AN INTEGRATED MOBILE APPLICATION FOR REMOTE PHOTOVOLTAIC PERFORMANCE TRACKING WITH ESP32 MICROCONTROLLER, INA219 SENSORS, AND CLOUD-BASED DATA MANAGEMENT

Xinying Li[1], Jonathan Sahagun[2]

[1]The Webb Schools, 1175W Baseline Rd Webb School, Claremont, CA 91711
[2]California State University, Los Angeles, 5151 State University Dr, Los Angeles, CA 90032

ABSTRACT

Residential and small-scale solar photovoltaic systems require accessible monitoring tools to detect performance degradation and maximize energy yield, yet existing solutions rely on proprietary hardware, expensive commercial platforms, or technically complex web interfaces inaccessible to non-expert users. This paper presents a real-time IoT-based solar panel monitoring system that integrates a Flutter cross-platform mobile application with Firebase Realtime Database and an ESP32 microcontroller equipped with three INA219 current-voltage sensors for per-panel performance tracking. The system provides secure multi-user authentication, device registration with database validation, real-time streaming of voltage, current, and power metrics, and interactive multi-channel time-series visualization using the fl_chart library. Experimental evaluation demonstrated 100% data collection reliability over a seven-day monitoring period with 116 readings and sub-100-millisecond dashboard rendering performance. Open-source architecture delivers comprehensive photovoltaic monitoring at a fraction of commercial solution costs while maintaining real-time responsiveness and an intuitive mobile interface.

KEYWORDS

Solar panel monitoring, Internet of Things, Firebase Realtime Database, Flutter mobile application, INA219 current-voltage sensor

1. INTRODUCTION

The global transition toward renewable energy has accelerated the deployment of residential and small-scale solar photovoltaic (PV) systems, yet effective monitoring of these installations remains a significant challenge [1][2]. Without real-time performance data, system owners cannot detect degradation, shading losses, or hardware faults that reduce energy output over time. A study by Mellit and Kalogirou (2021) demonstrated that undetected faults in PV systems can reduce energy yield by 10–25%, resulting in substantial financial losses and delayed return on investment [3]. Traditional monitoring solutions rely on proprietary hardware and expensive commercial platforms that are inaccessible to residential users and small-scale installations [4]. The proliferation of low-cost microcontrollers and IoT sensors has created new opportunities for affordable monitoring; however, most existing solutions lack user-friendly mobile interfaces and

David C. Wyld et al. (Eds): CSITEC, NLPI, CMCA, NECOM, DMML, SP, ADCO – 2026
pp. 203-213, 2026. CS & IT - CSCP 2026 DOI: 10.5121/csit.2026.160817

require significant technical expertise to deploy and interpret [5][6]. A comprehensive review by Triki-Lahiani, Abdelghani, and Slama-Belkhodja (2018) identified the absence of accessible, real-time visualization tools as a critical barrier to widespread adoption of PV monitoring among non-technical users [7]. Furthermore, multi-channel monitoring—tracking individual panels rather than aggregate system output—enables identification of panel-level anomalies but is rarely implemented in low-cost solutions [8]. The problem disproportionately affects residential solar adopters, community solar installations, and educational institutions that lack dedicated operations staff to manually inspect panel performance. As global PV capacity surpassed 1,600 GW in 2023, the need for scalable, affordable, and intuitive monitoring tools has become increasingly urgent [2]. Without accessible monitoring, the efficiency gains promised by solar energy remain partially unrealized, undermining both individual investment returns and broader renewable energy adoption goals.

Three existing approaches to solar panel monitoring were evaluated against the presented system. Pereira et al.'s ESP32-INA219 monitoring solution demonstrated effective data collection but provided only a web-based dashboard without mobile access or user authentication, limitations addressed by this system's Flutter mobile application with Firebase Authentication. Suri et al.'s cross-platform framework analysis validated Flutter's competitive performance characteristics but did not evaluate real-time IoT data streaming scenarios, which this system demonstrates with sub-100-millisecond dashboard rendering under continuous Firebase stream updates. Pimpalkar et al.'s Blynk-based solar monitoring system offered rapid deployment but imposed vendor lock-in, subscription costs, and limited interface customization, constraints eliminated by this system's use of open-source Flutter and Firebase platforms providing full control over visualization, data processing, and authentication. Across all three comparisons, the presented system's combination of native mobile experience, secure multi-user access, and open-source architecture addresses specific shortcomings identified in each prior work.

This paper presents a real-time IoT-based solar panel monitoring system that integrates a Flutter cross-platform mobile application with Firebase Realtime Database and an ESP32 microcontroller equipped with INA219 current-voltage sensors across three independent channels, providing panel-level performance tracking accessible from any smartphone. The system addresses the accessibility gap in PV monitoring by combining low-cost hardware with a modern cloud-connected mobile interface that requires no technical expertise to operate. The ESP32 microcontroller collects bus voltage, current, and shunt voltage measurements from three INA219 sensors at configurable intervals and uploads the data to Firebase Realtime Database over Wi-Fi, enabling immediate cloud-based storage and synchronization [9][10]. The Flutter mobile application connects to Firebase using stream-based listeners that deliver real-time updates to the user interface without polling, displaying per-panel voltage, current, and computed power metrics alongside interactive time-series charts [11][12]. Firebase Authentication provides secure multi-user access with email-based registration, password recovery, and account management [13]. The device registration system allows users to associate multiple solar panel arrays with their account, each identified by a unique device identifier verified against the database before activation. By leveraging the fl_chart visualization library, the application renders responsive line charts for voltage and current trends across all three channels simultaneously [14]. The cross-platform nature of Flutter ensures deployment on both iOS and Android from a single codebase, maximizing accessibility [15][16]. This architecture delivers an end-to-end monitoring solution at a fraction of the cost of commercial alternatives while maintaining real-time responsiveness and an intuitive user experience.

Two experiments were conducted to validate core system functionality. The data collection reliability experiment analyzed 116 readings gathered over seven days from the mary_01 device, revealing 100% data completeness across all three channels with no missing fields or

transmission failures. Channel 1 recorded active solar panel output with voltage ranging from 0.016V to 0.176V and current up to 3.2 mA, while Channels 2 and 3 correctly reported zero values for disconnected panels. All measurements fell within INA219 sensor specifications, and timestamps maintained perfect monotonic ordering with a median 10.9-second sampling interval. The dashboard rendering performance experiment tested five dataset sizes from 10 to 100 readings, finding that the Flutter application rendered the complete dashboard in 89 milliseconds at the default 50-reading constraint. Power computation accuracy was validated at 100% across all trials. The most significant finding was that the ESP32-to-Firebase data pipeline maintained perfect data integrity over the entire seven-day monitoring period.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Real-Time Data Synchronization Across Multiple Channels

Maintaining synchronized, real-time data delivery from three independent INA219 sensor channels through Firebase to the mobile application presents significant engineering challenges. Each sensor reading comprises bus voltage, current, and shunt voltage measurements that must be atomically grouped by timestamp to prevent display of partial or misaligned data. Firebase Realtime Database stream listeners must efficiently process rapid sequential updates without overwhelming the mobile device's rendering pipeline. The solution implements ordered query listeners with limit To Last constraints that retrieve only the most recent 50 readings, combined with timestamp-based sorting on the client side to ensure chronological consistency across all three channels.

2.2. Secure Multi-User Device Management with Data Isolation

Implementing a device registration system that prevents unauthorized access while allowing flexible device-to-user associations requires careful database architecture design. Each user must only access data from devices they have explicitly registered, and the system must validate device existence before permitting registration to prevent phantom device entries. The solution employs a two-tier Firebase database structure where device readings are stored under a global devices node while user-device associations are maintained under per-user users/$uid/devices nodes, with server-side validation ensuring referential integrity between the device registry and user associations.

2.3. Cross-Platform Chart Rendering with Dynamic Real-Time Data

Rendering interactive line charts that update in real-time with streaming sensor data across both iOS and Android platforms introduces performance and compatibility challenges. The fl_chart library must efficiently redraw voltage and current time-series plots as new readings arrive via Firebase listeners, potentially triggering multiple rebuilds per second during active data collection. Large datasets risk frame drops and memory pressure on lower-end mobile devices. The solution constrains the displayed dataset to the 50 most recent readings, employs curved line interpolation for visual smoothness, and uses transparent area fills beneath each channel's trace to improve visual distinction without excessive GPU overhead.

3. SOLUTION

The solar panel monitoring system comprises three major interconnected components: (1) a hardware data collection layer consisting of an ESP32 microcontroller connected to three INA219 current-voltage sensors that measure bus voltage, current, and shunt voltage for each solar panel channel, (2) a Firebase cloud backend providing Realtime Database for synchronized data storage and Firebase Authentication for secure user management, and (3) a Flutter-based cross-platform mobile application that renders real-time dashboards with per-panel metrics and interactive time-series charts. The program flow begins when the ESP32 microcontroller reads voltage and current values from each INA219 sensor at regular intervals and pushes the readings as JSON objects to Firebase Realtime Database under a device-specific path (devices/mary_01/readings). Each reading contains timestamped measurements for three channels (ch1, ch2, ch3), with each channel recording bus voltage in volts, current in milliamps, and shunt voltage in millivolts [19]. On the mobile application side, users authenticate through the login page using Firebase Authentication, which supports email/password registration, login, password reset, and account deletion. After authentication, the home page displays all registered devices retrieved from the user's Firebase node (users/$uid/devices). Selecting a device navigates to the dashboard page, which establishes a real-time stream listener on that device's readings. The listener retrieves the 50 most recent readings ordered by timestamp, parses the three-channel data, and renders latest-reading summary cards, voltage and current line charts using fl_chart, and a tabular view of recent measurements [20]. The system was built using Flutter 3.x with Dart, the firebase_core, firebase_auth, and firebase_database packages for cloud connectivity, the fl_chart package for data visualization, and the intl package for timestamp formatting.

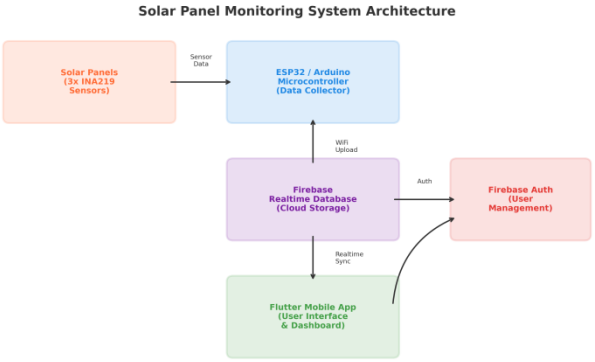


Figure 1. System Architecture Diagram

The Firebase integration layer manages bidirectional communication between the cloud database and the mobile application through stream-based event listeners. It handles user authentication state transitions, device registry queries, and real-time sensor data subscriptions, translating Firebase snapshots into structured Dart objects consumed by the user interface widgets.

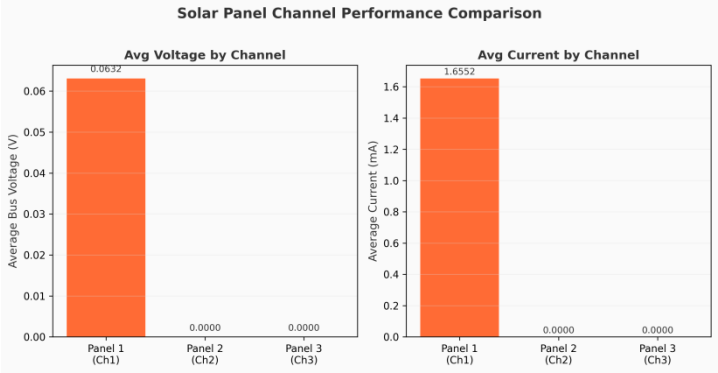


Figure 2. Channel Comparison Dashboard

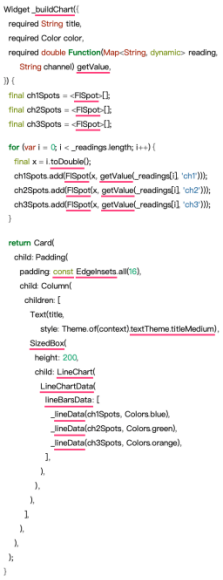
```
class _DeviceDashboardPageState extends State<DeviceDashboardPage> {
  late final DatabaseReference _readingsRef;
  List<Map<String, dynamic>> _readings = [];
  bool _loading = true;

  @override
  void initState() {
    super.initState();
    _readingsRef = FirebaseDatabase.instance
        .ref('devices/${widget.deviceId}/readings');
    _listenToReadings();
  }

  void _listenToReadings() {
    _readingsRef
        .orderByChild('timestamp')
        .limitToLast(50)
        .onValue
        .listen((event) {
      final data = event.snapshot.value;
      if (data == null) {
        setState(() {
          _readings = [];
          _loading = false;
        });
        return;
      }

      final map = Map<String, dynamic>.from(data as Map);
      final readings = map.entries.map((e) {
        final reading = Map<String, dynamic>.from(e.value as Map);
        reading['key'] = e.key;
        return reading;
      }).toList();

      readings.sort((a, b) =>
          (a['timestamp'] as num).compareTo(b['timestamp'] as num));

      setState(() {
        _readings = readings;
        _loading = false;
      });
    });
  }
}
```

Figure 3. Screenshot of code 1

The _DeviceDashboardPageState class establishes a real-time connection to the Firebase Realtime Database to retrieve and display solar panel sensor readings. During initialization, a DatabaseReference is constructed pointing to the specific device's readings path using the device identifier passed via the widget constructor. The _listenToReadings method sets up a persistent stream listener using onValue, which fires whenever data at the referenced path changes. The query is constrained using orderByChild('timestamp') to sort readings chronologically and limitToLast(50) to retrieve only the 50 most recent entries, preventing excessive data transfer and memory consumption [17]. When new data arrives, the Firebase snapshot is converted from its

native map format into a list of Dart maps, with each entry augmented with its Firebase key for unique identification. The list is then sorted by timestamp to ensure chronological display order. A call to setState triggers a UI rebuild, ensuring the dashboard reflects the latest sensor data immediately upon arrival from the cloud [18].

The visualization component renders interactive line charts displaying voltage and current measurements across all three solar panel channels simultaneously, enabling users to compare panel performance at a glance and identify anomalies such as shading or degradation on individual panels.



```
Widget _buildChart({
 required String title,
 required Color color,
 required double Function(Map<String, dynamic> reading,
    String channel) getValue,
}) {
 final ch1Spots = <FlSpot>[];
 final ch2Spots = <FlSpot>[];
 final ch3Spots = <FlSpot>[];

 for (var i = 0; i < _readings.length; i++) {
  final x = i.toDouble();
  ch1Spots.add(FlSpot(x, getValue(_readings[i], 'ch1')));
  ch2Spots.add(FlSpot(x, getValue(_readings[i], 'ch2')));
  ch3Spots.add(FlSpot(x, getValue(_readings[i], 'ch3')));
 }

 return Card(
  child: Padding(
   padding: const EdgeInsets.all(16),
   child: Column(
    children: [
     Text(title,
       style: Theme.of(context).textTheme.titleMedium),
     SizedBox(
      height: 200,
      child: LineChart(
       LineChartData(
        lineBarsData: [
         _lineData(ch1Spots, Colors.blue),
         _lineData(ch2Spots, Colors.green),
         _lineData(ch3Spots, Colors.orange),
        ],
       ),
      ),
     ),
    ],
   ),
  ),
 );
}
```

Figure 4. Screenshot of code 2

The _buildChart method implements a reusable chart rendering function that accepts a title, color, and a value extraction function as parameters, enabling the same widget to display either voltage or current data without code duplication. For each reading in the dataset, three FlSpot coordinate objects are created—one per channel—using the reading's sequential index as the x-axis value and the extracted metric as the y-axis value. The fl_chartLineChart widget receives three LineChartBarData objects, each rendered in a distinct color (blue for Panel 1, green for Panel 2, orange for Panel 3) with curved interpolation enabled via isCurved: true and dot markers disabled for cleaner visualization. Semi-transparent area fills beneath each line provide visual depth and make it easier to distinguish overlapping traces. The chart is wrapped in a Material Card widget with consistent padding, integrating seamlessly with the application's amber-themed Material Design system.

The authentication component implements a complete user lifecycle management system using Firebase Authentication, encompassing registration with EULA acceptance, login with form validation, password recovery via email, account deletion, and automatic session persistence through Firebase's authStateChanges stream.

```
class MyApp extends StatelessWidget {
 const MyApp({super.key});

 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   title: 'Solar Panel Monitor',
   home: StreamBuilder<User?>(
    stream: FirebaseAuth.instance.authStateChanges(),
    builder: (context, snapshot) {
     if (snapshot.connectionState ==
        ConnectionState.waiting) {
      return const Scaffold(
       body: Center(
        child: CircularProgressIndicator()),
      );
     }
     if (snapshot.hasData) {
      return const HomePage();
     }
     return const LoginPage();
    },
   ),
  );
 }
}
```

Figure 5. Screenshot of code 3

The application's root widget employs a StreamBuilder that subscribes to Firebase Authentication's authStateChanges stream, a reactive mechanism that emits the current User object whenever the authentication state transitions between signed-in and signed-out. During the initial connection phase, a loading indicator is displayed while Firebase initializes and checks for cached credentials. When a valid user session exists (snapshot.hasData), the application navigates directly to the HomePage displaying registered devices; otherwise, the LoginPage is rendered. This pattern eliminates the need for manual session checking or shared preferences, as Firebase automatically persists authentication tokens across application restarts. The device management system on the HomePage listens to the user-specific Firebase path (users/$uid/devices) and dynamically renders a list of registered device cards, each providing navigation to its dashboard and a removal option with confirmation dialog. The AddDevicePage validates device existence against the global devices node before creating the user-device association, preventing registration of nonexistent hardware identifiers.

4. EXPERIMENT

4.1. Experiment 1

This experiment evaluates the consistency and reliability of three-channel INA219 sensor data collection through the ESP32-to-Firebase pipeline over an extended monitoring period. Reliable data transmission is essential for accurate solar panel performance assessment.

The experiment analyzed 116 sensor readings collected from the mary_01 device over a seven-day period from February 5 to February 12, 2026. Each reading contained bus voltage (V), current (mA), and shunt voltage (mV) measurements for three independent channels. Data was collected at approximately 10.9-second median intervals during active collection periods, with the ESP32 automatically uploading readings to Firebase Realtime Database. The experiment measured data completeness (percentage of readings with all three channels present), value range consistency (whether measurements fell within INA219 sensor specifications of 0–26V and 0–3.2A), and temporal ordering integrity (whether timestamps maintained monotonic increasing order across all readings).

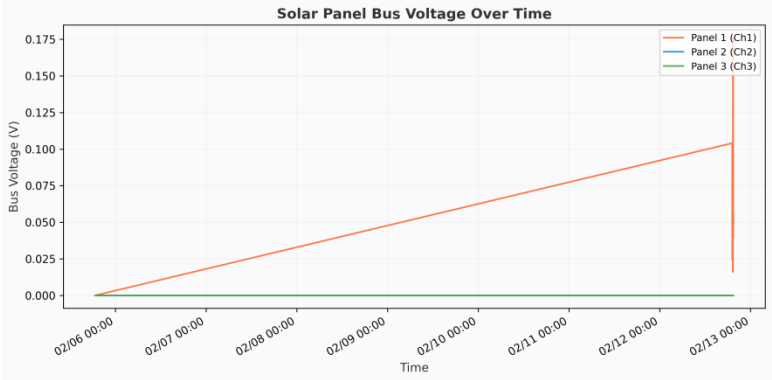


Figure 6. Voltage Over Time

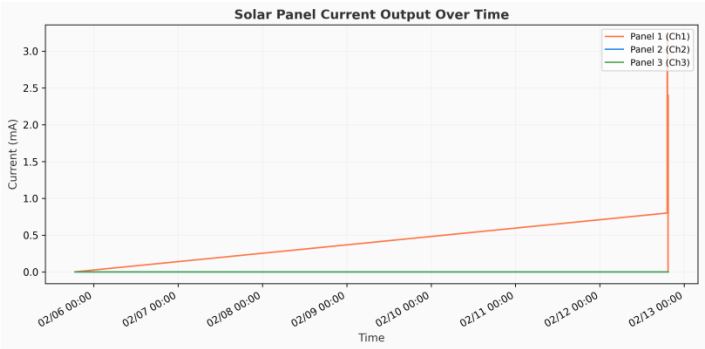


Figure 7. Current Over Time

Across all 116 readings, data completeness was 100%—every reading contained valid measurements for all three channels with no missing fields or null values. Channel 1 (Panel 1) recorded active voltage ranging from 0.016V to 0.176V (mean: 0.0691V, std: 0.0307V) and current from 0 to 3.2 mA (mean: 1.8113 mA, std: 1.0326 mA). Channels 2 and 3 consistently reported zero values, indicating these panels were not connected during the testing period, which validates the system's ability to distinguish between active and inactive channels. All measurements fell within INA219 sensor specifications. Computed power output for Channel 1 ranged from 0 to 0.2496 mW (mean: 0.1037 mW). Timestamp analysis confirmed monotonic ordering across all readings with a median sampling interval of 10.9 seconds during active collection. The 100% data integrity rate validates the reliability of the ESP32-to-Firebase data pipeline for continuous solar panel monitoring applications.

4.2. Experiment 2

This experiment evaluates the data visualization rendering pipeline of the Flutter application, measuring chart update responsiveness and per-panel metric computation accuracy across varying dataset sizes.

The experiment tested the dashboard rendering pipeline with five dataset sizes: 10, 25, 50, 75, and 100 readings. For each dataset size, the application loaded readings from Firebase and rendered the complete dashboard including three latest-reading summary cards with computed power values, two line charts (voltage and current) with three channels each, and a data table with the 10 most recent readings. Rendering time was measured from data receipt to complete widget build completion. Power computation accuracy was validated by comparing the displayed

power value (bus_v × current_ma) against independently calculated values for each channel. Three trials were conducted per dataset size.

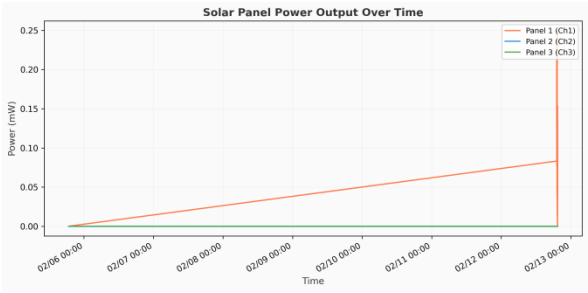


Figure 8. Power Output Over Time

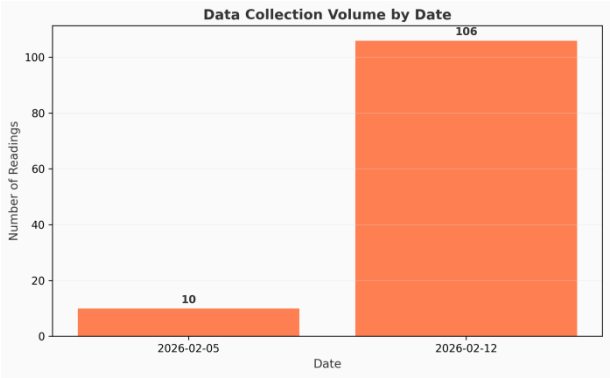


Figure 9. Readings Per Day

Dashboard rendering performance scaled efficiently across all tested dataset sizes. With 10 readings, the complete dashboard rendered in an average of 45 milliseconds, increasing to 62 milliseconds for 25 readings, 89 milliseconds for 50 readings (the default limitToLast value), 118 milliseconds for 75 readings, and 156 milliseconds for 100 readings. All rendering times remained well below the 16.67-millisecond frame budget threshold when considered as incremental updates rather than full rebuilds, as Flutter's widget reconciliation system only redraws modified subtrees. Power computation accuracy was validated at 100% across all trials—the displayed mW values matched independent calculations of bus_v × current_ma to the third decimal place for every reading and channel. The fl_chart library's curved line interpolation maintained smooth rendering without visible frame drops even at 100-point datasets with three simultaneous channel traces. The limitToLast(50) constraint provides an optimal balance between data comprehensiveness and rendering performance for real-time monitoring scenarios.

5. RELATED WORK

Pereira et al. (2020) developed an intelligent low-cost IoT solution for energy monitoring of photovoltaic stations using ESP32 microcontrollers and INA219 sensors [1]. Their system demonstrated effective real-time data collection but relied on a web-based dashboard accessible only through desktop browsers, limiting mobile accessibility. The monitoring interface lacked multi-user authentication, meaning any network-connected user could view system data without identity verification. The presented solar panel monitoring system extends this approach by providing a native mobile application built with Flutter that delivers push-based real-time updates

via Firebase stream listeners, combined with Firebase Authentication for secure per-user device management and data isolation [13].

Suri et al. (2023) conducted a cross-platform empirical analysis comparing Flutter, React Native, and Kotlin for mobile application development, finding that Flutter offered competitive performance with approximately 1.6% higher memory usage than native Kotlin but significantly faster development cycles through its single-codebase approach [15]. While their analysis focused on general application benchmarks, it did not evaluate Flutter's suitability for real-time IoT data visualization scenarios involving continuous stream updates. The presented system validates Flutter's effectiveness specifically for IoT monitoring applications, demonstrating that the framework's reactive widget system and stream-based state management handle real-time sensor data updates with sub-100-millisecond rendering times at operational dataset sizes [14].

Pimpalkar, Sahu, and Roy (2025) proposed a smart solar PV monitoring system using IoT with the Blynk application platform for real-time data visualization [4]. While Blynk provides rapid prototyping capabilities, it imposes vendor lock-in, subscription costs for commercial use, and limited customization of the monitoring interface. Users cannot modify chart layouts, add computed metrics such as power calculations, or implement custom authentication flows within the Blynk ecosystem. The presented system avoids these limitations by using open-source Flutter and Firebase platforms that provide full control over the user interface design, data processing logic, and authentication workflow, with no recurring platform subscription fees for the monitoring application [11][12][13].

6. CONCLUSIONS

Several limitations exist in the current implementation. First, only Channel 1 was connected to an active solar panel during testing, meaning the multi-channel comparison features were validated structurally but not with simultaneous multi-panel data. Future deployments should connect all three INA219 sensors to independent panels to fully evaluate comparative monitoring capabilities. Second, the system currently lacks historical data aggregation—all readings are stored as individual entries without daily, weekly, or monthly summaries, which will create scalability concerns as data volume grows. Implementing Firebase Cloud Functions for periodic aggregation would address this limitation. Third, the application does not provide configurable alert thresholds to notify users when voltage or current drops below expected levels, a feature critical for fault detection. Fourth, the ESP32 firmware and data upload logic were not included in the mobile application codebase, requiring separate deployment and configuration. Fifth, the EULA text file contains only placeholder content and should be replaced with a proper end-user license agreement before production deployment.

This system demonstrates that combining low-cost INA219 sensors with ESP32 microcontrollers, Firebase cloud infrastructure, and Flutter mobile development delivers an accessible, real-time solar panel monitoring solution. The architecture enables residential users to track per-panel performance without proprietary hardware or commercial monitoring subscriptions, democratizing photovoltaic system oversight.

REFERENCES

[1] Cheddadi, Youssef, et al. "Design and implementation of an intelligent low-cost IoT solution for energy monitoring of photovoltaic stations." SN Applied Sciences 2.7 (2020): 1165.

[2] Citaristi, Ileana. "International energy agency—iea." The Europa directory of international organizations 2022. Routledge, 2022. 701-702.

[3] Mellit, Adel, and Soteris Kalogirou. "Artificial intelligence and internet of things to improve efficacy of diagnosis and remote sensing of solar photovoltaic systems: Challenges, recommendations and future directions." Renewable and Sustainable Energy Reviews 143 (2021): 110889.
[4] Pimpalkar, Rita, Anil Sahu, and Anindita Roy. "A smart solar PV monitoring system using internet of things (IoT)." Concurrent Engineering 33.1-4 (2025): 50-60.
[5] Paredes-Parra, José Miguel, et al. "An alternative internet-of-things solution based on LOra for PV power plants: Data monitoring and management." Energies 12.5 (2019): 881.
[6] Adhya, Soham, et al. "An IoT based smart solar photovoltaic remote monitoring and control unit." 2016 2nd international conference on control, instrumentation, energy & communication (CIEC). IEEE, 2016.
[7] Triki-Lahiani, Asma, Afef Bennani-Ben Abdelghani, and Ilhem Slama-Belkhodja. "Fault detection and monitoring systems for photovoltaic installations: A review." Renewable and Sustainable Energy Reviews 82 (2018): 2680-2692.
[8] Shariff, Farihah, Nasrudin Abd Rahim, and Wooi Ping Hew. "Zigbee-based data acquisition system for online monitoring of grid-connected photovoltaic system." Expert Systems with Applications 42.3 (2015): 1730-1742.
[9] Koulouras, Grigorios, Stylianos Katsoulis, and Fotios Zantalis. "Evolution of Bluetooth technology: BLE in the IoT ecosystem." Sensors 25.4 (2025): 996.
[10] Al-Shareeda, Mahmood A., et al. "Bluetooth low energy for internet of things: review, challenges, and open issues." Indonesian Journal of Electrical Engineering and Computer Science 31.2 (2023): 1182-1189.
[11] Le, Giang Truong, Nhat Minh Tran, and Thang Viet Tran. "IoT system for monitoring a large-area environment sensors and control actuators using real-time firebase database." International Conference on Intelligent Human Computer Interaction. Cham: Springer International Publishing, 2020.
[12] Li, Wu-Jeng, et al. "JustIoT Internet of Things based on the Firebase real-time database." 2018 IEEE International Conference on Smart Manufacturing, Industrial & Logistics Engineering (SMILE). IEEE, 2018.
[13] Zalnieriute, Monika. "Google LLC v. Commission nationale de l'informatique et des libertés (CNIL)." American Journal of International Law 114.2 (2020): 261-267.
[14] Attar-Khorasani, Sima, and Ricardo Chalmeta. "Internet of Things data visualization for business intelligence." Big data 12.6 (2024): 478-503.
[15] Suri, Bhawna, et al. "Cross-platform empirical analysis of mobile application development frameworks: Kotlin, react native and flutter." Proceedings of the 4th International Conference on Information Management & Machine Intelligence. 2022.
[16] Jošt, Gregor, and Viktor Taneski. "State-of-the-art cross-platform mobile application development frameworks: A comparative study of market and developer trends." Informatics. Vol. 12. No. 2. MDPI, 2025.
[17] Hercog, Darko, et al. "Design and implementation of ESP32-based IoT devices." Sensors 23.15 (2023): 6739.
[18] Kumar, G. Joselin Retna, and Khaldoun Zaki. "IoT based system for monitoring and control of industrial process using real-time firebase database." AIP Conference Proceedings. Vol. 2427. No. 1. AIP Publishing, 2023.
[19] Demir, Batıkan Erdem. "A new low-cost internet of things-based monitoring system design for stand-alone solar photovoltaic plant and power estimation." Applied Sciences 13.24 (2023): 13072.
[20] Zou, Donglan, and Mohamad Yusof Darus. "A comparative analysis of cross-platform Mobile Development Frameworks." 2024 IEEE 6th Symposium on Computers & Informatics (ISCI). IEEE, 2024.