INSTITUTO SUPERIOR
DE TECNOLOGIAS
AVANÇADAS DE LISBOA

From Code to Components: Analyzing the Shift from Traditional Mobile Development to Low-Code Platforms

Mircea Andrei Avram

Dissertação para obtenção do Grau de Mestre em

INFORMÁTICA

Presidente: Professor Doutor Pedro Ramos dos Santos Brandão

Arguente: Professor Doutor Lúcio Miguel Studer Ferreira

Orientador: Professor Doutor Paulo André Reis Duarte Branco

Setembro, 2024

ISTEC

Instituto Superior de Tecnologias Avançadas

Campus Académico do Lumiar, Lisboa

Dissertação

Mestrado em Informática

Mircea Andrei Avram

Dissertação realizada sob a orientação científica do Professor Doutor Paulo André Reis Duarte Branco apresentada no ISTEC - Instituto Superior de Tecnologias Avançadas de Lisboa para obtenção de grau de Mestre em Informática.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my entire family for their infinite support throughout this journey. I am especially thankful to my parents for allowing me to have this opportunity in life and giving me everything that I ever needed, their constant encouragement and for being there for me every step of the way. I am also grateful to my cousin, for his constant support and encouragement every time that I felt I wanted to give up. Your love and belief in me have been a source of strength and motivation.

To my girlfriend, thank you for always believing in me and my capabilities. Your faith in me has been an anchor throughout this challenging process, and I am incredibly grateful for your continuous support, love and reassurance.

To my friends, thank you for your encouragement, companionship, and understanding throughout this journey. Your support has meant the world to me and helped me stay grounded during difficult times.

I would also like to extend a special thanks to my professor and tutor, Paulo Duarte. Your guidance and mentorship have been invaluable throughout this degree. Thank you for taking me under your wing and helping me navigate this journey with your expertise, patience, and dedication.

Abstract

In recent years, software development has changed due to the emergence and increase of low-code development platforms. These platforms, like OutSystems, promise to revolutionize the development process by offering a more accessible and efficient means of creating software applications. The software development community is still divided on the ramifications of this new approach compared to more conventional development techniques.

The accelerating demand for software solutions across various industries has underscored the need for agile and efficient development methodologies. The growth of low-code platforms represents a promising solution for meeting these demands by allegedly enabling rapid application development, reducing the dependency on expert programming skills, and streamlining the software development lifecycle.

The aim of this project is to conduct a comprehensive and comparative analysis between low-code mobile development, particularly using the OutSystems platform, and conventional mobile software development methods. By evaluating the strengths, weaknesses, and potential trade-offs of each approach, this study intends to contribute valuable insights into the practical implications of adopting low-code platforms in the software development lifecycle.

Through a literature review, hands-on development experiences, and insights from developer interviews, the study reveals that low-code platforms, specially OutSystems, can accelerate development and offer a gentler learning curve, making them accessible to non-developers. However, these platforms also face limitations in flexibility and scalability when dealing with highly specialized or complex applications. In contrast, traditional programming provides superior customization, flexibility, and scalability but requires more time and specialized knowledge. The findings suggest that the choice between these approaches should be based on project requirements, team expertise, and the application's complexity. By leveraging the strengths of both methods, organizations can optimize their software development processes and achieve their digital transformation goals more effectively.

Keywords: Development lifecycle; Low-code; Mobile; Software development; Strengths; Weaknesses.

Resumo

Nos últimos anos, o desenvolvimento de *software* tem mudado devido ao surgimento e aumento das plataformas de desenvolvimento *low-code*. Estas plataformas, como a OutSystems, prometem revolucionar o processo de desenvolvimento ao oferecer um meio mais acessível e eficiente de criar aplicações. A comunidade de desenvolvimento ainda está dividida sobre as implicações desta nova abordagem em comparação com as técnicas de desenvolvimento mais convencionais.

A crescente procura por soluções de *software* em várias indústrias aumentou a necessidade de metodologias de desenvolvimento ágeis e eficientes. O crescimento das plataformas *low-code* representa uma solução promissora para satisfazer estas necessidades, alegadamente permitindo um desenvolvimento rápido de aplicações, reduzindo a dependência de competências especializadas em programação e simplificando o ciclo de vida do desenvolvimento de *software*.

O objetivo deste estudo é realizar uma análise abrangente e comparativa entre o desenvolvimento *mobile* em *low-code*, particularmente utilizando a plataforma OutSystems, e os métodos convencionais de desenvolvimento de *software mobile*. Ao avaliar os pontos fortes, pontos fracos e os possíveis compromissos de cada abordagem, este estudo pretende contribuir com *insights* valiosos sobre as implicações práticas da adoção de plataformas *low-code* no ciclo de vida do desenvolvimento de *software*.

Através de uma revisão de literatura, protótipos de desenvolvimento e entrevistas com programadores, o estudo revela que as plataformas low-code, particularmente a OutSystems podem acelerar o desenvolvimento, oferecendo uma curva de aprendizagem mais suave, tornando-as acessíveis a não-programadores. No entanto, enfrentam limitações em termos de flexibilidade e escalabilidade quando aplicadas a áreas altamente especializadas ou complexas. Por outro lado, a programação tradicional demonstra ser mais personalizável, flexível e escalável, apesar de requerer mais tempo e conhecimento especializado. Este estudo indica que a escolha entre estas abordagens deve basear-se nos requisitos do projeto, na experiência da equipa e na complexidade da aplicação. Ao aproveitar os pontos fortes de ambos os métodos, as organizações podem otimizar os seus processos de desenvolvimento e atingir os seus objetivos de transformação digital de forma mais eficaz.

Palavras-chave: Ciclo de vida do desenvolvimento; Desenvolvimento de *software*; Low-code; *Mobile*; Pontos fortes; Pontos fracos.

Contents

List of Figures

List of Tables

Glossary

API - Application Programming Interface

BPT - Business Process Technology

DAO - Data Access Object

DSML - Domain-Specific Modeling Language

IDE - Integrated Development Environment

JSON - JavaScript Object Notation

LCDP - Low-Code Development Platform

POC - Proof of Concept

REST - Representational State Transfer

SOAP - Simple Object Access Protocol

SQL - Structured Query Language

UI - User Interface

WB - Web Block

1. Introduction

The rapid pace of technological advancement and the increasing demand for efficient software solutions have driven significant changes in the field of software development. Traditional programming methods, which rely heavily on detailed coding and platform-specific languages, have been the foundation of software development for decades. However, the advent of low-code development platforms presents a compelling alternative that promises to simplify and accelerate the development process. This research aims to explore and compare these two approaches, focusing specifically on the low-code platform OutSystems and traditional mobile programming methods.

The primary objective of this study is to evaluate the differences in mobile software development between OutSystems and traditional methods, examining key factors such as development speed, learning curve, flexibility, customization, maintainability, and scalability. As businesses increasingly seek to innovate and adapt quickly to market changes, understanding the practical implications of using low-code platforms versus traditional programming is crucial for making informed decisions about development strategies.

This study holds importance as it addresses an issue facing modern software development: the need for rapid, scalable, and efficient solutions in an increasingly competitive and digitally driven world. With businesses under constant pressure to innovate and deliver new software products and services faster, the choice between traditional programming methods and emerging low-code platforms has become a strategic decision. Understanding the differences between these approaches is crucial for organizations that are exploring digital transformation, as the right development methodology can substantially impact time-to-market, resource allocation, and overall business. This research provides valuable insights that can help companies make informed decisions about their development strategies and help developers choose a career path.

OutSystems, a low-code development platform, enables rapid application development through a visual interface and pre-built components, minimizing the need for extensive coding. This approach is particularly appealing for organizations looking to expedite their digital transformation efforts and reduce dependency on specialized programming skills. On the other hand, traditional programming methods, utilizing languages like Java or Swift, offer greater control and customization but require a deeper understanding of coding and longer development times.

The motivation for this study originates from the increasing adoption of low-code platforms like OutSystems by businesses seeking to accelerate software development without sacrificing quality. Despite their growing popularity, there is still a gap in the understanding of how low-code platforms compare to traditional programming methods, especially in terms of flexibility, scalability, and long-term maintainability. The hands-on experience in this study, combined with qualitative interviews with experienced developers, allows for a thorough exploration of these factors, offering a clearer perspective than theoretical analysis alone. By examining real-world use cases, this study seeks to illuminate both the practical advantages and limitations of each approach, addressing the challenges that organizations face when selecting the right tools and methodologies for their projects.

The structure of this document is designed to explore the following research question: "What are the differences in software development between the low-code platform OutSystems and traditional methods of mobile programming?". It begins with an extensive literature review that dives into the theoretical foundations of low-code platforms and traditional programming methods, setting the stage for the comparative analysis. Following this, the document details the hands-on development experiences with both OutSystems and Kotlin, which serves as the basis for evaluating the practical implications of each development approach. The next section presents the methodology, explaining the research design and the qualitative interviews conducted with developers to gain deeper insights into their experiences. The findings from the interviews are then synthesized into a discussion that compares the two methodologies on factors like development speed, learning curve, flexibility, customization, maintainability, and scalability. The document concludes with an analysis of the study's findings, future possibilities and some limitations of this study.

2. Literature Review

2.1 Digital Transformation

We understand Digital Transformation as a set of emerging technologies that aim to create new business opportunities and enhance existing ones. Many authors debate this term, both in academia and business (Vial, 2019).

With the emergence of the internet and the development of communication technologies, greater connectivity between organizations was experienced, which was possible through various devices that combine the most varied functions. One of the variables for the success of this process refers to the mastery in implementing these technologies (Matt et al., 2015).

Throughout these transformations, it is necessary to determine which digital technologies are best suited for the company and what impact they will have on the global structure. Therefore, adopting specific strategies for these transformational moments is recommended (Matt et al., 2015).

For businesses, implementing a digital transformation strategy is an option since it streamlines the process and reduces barriers. For this to take place, there must be a designated individual within the company who will oversee the course of reforms (Matt et al., 2015).

2.2 Low-Code Development Platform(s)

Digital Transformation is an inevitable topic for any company aiming to grow in the years to come. Organizations must provide resources that accelerate business processes and boost overall productivity in order to execute the digital transformation process as smoothly as possible. It is in this context that Low-Code development platforms emerge (Richardson & Rymer, 2014).

According to Richardson and Rymer (2014), it was in a report published in 2014 by two Forrester analysts that the term Low-Code first appeared. They also mentioned the use of programs where the amount of manual code development is minimized due to the drag-and-drop visual interface.

The acronym for "Low-Code Development Platform(s)" is LCDP. Stated differently, LCDP is software that offers an environment in which applications can be created with minimal programming requirements (Rymer, 2017). Low-Code platforms are built on visual interfaces and use a model-driven design, where the user creates objects and then adds functions to them using a predefined logical framework. The models they offer have code behind them that makes visual programming possible. With this kind of development, the user can simply see a list of options, and the software will automatically write the item's code when it is generated (Rymer, 2017).

Rymer (2017) also mentions that programming skills are not necessary for these platforms, as they can provide applications that are 100% functional, excluding the addition of exclusive functionality that the basic software does not offer. Tools for low-code development are becoming more and more crucial as businesses go digital. These are a more appealing option for various management levels since they enable programs to be developed more quickly and easily while also satisfying the various and ever-changing needs of consumers.

2.3 OutSystems Overview

OutSystems is a platform that enables visual, model-driven development of applications that can run in the cloud or on local infrastructures (Sahay et al., 2020). Furthermore, it allows integration with AI and DevOps, as well as the extension of components developed in traditional code.

This platform features the OutSystems UI framework that allows the rapid development of graphical interfaces for a web or mobile application from an existing template or from an empty page. These interfaces are created by dragging and dropping components to the screens (Alexander, 2021).

At OutSystems, application development takes place in Service Studio - the platform's visual, low-code development environment (Alexander, 2021). However, it is possible to create graphical interface prototypes or create workflows using Experience Builder or Workflow Builder, respectively (OutSystems, 2023). For integration with external services, the Integration Studio tool is used, which allows the management of extensions that correspond to the set of structures, entities and actions (Alexander, 2021). In addition, OutSystems supports the solution development lifecycle, through (OutSystems, 2023):

- deployment of all solution components and components for one-click integration.

- automatic monitoring of applications with the presentation of dashboards that make it possible to identify critical issues or problems through analysis.
- configuration management of running applications, simplifying the deployment of applications in production and ensuring continuous delivery. This platform has been increasingly used to automate business processes, however, according to Vincent (2019) the platform presents some challenges in modeling more complex decisions, which allows us to assess that it is not yet competitive in this context. Another challenging aspect of the platform is related to strategy, since Vincent et al. (2020) state that OutSystems favors programmers, rather than citizen developers, and competing platforms tend to encourage both profiles.

2.3.1 OutSystems Service Studio

Service Studio, which can be seen in Figure 1, is the development environment where mobile and web apps can be developed. The picture illustrates how similar it is to a traditional Integrated Development Environment (IDE). It also shows a sidebar with multiple pre-built widgets that can be utilized to build a mobile screen or a webpage. Furthermore, it can be observed in Figure 1's upper-right corner that four tabs are present: Processes, Interface, Logic, and Data (OutSystems Documentation, 2023).

2.3.2 OutSystems Interface

It is possible to design new screens and Web Blocks (WB) by starting from the interface. For example, a web screen has been defined in Figure 1 utilizing the WB Menu (the top menu that is shared by all web pages). Additionally, WBs can be used inside other WBs in addition to web screens; however, a WB cannot reference itself because doing so would lead to an infinite number of calls. The platform is prepared against that. Another crucial thing to remember about WBs is that they have their own scope and are unable to implicitly access the scope of the screen or WB they are in. Similarly, the parent screen or WB containing another WB is unable to directly access its child variables. Events, which are an asynchronous way of sending information to the parent and passing its variables. Using WBs offers the benefit of maintainability, as they only need to be created or updated once if changes are required across multiple screens utilizing them.

Another benefit is the encapsulation and abstraction of logic from a screen (OutSystems Documentation, 2023).

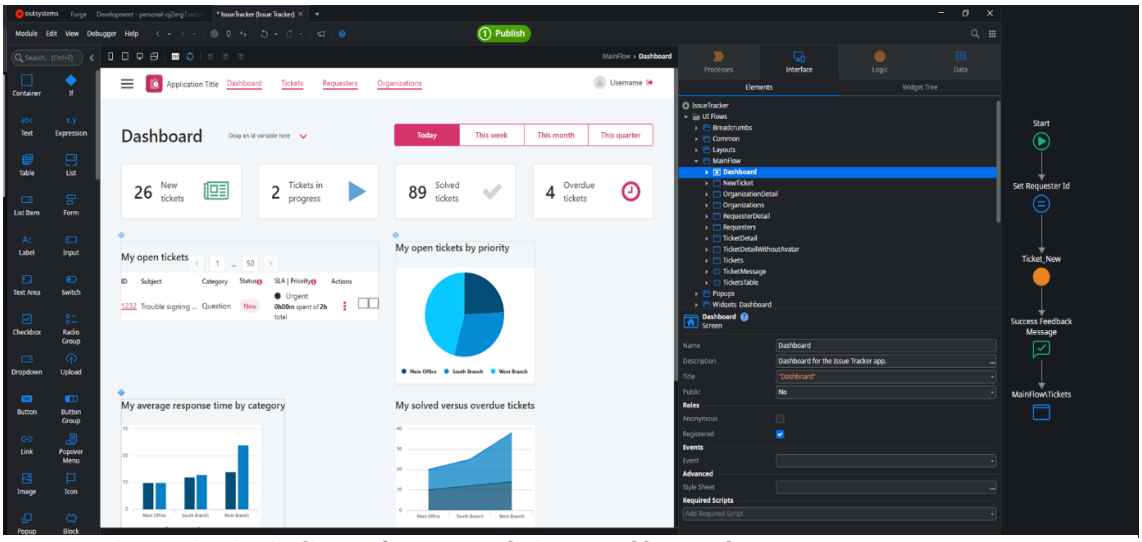

Figure 1 Service Studio interface: screen designing and logic code.

2.3.3 OutSystems development models

OutSystems provides three application development models: Traditional Web, Reactive and Mobile. Each is designed to meet specific development needs and user experiences. Traditional Web is designed for server-side rendering, it is ideal for applications that require complex server-side logic and can tolerate full page reloads, making it suitable for many enterprise-grade applications. Reactive, offers a client-side execution model ideal for creating responsive and interactive web applications. It leverages asynchronous data fetching and partial page updates for a smoother user experience on web applications, reducing server load and improving responsiveness. OutSystems Mobile apps extend the platform's capabilities into mobile application development, allowing for the creation of native mobile apps with offline capabilities, access to device sensors and a mobile optimized UI, all while maintaining the benefits of the low-code approach. This model focuses on using the full potential of modern smartphones, providing the best possible experience for users on the go.

2.3.4 OutSystems Data

First of all, OutSystems allows the construction of database entities (tables) and their properties. Aggregates and SQL can both be used to retrieve entities on the screen. Aggregates are essentially visual query builders that allow developers to perform operations on data without needing to write complex SQL queries by hand. They are designed to simplify the process of data retrieval and manipulation. The platform optimizes Aggregates but other things like bulk operations and unions, for example, cannot be accomplished without SQL.

Other than that, building structures is another possibility. To put it simply, these are custom data types built up of attributes from both simple and complex data types. Even the datatype of another structure might be a part of a structure. The ability to transform data structures to JSON and vice versa using methods included in the OutSystems platform is one of its many advantages. Lastly, there's the option to create Resources, which are files kept in directories, and Site Properties, which are variables that are rarely changed (OutSystems Documentation, 2023).

2.3.5 OutSystems Logic

OutSystems (2023) states that actions are required in the OutSystems ecosystem when encapsulating logic that carries out the application's business rules. Figure 1 shows on the right side a part of the front end logic. Using an interface similar to a flowchart, OutSystems enables programming without the need for written code. When developing application logic, traditional web development and Reactive development differ in a few ways. To retrieve data from the database for later display on the standard web, it can be done using a preparation. Since the preparation is called automatically prior to the rendering, there is no need to call it. On the other hand, when a user interacts with the screen (by clicking on a button or link, for example), screen actions enable the website to execute server-side logic.

However, because Reactive fetches data from the database asynchronously, no preparation is required. Moreover, actions can now be performed client-side in addition to server-side. Actions on the client side (formerly called screen actions) can call actions on the server side, but not the other way around. Calling several server actions within a client action is also not recommended from a performance standpoint, since this would result in many server calls, which could slow down screen rendering and worsen user experience. Each Aggregate may also include an after-fetch event that allows for the execution of additional logic. Lastly, there are actions

known as data actions, which are typically used to run SQL queries and are executed asynchronously, just like the Aggregates. Reactive also enables data to be fetched from the database on demand, at any time, or when the user interacts with the screen. Moreover, if the outcome of one aggregate is needed for another, OutSystems optimizes applications by running Aggregates in a particular order.

Additionally, representational state transfer (REST) application programming interface (API) and simple object access protocol (SOAP) web services can be exposed or consumed and it also allows for roles to be managed. Roles can be set on screens and can decide which elements appear inside a page. A function named CheckRole is offered by OutSystems and returns true or false based on whether the user possesses the RoleName permission. Not to mention, you can make exceptions. It is possible to trigger exceptions in the code and develop handlers for them after creating them.

2.3.6 OutSystems Processes

Timers and Asynchronous Processes, sometimes referred to as Business Process Technology (BPT), are tools for performing asynchronous activities. BPTs enable the management of business processes according to an entity's life cycle. For example, in order to schedule the execution of a process to begin automatically when a new invoice is created in an application that manages invoices, or it can pause process execution prior to the invoice payment task and wait for the invoice to be updated as "cleared for payment" by the chief officer, BPTs are utilized for it. A trigger in an operation or the creation of an entity can initiate a process as well. Having said that, BPTs and threads are comparable. A process instance is established upon startup, and an activity instance is created each time a process activity is located.

It remains to say that BPTs are intended for tasks that require lengthy times for an event to happen. OutSystems advises using Light Weight BPTs for large-scale batch processing scenarios such as event brokers (OutSystems Documentation, 2023).

2.3.7 OutSystems Service Center

OutSystems Service Center, a web-based management console that monitors and manages OutSystems environments, applications, and infrastructure as seen in Figure 2. It offers strong tools and capabilities to operations teams, administrators, and developers to efficiently handle maintenance, monitoring, troubleshooting, and application deployment activities. In order to ensure the efficient running and optimization of OutSystems applications, Service Center is essential to supporting the application lifecycle from development to production (Pereira, 2023).

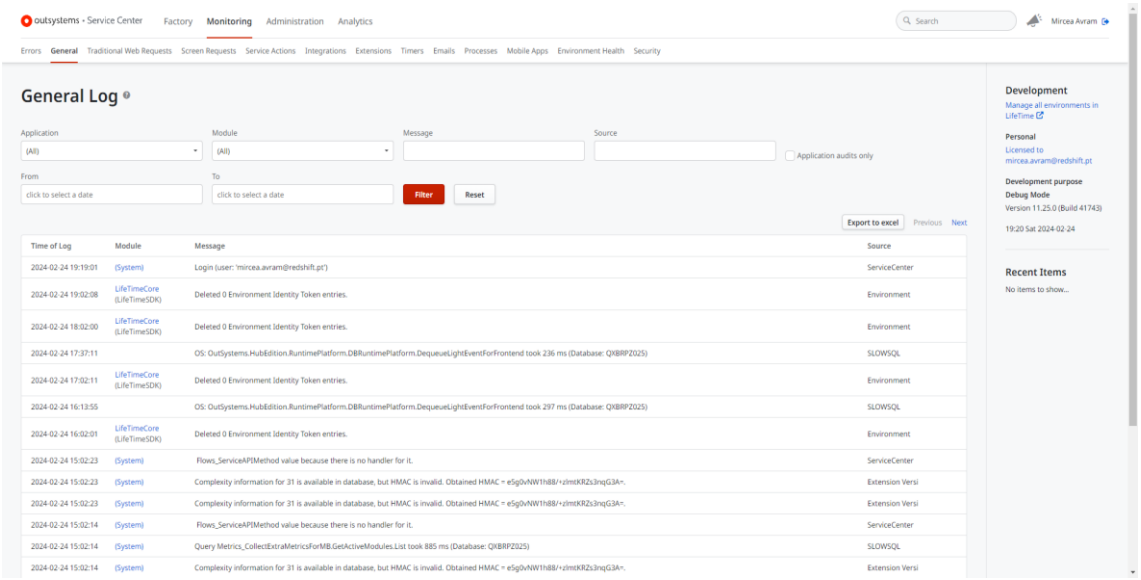

Figure 2 Service Center interface.

2.3.8 OutSystems LifeTime

OutSystems LifeTime is a robust management console that was created to streamline the whole lifecycle of applications created on the OutSystems platform. With its ability to provide a centralized view and control over many environments (usually development as shown in Figure 3, testing, staging, and production), it serves as an orchestration tool for the deployment, management, and monitoring of applications and environments. By improving teamwork between development, operations, and business teams, LifeTime guarantees that applications are deployed and managed effectively over their entire lifecycle (Sousa, 2022).

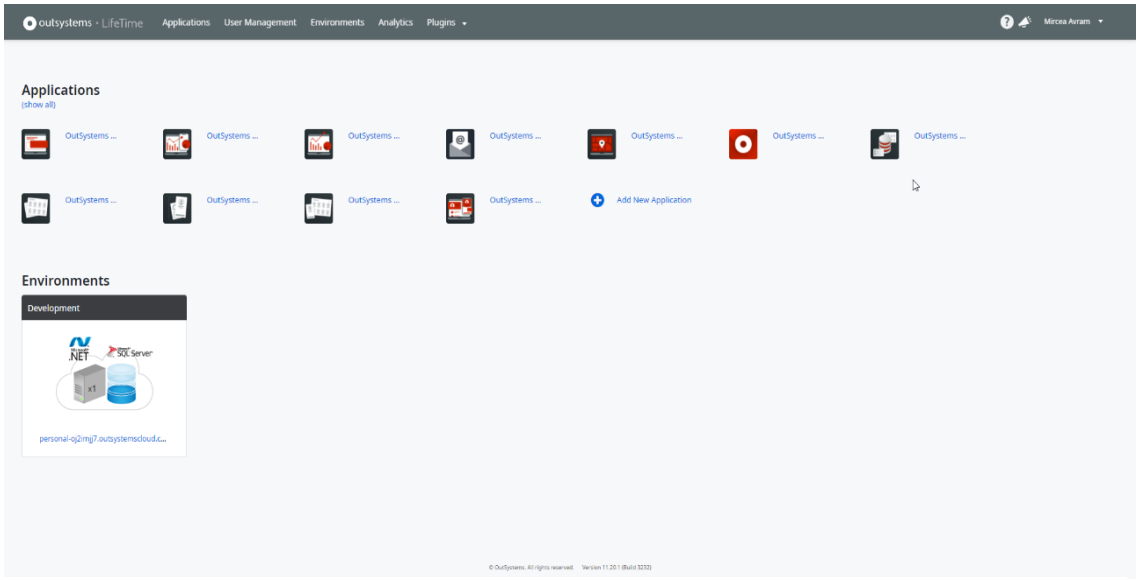

Figure 3 LifeTime interface.

2.3.9 OutSystems Integration Studio and Forge

OutSystems Integration Studio is a development environment expressly made for building, managing, and debugging integrations between OutSystems apps and external databases, systems, or custom code. Through integration with third-party systems, services, and legacy systems, developers can increase the functionality of OutSystems applications and facilitate a smooth transfer of data between various platforms and technologies (Duro, 2022). OutSystems offers an open-source repository called OutSystems Forge, which is a platform for community collaboration. Developers can exchange, find, and download reusable application modules, connectors, UI elements, samples, and templates from this marketplace. Using these components can speed up the creation of apps on the OutSystems platform by letting programmers take advantage of pre-built features and industry best practices without having to start from scratch (Habraken, 2017).

2.4 Mendix overview

Mendix is a low-code platform for developing applications that places an emphasis on visual modeling and the reduction of technical complexity. This allows a wide range of users to use it, such as business analysts, developers, and citizen developers. Mendix abstracts a significant amount of the coding process, unlike traditional programming languages with rigid syntax rules, enabling developers to create applications through a visual interface as opposed to writing code by hand. Therefore, the idea of "Mendix syntax" deviates from traditional programming languages and is more in line with visual modeling methodologies (Mendix, n.d.).

Mendix uses a visual development environment that allows developers to design applications by using model driven development techniques and drag and drop components. Developers use visual editors and wizards to define the application logic, data models, user interfaces, and integrations rather than writing code. Without requiring in-depth coding knowledge, users can focus on quickly developing applications and finding solutions to business needs (Mendix, n.d.).

Additionally, Mendix uses domain-specific modeling languages (DSMLs) to represent data modeling, workflow design, and user interface layout, among other aspects of application development. These DSMLs give users a standardized and user-friendly means of expressing application requirements and logic. These modeling languages are not strictly a "syntax" in the sense of the word, but they are the basis for creating Mendix applications and specifying their behavior (Mendix, n.d.).

Moreover, users can increase the functionality of their applications by integrating the variety of predefined modules, widgets, and microflows that Mendix offers. It encapsulates standard application patterns and best practices into these building blocks enabling users to assemble applications quickly and effectively. Mendix creates executable code based on the models and configurations defined by users, guaranteeing that applications are reliable, scalable, and maintainable (Mendix, n.d.).

2.5 Traditional mobile development

Over time, conventional approaches to mobile programming have gone through substantial changes in response to the growing complexity of mobile devices, operating systems, and user demands. Fundamentally, traditional mobile development uses native programming languages specific to each mobile platform. For Android it is used Java or Kotlin, and for iOS, Objective-C or Swift (Charland & Leroux, 2011). These methods prioritize performance, having access to all of the device's functions, and a high level of customization. Nevertheless, they raise the complexity and expense of development since they demand specific knowledge from engineers and necessitate maintaining distinct codebases for every platform.

There have been several significant turning points in the development of conventional mobile programming techniques. At first, smartphone apps were simple, limited to basic games or limited productivity tools. Rich media, real-time data synchronization, complex interactions, and other features were added to mobile applications as smartphones became more powerful (Rogers et al., 2009).

Rogers et al. (2009) further suggest that platform-specific development, performance optimization, and direct access to device capabilities are characteristics of traditional programming approaches. Native applications provide optimal performance and responsiveness by running directly on the operating system of the device. Developers that can access device features like the accelerometer, GPS, camera, and more by using platform-specific APIs offering a richer user experience.

Conventional approaches to mobile programming methods also face several challenges. Keeping up with distinct codebases for every platform demands additional time and resources, increasing the expense and duration of app development. Simultaneously updating apps on several platforms can be difficult and inconsistent, requiring more maintenance work. Furthermore, it is important for developers to possess proficiency in several programming languages and development environments, hence elevating the learning curve and restricting the flexibility of resources (Malavolta et al., 2015).

Continuing with this line of thought, Dalmasso et al. (2013) argue that all things considered, conventional approaches to mobile development, which provide excellent performance, device-specific optimizations, and access to a multitude of device characteristics, have been essential in the development of mobile applications. They do, however, also come with difficulties in the form of complexity, expense of development, and need for specialized

knowledge. In order to overcome these obstacles and accelerate the app development process, companies are looking into alternative strategies like cross-platform development and low-code platforms as the mobile landscape changes.

2.6 Android Studio overview

The official integrated development environment (IDE) for creating Android apps is called Android Studio, and it gives programmers all the tools they need to plan, create, and test Android apps. Since its release in 2013, Android Studio, which Google launched, has grown to be the go-to tool for Android developers because of its extensive feature set, smooth platform integration, and regular updates that accommodate the newest innovations and industry best practices. Android Studio is built on top of the IntelliJ IDEA platform and customized specifically for Android development. It offers a rich and intuitive user interface, powerful code editing and debugging tools, and seamless integration with other developer tools and services. With Android Studio, developers can create high-quality Android apps efficiently and effectively, whether they are beginners or experienced professionals (Google, 2024).

Google (2024), states the following as key features of Android Studio:

- **Intelligent Code Editor**: Android Studio includes a smart code editor with features like code completion, code refactoring, and syntax highlighting, making it easy for developers to write clean and maintainable code.
- **Visual Layout Editor**: The Layout Editor allows developers to design app layouts visually, with drag-and-drop support for UI components and real-time previews of the app's appearance on different screen sizes and orientations.
- **Performance Profiling Tools**: Android Studio offers powerful performance profiling tools that help developers identify performance bottlenecks, memory leaks, and other issues affecting app performance. These tools enable developers to optimize their apps for better responsiveness and efficiency.
- **Built-in Emulator**: Android Studio includes a built-in Android Emulator that allows developers to test their apps on virtual devices with different configurations and different Android versions. The emulator provides a fast and reliable way to test app behavior without the need for physical devices.
- **Integration with Google Services**: Android Studio integrates seamlessly with Google Play services, Firebase, and other Google APIs, allowing developers to

access a wide range of features such as cloud storage, authentication and analytics.

- **Version Control Integration**: Android Studio supports version control systems like Git, enabling collaborative development and easy management of project changes.

2.6.1 Android Studio Interface

Android Studio's interface is designed to enhance productivity and provide an intuitive development environment for users of all skill levels. The interface is highly customizable and offers a range of tools and panels that streamline the app development process. Developers are greeted with a Welcome Screen that provides easy access to essential actions such as creating new projects or opening existing ones. The Editor Window serves as the central workspace, equipped with a powerful code editor featuring syntax highlighting, code completion, and refactoring tools, facilitating efficient code writing and editing (Google, 2023).

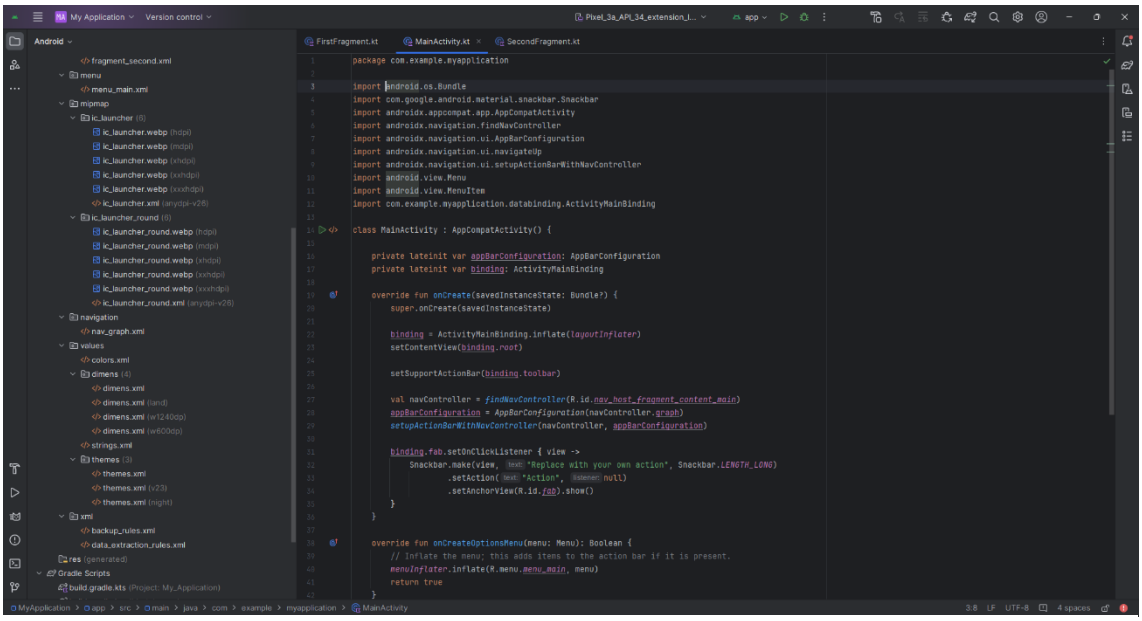

Figure 4 Android Studio Interface

The Project Explorer on the left side of Figure 4 provides a hierarchical view of the project structure, aiding developers in navigating through directories, files, and resources. This intuitive layout allows for quick and convenient access to specific files for editing or management. Android Studio offers additional functionalities, including debugging tools like Logcat, build variant

management, and integration with version control systems like Git, contributing to a comprehensive development environment (Google, 2023).

Moreover, Android Studio's design tools, such as the Layout Editor and Resource Manager, enable developers to create and preview user interfaces. The Layout Editor enables visual design of XML layouts with drag-and-drop functionality, while the Resource Manager facilitates the organization and management of app resources such as images. Combined with the Navigation Bar for quick access to common actions and the Status Bar for monitoring progress and errors, Android Studio's interface provides a cohesive and efficient environment for Android app development (Google, 2023).

2.6.2 Kotlin overview

Kotlin is a statically typed programming language created by JetBrains and due to its short syntax and Java compatibility has become very popular both inside and outside of the Android development community. Modern application development prefers Kotlin because of its syntax, which is optimized to decrease repetitive code and increase readability. Null safety is a defining characteristic of Kotlin, avoiding the frequent NullPointerExceptions that arise in Java programs. Explicit nullable and non-nullable types are used to do this, guaranteeing that nullability problems are discovered at compilation as opposed to runtime (Kotlin, n.d.).

Another significant aspect of Kotlin's syntax is its support for extension functions, allowing developers to extend the functionality of existing classes without inheriting from them. This feature makes it possible to add new methods to classes in a decoupled way, which results in code that is more understandable and compact. Additionally, Kotlin offers a simplified method that requires less repetitive code to manage data classes. By simply prefixing a class definition with the data keyword, Kotlin automatically generates essential functions like equals(), hashCode(), and toString() among others, making data handling more efficient and error-free (Kotlin, n.d.).

In addition, Kotlin places a strong emphasis on the aspects of functional programming, such as strong function support, immutable collections, and a wide range of higher-order functions. These characteristics enable expressive and compact code that is simple to develop and maintain, especially when combined with Kotlin's robust type inference. The features and syntax of the language are carefully crafted to enhance developer efficiency and program performance, which makes Kotlin a desirable choice for modern software development projects (Kotlin, n.d.).

Furthermore, Kotlin includes powerful features such as smart casts and string interpolation, which contributes to code clarity. Smart casts eliminate the need for explicit type casting in many cases by automatically narrowing the type of variables within conditional blocks. String interpolation, on the other hand, simplifies the process of constructing strings by allowing variables and expressions to be embedded directly within strings using the ${} syntax (Kotlin, n.d.).

2.7 Xcode overview

One of the most important tools for developers in the Apple ecosystem is Xcode, which is the integrated development environment (IDE) for creating apps for macOS, iOS, iPadOS, watchOS, and tvOS. Xcode simplifies the entire app development process, from the initial concept to the final deployment, by providing an extensive set of features and capabilities. The core of Xcode is its user-friendly interface, which offers developers a single workspace where they can easily manage project assets, write code, create user interfaces, and debug apps (Apple Inc., 2023).

Apple Inc. (2023) also claims that one of the main features is its code editor, which helps developers write clear and effective code by providing powerful capabilities like syntax highlighting, code completion, and intelligent code suggestions. Additionally, Xcode comes with an integrated visual design tool called Interface Builder that lets developers construct user interfaces using a drag-and-drop interface, facilitating quick prototyping and iteration.

Another standout feature of Xcode is its debugging tools, which give developers effective tools to find and fix problems in their apps. With the built-in debugger, developers may step through code, examine variables, and efficiently diagnose issues by getting real-time insights into how the application behaves. A smooth and responsive user experience is ensured by the performance analysis tools included in Xcode, which assist developers in optimizing their projects for speed, memory utilization, and energy efficiency (Apple Inc., 2023).

Xcode provides a smooth integration with Git and other version control systems, allowing team cooperation and effective code management in terms of project management. Additionally, developers can easily find answers to their questions and stay up to date with the newest technologies and best practices with Xcode's extensive documentation and support resources (Apple Inc., 2023).

Furthermore, Xcode is a reliable and adaptable integrated development environment that meets the requirements of developers working in the Apple ecosystem. With Xcode's robust feature set, user-friendly interface, and smooth connection with Apple's development ecosystem, developers can easily create high-caliber applications for all Apple platforms (Apple Inc., 2023).

2.8 Swift overview

Since its 2014 release by Apple, Swift has grown to be one of the most used programming languages for creating apps for iOS, macOS, watchOS, and tvOS. Swift offers a modern syntax that makes developing and maintaining code more understandable and efficient. It is well known for its simplicity, safety, and performance. Apple's dedication to giving developers a language that is both strong and approachable with frequent updates bringing new features and enhancements has driven the evolution of Swift (Apple Inc., 2024).

Swift includes features like optionals, which assist developers in handling nil values in a type-safe manner and lowers the possibility of runtime errors. Additionally, Swift's strong type system and type interface inference ensure that types are checked when compiling, catching errors early in the development process and improving code reliability (Apple Inc., 2024).

Furthermore, Swift's expressive and compact syntax allows programmers to build code that is easier to read and maintain. Developers may approach complicated concepts in a clear way by using features like type inference, closures, and generics. Swift also presents innovative programming ideas like protocol-oriented programming, which promotes the use of protocols and protocol extensions to create code that is reusable and composable (Apple Inc., 2024).

Moreover, one other main advantage of Swift is the ability to integrate with existing Objective-C codebases allowing developers to use already existing libraries and frameworks with ease. In terms of performance, Swift is better than Objective-C in areas like memory management and string handling. These performance enhancements, coupled with Swift's modern syntax and safety features, make it an ideal choice for building high-performance and reliable applications within the Apple ecosystem (Apple Inc., 2024).

2.9 Literature Review Summary

Digital Transformation is understood as the implementation of emerging technologies aimed at creating new business opportunities and enhancing existing ones. It requires organizations to carefully select digital technologies and implement them effectively, with a focus on reducing barriers and improving business processes (Vial, 2019; Matt et al., 2015). Low-code development platforms, like OutSystems, have emerged in this context as tools that enable rapid development through visual interfaces and model-driven designs (Richardson & Rymer, 2014). These platforms allow users to build applications with minimal programming by leveraging drag-and-drop functionalities, making them appealing for organizations seeking faster and more efficient development cycles (Rymer, 2017).

OutSystems stands out for enabling the visual development of applications that can run both on the cloud and local infrastructures. It includes tools such as Service Studio for visual development, Integration Studio for managing external services, and a UI framework that facilitates rapid interface design. OutSystems supports the entire solution lifecycle with built-in monitoring, deployment, and configuration management features. However, the platform faces some challenges in modeling complex decisions (Vincent, 2019; Vincent et al., 2020).

The review also covers traditional mobile development approaches, which involve using native programming languages such as Java, Kotlin, Swift, and Objective-C for specific platforms like Android and iOS (Charland & Leroux, 2011). These methods provide superior performance, access to device-specific features, and a high level of customization, but they also introduce complexity and higher costs due to the need to maintain separate codebases for each platform (Rogers et al., 2009; Malavolta et al., 2015). Native mobile apps have evolved from simple tools to sophisticated applications with complex interactions, driven by the increasing power of smartphones and the need for real-time data synchronization (Rogers et al., 2009).

In addition, this review touches on other tools and platforms like Mendix, another low-code platform that emphasizes visual modeling and simplifies the coding process, enabling business analysts and non developers to engage in application development (Mendix, n.d.). Meanwhile, Android Studio and Xcode are discussed as the primary integrated development environments (IDEs) for traditional mobile programming, offering extensive features like visual layout editors, intelligent code editors, performance profiling tools, and integration with platform-specific services (Google, 2024; Apple Inc., 2023).

Finally, Swift and Kotlin are identified as modern programming languages that optimize coding practices and enhance developer productivity. Swift offers a simplified syntax, safety

features like optionals, and seamless integration with legacy Objective-C codebases, making it a powerful language for Apple's ecosystem (Apple Inc., 2024). Kotlin, meanwhile, has gained popularity for Android development due to its condensed syntax, null safety features, and support for functional programming possibilities (Kotlin, n.d.).

Following this literature review, the *hands-on experience* chapter will explore practical applications of both OutSystems and Kotlin, providing some insights into their performance, usability, and development processes.

3. Hands-on experience

To explore and address the research question concerning the distinctions between the conventional approaches of mobile programming and the low-code platform OutSystems, a Proof of Concept (POC) application for tracking expenses was developed. To compare and gain firsthand experience with the advantages and disadvantages of each development approach, this POC will function as a practical investigation of the two approaches. The expense tracker app will represent essential functionalities commonly found in such applications, including recording expenses, categorizing transactions, generating reports, and setting budget limits. Additionally, to ensure clarity, this study will outline user stories detailing the basic functionalities expected from the app. These user stories will provide a clear roadmap for development, guiding the implementation of features and ensuring that the POC effectively demonstrates the capabilities of both OutSystems and traditional coding methods in delivering a functional mobile application. Through this hands-on approach, the study aims to gain valuable insights into the development process, usability, and performance of both approaches, ultimately contributing to a comprehensive analysis of their comparative advantages and challenges.

3.1 The requirements

1. **Manual Expense Logging:** Users should have the ability to manually input their expenses into the app. This includes entering the amount spent, the date of the transaction, and selecting the appropriate expense category. This feature allows users to record their spending as they go about their day-to-day activities.
2. **Expense Categorization:** After entering an expense, users should be able to categorize it into predefined categories (e.g., groceries, transportation, utilities). Categorization helps users understand their spending patterns and identify areas where they can potentially cut costs.
3. **Budget Setting:** Users should be able to set budgets for different expense categories. This feature enables users to establish spending limits and stay within their financial goals.
4. **Expense Summary:** Users should have access to a summary of their expenses for a selected period (e.g., daily, weekly, monthly). This summary provides users with a quick overview of their spending habits and allows them to track their progress over time.

5. **Visual Representations:** The app should provide visual representations such as charts or graphs to illustrate spending patterns. Visualizations make it easier for users to understand vast amounts of financial information and identify trends.

6. **Budget Notifications:** Users should receive notifications when they approach or exceed their budget limits for specific categories. These notifications serve as reminders to stay on track with their spending goals and avoid overspending.

7. **Expense Editing/Deletion:** Users should be able to edit or delete expense entries if they make a mistake or need to make changes. This feature provides flexibility and ensures that users can maintain accurate records of their expenses.

8. **User-Friendly Interface:** The app should have a simple and intuitive user interface to enhance the user experience. A clean and easy-to-navigate interface ensures that users can quickly access the app's features and accomplish their tasks.

3.2 Visual guidance

To act as a guide for the Proof of Concept (POC) expense tracker application, simple visual mockups were created. These mockups offer a visual idea of the user interface for the app, highlighting important screens and features like the budget overview, category management, transaction history, and expense entry form as shown in Figure 5 and 6.

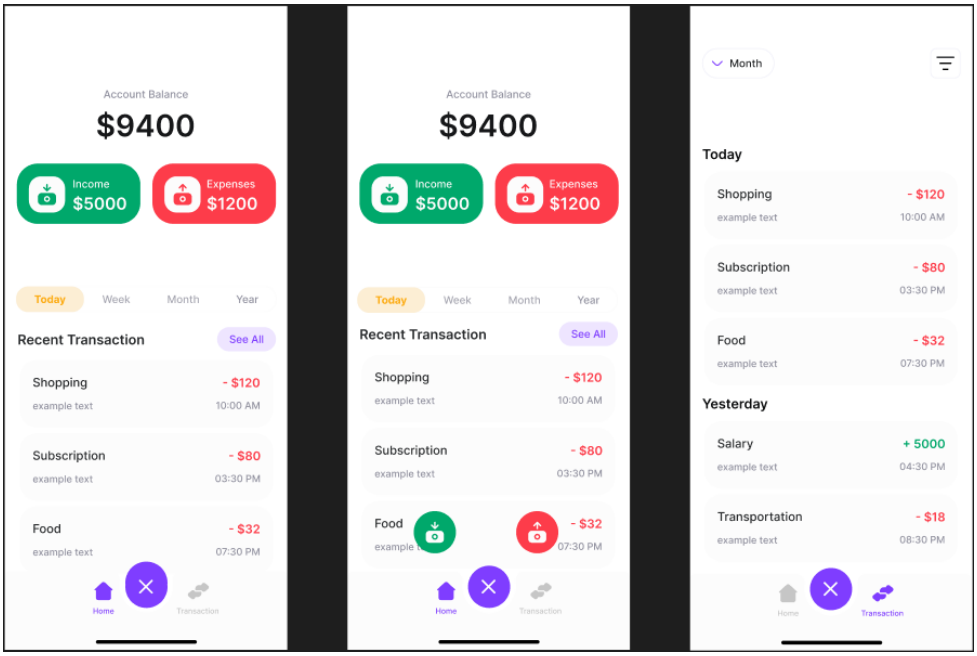

Figure 5 Mockup Main Screen & Transactions

The final POC implementation may vary between the version developed in OutSystems and the one in Kotlin, even though the mockups provide a starting point for the design and development process. The final appearance and feel of the applications may depend on several factors, including technical considerations, development capabilities, and platform-specific design guidelines. Nonetheless, the mockups will serve as valuable guidance throughout the development process, helping to ensure that the core functionalities are accurately represented in both versions of the POC.

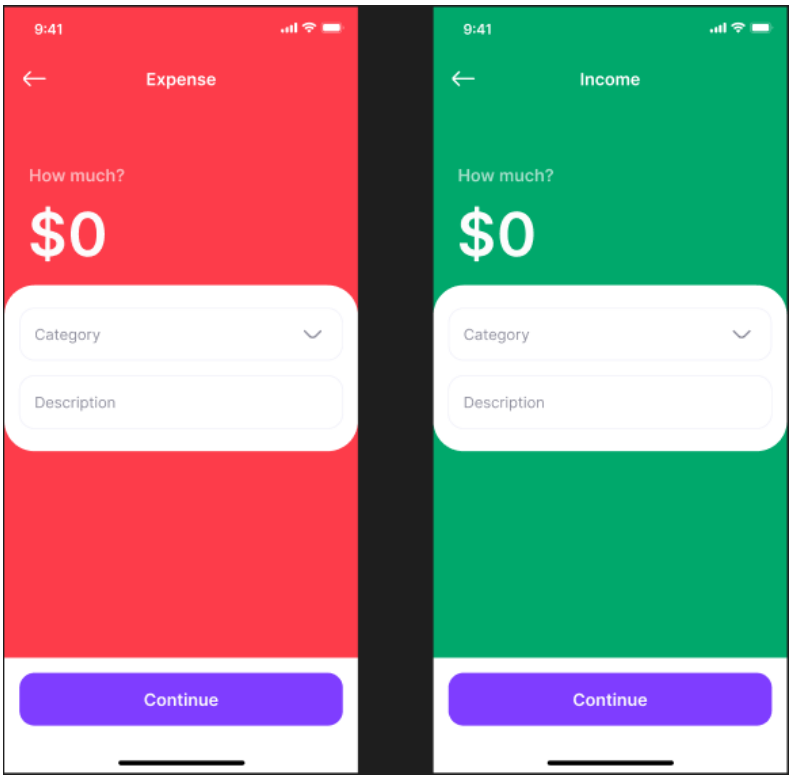

Figure 6 Mockup Expense & Income Form

3.3 Overview of the developed OutSystems POC

Home Screen

The home screen, Figure 7, provides a dashboard view of the user's financial summary. It displays an overview of the current month’s expenses, current month’s incomes and a list of the most recent expenses that can be filtered with the available onscreen timeframes. The user can quickly access key functionalities from the home screen, including adding a new expense, adding a new income and viewing transaction history.

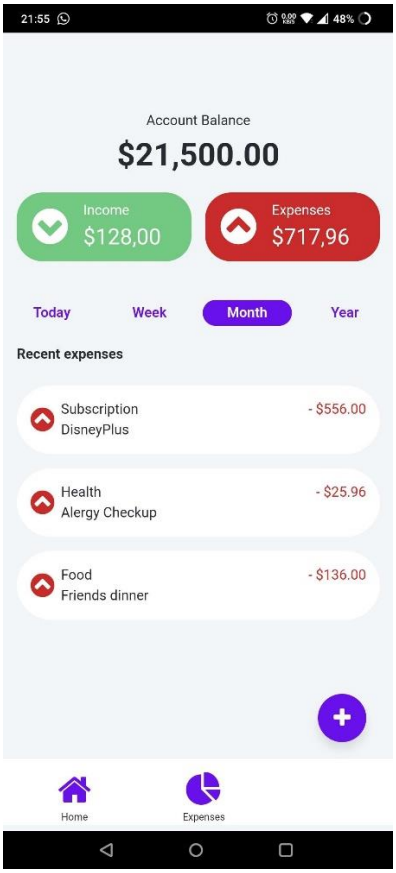

Figure 7 OutSystems Home Screen

Expense and Income Entry Screen

The expense and income entry form are designed to allow users to record their expenses and incomes quickly and easily. Users can enter the amount, select a category (such as food, transportation, entertainment, etc.), and add a brief description on the respective screens as seen in Figure 8. The same goes for the income form where users can enter the amount, select a category (such as salary, friends, family, etc.), and add a brief description. This form ensures that all necessary details are captured for each expense, enabling accurate tracking and categorization.

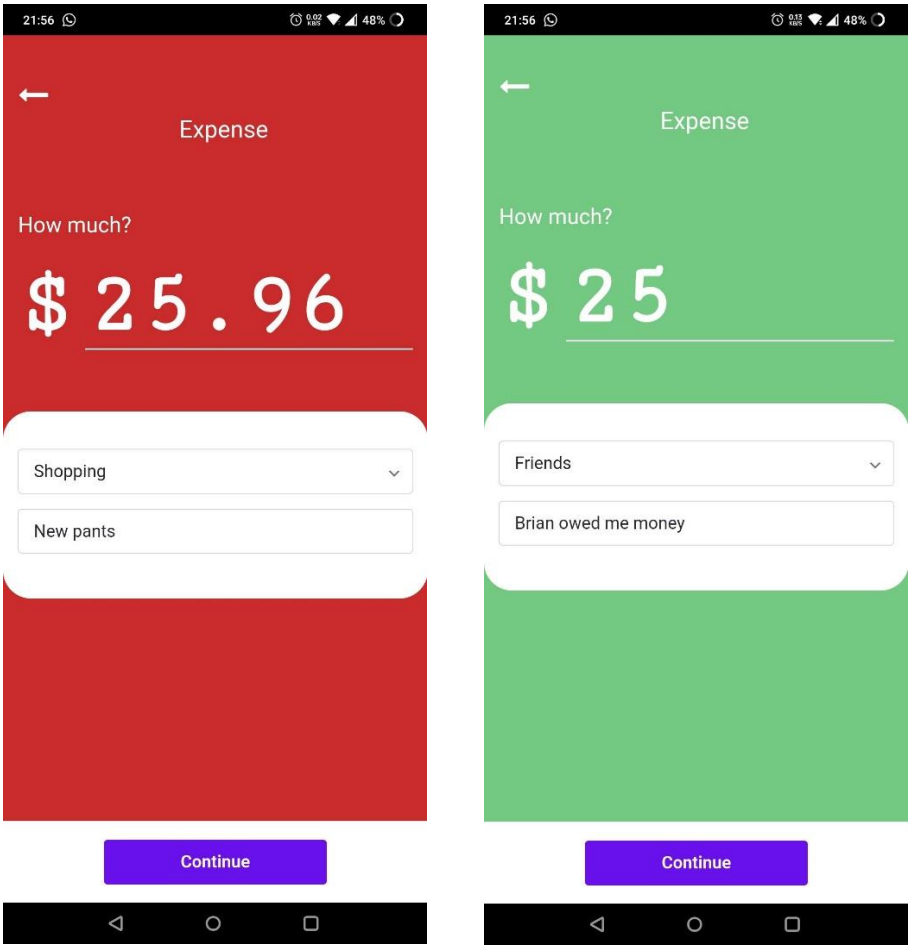

Figure 8 OutSystems Expense and Income Entry Screens

Expenses Screen

The transaction history screen, Figure 9, provides a detailed list of all recorded expenses. Users can view transactions in a chronological order that can be filtered with the available onscreen timeframes. Each entry displays the amount, category, date, and description, allowing users to review their expenses.

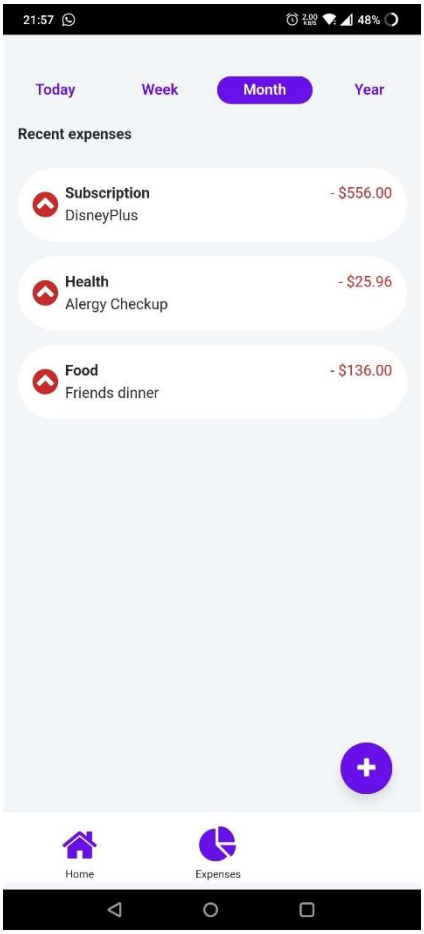

Figure 9 OutSystems Expenses Screen

3.4 OutSystems developing experience

It was developed a basic version of the Proof of Concept (POC) expense tracker application using the OutSystems low-code platform. This application serves as a practical demonstration of the capabilities and efficiencies of OutSystems in developing functional mobile applications. The primary goal of this POC is to compare the development process and outcomes with traditional mobile programming methods, specifically using Kotlin.

Developing the expense tracker application in OutSystems showcases the strengths and efficiencies of the low-code platform. The development of this project leveraged most of OutSystems' tools and features to streamline the development process and deliver a functional application as fast as possible.

The development started with setting up the OutSystems environment, which was straightforward and user-friendly. OutSystems provides a cloud-based development environment, eliminating the need for complex local installations. Once the environment was set up as shown in Figure 10, it was developed a new application project, which included configuring basic settings such as the app name, description, and primary template to be used.

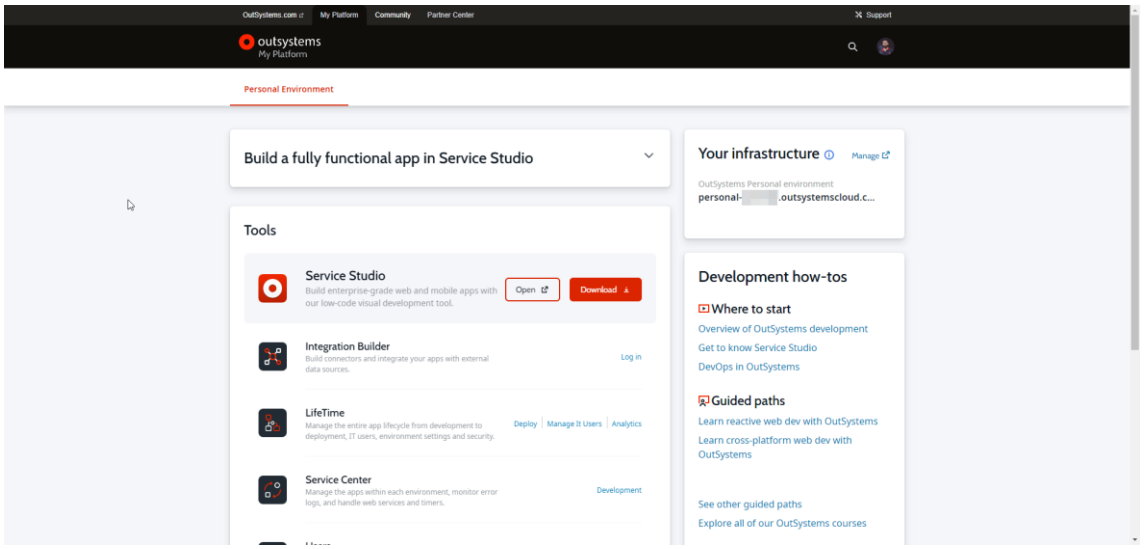

Figure 10 OutSystems environment setup

One advantage of OutSystems is its visual development. Using Interface Builder, the user interface was designed by dragging and dropping components onto the canvas. For the home screen were used, icons, expressions, button groups and list widgets to display the information on the screen.

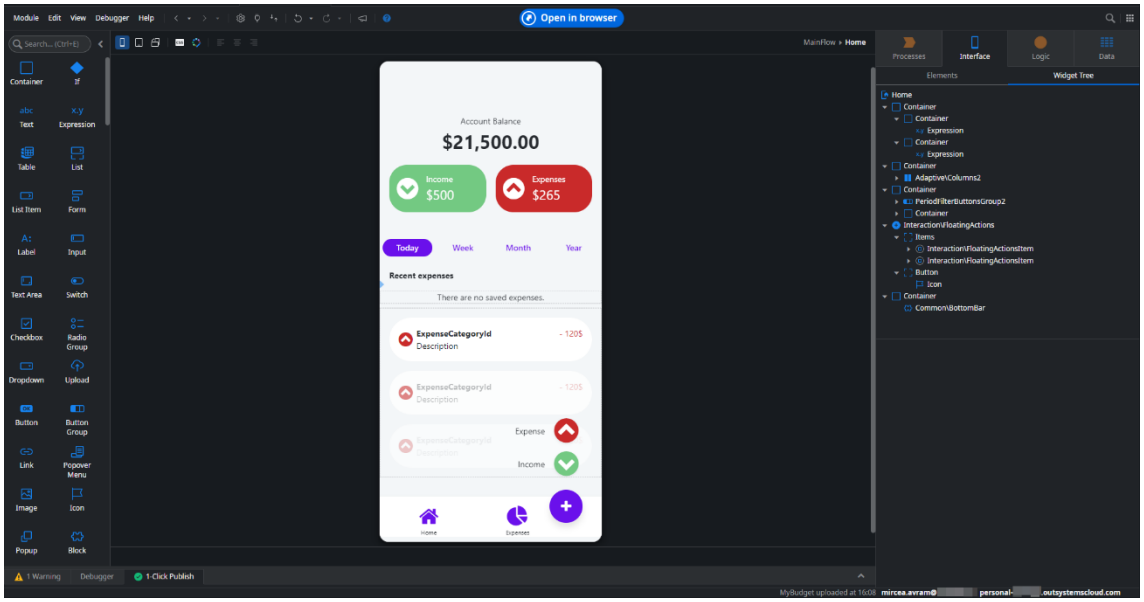

Figure 11 OutSystems Home Screen development interface

The expense and income entry form screen were designed using input widgets, allowing users to input data with ease. OutSystems' visual editor allowed to define input fields for the expense amount, a dropdown widget for category selection and description input. The ability to preview changes in real-time is beneficial, ensuring that the UI was user-friendly and visually appealing.

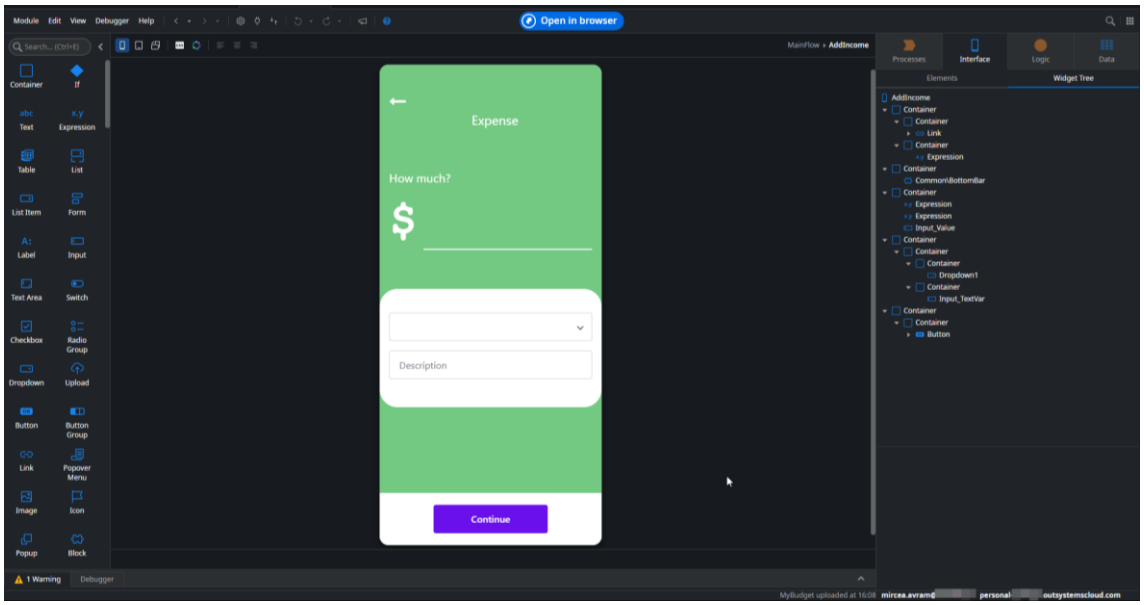

Figure 12 OutSystems Income Entry Screen development interface

OutSystems excels in reducing the complexity of traditional coding through visual programming. To implement business logic, it was used OutSystems' action flows, which are visual representations of logical workflows. For example, when a user submits a new expense, an

action flow validates the input data, updates the expenses history, and recalculates the budget overview.

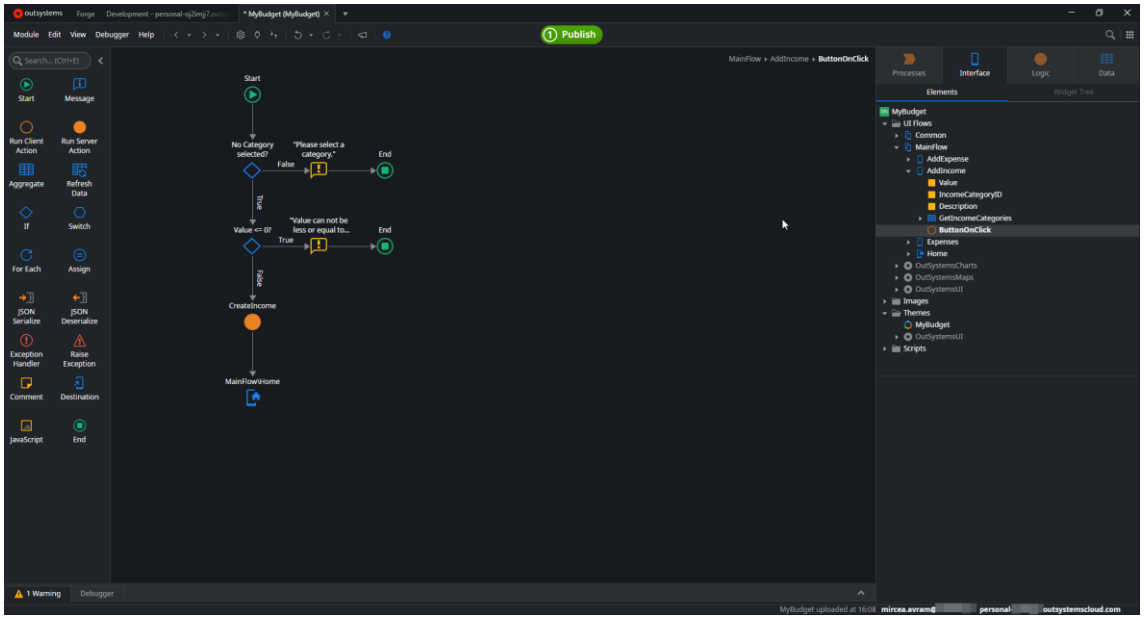

Figure 13 OutSystems Submit logic with data validation

The logic for filtering the expenses list by the selected timeframe was implemented using OutSystems' built-in functions and aggregate operations. This visual approach to programming significantly reduced the time needed to write and debug code, as it’s easy to see and modify the logic flow.

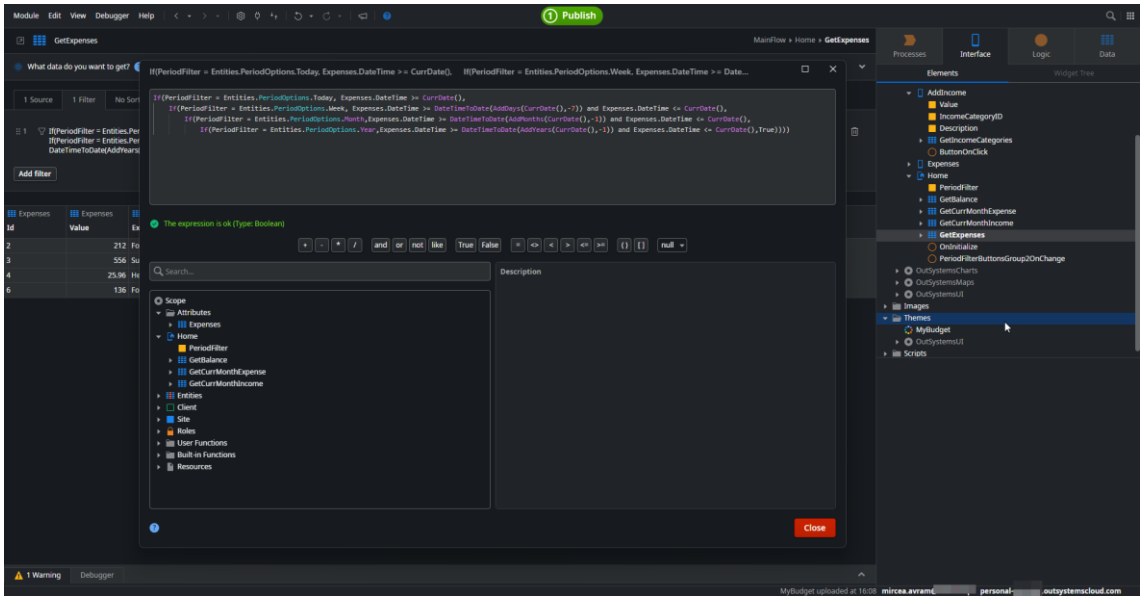

Figure 14 OutSystems aggregate filtering dates

OutSystems provides a data modeling tool that allows developers to define and manage data entities visually. Entities for expenses, incomes, and categories were created directly within the platform. The relationships between these entities were defined using simple attribute references, summing up the process of setting up the database schema.

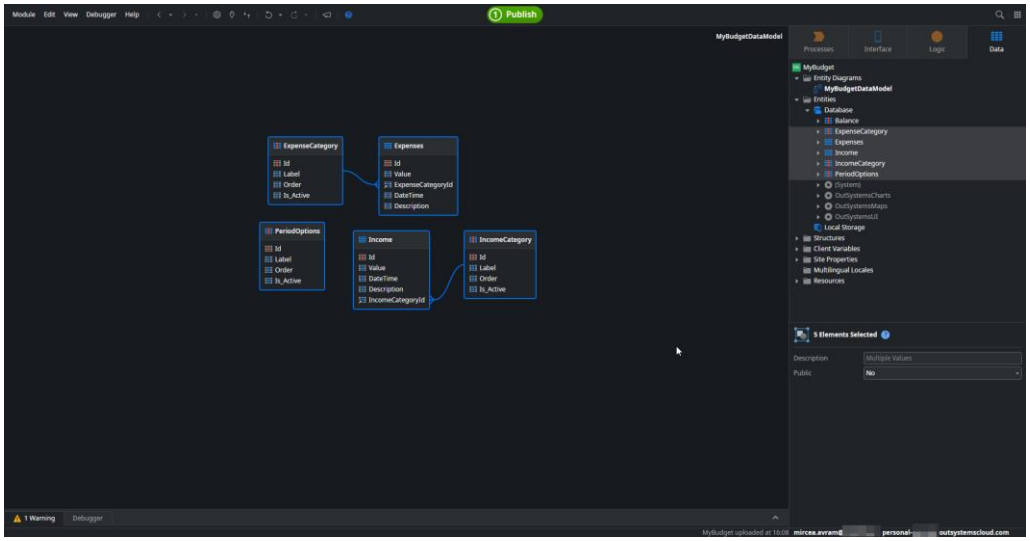

Figure 15 OutSystems database diagram

Testing and debugging were important parts of the development process. OutSystems offers built-in debugging tools that allow developers to step through action flows, inspect variable values, and diagnose issues in real-time. This ensures that the application logic was correct and that all functionalities worked as intended after going through some testing of the expense entry form, expense filtering, and budget calculations to ensure accuracy. The visual debugging tools made it easy to identify and fix any issues that appeared during testing.

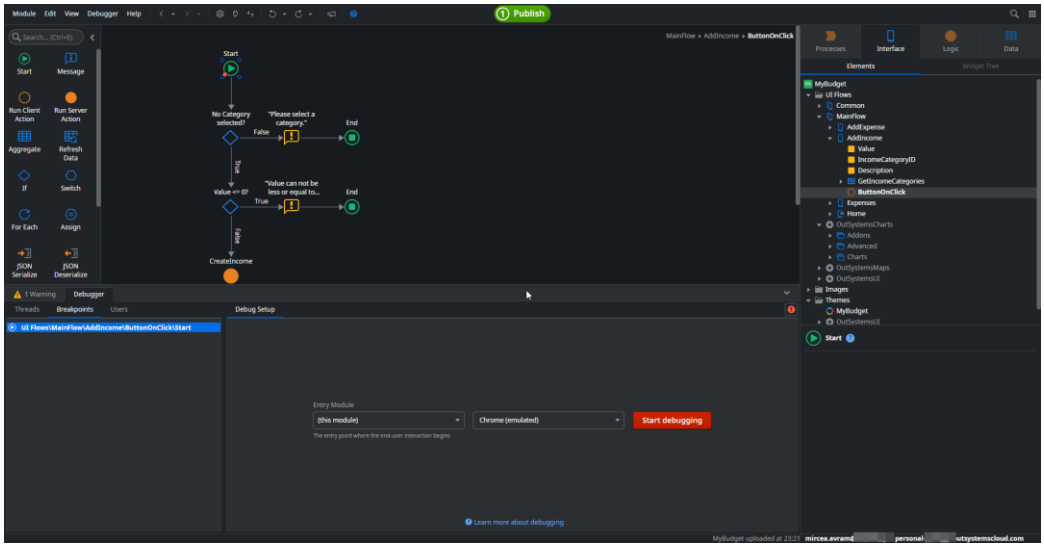

Figure 16 OutSystems breakpoint and debug options

3.5 Overview of the developed Kotlin POC

It was also developed a basic version of the Proof of Concept (POC) expense tracker application using Kotlin. This project serves to compare the differences in software development between traditional mobile programming methods and the low-code platform OutSystems. The primary aim of this POC is to provide a practical basis for evaluating the strengths and challenges of using Kotlin for mobile app development compared to OutSystems.

Home Screen

The home screen serves as the dashboard, providing users with an overview of their financial status as seen in Figure 17. It displays the total expenses, the budget, categorized expenditure, and the remaining balance. The home screen allows users to navigate to the add transaction screen.

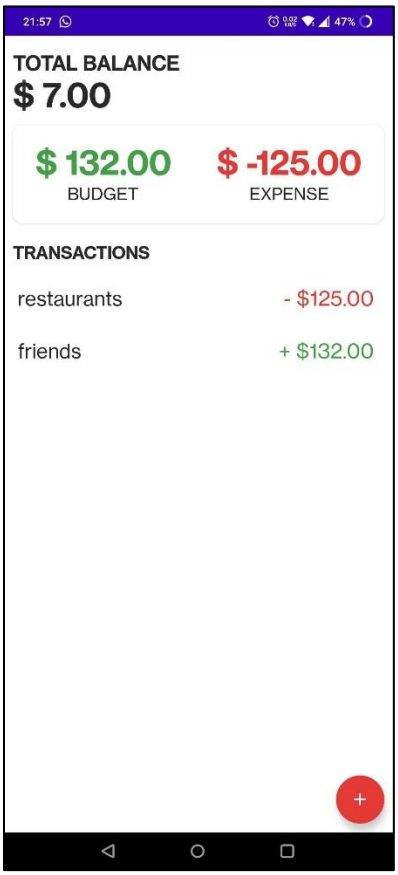

Figure 17 Kotlin POC Home Screen

Transaction Entry Screen

The transaction entry screen allows users to record their expenses and incomes. Users can input the amount, the category, and add a brief description. The form ensures that all necessary details are captured, providing accurate tracking and categorization of expenses.

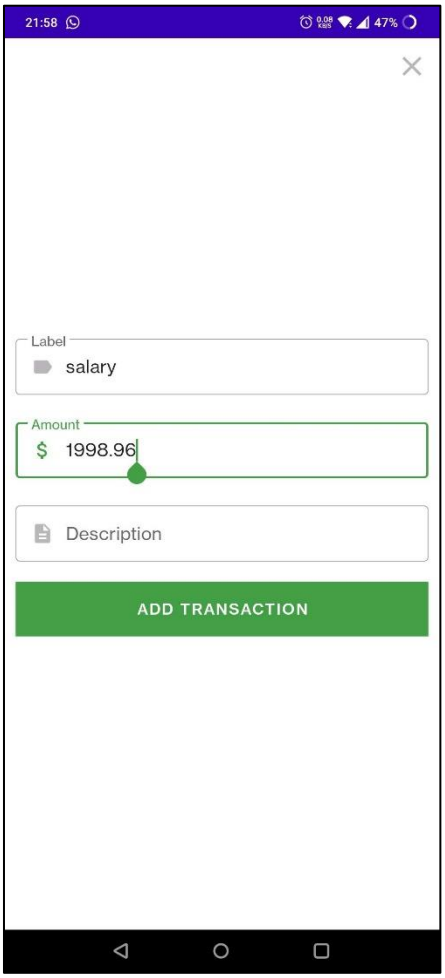

Figure 18 Kotlin Transaction Entry Screen

3.6 Kotlin developing experience

Developing the expense tracker application in Kotlin provided a hands-on experience with traditional mobile programming. There were some initial challenges in getting used with Kotlin' syntax and interface. However, while developing, some of the initial blockers were overcome and the development process became a bit more intuitive.

Development started with setting up the Android Studio IDE. Configuring the development environment involved setting up the project structure, dependencies, and necessary libraries as shown in Figure 19. Android Studio's robust support for Kotlin made the setup process straightforward, although there was a learning curve in understanding the best practices for project configuration and what libraries to use.

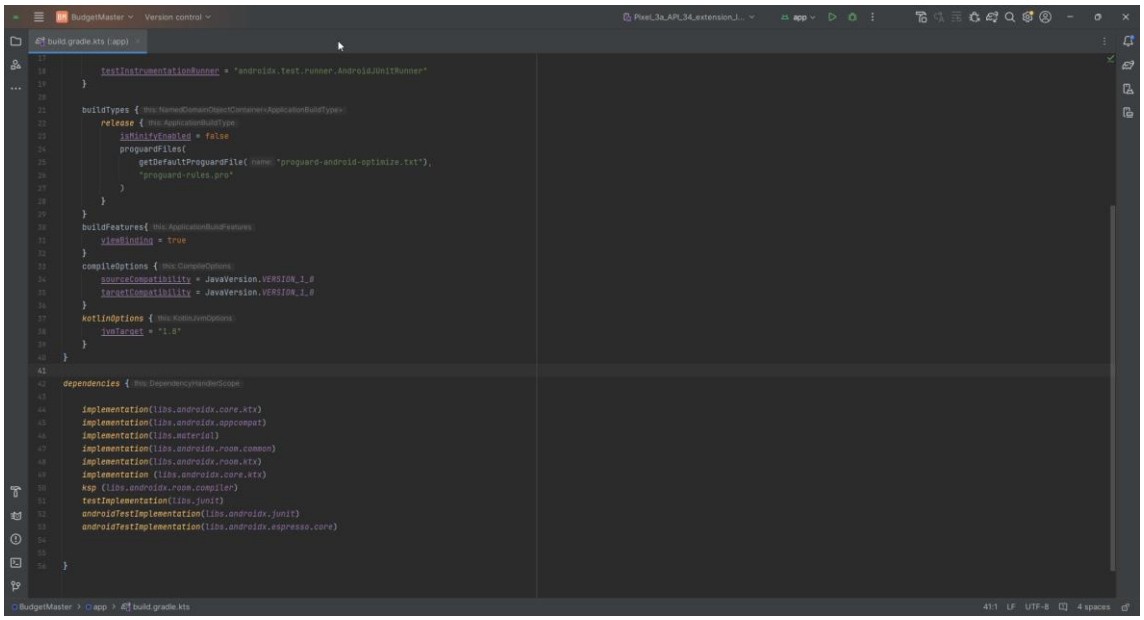

Figure 19 Kotlin configurations and dependencies file

Designing the user interface in Kotlin involved using XML for layout definitions and Kotlin code for activity and fragment management. The Android Studio Layout Editor was instrumental in visually designing the UI components, such as buttons, text fields, and lists. Integrating these components with Kotlin code required some time over forums and documentation to get a clear understanding of Android's activity lifecycle and view binding.

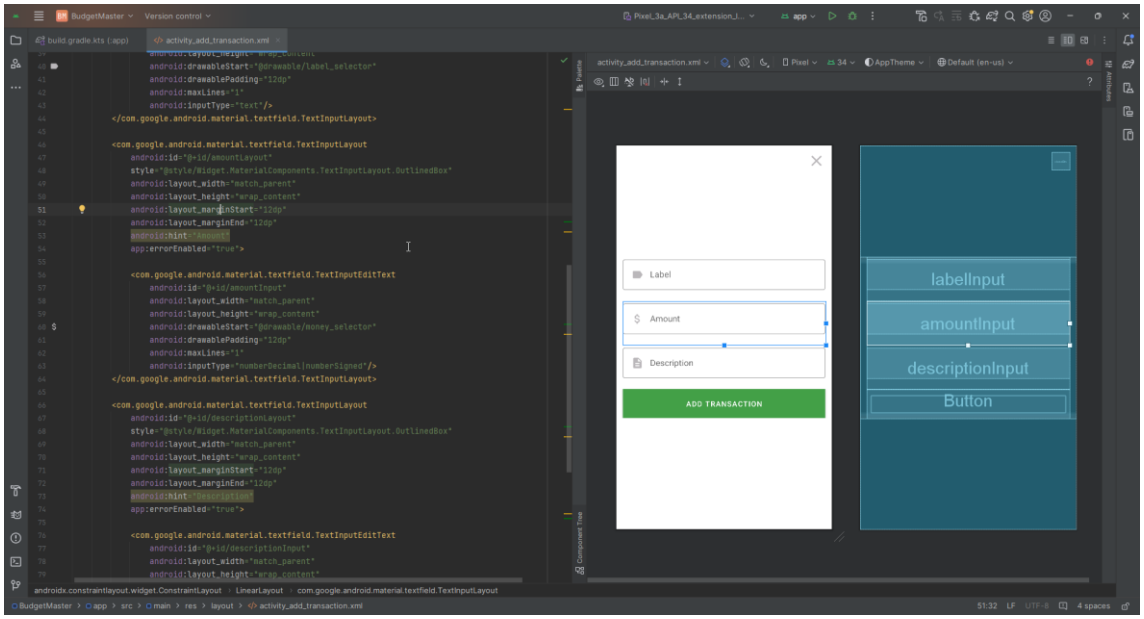

Figure 20 Kotlin screen design process

The business logic was implemented using Kotlin's modern syntax features, such as null safety and extension functions. The application's data model was defined and functions for managing expense entries and calculating budget were developed. Despite some initial challenges with Kotlin's syntax, deprecated libraries and compiling errors, the documentation and developers' forums helped understand and fix the known issues.

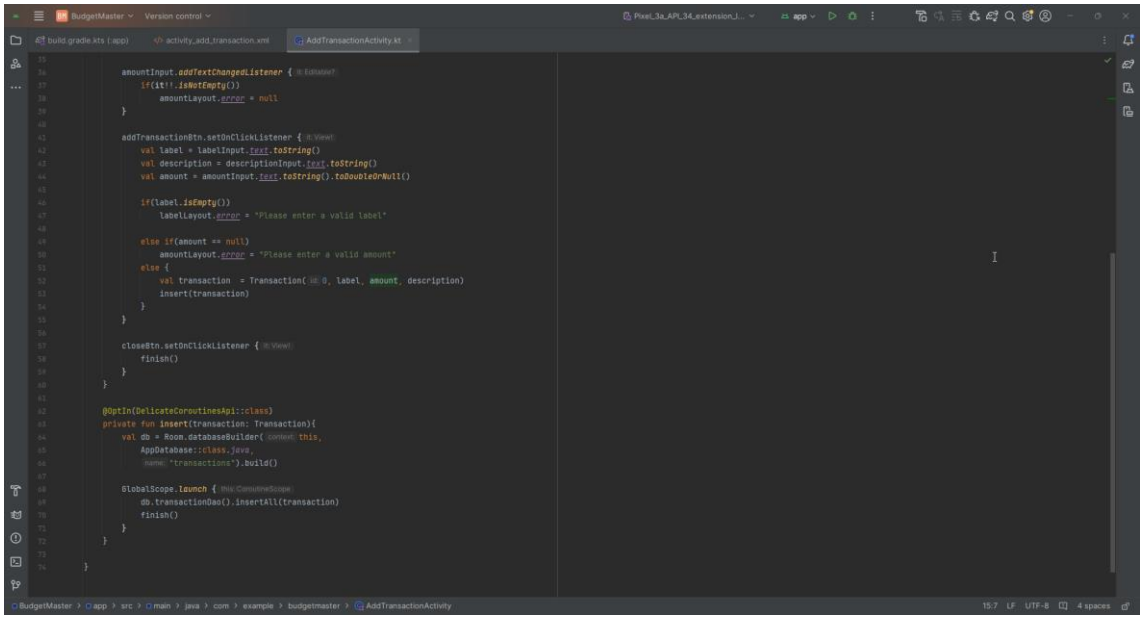

Figure 21 Kotlin code responsible for validating and registering a transaction

For data storage, it was implemented a Data Access Object (DAO) pattern using local in-memory storage to manage the application's data. The DAO pattern provides an abstract interface

for data operations, ensuring that data access logic is decoupled from the business logic, which enhances code maintainability and testability. It was also implemented a DAO interface that defines the operations for adding, updating, deleting, and querying transaction entries. By using local data structures such as lists to store the data in memory, the implementation remained lightweight and straightforward.

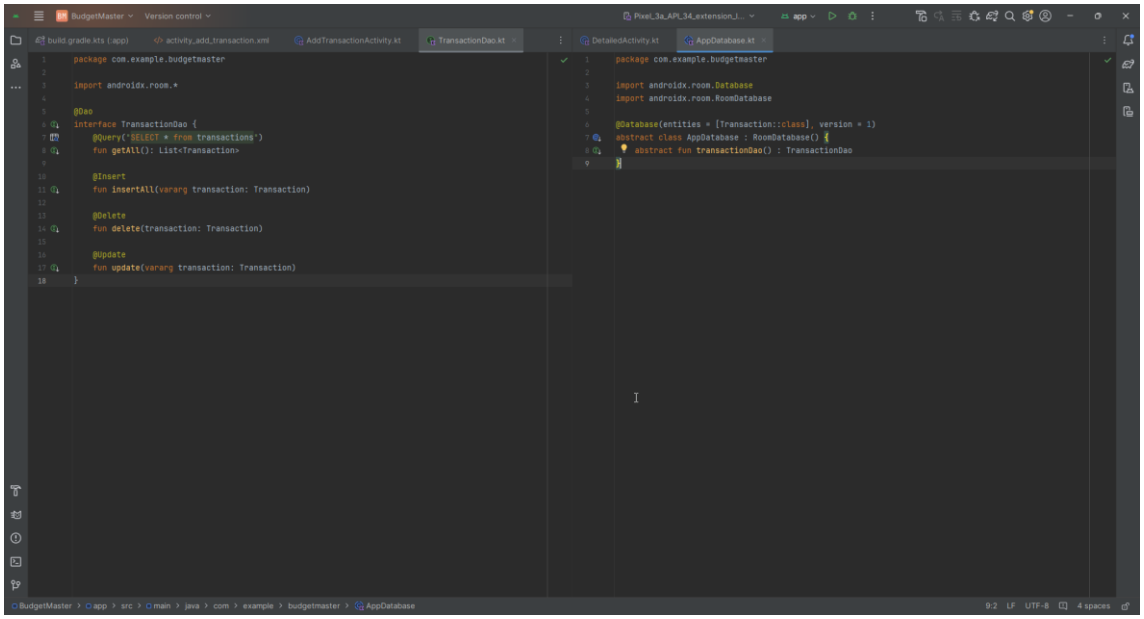

Figure 22 Kotlin DAO storage

Testing and debugging were critical aspects of the development process. Android Studio's built-in tools allowed for effective debugging, allowing to step through code, inspect variables, and resolve issues. Logcat command-line tool was a life saver when talking about errors. The errors are not the most user-friendly, but most of the time they had key information about what was wrong with the written code.

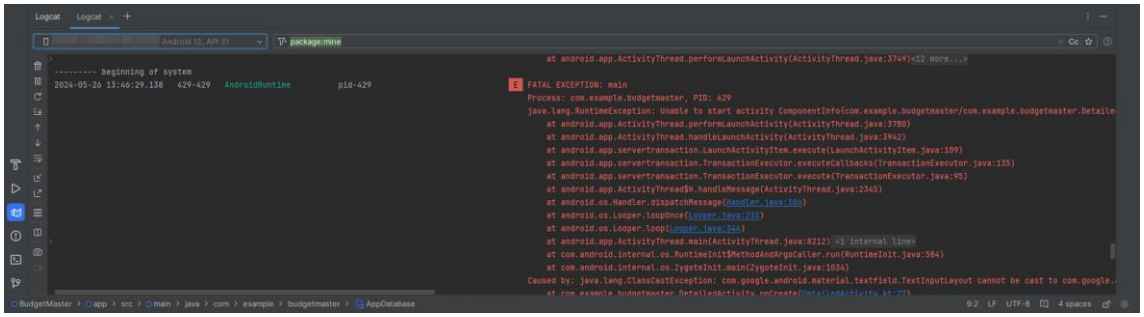

Figure 23 Kotlin Logcat

3.7 Hands-on experience Summary

To address the research question of this study, a proof of concept (POC) expense tracker application was developed using both OutSystems and Kotlin. This POC served as a practical investigation of each approach strengths and challenges, focusing on key functionalities such as expense logging, categorization and visual summaries. Additionally, user stories were outlined to guide development, ensuring that the application met essential functionality requirements.

The OutSystems POC was developed in the platform's cloud-based Service Studio, leveraging its visual interface and drag-and-drop components to rapidly design the user interface and implement business logic. The home screen and transaction entry forms were created using widgets, while action flows and aggregate operations simplified business logic development, such as validating expenses and recalculating budgets. OutSystems built-in debugging and data modeling tools enhanced the efficiency of development, allowing for real-time testing and quick issue resolution. The experience demonstrated OutSystems' strengths in reducing development complexity and speeding up the process but also highlighted limitations in customization.

In contrast, developing the same POC using Kotlin in Android Studio presented a more traditional coding experience, requiring manual configuration of the project structure and dependencies. UI design was handled using XML layouts, and Kotlin's modern syntax features were used to implement its logic. Despite initial challenges in adapting to Kotlin's syntax and resolving compiling issues, Android Studio's debugging tool, Logcat, proved essential in testing and fixing errors. Local in-memory storage using the Data Access Object (DAO) pattern was implemented for data management, offering control over the application's storage. The Kotlin POC development process required more time investment but offered deeper insights about its platform's customization and flexibility.

The hands-on development of the expense tracker app in both OutSystems and Kotlin provided invaluable insights into the comparative strengths and challenges of low-code versus traditional programming methods. Following this chapter, the *Methodology* chapter will detail the research approach, data collection, and analysis methods employed to further explore these differences.

4. Methodology

This chapter outlines the research technique used in this work, breaking down the research process into two main areas of focus: the approach used to conduct the research and the research method that was used.

This research aims to conduct a comparative analysis between software development using the low-code platform OutSystems and traditional mobile programming methods, specifically focusing on the development of a mobile application using OutSystems Service Studio. The methodology will also include analysis on the existing literature and an in-depth study of the tools used in the development of this project.

4.1 Research Method

Qualitative and quantitative approaches are the two primary categories of research methodology. I decided to use qualitative methods for this research, which are defined by Wainer (2007) as those that involve a more in-depth analysis of a system in the environment in which it is inserted or intended to be used.

Wainer (2007) also states that qualitative methods are different from quantitative methods in a way that they deal with variables that can only be observed rather than measured. The distinction is clear and straightforward. Unlike quantitative approaches, which originate in the natural sciences, qualitative methods are derived from the social sciences.

4.1.1 Qualitative Approach

This study employs a qualitative research method, which places emphasis on the gathering and examination of non-numerical data to investigate the distinctions in software development between OutSystems mobile and conventional approaches to mobile programming. Because it enables a thorough examination of people's viewpoints, experiences, and perceptions of which may not be properly captured by quantitative methods alone. Qualitative research is especially well-suited for this kind of investigation (Creswell & Poth, 2018).

According to Creswell & Poth (2018), the philosophy behind the qualitative research approach is to comprehend and explain the complex realities of social processes and human experiences. Its foundation is the idea that people create their own subjective interpretations of

the world, which make it impossible for quantitative measurements to fully represent. In order to understand the underlying meanings, patterns, and processes that influence people's thoughts, behaviors, and relationships, qualitative research places a strong emphasis on examining the depth, richness, and contextuality of human experiences.

Qualitative research is distinguished by its flexibility, which enables researchers to modify their methodologies and methods to fit the particulars of the event that they are studying as well as the research context. It is by its very nature exploratory and iterative, frequently including an inductive process of gathering and analyzing data, where ideas and hypotheses are not imposed from above but rather arise from the facts. In order to obtain participants' lived experiences and viewpoints, qualitative researchers work closely with them in an effort to build relationships, confidence, and empathy (Merriam, 2009).

Rubin & Rubin (2012) state that qualitative research methods comprise a wide range of approaches, such as document analysis, focus groups, observations, and interviews, each with unique benefits for capturing various facets of the investigated topic. In example, interviews are a key technique in qualitative research because they enable researchers to engage individuals in discussion and extract insightful stories, thoughts, and rich narratives. Researchers can probe deeply into participants' experiences, perceptions, and interpretations through open-ended questioning and probing, revealing underlying motivations, meanings, and emotions.

4.1.2 Justification for Qualitative Design

The decision to opt for a qualitative approach is grounded in the need to understand the underlying reasons, motivations, and challenges associated with software development practices using OutSystems and traditional methods. While quantitative methods excel at providing numerical comparisons of development metrics, they may fall short in capturing the nuanced experiences and perceptions of developers (Merriam, 2009). Given the complex and multifaceted nature of software development, qualitative research offers a more holistic and in-depth understanding of the subject matter (Creswell & Creswell, 2017). By employing a qualitative design, this study aims to uncover the subjective experiences and perceptions of developers to address the research question effectively.

4.1.3 Phenomenological Research Design

A phenomenological research design will be used for this study, which focuses on examining people's personal interpretations and lived experiences related to a certain occurrence.

The phenomenological methodology is well-suited to this study because it allows this research to delve deeply into developers' experiences with both OutSystems and traditional approaches, exposing their distinct views and insights.

The phenomenological research design is a qualitative methodology that involves several key elements, including selecting a phenomenon of interest, recruiting participants who have experienced the phenomenon, and collecting descriptive data through methods such as interviews or observations. By using these techniques, researchers can extract comprehensive descriptions that capture participants' feelings, perceptions, experiences, and uncover valuable insights of the occurrence (Creswell & Poth, 2018).

Creswell & Poth (2018) also states that data analysis plays a crucial role in phenomenological research as it enables researchers to uncover the fundamentals of the phenomenon under investigation. Through rigorous analysis of participants' narratives and descriptions, researchers can identify recurring themes, patterns, and meanings that provide insight into the lived experiences of individuals. In addition to helping in the comprehension of the phenomenon itself, this analytical process clarifies the larger ramifications and importance of these experiences for both individuals and society.

4.1.4 Data Collection Method

Qualitative data will be collected through structured interviews that will allow for in-depth exploration of developers' experiences, perceptions, and challenges, providing additional insights from the sample of participants (Creswell & Poth, 2018). Data collection methods will be flexible and iterative, allowing this research to adapt questions and based on emerging themes and insights (Merriam, 2009).

4.2 Methodology Summary

This chapter outlines the research approach used to conduct a comparative analysis of software development between the low-code platform OutSystems and traditional mobile programming methods. A qualitative research method was used, focusing on gathering and analyzing non-numerical data to investigate developer experiences and perceptions. Qualitative methods were chosen for their ability to explore in-depth the complexities of software development practices, which cannot be fully captured by quantitative metrics (Creswell & Poth, 2018).

The study uses a phenomenological research design to examine developers' lived experiences with both OutSystems and traditional programming methods. This approach enables the research to dive into developers' subjective views, revealing the deeper meanings and patterns behind their development practices (Creswell & Poth, 2018; Merriam, 2009). Data was collected through structured interviews, allowing participants to describe their personal experiences, challenges, and insights regarding the development processes. These interviews provided descriptive data that contributed to a comprehensive understanding of the differences between the two development approaches.

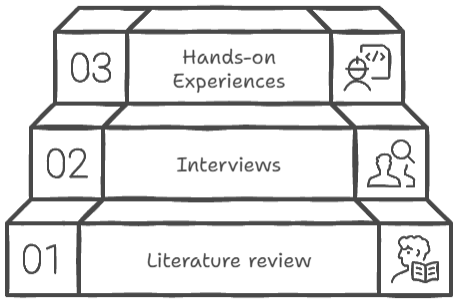

Figure 24 Steps of this study

Following this, is the *Analyzing the results* chapter, where the collected data will be thoroughly examined and interpreted in relation to the research question.

5. Analyzing the results

5.1 Introduction

The objective of this comparative analysis is to evaluate the differences in software development between the low-code platform OutSystems and traditional mobile programming methods. This analysis is based on a literature review, hands-on development experiences, and qualitative insights gathered from developer interviews. The specific interview questions used to gather these insights are listed below as well as some information about each interviewed subject. These sources provide a multi-dimensional view of the strengths and challenges associated with each development method. The interviews included six questions designed to gather detailed responses about the developers' experiences with both OutSystems and traditional programming methods, focusing on aspects such as development speed, learning curve, flexibility, customization, maintainability, and scalability. The decision to create a custom interview, rather than using an existing model, was driven by the specificity of the research question. Due to the unique focus of this study on comparing low-code platforms to traditional mobile development, no established interview framework was found that adequately addressed both methodologies. Therefore, a tailored approach was necessary to gather relevant insights specific to this comparative analysis.

This custom interview was based on the literature review, ensuring that the questions reflected key factors such as development speed, learning curve, flexibility, customization, maintainability, and scalability. It is important to note that this interview was not designed as a general model for future studies, but rather as a targeted tool to assist in comparing the two development methodologies within the context of this research. The intent was to gain a deeper understanding of developer experiences and perceptions, directly aligning with the objectives of the study, rather than establishing a reusable interview framework.

Table 1 Interviewed subjects

	Gender	*Age*	*OutSystems Experience (years)*	*Traditional Experience (years)*
Developer 1	Male	49	18	4
Developer 2	Male	25	6	0
Developer 3	Male	35	8	8
Developer 4	Male	24	5	0
Developer 5	Male	32	6	2
Developer 6	Male	25	2	0
Developer 7	Female	27	3	1

5.1.1 Interview questions

Question 1

Could you share your gender, age, and years of experience in the field of Information Technology (IT)? Additionally, if applicable, how many years of experience do you have working with low-code platforms, such as OutSystems, and how many years of experience do you have in traditional coding methods, if any?

Question 2

Can you describe your experience with both traditional mobile programming methods and developing applications using the low-code platform OutSystems? What are some key differences you've observed in terms of development workflow, tools, and overall experience?

Question 3

When considering factors such as development speed, scalability, and maintainability, how do you describe the effectiveness of OutSystems compared to traditional mobile programming methods? Are there any specific aspects of each approach that stand out to you as advantageous or challenging?

Question 4

In your opinion, how does the learning curve of using OutSystems compare to the traditional mobile programming languages? Have you encountered obstacles or advantages related when adopting OutSystems compared to traditional development practices?

Question 5

Can you share insights into the flexibility and customization capabilities offered by OutSystems compared to traditional mobile programming methods? Do you think that OutSystems is suitable for a wide range of project requirements and use cases, or are there limitations?

Question 6

From a project management perspective, how does the use of OutSystems impact collaboration, communication, and project delivery compared to traditional mobile programming methods? Have you observed any differences in team dynamics, stakeholder engagement, or overall project success metrics when using OutSystems compared to traditional development approaches?

5.2 Literature Review Overview

The literature review highlights the growing importance of digital transformation and the role of low-code development platforms like OutSystems in accelerating this process. Low-code platforms, characterized by their visual interfaces and drag-and-drop functionalities, allow for rapid application development with minimal hand-coding. This approach contrasts sharply with traditional mobile programming methods, which involve writing detailed code in languages like Java, Kotlin, Swift, and Objective-C.

OutSystems, a low-code platform, offers tools for both web and mobile application development, emphasizing ease of use and quick deployment. The platform's ability to integrate with existing systems and its support for modern development practices and AI integration further underscores its relevance in software development.

Contrarily, traditional mobile programming methods, while offering greater control and customization, require extensive knowledge of specific programming languages and frameworks. These methods typically involve more complex development cycles and longer deployment times due to the necessity of writing and maintaining large codebases.

5.3 Hands-On Development Experience first impressions

To help evaluate the differences between these approaches, a proof-of-concept (POC) expense tracker application was developed using both OutSystems and Kotlin.

The OutSystems POC was developed using Service Studio, OutSystems' visual development environment. The process began with setting up the development environment, which was straightforward due to OutSystems' cloud-based infrastructure. The user interface was designed using drag-and-drop components, allowing for rapid prototyping and real-time previews. Key functionalities, such as expense entry, income tracking, and budget visualization, were implemented using OutSystems' visual action flows and aggregate operations for data manipulation.

OutSystems' built-in debugging tools and visual data modeling capabilities significantly reduced development time and complexity. The platform's emphasis on reusable components and templates facilitated quick iteration and ensured consistent application architecture.

Developing the same POC in Kotlin involved setting up Android Studio, configuring the project structure, and managing dependencies. The user interface was designed using XML layouts, and business logic was implemented using Kotlin's modern syntax features, such as null

safety and extension functions. Despite initial challenges with Kotlin's syntax and deprecated libraries, the robust documentation and community support helped resolve these issues.

Data storage in the Kotlin POC was handled using local in-memory structures with a DAO pattern, which provided a clean separation between business logic and data access. Testing and debugging were carried out using Android Studio's integrated tools, such as Logcat, which provided detailed error information and facilitated efficient problem resolution.

5.4 Interview first impressions

Interviews with developers who have experience in both low-code and traditional mobile development provided qualitative insights into the comparative advantages and challenges of each approach.

Developers highlighted the rapid development capabilities of OutSystems, noting its high efficiency in building and deploying applications. Developer 1 stated that the platform's visual development environment was easy to use and allowed for quick production of functional applications "…the platform has evolved and today the generation of mobile apps is easier and more intuitive…" (Developer 1). However, some developers, like Developer 3, pointed out limitations in customization and performance when compared to traditional coding methods "…the main challenges arise when demanding higher performance or requiring greater access to the underlying code language." (Developer 3).

On the other hand, developers with experience in traditional programming emphasized the greater flexibility and control offered by native development tools. Developer 3 mentioned that development times were longer "When utilizing traditional code, customization and flexibility are greater, albeit at the cost of increased time consumption…" (Developer 3), the ability to fine-tune performance and create highly customized solutions was invaluable. The developers also noted the importance of understanding coding syntax and logic, which can lead to more robust and maintainable applications.

5.5 In depth comparative analysis

The objective of this comparative analysis is to evaluate the differences in software development between the low-code platform OutSystems and traditional mobile programming methods. This analysis is based on the literature review, hands-on development experiences, and

qualitative insights gathered from developer interviews. These sources provide a view of the strengths and challenges associated with each development method.

5.5.1 Development Speed and Efficiency

The literature review emphasizes the rapid development capabilities of low-code platforms like OutSystems. Richardson and Rymer (2014) highlight that low-code platforms significantly reduce development time through visual interfaces and pre-built components. This was corroborated in the hands-on experience, where the OutSystems POC was developed quickly using drag-and-drop tools and visual action flows. Developer 1 also noted that OutSystems allowed for the rapid production of functional applications, significantly cutting down the time required for initial development phases "OutSystems provides a rapid application development platform (…) which allows us to visually and intuitively build applications effectively and securely." (Developer 1).

In contrast, traditional programming methods, as described in the literature and by Developer 2, involve a longer development cycle due to the need for writing and maintaining detailed codebases in traditional languages "…I can confidently say OutSystems allows developers to build apps way faster compared to those built with traditional mobile methods." (Developer 2). The hands-on experience with Kotlin POC confirmed this, as it required setting up Android Studio, configuring dependencies, and managing XML layouts, all of which contributed to a lengthier development process. This aligns with the findings of Charland and Leroux (2011), who describe the time-intensive nature of traditional development.

5.5.2 Learning Curve

The literature review and interviews indicate that OutSystems has a gentler learning curve compared to traditional programming methods. Low-code platforms are designed to be accessible to users with minimal coding experience, making them suitable for a broader range of developers. This was evident in the hands-on experience, where the intuitive visual development environment of OutSystems facilitated quick onboarding and ease of use. Developer 3 mentioned that even those with limited programming backgrounds could quickly learn to develop applications using OutSystems "…people with limited IT knowledge Low-Code is easier to learn and understand than traditional code languages …" (Developer3).

On the other hand, traditional programming requires a deep understanding of specific languages and development environments. Developer 2 pointed out the steeper learning curve

associated with mastering traditional languages, which was also reflected in the hands-on experience “The learning curve in OutSystems is less aggressive since we don't need to be worried about all the syntax needs that traditional development requires.” (Developer 2). The extensive setup and configuration required for Android Studio and the detailed coding in Kotlin presented a significant barrier to entry, which is consistent with the literature's emphasis on the specialized knowledge required for traditional programming (Malavolta et al., 2015).

5.5.3 Flexibility and Customization

Traditional programming offers greater flexibility and customization, as highlighted by both the literature and Developer 3. The ability to write detailed code allows developers to fine-tune performance and tailor applications to specific needs “When utilizing traditional code, customization and flexibility are greater…” (Developer 3). The hands-on experience with Kotlin demonstrated this flexibility, as the custom code provided precise control over application behavior.

However, the literature review and Developer 3 noted limitations in OutSystems regarding extensive customization “When utilizing traditional code, customization and flexibility are greater…” (Developer 3). While OutSystems excels in rapid development and ease of use, its predefined components and frameworks can constrain highly specialized applications.

5.5.4 Maintainability and Scalability

OutSystems provides robust tools for maintainability, such as visual debugging and modular development practices. The platform's emphasis on reusable components and templates ensures consistent application architecture, which simplifies maintenance. Developer 1 highlighted the ease of managing and updating applications developed in OutSystems “…it becomes simpler to manage the entire life cycle of applications, as with this platform any change or adaptation is also easier and faster.” (Developer 1), which was supported by the hands-on experience where built-in debugging tools facilitated efficient problem resolution.

However, the literature review suggests that scalability can be an issue for low-code platforms when dealing with large, complex projects. This was echoed by Developer 3, who pointed out that while OutSystems is excellent for fast development cycles, traditional programming methods offer superior scalability and maintainability for larger applications “Traditional coding tends to scale better than Low-Code because effective language coding entails fewer restrictions…” (Developer 3). The hands-on experience with Kotlin confirmed this, as it

was possible to note the detailed control over the codebase allowing for better management of logic.

5.5.5 Developer Experience and Preference

Interviews revealed that developer experience and preference play a significant role in the choice of development method. Developer 3, with extensive experience in traditional programming, preferred the control and flexibility it offered despite the longer development times. Conversely, Developer 1 and 2, who had more experience with low-code platforms, appreciated the speed and ease of use provided by OutSystems. This variability in preferences underscores the importance of aligning the development approach with the team's expertise and project requirements.

5.5.6 Concluding the comparative analysis

The comparative analysis highlights distinct advantages and challenges associated with low-code and traditional development methods. OutSystems excels in rapid development, ease of use, and maintainability, making it ideal for projects with tight deadlines and limited resources. Traditional programming methods, while requiring more time and specialized knowledge, offer unparalleled flexibility, customization, and scalability, making them suitable for complex and high-performance applications. The choice between these approaches should be based on project requirements, team expertise, and the specific needs of the application. By leveraging the strengths of both methods, organizations can optimize their software development processes and achieve their digital transformation goals effectively.

5.6 Result analysis Summary

This chapter presents a comparative analysis between software development using the low-code platform OutSystems and traditional mobile programming methods, based on insights from the literature review, hands-on development experiences, and qualitative interviews with developers. The interviews, consisting of six custom questions, focused on development speed, learning curve, flexibility, customization, maintainability, and scalability. The custom interview was created due to the unique focus of the study, as existing frameworks were not suitable for addressing the specific comparison between low-code and traditional methods.

Development Speed and Efficiency: The literature review and hands-on experiences demonstrate that OutSystems significantly accelerates development due to its visual interface and pre-built components. Developers confirmed that applications could be built much faster using OutSystems compared to traditional methods, which require more time for setup, coding, and debugging.

Learning Curve: OutSystems offers a gentler learning curve, making it accessible to developers with limited coding experience. This was supported by both the literature review and developers, who noted that OutSystems facilitated the onboarding. In contrast, traditional programming methods require in-depth knowledge of coding languages and environments, making the learning process more time-consuming and challenging.

Flexibility and Customization: Traditional programming methods provide greater flexibility and customization, allowing developers to fine-tune applications to meet specific needs. While OutSystems is ideal for rapid development, it is limited in its ability to handle highly specialized applications due to its predefined components and frameworks.

Maintainability and Scalability: OutSystems offers strong maintainability features, such as visual debugging and reusable components, which simplify application management. However, scalability can be an issue for larger and more complex projects, where traditional programming methods provide superior control and adaptability.

The analysis revealed that developer preferences vary depending on their experience with each development method. Those with more experience in traditional programming valued control and flexibility, while developers familiar with low-code platforms appreciated the speed and ease of development.

After this, the final chapter of the study, the *Conclusions* chapter. With all the gather information from this study we will answer the research question, leave some suggestions for future studies and some limitations of this one.

6. Conclusions

The research question guiding this study was: "What are the differences in software development between the low-code platform OutSystems and traditional methods of mobile programming?". To address this question comprehensively, a thorough analysis was conducted, integrating insights from a literature review, hands-on development experiences with both OutSystems and Kotlin, as well as qualitative insights gathered through interviews with developers experienced in both methodologies. This approach allowed for a deeper understanding of the strengths, challenges, and practical implications of each development method, contributing with valuable knowledge to both academic and professional fields.

6.1 Development Speed

The findings consistently indicate that OutSystems significantly accelerates the development process. The platform's visual interface, drag-and-drop functionalities, and pre-built components enable rapid prototyping and deployment. This is emphasized both in the literature review and supported by the hands-on development experience. Developers highlighted that applications could be built in a fraction of the time required by traditional programming methods. The ease of use and rapid iteration cycle facilitated by the visual tools in OutSystems provided a clear advantage in scenarios where speed was crucial, allowing to quickly transition from concept to product. The literature points to similar conclusions, indicating that OutSystems and other low-code platforms can drastically reduce time-to-market for applications, which is critical in fast-paced business environments where timely delivery of software solutions is tied to competitive advantage.

However, while the speed of development in OutSystems is faster, it is essential to consider that the accelerated timeline is most effective for straightforward projects. Larger, more complex applications with extensive customization needs may not benefit as significantly from this speed, as performance and structural issues may arise. In these cases, traditional programming, although slower, may result in a more robust and scalable product, particularly when more custom logic is required.

6.2 Learning Curve

OutSystems offers a gentler learning curve compared to traditional programming. Its user-friendly visual environment enables developers with limited coding experience to quickly start building functional applications, a point emphasized both in the literature and supported by the interviewed developers. OutSystems simplifies complex tasks and abstracts away many technical details, making the development process accessible to a wider range of individuals,

including non-developers. This accessibility provides organizations with the ability to distribute development work across teams with varying levels of technical expertise, thereby accelerating the overall development process and allowing for more collaborative, cross-functional teams.

The hands-on experience demonstrated how the intuitive interface of OutSystems reduced the barriers for development. However, traditional programming methods require a thorough understanding of specific languages such as Kotlin, Java, or Swift, as well as experience in their respective development environments. This deep technical knowledge can be a barrier for beginners. The hands-on development with Kotlin confirmed this, as it involved complex setup of the environment, manual configuration, and specific syntax for coding. For developers trained in these languages, the result shows more control and flexibility, but the initial investment in learning and skill acquisition is significantly higher. This creates a contrast where OutSystems shines in environments where speed and accessibility are prioritized, but traditional methods become advantageous in cases where technical depth is required.

6.3 Flexibility and Customization

While OutSystems excels in speed and ease of use, its limitations become evident when it comes to flexibility and customization. Both the literature review and Developer 3 pointed out that low-code platforms, while powerful, may struggle with highly specialized applications that require advanced customizations. The visual interface and pre-defined components of OutSystems are designed for quick assembly but can become restrictive when more intricate or unique functionality is needed. In the hands-on experience, while no significant challenges were encountered with implementing basic customizations, it was clear that more complex custom features could potentially push the boundaries of what OutSystems could easily handle.

In contrast, traditional programming methods offer much more flexibility and customization, allowing developers to tailor applications to precise business requirements. The ability to write detailed code means that traditional programming can meet a wider variety of specific use cases, making it ideal for projects where customization is a must. The hands-on experience with Kotlin illustrated this advantage, as the traditional method allowed for the implementation of fine-tuned logic and control over the application's behavior. This level of customization is difficult to replicate with low-code platforms, highlighting traditional programming's suitability for projects requiring a high degree of specificity.

6.4 Maintainability and Scalability

OutSystems provides robust tools for maintainability, such as visual debugging and modular development practices. These features facilitate updates and ongoing maintenance of applications, as the modular design allows components to be reused and updated independently. For small to mid-sized applications, this can result in significant time savings when managing long-term maintenance. The hands-on experience with OutSystems demonstrated how these features make it easier to keep applications up-to-date and resolve issues without deep coding knowledge.

However, the literature review and Developer 3 both showed concerns about the scalability of low-code platforms when dealing with large, complex projects. The predefined components and framework limitations that enable rapid development can also restrict the scalability of applications, particularly as projects grow in size and complexity. For businesses that require highly scalable and high-performance solutions, traditional programming methods offer superior scalability and maintainability. As demonstrated in the Kotlin POC, the control over the codebase would allow for better management of complex logic and more efficient handling of large-scale operations. Though traditional methods require more effort in managing large codebases, they are better suited for organizations with long-term, large-scale projects that demand ongoing flexibility, specificity and high-performance.

6.5 Last thoughts

In conclusion, the choice between low-code platforms like OutSystems and traditional programming methods should be made based on specific project requirements, the level of expertise within the development team, and the complexity of the application. OutSystems offers significant advantages in development speed and accessibility, making it ideal for projects with tight deadlines, smaller teams, or limited technical resources. The platform's ease of use and quick learning curve provide a practical solution for rapidly deploying applications, particularly in fast-paced environments.

On the other hand, traditional programming methods, though requiring more time and specialized knowledge, provide unmatched flexibility, customization, and scalability, making them suitable for complex, high-performance applications. These methods offer the control needed to handle highly specific requirements and larger-scale applications that low-code platforms may struggle to accommodate. By understanding and leveraging the strengths of both approaches, organizations can optimize their software development processes and effectively achieve their digital transformation goals. Future studies could expand on this comparison by exploring the performance and resource utilization of applications developed in both

environments or investigating long-term maintainability and user experience across these two methodologies.

6.6 Future Studies

This study lays the groundwork for future research in several areas. One potential direction is a quantitative analysis comparing the performance and resource utilization of applications developed using low-code platforms versus traditional methods. Performance metrics such as load times, response rates, memory consumption, and scalability under high user loads could be key focus areas, providing data that highlights how each approach holds up under real-world usage conditions. Another area of quantitative research could involve benchmarking development time, error rates, and productivity across multiple teams using different methodologies to evaluate how low-code platforms impact development efficiency and code quality over time.

Another avenue could explore the long-term maintainability and total cost of ownership of applications built on low-code platforms compared to traditionally developed ones. This research could include a study that tracks application updates, maintenance efforts, and the cost of making iterative changes across both low-code and traditionally coded applications. It would be insightful to examine how low-code platforms handle increasing complexity over time, especially in terms of technical debt accumulation, as well as how well they integrate with evolving technology ecosystems like cloud services, APIs, and frameworks. Additionally, it could assess whether low-code platforms prove more cost-effective in the long run or if traditional methods, with their more flexible customization, reduce expenses associated with platform limitations.

Additionally, investigating the user experience and satisfaction of end-users with applications developed through these differing methodologies could provide valuable insights. This could involve gathering feedback from end-users regarding the usability, performance, and overall experience of applications developed through both low-code platforms and traditional coding approaches. Understanding how end-users perceive and interact with these applications would not only help refine development practices but also aid organizations in making decisions that align with customer satisfaction. Another potential avenue could explore developer satisfaction and retention when working with low-code platforms versus traditional programming methods, as well as how these approaches affect team dynamics, skill development, and overall productivity within an organization.

By expanding on these areas, future research can further inform the decision-making processes around choosing the most appropriate software development methodologies, offering a deeper understanding of the practical, financial, and technical implications of low-code platforms versus traditional methods.

6.7 Study limitations

While this study provides valuable insights into the comparative strengths and challenges of low-code platforms like OutSystems versus traditional programming methods, it is not without limitations. One significant limitation is the scope of the hands-on development experiences, which focused on a single proof-of-concept (POC) application, potentially limiting the findings to more complex or varied project types. Additionally, the qualitative nature of the developer interviews, while rich in detail, may introduce biases that could affect the broader applicability of the results. Finally, the study did not account for all potential variables, such as the impact of team size, or project management practices, which could influence the outcomes of software development projects.

7. References

Alexander, F. (2021, February 11). Build it fast, right, and for the future: our favorite things. *OutSystems*. https://www.outsystems.com/blog/posts/few-of-our-favorite-things/

Apple Inc. (2023). *Apple Developer Documentation*. Retrieved March 18, 2024, from https://developer.apple.com/documentation/

Apple Inc. (2024). *Swift.org*. Retrieved March 20, 2024, from https://www.swift.org/documentation/

Charland, A., & LeRoux, B. (2011). Mobile application development: Web vs. native. Communications of the ACM, 54(5). https://doi.org/10.1145/1941487.1941504

Creswell, J. W., & Poth, C. N. (2018). Qualitative Inquiry and Research Design Choosing Among Five Approaches (4th Edition ed.). In Journal of Materials Processing Technology (Vol. 1, Issue 1).

Creswell, John W and Creswell, J. D. (2017). Qualitative, quantitative, and mixed methods approaches. Research Design Qualitative Quantitative and Mixed Methods Approaches. In Research Design (Vol. 4, Issue June).

Dalmasso, I., Datta, S. K., Bonnet, C., & Nikaein, N. (2013). Survey, comparison and evaluation of cross platform mobile application development tools. 2013 9th International Wireless Communications and Mobile Computing Conference, IWCMC 2013. https://doi.org/10.1109/IWCMC.2013.6583580

Duro, J. (2022, March 30). OutSystems — Create a .NET extension — Date difference in months. Medium. https://itnext.io/outsystems-create-an-net-extension-date-difference-in-months-cedb4f5f4289

Google. (2023, April 12). *Get to know the Android Studio UI*. Retrieved March 7, 2024, from https://developer.android.com/studio/intro/user-interface

Google. (2023, July 25). *Build and run your app*. Retrieved March 7, 2024, from https://developer.android.com/studio/run

Google. (2024, March 4). *Meet Android Studio*. Retrieved March 5, 2024, from https://developer.android.com/studio/intro

Habraken, M. (2017, October 27). May the Forge be with you. . . - Martijn Habraken - Medium. Medium. https://medium.com/@m.habraken/may-the-forge-be-with-you-e8c61341d3cb

Kotlin. (n.d.). *Kotlin Documentation*. Retrieved March 8, 2024, from https://kotlinlang.org/docs/home.html

Malavolta, I., Ruberto, S., Soru, T., & Terragni, V. (2015). Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation. Proceedings - 2nd ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft 2015. https://doi.org/10.1109/MobileSoft.2015.15

Matt, C., Heß, T., & Benlian, A. (2015). Digital Transformation Strategies. *Business & Information Systems Engineering*, *57*(5), 339–343. https://doi.org/10.1007/s12599-015-0401-5

Mendix. (2024). *Mendix Documentation*. Retrieved March 24, 2024, from https://docs.mendix.com/

Merriam, S. B. (2009). Qualitative research: A guide to design and implementation. In The JosseyBass higher and adult education series (Vol. 2nd). https://doi.org/10.1097/NCI.0b013e3181edd9b1

OutSystems. (2023, December 21). *Data*. Retrieved February 24, 2024, from https://success.outsystems.com/documentation/11/reference/outsystems_language/data/

OutSystems. (2023, December 21). *Getting Started*. Retrieved December 28, 2023, from https://success.outsystems.com/documentation/11/getting_started/

OutSystems. (2023, December 21). *Interfaces*. Retrieved December 28, 2023, from https://success.outsystems.com/documentation/11/reference/outsystems_language/interfaces/

OutSystems. (2023, December 21). *Interfaces*. Retrieved February 13, 2024, from https://success.outsystems.com/documentation/11/developing_an_application/implement_application_logic/actions_in_web_applications/

OutSystems. (2023, December 21). *Logic*. Retrieved December 28, 2023, from https://success.outsystems.com/documentation/11/reference/outsystems_language/logic/

OutSystems. (2023, December 21). *Processes*. Retrieved February 24, 2024, from https://success.outsystems.com/documentation/11/developing_an_application/use_processes_bpt/processes/

OutSystems. (2023, December 21). *Service Studio*. Retrieved December 28, 2023, from https://success.outsystems.com/documentation/11/getting_started/service_studio_overview/

Pereira, R. (2023, September 21). OutSystems 11: How to use Service Center - ITNEXT. Medium. https://itnext.io/outsystems-11-how-to-use-service-center-aef5c051bc56

Richardson, C. & Rymer, J. R. (2014). New Development Platforms Emerge For Customer-Facing Applications. *Forrester*

Rogers, R., Lombardo, J., Mednieks, Z., & Meike, B. (2009). Android Application Development: Programming with the Google SDK. O'Reilly Media, Incorporated.

Rubin, H., & Rubin, I. (2012). Qualitative Interviewing (2nd ed.): The Art of Hearing Data. In Qualitative Interviewing (2nd ed.): The Art of Hearing Data. https://doi.org/10.4135/9781452226651

Rymer, J. R. (2017). Low-Code Development Platforms For AD & D Pros. *The Forrester Wave*

Sahay, A., Indamutsa, A., Ruscio, D.D., Pierantonio, A., (2020). Supporting the understanding and comparison of low-code development platforms, in: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). *Presented at the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 171–178. https://doi.org/10.1109/SEAA51224.2020.00036

Sousa, J. (2022, January 29). Lifetime — Create a Governance Model that will make you proud. Medium. https://medium.com/@neves.sousa/lifetime-create-a-governance-model-that-will-make-you-proud-81885a457e7a

Vial, G. (2019). Understanding digital transformation: A review and a research agenda. *Journal of Strategic Information Systems*, *28*(2), 118–144. https://doi.org/10.1016/j.jsis.2019.01.003

Vincent, P. (2019, November 25). *Gartner: What to consider before adopting low-code development*. ComputerWeekly.com. https://www.computerweekly.com/feature/Gartner-What-to-consider-before-adopting-low-code-development

WAINER, J. (2007). Métodos de pesquisa quantitativa e qualitativa para a ciência computação. In T. Kowaltowski & K. Breitman (Orgs.), Atualização em Informática da Sociedade Brasileira de Computação

Yin RK. (2016). Qualitative Research: From start to finish. Second Edition. In THE GUILFORD PRESS. New York London.

8. Appendix

8.1 Interview answers

8.1.1 Developer 1

A1: Male; Age: 49; Experience in the IT area: 18; OutSystems Experience: 18; Traditional coding experience: 4

A2: Over time I have developed or been part of teams that developed mobile apps with OutSystems. I've never done it with traditional methods, but I've heard a lot about Xamarin.

The first experience was with OutSystems 9, where it was still necessary to use a lot of JavaScript and an emulator called OutSystems Now was used. I would say that for the first steps it wasn't bad, but the "wow factor" was still missing here.

Over time, the platform has evolved and today the generation of mobile apps is easier and more intuitive, we can generate native apps or PWA. The development method is similar to that of any Reactive application, using the various accelerators and widgets that the platform makes available.The main challenges are synchronization management and offline.

A3: OutSystems provides a rapid application development platform (commonly called RAD), which allows us to visually and intuitively build applications effectively and securely. We are talking about WEB and Mobile applications supporting the 2 best-known providers on the market, iOS and Android.

How is this done? Basically, the development of applications using the OutSystems platform is based on an IDE capable of building pages and functionalities visually, through drag and drop of properly coded objects.

OutSystems has a "Full-stack visual language, which allows you to develop all layers of the application". This visual language is simpler to understand, easier to convey knowledge and allows the programmer to focus on what he wants to implement, rather than how to implement it.

Also, for this reason, it becomes simpler to manage the entire life cycle of applications, as with this platform any change or adaptation is also easier and faster.

Therefore, every aspect of the OutSystems platform is designed to help you build better applications, faster. And all the added value of acceleration is for both the development of web and mobile applications.

A4: The learning curve in OutSystems is very high. However, the developer must have good concepts of logic. Only then will you be able to use all the capabilities that the platform offers. It is not necessary to know code syntax, as it is all "embedded" in the various components we use.

In the traditional method, the main problem is that the developer must know the syntax he is going to use, and sometimes, whether due to lack of knowledge or lack of attention, many difficult mistakes are made after finding them. There are applications with thousands of lines of code. With OutSystems this problem does not happen, because there is a very useful feature called TrueChange that immediately gives us an indication of errors.

A5: Yes, OutSystems is suitable for a wide range of project requirements. You can do all with OutSystems.

With OutSystems, we can create from scratch, customize pre-built templates or import existing templates, ensuring a good image of the application, such as performance and functionality.

In other words, one of the advantages of OutSystems is that the platform provides the building blocks to deliver these experiences faster. We have ready access to predefined UI patterns and page templates with layouts, widgets, components, styles and logic that we can add to our applications using a (What You See Is What You Get) type.

All this using a very pleasant visual model, avoiding boring moments and confusing code.

In short, this platform can extend with external code (.net), and the nature of the applications are entirely based on standard code.

A6: The main methodology used in Low-Code projects is SCRUM.

Scrum can significantly improve the effectiveness and efficiency of communication projects by promoting collaboration, flexibility, transparency and continuous improvement.

The team feels involved in the decisions made to bring the project to success. Then the stakeholders themselves, through the various demos they see, normally every 15 days, also feel integrated into the project.

It's true that all of this can also be applied to traditional projects, but the speed with which it is done and shows something functional is something that only Low-Code can provide.

8.1.2 Developer 2

A1: Male, 25y; 5.5 years in IT; 5.5 years in Outsystems.

A2: While in school, I used traditional mobile methods to develop an android app.

After working with Outsystems for a couple of years, I can confidently say Outsystems allows developers to build apps way faster compared to those built with traditional mobile methods.

Also, devs get prepared for real world scenarios quicker in outsystems since the learning curve is less aggressive.

The outsystems tools end up being easier to use aswell.

A3: Outsystems allow developers to easily build reusable blocks that be referenced by other applications. This helps alot with maintainability and development speed since we don't need to rebuild the same logic/ui repeatedly.

A4: The learning curve in outsystems is less aggressive since we don't need to be worried about all the syntax needs that traditional development requires. The platform ends up focusing 90% on logic flows instead of code.

A5: At the moment there are still some limitations in Outsystems. But since we have Integration studio available at any time, every time we found something that we need and Outsystems doesn't have it yet, we can always build extensions with C# or Java, which unblocks the limitations most of the times.

A6: No answer for this.

8.1.3 Developer 3

A1: Male; 35 years old; with 16 years of software development experience, including 8 years specializing in Outsystems.

A2: OutSystems simplifies the development of various applications, reducing development time. Its main advantage lies in creating generic code that can be applied across different infrastructures and types of applications.

A3: When comparing the same code:

Scalability: Traditional coding tends to scale better than Low-Code because effective language coding entails fewer restrictions compared to using a framework.

Maintainability: Low-Code offers easier maintainability than Traditional coding because visual flow representation simplifies understanding.

Development Speed: Low-Code development is faster than Traditional coding due to numerous accelerators such as Aggregates, UI Templates, and basic security features.

A4: For people with limited IT knowledge Low-Code is easier to learn and understand than traditional code languages because it requires managing fewer variables; Low-Code platforms already handle some variables. However, the main challenges arise when demanding higher performance or requiring greater access to the underlying code language.

A5: When utilizing traditional code, customization and flexibility are greater, albeit at the cost of increased time consumption due to the need to select from numerous libraries to achieve specific functionalities. OutSystems excels in catering to a wide range of projects. However, when it comes to requiring code for a particular device or machine, especially in cases of specificity, employing OutSystems or Low-Code platforms can be more challenging.

A6: With Low-Code, we can respond more rapidly to market demands and accommodate changes requested by clients more efficiently.

8.1.4 Developer 4

A1: Male; age 24; 5 years of experience with low code. No experience with traditional coding methods.

A2: I can only describe my experience with low code, and it's a fast and productive way of developing web reactive or mobile apps.

A3: They normally take much less time to build and to make them available to the customer. The word low code is no joke, we still need to have good logical thinking but it's much less error prone than traditional methods.

A4: The learning curve it's much easier and faster, with low code because there is a beginner friendly tool like OutSystems with a lot of curses to attend that make a person not familiar with programing sense it's not as hard to work with then a traditional tool.

A5: There is plenty of flexibility with OutSystems, namely the integration studio that enables us to use .net and other types of languages to be implemented in OutSystems action.

A6: The collaboration impact is huge because there is a faster production release timeline, then traditional methods making the communication a key factor to project success.

8.1.5 Developer 5

A1: Male, 32, 8 years of experience total, 6 in low code

A2: The big difference is the time to deliver, which is significantly lower if you use low code.

A3: OutSystems is great, offering tools to accomplish the described factors. The main difference for traditional tools is again the time we need to accomplish the goals.

A4: OutSystems provides very good documentation and is easier to learn than other traditional languages.

A5: I see no differences between OutSystems and traditional regarding the described topics, that said, yes, OutSystems is suitable for a wide range of projects, no limitations.

A6: No differences in my opinion, OutSystems is good for fast development but has no advantages in management perspective in my opinion.

8.1.6 Developer 6

A1: Male, 25, 2 years of experience in IT. I have 2 years working with OutSystems, and 0 years of work with traditional coding.

A2: The coding experience with OutSystems is easier to understand and since it has a drag and drop interface the development speed is way faster. Reusability, one of the biggest advantages OutSystems provides is that you can reuse your work or the work of others. Once a module exists on the platform, its publicly exposed actions and data can easily be referenced in another application, with traditional coding I believe that isn't possible.

A3: In my opinion OutSystems is superior in all the factors mentioned above compared to traditional coding. We can develop faster with the drag and drop interface, scalability is also possible, it's easier to take an older application and implement more features on it since the flowcharts are easier to interpret and understand its structure. Maintainability I believe is also easier (I haven't had any experience on maintaining a traditional coding application) since there is a common architecture canvas that is used by everyone using OutSystems, any developer that has some experience can navigate through several new/unexplored apps easier since most of them are built using the same architecture. So far, no specific aspect has been challenging.

A4: I have never had working experience with traditional coding, but I have 5 years of study with traditional coding and to this day I still know nothing about it just know the basics, but with OutSystems the adaption was easier, it took me a couple of months on the job to get the hang of it and understanding how the platform works. I would say that OutSystems is way easier to learn than traditional coding, I believe that anyone with no preparation for this area of work could learn OutSystems easier and faster than they could learn traditional coding.

A5: I would say that OutSystems is just as customizable as traditional mobile programming. Even though OutSystems is a low-code platform it allows the developer to create extensions using .Net code and use it in its flow. It also allows the developer to create an application theme, customized widgets and so on using CSS.

A6: Honestly no idea, since I have never been in a work environment with traditional mobile programming, I do not have the knowledge to comment on this.

8.1.7 Developer 7

A1: Female, 27, 3 years of experience in OutSystems and 1 year in Python.

A2: I used Python to program a collaborative robot and I've been using OutSystems to develop web applications. I find OutSystems easier to develop and see results compared to Python.

A3: Both technologies are quick to develop. I think OutSystems is better to maintain, as it is more "understandable" from a developer point of view.

A4: OutSystems is easier to learn, compared to Python. However, when reaching a huge OutSystems factory, some problems such as deploys, or maintainability appear.

A5: OutSystems offers robust integration capabilities, allowing developers to easily connect to existing systems, databases, and APIs. This is crucial for projects that require interoperability with other systems or services.

A6: With OutSystems' low-code approach, communication between developers and non-technical stakeholders is often enhanced. The visual nature of development allows for clearer demonstrations of project progress and functionality, leading to better communication and alignment on project goals.