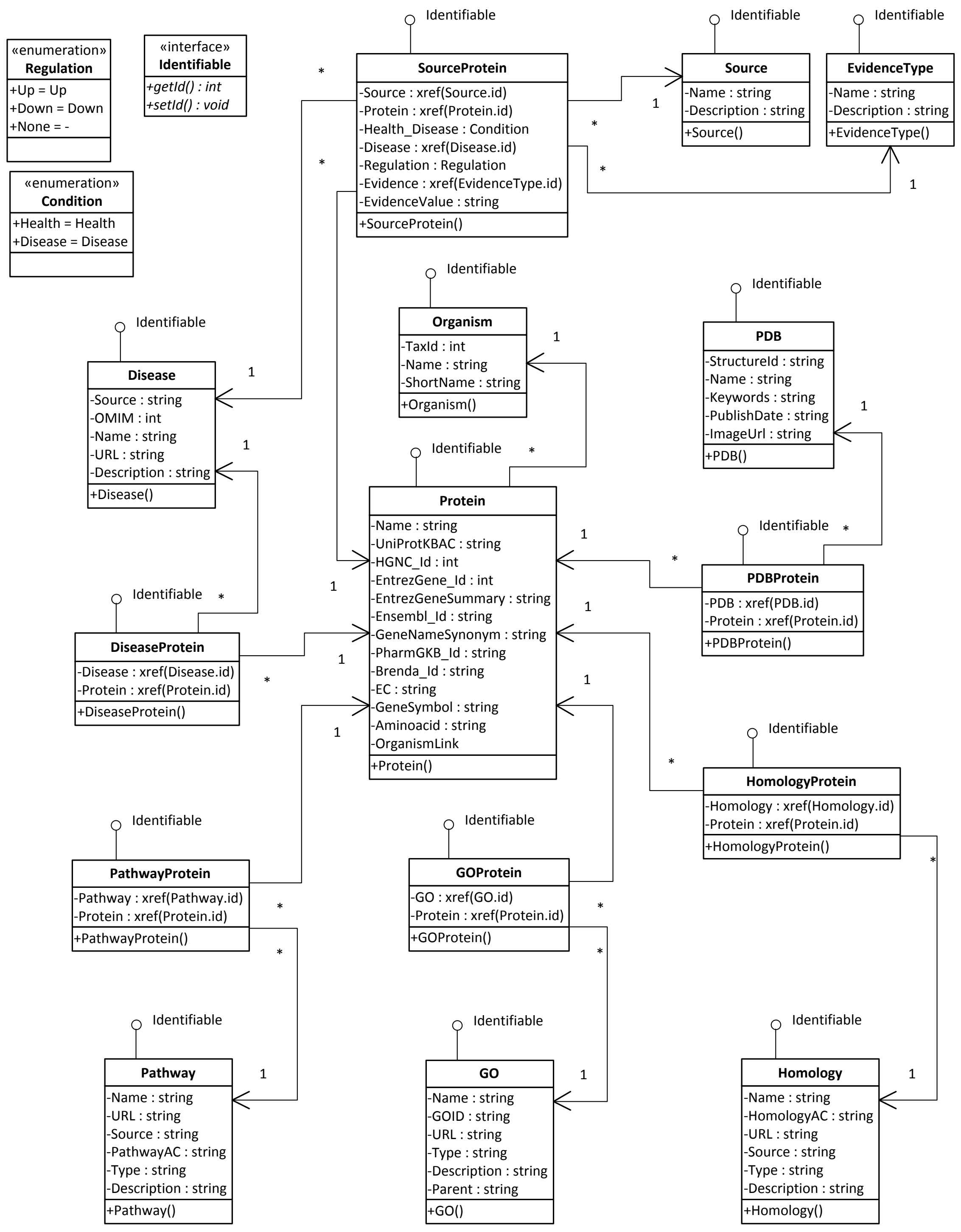


Figure 15: Class diagram for Molgenis generator

4.3 Importing Data

A key objective of the OralCard project was to create tools to obtain information specific to each of the elements that build up the system (proteins, diseases, pathways, and others). For this, we carried out a first survey of sources where this information would be available.

This data fetch is made easier using Molgenis. It is bundled with a Database API[12] that has the advantage of hiding complex SQL commands.

To import and filter the information needed in our database, we used Java as the programming language because it is highly compatible with the most APIs provided by the resources that we will refer through this document (particularly Molgenis).

Importing Protein's Data

For the particular case of proteins, the data source used was UniProt, the most complete resource for protein sequence and functional information. It identifies each protein with an accession code, which we will denominate as the UniProtKBAC (The UniProt Knowledgebase Accession Code), and it is composed by one letter and several digits (ex. *P26572*). This code is used as an input for the UniProt web service, which returns a descriptive XML file with the information about the protein. This web service is accessible in the format http://www.uniprot.org/uniprot/P26572.xml, where *P26572* is the UniProtKBAC.

In order to parse the provided XML files from Uniprot, we have used the DOM parser Java library from W3C. We could use the StAX[13] (Streaming API for XML) instead, which is also capable of reading XML files. The Document Object Model (DOM) is an abstract data structure that represents XML documents as trees made up of nodes. The *org.w3c.dom* package contains various interfaces that represent elements, attributes, parsed character data, comments, and processing instructions. All of these are subinterfaces of the *Node* interface, which gives basic methods for navigation and trimming the tree [70].

A parser reads an XML document from a stream and builds a *Document* object that is a tree representation of the whole document. The developer can then call

[12] http://www.molgenis.org/wiki/MolgenisDatabaseApi
[13] http://stax-utils.dev.java.net/

Document methods and other DOM services to navigate the tree and extract the desired information [70].

Figure 16 shows an excerpt of some of the information provided in a UniProt XML file (in this particular case, the information regards to the *P26572* protein). The node *uniprot* is a child node of the document's root, with several elements. From these elements, we extract the necessary information, for instance, the *name* of the *entry*, the *recommendedName* of the *protein*, the *name* of the *organism* as its NCBI Taxonomy id.

The UniprotJAPI[14], from EMBL-EBI (European Bioinformatics Institute) is also being used in order to fetch ids for external databases, such as HGNC, BRENDA or PharmGKB.

[14] http://www.ebi.ac.uk/uniprot/remotingAPI/

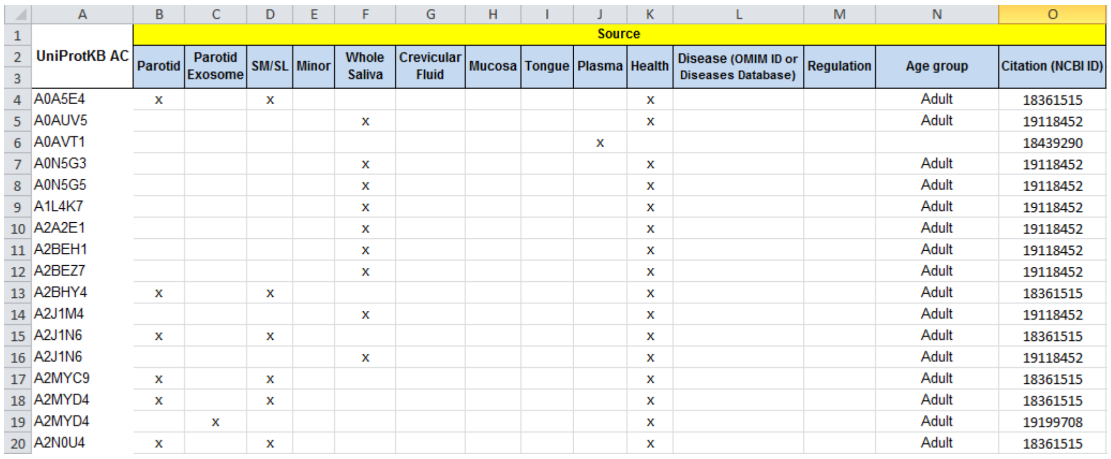
```xml
<?xml version='1.0' encoding='UTF-8'?>
<uniprot xmlns="http://uniprot.org/uniprot" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
       xsi:schemaLocation="http://uniprot.org/uniprot
http://www.uniprot.org/support/docs/uniprot.xsd">
       <entry dataset="Swiss-Prot" created="1992-08-01" modified="2011-04-05"
              version="106">
              <accession>P26572</accession>
              <accession>A8K404</accession>
              <accession>D3DWR1</accession>
              <accession>Q6IBE3</accession>
              <name>MGAT1_HUMAN</name>
              <protein>
                     <recommendedName ref="1">
                            <fullName>Alpha-1,3-mannosyl-glycoprotein 2-beta-N-
acetylglucosaminyltransferase</fullName>
                     </recommendedName>
                     <alternativeName>
                            <fullName>N-glycosyl-oligosaccharide-glycoprotein N-
acetylglucosaminyltransferase I</fullName>
                            <shortName>GNT-I</shortName>
                            <shortName>GlcNAc-T I</shortName>
                     </alternativeName>
              </protein>
              <gene>
                     <name type="primary">MGAT1</name>
                     <name type="synonym">GGNT1</name>
                     <name type="synonym">GLCT1</name>
                     <name type="synonym">GLYT1</name>
                     <name type="synonym">MGAT</name>
              </gene>
              <organism>
                     <name type="scientific">Homo sapiens</name>
                     <name type="common">Human</name>
                     <dbReference type="NCBI Taxonomy" id="9606" key="2" />
                     <lineage>
                            <taxon>Catarrhini</taxon>
                            <taxon>Hominidae</taxon>
                            <taxon>Homo</taxon>
                     </lineage>
              </organism>

       </entry>
       <copyright>
              Copyrighted by the UniProt Consortium, see http://www.uniprot.org/terms
              Distributed under the Creative Commons Attribution-NoDerivs License
       </copyright>
</uniprot>
```

Figure 16: Excerpt of a UniProt XML file

As a starting point, we have used a previously created XLSX file with 17 738 UniProt entries (Figure 17, Appendix A-1). This file contains a list of UniProtKBACs, the biological sources in which the protein was found, information regarding to the situation the protein was found (health or disease situation), the regulation (up regulation or down regulation), the age group and the corresponding NCBI citation. In order to read this XLS file through the Java environment, we are using the OpenCSV[15] parser library.

[15] http://opencsv.sourceforge.net/



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	UniProtKB AC	Source													
2–3		Parotid	Parotid Exosome	SM/SL	Minor	Whole Saliva	Crevicular Fluid	Mucosa	Tongue	Plasma	Health	Disease (OMIM ID or Diseases Database)	Regulation	Age group	Citation (NCBI ID)
4	A0A5E4	x		x							x			Adult	18361515
5	A0AUV5					x					x			Adult	19118452
6	A0AVT1									x					18439290
7	A0N5G3					x					x			Adult	19118452
8	A0N5G5					x					x			Adult	19118452
9	A1L4K7					x					x			Adult	19118452
10	A2A2E1					x					x			Adult	19118452
11	A2BEH1					x					x			Adult	19118452
12	A2BEZ7					x					x			Adult	19118452
13	A2BHY4	x		x							x			Adult	18361515
14	A2J1M4					x					x			Adult	19118452
15	A2J1N6	x		x							x			Adult	18361515
16	A2J1N6					x					x			Adult	19118452
17	A2MYC9	x		x							x			Adult	18361515
18	A2MYD4	x		x							x			Adult	18361515
19	A2MYD4		x								x			Adult	19199708
20	A2N0U4	x		x							x			Adult	18361515

Figure 17: Registration of researched UniProts on XLSX file (client version)

For each input read, we first analyse the corresponding UniProt XML file and check its organism. If it is not available in our database, we insert a new record and link the protein to it. Next we check for the sources and its evidences. When a repeated UniProtKBAC is found in the XLSX file, we simple merge the new information into the existing one. We also extract from the protein XML file all the necessary external ids (concerning to homologies, pathways, diseases, PDB's ids, and gene ontologies).

Importing Disease's Data

In order to retrieve data regarding to a specific disease, we used the NCBI Entrez Programming Utilities[16]. It consists on several REST tools that provide access to Entrez data outside of the regular web query interface and it's helpful for retrieving search results in the Java environment. We need to provide three parameters to the *EFetch* web service URL[17] : OMIM as the database, the disease OMIM id, and XML as the format. After that we get a descriptive XML file, just like the UniProt output previously described.

Occasionally this web service is down. As an alternative, we fetch the disease name using a similar service, the *ESummary*[18]

[16] http://www.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html
[17] http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi
[18] http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi

Importing Pathway's Data (KEGG)

With the aim of fetching information regarding pathways, we used the KEGG API[19]. It consists on a SOAP/WSDL and REST interface to the KEGG system (built using Java), allowing for searching biochemical pathways in cellular processes or analysing the universe of genes in the completely sequenced genomes.

As input, we provide the KEGG id presented in the UniProt XML output and as a result we obtain the corresponding pathways codes (ex. *hsa:4245* results on *hsa00510* and *hsa01100*).

Importing PDB's Data

As the importation of diseases or proteins, fetching PDB data goes in the same line, using a REST web service[20]. It requires the structure id contained in the UniProt XML file in order to output a result. For instance, for the *structureId* 1A85 it outputs the result shown in the Figure 18.

```
<?xml version='1.0' standalone='no' ?>
<PDBdescription>
       <PDB structureId="1A85" title="MMP8 WITH MALONIC AND ASPARAGINE BASED INHIBITOR"
              pubmedId="9655333" expMethod="X-RAY DIFFRACTION" resolution="2.00"
              keywords="COMPLEX (COLLAGENASE/INHIBITOR)" nr_entities="4"
              nr_residues="158" nr_atoms="1537" publish_date="1998-04-03"
              revision_date="1999-04-27"
              audit_authors="Brandstetter, H., Roedern, E.G.V., Grams, F., Engh, R.A."
              citation_authors="Brandstetter, H., Engh, R.A., Von Roedern, E.G.,
Moroder, L., Huber, R., Bode, W., Grams, F."
              status="CURRENT" />
</PDBdescription>
```

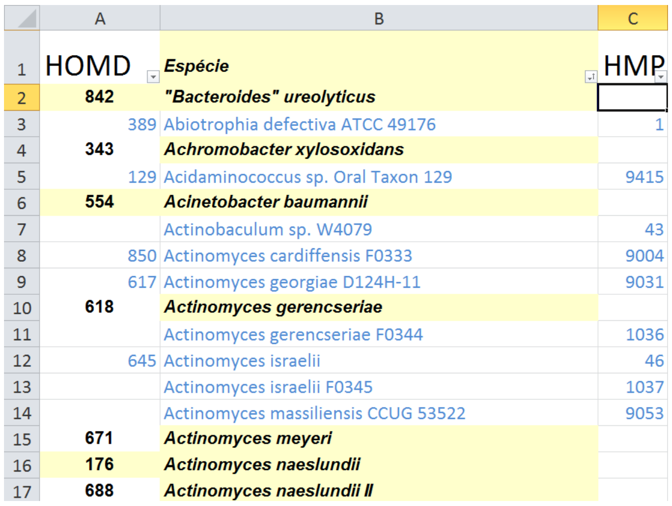
Figure 18: PDB XML file for Structure Id 1A85

Here, we extract the title, keywords and the date of publish. As in the previous cases, data filtering is done by using the DOM parse library from W3C.

Importing Proteins from other Organisms

Another requirement for OralCard was to import proteins regarding to organisms other than human. These specific organisms are described in an XLSX file, organized by rows (Figure 19, Appendix A-2):

[19] http://www.genome.jp/kegg/soap/
[20] http://www.pdb.org/pdb/rest/describePD



	A	B	C
1	HOMD	*Espécie*	HMP
2	**842**	***"Bacteroides" ureolyticus***	
3	389	Abiotrophia defectiva ATCC 49176	1
4	**343**	***Achromobacter xylosoxidans***	
5	129	Acidaminococcus sp. Oral Taxon 129	9415
6	**554**	***Acinetobacter baumannii***	
7		Actinobaculum sp. W4079	43
8	850	Actinomyces cardiffensis F0333	9004
9	617	Actinomyces georgiae D124H-11	9031
10	**618**	***Actinomyces gerencseriae***	
11		Actinomyces gerencseriae F0344	1036
12	645	Actinomyces israelii	46
13		Actinomyces israelii F0345	1037
14		Actinomyces massiliensis CCUG 53522	9053
15	**671**	***Actinomyces meyeri***	
16	**176**	***Actinomyces naeslundii***	
17	**688**	***Actinomyces naeslundii II***	

Figure 19: Excerpt of XLSX file containing other organisms

The goal was to confront this organisms with a list of gene ontologies ids, provided in another document (Appendix A-3). This document is presented in the DOCX file format, and we are reading it using the Apache POI[21], a Java API for Microsoft documents.

The UniProt REST web service[22] makes easy the search and retrieval task. By building a simple query, it outputs a list of proteins that match our parameters (Figure 20). For each retrieved protein, the fetching data process is done in the same way as previously described

```
http://www.uniprot.org/uniprot/?query=

organism:"Bacillus subtilis"+

AND + reviewed:yes +

AND +(go:0042597 + OR + go:0005737 + OR + go:0005634)

&format=list
```

Figure 20: Example of a query to the UniProt REST web service

Runtime

In order to fulfil the database, and resuming the previous sections, we need to execute three steps. The first one is to create the oral cavity sources in the *Source* table. This is done running the code provided in the file *StartUpDatabase.java*.

[21] http://poi.apache.org
[22] http://www.uniprot.org/uniprot

Next, we update the database with the proteins and their relationships provided in the XLSX file (Figure 17). This is done by running the script contained in the file *UpdateProteinsDatabase.java*.

Finally, with the aim of updating the OralCard database with proteins regarding to other organisms, we run the script *UpdateProteinsOtherOrganisms.java*. After these steps, our database is completely fulfilled and ready to be searched.

4.4 Frontend Development

After deploying our initial web interface generated by the Molgenis framework, we needed tools for visualization, for simple query of the database, and for insert, update and delete records.

We also needed a customized and detailed view for each entity present on the database, as also others features like counting the proteins for each source or presenting search results in real time.

Considering these aspects, we developed a frontend for the OralCard project where these features are available. This development was done using the Stripes framework (described in the section 2.5.1) and the NetBeans IDE 7.0[23].

For the development of the OralCard frontend, we decided for the following architecture (Figure 21): J2EE technology, MySQL for storage, Hibernate as the persistence framework, and Stripes for the business and client layer. jQuery was also used along with Stripes in order to improve the client side of the application. We kept the database we have created previously using Molgenis, and from that we built the top tree.

[23] http://netbeans.org/

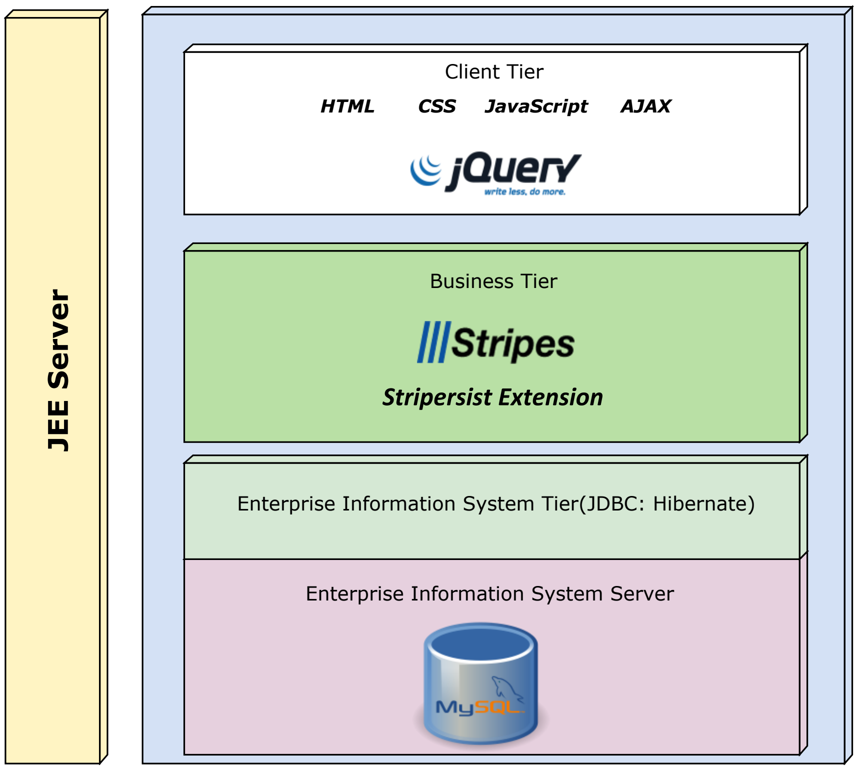


Figure 21: OralCard Frontend Architecture

For the development of this system, we chose the Stripes framework, as mentioned previously in the section 2.5.

Mapping the Database

With the aim of accessing the database and transform it to a Java model, we had three possible solutions. The first one was using one of the generated web services, REST. It has several important advantages, like scalable component interactions, general interfaces, independently deployed connectors, reduced interaction latency and strengthened security. Molgenis REST interface outputs JSON[24] objects, which are a lightweight data-interchange format, easy for machines to parse and generate, and gradually a replacement for XML objects.

The downside of it, as a generated web service, is that it only supports GET/PUT/POST/DELETE, and one need to have previous knowledge of the corresponding database entry id (Figure 22). In order to make it work, we had to fetch an entire table, query it to find the corresponding database entry id, and then make the

[24] http://json.org/

transaction. This would take a large amount of time, especially for tables with thousands of entries.

REST Query:
http://bioinformatics.ua.pt/oralome/api/rest/json/protein/2

JSON Object:

```
{
   "protein":{
      "readonly":false,
      "id":2,
      "__Type":"Protein",
      "___Type_options":[
         {
            "label":"Protein",
            "value":{
               "@xsi.type":"xs:string",
               "$":"Protein"
            }
         },
         {
            "label":"Interaction",
            "value":{
               "@xsi.type":"xs:string",
               "$":"Interaction"
            }
         }
      ],
      "name":"Ig heavy chain V-III region BUR",
      "uniProtKBAC":"P01773",
      "aminoacid":"QVQLVESGGGVVQAGTSLRLSCTASAFNLSDYAM
                   HWVRQAPGKGLZWVALISYGGSBTYYADSVRGR
                   FTISRBISKBTLYLZMKTLRTEDTAVYYCAKLI
                   AVAGTRBFWGQGTLVTVSL",
       "organismLink":483,
      "_organismLink_Name":"Homo sapiens"
   }
}
```

Figure 22: REST Query on a specified protein

Also, this web service does not support general queries for others attributes. For instance, it is impossible to query the database in order to fetch the details of a protein named *Ig heavy chain V-III region BUR* without knowing its id.

The second option was to use another generated service, SOAP. Despite the architecture being different, it also has the same main access problem of REST.

Having Molgenis this downside, and with the aim of solving this problem, we could develop our own REST or SOAP interface, personalized in such way we could make any kind of query to any attribute of any table. But there are frameworks developed for the purpose of interacting with an existing database, and that's why we chose the third solution: querying directly the MySQL database.

In order to make the link between the Java model objects and the database (previously created using Molgenis), there are several solutions available: using plain JDBC, using a library that facilitates the interaction with JDBC but places on the developer the responsibility of writing SQL, using an ORM framework, and others.

In this particular case, we decided to use JPA and Hibernate[25]. JPA is Sun's standard persistence specification and there is a library specifically designed to integrate JPA with Stripes. Concerning to the specification, we chose Hibernate because it is widely used, and some previous experience with it was acquired. Other solutions could be used, like OpenJPA[26] or JPOX[27].

Stripersist[28] is a Java library that facilitates the integration of JPA in Stripes, and we are using it in order to save some time around needed configurations.

The first step was to convert the relational database to an object-oriented domain model. To achieve this goal, we used the Netbeans functionality of importing *Entity Classes from Database*. It uses the JDBC connector for MySql, and creates a Java Persistence API entity class for each selected database table, complete with named query annotations, fields representing columns, and relationships representing foreign keys.

In order to set the JPA configuration, we need to place the *persistence.xml* file. It will contain some information telling JPA to use Hibernate, and configure Hibernate to use the MySQL database (URI, username, password, and others). The next step is to tell Stripes framework to use Stripersist, by referring it as an extension package.

The Business and Client Tier

For the business layer, Stripes provides an extension called Stripersist, which makes the integration of third-party libraries quite simple. It is responsible of creating methods for creating, synchronizing and removing the data in the database, much like an EJB container is responsible for. With a relational database previously created using MySQL server, an object-to-relational mapping is applied fully automatically.

[25] http://www.hibernate.org/
[26] http://openjpa.apache.org/
[27] http://www.jpox.org/
[28] http://www.stripes-stuff.org/

In order to configure the Stripersist, we need to place the JPA configuration in the *persistence.xml* file. This is where we tell JPA to use Hibernate and configure it to use the MySQL database, as described (Enterprise Information System Tier). Since the model classes have been generated using the tool provided by the NetBeans IDE (Entity Classes from Databases), we do not need to worry about adding JPA annotations. This has already been done automatically (example: adding the *@Entity* annotation).

The next step is to create methods to read data from the database. We implemented a Java DAO interface for each entity present in our database that includes methods for reading an entire table or a specific entry, as also methods for searching. For instance, for the *Protein* entity, we have a *ProteinDao* interface and a *ProteinDaoImpl* class which will implement those methods of reading and searching. This implementation class will also extend a *BaseDaoImpl* class, which contains the core methods for reading, committing and finding entries. The *BaseDaoImpl* is where we take the advantage of the Stripersist package. For instance we can read a specific entry by invoking the command described below (Figure 23).

```
public T read(ID id) {
    return Stripersist.getEntityManager().find(entityClass, id);
}
```

Figure 23: Using Stripersist for reading from database

The *entityClass* can be a *Protein*, and the id identifies the entry we want to read. This automatically returns an object related to the entity we read.

Using this approach the DAOs can use Stripersist and JPA easily, and the rest of the application can use the DAOs without being exposed to the details of the persistence layer.

In addition, we are using Spring[29] for dependency injection. Spring is an open source application framework for the Java platform, and dependency injection is the concept of providing, from the outbound, implementations to classes that need them. Using this methodology, classes have references only to interfaces, and not to any specific implementation. It becomes easier to use different implementations by

[29] http://www.springsource.org

changing only the configuration. The code becomes more flexible, and testing the application is easier: we only have references to interfaces [27].

For instance, in order to establish a relationship between the *ProteinDaoImpl* and the *BaseActionBean* (which is responsible of invoking instances of the DAOs), the developer only needs to incorporate the *@Repository* (Figure 24) and the *@SpringBean* (Figure 25) annotations.

```
@Repository("proteinDao")
public class ProteinDaoImpl extends BaseDaoImpl<Protein, Integer> implements ProteinDao
```

Figure 24: Using the *@Repository* annotation in the ProteinDaoImpl

```
@SpringBean protected ProteinDao proteinDao;
```

Figure 25: Using the *@SpringBean* annotation in the *BaseActionBean*

Regarding the client tier, we are using JSP combined with Stripes. The main purpose is to write a JSP that retrieves the information from the action beans that were developed using the Stripes framework. With these actions beans, we can redirect requests to specific web pages, and take full control of the web flow.

We are using JavaScript and the jQuery framework to improve the user interface. jQuery[30] is a free, open source, cross-browser JavaScript library that was designed to simplify the client-side scripting of HTML. Several jQuery plugins, such as DataTables[31], were used in order to transform raw HTML information, into JavaScript, user friendly, data holders.

In order to achieve the autocomplete feature while querying the OralCard database, we used the jQuery Autocomplete plugin with Stripes. We implemented an action bean which will be invoked by an AJAX request, as long as the user is typing the

[30] http://jquery.com/
[31] http://www.datatables.net/

search string. An AJAX response will be delivered to the plugin, which will present the matched results in a dropdown list.

Finally, the OralCard web application takes advantage of the CSS benefits. It is designed to separate the document content written in JSP from the document presentation, including elements such as the page layout, used colors and fonts. The OralCard web application has its style defined in two separate files: The *home.css* for the home screen design, and the *style.css* for the rest of the application.

4.5 Summary

This section described the technical aspects considered for the development of the OralCard web application. First we explained how the Molgenis framework works, specifying its architecture. It requires the developer to describe a XML file with the entities that will compose the system (organism, protein, disease, gene ontology, and others). We also were required to describe tables that would make the link between two main entities. For instance, we need an association class *PDBProtein* that will establish a relation between a *Protein* and a *PDB*.

We presented the class diagram reflecting the specification made in the *molgenis_db.xml* file, the place we insert and described our tables.

After determining all the requirements, Molgenis generates a Java model, SQL tables and a trivial web user interface. The system is now ready for importing data. In order to achieve this task, we have described the methods chosen to gather the necessary information for insertion in the generated tables.

The second part of the project was related to the development of a web user interface. In this part we described the frontend architecture, the tools used (MySQL, Hibernate, Stripes, and jQuery) and some technical aspects concerning to the implementation itself.

In the next chapter they will be present the results achieved after the work implementation.

5. Results

5.1 Importing Data

In order to fetch the relative data for the OralCard web application, we designed two scripts. The first one will read the UniProtKBACs from the provided XLSX file (Appendix A-1), and retrieve the necessary data, inserting it in the Oralome database. The second script will read the microorganisms from a XLSX file (Appendix A-2). For each microorganism, it will build a REST request with the gene ontology ids provided in the DOCX file (Appendix A-3). This REST request will generate a REST response with the UniProtKBACs that match the query.

After querying the Uniprot web service with all the given microorganisms and gene ontology ids, we managed to get 54 805 UniProt Knowledgebase accession codes. Next, we repeat the retrieval process made in the first script.

For the first task, we used an approach with no threads. Each protein *y* has a time window *x* necessary for retrieving all the necessary data, and the next protein *y+1* will be explored when the protein *y* has finished (Figure 26). This method has the advantage of not overloading the servers used to retrieve information, but it is considerable slow. In order to fetch data from 11 413 proteins (Appendix A-1) using a broadband connection, the application took around 48 hours. This slowness is derived from the delay window between a request made to the NCBI web services and its response. The Uniprot web service's response has a delay window significantly short, taking out some situations in which the service is unresponsive, forcing the application to wait a random time in order to repeat the request.

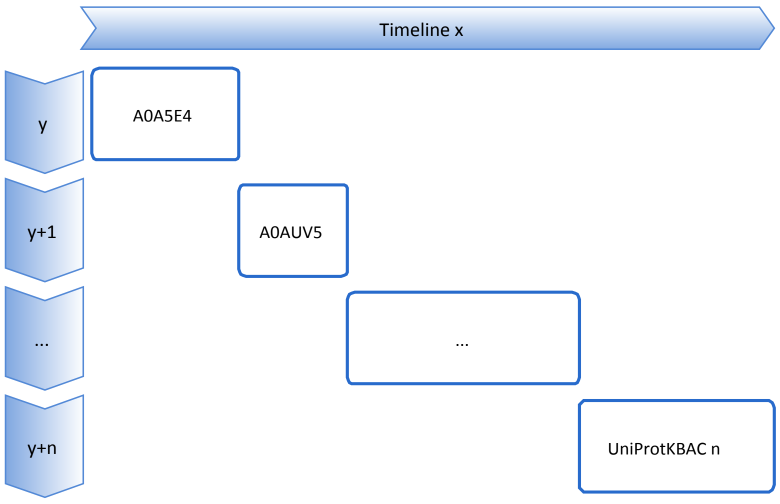


Figure 26: Importing proteins without multithreading

With the goal of speeding up this process, we used threads in our Java code. Having a multithreaded application means to deliver its potent power by running many threads concurrently within a single program. The operating system can treat the program as a bunch of separate and distinct processes. We tried to achieve a compromise between the number of threads, the number of proteins per thread, and occupied memory. Using a personal computer with 4GB of RAM, we managed to create a thread *y* for each group of 500 proteins (Figure 27). Using this approach, the application was able to retrieve 66 218 proteins in about 28 hours (using the two scripts).

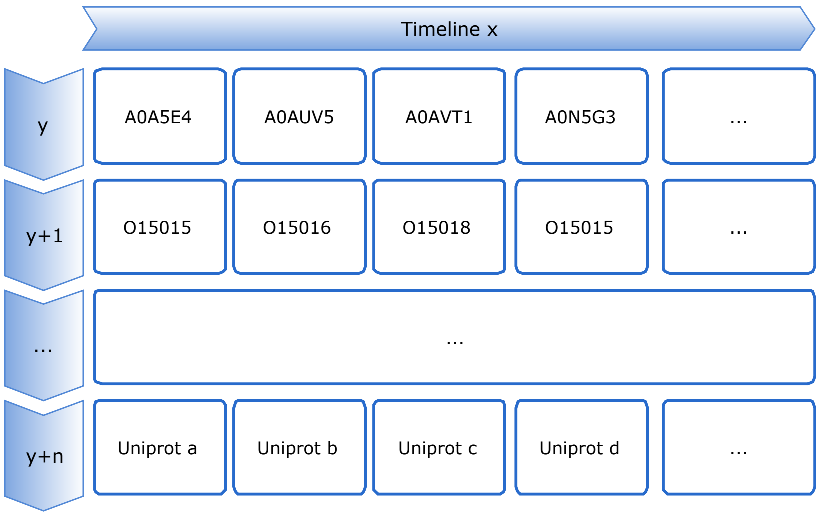


Figure 27: Importing proteins using multithreading

The second approach is significantly faster, but forced us to consider the denial of service from the external web services (NCBI, Uniprot and RCSB PDB), which sometimes happens for a short time window. The KEGG API only supports one connection to the web service at a time, and this can reduce the throughput, but not significantly.

The retrieved information is available at http://bioinformatics.ua.pt/oralome.

5.2 Frontend

5.2.1 The Home Page

The OralCard web portal should provide search functionalities for every entity contained in the database. For that, we included a home page with a text box where the user is able to search for anything inside the domain of proteins, diseases and gene ontologies (Figure 28). These can be a protein name or UniProtKBAC, a disease name or its OMIM accession code, gene ontology name or its accession code, or even an organism name.

As long as the user is typing a search string, the web application is constantly querying the database in order to present a dropdown list with suggestions. This is done using the AJAX possibilities (Figure 29).

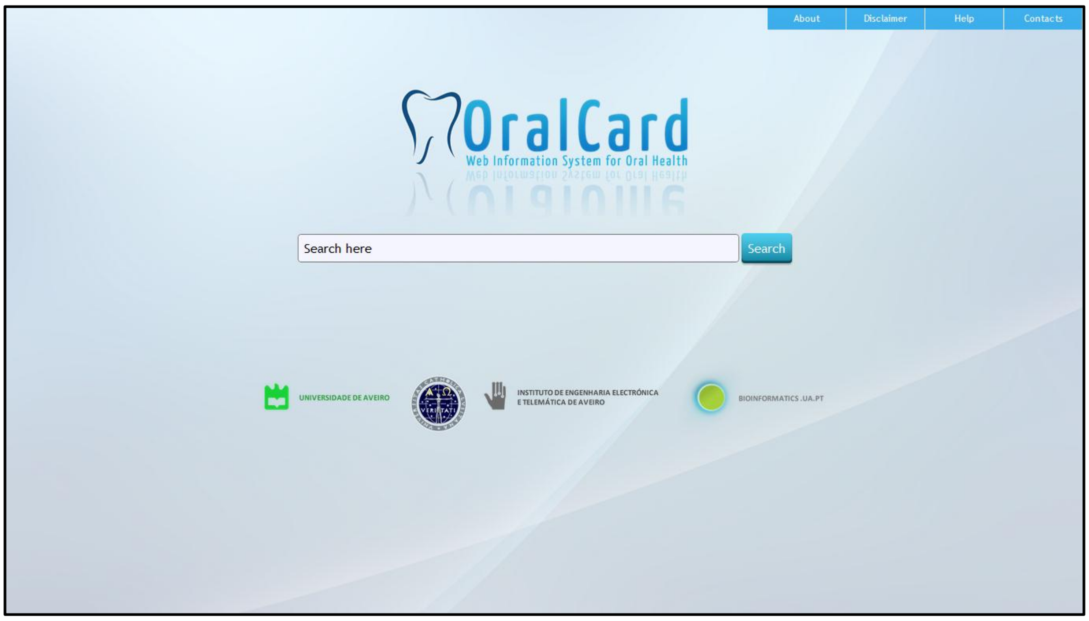

About
Disclaimer
Help
Contacts
OralCard
Web Information System for Oral Health
Search here
Search
UNIVERSIDADE DE AVEIRO
INSTITUTO DE ENGENHARIA ELECTRÓNICA E TELEMÁTICA DE AVEIRO
BIOINFORMATICS.UA.PT

Figure 28: The OralCard home page

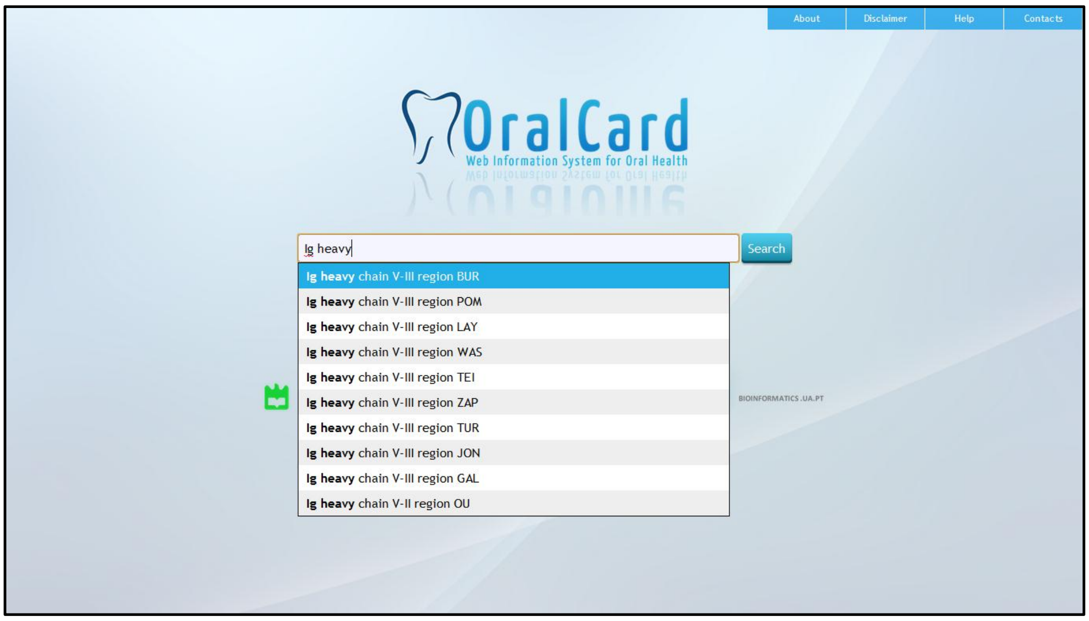

About
Disclaimer
Help
Contacts
OralCard
Web Information System for Oral Health
Ig heavy
Search
Ig heavy chain V-III region BUR
Ig heavy chain V-III region POM
Ig heavy chain V-III region LAY
Ig heavy chain V-III region WAS
Ig heavy chain V-III region TEI
Ig heavy chain V-III region ZAP
Ig heavy chain V-III region TUR
Ig heavy chain V-III region JON
Ig heavy chain V-III region GAL
Ig heavy chain V-II region OU
BIOINFORMATICS.UA.PT

Figure 29: Suggestions in the home page while searching

In order to restrict a query to specific tables, the user can start his search with the entity type. For instance, searching for an organism *Homo sapiens* is made easy by first typing the key word *organism*. Suggestions only matters to the organism table (Figure 30).

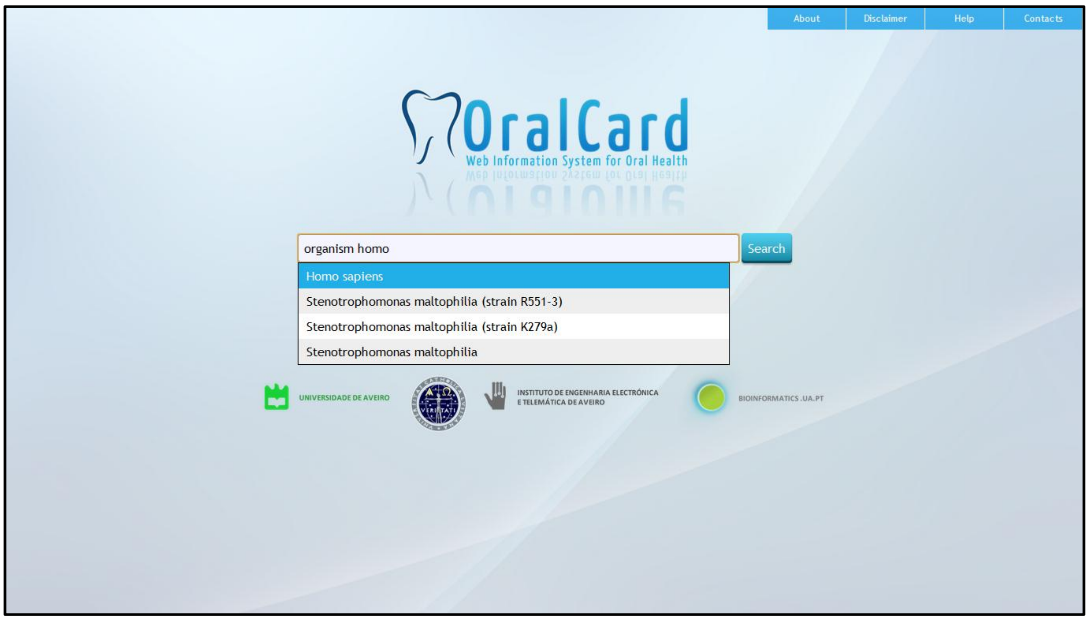

About
Disclaimer
Help
Contacts
OralCard
Web Information System for Oral Health
organism homo
Search
Homo sapiens
Stenotrophomonas maltophilia (strain R551-3)
Stenotrophomonas maltophilia (strain K279a)
Stenotrophomonas maltophilia
UNIVERSIDADE DE AVEIRO
INSTITUTO DE ENGENHARIA ELECTRÓNICA E TELEMÁTICA DE AVEIRO
BIOINFORMATICS.UA.PT

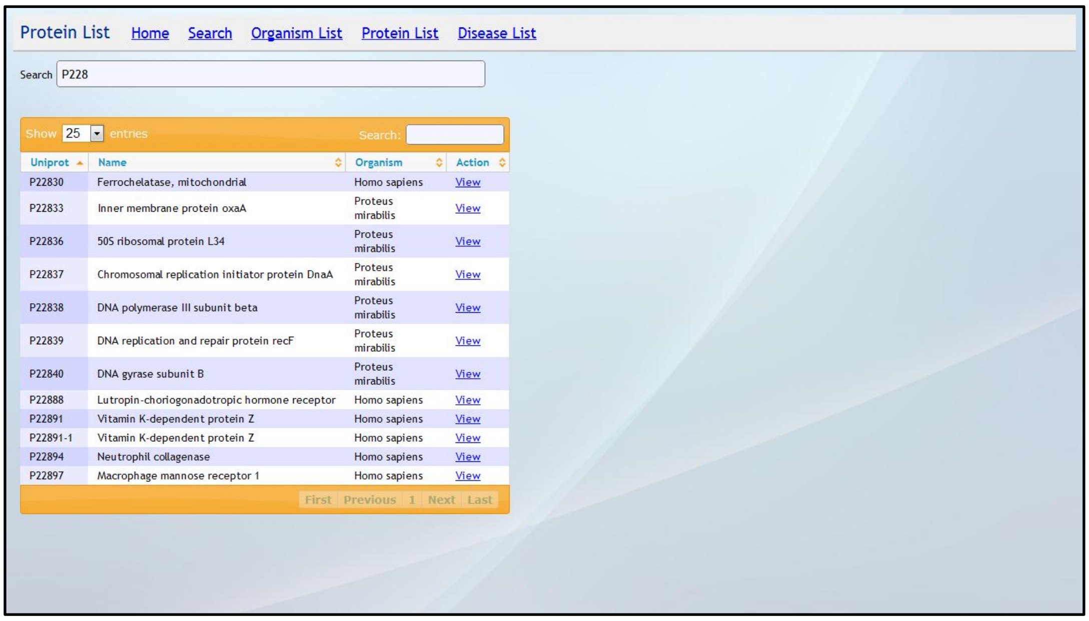
Figure 30: Specifying the organism entity while searching

We also included 4 sections in the home page: *About*, *Disclaimer*, *Help* and *Contacts*. These will serve the purposes of user assistance and general information about the web application.

5.2.2 Protein Search and Details

We designed a specific web page for each entity. The page designed for the protein has a search box where the user can input a query. This query can be a protein name or a UniProtKBAC. Live results will be displayed using AJAX and a JavaScript data table. General information is given in order to identify the protein easily: The UniProt code, the protein's name and the organism which is related.

In the following illustration (Figure 31) the user is searching for a UniProtKBAC, starting with *P228*. The table is refreshed with all the proteins that have some similarity with the query. This table can be filtered itself using its search box (for instance, the user can filter the results for the *Homo sapiens* organism). Finally, the table can be sorted using each column.



Protein List Home Search Organism List Protein List Disease List

Search P228

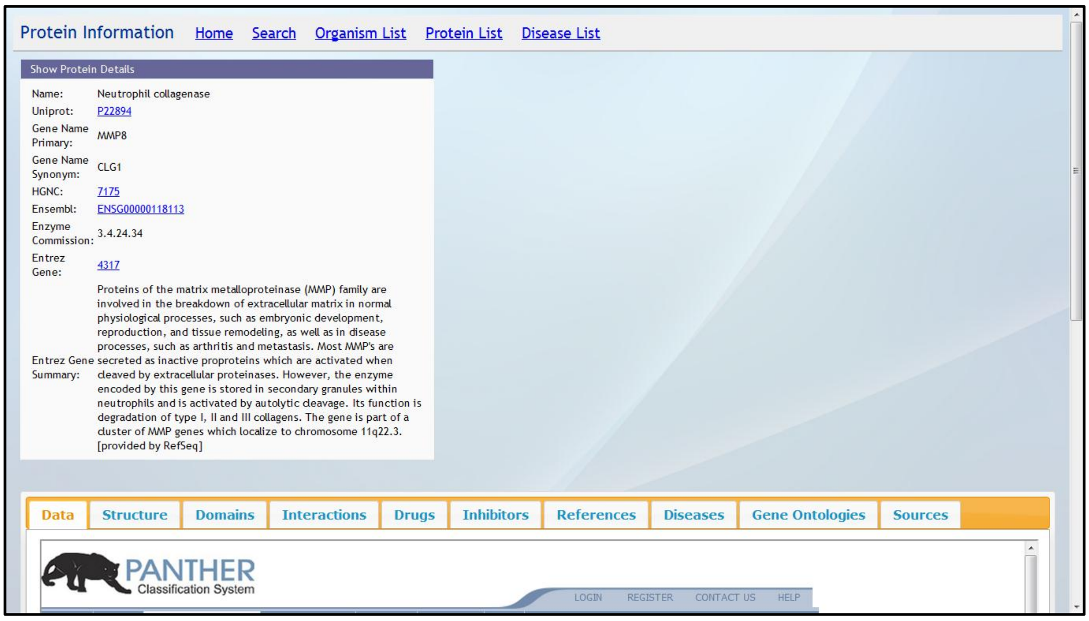
Show 25 entries Search:

Uniprot	Name	Organism	Action
P22830	Ferrochelatase, mitochondrial	Homo sapiens	View
P22833	Inner membrane protein oxaA	Proteus mirabilis	View
P22836	50S ribosomal protein L34	Proteus mirabilis	View
P22837	Chromosomal replication initiator protein DnaA	Proteus mirabilis	View
P22838	DNA polymerase III subunit beta	Proteus mirabilis	View
P22839	DNA replication and repair protein recF	Proteus mirabilis	View
P22840	DNA gyrase subunit B	Proteus mirabilis	View
P22888	Lutropin-choriogonadotropic hormone receptor	Homo sapiens	View
P22891	Vitamin K-dependent protein Z	Homo sapiens	View
P22891-1	Vitamin K-dependent protein Z	Homo sapiens	View
P22894	Neutrophil collagenase	Homo sapiens	View
P22897	Macrophage mannose receptor 1	Homo sapiens	View

First Previous 1 Next Last

Figure 31: The protein search web page

Selecting the *view* link, we have access to the details of the selected protein (Figure 32). We have included a summary of the protein (with its gene name, gene synonym, HGNC, and others), and a JavaScript tab table where the user can select specific information using different tools: *Data*, *Structure*, *Domains*, *Interactions*, *Drugs*, *Inhibitors*, *References*, *Diseases*, *Gene Ontologies*, and *Sources.*



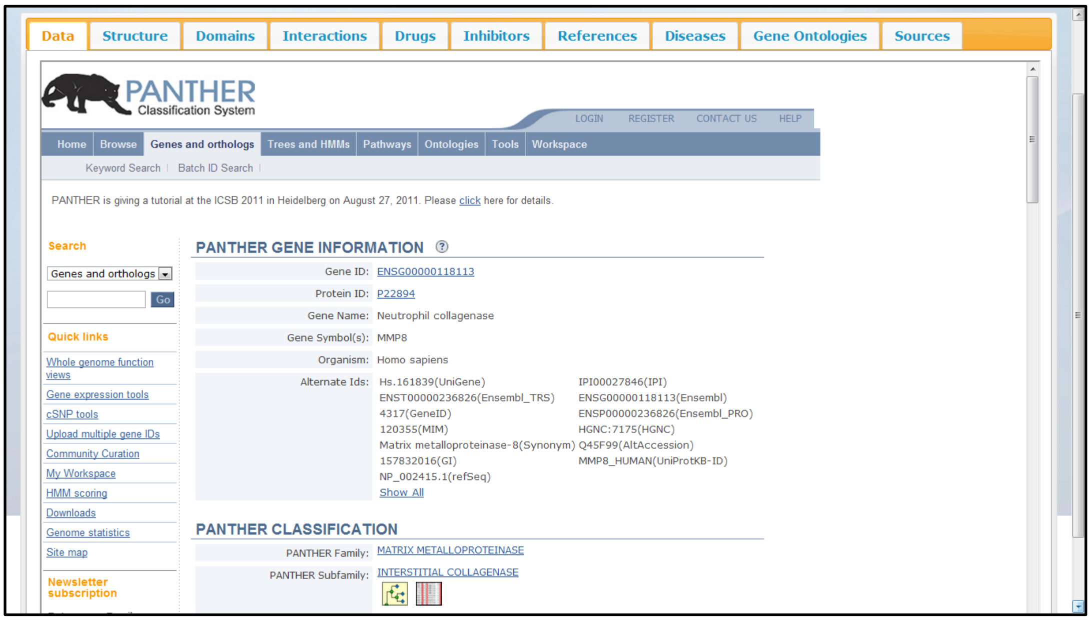
Protein Information Home Search Organism List Protein List Disease List

Show Protein Details

Name:	Neutrophil collagenase
Uniprot:	P22894
Gene Name Primary:	MMP8
Gene Name Synonym:	CLG1
HGNC:	7175
Ensembl:	ENSG00000118113
Enzyme Commission:	3.4.24.34
Entrez Gene:	4317
Entrez Gene Summary:	Proteins of the matrix metalloproteinase (MMP) family are involved in the breakdown of extracellular matrix in normal physiological processes, such as embryonic development, reproduction, and tissue remodeling, as well as in disease processes, such as arthritis and metastasis. Most MMP's are secreted as inactive proproteins which are activated when cleaved by extracellular proteinases. However, the enzyme encoded by this gene is stored in secondary granules within neutrophils and is activated by autolytic cleavage. Its function is degradation of type I, II and III collagens. The gene is part of a cluster of MMP genes which localize to chromosome 11q22.3. [provided by RefSeq]

Figure 32: Protein Details

The first tab, *Data*, contains a frame that contains gene information and classification using the Panther[32] classification system.



PANTHER is giving a tutorial at the ICSB 2011 in Heidelberg on August 27, 2011. Please click here for details.

PANTHER GENE INFORMATION

Gene ID:	ENSG00000118113
Protein ID:	P22894
Gene Name:	Neutrophil collagenase
Gene Symbol(s):	MMP8
Organism:	Homo sapiens
Alternate Ids:	Hs.161839(UniGene) IPI00027846(IPI) ENST00000236826(Ensembl_TRS) ENSG00000118113(Ensembl) 4317(GeneID) ENSP00000236826(Ensembl_PRO) 120355(MIM) HGNC:7175(HGNC) Matrix metalloproteinase-8(Synonym) Q45F99(AltAccession) 157832016(GI) MMP8_HUMAN(UniProtKB-ID) NP_002415.1(refSeq) Show All

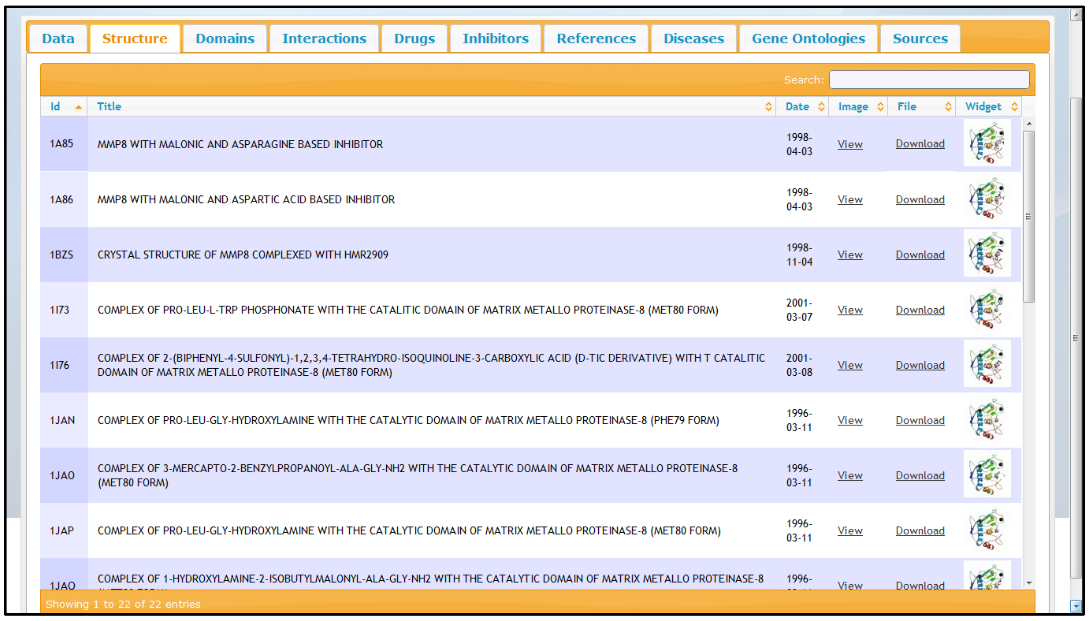
PANTHER CLASSIFICATION

PANTHER Family:	MATRIX METALLOPROTEINASE
PANTHER Subfamily:	INTERSTITIAL COLLAGENASE

Figure 33: Phanter frame

The second tab, *Structure* (Figure 34), covers the related PDBs that are referred by the protein. There is information on the PDB id, title, date and links for the sequence image (using a JavaScript gallery, Figure 35) and a widget where the user can explore its structure (Figure 36).

[32] http://www.pantherdb.org/



Id	Title	Date	Image	File	Widget
1A85	MMP8 WITH MALONIC AND ASPARAGINE BASED INHIBITOR	1998-04-03	View	Download	
1A86	MMP8 WITH MALONIC AND ASPARTIC ACID BASED INHIBITOR	1998-04-03	View	Download	
1BZS	CRYSTAL STRUCTURE OF MMP8 COMPLEXED WITH HMR2909	1998-11-04	View	Download	
1I73	COMPLEX OF PRO-LEU-L-TRP PHOSPHONATE WITH THE CATALITIC DOMAIN OF MATRIX METALLO PROTEINASE-8 (MET80 FORM)	2001-03-07	View	Download	
1I76	COMPLEX OF 2-(BIPHENYL-4-SULFONYL)-1,2,3,4-TETRAHYDRO-ISOQUINOLINE-3-CARBOXYLIC ACID (D-TIC DERIVATIVE) WITH T CATALITIC DOMAIN OF MATRIX METALLO PROTEINASE-8 (MET80 FORM)	2001-03-08	View	Download	
1JAN	COMPLEX OF PRO-LEU-GLY-HYDROXYLAMINE WITH THE CATALYTIC DOMAIN OF MATRIX METALLO PROTEINASE-8 (PHE79 FORM)	1996-03-11	View	Download	
1JAO	COMPLEX OF 3-MERCAPTO-2-BENZYLPROPANOYL-ALA-GLY-NH2 WITH THE CATALYTIC DOMAIN OF MATRIX METALLO PROTEINASE-8 (MET80 FORM)	1996-03-11	View	Download	
1JAP	COMPLEX OF PRO-LEU-GLY-HYDROXYLAMINE WITH THE CATALYTIC DOMAIN OF MATRIX METALLO PROTEINASE-8 (MET80 FORM)	1996-03-11	View	Download	
1JAQ	COMPLEX OF 1-HYDROXYLAMINE-2-ISOBUTYLMALONYL-ALA-GLY-NH2 WITH THE CATALYTIC DOMAIN OF MATRIX METALLO PROTEINASE-8	1996-	View	Download	

Showing 1 to 22 of 22 entries

Figure 34: List of PDBs related to the protein

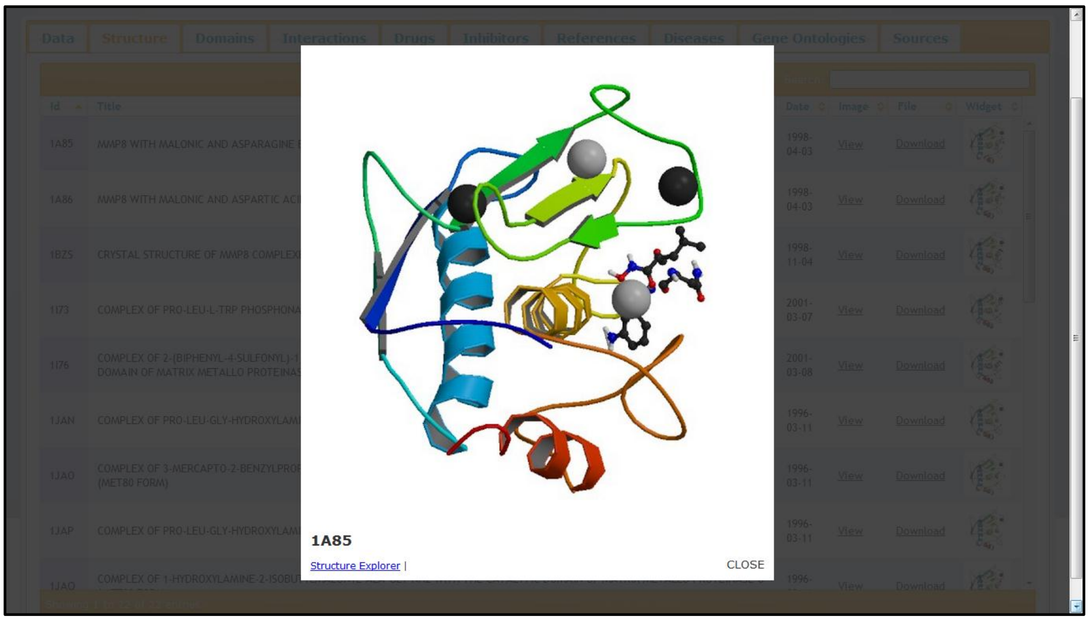
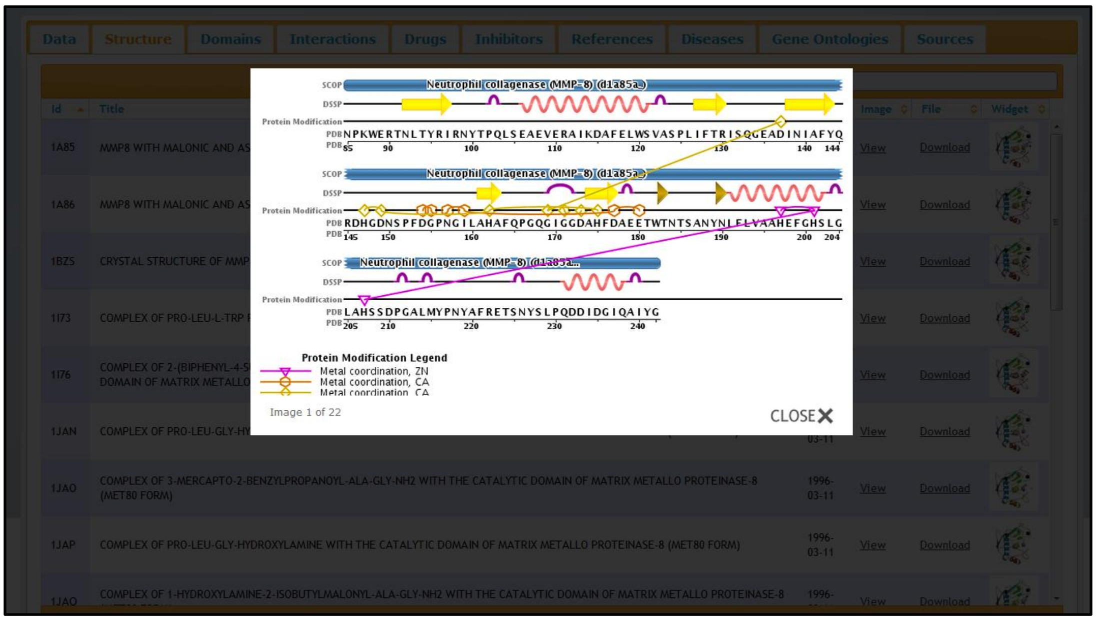


Figure 35: PDB's sequence image

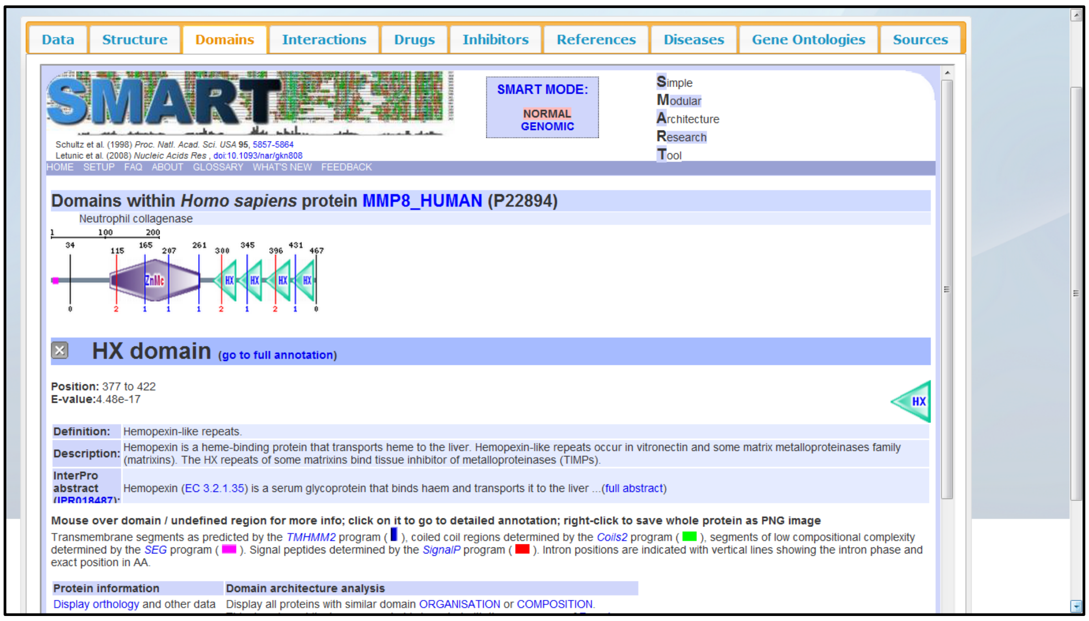


Data
Structure
Domains
Interactions
Drugs
Inhibitors
References
Diseases
Gene Ontologies
Sources
SCOP
DSSP
Protein Modification
PDB
Neutrophil collagenase (MMP-8) (d1a85a_)
NPKWERTNLTYRIRNYTPQLSEAEVERAIKDAFELWSVASPLIFTRISQGEADINIAFYQ
RDHGDNSPFDGPNGILAHAFQPGQGIGGDAHFDAEETWTNTSANYNLFLVAAHEFGHSLG
LAHSSDPGALMYPNYAFRETSNYSLPQDDIDGIQAIYG
Protein Modification Legend
Metal coordination, ZN
Metal coordination, CA
Metal coordination, CA
Image 1 of 22
CLOSE

Id	Title		Image	File	Widget
1A85	MMP8 WITH MALONIC AND AS		View	Download	
1A86	MMP8 WITH MALONIC AND AS		View	Download	
1BZS	CRYSTAL STRUCTURE OF MMP		View	Download	
1I73	COMPLEX OF PRO-LEU-L-TRP		View	Download	
1I76	COMPLEX OF 2-(BIPHENYL-4-S DOMAIN OF MATRIX METALLO		View	Download	
1JAN	COMPLEX OF PRO-LEU-GLY-HY	03-11	View	Download	
1JAO	COMPLEX OF 3-MERCAPTO-2-BENZYLPROPANOYL-ALA-GLY-NH2 WITH THE CATALYTIC DOMAIN OF MATRIX METALLO PROTEINASE-8 (MET80 FORM)	1996-03-11	View	Download	
1JAP	COMPLEX OF PRO-LEU-GLY-HYDROXYLAMINE WITH THE CATALYTIC DOMAIN OF MATRIX METALLO PROTEINASE-8 (MET80 FORM)	1996-03-11	View	Download	
1JAQ	COMPLEX OF 1-HYDROXYLAMINE-2-ISOBUTYLMALONYL-ALA-GLY-NH2 WITH THE CATALYTIC DOMAIN OF MATRIX METALLO PROTEINASE-8	1996-	View	Download	

Figure 36: PDB Structure and link for its explorer

For the *Domains* tab, we framed the SMART tool (Figure 37). It features a JavaScript widget where the researcher can explore different domains within the protein, by just dragging the mouse over it (in the *P22894* protein case, the HX domain and the ZnMc domain).



Schultz et al. (1998) *Proc. Natl. Acad. Sci. USA* **95**, 5857-5864
Letunic et al. (2008) *Nucleic Acids Res* , doi:10.1093/nar/gkn808

Domains within *Homo sapiens* protein MMP8_HUMAN (P22894)

Neutrophil collagenase

HX domain (go to full annotation)

Position: 377 to 422
E-value:4.48e-17

Definition:	Hemopexin-like repeats.
Description:	Hemopexin is a heme-binding protein that transports heme to the liver. Hemopexin-like repeats occur in vitronectin and some matrix metalloproteinases family (matrixins). The HX repeats of some matrixins bind tissue inhibitor of metalloproteinases (TIMPs).
InterPro abstract (IPR018487):	Hemopexin (EC 3.2.1.35) is a serum glycoprotein that binds haem and transports it to the liver ...(full abstract)

Mouse over domain / undefined region for more info; click on it to go to detailed annotation; right-click to save whole protein as PNG image

Transmembrane segments as predicted by the *TMHMM2* program (▮), coiled coil regions determined by the *Coils2* program (🟩), segments of low compositional complexity determined by the *SEG* program (🟪). Signal peptides determined by the *SignalP* program (🟥). Intron positions are indicated with vertical lines showing the intron phase and exact position in AA.

Protein information	Domain architecture analysis
Display orthology and other data	Display all proteins with similar domain ORGANISATION or COMPOSITION.

Figure 37: The SMART tool showing the current protein

With the *Interactions* tab, the researcher can generate a diagram showing the protein interaction. This is done using the STRING[33] web tool wich requires the input of two parameters: the degree of confidence (low, medium, high or highest), the network flavor (evidence, confidence or actions) and the maximum number of proteins the user want to be shown (Figure 38).

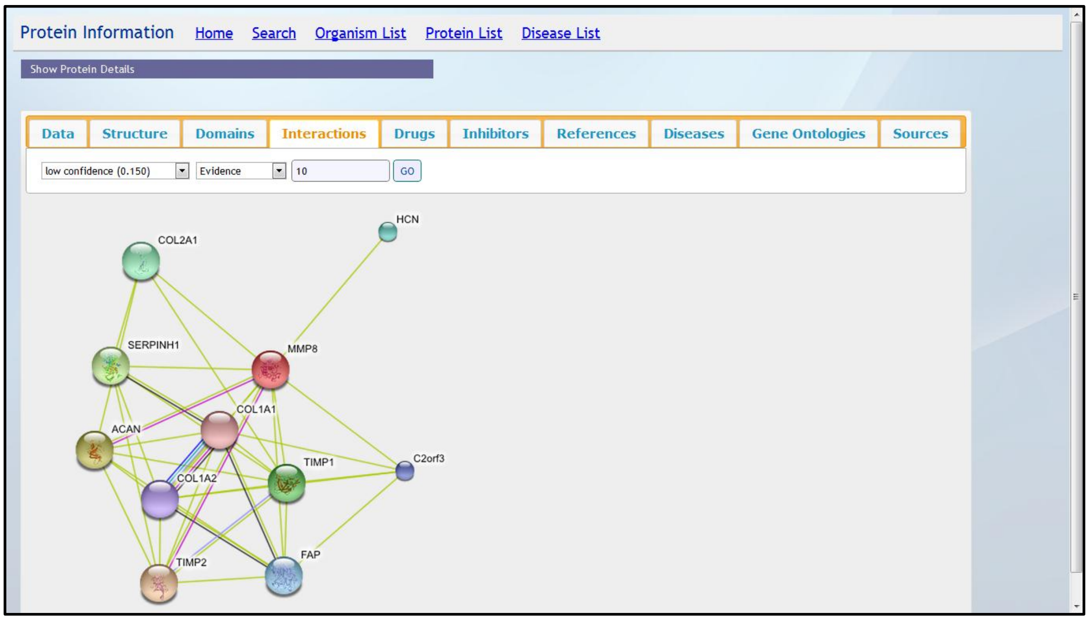
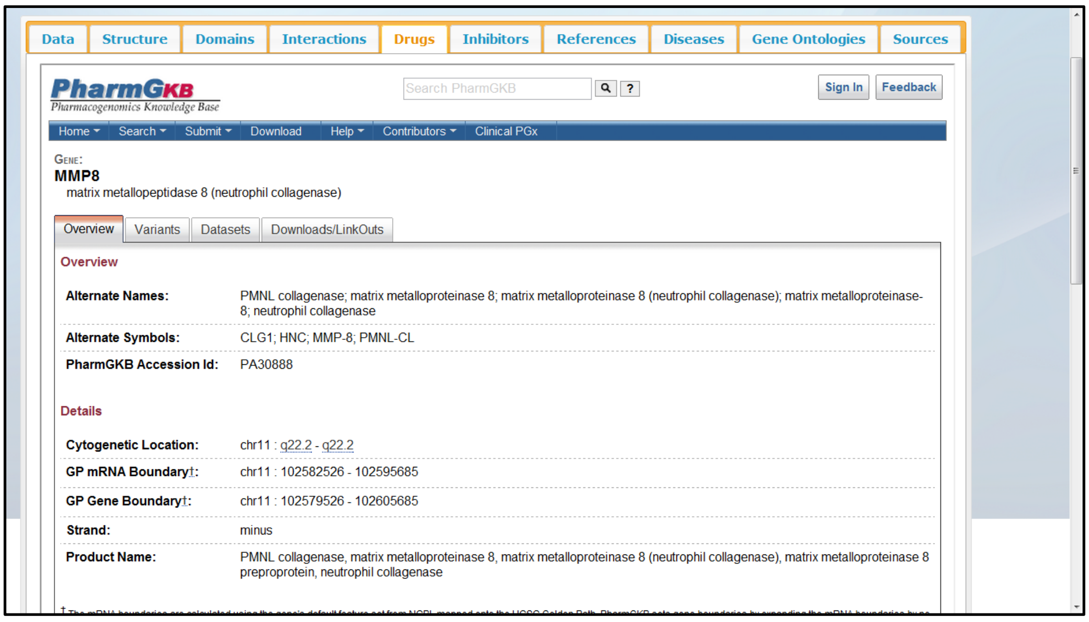


Figure 38: The STRING protein interaction tool

In the *Drugs* tab, we have used the PharmGKB[34]. In the particular case where the protein does not have any associated drug, a description of its gene is presented. Fr instance, the protein *P22894* is not related to any drug, thus a description of the MMP8 gene is presented (Figure 39).

[33] http://string-db.org/
[34] http://www.pharmgkb.org/



GENE:
MMP8
matrix metallopeptidase 8 (neutrophil collagenase)

Overview

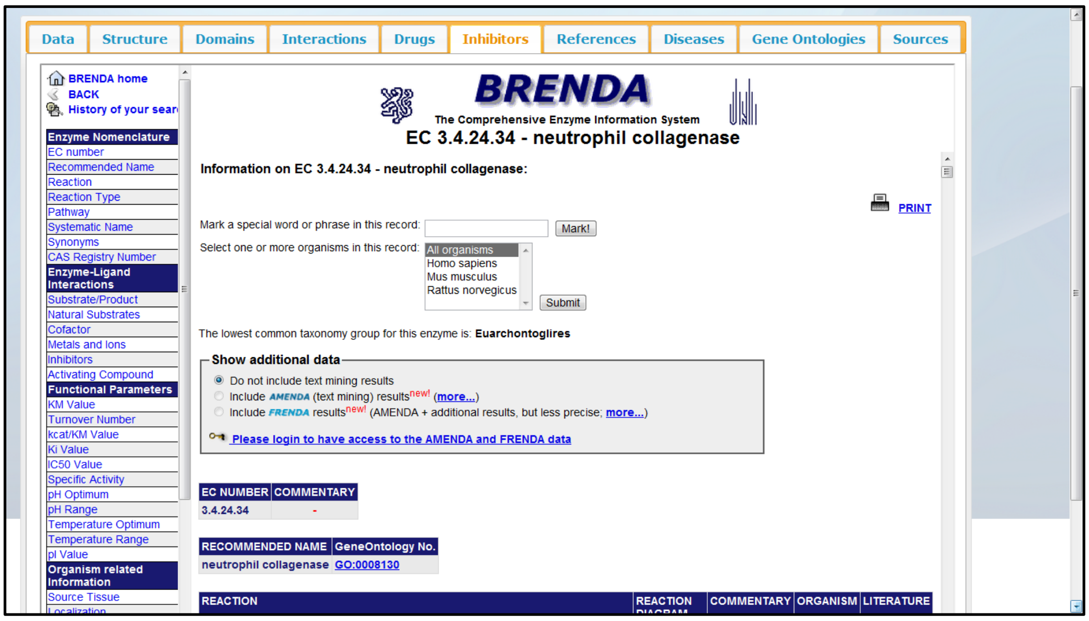
Alternate Names:	PMNL collagenase; matrix metalloproteinase 8; matrix metalloproteinase 8 (neutrophil collagenase); matrix metalloproteinase-8; neutrophil collagenase
Alternate Symbols:	CLG1; HNC; MMP-8; PMNL-CL
PharmGKB Accession Id:	PA30888

Details

Cytogenetic Location:	chr11 : q22.2 - q22.2
GP mRNA Boundary†:	chr11 : 102582526 - 102595685
GP Gene Boundary†:	chr11 : 102579526 - 102605685
Strand:	minus
Product Name:	PMNL collagenase, matrix metalloproteinase 8, matrix metalloproteinase 8 (neutrophil collagenase), matrix metalloproteinase 8 preproprotein, neutrophil collagenase

Figure 39: The PharmGKB frame

Using the *Inhibitors* tab the OralCard application presents a frame where it links to BRENDA[35], An information system that represents one of the most complete enzyme repositories. Enzymes are classified according to the Enzyme Commission (EC) list. We use the protein's EC number and present the BRENDA tools to show additional data. In the given example (Figure 40) we present the frame related to the *P22894* protein's EC (3.4.24.34).



EC 3.4.24.34 - neutrophil collagenase

Information on EC 3.4.24.34 - neutrophil collagenase:

Mark a special word or phrase in this record:

Select one or more organisms in this record: All organisms, Homo sapiens, Mus musculus, Rattus norvegicus

The lowest common taxonomy group for this enzyme is: **Euarchontoglires**

Show additional data

- [✓ Do not include text mining results]
- Include AMENDA (text mining) results new! (more...)
- Include FRENDA results new! (AMENDA + additional results, but less precise; more...)

Please login to have access to the AMENDA and FRENDA data

EC NUMBER	COMMENTARY
3.4.24.34	-

RECOMMENDED NAME	GeneOntology No.
neutrophil collagenase	GO:0008130

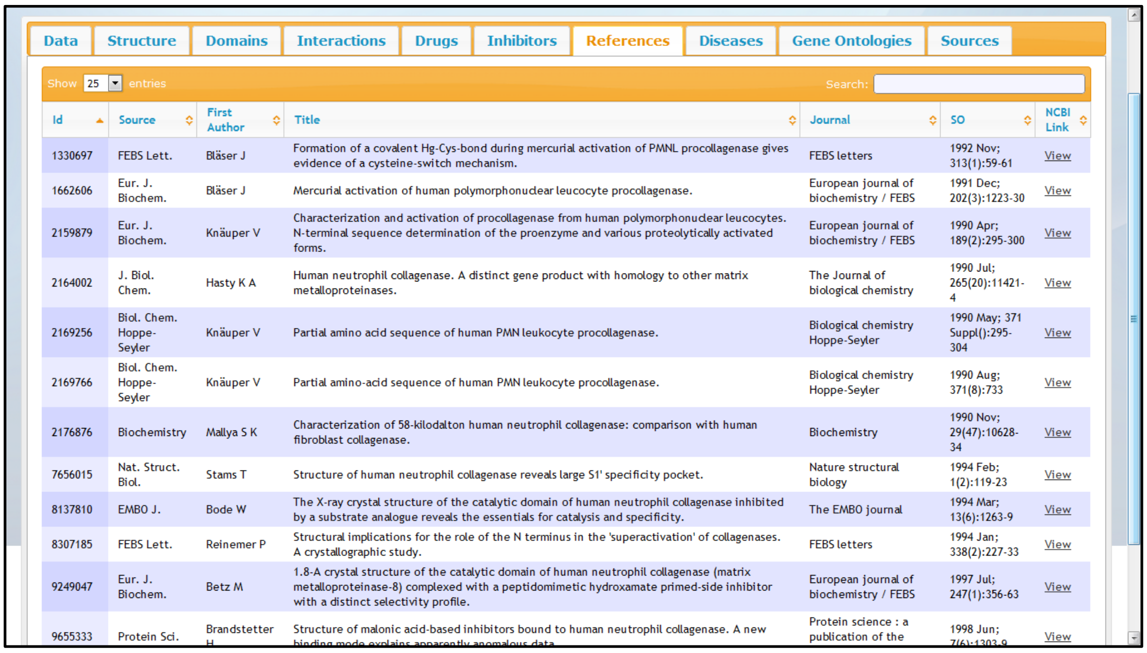
REACTION	REACTION DIAGRAM	COMMENTARY	ORGANISM	LITERATURE

Figure 40: The BRENDA frame

[35] http://www.brenda-enzymes.info/

Concerning to the *References* tab, we rely on the PubMed[36] database to present the reference details for the selected protein. For a first approach, we resorted to the NCBI SOAP web service for Java in order to retrieve the journal articles citations related to the selected proteins. Thus, because of memory issues, the destination server for the OralCard application could not manage to use this web service. For this reason, we recurred to NCBI web service EFetch for the PubMed database, where a XML is retrieved and parsed.

The OralCard web application presents the user a JavaScript table (Figure 41) with details concerning to the referred citation (citation id, source, last author, title, journal name, publish date, and a link to the PubMed citation where more details are available).



Id	Source	First Author	Title	Journal	SO	NCBI Link
1330697	FEBS Lett.	Bläser J	Formation of a covalent Hg-Cys-bond during mercurial activation of PMNL procollagenase gives evidence of a cysteine-switch mechanism.	FEBS letters	1992 Nov; 313(1):59-61	View
1662606	Eur. J. Biochem.	Bläser J	Mercurial activation of human polymorphonuclear leucocyte procollagenase.	European journal of biochemistry / FEBS	1991 Dec; 202(3):1223-30	View
2159879	Eur. J. Biochem.	Knäuper V	Characterization and activation of procollagenase from human polymorphonuclear leucocytes. N-terminal sequence determination of the proenzyme and various proteolytically activated forms.	European journal of biochemistry / FEBS	1990 Apr; 189(2):295-300	View
2164002	J. Biol. Chem.	Hasty K A	Human neutrophil collagenase. A distinct gene product with homology to other matrix metalloproteinases.	The Journal of biological chemistry	1990 Jul; 265(20):11421-4	View
2169256	Biol. Chem. Hoppe-Seyler	Knäuper V	Partial amino acid sequence of human PMN leukocyte procollagenase.	Biological chemistry Hoppe-Seyler	1990 May; 371 Suppl():295-304	View
2169766	Biol. Chem. Hoppe-Seyler	Knäuper V	Partial amino-acid sequence of human PMN leukocyte procollagenase.	Biological chemistry Hoppe-Seyler	1990 Aug; 371(8):733	View
2176876	Biochemistry	Mallya S K	Characterization of 58-kilodalton human neutrophil collagenase: comparison with human fibroblast collagenase.	Biochemistry	1990 Nov; 29(47):10628-34	View
7656015	Nat. Struct. Biol.	Stams T	Structure of human neutrophil collagenase reveals large S1' specificity pocket.	Nature structural biology	1994 Feb; 1(2):119-23	View
8137810	EMBO J.	Bode W	The X-ray crystal structure of the catalytic domain of human neutrophil collagenase inhibited by a substrate analogue reveals the essentials for catalysis and specificity.	The EMBO journal	1994 Mar; 13(6):1263-9	View
8307185	FEBS Lett.	Reinemer P	Structural implications for the role of the N terminus in the 'superactivation' of collagenases. A crystallographic study.	FEBS letters	1994 Jan; 338(2):227-33	View
9249047	Eur. J. Biochem.	Betz M	1.8-A crystal structure of the catalytic domain of human neutrophil collagenase (matrix metalloproteinase-8) complexed with a peptidomimetic hydroxamate primed-side inhibitor with a distinct selectivity profile.	European journal of biochemistry / FEBS	1997 Jul; 247(1):356-63	View
9655333	Protein Sci.	Brandstetter H	Structure of malonic acid-based inhibitors bound to human neutrophil collagenase. A new binding mode explains apparently anomalous data	Protein science : a publication of the	1998 Jun; 7(6):1303-9	View

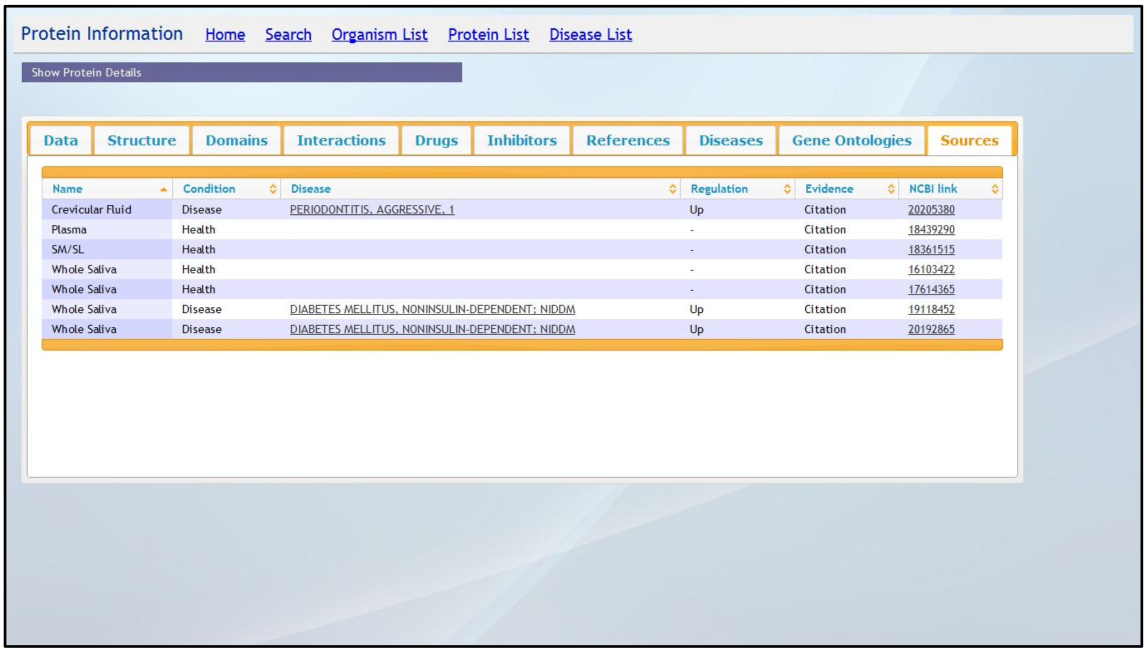
Figure 41: The References tab

The last two tabs refer to diseases and gene ontologies that are related to the selected protein in the OralCard database. The application presents a link to the respective entity where the user can search for further data.

The *Sources* tab presents information contained in the XLSX file (Appendix A-1): source name, condition (health or disease), a link to the disease page, regulation (up

[36] http://www.ncbi.nlm.nih.gov/pubmed/

or down), evidence type, and finally a link to the PubMed reference that supports the relation (Figure 42).

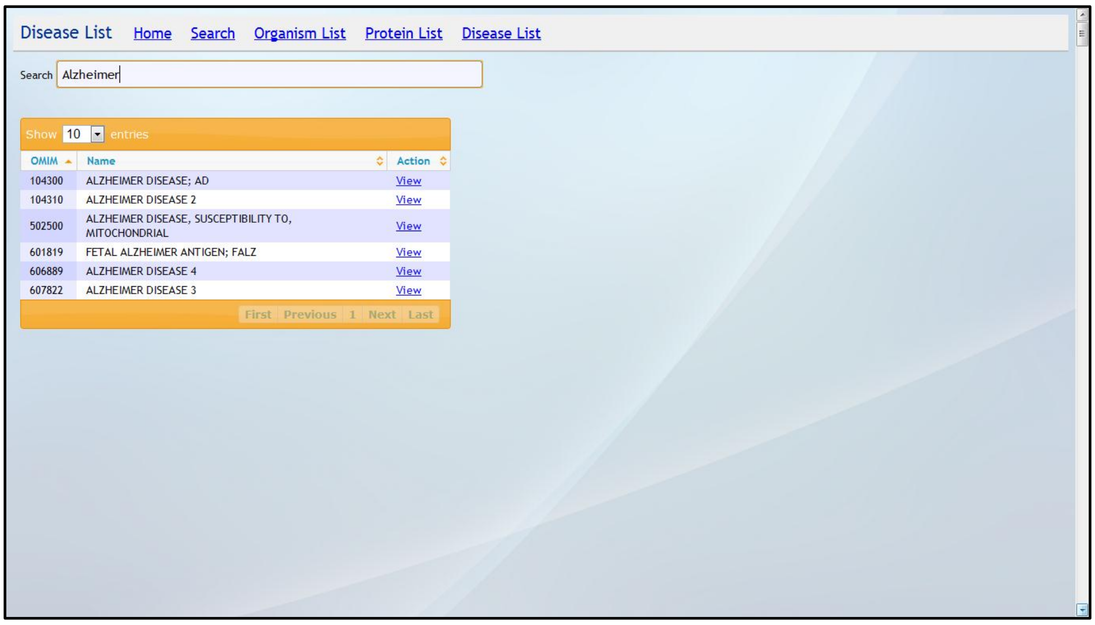


Name	Condition	Disease	Regulation	Evidence	NCBI link
Crevicular Fluid	Disease	PERIODONTITIS, AGGRESSIVE, 1	Up	Citation	20205380
Plasma	Health		-	Citation	18439290
SM/SL	Health		-	Citation	18361515
Whole Saliva	Health		-	Citation	16103422
Whole Saliva	Health		-	Citation	17614365
Whole Saliva	Disease	DIABETES MELLITUS, NONINSULIN-DEPENDENT; NIDDM	Up	Citation	19118452
Whole Saliva	Disease	DIABETES MELLITUS, NONINSULIN-DEPENDENT; NIDDM	Up	Citation	20192865

Figure 42: The Source relations tab

5.2.3 Disease Search and Details

The disease dedicated search page, as the protein search, presents an input box where the user can make a query. This query can be an OMIM number or a disease’s name. The following example represents a search on the term *Alzheimer* (Figure 43).



Disease List Home Search Organism List Protein List Disease List

Search Alzheimer

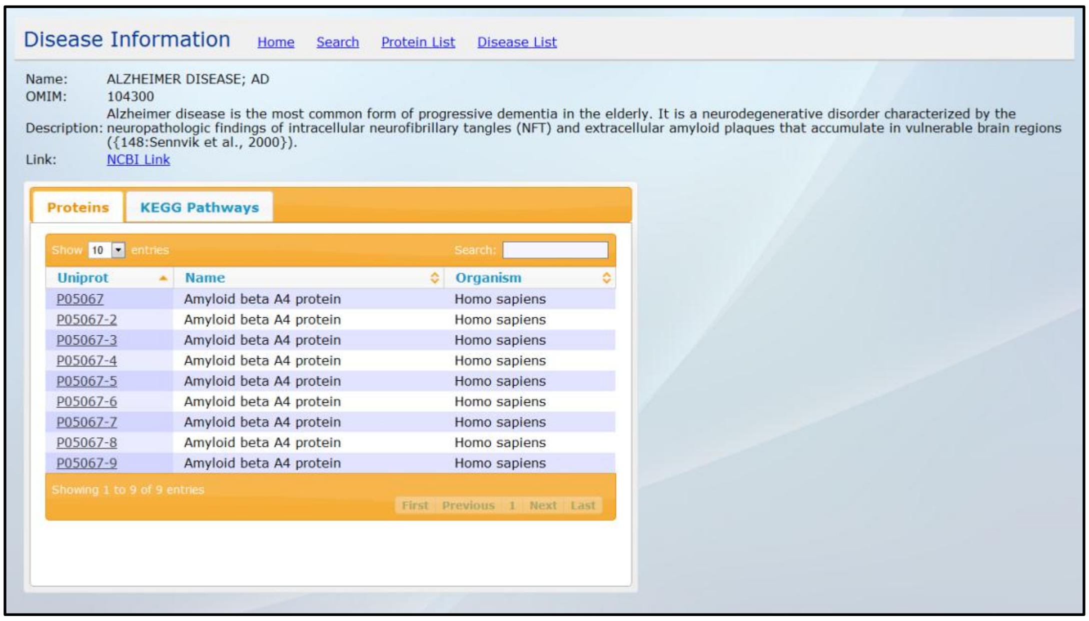
Show 10 entries

OMIM	Name	Action
104300	ALZHEIMER DISEASE; AD	View
104310	ALZHEIMER DISEASE 2	View
502500	ALZHEIMER DISEASE, SUSCEPTIBILITY TO, MITOCHONDRIAL	View
601819	FETAL ALZHEIMER ANTIGEN; FALZ	View
606889	ALZHEIMER DISEASE 4	View
607822	ALZHEIMER DISEASE 3	View

First Previous 1 Next Last

Figure 43: The disease search web page

For each disease, a short description and a detailed table are shown (Figure 44). We have also integrated a tab in order to consult the KEGG pathways related to the disease. Here the user can go directly to a dedicated website or consult an illustration of the selected pathway (Figure 45).



Disease Information Home Search Protein List Disease List

Name:	ALZHEIMER DISEASE; AD
OMIM:	104300
Description:	Alzheimer disease is the most common form of progressive dementia in the elderly. It is a neurodegenerative disorder characterized by the neuropathologic findings of intracellular neurofibrillary tangles (NFT) and extracellular amyloid plaques that accumulate in vulnerable brain regions ({148:Sennvik et al., 2000}).
Link:	NCBI Link

Proteins KEGG Pathways

Show 10 entries Search:

Uniprot	Name	Organism
P05067	Amyloid beta A4 protein	Homo sapiens
P05067-2	Amyloid beta A4 protein	Homo sapiens
P05067-3	Amyloid beta A4 protein	Homo sapiens
P05067-4	Amyloid beta A4 protein	Homo sapiens
P05067-5	Amyloid beta A4 protein	Homo sapiens
P05067-6	Amyloid beta A4 protein	Homo sapiens
P05067-7	Amyloid beta A4 protein	Homo sapiens
P05067-8	Amyloid beta A4 protein	Homo sapiens
P05067-9	Amyloid beta A4 protein	Homo sapiens

Showing 1 to 9 of 9 entries

First Previous 1 Next Last

Figure 44: The disease details web page with related proteins

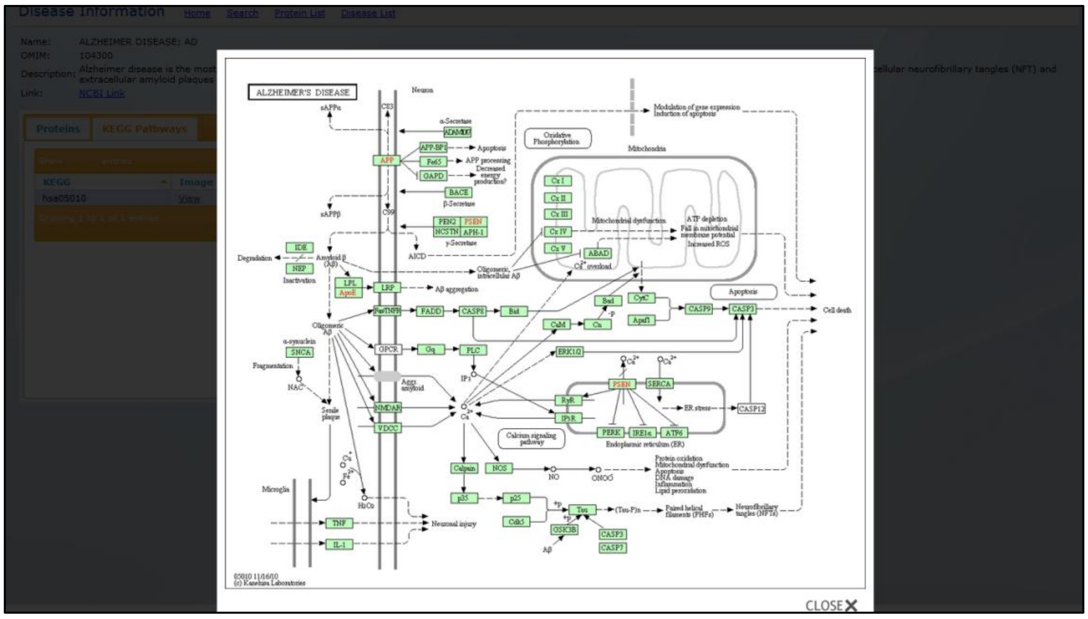


Figure 45: A KEGG pathway related to the selected disease

5.2.1 Direct access using URL Bindings

One of the requirements for the OralCard web application was to have a direct access to a specific page for each entity. Using this direct approach, a user that already knows the accession code for a protein, disease or gene ontology, can input the respective entity address in order to consult its details.

We used the Stripes feature to achieve this goal. It is designated as URL bindings, and allows the developer to customize the application's URL fashion. Using the *@UrlBinding* annotation in our Java code, and some tweaks in the *web.xml* file, we can customize the application's URL for each action.

In the OralCard web application a user can have direct access to a protein details page using the following link:

```
http://bioinformatics.ua.pt/OralCard/proteins/view/P22894
```

The *P22894* refers to the desired UniProtKBAC, and can be any of those contained in the application's database.

The same situation can be applied to a disease. The user can have direct access to a disease details page using the following link:

```
http://bioinformatics.ua.pt/OralCard/diseases/view/104300
```

The *104300* number refers to the desired OMIM, and it can be any of those contained in the application’s database.

5.3 Summary

In this section we presented the results achieved for the OralCard project. The first part of the assignment was intended to import information from several external services. We realized that an approach using Java threads would reduce significantly the time consumption in this task. This difference of time was due to the delay of used web services, such as UniProt or the Entrez Utilities.

The second part of the assignment was to develop a customized frontend reflecting the results presented in the database. We have shown some screenshots of the application (home page, autocomplete feature, customized views for proteins and diseases).

Finally we described the customized links for each entity. This feature was implemented recurring to a Stripes tool, denominated *URL Bindings*. This allows a rapid and direct access to a previously selected protein or disease.

In the next chapter we will present general conclusions regarding the OralCard project.

6. Conclusions and Future Work

The presented OralCard web application fulfils the proposed objectives of enabling non database experts to benefit from the database’s storage and retrieval power and scalable data management. The OralCard system provides a way for the non-expert users to access data from proteins, diseases and gene ontologies, in a transparent and easy way. It was built to aid the user to find in the least possible amount of time, the looked-for entity, providing several important data about it, as of different views retrieved from several major external databases.

In terms of implementation options, we decided to take advantage of the on-going project Molgenis, a local microarray database. It was designed for *DRY* software engineers and bioinformaticians to take better decisions regarding genomics experiment information management. Along with the Molgenis database API, we designed tools for data retrieval using well-known web services, such as Uniprot and Entrez Utilities, from NCBI.

Web services from biological dedicated services have a major importance for bioinformaticians, since they make easy the retrieval of information collected in several warehouses. Due to its availability and its importance, they are highly requested, and can fail without warning. For this reason, it is of major importance to develop methods to backup these situations of denial of service, or simple its absence.

Web services also have delays between requests and responses. For this reason, we concentrated on reducing these *idle* situations by creating Java threads for groups of proteins. This will improve the throughput of the imported proteins reducing considerably the data fetch time.

In the future, it would be important to develop a GUI to use the functionalities of importing proteins information. Since this process takes a couple of hours to be concluded, a GUI with controls would provide pause and resume functionalities. Also, it would have a major importance the system to automatically update the OralCard database, checking for updates within the external biological services.

Molgenis generates automatically a simple web user interface, which is of major importance while downloading data, since the developer can check at any time the correctness of this information.

In the OralCard fronted implementation, we decided to use Stripes to develop the business tier and the client tier. Through a dedicated extension, we were able to create a persistence layer without much effort, and we did not have to repeat the common tasks to every entity that resides in our database.

The usage of object-relational mapping solutions based on Hibernate and Stripersist (using a MySQL backend database), demonstrated to be a good choice for a swift and effective development of the data access application layer. Hibernate Annotations allow a very organized and easy way to bind the domain model classes with the respective tables in the relational database. Stripersist includes easy to use tools to establish the relationship between the domain model classes and the frontend DAO classes.

In terms of web interface development choices, Stripes also includes reusable methods to implement a MVC pattern in our web application. It uses action beans that can be used through JSP interactions, making easy the most common Java web development tasks for the frontend.

The decision to use Stripes, combined with jQuery, has proved to be a good choice for web development, since it allowed intuitive development of an advanced web based portal. Stripes does not have good documentation has other well-known frameworks, such as GWT, making it hard to integrate with other frameworks functionalities, such as the auto completion with jQuery. Nevertheless, since the basics are assimilated it is very natural to use.

jQuery is a framework that was useful to model some user interface details, but it was most used while using the autocomplete feature and for showing images using a gallery. Since it is a widely used JavaScript framework, it was very easy to find documentation to support some implementations.

In order to verify if the developed work was in the route of achieving the proposed objectives, we demonstrated our results to the Portuguese Catholic University (Viseu) biomedical researchers. The provided feedback was very satisfactory, and gave us the green flag to carry on with the project and include more functionality to the web application.

7. Appendix

7.1 Appendix A - CD

This CD includes the following files:

- **(1) Oralome – Human Proteins.xlsx:** contains data inserted manually by the researchers (UCP-PV). It includes a list of uniprot codes, the source where the protein was identified, a health or disease context, regulation, age group, and an evidence (NCBI citation);
- **(2) HMP and HOMD list.xlsx:** contains a list of microorganisms which are found in association with both healthy and diseased humans, and species that are present in the human oral cavity (provided by the UCP-PV researchers) ;
- **(3) Gene Ontology Codes.docx:** contains a list of gene ontology codes (provided by the UCP-PV researchers).

8. References

1. Heerschop, E., *Development of a Database Abstraction Layer.* Order. **501**: p. 2783.
2. Fokkema, I.F.A.C., J.T. den Dunnen, and P.E.M. Taschner, *LOVD: Easy creation of a locus specific sequence variation database using an "LSDB in a box" approach.* Human mutation, 2005. **26**(2): p. 63-68.
3. D O'Connor, B., et al., *GMODWeb: a web framework for the Generic Model Organism Database.* Genome Biology, 2008. **9**(6): p. R102.
4. Swertz, M.A., et al., *Molecular Genetics Information System (MOLGENIS): alternatives in developing local experimental genomics databases.* Bioinformatics, 2004. **20**(13): p. 2075.
5. Magnani, M. and D. Montesi, *A Survey on Uncertainty Management in Data Integration.* J. Data and Information Quality, 2010. **2**(1): p. 1-33.
6. Lodhi, F. and M.A. Ghazali, *Design of a simple and effective object-to-relational mapping technique*, in *Proceedings of the 2007 ACM symposium on Applied computing*2007, ACM: Seoul, Korea. p. 1445-1449.
7. Zaninotto, F. and F. Potencier, *The definitive guide to symfony*2007: Apress.
8. Thomas, D., et al., *Agile web development with rails, Second Edition*2007: Pragmatic bookshelf.
9. Hart, A.M., *Hibernate in the classroom.* J. Comput. Small Coll., 2005. **20**(4): p. 98-100.
10. O'Neil, E.J., *Object/relational mapping 2008: hibernate and the entity data model (edm)*, in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*2008, ACM: Vancouver, Canada. p. 1351-1356.
11. Ozkaya, I., R. Kazman, and M. Klein, *Quality-Attribute Based Economic Valuation of Architectural Patterns*, in *Proceedings of the First International Workshop on The Economics of Software and Computation*2007, IEEE Computer Society. p. 5.
12. Hansen, S. and T.V. Fossum, *Refactoring model-view-controller.* J. Comput. Small Coll., 2005. **21**(1): p. 120-129.
13. Sasine, J.M. and R.J. Toal, *Implementing the model-view-controller paradigm in Ada 95*, in *Proceedings of the conference on TRI-Ada '95: Ada's role in global markets: solutions for a changing complex world*1995, ACM: Anaheim, California, United States. p. 202-211.
14. Stoughton, A., *A functional model-view-controller software architecture for command-oriented programs*, in *Proceedings of the ACM SIGPLAN workshop on Generic programming*2008, ACM: Victoria, BC, Canada. p. 1-12.

15. Klopper, R., S. Gruner, and D.G. Kourie, *Assessment of a framework to compare software development methodologies*, in *Proceedings of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*2007, ACM: Port Elizabeth, South Africa. p. 56-65.
16. Coleman, G. and R. Verbruggen, *A quality software process for rapid application development.* Software Quality Journal, 1998. **7**(2): p. 107-122.
17. Agarwal, R., et al., *Risks of rapid application development.* Commun. ACM, 2000. **43**(11es): p. 1.
18. Manfredi, R. and P. Fouche, *Web application framework*, 2005, EP Patent 1,594,049.
19. *Getting Started with Rails - What is Rails?* 23-02-2011]; Available from: http://guides.rubyonrails.org/getting_started.html#what-is-rails.
20. *Agile Manifesto*. 11-03-2011]; Available from: http://agilemanifesto.org/.
21. *Symfony - Open-Source PHP Web Framework*. 23-02-2011]; Available from: http://www.symfony-project.org/.
22. *Symfony - About*. 23-02-2011]; Available from: http://www.symfony-project.org/about.
23. Sarang, P., *Practical Liferay: Java-based Portal Applications Development*2009: Springer.
24. Yuan, J.X., *Liferay Portal Enterprise Intranets*2008: Packt Publishing Limited.
25. Swertz, M., et al., *The MOLGENIS toolkit: rapid prototyping of biosoftware at the push of a button.* BMC Bioinformatics, 2010. **11**(Suppl 12): p. S12.
26. *Stripes Home*. 23-02-2011]; Available from: http://www.stripesframework.org.
27. Daoud, F., *Stripes:... and Java web development is fun again (Pragmatic Programmers)*2008: Pragmatic Bookshelf.
28. Hanson, R. and A. Tacy, *GWT in Action: Easy Ajax with the Google Web Toolkit*2007: Manning Publications Co. Greenwich, CT, USA.
29. Chen, H. and R. Cheng, *ZK: Ajax without JavaScript framework*2007: Springer.
30. Dobecki, M. and W. Zabierowski. *Web-based content management system*. IEEE.
31. Naidu, P.G., M.J. Palakal, and S. Hartanto, *On-the-fly data integration models for biological databases*, in *Proceedings of the 2007 ACM symposium on Applied computing*2007, ACM: Seoul, Korea. p. 118-122.
32. Santos, R.J. and J. Bernardino, *Real-time data warehouse loading methodology*, in *Proceedings of the 2008 international symposium on Database engineering \&\#38; applications*2008, ACM: Coimbra, Portugal. p. 49-58.
33. Chaudhuri, S. and U. Dayal, *An overview of data warehousing and OLAP technology.* SIGMOD Rec., 1997. **26**(1): p. 65-74.
34. Weng, L., et al. *An approach for automatic data virtualization*. in *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on*. 2004.
35. Lans, R.v.d., *Clearly Defining Data Virtualization, Data Federation, and Data Integration.* BeyeNetwork - Powell Media, LLC, 2010.
36. Loshin, D., *Data Integration Alternatives - Managing Value and Quality; Using a Governed Approach to Incorporating Data Quality Services Within the Data Integration Process.* Pitney Bowes - Business Insight, 2010: p. 3.
37. de Almeida, P.V., et al., *Saliva composition and functions: a comprehensive review.* J Contemp Dent Pract, 2008. **9**(3): p. 72-80.

38. Greabu, M., et al., *Saliva—a diagnostic window to the body, both in health and in disease.* J Med Life, 2009. **2**: p. 124-132.
39. Nagler, R.M., *Saliva as a tool for oral cancer diagnosis and prognosis.* Oral oncology, 2009. **45**(12): p. 1006-1010.
40. Shpitzer, T., et al., *Salivary analysis of oral cancer biomarkers.* British journal of cancer, 2009. **101**(7): p. 1194-1198.
41. Rudney, J., R. Staikov, and J. Johnson, *Potential biomarkers of human salivary function: a modified proteomic approach.* Archives of oral biology, 2009. **54**(1): p. 91-100.
42. Gonçalves, L.D.R., et al., *Comparative proteomic analysis of whole saliva from chronic periodontitis patients.* Journal of proteomics, 2010. **73**(7): p. 1334-1341.
43. Streckfus, C.F., et al., *Breast cancer related proteins are present in saliva and are modulated secondary to ductal carcinoma in situ of the breast.* Cancer investigation, 2008. **26**(2): p. 159-167.
44. Hu, S., et al., *Salivary proteomic and genomic biomarkers for primary Sjögren's syndrome.* Arthritis & Rheumatism, 2007. **56**(11): p. 3588-3600.
45. Rao, P.V., et al., *Proteomic identification of salivary biomarkers of type-2 diabetes.* Journal of Proteome Research, 2009. **8**(1): p. 239-245.
46. Livnat, G., et al., *Salivary profile and oxidative stress in children and adolescents with cystic fibrosis.* Journal of Oral Pathology & Medicine, 2010. **39**(1): p. 16-21.
47. Giusti, L., et al., *Specific proteins identified in whole saliva from patients with diffuse systemic sclerosis.* The Journal of Rheumatology, 2007. **34**(10): p. 2063.
48. Seymour, G.J., M.P. Cullinan, and N.C. Heng, *Oral Biology: Molecular Techniques and Applications (Methods in Molecular Biology)*2010: Humana Press.
49. Wong, D.T., *Salivary diagnostics powered by nanotechnologies, proteomics and genomics.* The Journal of the American Dental Association, 2006. **137**(3): p. 313.
50. Helmerhorst, E. and F. Oppenheim, *Saliva: a dynamic proteome.* Journal of dental research, 2007. **86**(8): p. 680.
51. Chen, T., et al., *The Human Oral Microbiome Database: a web accessible resource for investigating oral microbe taxonomic and genomic information.* Database: the journal of biological databases and curation, 2010. **2010**.
52. Nelson, K.E., et al., *A catalog of reference genomes from the human microbiome.* Science (New York, NY), 2010. **328**(5981): p. 994-999.
53. Xie, H., et al., *Proteomics analysis of cells in whole saliva from oral cancer patients via value-added three-dimensional peptide fractionation and tandem mass spectrometry.* Molecular & Cellular Proteomics, 2008. **7**(3): p. 486.
54. Lau, J.Y., *Protein Structure Database for Structural Genomics Group*, 2005, Rutgers, The State University of New Jersey.
55. Bairoch, A., et al., *The universal protein resource (UniProt).* Nucleic acids research, 2005. **33**(suppl 1): p. D154.
56. Boeckmann, B., et al., *The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003.* Nucleic acids research, 2003. **31**(1): p. 365.
57. George, D.G., W.C. Barker, and L.T. Hunt, *The protein identification resource (PIR).* Nucleic acids research, 1986. **14**(1): p. 11.
58. Hubbard, T., et al., *The Ensembl genome database project.* Nucleic acids research, 2002. **30**(1): p. 38.

59. Kersey, P.J., et al., *Technical Brief The International Protein Index: An integrated database for proteomics experiments.* Proteomics, 2004. **4**(1985): p. 1988.
60. Pruitt, K.D. and D.R. Maglott, *RefSeq and LocusLink: NCBI gene-centered resources.* Nucleic acids research, 2001. **29**(1): p. 137.
61. Drysdale, R.A. and M.A. Crosby, *FlyBase: genes and gene models.* Nucleic acids research, 2005. **33**(suppl 1): p. D390.
62. Stein, L., et al., *WormBase: network access to the genome and biology of Caenorhabditis elegans.* Nucleic acids research, 2001. **29**(1): p. 82.
63. Berman, H.M., et al., *The protein data bank.* Nucleic acids research, 2000. **28**(1): p. 235.
64. *Disease Definition*. 25-05-2011]; Available from: http://www.biology-online.org/dictionary/Disease.
65. Hamosh, A., et al., *Online Mendelian Inheritance in Man (OMIM), a knowledgebase of human genes and genetic disorders.* Nucleic acids research, 2005. **33**(suppl 1): p. D514.
66. Ekins, S., et al., *Pathway mapping tools for analysis of high content data.* METHODS IN MOLECULAR BIOLOGY-CLIFTON THEN TOTOWA-, 2006. **356**: p. 319.
67. Kanehisa, M. and S. Goto, *KEGG: Kyoto encyclopedia of genes and genomes.* Nucleic acids research, 2000. **28**(1): p. 27.
68. Ashburner, M., et al., *Gene Ontology: tool for the unification of biology.* Nature genetics, 2000. **25**(1): p. 25.
69. Harris, M., et al., *The Gene Ontology (GO) database and informatics resource.* Nucleic acids research, 2004. **32**(Database issue): p. D258.
70. Harold, E.R., *Processing XML with Java: a guide to SAX, DOM, JDOM, JAXP, and TrAX*2003: Addison-Wesley Professional.