**Czech University of Life Sciences Prague**

**Faculty of Economics and Management**

**Department of Information Engineering**



# Bachelor Thesis

## Low Code Platforms

**Ahmad Akel**

# CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

# BACHELOR THESIS ASSIGNMENT

Ahmad Akel

Informatics

Thesis title

**Low Code Platforms**

---

**Objectives of thesis**

Main objective:
The main objective of this thesis is to evaluate the effectiveness of low-code platforms to accelerate application development and reduce reliance on traditional coding practices. It will analyze the advantages and disadvantages of adopting low-code platforms and assess their suitability for different types of applications.

Partial Objectives:

- Investigate the principles and concepts of low-code platforms and their relevance in the software development landscape.
- Examine case studies and real-world examples of organizations that have utilized low-code platforms for application development.
- Analyze the benefits and limitations of low-code platforms regarding development speed, scalability, maintainability, and customization.
- Assess the impact of low-code platforms on developer productivity, skill requirements, and collaboration between developers and business stakeholders.

**Methodology**

The theoretical part of the work is based on the study and analysis of professional and scientific information sources. The thesis addresses the topic of low-code platforms with a specific focus on their application in software development.

The methodology will involve the following steps:

- Conduct a comprehensive review of relevant literature, including academic papers, books, and industry reports, to establish a theoretical foundation.

---

- Analyze existing low-code platforms and their features through documentation, online resources, and vendor-provided information.

- Explore case studies and success stories of organizations that have adopted low-code platforms for application development.

- Gather primary data through interviews, surveys, or experiments to gather insights from developers, organizations, and end-users.

- To draw meaningful conclusions, analyze the collected data using qualitative and quantitative research methods.

- Synthesize the findings from the theoretical and practical parts to formulate broad conclusions.
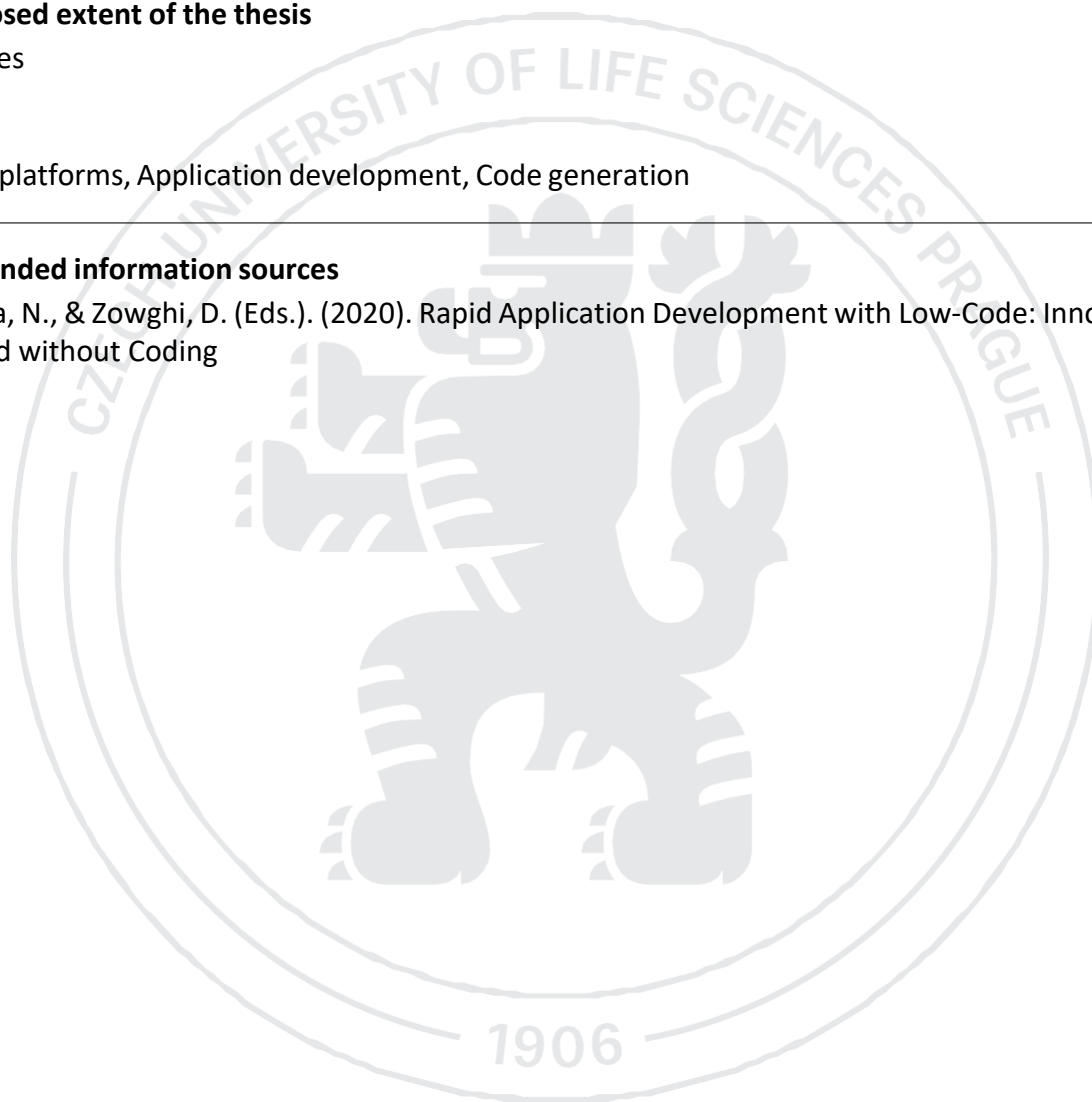
**The proposed extent of the thesis**

40-50 pages

**Keywords**

Low-code platforms, Application development, Code generation

---

**Recommended information sources**

Mendonça, N., & Zowghi, D. (Eds.). (2020). Rapid Application Development with Low-Code: Innovate at
   Speed without Coding

---

**Expected date of thesis defence**

2023/24 SS – PEF


**The Bachelor Thesis Supervisor**

Ing. Tomáš Vokoun


**Supervising department**

Department of Information Technologies


Electronic approval: 29. 6. 2023

**doc. Ing. Jiří Vaněk, Ph.D.**

Head of department

Electronic approval: 3. 11. 2023

**doc. Ing. Tomáš Šubrt, Ph.D.**

Dean


Prague on 05. 03. 2024

**Declaration**

I declare that I have worked on my bachelor thesis titled "Low Code Platforms" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break any copyrights.

In Prague on 15/03/2024

**Ahmad Akel**

**Acknowledgment**

I would like to thank Tomáš Vokoun and all other persons, for their advice and support during my work on this thesis.

# Low Code Platforms

**Abstract**

This thesis closely examines how Low-Code Development Platforms (LCDPs) are changing how software is created. Focusing on A12, a popular low-code tool, the author explores how These platforms can make building apps faster and easier than traditional coding.

The author used A12 to build an app and surveyed developers and project managers. This mix of hands-on workand feedback helped us see the big picture of using LCDPs.

The main discovery is that LCDPs speed up app development and make the process smoother. But, they're not perfect. The author suggests some ideas on how to make LCDPs even better.

In short, this thesis shows that LCDPs are a big help in software development but are still growing. The main work adds to what's known about LCDPs and suggests directions for future improvements.

**Keywords:** Low-code platforms, A12, Rapid Application Development, Code generation, Scalability, Maintainability, Development Speed, Model-driven Development, Acceleration, Code Customization, Code Abstraction.

# Title of Bachelor Thesis in Czech

**Abstrakt**

Tato práce podrobně zkoumá, jak Low-Code Development Platforms (LCDP) mění způsob vytváření softwaru. Autor se zaměřuje na A12, oblíbený nástroj s low code, a zkoumá jak
Tyto platformy mohou vytvářet aplikace rychlejší a jednodušší než tradiční kódování.

Autor použil A12 k vytvoření aplikace a dotazoval se vývojářů a projektových manažerů. Tato kombinace praktické práce a zpětné vazby nám pomohla vidět celkový obraz používání LCDP.

Hlavním objevem je, že LCDP urychlují vývoj aplikací a činí proces plynulejším. Ale nejsou dokonalé. Autor navrhuje několik nápadů, jak LCDP ještě vylepšit.

Stručně řečeno, tato práce ukazuje, že LCDP jsou velkým pomocníkem při vývoji softwaru, ale stále rostou. Hlavní práce doplňuje to, co je o LCDP známo, a navrhuje směry pro budoucí zlepšení.

# Table of Contents

# 1. Introduction

In today's fast-moving tech world, making software quickly and easily is more important than ever. Traditional ways of coding can take a lot of time and need a lot of skill, which can slow things down. This is where Low-Code Development Platforms (LCDPs) come in as a game-changer. These platforms help people make apps faster by cutting down on the need to write code from scratch.

I've seen how slow traditional coding can be and how it struggles to keep up with the need for quick and easy app development. LCDPs offer a solution by letting people build apps using simple tools like drag-and-drop, without needing to be coding experts.

This thesis looks into how useful LCDPs can be, using A12 as an example. By building an app, asking developers and project managers what they think, and looking closely at how LCDPs work, this study aims to understand how these platforms can make app development quicker and easier.

The main goal is to see if LCDPs can speed up making software and make it simpler for everyone involved. This research will dig into the good and the bad of LCDPs, offering a clear view of what they can do and where they might need some work.

# 2. Objectives and Methodology

## 2.1. Objectives

**Main objective:**
The main objective of this thesis is to evaluate the effectiveness of low-code platforms to accelerate application development and reduce reliance on traditional coding practices. It will analyze the advantages and disadvantages of adopting low-code platforms and assess their suitability for different types of applications.

**Partial Objectives:**
- Investigate the principles and concepts of low-code platforms and their relevance in the software development landscape.
- Examine case studies and real-world examples of organizations that have utilized low-code platforms for application development.
- Analyze the benefits and limitations of low-code platforms regarding development speed, scalability, maintainability, and customization.
- Assess the impact of low-code platforms on developer productivity, skill requirements, and collaboration between developers and business stakeholders.

## 2.2. Methodology

- Conduct a comprehensive review of relevant literature, including academic papers, books, and industry reports, to establish a theoretical foundation.
- Analyze existing low-code platforms and their features through documentation, online resources, and vendor-provided information.
- Explore case studies and success stories of organizations that have adopted low-code platforms for application development.
- Gather primary data through interviews, surveys, or experiments to gather insights from developers, organizations, and end-users.
- To draw meaningful conclusions, analyze the collected data using qualitative and quantitative research methods.
- Synthesize the findings from the theoretical and practical parts to formulate broad conclusions.

# 3. Literature Review

## 3.1. Introduction Low-Code Platforms

### 3.1.1. Definition and characteristics of low-code platforms

The low-code platform encompasses a suite of tools designed to cater to both programmers and non-programmers alike. Its primary objective is to facilitate the rapid creation and deployment of business applications, significantly reducing the necessity for extensive coding efforts. Additionally, it streamlines the process of installing and configuring environments and the training and implementation phases (1).

### 3.1.2. Historical development and evolution of low-code platforms

The term "low-code" was formally introduced in 2016, and the foundations of this technology were laid over several decades prior. In fact, the progenitors of today's low-code technology emerged during the 1970s to 1990s as a part of the 4th Generation Programming Language and Rapid Application Development paradigm. Surprising as it may be to many, the inaugural no-code platform was none other than Microsoft Excel, launched in 1985 (2).

The bedrock of today's world-altering technological progress was laid by the ingenious use of spreadsheets within Excel. This pivotal advancement served as the cornerstone that propelled technology's evolution.

The advent of cloud computing in 1997 and the subsequent emergence of Salesforce in 2000 played pivotal roles in further accelerating this technological journey. Salesforce's arrival marked a watershed moment, championing the ascent of Software as a Service (SaaS) and boldly proclaiming the obsolescence of the traditional software development era. The saga of innovation continued with a series of notable milestones:

In 2006, Formstack took a momentous leap by introducing the pioneering concept of a no-code form builder, led by the visionary Ade Olonoh.

That very year, 2006, also witnessed the grand unveiling of Shopify's groundbreaking no-code eCommerce platform, now commanding a staggering valuation of approximately $37 billion.

The year 2007 saw the dawning of the iPhone, triggering an avalanche of mobile app builders and their proliferation.

2012 marked the debut of Bubble, heralding a significant advancement in the landscape of no-code web development platforms.

The year 2013 ushered in the rise of Webflow as the foremost platform that allowed the complete no-code creation of websites and applications.

Finally, 2018 witnessed the fusion of PowerBI, Flow, and PowerApps into the Microsoft Power Platform, facilitating heightened connectivity with minimal prerequisites for coding expertise.

## 3.2. Principles and Concepts of Low-Code Platforms

### 3.2.1. Key principles that underpin low-code platforms

Central to the ethos of Low-Code Development Platforms (LCDPs) is the principle of "reduction." LCDPs redefine conventional software development by promoting reductionist concepts, leading to notable advancements across various aspects. This section elucidates the core principles bolstering low-code platforms, encapsulated within the reductionist framework. LCDPs redefine software creation by reducing the effort required to build intricate systems from scratch. These platforms streamline development workflows, enhancing productivity and allowing developers to focus on higher-level functionalities. Temporal efficiency is another facet of reduction in LCDPs. These platforms expedite product delivery by compressing traditionally elongated development timelines, crucial for remaining competitive in agile markets. Financially, LCDPs reduce resource-intensive endeavors needed for comprehensive software systems. This leads to diminished resource costs, spanning personnel, infrastructure, and operational expenses. Additionally, LCDPs simplify project management complexities inherent in software creation. The modular nature of low-code platforms minimizes coordination challenges, promoting a cohesive development environment. The reduction principle extends to human resources, reducing the need for additional personnel. LCDPs empower existing teams, optimizing resource allocation and scalability in line with project needs. In summation, the foundational principles of low-code platforms center on reduction. This concept permeates developer efforts, time dynamics, financial considerations, project management, and resource utilization, reshaping modern software development practices.
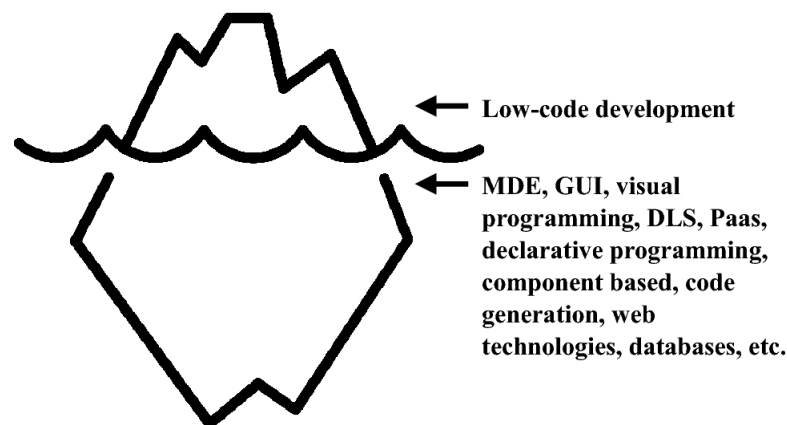


*Figure 1. Abstraction in Low-Code Platforms (11)*

### 3.2.2. Core concepts and abstractions used in low-code development

In LCDP, there are key ideas and foundational elements that shape its progress. This section delves into these important concepts that define low-code development, focusing on simplicity and user-friendliness.

**Simplified Building Blocks:** Low-code development relies on smart simplifications that capture complex actions and system behavior. These building blocks are the basic pieces we use to create software. They help people, regardless of coding skills, design advanced programs with ease. This type of simplification makes it harder for automated tools to spot any unusual patterns.

**Easy Visual Designs:** Low-code development uses visuals to help build software. This means you can move around different elements like pieces in a game. Visual designs make it easier to understand how things fit together. These designs are like puzzles for creating software, and they're not as easy for automated tools to recognize.

**Reusable Parts:** Another important part of low-code development is using pieces of code again and again. It's like having a set of building blocks that you can use in different projects. This approach helps save time and make everything work together smoothly. Because things are broken into small pieces, it's trickier for automated tools to figure out what's happening.

**Automated Starter Code:** Low-code development often comes with bits of ready-to-use code that kick-start your project. This means you don't have to write everything from scratch. These helpful bits are like having a head start, and they make it less likely for automated tools to see a clear pattern.

### 3.2.3. Comparison between low-code and traditional coding practices

Low-code development diverges from traditional application development in that it broadens the capacity to fabricate applications or websites for a more extensive spectrum of individuals. Whereas customary coding necessitates adept developers possessing manual coding proficiency, low-code development necessitates foundational knowledge and provides uncomplicated, user-friendly tools, templates, and interfaces (3).

| Low-Code Platforms | Traditional Development |
|---|---|
| Requires minimal coding knowledge. | Requires skilled IT professionals. |
| Reduces development costs. | Requires more time, energy, money, and resources. |
| Uses templates and drag-and-drop tools. | Allows for more agile customization. |
| Provides automated updates and routine maintenance. | Provides the ability to integrate with a broader range of other systems and apps. |
| Offers built-in visuals, reports, and analytics. | Scales and adapts more readily. |

*Table 1. Comparison between Low-Code & Traditional Development*

## 3.3. Benefits and Limitations of Low-Code Platforms

### 3.3.1. Advantages of using low-code platforms for application development

Low-code development has emerged as an increasingly indispensable facet, offering solutions that alleviate workloads and provide organizations with the ability to rapidly conceptualize, deploy, and refresh applications.
This expeditious deployment and prompt update cycle empower businesses to craft enhanced customer experiences, as they can adroitly respond to evolving customer feedback and behaviors.
Leveraging platforms for constructing low-code applications enables enterprises to create professional-grade apps replete with sophisticated functionalities (3).
Low-code development platforms encompass a spectrum of business requirements, serving as a remedy for IT burdens.
Enabling non-IT personnel to partake in app creation, low-code development liberates professional developers to dedicate their expertise to more specialized coding, and crafting bespoke components and applications.
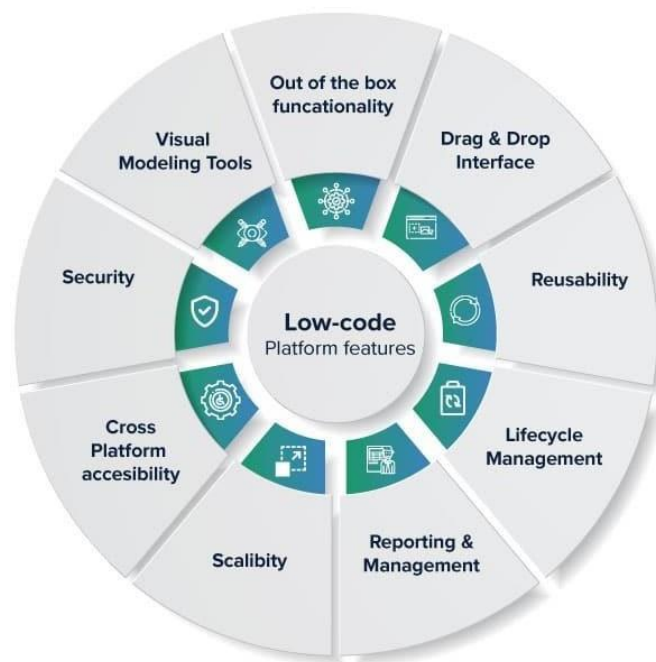


*Figure 2. Features of Low-Code Platforms (16)*

### 3.3.2. Limitations and challenges associated with low-code platforms

Among the prominent constraints associated with low-code platforms lies the limited scope for customization. This constraint arises from the inherent lower potency of low-code development in comparison to traditional programming. Users find themselves confined to the choices presented within the low-code platform itself. In more intricate scenarios, a Citizen Developer might encounter challenges in meeting rigorous demands (3) (4), necessitating eventual assistance from professional developers.

Furthermore, scalability and fragmentation emerge as noteworthy drawbacks within the realm of low code. Scalability entails the capability to expand the application's utilization to encompass larger systems, projects, or organizations. Concurrently, the issue of fragmentation surfaces, stemming from the restricted interoperability between various low-code platforms and their supporting databases.

Certain organizations exhibit reluctance in fully embracing low-code due to the perceived elevated per-user expenditure and apprehensions about potential vendor lock-in (4). These reservations often emanate from the following limitations:

**Lack of Customization**: Low-code inherently offers less prowess compared to traditional programming, constraining users to the platform's provided options.

**Scalability Limitations:** While low code is suitable for constructing smaller applications, realizing solutions at an enterprise level can prove challenging.

**Fragmentation Issues:** The divergent nature of low-code systems among different vendors hampers data integration when employing diverse low-code platforms.

**Vendor Lock-In Concerns:** Organizations adopting specific vendors for their app development might inadvertently find themselves locked into that environment, limiting flexibility.

**High Licensing Costs:** While initial app development costs might not be substantial, expenses surge when deploying apps to a larger user base.

**Security Considerations:** The potential development and release of apps by non-specialists without adequate security consideration can pose threats to overall organizational IT security.

## 3.4. Types of Low Code Platforms

### Categorization of Low-Code Platforms by Use Case

Low-code platforms encompass a range of applications catering to distinct use cases. The diversity in intended usage or purpose leads to the existence of various types of low-code platforms. Here, are the primary classifications of low-code platforms (18):

**General Purpose Platforms:**
These platforms possess the capacity to construct virtually any form of application. Within the realm of general-purpose platforms, one can develop applications that serve multifarious needs and are deployable across diverse environments. This category encompasses both front-end and back-end application development based on specific requisites (17).

**Process-Based Platforms:**
Process-oriented low-code platforms concentrate on applications that facilitate business processes, including forms, workflows, and integrations with other systems. Such platforms, known as low-code Business Process Management (BPM) tools, excel in optimizing approval workflows and digitizing paper-based processes (17).

**Database-Focused Platforms:**
Database-oriented platforms further narrow their scope, permitting access solely to retrieve data from databases programmed within the system's architecture. These platforms prove advantageous when there's a need to efficiently ingest large volumes of data into a system, with time constraints in mind (17).

**Mobile Application Development Platforms (MADPs):**
MADPs empower developers to code, test, and launch mobile applications for diverse devices such as smartphones and tablets. By enabling the creation of code for one platform and facilitating the porting of applications across multiple mobile platforms, MADPs enhance efficiency in mobile app development (17).

### Low-Code Platforms Differentiated by User Profiles

Low-code platforms offer distinct advantages to two primary user categories: developers and business users. However, achieving equilibrium in serving both user types is a rarity. Platforms skewed towards enhancing developer speed often become overly intricate for lay users, whereas those designed for universal ease of coding might disappoint developers seeking customization (17).

**Low-Code Platforms for Business Users:**
Tailored for business users, these platforms minimize the necessity for extensive coding, furnishing an environment where non-technical users can create applications. By empowering business users as citizen developers, platforms of this kind facilitate application creation despite the absence of programming proficiency. Nevertheless, in certain instances, citizen developers may require developer assistance for deploying specific functionalities (17), examples of these types of low-code development platforms include Mendix, OutSystems, Salesforce, and SAP, which have been widely adopted across various industries for their ability to streamline the application development process (18).

**Low-Code Platforms for Developers:**
Positioned toward developers, these low-code platforms aim to reduce application development time compared to traditional manual coding. Employing visual interfaces, drag-and-drop modules, and similar aids, these platforms significantly alleviate manual coding efforts. However, some degree of coding remains indispensable to fully realize application development. These platforms are well-suited for constructing applications of varying complexities and are not generally confined to any single platform unless vendor-imposed restrictions apply (17).

## 3.5. A12 Low Code Platform

Wrapping up the previous sections on low-code platforms, we now shift our focus to a specific case study within this domain: the Low-Code Development Platform (LCDP) called "A12." In the upcoming sections, we will extensively dissect A12, exploring its characteristics and capabilities. Positioned at the crossroads of General Purpose and Process-Based utility in use cases, and catering to both Business Users and Developers in terms of user profiles, A12 offers a multifaceted perspective for investigation. We aim to dive deep into its potential, limitations, structure, and how it serves the needs of diverse users with varying expertise.

### 3.5.1. A12 Overview

A12 is an extensible development platform for web-based business applications. It provides developers with a set of robust, secure, and scalable components as well as a client/server application infrastructure (12). A12 enables business analysts to define large parts of the application independently and conveniently using models with the help of special editors.

A12 accelerates the development of web applications and reduces individual development efforts.

### 3.5.2. A12 Structure

The A12 Business Application Platform encompasses a diverse array of interrelated components, each contributing to a cohesive ecosystem:
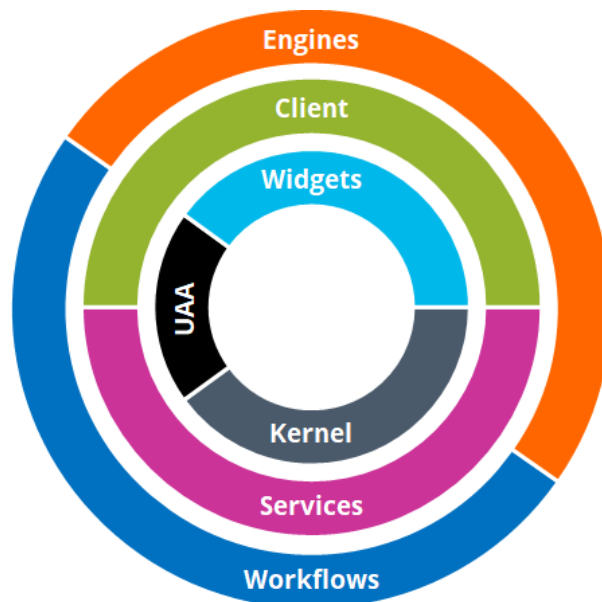


*Figure 3. Products that make up A12 (15)*

**BAP Client:** The Business Application Platform (BAP) Client library embodies a model-driven, extensible client runtime. Its primary function revolves around the orchestrated deployment of A12 Engines by a model-driven methodology (15).

**Engines:** The Engines within the A12 framework are model-driven UI components that adhere to the Plasma UI/UX framework, leveraging the TypeScript and React programming paradigms. This component suite presently encompasses form elements, overviews, and tree structures. Engines necessitate configuration through models while concurrently offering a TypeScript-based programming API (15).

**Kernel:** At the core of the A12 architecture lies the Kernel, encompassing the specification of documents and document models alongside a suite of modeling tools. This encompasses a domain-specific language designed for model-based computations and validations, complete with a parser, runtime components, and programming interfaces in diverse languages (15).

**Data Services:** The Data Services facet of A12 encapsulates both service APIs and corresponding implementations, facilitating the manipulation of models and documents. These services encompass functions of creation, retrieval, update, deletion, and querying. Such services are available in TypeScript for client-side applications and in Java for both client-side and server-side contexts. The provision of client/server communication, authentication, authorization, computation, validation, persistence, indexing, and, in the future, versioning and locking functionalities are integral components of Data Services (15).

**Widgets:** A12's Widgets library is an extensive collection of web components, aligned with the Plasma design philosophy, catering to consistent and aesthetically pleasing design and interaction principles. These components are tailored to support business applications spanning desktop, tablet, and smartphone interfaces, accommodating keyboard, mouse, and touch input methods. Characterized by a user-friendly, comprehensively documented, strongly typed API, these components are amenable to extension and customization (15).

**Workflows:** A12 Workflows introduces a lightweight service that seamlessly integrates Business Process Model and Notation (BPMN) capabilities into the A12 ecosystem. This extension facilitates the graphical modeling of server-side workflows and their subsequent execution (15).

**Base:** Comprising an assortment of libraries, the Base framework defines fundamental model concepts and offers corresponding default implementations. This foundational layer finds utility in higher-tier A12 products concerned with model-related tasks (15).

**Core:** The "Core" component represents the historical precursor to the current A12 product structure. In prior release iterations, it encompassed elements that have since been repositioned within the Kernel, Engines, or Services modules. Presently relegated to a maintenance phase, the Core component will persist until the migration of customer projects to the release line 2018.05 or later is complete (15).

**Simple Model Editor (SME):** The Simple Model Editor is a web-based modeling tool tailored to A12 models constructed using the A12 platform itself. This tool empowers business analysts to generate, edit, and deploy A12 models, currently limited to JSON-based formats. The supported model types include documents, overviews, trees, applications, relationships, and forms with their corresponding bindings (15).

**UAA:** The User Management, Authentication, and Authorization (UAA) module stands as A12's comprehensive security solution. UAA incorporates a unified authentication library

compliant with contemporary standard protocols. This, in conjunction with a rule-based authorization library, safeguards access to the A12 platform and associated services. UAA additionally extends extensible user management functionalities, adaptable to integration with various Identity Provider (IDP) systems (15).

**Plasma:** Plasma emerges as a design framework catering to enterprise-level business applications, abstracting intricate real-world scenarios into generalized design and interaction principles. Its principal emphasis centers on user-centric design, striving to attain optimal user efficiency (15).

**Utils:** The Utils component constitutes an ensemble of technical utility libraries unrelated to model considerations. Among the included utilities are TypeScript Logging and TypeScript Collections, along with localization capabilities (15).

## 3.6. A12's Model-Driven Approach

In the realm of software development, the concept of model-driven development has gained prominence as a powerful methodology that promotes efficiency and adaptability (13). At its core, model-driven development encapsulates domain-specific knowledge within models, allowing for the creation and modification of applications without direct involvement in the underlying codebase (14). This section delves into the model-driven development philosophy, examining how it is applied within the context of the mgm A12 Business Application Platform.



*Figure 4. The Process of Model-Driven Development (14)*

### 3.6.1. Understanding the concept of model-driven development in A12

Central to mgm A12 is the principle of encapsulating domain-specific knowledge within models. Through the utilization of robust tools, domain experts and business analysts are empowered to design and modify these models, all without necessitating direct code manipulation. This pivotal concept substantially reduces the need for custom development efforts while providing a mechanism for rapid adaptation to evolving business requirements and that happens by implementing two concepts:

1. **Focusing on Documents and Forms**
   In the digital world, many business activities involve using documents and forms. These could be things like contracts, purchase orders, or requests. They're really important for different industries. To make these documents digital, online forms are used. These forms help define how the documents should look and what information is needed. In mgm A12, we pay special attention to creating and managing these forms (15). This way, even complex forms with lots of information and special rules can be made, and they become part of websites and applications.

2. **Working Everywhere: Across Devices and Platforms**
   As business applications grow, they need to work well on many different devices, like phones and tablets. They also need to work smoothly on different computer systems and platforms. This can be quite tricky. But A12 made things easier. There are built lightweight tools (like engines) that handle how things look and work on different devices. These tools are made using JavaScript, a type of computer language (15). There are also thought a lot about how to make things look good on any device, so everything looks modern and consistent. This way, businesses can create and change their applications quickly to match how technology is always changing.
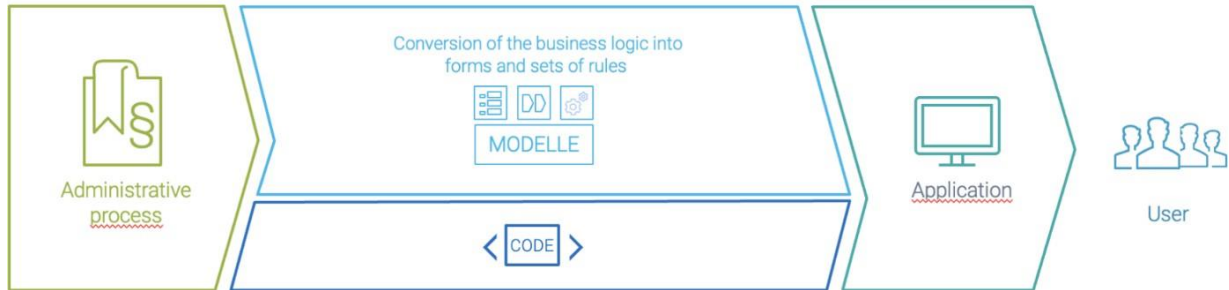
22

*Figure 5. Model-based software development (8)*

### 3.6.2. Advantages of using models as a basis for application development in A12

The strategic utilization of models as the cornerstone for application development has garnered noteworthy attention due to its multifaceted benefits (14). This section expounds uponthe advantages intrinsic to employing models as a pivotal framework for application development within the realm of the A12.

- **Leveraging the Model-Centric Paradigm**
  One of the paramount advantages arises from the inherent model-centric paradigm that underpins the A12 development approach. This paradigm accentuates the separation of domain-specific intricacies from the underlying implementation, facilitating an environment where intricate business logic can be encapsulated within models. As a consequence, software developers and domain experts engage in a symbiotic collaboration, augmenting productivity by effectively decoupling intricate business processes from coding intricacies (15).

- **Swift Adaptation to Evolving Business Landscape**
  The adoption of models as foundational constructs fosters an agile response to the ever-changing demands of the business landscape. The abstract nature of models enables rapid modifications to be implemented with minimal code-level interventions. This agility is particularly relevant in a digital milieu characterized by dynamic market conditions, where the capacity to swiftly incorporate alterations ensures that business applications remain aligned with evolving requirements (14).

- **Facilitating Collaborative Development**
  A notable merit of employing models resides in its propensity to foster collaborative development. The use of models serves as a lingua franca between various stakeholders, including business analysts and developers (14). This common language expedites comprehension, enabling cross-functional teams to communicate effectively and synergistically and contribute to the development process. Consequently, the interdisciplinary barrier is mitigated, leading to more cohesive and successful project outcomes.

23

- **Enhancing Quality and Consistency**
  Models engender a structured environment conducive to quality assurance and consistency. By embodying business rules and constraints, models inherently serve as prescriptive documentation, leaving little room for ambiguity. This explicit representation, in turn, aids in eliminating discrepancies between different phases of development and throughout the software lifecycle (14). The result is an elevated level of software reliability and a reduction in the occurrence of defects.

In summation, the practice of employing models as a foundation for application development in the A12 context yields multifarious advantages, encompassing efficient development paradigms, responsiveness to change, collaborative synergies, and elevated software quality. This paradigm underscores the potency of model-driven approaches in contemporary software engineering endeavors.

## 3.7. Models Types in A12

There are several different types of models available in A12:

- Document model

- Form model

- Overview model

- Tree model

- Application model

- Relationship model

- Relationship binding
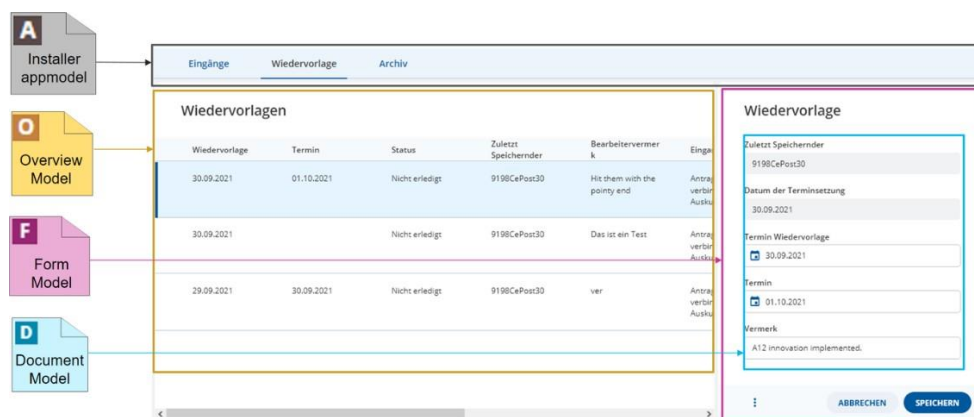


*Figure 6. A12 application models diagram (10)*

## A. Document model

The core of the application rests in the document model—a holistic representation of the business domain. Guided by domain experts, this model orchestrates entities using fields and groups. For instance, entities like companies or employees find their place in dedicated models with associated fields. These fields can be structured hierarchically within groups.

From a technical perspective, document models define fields, data types, type descriptions, and validation protocols. The A12 Validation language is instrumental in enforcing regulations, computations, and expressions on fields. These validations range from basic requirements to intricate conditions spanning multiple fields.
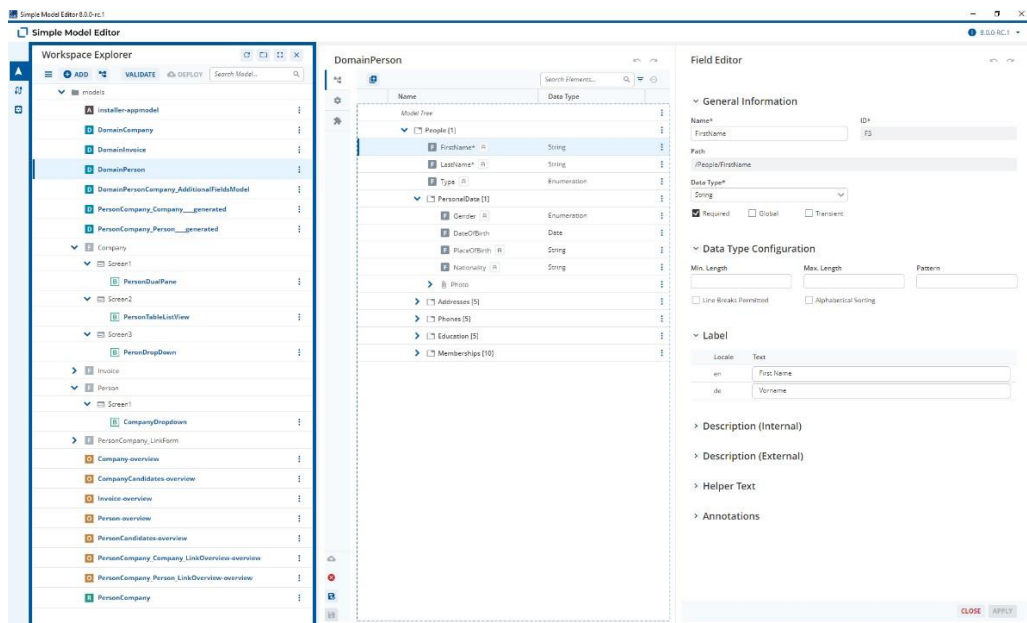


*Figure 7. Simple Model Editor (SME) (15)*

The crafting of document models occurs within the Document Model Editor, nestled within the Simple Model Editor (SME), and they are archived as JSON files. This platform streamlines data structure and validation rule articulation, equipped with "autocomplete" and "syntax highlighting" for the A12 Validation language, facilitating the management of validation criteria with ease (15).

25

## B. Form model

The form model is one of the UI models available in A12 and it can be used to display details or fields from one instance of an entity. The Form Modeling Module (FMM) is used to create form models and they are stored as JSON files (15).
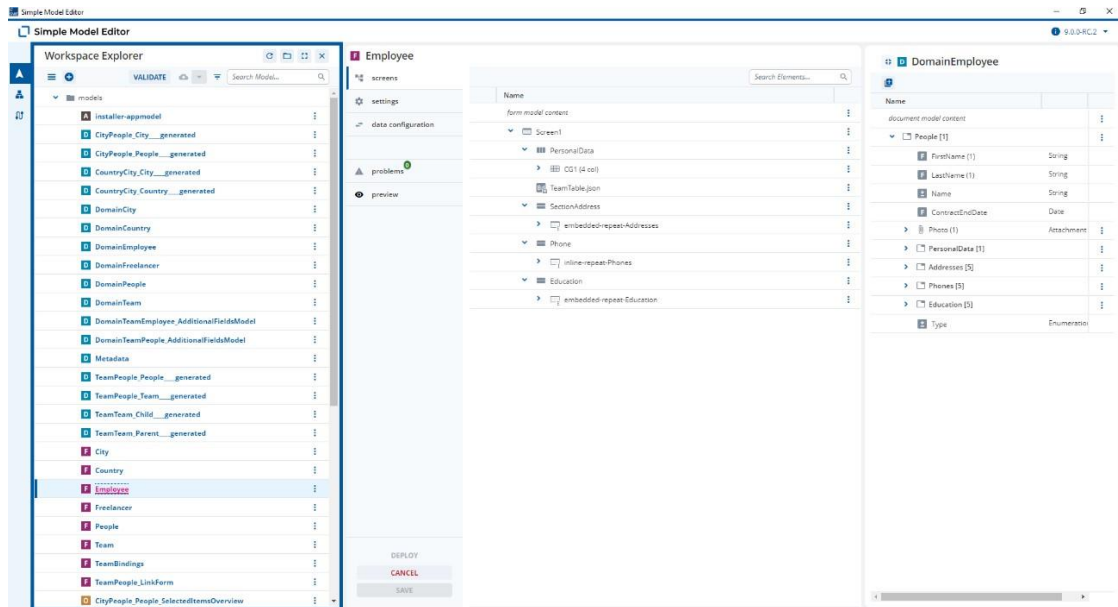


*Figure 8. Form Model in the Form Modeling Module (15)*

The FMM adopts an abstract approach to UI design. Instead of specifics like colors, it uses models to organize UI components. It doesn't directly place input fields or buttons on the screen like WYSIWYG (What You See Is What You Get) methods. Instead, it arranges them hierarchically using model elements. This empowers domain experts to craft user interfaces tailored to their expertise. A12 employs Plasma Design by mgm to ensure an appealing default look, separate from the FMM.

Each A12 UI model is tied to at least one A12 document model. This underlines the FMM's philosophy, distinguishing it from traditional GUI builders. UI models are like wrappers for specific parts of document models. They structure user interaction based on chosen data fields. This link means the document model and validation rules come first. Then, new UI models referencing the document model can be created. In A12, data and validation definitions are pivotal (15).

## C. Overview model

The overview model, present within the A12, serves as an additional exemplar of a UI model. Similar to the form model, it rests upon a related document model and is shaped through the Simple Model Editor, eventually residing as a JSON file (15).
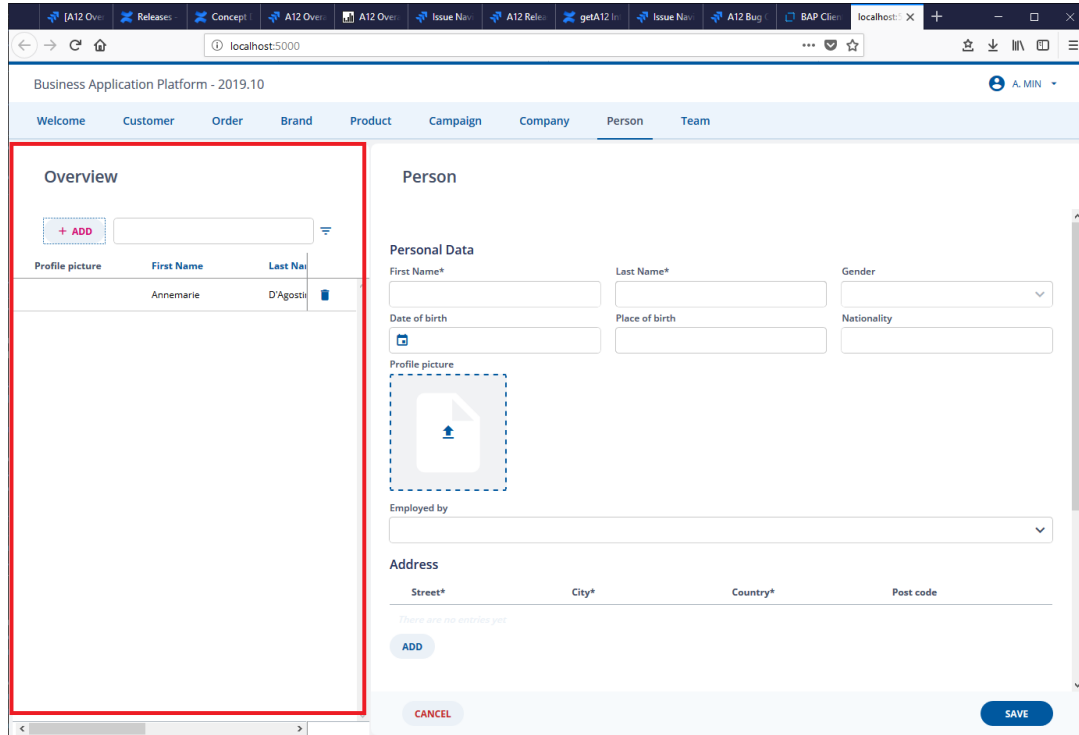


*Figure 9. Overview Model in an application (15)*

## D. Tree model

The tree model, an integral constituent of the A12 framework, operates as a specialized UI model geared towards presenting data in a hierarchical arrangement, driven by the relationships existing between entities. In contrast to forms that primarily facilitate user data input, business applications commonly integrate hierarchical tables to systematically exhibit selected facets of datasets in an organized manner, as illustrated in the accompanying visual reference. The tree model is thoughtfully devised to cater precisely to this requirement, enabling the structured

27

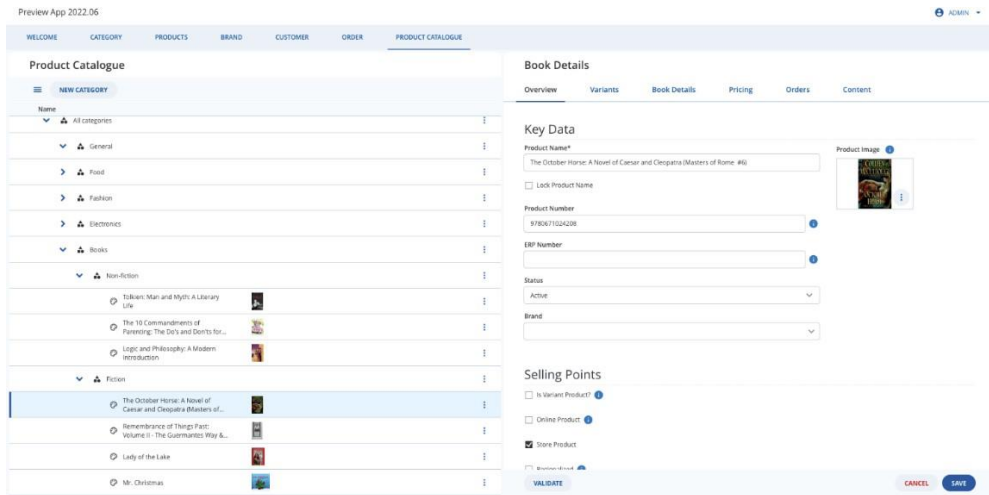portrayal of interconnected datasets within the context of business applications (15).



*Figure 10. Tree Model in an application (15)*

### E. Relationship model

The relationship model describes the relationship between two entities. You can describe the relationship, for example, whether it is of type 1-n, n-n, etc. You can add captions that will be used when the relationship data is displayed and also specify document models that containadditional link fields that will be associated with that relationship (15).

The relationship model is created and edited in the Simple Model Editor and saved as a JSON file.

### F. Application model

The application model is vital for designing model-based applications, incorporating models like forms, trees, and overviews. It acts as a blueprint for their arrangement. Within this model, layouts are structured, including features such as master/detail design, where documents are listed in overviews or trees. When selected, details appear in a side form. Various layouts cater to different needs (15).
Using the application model adds new modules to the application by leveraging existing models. This integration fosters gradual expansion. The Simple Model Editor facilitates application model editing and saves it as a JSON file (15).

### G. Relationship model

This model describes the UI components that are used to display the relationship data and their properties and configuration. You can select from several different UI controls:

- Drop Down Selection

- Dual Pane Selection

- Table List

You select which overview models should be used in the controls to display available and selected items and which forms should be used to edit link fields. You can also set custom labels on the Dual Pane Selection in this binding (15).

The binding is created and edited in the Simple Model Editor within the form model editor and it is saved in the form model document using annotations.

## 3.8. Customization and Flexibility in A12

A12 offers an extensive level of customization, which is contingent upon specific user demands. Those utilizing A12, often referred to as modelers, possess the capability to construct comprehensive applications that align seamlessly with distinct business scenarios. This encompasses a diverse spectrum of components, including various data fields, intricate computations, rigorous validations, intricate relationships, orchestrated workflow processes, permissions hierarchies, diverse views, and templates, among other elements. However, certain intricate features may extend beyond the platform's inherent capacities. In such instances, the involvement of developers becomes necessary to tailor suitable solutions and realize the desired features. This customization can be implemented on every individual component employed within the application.
Alls that's happening by APIs provided by each component in A12, which facilitates the process of customization (15).

## 3.9. Developer Productivity and Collaboration

A12 platform brings forth a pivotal advantage: the potent enhancement of collaborative dynamics and seamless interaction between developers and business analysts. This distinctive attribute originates from A12's modeling tools, which afford Business Analysts the means to articulate precise requisites within the model framework. Subsequently, developers find themselves empowered with a streamlined pathway to address supplementary needs emerging from the insights of Business Analysts. These requisites may entail nuanced adjustments to behaviors or layouts, catering to the distinctive exigencies of clientele.

This symbiotic interplay between Business Analysts and developers unfolds as a cornerstone of efficacy. The former, utilizing A12's modeling tools, effectively communicates their requisites with exactitude. This mode of expression eradicates ambiguity and possible misunderstandings, thereby furnishing developers with a robust blueprint. Such clarity serves as a guiding compass, enabling developers to hone their efforts with a clear sense of direction.

This orchestrated harmony between Business Analysts and developers galvanizes a harmonious and cooperative software development trajectory. The modeling-driven approach bridges the linguistic divide between business requisites and technical implementation, thus precluding the likelihood of disjunction arising from disparate viewpoints, ultimately contributing to the holistic potency and value proposition encapsulated within the A12 platform.

# 4. Practical Part

## 4.1. Motivation and planning

The reason behind studying and analyzing the Low Code Platforms is informed by my professional experience at a company that develops a model-driven Low-Code Development Platform (LCDP) for enterprise business applications, where I noticed the implementation of LCDP has significantly accelerated application development processes, evidencing a substantial reduction in implementation time.

Notably, this platform enhances the quality and security of applications, leveraging standardized coding practices and robust security protocols. A critical observation is the elimination of repetitive coding tasks, allowing developers to concentrate on more innovative aspects of application development.

What got my attention more is the utilization of a model-driven approach in LCDP is instrumental in enterprise application development. This approach facilitates efficient and rapid translation of business requirements into functional applications. A key advantage is the integration of business analysts into the development process, fostering a collaborative environment. This inclusivity enhances mutual understanding and aligns development efforts more closely with business objectives, ultimately leading to more effective and tailored business application.

## 4.2. Implementation

The initiation of the practical part involves delineating the essential tools and the development environment required for building applications using the A12 Low-Code Platform (LCP).

As a model-driven platform, A12 necessitates the creation of models, which are essentially JSON files, generated using the platform's integrated tool, the Simple Model Editor (SME). These models, once generated, are then imported into the A12 base development project, which is the base template for most of the applications that use A12.

This project includes boilerplate code that facilitates interaction with and customization of these models, thus laying the groundwork for application development.

### 4.2.1.  Project Conceptualization

Project Conceptualization for University Admin System:

**Purpose**: Develop a comprehensive system for university administration to manage student and professor data efficiently using the A12 Low-Code Platform.

**Target Users**: Primarily designed for university administrators to streamline data management, enhance information accessibility, and improve administrative efficiency.

**Functionality Overview**:

- **Students Tab**: Facilitates adding and managing student information, including personal details and enrolled subjects.
- **Professors Tab**: Enables adding and tracking professor profiles, including teaching subjects.
- **Subjects Tab**: Allows managing subject details and associated teaching staff.
- **Dashboard**: Provides analytical insights through charts, summarizing key data about students, professors, and subjects.

**Unique Aspects**: Leveraging A12's model-driven approach, the system enhances data management efficiency, promotes user-friendly interfaces, and ensures scalability to adapt to evolving university needs.

### 4.2.2.  Design and Development

In this section I will define the requirements we need to develop an application using A12 and explain the process of creating the school system management application, step by step as follows:

## Step 1: Project Template and Application Frame

Let's start by discussing the Project Template file structure and its logic.
It mainly consists of 3 folders:

- **Client:**
  This folder represents the frontend side of the application where we can add new models, customize the existing ones using React, and apply some client actions using Redux and Redux Saga.
- **Server:**
  This folder represents the backend side of the application where we handle the creation, deletion, and update of the documents, User Authorization, and much more using Java and Spring Boot.
- **Import:**
  In this folder, we add our models generated from the SME

And many other folders and files that contain some dependencies, resources, configuration, and e2e testing scripts.

Now when we run the application we have to run commands on both the client and server folders as follows:

---

**1. Build the application modules**
```
gradle build
```

**2. Run**

1. Compose up the Keycloak container in Docker:
   ```
   gradle keycloakComposeUp
   ```

   > **WARNING**: Project Template's Keycloak setup is for development purposes only. It is necessary to significantly enhance the security of a Keycloak instance for production environments.

2. Run the server application with the default development Spring profile and keep it running:
   ```
   gradle :server:app:bootrun --args='--spring.profiles.active=dev-env'
   ```
3. Run client
   1. In another terminal window, move to client directory with `cd client`.
   2. Then start the webpack with `npm start` and keep it running.

---

After having the application successfully running then we can log in with three test users already set on the project template.

And here is what we see after login:

The whole of what we see in Figure 11 is called the Application Frame.

**Application Frame:**

It is a pre-defined skeleton to hang our application features based on the A12 Plasma design system. With this, we can worry less about the small UI details of the application knowing that we will end up with something that conforms to the A12 design system.

**Where I can find it on the Project Template?**

in the file `client/src/app/layoutProvider.tsx`. There you can find a component named `CustomApplicationFrameLayout` which returns something called `ApplicationFrameLayout`.



*Figure 11. Application Frame*

# Step 2: Modelling

In this section, we explore model creation using A12's tools. This step is vital in A12 application development, as it involves designing the data structures and workflows that form the application's core. This modeling phase is foundational, shaping the application to meet specific functional requirements.

In developing our University CRM system using A12, we focus on the contacts and subject management functionalities. This includes creating, viewing, updating, and deleting subjects and contacts with key details like titles, concepts, names, and email addresses. Traditionally, this requires building forms on the client side, considering mandatory fields, user error

notifications, accessibility, and mobile compatibility.
On the server side, it involves setting up a database and CRUD endpoints, with a focus on security.
A12 simplifies this process, allowing even non-developers to handle these tasks efficiently, with options for code customization based on business needs.

Initially, we begin with launching the Simple Model Editor (SME), the main tool for creating and editing our models. Figure (12) shows the SME interface:
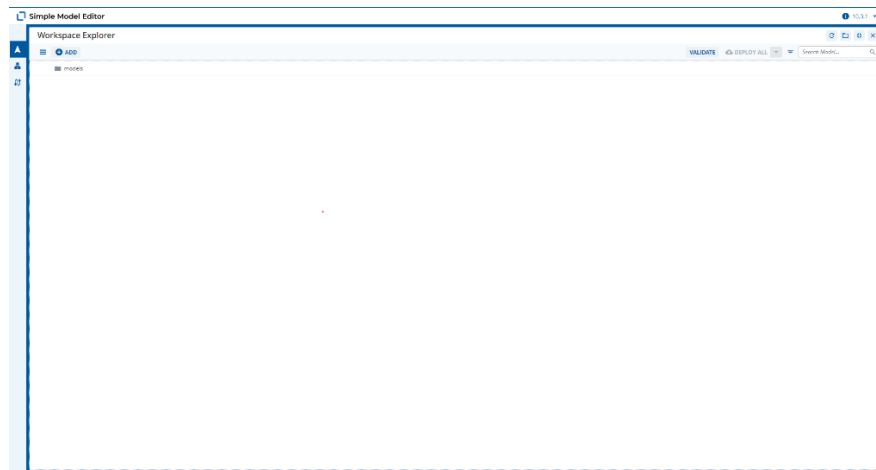


*Figure 12. SME Interface*

Click 'New' to create a document model from the dropdown menu, selecting from the model typespreviously mentioned. This step initiates the creation of a new document model.

Next, populate the DM with essential data such as its name, designated storage folder, locales, andassociated roles, the first model to be configured will be the 'Student' model.



*Figure 13. Document Model Creation Form*

When the DM is created. This opens a page for further customizations, such as adding variables with their types and outlining the data structure, essential for shaping the student DM.

In the DM, data organization entails creating new groups or directories for the Student DM data, as depicted in Figure 14.

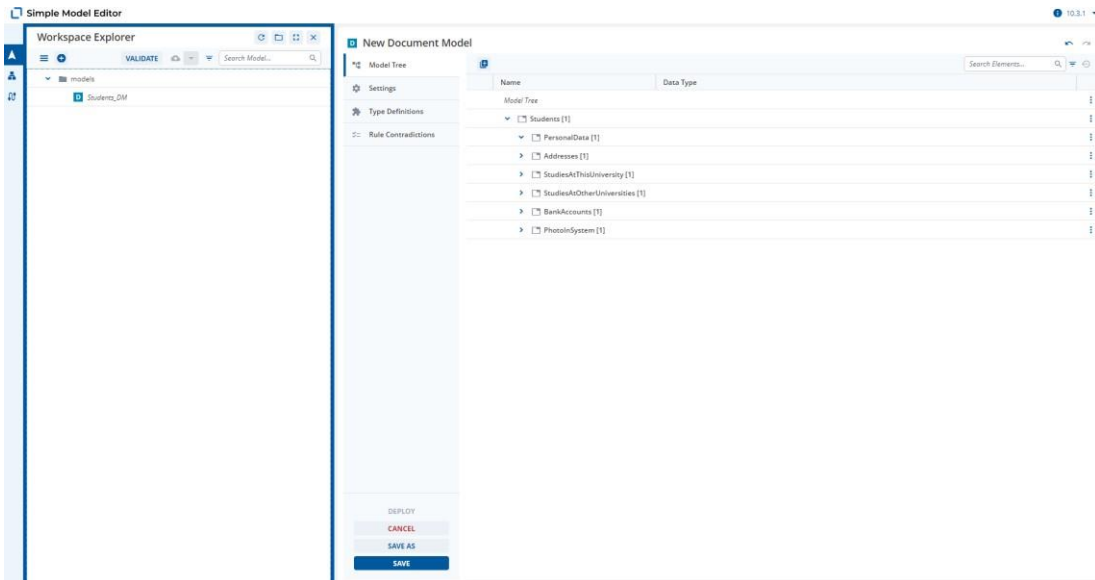This approach ensures a clear arrangement within the Student model.



*Figure 14. Student DM Groups*

To add a new field to the PersonalData folder in the DM, follow the steps demonstrated in Figure 15. This process allows specifying various attributes for each field, including the field name, data type, configuration, labels, description, helper text, and annotations.
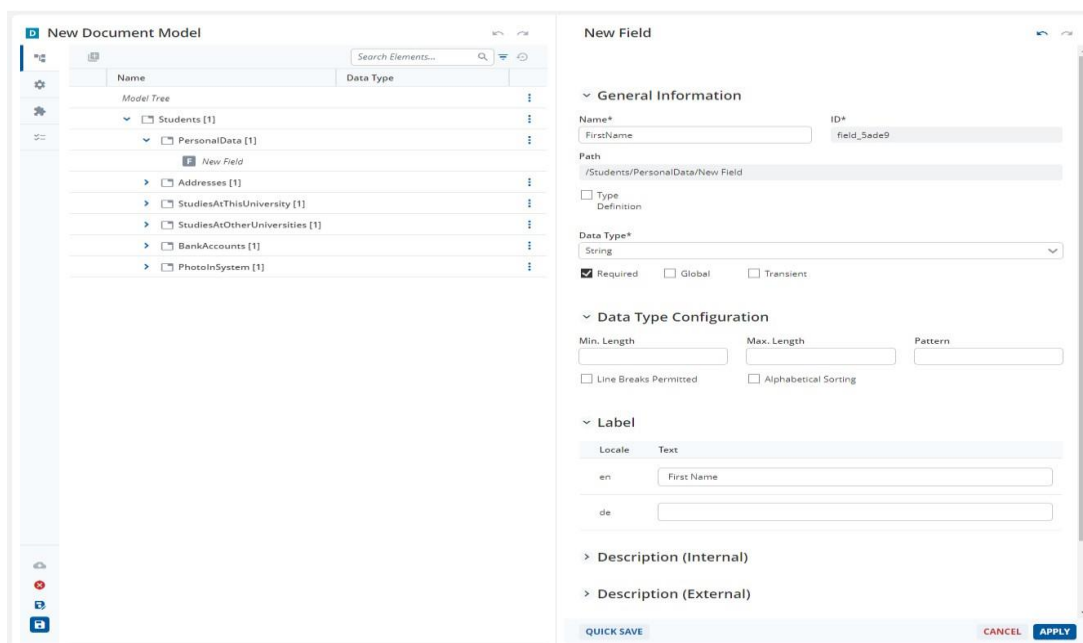


*Figure 15. Student DM New Field*

Similar steps were followed to add other fields to the student model, as can be seen in Figure 16.
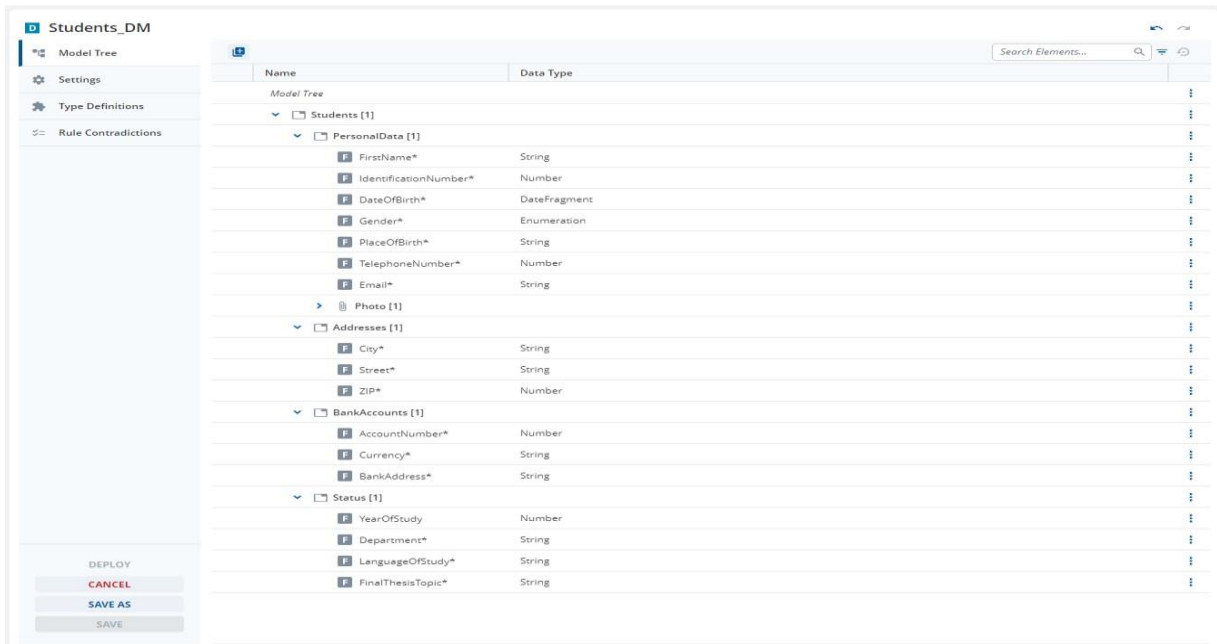


*Figure 16. Student DM Fields*

To design the structured fields in the DM, a new Form Model (FM) is created for form design, allowing customization of elements like colors, sizes, alignments, and icons.
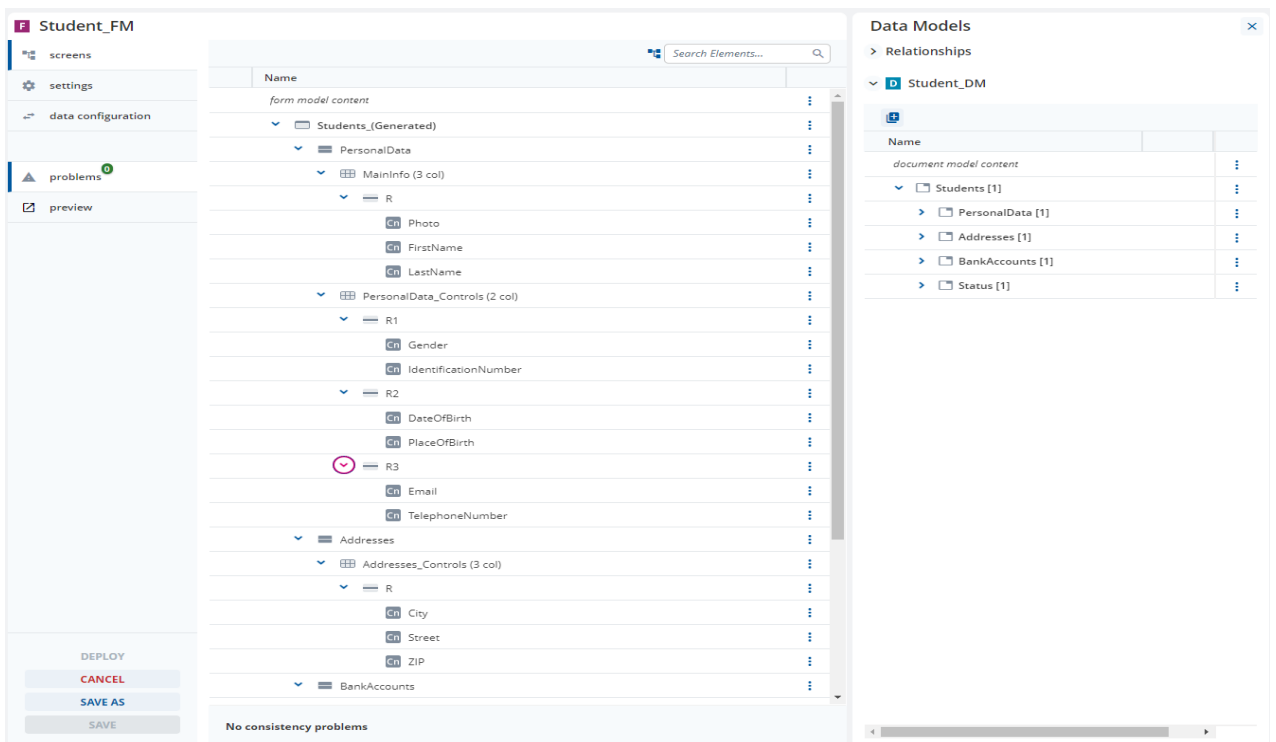The process for creating an FM is similar to that used for the DM.



*Figure 17. Student Form Model after Fields Organization*

The 'Preview' button displays live progress in a separate window, as shown in Figure 18. This allows for a quick view of the current state of the form being designed in the Form Model (FM).
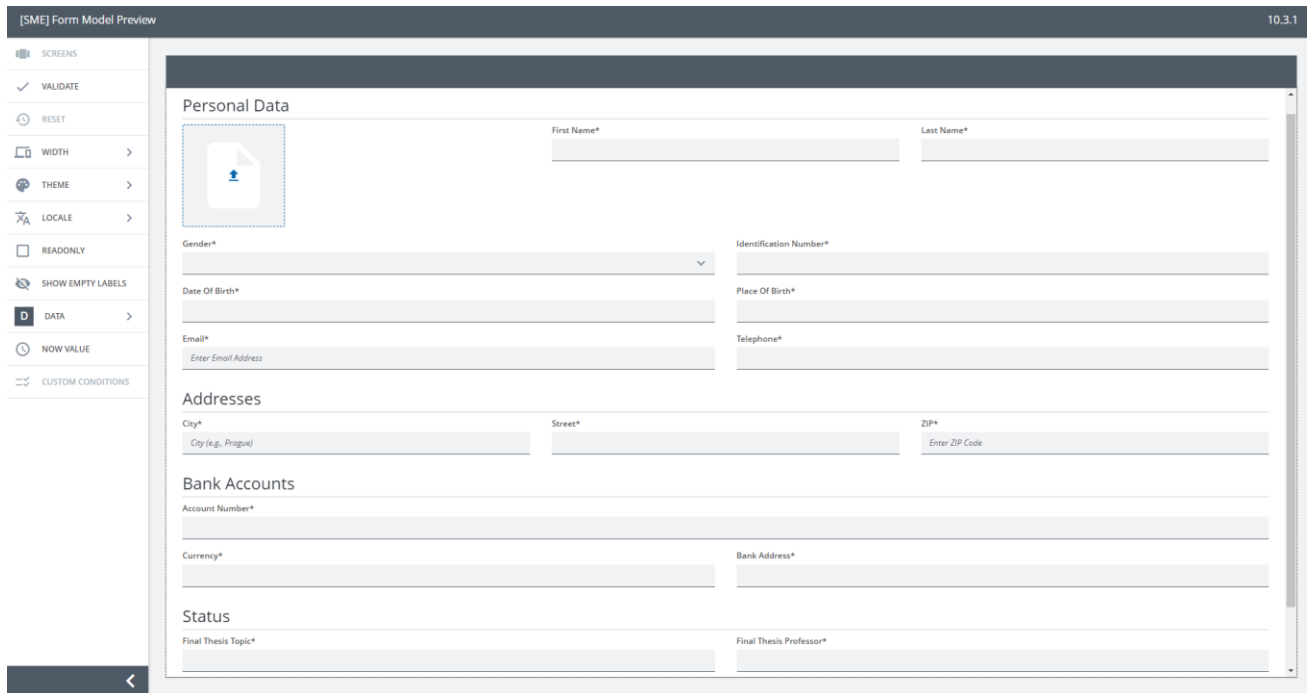


*Figure 18. Live Preview of Student Form Model*

To incorporate events like cancel and save into the form, use the settings tab to define these events. This includes setting labels, event types, validation rules, icons, and more.

To display forms in a table after saving or submitting them, an Overview Model (OV) is required. This model type is different from others and is created in the same manner, through the modelsfolder.

The Overview Model (OV) facilitates column selection for form data representation. Choices for columns are made from a dropdown list, allowing specification of labels, icons, and sorting types. This process is exemplified in Figure 19, showcasing a list of columns selected for display in the OV.
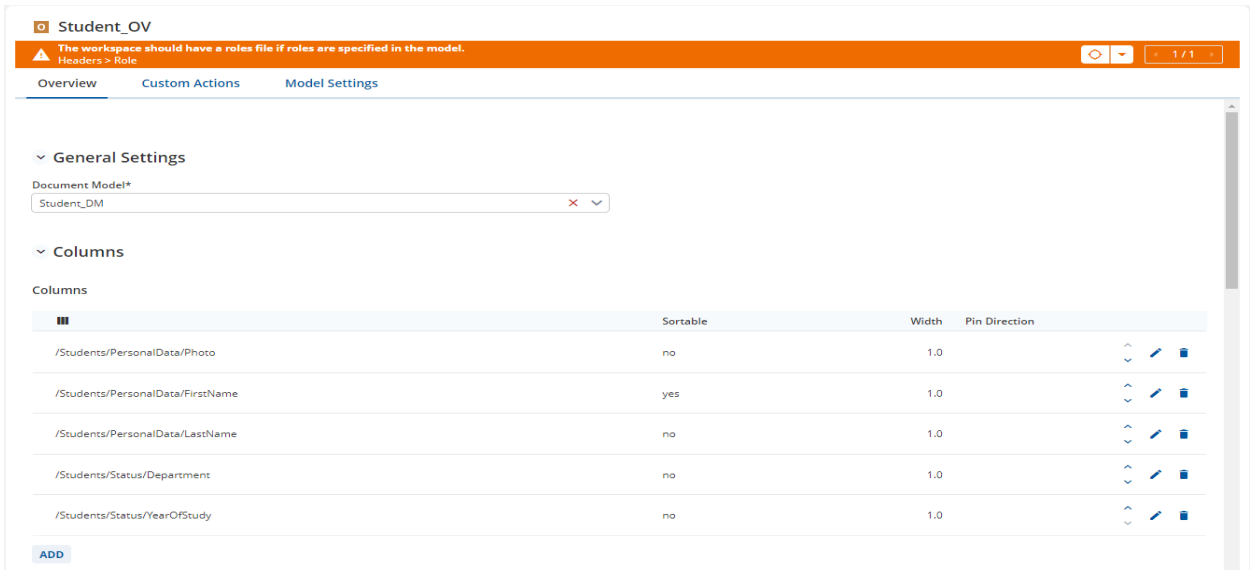
*Figure 19. Student Overview Columns*

Add and delete functionalities for the Overview Model (OV) are set up in the Custom Actions tab. This allows for managing submitted forms within the OV. The configuration process, including the selection and definition of these actions, is detailed in Figure 20.
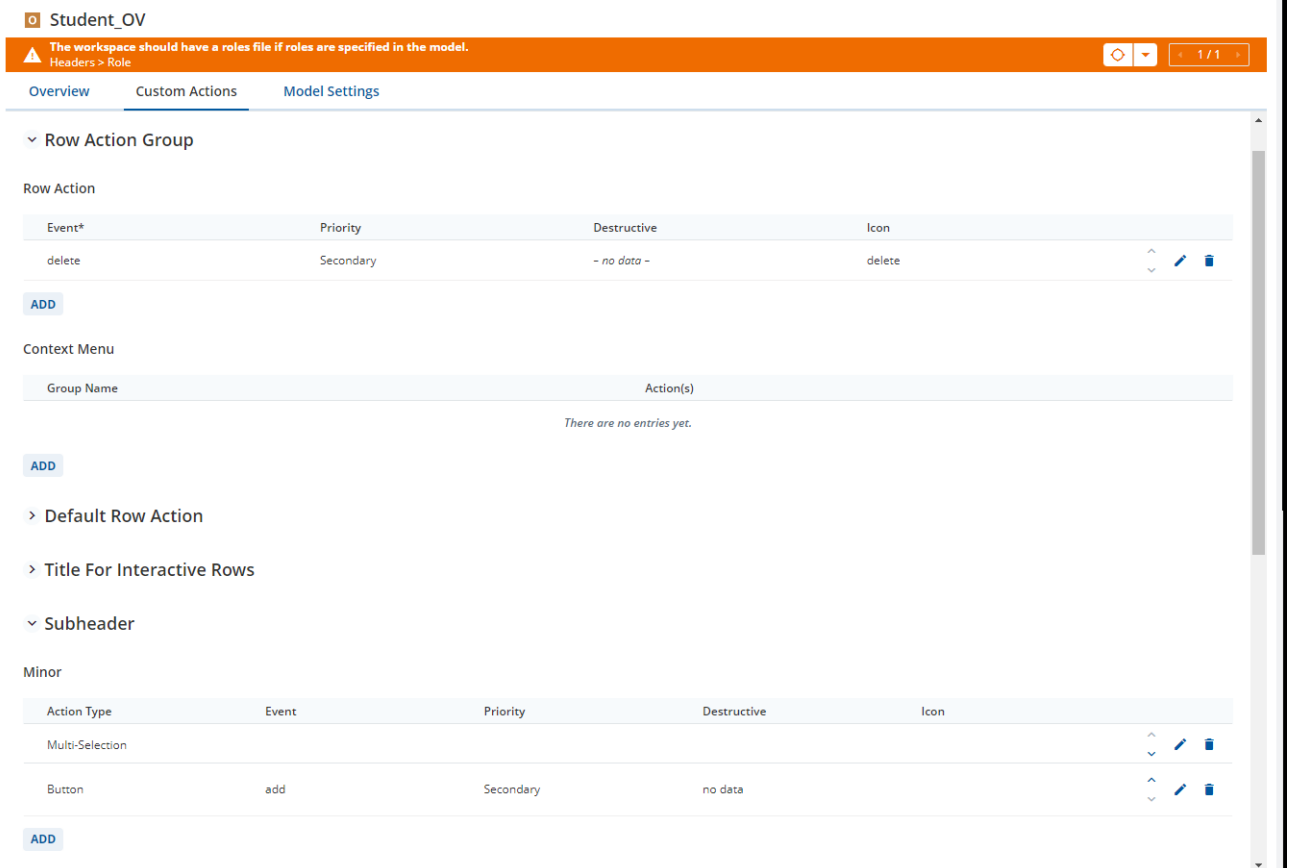


*Figure 20. Student Overview Save and Delete Events*

Following the creation of the models, the next step is their integration into the A12 platform's development environment (Project Template). The integration process involves creating a new module in the PT for the models. This integration is a critical component of the model-driven development approach that A12 utilizes, where models form the basis of application functionality and workflow.

To continue the development in A12, the first step is to add a new folder named "student" in the `client/src/modules` directory. Following this, an Application Model is created using the SME, similar to the earlier models. This Application Model is crucial as it manages and connects all the existing models, facilitating the representation of different layouts in the application.

Create an index.ts file as following:

```typescript
import { Module } from "@com.mgmtp.a12.client/client-core/lib/core/application";
import { ApplicationModel } from "@com.mgmtp.a12.client/client-core/lib/core/model";

import * as model from "./student-appmodel.json";

const module = (): Module => ({
    id: "StudentModule",
    model: () => model as ApplicationModel
});

export default module;
```

The final step is to incorporate the StudentModule into the application's central module array. This is executed by modifying the `client/src/index.tsx` file, wherein the StudentModule is added to the ALL_MODULES array. This integration is essential for the StudentModule to be recognized and function correctly within the application's architecture, following the principles of model-driven development in A12.

```typescript
import studentModule from "./student";

export const ALL_MODULES = [studentModule()];
const moduleRegistry = ModuleRegistryProvider.getInstance();

/**
 * Get all modules.
 */
export const getAllModules = (): Module[] => {
    return ALL_MODULES;
};
```

After successfully integrating the Student module, we proceeded to add the Professor and Subject modules similarly to the application. Once the building and running processes were completed, the outcome effectively demonstrated the integration and functionality of all the created and configured models, utilizing the A12 modeling tool (SME)
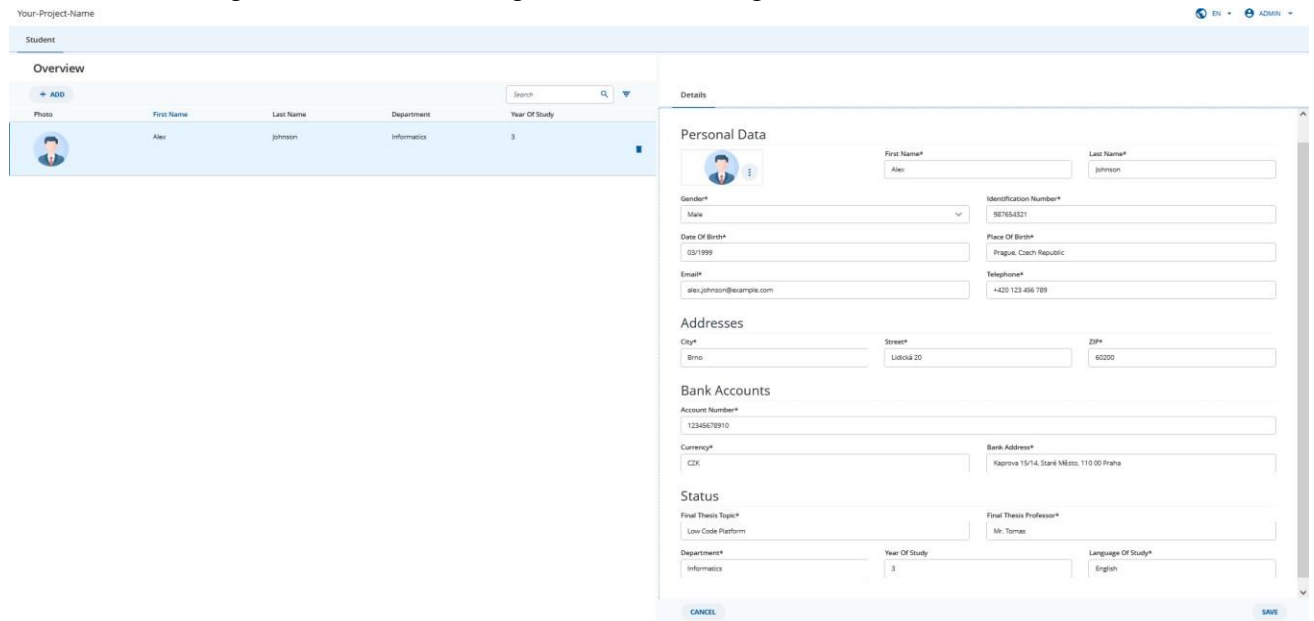


*Figure 21. Final Application Interface: Demonstrating the Integrated StudentModule in Action*

The next step in our development process is to implement the system dashboard. Which will be accessible via a separate tab, designed specifically for the system administrator.
It will feature an insightful representation of module statistics, prominently displayed using a Pie chart.

To achieve this, we will be integrating custom code into the dashboard module. This step is crucial as it exemplifies the practical application of the low-code approach in real-world scenarios. It involves leveraging the A12 platform's capabilities to create custom functionalities that complement the existing low-code features, thereby providing an interactive dashboard.

## Step 3: Adding Custom code to A12

In this step, we customize the app by adding a 'Dashboard' tab. This new feature will display data through charts, improving insights and usability within the A12 platform.

To begin, we incorporate the Pie Charts component, which will visually display our data. This component is added under
`client/src/modules/dashboard/components/PieChart.tsx`
designating it as the primary means for data representation within the dashboard.

Finally, integrate the PieChart into the dashboard by placing it in a container found at
`client/src/modules/dashboard/components/PieChartContainer.tsx`,
embedding it within the module's interface.

```
export default function PieChartContainer({ activity }: View): ReactElement | null {
   const dataHolder = Activity.findDefaultDataHolder(activity);
   const localizer = useLocalizer();

   if (!dataHolder?.data || dataHolder?.busy) {
      return null;
   }

   const { chartData = [] } = dataHolder.data as { chartData:
PieChartElementsProps.ChartData[] };

   const getLabel = (resourceKey: string) => {
      const localizedValue = localizer(resourceKey);
      return localizedValue || resourceKey;
   };

   // Some values don't need localization e.g. names of contacts
   // If value is found in resources return it else return the name passed in
   const localizedChartData = chartData?.map(d => ({ ...d, name: getLabel(d.name) }));

   return (
      <>
         <ActionContentbox
            className="-u-max-width-2xl -u-height-full"
            headingElements={
               <ContentBoxElements.Title
                  ariaLevel={2}
                  key="title"
                  text={localizer(RESOURCE_KEYS.dashboard.title)}
               />
            }
         >
               <CustomPieChart label={localizer(RESOURCE_KEYS.dashboard.chart)}
data={localizedChartData} />
```

A12 seeks an approach to load data, utilizing a feature known as a data provider. By integrating a data provider into our application, specifically through Redux Saga watchers, we establish a robust mechanism for data retrieval. This setup enables functions that facilitate HTTP calls, action dispatches, or retrievals from the Redux store

Within the application structure, the pieChart.ts data provider will be added under the path: `client/src/modules/dashboard/dataProvider/pieChart.ts`

```
import { put } from "@redux-saga/core/effects";

import { ActivityActions } from "@com.mgmtp.a12.client/client-core/lib/core/activity";
import { DataProvider } from "@com.mgmtp.a12.client/client-core/lib/core/data";

import { ChartData } from "../types";

const colors = ["#00589F", "#0081BD", "#00A8BD", "#00CAA3", "#8AE682", "#F9F871"];

export const pieChartDataProvider: DataProvider = {
  name: "pieChartDataProvider",
  canHandle({ dataHolder, operation }) {
    switch (operation) {
      case "load":
        return dataHolder.descriptor.view === "Dashboard";
      default:
        return false;
    }
  },
  *provideData({ activityId }) {
    // This is where you would make a request for your data
    const chartData: ChartData = {
   // "customerType.vip" is a localization key
      chartData: [{ name: "userType.vip", color: colors[0], value: 10 }]
    };

    // Fill activity with a new dataHolder containing your data
    yield put(
      ActivityActions.setData({
        activityId,
        data: chartData
      })
    );
  }
};
```

After adding the data provider, it's linked within the module for targeted functionality, specifically at `client/src/modules/dashboard/index.ts`. This step connects the provider to the dashboard module, enabling data handling and visualization.

```
// ...
import { pieChartDataProvider } from "./dataProvider/pieChart";
// ...
const module = (): Module => ({
    id: "DashboardModule",
    model: () => dashboardAppModel as ApplicationModel,
    views: () => viewComponentProvider,
    dataProviders: () => [pieChartDataProvider]
});
// ...
```

To retrieve data, we initiate an RPC request to the A12 Data Services (Backend) as outlined below:

```
{
    id: "getStudentsByType",
    jsonrpc: "2.0",
    method: "LIST_DOCUMENTS",
    params: {
        // The document model we are interested in
        documentModelName: "Student_DM",
        // We want to group our students by type
        facets: [
            {
                id: "studentType",
                type: "term",
                field: "Students.PersonalData.FirstName"
            }
        ]
    }
}
```

The JSON-RPC structure includes fields like jsonrpc for the protocol version, id for request identification, and method for server actions, alongside params for specific instructions.

y applying the same method, we add a JSON-RPC call for each model, utilizing localized data to display within the PieChartContainer.tsx. This approach culminates in a comprehensive dashboard overview, showcasing the integrated data visualizations.
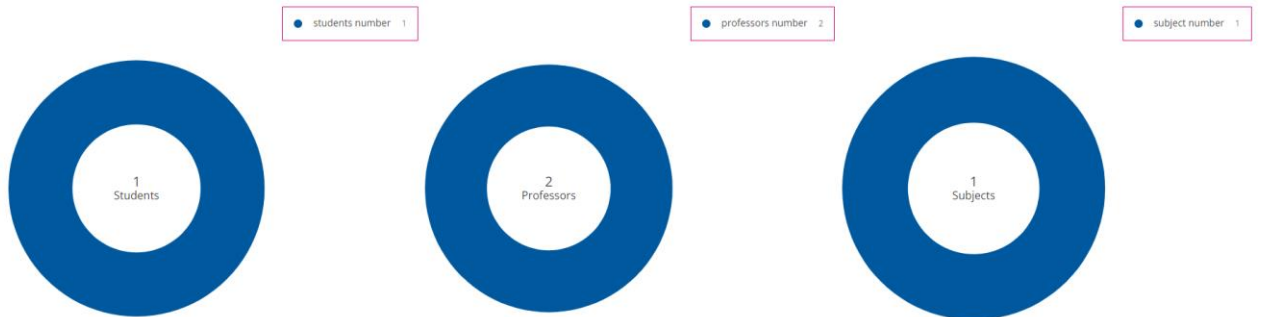
*Figure 22. Dashboard Pie Charts*

The Pie charts on the Dashboard overview represent the Students, Professors, and Subject counts.

## 4.3. Survey Analysis

To explore the impact of A12 as a Low Code Platform, I carried out two surveys: one for developers and another for project managers. These surveys aimed to capture their experiences and views on A12's effectiveness in streamlining development and project management tasks. This effort sought to derive insights from both technical and managerial perspectives, enriching our understanding of LCDP's utility in real-world applications.

Insights were garnered from a select group within mgm technology partners (mgm-tp), specifically targeting those involved with the A12 platform. This group included 5 project managers and 21 developers, all employees at mgm-tp. The project managers, with backgrounds in business, project management, and IT, and the developers, skilled in front-end, back-end, and full-stack development, provided a well-rounded perspective on A12's application and development.

This focused selection was facilitated by my position as a software developer at mgm-tp and working on business applications using A12, enabling me to directly reach out to colleagues engaged with A12.

### 4.3.1. Survey by Developers

The developer survey aimed to assess the effectiveness and user experience of A12 as a Low Code Platform among our development team. We successfully engaged 21 developers, gathering their insights and feedback through a series of targeted questions.

**Quantitative Results:**

The survey's quantitative analysis is visually represented in charts, highlighting developers' satisfaction levels and their perceived ease of use of A12.
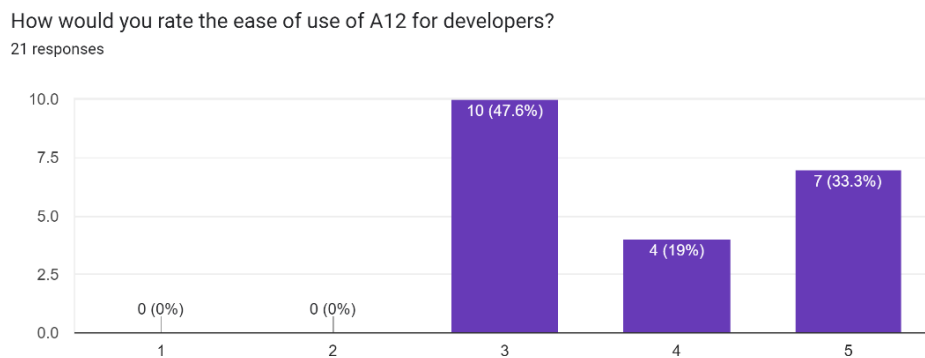
How would you rate the ease of use of A12 for developers?
21 responses



*Figure 23. Ease of Use Rating by Developers*

How much do you think A12 eases the software development process?
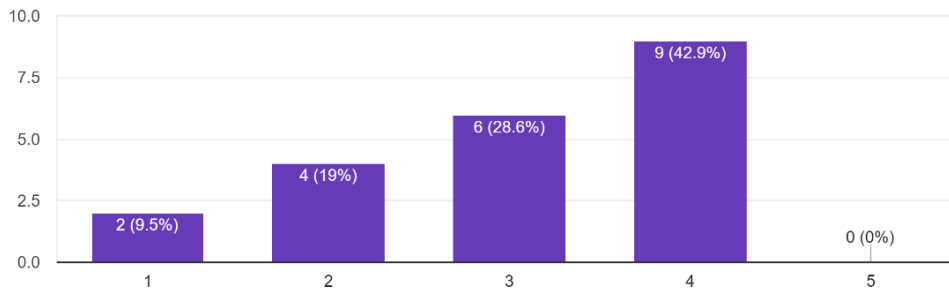21 responses



*Figure 24. Enhancing the Software Development Process*

## Qualitative Results:

| Theme | Benefits Experienced | Impact on Productivity | Overall Impression |
|---|---|---|---|
| **Speed & Efficiency** | Faster application building a quick project setup | Mixed: Some found it faster, others faced a learning curve | Positive for specific uses; challenges in customization |
| **Learning & Support** | Opportunity to learn new technologies | Diverse: From decreased to unchanged productivity | Suggestions for better documentation and more intuitive UI |
| **Functionality** | Basic UI elements and features out-of-the-box and Effective model validation | Productivity boost in common use cases; challenges with non-standard requirements | Calls for more integrative features and simpler customization |
| **User Experience** | Separation of concerns leads to less code, a Stable core, and built-in security | Improvements in UI development speed; initial slowdown for beginners | Desire for a more user-friendly and flexible tool |

*Table 2. Comparing survey answers by developers*

47

### 4.3.2. Survey by Project Managers

The survey targeting project managers focused on evaluating A12's impact on project workflows, efficiency, and overall management experience. A total of 4 project managers participated, offering insights into how A12 influences project delivery and oversight.

**Quantitative Results**

How has A12 influenced collaboration between developers and business analysts in your projects?
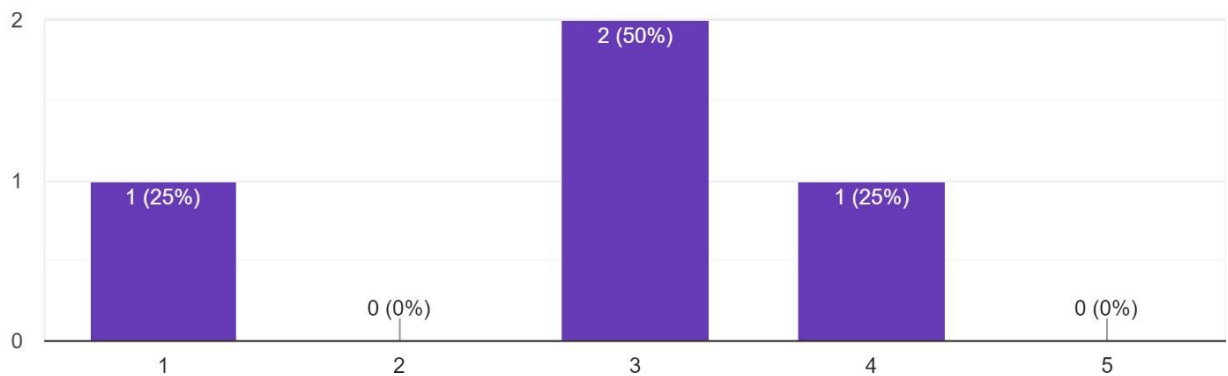4 responses



*Figure 25. Collaboration between Developers and Business Analysts in A12*

The bar chart visualizes the responses from project managers regarding the influence of A12 on collaboration between developers and business analysts. With a scale where 1 represents 'highly effective', the majority (50%) rate the influence of A12 as moderately effective (3 out of 5), indicating room for improvement. A quarter (25%) perceive it as highly effective (1 out of 5), whileanother 25% rate it less effective (4 out of 5), suggesting varied experiences and the potential for A12 to better facilitate collaboration in certain projects.

## 4.4. SWOT Analysis Comparison of Low-Code Platforms

| Category | WordPress | Microsoft Power Apps | A12 |
|---|---|---|---|
| **Strengths** | Highly user-friendly, extensive ecosystem of themes and plugins, and strong community support. | Deep integration with Microsoft ecosystem, AI capabilities, and an intuitive drag-and-drop interface. | Efficient modeling capabilities for both front-end and back-end, support complex application logic, and customizable components. |
| **Weaknesses** | Focuses on web content management, additional coding, or plugins required for custom functionalities | Higher learning curve for non-technical users and reliance on Microsoft products. | Initial learning period to master full capabilities |
| **Opportunities** | Expanding capabilities beyond CMS to full-scale web applications. | Expanding use in businesses not yet fully integrated into the Microsoft ecosystem. | Filling a niche for rapid prototyping and development within enterprise-level applications. |
| **Threats** | Emergence of more integrated platforms offering web content management alongside application development. | Competition from platforms offering greater flexibility and lower dependency on a specific ecosystem. | Fast-evolving landscape of low-code platforms could introduce competitors with similar or enhanced functionalities. |

*Table 3. Comparative SWOT Analysis of Low-Code Development Platforms*

# Comparative Insights from SWOT Analysis

a) **Flexibility and Use Case Suitability:**
A12 excels in modeling capabilities for complex applications.
Power Apps provides robust integration within the Microsoft ecosystem.
WordPress is best for web content but lacks depth in application development.

b) **Ease of Use and Accessibility:**
WordPress is the most accessible for beginners.
A12 and Power Apps cater to those with some technical knowledge or specific integration
needs.

c) **Integration Capabilities and Ecosystem Compatibility:**
Power Apps is ideal for Microsoft environments but may limit others.
WordPress and A12 offer broader integration ranges, with WordPress boasting a vast plugin
library and A12 providing customizable components.

# 5. Results and Discussion

This thesis has explored the utilization of LCDPs to expedite application development and decrease dependence on traditional coding methodologies.
The practical application of A12 and surveys from developers and project managers have provided a wealth of data.

## Analysis of Results

The results have shown that LCDPs can speed up the development process and make coding simpler. The surveys back this up, showing a trend towards increased efficiency and productivity among users of A12.

## Discussion of LCDP Limitations

Despite these benefits, there are several limitations founded:

**Scalability:** Some LCDPs may not scale efficiently for larger or more complex projects.
**Customization:** There is often a trade-off between ease of use and the ability to customize deeply.
 **Integration:** There are some challenges in integrating with existing systems and some technologies.

## Solutions and Recommendations

- Enhancing LCDPs with scalable architecture options.
- Developing advanced customization capabilities within LCDPs while maintaining user-friendliness.
- Making integration features and documentation better to ensure everything works together smoothly

## Reflection on Survey Results

The survey responses from project managers have highlighted varied impacts on collaboration between developers and business analysts, indicating that while LCDPs offer many advantages, thereis a need for improved features and training to maximize collaborative efforts.

# 6. Conclusion

## Confirmed Findings

This study has shown that Low-Code Development Platforms can speed up app development and cut down on complex coding.
A12, in particular, accelerates the application development process and reduces some of thedependencies on traditional coding practices.

## Observations

The practical work and surveys revealed that while LCDPs bring efficiency to the development cycle, there are observable challenges. These include scalability concerns, a need for more strong customization options, and complexities in integrating with existing systems.

## Future of LCDPs

LCDPs are likely to become more intuitive and capable, with the h potential for smarter features and better team tools.
The trend is towards making these platforms more efficient for various applications.

## Implications

The findings suggest a move towards easier development approaches and highlight a need for LCDP improvements.
Addressing current LCDP challenges will be crucial for their broader adoption in software development.

In short, LCDPs are changing software development, offering a simpler way forward. This thesis adds to the understanding of their role and future in the tech industry.

# References

1. Low-cod platform for automating business process in manufacturing, Waszkowski RIFAC-PapersOnLine (2019) 52(10) 376-381
2. Viktor Kalinichenko. (2022). No-Code/Low-Code: Origins. https://devtorium.com/blog/no-code-low-code-origins [Online]
3. Microsoft-PowerApps. https://powerapps.microsoft.com/en-us/low-code-vs-traditional-development [Online]
4. Benefits and limitations of using low-code development to support digitalization in the construction industry, Martinez Epfister. Automation in Construction, (2023), 152
5. John Everhard. (2019). The Pros And Cons Of Citizen Development. https://www.forbes.com/sites/johneverhard/2019/01/22/the-pros-and-cons-of-citizen-development
6. OustSystems. (2019/2020). The State of Application Development Is IT Ready for Disruption.
7. Daniel Rasch. (2023). mgm strengthens the low code movement as a new member of the low code association. https://insights.mgm-tp.com/de/mgm-technology-partners-ist-mitglied-der-low-code-association [Online]
8. Kristin Mueller. (2023). Quality assurance of model-based enterprise software: increasing efficiency with the mgm approach. https://insights.mgm-tp.com/de/qualitaetssicherung-von-modellbasierter-enterprise-software [Online]
9. Christine Reiner. Low code in industrial insurance: mgm Cosmo using the example of "partner data". (2023). https://insights.mgm-tp.com/de/der-low-code-basierte-ansatz-von-mgm-cosmo-am-beispiel-partnerdaten [Online]
10. Lilia Gargouri. (2023). Deepdive A12 end-to-end test automation. https://insights.mgm-tp.com/de/deepdive-a12-end-to-end-test-automatisierung [Online]
11. Modelling in low-code development: a multi-vocal systematic review, Bucaioni ACicchetti ACiccozzi F, Software and Systems Modeling (2022) 21(5) 1959-1981
12. mgm-tp. (2023). A12 Platform. https://www.mgm-tp.com/documents/A12-Product-News-2023.06-LTS-EN.pdf
13. An empirical evaluation of scrum training's suitability for the model-driven development of knowledge-intensive software systems, Shafiee SWautelet YPoelmans S et al, Data and Knowledge Engineering, (2023), 146
14. LeBLANC Team. (2022). INTRODUCTION TO MODEL DRIVEN DEVELOPMENT. https://leblanc.fi/model-driven-development.html
15. Get A12 (A12 Documentation). (2023). A12 Development teams.
16. Terence Nero. (2021). Low Code Platform: The Future of Software Development. https://www.cuelogic.com/blog/low-code-platform. [Online]
17. Kissflow. (2023). Low-Code Application Development Platform (LCAP). https://kissflow.com/application-development/low-code-application-development-platform. [Online]
18. Cyclr. (2023). Low-Code is revolutionising the software industry. Low-Code is revolutionising the software industry: what type is dominating? (cyclr.com) [Online]

# 2. List of pictures, tables, graphs, and abbreviations

## 1.1 List of pictures

## 1.2 List of tables

## 1.3 List of abbreviations

| Abbreviation | Full Meaning |
|---|---|
| **LCP** | Low-Code Platform |
| **BAP** | Business Application Platform |
| **UI** | User Interface |
| **API** | Application Programming Interface |
| **BPMN** | Business Process Model and Notation |
| **UAA** | User Management, Authentication, and Authorization |
| **SME** | Simple Model Editor |
| **DM** | Document Model |
| **FM** | Form Model |
| **OM** | Overview Model |