Plantonome: A Cross-Platform Application for Precision Agriculture

Anass DEROUSSI[1,*], *Abdessalam* Ait Madi[1,**], and *Imam* Alihamidi[1,***] *zakaria* chabou[1,****] *Adnane* Addaim[1,2,†]

[1]Laboratory of Advanced Systems Engineering, National School of Applied Sciences, Ibn Tofail University, Kenitra 14000, Morocco,

[2]Department of Electrical Engineering, Mohammadia School of Engineers, Mohamed V University, Rabat, Morocco

Abstract. In recent years, there has been growing interest in leveraging the Internet of Things (IoT) and Artificial Intelligence (AI) technologies for agriculture. A significant challenge for developers in this field is creating applications that provide precise data about plants, facilitating the smart automation of plant management.

This paper presents Plantonome, an open-source application developed using the Flutter software development kit (SDK) and the Dart programming language. Designed to integrate with IoT devices, Plantonome quickly and accurately identifies ornamental plant genera or species using the Plant.id API for plant image analysis. The application also utilizes a NoSQL database for storing user data and plant preferences, and it includes a dataset of ornamental plants with details such as name, brightness, temperature, and humidity requirements. The development approach outlined in this paper accelerates the creation process and results in a high-performing application with a flexible user interface and smooth user experience. The application, tested on Android 5.0 (API level 21) or higher, achieved an accuracy of 94.64% for plant identification and received highly positive feedback regarding its functionality, usability, and efficiency. This work offers significant benefits to researchers and startups aiming to develop cross-platform applications that can automate various agricultural tasks, contributing to advancements in smart agriculture.

1 Introduction

Planting in urban areas is driven by several factors today. A range of studies have highlighted the benefits of interior plants for physical and mental health, contributing to well-being, air purification, carbon dioxide conversion to oxygen, pollutant absorption, and providing a satisfying hobby [1],[2]. This has gained importance with more time being spent indoors, leading to a surge in demand for ornamental plants.Nevertheless, the lack of adequate information

*e-mail: anass.deroussi@uit.ac.ma
**e-mail: abdessalam.aitmadi@uit.ac.ma
***e-mail: imam.alihamidi@uit.ac.ma
****e-mail: zakaria.chabou@uit.ac.ma
†e-mail: adnane.addaim@uit.ac.ma

about plants often leads to their loss of vitality.The advent of the internet of things (IoT) and artificial intelligence (AI) technologies has sparked significant interest in their application to the field of agriculture. A key challenge faced by developers in this arena is the creation of an application capable of providing accurate and comprehensive data on plants, thereby facilitating the intelligent and automated control of these plants.

The proposed application, Plantonome, addresses these challenges by providing comprehensive information on ornamental plants, such as their temperature, soil, watering needs, and cultivation methods.Thereby facilitates automatic plant control, reducing the workload in plants cultivation. The application aims to foster interest in nature, fill the taxonomic gap, improve agricultural production in urban areas, and enhance individuals' physical and mental health.

Plantonome built as an open-source application compatible with IoT devices and capable of swiftly and precisely identifying ornamental plant genera or species using a href<http://Plant.id>Plant.id API for plant image analysis, a Cloud Firestore for storing user data and plant preferences, and a tailor-made dataset. The application's design, facilitated by the Flutter software development kit (SDK), the Dart programming language,and provider state managment solution wich simplifies the task of building a smart IoT solution for agriculture.

The subsequent sections of this work present a detailed discussion on the materials used in creating a cross-platform application, the application architecture, the protocols used, and the development process. The process includes a definition phase where the application's requirement specification is defined, a development phase where the specifications are implemented, an operation phase involving testing to ensure that the application is working as expected, and a validation phase where multiple application versions are created and rigorously tested.

2 Review of Literature and Related Works

This section reviews a variety of market applications designed for plant species identification, which are relevant to the research conducted in this thesis. These applications serve a broad audience, including plant enthusiasts, gardeners, hikers, and house plant explorers, who are interested in categorizing and learning about plants. The manual interpretation of plant species can be challenging and inaccurate due to individual visual perception variances. Therefore, applications providing reliable information about plants are indispensable. Such applications include 'Plant.id', 'FlowerChecker', 'Seek', 'iNaturalist', 'Flora Incognita', 'Google Lens', 'PlantNet', 'PictureThis', 'Garden Compass', 'PlantSnap', 'Name That Plant', 'Candidate Gardening', 'iPlant', 'PlantID', 'Algobase', 'Leafsnap', 'GardenAnswers', 'Plantix', 'Planta', and 'What's That Flower'. This review discusses some of these applications and presents the tools, hardware, datasets, and algorithms used behind them.

hout an internet connection, leveraging the extensive 'iNaturalist' database [3]. The 'Flora Incognita' application allows the identification of over 4800 plant species, using a deep learning CNN algorithm [4]. It is based on a large database of plant observations, and is available on Android, iOS, and Harmony OS devices [5].

The 'PlantNet' application is a research and educational tool focusing on plant biodiversity. While this application ranked fifth in terms of performance, it still managed to correctly identify nearly 50% of genus attempts. 'PlantNet' requires user input to classify an image and relies on a growing database, currently boasting over 38632 species [6].

Other applications, such as 'PlantSnap', aim to create a digital interface between people and nature. 'PlantSnap' has over 600,000 plant species and uses the Imagga's Custom Categorisation API in the DGX Station from NVIDIA, which costs $\$149,000$ per unit. It

can recognize 90% of known plants and trees, with over 316,000 species in its searchable database.

The success of these applications depends heavily on the quality of the image provided. It is critical to avoid confusing backgrounds, particularly for grasses and sedges. Despite this, all of the tested applications handle photos of flowers, leaves, or whole plants well.

For this research, we aim to improve the efficiency of AI (Artificial Intelligence) for plant detection. To achieve this, we exploit the 'Plant.id' API to maximize accuracy in identifying plant species using images from a gallery or camera.

3 Material and Methodology

This section delves into the methodologies and frameworks employed in the creation of the plant identification application 'Plantonome'. In the realm of botanical research and application development, especially in the sectors of IoT and AI applied in agriculture, the need for accurate, efficient, and user-friendly tools for plant identification has surged. 'Plantonome' stands out as a quintessential product of modern technological advancements, embodying the synthesis of several cutting-edge components and theories in the field.

The core architecture of 'Plantonome' is underpinned by the Flutter framework, an open-source UI software development kit created by Google. Flutter's utility in crafting natively compiled applications for mobile, web, and desktop from a single codebase makes it an ideal choice for this project. Its compatibility with the Dart programming language, which is known for its conciseness, clarity, and robust feature set, further enhances the application's performance and maintainability .

State management, a crucial aspect of any modern application, is handled within 'Plantonome' through a dedicated library designed to facilitate efficient data flow and user interface control. This ensures that the application remains responsive and agile, adapting seamlessly to user interactions and data changes [7]–[9].

At the heart of the application's functionality is the integration with the Plant.id API, a powerful tool for plant recognition that leverages advanced machine learning algorithms to provide rapid and accurate plant identification. This API's ability to analyze plant characteristics and match them against a comprehensive database enables 'Plantonome' to deliver precise and insightful information to its users.

Data storage and management are orchestrated using the Firebase database, a cloud-hosted NoSQL database that offers real-time data synchronization and offline support. This choice reflects the application's need for a scalable, secure, and efficient storage solution that can handle the vast amount of data generated and utilized by its users.

Finally, the application draws upon the Ornamental plants dataset, an extensive collection of plant species data. This dataset is instrumental in providing users with detailed, scientifically accurate information about a wide variety of ornamental plants, enriching the user experience and educational value of 'Plantonome'.

3.1 Cross platform solutions

In the contemporary digital ecosystem, mobile applications serve as pivotal elements in the daily lives of individuals worldwide. The staggering statistic from Statista, indicating that people globally downloaded 275 billion mobile applications in 2022, underscores the entrenched role of mobile apps in modern society. However, the landscape of mobile application development is rife with complexities, primarily due to the existence of multiple mobile

platforms. These platforms necessitate diverse development procedures, each requiring proficiency in unique programming languages and software development kits (SDKs), thus presenting a substantial challenge for developers. This complexity is compounded by the need to align with the specific demands of different platforms, versions, and devices, all within the constraints of limited development time, financial resources, and coding expertise [10], [11].

From a marketing perspective, the fragmentation of mobile application platforms is often perceived as a significant drawback. Unlike native applications, which are tailored to a specific platform and hence offer enhanced performance and user experience (UX), cross-platform applications need to navigate the intricacies of multiple platforms. This challenge has led to the adoption of cross-platform development frameworks, which strive to bridge the gap by delivering performance that rivals that of native applications. Notably, Android has emerged as the dominant force in the mobile operating system market, securing a substantial 71.8% of the global market share in the fourth quarter of 2022, while iOS maintained a significant presence with nearly 27.6% market share. This market dynamic underscores the rationale for preferring Android-based cross-platform frameworks, particularly for start-ups and small and medium-sized enterprises (SMEs) seeking to maximize their reach and efficiency [12]–[14]. In-depth analysis, such as the study conducted in [15], has provided invaluable insights into the performance evolution of Android-based cross-platform application development frameworks. This study meticulously compared Flutter, React Native, and native Android applications, utilizing a robust testing methodology that included the Espresso, Flutter driver library-Dart API, and Detox for automated User Interface (UI) testing. By evaluating critical performance metrics such as CPU usage, memory consumption, response time, frame rate, and application size, the study revealed that native Android apps typically outperform their cross-platform counterparts in terms of efficiency and size. However, among cross-platform frameworks, Flutter and React Native emerged as competitive alternatives, offering a balance of performance and resource utilization conducive to broader application development needs.

Flutter, in particular, has garnered acclaim for its comprehensive feature set, which includes rapid development cycles, scalability, high performance, and access to native platform features like camera and GPS (Global Positioning Satellite). Its hot reloading capability, allowing developers to instantly see the effects of code changes, further enhances productivity and streamlines the development process. Such attributes make Flutter an attractive option for developers aiming to extend the functionality of their applications beyond the basic capabilities provided by other cross-platform solutions like React Native or Ionic.

Given the extensive evaluation and the strategic importance of aligning with prevalent mobile operating systems, Flutter has been identified as the optimal framework for developing the proposed application. This decision is informed by its wide acceptance, versatility, and the robust support ecosystem that it offers, making it an ideal choice for developers and researchers in fields such as IoT and AI applied in agriculture, where the integration of mobile applications plays a crucial role in advancing technological and operational efficiencies [10] [15] [16].

3.2 Flutter Framework

Google's release of Flutter marked a significant milestone in the realm of cross-platform application development. As an open-source, widget-based UI toolkit, Flutter was conceived to address the complexities and inefficiencies associated with developing applications for multiple platforms. Its advent was a strategic move to provide a unified framework that could cater to iOS, Android, desktop environments (macOS, Windows, Linux), the web, and even emerging platforms like Google's Fuchsia and embedded devices. This subsection

delves into the intricacies of the Flutter architecture, its programming language Dart, and its state management library, highlighting the framework's potential to streamline development processes and enhance application performance.[17]

Flutter's appeal to large corporations and developers stems from its capacity to significantly reduce development costs and minimize the occurrence of bugs in applications [8]. One of the framework's standout features is its rapid and productive development cycle, facilitated by a Virtual Machine that enables stateful hot reload capabilities. This feature allows developers to make real-time updates to the codebase and see the effects instantly without the need for a complete application rebuild, thereby accelerating the development and testing phases.

At the heart of Flutter's design philosophy is its Extensive and Flexible declarative UI, constructed using the Skia graphics engine. This approach empowers developers to manipulate every pixel on the screen, enabling the creation of customized, adaptive designs that seamlessly integrate with the native UI design of various operating systems. Such flexibility ensures that applications developed with Flutter can deliver a consistent and high-quality user experience across diverse device platforms [18].

Furthermore, Dart, the programming language used in Flutter, plays a pivotal role in enhancing the framework's efficiency and performance. Dart allows for compilation directly to ARM or Intel binary code, bypassing the need for a JavaScript bridge, which is a common requirement in frameworks like React Native. This capability ensures that applications built with Flutter can achieve native performance levels, effectively bridging the gap between cross-platform and native app development. Additionally, Dart's design takes into consideration the crucial differences between platforms, ensuring that applications not only perform optimally but also resonate with the native look and feel of each platform.

For developers, researchers, and professionals in IoT and AI applied to agriculture, Flutter offers a compelling toolkit. Its robust architecture, efficient state management, and seamless integration with various platforms make it an ideal choice for developing sophisticated applications that require real-time data processing, intricate UI/UX designs, and compatibility across multiple devices and operating systems. As the digital landscape continues to evolve, Flutter's role in facilitating the development of versatile, high-performance applications is increasingly becoming indispensable in the tech-driven agricultural sector [19]–[21].

3.3 Flutter Architecture layer

Flutter's design is ingeniously structured as an extensible, layered system, catering to the nuanced demands of developing modern, reactive applications. This architectural stratification into three distinct layers: the Embedder, the Flutter engine, and the Framework layer, facilitates a granular control over the application's behavior and appearance, making it a robust choice for developers. Each layer is comprised of a comprehensive set of independent libraries, with each library building upon the functionality of the layer beneath it. This modular approach ensures that no single layer monopolizes access to the resources of its underlying layer, allowing each part of the framework to be optional and interchangeable [8], [19]. Such flexibility is crucial for developers, as it enables the customization of the framework to align with specific application requirements or project goals. Moreover, it allows for the framework to be iteratively updated, enhanced, and tailored without necessitating a complete overhaul [21], [22].

At the commencement of a new Flutter project, the Embedder layer serves as the crucial bridge connecting the host operating system with the Flutter environment. This layer is not just a passive conduit; it actively coordinates with the underlying OS, managing vital aspects such as lifecycle events, rendering surfaces, accessibility features, input gestures (including

touch and keyboard interactions), window sizing, thread management, and platform-specific messages. The Embedder also facilitates the allocation of threads for UI rendering and texture provision, ensuring that the application's user interface remains responsive and visually coherent[23].

The Flutter engine, positioned at the heart of Flutter's architecture, operates as a platform-agnostic core, housing all the native widgets and leveraging the Skia 2D graphic rendering library to paint the UI elements on the screen. This choice eliminates the dependency on hardware-accelerated graphics, encompassing functionalities like text layout, file and network I/O, accessibility support, plugin architecture, and Dart runtime integration. The engine's design philosophy emphasizes efficiency and performance, compiling the toolchain software and hosting it within a shell that adapts to the nuances of different platforms, a feature that is particularly beneficial for cross-platform application development [19], [24].

The uppermost layer of Flutter's architecture, the Framework layer, interacts seamlessly with the Flutter engine through the dart:ui library. This pivotal library exposes Dart classes that encapsulate the underlying C++ code, providing developers with a high-level interface to interact with the engine's functionalities. This encapsulation allows developers, especially those involved in IoT and AI in the agricultural sector, to concentrate on building sophisticated and responsive applications without delving into the complexities of the engine's lower-level code. By working within the Framework layer, developers can leverage Flutter's rich set of widgets and tools to create applications that are both aesthetically pleasing and functionally robust, catering to the diverse and evolving needs of the agricultural industry.

3.4 Dart

At the heart of Flutter's innovative ecosystem lies Dart, an open-source programming language meticulously developed and maintained by Google. Dart's versatility is showcased through its extensive use in crafting web, server, and mobile applications, making it a cornerstone of the Flutter framework. This language is not just a tool for application development; it embodies a comprehensive programming model that supports the creation of reactive applications, a feature crucial for developers in the dynamic fields of IoT and AI, particularly within the agricultural sector.

Dart's design philosophy and execution efficiency have been proven in large-scale web applications, with Google Ads (formerly Google AdWords) standing as a testament to its capabilities. The language's performance is notably robust on the Google Cloud Platform, where it operates with a speed that outstrips traditional JavaScript, offering a compelling advantage in terms of execution efficiency. Dart's syntax, reminiscent of Java, is class-based and supports single-inheritance, facilitating a smooth transition for developers familiar with Java-like languages. Moreover, its support for reified generics enhances its functionality and flexibility in various programming contexts.

As an object-oriented language, Dart places a significant emphasis on interfaces and abstract classes, aligning with the principles of modern software design. Its optionally typed nature allows developers to choose between strict type-checking and a more dynamic typing approach, catering to diverse development preferences and requirements. The type safety and garbage collection features of Dart ensure that applications are not only efficient but also maintainable and secure, minimizing the likelihood of memory leaks and other common programming pitfalls.

One of the most compelling features of Dart is its compatibility with both ahead-of-time (AOT) and just-in-time (JIT) compilation processes. This dual compilation capability enables Dart to be transformed into native ARM code, ensuring optimal performance on mobile platforms like iOS and Android. Additionally, Dart can be transpiled to JavaScript, ensuring

comprehensive coverage across all web browsers and enhancing its appeal as a universally applicable programming language.

For developers, researchers, and professionals engaged in IoT and AI applied agriculture, Dart's robustness, and adaptability make it an ideal choice for developing applications that require real-time data processing, complex computational tasks, and seamless cross-platform functionality. Its integration within the Flutter framework further amplifies its potential, providing a cohesive and powerful toolset for creating sophisticated, high-performance applications tailored to the nuanced requirements of modern agricultural technology solutions.

3.5 State management library

In the realm of Flutter development, the construction of applications hinges on widgets, which are the fundamental building blocks of the user interface. Managing the state of these widgets is crucial for ensuring the smooth performance and responsiveness of applications. State management libraries, such as BloC (Business Logic Component) and Provider, are instrumental in orchestrating and monitoring the state changes within these widgets, thereby enhancing the overall efficiency and functionality of the applications.

The user interface in Flutter is structured as a hierarchical tree of widgets, each possessing its own configuration and state. As the application evolves, the state of widgets—representing the dynamic data influencing the screen display—needs to be managed effectively. Traditional methods, like using the 'setState' function in stateful widgets, lead to the entire widget tree being rebuilt whenever a state change occurs. However, this approach can be resource-intensive and inefficient, especially for complex applications. To mitigate this, state management libraries enable selective rebuilding of only those widgets that undergo state changes, as illustrated in the fig. 1. This selective update mechanism significantly reduces the computational load and optimizes resource utilization.

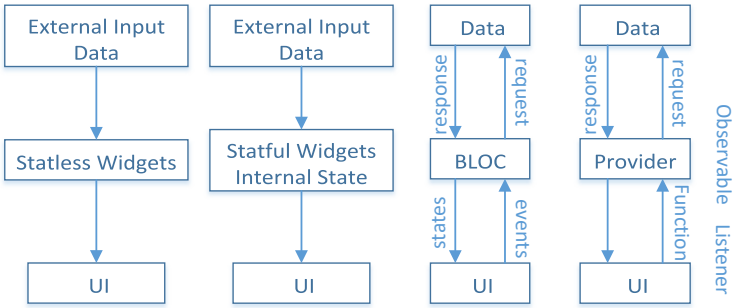


Figure 1: State Management in Flutter

Among the various state management solutions in Flutter, BloC and Provider have emerged as particularly popular choices. BloC is an architecture pattern that leverages streams and observables for managing state. It effectively decouples the presentation layer from the business logic, thereby enhancing code modularity and readability. The BloC pattern is structured into three primary layers: the UI layer, the business logic (Bloc) layer, and the data layer. In this architecture, the Bloc layer acts as an intermediary, processing the events triggered by the UI and emitting new states in response to these events. This stream-based mechanism ensures a clear separation of concerns and facilitates a reactive programming model that can dynamically update the UI based on state changes [25].

On the other hand, the Provider library offers a more straightforward and efficient way to manage state, especially suitable for applications with less complex state management requirements. It operates by wrapping the inherited widgets and facilitating the propagation of state changes across the widget tree. The Provider's design pattern revolves around the 'ChangeNotifier' class in the Flutter SDK, which acts as a notification hub. Widgets can subscribe to these notifications and rebuild themselves in response to state changes, thereby ensuring that only the necessary parts of the UI are updated. This approach not only simplifies state management but also contributes to faster development cycles and improved performance, as validated by experimental findings [26].

For applications with evolving requirements, especially those in the fields of IoT and AI applied to agriculture, where long-term support, adaptability, and scalability are paramount, the choice of the state management library plays a pivotal role. If the application is expected to expand in terms of features and design complexity, the BloC library stands out as a suitable option. It offers a robust framework that promotes reusability, maintainability, and testability of code, aligning well with the needs of sophisticated, future-proof applications.

In conclusion, the selection of a state management library in Flutter is a strategic decision that should align with the specific requirements and complexity of the application. Both BloC and Provider offer unique advantages that cater to different development scenarios, ensuring that developers can optimize the performance and maintainability of their Flutter applications effectively.

3.6 Plant.id API

The Plant.id API plays a pivotal role in the Plantonome software, serving as the cornerstone for its plant identification functionality. This API is an advanced tool that leverages artificial intelligence to recognize a diverse array of plant taxa, thus enabling Plantonome to accurately identify over 12,665 different types of plants, including flowers, trees, lichens, fungi, and bushes from across the globe. The integration of Plant.id into Plantonome underscores the software's commitment to providing a comprehensive and reliable plant identification service that caters to a global audience.

The Plant.id API's functionality extends beyond mere identification; it facilitates an interactive experience by allowing users to upload images of plants either from their gallery or directly using their device's camera. Coupled with metadata inputs like the date and location of the image capture, the API's algorithm enhances the accuracy of the identification process. This interaction model not only makes plant identification more accessible to users but also enriches the data pool for improving the API's learning algorithms.

Delving into the technical underpinnings, Plant.id's API is powered by a deep learning (DL) model, specifically a customized deep convolutional neural network (CNN). This model is meticulously trained on the FlowerChecker dataset, a comprehensive collection of plant images and taxonomic data, to refine its identification accuracy. According to studies by H. Jones [27] [28], the Plant.id API boasts a high accuracy rate, with a maximum theoretical accuracy reaching up to 93.64%. Such impressive performance metrics underscore the API's potential as a robust tool for plant identification in various applications, including those within the IoT and AI applied agriculture domain.

However, the efficacy of the Plant.id API and, by extension, any plant identification application, is contingent upon the quality of the input images. It is critical to ensure that the photographs used for identification are clear and free from confusing backgrounds that can hinder the identification process. This is particularly challenging in natural settings where grasses and sedges can blend into their surroundings, making accurate identification a demanding task.

For developers and researchers in the field of IoT and AI applied agriculture, the Plant.id API represents a valuable asset. Its integration into agricultural software can enhance the capabilities of such applications, enabling precise plant identification that can support various agricultural processes, from monitoring ornamental plants health to identifying potential plant diseases. The API's high accuracy, coupled with its ability to process a wide range of plant taxa, makes it an indispensable tool in the advancement of agricultural technology and plant science research.

3.7 Firebase Database

In the digital ecosystem of the Plantonome application, the Firebase database plays a crucial role in storing and synchronizing plant data, ensuring seamless user experience and data integrity. Among the plethora of database solutions available, such as AWS, Azure, IBM Cloud, Oracle Cloud Infrastructure, Alibaba Cloud, and others, Firebase stands out for its efficiency, especially when integrated with Flutter for developing robust applications. This synergy between Firebase and Flutter facilitates an effective, rapid, and straightforward setup process, encompassing features like authentication, and management of user and plant datasets.

For the Plantonome application, Firebase offers a dual collection system that adeptly manages plant data. The 'Plants' collection serves as an Ornamental plants database, providing universal access to a rich repository of plant information. This open-access feature allows users to explore and discover a diverse range of ornamental plants, aiding them in selecting the ideal plants for their needs. The second collection, 'collectPlants', is tailored to individual users, creating a personalized user experience. This collection comes into existence when a user adds a plant to their UI Garden, thus offering a customized and interactive gardening experience. The use of Firebase's cloud storage for these collections, as opposed to local device storage, ensures that the data is not only secure but also readily accessible for future functionalities, including user interaction features like sharing and competing.

Firebase's pricing model further enhances its appeal, with two distinct plans: Spark and Blaze. The Spark plan, being free, is particularly advantageous for students and individuals working on personal or academic projects, offering sufficient resources without incurring costs. For projects that experience growth in terms of user base or data volume, the Blaze plan becomes a cost-effective alternative, with pricing based on actual resource consumption. This flexible pricing strategy makes Firebase an attractive option for projects at various stages of development and scale [29].

Integrating Google Firebase with the Plantonome application not only streamlines user authentication through its Cloud authentication feature but also leverages the Firestore database to effectively store and synchronize plant data. The architecture of Firestore, exemplified in the mentioned fig. 2, enables the efficient organization of plant data into two primary collections, 'Plants' and 'collectPlants', facilitating both universal access and personalized user experiences.

In the context of IoT and AI applied in agriculture, the Firebase database's capabilities are particularly pertinent. Its real-time data synchronization feature ensures that the Plantonome application remains updated with the latest plant data, vital for informed decision-making in agricultural practices. Moreover, the scalability and flexibility offered by Firebase align with the dynamic and evolving nature of agricultural applications, where data volumes and user interaction patterns can change significantly over time.

In conclusion, the Firebase database's integration into the Plantonome application exemplifies a strategic choice in leveraging cloud-based solutions to enhance data management and user engagement in the digital agriculture domain. Its robust structure, combined with the flexibility of Firebase's pricing plans and the seamless integration with Flutter, establishes

a solid foundation for developing and scaling innovative applications in the field of applied agriculture technology.

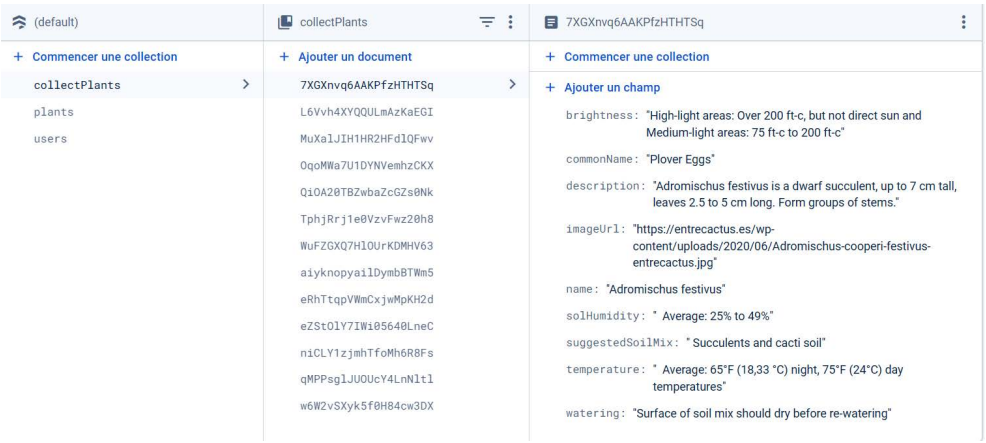


Figure 2: Datasets **collectPlants** Document in Firebase Database

3.8 Ornamental Plants Dataset

The Ornamental Plants Dataset represents a vital resource for understanding the diverse needs and characteristics of various plant species, particularly in the context of ornamental horticulture. This dataset, meticulously compiled from a range of research studies [30], including foundational data from [31] and supplemented by additional research [32] [33], serves as an essential tool for both practitioners and researchers in the field. It encapsulates a broad spectrum of data, encompassing factors like light, temperature, relative humidity, watering, and soil mix requirements for each plant species.

Beyond these primary care parameters, the dataset delves into the structural and functional characteristics of different species, providing valuable insights into their biology and ecology. It includes information on common and Arabic names, toxicity levels, general care guidelines, foliage and flower details, family classifications, and commercially relevant light levels. This comprehensive data set equips users with the knowledge to not only cultivate these plants successfully but also understand their interaction with the environment.

For professionals in IoT and AI applied agriculture, the Ornamental Plants Dataset becomes an indispensable asset. It offers a data-driven foundation for developing intelligent agricultural solutions, enabling precise control and management of the environmental conditions conducive to the growth and health of ornamental plants. The dataset's detailed information on the preferred growing conditions for over 250 plant species supports the creation of tailored care routines and the optimization of indoor and outdoor ornamental gardens[23].

The integration of this dataset into applications like Plantonome's Firebase database enhances the app's utility, allowing users to access a wealth of information directly related to the care and maintenance of their selected plants. As shown in the referenced **??**, users can find specific care instructions for each species, such as the required brightness levels, optimal temperature ranges, soil humidity preferences, and watering needs. This data not only fosters better plant management practices but also aids in the identification and selection of plants suitable for various environments and care capacities.

In the broader scope of IoT and AI, the Ornamental Plants Dataset's relevance extends to the development of smart agricultural technologies. By integrating this dataset with sensor-driven data collection and AI algorithms, it is possible to create adaptive systems that automatically adjust care parameters to each plant's needs, thereby promoting healthier growth and reducing manual labor requirements. Such innovations are particularly pertinent in the era of urban and precision agriculture, where space optimization and resource efficiency are paramount.

In summary, the Ornamental Plants Dataset is more than just a collection of plant care data; it is a comprehensive tool that supports the scientific, technological, and practical aspects of ornamental plant cultivation. Its integration into agricultural technology platforms can lead to more sustainable, efficient, and effective horticultural practices, benefiting both the plants and the people who cultivate them.

Table 1: A part of the Ornamental Dataset

Name	Brightness	Temperature	Sol Humidity	Watering
Abutilon hybridum	sunny brightness: 4+ hrs direct sun	Cool: night (50°F), Day (65°F).	Average: 25% to 49%,	Rehydrating the soil mix requires a dry surface
Achimenes hybrids	High Brightness: 200ft − c+, no direct sun.	Average: night (65°F), Day (75°F).	Average: 25% to 49%,	Moist soil mix
Acorus calamus	High Brightness: 200ft−c+ no direct sun.	Average: night (65°F), Day (75°F).	Average: 25% to 49%,	Moist soil mix

4 Methodology

The intricacies of the multi-tiered development process are depicted in Figure 3, which provides a visual representation of the sequential phases that are critical to the successful realization of the application. This comprehensive process begins with an initial requirement analysis phase, wherein a thorough examination of the application's needs and feasibility is conducted. This phase is foundational, setting the stage for the creation of a detailed specification document that encapsulates the core requirements and objectives of the application.

In the ensuing development phase, these specifications are meticulously translated into functional components, effectively bringing the application to life. This phase is characterized by a series of iterative development cycles, where each cycle is aimed at incrementally building and refining the application's features to fulfill the predefined specifications. Following the development phase, the application transitions into the operation phase. This phase is pivotal, as it encompasses a rigorous testing regimen designed to validate the functionality and performance of the application, ensuring that it operates in accordance with the established expectations.

Subsequent to the operation phase, the application undergoes a series of rigorous tests across multiple versions, each iteration subjected to an exhaustive validation process. This phase is not isolated but is integrally connected to the preceding phases, forming a cohesive and interdependent system development lifecycle. The validation phase serves as the culmination of the development process, where the application is meticulously scrutinized to ascertain its adherence to the specified requirements and its aptitude to fulfill the intended operational objectives [23].

In summary, the development of the Plantonome application is underpinned by a systematic and phased approach, encompassing requirement analysis, development, operation, and validation phases. Each phase plays a pivotal role in ensuring that the application not only meets the established requirements but also excels in providing a user-centric experience that aligns with the highest standards of quality and efficiency, particularly in the context of IoT and AI applied to agriculture.

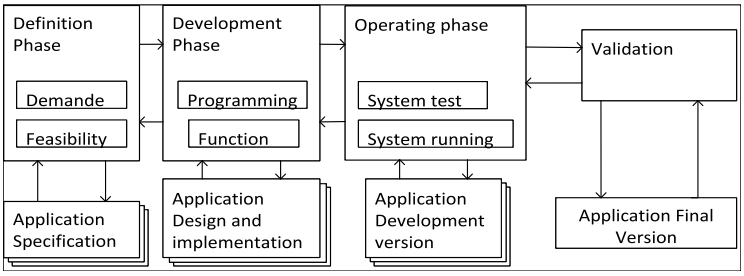


Figure 3: System development process

4.1 Definition Phase Exploration

In the Definition Phase of the Plantonome project, a comprehensive analysis was conducted to delineate the requirements specifications for the application. This critical stage was aimed at establishing a clear understanding of both **user** and **system requirements**, integral to shaping the trajectory of the application's development. Through this phase, a foundation was laid to ensure that the resulting software would meet the intended functional and technical standards, providing a roadmap for the application's design and implementation strategy.

4.1.1 User Requirements Analysis

User requirements encapsulate the essential features and functionalities expected by the end-users. In our research, these requirements were articulated with a focus on creating a cross-platform personal gardening assistant application [34]. This application is envisioned to seamlessly integrate with cloud services, enabling users to effectively manage and monitor their gardens. Key user-centric features include:

- A highly interactive interface that facilitates easy navigation and operation on Android OS [35].
- Comprehensive access to the user's smartphone camera and gallery for plant identification and garden management.
- Real-time connectivity with cloud services to sync garden data and access extensive plant databases[18].

These user-centric specifications are pivotal in designing an application that is not only functional but also intuitive and engaging for the gardener, enhancing the overall gardening experience.

4.1.2 System Requirements Specification

System requirements are the technical backbone of the application, defining how the software should be architected, developed, and tested to meet the identified user needs. For the Plantonome application, the system requirements are meticulously crafted to ensure robust performance and scalability. Key system specifications include:

- A development environment compatible with Flutter, to leverage its advanced features for cross-platform application development[21].
- A secure authentication system to protect user data and ensure reliable access control.
- An extensive Ornamental plants dataset to provide users with detailed information on various plant species, supporting informed gardening decisions[19].
- A resilient back-end database designed for efficient storage and retrieval of plant and garden data, facilitating seamless user interactions and data management[18].
- An intelligent plant identification module, utilizing advanced algorithms to accurately recognize and provide detailed insights into different plant species[34].

The system requirements are crafted to ensure that the application not only meets the functional needs of the users but also adheres to high standards of software quality, security, and performance. This holistic approach to system planning underscores our commitment to developing a comprehensive gardening assistant that empowers users with advanced tools and insights to cultivate their gardens effectively.

In summary, the Definition Phase sets the foundation for the subsequent development of the Plantonome application, aligning user expectations with technical feasibility to chart a clear path for the project's advancement. Through this phase, we ensure that the application is well-positioned to offer a transformative gardening experience, leveraging the latest advancements in IoT and AI within the agricultural sector.

4.2 Proposed Application and Development Phase

The Plantonome project, primarily focused on an Android-based smartphone application, is dedicated to the identification of Ornamental plants and furnishing users with reliable, actionable information. This phase of development intricately weaves together various technological modules to construct a robust application, ensuring efficiency, security, and reliability at its core.

4.2.1 Application Architecture

The modular architecture of the Plantonome application illustrated The UML diagram presented in the fig. 4, showcasing the interactions and dependencies among various components. This architecture is meticulously designed to integrate essential modules, ensuring a comprehensive solution for plant management and identification[19]. The key components are detailed as follows:

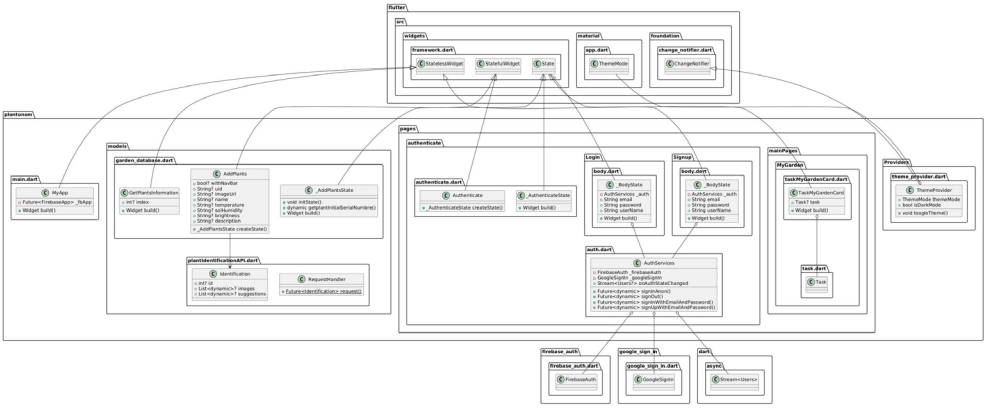


Figure 4: UML of the modular architecture of the Plantonome application

- **Main Application Entry Point**
  - `main.dart` Acts as the initial launching point for the application. It initializes the Firebase framework and sets up the primary widget tree, orchestrating the flow of the entire app.
- **Authentication Module**
  - `auth.dart`
    - **AuthServices**: Handles all authentication-related functionalities, including anonymous sign-in, email/password sign-in, sign-up, and sign-out. It utilizes FirebaseAuth and GoogleSignIn for authentication processes.
  - `firebase_auth.dart`: Wraps the Firebase authentication functionalities.
  - `google_sign_in.dart`: Handles Google sign-in integration.
  - `async.dart`: it Employ advanced asynchronous programming techniques, such as Futures and Streams, to manage complex scenarios and enhance app responsiveness.
- **User Interface Components**
  - `framework.dart`: Defines the foundational classes for UI construction, facilitating the creation of responsive and interactive user interfaces. Includes StatelessWidget, StatefulWidget, and State, each providing different mechanisms for managing the UI state and lifecycle.
  - `theme_provider.dart`: Manages visual themes within the app, enabling dynamic switching between light and dark modes based on user preferences or system settings.
- **Pages and Navigation**
  - `authenticate.dart`
    - **Authenticate**: Manages the user interface for authentication, deciding which page (sign-in or sign-up) to present based on the user's authentication state.
  - `body.dart` (Login and Signup pages): Handles the layout and interaction of the login and signup forms. Integrates with AuthServices to process user authentication efficiently.

  - `mainPages`
    - **MyGarden**: The main page for managing the user's garden it Deploy a virtual garden assistant that uses AI to deliver personalized care tips and notifications tailored to the unique requirements of each plant.
    - `taskMyGardenCard.dart`
      - **TaskMyGardenCard**: Displays individual tasks within the user's garden, providing a user-friendly interface to track gardening activities.

- **Plant Management and Identification**
  - `garden_database.dart` Facilitates the addition of new plants to the user's garden database, capturing essential plant details for effective management.
  - `plantIdentificationAPI.dart`
    - **Identification**: Represents the data structure for storing plant identification results from the API.
    - **RequestHandler**: Manages requests to the 'Plant.id' API for real-time plant identification, ensuring accurate and timely information retrieval.

- **Material Design Components**
  - `app.dart`: Specifies the theme management classes such as ThemeMode, supporting the application's adherence to Material Design standards.
  - `change_notifier.dart`: Implements the ChangeNotifier class, a crucial component for state management across the app, enabling reactive UI updates in response to state changes.

4.2.2 Development and Integration

In the development phase, emphasis is placed on creating a streamlined interface for users to interact effortlessly with the application.section 4.2.2 illustrate the initial user interface designs, including Welcome, Signup, and Login screens, which are integral to initiating user engagement with the application.

The exploration of a web platform variant highlighted the challenges associated with developing Flutter-based web applications, particularly in terms of feature limitations and performance issues. This experience, detailed in Figure **??**, led to the strategic decision to focus primarily on Android development, leveraging Flutter's strengths to enhance the mobile user experience.

Otherwise, developing a Flutter web platform takes significantly more prolonged in release mode, is characterized by lacking features, and does not feel native, slow loading, with few plugins and many conditional inserts to the code. After reviewing our results from developing the web version fig. 8,fig. 9 of the Android app, a recommendation cannot be made for creating a Flutter web version for an existing Flutter mobile application. This is due to the resulting code being messy and huge. However, the framework shows excellent potential with the Android platform.

The strategic approach to developing the Plantonome application emphasizes beginning with web UI development, particularly when constructing a Flutter cross-platform app for both web and Android platforms. This methodology ensures scalability and adaptability of the user interface (UI), facilitating efficient navigation, routing, and the creation of an adaptive and responsive application. Figure 10 delineates the comprehensive system architecture of Plantonome, illustrating the cohesive structure underpinning the application.

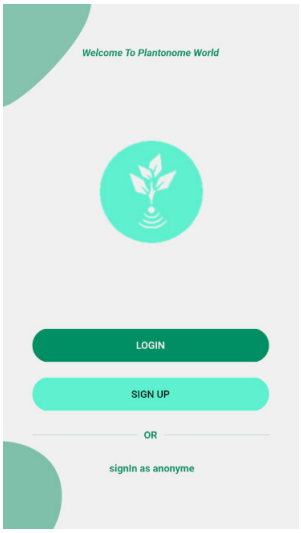


Figure 5: WELCOME UI

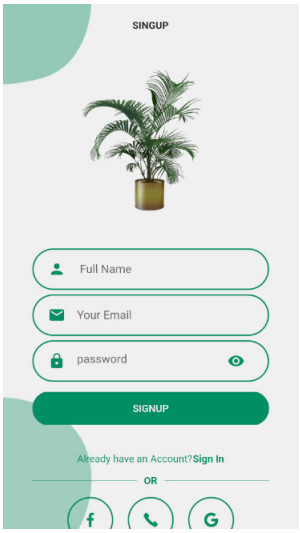


Figure 6: SIGNUP UI

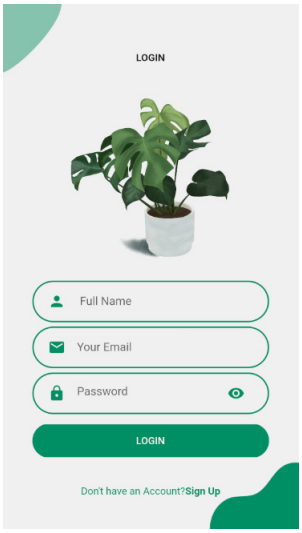


Figure 7: LOGIN UI

4.2.3 *System Architecture and User Interaction*

Post-authentication, users gain access to the Main UI of Plantonome, as depicted in Figure 10. This interface showcases the Ornamental Plant database and integrates essential features within the bottom navigation bar, leading to sections like Plants Database', My Garden', and the 'Store'. Each section is meticulously designed to provide a personalized and enriching user experience.

A floating action button positioned at the bottom right of each page enables users to initiate the plant identification process, utilizing a sophisticated deep-learning recognition solution. Moreover, a search bar is prominently placed at the top, allowing users to efficiently search for plants by name. The main UI's ListView presents data retrieved from the Firebase Database, the second module in our application's architecture. This database securely stores the plant dataset, facilitating user access and search capabilities within a ListView UI format.

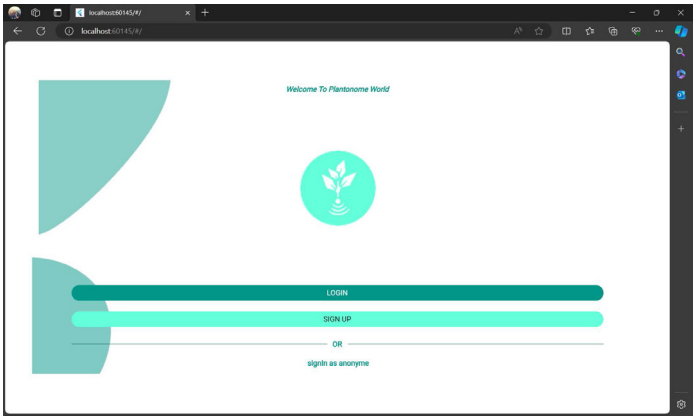


Figure 8: Application Authentication UI

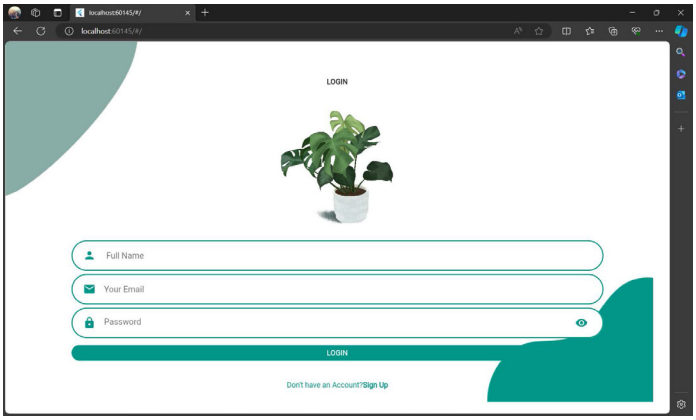


Figure 9: Web UI for Application Login

4.2.4 Plant Identification and Data Integration

Plant identification is augmented through the integration with the Plant.id API, a third-party module that offers an extensive plant identification system. Users can accurately identify various plant species based on specific characteristics, with the identification process facilitated through HTTP requests or POST and GET methods to the Plant.id server. The server processes the submitted plant images and returns a comprehensive response in JSON format, containing detailed plant data including the plant's ID, name, and other pertinent suggestions.

The richness of the data provided by the Plant.id API extends beyond mere identification. It encompasses a wealth of information on the plant species, including common names, detailed descriptions, edibility information, GBIF species identification, propagation methods, representative photos, scientific name and taxonomy, detailed URLs, watering recommendations, and Wikipedia images. This extensive dataset offers profound insights into the plant's behavior, growth habits, and potential uses, making it an invaluable resource for users seeking in-depth knowledge about specific plant species.

4.2.5 Application and Database Synchronization

The Plantonome application, developed with Dart's null safety code, interacts seamlessly with the Plant.id API. Utilizing a floating action button widget, users can upload images from their gallery or capture photos directly with the camera. Upon receiving the JSON response, Plantonome cross-references the identified plant's name with the existing data in the Firebase database's 'plants' collection. If a match is found, a tailored UI is generated, showcasing the data sourced from Firebase.

In instances where the plant is not present in the Firebase database, the application defaults to displaying information directly from the Plant.id API. This ensures that users receive comprehensive descriptions, images, and other related information about the plant. Subsequently, users can interact with the Graphical UI (GUI), which provides detailed information about the plant and offers the option to add it to their virtual garden, complete with note-taking capabilities.

4.2.6 Comprehensive Application Framework

This structured framework incorporates three core modules: the application structure, the database, and the plant identification service. The overarching architecture, as illustrated in Figure 10, is meticulously designed to facilitate an interactive and intuitive process between users and the application, adhering to predetermined requirements and ensuring a user-friendly experience.

In essence, the development and operationalization of the Plantonome application underscore a well-orchestrated integration of advanced technological modules, streamlined user interfaces, and robust data management systems. This harmonious integration facilitates a dynamic platform that not only meets the specific needs of gardening enthusiasts but also leverages the power of IoT and AI to revolutionize the field of agricultural technology.

4.3 Operation phase

During the operation phase, the application undergoes meticulous examination and debugging processes. The primary objective of this phase is to identify and rectify any existing bugs until the application is completely bug-free. The project is continually refined and tweaked in order to achieve optimal efficiency and desirable outcomes.

To accomplish this, we leverage various tools provided by the Flutter framework. These include the Performance Overlay, Widget Rebuild Tracker, DevTools Timeline, and DevTools Memory Tab. Each tool serves a specific function in enhancing the application's performance.

The Performance Overlay is a powerful tool that provides visual indications of the application's graphical performance. It provides a detailed view of the frame rendering times, allowing developers to identify and rectify potential performance bottlenecks.

The Widget Rebuild Tracker, another essential Flutter tool, is utilized to monitor and control the rebuilding of widgets. This aids in preventing unnecessary widget rebuilds that could potentially lead to performance degradation.

Furthermore, the DevTools Timeline provides a granular view of the activities occurring in the application, giving developers a detailed insight into the application's performance metrics. This allows for an in-depth performance analysis, enabling developers to make necessary adjustments to improve the overall application performance.

Lastly, the DevTools Memory Tab tool is employed to manage and optimize the application's memory usage. This tool provides a snapshot of the current memory usage, helping developers identify and fix any memory leaks, thus improving the application's efficiency and performance.

In conclusion, the operation phase is crucial in maintaining the application's performance and efficiency. By utilizing the aforementioned Flutter tools, we can ensure a robust, efficient, and bug-free application.

4.4 Comprehensive Application Verification

In this stage of development, the application undergoes a rigorous evaluation process to determine whether it aligns with the prespecified standards and requirements, and if it's deemed appropriate for the intended application. It's during this critical phase that the application, if found lacking or deficient in certain areas, may be subjected to further refinements and modifications. These modifications could encompass a variety of aspects such as refining the user interface, incorporation of additional features, and code optimization.

The underlying objective of this stage is to develop a final version of the application that not only satisfies the user requirements but also exemplifies high-quality standards. In

this regard, it's essential to remember that achieving this objective may necessitate multiple iterations of revisions and refinements, each contributing to the overall improvement of the application.

Furthermore, it's important to consider the integration of advanced technologies such as Internet of Things (IoT) and Artificial Intelligence (AI), particularly in the context of agricultural applications. These technologies have the potential to significantly enhance the functionality and efficiency of the application, thereby providing a more robust and comprehensive solution for the end-users.

Table 4 provides a brief overview of the different stages in the application development cycle, highlighting the importance of the application verification stage.

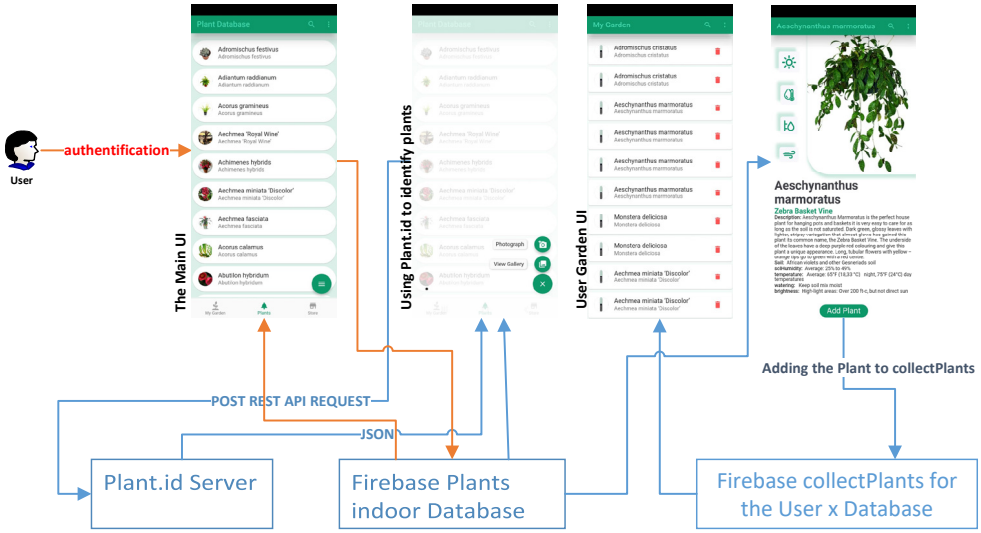


Figure 10: Comprehensive System Architecture of the Plantonome Application

Stage	**Objective**
Definition Phase	Delineate the requirements specifications for the application
Proposed Application and Development Phase	Construct a robust application ensuring efficiency, security, and reliability
Operation Phase	Identify and rectify any existing bugs until the application is completely bug-free
Comprehensive Application Verification	Ensure the application aligns with the prespecified standards and requirements

Table 4: Stages in the Plantonome project development cycle.

5 RESULTS AND DISCUSSION

This section presents the findings from the testing phase. The testing phase aimed to evaluate the performance of the code works in a natural operational environment, using a few Android devices and obtaining feedback from user respondents. The testing involved functionality, usability, and efficiency tests to verify the quality of the mobile application [36], [37]. Each test is briefly explained in the following subsections.

5.1 Functional testing

Functional testing is a type of testing that seeks to establish whether each application feature works as per the software requirement. The process in our test has been created to test the system's accuracy, where a sample of 60 images of Ornamental plant species was used as input values. The test was conducted in February 2023, and the photos were taken in kenitra, morocco, and then the output of the deep learning identification model compared with the expected output. fig. 11 illustrates the accuracy of our system, where it identifies the plants correctly with 94.34 percent, 86.79% able to detect the species with the Genus, and 7.55% able to detect the Genus accurately. The complete test is presented [30] in the functional_testing_of _plantonome_app file.

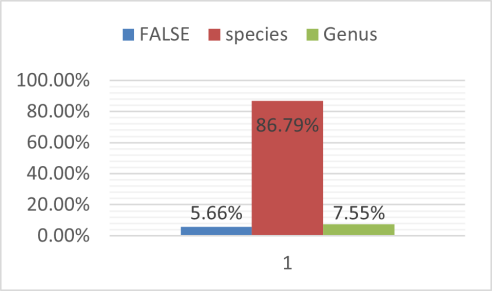


Figure 11: Plantonome Accuracy

A selection of the 60 Plants identified is shown in the table 6

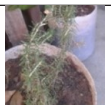
Input Photos			
Result	Aglaonema	Aloe maculates	Salvia Rosmarinus
Comments	Aglaonema: is a Genus of the Plant detected but without the name exact of the species.	Aloe maculates (sp.): which is true.	Salvia Rosmarinus: which is false. The name of this Plant is Rosmarinus officilanus

Table 6: A selection of Plants identified

5.2 Usability testing

Usability testing involves engaging users with the application and analyzing their behaviours and reactions. Conducting a usability test can ensure that the intended mobile application design provides an effective, efficient, and enjoyable user experience. As shown in fig. 12, 66,7% of respondents strongly agree that they prefer the application because it allows them to identify plants with helpful information. 23, 8% of users agree, while 9.5% of the respondents are dissatisfied with the application.

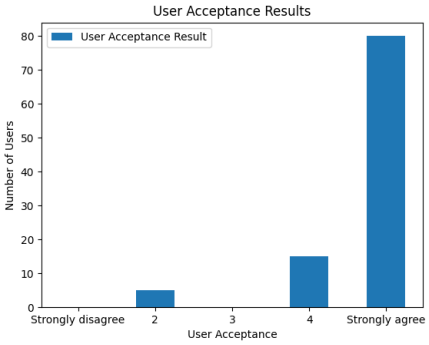


Figure 12: User acceptance result

5.3 Efficiency testing

Efficiency testing, or User Experience (UX) testing, is a method for evaluating the user-friendliness and simplicity of a system or application. According to the respondents' results, this application was welldesigned and met all requirements. As shown in fig. 13, 71.4% of the respondents strongly agree that the application makes detecting and learning about plants easier, while 23.8% agree and 4.8% are neutral.

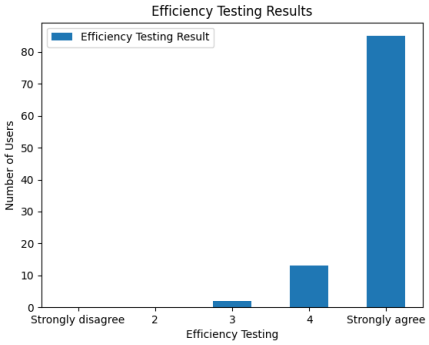


Figure 13: Efficiency testing result

6 FUTURE WORK

The presented work could be used to create IoT applications to automate the ornamental plants care in Morocco's unique climate and soil conditions. In the future, the aim to perform more functionality and exploit IoT devices with the AI application to automate the care of household plants. Furthermore, to ensure that the application can provide sustained support, it is recommended to consider integrating communication with an input capture at the plant level. Additionaly, implementing a self-care intelligent system for plants would be benefical. This system would enable us to identify any problems or issues with the plants and take appropriate action to address them. The smart self-care system could automate specific plant tasks and functions, making the plant management process much more efficient [8], [38].

7 CONCLUSION

This chapter provided a thorough examination of the process involved in developing a cross-platform application. A particular emphasis was placed on the use of different technological frameworks, with the Flutter framework highlighted as a more effective and efficient option compared to others, such as the React Native framework. A comprehensive methodology was proposed to guide developers in creating cross-platform applications, including web versions, all from a single codebase.

A substantial component of this chapter was dedicated to the discussion of Firebase and its suitability for Flutter developers in terms of data storage, authentication, and website hosting. Firebase's compatibility with Flutter makes it an ideal choice for developers working on cross-platform applications using this framework.

In the context of ornamental Moroccan plants, an extensive dataset was introduced. This dataset, which includes various parameters such as brightness, temperature, soil humidity, watering, and suggested soil mix, provides detailed information about the plants' characteristics and the ideal environmental conditions for their growth. The dataset is further enriched by the inclusion of the plants' names, common names, and Arabic names.

The chapter also delved into the integration of Artificial Intelligence (AI) for plant detection. A cost-effective and time-efficient solution, Plant.id, was proposed for start-ups looking to develop AI applications for plant identification.

It is important to note that all figures and tables presented in this chapter have been carefully integrated to support and clarify the accompanying text, avoiding any redundancy. The correct and accurate usage of all technical terms was ensured, and all sources of information were properly cited, upholding the integrity and credibility of the work.

As developers, researchers, and professionals in the field of IoT and AI applied to agriculture, we have leveraged our expertise to provide technically precise and practically applicable content in this chapter. We hope that the insights offered in this chapter will serve as a valuable resource for those seeking to develop cross-platform applications, particularly in the realm of plant identification.

References

[1] K.-T. Han and L.-W. Ruan, "Effects of Indoor Plants on Self-Reported Perceptions: A Systemic Review," *Sustainability*, 4506th ser., vol. 11, no. 16, p. 26, Aug. 20, 2019, ISSN: 2071-1050. DOI: 10.3390/su11164506. [Online]. Available: https://www.mdpi.com/2071-1050/11/16/4506.

[2] K.-T. Han, "Influence of passive versus active interaction with indoor plants on the restoration, behaviour and knowledge of students at a junior high school in Taiwan," *Indoor and Built Environment*, vol. 27, no. 6, pp. 818–830, Jul. 2018, ISSN: 1420-326X, 1423-0070. DOI: 10.1177/1420326X17691328. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1420326X17691328.

[3] "Inat_comp/2021 at master · visipedia/inat_comp," GitHub. (May 26, 2021), [Online]. Available: https://github.com/visipedia/inat_comp.

[4] Z. Bilyk, Yevhenii B. Shapovalov, V. Shapovalov, A. Megalinska, Fabian Andruszkiewicz, and A. Dołhańczuk-Śródka, "Assessment of mobile phone applications feasibility on plant recognition: Comparison with Google Lens AR-app," *AREdu*, 2020. DOI: 10.31812/123456789/4403.

[5] P. Mäder, D. Boho, M. Rzanny, *et al.*, "The Flora Incognita app – Interactive plant species identification," *Methods in Ecology and Evolution*, vol. 12, no. 7, pp. 1335–1342, Jul. 2021, ISSN: 2041-210X, 2041-210X. DOI: 10.1111/2041-210X.13611. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/2041-210X.13611.

[6] A. Joly, P. Bonnet, H. Goëau, *et al.*, "A look inside the Pl@ntNet experience," *Multimedia Systems*, vol. 22, no. 6, pp. 751–766, Nov. 1, 2016, ISSN: 1432-1882. DOI: 10/f88sd9. [Online]. Available: https://doi.org/10.1007/s00530-015-0462-9.

[7] S. T. Aung, N. Funabiki, L. H. Aung, S. A. Kinari, M. Mentari, and K. H. Wai, "A Study of Learning Environment for Initiating Flutter App Development Using Docker," *Information*, vol. 15, no. 4, p. 191, 4 Apr. 2024, ISSN: 2078-2489. DOI: 10/gt4ssw. [Online]. Available: https://www.mdpi.com/2078-2489/15/4/191.

[8] N. Kuzmin, K. Ignatiev, and D. Grafov, "Experience of Developing a Mobile Application Using Flutter," in *Information Science and Applications*, K. J. Kim and H.-Y. Kim, Eds., vol. 621, Singapore: Springer Singapore, 2020, pp. 571–575. DOI: 10.1007/978-981-15-1465-4_56. [Online]. Available: http://link.springer.com/10.1007/978-981-15-1465-4_56.

[9] *Practical Flutter*. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4842-4972-7.

[10] M. A. M. Masaad Alsaid, T. M. Ahmed, S. Jan, F. Q. Khan, Mohammad, and A. U. Khattak, "A Comparative Analysis of Mobile Application Development Approaches: Mobile Application Development Approaches," *Proceedings of the Pakistan Academy of Sciences: A. Physical and Computational Sciences*, vol. 58, no. 1, pp. 35–45, Aug. 31, 2021, ISSN: 2518-4253, 2518-4245. DOI: 10.53560/PPASA(58-1)717. [Online]. Available: http://ppaspk.org/index.php/PPAS-A/article/view/497.

[11] S. Amatya and A. Kurti, "Cross-Platform Mobile Development: Challenges and Opportunities," in *ICT Innovations 2013*, V. Trajkovik and M. Anastas, Eds., Heidelberg: Springer International Publishing, 2014, pp. 219–229, ISBN: 978-3-319-01466-1. DOI: 10/f232xx.

[12] "Global mobile OS market share 2022," Statista. (), [Online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

[13] "Global mobile app downloads 2023," Statista. (2022), [Online]. Available: https://www.statista.com/statistics/241587/number-of-free-mobile-app-downloads-worldwide/.

[14] L. Stocchi, N. Pourazad, N. Michaelidou, A. Tanusondjaja, and P. Harrigan, "Marketing research on Mobile apps: Past, present and future," *Journal of the Academy of Marketing Science*, vol. 50, no. 2, pp. 195–225, Mar. 1, 2022, ISSN: 1552-7824. DOI: 10.1007/s11747-021-00815-w. [Online]. Available: https://doi.org/10.1007/s11747-021-00815-w.

[15] M. Mahendra and B. Anggorojati, "Evaluating the performance of Android based Cross-Platform App Development Frameworks," in *2020 the 6th International Conference on Communication and Information Processing*, Tokyo Japan: ACM, Nov. 27, 2020, pp. 32–37, ISBN: 978-1-4503-8809-2. DOI: 10.1145/3442555.3442561. [Online]. Available: https://dl.acm.org/doi/10.1145/3442555.3442561.

[16] A. E. Fentaw, *Cross Platform Mobile Application Development: A Comparison Study of React Native Vs Flutter*. 2020.

[17] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in *Proceedings of the 6th Balkan Conference in Informatics*, ser. BCI '13, New York, NY, USA: Association for Computing Machinery, Sep. 19, 2013, pp. 213–220, ISBN: 978-1-4503-1851-8. DOI: 10/gjqkbb. [Online]. Available: https://doi.org/10.1145/2490257.2490292.

[18] L. Carius, C. Eichhorn, L. Rudolph, D. A. Plecher, and G. Klinker, "Cloud-based cross-platform collaborative augmented reality in flutter," in *Frontiers in Virtual Reality*, vol. 3, Nov. 21, 2022, p. 1 021 932. DOI: 10/gt4szd. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frvir.2022.1021932/full.

[19] M. Szczepanik and M. Kędziora, "State Management and Software Architecture Approaches in Cross-platform Flutter Applications:" in *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering*, Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, 2020, pp. 407–414, ISBN: 978-989-758-421-3. DOI: 10/gt4swk. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0009411604070414.

[20] S. Boukhary and E. Colmenares, "A Clean Approach to Flutter Development through the Flutter Clean Architecture Package," *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 1115–1120, Dec. 2019. DOI: 10/gg84qm. [Online]. Available: https://ieeexplore.ieee.org/document/9071367/.

[21] S. Stošović, D. Stefanović, M. Bogdanović, and N. Vukotić, "THE USE OF THE FLUTTER FRAMEWORK IN THE DEVELOPMENT PROCESS OF HYBRID MOBILE APPLICATIONS," *KNOWLEDGE - International Journal*, vol. 54, no. 3, pp. 477–483, Sep. 30, 2022, ISSN: 2545-4439, 1857-923X. DOI: 10/gt4sxc. [Online]. Available: https://ikm.mk/ojs/index.php/kij/article/view/5597.

[22] I. Jamali, "Identifying Trade-offs Associated with Cross-platform Mobile Development Tools," University of Saskatchewan Saskatoon, May 2022, 166 pp.

[23] G. W. Wiriasto, R. W. S. Aji, and D. F. Budiman, "Design and development of attendance system application using android-based flutter," in *2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*, Surabaya, Indonesia: IEEE, Oct. 2020, pp. 1–6. DOI: 10.1109/ICVEE50212.2020.9243190.

[24] A. D. I. A. Kadir, M. R. N. M. Alias, D. R. M. Dzaki, A. Azizan, and N. M. Din, "Mobile IoT Cloud-based Health Monitoring Dashboard Application for The Elderly," in *2022 4th International Conference on Smart Sensors and Application (ICSSA)*, Kuala Lumpur, Malaysia: IEEE, Jul. 26, 2022, pp. 161–166, ISBN: 978-1-66549-981-1. DOI: 10.1109/ICSSA54161.2022.9870913. [Online]. Available: https://ieeexplore.ieee.org/document/9870913/.

[25] D. Slepnev, "State Management Approaches in Flutter," p. 98,

[26] R. R. Prayoga, G. Munawar, R. Jumiyani, and A. Syalsabila, "Performance Analysis of BLoC and Provider State Management Library on Flutter," *Jurnal Mantik*, vol. 5, no. 36, p. 7, Aug. 2021, ISSN: 1591-1597. [Online]. Available: https://iocscience.org/ejournal/index.php/mantik/article/view/1539.

[27] H. G. Jones, "What plant is that? Tests of automated image recognition apps for plant identification on plants from the British flora.," *Aob Plants*, vol. 12, no. 6, 2020. DOI: 10.1093/aobpla/plaa052. pmid: 33173573.

[28] H. Jones, "Artificial Intelligence for plant identification on smartphones and tablets," *BSBI NEW*, pp. 34–40, Apr. 2020.

[29] T. Huynh, "A flashcard mobile application development with Flutter."

[30] a. deroussi, *Anassdrs/indoorPlantsDataSet: V0.1.0-alpha*, version V0.1.0-alpha, Zenodo, Aug. 10, 2022. DOI: 10.5281/zenodo.6980168. [Online]. Available: https://zenodo.org/records/6980168.

[31] B. V. Pennisi, *Growing indoor plants with success*, May 2020.

[32] A. C. Barr, "Household and garden plants," in *Small Animal Toxicology*. Elsevier, 2013, pp. 357–400. DOI: 10.1016/B978-1-4557-0717-1.00027-2.

[33] W. R. Roberts and E. H. Roalson, "Phylogenomic analyses reveal extensive gene flow within the magic flowers (achimenes)," *Am J Bot*, vol. 105, no. 4, pp. 726–740, Apr. 2018. DOI: 10.1002/ajb2.1058.

[34] V. P. Patil, D. Jagtap, S. Khodke, O. Jagdale, and A. Shitole, "Flutter-Modern and Easy Technology to Build Applications," *International Journal for Research in Applied Science and Engineering Technology*, vol. 11, no. 2, pp. 458–461, Feb. 28, 2023, ISSN: 23219653. DOI: 10/gt4s2n. [Online]. Available: https://www.ijraset.com/best-journal/flutter-modern-and-easy-application-development-platform.

[35] G. W. Wiriasto, R. W. S. Aji, and D. F. Budiman, "Design and Development of Attendance System Application Using Android-Based Flutter," *2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*, pp. 1–6, Oct. 3, 2020. DOI: 10/gt4s2k. [Online]. Available: https://ieeexplore.ieee.org/document/9243190/.

[36] M. I. M. Ariff *et al.*, "Mobile development: Learn du'a for early childhood learners," *Bulletin EEI*, vol. 11, no. 4, pp. 2253–2261, Aug. 2022. DOI: 10.11591/eei.v11i4.3860.

[37] S. Shah, N. Jain, S. Shah, P. J. Bide, and P. Bide, "A flutter application for farmers," in *2021 Asian Conference on Innovation in Technology (ASIANCON)*, 2021. DOI: 10.1109/asiancon51346.2021.9544511.

[38] Miquido. "Flutter architecture: Provider vs bloc - miquido blog." (Apr. 2020), [Online]. Available: https://www.miquido.com/blog/flutter-architecture-provider-vs-bloc/.