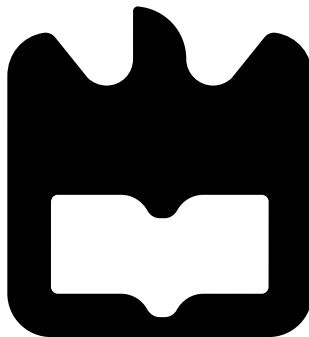




**Miguel Filipe  
Oliveira Dinis**

**Sistema de Informação com Conteúdos Dinâmicos  
Information System with Dynamic Content**







Universidade de Aveiro  
2021

**Miguel Filipe  
Oliveira Dinis**

**Sistema de Informação com Conteúdos Dinâmicos  
Information System with Dynamic Content**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor (Carlos Manuel Azevedo Costa), Professor associado c/ agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.



**o júri / the jury**

presidente / president

Professor Doutor Joaquim Arnaldo Carvalho Martins  
Professor Catedrático, Universidade de Aveiro

vogais / examiners committee

Doutor Eriksson Jorge Melicio Monteiro  
Chefe de Gabinete de Tecnologia, Gabinete de Tecnologia, da Smart Solutions - Cabo Verde  
(Arguente Principal)

Professor Doutor Carlos Manuel Azevedo Costa  
Professor Associado C/ Agregação, Universidade de Aveiro (Orientador)



**agradecimentos /  
acknowledgements**

Agradeço especialmente aos meus orientadores Luís Bastião Silva da BMD Software e ao Professor Doutor Carlos Costa pelo acompanhamento, ajuda e incentivo que sempre me prestaram ao longo deste percurso.

Aos meus pais e irmã pelo suporte e apoio constante durante os últimos 5 anos, bem como aos meus colegas de curso e amigos.

A todos, um muito obrigado.





## Palavras Chave

Sistemas de Informação, Plataformas Low-Code/No-code, Desenvolvimento de Software Agile, Desenvolvimento orientado a modelos, Bases de Dados NoSQL, Esquema de Bases de Dados Dinâmico, Aplicações na Nuvem.

## Resumo

Os sistemas de informação modernos têm requisitos funcionais que podem variar significativamente entre organizações ou contextos regionais, para a mesma área aplicacional. Isto resulta na existência de várias versões de software e respetivos modelos de dados. Os métodos tradicionais de desenvolvimento são pouco flexíveis e eficazes para lidar com a implementação de novos requisitos em tempo útil. Uma simples adição de um elemento de informação num formulário gráfico obriga à alteração do modelo de dados e respetiva ação nas tabelas do sistema, resultando num processo pouco dinâmico de criação de conteúdos que devem ser guardados na base de dados. Com o advento da computação na cloud, têm surgido soluções tecnológicas que permitem desenvolver aplicações com recurso a pouco ou nenhum código. Estas soluções podem inclusive ser operadas por utilizadores finais com conhecimento do domínio aplicacional. Esta dissertação teve como objetivo desenhar e implementar uma plataforma de criação de sistemas de informação com conteúdos dinâmicos. O resultado foi uma solução web multiplataforma de baixo custo que permite um desenvolvimento rápido, intuitivo e dinâmico de conteúdos, através do desenho de modelos a partir de uma interface visual, para utilizadores sem conhecimentos de engenharia de software. Em termos tecnológicos destaca-se o facto de a plataforma integrar soluções open-source robustas que, associada a uma abordagem de metadados, permite uma abstração relativamente à camada de persistência de dados.



**Keywords**

Information Systems, Low-Code/No-code Platforms, Agil Software Development, Model-Driven Development, NoSQL Databases, Dynamic Database Schema, Cloud Application.

**Abstract**

Modern information systems have functional requirements that can vary significantly between organizations or regional contexts for the same application area. This results in the existence of multiple software versions and their respective data models. Traditional development methods are not very flexible and effective in dealing with the implementation of new requirements in a time frame. A simple addition of an information element in a graphical form requires a change in the data model and its action in the system's tables, resulting in an undynamic process of creating content that must be stored in the database. With the advent of cloud computing, technological solutions have emerged that allow the development of applications using little or no code. These solutions can even be operated by end users with knowledge of the application domain. This dissertation aimed to design and implement a platform to create information systems with dynamic content. The result was a low cost multiplatform web solution that allows a fast, intuitive and dynamic content development, through the design of models from a visual interface, for users with no knowledge of software engineering. In technological terms, the platform integrates robust open-source solutions, which, associated with a metadata approach, allows an abstraction regarding the data persistence layer.



# Contents

<b>Contents</b>	<b>1</b>
<b>List of Figures</b>	<b>3</b>
<b>List of Tables</b>	<b>5</b>
<b>Acronyms</b>	<b>6</b>
<b>1 Introduction</b>	<b>9</b>
1.1 Overview . . . . .	9
1.2 Scenario . . . . .	9
1.3 Objectives . . . . .	10
1.4 Dissertation structure . . . . .	11
<b>2 Background</b>	<b>13</b>
2.1 Information Systems . . . . .	13
2.2 Cloud computing . . . . .	14
2.3 Low-code and No-code Platforms . . . . .	15
2.4 Model-Driven Development (MDD) . . . . .	16
2.5 Databases . . . . .	16
2.5.1 Relational Databases . . . . .	16
2.5.2 Non-relational databases . . . . .	18
2.5.3 Search Engines . . . . .	19
2.5.4 Relational and non-relational databases comparison . . . . .	20
2.5.5 NewSQL, Polyglot Persistence and Multi-model databases . . . . .	22
2.6 Related work on the development of dynamic platforms . . . . .	24
2.6.1 Dynamic GUI forms . . . . .	24
2.6.2 Database schema evolution . . . . .	25
<b>3 Low-code development platforms</b>	<b>29</b>
3.1 Open Source solutions . . . . .	29
3.1.1 Convertigo . . . . .	29
3.1.2 Joget . . . . .	30

3.1.3	Budibase . . . . .	32
3.1.4	OpenXava . . . . .	33
3.1.5	Skyve . . . . .	36
3.1.6	Other open source solutions . . . . .	38
3.2	Proprietary solutions overview . . . . .	40
3.3	Results and considerations . . . . .	41
<b>4</b>	<b>DynamicIS Proposal</b>	<b>45</b>
4.1	System Requirements . . . . .	45
4.1.1	Non-functional Requirements . . . . .	45
4.1.2	Functional Requirements . . . . .	46
4.2	System Architecture . . . . .	50
4.3	System Technologies . . . . .	53
4.3.1	Server-side . . . . .	55
4.3.2	Client-side . . . . .	56
4.4	Skyve Integration . . . . .	58
4.4.1	DynamicIS REST API . . . . .	58
4.4.2	Metadata flow . . . . .	60
4.5	Persistence . . . . .	63
4.5.1	Couchbase integration . . . . .	64
4.5.2	Database model . . . . .	66
4.6	System Implementation . . . . .	70
4.6.1	Authentication . . . . .	71
4.6.2	Front-end capabilities . . . . .	71
4.6.3	Form and Result page building . . . . .	73
4.6.4	Advanced Form customization . . . . .	74
4.6.5	Project generation . . . . .	76
<b>5</b>	<b>Results</b>	<b>81</b>
5.1	General aspects and Authentication . . . . .	81
5.2	Projects . . . . .	81
5.3	Project page . . . . .	83
5.4	Module page . . . . .	83
5.5	Form page . . . . .	84
5.6	Result page . . . . .	89
5.7	Test and validation . . . . .	91
<b>6</b>	<b>Conclusion</b>	<b>95</b>
6.1	Final considerations . . . . .	95
6.2	Future Work . . . . .	96
	<b>References</b>	<b>97</b>

# List of Figures

2.1	Cloud computing service model, text adapted from [7]	14
2.2	Layered architecture of Low-code Development Platform (LCDP) based on [13]	17
2.3	Example of Polyglot persistence in an E-commerce platform	23
2.4	High-level Abstraction of the Form based dynamic database concept, based on [35]	26
3.1	Convertigo Studio IDE Interface	31
3.2	Initial page of Joget web platform	31
3.3	Joget drag and drop form builder interface	32
3.4	SELECT query run in a MySQL bash showing the table automatically created from the form defined by the developer in the UI.	32
3.5	View of the form page running on the web application created with Joget	33
3.6	Budibase front-end design page	34
3.7	Example of two entities created from Java classes	35
3.8	Automatically generated UI for Medic and Patient entities	35
3.9	Patients list UI	36
3.10	XML document specifying the entity "Nurse" in Skyve metadata	37
3.11	XML document specifying the "MedicStaff" module in Skyve metadata	38
3.12	UI form for the "Nurse" entity and lateral navigation menu generated by Skyve	39
3.13	UI view containing all created Nurse entities	39
4.1	DynamicIS platform use case diagram	47
4.2	DynamicIS Web App architecture	51
4.3	DynamicIS web app project structure	52
4.4	HTTP routing excerpt screenshot present in "routes" file	55
4.5	UML activity diagram showing the process of defining a model (Form)	62

4.6	UML activity diagram showing the process of converting DynamicIS metadata to Skyve understandable metadata in project generation . . . . .	63
4.7	Document-oriented data model of DynamicIS based on [43] proposal . . . . .	67
4.8	ID and iframe present in the settings of a Google calendar . .	75
4.9	Project generation components . . . . .	77
5.1	Authentication page . . . . .	82
5.2	List of projects . . . . .	82
5.3	New project page . . . . .	83
5.4	Project page . . . . .	84
5.5	Module page . . . . .	85
5.6	Form page . . . . .	86
5.7	Field of type Association Aggregation in "Appointment" form	86
5.8	Form field description page . . . . .	87
5.9	View for advanced form customization . . . . .	87
5.10	View Container dialogue . . . . .	88
5.11	Select component dialogue . . . . .	89
5.12	Google calendar component page . . . . .	89
5.13	HTML component page . . . . .	90
5.14	Redirect action component page . . . . .	90
5.15	Result page configuration with default query . . . . .	91
5.16	Result page for the "Medic" entity with a configured query . .	91
5.17	Generate project page . . . . .	92
5.18	Side navigation menu containing the structure of the IS . . .	93
5.19	Result page generated for the IS . . . . .	93
5.20	Advanced view created for "Appointment" form in the IS . . .	94
5.21	Example of generated form with different field arrangements.	94



# List of Tables

3.1	Open source LCDP core features comparison. . . . .	42
4.1	DynamicIS REST API endpoints . . . . .	60

# Acronyms

- ACID** Atomicity, Consistency, Isolation, Durability. 18, 19, 21
- API** Application Programming Interface. 34, 40, 54, 58–62, 64, 65, 75
- AWS** Amazon Web Services. 14
- BASE** Basically Available, Soft state, Eventually consistent. 19
- CAP** Consistency, Availability, and Partition tolerant. 19
- CDN** Content Delivery Network. 71
- CRUD** Create, Read, Update and Delete. 35, 36
- DOM** Document Object Model. 62
- DSL** Domain Specific Language. 16, 39
- EIS** Enterprise Information Systems. 13
- GUI** Graphical user interface. 1, 24, 25
- HTML** HyperText Markup Language. 61
- HTTP** Hypertext Transfer Protocol. 52, 57, 60
- IaaS** Infrastructure-as-a-Service. 14
- IDE** Integrated Development Environment. 3, 27, 31, 37–39, 76
- IS** Information System. 4, 13, 31, 69, 76, 84, 93, 94
- JDBC** Java Database Connectivity. 64
- JSON** JavaScript Object Notation. 10, 19, 55, 65, 66, 75
- LCDP** Low-code Development Platform. 3, 5, 15, 17, 25, 42

**MDD** Model-Driven Development. 1, 16

**PaaS** Platform-as-a-Service. 14, 15, 71

**PWA** Progressive Web App. 30

**RDBMS** Relational Database Management System. 10, 17, 18, 21, 23

**REST** REpresentational State Transfer. 33, 36, 40

**SaaS** Software-as-a-Service. 14, 53, 71, 78

**SDK** Software Development Kit. 56, 64

**SDLC** Systems Development Life Cycle. 25

**SQL** Structured Query Language. 15, 17, 18, 22, 30, 33, 36, 39

**UI** User Interface. 25, 30, 32, 34, 37, 39, 43, 46

**WAR** Web Application Resource. 38

**XML** Extensible Markup Language. 3, 34, 36–38, 62, 78



# Chapter 1

## Introduction

### 1.1 Overview

In an increasingly digital world, with growing amounts of information being produced, there follows a tendency to create information systems needed for the most varied areas of application. Business firms and other organizations rely on modern information systems to carry out and manage their operations.

The adoption and evolution to web and cloud technologies requires most of the time a migration that needs to be fast and cost-effective. Not only for large companies that are built all around information systems but also for small organizations or even individuals who want to make their business digital. However, for its small size is not justified to spend many resources and hire development services to create their information systems.

In addition, modern information systems have functional requirements that can vary within and outside their application area, usually resulting in multiple application versions due to the constant changes that need to be made. The fact that most solutions are developed depending on the context of the problem makes their extensibility and adjustment more complex.

### 1.2 Scenario

A traditional information system, such as an enterprise system, results from a design process where the functional requirements of the domain are captured, and the implementation is based on highly standardized relational database systems with rigid data models. If there is a need to change a simple element of information in a front-end form, it may also be necessary to build some components or rebuild the entire application, resulting in a not very dynamic process of creating contents that must be stored in the database.

More recently technologies have emerged to overcome the inflexibility of relational databases, referred to as NoSQL. These technologies can have differentiated application purposes supporting multi-paradigms, from logical and physical data models based on structured data, semi-structured data and documents. The most common technologies support document-oriented data modeling (e.g. MongoDB), key/value (e.g. Redis), columnar (e.g. Cassandra) and graph (e.g. Neo4j).

The real world applications require the data integrity offered by a Relational Database Management System (RDBMS) in addition to the benefits offered by NoSQL databases. So, in the context of information systems, NoSQL technologies can be used in parallel with the relational model to serve specific purposes (Polyglot persistence). For example, storing dynamic content in a JavaScript Object Notation (JSON) structure that is schemaless and static content in a well-defined data model in a relational database.

It is important to note that the search time using dynamic selection attributes is penalized, unless the creation of complementary structures (i.e. indexes), and even that requires a degree of customization appropriate to the model applied.

In an increasingly competitive and highly demand market, such as software development, it is becoming more and more necessary to use tools that help and promote rapid and flexible development which, in turn, bring some challenges, already present in the literature for some time [1]. For this, Low-code and No-code development platforms are increasingly being used due to the ease and speed of learning and also the ability to adapt to different contexts. Moreover, Gartner predicts low-code application platforms will be used for 65% of all application development activity by 2024 [2]

### 1.3 Objectives

This research and dissertation proposes a platform that allows the creation of an information system with dynamic content. It aims to be efficient in terms of attribute search times, scalable and with the possibility of integration in different scenarios. The proposed solution should promote a No-Code web development environment, highlighting important features such as fast, dynamic, and user-friendly development for individuals without technical knowledge of software engineering.

## 1.4 Dissertation structure

This dissertation is organized into six chapters:

- The first chapter presents the introduction, briefly addressing the current context regarding the development of information systems as well as the intended objectives.
- The second chapter presents the background information as well as core concepts for the development of software that provides dynamism and flexibility in the creation of information systems.
- The third chapter presents an analysis of state of the art similar platforms that are integrated in the ambit of a rapid, dynamic and flexible development as well as comparisons between them.
- The fourth chapter presents the DynamicIS proposal, with its system requirements, architecture and implementation, along with details of its components.
- The fifth chapter is the presentation of the results as well as screenshots of the actual work with descriptions of the interface, use cases of the platform and its validation.
- The sixth chapter presents a summary and conclusions regarding this dissertation as well as work that could be done in the future.
- Finally, the References/Bibliography used to produce this work.





## Chapter 2

# Background

*This chapter introduces the state of the art with the current knowledge and advances made in the area of software development for information systems as well as associated software engineering concepts.*

### 2.1 Information Systems

An Information System (IS) is defined as a set of components for collecting, storing and processing data, as well for providing information, knowledge, and digital products [3]. Nowadays, it is rare a company/organization not have an IS to support its business strategy or management. Thus, associated with the enterprise sector, the Enterprise Information Systems (EIS) are the key IT assets for industrial enterprises to organize, plan, schedule, and control their business processes [4].

In an increasingly digital world and with the constant presence of the web, an ever growing volume of information is generated and needs to be processed by these systems as well a growing number of companies have a business model that depends exclusively on their information systems. The increasingly competitive web market leads to the need for greater flexibility, dynamism, and speed in the development of these systems. For these reasons, we are witnessing an evolution of the traditional development of information systems, driven by new methodologies and trends such the Agile Software development and the advent of Cloud computing.

## 2.2 Cloud computing

Nowadays, there is a solid and robust network infrastructure capable of transmitting large amounts of data very quickly. This allowed an easy and convenient way to make services available over the web, and that is where the term "Cloud computing" comes from and became a mainstream in 2006 [5]. Defined by NIST as: "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (...) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [6] Hence, Cloud computing allows companies or users to concentrate on their business model, strategy or product without having to worry about everything that involves the software, platform or infrastructure of their service. Cloud computing is classified with a three-tier service model: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). The Figure 2.1 describes each of the layers of services provided by the cloud.

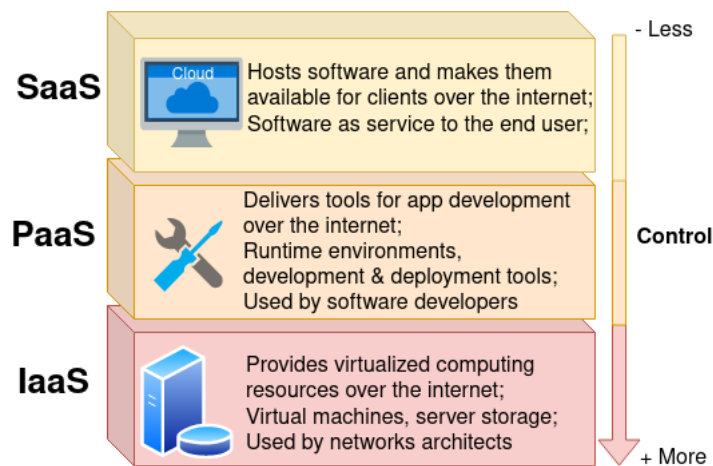


Figure 2.1: Cloud computing service model, text adapted from [7]

In addition to the above, Cloud computing offers a modern way to make available, or even, create software, through a client (browser). Solutions based on Cloud technologies have become very popular and have been developed by large companies such as Amazon through Amazon Web Services (AWS), which in turn are used by developers to create their own platforms for various fields like the medical sector [8].

Also, the new technologies inherent to Cloud computing have accelerated the emergence of platforms made available through the web referred to as "Low-code" that allow a rapid development of software. Moreover, cloud enables businesses to use less-expensive technology to rapidly deploy their low-code solutions, thus proving to be the ideal complement because both arise in the context of providing an increase in productivity combined with

reduced costs, greater agility and scalability [9].

## 2.3 Low-code and No-code Platforms

Low-code Development Platform (LCDP) refers to a platform that allows faster development and easy deployment of fully functional software, usually by abstracting code into visual elements or high-level programming such as model-driven and metadata-based languages. Moreover, some of these platforms are made available through a cloud computing environment via the PaaS model. Intended to be accessible to citizens without programming knowledge, known as "Citizen developers" i.e. a persona who is not used to the IT environment and who performs businesses in other fields [10]. No-code platforms imply that the user of the platform does not have to write any code, while a low-code platform requires some programming skills.

LCDP have had a great acceptance because they meet an increasingly dynamic and high demand market of software solutions and services as they provide more speed in its development and delivering. Not all companies are able to meet this high demand due to lack of resources, time, or even inability to hire (due to scarcity) highly qualified programmers, so these platforms end up filling these needs.

As an example, the COVID-19 pandemic situation has increased the search for software that needs to be development in a short period of time and with changing requirements, this promoted a even more adoption of LCDP [11] [12].

These platforms allow the developer to focus on the data model or the business model without having to worry about coding the other elements of the software because the platform takes care of that automatically. The platform should be able to build most of the infrastructure and deal with the software deployment so that the developer who has knowledge about a certain subject inherent to a data model should be able to build forms without having to handle the database structure, such as SQL, table schemas, etc. In the Figure 2.2 is shown a generic architecture of the low-code platforms based and adapted from [13] consisting of four layers: Application Layer (the graphical interface in which the user interacts), Service Integration, Data Integration and Deployment.

Regardless the advantages of these platforms, it must be said that they are not, at the moment, the solution for every type of requirement, which is caused by their limitation in terms of customization and the impossibility of supporting every type of features for certain situations, which end up needing programming through traditional methods [14].

The fact that these platforms speed up the development of systems and allow a certain dynamism at the level of contents to be implemented in an information system makes them the most obvious object of study to be

researched, and therefore they were taken as the basis for the development of the software presented as a solution result of this dissertation.

In the next chapter some of the best known low-code platforms will be analyzed in order to meet a survey of requirements and to understand how this type of platforms can generate information systems with dynamic content.

## 2.4 Model-Driven Development (MDD)

The concept of Model-Driven Development (MDD) is closely associated with low-code development platforms because it is a way to transform ideas into applications that deliver business value through abstraction, automation, and openness [15]. MDD states that only the model of an application needs to be developed, and the rest is automatically generated. In this way the realization of ideas into applications becomes easier because models are high-level interpretations (usually visual components) that abstract complex programming languages with rigid syntax and enable more effective collaboration and communication between business domain experts and software developers by breaking the language barrier. No-code and Low-code development platforms are driven by this development concept because they handle the models and generate their processes and configurations in an automated way by transforming them into a programming language (source code, general-purpose language, such as Java, C) bringing benefits for example in terms of efficiency, development time and reduction of human error. Each of the models is designed through a Domain Specific Language (DSL) with its logic that handles all the technical aspects of the application with a higher level of abstraction optimized for a specific class of problems and that allows a better integration between the technical development team and the domain experts.

## 2.5 Databases

### 2.5.1 Relational Databases

Humans have been storing information for a long time, even before the invention of the computer. Databases are inherently related to information systems because they are one of their primary components. However, the concept of Relational Database as we know it today, was conceived by a computer scientist from IBM named E.F. Codd in the 1970s [16].

The relational model is an intuitive way of representing data in tables, where each row in a table is a record with a unique ID (key) and the columns of the table holds attributes of the data. This type of databases stores and provides access to data points that are related to one another and the

## LCDP architecture Layers

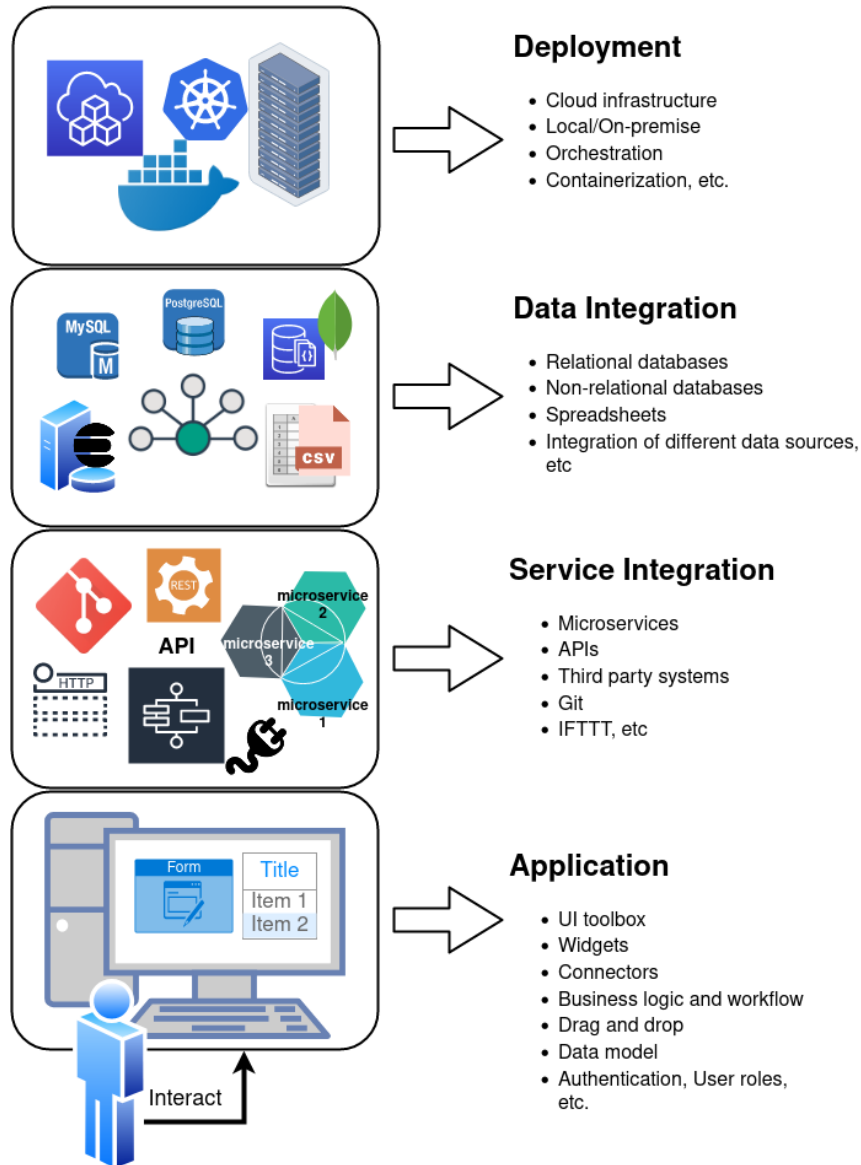


Figure 2.2: Layered architecture of Low-code Development Platform (LCDP) based on [13]

software used to store, manage, query and retrieve data stored in a relational database is referred to as Relational Database Management System (RDBMS) [17].

In 1979 a company called Oracle introduces the first commercial Structured Query Language (SQL) RDBMS and by the 1980s SQL had become

the standard language for relational systems. Relational databases continued to improve, and have been market leaders for the past 40 years. Currently it continues to be topping the *DB-Engines Ranking* for popularity.<sup>1</sup> The long time of existence gives them a great level of maturity, documentation and ability to adapt to various applications, and even today they are the type of database recommended in most systems to be developed, especially solutions that need a high level of consistency in handling and storing data.

Relational databases are thus characterized by the implementation of Atomicity, Consistency, Isolation, Durability (ACID) properties, and are mostly chosen by developers looking for these features [18]:

**Atomicity** - all changes to data, associated to a transaction, are performed as if they are a single operation.

**Consistency** - data is in a consistent state when a transaction starts and when it ends.

**Isolation** - the intermediate state of a transaction is invisible to other transactions.

**Durability** - after a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.

## 2.5.2 Non-relational databases

Although relational databases are still very much present in modern computing applications, new requirements and use cases are emerging due to the advent of the internet, cloud and big data. Never before has there been such a large amount of data being produced, manipulated and stored, and relational database systems have not proven to cope with the flexibility of these new types of data, which are very often unstructured or semi-structured data. Besides that, RDBMS were mostly designed to work well when scaling vertically and do not perform properly with the huge amounts of data of nowadays systems that needs to be distributed due to physical constraints or operational requirements. They have little support and poor performance when it comes to scaling horizontally.

Then a new type of database called non-relational, often called "NoSQL" or "Not Only SQL" began to emerge to support today's needs. The term NoSQL was first used in 1988 to name a relational database that did not have a SQL interface. Generally speaking, a NoSQL database is one that uses different ways and methods of data storage when compared to RDBMS [19].

---

<sup>1</sup><https://db-engines.com/en/ranking>

The NoSQL databases are built to scale easily while tolerate node failures with minimal disruption and, as opposed to relational databases that relied on ACID properties, many of NoSQL databases works with the database model BASE (Basically Available, Soft state, Eventually consistent) properties which aims to provide high levels of availability and resilience even though this may compromise consistency for a few moments [20].

In addition, when developing a system with a non-relational database, one must take into account that if it is a distributed system, it can only operate with two of the three criteria in the CAP theorem which means Consistency, Availability, and Partition tolerant [21].

In order to distinguish NoSQL database systems, they are classified according to their data model, i.e. how they store and access data. The four main categories of data models are: key-value stores, column-family stores, document stores, and graph databases [22]. It should be noted that at the time of writing, the *MongoDB* database management system, which is classified as a document store, is in the top 5 of the *DB-Engines Ranking*<sup>2</sup> which demonstrates its great adoption and popularity by developers.

The key points to consider when looking for a document store are that data is stored without the need of a schema definition, usually data is semi-structured in JSON format. Furthermore, the fact that it contains attribute metadata associated with stored content enables a way of query data based on its content, all this combined with the ease of running on distributed systems. These characteristics enables greater convenience when it comes to handling data provided by the *Big data*, which explains its great adoption, where the data comes in the most varied formats, large volumes and where flexibility is an essential factor.

### 2.5.3 Search Engines

In addition to the database systems discussed above, it is worth mentioning Search Engines, which are currently considered NoSQL database management systems dedicated to the search of data content. The most popular search engines are Elasticsearch<sup>3</sup> and Solr<sup>4</sup> that allow distributed full text search for high scalability. Both solutions are very effective when it comes to searching through huge amounts of unstructured information present in Big Data and support schemaless data structures with ease in indexing unstructured data and also the use of dynamic fields without the need for prior definition of the index schema.

In this context, an index is a mechanism for fast content discovery, which contains multiple Types that in turn contain documents (such a JSON con-

---

<sup>2</sup><https://db-engines.com/en/ranking>

<sup>3</sup><https://www.elastic.co/elasticsearch/>

<sup>4</sup><https://solr.apache.org/>

taining key-value properties)<sup>5</sup>. It can be compared to the structure of a relational database in the sense that the indices represent the database itself, the "Types" represent typical tables and the documents are the columns with respective rows. One of the capabilities of indexes, as being logical partitions of optimized collections of documents, is that they facilitate scalability due to another concept present, the "Shards". These allow to horizontally divide the indexes into shards, which allows for a more efficient distributed structure, since the indexes have no size limit on the documents they can store and could cause performance problems.

#### 2.5.4 Relational and non-relational databases comparison

Since there is so much choice when it comes to database management systems belonging to the two database families (relational and non-relational) and since nowadays it is not only relational database systems that are at an advanced stage of maturity, there is doubt when it comes to choosing the most viable and effective option at the time of developing software. This is why it is important to conduct studies that make a comparison not only based on direct performance of execution times but also on non-functional requirements and key properties for specific applications.

From the different studies analyzed and that directly compare relational databases with non-relational databases [23] [24] [25] [26] [27], there is a consensus that for a large amount of data the non-relational database is the best option because it executes queries with better response times than the relational one. This is explained by the advantages that a non-relational database system has in allowing better execution in a distributed and scalable environment.

In addition, non-relational data models have other advantages when used in specific Big Data applications such an E-Commerce product catalogue system as shown in [28]. In this study the authors compared the document-oriented data model with the relational model and identified advantages:

**Reduce Redundancy** - any irrelevant field in relational model holds null value and a document-oriented does not need to store a null value.

**Increase the Flexibility** - in relational data model, it is not possible to add any extra field out of its predefined fields. In non-relational data model there is flexibility due to the fact that it is not restricted to a fixed scheme.

**Unlimited Fields** - relational databases supports a limited number of columns in one single table. In non-relational unlimited number of fields can be added.

---

<sup>5</sup><https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html>



**Simplify Data Access** - in non-relational data model, every document is complete with its all required fields and does not refer to other documents for other fields.

Within the family of non-relational databases there are also several articles describing the characteristics of each NoSQL technology, concluding that each database is best for a particular use case scenario. In [19] the authors compared and tried to find the best non-relational database in a quality attribute perspective (performance is only a single aspect that influences the database choice). Similarly, the study done in [21] compares the different NoSQL models in terms of Persistence, Replication, Sharding, Consistency, Query method and Implementation language. Another interesting approach was taken by the authors of [29] which elaborated a binary decision tree that maps trade-off decisions to example applications and potentially suitable database systems. In this way, developers can select a particular technology by choose one set of desirable properties over another in order to better fit the requirements needed.

As shown in the exhaustive study carried out by Kamil Kolonko [26], in a direct comparison of performance, executing several types of queries, between the two most used databases of each family, in this case *MongoDB* for the non-relational and *OracleDB* for the relational, *MongoDB* outperformed *OracleDB* in read, scan, insert, update and read-modify-write operations, but this does not mean that *OracleDB* has no advantage as in fact it was also shown to have better consistency in the operations executed. As a matter of fact the relational databases are well suited for applications requiring complex querying, and can be considered as a way to overcome the ACID challenges.

It is worth pointing out that Relational databases are widely used in most of the applications and have good performance when they handle a limited amount of data, as shown in [27] when comparing MongoDB with MSSQL 2014 performing various operations. The difference between the results of each database was not noticeable until around 1.000 records, stating that relational databases, namely MSSQL is suitable for small and medium applications.

In addition to data volume requirements, non-relational databases are a very interesting option when it comes to increasing agility, since by being scheme-less the data model structure of the application is not predefined and can be fixed up during the building of application. In fact the non-relational data model implies the dynamic schema that therefore supports agile development, that is, adapting to evolving application requirements, easily make changes to an application without interruption change the schema as the data, application requirements or business evolves. Contrary, when the data of a use case has certainty about the features, relational data model is perfect for that (RDBMS supports only specific and with rigid schema allowing

small variations to the data).

In conclusion, the choice of database model should be mostly based on factors as the amount of data, the flexibility of schema, the budget, the amount of transactions that would be made and how frequent they are called [27].

### 2.5.5 NewSQL, Polyglot Persistence and Multi-model databases

The technologies that involve databases are constantly changing and adapting, not only due to the necessity derived from the Cloud Era and Big Data but also due to the search for solutions with increasing performance.

In addition to the technologies discussed in the previous section, other paradigms have arisen and are becoming established as a way to address changing requirements. As an example of this, an approach referred to as "NewSQL" has emerged in recent years [30]. Designed for a distributed architecture while at the same time having characteristics that belong to relational database systems, NewSQL systems can then be considered an evolution or modification of traditional SQL systems that aims to overcome some of their disadvantages, introducing NoSQL features such as horizontal scalability and the ability to run on multiple nodes and datacenters [21].

*NuoDB*<sup>6</sup> and *CockroachDB*<sup>7</sup> are examples of NewSQL distributed database systems, as they allow capabilities such as fast transactional interactivity, support for queries and the ACID operations typical of SQL thus having high consistency and at the same time allowing data replication and scalability. This type of system can be useful as an alternative to certain NoSQL applications where data analytics in combination with high consistency are required and distributed usage is necessary.

When analyzing the specific characteristics of each type of database system, be it relational or non-relational, it can be concluded that none of these are perfect and each one of them has strong and weak characteristics for a certain type of data or usage requirement. In this scenario, it has emerged a concept referred to as "Polyglot Persistence" where it generally involves using different types of databases, as appropriate, in different parts of the target system [31]. In other words, it means using the right tool for the right use case so that complex applications can take advantage of using different data models to store and process different parts of their data.

There is another approach to dealing with the variety of data in a system, which resembles the concept of polyglot persistence, and that is multi-model databases. These two concepts have in common the goal of dealing with data from different models using the best performing system for that purpose, but they contrast in the sense that while in polyglot persistence different

---

<sup>6</sup><https://nuodb.com/>

<sup>7</sup><https://www.cockroachlabs.com/>

independent databases are used and then a mediator is applied to unify and integrate them in order to answer the queries, in multi-model databases the goal is to build a single database system to manage the various data models with a fully integrated back-end [32].

As shown in Figure 2.3, developers of an online commerce platform can take advantage of polyglot persistence by using the best fitting database system for a certain type of data.



Figure 2.3: Example of Polyglot persistence in an E-commerce platform

Thus a document based NoSQL database such as MongoDB<sup>8</sup> or Couchbase<sup>9</sup> could be used to store all the products in the store as they are change-prone data types that benefit from the flexible schema of these database systems. To handle payments, traditional RDBMS are best because here can take advantage of the consistency required for transactions as well as the well-structured data. User sessions take advantage of the use of a key-value store such as Redis<sup>10</sup> as fast writes and reads are required without the need for durable data. For search capabilities, a search engine such as Elas-

<sup>8</sup><https://www.mongodb.com/>

<sup>9</sup><https://www.couchbase.com/>

<sup>10</sup><https://redis.io/>

ticsearch is best because it provides fast, indexed text searching capabilities and suggestions. Finally, to shape product suggestions and recommendations from the relationships between customers and their preferences, one can take advantage of a graph-based database system like Neo4j<sup>11</sup>.

Although polyglot persistence is a theoretically good solution, it is still immature because there are some challenges, such as the fact that there is no unified query language and no guarantee of interoperability or cross-database consistency. At the moment, the solutions developed in this sense need application code in order to unify the interactions and deal with complex queries from all the databases in the polyglot system [31].

Meanwhile multi-model systems like OrientDB<sup>12</sup> try to take advantage of the benefits of polyglot persistence in a single system, corresponding to the "one size fits a bunch" viewpoint, and is an approach that has been shown to have practical applicability. However, as demonstrated in [32], they are still an immature solution compared to the already robust and verified relational databases systems.

## 2.6 Related work on the development of dynamic platforms

Before addressing the most recent solutions, specifically the existing Low-code Development Platforms, it is worth looking at some studies and approaches that have been made in order to achieve a certain flexibility and dynamism in the design of information systems.

### 2.6.1 Dynamic GUI forms

Motivated by the amount of data that is required in the field of biomedical research, where there is a need to quickly collect data either by personal interviews, online questionnaires or other data sources, a solution that dates back to 2008 was developed [33]. The authors created a platform consisting of a flexible database. This platform allowed, in a user-friendly, fast, secure and reliable way, to create the systems for obtaining and managing these data collections, such as an information system. Briefly, the dynamism of the data model design was achieved through a data repository based on entity-attribute-value with classes and relationships (EAV/CR) where the surveys (forms) are stored. This way the information system developer can make updates to forms on the fly without any manual database table changes. Furthermore, the platform was supported by a conversion system in which the EAV/CR model with metadata is transformed to a table in the relational database and attributes are transferred to table columns, thus

---

<sup>11</sup><https://neo4j.com/>

<sup>12</sup><https://orientdb.org/>

allowing to offer the query power and simplicity of relational databases to external clients.

In another study [34], dating from 2010, the authors focused on an approach where they aim to decrease database system development time and amount of written code by inferring database code from the Graphical user interface (GUI). They refer to this process as GUI analysis development (GUIDE), in which a simple information system can be created by intuitively designing forms that are in turn analyzed (Form-oriented analysis) by the platform which then automatically creates the underlying database structure. This approach allows more flexibility for the developer (who does not need extensive programming knowledge) and reduces the time spent on database design in the Systems Development Life Cycle (SDLC).

Similar to [33], more recently there has been a work that took a dynamic form-based approach with creation and modification of the database schema [35]. This in turn was another attempt to innovate the traditional relational database design approach motivated by the need for healthcare-IT where it is intended that groups of researchers/knowledgeable staff in the data domain were able to create systems with flexibility to meet their dynamic and variable requirements without relying on EMR (Electronic Medical Record) which are often costly and not very dynamic. The key points of this solution are that there is a relational database that stores the metadata regarding the UI built by the user and another that stores the actual data entered. In addition the system has available templates based on the metadata of the forms that are displayed as UI and that users can modify. These changes make incremental modifications to the database schema while maintaining consistency. The Figure 2.4 shows a high-level abstraction diagram of the system developed in [35].

These studies show a similar approach when it comes to how the dynamism of the content of information systems is achieved. This dynamism is achieved at the user level because the user is able to flexibly modify forms with an abstraction from the back-end, in this case the database. The concept of using a forms metadata store allows for customizing schemas and then aligning the back-end storage data as the schema evolves. Allied to this was the feasibility of the database being designed from the intuitive forms by users with little knowledge of database administration as the platform takes care of transforming the forms to build the database. All these features such as automatic code generation, the design of the data model from interactive forms and the reduction of development time are features present in the current LCDP that are analyzed in the next chapter.

## 2.6.2 Database schema evolution

For some time now, techniques and methods have been present in the literature that aim to ease the database schema evolution concept, which

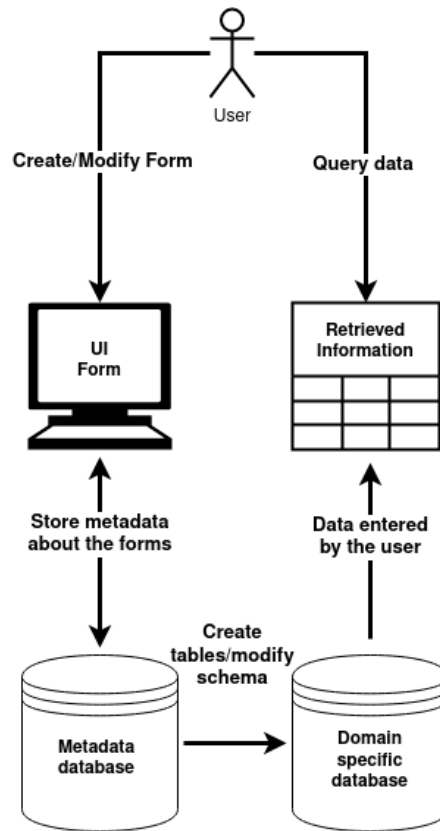


Figure 2.4: High-level Abstraction of the Form based dynamic database concept, based on [35]

is defined by the ability of a database system to respond to changes in the real world by allowing the schema to evolve [36]. It is considered one of the biggest obstacles when upgrading software, particularly information systems, and it is getting more complex because unlike traditional information systems, the new web-based ones are susceptible to much more frequent changes in the schema due to the dynamics and new requirements needed today, which combined with the agile development paradigm creates a barrier for developers at the time of continuous delivery because mainly relational databases are not naturally prepared for this flexibility.

To overcome this complex challenge that creates down-times (nowadays not tolerable) and costly expenses, there are some projects that tackle these problems and avoid developers having to make time-consuming and error-prone manual changes to databases every time there is an evolution in the schema. It is the case of the authors of [37] who worked on the PRISM project where they developed, among others, a language of Schema Modification Operators. The solution predicts old queries and updates to work on

the new schema versions and perform data migrations.

Even with non-relational (NoSQL) databases where most do not need a schema (schemaless), the application code usually assumes some sort of domain model mostly involving object mapper libraries for the persisted entities, the database may no longer be synchronized with the application code and therefore there is a need to maintain an evolution [38]. To assist in some of these aspects there are tools. For example, the "Dynamic mapping"<sup>13</sup> in Elasticsearch that automatically detects and adds new fields to the index, or the framework for controlled schema evolution in NoSQL databases developed by the authors of [39] which consists of an integrated solution in an IDE to assist developers when it comes to managing the combined evolution of the application code and the schema.

Furthermore, in the context of the agile development methodology, there are techniques that make it possible to apply continuous integration and continuous deployment in the development and design of the database to evolve as an application develops. These techniques were explained in Martin Fowler's Evolutionary Database [40] and were the basis for two leading open source tools for version control and change migration: Liquibase<sup>14</sup> and Flyway<sup>15</sup>. Both tools allow versioning and organizing database changes as well as deploying and tracking those changes. Although they have similarities in the capabilities they support, one major distinction is that Liquibase has also support for NoSQL database types.

---

<sup>13</sup><https://www.elastic.co/guide/en/elasticsearch/reference/current/dynamic-mapping.html>

<sup>14</sup><https://www.liquibase.org/>

<sup>15</sup><https://flywaydb.org/>





## Chapter 3

# Low-code development platforms

*This chapter presents the current Low-code and No-code platform solutions for the development of software, where information systems are included. The stated software systems are divided into two categories, open source and proprietary solutions. A brief overview of each one will be presented as well as comparison tables between the solutions reviewed with emphasis on the open source solutions.*

### 3.1 Open Source solutions

The low-code open source platforms were chosen to be studied in more detail due to its possibility to be extended or adapted as their source code is open and can be inspected. On the other hand, closed or "proprietary" solutions require licenses or some form of subscription, which makes their use in the academic environment more complicated. However, it should be noted that some of open source reviewed platforms contains paid versions with additional features. The following solutions were found by searching in the web using the key words "open source low-code platform". In addition, in order to perform a more comprehensive test, the open-source solutions were installed and configured on a localhost environment.

#### 3.1.1 Convertigo

Convertigo<sup>1</sup> is a Full Stack platform with multiple components for a Low-code and No-code development environment. This platform enables the quick building of enterprise grade mobile, tablet and desktop applications with the features provided depending and divided into editions.

---

<sup>1</sup><https://www.convertigo.com>

It includes an Eclipse-based integrated environment with all the layers (Back-end and Front-end) provided by the Convertigo platform. On this platform, it is possible to develop the back-end infrastructure of the system through a visual interface without writing extensive code and to create Front-end cross-platform client applications, by defining pages and graphical components through a UI based on drag and drop. Among the features offered for low-code back-end development are the setup of "connectors" to back end systems like databases and defining "transactions" for data exchange and data biding. It also allows the implementation of micro services, server side business logic, screen flow, local business logic and test cases.

One of the interesting front-end features is that the platform includes a pallet of components to build the UI and that can be dragged into a tree view in order to place components inside components. It also contains a built-in window for viewing in real time the application produced automatically through Ionic<sup>2</sup>/Angular<sup>3</sup> and TypeScript code automatically generated as the UI components are added.

Convertigo is a very robust platform and although it allows a lot of functionality, only its core is available in the open source "community edition" for academic use. This edition, despite providing the Low-code back-end, does not offer paid features such as more SQL and NoSQL connectors for easy data biding. Also one of the most distinguishing features of the Convertigo platform, "Full sync" is not included in the free open source platform. This feature enables mobile applications to handle offline data, i.e. allows the use of the application in an offline state. It replicates and synchronizes the data on the Convertigo Server through a NoSQL database.

### 3.1.2 Joget

Joget<sup>4</sup> is an open source low-code platform that is accessible to the developer through a web-based interface, i.e. using a browser (Figure 3.2) Furthermore the server running the platform can be installed independently of the operating system and the database as long as it supports Java and it is a SQL based database.

It is also divided into editions, the open source "Community Edition" already contains many features such as form building, through a drag and drop UI (Figure 3.3), tabular data lists with the information coming from the forms which again can be managed through the choice of columns without the need to write any query from SQL code. It also contains a UI with several components for creating responsive "Userviews" that embraces Google's Material design in a Progressive Web App (PWA) format and a drag and drop process builder for designing workflow processes through diagrams.

---

<sup>2</sup><https://ionicframework.com/>

<sup>3</sup><https://angular.io/>

<sup>4</sup><https://www.joget.org/>

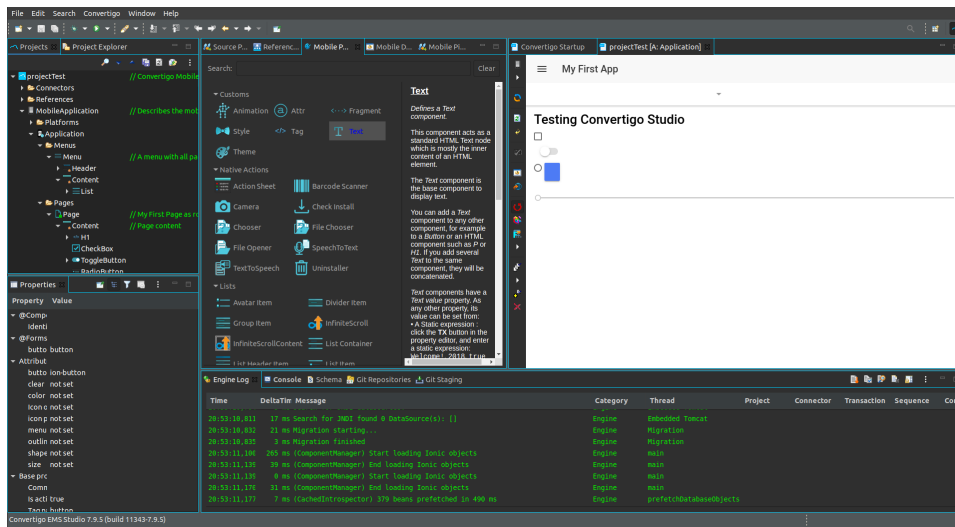


Figure 3.1: Convertigo Studio IDE Interface

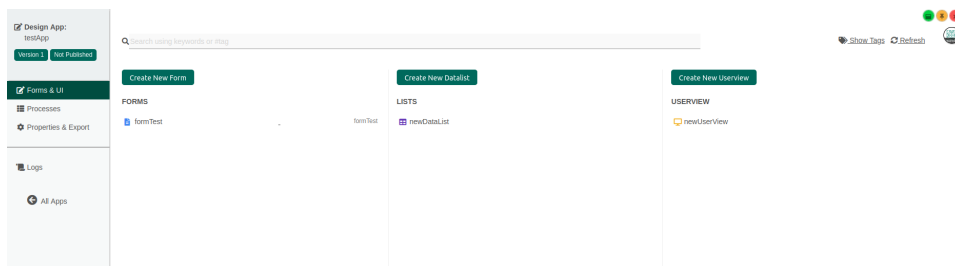


Figure 3.2: Initial page of Joget web platform

The platform supports a dynamic plugin architecture that makes it extendable and adaptable to more complex requirements without the developer having to change the core source of the platform.

When testing Joget on a localhost environment using MySQL database, some of the strengths that were highlighted were the drag and drop UI containing functions that are very accessible and easy to learn, requiring no code to be written to create a complete information system. Even the operation of the database is completely abstracted from the user as shown in Figure 3.4. The platform automatically builds the tables in the database and modifies them as the developer designs the data model (forms). The generation and deployment of the created application is just a button away. Figure 3.5 shows the newly created IS with the form defined in the UI presented in Figure 3.3.

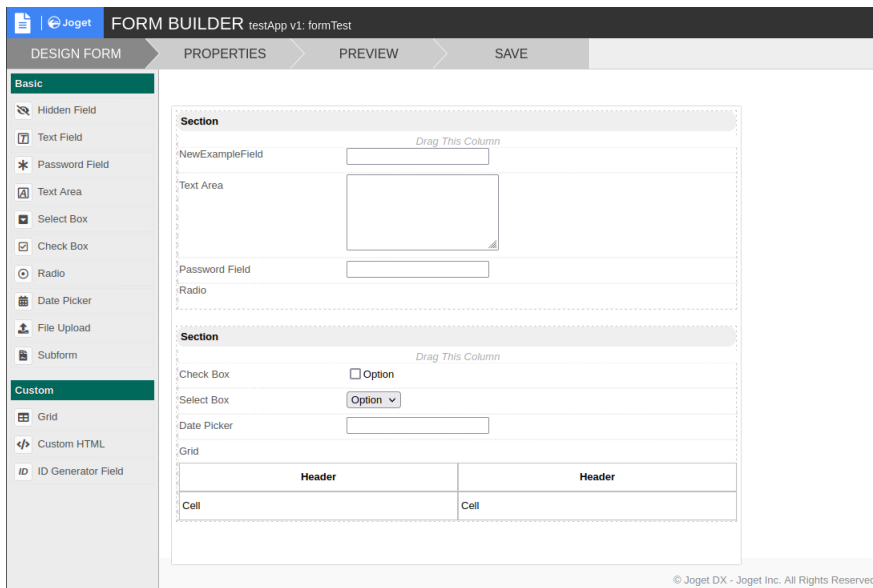


Figure 3.3: Joget drag and drop form builder interface

```
mysql> SELECT * FROM app_fd_formTest;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | dateCreated | dateModified | createdBy | createdByName |
| c_field6 | c_field8 | c_field3 | c_field2 | c_field5 | c_field4 |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 7d545b7d-254b-44a7-9673-e03e37fc5e4c | 2021-10-01 14:14:52 | 2021-10-01 14:14:52 | admin | Admin Admin |
| 2021-10-08 | [ ] | Test text area | | |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Figure 3.4: SELECT query run in a MySQL bash showing the table automatically created from the form defined by the developer in the UI.

### 3.1.3 Budibase

Budibase<sup>5</sup> is the youngest full-stack low-code platform reviewed. According to its open source github repository, it started being developed in 2019 and currently has very active commits. It is also the platform that is closest to a No-Code platform due to more abstractions in visual components. It is noticeable that the developers have taken extra care in its visual design in order to facilitate learning by those who use it.

The front-end can be built very easily by selecting visual components with menus and properties (Figure 3.6) that generate UI using Svelte framework<sup>6</sup> and can be changed without using code. This approach to a No-code platform is intended to bring the obvious advantage of ease of use for users with little programming knowledge and eases the learning curve, plus devel-

<sup>5</sup><https://budibase.com/>

<sup>6</sup><https://svelte.dev/>

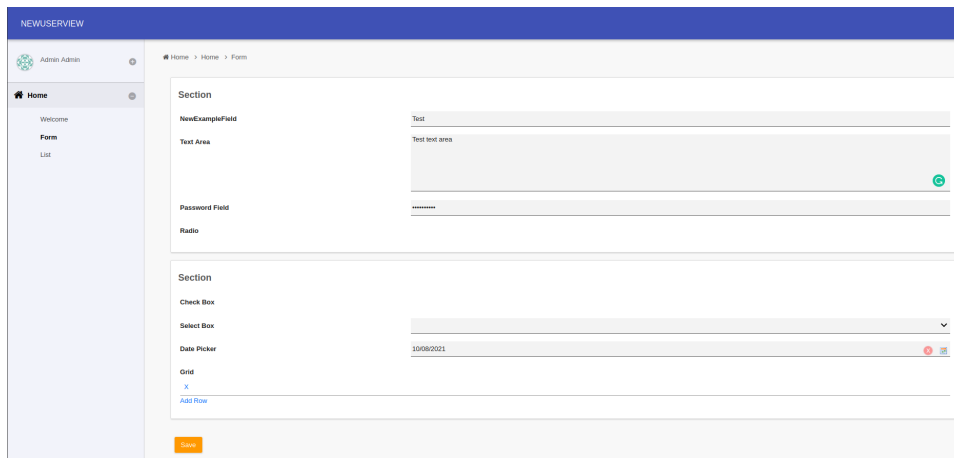


Figure 3.5: View of the form page running on the web application created with Joget

opment time is greatly reduced as the platform reduces the amount of code needed. On the other hand, the customization flexibility is not so high.

On this platform it is possible to select connectors to a couple of predefined data sources available, with options to choose SQL databases, NoSQL and even use REST APIs. In addition, the platform also contains a built-in database where tables and their fields can be created directly from the UI or imported from CSV. An interesting feature is the fact that more than one data source can be used in the same application, i.e. create a polyglot system.

Besides the choices of data sources and front-end creation, workflow automation is also easily created using components and visual diagrams. In the test performed, it was possible to create an action, such as being redirected to another page after submitting a form, all programmed quickly through the platform's UI.

It should be noted that in its architecture, this platform contains a NoSQL database, CouchDB, to store the metadata for all the applications created, in order to be able to support data replication.

Despite being a fairly young platform, it seems to be an option to be taken into account, in the test performed it was the platform with the easiest deployment, most intuitive (with a design that is close to the current modern web apps) and with the easiest learning curve while allowing great flexibility.

### 3.1.4 OpenXava

Like the previously analyzed solution "Convertigo", OpenXava<sup>7</sup> is an Eclipse-based framework. It is a solution that was first developed in 2005

<sup>7</sup><https://www.openxava.org/>

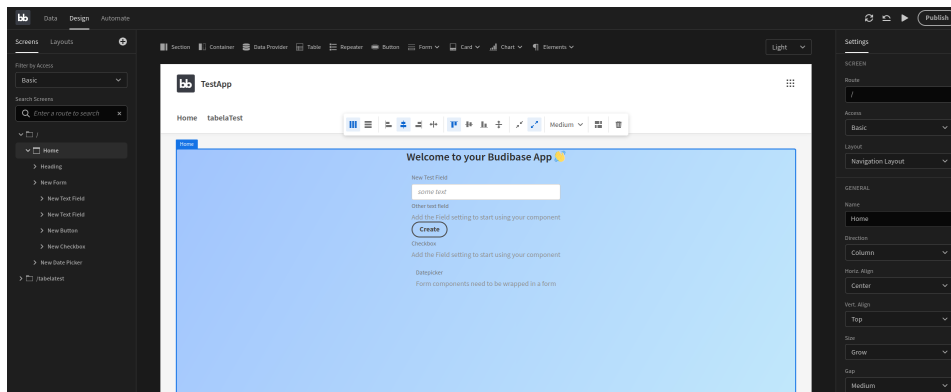


Figure 3.6: Budibase front-end design page

and since then has been a very active project with frequent updates, a large community and extensive documentation. It has a model-driven development approach, more specifically java-domain-driven. The core of the application is the development of Java classes that represent entities complemented by OpenXava's built-in annotations and other JPA's annotations (Java Persistence API). It is an Enterprise Java development solution with a fast learning curve, especially for developers with basic Java knowledge, but also to users that want to make their work faster and with less code writing. The platform generates a lot of code automatically (such as the user interface and authentication system) and it is flexible enough to develop real life complex business applications through agile development as it allows a very fast initial development and allows to make and view changes instantly by simply restarting and run the application.

OpenXava is extensible, customizable, integrable and includes support to call web services using JAX-RS, without adding extra libraries. Automatically generates and stores metadata for all entities being accessible for the developer to use more generic code.

The layout of the automatically generated web interface, can also be customized from the creation of views (always in Java annotations). For example, putting UI form fields on the same row is as simple as separating the same attributes by commas in the code. It is also possible to add validations, business logic and user interface behavior. Furthermore, by default, OpenXava allows to manage the entities created to perform the most common tasks such as adding, modifying, removing or searching from tables, generating PDF reports and exporting CSV (for example to import content into tables), all done by controllers which can be modified through XML fragments accessible to the developer.

The platform contains a database manager, to facilitate connections and transactions with the databases and supports any relational database as long as it is compatible with Hibernate. In the tests performed, although

OpenXava handles the evolution of the database schema, such as the automatic creation of tables and adding columns when the developer adds a new attribute in a Java class, it falls a bit short because it does not add the desired default values only when adding new records to the database, thus there is a need to rely on manual intervention for the evolution of the scheme.

In a few minutes, a CRUD application of a simple hospital management system is easily created just by defining two Java classes, one for the entity "Medic" and another for "Patient" as shown in Figure 3.7.

```

Patient.java
package com.yourcompany.medicplatform.model;

import javax.persistence.*;
import org.openxava.annotations.*;
import lombok.*;

@Entity
@Getter @Setter
public class Patient {

    @Id
    @Column (length=10)
    int number;

    @Column (length=10)
    @Required
    String name;

    @Stereotype("MEMO") // This is for a big text, a text area or equivalent will be
    String pathologies ;
}

Medic.java
package com.yourcompany.medicplatform.model;

import javax.persistence.*;
import org.openxava.annotations.*;
import lombok.*;

@Entity
@Getter @Setter
public class Medic {

    @Id
    @Column (length=10)
    int number;

    @Column (length=10)
    @Required
    String name;

    @ManyToOne( // The reference is persisted as a database relationship
        fetch=FetchType.LAZY, // The reference is loaded on demand
        optional=true) // The reference can have no value
    @DescriptionsList // Thus the reference is displayed using a combo
    Patient patient;
}

```

Figure 3.7: Example of two entities created from Java classes

After running the application, the platform almost instantly creates the interface shown in Figure 3.8 and all the database back-end. The forms are also readily functional, after adding a Patient by filling out the form, a list (Figure 3.9) can be accessed with the records that are stored in the database.

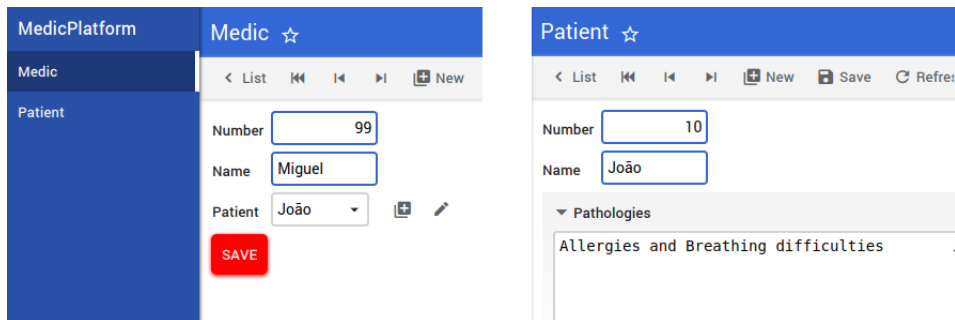


Figure 3.8: Automatically generated UI for Medic and Patient entities

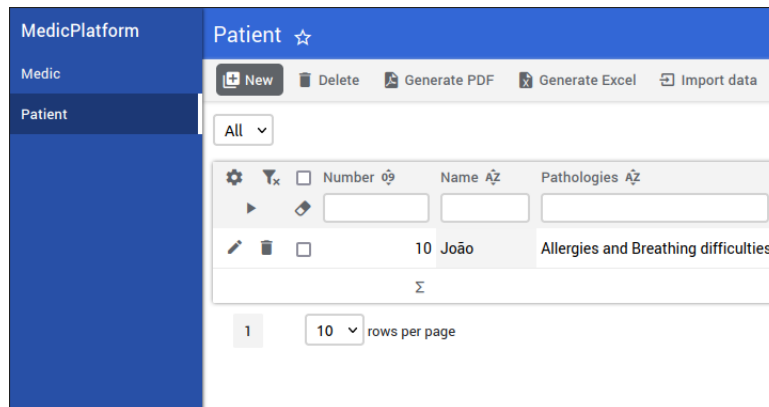


Figure 3.9: Patients list UI

### 3.1.5 Skyve

The Skyve<sup>8</sup> concept and prototype started to be developed in 2005 as part of Biz Hub Australia<sup>9</sup> and evolved until it became open-source in 2014. Nowadays, it is updated frequently being the result of years of experience in the industry and is presented in a mature state that intends to be the base platform for creating high-quality, secure, scalable and maintainable applications.

Therefore, Skyve is an open-source low-code Enterprise Platform that integrates a set of solutions (also open-source) and that distinguishes itself by using a declarative approach, and a Meta-driven development, i.e., from a high-level specification of metadata described in XML it is possible to build applications ready to use and in a consistent state without the need to use a programming language. This approach to a declarative language makes it possible to ensure security aspects (using the Spring Framework<sup>10</sup>), menus, navigation and CRUD activities (also includes a REST interface for CRUD operations on the database), while much work is abstracted from the developer, such as the generation of automatic code that is left to the platform.

A notable feature in Skyve is the automatic database creation and manipulation along with the searching and filtering support, this way the developer avoids SQL creation which can result in security vulnerabilities and other inconsistencies. The entire schema object creation process is handled automatically by Skyve via Hibernate<sup>11</sup> with the ability to easily change the database system to be used. Furthermore, Skyve includes, besides the structured database persistence repository, a NoSQL/content repository. This

<sup>8</sup><https://skyve.org/skyve-enterprise>

<sup>9</sup><https://www.bizhub.com.au/>

<sup>10</sup><https://spring.io/>

<sup>11</sup><http://hibernate.org/orm/>



dual persistence mechanism is transparent to the developer and a "Apache Lucene"<sup>12</sup> based automatic content management with NoSQL storage is included with Skyve (with the possibility of using other solutions like Elastic). It also includes, within the persistence handling, a tool that allows to easily backup to another database, supporting any database system as long as it is compatible with Hibernate. Skyve is also operating system and platform independent as its engine was created in "Java EE" and is compatible with any application server, such as Wildfly<sup>13</sup>.

Skyve allows to automatically generate highly usable UI, using open source tools such as PrimeFaces<sup>14</sup> and SmartClient<sup>15</sup>, to meet the domain model defined in the XML specified metadata. The evaluation performed demonstrates the steps required to build a ready to use application using the Skyve engine in the Eclipse IDE.

The first step was to create a project from the Skyve website, this way it was possible to download a quick-start project that can be imported as a Maven project<sup>16</sup> in Eclipse, which already includes the necessary dependencies specified in pom.xml. After that, the necessary configurations for the Wildfly server and the PostgreSQL database were made.

Next, two entities "Nurse" and "Doctor" were specified using the Skyve metadata nomenclature in two XML documents (Figure 3.10 demonstrates the Nurse) and also a file representing the "MedicStaff" module (Figure 3.11) which contains the two entities and where the Roles are specified, i.e. the group of users who have access to these Documents, making them visible in the side menu automatically generated in the UI.

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?><document xmlns="http://www.skyve.org/xml/docu
2   <persistent name="Nurse_MedicStaff"/>
3   <singularAlias>Nurse</singularAlias>
4   <pluralAlias>Nurses</pluralAlias>
5   <bizKey expression="Nurse-{name}"/>
6   <attributes>
7     <text name="name" persistent="true" required="true">
8       <displayName>Name</displayName>
9       <length>10</length>
10    </text>
11    <integer name="age" persistent="true" required="false">
12      <displayName>Age</displayName>
13    </integer>
14    <date name="birthDate">
15      <displayName>Birth date</displayName>
16    </date>
17    <association name="doctor" type="aggregation" required="false">
18      <displayName>Associated doctor</displayName>
19      <documentName>Doctor</documentName>
20    </association>
21  </attributes>
22 </document>

```

Figure 3.10: XML document specifying the entity "Nurse" in Skyve metadata

<sup>12</sup><http://lucene.apache.org/>

<sup>13</sup><https://www.wildfly.org/>

<sup>14</sup><https://www.primefaces.org/>

<sup>15</sup><https://www.smartclient.com/>

<sup>16</sup><https://maven.apache.org/>

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?><module xmlns="http://www.skyve.org/xml/module" xmlns:ns
2 <homeRef>list</homeRef>
3 <homeDocument>Doctor</homeDocument>
4 <documents>
5 <document ref="Doctor"/>
6 <document ref="Nurse"/>
7 </documents>
8 <roles>
9 <role name="Viewer">
10 <description><![CDATA[Enough privileges to view Medic Staff documents.]]></description>
11 <privileges>
12 <document name="Doctor" permission="R_C"/>
13 <document name="Nurse" permission="R_C"/>
14 </privileges>
15 </role>
16 <role name="Maintainer">
17 <description><![CDATA[Enough privileges to create and edit Medic Staff documents.]]></description>
18 <privileges>
19 <document name="Doctor" permission="CRUDC"/>
20 <document name="Nurse" permission="CRUDC"/>
21 </privileges>
22 </role>
23 </roles>
24 <menu>
25 <list document="Doctor" name="Doctors">
26 <role name="Maintainer"/>
27 <role name="Viewer"/>
28 </list>
29 <list document="Nurse" name="Nurses">
30 <role name="Maintainer"/>
31 <role name="Viewer"/>
32 </list>
33 </menu>
34 </module>

```

Figure 3.11: XML document specifying the "MedicStaff" module in Skyve metadata

The next step is to execute the commands with pre-configured maven run configurations in Eclipse. The first command executed was the *"mvn skyve:generateDomain"* which generates the application domain, validates and compiles the metadata (XML files) in the project, checking that the domain is in a valid state. After execution, Skyve automatically generates the Java objects classes for the defined entities and also a file hbm.xml with mapping metadata that Hibernate uses to determine how to load and store objects of the persistent class. At this point, the tables for each entity are also created in the database. Finally it is executed the maven command *"mvn clean compile war:exploded"* to update the resources with the compiled project, creating a WAR file ready to be executed by the server. The Figure 3.12 and 3.13 shown the generated UI executing in a web browser and running in the WildFly server. Since Skyve is used in strict integration with the platform created for the purpose of this dissertation "DynamicIS", it will be discussed more in detail in Chapter 4.

### 3.1.6 Other open source solutions

There are other open source solutions that, despite not having been tested or analyzed in depth in the scope of this dissertation work, are worth mentioning because they present some interesting characteristics. Jmix<sup>17</sup>, formerly "Cuba Platform" is a plugin for IntelliJ IDEA based on Spring Boot and that intends to turn this IDE into a Rapid Application Development tool

<sup>17</sup><https://www.jmix.io/>

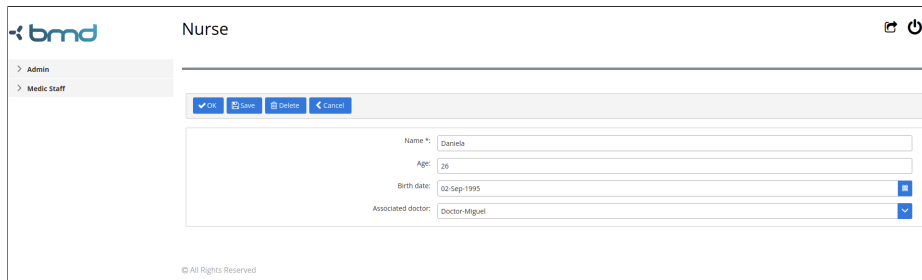


Figure 3.12: UI form for the "Nurse" entity and lateral navigation menu generated by Skyve

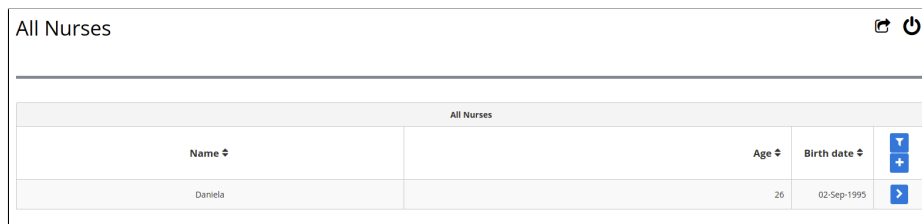


Figure 3.13: UI view containing all created Nurse entities

offering abstractions and extensive code generation in terms of data model design, data manipulation, business logic, security and UI creation. There is a gain in flexibility as the developer only has to design the data model from a UI and the platform is in charge of creating the data model schema in the database automatically and synchronizing (using the open source tool "Liquibase") changes as the data model and application code evolves. It is an interesting solution for developers who are familiar with IDE and Java programming but still want a way to increase their productivity, gain flexibility and decrease development time.

OSBP<sup>18</sup> project is another solution worth referring to. Its main developer is a software company "Compex" with more than 30 years of experience in Enterprise Resource Planning (ERP) solutions. Like the previous solution, OSBP is also a plugin for an IDE, this time for Eclipse Ecosystem, and combines No-code with Low-code in a model-driven architecture with automated application development, enabling the development of enterprise or any other applications in a fast and flexible way. Its multilayered architecture (data, business and presentation layer), represented in models and DSL, allows developers to modify and extend the application to meet new requirements. This platform is integrated with other open source frameworks and is not tied to a specific technology, being able to create solutions with different database systems (SQL and NoSQL) together in a single application.

<sup>18</sup><https://www.eclipse.org/osbp/index.html>

Running in a drag-and-drop web development environment, it is possible to identify two solutions, Saltcorn<sup>19</sup> and Corteza<sup>20</sup> platforms.

Saltcorn is implemented in Javascript and contains a multi-tenancy architecture (from a single host it is possible to run several applications). It is targeted for the creation of Content Management Systems (CMS) with structured data supported by a well understood and robust relational model. Among its features are: the creation of tables, views (display of information through queries defined in the UI), user and authorization management (user roles), front-end, events and actions, all with an error-free and plugin-extensible development environment.

The Corteza project, developed by Crust Technology (its primary contributor), is a scalable self hosted architecture to build organization's critical cloud web applications quickly. It provides a secure platform and is compatible with various protocols and standards such as REST API. It allows the creation of modules (forms that translate into tables) and workflow logic from the schema design, automatically generating responsive applications.

Finally, the platforms Appsemble<sup>21</sup> and Gramex<sup>22</sup> both take a component based approach (or pre programmed code blocks) to build data apps. It greatly simplifies the process of building an application, with fewer steps, less time and fewer bugs but at the expense of customization flexibility.

## 3.2 Proprietary solutions overview

Unlike open-source software, proprietary software belongs to an individual, group or company. Its source code is not publicly available and only the owner of the software is involved in its development. Therefore, the solutions present in this section, which have a different market target than the previous ones analyzed, have not been tested or reviewed in detail because their source code cannot be modified, extended or integrated into other solutions. Next, some of the most popular and leading solutions have been reviewed from their official websites or from third party studies in order to get an overview of their features.

These solutions are intended to be a tool often used in larger markets, offering their services and platform through licenses or paid subscriptions. Their growth and increasing adoption is an attraction for companies already established in the market for a long time and with robust solutions such as Microsoft and Salesforce that intend to evolve and adapt to this new paradigm and business model of low-code platforms. According to a Forrester report, the low-code market is expected to represent \$21B in spending

---

<sup>19</sup><https://saltcorn.com/>

<sup>20</sup><https://cortezaproject.org/technology/core/corteza-low-code/>

<sup>21</sup><https://appsemble.com/en/>

<sup>22</sup><https://gramener.com/gramex/>

by 2022 [41].

A recent Gartner report identifies the market leaders being: Microsoft<sup>23</sup>, Outsystems<sup>24</sup>, Mendix<sup>25</sup>, Salesforce<sup>26</sup> and Appian<sup>27</sup> [2].

In summary, Microsoft PowerApps and Salesforce Lightning platform are distinguished as being products that aim to extend the panoply of solutions from these two companies, offering integration with their own products as is the case of PowerApps which as deeper integration with many services in the Microsoft ecosystem such as Excel, Azure database or similar connectors to legacy systems. These proprietary platforms also contain pre-built templates or applications and components that can be reused. They are ready to use and can be obtained from stores within the platform itself such as Salesforce's AppExchange marketplace. Another feature, is Collaborative development support tools such as Mendix that allows collaborating project management in real time with peers and one of the oldest platforms, Appian, that supports collaborative task management. Support for very recent technologies such as some kind of artificial intelligence or machine learning is also present in the leading platforms, such as the decision engine with AI-enabled complex logic in Appian, AI libraries for discovery, prediction and voice services (Einstein) in Salesforce. The Outsystems platform, which allows the development and deployment of enterprise-grade applications quickly in a collaborative environment, also features AI-powered tools. These tools comprise an AI services layer, to provide automation enhancing the entire application lifecycle. Overall, the AI technologies available range from creating AI chatbots, to assist developers create the business logic of their applications.

It should be noted that Salesforce, Mendix and Outsystems support the creation of solutions for large companies as they are used to create large and scalable applications, while Appian's platform is more oriented towards small and medium scale companies and presents cheaper solutions.

### 3.3 Results and considerations

The current paradigm when it comes to low-code platforms is promising. It is a concept that is evolving quickly in software engineering as a result of the need to create applications faster, easier, and with a higher level of flexibility.

Although the proprietary low-code platforms are not susceptible to extension or modification they were briefly reviewed in this dissertation in or-

---

<sup>23</sup><https://powerapps.microsoft.com/en-us/>

<sup>24</sup><https://www.outsystems.com/platform/>

<sup>25</sup><https://www.mendix.com/platform/>

<sup>26</sup><https://www.salesforce.com/products/platform/lightning/>

<sup>27</sup><https://appian.com/platform/low-code-development/low-code-application-development.html>

der to have an overview of all the capabilities and features involved in these applications. The proprietary platforms are safer choices for large companies, due to their great ability to create very scalable applications, and since they are paid platforms there are greater guarantees when it comes to support and assistance given by the platform’s own developers. An innovative feature that distinguishes these platforms from open-source platforms is the fact that they have artificial intelligence or machine learning technology that is not found in any of the open source platforms analyzed, but also an advanced level of support for collaborative development.

The table 3.1 presents a summary of core features of the low-code open-source platforms in a comparative overview. Cell color indicates the feature support:

- Green: Supported;
- Yellow: Partially supported;
- Red: Not supported;

Feature/Platform	Convertigo	Joget	Budibase	OpenXava	Skyve
Cloud/ Web-based UI	Eclipse IDE based			Eclipse IDE based	IDE, Maven
Web-based/ mobile apps	Ionic, Angular	PWA	Svelte	Ajax	PrimeFaces/ Ajax SmartClient
Drag-and-drop UI	Only app GUI building				
Extensible		Dynamic plugin architecture			
Integration capabilities					
Workflow/ business logic		Drag-and-drop workflow creator	Drag-and-drop workflow creator	Auxiliary functions	Auxiliary functions/ Metadata
Multiple back-end support			Polyglot database support	Relational database supported by Hibernate	Relational database supported by Hibernate, Lucene based content manager
Database abstraction	SQL needed			SQL needed	
Easy app deployment				Any Java EE server	Maven commands, Any Java EE server

Table 3.1: Open source LCDP core features comparison.

Despite being a robust platform, Convertigo, in the open-source edition, does not contain more advanced features regarding database abstraction, losing some of the dynamism that is intended. The developer does not discard SQL, as he needs to create and modify tables in the system. The documentation is also not very complete which makes it difficult to understand some aspects and even to extend the platform.

OpenXava despite being embedded in an Eclipse based IDE such as Convertigo, follows a more traditional programming approach, not offering

No-code oriented capabilities such as drag-and-drop for UI building. However it does contain useful mechanisms and tools for faster development for developers with Java knowledge.

With more intuitive and fully drag-and-drop interfaces on the web are the Joget and Budibase platforms, which distinguish themselves from the others, for example, with the possibility of creating workflows visually. Nevertheless Skyve is the most interesting platform due to its structure and organization oriented towards the development of enterprise applications with security, performance and scalability aspects that the developer does not have to deal with directly. The complex capabilities of handling the database in terms of automatic schema changes and the total abstraction of the database for the developer provides more flexibility and level of desired dynamic content. In addition Skyve documentation is very extensive and detailed, the source code is well organized and understandable. The robustness of the platform and its feasibility of integration/extension were decisive characteristics for which it was chosen to integrate DynamicIS, the platform proposed which will be detailed in the next chapter.





## Chapter 4

# DynamicIS Proposal

*This chapter presents in detail the system developed, "DynamicIS", including functional and non-functional requirements. They were gathered from the user interactions, background technologies and related platforms. The proposed architecture and the technologies involved are presented. It is also described the processes that led to the implementation of the system as a whole.*

### 4.1 System Requirements

The development of the DynamicIS platform started with the specification and description of its capabilities, functionalities and behavior, in terms of non-functional as well as functional requirements. These requirements were defined mainly in iterations with the client of this proposal, "BMD Software", and were also based on similar analyzed platforms.

#### 4.1.1 Non-functional Requirements

In order to be a solution that meets the competitive expectations of the market and provides a good user experience, DynamicIS needs to behave in specific ways and meet the following non-functional requirements:

- **Performance and scalability** - The platform should respond quickly to user interactions with the UI, there should be no long response times when requests are made to the server. Pages and their components, including data from the database, should be processed without delays, errors and data losses. In addition, the platform including the database should be able to scale with support to large amounts of data without a decrease in performance.
- **Compatibility** - Following the trend of current software solutions, the platform should be accessible independently of the operating system

and should run on the web on different devices (mobile or pc) as well as being compatible with the latest web browsers.

- **Extensibility** - Given the nature and concept of the platform and the need to provide dynamism, it must be built with an architecture that supports flexibility to be extended or modified as new requirements and tools/frameworks need to be integrated.
- **Usability** - One of the key aspects of this platform is that it is oriented to end-users without programming knowledge in a No-code style of development. For this, it needs to be easy to use and learn, leading the user to execute the main functions without many steps and without being error-prone. Thus the platform should have a responsive, web-style interface design that leads to efficient development by the user.
- **Security** - As is already the norm in information systems, this platform must have security mechanisms that address issues of privacy in data access as well as protection against the integrity and loss of data.

#### 4.1.2 Functional Requirements

The set of functional requirements intended for the DynamicIS platform were obtained from user's interactions and adapted according to the used technologies specifications. Certain functionalities of related development platforms were also considered, specifically the "Skyve" platform, in order to achieve a better integration, allowing the user to quickly create and modify an information system from the definition of the data model. In this way the actors that will use the platform can be either Citizen Developers with knowledge of the data domain for the system they intend to create or more advanced Developers who intend to make a quick startup in the development without the need to write code. In both cases the platform must offer functionalities that make it possible to specify the data model, manage and organize it, create user interfaces, and generate/deploy the system from a button action, while the content of the information system created must be able to be easily modified or extended without major downtime by using UI buttons with auxiliary drag-and-drop functionality. The diagram present in the Figure 4.1 shows the set of main operations a developer might perform on the DynamicIS platform.

The following list presents detailed descriptions of the use cases specified in the previous diagram.

- **Log-in/Log-out** - A user should enter his log-in credentials in an authentication page in order to access the platform and the projects created by him. In addition, the user must be able to log out in order to terminate his session and thus the browser no longer display the user's content, redirecting the user to the authentication page.



Figure 4.1: DynamicIS platform use case diagram

- **Create/Remove project** - A user should be able to create a project by clicking on a button that redirects to a new page to fill the project information, such as the project name, the information system customer and endpoint of connection to the Skyve server. A project is an information system containing forms, views and result pages. The

user should be able to remove a project and the platform deletes all content belonging to that project from the database.

- **Edit Project Info** - Projects created by an authenticated user must be visible by that user and, after one is selected, the information in the project should be able to be edited by the user at any time, such as the project name, the endpoint connecting to the Skyve server, the name of the customer for whom the information system is being developed, and an image that will be the system's logo.
- **Import image logo** - A user should be able to import an image for the project logo, which will be visible in the UI of the generated information system.
- **Create/Remove module** - Within a project, a user should be able to create modules or remove them and all the content contained in the module. In the context of this platform, a module contains form pages, views and result pages associated with each other. All the content belonging to the same module can be accessed by clicking on the name of the module from a menu on the UI of the information system generated from the project.
- **View and reorder module** - On the project page, a user should be able to view the modules contained in that project as well as arranging them by drag-and-drop. The order of the modules will be reflected in the order they appear in the menu of the generated information system.
- **Create/Remove form page** - Within a module, a user should be able to create simple form pages. The user should give the form a name and add fields to it. Fields can be removed or reordered via drag-and-drop. Each field should consist of an option to add a description, a checkbox to identify if the field is required, a list box with the desired field type. If the selected field type requires any further option, a new element should appear to specify this option. Within the possible field types, the platform should provide text, numbers, dates, check-boxes, combo-boxes, files, images, and also associations between existing forms existing in the project. The user should be assisted of error messages after building the form, so that in this way the platform can build a consistent and error-free information system. Finally, the user should also be able to delete a form, and the platform deletes all content associated with it from the database.
- **Import form** - Within a module, there should be a list box with all the forms in the project that are from other modules so that a user can select and import them and quickly create a form from another.

- **Create/Remove result page** - Within a module, a user should be able to create results pages. The platform should allow the user to enter a name for the page and be able to choose a form for which want to create a results table. In addition, should be able to choose, from the available fields of the selected form, the ones to be visible as well as allow to choose a name for each field to be displayed in the table. If the user does not select specific fields, then all the fields of the table will be shown in the generated project. If the user selects a field of type image, then options should appear to select the option to show a thumbnail of the image in the table, as well as its dimensions, or just a link to the image source.
- **Create a view page** - Within the form building page, a user should have an option to create a "view" that consists of a page with more advanced customization. In the view, the user should be able to create layouts by inserting containers for vertical or horizontal arrangement of components, tabs and forms. In the forms, the user should be able to add columns and rows to insert the fields that were created in the forms page. Should also be able to add components such as Calendars, HTML code blocks and buttons. Also, on this page, the platform must provide a navigation view containing the organization of the elements arranged on the view page to be created. If the user does not create a view, the platform after generating the project will create a simple page for the form created in the forms page.
  - **Add a Calendar** - A user should be able to add a calendar on the view page and bind form fields to the scheduling of new events in the calendar.
  - **Add HTML code block** - For more advanced customization, or even integration of JavaScript or iframe components the user should be able to add HTML files with the ability to edit them in the platform in order to create blocks of HTML code.
  - **Add an action** - The user should be able to create a simple workflow, through redirection actions, in order to produce a sequence of steps to follow across different pages. This way, it should be able to create a button to redirect to another page, through the choice of forms and result pages in the project. The user should also be able to add the redirection behavior after the action of filling and saving the form in the created project.
- **View an embedded link of a form/view or result page** - The platform must provide a URL for all created forms/views or result pages. That way, the user can use this link to embed the form in other applications.

- **Group forms and/or result pages** - The user should be able to create groups by dragging forms to other forms, forms with result pages or results with results and also drag both to a previously created group. The platform should indicate how many elements of each type are within the group and should allow to change the name of the group as well as remove it. In this context, the groups are used to divide the modules so that they are displayed together in the UI menu of the generated project.
- **Reorder groups, forms and result pages** - Through drag-and-drop, the user should be able to reorder the elements within a module (groups, forms and result pages). The platform should allow switching between grouping and reordering of elements.
- **Edit form, results page or view** - At any time, a user should be able to view the forms, results page or views present in the project, specifically within each module, with the possibility of updating or editing each one.
- **Generate a project** - The user should be able to generate the project on click away and view the generation progress via a progress bar, as well as more detailed messages of the generation status and communication with the Skyve server.

## 4.2 System Architecture

Although it is built from scratch and is designed with the concept of expansion and integration with different platforms, DynamicIS was developed in order to have the Skyve open-source platform, discussed in the previous chapter, as the bridge between the content of an information system and its deployment. Thus, the developed platform adapts functionality from Skyve Core in order to integrate its own specific metadata into functional Skyve metadata that generates the information system.

DynamicIS takes advantage of Skyve's deployment mechanisms and capabilities but is independent of it. The contents of the information system can be created and changed at any time on the DynamicIS platform even if the Skyve server is not running, although can only be deployed once a Skyve server instance is online. To better illustrate how the system constituents are integrated, the Figure 4.2 shows a diagram of the DynamicIS system architecture.

The DynamicIS web app follows a client-server architecture. In this sense, the server is in charge of managing most of the resources to be consumed by the client. This way, the client, that is a web browser interface, can run on most devices, including mobile, without needing to be very robust in terms of hardware resources. Furthermore, this architecture provides

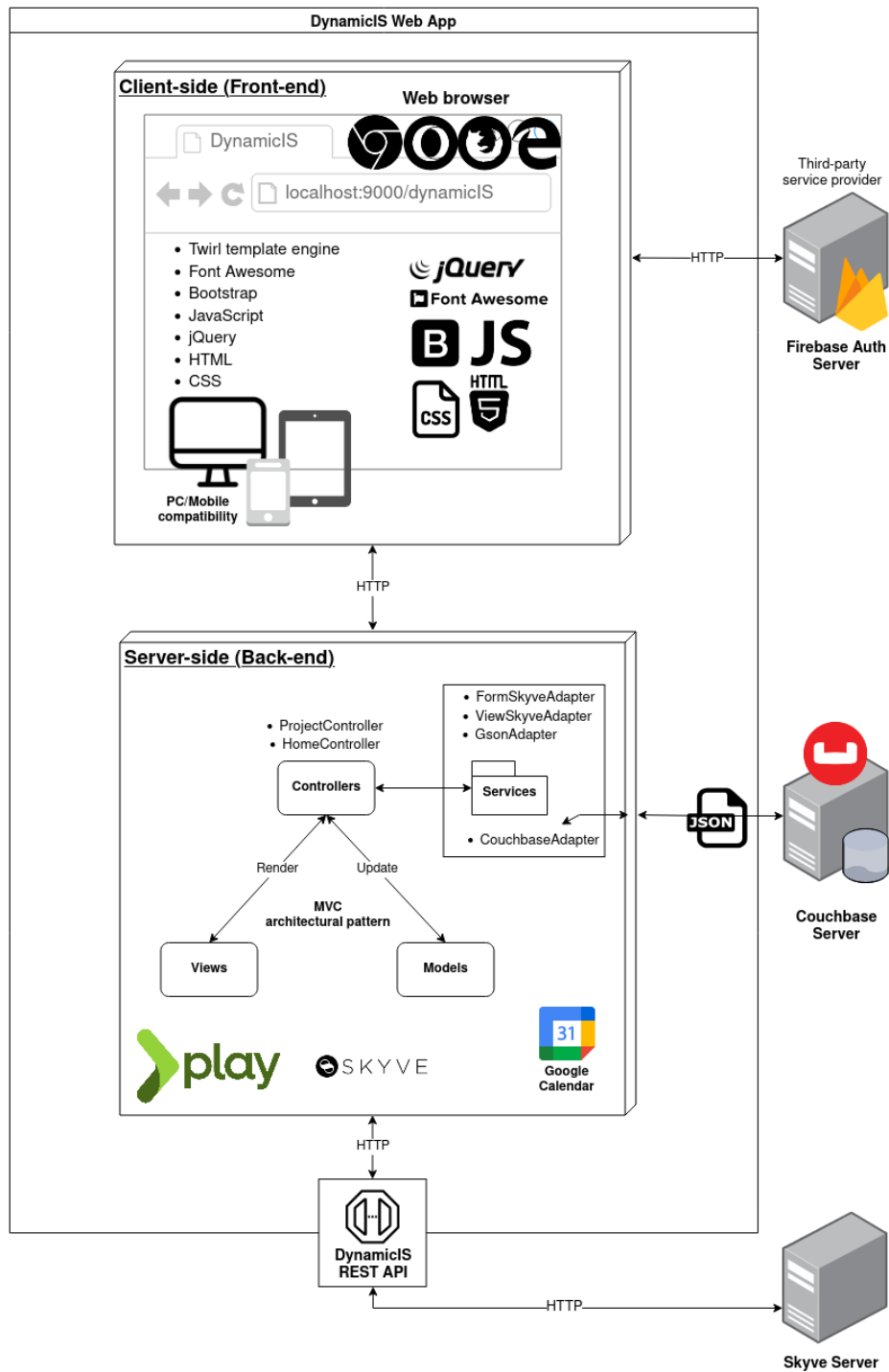


Figure 4.2: DynamicIS Web App architecture

a transparent interface for clients, since they have no knowledge of the application's specifications and business logic, such as database operations. In this sense, the client receives the users' inputs and makes a request to the server through the HTTP communication protocol, the server processes the request and sends a response that will be rendered in the client.

DynamicIS web app follows the MVC architectural pattern, which consists in the separation of the application in interconnected layers: Model, View and Controller. This pattern allows for a promotion of responsibility segregation development in which the representation of the data domain and how it is stored (Model), the rendering of interfaces and interactions with the user (View) and the processing of actions that can update the model (Controller), are separated. Besides the Controllers, services were also developed and correspond to useful classes with their own logic that are used by the controllers in order to obtain a better organization and logical separation. It is worth noting that the MVC pattern is naturally supported by the framework chosen for the development of DynamicIS, which will be discussed in the next section. The project structure of DynamicIS is represented in the Figure 4.3.

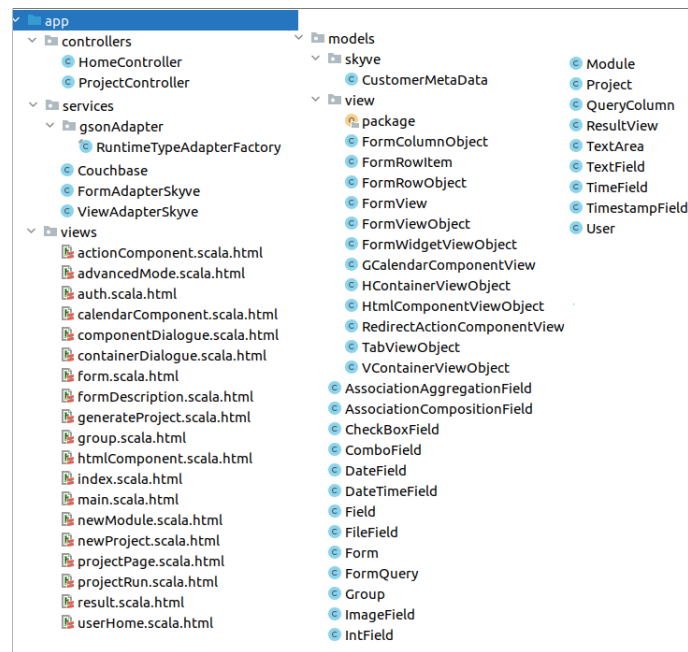


Figure 4.3: DynamicIS web app project structure

Finally, DynamicIS as a whole, follows a service oriented approach as the Firebase authentication server, the Couchbase server and the Skyve server instance are independent components that communicate with the Dynamic Web App through standard communication protocols and APIs. This way,



this system allows multi-tenancy providing SaaS. That is, from a Dynamic Web App instance, it is possible to manage several Skyve server instances that in turn deploy different information systems.

### 4.3 System Technologies

Play Framework<sup>1</sup> was the choice for the development of the DynamicIS Web app. One of the reasons for this choice was the need to support Java natively in order to be compatible with the dependencies and libraries of the Skyve core source code that was also developed in Java. Another feature is the integration of sbt<sup>2</sup> as a build tool, which besides allowing a better integration with Maven projects, has the ability to allow live compiling and hot reloading, enabling a more effective development. Additionally, the Play Framework contains very useful features oriented towards web application development, such as natively supporting Akka<sup>3</sup> in order to allow fully asynchronous model and highly-scalable applications with minimal resource consumption, a feature that was intended given the future-proof nature and architecture of the Dynamic Web platform. Another characteristic of Play Framework is the fact that it is a Full-Stack development tool and therefore also has a "Twirl" template engine that allows for a very dynamic rendering of web pages on the client-side. That said, it is also worth pointing out that Play is the framework used in BMD<sup>4</sup> projects, which further supported the choice of this technology.

In order to have extra functionality, it was necessary to declare some dependencies. This is done through the mechanism that Play Framework implements (via sbt) for managed dependencies. Since it was used specific Skyve Core functions for integration there was a need to add this dependency. However, as out of the box sbt uses standard Maven2 repository, it was necessary to add a resolver to find the Skyve repository. Furthermore, as part of sbt, it was also possible to add WebJars that provide a convenient mechanism to add client side dependencies packed into Jar archive files.

Going into detail about the libraries added in DynamicIS, demonstrated in Listing 4.1, the "guice" is provided and supported by Play and allowed the use of the Dependency Injection design pattern, which is present in other frameworks such as SpringBoot<sup>5</sup>. In that sense, the "guice" library allowed the use of the "@Inject" annotation in the constructors to permit the creation of dependent objects outside the class, separating the responsibility of the object creation in the class that will use it and allowing the injection

---

<sup>1</sup><https://www.playframework.com/>

<sup>2</sup><https://www.scala-sbt.org/>

<sup>3</sup><https://akka.io/>

<sup>4</sup><https://www.bmd-software.com/>

<sup>5</sup><https://spring.io/projects/spring-boot>

mechanism to wire together the components without the need to do it manually. In addition to the "Inject" annotation, the "@Singleton" annotation, also provided by this library, was used. This annotation allows the class that uses it, when injected, to create only one instance of that class. This mechanism is useful for complex and expensive components to create. In the DynamicIS web app, the @Singleton was used, for example, in the service class that connects to Couchbase.

```
resolvers +=
"skyve".at("https://repo.skyve.org/repository/skyve/")

libraryDependencies += Seq(
  guice,
  javaWs,
  "org.webjars" %% "webjars-play" % "2.8.0-1",
  "org.webjars" % "bootstrap" % "4.6.0",
  "org.webjars" % "font-awesome" % "5.15.2",
  "com.couchbase.client" % "java-client" % "3.1.2",
  "dom4j" % "dom4j" % "1.6.1",
  "javax.xml.bind" % "jaxb-api" % "2.3.0",
  "com.sun.xml.bind" % "jaxb-core" % "2.3.0",
  "com.sun.xml.bind" % "jaxb-impl" % "2.3.0",
  "com.google.code.gson" % "gson" % "2.8.6",
  "org.webjars" % "jquery" % "3.6.0",
  "org.skyve" % "skyve-core" % "7.0.3",
  "org.apache.commons" % "commons-lang3" % "3.12.0"
)
```

Listing 4.1: DynamicIS web app library dependencies

Another library supported by Play is "javaWs", which stands for "Web-Service". This library allows calls to other HTTP services and also makes asynchronous HTTP calls, which was the case used in DynamicIS to communicate with the REST API developed for the Skyve server. This allows the generation of a DynamicIS project in a information system, deployed on the Skyve server, as will be demonstrated in the implementation section.

For the XML manipulation and processing, required to create metadata structures to be integrated with Skyve, "dom4j" and "jaxb" were used. Also, to assist in the string manipulation process, the "commons-lang3" library was used, which provides extra methods to the standard Java libraries.

In addition to the other libraries that will be mentioned later, "GSON"<sup>6</sup> has been included, which is an ObjectMapper to allow the conversion of Java objects into JSON representations. This library was fundamental for storing data in Couchbase which is oriented to JSON documents, as will be demonstrated in the Persistence section. An interesting feature of GSON, that differs from other ObjectMapper libraries, is that it does not require

---

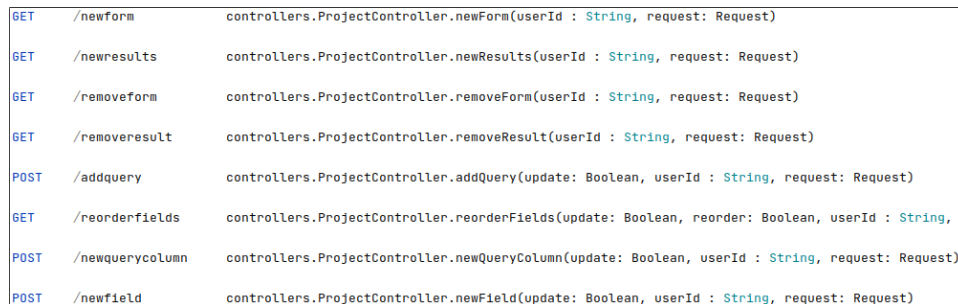
<sup>6</sup><https://github.com/google/gson>

the insertion of annotations into Java object classes as GSON infers them on its own.

Finally, it is important to state that this project was developed using only open-source or free tools. It was tested with Skyve 7.0.3 via Wildfly 21 server and PostgreSQL database. The DynamicIS web app was programmed to be used with a Couchbase database (tested with version "Server 6.6").

### 4.3.1 Server-side

As mentioned in the architecture section, the Play framework has the MVC pattern as its development orientation. The request life cycle boils down to an HTTP request received by the framework, which in turn resolves the request using the routes file that maps the URI to the "HomeController" and "ProjectController" controllers and invokes an action on it. This action can update one of the Models or fetch information from them using Couchbase database requests. Finally, the Controller renders a View and pops up the necessary information that will be sent to the Client through an HTTP response. The Figure 4.4 demonstrates an excerpt of the routes with GET and POST methods used in the application. Each of these routes has a Controller function associated with parameters and data coming from the request.



GET	/newform	controllers.ProjectController.newForm(userId : String, request: Request)
GET	/newresults	controllers.ProjectController.newResults(userId : String, request: Request)
GET	/removeform	controllers.ProjectController.removeForm(userId : String, request: Request)
GET	/removeresult	controllers.ProjectController.removeResult(userId : String, request: Request)
POST	/addquery	controllers.ProjectController.addQuery(update: Boolean, userId : String, request: Request)
GET	/reorderfields	controllers.ProjectController.reorderFields(update: Boolean, reorder: Boolean, userId : String, request: Request)
POST	/newquerycolumn	controllers.ProjectController.newQueryColumn(update: Boolean, userId : String, request: Request)
POST	/newfield	controllers.ProjectController.newField(update: Boolean, userId : String, request: Request)

Figure 4.4: HTTP routing excerpt screenshot present in "routes" file

The database chosen to integrate this system was Couchbase Server<sup>7</sup>. This is distinguished as a NoSQL, distributed, cloud-native database with replication support. It is a multimodal database oriented to JSON documents. In addition, it contains a query language (N1QL) with a SQL-like syntax for querying JSON documents. These features are essential and were the reason for the choice of this technology because they are in line with what is intended with the DynamicIS platform. The fact that DynamicIS allows multi-tenancy leads to the need for a database that supports a distributed and highly scalable architecture while offering consistent and fast

<sup>7</sup><https://www.couchbase.com/products/server>

performance. The agility and flexibility enabled by JSON's schemaless nature allows it to meet the development dynamism implicit in the DynamicIS platform that aims to be future proof with the possibility of integrating and being extended to other types of metadata. Couchbase Server was integrated into the system using the Couchbase Java Client SDK that allows DynamicIS to access a Couchbase cluster and perform database operations.

### 4.3.2 Client-side

One of the objectives of the DynamicIS web app is to be an intuitive application, user-friendly and, at the same time, allows a fluid use without long waiting times. In addition, a differentiating feature of DynamicIS is the No-Code development approach that aims to be compatible with mobile devices and therefore has to have a responsive user interface capable of adapting to different screen sizes.

To achieve this result and in order to make the application as efficient as possible the Twirl template engine embedded in the Play Framework and native JavaScript were used as essential elements. jQuery<sup>8</sup> was also used as it is a fast and small feature-rich JavaScript library that simplifies JavaScript usage in certain aspects such as manipulating the HTML document. In Listing 4.2 a use of jQuery in the script is demonstrated in which it executes a function after the HTML document is loaded using the ".ready()" method. This differs from the native JavaScript method "onload" because the onload event is only triggered after all the content of the document is loaded, causing a longer delay. In addition, Bootstrap<sup>9</sup> was used to achieve an application with a responsive design compliant with a mobile layout. Finally, Font Awesome<sup>10</sup> icons were used to give a more appealing and intuitive look to the user. Both libraries were added through Webjars.

```
@if(!serverReady){
  <script>
    $(document).ready(function(){
      var r = jsRoutes.controllers.
        ProjectController.
        testServerConnection(true, "@(userId)");
      window.location.assign(r.url);
    });
  </script>
}
```

Listing 4.2: JavaScript function to trigger the testServerConnection method from the browser using jsRoutes.

The twirl template engine allows to create simple text files containing, in addition to HTML and JavaScript, logic with Scala<sup>11</sup> code blocks. A tem-

---

<sup>8</sup><https://jquery.com/>

<sup>9</sup><https://getbootstrap.com/>

<sup>10</sup><https://fontawesome.com/>

<sup>11</sup><https://www.scala-lang.org/>

plate is like a function and can receive parameters such as objects from the application model. Other possibilities include iterating through for-loops and if-blocks in order to build simple logic in the Client. The declaration of reusable Scala code blocks is also possible and this has been done in DynamicIS web app where all template files include the "main.scala.html". All templates files of this project are contained within the "view" folder and end in ".scala.html", having to follow this naming convention in order to be recognized by the template engine and generate a class with a "render()" method. Listing 4.3 shows a simple method of the HomeController that, after being invoked by the HTTP request, will send a response, rendering the page defined in the folder view as "auth.html.scala" which is the authentication page of the DynamicIS web app.

```
public Result authPage(Http.Request request){
    return ok(views.html.auth.render(request));
}
```

Listing 4.3: HomeController method for rendering the authentication page.

An important feature provided by the Play Framework routing system is the ability to generate JavaScript code in order to handle routing from the client-side back to the server-side. In the context of the DynamicIS web app, this capability was used to invoke certain actions on the back-end from the client. An example use of this feature was the one that was developed on the page to generate a project. After the page is loaded by the client browser, it will invoke a method of the "ProjectController", "testServerConnection" (represented in the Listing 4.2) in order to DynamicIS server test the connection to the Skyve server. For the Play routing engine to generate this route, it is first necessary to expose it as shown in Listing 4.4 with a code excerpt from the "generateProject.scala.html" page template.

```
@helper.javascriptRouter("jsRoutes")(
  routes.javascript.ProjectController.generateSkyve,
  routes.javascript.ProjectController.testServerConnection
)
```

Listing 4.4: Embedded router generated inside a Scala template.

As for the authentication interface, it was used the open-source FirebaseUI on the client in order to promote the segregation of services and to take better advantage of Google's authentication system that eliminates boilerplate code and promotes good practices. Thus, FirebaseUI is an open-source JavaScript library for Web that was customized and integrated into the client interface of DynamicIS web app's authentication page. It will be explained, in more detail, in the implementation section.

## 4.4 Skyve Integration

As discussed in Chapter 3 of this dissertation, Skyve open-source was the platform chosen to integrate the DynamicIS platform and thus achieve a solution that allows the creation of information systems with dynamic content. The choice for Skyve was mainly due to the fact that it has automatic mechanisms to manage the database structure. Along with the specification of metadata, it creates a higher abstraction for the developer when dealing with relational databases. Thus, the information systems deployed by Skyve, has its data, and user's entered data, stored in a relational database, whereas DynamicIS contains an independent NoSQL database. This NoSQL database stores only the metadata of the structures (such as forms) allowing higher flexibility to make changes. This approach of having one database for the metadata and another for the user-entered data was based on the [35] study. The main difference is that the database used in that study for the metadata was also relational, while in DynamicIS it is non-relational.

The database option for the users information system was relational due to the fact that this technology stills very established in the market taking into account their 40 years of experience. So, for most information systems is continuous to be the best solution. Besides that, many companies would not choose to migrate data to a non-relational database. This coupled with the fact that it is easier to find programmers with experience in SQL with relational databases and that, in this type of information system, it is preferable to provide the user with data in a consistent state [42] which is a characteristic of this type of database.

That said, a significant part of the work needed was to understand the whole organization and functioning of Skyve, this was done by reading its official developer documentation<sup>12</sup> as well as, in more detail, sections of its source code<sup>13</sup>.

### 4.4.1 DynamicIS REST API

The way DynamicIS web app and Skyve's server communicate is crucial since the goal is to achieve an integration where neither platform is dependent on the other in a service-oriented architecture. For this purpose it was developed the DynamicIS REST API that only needs to be included as a directory within the "src/main/java" of a Skyve project. The structure of the API is present in Listing 4.5.

---

<sup>12</sup><https://skyvers.github.io/skyve-dev-guide/>

<sup>13</sup><https://github.com/skyvers/skyve>

```

dynamicISRest
|-- JaxRSActivator.java
|-- RestUserPersistenceFilter.java
|-- RestDynamicISService.java

```

Listing 4.5: DynamicIS REST directory structure

The "JaxRSActivator.java" file makes use of the JAX-RS API which defines interfaces and annotations for creating web services. This class is needed because it contains the resource registry for the server runtime since it extends the class "Application" and put it in the application's classpath that will call the "RestDynamicISService.class".

In addition, the "RestUserPersistenceFilter.java" is needed so that incoming REST requests are able to query and interact with the datastore. For this filter to be active and to comply with the security issues imposed by Skyve, it is necessary to edit the web.xml which is located in "/src/main/webapp/WEB-INF/web.xml" of the Skyve project and insert the url-pattern through which the Rest API will be accessed. Furthermore, it is also necessary to update the Spring security settings to allow access in the pattern specified in web.xml, present in the directory "src/main/java/org/skyve/impl/web/spring/SpringSecurityConfig.java".

Finally, the developed class "RestDynamicISService.java" contains all the endpoints defined by the paths (Annotation @Path) with the HTML methods (@POST and @GET) and with the consumed data type for all the interactions needed for the communication between DynamicIS and the Skyve server. Listing 4.6 demonstrates one of the API endpoints that is used to insert a new logo into the Skyve generated information system. This endpoint can be accessed from the DynamicIS Web app using the path specified in the @Path annotation and the base endpoint defined in the class "JaxRSActivator.java". In this case, the access to this API method is provided from "{skyveURL}/dynamicis/updatelogo/{customer}".

The DynamicIS REST API, used for the communication between the DynamicIS Web app and the Skyve server, will be discussed in more detail in the implementation section.

```

//Update customer image logo
@Path("/updatelogo/{customer}")
@POST
@Consumes(MediaType.TEXT_PLAIN)
public Response updateLogo( @PathParam("customer") String customer, String encodedContent) {
    try {
        assert(customer != null);
        assert(encodedContent != null);

        String dir = sourceProjectURL + "src/main/java/customers/" + customer +
            "/resources";
        byte[] fileContent = Base64.getDecoder().decode(encodedContent);
        Files.write(Paths.get(dir + "/logo.png"), fileContent);
    }
    catch (Throwable t) {
        t.printStackTrace();
        AbstractRestFilter.error(null, response, t.getLocalizedMessage());
    }
    return Response.status(Response.Status.OK).entity("Logo updated!").build();
}

```

Listing 4.6: DynamicIS Rest API endpoint for inserting a new image logo

The table 4.1 presents all the endpoints that are part of the API, containing their path, HTTP method, MediaType (the type of data the class accepts to consume) and a brief description of the function to be performed in the Skyve server.

Path (/dynamicis)	Http Method	MediaType	Description
/newdocument/{modulename}/{docname}	POST	APPLICATION_XML	Create new XML metadata form
/newview/{modulename}/{docname}	POST	APPLICATION_XML	Create new XML metadata view
/newcustomer/{customername}	POST	APPLICATION_XML	Create new XML metadata customer
/cleanmodule/{modulename}	GET		Remove forms and the directory of specific module
/removerole/{modulename}	GET		Remove roles for specific module
/testserver	GET		Test if Skyve server is ready and listening
/assignroles/{modulename}/{moduleupper}	GET		Assign roles to new created module
/updatemodule/{modulename}	POST	APPLICATION_XML	Update module XML file with new form
/updatelogo/{customer}	POST	TEXT_PLAIN	Update customer image logo
/newGCredentials/{credentialsFileName}	POST	TEXT_PLAIN	Create new JS credentials file for Google calendar
/newCustomAction/{className}/{moduleName}/{formName}	POST	TEXT_PLAIN	Create new Java generated action file
/applymavencommands	GET		Trigger Maven commands to generate project

Table 4.1: DynamicIS REST API endpoints

#### 4.4.2 Metadata flow

In brief, the tables created in the relational database of the Skyve server, which will be the mechanism for storing data from the information systems, are generated from metadata that in turn are created from the graphical interface of the DynamicIS web app. This concept follows the model-driven-development methodology because the user is only in charge of creating the models through the visual components of DynamicIS, which in turn is responsible for generating and processing the metadata necessary for the Skyve server to recognize.

In this way, and although Skyve works from a relational database with schema, this schema is abstracted from the user. Taking into account that



the user can at any time change the structures of his models with considerable flexibility without directly affecting the schemas of the Skyve tables. This is due to the fact that the databases of the two systems work independently and there are only changes in Skyve when the user gives the order to generate the project. Skyve contains a well-defined relational database allowing the generated information systems to take advantage of all the capabilities, has discussed in Chapter 2.

Therefore, from the moment the user defines his models (Forms, Result pages and views) until they are recognized as metadata by Skyve which will create the structure of the information system, there is an inherent process that can be explained by two phases: Definition of the model with storage in Couchbase (shown in the activity diagram of the Figure 4.5) and project generation (Figure 4.6).

The process represented in the diagram of Figure 4.5 starts with a user entering the page to build a form (the building process of the other models follows a similar pattern). If this form exists in the database, i.e., the form has already been created and the user is editing it, the client sends a request to the server's "ProjectController" to get the form which in turn will be retrieved from the Couchbase database through the service developed for that purpose. Since Couchbase is a document-oriented database, it stores the data in JSON that need to be converted to a Java Object of type Form via the GSON object mapper. Since the JSON of the form contains a list of Fields that can be of various types (Field extends other objects), the GSON library cannot map this inheritance natively so it was necessary to create a type adapter for this purpose. It was developed a function that creates a JSON object, iterates through the subtypes (identified by the key "type"), and registers the subtypes of the Field object, so that GSON can correctly parse it. After that a Java object of type Form is sent in response to the client that in turn processes the fields and dynamically builds the Form. On the other hand, if the Form does not exist in the database, the user builds it from the DynamicIS visual interface and submits the form. The server receives the request containing the processed fields from the <form> element of the HTML and creates a Java object of type Form. Then, the object is converted to JSON (using the GSON mechanism explained earlier) and stored in the Couchbase.

The second phase of the metadata flow begins when a user gives the order to generate the project. At this point, DynamicIS and the Skyve server start communicating through the DynamicIS Rest API. The process is described in outline in the activity diagram in the Figure 4.6 and will be explained in more detail in the implementation section.

As depicted in the activity diagram, the metadata stored in Couchbase will be requested. Note that the diagram only shows the process for one Form, although if there are more, this process is looped.

After the metadata is obtained from Couchbase, it is converted to a

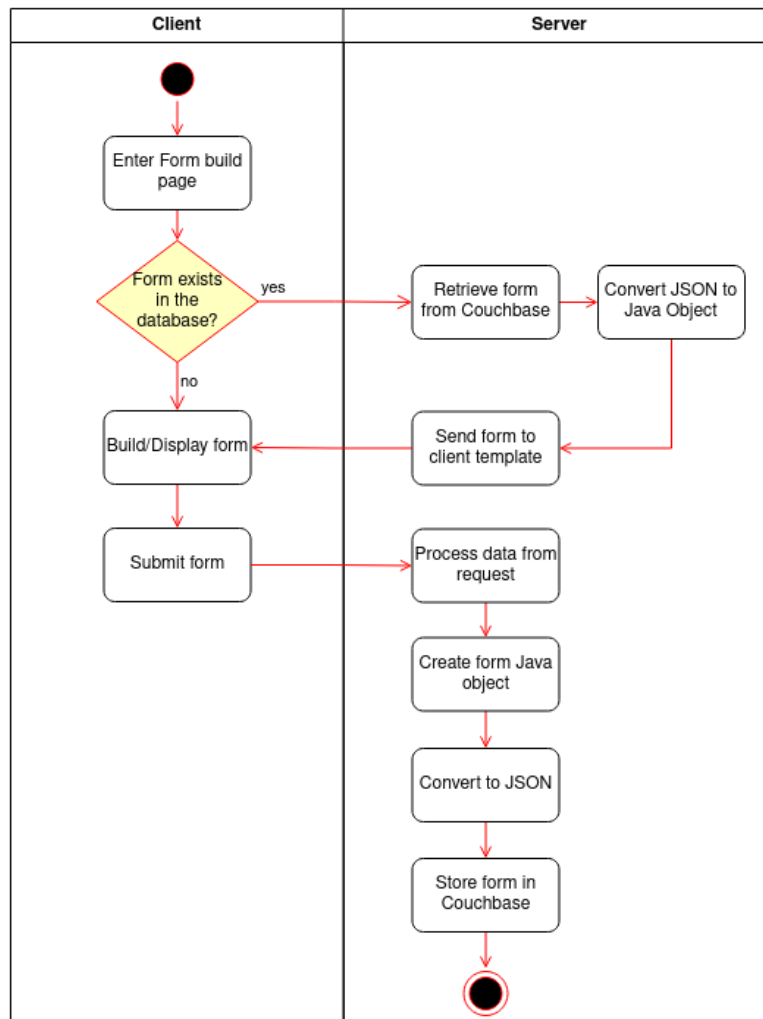


Figure 4.5: UML activity diagram showing the process of defining a model (Form)

Java Object of type "Form" (mechanism explained before). Then the service "formSkyveAdapter" will create a metadata class recognized by Skyve, which contains annotations of the "dom4j" library that allow to marshall the class, i.e., transform the representation of the object in an XML string. After that, the string is parsed to a DOM XML, using Play Framework utilities, in order to be sent by DynamicIS REST API, which consumes a MediaType.APPLICATION\_XML. After the Skyve server receives the metadata, the domain can be generated, building the tables and their schemas in Skyve's relational database (this part of the process is more detailed in the implementation section).

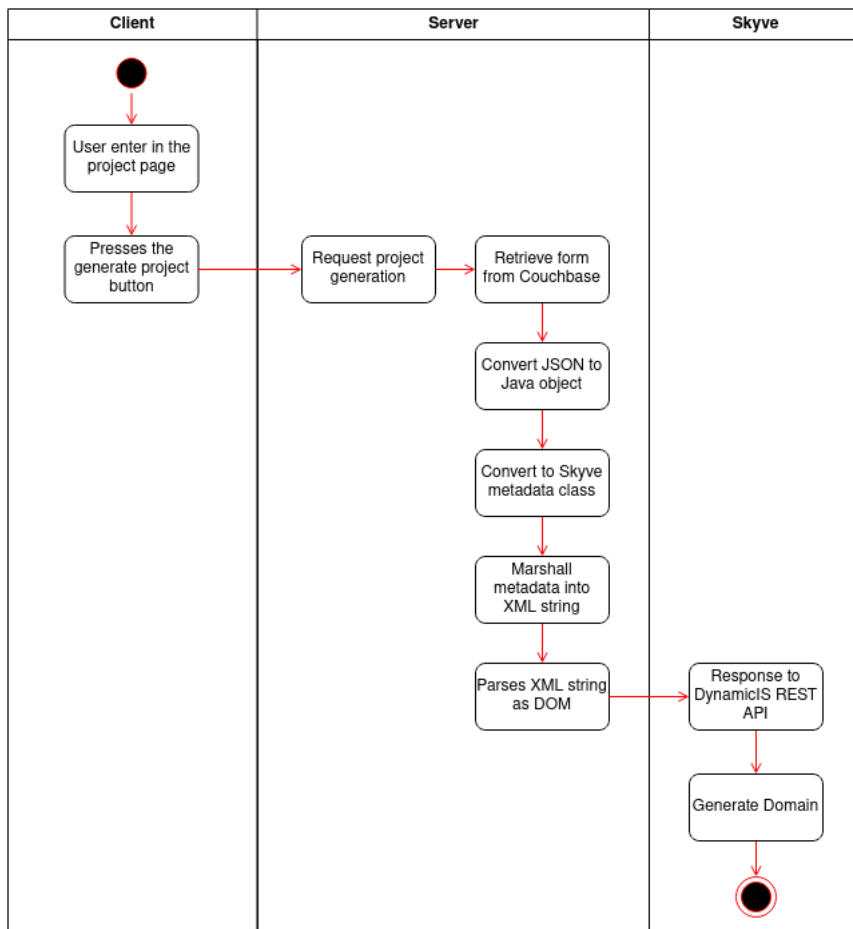


Figure 4.6: UML activity diagram showing the process of converting DynamicIS metadata to Skyve understandable metadata in project generation

## 4.5 Persistence

The data persistence mechanism chosen for integrating DynamicIS took into account mainly the ability to be highly scalable and flexible so that the application can progressively evolve along with the data model. Thus, the Couchbase database stores data in JSON structures that are extremely convenient for web-application programming due to its fast serialization and deserialization processes and its flexibility. That said, Couchbase’s storage unit is the document, which refers to a database entry and represents a single instance of an object in the application code. A document can be compared to a row in a table of a relational database with the difference that it provides flexibility because each document can contain JSON with varying schemas, i.e. different key-value pairs and the application is the one that defines and manages the structure of the documents since Couchbase does not enforce

uniformity. Given these characteristics, DynamicIS, being designed with Couchbase, can be extended to integrate other platforms (besides Skyve) that contain different properties and structures. So, if there is a change in the documents, they do not have all to be updated in the same way. Another important feature of the documents is that they can be highly self-contained, meaning that they have the advantage of supporting scalability, replication and can be accessed with low latency without other documents needing to be accessed<sup>14</sup>. In addition, documents can contain nested structures, allowing them to express relationships without referencing other data structures, like tables in the relational databases, allowing a more natural approximation to the hierarchy of data present in the application objects.

#### 4.5.1 Couchbase integration

The Play framework provides a plugin to manage JDBC connection pools for relational databases making the process of setting up and using data sources easier. Since non-relational databases are variable, they currently lack a common API and are not natively supported by Play. However, Couchbase server contains a Java SDK API that allows Java applications to access a Couchbase cluster easily. So, for the purposes of organization and segregation of services in the DynamicIS web app, the class "CouchbaseAdapter" was created.

"CouchbaseAdapter" is a service class that manages all operations and queries to the Couchbase server and serves as a mediator between the Couchbase API and DynamicIS. Couchbase supports "Indexes" which are necessary to make the data available to be found when querying. When CouchbaseAdapter is instantiated, it creates the necessary primary indexes to be able to perform queries. Next, it connects to the Couchbase Server Cluster and gets references to the buckets as shown in Listing 4.7.

```
public Couchbase() {
    ...
    this.connectionString = "localhost";
    this.username = "Administrator";
    this.password = "password";
    this.bucketName = "forms";
    this.bucketNameProjects = "projects";
    this.bucketNameModules = "modules";

    //Connect to Couchbase Server cluster
    try {
        this.cluster =
            Cluster.connect(connectionString, username, password);
    } catch (NullPointerException ex) {
        log.warn("Cant connect to Couchbase cluster");
    }

    //Get a Bucket references
    this.bucketForms = cluster.bucket(bucketName);
    this.bucketProjects = cluster.bucket(bucketNameProjects);
}
```

---

<sup>14</sup><https://docs.couchbase.com/server/current/learn/data/document-data-model.html>

```

    this.bucketModules = cluster.bucket(bucketNameModules);

    //Create primary indexes
    createPrimaryIndex(primaryIndexForms);
    ...
}

```

Listing 4.7: Connection to Couchbase Server (excerpt from "CouchbaseAdapter" class)

In the context of how Couchbase operates, a bucket is a way to logically group collections of documents. For the DynamicIS Web app, buckets were created to group documents belonging to the main entities: projects, modules, forms, groups, views, results and users (which will be detailed in the data model section).

Although Couchbase supports "N1QL", which is a JSON query language that can be used to perform many single-document operations with a SQL-like syntax, at its core is used a high-performance key-value store. This way, using the API for crud key-value operations is much more efficient because the requests can go directly to the correct node<sup>15</sup>.

It was with the performance of the key-value mechanism in mind that most database operations by DynamicIS are done through that API and not through "N1QL" queries. Since operations using the API for key-value are performed directly on a single document, they need to be provided with the ID that is unique to that document.

Thus, DynamicIS data model was designed in a way that some documents contain a field referencing the IDs of the documents they are related to. For example, whenever a new user is registered, through Firebase authentication, it is created and stored in the Couchbase with the ID provided by Firebase. When this user creates a DynamicIS project, it saves the ID of that project in his Couchbase document. Therefore, when the user logs into the DynamicIS web app, his Couchbase document is retrieved from the ID provided by the authentication. Then DynamicIS will iterate over the IDs of his projects, included in the user document, retrieving them individually through the key-value API. This way, the data retrieval is more efficient because complex queries are not used. Besides that, the documents retrieved from the database are smaller since they do not contain the entire entities of the objects but only ID references.

Listing 4.8 demonstrates the function of the class "CouchbaseAdapter" that performs the operation of retrieving a Project from the database using its ID. The Couchbase API key-value method that retrieves the document (still in JSON format) is the "get" and the parameter "RawJsonTranscoder.INSTANCE" is used to specify that the retrieved data is in JSON format, since the serialization to transform it into a Project object is done after using the GSON library.

---

<sup>15</sup><https://docs.couchbase.com/java-sdk/current/howtos/kv-operations.html>

```

public Project retrieveProject(String document_id) {
    GetResult projectResult =
        collectionProjects.get(document_id,
            GetOptions.getOptions().
                transcoder(RawJsonTranscoder.INSTANCE));

    String returnedJson =
        projectResult.contentAs(String.class);

    Gson gson = new Gson();
    Project project =
        gson.fromJson(returnedJson, Project.class);
    return project;
}

```

Listing 4.8: Couchbase method to retrieve a project (excerpt from "CouchbaseAdapter" class)

## 4.5.2 Database model

Unlike typical relational database systems that use the Entity-Relationship Model for the representation of the data model, DynamicIS is based on a non-relational document-oriented database. As it happens, this database stores documents in JSON format and it is difficult to abstract the document model to an Entity-Relationship Model. For this reason, it was opted to apply the authors's proposal of [43] for a standard approach to building data visualizations in the form of ER diagrams for NoSQL, more specifically document-oriented. Therefore, in order to better understand the database implementation of the DynamicIS system, a data model diagram was created, represented in Figure 4.7.

The diagram is organized into seven main entities represented in the shape of a document. Each document represents a class in the application but also groups (or collections) of documents that follow an identical structure, the so-called "Buckets" of Couchbase. The arrows between the documents represent a relationship between them, which is equivalent to a reference made by the ID of one document as an attribute of another document. For example, all documents of the entity User contain a list of IDs for entities of the type "Project" and the equivalent one-to-many notation as in the UML is used. In addition to this type of relationship through ID references, a document can have one or more embedded documents and in the Java class this translates to an attribute of an Object type and not just its reference in a String. In the case of embedded documents, the same notation of the UML is used. For example, the Document of the "Form" entity contains several documents of the "Field" entity, thus also constituting a one-to-many relationship. Hence, taking the Form Document as example, its JSON structure will be mapped (using the GSON library) in such a way that its key-values correspond to the following attributes of the developed Java Class represented in the Listing 4.9:



As can be seen in Listing 4.9, following object oriented programming practices, the attributes declared in the class are preceded by the keyword "private" in order to be protected from unexpected changes from outside this class. For the outside classes to be able to access the attributes of this class, getters and setters were created (not represented in the Listing).

Considering this, each of the declared attributes corresponds to a key in JSON and this does not have to be strictly followed as the application will still run if there are more keys in JSON. Also there is a "formViewId" attribute that corresponds to a reference to the "FormView" entity and a list of "Field" objects that correspond to documents embedded in the JSON of this entity as shown in Listing 4.10 which represents a document of type Form in Couchbase.

```
{
  "couchbaseID": "dd76afa9-e2a3-4ed4-b59a-7cf34c829129",
  "name": "Nurse",
  "icon": "fa-user-o",
  "fields": [
    {
      "length": 30,
      "idx": 0,
      "fieldName": "Name",
      "required": true,
      "type": "TextField",
      "description": "Nurse name"
    },
    {
      "defaultValue": 20,
      "hasDefaultValue": true,
      "idx": 1,
      "fieldName": "Age",
      "required": true,
      "type": "IntField",
      "description": ""
    }
  ],
  ...
}
```

Listing 4.10: Example of "Form" Document stored in Couchbase

That said, a description for each entity will now be presented in order to explain its functionality in the system (all the specified classes are located in the model folder of the DynamicIS web app development framework).

- **User** - As in almost every other software system, there is a need to store information about the user who interacts with the system. That is why this entity exists and stores information about the "uid" (which is acquired through Firebase's authentication service), "couchbaseID" which is the document id in Couchbase, the user's email used as authentication element and a list of Projects created by that user.
- **Project** - The purpose of DynamicIS is the creation of information systems and, in this context, this entity serves to store information



about a project that basically contains all the necessary content for the development of the information system. Thus, this entity has a project name, information to access the Skyve URL, project directory on the machine that contains the Skyve server, name of the Customer for which the information system is being created and also a list of modules created the last time the project was generated. In addition, the Project keeps a list of all the modules it consists of.

- **Module** - An information system usually contains sections that group together information with related content. For example, an information system should have a "Staff" module with information about doctors, nurses, etc. It is in this sense that this entity exists, in order to store the metadata of forms, results pages and groups. Note that a module will be a section in the navigation menu of the generated information system, so it needs to have a name.
- **Group** - This entity is used to organize result pages and forms more specifically than in modules. In the generated information system, all the content within a group (which also has a name) will be entered in a subsection of the menu.
- **Form** - Forms are a fundamental part of any information system. They are used to fill in information about a certain subject by those using the system and generally the system revolves around them. This entity was created for the purpose of storing metadata about Forms. As such, it contains a name (which is present in the navigation menu of the information system) the name for a FontAwesome library icon, a url to embed the form (outside the information system) and contains a list of all the fields that the user will add to the form. Although not represented in the diagram in the Figure 4.7, the Fields are extended by other child entities that represent subtypes of fields and that correspond to most of the types used in forms which will represent a SQL data type in the tables of the IS generated by Skyve, they are: Text, TextArea, TimeField, Timestamp, Integer, Image, File, DateTime, Data, Combo box, Checkbox, Aggregation Association and finally Composition Association. The fields enumerated are the possible choices of fields for selection on a form, each of which has different characteristics and constraints.
- **FormView** - Some DynamicIS users may need to create more customized form views and not just define the fields of a form. This entity was created in order to provide a more advanced means of customization to the Form entity (and is therefore referenced there) either in terms of layout and element arrangement or by adding components other than fields. Thus, this entity contains another entity "For-

mViewObject" which consists of a type of object that can be added to the form view. It should be noted that although it is not represented in the diagram, a "FormViewObject" contains other "FormViewObjects" that extend this class: "VContainer" and "HContainer" which represent vertical and horizontal containers, "Tab", which represents a tab layout, three components which are "HtmlComponent" (block of html code) , "GCalendar" (Google Calendar) and "ActionComponent" (buttons with actions). Also, the entity "FormWidgetView" that represents a form, "FormRow" and "FormColumn" that represent columns and rows of a form and finally "FormRowItem" that represents an item inside a row that can be a field.

- **ResultView** - Besides the Form, a ResultView is another fundamental piece of an information system, because it is the entity that represents the tables with a view of the contents obtained from the forms. Thus, this entity contains another entity, "Query", which represents the content to be made available about a given form in the generated information system. A "Query" contains another entity "Column" which specifies the attribute of the form element to be represented, containing the name that will be visible in the table to the information system user as well as other characteristics. The "Column" entity is an example of the flexibility of the NoSQL schemaless because there are "Column" documents that represent an image and will have more attributes, such as the size of the thumbnail of the image to be displayed, allowing documents of the same type to be different, in other words, a different scheme.

## 4.6 System Implementation

The implementation of DynamicIS started with studying the Play Framework, its capabilities and working methodology. In addition, it was necessary to understand Skyve engine, specifically the mechanism underlying the manipulation of the database structure of the information system, in order to the implementation of a flexible integration between the two platforms.

Overall DynamicIS allows a fast, easy (No-Code) way to implement an Information System based on domain definition through form building but also with more advanced components. Thus, the DynamicIS platform is in charge of creating and storing the metadata concerning the entire composition of the information system, allowing for a quick setup and modification.

That said, in addition to what was covered in the previous sections, in terms of Persistence and Integration with Skyve, this section presents more implementation considerations and how its components work together. In

the next chapter, some screenshots of DynamicIS running with the implementation of the following sections will be presented.

#### 4.6.1 Authentication

Like most applications, DynamicIS needs to recognize the identity of a user in order to secure the data belonging to him and provide that data regardless of the platform with which the user uses DynamicIS.

In order to ensure segregation of responsibilities in a service oriented architecture approach, the choice was made to leverage Google Firebase Authentication, which contains robust components and a back-end to manage the registration flow, login in and log out. Besides the Firebase authentication back-end, it was also taken advantage of FirebaseUI which is an open-source JavaScript library that allows communication through the client with the Firebase back-end.

The first step was to create a project registration for the DynamicIS application, as well as adding its authorized access domain in Firebase console. Then, it was necessary to insert a CDN to obtain the Firebase library in the "main.scala.html" view of the DynamicIS web app template so that, as all the other views extend this view, it will be possible to ensure that the user authentication status is maintained by all the views. Only email and password authentication was used for proof of concept, although Firebase supports multiple providers. The "auth.scala.html" template page is the first to be called when DynamicIS is launched and it is this page that starts the FirebaseUI with the component for the user to log in or register. It is in the FirebaseUI configuration script that callbacks are added after an action. In this case, a callback is added to the DynamicIS back-end controller to create a new user and store a reference to this user (using the ID created by Firebase) in the User entity (explained in the Persistence section). In addition to this callback, another one is triggered in case a user successfully logs in (this check is done through the Firebase back-end) and that calls a method in the DynamicIS controller to retrieve the User's projects and load the User's home page. It should be noted that the callbacks, as they are done from the client, take advantage of the JavaScript routing system provided by PlayFramework in order to call a method from a DynamicIS back-end controller.

#### 4.6.2 Front-end capabilities

DynamicIS platform intends to provide a conceptual model that approaches a PaaS for SaaS development through DynamicIS web app integrated with the Skyve server that allows the deployment of the software (Information System). In this case, the DynamicIS web app is a platform with a No-Code development environment that allows the quick and easy

creation of Information Systems for a broad group of people whether they have programming knowledge or not. Unlike more manual code writing oriented environments, DynamicIS is focused on visual components with drag and drop capabilities while having an intuitive and mobile friendly interface. In DynamicIS, there is no text editor but rather components that generate the text (code and metadata).

On the DynamicIS interface pages it is possible to visualize the content created by a user, whether its projects, inside the projects the modules and in this one the forms and results pages. Besides the visualization of this content, the platform also allows a drag-and-drop reorganization of the content on the same page, and this organization will correspond to changes in the disposition of elements in the information system that will be generated. To provide content from the back-end to the front-end of DynamicIS, the Play Framework's twirl template allows the inclusion of parameters in the template since they work as functions (in this case in Scala language).

The example shown in Listing 4.11, presents a function of the template for the preview page of a User project. The first parameter is the project, an object of type "Project" present in the models folder. The second parameter refers to a list of modules contained in the project of type "module". The third parameter refers to the user ID which is of type "String" (standard Scala and Java data type). All these parameters are loaded from a function present in a back-end controller.

```
@(project: models.Project, modules: Seq[models.Module],  
  userID : String)(implicit request: Request)
```

Listing 4.11: Parameters of "ProjectPage.scala.html" located at the top of template page

The display of the contents in the interface is in general done from an iteration between all the IDs present in the Project. If it is on a Module page, after the iteration, a check is made to distinguish the ID in order to verify if it belongs to a Form, ResultPage or Group and thus associate an action to the respective function in the back-end.

Drag-and-drop is controlled from a JavaScript script that handles and adds a Listener to the occurrence of an event, which in this case is the end of Drag. This event triggers through JSRoutes a function to reorder the items which takes as parameter the IDs of the item that started drag-and-drop and the item that ends it, and makes the respective index updates so that the order is updated. In case the user is in "Group Mode", the script does not reorder the elements but rather groups the items and triggers a different controller function for this purpose.

That said, it is worth mentioning that the front-end operation is dynamic and based on the content loaded from the back-end controllers, and although not very complex (to not overload the client) the logic present in the twirl

template engine allows to create highly flexible pages without resorting to other frameworks.

### 4.6.3 Form and Result page building

Forms are important parts that integrate information systems, as such DynamicIS allows a developer to create them from a graphical interface. In this interface, an index-based approach was used to handle the fields that are dynamically added through the client. Whenever a field is added to a form, it will be identified with an index so that, when the back-end receives the form request, it can distinguish fields as they can be reordered from drag-and-drop and as they can be extended from the parent class "Field". Each field thus has to be processed individually on the backend because they have different properties. For example, a field of the subtype "ComboBox" needs to have features that allow the user to add options to it. After form submission, the backend also validates the form by presenting error messages to the client if the form is not valid.

Just as it is possible to enter information using forms, an information system also needs to allow visualization of the information entered and stored in the database. For this reason, DynamicIS contains a mechanism to create Result Pages with developer-defined queries. The mechanism is similar to that of form fields but in this case instead of fields it is possible to choose columns to be available for viewing in a table associated with the attributes of a specific form. After creating a result page, Couchbase stores the metadata related to the queries and in the project generation phase (described in the Project Generation section) they will be converted into queries recognizable by Skyve.

For all Form and Result pages created from DynamicIS, it provides a link that can be embedded. This can be useful if external services want to access the GUI of the information system and embed it via an iframe but without the navigation menus that are included by default in all Skyve pages. To implement this functionality, it was necessary to make changes to Skyve's default Routing mechanism. In this case, it was necessary to add the code excerpt present in Listing 4.12 to Skyve's "router.xml" file, which directs where to forward the request.

```
<uxui name="public">
  <route outcome="/external/editembed.xhtml" >
    <criteria webAction="e" />
  </route>
  <route outcome="/external/listembed.xhtml" >
    <criteria webAction="l" />
  </route>
</uxui>
```

Listing 4.12: Excerpt of code added to Skyve "router.xml" file

Skyve makes use of JavaServer Faces and XHTML for the definition of client web pages and therefore two pages "editembed.xhtml" and "listembed.xhtml" were created and adapted from the original Skyve "edit" and "list" pages in order to embed only the form and the results. It was also necessary to adapt Skyve's "home.jsp" file because this is where the requests go first. This is where they are analyzed and checked to see if are valid and then the "router.xml" file is called so that the request is redirected to the specified pages. The changes included the addition of the parameter "p" (of the url) for "public" so that the forms that can be embedded are made publically available. In this case, Skyve's routing system will provide a link. For example, a "Medic" form belonging to the "MedicStaff" module of the "dynamicis" project can be accessed in a widget style through the link: `http://{domain}/dynamicis?a=e&m=MedicStaff&d=Medic&p=public`

#### 4.6.4 Advanced Form customization

Besides creating forms by just defining their fields, DynamicIS allows advanced form customization to meet more specific requirements. This customization not only allows to adjust the layout of the form, but also to add more components such as HTML code blocks, actions and calendars. Both, advanced form design and the regular design features of DynamicIS, allow the developer to make no mistakes when building the information system because everything is controlled. The developer defines components through a user interface that imposes rules, unlike Skyve's characteristic low-code that allows several mistakes to the developer and requires some time to get used to metadata specification.

One of the features implemented on the views page was a tree-view style side navigation menu. Given the nature of the organization of objects within a view, these can be of various sub-types and are organized in lists within lists. So, this navigation menu has to be built dynamically in the client using recursion to traverse all objects within objects. For this, the template receives as argument a JSON string which contains the object lists, then a JavaScript script converts the JSON string into a JavaScript object to be manipulated. Next, the HTML elements are created recursively using the method "Element.classList", which allows to manipulate HTML elements in order to create a ListGroup using also Bootstrap classes. Since each element in the menu contains a link to select an object (and each object has different behaviors in the view) it is also necessary to create this link dynamically through back-end calls to a function that goes through all the elements recursively and creates a link for the element to be selected that redirects the user to it when selected.

Calendars and scheduling are features often adopted in information systems. However, in the analyzed platforms, it is not a native feature or they do not have tools that assist the process of creating calendars for the

information system. For this reason, a component was implemented in DynamicIS in the advanced view that allows calendars to be created in an easier way. The choice was to use Google Calendar and its API, a system very used by the community that contains useful synchronization across multiple devices. A user can bind specific fields of a form (one text field, and two of type Date) to automatically create events in the calendar whenever he fills those fields of the form. This way, for example, a doctor can register an appointment by filling out a form in the information system and an event is created in the calendar that can be accessed even from the smartphone (it does not have to be in the information system) as long as the google calendar has the same account associated.

That said, in order for a developer to create a calendar, first needs to create a Google service account. These accounts are a special type of accounts to be used in applications. This way, the application (in this case a DynamicIS project) can make authorized calls to the Google Calendar API. It is also necessary for the developer to create (from the service account console) a credentials JSON file that can be dragged into the DynamicIS interface which automatically associates with the calendar. The developer just has to put the calendar ID and iframe so that it can be embedded. This information is easily accessible and can be found in the settings of a Google calendar as can be seen in Figure 4.8 with the red and green arrows.

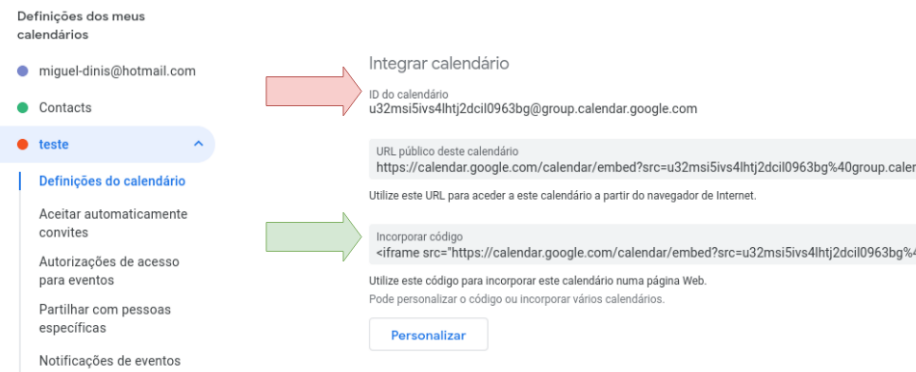


Figure 4.8: ID and iframe present in the settings of a Google calendar

After containing this information, DynamicIS dynamically generates Java code that binds the form fields and creates the events automatically by calling the Google calendar API.

Automatic code generation is a common and important feature in Low-code/No-code platforms and in DynamicIS, besides the code generated for the Google Calendar component, it was also necessary to implement a mechanism for actions that can occur in the client (IS) and that trigger an event in the Skyve server (such as redirection actions). This mechanism was implemented based on the adaption of Skyve functions and metadata. So,

when a project is being generated, DynamicIS will iterate over all the components of a View, and each of these, when necessary, will generate Java files that will be transmitted through the `"/newCustomAction/"` endpoint of the DynamicIS REST API to the a specific directory in the Skyve server project structure.

#### 4.6.5 Project generation

DynamicIS is a system that explores Skyve's Low-code capabilities to create a No-Code platform with more automatic code generation and greater flexibility. However, since each one can work independently, it makes it possible for a system to be developed by two different entities. A domain knowledgeable person such as a doctor can build a form using DynamicIS from his tablet and see the result almost instantly in a fully functional IS, and a developer with more knowledge can later make more specific modifications or adaptations from Skyve using an IDE.

As shown in the Figure 4.9, this integrated system consists of three layers, with the DynamicIS web app (No-code) and DynamicIS Rest API (Communication) created as part of this dissertation proposal. Minor changes were made to the Skyve level for configuration purposes, so any project created with Skyve can be easily connected to DynamicIS in order to be extended.

The components that are part of the DynamicIS web app and participate in the process of generating a project consist of the ProjectController that performs operations on the database from the "CouchbaseService" service, which in turn makes use of Models (Java object classes). The "Project-Controller" also makes use of two auxiliary components that build Skyve's specific metadata classes. The method in charge of generating a project, that transforms what was created in DynamicIS into useful content for Skyve, is the `"generateSkyve()"` and will be explained next.

Before invoking the `generateSkyve()` method, the client first makes a call to the `"testServerConnection()"` method which in turn invokes the `"/test-server"` endpoint of the DynamicIS REST API. If it has a positive response, then the client does the routing to call `generateSkyve()`. It should be noted that the call to `"testServerConnection()"` is also made after a timeout if the Skyve server does not respond and then the client displays a button for the user to restart the operation. That said, `"generateSkyve()"` is composed of 6 stages that must be executed sequentially, and an error in one stage does not allow to proceed to the next one. This staged process is responsible for creating an information system running on the Skyve server from a project created in DynamicIS. Most of the interactions between the two systems occur in the process, via the DynamicIS REST API:

- 1) It starts by executing a function that checks the modules that have been removed since the last generation of the project. It then iterates



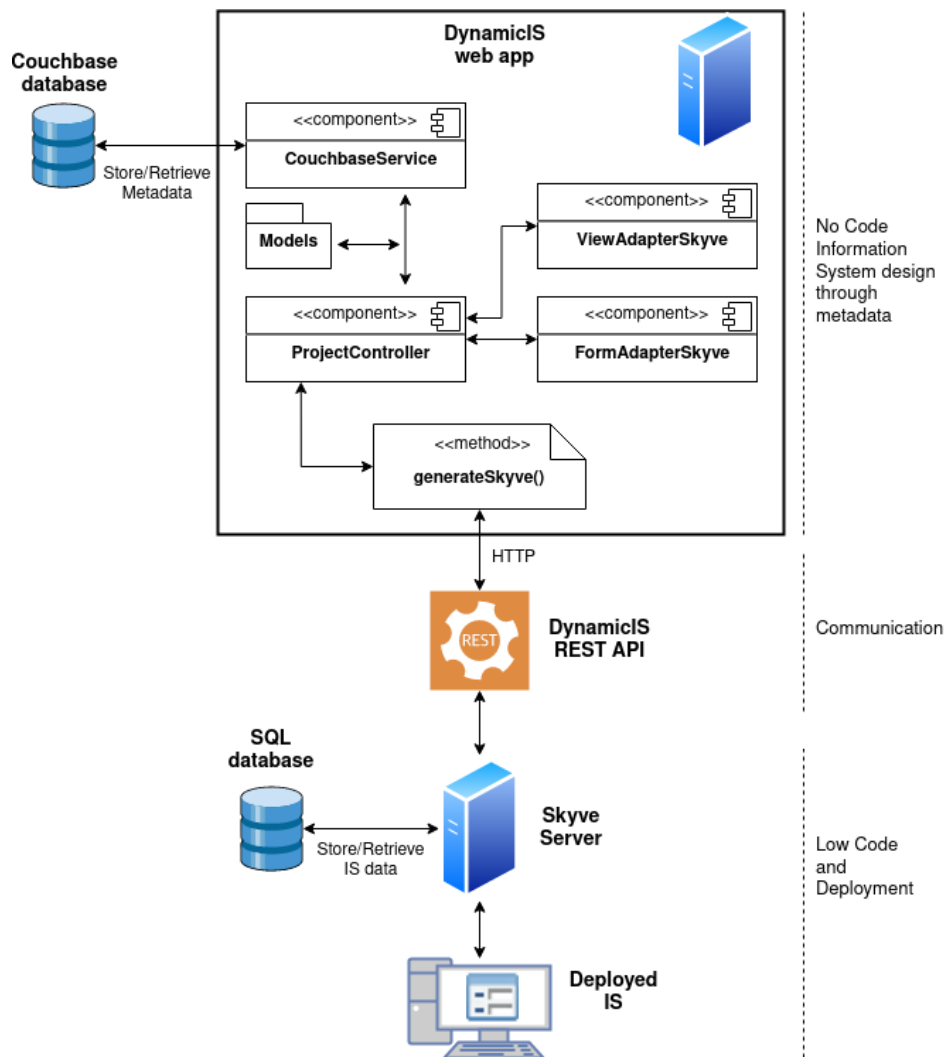


Figure 4.9: Project generation components

over those modules and through the "/removeRole" endpoint, using the Play WebService (WS) library, it communicates with the DynamicIS REST API. Note that the WS was used with the method "toCompletableFuture.get()" in order to make an HTTP asynchronously (non-blocking) but, since it has the .get(), it expects the call to return a response. All DynamicIS REST API calls work using this pattern, so that they execute sequentially. Already in the DynamicIS API, the system will execute a query on the Skyve database in order to eliminate the "Roles" for these modules. This procedure is necessary because Skyve uses roles and does not allow users to access modules if they have no role (they are not authorized). However, if there is a

role in the database for a module that no longer exists, it generates an error. Next, it is necessary to track the modules removed since the last generation of the project, so that a developer using DynamicIS can make changes to the modules without bothering with further configurations. After that, there is a call to the API endpoint `"/cleanmodule/"` in order to delete the directories and files belonging to modules that are no longer used in the Skyve server directory. Whenever a phase ends, the client reloads and updates the project generation progress along with a status message describing the phase procedure.

- 2) It iterates over each module in the project, makes the necessary consistency checks and, within each module, it checks its forms. If a form has association fields to forms (i.e. relationships between tables) from other modules, Skyve needs the metadata file of the module containing that form to reference the other module, so this process is done automatically in DynamicIS. Also, if a form has been customized through a view, another Skyve XML metadata file will be sent via the DynamicIS API endpoint `"/newview"`, as Skyve distinguishes between the two types of forms. This endpoint will send an XML document created with the help of the DynamicIS `"ViewSkyveAdapter"` service, which in general creates a Skyve metadata class corresponding to a view from the metadata stored in DynamicIS, specifically in Couchbase. For regular forms (without advanced customization), the `"FormSkyveAdapter"` service will be used in a similar process to the one explained above, but this time the endpoint for transmitting the XML document will be `"/newdocument"`. Also on a module, the existence of Result Pages is verified. The queries created, corresponding to columns that will be displayed in the information system within a module, are added to the Skyve metadata document. Next, the groups created are verified and added to the module. Finally, being completed the operations on a module, it will be converted into an XML document of Skyve specific metadata and be sent to the Skyve server from the endpoint `"/updatemodule"`.
- 3) Skyve supports multi-tenant SaaS with the possibility to customize an application for each specific customer. For each customer, it needs a metadata XML file with settings and declarations concerning that customer, the modules accessible by it, the module that will be started as home page and other aspects such as the system language. This stage is responsible for creating that XML document and sending it to the Skyve server through the `"/newcustomer"` endpoint.
- 4) This stage is in charge of sending an image that will be the information system's logo. This will appear in the upper left corner above the generated system navigation menu as well as on the default authenti-

cation page. For this, the image will be converted from a sequence of bytes to a String using the Base64 encoding scheme. It will be sent by the endpoint `"/updatelogo"`, and on the API Skyve side, the inverse process of decoding will be done.

- 5) Orders are given to the Skyve server to proceed with the execution of Maven commands via the `"/applymavencommands"` endpoint. Skyve contains Maven targets utilities to assist developers in some processes. These targets can be executed from commands in the DynamicIS API beginning with `"mvn"` which will be executed using the `java.Lang.Runtime` that allows the application to interact with the environment it is running in. In total, 3 commands will be executed sequentially that will generate the information system domain, validate and compile the metadata (XML files), update the directory with the compiled project, restart the Skyve server and deploy the project. At this point, an information system is already deployed with the necessary tables in the relational database.
- 6) The last stage is used to register the last modules generated in the current project so that it is synchronized with the next generation of the project. Also, since the information system is now running, a button will appear on the client so that a user can be redirected to a page with the information system preview.



# Chapter 5

## Results

*This chapter covers the results obtained. Besides screenshots of the DynamicIS App interface, the actions and options available to the user will be briefly described. Throughout the presentation of the interface, an example of an information system will be developed based in the scope of a typical use case for a medical information system. In the final section, the project will be generated in order to validate the solution.*

### 5.1 General aspects and Authentication

Since the DynamicIS app was designed as a No-code platform, it is assumed that it can have a more generalized usage by users already accustomed to the mobile application paradigm. DynamicIS takes into account responsiveness and adaptation to different screen sizes through the layout of its elements and a design based on buttons and intuitive simple interface.

Figure 5.1 shows the initial interface of a DynamicIS workflow and corresponds to the authentication page. The application contains a top bar in blue color that remains on every page, where it is possible to click on the DynamicIS logo to be redirected to the main page containing the user projects, if logged in. In the upper right corner, it is the email address of the logged-in user and a log-out button to clear the entire browser session. It should be noted that, if a user enters an email address that does not exist in the system, the interface presents an option to create an account on the fly. After a user logs in, it will redirect to its homepage.

### 5.2 Projects

A project is defined as the set of all the components that will be integrated in order to originate the information system. After a user logs in, all his previously created projects will be shown with the icon of a folder as shown in Figure 5.2. It is on this same page that a user can create a new

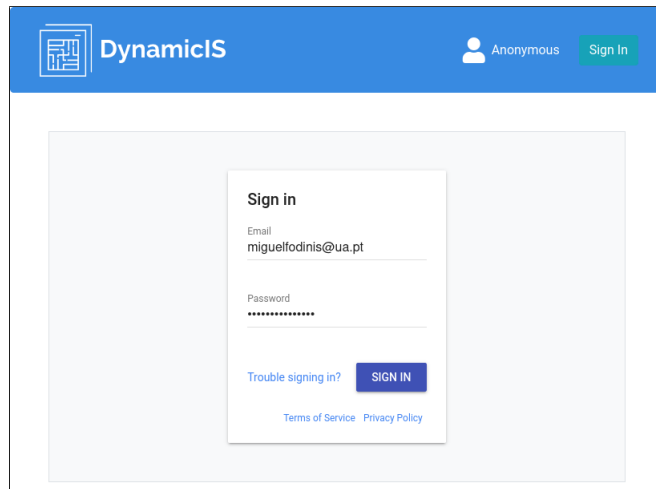


Figure 5.1: Authentication page

project or edit an existing one by clicking on it. If the user wants to create a new one, a new page will be displayed (Figure 5.3) in order to fill in the project information and the respective fields needed for the integration with the Skyve server.

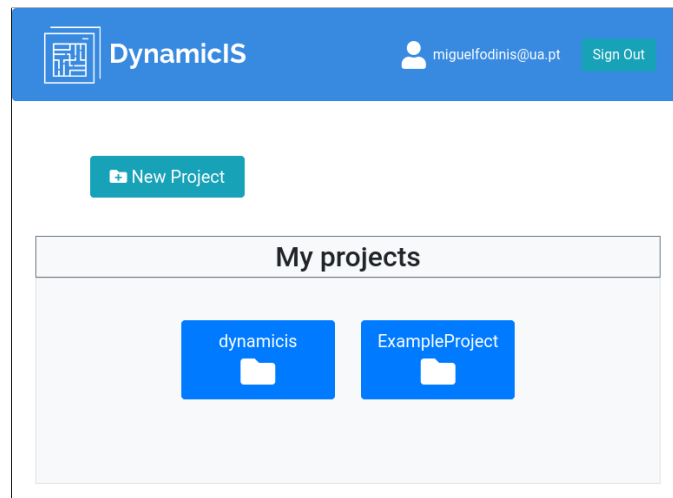
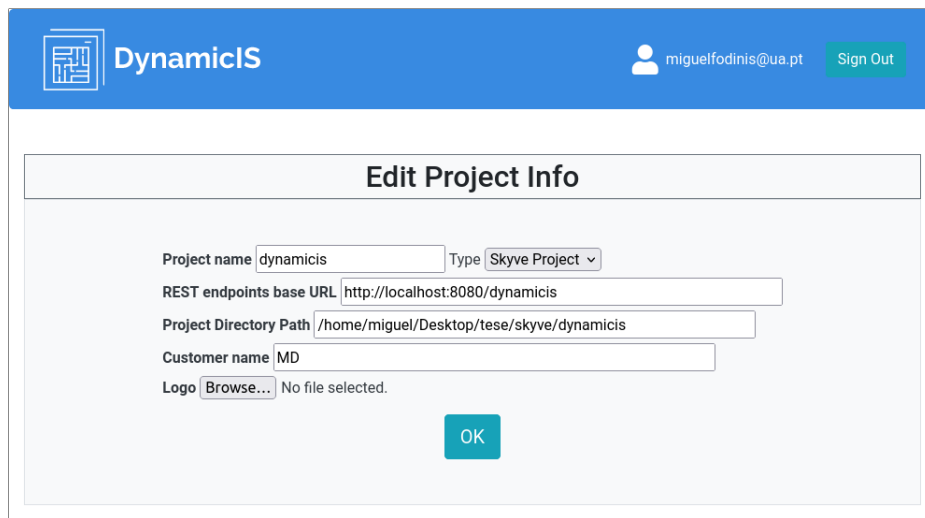


Figure 5.2: List of projects

It is on this page that the base URL for connecting to the DynamicIS REST API is set, as well as the root directory of the Skyve project present on the machine where the server is running. This is also where the choice of the information system logo is made which will be stored in the DynamicIS database, because the transmission of the logo to the Skyve server is only made when the project is generated. Note that all these settings can be

edited at any time on a project page, as will be demonstrated next. This approach to connecting to a Skyve server makes it possible to quickly instantiate a similar project that needs minor changes to another customer from the same project created in DynamicIS, saving a lot of development time in the design of forms and database structure.



The screenshot shows the 'Edit Project Info' form in the DynamicIS application. The form is titled 'Edit Project Info' and contains several input fields: 'Project name' (dynamicis), 'Type' (Skyve Project), 'REST endpoints base URL' (http://localhost:8080/dynamicis), 'Project Directory Path' (/home/miguel/Desktop/tese/skyve/dynamicis), 'Customer name' (MD), and 'Logo' (Browse... No file selected). An 'OK' button is located at the bottom center of the form.

Figure 5.3: New project page

### 5.3 Project page

When a user clicks on a project it will be redirected to that project's page. It is on this page that is showed the existing modules that compose it. It will be also provided functionalities to generate the project, edit project information, create new modules and remove the project (deleting all its data from the database). Figure 5.4 (the top bar has been omitted from the screenshot) shows this page with 3 modules created. Each new module created will be displayed in the "Modules" container and these can be dragged at any time to change their order. The order of the modules changes their positioning in the side navigation menu of the information system (as shown in Figure 5.18) generated by Skyve.

### 5.4 Module page

When a user clicks on a module, it will be redirected to the module page. On this page, it is possible to see all the items that make up the module, such as the Forms, and the Result Pages. The user can edit them at any time or drag them (just like on the modules page) to change their arrangement that

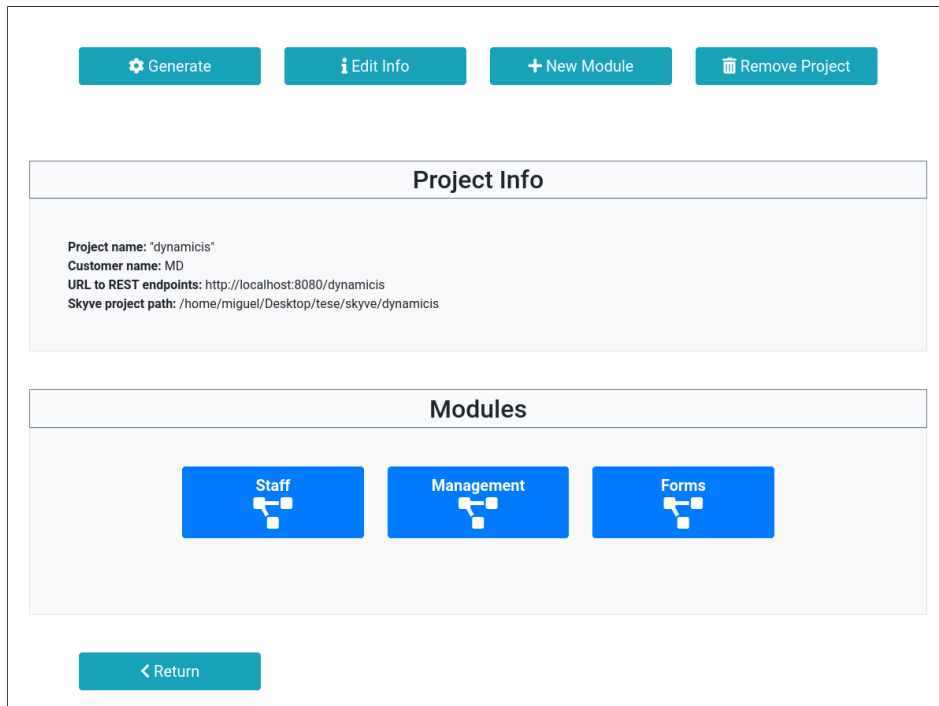


Figure 5.4: Project page

will be reflected in the information system. In addition, it is also possible to group forms and results into groups by clicking on the "Group mode" button, where dragging stops rearranging and switches to grouping mode. Figure 5.5 shows the created "Staff" module containing two forms and a group (in gray) consisting of two results pages. Here, everything is dynamic, and the order, groups and their names can be changed at any time. The "import form" option allows to import a form from another module in a way that allows to quickly change a similar form by importing his content.

## 5.5 Form page

It is on the Form page that the fundamental part of an information system, i.e. the form, is created. A user can add form fields dynamically, even if the form have been submitted and the information system has been generated, because forms can be updated at any time. Figure 5.6 shows the form constructed for the entity "Medic". Each new field created contains 3 generic elements to fill out: the option to add a description (tooltip that appears next to the field in the generated IS), a checkbox to make the field mandatory to fill out, the name of the field. In the DynamicIS platform, it is possible to choose among different types of field (with filling constraints) that present new and specific options. Figure 5.7 shows a field



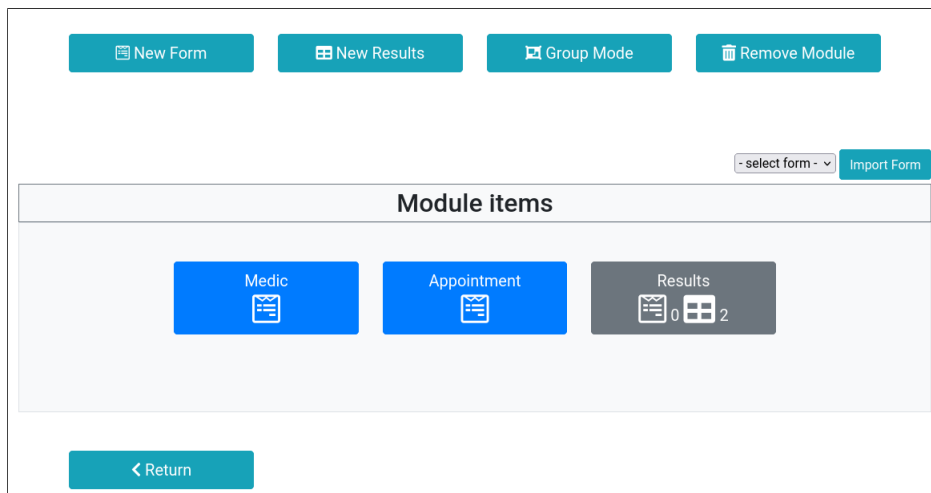


Figure 5.5: Module page

of the association type aggregation of the form "Appointment" for making an association between two forms (corresponding to relations between tables in the information system database). In this case, the "Aggregation" field has two selection boxes available to select from among the available forms previously created. Adding tooltips is done by clicking the "Add Description" option which displays the page shown in Figure 5.8. After the description is added, the icon turns blue to show that a description has already been created. Once again, it is possible to reorder items, in this case fields, by clicking the "Reorder Fields" button, which makes the Remove icon change to a Reorder Icon and makes it possible to drag fields to change their position on the form. A user can change the form icon that will appear in the navigation menu of the information system by setting a Font Awesome icon. Also, as already discussed in previous sections, a link is available so that the form can be embedded in another external website by creating a special widget. One of the goals of DynamicIS is to create an error-free system for the developer. For this reason the entire workflow is protected. As shown in Figure 5.6, if a user clicks the "Create Form" button without having any field selected as "Required", a red warning message will appear informing the user that the form cannot be created and registered in the database (different messages appear, depending on the condition, to ensure a consistent form creation).

This is the simplest and quickest way to define a form template, and is enough to generate a fillable form in the information system, with a layout where each field take up one line. If a user wants a more advanced customization with other components than just fields, can do this by clicking on the "Advanced Mode" button in order to be redirected to the page presented in Figure 5.9 where a view for the form can be designed.

Figure 5.6: Form page

Figure 5.7: Field of type Association Aggregation in "Appointment" form

Here the user can organize the items of a view into containers (Figure 5.10) that automatically arrange elements inside them vertically or horizontally, and can also place containers in separate Tabs. There are two possible ways to navigate over the items in the view, either through the "Navigation

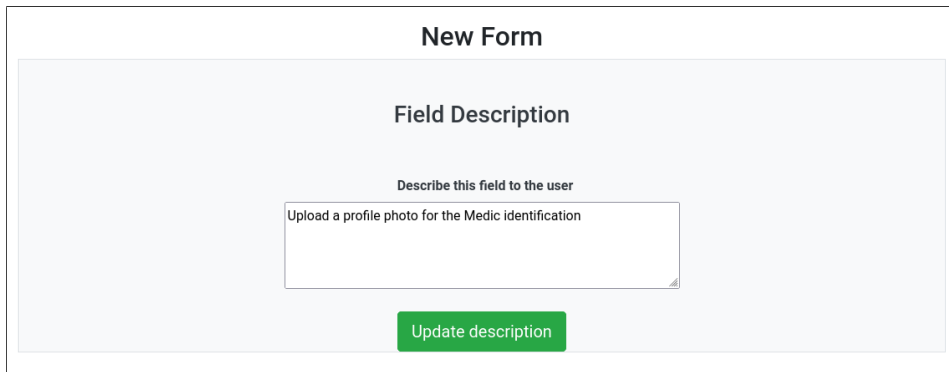


Figure 5.8: Form field description page

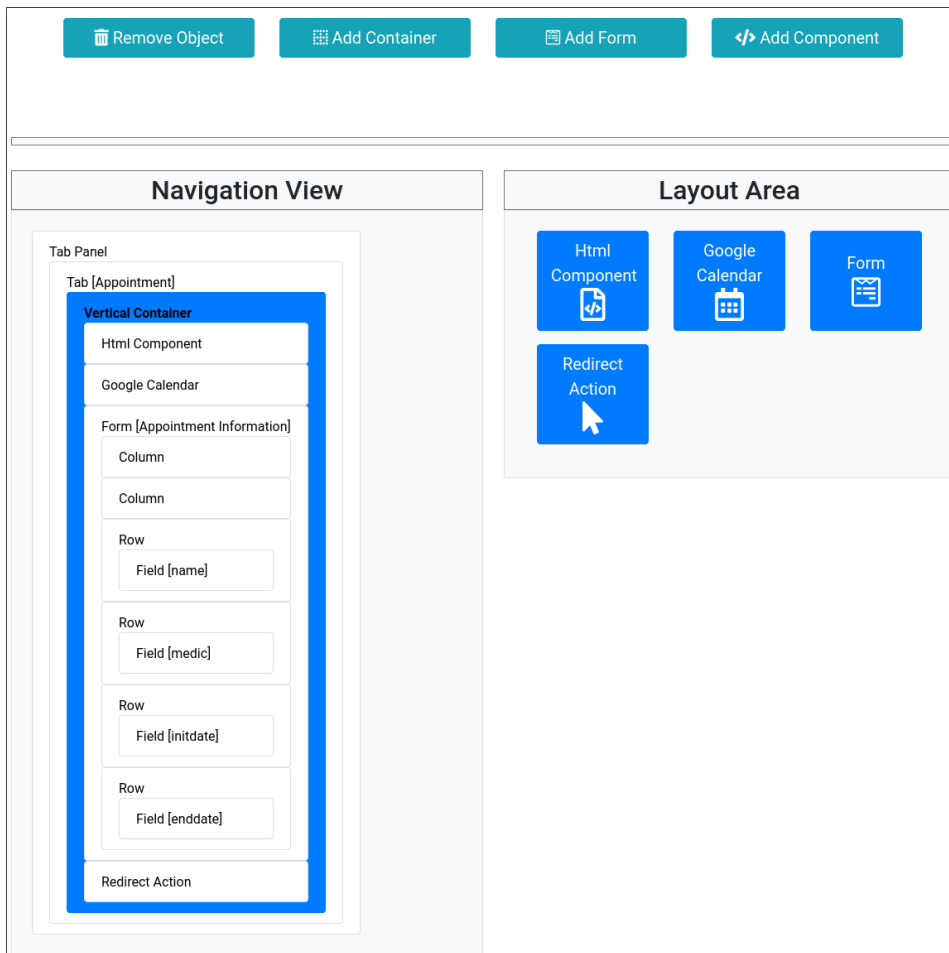


Figure 5.9: View for advanced form customization

View" menu, which allows to see the complete structure of the View and where the item currently selected is in blue, or through the "Layout Area" that allows to browse the items contained within the currently selected item with the possibility to select an item to change its settings, such as the alignment of a form's fields, the option to add a border and a name to the form, etc. Note that the button options that are at the top differ depending on the item currently selected by the user, for example it is only possible to place fields inside a form, so this option button is not shown as the user in the screenshot of Figure 5.9 has a "vertical container" item selected.

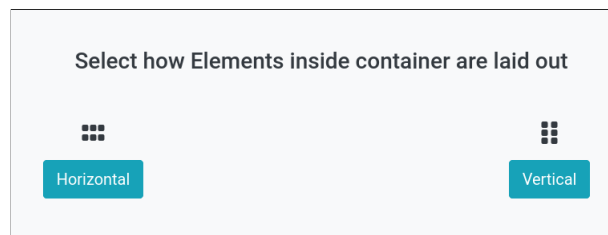


Figure 5.10: View Container dialogue

Also, on this customization page, components can be added in addition to the forms (Figure 5.11) in order to give extra capabilities to the information system. As mentioned before, the ability to support calendars is a very recurring feature in information systems and DynamicIS contains a mechanism to facilitate this implementation through the Google Calendar Component (Figure 5.12). On the Calendar component page, in addition to a user filling out information to associate with a Google calendar, it also has selection boxes to associate compatible form fields with the creation of calendar events automatically as a user in the information system fills out these fields. The HTML block component allows more advanced users to add HTML or even JavaScript snippets to the page, these will automatically be placed in a `<div>` in the generated information system interface. Figure 5.13 shows the page of the component that allows this, where it is possible to drag a file with the code that will be automatically placed in the text area on the left and can be edited there. Finally, it is possible to add an action, i.e. a button that triggers a certain action when clicked by a user of the information system. In this case, it is shown in Figure 5.14, the "Redirect Action" that allows, after saving the contents of a form, to redirect the user to another form page in order to create form filling flows. Here a user has the option to choose a dialog message that appears after clicking the button, the text that appears on the button as well as the icon and the redirection form, within the existing ones in the project.

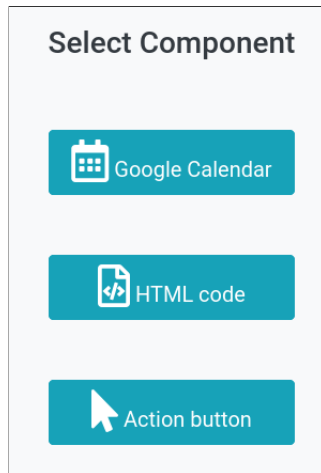


Figure 5.11: Select component dialogue

 A configuration page titled "Google Calendar component" with a light gray background. It is divided into two main sections: "Calendar info" on the left and "Drag credentials JSON file" on the right.
   
 In the "Calendar info" section:
 

- "Calendar Name" is a text input field containing "AppointmentCalendar".
- "Google Calendar id" is a text input field containing "group.calendar.google.com".
- "Fields to bind events" section contains three dropdown menus: "Event name" (selected "Name"), "Init date" (selected "InitDate"), and "End date" (selected "EndDate").
- "Google calendar iframe (leave empty if not to show)" is a text area containing the following code:
 

```
<iframe src="https://calendar.google.com/calendar/embed?src=u32msi5ivs4lhtj2dcil0963bg%40group.calendar.goo
```

 In the "Drag credentials JSON file" section:
 

- A dashed rectangular box indicates a drop zone. Inside the box, the text "skyve-1619382504045-460a153c2ce7.json" is visible.

 At the bottom center of the page is a teal button labeled "Add Calendar".

Figure 5.12: Google calendar component page

## 5.6 Result page

Once the information has been entered into the information system by filling out forms, it is necessary to have pages with tables to make this information available in an organized way. In DynamicIS, these pages are created through Result Pages. The configuration of a Result Page is shown in Figure 5.15 to demonstrate the results of the entity "Appointment". The

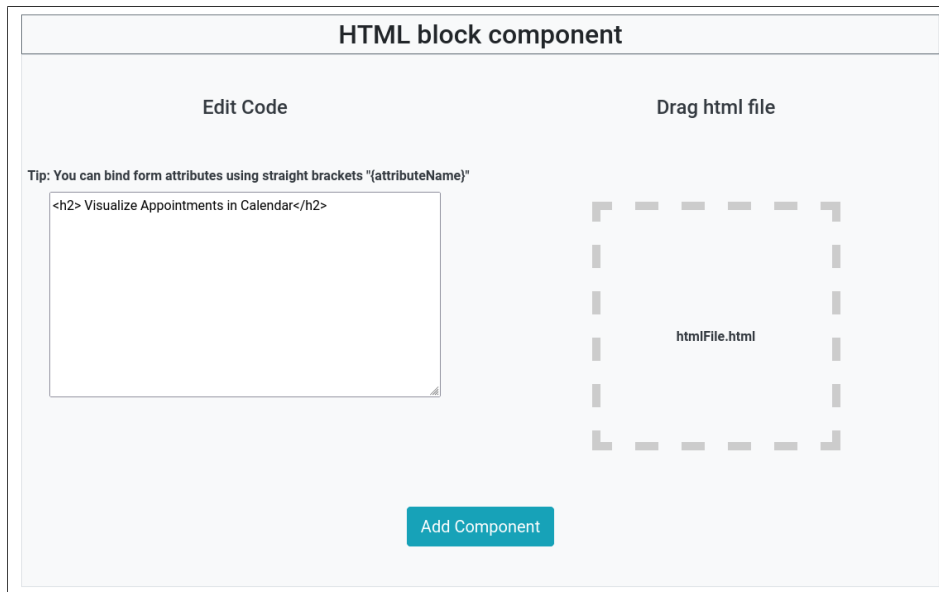


Figure 5.13: HTML component page

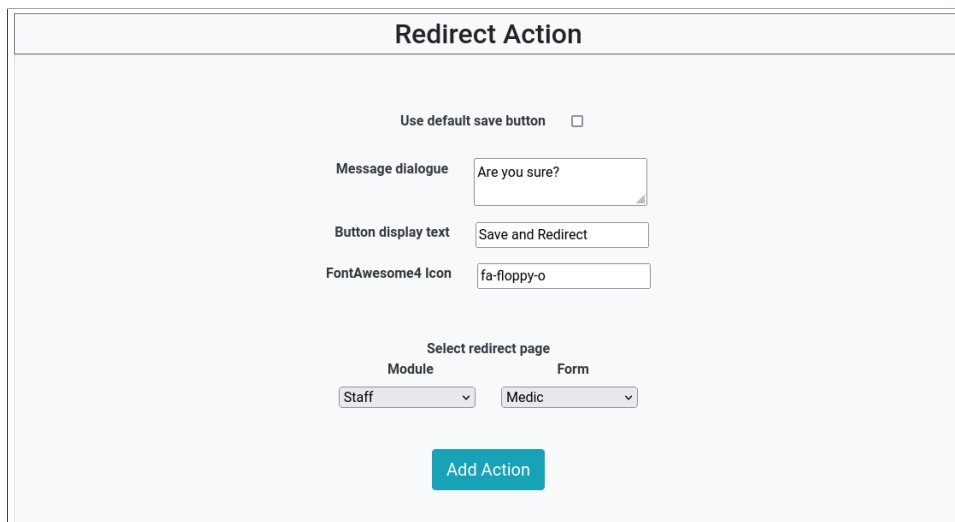


Figure 5.14: Redirect action component page

configuration is done by choosing the associated form and therefore the fields that will be shown in the result table (from the fields that compose the form). If a user does not click on the "Add Query" option, DynamicIS will automatically create a result page that simply shows a table with all the fields that compose the form with default settings.

If a user wants to choose specific fields, as shown in Figure 5.16 for the entity "Medic", it is possible in some fields, such as the image type, to set

The screenshot shows a configuration form titled "New Results". It has two columns. The left column contains a text input for "Results name" with the value "AppointmentResults" and a dropdown menu for "Forms" with the value "Appointment". The right column contains a text input for "FontAwesome4 Icon" with the value "fa-user-o" and a note "None for default Query (all form fields except associations)". At the bottom right is a blue button labeled "Add Query".

Figure 5.15: Result page configuration with default query

more different configurations.

The screenshot shows a configuration form titled "New Results" for the "Medic" entity. It has two columns. The left column contains a text input for "Results name" with the value "MedicResults" and a dropdown menu for "Forms" with the value "Medic". The right column contains a text input for "FontAwesome4 Icon" with the value "fa-user-o". Below these are four rows of attribute configurations, each with a "Remove" button (trash icon), an "Attribute" dropdown, and a "Display name" input. The attributes are: "Name" (display name "Medic Name"), "Phone Number" (display name "Phone Number"), "Gender" (display name "Gender"), and "[image\_field]-Photo" (display name "Medic Photo", view "thumbnail", height "80", width "80"). At the bottom center is a blue button with a "+" sign, and at the bottom right is a blue button labeled "Create Results".

Figure 5.16: Result page for the "Medic" entity with a configured query

## 5.7 Test and validation

After a user is satisfied with solution designed and wants to create the information system from the project, it can do so by clicking on the "Generate Project" button that is located on a project homepage. It will be redirected to the page shown in Figure 5.17, which contains a progress bar and displays messages informing of the operations performed on the Skyve server that communicates through the DynamicIS REST API. At the end of the

generation, an option to run the project appears, which is made available to preview through an iframe embedded in DynamicIS web app.

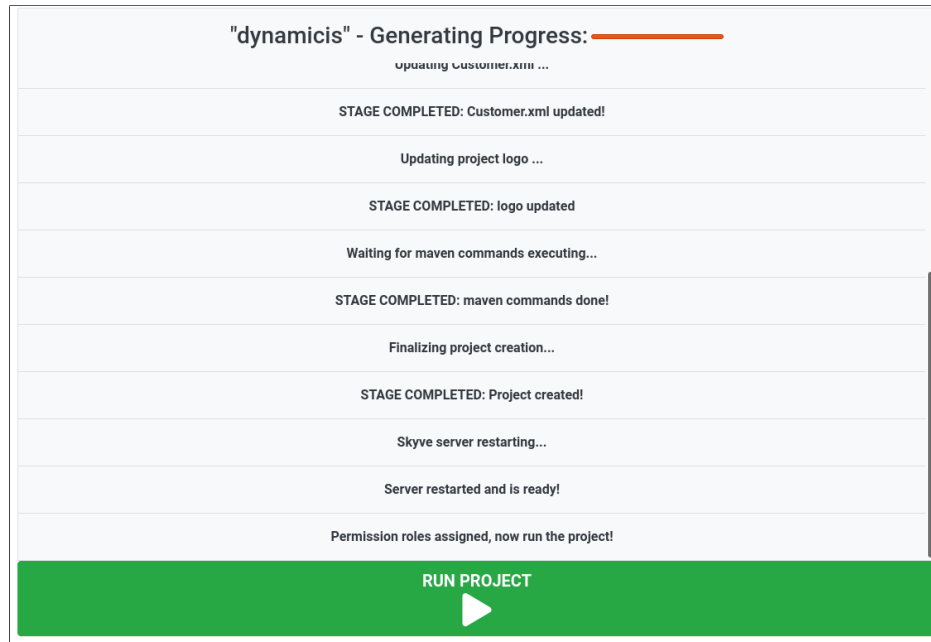


Figure 5.17: Generate project page

It is important to note that the process of project generation in an information system is relatively fast, for a first startup with new forms where it is necessary for Skyve to create new tables in the database, it takes an average of 35 seconds from the start of the generation until the solution availability. A next generation of a project with only form changes, such as adding a new field or other minor updates, only takes an average of 27 seconds. These metrics are important because, in a production, information system they mean how long the system is down and can not be used.

While the changes are being made in the DynamicIS web app, the information system is running normally as the two systems work independently, also allowing for greater security in the information stored in the database. Another significant aspect is the fact that as DynamicIS is oriented to No-Code and implicitly to the use of less experienced developers. The removal of a form does not imply the removal of a table, nor of its information in the information system's database, it only implies system changes in the information system user interface. This way, the administration of the database can be left to more experienced developers not allowing errors and even worse, the loss of important data.

That said, the next screenshots are the actual interface of the proof-of-concept information system created with the DynamicIS web app. Figure 5.18 demonstrates the side navigation menu, containing at the top the logo



that was loaded when the project was created, in blue the tabs corresponding to the modules created, and in the Staff module it is also possible to see the "Results" group created to group all the results pages, forming a sub-section within the module.

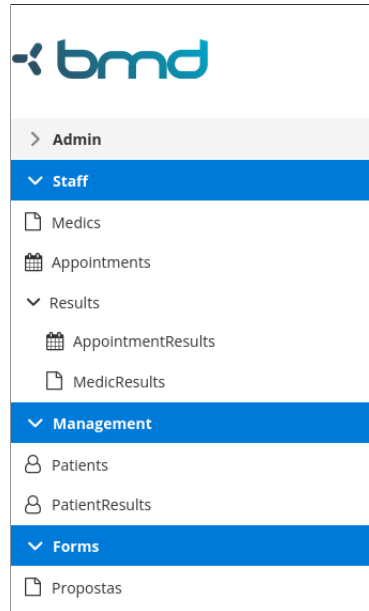


Figure 5.18: Side navigation menu containing the structure of the IS

Figure 5.19 demonstrates the results page for the entity/form "Medic" with a table and a record that was filled in through the form for demonstration. Here can be seen the fields defined previously in the DynamicIS interface (Figure 5.16) for visualization.

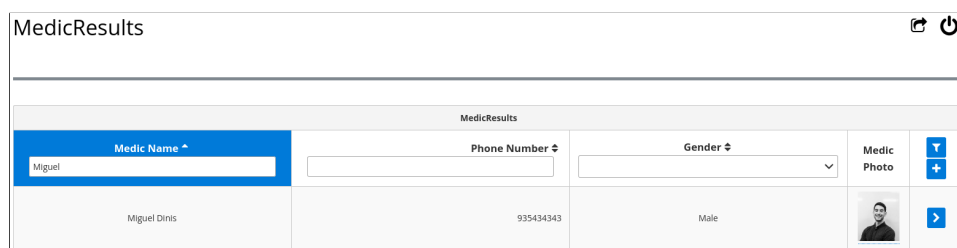


Figure 5.19: Result page generated for the IS

The result of generating the defined advanced view (Figure 5.9) in the information system is shown in Figure 5.20 which contains the calendar component as well as the created form. In this page, a user fills in the fields of the "Appointment Information" and, after clicking the "Register Appointment" button, an event will be automatically created in the calendar from that information.

Figure 5.20: Advanced view created for "Appointment" form in the IS

Another form was created from the DynamicIS web app contained within the "Forms" module to demonstrate an example of forms with a different layout that was quickly customized through the advanced view. Figure 5.21 demonstrates the result interface in the generated information system.

Figure 5.21: Example of generated form with different field arrangements.

## Chapter 6

# Conclusion

*This chapter is allusive to the conclusion, presenting a summary, as well as final considerations and work that could be done in the future within the topic of this dissertation.*

### 6.1 Final considerations

Today we are witnessing a rapid technological transformation of software development methods. This is caused largely due to the advent of the internet and cloud computing, and many companies are taking advantage of these capabilities to provide tools and platforms that are more intuitive, faster and accessible to a wider range of users. Information systems development is one of the sectors that benefit most from these platforms, with more and more data being generated and consumed around the world. It is necessary to create systems that are more capable, scalable, developed in a fast way, and most importantly, that provide features that make them more flexible and dynamic, adaptable to the constant demand for different requirements.

In this sense, the fast-growing Low-code and No-code platforms coupled with new approaches to databases are the solutions that best enable development adapted to current needs. They allow development time to be shortened by reducing the writing of code and boilerplate code through automatic generation mechanisms, following the motto of not reinventing the wheel. They are allowing more individuals with less experience to develop robust solutions at lower cost and with increased flexibility.

Nowadays, open-source platforms emerge as convenient and capable solutions, although few really offer a No-code environment without the need to write and learn some type of code, as these features are more exploited on proprietary platforms.

It was with the dynamism offered by Low-code solutions in mind that the DynamicIS platform was created. This platform allows the development of

a complete information system by defining its fundamental basis, the forms. Thus the entire development environment is No-code based on buttons and drag-and-drop allowing a domain knowledgeable person with no programming experience to turn his ideas into a ready-to-use information system, in a short time and in a very intuitive way through any device running a web browser. At its core, DynamicIS contains a dynamic metadata storage mechanism that can be displayed and organized as a template that can be updated by the developer at any time. DynamicIS was developed as a proof of concept in strict integration (but at the same time independent) with the open-source Skyve platform, a Low-code solution, taking advantage of its robust work at the level of metadata processing in the database and production of the final information system.

Without a doubt, all this evolution has proven to be very interesting, but we must take into account that although these types of platforms contain more and more capabilities, they still present limitations at the level of more complex requirements, and therefore, more classical approaches by developers with programming experience are still necessary.

## 6.2 Future Work

This work constitutes the basis of a platform capable of creating a simple information system, and it is structured in a way that it can be extended to create more complex components to integrate the range of its capabilities. Something interesting that could be a great addition to this work, would be to integrate a No-Code mechanism to create more complex workflows in which the developer could design logic and actions that would occur in the system if a certain event happened. Another capability that would enrich this platform would be a mechanism to automatically learn database schemas from existing systems, i.e. by connecting DynamicIS to a system currently in production, it would provide all the form structures automatically so that they could be easily modified.

Finally, it would provide extra value to the solution, the upgrade to a fully self-contained platform, that would run in a cloud environment, without the need for the developer to instantiate external servers to deploy the information system.

# References

- [1] R. Kadia. “Issues Encountered in Building a Flexible Software Development Environment: Lessons from the Arcadia Project”. In: *SIGSOFT Softw. Eng. Notes* 17.5 (Nov. 1992), pp. 169–180. ISSN: 0163-5948. DOI: 10.1145/142882.143768. URL: <https://doi.org/10.1145/142882.143768>.
- [2] Paul Vincent et al. *Magic Quadrant for Enterprise Low-Code Application Platforms*. 2019.
- [3] Vladimir Zwass. *Information system*. *Encyclopedia Britannica*. URL: <https://www.britannica.com/topic/information-system>.
- [4] Nan Niu, Li Da Xu, and Zhuming Bi. “Enterprise Information Systems Architecture—Analysis and Evaluation”. In: *IEEE Transactions on Industrial Informatics* 9.4 (2013), pp. 2147–2154. DOI: 10.1109/TII.2013.2238948.
- [5] Paulo Neto. *Demystifying Cloud Computing*.
- [6] Peter Mell and Tim Grance. *The NIST Definition of Cloud Computing*. 2011. URL: <https://csrc.nist.gov/publications/detail/sp/800-145/final>.
- [7] Sagar Tambe. *What Are the Types of Cloud Models?* Apr. 2019. URL: <https://www.cuelogic.com/blog/3-types-of-cloud-computing-services>.
- [8] Desislava Ivanova, Plamenka Borovska, and Stefan Zahov. “Development of PaaS using AWS and Terraform for medical imaging analytics”. In: vol. 2048. Dec. 2018, p. 060018. DOI: 10.1063/1.5082133.
- [9] Joe McKendrick. *Low-code and no-code prepare enterprises for an 'unknown future'*. Mar. 2021. URL: <https://www.zdnet.com/article/low-code-and-no-code-have-evolved-very-quickly-over-the-past-year/>.
- [10] *Gartner Glossary for Citizen Developer*. URL: <https://www.gartner.com/en/information-technology/glossary/citizen-developer>.

- [11] Cliff Saran. *Covid-19 drives growth in low-code development tools*. Feb. 2021. URL: <https://www.computerweekly.com/news/252496408/Covid-19-drives-growth-in-low-code-development-tools>.
- [12] Roberto Torres. *Low code helped put out COVID-19 fires. How can leaders sustain momentum?* June 2020. URL: <https://www.ciodive.com/news/low-code-implementation-coronavirus-2020/580024>.
- [13] Apurvanand Sahay et al. “Supporting the understanding and comparison of low-code development platforms”. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2020, pp. 171–178. DOI: 10.1109/SEAA51224.2020.00036.
- [14] Agnes Hallberg. “Using Low-Code Platforms to Collect Patient-Generated Health Data: A Software Developer’s Perspective”. In: (2021).
- [15] Johan den Haan. *Low-Code Principle #1: Model-Driven Development, The Most Important Concept in Low-Code*. Jan. 2020. URL: <https://www.mendix.com/blog/low-code-principle-1-model-driven-development>.
- [16] Kristi Berg, Dr. Tom Seymour, and Richa Goel. “History Of Databases”. In: *International Journal of Management & Information Systems (IJMIS)* 17 (Dec. 2012), p. 29. DOI: 10.19030/ijmis.v17i1.7587.
- [17] *What is a Relational Database (RDBMS)?* URL: <https://www.oracle.com/database/what-is-a-relational-database/>.
- [18] *ACID properties of transactions*. URL: <https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions>.
- [19] João Lourenço et al. “NoSQL Databases: A Software Engineering Perspective”. In: *Advances in Intelligent Systems and Computing* 353 (Jan. 2015), pp. 741–750. DOI: 10.1007/978-3-319-16486-1\_73.
- [20] Cristofer Zdepski, Tarcizio Alexandre Bini, and S. N. Matos. “An Approach for Modeling Polyglot Persistence”. In: *ICEIS*. 2018.
- [21] Tariq N. Khasawneh, Mahmoud H. AL-Sahlee, and Ali A. Safia. “SQL, NewSQL, and NOSQL Databases: A Comparative Survey”. In: *2020 11th International Conference on Information and Communication Systems (ICICS)*. 2020, pp. 013–021. DOI: 10.1109/ICICS49469.2020.239513.
- [22] Antonios Makris et al. “A Classification of NoSQL Data Stores Based on Key Design Characteristics”. In: *Procedia Computer Science* 97 (2016). 2nd International Conference on Cloud Forward: From Distributed to Complete Computing, pp. 94–103. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2016.08.284>. URL: <https://doi.org/10.1016/j.procs.2016.08.284>.

<https://www.sciencedirect.com/science/article/pii/S1877050916321007>.

- [23] Antonio Celesti et al. “An OAIS-Based Hospital Information System on the Cloud: Analysis of a NoSQL Column-Oriented Approach”. In: *IEEE Journal of Biomedical and Health Informatics* 22.3 (2018), pp. 912–918. DOI: 10.1109/JBHI.2017.2681126.
- [24] Roman Čerešňák and Michal Kvet. “Comparison of query performance in relational a non-relation databases”. In: *Transportation Research Procedia* 40 (2019). TRANSCOM 2019 13th International Scientific Conference on Sustainable, Modern and Safe Transport, pp. 170–177. ISSN: 2352-1465. DOI: <https://doi.org/10.1016/j.trpro.2019.07.027>. URL: <https://www.sciencedirect.com/science/article/pii/S2352146519301887>.
- [25] Cornelia A. Györödi et al. “Performance Analysis of NoSQL and Relational Databases with CouchDB and MySQL for Application’s Data Storage”. In: *Applied Sciences* 10.23 (2020). ISSN: 2076-3417. DOI: 10.3390/app10238524. URL: <https://www.mdpi.com/2076-3417/10/23/8524>.
- [26] Kamil Kolonko. “Performance comparison of the most popular relational and non-relational database management systems”. In: (2018).
- [27] Roxana Sotoc Cornelia Györödi Robert Györödi. “A Comparative Study of Relational and Non-Relational Database Models in a Web-Based Application”. In: (2015).
- [28] Jannatul Maowa. “A Comparative Study on Big Data Handling Using Relational and Non-Relational Data Model”. In: (2017).
- [29] Friedrich Gessert F Wingerath W. “NoSQL database systems: a survey and decision guidance.” In: (2017).
- [30] Matthew Aslett. *What we talk about when we talk about NewSQL*. Apr. 2011.
- [31] Pwint Phyu Khine and Zhaoshun Wang. “A Review of Polyglot Persistence in the Big Data World”. In: *Information* 10.4 (2019). ISSN: 2078-2489. DOI: 10.3390/info10040141. URL: <https://www.mdpi.com/2078-2489/10/4/141>.
- [32] Jiaheng Lu and Irena Holubová. “Multi-Model Databases: A New Journey to Handle the Variety of Data”. In: *ACM Comput. Surv.* 52.3 (June 2019). ISSN: 0360-0300. DOI: 10.1145/3323214. URL: <https://doi.org/10.1145/3323214>.
- [33] Margus Jäger et al. “Flexible Database Platform for Biomedical Research with Multiple User Interfaces and a Universal Query Engine.” In: vol. 187. Jan. 2008, pp. 301–310. DOI: 10.3233/978-1-58603-939-4-301.

- [34] Meredith A. Barnes, Mathys C. du Plessis, and Brenda Scholtz. “Toward Database Inference by GUI Analysis: A Case Study”. In: *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*. SAIC-SIT '10. Bela Bela, South Africa: Association for Computing Machinery, 2010, pp. 346–349. ISBN: 9781605589503. DOI: 10.1145/1899503.1899542. URL: <https://doi.org/10.1145/1899503.1899542>.
- [35] Kunal Malhotra, Shibani Medhekar, and Shamkant Navathe. “Towards a Form Based Dynamic Database Schema Creation and Modification System”. In: vol. 8484. June 2014. ISBN: 978-3-319-07880-9. DOI: 10.1007/978-3-319-07881-6\_40.
- [36] John F. Roddick. “Schema Evolution in Database Systems - An Annotated Bibliography”. In: *SIGMOD record* 21.4 (1992), pp. 35–40.
- [37] Carlo Curino, Hyun J. Moon, and Carlo Zaniolo. “Automating Database Schema Evolution in Information System Upgrades”. In: *Proceedings of the 2nd International Workshop on Hot Topics in Software Upgrades*. HotSWUp '09. Orlando, Florida: Association for Computing Machinery, 2009. ISBN: 9781605587233. DOI: 10.1145/1656437.1656444. URL: <https://doi.org/10.1145/1656437.1656444>.
- [38] Stefanie Scherzinger and Sebastian Sidortschuck. “An Empirical Study on the Design and Evolution of NoSQL Database Schemas”. In: Oct. 2020, pp. 441–455. ISBN: 978-3-030-62521-4. DOI: 10.1007/978-3-030-62522-1\_33.
- [39] Stefanie Scherzinger, Thomas Cerqueus, and Eduardo Cunha De Almeida. “ControVol: A framework for controlled schema evolution in NoSQL application development”. In: *2015 IEEE 31st International Conference on Data Engineering*. 2015, pp. 1464–1467. DOI: 10.1109/ICDE.2015.7113402.
- [40] Joe McKendrick. *Evolutionary Database Design*. May 2016. URL: <https://martinfowler.com/articles/evodb.html>.
- [41] John R Rymer et al. *The forrester wave™: Low-code development platforms for ad&d professionals, q1 2019*. 2019.
- [42] Cory Nance et al. “Nosql vs rdbms-why there is room for both”. In: (2013).
- [43] Harley Vera et al. “Data modeling for NoSQL document-oriented databases”. In: *CEUR Workshop Proceedings*. Vol. 1478. 2015, pp. 129–135.