

Design of mobile application for communication and user interface of ESP32 potentiostat system

Retno Supriyanti¹, Wahyu Widanarto², Putra Dwi Susanto¹, Madya Ardi Wicaksono³, Syafrudin Rais Akhdan⁴, Muhammad Alqaaf⁴

¹Department of Electrical Engineering, Jenderal Soedirman University, Purbalingga, Indonesia

²Department of Physics, Jenderal Soedirman University, Purwokerto, Indonesia

³Department of Medical, Jenderal Soedirman University, Purwokerto, Indonesia

⁴Department of Informatics Science, Nara Institute of Science and Technology, Ikoma, Japan

Article Info

Article history:

Received Oct 23, 2024

Revised Sep 28, 2025

Accepted Oct 9, 2025

Keywords:

Bluetooth low-energy

Cyclic voltammetry

ESP32

Mobile application

Potentiostat

ABSTRACT

The potentiostat utilizing the ESP32 has a 12-bit analog-to-digital converter (ADC), meaning the maximum value for ADC voltage readings on the ESP32 is 4095. These ADC readings are then converted into actual voltage units, ensuring more accurate measurements on the potentiostat. To facilitate the use of the ESP32 potentiostat, a mobile application must be designed as a user interface for data communication. The application will be developed on a mobile platform using a Bluetooth low energy (BLE) communication channel for easier access. The development process will utilize visual studio code as the code editor and programming languages like Dart and Flutter. The resulting application will feature a user-friendly dashboard, display data in a cyclic voltammetry graph, and store data in comma-separated values (CSV) files or images in the phone's memory. This stored data will simplify observing results obtained from the ESP32 potentiostat.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Retno Supriyanti

Department of Electrical Engineering, Jenderal Soedirman University

Kampus Blater, Jl. Mayjend Sungkono KM 5, Blater, Purbalingga, Central Java, Indonesia

Email: retno_supriyanti@unsoed.ac.id

1. INTRODUCTION

Technology supports human needs in all aspects of life, including health. Biosensors are one of the technologies that emerged when fast, sensitive, and easily transportable instruments and detection were needed. They are bioanalytical devices for detecting very low biomarkers. Biosensors are made with nanomaterials whose surfaces can be engineered for specific purposes [1]. One tool in biosensors is the screen-printed electrode (SPE), which detects an analyte. SPE is a low-cost electrode whose surface can be modified to improve the material's detection ability [2]. SPE requires a potentiostat to analyze the results of electrochemical reactions on the working electrode's surface. SPEA potentiostat is an electronic device that flows voltage on the working electrode to the reference and flow current between the working electrode and the counter. The results of both flows will be used as observation results from the potentiostat. A potentiostat is required to analyze the results of electrochemical reactions on the working electrode's surface [3]. Biosensor measurements using potentiostats can be done with several measurement methods, such as cyclic voltammetry (CV), linear sweep voltage metering (LSV), and chronoamperometry (CA). One of the potentiostats developed uses the ESP32 device as a microcontroller controller and reader. ESP32 was chosen because it is relatively affordable and has pins with quite complete functions. A potentiostat was developed using CV analysis, the most common medical method. CV is produced by measuring the current read in the electrochemical cell when voltage is applied to the working electrode. In biosensors, potentiostats are often

used to detect specific biological interactions through electrochemical detection. These devices measure the electrical changes due to biochemical reactions, usually involving enzymes, antibodies, or deoxyribonucleic acid (DNA). The advantages of using potentiostats as a biosensor are as follows: i) high sensitivity: potentiostats can detect minimal changes in current, making them suitable for detecting low concentrations of analytes, ii) real-time monitoring: they allow for real-time and continuous monitoring of biological reactions, and iii) versatility: potentiostats can be used with various biosensors, including enzyme-based, immuno-based, and DNA-based sensors.

Several research studies have discussed using potentiostats as biosensors, including the following: Radogna *et al.* [4] use a wireless potentiostat named ElectroSense as an electrochemical biosensor interface. This system is used for non-invasive blood sugar monitoring. In principle, they use digital to analytical conversion on a microcontroller by utilizing pulse width modulation to generate signals. This feature enables the internet of things (IoT) in everyday healthcare. Kaci *et al.* [5] developed a portable biosensor to detect bacteria. The biosensor consists of a potentiostat that can control a screen-printed platform with a MoS₂ surface and a DNA probe. The current response was obtained from thionine-functionalized carbons (Ty-CDs), monitored as electrochemical indicators. Archer *et al.* [6] compared several biosensors for salivary biomarker detection that can be used to identify and monitor systemic diseases. Eleven studies were identified, eight of which discussed continuous biomarker monitoring, and the remaining three discussed point-of-care applications. All sensors showed excellent sensitivity. Souza *et al.* [7] developed a complementary metal-oxide-semiconductor (CMOS) potentiostat designed for electrochemical cells. The proposed topology is a circuit interface that controls the voltage polarization at the sensor electrode and supports current measurement in oxidation and reduction processes in carbon analysis. Suganthan *et al.* [8] researched protein-based biosensors to detect the pathogenic bacteria *C. jejuni* national collection of type cultures (NCTC) 11168 in food. Bounegru *et al.* [9] explored the development and performance of a SPE modified using graphene (GPH), poly(3,4-ethylene dioxythiophene), and tyrosinase (Ty) designed for water analysis. Ekonomou *et al.* [10] developed an electrochemical biosensor to measure glycerol levels in wine. Kazemzadeh-Beneh *et al.* [11] developed a biosensor to detect levels of *Candidatus Liberibacter asiaticus* (CLas), the leading cause of citrus fruit disease. Fiska *et al.* [12] researched the development of non-invasive biosensors for glucose detection in sweat. Elnagar *et al.* [13], using an electrospinning technique, researched an impedimetric electrochemical biosensor formed by nanofiber (NF) from bio-copolymers. This biosensor combines several environmentally friendly polymers, such as sodium alginate (SA) and polyethylene oxide (PEO), as antifouling polymers, with folic acid as a biorecognition element. Thongkhao *et al.* [14] developed a biosensor based on a polyurethane-polyaniline-m-phenylenediamine layer, using a screen-printed gold electrode, to detect lactic acid in the blood. Stoikov *et al.* [15] developed a biosensor for uric acid detection based on an electrochemical cell with a modified screen-printed graphite electrode (SPE). Filho *et al.* [16] developed a biosensor embedded with machine learning. Ashoori *et al.* [17] proposed a new circuit architecture by utilizing dynamic voltages at the electrodes of electrochemical cells to increase the voltage range used. Mari *et al.* [18] developed an electrochemical biosensor based on hierarchical polyglutamic acid/ZnO coupled with gold nanoparticles for detecting α -synuclein in human blood plasma. In this research, we create a mobile application that displays CV graphs from potentiostat measurement results. The application and the ESP32 Potensiostat device can be connected using Bluetooth low energy (BLE) or Wi-Fi. We chose communication using BLE. We chose BLE because it is a low-power communication option in the ESP32 communication options other than Wi-Fi. This mobile application is expected to read voltage and current data from the ESP32 potentiostat. Then, the application saves the data into a comma-separated values (CSV) file. So, the application can display data information that matches the measurement results on the ESP32 potentiostat. Psotta *et al.* [19] created a portable potentiometer to detect *E. coli* bacteria in urine. The developed prototype utilizes modified SPEs. Sensor output is wirelessly transferred via Bluetooth and recorded on mobile phones and laptops. Hossain and Tabassum [20] evaluated a hybrid physicochemical sensor array simultaneously measuring vapor pressure. Our previous research [21]-[23] researched image processing in medical images. In other research, [24] GPH was synthesized from microwave-irradiated activated carbon for use in supercapacitor electrodes. This paper focuses on the principles of mobile application communication with microcontrollers, creating an exemplary user interface, and how data obtained from the microcontroller can be stored in mobile phone memory. For that reason, we limit the discussion in this paper to the mobile application, which can communicate with the microcontroller and display information in the form of CV graphs. The CV graph can be stored on the mobile device in the form of a CSV file; the mobile application can display the creation of graphs in real time and does not discuss the ESP32 potentiostat electronic circuit.

2. RESEARCH METHOD

2.1. Application design

This application is planned to accommodate several features, as listed in Table 1. To run a feature, application development requires a package or plugin with the functionality according to the expected feature. Plugins make it easier to create application features. Developing applications to run these features certainly requires packages or plugins with the functionality according to the expected features. Using plugins makes it easier to create application features.

Table 1. Application features

No.	Feature	Description
1	The application can connect with the ESP32 potentiostat using BLE.	Users can scan for available Bluetooth devices and connect the ESP32 potentiostat device to the mobile device.
2	The application can receive data from ESP32 potentiostat.	The application allows data to be received from the ESP32 potentiostat in serial form. The data obtained is also displayed on the serial monitor on the Arduino IDE.
3	The application can plot the data received from the ESP32 potentiostat into a CV graph.	The received data is arranged to appear on the screen as a CV graph that compares the current and voltage variables against time.
4	The application can save data and graphics for the phone.	The data obtained can be saved as a CSV file or image. Then, the graph formed can be saved as an image.

The application is designed to run on Android 5.0 and above, with a minimum software development kit (SDK) requirement of 21. This is necessary because the packages used in the application only support development from this SDK version onward. These packages act as intermediaries, facilitating communication between the application and the ESP32. Therefore, the packages must support BLE to develop Android applications using Flutter.

2.2. User interface design

In making a mobile application, it is necessary to create a user interface design to make it easier for developers to develop applications. We created a user interface design using the Figma application as the initial framework for building the application. After the application was designed in Figma, we implemented the application in vs. code using the Flutter framework. The application has three main page views: home, graph, and about. The home page is the first page of the application and displays a menu that users can select. The graph page is used to create graphs from .csv format files. The about page displays information about ongoing research and research contributors. Figure 1 shows the user interface design of the three main pages (home (a), graph (b), and about (c)) we developed in this research.

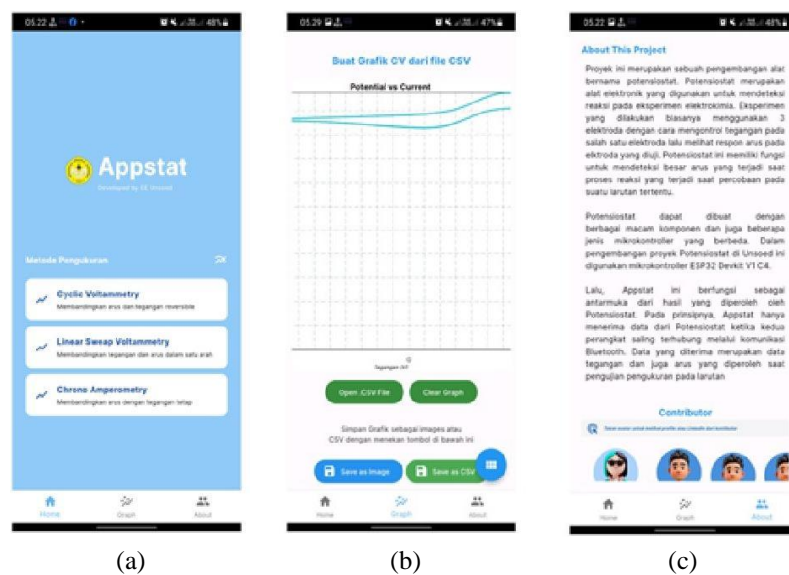


Figure 1. Applications on the main page; (a) home, (b) graph, and (c) about

2.3. Data communication flow

The microcontroller device communicates with the smartphone using BLE in the data communication we developed. This condition allows no internet connection between the microcontroller device and the smartphone. BLE is a communication that will enable low power on the microcontroller device.

BLE communication provides guaranteed data security through AES-128 encryption, which operates effectively in the BLE environment. One of the key advantages of BLE is its low power consumption. This is achieved by allowing data transmission to occur over a short period, followed by inactive phases known as "nap," "sleep," or "deep sleep." Combining a small duty cycle and efficient use of radio waves results in significantly lower power consumption compared to Bluetooth classic. The power required for transmission ranges from -20 dBm (0.01 mW) to +20 dBm (100 mW) [25]. The ESP32 already supports BLE communication that can be activated and given a universally unique identifier (UUID) to indicate the difference between each available BLE device. This UUID will be helpful in communication as a marker for the data communication path between the smartphone and the microcontroller device. Applications with microcontrollers can communicate with each other when the smartphone recognizes the UUID of the microcontroller device. Figure 2 is a description of the data communication that occurs between the ESP32 and the smartphone.

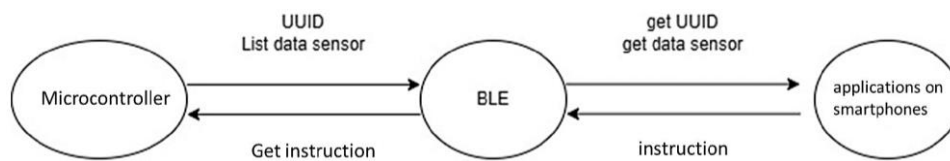


Figure 2. Data communication flow

The device will send data to the application directly after both are connected in the data communication flow. Data is transmitted in a two-dimensional array format, where each index represents a current and voltage value. Data sent from the microcontroller is directly plotted into a graph displayed when the device is connected. Service characteristics are a container for sending data from the device to the application. The maximum data that can be sent through one characteristic is 512 bytes. Maximum transmission unit (MTU) is the maximum amount of data sent in one BLE packet. MTU in default conditions is 23 bytes.

In this scenario, BLE communication is facilitated using the BLE Device.h, BLE Server.h, BLE Utils.h, and BLE 2902.h libraries. These four libraries enable the ESP32 to utilize BLE functionality effectively for sending and receiving data. Each piece of data is transmitted using functions from the BLE Device.h library, configured with the appropriate UUIDs for both transmitting (TX) and receiving (RX) conditions between the application and the microcontroller. The ESP32 operates as a server and a client in this one-to-one communication setup. Communication between the application and the microcontroller is smooth, with a delay of no more than 200 milliseconds. Initially, the Android device can connect directly to the microcontroller, allowing the user to start the measurement on the application simply. The data is then processed automatically for delivery.

3. RESULTS AND DISCUSSION

The main page we designed, as in Figure 1, is a three-screen display of home, graph, and about. The Home () class creates a home tab display using several widgets to organize the display. The Home () class returns a value as a SafeArea () widget, ensuring that the display does not exceed the application area it should be. With this widget, the application does not exceed the upper limit on the battery indicator and notification tabs. Then, the Center () widget is used to place the widget at the Center of the screen. The child and children parameters are parameters used to add widgets to other widgets; child only allows one widget to be accommodated, and children can accommodate more than one widget. The combination of Column() and Row() widgets is a method used to set the position of each widget to be aligned horizontally or vertically. This arrangement makes developing a reasonably complex display in the application easy. Custom widgets are used to create a little by calling a class defined in another file. So, to create a list display, we only need to contact the ClickList() class by providing the title, subtitle, and widget parameters to move to another display. The display is in the form of a graph and also several buttons have specific functions, namely the button to open the CSV file, precise the graph, save as an image, and save as CSV. The button is created using a floating button widget that can accommodate certain functions. Each button performs a different

function, defined outside the widget build. The graphic display uses `graphCustomWidget()`, a custom widget for creating graphs with the `LineChart` widget. The `LineChart()` widget comes from the `fl_chart` package and has a feature for creating graphs to suit application needs. Creating a graph with `LineChart()` requires a data list in `FLSpot()` format by taking values from the sensor. Dart. `FLSpot()` is a class on `fl_chart` to provide correlated points between the x-axis and y-axis. In this widget, several parameters are set, starting from the background color, the title on each side, and the thickness of the graph. The About page contains information about the Appstat application. It is displayed using simple widgets, namely `Text` and `Images` arranged using `Columns()` and `Rows()`. The `Profile()` widget displays avatars and direct profiles whenever pressed.

The ESP32 Bluetooth communication feature with Flutter can be connected using the `flutter_blue_plus` package on Flutter and the `ESP32_BLE_Arduino` library on the ESP32 potentiostat. `Flutter_blue_plus` on Flutter uses the Bluetooth feature available on Android devices. `Flutter_blue_plus` has a minimum SDK development requirement on Android devices, namely SDK 19 or on a minimum operating system version of Android 4.4 (KitKat). An additional program is required for BLE to activate the Bluetooth feature on the ESP32 potentiostat. The BLE functions that are set include `UUID` device, `BLE` server, and `notify BLE`. The `UUID` setting is intended so that BLE on the ESP32 Potensiostat has a device identity as a sender of information to the connected device. The ESP32 Potensiostat sends information when the device is connected to the application. Then, check the connection using `MySeverCallbacks`, which produces an actual value on a device connected when the device is connected. The exact value makes the ESP32 Potensiostat device send data by running `BLEDevice::startAdvertising()`. Data is stored in the `send_voltage` and `send_current` variables to be sent to the application. Both variables are entered into the `str` variable of type string. Then, `characteristic ->setValue((char *)str.c_str())` is used to set the value sent to the BLE characteristic. The distinct object calls the `notify()` function to inform the application whenever there is a new value from the ESP32 potentiostat. The Appstat application is designed to communicate via Bluetooth with the ESP32 potentiostat device. In connecting the device to the application, a screen display is required that displays the BLE device name from the ESP32 potentiostat. Using the `flutter_blue_plus` package, the screen display can display BLE devices around the smartphone. The `ble_scanner.dart` file is a source code file used to create a list of Bluetooth devices. In the source code, there are two `StreamBuilders` that have different uses. The first `StreamBuilder` displays devices connected to the application as `ListTile`. The second `StreamBuilder` maps the results of scanning devices from the data stream. Each data obtained is mapped to the result parameter of the `ScanResultTile` class. The class has an `onTap` parameter to direct to the `SensorPage()` page that processes the data into a graph. The result of `_buildTile` is the device name and ID, which are then used in the `ExpansionTile` widget. In `ExpansionTile`, three parameters need to be entered: `leading`, `title`, and `trailing`. In the source code, the `leading` section is filled with the value of `RSSI`, the `title` is filled with the name and ID of `_buildTile`, and the `trailing` section is filled with the button to connect the BLE device. The `onTap` parameter on `ScanResultTile` is directed to the `SensorPage()` class located in the sensor. Dart source code. This source code defines the `service_uuid` and `characteristic_uuid` of the ESP32 Potensiostat device. So, if the `UUID` of the BLE device is different, the application cannot connect to it. `ServiceUUID`, `characteristic UUID`, `service UUID`, and `characteristic UUID` are variables that hold the `UUID` of the device that can connect to the application. The `UUID` is the same as that used by the ESP32 Potensiostat device. The `discoverServices()` function is `async`, indicating that the function works repeatedly. Then, all available services from the Bluetooth device are listed and stored in the `services` variable. A loop is performed for each service value compared to the `service_uuid` and `characteristic_uuid` values to get the service. When the value is the same as the value that has been defined, the function `characteristic.notifyValue` is run to receive the value from the ESP32 potentiostat device via the BLE characteristic. The stream variable is filled with the value from `characteristic.onValueReceived` or the value received in the characteristic. The `_tempdata` variable holds the values from the device in the form of a list, the voltage variable is used to store the value of `_tempdata[0]`, and the current variable is used to store the value of `_tempdata[1]`. The selection of index 0 and 1 adjusts the order of values given by the ESP32 potentiostat. Next, the `dataVoltageCurr` and `dataset` variables are used as data containers entered into the `CVPage()` class parameter in `cv`. Dart. The `dataVoltageCurr` variable is used as a value that is stored in a CSV file, and the `dataset` fills the `graphCustomWidget()` parameter. The first `StreamBuilder` contains data received from the `notifyServices()` function. This Builder returns a value in the form of a `CVPage()` class display widget with the values that are parameters in the class, such as `voltage1`, `current2`, and `widget.device`, `dataspot`, `dataVoltageCurr`, `_ListTegangan`, and `_ListCurrent`. When the Builder does not get a data stream, the application displays "Check The Stream" text on the screen.

Displaying graphs on Appstat can use several packages to create graphs by utilizing data available in variables or lists. The graph display settings are set in the `graphCustomWidget()` widget. The widget used is `LineChart()` to create a line graph using two data for the vertical and horizontal axes. The horizontal axis is set for voltage data from when receiving characteristics. The data sent to the `CVPage()` class is converted into a list format containing the `FLSpot(x,y)` value, with `x` being the voltage data for the horizontal axis and `y`

being the current data for the vertical axis. When receiving these characteristics, a list containing `FLSpot(x,y)` is more feasible than returning two voltage and current data to `CVPage()`. Returning voltage and current values separately to `CVPage()` creates `FLSpot`. A list containing `FLSpot(x,y)` data is defined in the `_SensorPageState()` class named `dataset`. The `StreamBuilder` section of each data received in the current and voltage variables is transferred to the `voltage1` and `current2` variables. After that, `voltage1` and `current2` are entered into `FLSpot(voltage1,current2)` in the `dataspot` List using the `dataspot.add(FLSpot(voltage1,current2))` command. Because it is in the `StreamBuilder()` widget, `FLSpot(voltage1,current2)` continues to be added to the `dataspot` List until the BLE ESP32 potentiostat finishes sending data. This allows the displayed graph to continue to be updated to adjust the values in the `dataset` list. The list data spot value fills one `CVPage()` class parameter. The `dataset` is included in the widget—data variable in the `CVPage()` class. A custom widget named `graphCustomWidget()` creates a graph using the widget—data parameter, the `dataset` variable from `SensorPage()`. In `LineChartBarData()`, other parameters are set, such as `color` to set the color of the resulting line, `curvature` to make the line non-angular, and `dot data` to set the value points to be removed so only the line remains. The `Graph()` page has a feature to create graphs from .csv files. A function must import the CSV file and convert its data into a list to accommodate this feature. The `file_picker` package transforms the selected File into a variable called `result`. When this result is not null, the command to open and read the File continues using `File(filePath).openRead()`. The `filePath` variable used as a parameter in the `File()` method results from the `result.files.first.path!`. The `!` A sign at the end ensures the value cannot contain null (empty). Next, convert the value to `utf8` and make CSV into a list using the function `.transform(const CvsToListConverter().transform toList())`. After that, the values obtained in the list fields are mapped into the `FLSpot` list. The repetition (read: iteration) is as many as the number of list field indices. `SetState()` is intended to reset the values of the data list and `datasets`.

The graph that is successfully created is not only displayed on the application. However, the application also has a feature to save graphs in image format and CSV files. Keeping this file requires permission to write the file to the phone's memory. This permission setting is in the `AndroidManifest.xml` file in the `Android/app/src/res` directory. Permission allows the application to read, write, and manage storage on Android devices. This permission setting only applies to Android devices. If the application is to be run on an IOS device, then a different permission is required for the application to be able to manage storage. After adding permissions, the next step is to add the necessary packages to `Pubspec. yaml` to save files in image and CSV format. The `screenshot` package and `image_gallery_saver` package capture graphics and save them in image format. The captured images will be put into the Pictures folder. Furthermore, the CSV and `path_provider` packages save graphic data into a CSV file. The function to save files in CSV is created in a different file to make it easier to manage the code. The file for the function is named `helper_csv.dart`, and it is in the same folder as `cv`. Dart. A separate file allows the function to be used in other source code. The function has a void `saveCSV` method with a list parameter of the double list data type. A double list is intended so that the data stored is a two-dimensional list. The `saveCSV` method calls the `mobileSaveSCV` function. The parameters in the function are the parameters entered in `saveCSV`. The second function uses the `async` type, and the nature of `async` is similar to that of a stream. Data is converted from list form to CSV using `ListToCsvConverter().convert(list)`. After that, the function takes the local directory on the phone using the `await getDownloadsDirectory` command from the `path_provider` package. The CSV file is saved to the storage directory with the name set in the file variable. When it fails to save, the application debug issues an error message that it cannot save the file. The second feature is to make the graph can be saved in the form of an image. To do this, two packages, `screenshot` and `image_gallery_saver`, are needed. The function's source code is placed in the same file as before the widget build. The function enters the state of the `_CVPageState()` class in the `cv`—dart file. A screenshot controller from the `screenshot` package must be defined to capture an image. The `screenshotController` object is used as the controller name. The controller is used to save the `toGallery()` function. The `screenshotController` object captures the image and is given in the `save screenshot` function parameter. In addition, when the `savetoGallery()` function is called, a notification appears in the form of a black bar at the bottom of the screen that displays "Image Saved to Gallery." The `save screenshot` function saves images in the gallery with the name format "AppStat\$dates," where the date is the time when the screenshot was taken. To take a screenshot, the `Container` widget that holds the graph must be wrapped with the `Screenshot` widget. The `screenshotController()` object fills the controller parameters on the `Screenshot()` widget. The `savetoGallery()` function runs when the "Save as Image" button is pressed.

Testing is done by testing all the application features planned in Table 1. Testing only refers to the application's functionality. Testing does not pay attention to changes in the display when causing the application's display to fail. Table 2 shows the results and descriptions of the functionality testing performed.

Table 2 shows that the application can run all features. In this experiment, we used Samsung A50s (Android 11, API 33), which experienced problems plotting graphs using CSV files on the `Graph()` tab. The limited remaining phone memory can cause this. Therefore, it is necessary to optimize memory usage in the

AppStat application. Another obstacle is when the display repeats the ESP32 potentiostat measurement. When repeating the measurement, the application can repeat the measurement correctly. However, the application displays a gray screen when receiving data for the first time, which indicates that this application still has bugs in its appearance.

Table 2. Testing results

No	Feature	Result	Description
1	The application can connect with the ESP32 potentiostat using BLE.	Success	ESP32 potentiostat can be connected to the application if the device has turned on the Bluetooth feature.
2	The application can receive data from the ESP32 potentiostat.	Success	ESP32 potentiostat can send data through characteristics defined on the ESP32 potentiostat, and applications can read these characteristics.
3	The application can plot the data received from the ESP32 potentiostat into a CV graph.	Success	The data received when the ESP32 potentiostat device is connected to the application is directly plotted into a graph.
4	The application can save data and graphics for the phone.	Success	The application can save data into two file formats: .jpg and .csv. The .csv format stores data as numbers, while the .jpg format stores graphs as images.

4. CONCLUSION

The application can connect to the ESP32 potentiostat via BLE using the same UUID, where the ESP32 potentiostat can send data via BLE using the characteristic notify. In addition, this application can save data in the form of CSV or image files. For further research, we will improve the capabilities by adding features to the application. Some notes in this research are that the graphs displayed in the application could be more interactive because the package support could be better, so we will use other graph-making packages such as Syncfusion on Flutter. This package allows the creation of more interactive graphs, such as zooming, panning, and viewing values at each point. In addition, we will optimize the application's memory usage when running the feature. Integrating a database system with the cloud application is essential to enhancing development. This integration facilitates advanced data analysis using AI tools, supporting a more comprehensive evaluation of the measurement results obtained. This can be done using an excellent architectural concept for writing flutter source code. We maximized the appearance of the user interface of each page so that it can be displayed more simply and have a workflow that users can better understand. We will also add features so that the application can enter measurement parameters such as minimum voltage, maximum voltage, scan rate, and sampling frequency so that the application can change the measurement scheme carried out by the ESP32 potentiostat. In this case, we will use other frameworks, such as react native, which is based on JavaScript, to expand development and also as a comparison with applications that use Flutter. However, the application's runtime will likely take longer because it is JavaScript-based.

FUNDING INFORMATION

Jenderal Soedirman University funded this research through the International Research Collaboration Scheme 2024, contract numbers 705/UN23/PT.01.02/2024.




REFERENCES

- [1] B. Purohit, P. R. Vernekar, N. P. Shetti, and P. Chandra, "Biosensor nanoengineering: Design, operation, and implementation for biomolecular analysis," *Sensors International*, vol. 1, pp. 1-19, 2020, doi: 10.1016/j.sintl.2020.100040.
- [2] S. Tajik, H. Beitollahi, S. A. Ahmadi, M. B. Askari, and A. Di Bartolomeo, "Screen-printed electrode surface modification with NiCo₂O₄/rgo nanocomposite for hydroxylamine detection," *Nanomaterials*, vol. 11, no. 12, 2021, doi: 10.3390/nano11123208.
- [3] A. W. Colburn, K. J. Levey, D. O'Hare, and J. V. Macpherson, "Lifting the lid on the potentiostat: a beginner's guide to understanding electrochemical circuitry and practical operation," *Physical Chemistry Chemical Physics*, vol. 23, no. 14, pp. 8100-8117, 2021, doi: 10.1039/D1CP00661D.
- [4] A. V. Radogna, L. Francioso, E. Sciurti, D. Bellisario, V. Esposito, and G. Grassi, "A Wireless Potentiostat Exploiting PWM-DAC for Interfacing of Wearable Electrochemical Biosensors in Non-Invasive Monitoring of Glucose Level," *Electronics*, vol. 13, no. 6, pp. 1-12, 2024, doi: 10.3390/electronics13061128.
- [5] K. Kaci *et al.*, "Multiplex Portable Biosensor for Bacteria Detection," *Biosensors*, vol. 13, no. 11, pp. 1-18, 2023, doi: 10.3390/bios13110958.
- [6] N. Archer, S. Ladan, H. T. Lancashire, and H. Petridis, "Use of Biosensors within the Oral Environment for Systemic Health Monitoring—A Systematic Review," *Oral*, vol. 4, no. 2, pp. 148-162, 2024, doi: 10.3390/oral4020012.
- [7] A. K. P. Souza, C. A. de M. Cruz, É. C. D. e S. J., and F. B. Pontes, "Current Mirror Improved Potentiostat (CMIPot) for a Three Electrode Electrochemical Cell," *Sensors*, vol. 24, no. 5897, pp. 1-12, 2024, doi: 10.3390/s24185897.
- [8] B. Suganthan *et al.*, "A Bacteriophage Protein-Based Impedimetric Electrochemical Biosensor for the Detection of *Campylobacter jejuni*," vol. 14, no. 8, pp. 1-17, Aug. 2024, doi: 10.3390/bios14080402.
- [9] A. V. Bounegru, C. Iticescu, L. P. Georgescu, and C. Apetrei, "Development of an Innovative Biosensor Based on Graphene/PEDOT/Tyrosinase for the Detection of Phenolic Compounds in River Waters," *International Journal of Molecular*




- Sciences*, vol. 25, no. 8, pp. 1-24, Apr. 2024, doi: 10.3390/ijms25084419.
- [10] S. I. Ekonomou, A. Crew, O. Doran, and J. P. Hart, "Development of a Disposable, Amperometric Glycerol Biosensor Based on a Screen-Printed Carbon Electrode, Modified with the Electrocatalyst Meldolas Blue, Coated with Glycerol Dehydrogenase and NAD⁺: Application to the Analysis of Wine Quality," *Applied Sciences*, vol. 14, no. 14, pp. 1-12, 2024, doi: 10.3390/app14146118.
 - [11] H. Kazemzadeh-Beneh, M. R. Safarnejad, P. Norouzi, D. Samsampour, S. M. Alavi, and D. Shatterreza, "Development of label-free electrochemical OMP-DNA probe biosensor as a highly sensitive system to detect of citrus Huanglongbing," *Scientific Reports*, vol. 14, no. 1, pp. 1-16, 2024, doi: 10.1038/s41598-024-63112-w.
 - [12] V. Fiska *et al.*, "DEMIGOD: A Low-Cost Microcontroller-Based Closed-Loop System Integrating Nanoengineered Sweat-Based Glucose Monitoring and Controlled Transdermal Nanoemulsion Release of Hypoglycemic Treatment with a Software Application for Noninvasive Personalized Diab," *Micromachines*, vol. 15, no. 7, pp. 1-22, Jul. 2024, doi: 10.3390/mi15070887.
 - [13] N. Elnagar, N. Elgiddawy, W. M. A. El Roubi, A. A. Farghali, and H. Korri-Yousoufi, "Impedimetric Detection of Cancer Markers Based on Nanofiber Copolymers," *Biosensors*, vol. 14, no. 2, pp. 1-18, 2024, doi: 10.3390/bios14020077.
 - [14] P. Thongkhao, A. Numnuam, P. Khongkhow, S. Sangkhatat, and T. Phairatana, "Disposable Polyaniline/m-Phenylenediamine-Based Electrochemical Lactate Biosensor for Early Sepsis Diagnosis," *Polymers (Basel)*, vol. 16, no. 4, pp. 1-14, 2024, doi: 10.3390/polym16040473.
 - [15] D. Stoikov *et al.*, "Flow-Through Amperometric Biosensor System Based on Functionalized Aryl Derivative of Phenothiazine and PAMAM-Calix-Dendrimers for the Determination of Uric Acid," *Biosensors*, vol. 14, no. 3, pp. 1-14, 2024, doi: 10.3390/bios14030120.
 - [16] J. I. de O. Filho, M. C. Faleiros, D. C. Ferreira, V. Mani, and K. N. Salama, "Empowering Electrochemical Biosensors with AI: Overcoming Interference for Precise Dopamine Detection in Complex Samples," *Advanced Intelligent Systems*, vol. 5, no. 10, pp. 1-8, Aug. 2023, doi: 10.1002/aisy.202300227.
 - [17] E. Ashoori, D. Goderis, A. Inohara, and A. J. Mason, "Wide Voltage Swing Potentiostat with Dynamic Analog Ground to Expand Electrochemical Potential Windows in Integrated Microsystems," *Sensors*, vol. 24, no. 9, pp. 1-12, 2024, doi: 10.3390/s24092902.
 - [18] G. M. Di Mari *et al.*, "Pain-Free Alpha-Synuclein Detection by Low-Cost Hierarchical Nanowire Based Electrode," *Nanomaterials*, vol. 14, no. 2, pp. 1-14, 2024, doi: 10.3390/nano14020170.
 - [19] C. Psotta, V. Chaturvedi, J. F. Gonzalez-Martinez, J. Sotres, and M. Falk, "Portable Prussian Blue-Based Sensor for Bacterial Detection in Urine," *Sensors*, vol. 23, no. 1, pp. 1-14, 2023, doi: 10.3390/s23010388.
 - [20] N. I. Hossain and S. Tabassum, "A hybrid multifunctional physicochemical sensor suite for continuous monitoring of crop health," *Scientific Reports*, vol. 13, no. 1, pp. 1-19, 2023, doi: 10.1038/s41598-023-37041-z.
 - [21] R. Supriyanti, A. S. Aryanto, M. I. Akbar, E. Sutrisna, and M. Alqaaf, "The effect of features combination on coloscopy images of cervical cancer using the support vector machine method," *IAES International Journal of Artificial Intelligence*, vol. 13, no. 3, pp. 2614-2622, 2024, doi: 10.11591/ijai.v13.i3.pp2614-2622.
 - [22] R. Supriyanti, F. F. R. Wibowo, Y. Ramadhani, and H. B. Widodo, "Comparison of conventional edge detection methods performance in lung segmentation of COVID19 patients," *AIP Conference Proceedings*, vol. 2482, no. 1, p. 100001, Feb. 2023, doi: 10.1063/5.0111683.
 - [23] R. Supriyanti, S. L. Dzihniza, M. Alqaaf, M. R. Kurniawan, Y. Ramadhani, and H. B. Widodo, "Morphological features of lung white spots based on the Otsu and Phansalkar thresholding method," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 33, no. 1, pp. 530-539, Jan. 2024, doi: 10.11591/ijeecs.v33.i1.pp530-539.
 - [24] W. Widanarto *et al.*, "Microwave irradiation-induced yield enhancement of coconut shell biomass-derived graphene-like material," *Physica Scripta*, vol. 99, no. 6, p. 65949, 2024, doi: 10.1088/1402-4896/ad4691.
 - [25] G. Koulouras, S. Katsoulis, and F. Zantalis, "Evolution of Bluetooth Technology: BLE in the IoT Ecosystem," *Sensors*, vol. 25, no. 4, pp. 1-37, 2025, doi: 10.3390/s25040996.

BIOGRAPHIES OF AUTHORS






Retno Supriyanti    is a professor at Department of Electrical Engineering, Jenderal Soedirman University, Indonesia. She received her Ph.D. in March 2010 from Nara Institute of Science and Technology Japan. Also, she received her M.S. degree and Bachelor's degree in 2001 and 1998, respectively, from the Department of Electrical Engineering, Gadjah Mada University, Indonesia. Her research interests include image processing, computer vision, pattern recognition, biomedical application, e-health, tele-health, and telemedicine. She can be contacted at email: retno_supriyanti@unsoed.ac.id.






Wahyu Widanarto    completed his undergraduate degree majoring in Physics at Brawijaya University, his master's degree majoring in Physics at the Bandung Institute of Technology, and his Ph.D. at the Universitaet der Bundeswehr Munich Germany in the field of Institut fuer Physik, Mikrosystemtechnik. His current research is on the development of carbon materials for sensors. He can be contacted at email: wahyu.widanarto@unsoed.ac.id.






Putra Dwi Susanto    received his Bachelor degree from Department of Electrical Engineering, Jenderal Soedirman University Indonesia. His research interest is embedded system. He can be contacted at email: putra.susanto@mhs.unsoed.ac.id.






Madya Ardi Wicaksono    is an academic staff at Public Health and Community Medicine, Jenderal Soedirman University, Indonesia. He received his M.S. degree from Bogor Agricultural University, Indonesia, and graduated as Medical Doctor from Brawijaya University, Indonesia. His research interest including tropical medicine, metabolic diseases, and community nutrition. He can be contacted at email: madya.wicaksono@unsoed.ac.id.



Syafrudin Rais Akhdan    received his bachelor degree from Department of Electrical Engineering, Jenderal Soedirman University, Indonesia; M.S. degree in Nara Institute of Science and Technology, Japan. He can be contacted at email: syafrudin.rais_akhdan.ss2@nais.ac.jp.



Muhammad Alqaaf    received his bachelor degree from Department of Electrical Engineering, Jenderal Soedirman University, Indonesia; M.S. degree in Nara Institute of Science and Technology, Japan. Currently he is a doctor student in Nara Institute of Science and Technology, Japan. His research interest image processing field and bioinformatics. He can be contacted at email: muhammad.alqaaf_subandoko.mb5@is.naist.jp.