INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Pacote de Umbraco para configurar conteúdo para um cliente móvel

TIAGO RAFAEL PINTO LACERDA
Junho de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Umbraco Package to Configure Content for a Mobile Client

Tiago Lacerda

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Cybersecurity and Systems Administration

Supervisor: Dr. Paulo Gandra de Sousa

Evaluation Committee:
President:
Dra. Isabel Sampaio, Professor, DEI/ISEP

Members:
Dra. Isabel Azevedo, Professor, DEI/ISEP
Dr. Paulo Baltarejo Sousa, Professor, DEI/ISEP

Porto, June 29, 2024

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

Therefore the work presented in this document is original and authored by me, having not previously been used for any other end.

I further declare that I have fully acknowledged the Code of Ethical Conduct of P.PORTO.

ISEP, Porto, June 29, 2024

Dedicatory

I dedicate this thesis to my late grandmother, with the hope of being the first of many grandchildren to achieve a Master's degree.

Abstract

This thesis aims to address the demand of digitalization by developing a package capable of being installed in any Umbraco project that will allow users to create and customise their desired mobile app. That structure is then extracted in such a way that a code generator is able to create a cross-platform mobile solution that will give developers a jump start to customize the solution according to more specific needs, trying to achieve not only the replication of the existing functionalities in the website, but also to introduce specific features that take advantage of components available exclusively on mobile devices. This initiative responds to client demands for rapid mobile app development from existing Umbraco websites, while also aiming to enrich the Umbraco community with enhanced functionalities. The systematic mapping review conducted aimed to consolidate comprehensive knowledge of Umbraco, content management systems, packages, and elements of code generation. A proof of concept was successfully developed, compatible with any Umbraco website version 12 or higher. This package enables users to customize mobile applications directly within the Umbraco backoffice interface, facilitating generation with minimal effort. Agap2IT supports continuous integration pipelines, enabling ongoing mobile app generation. The initial goal was to develop both the package and the code generator within this project. However, because of the approaching deadlines and the effect on our submission for the Umbraco Awards, the company decided to form a team of two developers. One developer would handle the code generator, while I would be responsible for creating and distributing the Umbraco package, which will be the primary focus of this thesis. This project has the potential to make cross-platform mobile app development more accessible, empowering users with no specialized programming skills to leverage the familiar Umbraco Backoffice environment for app creation.

Keywords: umbraco, content management system, packages, plugins, mobile app generation

Resumo

Na era digital atual, as empresas precisam de inovar constantemente para se manterem competitivas. A digitalização oferece inúmeras oportunidades para melhorar a eficiência operacional, aumentar o interesse do cliente e explorar novos mercados. No entanto, desenvolver e manter aplicações móveis é desafiante e caro, especialmente para empresas que já investiram significativamente nos seus websites. É aqui que o Umbraco, um sistema de gestão de conteúdos popular e flexível, entra em cena.

O Umbraco é amplamente utilizado pela sua interface amigável e flexibilidade. A criação de um pacote que permite a geração de aplicações móveis a partir de sites Umbraco atende a uma necessidade crescente de soluções rápidas e eficientes de desenvolvimento de aplicações. Este pacote não só beneficia os clientes da empresa, mas também a comunidade Umbraco, promovendo a inovação e a colaboração.

Para desenvolver a solução proposta, foi realizada uma revisão sistemática do conhecimento existente sobre Umbraco, sistemas de gestão de conteúdo, pacotes e geração de código. A revisão sistemática envolveu a análise de documentos técnicos, casos de estudo e a exploração de pacotes existentes no ecossistema Umbraco.

Após esta revisão, a equipa, composta por dois indivíduos, focou-se no desenvolvimento, adotando a metodologia Scrum e organizando o trabalho em sprints para garantir um progresso contínuo e bem estruturado. Este método ágil permitiu a adaptação rápida a novos insights e desafios, mantendo o foco nos objetivos principais do projeto.

A prova de conceito desenvolvida demonstrou a viabilidade da solução. O pacote pode ser instalado em qualquer site Umbraco versão 12 ou superior, e oferece uma interface intuitiva no backoffice para a personalização de aplicações móveis. Os utilizadores podem definir a aparência e o comportamento das aplicações diretamente na interface do Umbraco, sem a necessidade de conhecimentos técnicos avançados.

Uma das funcionalidades do pacote é a integração com pipelines de integração contínua, que permite a geração automática de aplicações móveis. Isto reduz significativamente o tempo e o esforço necessários para criar e manter aplicações móveis, permitindo que as empresas se concentrem em melhorar a experiência do utilizador e expandir as suas ofertas digitais.

O principal desafio enfrentado durante o desenvolvimento do PoC foi a restrição de tempo. Com mais tempo, seria possível incorporar mais funcionalidades e melhorar o design das aplicações móveis geradas. No entanto, mesmo com as limitações, o PoC demonstrou claramente o potencial da solução proposta.

Este projeto tem o potencial de revolucionar a maneira como as empresas desenvolvem aplicações móveis. Ao simplificar o processo de geração de aplicações e torná-lo acessível a um público mais amplo, o pacote para Umbraco pode reduzir os custos e o tempo associados ao desenvolvimento de aplicações móveis. Além disso, ao integrar a geração de aplicações móveis com o backoffice do Umbraco, as empresas podem gerir de maneira centralizada

os seus conteúdos web e móveis, melhorando a eficiência operacional e a consistência da marca.

A contribuição deste trabalho para a comunidade Umbraco também é significativa. Ao disponibilizar um pacote que facilita a criação de aplicações móveis, este projeto incentiva a inovação e a colaboração dentro da comunidade, fortalecendo ainda mais o ecossistema Umbraco.

Em suma, esta tese apresenta uma solução inovadora que aborda a necessidade urgente de digitalização nas empresas, focando na criação de um pacote para Umbraco que permite a geração rápida e eficiente de aplicações móveis multiplataforma. O desenvolvimento da prova de conceito demonstrou a viabilidade e o potencial desta solução, apesar das limitações de tempo. Com mais desenvolvimento, o pacote pode oferecer funcionalidades ainda mais avançadas e um design aprimorado, transformando a maneira como as empresas abordam o desenvolvimento de aplicações móveis e contribuindo significativamente para a comunidade Umbraco.

Acknowledgement

I would like to express my gratitude to my family for their support throughout the writing of this thesis. Special thanks go to my girlfriend, who helped me review this document, and to my friend, who joined this journey with me five years ago. It feels like just yesterday we began our bachelor's degree, and now we are on the final stretch of our master's degree.

I am also grateful to the company for giving me the opportunity to undertake this project with them, and for providing a dedicated team to manage the project and assist me with my questions.

Finally, I wish to extend my sincere thanks to my thesis supervisor for their invaluable insights and the time he invested in reviewing this document, enabling me to achieve the best possible outcome for this project.

Contents

List of Figures **xv**

List of Tables **xvii**

List of Abbreviations **xix**

1 Introduction **1**
1.1 Problem Definition . . . 1
1.2 Objectives . . . 2
1.3 Methodology . . . 2
1.3.1 Research Questions . . . 2
1.3.2 Search Process . . . 3
1.3.3 Work Methodology . . . 3
1.4 Ethical Considerations . . . 4

2 Literature Review **5**
2.1 Introduction . . . 5
2.2 Umbraco . . . 6
2.3 Historical context and evolution of Umbraco. . . . 8
2.4 Comparison with Other CMS Platforms . . . 9
2.5 Considerations about Umbraco Packages . . . 10
2.5.1 Security . . . 11
2.5.2 Creating and Publishing . . . 11
2.5.3 Popular Packages . . . 12
2.6 Code Generation . . . 13
2.6.1 Model-Driven Development . . . 13
2.6.2 Models . . . 13
2.6.3 Advanced Code Generation Techniques . . . 14
Template-based code generation . . . 14
API-based Generators . . . 15
2.6.4 Advantages and Challenges . . . 16
2.6.5 Domain Specific Languages (DSLs) . . . 17
2.6.6 Program Synthesis . . . 18
2.6.7 Artificial Intelligence . . . 18
2.6.8 Tools . . . 19
Template-based . . . 19
API-based . . . 20
DSLs . . . 20
GPT . . . 20
2.6.9 Discussion . . . 20

3 Solution **23**
3.1 Umbraco Package . . . 24
3.1.1 Analysis . . . 24
Document Types Functional Requirements . . . 25
3.1.2 Design . . . 28
3.1.3 Implementation . . . 31
3.1.4 NuGet Installation . . . 46
3.2 Middleware . . . 48
3.2.1 Analysis . . . 48
3.2.2 Design . . . 50
3.2.3 Implementation . . . 51
3.3 Pipeline . . . 52
3.3.1 Analysis . . . 52
3.3.2 Design . . . 53
3.3.3 Implementation . . . 54
3.4 Mobile Generator . . . 57
3.4.1 Analysis . . . 57
3.4.2 Design . . . 58
3.4.3 Implementation . . . 59

4 Solution Validation **61**

5 Conclusion **71**
5.1 Future Work . . . 72
5.2 Umbraco Awards . . . 72

Bibliography **75**

List of Figures

2.1 Umbraco Sections [2] . . . 6
2.2 TBCG . . . 14

3.1 UmApp High Level Component Diagram . . . 23
3.2 Umbraco Backoffice . . . 24
3.3 Umbraco App Creation . . . 25
3.4 Plugin Architecture . . . 28
3.5 Source List Plugin . . . 29
3.6 Child Prop Plugin . . . 30
3.7 UmApp UI Plugin . . . 31
3.8 UmApp Solution . . . 44
3.9 Middleware Use Case Diagram . . . 48
3.10 Middleware Register Page . . . 49
3.11 Middleware Login Page . . . 49
3.12 Middleware Main Page . . . 50
3.13 Middleware Architecture . . . 50
3.14 Generator Pipeline . . . 53
3.15 Generator Architecture . . . 58

4.1 Futebol Manager Website . . . 61
4.2 UmApp in Settings tree . . . 62
4.3 UmApp Main Page . . . 62
4.4 UmApp Document Types . . . 63
4.5 Football App . . . 64
4.6 UmApp CardsList . . . 64
4.7 UmApp CardsList filled . . . 65
4.8 UmApp CardsGrid . . . 65
4.9 Mobile structure . . . 66
4.10 Pipeline Runs . . . 66
4.11 Pipeline Last Run . . . 67
4.12 My Apps View . . . 67
4.13 Main Page . . . 68
4.14 Main Page scroll down . . . 68
4.15 Details Page . . . 69
4.16 Menu . . . 70
4.17 Trainers Page . . . 70

5.1 UmApp NuGet . . . 72
5.2 UmApp Umbraco Marketplace . . . 73

List of Tables

3.1 Properties of Mobile App 25
3.2 Properties of Page 26
3.3 Properties of Cards Grid 26
3.4 Properties of Cards List 26
3.5 Properties of List 27
3.6 Properties of Story Carousel 27
3.7 Properties of Navigation Button 27
3.8 Properties of Modal 27
3.9 Properties of Splash Screen 27
3.10 Properties of Form 28

5.1 Objectives retrospective 71

List of Abbreviations

AI	**A**rtificial **I**ntelligence
API	**A**pplication **P**rogramming **I**nterface
CEO	**C**hief **E**xecutive **O**fficer
CMS	**C**ontent **M**anagement **S**ystem
CSS	**C**ascading **S**tyle **S**heets
DSL	**D**omain **S**pecific **L**anguage
GPL	**G**eneral **P**urpose **L**anguage
LLM	**L**arge **L**anguage **M**odel
MDA	**M**odel **D**riven **A**rchitecture
MDD	**M**odel **D**riven **D**evelopment
MDE	**M**odel **D**riven **E**ngineering
ML	**M**achine **L**earning
MVC	**M**odel-**V**iew-**C**ontroller
NLP	**N**atural **L**anguage **P**rocessing
OCL	**O**bject **C**onstraint **L**anguage
OMG	**O**bject **M**anagement **G**roup
PBI	**P**roduct **B**acklog **I**tem
SQL	**S**tructured **Q**uery **L**anguage
TBCG	**T**emplate **B**ased **C**ode **G**eneration
UML	**U**nified **M**odeling **L**anguage
VHDL	**V**HSIC **H**ardware **D**escription **L**anguage

Chapter 1

Introduction

With the increase in digitalization, companies are looking for solutions to remain competitive in the market.[1] These solutions include websites advertising their products and services, and/or mobile applications, with the aim of reaching the largest number of users.

This thesis aims to address this demand by developing a package capable of being installed in any Umbraco project that will allow users to create and customise their desired mobile app. That structure is then extracted in such a way that a code generator is able to create a cross-platform mobile solution that will give developers a jump start to customize the solution according to more specific needs. Trying to achieve not only the replication of the existing functionalities in the website, but also to introduce specific features that take advantage of components available exclusively on mobile devices.

The initial goal was to develop both the package and the code generator within this project. However, because of the approaching deadlines and the effect on our submission for the Umbraco Awards, the company decided to form a team of two developers. One developer would handle the code generator, while I would be responsible for creating and distributing the Umbraco package, which will be the primary focus of this thesis.

1.1 Problem Definition

Founded in September 2005, Agap2IT is an European organization in the field of Information Systems, Science and Technology. Agap2IT currently has a large client base that uses websites developed in Umbraco and, as a result, new challenges to develop mobile applications with the same purpose are proposed by these clients. In order to take advantage of the features and capabilities of a mobile device, it is necessary to develop native applications for both Android and iOS systems.

Agap2IT is therefore aiming to enhance the projects' workflow efficiency, from providing a feature-rich and dependable package to reducing time to development. In this context, not only does this package provides an initial skeleton of the application to the developers, but also enables them to focus and dedicate the remaining time in developing new functionalities, of more specific or complex nature, or even translating customized Umbraco components when not automatically possible.

This solution have the potential to prevent Agap2IT from facing higher development costs and time, while securing their current clients by addressing their needs and more, increasing their customer base by providing more cost-efficient solutions than other competing companies, and as a result strengthening its position on the market.

In addition, by submitting this innovative package to Umbraco marketplace, it contributes to the community, which is in alignment with the company's values.

1.2 Objectives

After understanding the problem that originated this work, is important to define what objectives must be accomplished.

- **Development of an Umbraco Package:** Develop a proof of concept that allows the generation of cross-platform mobile applications.
- **Automation in the extraction of Umbraco structure:** To make this solution adaptable to any Umbraco website, it is necessary to create an automated algorithm that extracts the website's structure and exports it in the expected format.
- **Testing:** To evaluate the precision and efficiency of this solution, several tests in Umbraco websites must be performed.
- **Contribution to Umbraco's Community:** Make the Umbraco package available to the community through the marketplace, providing more resources and functionalities for other developers.

1.3 Methodology

This systematic mapping review aimed to identify and examine significant studies concerning Umbraco as a content management system, as well as the development of plugins and packages within its ecosystem. Additionally, research was conducted in the area of code generation, with a focus on various techniques and tools for generating code. Inclusion criteria for studies with publication dates after 2000 and available in English or Portuguese. The search terms and keywords employed include 'umbraco', 'content management system', 'packages,' and 'plugins.'

1.3.1 Research Questions

To address the challenges related to extracting Umbraco's website structure and developing distributive packages, this study is guided by the following research questions:

RQ1 - How can we extract the Umbraco website structure?

Justification: Identifying an effective approach to extract the Umbraco structure is essential for this thesis. An automated method is necessary to collect all website data for the tool that creates the mobile application. A reliable extraction technique will allow for greater scalability and flexibility across different Umbraco websites, improving the tool's versatility and value.

RQ2 - What kind of packages exist and how can we create a package?

Justification: Understanding the different types of packages available in the Umbraco ecosystem and the process of creating a package is crucial for the successful distribution of the final solution. This knowledge will help in designing a package that is not only functional but also adheres to best practices and standards within the Umbraco community.

RQ3 - What are the requisites to publish a package in the Umbraco marketplace?

Justification: Publishing the final solution as an Umbraco package is essential for contributing to community development efforts and extending the tool's reach. Understanding the requirements and guidelines for publishing in the Umbraco marketplace will ensure that the package meets all necessary standards for quality, security, and usability.

1.3.2 Search Process

The initial search involved querying academic databases such as Google Scholar, ACM Digital Library, and b-on. After thorough search, websites and documentations from several tools had to be included to give a better understanding of the literature review. The results were filtered based on the publication date and language criteria. The review process began with a review of titles and abstracts to identify papers that fit the research questions.

The search queries were constructed using Boolean operators to combine relevant keywords. The following exemplifies the approach applied to Google Scholar:

"umbraco" OR "content management system" OR "packages" OR "plugins" AND ("2010" TO *)

This query initially limited results to articles published from 2010 onward. However, during an extensive search, it became evident that numerous relevant articles and books predated this timeframe. Consequently, the date filter was adjusted to contemplate information from the year 2000 onward, ensuring a more comprehensive inclusion of valuable content.

This methodology ensured a systematic and rigorous approach to identify, select, and analyze relevant studies on Umbraco website structure extraction and the production of distributive packages within the specified parameters. A focus on titles and abstracts initially simplified the identification process, followed by a more detailed review of selected articles to uncover valuable insights.

1.3.3 Work Methodology

The work methodology for the development of this project was structured around the Scrum framework, utilizing two-week sprints to organize and manage the work efficiently. The sprint would have four different phases:

- **Sprint Planning**

  At the start of each sprint, a sprint planning session was held to outline the objectives and tasks for the next two weeks. This involved decomposing product backlog items (PBI) into smaller tasks to facilitate estimation and prioritize the tasks accordingly.

- **Daily Meetings**

  Every day, a brief meeting, usually around 20 minutes long, takes place to provide feedback on the previous day's work and discuss any potential obstacles.

- **Sprint Review**

  At the conclusion of every sprint, a sprint review session was held to present the completed work. These meetings also provided an opportunity to brainstorm about the project, determine new directions, assign new tasks or features, and decide on any that might be eliminated.

- **Demo**

  Every month, a demonstration is conducted with the project's stakeholders, which include the CEO and the company's tech leads. During these demonstrations, a comprehensive presentation of the project is given, and new feedback is gathered to enhance the quality of the final product.

To achieve this solution, a deeper understanding of various key concepts is essential. The first chapter introduces the problem outlined by Agap2IT and details the methodology employed in this research. The second chapter provides an overview of the CMS Umbraco, comparing it with other similar systems, explaining packages and their submission process to the web. Additionally, this chapter delves into important concepts of code generation such as model-driven development, DSLs, and various code generation techniques and tools. Chapter 3 will cover the implementation details of the solution, describing the various components involved. Chapter 4 will focus on validating the solution to demonstrate the tool's efficiency, and Chapter 5 will present the conclusions about the project.

1.4 Ethical Considerations

This document was prepared with careful attention to ethical considerations.

- **Artificial Intelligence Usage:** Certain paragraphs of this thesis have been rephrased using artificial intelligence tools. This was done with fairness and precision, ensuring that no sensitive information was included in the processed text. This approach upholds the academic principles of honesty and integrity, ensuring proper credit is given to the original authors.
- **Confidentiality and Review Process:** No details in this document were disclosed without a company representative's prior review. Ensuring no information was revealed, tests on client websites were excluded; a demonstration website was developed instead for validation purposes.

Chapter 2

Literature Review

2.1 Introduction

This literature review aims to address the research questions presented earlier in this thesis through an extensive analysis of Umbraco and its ecosystem. The review begins with an introduction to Umbraco, detailing its features, capabilities, and its place in the evolution of content management systems (CMS).

Following this, a comparative analysis with other CMS platforms will be conducted. This comparison will highlight the utility and advantages of using Umbraco while also discussing its disadvantages relative to other CMS platforms. By exploring these comparisons, the review aims to provide a nuanced understanding of Umbraco's strengths and areas for improvement.

The review will subsequently examine the importance of packages and plugins in open-source platforms such as Umbraco. It will explore the creation process of these packages, their significance, and the steps required to publish them in marketplaces and package managers. This section will include practical examples of successful Umbraco packages, illustrating their impact and importance within the community. Additionally, it will discuss future steps to enhance Umbraco as a CMS and examine current trends in CMS development.

A dedicated section will focus on code generation and model-driven development, topics initially central to the project's scope. Despite the project's shift in focus, the research conducted in these areas remains relevant and valuable to the community. This inclusion will provide a broader context in the area of code generation which is a crucial part of the final solution.

By addressing these topics, this literature review aims to provide a comprehensive understanding of Umbraco and its ecosystem, offering insights into its strengths, areas for improvement, and future directions. This in-depth analysis will support the overall objectives of the thesis and contribute valuable knowledge to the CMS community.

2.2 Umbraco

Umbraco is an open-source content management system for publishing content on the web. It is written in C# and uses Microsoft's ASP.NET framework. Umbraco is known for its flexibility, scalability, and ease of use, making it a popular choice for building websites and web applications. [2]

Setting up the latest version of Umbraco requires .NET 8 or higher and a suitable database like SQL Server or SQLite. Following these prerequisites, Umbraco can be installed either through the NuGet Package Manager or using the Umbraco installer.

Once the Umbraco project is created and initial configuration is completed via the Umbraco wizard, users are presented with the backoffice interface for website management and development.

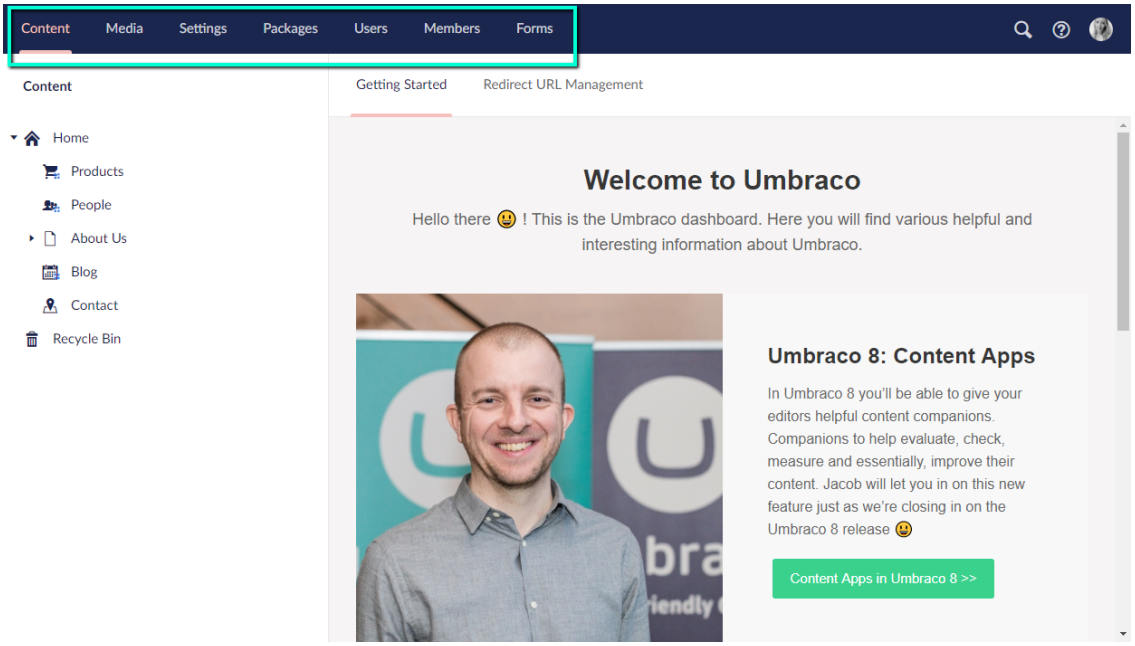

Figure 2.1: Umbraco Sections [2]

Figure 2.1 illustrates the initial view visible upon admin login. On the left side, a content tree displays nodes representing the website's content, allowing customization of component display and information within each node. The navbar holds different sections of the backoffice, a section in Umbraco is where you perform specific tasks related to a particular area of Umbraco.

- **Content**

  As mentioned before, this content is organized in a tree structure represented by nodes. Each node represents a document type defined previously. Its in here, where its possible to manage all the content and data of the different nodes, making it easy and simple for maintenance.

- **Media**

  Umbraco also has a media library for managing images, videos, and other media assets.

- **Settings**

  The Settings section of Umbraco allows users to manage website layout files, languages, and define media and content types. It also provides access to the Log Viewer for browsing through log files. It consists in the following major areas:

  - **Document Types**

    A document type is a data container in Umbraco where we can add content structures and properties to input data. Each property has a Data Type like text string, number, or rich text body. Some examples of document types could be a page, a modal, or a list.

  - **Data Types**

    A Data Type defines the type of input for a property. There are many built-in data types with different complexities, from textstrings or numbers to multi node tree picker or grid layouts. Its also possible to create new data types and customize them in the most appropriate way to our scenario.

  - **Templates**

    As mentioned before, Umbraco uses the .NET Framework so in order to display the document types in a website, we need to create templates. Templates are nothing more than Razor Views[1] from ASP.NET MVC. Templating also allows Master templates and partial views to make the code more efficient and more readable.

- **Packages**

  Within this section, users can browse and install packages into their Umbraco solutions. It provides an overview of all installed packages and allows for uninstalling those that are no longer needed.

- **Users**

  Users in Umbraco refer to individuals who access the Backoffice, where they can manage, create, and customize user accounts and user groups. Users typically have administrative or editorial roles, allowing them to edit content, manage settings, and perform administrative tasks within the CMS.

- **Members**

  Members refer to individuals who interact with the front-end of the website. They are typically end-users or visitors who register or subscribe to the website, creating member accounts. Members can have profiles, subscribe to newsletters, access restricted content, and interact with features that are designed for registered users.

- **Forms**

  Umbraco Forms is featured in the top bar of the Umbraco Backoffice, reflecting its pivotal role in website management. This package allows administrators to install it directly from the Backoffice with a simple click, facilitating easy implementation without requiring extensive technical expertise.

[1] Razor is a markup syntax for embedding .NET based code into webpages. The Razor syntax consists of Razor markup, C#, and HTML.

Umbraco offers a robust API framework that includes both services and helpers to facilitate development tasks within the CMS environment. Services play a vital role in interacting with Umbraco data, providing functionalities to modify, retrieve, and delete core entities stored in the database. These services enable developers to create, update, and delete different Umbraco entities programmatically, ensuring flexibility and efficiency in customizing and managing content within Umbraco applications.

Umbraco uses dependency injection to provide these services, simplifying their integration into custom code using the following syntax:

```
    public class CustomController
{
    private readonly IContentService _contentService;


    public ContentController(IContentService contentService)
    {
        _contentService = contentService;
    }


    public ActionResult PerformAction()
    {
        var someContent = _contentService.GetById(1234);
    }
}
```

Listing 2.1: Umbraco Controller [2]

Currently, Umbraco has 12 services: AuditService, ConsentService, ContentService, ContentTypeService, DataTypeService, EntityService, FileService, LocalizationService, MediaService, MemberService, MemberTypeService, MemberGroupService.

2.3 Historical context and evolution of Umbraco.

Based on Umbraco website [3], Umbraco started in 1999 when Niels Hartvig made the first sketches and prototypes to a simple framework for building and maintaining websites. Umbraco's name was inspired by the founder desire for uniqueness and his use of an "Umbraconøgle" (Allen key in Danish) to successfully assemble IKEA furniture.

In 2003, Umbraco 1.0 was announced, built with classic ASP, VB.NET, and C#, introduced key concepts like Document Types, Macros, and XSLT, but lacked performance and scalability.

In 2004, Umbraco 2.0 was already 100% .NET based and a beta version was released in October of the same year. In 2005, Umbraco 2.0 was released as an open-source CMS which represented a great step for the company. The first ever Umbraco Developer conference, called CodeGarden took place in Copenhagen.

In the following years, versions 3.0 and 4.0 were released until 2009, when the dedicated Umbraco Community site, Our Umbraco, was launched at Codegarden 2009. Until this day, it remains the essential resource for technical documentation and peer support for Umbraco developers.

In 2013, Umbraco source code became available in GitHub and this made possible for developers to contribute to the source code by creating issues and making pull requests in order to fix bugs and suggest features, achieving 250 pull requests in that year. Version 6.0 and 7.0 were also release in that year.

In November 2015, Umbraco Cloud was launched to streamline building, hosting, and deploying Umbraco sites. In 2017, Umbraco celebrated the 400,000 active installs of Umbraco.

In 2019, Umbraco released a major version, 8.0, 6 years after the last version which became most used version in the following years.

In 2021, Umbraco 9.0 comes to the market using .NET 5 and ASP.NET Core framework . The CodeGarden conference counts with 2400 attendees and the company has already 105 employees.

In recent years, Umbraco has launched several versions, with the most recent being version 14, released on May 30, 2024. This version features new APIs and a redesigned backoffice implemented using TypeScript and Web Components, replacing the old AngularJS framework used in previous versions.

2.4 Comparison with Other CMS Platforms

A CMS is a system used to manage the content of a Web site. A CMS is a computer application that allows publishing, editing, modifying, and organizing, deleting, and maintaining content from a central interface. [4]

According to [5], the top 3 CMS in 2024 are WordPress, Joomla, and Drupal.

- **WordPress**

  WordPress is one of the most popular CMS in the world, it was launched in 2003 by Matt Mullenweg and Mike Little. Is also open-source which means its source code is available for the community and can be modified and distributed according to everyone needs. According to W3Techs [6], WordPress is used by 62.7% of all the websites who use content management system, which makes 43.4% of all websites. WordPress was initially designed as a blogging platform, in the last several years WordPress has changed itself as a useful content management system. One of the main advantages of WordPress is the large number of plugins released by independent developers. [7] There are other advantages associated to the use of WordPress, like the large community that holds this CMS and contributes actively with new updates to enhance the security and add new functionalities through the use of plugins. But there are also some disadvantages, most of them rely on security vulnerabilities due to its popularity, WordPress is a common target for malicious eyes.

- **Joomla**

  According to the Joomla website [8], it was first released in 2005 as a fork of Mambo, another CMS that was very popular in the 2000s. This CMS, developed in PHP and using MySql databases, provides functionalities very similar to WordPress. As per W3Techs [6], Joomla is employed by 2.6% of all websites utilizing a content management system, which represents 1.7% of all websites. Joomla includes built-in support for multiple languages, making it a great option for multilingual websites without the need for additional plugins. However, its complexity might be an issue

today, as it requires PHP knowledge. According to the TIOBE Index for 2024 [9], PHP has only a 1.22% rating, indicating a shortage of developers specialized in this technology.

- **Drupal**

  Drupal is also another CMS open-source that currently holds the top 3 on the most used CMS worldwide. It was launched in 2001 by Dries Buytaert and at today's date, is used by 1.8% of all the websites that use a content management system, which makes up 1.2% of all websites, according to W3Techs [6]. Drupal is also based on PHP which brings the same disadvantages as Joomla and some authors, including Will Morris [10], might even affirm that Drupal is more complex and has a more steep learning curve because requires more technical expertise using technologies like PHP, HTML, CSS and JavaScript. On the other hand, Drupal is more known for its strong focus on security and for its reputation for being both highly-scalable and light on resources.

In a comparative study conducted by Patel [7], it was found that the performance of these three major CMS platforms are quite similar when tested under identical conditions. However, WordPress demonstrated slightly higher performance metrics in most areas evaluated in the study. This marginally superior performance is one of the factors contributing to WordPress's significant lead in market share over other CMS technologies.

To sum up, the most important aspect that defines the optimal CMS for any situation is the simplicity and effectiveness of content creation and administration. Importantly, the presence and excellence of packages and plugins distinguish a top-tier CMS. These plugins offer flexible solutions for various needs, allowing users to enhance functionality and tailor their systems to specific demands.

Umbraco, conversely, is a platform deeply intertwined with Microsoft technologies such as .NET and Azure, making it a clear option for a company like Agap2IT, a Microsoft Gold Partner. As per the Umbraco Marketplace, there are nearly 400 packages available for Umbraco, which is significantly less than the 60,000 plugins available in the WordPress ecosystem, underscoring the need to contribute new and valuable packages to the Umbraco community.

2.5 Considerations about Umbraco Packages

This part emphasizes important aspects related to Umbraco packages, concentrating on their security, reliability, and the procedures for developing and releasing them in a marketplace. It also presents successful examples of Umbraco packages that have had a notable impact on the community.

2.5.1 Security

As previously stated, plugins (or packages for Umbraco) are crucial for their success, but the majority of these plugins are community-created. This situation raises concerns, especially in platforms like WordPress, which has over 60,000 plugins available in the marketplace. Some of these plugins may contain vulnerabilities or malicious code that could harm users who download and install them. Some considerations a user must have when downloading a unknown plugin are:

- **Authentic Sources**

  Looking for plugins with trusted sources is the first step to make sure they are not malicious. Browsing the official marketplace of plugins is mandatory because they each new plugin submitted to the marketplace undergoes a review process to ensure it meets quality standards. Also, performing an investigation on the author of the plugin could help, a author with another known plugins will transmit more security. [11]

- **Reviews and Ratings**

  Even if a plugin is available in an official marketplace, reading reviews can help identify any potential issues reported by other users.

- **Update frequency and Documentation**

  A plugin that is regularly updated is less likely to have vulnerabilities. Additionally, a plugin with extensive and useful documentation aids developers in correctly implementing its functionalities, thereby minimizing potential security breaches.

2.5.2 Creating and Publishing

Since Umbraco is part of the .NET ecosystem, their packages are firstly hosted in the NuGet repository in nuget.org to make them available for download in any .NET project. According to Balliauw and Decoster, NuGet is a free, open source, package management tool for the .NET platform, C++, and JavaScript. A NuGet package can consist of assemblies, configuration files, source code, and more. [12]

According to Microsoft official documentation [13], creating a new NuGet package involves using a .NET Class Library, which is a specific type of project in .NET. This class library should encapsulate all the functionalities intended for the NuGet package. Additionally, metadata can be added directly to the .csproj file, including details such as version, authors, company, package ID, licenses, and tags.

Once the project is fully prepared, the following command is used to pack the project:

```
dotnet pack
```

Executing this command generates a .nupkg file, which is the NuGet package. This package includes a .nuspec file containing all the package's configuration details and references to the compiled DLLs. The .nuspec file also holds metadata that defines the package's dependencies, framework compatibility, and other relevant information.

Now, this results in a NuGet package that can be used locally. To make it available to the community, publishing it to NuGet.org is essential. The process is straightforward and involves creating an account on the NuGet.org website and generating an API key.

This API key can then be used with the following command to push the NuGet package to the repository:

```
dotnet nuget push Project.nupkg --api-key xxxxxxxxxxx --source
https://api.nuget.org/v3/index.json
```

To improve the security and reliability of a package, it can be signed. This involves getting a paid certificate from a public certificate authority and exporting it as a .CER file. After obtaining the certificate, it can be uploaded directly through the management section on the NuGet website, or a new package version can be pushed using the –certificate-path flag with the certificate's path.

Once the NuGet is available in the repository, publishing it to the Umbraco Marketplace is quite straightforward. When logging in to Our Umbraco website, there will be a section called Packages where it is possible to submit a request to add a new package. In this request, a link to the NuGet repository must be provided and some metadata information, similiar to the publish of the NuGet previously explained. After submission, this process will be reviewd by the Umbraco team to ensure that it meets their guidelines and standards. After approval, the package will be available in the Umbraco marketplace.

2.5.3 Popular Packages

Over the years, many Umbraco packages were developed and posted in the marketplace but there a few that really made an impact in the community, for example, uSync.

uSync, an Umbraco package created by Kevin Jump in 2013, has received multiple versions and updates, and is currently at version 14.0.0 with 4 million downloads. uSync transfers the elements of Umbraco stored in a database to disk, allowing users to source control, copy, and move their Umbraco sites between computers and servers.

uSync also is an open-source project, which can be found here, and currently has 31 active contributors and serves as a great learning tool to develop new plugins.

2.6 Code Generation

As previously mentioned, this section will provide a comprehensive overview of code generation techniques and the tools available to achieve a final solution. It will begin with an introduction to model-driven development, followed by an exploration of advanced code generation techniques such as template-based and API-based methods. The section will then discuss the advantages and challenges associated with these methodologies. Additionally, other approaches like Domain-Specific Languages (DSLs), program synthesis, and artificial intelligence will be introduced. Finally, the section will present tools corresponding to each code generation technique, concluding with a discussion on the various techniques and identifying the most suitable one for the project.

2.6.1 Model-Driven Development

The history of Model-Driven Development (MDD) can be traced back to the early 2000s when the Object Management Group (OMG) introduced the Model-Driven Architecture (MDA) initiative. This initiative, which was made public in November 2000, marked the beginning of a new global trend called Model-Driven Engineering (MDE) [14]. The MDA concept aimed to create a family of languages using metamodeling[2], similar to the Unified Modeling Language (UML) [15].

MDE has a wider scope than MDD, while MDE tries to combine the process and analysis with the architecture, MDD, according to [16], is an approach aimed at increasing productivity and reducing time-to-market by enabling development at a higher level of abstraction.

This goal is achieve by the use of models that provide abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on the relevant ones. [17]

MDD also verifies the accuracy of models and generate source code, reducing the development time required to check and test the software, allowing programmers to focus on the modeling process. [18]

2.6.2 Models

UML is widely used as a modeling language in software development. Some tools, including Enterprise Architect and Visual Paradigm, provide the opportunity of automatically generate source code from UML diagrams. It takes in account the class diagrams to represent the system's structure, including classes, attributes, methods and relationships, and sequence diagrams to describe interactions between objects that represent business logic of the application.

[2]Metamodels are models that make statements about modeling. More precisely, a metamodel describes the possible structure of models in an abstract way.

2.6.3 Advanced Code Generation Techniques

Template-based code generation

Template-based code generation (TBCG) is a popular technique in MDE because follows the main principles of abstraction and automation. TBCG can be seen has a technique that uses an abstract and generalized template with the objective of producing source code. According to Syriani that cited Jörges [19], there is three main components in TBCG: the data, the template, and the output.

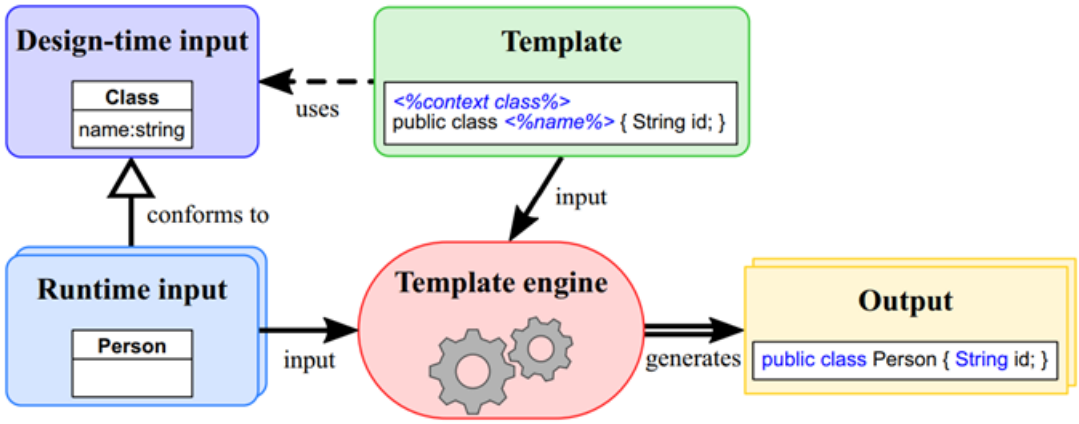

Figure 2.2: TBCG Components [19]

According to figure 2.2, we can see the main concepts of TBCG:

- **Design-time input**

  This component includes all the abstract models developed in the TBCG tool. These models follow the architecture that will be used to generate the Output.

- **Template**

  A template can be seen as a representation of the code structure and uses design-time inputs. Templates also have all the placeholders that will be inserted during the code generation process, based on runtime inputs.

- **Runtime input**

  Runtime input refers to the data and information provided during the code generation process, specifically when the templates are applied to the abstract models.

- **Template engine**

  The engine is the most important component of TBCG that is responsible for interpreting and processing templates at runtime. It takes all the inputs and replace the placeholders in the templates with actual data from the inputs.

- **Output**

  This is the final stage of TBCG and represents the source code generated by the template engine.

API-based Generators

Another popular technique is by using API based generators. Conceptually, these generators are based on the abstract syntax (the metamodel) of the target language, and are therefore always specific to one language, or more precisely to the target language's abstract syntax.[20]

A client can reach to an API endpoint and send inputs. Lets analyse a example in the .NET world given by Fowler[20]:

The objective is to generate this class:

```
public class Vehicle : object {
}
```

In order to generate this code, this is the C# code that will be executed in the API side:

```
CodeNamespace n = ...
CodeTypeDeclaration c = new CodeTypeDeclaration ("Vehicle");
c.IsClass = true;
c.BaseTypes.Add (typeof (System.Object) );
c.TypeAttributes = TypeAttributes.Public;
n.Types.Add(c);
```

This type of code generators are intuitive and easy to use, that is why they are so popular in the code generation world. However, the down side is that may have large amounts of constant code that need to be generated every time, instead of being part of a template like in TBCG previous explained.

2.6.4 Advantages and Challenges

According to Fowler[20], it is possible to cite several advantages on adopting MDD and code generation techniques:

- **Performance**

  Producing efficient and flexible code is one of the pillars of MDD, thanks to the level of abstracting when developing these solutions.

- **Code Volume**

  If at compile time is possible to determine which features will be needed at runtime, the generator only needs to generate those parts, resulting in smaller solutions.

- **Readability and Consistency**

  The use of frameworks often keeps complexity at a proprietary configuration level, making it difficult for developers to analyze certain program properties or functionalities. In contrast, generation tools produce code that adheres to common patterns and standards, similar to manually-written code, enhancing readability and consistency among developers.

- **Early Error Detection**

  By using code generators is possible to detect possible errors before compilation and usually with more information.

- **Plataform Compatibility**

  As mention before, the MDA aims for that the application logic can be programmed independently of the implementation platform, by using metamodeling.

MDD has several advantages and benefits, but like any others approaches, also have challenges and downsides:

- **Models Complexity**

  The more the models and levels of abstraction that are introduced to the software will result in more complex relationships between those models. The roundtrip problem might occur in these conditions which can lead to a change in a model affect some or all of its related artifacts.[21]

- **Expertise Required**

  Adopting MDD involves a learning curve for both modeling languages and the tools used for code generation. Each type of model requires a particular set of skills to produce and evolve effectively.[21]

2.6.5 Domain Specific Languages (DSLs)

Domain-Specific Languages (DSLs) are specialized programming languages designed for a specific domain or problem set, providing notations and abstractions closely aligned with that domain. [22]

General-purpose languages (GPLs) like Java and C# are widely used and commonly known by developers. These languages are designed to solve a wide variety of problems and offer a lot of techniques and ways to solve the same problem. On the other hand, a DSL designed to solve a specific problem can be more expressive and easier to use than GPLs, which means higher productivity and small maintenance costs.

DSLs allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL programs. [22]

According to Fowler[23], a DSL can be categorized into two types based on their form:

- **External DSL**

  These DSLs are written in their own language with the possibility of being processed by another host language and can be used in a wide variety of scenarios. For example, SQL, VHDL, CSS.

- **Embedded DSL**

  The DSLs are embedded in a GPL, can be seen has a library in the common languages to solve a determined problem. For example, LINQ that adds native data querying capabilities to .NET applications.

However, developing a DSL brings significant costs. Not only is needed a DSL developer with high knowledge, but also because of the difficulty in finding the proper scope to implement that DSL, and even the potential loss of efficiency when compared to hand-coded software or a program developed by a general-purpose programming language.

The development of a DSL normally follows two steps:

- **Analysis/Design**

  In this step, we identify the problem and gather all the information about it, then we develop a semantic of notions and operations that will solve this problem. We aim for expressiveness, readability, and abstraction, so that both developers and domain experts can understand and use the DSL.

- **Implementation**

  To implement a DSL, first the library must be developed and then a proper compiler must be choosen to translate the DSL and generate the source code. If none of the existent compilers in the market can translate the created DSL, a new compiler must be developed.

2.6.6 Program Synthesis

According to Copeland[24], program synthesis has been around since 1940s and 50s and lately have been getting much more attention because of the evolution of machine learning. Program synthesis transforms users inputs in natural language or high-level abstractions and generates source code to solve the problems proposed by the users. According to Jonathan, there are two different approaches to program synthesis:

- **Deductive**

  In this approach, the tool requires that the user provides logical rules and specifications to deduct the correct program structure.

- **Inductive**

  This approach can be seen as a more abstract technique where the tool takes a partial specification and constructs a program that satisfies it. Which may result in lower efficiency and more inaccurate approaches to solve a given problem.

2.6.7 Artificial Intelligence

In the last few years, artificial intelligence (AI) came to revolutionize the software development world. Machine learning (ML) methods have been used to create powerful language models for a broad range of natural language processing tasks. An important subset of this field is that of generating code of programming languages for automatic software development.[25]

One of the most popular models is GPT, a large language model (LLM) trained by OpenAI. GPT has been applied in some of the software development tasks, like programming numerical methods, code generation, solving programming bugs, code completion and even in practicing for computer science exam.[26]

In this particular scenario, GPT takes inputs in common language and uses natural language processing (NLP) techniques to interpret the user's request. These large language model uses Big Code, which consists in a vast collection of online software artifacts such as source code repositories, bug databases, and code snippets.[27]

This technique falls under the category of program synthesis with an inductive approach. In this method, a generator utilizes the input parameters of the desired solution, querying LLMs to retrieve source code for constructing the solution.

2.6.8 Tools

This section will explore the most relevant and open-source tools for the different approaches mentioned before.

Template-based

According to Luhunu [28], the template based tools can be classify in two groups, model-based or code-based. A model-based has a block template definition, which means that is composed by a list of block items and those items can have predefined attributes or placeholder content. On the other hand, a code-based has a mix of static and dynamic parts and often allow the use of functions to generate code, this gives more flexibility to the developers.

- **Acceleo**

  Acceleo is a practical implementation of a standard called MOF Model to Text Language (MTL), established by the OMG. In Acceleo, a file is made up of different sections called "typed definition blocks." These blocks are essentially templates where a developer specifies how each type of element in a model should be transformed into text.

- **Xpand**

  Xpand uses the same Acceleo principles but is less powerful because relies in a smaller subset of Object Constraint Language (OCL), a declarative language used to describe rules of transformation. On the other hand, Xpand can use functions defined in Xtend extensions or custom Java code.

- **Xtend2**

  Xtend2 is a complete programming language that builds upon Java and introduces a DSL that extends Java by incorporating features like lambda expressions and templates.

- **Velocity**

  Velocity is a Java-based template engine. Its a simple but powerful template language that requires Java objects as the input and based on that, custom functions in Java code will interpret the objects and generate source code.

- **T4**

  T4 is a similar tool when comparing to Velocity, the main difference is that it takes C# objects as the input and can generate C# source code simultaneously. This tool is integrated in Visual Studio and has two types of templates, run time and design time. According to Visual Studio documentation [29], a run time template is defined in the source code and can be called using functions to generate a piece of code, like a HTML page. On the other hand, a design time template takes as a input a configuration file and generates source code in C#.

API-based

As for API based generators there are several proposals for new tools that accomplish this purpose but we can highlight OpenAPI as the most used tool in this context and Roslyn a open-source compiler in the .NET ecosystem.

- **OpenAPI Generator**

  The OpenAPI Generator is a tool that automates the process of generating client libraries, server stubs, and API documentation based on an OpenAPI Specification.[30]

- **Roslyn**

  Roslyn is the open-source implementation of both the C# and Visual Basic compilers with an API surface.[31] According to Harrison[32], all the process of generating code is exposed through the API. This allows that developers interact with every step of the process making it simpler for debugging and provide a better end-to-end solution.

DSLs

A DSL can be made using different tools but the most known is the one provided by Microsoft. **DSL Tools** are hosted in Visual Studio and offers every tool necessary to develop a new DSL, since a graphic interface to add new elements and relationships to a validation engine to check for errors in the DSL.

GPT

OpenAI offers developers an API that facilitates the integration of GPT models into their applications. The OpenAI GPT API allows developers to make calls to the GPT models hosted by OpenAI, generating responses based on prompts provided to the system.

2.6.9 Discussion

With this research, it is possible to highlight several techniques to generate code. One path for a solution could be the use of models and translate them directly into source code, but that would require complete UML diagrams to represent all the business logic and system structure. Another would be the used of an advanced code generation technique like template-based or API-based to interpret the model and generate source code. Creating a DSL could be an option aswell, but that would require a specialized knowledge in this domain and that translates in high costs to the company. Finally, using an approach relative to AI would be also valid.

According to the problem previously explained by Agap2IT, the goal is to generate source code based on an existing application. That being said, using the UML code generation technique is no longer viable because that would require an extra effort from the developers to create UML diagrams complete enough to be able to generate correct source code. Creating a new DSL or using AI would require specialized developers which means an added cost to the company. So, the most logical solution would be to extract all the data from the website and use that as an input on a template-based or api-based tool.

For the first scenario, several tools were presented, Acceleo, Xpand, Xtend2, Velocity and T4. The first four were designed in Java and take Java objects as inputs, which would require an extra effort to convert the Umbraco structure from C# to Java. So, the natural

solution would be T4 which allows C# objects and is integrated in Visual Studio. For the second scenario, Roslyn seems a better solution because is also part of Visual Studio and its implemented in C#.

Following discussions with the company and a clearer understanding of the proposed solution, it was discovered that an internal web code generator had already been created. This generator, utilizing Roslyn, was seen as the best option because the team is already familiar with the technology, and some existing features can be adapted for the mobile code generator.

Chapter 3

Solution

This chapter will detail every step and component developed during this project. For a better understanding, it's possible to distinguish four different components: Umbraco Package, Middleware, Pipeline, and Mobile Generator.

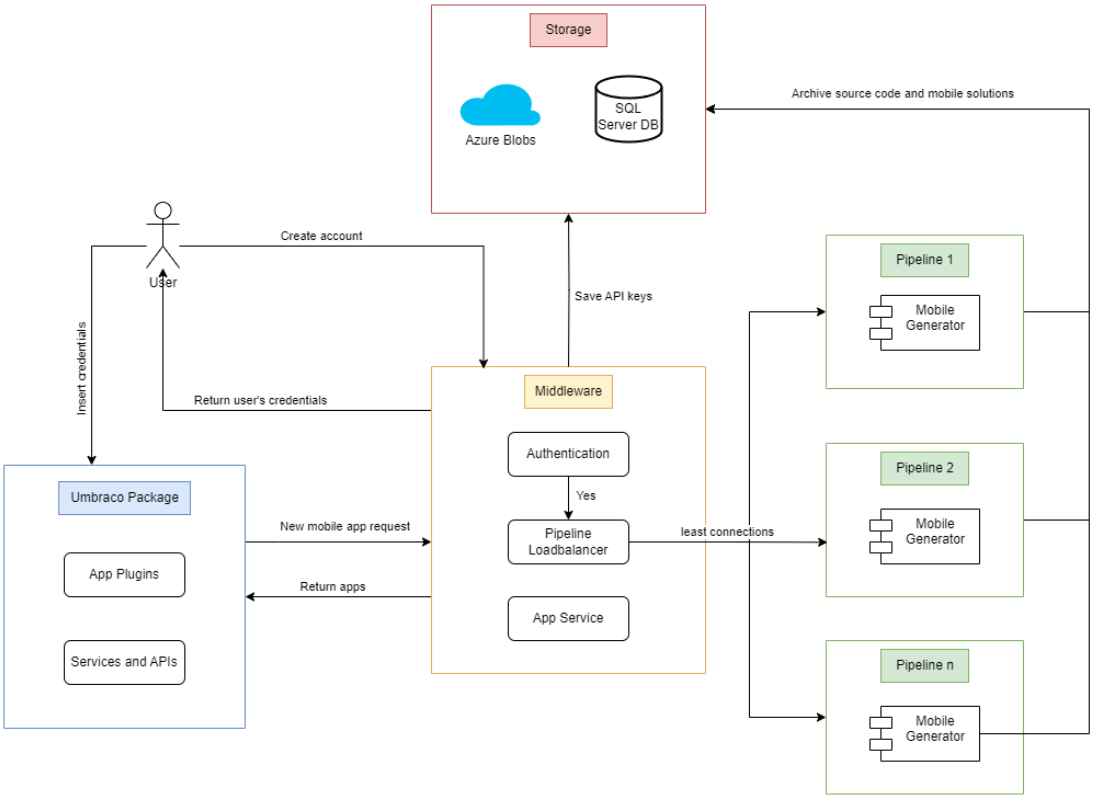

Figure 3.1: UmApp High Level Component Diagram

Figure 3.1 illustrates the main components previously mentioned, their connections, and user interaction methods. Users directly engage with the middleware to acquire a unique API key and to register this key within the Umbraco package. When creating a new mobile application, the Umbraco Package transmits a request containing all the essential configurations and user credentials to the Middleware. The Middleware confirms the request by checking the key, and upon successful validation, forwards it to a selected pipeline that utilizes a least connections algorithm to manage load distribution. This pipeline manages the creation using the Mobile Generator, after which the solutions are stored in Azure blobs.

3.1 Umbraco Package

Umbraco, as an open source CMS, encourages the collaborative development of packages by its community members. These packages can be composed by multiple plugins and extend Umbraco's functionality, enabling users to personalize their websites with features like custom data types, forms, or advanced e-Commerce solutions. Typically, developers craft plugins using technologies like AngularJS/JavaScript, HTML, and CSS for the user interface, complemented by C# for backend operations in the latest Umbraco version. These plugins are accessible through the Umbraco store or can be stored in local repositories for more customized needs. This approach significantly enhances Umbraco's versatility and adaptability, empowering users to create highly customized and feature-rich websites with ease.

3.1.1 Analysis

To create this Umbraco package, we must develop three distinct plugins aimed at enhancing the Umbraco backoffice. The initial plugin involves a new data type that enables the selection of a source list/document type from an existing website, with the constraint that only one document type can be chosen as its children at this stage. To solve this, an input feature will facilitate the selection of a document type from the available options in that source list. Once the source list and child document type are selected, another plugin will be developed. This plugin's purpose is to establish a connection between existing properties in the child document type and the properties supported by mobile document types. Lastly, we require a plugin to provide users with a user-friendly interface to configure their credentials and generate mobile applications.

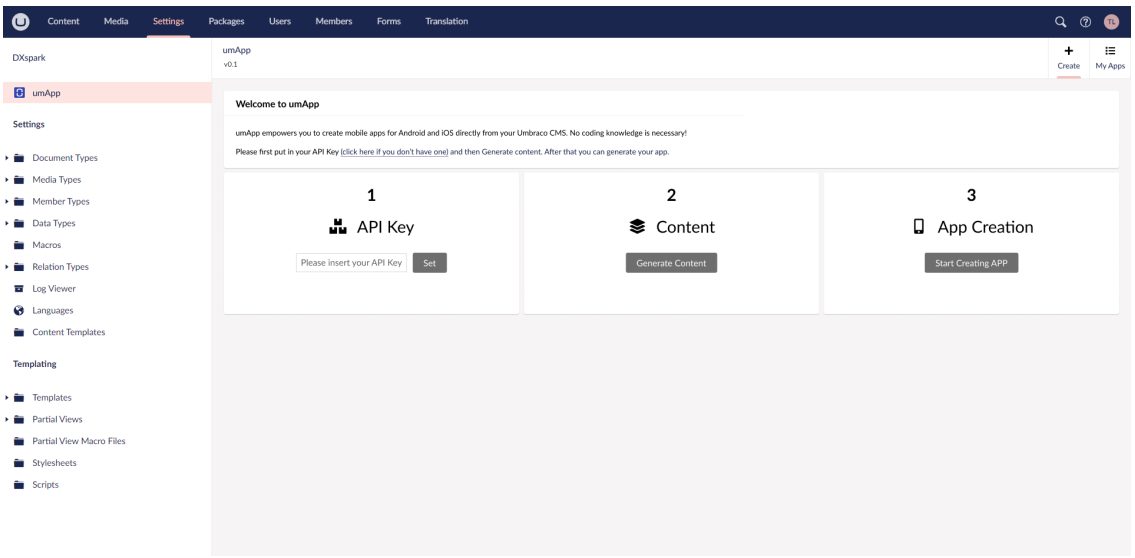

Figure 3.2: Umbraco Backoffice

Figure 3.2 represents the main screen of the plugin, in this page the user can do all tasks related to setting the API key, generate the necessary content and start the process of creating a new mobile application.

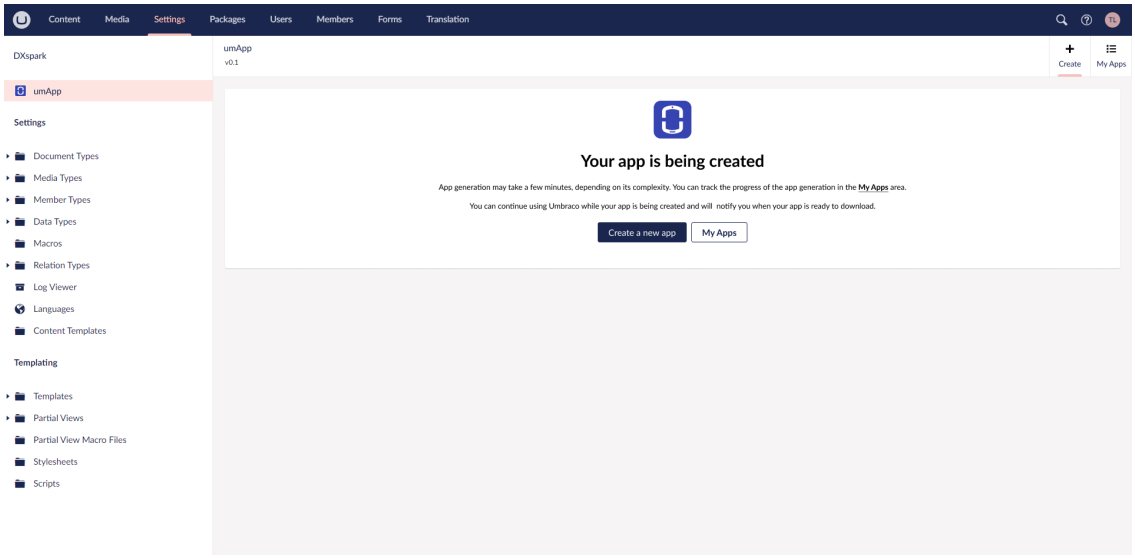

Figure 3.3: Umbraco App Creation

In Figure 3.3, the feedback screen is displayed when the user initiates the generation of a new mobile app. This screen offers the user the option to return to the main screen to generate additional apps or to be redirected to the page listing all generated apps.

Once the package is developed, it needs to be distributed to clients. A NuGet package emerges as the most practical solution, bundling all essential files, dependencies, and configurations required for effortless integration with each project.

Document Types Functional Requirements

This section will present detailed information about each component in a table format. The table will list all the properties for each document type along with their descriptions.

Starting by the main component, it's needed a document type that would represent the mobile application and hold basic configurations, Table 3.1 illustrates five different properties and the list of allowed document types:

Property	**Description**
Primary Color	Specifies the primary color theme used throughout the app.
Secondary Color	Specifies the secondary color theme utilized in the app.
Accent Color	Specifies the accent color used to highlight specific elements in the app.
Icon	Specifies the icon displayed for the app.
Login	Indicates whether the app requires user authentication or not.
Allowed document types	• Page • SplashScreen

Table 3.1: Properties of Mobile App

To build pages within the mobile app, it is also necessary a document type that would represent that. As indicated in Table 3.2, it does not possess any properties but includes all other document types as allowed document types:

Property	**Description**
Allowed document types	• Cards Grid • Cards List • Carousel Cards • Story Carousel • List • Modal • Navigation Button • Form

Table 3.2: Properties of Page

To represent a grid of cards, another document type is needed. As indicated in Table 3.3, it has six properties:

Property	**Description**
Source and Type	Specifies the data source and selects the type for the cards displayed within the grid.
Card Title	Defines the title of each card displayed within the grid.
Card Image	Specifies the image associated with each card in the grid.
Card Subtitle	Provides a subtitle or additional information for each card.
Span	Indicates the number of columns or rows each card occupies within the grid (must be between 1 and 4).
Orientation	Specifies the orientation of the grid layout (horizontal or vertical).

Table 3.3: Properties of Cards Grid

To represent a simple list of cards, another document type with four properties is needed, as we can see in Table 3.4:

Property	**Description**
Source and Type	Facilitates the specification of the data source and selection of the card type to be displayed within the list.
Card Title	Defines the title for each card listed.
Card Image	Specifies the image associated with each card in the list.
Card Subtitle	Provides additional information or a subtitle for each card listed.

Table 3.4: Properties of Cards List

To make simple lists in the mobile application, a document type with the properties presented in Table 3.5 are needed:

Property	Description
Source and Type	Enables you to specify the data source and type for the items displayed within the list.
Item Text	Defines the text or title for each item listed.
Item Icon	Specifies the icon associated with each item in the list.

Table 3.5: Properties of List

To present a carousel of stories, a document type with two properties is needed, according to Table 3.6:

Property	Description
Source and Type	Enables you to specify the data source and type for the stories displayed within the carousel.
Story Image	Specifies the image associated with each story in the carousel.

Table 3.6: Properties of Story Carousel

To facilitate navigation in the mobile application, we can solve this by adding a new document type with only one property, as showed in Table 3.7:

Property	Description
Button Label	Enables you to define the label or text displayed on the navigation button.

Table 3.7: Properties of Navigation Button

A modal is also a key component of any application, to allow this feature, we need a new document type with the properties showed in Table 3.8:

Property	Description
Modal Title	Defines the title of the modal.
Modal Body	Specifies the content or body text displayed within the modal.
Cancel	Represents the label for the cancel button within the modal.

Table 3.8: Properties of Modal

To allow the customization of the splash screen in the mobile application, a new document type holding only an image input is necessary, as we can see in Table 3.9:

Property	Description
Image	Allows you to select an image to be displayed as the splash screen when the application is launched.

Table 3.9: Properties of Splash Screen

A more advanced feature that would allow to insert new entries to a certain list in the Umbraco backoffice with the help of a form, needs a new document type with one property, as displayed in Table 3.10:

Property	Description
Source and Type	Enables you to specify the data source and type for the form.

Table 3.10: Properties of Form

3.1.2 Design

As mentioned above, we can see a typical plugin architecture in Figure 3.4.

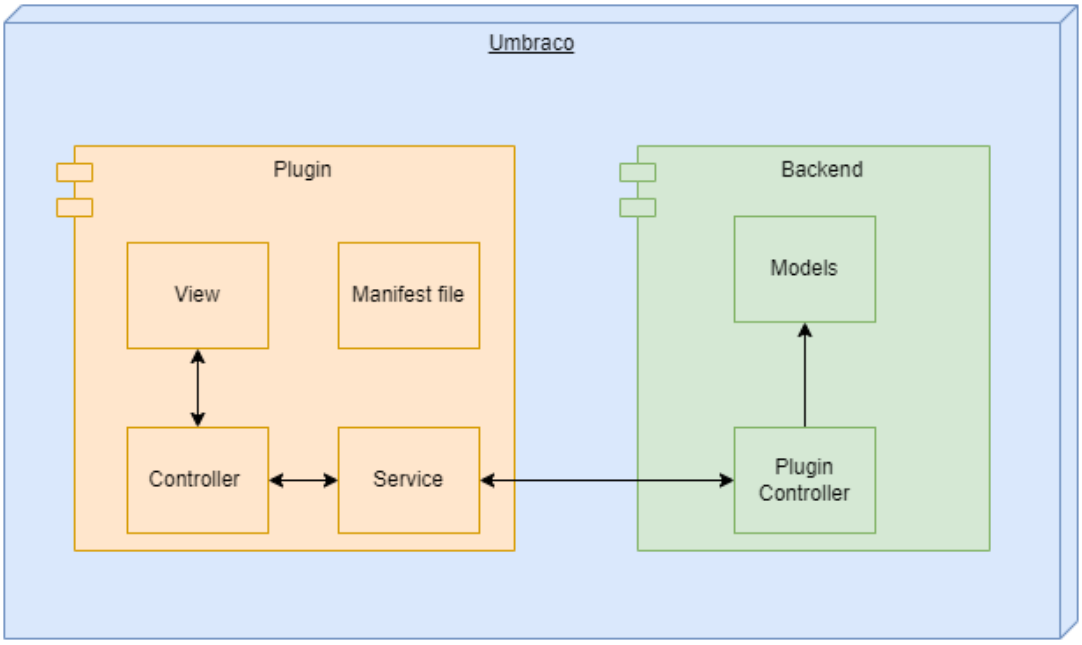

Figure 3.4: Plugin Architecture

The first two plugins are designed to extend the functionality of Umbraco by adding new data types. In this particular case, they do not require access to the Middleware also developed within the context of this project, as all the actions and information will be provided by the Umbraco API. This includes retrieving properties and information of Umbraco elements to create elements in the Umbraco backoffice UI. However, the last plugin requires access to the middleware because it is the service responsible for authenticating user requests to generate new apps and sending the configuration file from the Umbraco backoffice to the pipelines.

For enhanced clarity, the following sequence diagrams will delineate the behavior of these plugins.

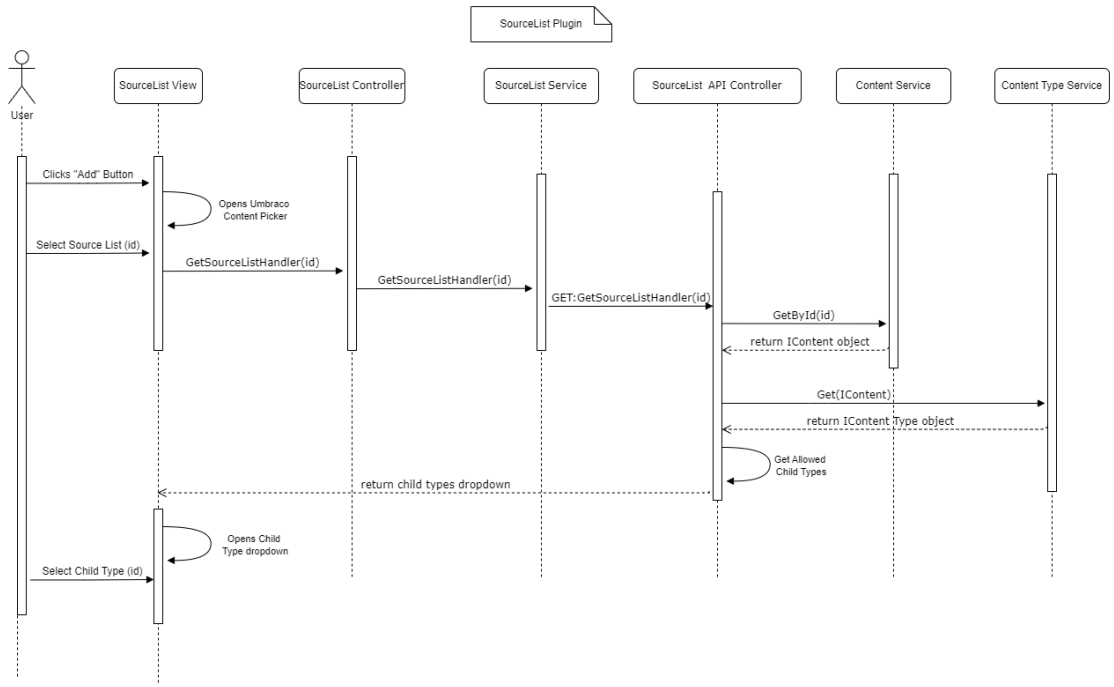

Figure 3.5: Source List Plugin

The diagram presented in Figure 3.5 will illustrate the workflow of the "Source List Plugin". First, the user must interact with the "Add" button that will trigger a native component from Umbraco named Umbraco Content Picker. This picker will allow the user to select a content node from their content tree and upon selection a request to the Source List Controller will retrieve all allowed content types for that document type. The controller will rely on the content service and content type service provided by the Umbraco API to retrieve that information. Finally, it presents these document types in a dropdown menu, facilitating user selection of the desired document type for binding the properties intended for translation into the mobile application.

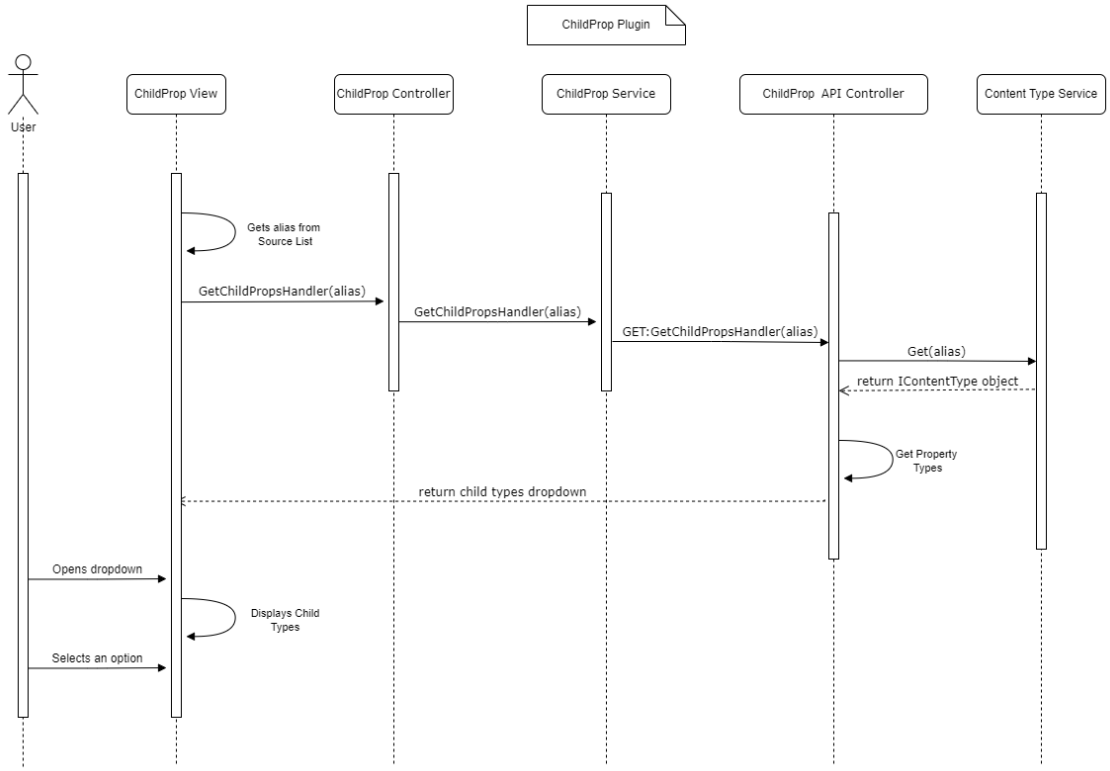

Figure 3.6: Child Prop Plugin

The second plugin depends on the use of the previous plugin, so once the user has selected the source list and child type, this plugin will retrieve all properties associated with that document type. As illustrated in the sequence diagram in Figure 3.6, the plugin will use the content type service to provide the properties for that document type and present them in a dropdown menu, allowing the user to bind each property as intended.

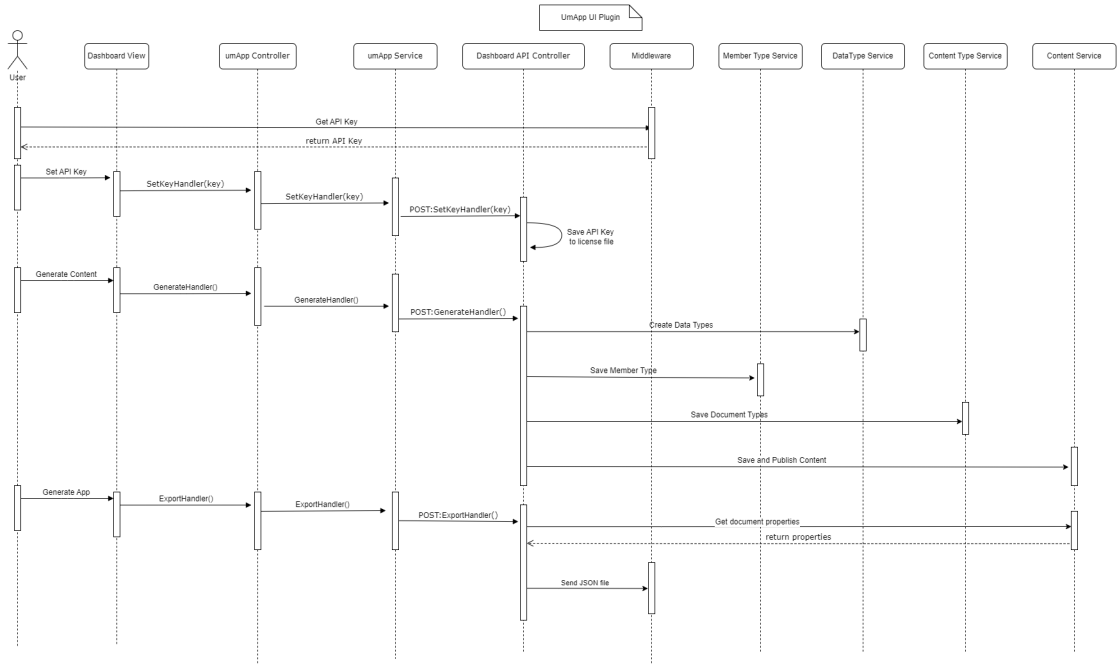

Figure 3.7: UmApp UI Plugin

Finally, Figure 3.7 illustrates the functionality of the main plugin, which allows the user to interact with this package. Initially, the user needs to obtain a unique API key from the Middleware and set it in the Umbraco backoffice. This key is stored in a license file, making it accessible to all company members and simplifying authentication for each request generated. Once the key is successfully set, the user can generate all necessary document types and data types to customize the mobile application using various Umbraco services. After customizing the mobile application, a new app can be generated, which will extract all properties from the document types and create a JSON file with the required configuration. This file is then sent to the Middleware, which will validate the request and initiate a pipeline to build and generate the new mobile application.

3.1.3 Implementation

These plugins were developed using Visual Studio 2022 and, for development and testing purposes, a new Umbraco project was created so that the plugins could be directly inserted in the Umbraco backoffice. The main plugin is the most complex and is composed by two different pages. One main page where the user can interact with the plugin and generate new apps and another where it is possible to download the source code and mobile apps. This part will describe the necessary steps to replicate this solution and provide a guide with code examples to help the community develop their own plugins. At present, Umbraco documentation is still incomplete and does not offer detailed information specifically about plugins.

As mentioned above, it is possible to define the content of a page using the html and css files. Umbraco provides some native components like *umb-box* and *umb-buttons* so the following example illustrates the view for setting a new API Key.

```
<umb-box class="umApp-group-box" ng-class="{ 'hidden': vm.export.generating }">
    <umb-box-content>
        <div class="umApp-group-box-title">
            <h2>1</h2>
        </div>
        <div class="umApp-group-box-title">
            <div class="umApp-icon">
                <svg width="33" height="32" viewBox="0 0 33 32" fill="none"
    xmlns="http://www.w3.org/2000/svg">
                    ...
                </svg>
            </div>
            <h2>Api Key</h2>
            <div class="umApp-icon-check" ng-class="{ 'hidden':
    !vm.apiKey.hasKey }">
                <svg width="25" height="24" viewBox="0 0 25 24" fill="none"
    xmlns="http://www.w3.org/2000/svg">
                    ...
                </svg>
            </div>
        </div>

        <div class="umApp-group-buttons">
            <div class="form-input">
                <form name="apiKeyForm" ng-submit="vm.setKey(vm.apiKey.key)">
                    <input class="key-input" placeholder="Please insert your
    API Key" ng-model="vm.apiKey.key"
    ng-pattern="/^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}$/"
    required />
                    <umb-button button-style="action" type="submit"
    ng-disabled="apiKeyForm.$invalid || vm.working === true" label="Set"
    state="group.state">
                    </umb-button>
                </form>
            </div>
            <span class="feedback-message" ng-class="{ 'green-text':
    vm.apiKey.id === 2, 'red-text': vm.apiKey.id === 3
    }">{{vm.apiKey.feedback}}</span>
        </div>
    </umb-box-content>
</umb-box>
```

Angular allows us to bind actions directly to buttons, with those actions being configured in the controller associated with the view. Also, it's possible to make front-end validations for expected inputs. In this scenario, visual effects such as the color of the feedback message and the check icon when the key is successfully set, but also validations for the input that is expected to be a GUID. When the form is submitted, a action "setKey" from the controller is triggered sending the input as a parameter.

```
function umAppController($scope, $q, $controller, $location, $window,
    notificationsService, umAppService, umAppHub, $rootScope) {
    var vm = this;
    ...

    function setKey(key) {
        if (vm.working === true) return;

        umAppService.setKeyHandler(getClientId(), key).then(function (results) {
            if (results.data.success) {
                vm.apiKey.id = 2;
                vm.apiKey.feedback = "Api Key set!";
                vm.apiKey.hasKey = true;
            } else {
                vm.apiKey.id = 3;
                vm.apiKey.feedback = "Please insert a valid key!";
            }
        }, function (error) {
            vm.apiKey.id = 3;
            vm.apiKey.message = "Error";
            notificationsService.error('Error', error.data.ExceptionMessage ??
    error.data.exceptionMessage);
        });
    }
    function verifyKey() {
        if (vm.working === true) return;
        umAppService.verifyStepsHandler(getClientId()).then(function (results) {
            if (results.data.success) {
                vm.apiKey.key = results.data.apiKey;
                vm.apiKey.hasKey = results.data.hasApiKey;
                vm.content.hasContent = results.data.hasContent;
            }
        }, function (error) {
            vm.status.id = 3;
            vm.status.message = "Error";
            notificationsService.error('Error', error.data.ExceptionMessage ??
    error.data.exceptionMessage);
        });
    }
}

angular.module('umbraco')
    .controller('umAppController', umAppController);
```

Within the UmAppController, which manages the main plugin's behavior, there are two functions associated with the API key. The setKey() function transmits the specified API key to the Umbraco backend for storage and future use in Middleware requests. The verifyKey() function ensures that the user does not need to re-enter the API key each time the browser is refreshed or the user logs off. These functions utilize the UmAppService, which handles HTTP requests to the backend and returns the data. The following example demonstrates two post requests to the server:

```
(function () {
    'use strict';

    function umAppService($http, $q) {

        var serviceRoot = Umbraco.Sys.ServerVariables.umApp.umAppService;

        var service = {
            setKeyHandler: setKeyHandler,
            verifyStepsHandler: verifyStepsHandler,
        };

        return service;

        function setKeyHandler(clientId, key) {
            return $http.post(serviceRoot + 'SetKeyHandler', {
                clientId: clientId,
                key: key,
            });
        }

        function verifyStepsHandler(clientId) {
            return $http.post(serviceRoot + 'VerifyStepsHandler', {
                clientId: clientId,
            });
        }

    }

    angular.module('umbraco.services')
        .factory('umAppService', umAppService);

})();
```

Ultimately, this request is directed to the backend, where a C# controller contains the necessary endpoints to handle these requests. The following example illustrates the endpoints for configuring and validating the key:

```
[PluginController("umApp")]
public class UmAppDashboardApiController : UmbracoAuthorizedJsonController
{
    private readonly IDashboardManager _dashboardManager;

    [ActivatorUtilitiesConstructor]
    public UmAppDashboardApiController(
        IDashboardManager dashboardManager)
    {
        _dashboardManager = dashboardManager;
    }
    [HttpPost]
    public UmAppActionResultViewModel SetKeyHandler(UmAppRequestViewModel
umAppRequest)
    {
        string host = $"{Request.Scheme}://{Request.Host}";
        Guid.TryParse(umAppRequest.Key, out var result);
        if (result == Guid.Empty)
        {
            return new UmAppActionResultViewModel { Success = false };
        }
        var serviceResult =
_dashboardManager.SetApiKey(umAppRequest.ClientId, host, umAppRequest.Key);
        if (serviceResult.Result.Succeeded)
        {
            return new UmAppActionResultViewModel() { Success = true };
        }
        return new UmAppActionResultViewModel() { Success = false,
ErrorDescription = serviceResult.Result.ErrorDescription };
    }
    [HttpPost]
    public UmAppStepsResultViewModel
VerifyStepsHandler(UmAppRequestViewModel umAppRequest)
    {
        string host = $"{Request.Scheme}://{Request.Host}";
        var serviceResult =
_dashboardManager.VerifySteps(umAppRequest.ClientId, host);
        if (serviceResult.Result.Succeeded)
        {
            UmAppStepsResultViewModel result =
UmAppStepsResultViewModel.FromBusinessModel(serviceResult.Result.Data);
            result.Success = true;
            return result;
        }
        return new UmAppStepsResultViewModel() { Success = false,
ErrorDescription = serviceResult.Result.ErrorDescription };
    }
}
```

In this controller, it is crucial to make certain configurations. The attribute [PluginController("umApp")] is essential for enabling communication between the plugin's frontend and backend. Additionally, this project is based in the Dependency Injection design pattern from C#, which enhances its flexibility, reusability, and maintainability for future developments. By injecting the dashboardManager into this controller, it ensures that the controller remains free of business logic, handling only the necessary validations and data transformations for the response.

The Dashboard Manager contains all the business logic of the plugin, for this particular case of saving API keys we can see these two methods:

```
    public async Task<Models.OperationResult<bool>> SetApiKey(string clientId,
    string apiKey)
{
    return await ExecuteOperationAsync(async () =>
    {
        _clientId = clientId;
        string folderPath = Path.Combine("umbraco", "Licenses");
        if (Directory.Exists(folderPath))
        {
            _logger.Warning("The directory already exists.");
            CreateApiKeyFile(folderPath, apiKey);
        }
        else
        {
            try
            { Directory.CreateDirectory(folderPath);
                _logger.Information("The directory has been created
    successfully.");
                CreateApiKeyFile(folderPath, apiKey);
            }
            catch (Exception ex)
            {
                _logger.Error($"Error creating directory: {ex.Message}");
            }
        }
        return Success(true);
    });
}
public void CreateApiKeyFile(string folderPath, string apiKey)
{
    string filePath = Path.Combine(folderPath, "umApp.lic");
    try
    {
        System.IO.File.WriteAllText(filePath, apiKey);
        _logger.Information("Text file created successfully.");
    }
    catch (Exception ex)
    {
       _logger.Error($"Error creating text file: {ex.Message}");
    }
}
```

According to Umbraco best practices, license files should be stored in a Licenses folder within the Umbraco root directory. An API key, which functions as a license to authenticate the use of a plugin, should be stored in this way to ensure it is accessible to every user within the organization. Therefore, the previous code will create the Licenses folder if it does not already exist and generate a file named umApp.lic to store the key.

To verify which steps have already been completed, the following code will attempt to retrieve the value of the license and the Umbraco container. The Umbraco container represents a folder in the backoffice that contains all the document types and data types necessary for building mobile apps.

```
    public async Task<Models.OperationResult<StepsBusinessModel>>
    VerifySteps(string clientId)
{
    return await ExecuteOperationAsync(async () =>
    {
        var result = new StepsBusinessModel();
        var license = LicenceHelper.GetLicense();
        if (license.HasValue)
        {
            result.ApiKey = license.Value.ToString();
        }
        result.HasApiKey = result.ApiKey != null ? true : false;

        EntityContainer container = ContentTypeService.GetContainers("umApp",
    1)?.FirstOrDefault();

        result.HasContent = container != null;

        return Success(result);
    });
}
```

Other examples, like Generate Content and Start Generating App follow the same flow with similar views, controllers and services. But they do differ in the functionality of the backend controller which will have to either generate every document type, data type and member type needed or extract the mobile structure to request a new mobile app generation.

Diving into the process of generating all the needed content, we can approach this from three different angles.

- **Creating Data Types**

  Using the Data Type Service from Umbraco API it is possible to create new data types or validate if certain data types exists. This proves to be helpful because the plugin uses native data types from Umbraco like ColorPickers and MediaPickers and we need to make sure they exist otherwise some functionalities do not work as intended. For those cases, the following code sample will make sure the data type exists and if not, create a new one:

```
    IDataType mediaPicker =
    DataTypeService.GetDataType("Umbraco.MediaPicker3");
if (mediaPicker == null)
{
    _propertyEditorCollection.TryGet("Umbraco.MediaPicker3", out
    IDataEditor mediaPickerEditor);
    var datatype = new DataType(mediaPickerEditor,
    _configurationEditorJsonSerializer)
    {
        Name = "Umbraco.MediaPicker3",
        DatabaseType = ValueStorageType.Ntext,
    };
    DataTypeService.Save(datatype);
}
```

  For the cases that we need to initialize data types custom made, those are the cases of umAppSourceList and umAppChildProps, the logic is the same, only need to reference their alias as showed in the next example:

```
    IDataType sourceList =
    DataTypeService.GetDataType("umAppSourceListEditor");
if (sourceList == null)
{
    _propertyEditorCollection.TryGet("umAppSourceListEditor", out
    IDataEditor sourceListEditor);
    var datatype = new DataType(sourceListEditor,
    _configurationEditorJsonSerializer)
    {
        Name = "umAppSourceListEditor",
        DatabaseType = ValueStorageType.Ntext,
    };
    DataTypeService.Save(datatype);
}
```

- **Creating Document Types**

  In this step, document types will be created and stored in a folder named UmApp. This organization ensures that all content generated by this plugin will be centralized, making it easier to locate and use. To create a new folder in the document types section, we can utilize the ContentType Service provided by the Umbraco API:

```
EntityContainer container = ContentTypeService.GetContainers("umApp",
    1)?.FirstOrDefault();

if (container == null)
{
    var attempt = ContentTypeService.CreateContainer(-1, Guid.NewGuid(),
    "umApp");
    var result = attempt.Result.Entity;
    ContentTypeService.SaveContainer(result);
    container = ContentTypeService.GetContainers("umApp",
    1)?.FirstOrDefault();
}
```

  Once the container is ready, the process of creating document types can start, for this example we will use the umAppCardsGrid that represents a grid of cards in the mobile app:

```
IContentType cardsGridCt = ContentTypeService.Get("umAppCardsGrid");
if (cardsGridCt == null)
{
    _hubContext.Clients.Client(_clientId).SendAsync("Add", "Create Cards
   Grid");
    cardsGridCt = new ContentType(ShortStringHelper, container.Id) {
   Alias = "umAppCardsGrid", Icon = "icon-list" };
    cardsGridCt.Name = "umAppCardsGrid";
    cardsGridCt.AllowedAsRoot = false;

    //Properties
    PropertyType sourceAndType = new PropertyType(stringhelper,
   "umAppSourceListEditor", ValueStorageType.Nvarchar);
    sourceAndType.Name = "Source and Type";
    sourceAndType.Alias = "sourceAndType";

    PropertyType cardTitle = new PropertyType(stringhelper,
   "umAppChildPropsEditor", ValueStorageType.Nvarchar);
    cardTitle.Name = "Card Title";
    cardTitle.Alias = "cardTitle";

    PropertyType span = new PropertyType(stringhelper, "Umbraco.TextBox",
   ValueStorageType.Nvarchar);
    span.Name = "Span";
    span.Description = "Number of columns or rows";
    span.Alias = "span";
    span.Mandatory = true;
```

```
        span.MandatoryMessage = "You have to define the number of columns or
    rows";
        span.ValidationRegExp = "^[1-4]$";
        span.ValidationRegExpMessage = "The number must be between 1 and 4";

        cardsGridCt.AddPropertyGroup("Properties", "Properties");
        cardsGridCt.AddPropertyType(sourceAndType, "Properties");
        cardsGridCt.AddPropertyType(cardTitle, "Properties");
        cardsGridCt.AddPropertyType(span, "Properties");

        ContentTypeService.Save(cardsGridCt);
    }
```

In this example, both umApp custom plugins are demonstrated, along with a property called span that indicates the number of columns or rows in the grid. This property is required and includes a Regex validation. Finally, the content type service will save this document type in the Umbraco database.

We can also take a look to the umApp document type that represents the mobile app, this instance not only creates a document type in the backoffice, but also creates a node in the Content section so that the user can start personalizing his mobile app. We can achieve this by using the ContentService as shown in the next lines of code:

```
    var appNode = ContentService.Create("umApp", -1, "umApp");
    ContentService.SaveAndPublish(appNode);
```

- **Creating Member Type**

  This plugin enables users to generate a mobile application featuring an authentication system that uses Umbraco's Member Service for user verification. To prevent the necessity of re-registering each user under a distinct member type, we utilize the default member type in Umbraco, called Member, and introduced a new set of properties specific to the umApp plugin. These properties allow users to choose if they want to receive newsletters and consent to share their personal information.

```
IMemberType umAppMember = _memberTypeService.Get("Member");
if (umAppMember != null)
{
    if (!umAppMember.PropertyTypeExists("personalData"))
    {
        umAppMember.AddPropertyGroup("umAppPrivacy", "umApp Privacy");
        IDataType dataType =
    DataTypeService.GetByEditorAlias("Umbraco.TrueFalse").FirstOrDefault();

        IPropertyType personalDataPt = new PropertyType(ShortStringHelper,
    dataType);
        personalDataPt.Alias = "personalData";
        personalDataPt.Name = "Personal Data Share";
        umAppMember.AddPropertyType(personalDataPt, "umAppPrivacy");
```

```
            IPropertyType newsletterPt = new PropertyType(ShortStringHelper,
        dataType);
            newsletterPt.Alias = "newsletter";
            newsletterPt.Name = "Newsletter";
            umAppMember.AddPropertyType(newsletterPt, "umAppPrivacy");
            _memberTypeService.Save(umAppMember);
        }
    }
```

Once the content is generated, the user can customize their mobile application and initiate the app generation process. This action triggers an API endpoint that performs several validations, including checking for a node with the umApp alias in the content tree and verifying if an API key is stored. Upon successful validation, a recursive function is executed to traverse all nodes beneath the umApp node.

```
    private void GetDocumentTypeAndChildren(int documentTypeId,
    ExportAppBusinessModel exportApp)
{
    IContent documentType = ContentService.GetById(documentTypeId);
    if (documentType != null)
    {
        IEnumerable<IContent> children;
        switch (documentType.ContentType.Alias)
        {
            case "umApp":
                ...
                break;

            case "umAppPage":
                ExportPageBusinessModel pageModel = new
    ExportPageBusinessModel() { ChildrenComponents = new
    List<ExportComponentBusinessModel>(), Properties = new Dictionary<string,
    string>() };
                pageModel.Id = documentTypeId;
                pageModel.Name = documentType.Name;
                pageModel.Type = documentType.ContentType.Alias;
                exportApp.Pages.Add(pageModel);

                children = ContentService.GetPagedChildren(pageModel.Id, 0,
    int.MaxValue, out _).ToList();
                bool hasChildren = children.Any();
                if (hasChildren)
                {
                    children.ForEach(child =>
                    {
                        GetChildrensFromPage(child, pageModel, exportApp, host);
                    });
                }
                break;

            ...
```

```
        }

        children = ContentService.GetPagedChildren(documentTypeId, 0,
    int.MaxValue, out _).ToList();
        bool hasChildren2 = children.Any();
        if (hasChildren2)
        {
            children.ForEach(child =>
            {
                GetDocumentTypeAndChildren(child.Id, exportApp, host);
            });
        }
    }
}
```

In this example, we demonstrate how to utilize Umbraco's services to obtain the properties of each node and the list of child nodes. This enables us to map the entire structure of the mobile application and recursively process it until completion. Additionally, a new function has been introduced to fetch the nodes nested within a page, thereby maintaining the original structure.

```
    private void GetChildrensFromPage(IContent child, ExportPageBusinessModel
    exportPage, ExportAppBusinessModel exportApp)
{
    ExportComponentBusinessModel componentModel = new
    ExportComponentBusinessModel() { Properties = new Dictionary<string,
    string>() };
    componentModel.Id = child.Id;
    componentModel.Name = child.Name;
    componentModel.Type = child.ContentType.Alias;

    foreach (var propertyGroup in child.Properties)
    {
        if (propertyGroup.PropertyType.PropertyEditorAlias ==
    "Umbraco.MediaPicker3")
        {
            var content = UmbracoHelper.Content(child.Id);
            if (content != null)
            {
                var value =
    content.Value<IPublishedContent>(propertyGroup.PropertyType.Alias);
                if (value != null)
                {
                    var url = $"{host}{value.Url()}";
                    componentModel.Properties[propertyGroup.PropertyType.Alias]
    = url;
                }
            }
        }
```

```
        else if (propertyGroup.PropertyType.PropertyEditorAlias ==
    "Umbraco.MultiNodeTreePicker")
        {
            ...

        }
        else
        {
            if (propertyGroup.PropertyType.PropertyEditorAlias ==
    "umAppSourceListEditor")
            {
                var childValue = child.GetValue(propertyGroup.Alias);
                var childId = int.Parse(childValue.ToString().Split("|")[0]);
                var content = ContentService.GetById(childId);
                IContentType childType =
    ContentTypeService.Get(childValue.ToString().Split("|")[1]);
                componentModel.Properties["sourceListId"] =
    content.Key.ToString();
                componentModel.Properties["childType"] =
    childType.Id.ToString();

            ...

            }
            else
            {
                var value = child.GetValue(propertyGroup.Alias);
                componentModel.Properties[propertyGroup.PropertyType.Alias] =
    value != null ? value.ToString() : "";
            }
        }
    }

    exportPage.ChildrenComponents.Add(componentModel);
}
```

In this example, we can see how different data types have unique methods for retrieving information. For instance, in the case of a Umbraco.MediaPicker3, which represents media content such as images or videos, the relevant information is the published URL, which can be used as a pointer in the mobile app. For custom plugins, like umAppSourceListEditor, a different extraction technique is required depending on how the plugin stores information in the Umbraco database. In this case, the data is stored in the format "id | childtype," so we must split the string and assign the values to the appropriate properties.

After finishing, the structure is converted into a JSON object and transmitted to the Middleware. A POST request is made with the JSON object and the API key.

Now that the Umbraco package is working, the next challenge is to pack it into a Nuget package. Visual Studio provides an option for packing a project on Build but it is also possible using the dotnet command:

```
dotnet pack UmApp.Nuget.csproj
```

Instead of including everything from a typical Umbraco project (like Startup, Program files, and the umbraco folder), the NuGet package only focuses on the essential parts for its functionality. These essential parts likely include plugins, controllers, and models. To achieve this, a separate class library was created containing only the necessary code for the package. This method simplifies the package and eliminates unneeded elements. Figure 3.8 shows the structure of the final solution.

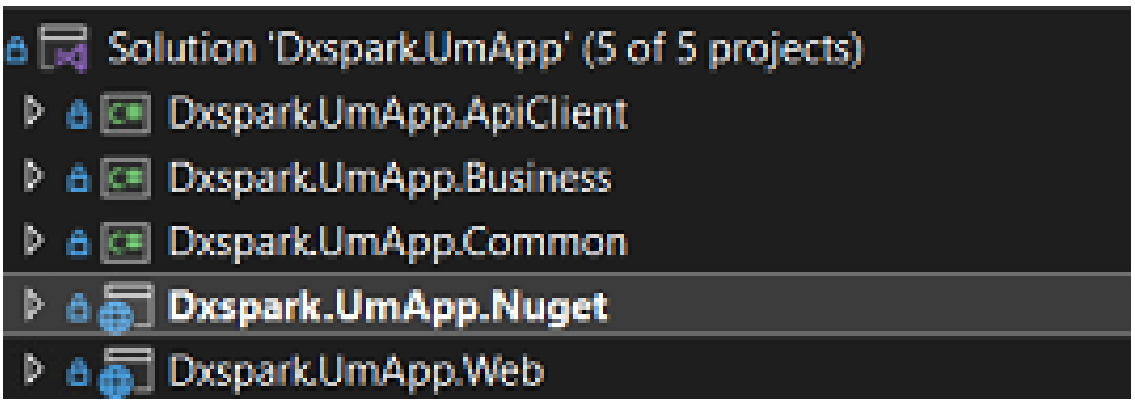

Figure 3.8: UmApp Solution

So, the Nuget project includes the controllers and plugins but it still needs all the business logic and communication to the Middleware that are stored in the other projects like Business, Common and ApiClient. To achieve this, when the Nuget is beeing created, it also needs to include the .dlls of the other projects marked as dependencies. To achieve this, some changes to the .csproj of the main project Nuget need to be made:

```
    <ItemGroup>
  <ProjectReference
    Include="..\Dxspark.UmApp.ApiClient\Dxspark.UmApp.ApiClient.csproj"
    PrivateAssets="All" />
  <ProjectReference
    Include="..\Dxspark.UmApp.Business\Dxspark.UmApp.Business.csproj"
    PrivateAssets="All" />
  <ProjectReference
    Include="..\Dxspark.UmApp.Common\Dxspark.UmApp.Common.csproj"
    PrivateAssets="All" />
</ItemGroup>


<PropertyGroup>
 <TargetsForTfmSpecificBuildOutput>$(TargetsForTfmSpecificBuildOutput);
 IncludeP2POutput</TargetsForTfmSpecificBuildOutput>
 <Version>$(VersionPrefix)</Version>
```

```
</PropertyGroup>

<Target Name="IncludeP2POutput" DependsOnTargets="ResolveReferences">
   <ItemGroup>
      <BuildOutputInPackage
    Include="@(ReferenceCopyLocalPaths-&gt;WithMetadataValue('ReferenceSourceTarget',
    'ProjectReference')-&gt;WithMetadataValue('PrivateAssets', 'all'))" />
   </ItemGroup>
</Target>

<ItemGroup>
   <None Include="build\**\*.*">
      <Pack>true</Pack>
      <PackagePath>buildTransitive</PackagePath>
   </None>
</ItemGroup>
```

Using these settings, we designate the dependency projects as private and filter for each asset marked as private to ensure that the DLLs are copied to the output/nuget directory. Then, we use the dotnet command to pack, which will result in a .nupkg file. To verify that the Umbraco plugins were copied successfully, they must appear inside a folder named staticwebassets, and we should also ensure that the DLLs appear inside the lib folder.

In the NuGet package, there will also be a .nuspec file. This XML file describes the contents of the NuGet package, including details such as its name, version, authors, description, and dependencies. Below is the .nuspec file for the UmApp NuGet package:

```
    <?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2013/05/nuspec.xsd">
  <metadata>
    <id>umApp</id>
    <version>1.0.0</version>
    <authors>Dxspark</authors>
    <description>Package Description</description>
    <repository type="git" commit="xxxxxxxxxxx" />
    <dependencies>
      <group targetFramework="net7.0">

        ...

        <dependency id="Umbraco.Cms" version="12.3.6" exclude="Build,Analyzers"
    />
      </group>
    </dependencies>
    <contentFiles>
      <files include="any/net7.0/appsettings-schema.json" buildAction="Content"
    />
      <files include="any/net7.0/appsettings-schema.Umbraco.Cms.json"
    buildAction="Content" />
    </contentFiles>
  </metadata>
```

```
</package>
```

After completing these steps, we have a local NuGet package. To make it available to the community, it needs to be uploaded to a NuGet repository, such as NuGet.org. While NuGet.org recommends that packages be licensed and signed for increased reliability, this is not mandatory. In this case, no license was included. Once uploaded, the package can be found here.

3.1.4 NuGet Installation

To install `umApp`, we need to add some configurations to our `Startup.cs` file.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddUmbraco(_env, _config)
        .AddBackOffice()
        .AddMumbraco()  // add this line
        .AddWebsite()
        .AddDeliveryApi()
        .AddComposers()
        .Build();
    services.AddMumbraco(_config);  // add this line
}
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
      app.UseDeveloperExceptionPage();
    }

    app.UseUmbraco()
        .WithMiddleware(u =>
        {
            u.UseBackOffice();
            u.UseWebsite();
        })
        .WithEndpoints(u =>
        {
            u.UseInstallerEndpoints();
            u.UseBackOfficeEndpoints();
            u.UseWebsiteEndpoints();
        });
    app.UseMumbraco(); // add this line
}
```

We also need to make sure that DeliveryApi is enabled in our `appsettings.json`:

```
{
  "$schema": "appsettings-schema.json",
  "Umbraco": {
    "CMS": {
      "DeliveryApi": {
        "Enabled": true
      },
    }
  },
}
```

Lately, Umbraco lists are built with Listviews, but at the moment they are not supported, when using the DeliveryApi.

3.2 Middleware

To optimize user and resource management within the system, the integration of a middleware is strategically positioned between the Umbraco package and the pipeline service. This middleware not only facilitates communication between these components but also provides unique API Key to users. This API key acts as an authentication credential and enables users to securely request the use of our pipelines to generate new mobile apps.

3.2.1 Analysis

This component serves dual purposes: facilitating user authentication and API key generation, and establishing a platform for managing pipelines and user accounts within our services. The first objective is to ensure secure access to our pipeline services by authenticating users and providing them with unique API keys. In addition, the platform also enables the management of pipelines and user accounts, optimizing resource distribution and access control. To enhance comprehension, the use-case diagram depicted in Figure 3.9 illustrates the potential activities available to the two user categories of this component. Here, 'User' denotes an individual associated with the Umbraco Plugin, while 'Admin' refers to the team responsible for managing pipelines and user account management.

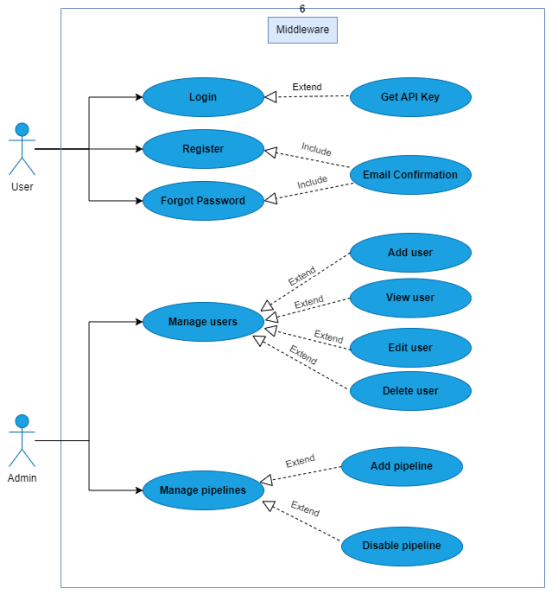

Figure 3.9: Middleware Use Case Diagram

For common users, the primary use cases include logging in, registering, and resetting their passwords. Upon logging in, users have the option to obtain an API key. The registration and password reset processes both include an email confirmation step to verify the user's email address. Administrators, on the other hand, have wider capabilities. They can manage user accounts, which comprehends adding new users, viewing user details, editing user information, and deleting user accounts. In addition, administrators can manage pipelines by adding new ones or disabling them.

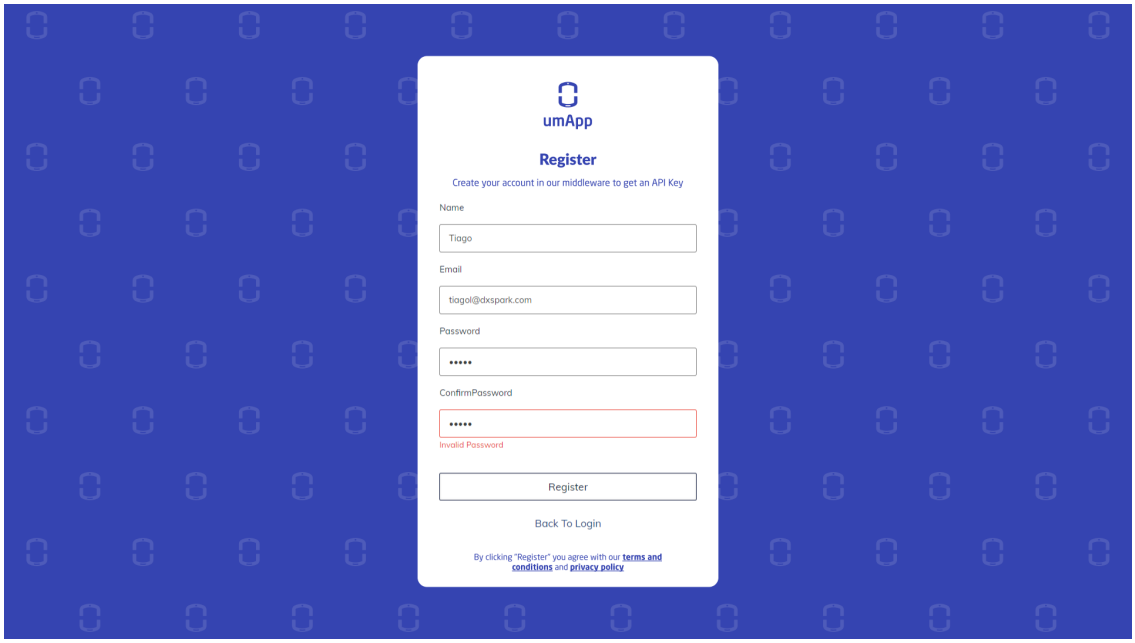

Figure 3.10: Middleware Register Page

Figure 3.10 represents a register example, which after confirming the account in the email, the user can login into their account.

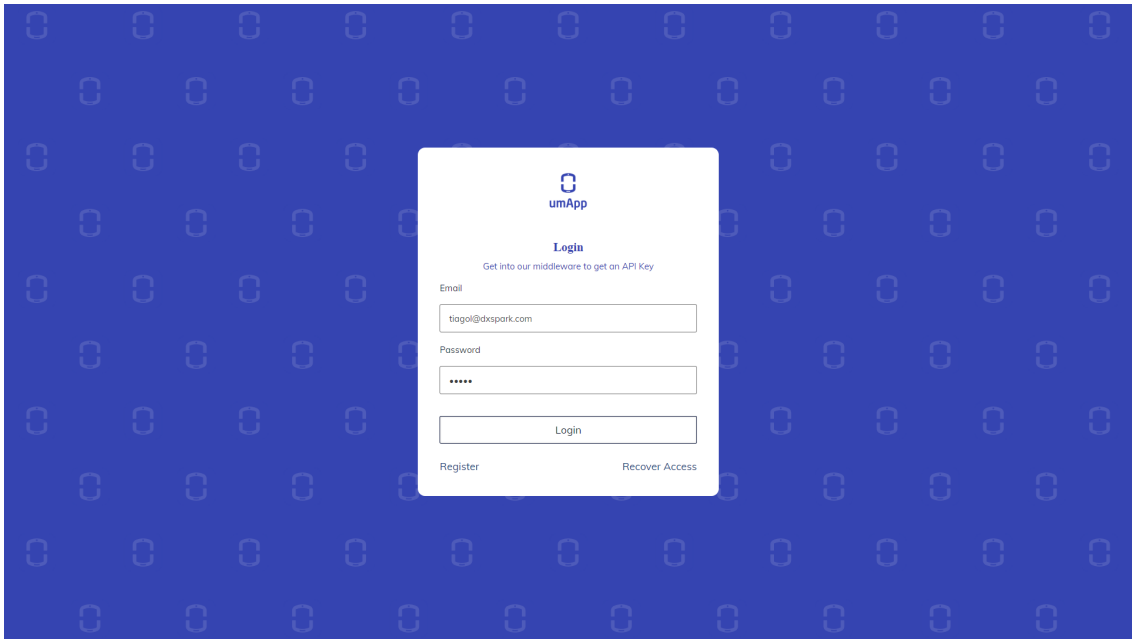

Figure 3.11: Middleware Login Page

After registration, users gain access to their accounts and can log in, as illustrated in Figure 3.11.

Upon successful authentication, users are directed to the main page, illustrated in Figure 3.12, where they are presented with their unique API Key.

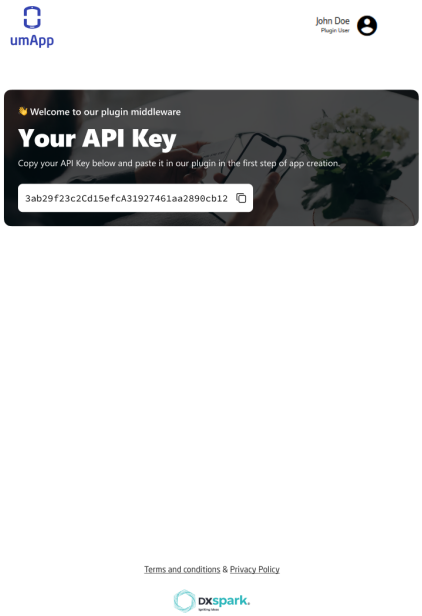

Figure 3.12: Middleware Main Page

3.2.2 Design

This component was developed in Asp.Net Core 8 so it follows the MVC architectural pattern. The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. In this company, we take a step forward and separate the application into five groups of components.

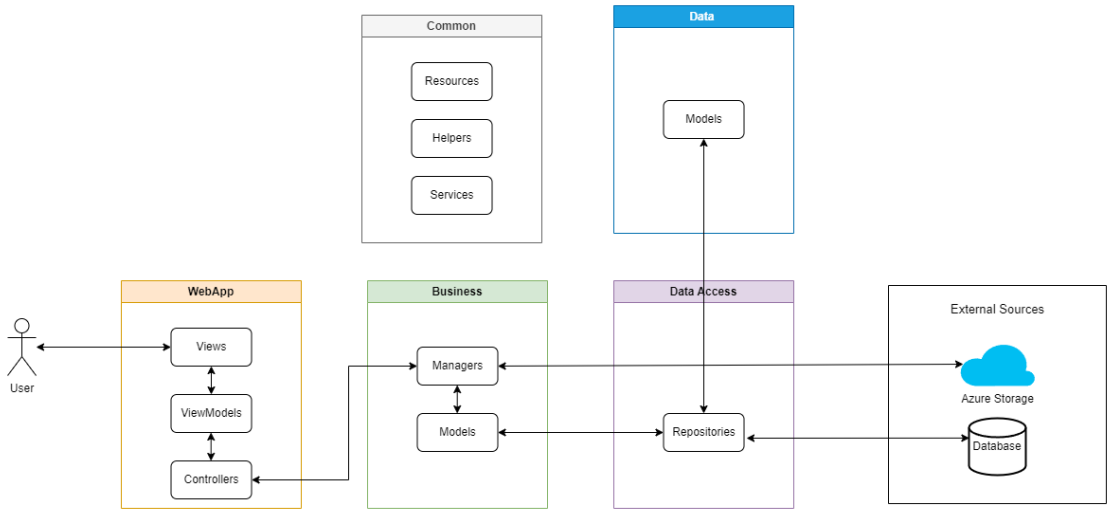

Figure 3.13: Middleware Architecture

Figure 3.13 represents the architecture of this project, and the different groups can easily be identified:

- **WebApp**

  This group contains all the views that will be available for the user to interact. Views with dynamic data require view models for processing and displaying information accurately, like formatting dates or strings. Controllers facilitate data exchange between views and business logic, updating view models as needed.

- **Business**

  The business layer hosts business managers and models, containing the core business logic. It acts as a bridge between views and external data sources such as Azure Storage or the Data Access layer.

- **Data Access**

  This layer contains repositories tasked with database interactions: inserting, updating, deleting, or retrieving data. Typically utilizing LINQ expressions, simplifying database interactions without directly executing SQL queries.

- **Data**

  The data layer contains the exact representations of the database, these models are then runned by EF Core Migrations to update the database. EF Core compares the current model against a snapshot of the old model to determine the differences and generates migration source files. Once a new migration has been generated, it can be applied to a database.

- **Common**

  The common layer contains resources, helpers, enums, and services applicable across the application. It serves as a versatile component accessible from any part of the application.

3.2.3 Implementation

The Middleware is a standard web-project with authentication and different private areas for a common user or a administrator. Starting by the authentication, this project was designed to provide different features of authentication, since login, registration and forget password. All these features are followed by an SMTP (Simple Mail Transfer Protocol) configuration that sends confirmation emails to register and recover the password.

When a new registration occurs, a unique API key is generated and linked to the user's profile. This key is displayed on the Home page, allowing the user to access and utilize it within the Umbraco plugin.

To deliver the information to the user, a new endpoint was created that returns all applications generated by a specific key in a paginated object.

In conclusion, to enable the management of users, pipelines, and applications, three distinct views were developed featuring tables and action buttons for adding, editing, or deleting entries.

3.3 Pipeline

This component consists of various pipelines designed to create mobile application solutions. Middleware manages these pipelines, which are hosted on Azure. Azure Pipelines, a cloud service from Microsoft Azure, facilitates continuous integration (CI) and continuous delivery (CD), automating the processes of building, testing, and deploying applications.

3.3.1 Analysis

In order to generate mobile application solutions, a pipeline must have the following input parameters:

- **Url**

  This is a string parameter that represents the endpoint from the Umbraco API that will provide the Json configuration file.

- **ProjName**

  This is also a string parameter defines the name of the solution generated, if no projname is provided, a default name of *UmApp* will take place.

- **AppGuid**

  A Guid is necessary to ensure that all generations are unique and no conflicts will happen when saving to the Azure storage.

- **FeedbackUrl**

  This string parameter points to the middleware endpoint and is not required because the default value already points to the production endpoint.

- **BuildGeneratedProject**

  This is a boolean parameter that indicates whether the solution must be built to generate mobile applications or not.

After generating the mobile solution, another job needs to send the information back to the provided FeedbackUrl, indicating the build state and the appGuid. This allows the Middleware to bind and register the location of the solution in Azure Storage.

3.3.2 Design

The pipeline consists of two different jobs. Job A has all the steps relative to generating and storing the solutions in Azure Storage. While Job B is responsible for sending feedback to the Middleware about Job A status and solution storage information.

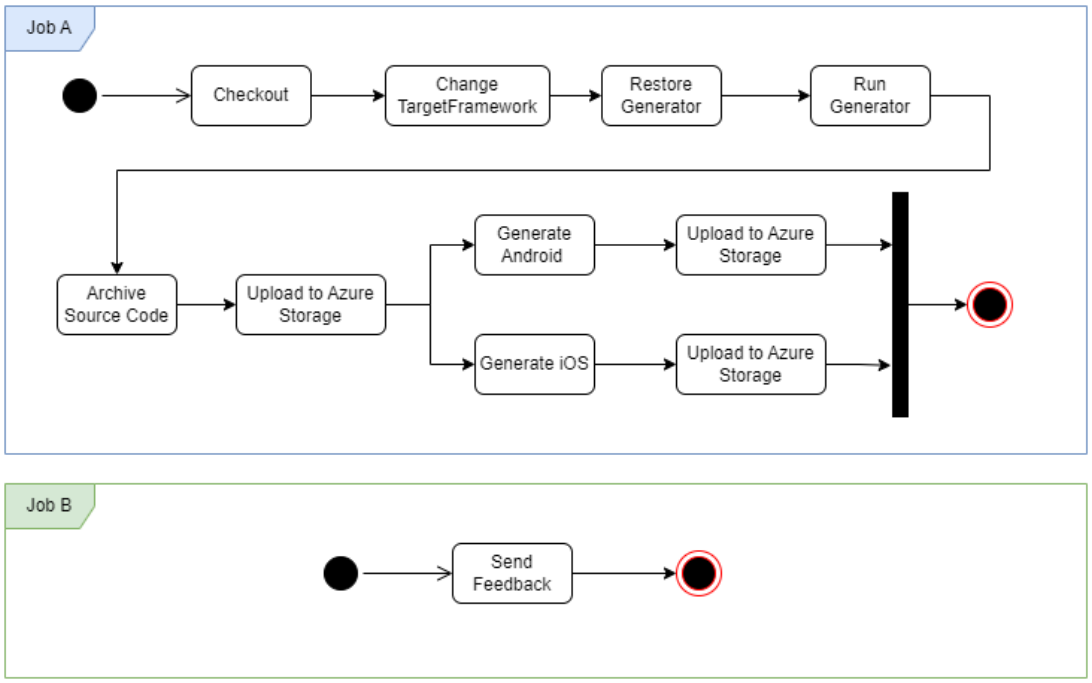

Figure 3.14: Generator Pipeline

Figure 3.14 illustrates in more detail which steps take place in each Job. Job A is composed by ten steps:

- **Checkout**

  This step will do a checkout to the Generator repository hosted in Azure.

- **Change Target Framework**

  This step ensures that the pipeline is running in the same .NET version as the MAUI solution.

- **Restore Generator**

  This step consists in building the generator so that it can be used to run the Json configuration.

- **Run Generator**

  After having the generator built, it will run with the Json configuration file and generate a MAUI solution.

- **Archive Source Code**

  This step stores the source code generated before in the pipeline artifacts.

- **Upload to Azure Storage**

  This step uploads the previous step to the Azure Storage, can be the source code, the apk, or the ipa file.

- **Generate Mobile Solutions**

  To enhance performance, the Generate Android and Generate iOS steps are executed concurrently, as they do not depend on each other.

Finally, Job B only has one step that is responsible for sending feedback to the Middleware with the Job A's build status and appGuid.

3.3.3 Implementation

This pipeline relies on the Generator as its main component, beginning with the checkout of the Generator repository. Following the checkout, it is important to confirm that the .NET version aligns with MAUI, making sure to use version 8. This can be accomplished through a PowerShell task:

```
task: PowerShell@2
    displayName: 'Change TargetFramework'
    inputs:
      targetType: 'inline'
      script: |
        (Get-Content
$(System.DefaultWorkingDirectory)\Generator\Generator.csproj -Encoding
UTF8) -replace '<TargetFramework>net7.0</TargetFramework>',
'<TargetFramework>net8.0</TargetFramework>' | Set-Content
$(System.DefaultWorkingDirectory)\Generator\Generator.csproj
```

Once this step is completed, we can build and run the Generator project with the Json configuration sent as parameter for the pipeline. We can use a DotNetCoreCLI task to build a .NET application and CmdLine to run the project with the desired configuration:

```
task: DotNetCoreCLI@2
    displayName: 'Restore Generator'
    inputs:
      command: 'restore'
      projects: '**/*.csproj'
      feedsToUse: 'config'
      nugetConfigPath: 'NuGet.config'
script: dotnet build --configuration $(buildConfiguration)
    displayName: 'Build Generator'
task: CmdLine@2
    displayName: 'Run Generator'
    inputs:
      script:
'$(System.DefaultWorkingDirectory)\Generator\bin\Release\net8.0\win-x64\Generator.exe
--url "${{ parameters.url }}"'
```

These tasks will result in a MAUI solution, that needs to be archived into a zip file and then sent to the Azure Storage. We can use the ArchiveFiles task to generate a zip file and a AzureCLI to make the connection to the storage:

```
    task: ArchiveFiles@2
        displayName: 'Archive source code'
        inputs:
          rootFolderOrFile:
    '$(System.DefaultWorkingDirectory)\Export\${{parameters.projname}}'
          includeRootFolder: true
          archiveType: 'zip'
          archiveFile: '$(Build.ArtifactStagingDirectory)/${{
    parameters.projname }}.zip'

    task: AzureCLI@2
        displayName: Upload to Azure Storage
        inputs:
          azureSubscription: 'UmAppServiceConnection'
          scriptType: 'bash'
          scriptLocation: 'inlineScript'
          inlineScript: |
            az storage blob upload --overwrite true --account-name xxxxx
    --account-key xxxxxx --container-name xxxxxx --name ${{ parameters.projname
    }}_${{ parameters.apkGuid }}.zip --file
    '$(Build.ArtifactStagingDirectory)/${{parameters.projname}}.zip'
```

To create both an Android and an iOS application, we can utilize a script to compile the MAUI solution into these apps. Starting with Android, we can use:

```
    - script: dotnet build
    $(System.DefaultWorkingDirectory)\Export\${{parameters.projname}}\${{parameters.projname}}.
    --configuration $(buildConfiguration) -f net8.0-android
    displayName: 'Build ${{parameters.projname}}.sln Android'
```

For the iOS scenario, we just need to change the -f option from the dotnet build to the iOS framework:

```
    - script: dotnet build
    $(System.DefaultWorkingDirectory)\Export\${{parameters.projname}}\${{parameters.projname}}.
    --configuration $(buildConfiguration) -f net8.0-ios
  displayName: 'Build ${{parameters.projname}}.sln iOS'
```

Once these steps are completed, a Job B will start to send the feedback back to the Middleware. Including only one CmdLine task to make a curl request:

```
job: Job_B
dependsOn: Job_A
condition: always()
variables:
  jobAStatus: $[ dependencies.Job_A.result ]
steps:
   checkout: none
   task: CmdLine@2
    displayName: 'Sending Feedback'
    inputs:
      script: |
        curl -k -X POST -i -H "Content-Type: application/json" -d
"{\"State\": \"$(jobAStatus)\", \"Id\": \"${{ parameters.apkGuid }}\"}" ${{
parameters.feedbackUrl }}
```

3.4 Mobile Generator

The Mobile Generator is a .NET solution designed to interpret a JSON configuration file and generate a MAUI solution, using Roslyn for code generation. This generator is used in the pipeline upon receiving a generation request from the Middleware.

3.4.1 Analysis

As previously mentioned, this generator is intended to accept a JSON configuration that outlines the structure of the mobile application. This structure consists of various components that the generator must identify in order to interpret and generate MAUI code accordingly. In this initial phase of development, the components available for mobile applications are:

- **umApp:** Represents the mobile application and has necessary configuration properties to the app.
- **umAppPage:** Represents a mobile page that can have child components.
- **umAppCardsGrid:** Represents a grid with the possibility to define the span, which means the number of columns or rows depending on the grid's direction.
- **umAppCarouselCards:** Represents a carousel of cards.
- **umAppCardsList:** Represents a list of cards.
- **umAppList:** Represents a list of simple elements.
- **umAppStoryCarousel:** Represents a carousel of stories.
- **umAppNavigationButton:** Represents a button that allows navigation between pages and components.
- **umAppModalButton:** Represents a custom button for a modal.
- **umAppSplashScreen:** Represents a custom image for when the app is started.
- **umAppForm:** Represents a form used to add new entries of a specific document type.

The Generator then processes these components to generate MAUI code, which includes XAML code for presenting the UI and replicating these features.

3.4.2 Design

This solution is composed by three projects, each of them has a unique purpose and Figure 3.15 details their components.

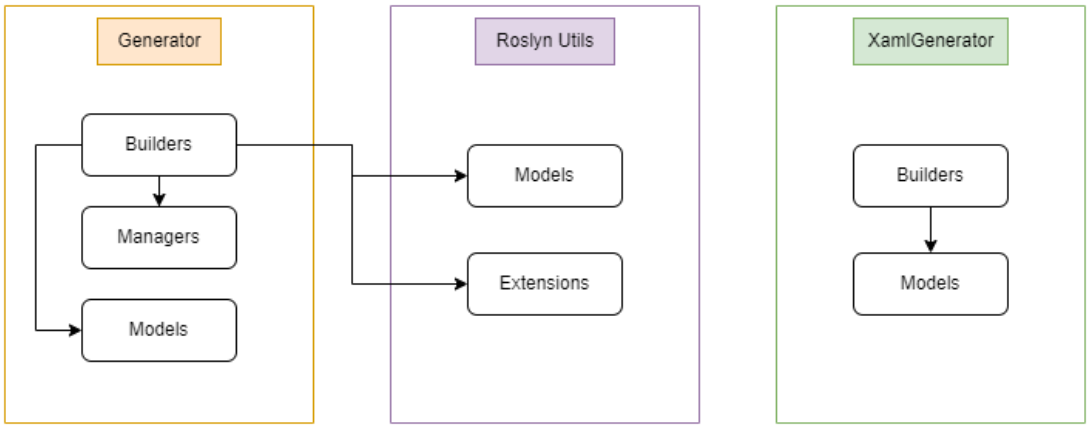

Figure 3.15: Generator Architecture

The Generator's primary role is to transform the input Json configuration into a C# object, which is then utilized for component creation. It primarily consists of builders that produce essential components for the mobile solution, such as pages, viewmodels, session managers, and the program class. These builders rely on models to create attributes for classes.

The Roslyn Utils package, developed by Agap2IT, provides a more familiar interface for generating code. It offers an intuitive method focusing on models and extensions for essential object-oriented programming components such as classes, constants, attributes, methods, parameters, and data types.

The XamlGenerator is exclusively dedicated to crafting XAML code for all components generated previously, including GridViews, Layouts, Labels, Images, and others. XAML, similar to HTML in web development, serves as a markup language within MAUI solutions. Enables the definition of user interface elements, layouts, and interactions in a platform-independent way through a declarative approach.

3.4.3 Implementation

When the Generator is invoked, the initial step involves converting the JSON file's content into a C# object. This facilitates a clearer understanding of the configuration and provides improved access to its properties.

Using a custom converter, it is possible to deserialize this information and create a structure. In the following example, we can observe how the pages of the mobile application are created:

```
    public umAppConfig? Convert(MUmbracoAppConfig? mumbracoAppConfig)
{
    if (mumbracoAppConfig == null)
        return null;

    Paths paths = new
    Paths(namingSchemeProvider.GetClassName(mumbracoAppConfig.Name));

    umAppConfig appConfig = new umAppConfig();

    appConfig.AppType = AppConfigType.UmApp;
    appConfig.AppName = mumbracoAppConfig.Name;
    appConfig.Pages = new List<Page>();
    appConfig.HasLogin = mumbracoAppConfig.HasLogin;
    appConfig.Urls["baseUrl"] = mumbracoAppConfig.BaseUrl;
    appConfig.Urls["contentUrl"] = mumbracoAppConfig.ContentUrl;

    foreach (MUmbracoPage mumbracoPage in mumbracoAppConfig.Pages)
    {
        ViewModel viewModel = CreateViewModel(mumbracoPage, paths);
        if (!appConfig.ViewModels.Any(it => it.Name == viewModel.Name))
        {
            appConfig.ViewModels.Add(viewModel);
        }

        Page page = CreatePage(mumbracoPage);
        if (!appConfig.Pages.Any(it => it.Name == mumbracoPage.Name))
        {
            appConfig.Pages.Add(page);
        }
    }

    ...

    return appConfig;
}
```

In this instance, each page listed in the configuration will be generated along with a ViewModel unique to that page. MAUI adheres to the MVVM (Model View ViewModel) architecture, indicating that the ViewModel will handle the content displayed in the view.

To generate the necessary code for the page, we leverage the abstraction layer provided by the Roslyn project. The RoslynClassBuilder shown in the following code snippet includes a method for class creation, allowing customization with typical .NET development attributes such as constructors, usings, methods, inheritance, partial, and static classes.

```
public class RoslynClassBuilder
{
private INamingSchemeProvider namingSchemeProvider;

public RoslynClassBuilder(INamingSchemeProvider namingSchemeProvider)
{
    this.namingSchemeProvider = namingSchemeProvider;
}

public SyntaxNode CreateClass(IList<string> usings,
    string classNamespace,
    SyntaxKind classModifier,
    string className,
    IList<IAttribute> classAttributes = null,
    IList<Inheritance> inheritance = null,
    ConstructorMetaData constructor = null,
    IList<PropertyMetaData> properties = null,
    IList<FieldMetaData> fields = null,
    IList<MethodMetaData> methods = null,
    bool isPartial = false,
    bool isStatic = false,
    bool addGenericTIdentifier = false)
}
```

This procedure closely resembles the ViewModel builder since, ultimately, both are C# classes with distinct attributes and methods. Once the pages are created, it is also possible to define which one will be the starting page of the mobile application. Since that information is not coming from JSON configuration yet, the first page on the list is being selected as the starting page.

Once the C# classes are ready, the next step is to generate the XAML views for these components. .NET provides some classes that are really helpful for this situation, like XmlWriter and StringBuilder so that ultimately all the generated code will be presented in a string format.[1]

[1] As previously mentioned, the specifics of the implementation of this generator are outside the scope of this thesis and are considered confidential.

Chapter 4

Solution Validation

The final step involves validating the solution to ensure it functions as expected. While it is theoretically possible to validate each of the four components individually, their interdependencies make this approach impractical. For instance, the middleware requires the pipelines to function, the generator depends on the configuration JSON file produced exclusively by the Umbraco package, and so forth. Therefore, the most effective method to test this solution is to create a comprehensive demo. By examining the different stages of the demo, we can determine if the final outcome aligns with our expectations. For confidentiality reasons, we cannot provide a specific example from DxSpark's clients. However, we have created a demo website for this purpose, showcasing a football website.

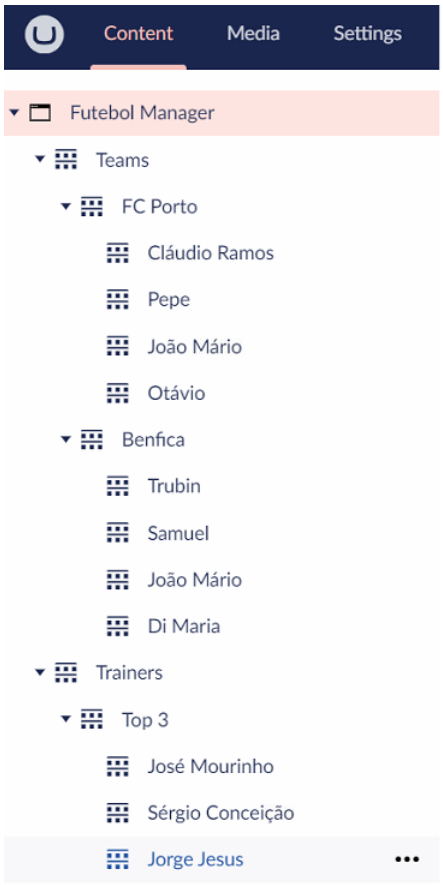

Figure 4.1: Futebol Manager Website

Figure 4.1 illustrates the website's content. Although the document types for each node are not relevant in this context, it is important to note that "Teams" and "Trainers" represent pages of the website. "FC Porto," "Benfica," and "Top 3" are lists, while the other nodes represent elements within those lists, such as team players or trainers.

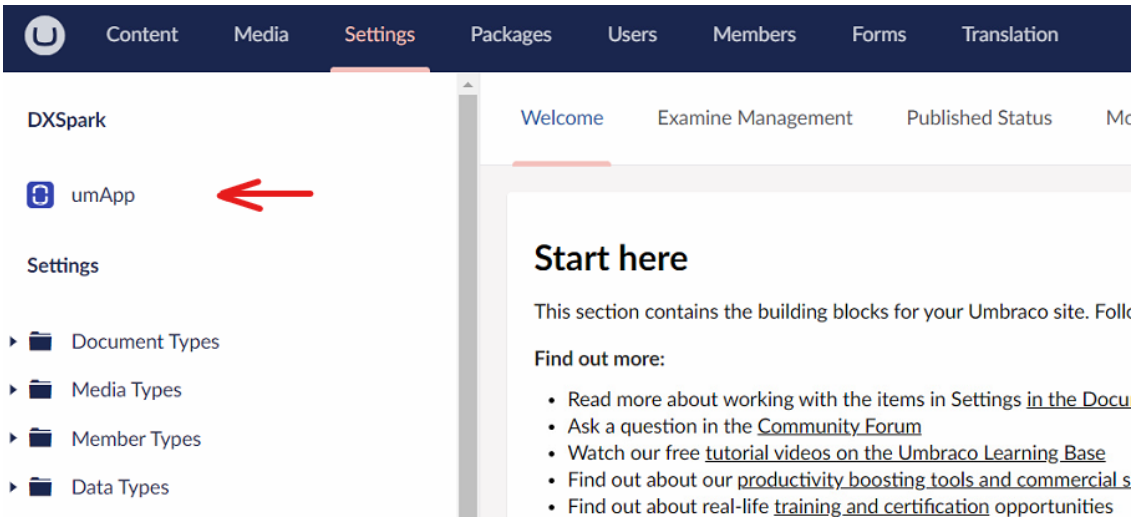

Figure 4.2: UmApp in Settings tree

Once the NuGet package is installed in the Umbraco project, additional details about the installation can be found in section 3.1.4. We can then go to the Settings tab, and if the installation was successful, a new icon named umApp will be visible, as shown in Figure 4.2.

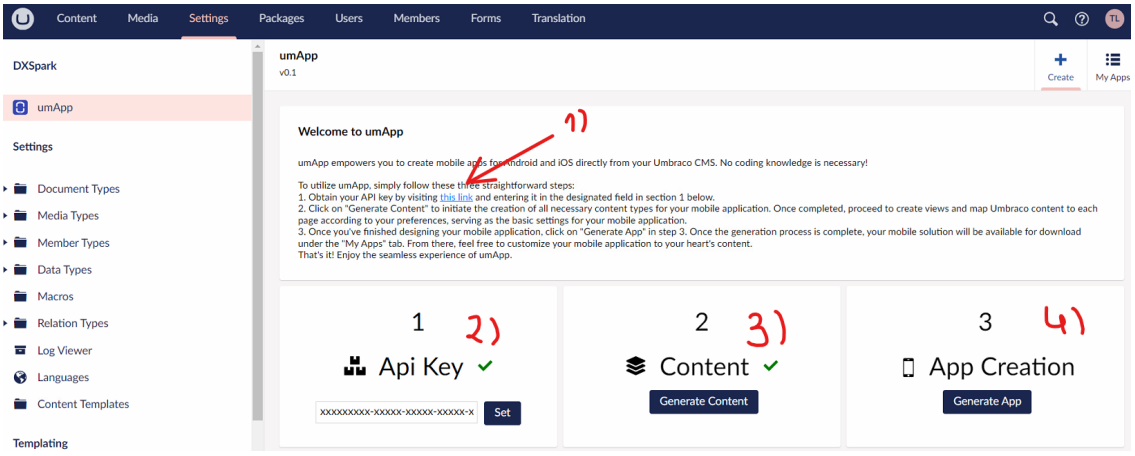

Figure 4.3: UmApp Main Page

Figure 4.3 represents the main page of the plugin, in this page we interact with most of UmApp functionalities:

- 1) By clicking this link, we go straight to the Middleware, where we can create a new account to obtain an API key.
- 2) After retrieving the key, we can set it by inserting on the input and it gets stored in Umbraco.

- 3) This action gets unlocked after setting a valid API key, once that is done, we can generate the content by clicking in the button. This results in a folder under the document types, as we can see in Figure 4.4:

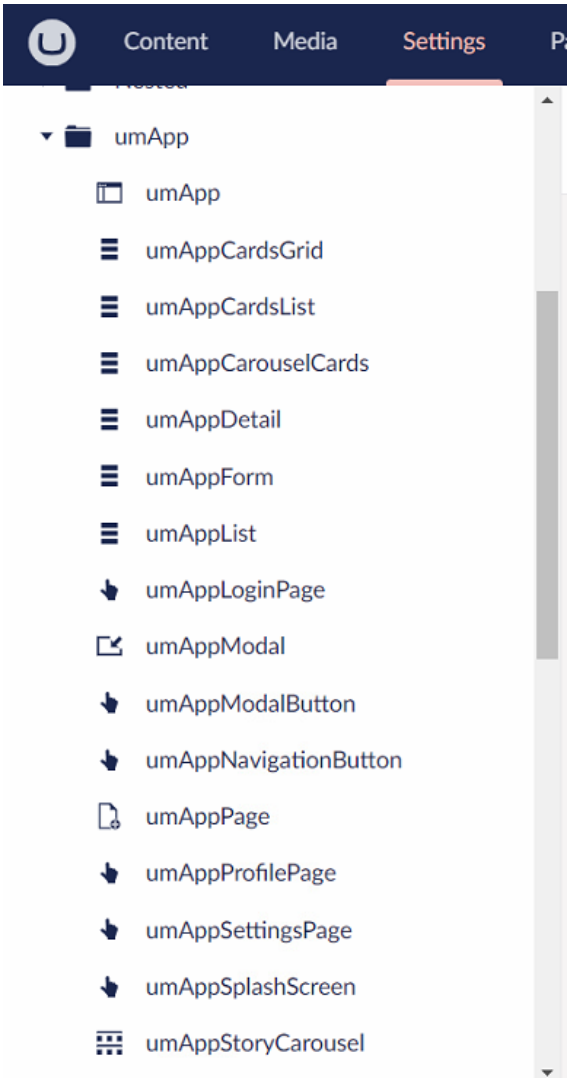

Figure 4.4: UmApp Document Types

Once the document types are ready for use, we need to structure and personalize our mobile application within the Content tree.

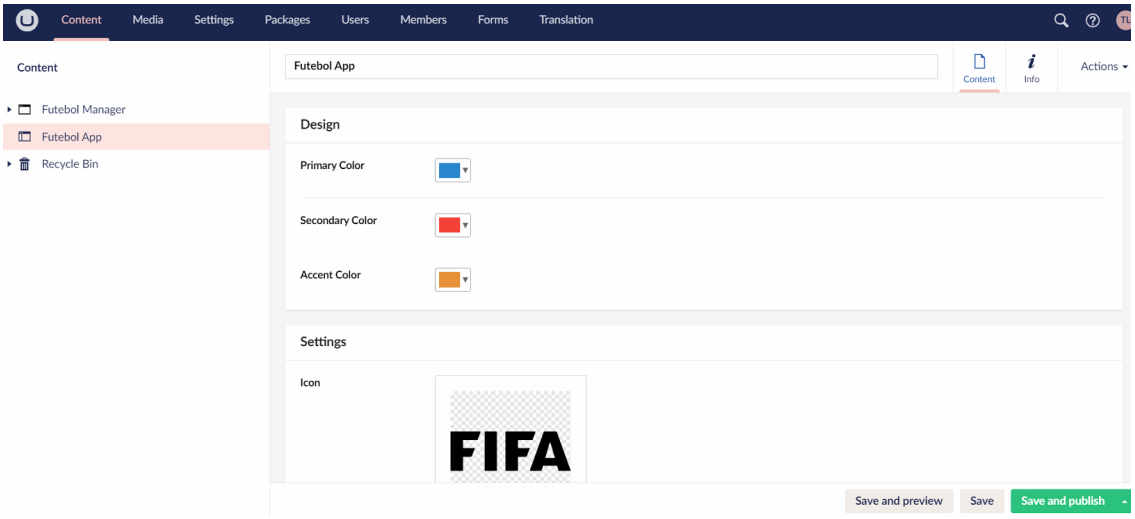

Figure 4.5: Football App

We will start by configuring the root element which holds configuration for themes, icons and if the application has authentication, as showed in Figure 4.5.

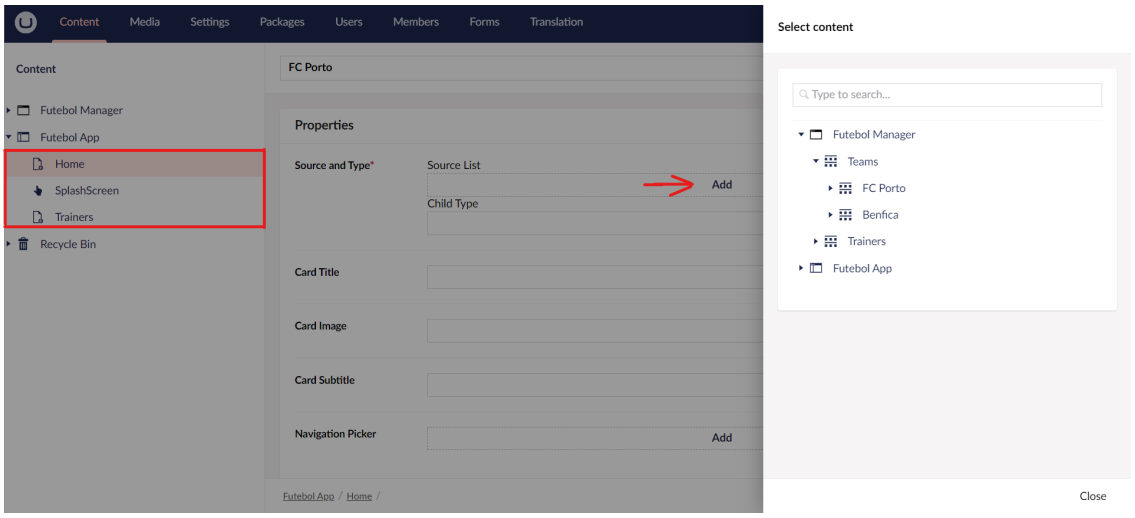

Figure 4.6: UmApp CardsList

In Figure 4.6, the left rectangle shows the creation of two pages, named "Home" and "Trainers", which represent pages in the mobile app. Additionally, a new node for configuring the SplashScreen is included, which contains an input for an image. On the remaining part of the page, we can see the creation of a CardsList component by selecting the source list from the original website through the "Add" button, which will activate a content picker from Umbraco.

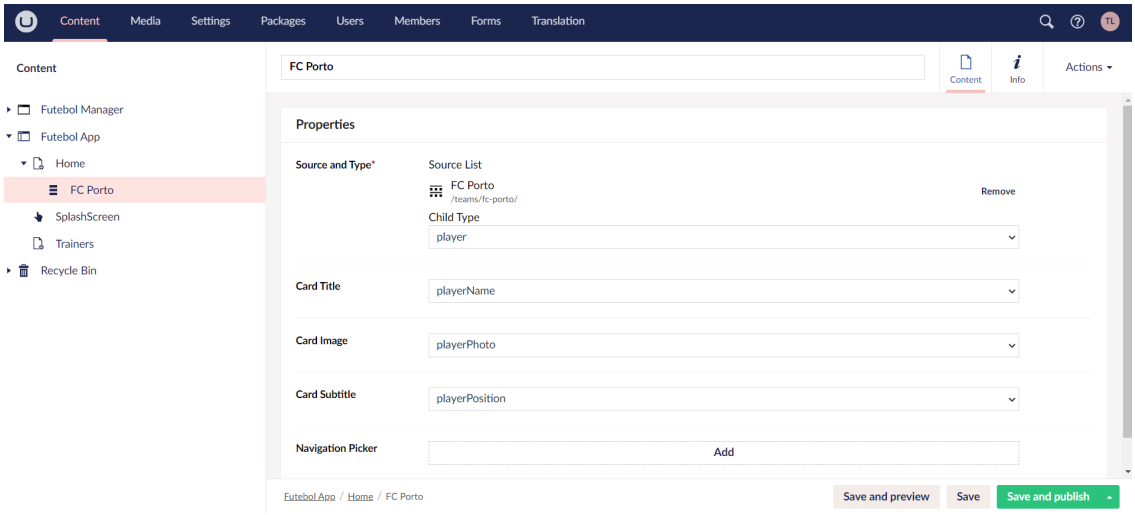

Figure 4.7: UmApp CardsList filled

Once a source list is selected, we have to choose the child type that we want to use to bind the properties. Figure 4.7 shows the component fully filled and ready to export.

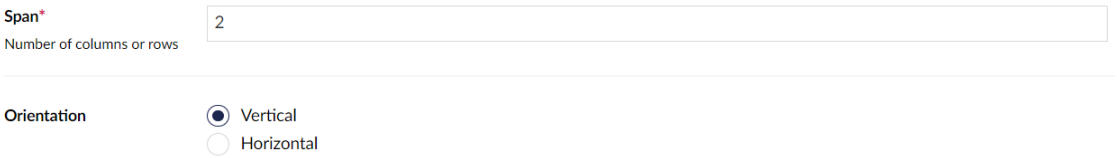

Figure 4.8: UmApp CardsGrid

To make a more comprehensive test, we also created a grid of cards to display the team "Benfica" from the original website, the configuration is pretty similar but it is also possible to configure the number of rows or columns and the orientation of the grid, as we can see in Figure 4.8.

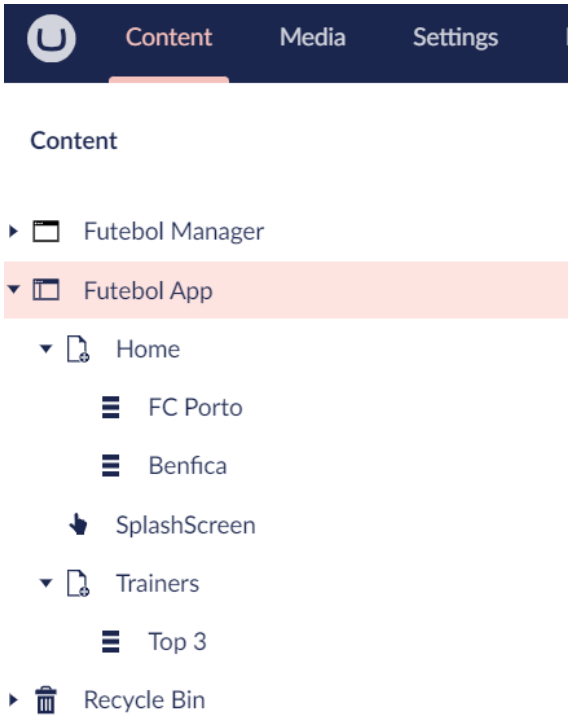

Figure 4.9: Mobile structure

After completing the remaining content, the final mobile structure is displayed in Figure 4.9. We are now prepared to begin the process of generating a new mobile app.

- 4) By pressing the "Generate App" button, the generation process starts. This action will activate the Middleware, which in turn will initiate an available pipeline to carry out the procedure.

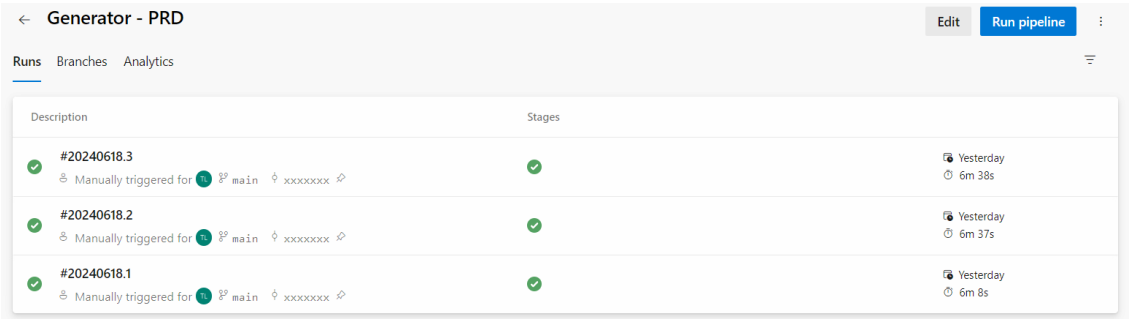

Figure 4.10: Pipeline Runs

Figure 4.10 shows the most recent three runs, from which we can infer that the pipeline requires an average of 6 minutes and 27 seconds to produce the mobile solution.

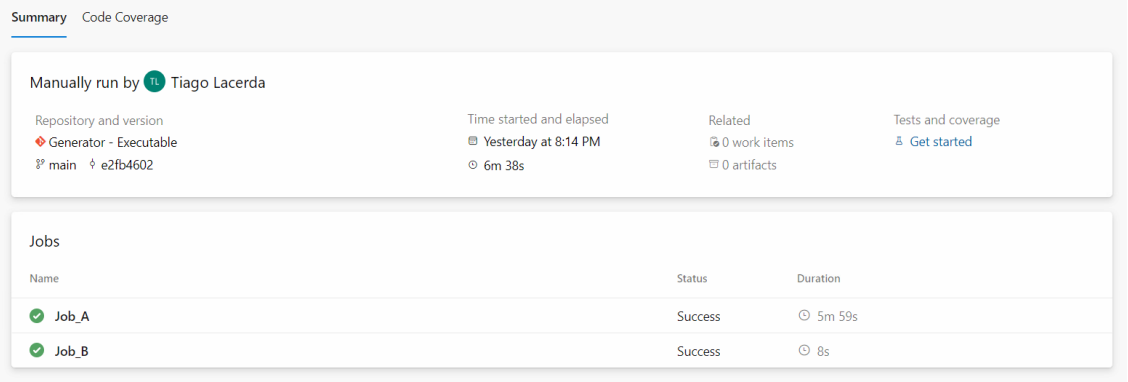

Figure 4.11: Pipeline Last Run

Figure 4.11 provides a detailed view, indicating that Job A takes approximately 6 minutes, when Job B takes less than 10 seconds to return the feedback to the Middleware.

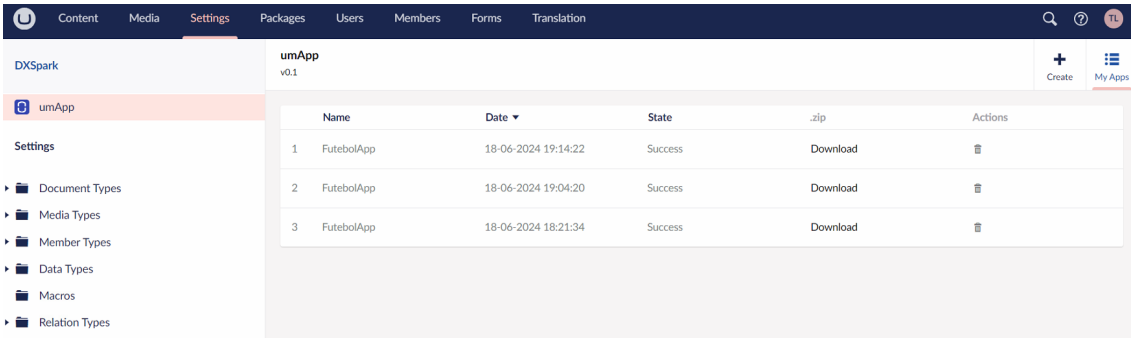

Figure 4.12: My Apps View

After the process is complete, a download link will appear in the My Apps section, as shown in Figure 4.12. This section provides feedback to the user and enables them to download the source code for the mobile solution. Currently, there are no links available for Android or iOS apps, as these features will be included in future versions of the Umbraco package.

After downloading the solution, we can utilize an emulator such as those available in Visual Studio to execute the mobile application.

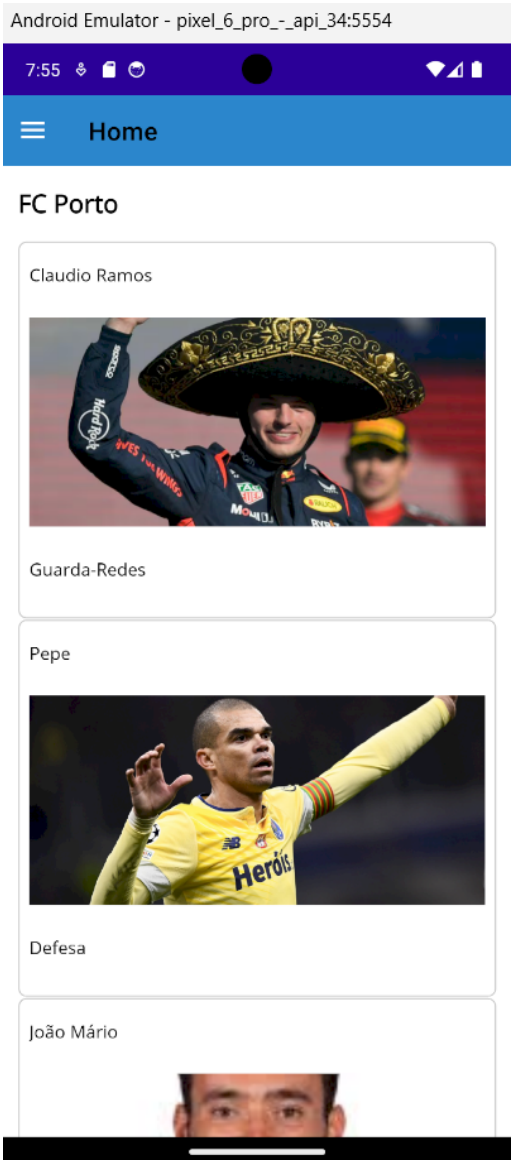

Figure 4.13: Main Page

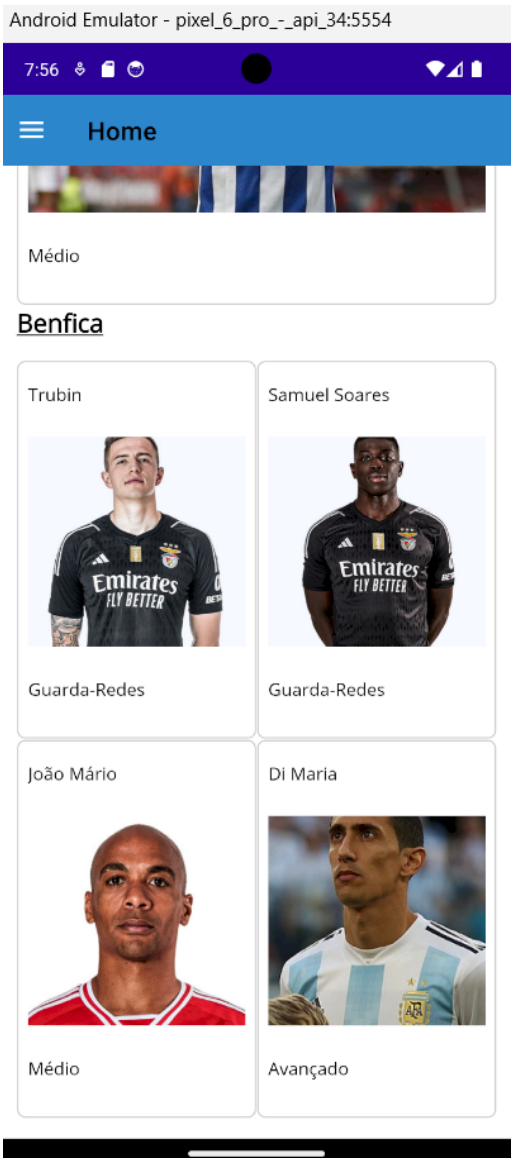

Figure 4.14: Main Page scroll down

Upon launching the application, the initial page displayed is Home, as illustrated in Figure 4.13 and Figure 4.14. This page contains the CardsList followed by the CardsGrid, as defined earlier.

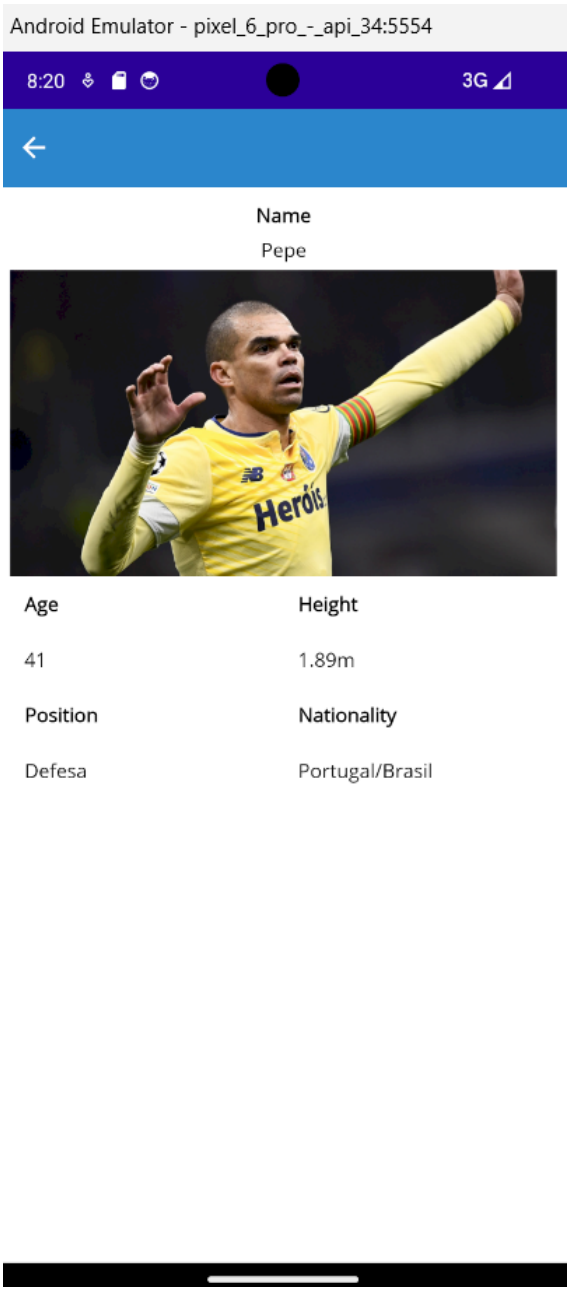

Figure 4.15: Details Page

By clicking in any element present on the list or the grid, we navigate to the details page of that element, as we can see in Figure 4.15. This page lists all the properties available for the selected document type.

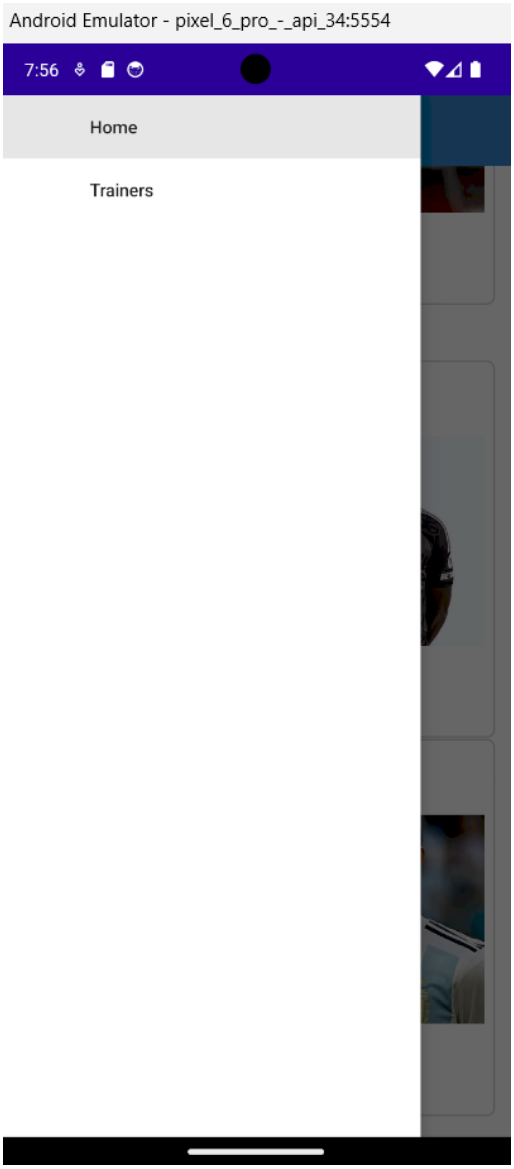

Figure 4.16: Menu

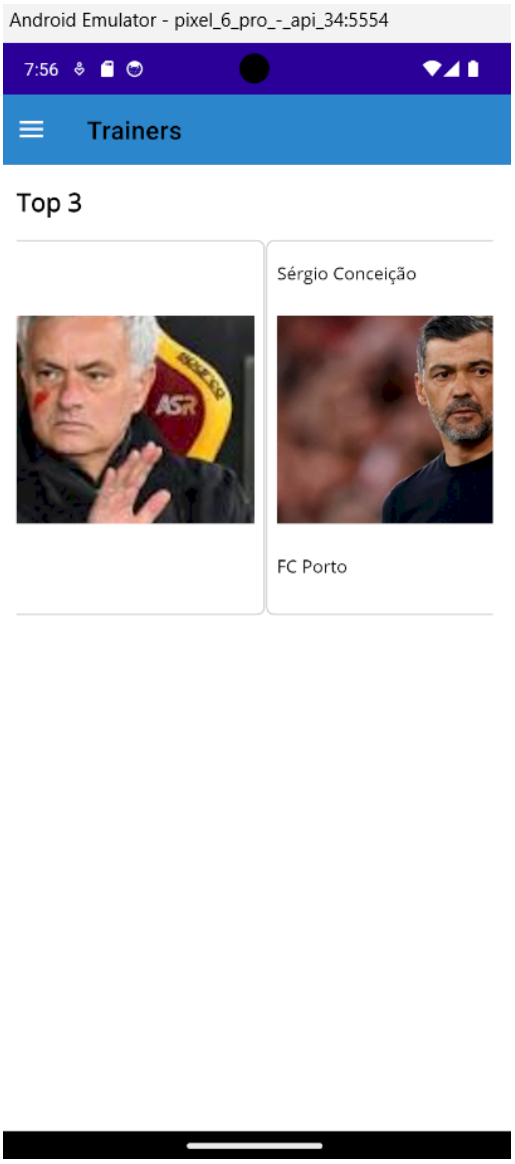

Figure 4.17: Trainers Page

Lastly, we can access other pages by clicking on the side menu, as shown in Figure 4.16. When we navigate to the Trainers page, a carousel showing the top 3 trainers' cards will be displayed, as depicted in Figure 4.17.

Following this demonstration, it can be concluded that the solution is successfully generating the mobile app according to the customizations made in the Umbraco backoffice. While the fundamental logic is functioning as intended, there are aspects that need improvement, such as the UI and the appearance of the components in the final mobile solution.

Chapter 5

Conclusion

This thesis emerged from the urgent need to generate mobile applications for Umbraco clients, allowing for customization and saving development time. As the research progressed, the complexity of the project became increasingly apparent. It was discovered that Umbraco packages were significantly more complex than initially expected, and the available documentation was often incomplete, especially in relation to code development.

As the development continued, it became evident that the best way to distribute this solution would be through NuGet packages. Additionally, it was found that Umbraco provides sufficient APIs to perform operations to extract information from both published content and content configured in the backoffice.

Consequently, a proof of concept was developed that enables the creation of all necessary configurations in the Umbraco back office and the customization of a mobile application. Although limited in this initial state, this proof of concept allows for the generation of applications using three external components: the Middleware, Azure Pipelines and a Roslyn Generator. This opens up a new world in the Umbraco ecosystem, transforming it from a standard CMS into a platform that also offers the capability to generate mobile applications automatically, significantly saving time and development costs.

Objective	**Status**
Development of an Umbraco Package	Completed
Automation in the extraction of Umbraco structure	Completed
Testing	Completed
Contribution to Umbraco's Community	Completed

Table 5.1: Objectives retrospective

By analysing Table 5.1, it is evident that every proposed objective was accomplished. The development of the proof of concept as an Umbraco package and the creation of an extraction algorithm capable of being used in every Umbraco environment using only the DeliveryAPI were successfully achieved. Additionally, all testing proved the efficiency and accuracy of this tool. The submission to the Umbraco marketplace was also approved, and more details about this can be found in section 5.2.

The project also faced time constraints, as the deadline for submitting the project to the Umbraco awards was very short. This limitation impacted the functionalities available in the proof of concept. Despite these constraints, the research and development conducted have

laid a solid foundation for future work and highlighted the potential for further advancements in the integration of mobile application generation with the Umbraco CMS.

5.1 Future Work

To build on this foundation, several improvements and extensions are proposed:

- Add more components to the Umbraco back office to support additional mobile components, such as tables, more complex forms, charts, etc.
- Implement two-factor authentication and Google authentication for increased security.

These enhancements will further extend the capabilities of the Umbraco ecosystem, enabling more sophisticated and customizable mobile applications and providing greater value to Umbraco clients.

5.2 Umbraco Awards

This project has also been submitted to NuGet.org, making it accessible for download by any member of the community as a NuGet package, compatible with Umbraco versions 12.0.0 to 13.3.2.

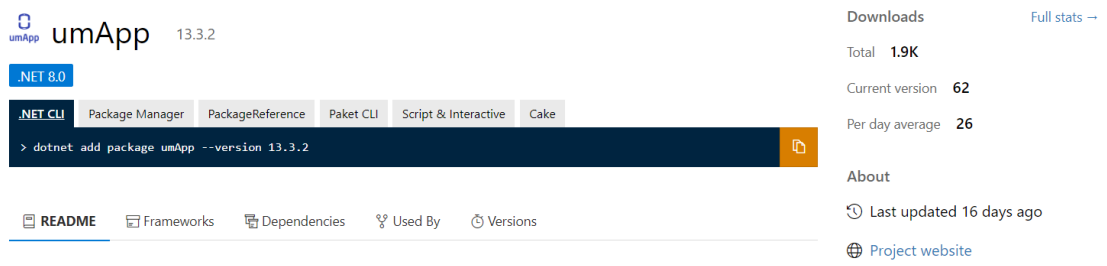

Figure 5.1: UmApp NuGet

As illustrated in Figure 5.1, the package is available on the official NuGet website and has already been downloaded nearly 2000 times. This distribution through NuGet ensures that the solution is readily accessible and can be easily integrated into any compatible Umbraco project, proving his impact and utility within the Umbraco community.

The package was also successfully added to the Umbraco Marketplace, as illustrated in Figure 5.2. The Umbraco Marketplace is a platform that showcases a variety of extensions and tools available for the Umbraco content management system.

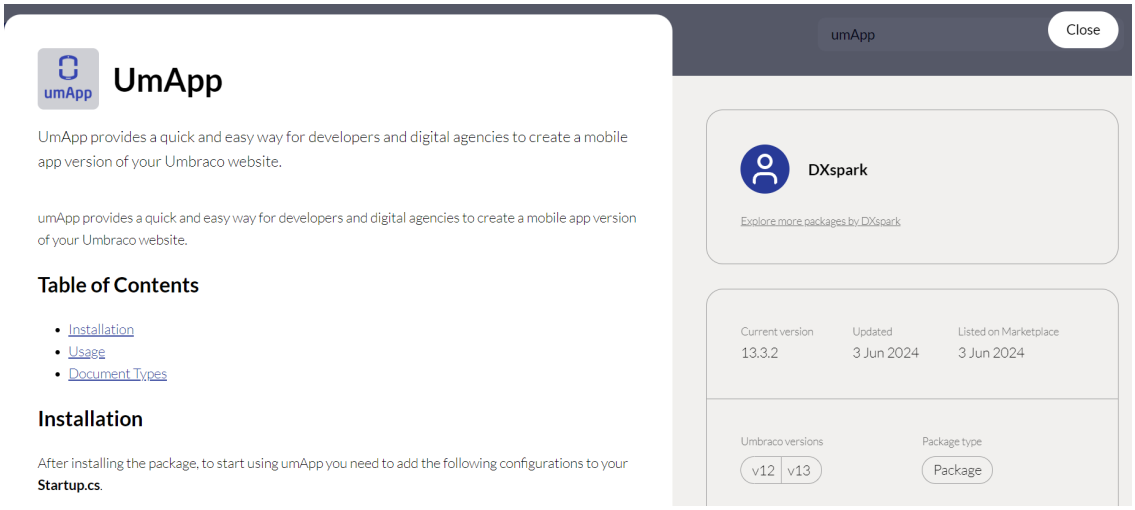

Figure 5.2: UmApp Umbraco Marketplace

Additionally, the package has been submitted for consideration in the Most Innovative Package category at the Umbraco Awards. The Umbraco Awards are an annual event held during the Umbraco Codegarden conference, which is the largest gathering of the Umbraco community. The awards aim to recognize and celebrate exceptional achievements within the Umbraco ecosystem across various categories. At the time of writing this thesis, the results are not yet announced.

Bibliography

[1] Igors Astapciks. *Council Post: Why Do Companies Need Digital Transformation?* Forbes. Section: Innovation. url: https://www.forbes.com/sites/forbestechcouncil/2023/03/20/why-do-companies-need-digital-transformation/ (visited on 01/05/2024).

[2] *Umbraco CMS Documentation*. url: https://docs.umbraco.com/umbraco-cms/ (visited on 12/30/2023).

[3] *The History of Umbraco from 1997 to now*. url: https://umbraco.com/about-us/the-history-of-umbraco/ (visited on 06/22/2024).

[4] Manoj Srivastav and Asoke Nath. "WEB CONTENT MANAGEMENT SYSTEM". In: *International Journal of Innovative Research in Advanced Engineering (IJIRAE)* 3 (Mar. 27, 2016), pp. 51–56. doi: 10.6084/M9.FIGSHARE.3504368.V1.

[5] Will M. *15 Best CMS Platforms to Start a Website in 2024*. Hostinger Tutorials. Aug. 12, 2018. url: https://www.hostinger.com/tutorials/best-cms (visited on 06/23/2024).

[6] *Usage Statistics and Market Share of WordPress, June 2024*. url: https://w3techs.com/technologies/details/cm-wordpress (visited on 06/23/2024).

[7] Savan K Patel, V.R. Rathod, and Satyen Parikh. "Joomla, Drupal and WordPress - a statistical comparison of open source CMS". In: *3rd International Conference on Trendz in Information Sciences & Computing (TISC2011)*. 3rd International Conference on Trendz in Information Sciences & Computing (TISC2011). ISSN: 2325-5927. Dec. 2011, pp. 182–187. doi: 10.1109/TISC.2011.6169111. url: https://ieeexplore.ieee.org/document/6169111?denied= (visited on 06/23/2024).

[8] *Joomla Content Management System (CMS) - try it! It's free!* Joomla! url: https://www.joomla.org/ (visited on 06/23/2024).

[9] *TIOBE Index*. TIOBE. url: https://www.tiobe.com/tiobe-index/ (visited on 06/23/2024).

[10] Will M. *Drupal vs Joomla: Ease of Use, Site Management, Customization, Performance and Security Compared*. Hostinger Tutorials. Oct. 5, 2018. url: https://www.hostinger.com/tutorials/drupal-vs-joomla (visited on 06/23/2024).

[11] Raheel Merchant. *How Do You Know if a WordPress Plugin is Secure? - HostPapa*. The HostPapa Blog. Feb. 8, 2022. url: https://www.hostpapa.com/blog/web-hosting/wordpress-plugin-vulnerabilities/ (visited on 06/23/2024).

[12] Maarten Balliauw and Xavier Decoster. "Creating Packages". In: *Pro NuGet*. Ed. by Maarten Balliauw and Xavier Decoster. Berkeley, CA: Apress, 2012, pp. 49–79. isbn: 978-1-4302-4192-8. doi: 10.1007/978-1-4302-4192-8_3. url: https://doi.org/10.1007/978-1-4302-4192-8_3 (visited on 06/22/2024).

[13] JonDouglas. *Create and publish a NuGet package with the dotnet CLI*. Aug. 21, 2023. url: https://learn.microsoft.com/en-us/nuget/quickstart/create-and-publish-a-package-using-the-dotnet-cli (visited on 06/29/2024).

[14] Jean Bézivin. "In Search of a Basic Principle for Model Driven Engineering". In: *Novatica/Upgrade* 5 (Jan. 1, 2004).

[15] Dragan Gasevic, Dragan Djuric, and Vladan Devedzic. *Model Driven Architecture and Ontology Development*. Journal Abbreviation: Model Driven Architecture and Ontology Development Publication Title: Model Driven Architecture and Ontology Development. Jan. 1, 2006. isbn: 978-3-540-32180-4. doi: 10.1007/3-540-32182-9.

[16] S. Sendall and W. Kozaczynski. "Model transformation: the heart and soul of model-driven software development". In: *IEEE Software* 20.5 (Sept. 2003). Conference Name: IEEE Software, pp. 42–45. issn: 1937-4194. doi: 10.1109/MS.2003.1231150. url: https://ieeexplore.ieee.org/document/1231150 (visited on 12/30/2023).

[17] Sami Beydeda, Matthias Book, and Volker Gruhn, eds. *Model-driven software development*. Berlin ; New York: Springer, 2005. 464 pp. isbn: 978-3-540-25613-7.

[18] Seiko Akayama et al. "Development of a modeling education program for novices using model-driven development". In: *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*. WESE '12. New York, NY, USA: Association for Computing Machinery, Oct. 12, 2012, pp. 1–8. isbn: 978-1-4503-1765-8. doi: 10.1145/2530544.2530548. url: https://doi.org/10.1145/2530544.2530548 (visited on 12/30/2023).

[19] Eugene Syriani, Lechanceux Luhunu, and Houari Sahraoui. "Systematic mapping study of template-based code generation". In: *Computer Languages, Systems & Structures* 52 (June 1, 2018), pp. 43–62. issn: 1477-8424. doi: 10.1016/j.cl.2017.11.003. url: https://www.sciencedirect.com/science/article/pii/S1477842417301239 (visited on 12/24/2023).

[20] Thomas Stahl and Markus Völter. *Model-driven software development: technology, engineering, management*. OCLC: ocm64453466. Chichester, England ; Hoboken, NJ: John Wiley, 2006. 428 pp. isbn: 978-0-470-02570-3.

[21] Brent Hailpern and P. Tarr. "Model-driven development: The good, the bad, and the ugly". In: *IBM Systems Journal* 45 (Feb. 1, 2006), pp. 451–461. doi: 10.1147/sj.453.0451.

[22] Arie Van Deursen, Paul Klint, and Joost Visser. "Domain-specific languages: an annotated bibliography". In: *ACM SIGPLAN Notices* 35.6 (June 2000), pp. 26–36. issn: 0362-1340, 1558-1160. doi: 10.1145/352029.352035. url: https://dl.acm.org/doi/10.1145/352029.352035 (visited on 12/30/2023).

[23] Martin Fowler. *DSL Guide*. martinfowler.com. Aug. 28, 2019. url: https://martinfowler.com/dsl.html (visited on 01/04/2024).

[24] B.J. Copeland. *Alan Turing's Electronic Brain: The Struggle to Build the ACE, the World's Fastest Computer*. OUP Oxford, 2012. isbn: 978-0-19-960915-4. url: https://books.google.pt/books?id=YhQZnczOS7kC.

[25] ENRIQUE DEHAERNE. *Code Generation Using Machine Learning: A Systematic Review | IEEE Journals & Magazine | IEEE Xplore*. url: https://ieeexplore.ieee.org/abstract/document/9849664 (visited on 01/05/2024).

[26] Adna Beganovic, Muna Abu Jaber, and Ali Abd Almisreb. "Methods and Applications of ChatGPT in Software Development: A Literature Review". In: *Southeast Europe Journal of Soft Computing* 12.1 (May 13, 2023). Number: 1, pp. 08–12. issn: 2233–1859. doi: 10.21533/scjournal.v12i1.251. url: http://scjournal.ius.edu.ba/index.php/scjournal/article/view/251 (visited on 01/05/2024).

[27] Man-Fai Wong et al. "Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review". In: *Entropy* 25.6 (June 2023). Number: 6 Publisher: Multidisciplinary Digital Publishing Institute, p. 888. issn: 1099-4300. doi:

10.3390/e25060888. url: https://www.mdpi.com/1099-4300/25/6/888 (visited on 01/05/2024).

[28] Lechanceux Luhunu and Eugene Syriani. "Comparison of the expressiveness and performance of template-based code generation tools". In: *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. SPLASH '17: Conference on Systems, Programming, Languages, and Applications: Software for Humanity. Vancouver BC Canada: ACM, Oct. 23, 2017, pp. 206–216. isbn: 978-1-4503-5525-4. doi: 10.1145/3136014.3136021. url: https://dl.acm.org/doi/10.1145/3136014.3136021 (visited on 12/31/2023).

[29] mgoertz-msft. *Code Generation and T4 Text Templates - Visual Studio (Windows)*. Mar. 10, 2023. url: https://learn.microsoft.com/en-us/visualstudio/modeling/code-generation-and-t4-text-templates?view=vs-2022 (visited on 01/01/2024).

[30] *Hello from OpenAPI Generator | OpenAPI Generator*. url: https://openapi-generator.tech/ (visited on 01/05/2024).

[31] *The .NET Compiler Platform*. original-date: 2015-01-11T02:39:03Z. Jan. 5, 2024. url: https://github.com/dotnet/roslyn (visited on 01/05/2024).

[32] Nick Harrison. *Code Generation with Roslyn*. Berkeley, CA: Apress, 2017. isbn: 978-1-4842-2210-2 978-1-4842-2211-9. doi: 10.1007/978-1-4842-2211-9. url: http://link.springer.com/10.1007/978-1-4842-2211-9 (visited on 01/05/2024).