Siguín, M., Blanco, T., Rossano, F., Casas, R. (2025). Digital Tools for Designers: Means for Wearable Prototyping Through Block Programming. In: Manchado del Val, C., Miralbes Buil, R., Peris Fajarnés, G., Moncho Santonja, M., Rizzi, C., Roucoules, L. (eds) Advances on Mechanics, Design Engineering and Manufacturing V. JCM 2024. Lecture Notes in Mechanical Engineering. Springer, Cham. https://doi.org/10.1007/978-3-031-72829-7_47

# Digital tools for designers: Means for wearable prototyping through block programming

Marta Siguín[1,2] [0000-0003-3689-7780], Teresa Blanco[1,2,*] [0000-0002-1831-3342], Federico Rossano[4] [0000-0002-6544-7685] and Roberto Casas[1,3] [0000-0001-5316-8171]

[1] Aragon Institute of Engineering Research (I3A), University of Zaragoza, 50018, Spain
[2] Departamento de Diseño y Fabricación, University of Zaragoza, 50018, Spain
[3] Departamento de Ingeniería Electrónica y Comunicaciones, University of Zaragoza, 50018, Spain
[4] Department of Cognitive Science, University of California, San Diego, USA
tblanco@unizar.es

**Abstract.** Current technological developments have favoured the popularisation of portable electronic devices. However, the demand for advances is increasing, not only in the technical field, but also in the functional and ergonomic domain. These devices must be small and robust; and sometimes there is no room for integrated interfaces, and these must be externalised. To respond to these requirements, User-Centred Design methodologies are used, where prototyping plays a fundamental role. The prototyping of wearables requires a complicated integrated design-technology approach that facilitates the evaluation of the devices with real users, sometimes without technical expertise and in hostile environments. In this paper we discuss the use of mobile application development platforms based on block programming as a tool for designers in wearable prototyping. This stimulates a comprehensive design from the first phases of the project. This type of platform provides a visual development environment, easy to use, with predefined components, and an open community behind it which favours a fast and accessible validation of the proposed designs. To provide means for designers who want to enter the field, we present the functions that have been identified as basic in a wearable prototyping application; the elements involved; and conclusions for structuring its design and development.

**Keywords:** User Centred Design, Visual programming, Rapid prototyping, App Inventor, Wearables design.

## 1 Introduction

We live in a dynamic technological context, rich in innovation and advances in sensorization and monitoring, communications, and artificial intelligence, where the interconnection of devices with other devices, with the environment and with people is becoming increasingly important. In this context, wearables, portable electronic devices, are gaining popularity, and there is a demand for the development of products that are not only technologically advanced, but also respond to the functional and ergonomic needs of users [1–3].

The design of wearables is an arduous task whose results are expected to be small and robust devices that require reduced or even non-existent interfaces. Therefore, we must resort to the use of wireless communications that connect the wearable with other external devices more suitable for interaction and that require an interface design and the communications themselves. Furthermore, the use environment can be hostile, and its users may not be collaborative or have little technological experience, resulting in complex requirements [4–6].

To meet these requirements, User-Centred Design approaches are necessary, where prototyping plays a crucial role in evaluating and iterating proposed designs. Wearable prototyping requires technological validation, where crude versions are often used [7]; while when the designs are formally and functionally validated, the electronic assembly is not usually respected; and this involves an integration whose evaluation becomes a complicated phase to carry out with non-technological users.

Visual block programming is an approach within the No-Code and Low-Code philosophies [8]. It is a programming system for beginners based on the existence of puzzle pieces representing typical syntaxes of text programming [9]; which ease and reduce the learning curve for the design of simple mobile applications. This is due to the creation of predetermined blocks based on development patterns, which provide a structural base and reduce syntax errors; are aided by a visual component; and nurtured by collaborative communities [10].

Visual block programming can help designers in rapid prototyping the performance of their wearables [11] in an accessible way, favoring a comprehensive evaluation of their designs from the beginning.

The aim of this document is to expose the basic functions of a mobile application for wearable prototyping, the elements involved, and conclusions for structuring its design and development. To facilitate its understanding and validate its application, we exemplify it with a case study. This case study is focused on the design of a Bluetooth wearable for virtual livestock fencing using App Inventor platform. We hope that this research will empower designers in the current technological context.

## 2 Materials and methods

### 2.1 Design context

The use case used to guide the exposition focuses on the design of a Bluetooth wearable for virtual livestock fencing. Virtual fencing is a new control system for extensive livestock that evolves the current electrified fences. This system requires a device worn by the animal that applies electrical and/or sound stimuli when the animal tries to exceed a previously defined virtual perimeter [12]. The functioning of these systems is based on associative conditioning and seeks to teach the animal that after the sound stimulus comes the electrical stimulus; with the aim of favouring an autonomous functioning in the field that dispenses with the electrical stimulus [13]. This conditioning is achieved through training.

These devices are applied on farms or in open fields, with unfavourable conditions for the use of technology. The dirt and environmental conditions; the need of using gloves and how it difficults to manipulate the device; the non-collaborative users -animals-; and the non-technical users -humans-; lead us to consider the implementation of Bluetooth Low Energy (BLE) connections in the device to control it through a personalised mobile application. This application allows (i) to control the device remotely, without interacting with the animal, and in real time; (ii) to obtain data wirelessly; (iii) and to design an interface adjusted to the technical knowledge of the human user.

**Objectives and technical functions**. The main objectives of the mobile application were 3: (i) validate the operation of the electronic device itself, and specifically in automatic mode; (ii) define the levels of use of each type of stimulus (sound and electrical); (iii) and facilitate the design of the stimulus sequence, both for training and for autonomous deployment. To achieve these objectives, the app needed the following technical functions:

- Management of the app-device connection.
- Configuration of usage parameters (name, intensity of the electrical stimulus, duration of the electrical stimulus, frequency of the sound stimulus)
- Activation of the actuators
- Selection and/or definition of virtual polygons
- Data download.

In addition, the app had to be easy to use and understand by the user, so the design includes large buttons and fonts (scalable according to the size configured on the smartphone), high colour contrast to mitigate bright conditions, explanatory texts, validation rules, floating messages with feedback, etc.

**Logic design**. The logical design of the application is the result of the detection of functionalities and tasks, extracted following User-Centred Design methods (interviews, user-persona method, co-design sessions, etc.).. The diagram (Fig. 1) includes a rough representation of the functionalities and the navigation screens that structure the logical order of the app. Each of the purple boxes represents the functions that users demand, which can be extrapolated to other fields of wearables use. In this type of diagrams, it is convenient to specify the tasks necessary for the achievement of the functions in detail, but they will be detailed in section 3 of this document.


**Fig. 1.** Reduced logic app design. Functions: purple; Screen 1: yellow. Screen 2: green; Screen 3: blue.

## 2.2 Bluetooth Low Energy

The power consumption of classic Bluetooth limits its applications in devices whose small size is crucial, such as wearables. Thus, BLE has emerged as an evolution: a type of connection that uses the lowest possible wireless power [14] and, therefore, reduces the necessary capacity of the power source. The BLE application layer has a hierarchical organization of data that enhances interoperability of devices [15]:

- Profiles: are sets of services and characteristics that define functionality of a device. For example, “Virtual Fence Profile” defines our device’s functionality.
- Services: are containers of characteristics that define the specific features that build a profile. For example, the "Electrical Stimulator" service could include the "Electrical Stimulus Duration" and "Pulse Rate" features that make up the electrical stimulus.
- Characteristics: are logical units that represent specific data or functionality of a BLE device. For example, the characteristic 'Duration of electrical stimulus' define how many milliseconds is the stimulus applied.

This is important to know, because these elements structure the functionalities of BLE devices and are the elements you interact with when programming the device and the app. [14].

### 2.3 App Inventor

App inventor is one of the existing visual block programming environments [16]. It was created by Google Labs and is currently maintained by the Massachusetts Institute of Technology (MIT). It is an open-source online tool and enacts accessibility to programming for a wide audience. In addition, it has a large collaborative community behind.

In App Inventor (see Fig. 2) we can distinguish 3 basic blocks of functionality: design ('Designer' mode), developer ('Blocks' mode) and implementation ('Connect' and 'Build' tabs).

In the "Designer" mode, the visual design of the mobile application is structured and configured. It has predefined components (buttons, text boxes, sensors, databases, Bluetooth elements, etc.), visible and non-visible, to be implemented in the viewer. The selection of these components is crucial as they will be the elements referred to in the programming.

In the "Blocks" mode, we find a series of syntactic blocks for general programming and, on the other hand, syntactic blocks specific to each of the components used in the "Designer". The blocks must be dragged into the viewer to carry out the programming.

The implementation of the app can be emulated on a smartphone via the "Connect" tab; or an installable (.apk) file can be generated from the "Build" tab. During the design of the app, it is recommended to emulate the application to validate the created code or parts of it. For this purpose, the best option is to emulate it on a real smartphone via "AI Companion".

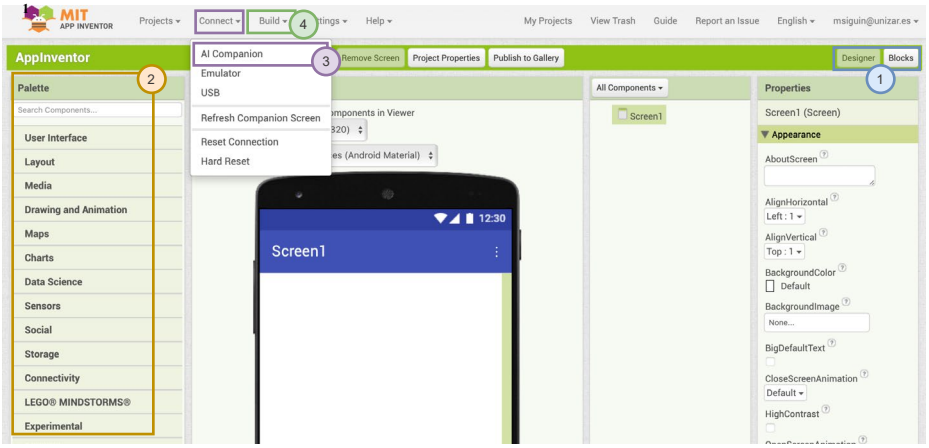


**Fig. 2.** App Inventor Interface. Element 1 (blue): Designer and Blocks modes. Element 2 (yellow): Component palette from Designer mode. Element 3 (purple): Connect tab and AI Companion option to emulate the app. Element 4 (green): Build tab.

# 3 Results

The analysis of wearables has enabled us to detect the basic functions that an app for Bluetooth control of an electronic device must have. In this section we summarise the tasks necessary to achieve these functions at a low level. These tasks must always be done, how to carry them out depends on the case.

## 3.1 Prototype

Before starting, it is important to understand the logic behind the App Inventor screen display. Within the platform, as many screens can be created as there are defined pages in the logic diagram. However, to use the same variables on multiple screens, they must be transferred from one screen to another. This means having to work with the databases as intermediate elements. Moreover, this does not work correctly in the emulator and the validation of the app must be done by building the .apk file each time, which makes the prototyping process difficult and slows down.

Therefore, to represent the pages in the "Designer" viewer, it is necessary to play with the display and hiding of layout elements (buttons, texts, etc.), that can be grouped vertically or horizontally. To facilitate the workflow in App Inventor, all pages and elements must be configured in the "Designer" before going into programming.

In the code block corresponding to the initialisation of the application, the display of the visual elements that appear when the application is executed must be configured. In addition, depending on the particularities of the application, other add-ons such as cloud credentials or permission requests have to be launched; or variables and lists have to be initialised.

Working with BLE within App Inventor requires a specific extension to be installed. This extension has been developed by MIT App Inventor itself and can be found in its official source (https://mit-cml.github.io/extensions/). Once the extension is installed, the first task to do is to define the wearable BLE services and characteristics as global variables.

**Device management.** In any mobile application with Bluetooth, connections must be managed: scanning and displaying devices and connecting or disconnecting them (see Fig. 3). To achieve this function, the following tasks are defined. Each of these tasks is represented by a logical block "When.Component.Action".

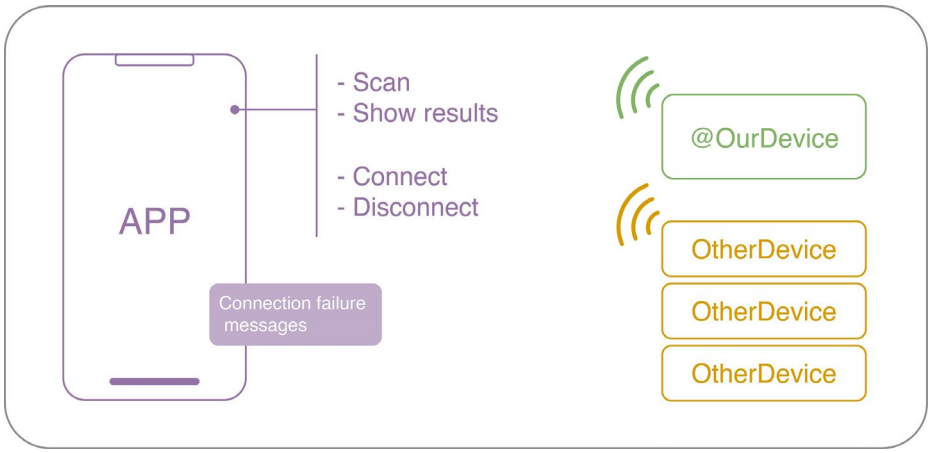


**Fig. 3.** Device management scheme. Any type of device with Bluetooth enabled, such as wearables, TVs, smartphones, etc., will be registered (in orange).

*Scan.* In our case study we have initialised the scan with a button. When this button is pressed, the user is asked for Bluetooth permissions. Changes also occur in texts, colours, and visibility of certain elements. These changes are not trivial because there are controls based on them.

*Show the list of devices and select one.* When devices are found, they shall be displayed in a list. When one of the devices in the list is tapped, scan stops and properties of the display components are modified (See Fig. 4 - left).

*Connecting the device/Disconnecting.* According to the control flow of our case study, based on the colour of the buttons, the procedure for connecting or disconnecting the device is called. When the device has connected, the navigation evolves, and the configuration of the page changes visually (See Fig. 4 - right). When the device is switched off after having been connected, it returns to the initial page setting.

*Connection failure.* If any type of connection error occurs, the user shall be notified and the list of BLE devices shall be redisplayed. The component type "Notifier" is used for this purpose.

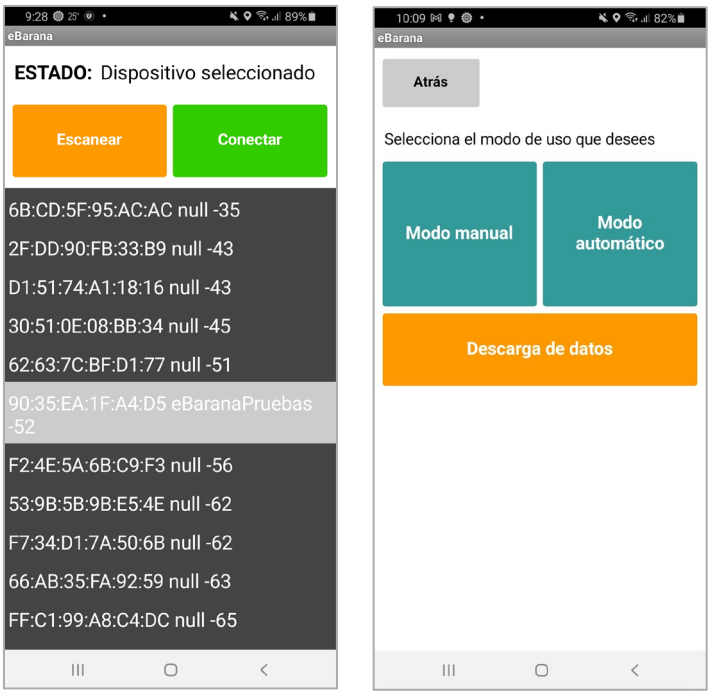


**Fig. 4.** Device management. Page with the list of Bluetooth devices detected in the scan; and selection of the desired one (left). Page after connection, where the operation of the device starts (right).

**Device operation. Configuration and actuation**. The operation of a Bluetooth device is based on the configuration and reading/writing of its characteristics (see Fig. 5). To achieve these functions, the following tasks are defined:

*Configure new value.* The user's input of values must be collected. This input can be done using various methods: sliders, text boxes, swiches, etc. In our case study, the input is stored in one of the labels placed as a component in the "Designer".

*Validate the configured data.* Through a conditional structure, the label used to store the value configured, must be checked. If the value meets the conditions, it is sent. Otherwise, a message is displayed to the user to modify it.

*Send the configured data.* A string with the value of the label is created and filled with zeros until it reaches the required length for sending. Finally, the byte writing process

is called, indicating the service and characteristic of the parameter to be configured. The length and type of the data depends on the configuration in the wearable firmware.

*Send the actuator-on characteristic.* The byte writing process is called again but with a different characteristic.

*Get data from a sensor.* In our use case study, we have not required the reading of any sensor, however, it is a function that can be recurrent in the programming of apps for wearables. The processes needed to carry out this function is “ReadFormat”; it obtains the data when the instruction is executed.

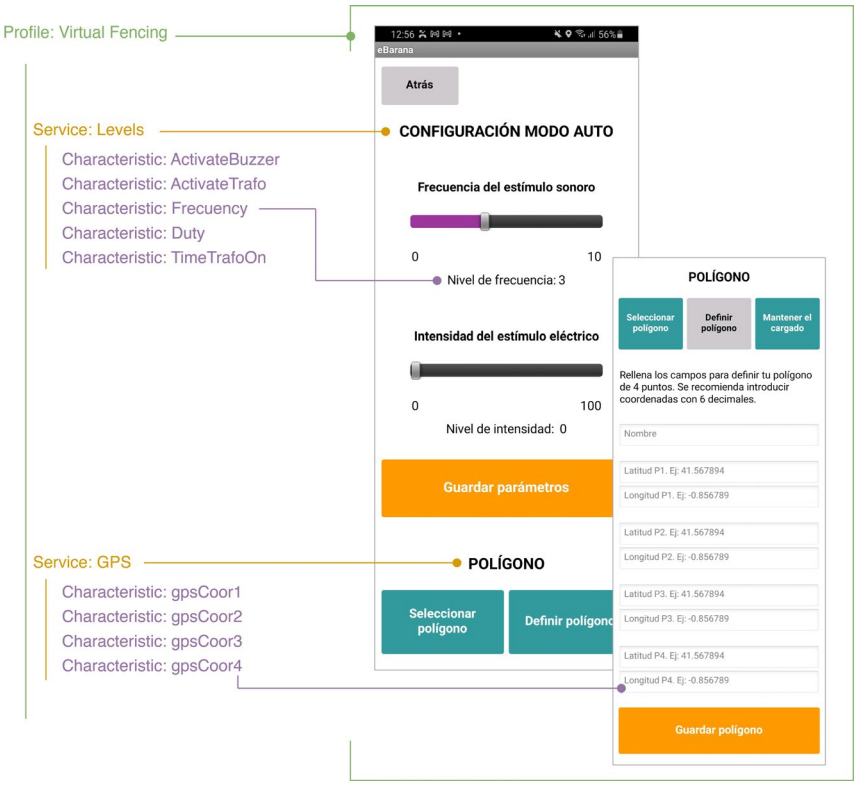


**Fig. 5.** Configuration and actuation BLE structure. It can be seen the automatic mode configuration page, where the actuation levels (levels service) can be modified and the points that define the virtual polygon (GPS service) can be introduced. Manual activation of the actuators is done on another page of the app and automatic activation is programmed in firmware.

**Device operation. Data Download.** Wearables usually have an internal memory for storing data. To download the data, a data service needs to be implemented. This service is different from the ones needed to configure and act on the device because this

function needs to subscribe to a “Notify” characteristic that changes the memory every time it detects an event that needs to be saved (see Fig. 6).

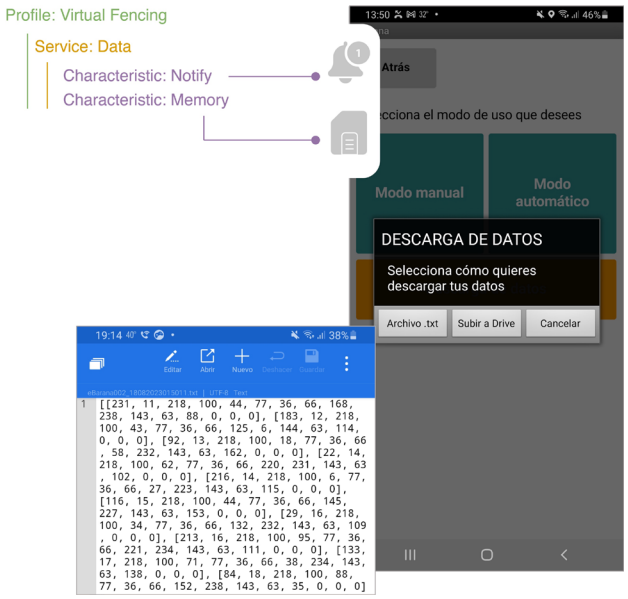


**Fig. 6.** Data download BLE structure. It can be seen the pop-up window that allows to decide how to save the data; and the format in which it is saved.

To achieve this function, the following tasks are defined:

*Subscribe to the Notify characteristic.* The "RegisterFormat" method must be called to register the characteristic. This saves data each time a change is detected.

*Make download request.* A memory characteristic is pointed to; and an AT type command is passed to it as a list. Feedback is given to the user when the request is made.

*Save data.* When data is received it is saved in a list.

*Store the data in a file (local or in the cloud).* For this, the user is previously asked about the desired type of file.

- Local .txt file: request permissions to save to the device's external memory; save the file; and notify the user.
- Google Sheets file in the cloud: name the website where the file will be hosted; make a "Get" call; specify the cells in which we want to save the data; and notify the user.

### 3.2 Field test

The design of the app has allowed users (veterinarians and livestock farmers) to be autonomous in the use of the device. With the proposed application, users can connect to the device, modify parameters, activate actuators, and download and visualise the collected data, to evaluate the device and advance the theoretical approach to virtual fencing. These functions can be extrapolated to the needs of other wearable application fields.

Initial uses of this virtual fence application showed that non-technical staff were able to configure and operate the device in the field (Fig. 7); in contrast to past experiences with applications provided by IoT developers, where staff couldn't use the wearable autonomously.

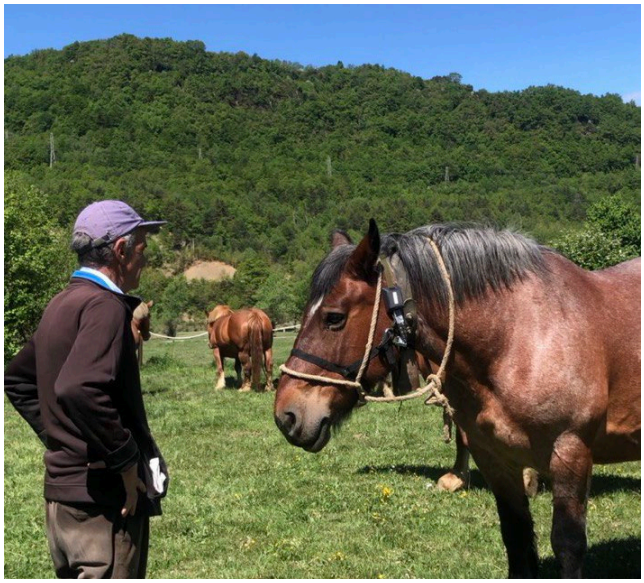

**Fig. 7.** Field test with livestock staff.

## 4 Conclusions and future work

Visual block programming platforms are a useful tool in wearable prototyping. They promote a whole process that is easy to implement for people with minimal programming experience. The first evaluations of the project showed good outcomes.

A systematised evaluation should be carried out to demonstrate the scope for designers to use these platforms in the prototyping of simple mobile applications. In addition, work must be done to establish a framework that facilitates the structuring of digital product design in relation to the workflow with this type of platform.

Designers do not have to become experts in technology, but they do have to take advantage of the tools available to them to complement their facilitating role and actively respond to the demands of society.

## Acknowledgements

This work was supported by Aragon Regional Government through the Program for Research Groups under Grant T27_23R (Howlab Research Group), through the Grant BOA20211216010, and through the Call "Grupos de Cooperación de Agentes del Sector Agrario Gobierno de Aragón 2020" (Proyecto Barana-Tech - https://esnepi.es/e-barana/); and by the Academic Senate of the University of California, San Diego (RG096635-COG506R).

## References

1. Ferraro, V., Ugur, S.: Designing wearable technologies through a user centered approach. In: Proceedings of the 2011 Conference on Designing Pleasurable Products and Interfaces. pp. 1–8. ACM, New York, NY, USA (2011). https://doi.org/10.1145/2347504.2347510.
2. Francés-Morcillo, L., Morer-Camo, P., Rodríguez-Ferradas, M.I., Cazón-Martín, A.: Wearable Design Requirements Identification and Evaluation. Sensors. 20, 2599 (2020). https://doi.org/10.3390/s20092599.
3. Blanco, T., Casas, R., Manchado-Pérez, E., Asensio, Á., López-Pérez, J.M.: From the islands of knowledge to a shared understanding: interdisciplinarity and technology literacy for innovation in smart electronic product design. Int J Technol Des Educ. 27, 329–362 (2017). https://doi.org/10.1007/s10798-015-9347-7.
4. Paci, P., Mancini, C., Price, B.A.: Understanding the Interaction Between Animals and Wearables. In: Proceedings of the 2020 ACM Designing Interactive Systems Conference. pp. 1701–1712. ACM, New York, NY, USA (2020). https://doi.org/10.1145/3357236.3395546.
5. Peter Wanga, H.: Designing a Machine Learning – Based Framework for Enhancing Performance of Livestock Mobile Application System. American Journal of Software Engineering and Applications. 4, 56 (2015). https://doi.org/10.11648/j.ajsea.20150403.13.
6. Siguín, M., Blanco, T., Rossano, F., Casas, R.: Modular E-Collar for Animal Telemetry: An Animal-Centered Design Proposal. Sensors (Basel). 22, 300 (2021). https://doi.org/10.3390/s22010300.
7. Graham, D., Zhou, G.: Prototyping Wearables: A Code-First Approach to the Design of Embedded Systems. IEEE Internet Things J. 3, 806–815 (2016). https://doi.org/10.1109/JIOT.2016.2537148.
8. Dhoke, P., Lokulwar, P.: Evaluating the Impact of No-Code/Low-Code Backend Services on API Development and Implementation: A Case Study Approach. In: 2023 14th International Conference on Computing Communication and Networking Technologies, ICCCNT 2023. Institute of Electrical and Electronics Engineers Inc. (2023). https://doi.org/10.1109/ICCCNT56998.2023.10306945.
9. Ricarose Vallarta, R.: OpenBlocks: An Extendable Framework for Graphical Block Programming Systems, (2007).

10. Bau, D., Gray, J., Kelleher, C., Sheldon, J., Turbak, F.: Learnable programming. Commun ACM. 60, 72–80 (2017). https://doi.org/10.1145/3015455.
11. Arbi, K.F., Kromba, I., Saffih, F., Ben-Ramdane, A., Slami, A., Hadjersi, A., Soulimane, S., Brixi Nigassa, M.E.: Intelligent IoT (I $^{2}$ oT) Biomedical Wearable System based on Smartphone Application. In: 2020 IEEE 5th Middle East and Africa Conference on Biomedical Engineering (MECBME). pp. 1–4. IEEE (2020). https://doi.org/10.1109/MECBME47393.2020.9265158.
12. Umstatter, C.: The evolution of virtual fences: A review. Comput Electron Agric. 75, 10–22 (2011). https://doi.org/10.1016/j.compag.2010.10.005.
13. Marini, D., Cowley, F., Belson, S., Lee, C.: The importance of an audio cue warning in training sheep to a virtual fence and differences in learning when tested individually or in small groups. Appl Anim Behav Sci. 221, 104862 (2019). https://doi.org/10.1016/j.applanim.2019.104862.
14. Afaneh, M.: Intro to Bluetooth Low Energy. (2018).
15. Heydon, Robin.: Bluetooth low energy: the developer's handbook. Prentice Hall (2013).
16. App Inventor Homepage, https://appinventor.mit.edu, last accessed: 2024/04/10.