haute école arc ingénierie
neuchâtel berne jura saint-imier le locle delémont

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Bachelor's in computer science and communication systems

Orientation : Software Computing

Collecte données techniques

By :

Nouidri Sami

Under the supervision of :
Prof. Marina Ninoslav
Haut Ecole Arc, HES-SO

St-Imier, HES-SO // P3, the 23 January 2024

ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude to NPS Information Systems Sàrl for their invaluable contribution and collaboration on this project. It was the company's initial proposal and vision that laid the foundation for this endeavor, providing us with a unique opportunity to delve into the realms of mobile applications for an industrial context & clientele.

Working alongside the team at NPS Information Systems Sàrl has been a remarkable journey of learning and mutual growth. Their expertise, insights, and commitment have been instrumental in guiding the project towards success. This collaboration has not only enriched my experience but has also significantly contributed to the project's achievements.

I would also like to express my sincere appreciation to M. Marina Ninoslav for his guidance and supervision throughout the development of this project, as his insightful feedback has been invaluable in steering the project towards its successful completion.

1 SUMMARY

The project, initiated by NPS Information Systems Sàrl and supervised by M. Marina Ninoslav, aimed to develop a specialized Android application using the Flutter framework to streamline measurement collection in industrial settings. The core functionality centered on using QR codes to identify machines or specific points within a facility, allowing users to scan these codes and enter corresponding measurements. This system was designed to replace traditional manual entry methods, thus improving accuracy and efficiency in data collection processes.

At its core, the application features an intuitive QR code scanning system, designed to rapidly and accurately gather data from specific machinery or checkpoints within an industrial environment. This approach significantly reduces the likelihood of manual entry errors and enhances the efficiency of the data collection process.

To ensure the reliability and accuracy of the data entered, A data validation routine was implemented. This routine meticulously checks the captured measurements against predefined minimum and maximum thresholds, guaranteeing that the data adheres to established industry standards before it is stored.

Central to the application's functionality is its integration with a cloud-based database using Firebase, which offers a secure and centralized platform for data storage and retrieval. Further enhancing the application's utility is a robust user authentication system, also powered by Firebase, which supports multiple user accounts and prevents unauthorized access, thereby bolstering security.

After months of development, the resulting application, whilst lacking some nice-to-have features, is a solid foundation to be expanded upon after we've gathered feedback, following its deployment. As of now, core systems, such as QR Code scanning and data validation routines have been proven to work in a test environment.

In summary, this project has successfully demonstrated the effectiveness of integrating modern technology into industrial operations. While it has achieved most of its goals, there is always room for further enhancements, such as the proposed addition of biometric authentication. The collaboration with NPS Information Systems Sàrl and the guidance of M. Marina Ninoslav have been pivotal in the project's success, and the results obtained serve as a testament to the hard work and dedication of all parties involved.

2 TABLE OF CONTENTS

Acknowledgements 1
1 Summary 2
3 Introduction 5
3.1 Objectives 6
3.1.1 Main 6
3.1.2 secondary 6
3.2 constraints 7
4 Problem Analysis 8
5 Project Analysis 9
5.1 main objectives 9
5.1.1 QR Code Management 9
5.1.2 Data Validation 9
5.1.3 Cloud data collection & database management 10
5.1.4 Authentication & user management 10
5.2 Secondary objectives 11
5.2.1 Daily round statistics calculated & displayed 11
5.2.2 Measurement auto-completion & mini-grid displaying DATA history 11
5.2.3 Biometric authentication 12
6 State of the art 13
7 Conception 15
7.1 UML diagram 15
7.2 Database schematic 17
7.3 Design Mockups 19
7.4 QR Code format choice 20
8 Implementation 21
8.1 Main Objectives 21
8.1.1 User Interface 21
8.1.2 QR Code management 23
8.1.3 Data Validation 24
8.1.4 Cloud data collection & Database management 26
8.1.5 Authentication & User management 27
8.2 Secondary Objectives 29
8.2.1 Daily round statistics & graph display 29
8.2.2 Measurement auto-completion & mini-grid display 32
9 Planning 33

10 Conclusion 34
11 Bibliography 35
12 Date and signature 36

3 INTRODUCTION

This report provides a detailed analysis of the “Collecte donnée techniques” project, focusing primarily on the utilization of QR code technology for efficient data collection and management in industrial environments. The report is structured to offer a comprehensive overview, beginning with an in-depth analysis of the project's goals and client demands, which serve as the foundational elements guiding the project's trajectory. We delve into the initial conception phase, where innovative ideas were conceptualized to address specific challenges in data acquisition and management, particularly in settings such as incineration plants where accurate data collection is crucial.

Following the conceptual framework, the report meticulously documents the implementation process. This section details the technical aspects of the project, from the integration of QR codes for capturing essential information at various operational stations to the development of robust verification routines ensuring data accuracy. Emphasis is placed on the user interface design and the application's functionality, which includes features like real-time data validation against predefined parameters and historical data analysis.

The conclusion of the report synthesizes the key findings and outcomes, offering a critical assessment of the project's success in meeting its initial objectives and client expectations. It also discusses potential areas for future enhancements, such as the incorporation of biometric authentication for improved security measures, as suggested by educational advisors.

3.1 OBJECTIVES

Below is our list of objectives and client demands, separated into two categories.

3.1.1 MAIN

1. QR Code management
2. Data Validation
3. Cloud data collection & database management
4. Authentication & user management

3.1.2 SECONDARY

1. Daily Round statistics calculated & displayed.
2. Measurement auto-completion & Mini-grid displaying data history.
3. Biometric authentication

3.2 CONSTRAINTS

This project is governed by a set of specific constraints that significantly influence its development and final outcome. A notable constraint is the use of Flutter, a robust and versatile framework developed by Google, as the primary technology for building the application. Flutter's ability to deliver high-fidelity, platform-specific UIs with a single codebase is instrumental in meeting our development goals efficiently. However, this choice also introduces certain limitations, particularly in terms of platform-specific functionalities and integration with third-party services.

Additionally, the project is targeted specifically at Android devices, which dictates our approach to UI design, performance optimization, and testing. This Android-centric focus enables us to leverage platform-specific features and optimizations but also means that the application's compatibility with other platforms, like iOS or web, is not a priority in the current development phase. These constraints, while challenging, also provide a clear direction and scope, ensuring a focused and efficient development process.

4 PROBLEM ANALYSIS

In the industrial maintenance sector, the traditional practice of manually recording machine measurements presents significant challenges, a problem our application seeks to address. Employees, tasked with conducting daily rounds to monitor facility machines, often relied on handwritten notes or Excel sheets for data recording. This method, while straightforward, was fraught with inefficiencies and potential for error.

The manual recording process was notably time-consuming and susceptible to inaccuracies. Handwritten data posed risks of illegibility and transcription errors, while manual entry into Excel sheets was prone to numerical mistakes. These inaccuracies held considerable implications, especially since such measurements often serve to assure insurance companies of the facility's proper maintenance. Any errors in data could lead to incorrect assessments of the facility’s condition, potentially causing compliance issues or oversight in maintenance needs.

Additionally, the manual method significantly impacted employee productivity. The labor-intensive process of writing down each measurement and subsequently transferring these records into digital formats for reporting consumed valuable time that could have been allocated to other maintenance or operational tasks. This inefficiency was compounded in facilities with numerous machines, where the data recording process became even more cumbersome.

Our application introduces a transformative solution by digitizing and streamlining the data collection process. With the app, employees can swiftly scan QR codes on machines, accurately capturing machine information, and enter measurements directly into their mobile devices. This approach not only minimizes the risk of human error but also facilitates immediate data upload to a centralized database.

The app's real-time data entry and processing capabilities allow for quicker decision-making and trend analysis, essential for maintaining insurance compliance. It enhances operational efficiency by reducing the time spent on manual data recording, thereby improving overall productivity. Furthermore, the app provides easy access to historical data, aiding in tracking maintenance history and identifying long-term trends.

In essence, our application effectively tackles the challenges associated with traditional machine measurement recording practices. By offering a digital and automated alternative, it not only ensures accuracy and reliability of data but also contributes to enhanced operational efficiency and informed decision-making in industrial maintenance.

5 PROJECT ANALYSIS

In this section, we will explore in detail the decisions made, the approaches adopted, and the key considerations for each objective. From data reading to performance evaluation, this analysis provides an in-depth insight into the strategic choices that have shaped the project's direction.

5.1 MAIN OBJECTIVES

5.1.1 QR CODE MANAGEMENT

OBJECTIVE

The objective is to effectively manage and interpret data from QR codes scanned at various operational points, such as machinery or stations within an industrial setting.

APPROACH

Implement a QR code scanning system within the app that can accurately decode and extract relevant information.

DECISIONS TAKEN

1. Chose a reliable and efficient QR code scanning library compatible with Flutter and optimized for Android devices (qr_code_scanner)
2. Dedicated an entire page to scanning and displaying QR code information.
3. Defined a string format for validation.

5.1.2 DATA VALIDATION

OBJECTIVE

To ensure the integrity and accuracy of the data entered by users after scanning QR codes.

APPROACH

Develop validation routines that compare user-entered data against predefined thresholds and parameters.

DECISIONS TAKEN

1. Added validation and comparison with Database thresholds when submitting a measurement
2. Verify the QR Code’s contents, such that it matches our pre-defined format.

5.1.3 CLOUD DATA COLLECTION & DATABASE MANAGEMENT

OBJECTIVE

Centralize data storage and management in a cloud database, facilitating access and manipulation of collected data.

APPROACH

Utilize cloud-based database solutions for storing and retrieving data efficiently.

DECISIONS TAKEN

1. Opted for Firebase as the backend database due to its compatibility with Flutter and ease of integration for real-time data operations.

5.1.4 AUTHENTICATION & USER MANAGEMENT

OBJECTIVE

Implement a secure system for user authentication, to prevent unauthorized access by non-personnel members. Such system should have the possibility to be expanded with additional features, such as e-mail password reset and access roles.

APPROACH

Integrate a robust authentication system that supports multiple users.

DECISIONS TAKEN

1. Chose Firebase Authentication to manage user access, offering features like email-password login and account recovery options.

5.2 SECONDARY OBJECTIVES

5.2.1 DAILY ROUND STATISTICS CALCULATED & DISPLAYED

OBJECTIVE

Provide users with statistical feedback on their data collection rounds, such as time taken and deviation from standard values.

APPROACH

Implement analytics to calculate and display relevant statistics post data entry rounds.

DECISIONS TAKEN

1. Chose a reliable charting library for Flutter (fl_chart).
2. Developed an in-app analytics dashboard that presents calculated statistics in an easy-to-understand format.

5.2.2 MEASUREMENT AUTO-COMPLETION & MINI-GRID DISPLAYING DATA HISTORY

OBJECTIVE

Enhance user experience by suggesting previous measurements for auto-completion and displaying recent data entries.

APPROACH

Incorporate features that recall and suggest past data entries, and display a mini-grid of historical data for quick reference.

DECISIONS TAKEN

1. Displayed a card with the three last measurement entries for a given data point on the QR scan page.

5.2.3 BIOMETRIC AUTHENTICATION

OBJECTIVE

Increase the security of the app by adding an additional layer of biometric authentication.

APPROACH

Implement a biometric authentication feature that requires user verification through fingerprints or facial recognition.

DECISIONS TAKEN

1. Researched various ways to implement biometric authentication in Flutter.

6 STATE OF THE ART

In the realm of industrial measurement and data collection, the integration of advanced technologies has significantly transformed operational methodologies. The current landscape is marked by several pioneering applications, each contributing uniquely to this evolving field.

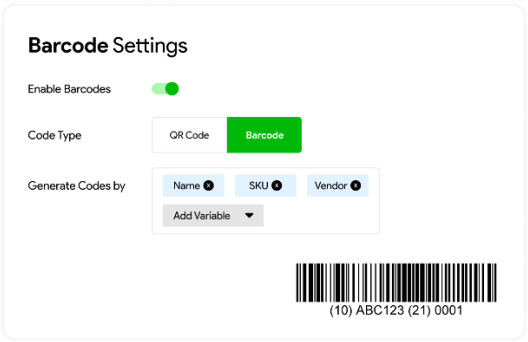

Figure 1 "IndusTrack's QR Code management system"

A notable trend among these applications is the adoption of QR code technology, which has revolutionized the way data is collected in industrial settings[1]. This shift towards QR codes signifies a broader movement towards efficient and paperless data capture methods. Applications like "IndusTrack" (specifically its inventory management systems) have been at the forefront of incorporating QR codes, specifically tailored to cater to the unique needs of sectors such as manufacturing and logistics.

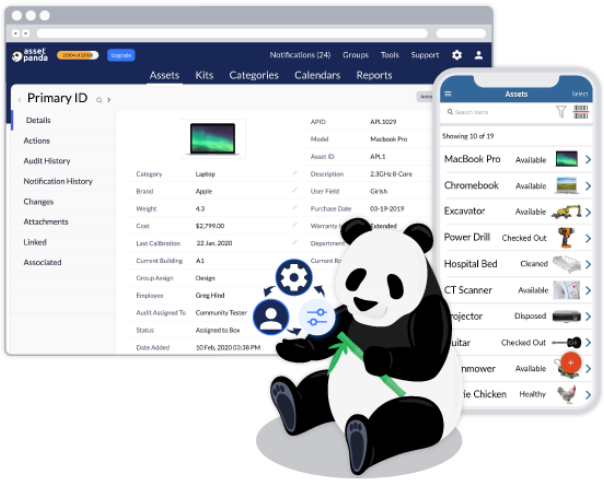

Figure 2 "AssetPanda's user interface"

Another key development is the use of cloud-based platforms for data management[2]. This approach, embraced by applications like "AssetPanda" and "Fiix," has enabled scalable storage solutions, real-time data access, and centralized data management. These features have been instrumental in enhancing coordination and decision-making processes across various industrial operations.

[1] *QR Codes for manufacturing, qr-code-generator, https://www.qr-code-generator.com/blog/qr-code-for-manufacturing/*

[2] *Advantages of cloud-based asset/data management, AssetPanda, https://www.assetpanda.com/resource-center/blog/what-are-the-advantages-of-cloud-based-asset-management/*

Figure 3 "Technical Data collection app using NFC tags"

The impact of these software solutions on industrial operations cannot be overstated. Their adoption has led to notable improvements in operational efficiency, data accuracy, and the overall decision-making process. In addition, **it is interesting to note that a similar application was developed by NPS Information Systems, relying on NFC tags instead of QR Codes**. Reduced manual errors, expedited data collection, and enhanced data-driven insights are among the key benefits that users have reported after switching to said application.

As the industry continues to advance, future developments in these applications are expected to delve deeper into automation, incorporating AI for predictive analytics, and improving IoT integration. This ongoing evolution is set to further revolutionize the landscape of industrial operations and data management, underlining the importance of these software solutions in the modern industrial milieu.

7 CONCEPTION

In this section, we will go over the various conceptual documents, used to define the project's architecture and eventual look.

7.1 UML DIAGRAM

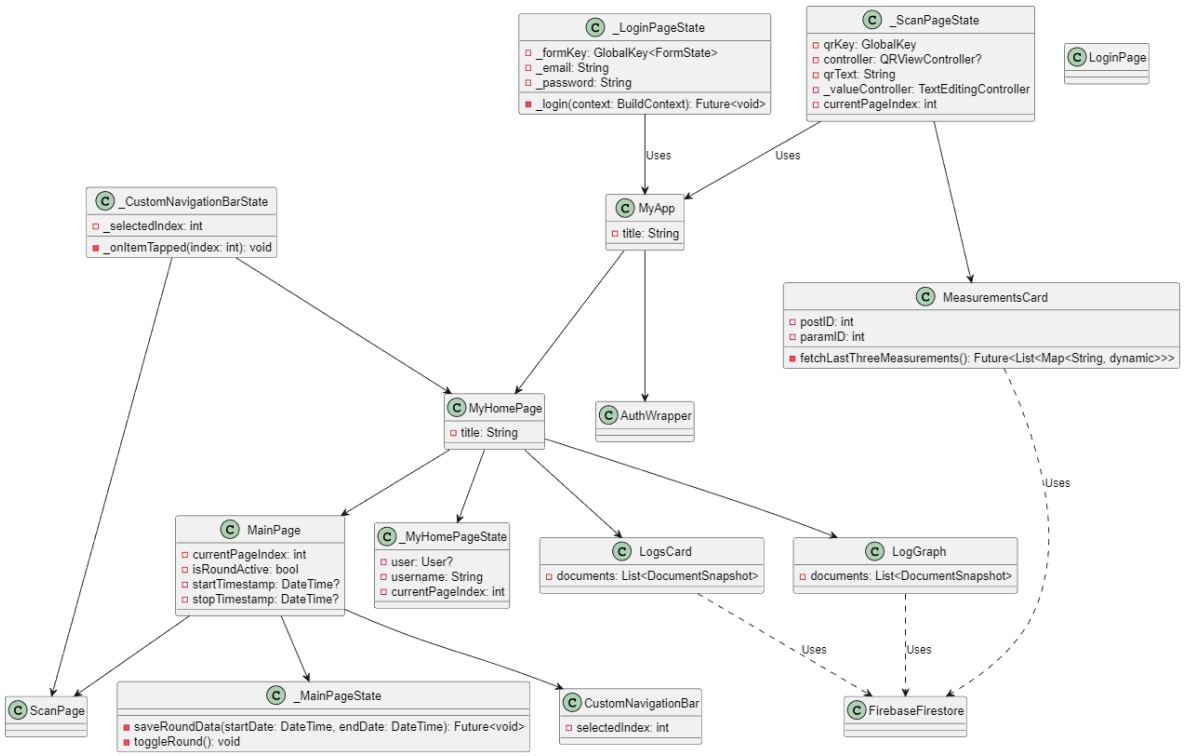

Figure 4 "UML Class diagram for this project"

This project architecture is characterized by a clear division of responsibilities across various classes, each serving a distinct role within the application's ecosystem.

At the top level, "**MyApp**" serves as the entry point of the application. It's a fundamental class that sets up the initial configuration, including the title and the home route. The application structure is defined here, with routes to different pages like MyHomePage, AuthWrapper, and LoginPage. This class acts as a centralized orchestrator, directing the flow to various sections of the app based on user interaction and authentication status.

MyHomePage is a critical component of the application, acting as the first interface displayed when starting the app. It's connected to the LogsCard and LogGraph widgets, which are responsible for displaying data in different formats. The LogsCard class presents presents a view of the last recorded round statistics, offering a concise and readable summary, while the LogGraph provides a graphical representation of the average measurement taken, such that the user can easily discern if a round's measures have been within the usual range.

The **AuthWrapper** class plays a pivotal role in managing user authentication. It utilizes FirebaseAuth to handle user sessions, determining whether to present the user with a login interface or direct them to the main content of the app. This mechanism ensures that user access is controlled and secure.

The **MainPage** class, along with its state _MainPageState, is central to navigation and user interaction within the app. It facilitates the switching between different pages like ScanPage and the home page. The

CustomNavigationBar and its state _CustomNavigationBarState further augment this navigation by providing a user-friendly bottom navigation bar, enabling seamless transitions between different sections of the application.

In the context of functionality, **ScanPage** and its state _ScanPageState are crucial for the QR code scanning feature. This class manages the process of scanning QR codes, extracting relevant data, and handling user input for measurement values. It interacts closely with the **MeasurementsCard** widget, which is designed to fetch and display the last three measures for a given data point, giving the user an idea of what values are expected.

The **LoginPage** and its state _LoginPageState handle user logins, interfacing with Firebase's authentication services. This class ensures that user credentials are correctly managed and authenticated, providing a secure entry point to the app.

7.2 DATABASE SCHEMATIC

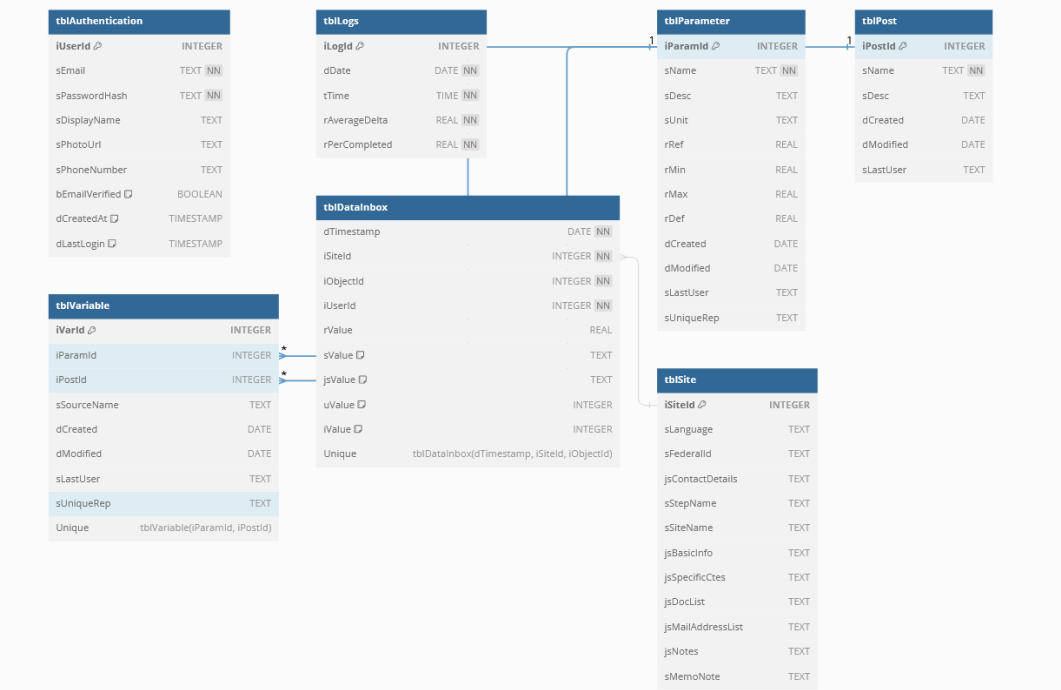

Figure 5 "Database Schematic"

The database structure for thisn application, comprising tables like TBL_DATAINBOX, TBL_LOGS, TBL_PARAMETER, and TBL_POST, reflects a well-considered approach aligned with the project's constraints and conceptual design. It is important to note that this layout is standard across NPS Information Systems' other applications, and has been slightly modified to accommodate the demands of this particular project.

TBL_DATAINBOX:

This table is pivotal for real-time data collection. Each entry, identified by dTimeStamp, represents a unique measurement instance, capturing the essential IDs (iParamID, iPostID, iSiteID, iUserID) that link the data to specific parameters, posts, sites, and users. The rValue field holds the actual measurement, while jsValue and sValue are reserved for additional data, offering flexibility for future enhancements.

TBL_LOGS:

This table plays a crucial role in aggregating and analyzing data over time. It records the start (dStartDT) and end (dEndDT) timestamps of data collection rounds, providing a clear timeframe for each set of measurements. The rAverageDelta and rPerCompleted fields are user for evaluating performance and operational efficiency, offering insights into average deviations and task completion rates. The design of TBL_LOGS aligns with the project's secondary objectives of providing users with actionable insights and statistical overviews of their data collection rounds.

To note : rPerCompleted has been kept, but it's underlying mechanic hasn't been implemented in the app's current version.

TBL_PARAMETER:

The structure of TBL_PARAMETER underpins the application's data validation and standardization features. Each parameter, uniquely identified by iParamID, includes details like the data parameter's name (sName), unit, and range values (rMin, rMax, rDef, rRef). This setup is crucial for ensuring that measurements collected are within acceptable ranges, enhancing the reliability of the data. The creation (dCreated) and modification (dModified) timestamps help in maintaining the record's integrity and tracking changes over time.

TBL_POST:

TBL_POST serves as a reference table for various posts or stations within the industrial environment. Each post, identified by iPostID, includes a name (sName) and an optional description (sDesc). The creation and modification timestamps (dCreated, dModified) are consistent with the other tables, maintaining a uniform approach to data management. This table is essential for contextualizing the measurements collected, linking them to specific locations or equipment within the facility.

7.3 DESIGN MOCKUPS

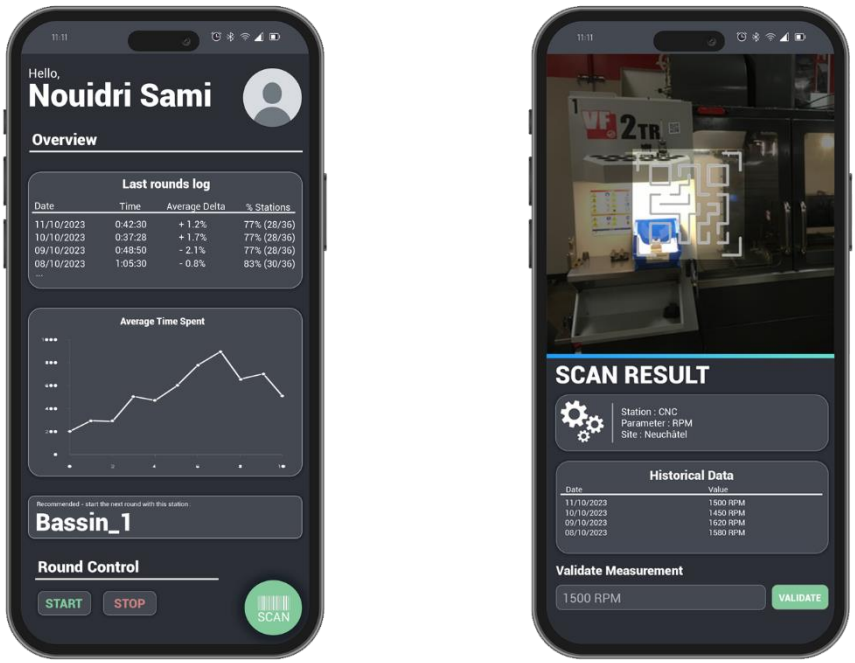

Figure 6 "Design mockups"

The following design mockups illustrate the initial component layout concept for this project. On the right, the home page's purpose is to give a general overview of the user's analytics and past activity, so as to get an idea of was previously done. In addition, round control was placed at the bottom, with the idea being that the user would start/stop rounds much like a chronometer, with two separate buttons. The "scan" button would serve as a navigation point to the scan page.

On the left, the scan page's "dual" layout was defined, with the top portion dedicated to a camera display with a QR Code outline, and the bottom portion used to display information about the scanned data point. Beneath all of this, and having read the data displayed, the user would then enter the measurement data in the given text box, and validate/send it with the "validate" button.

As previously mentioned, the home page's components include a view of past activity, in addition to a graph displaying the average measurement value taken, to determine whether a particular round had higher or lower than usual deviancy in the values measured. The "next post" component was envisioned as a way to suggest the nearest station or measurement point to start the round, but was eventually dropped, as it wasn't deemed a priority, especially when considering the time needed to implement it.

The scan page's displayed information includes the post and parameter ID, as well as site (for multi-site installations). In addition, a view of the last measurement entries for said post is displayed, so as to give an idea of what values should be entered.

7.4 QR CODE FORMAT CHOICE

Figure 7 "Model 1 QR Code used for testing"

In the industrial domain, QR codes are a pivotal tool, offering versatility and efficiency for tasks ranging from inventory management to equipment monitoring. The adoption of QR codes in our Flutter-based application was a strategic decision, influenced by the diverse QR code formats available and their respective features. Our application specifically employs the Model 1 QR code standard, a choice driven by the unique requirements of our project.

Model 1 QR codes, the original format, are characterized by their square shape and ability to encode a limited amount of data[3]. Despite the advent of more advanced formats like Model 2, which offers greater data capacity, and Micro QR codes, designed for space-constrained applications, the Model 1 format stood out for its simplicity and wide compatibility. Other formats, such as QR codes and Frame QR, although offering flexibility in size and additional branding opportunities, were not deemed necessary for our application's objectives.

Our selection of Model 1 QR codes was primarily guided by considerations of simplicity and compatibility. The straightforward structure of Model 1 QR codes ensures seamless compatibility with a wide range of scanning devices and software, a crucial factor for ensuring the application's accessibility and usability. The data capacity of Model 1 QR codes, while less than that of Model 2, is sufficient for our application, as it primarily encodes IDs and basic parameter information.

[3] *QR Code formats, qrcode.com, https://www.qrcode.com/en/codes/*

8 IMPLEMENTATION

In this section, we will go over the various ways in which we implemented the different tasks needed to accomplish the project's intended objectives.

8.1 MAIN OBJECTIVES

8.1.1 USER INTERFACE

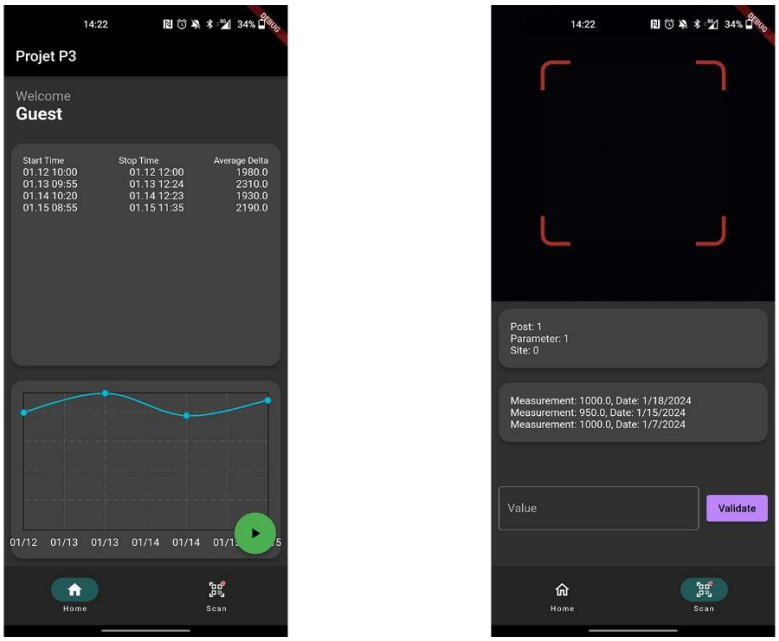

Figure 8 "Final result of interface elements"

One of the first tasks we've set out to do, was translate the design mockups into Flutter interface elements, so as to get a good feel of what remains to be done during development. Initially, this is done by adding empty CardViews to the home screen, which then helps define the various layout constraints needed to match the design mockup (such as Expanded, SingleChildScrollView, and other front-end spacers/components).

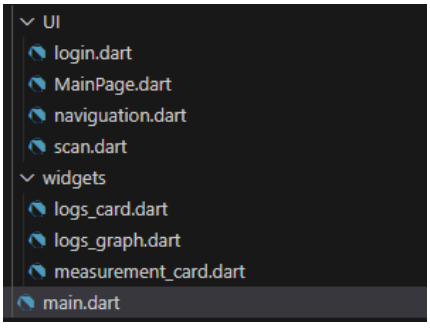

Figure 9 "Project Dart file structure"

At first, the green "play" button, which acts as a round control, was tied to a "main" dart script which acted as the application's home screen. Eventually, we had to create a separate *MainPage* to house the button, which acted as a container for *Main.dart* and *Scan.dart* pages, and was instrumental in keeping data consistency between pages. This allowed us to keep the start/stop mechanic on the home page, and simply hide the button when another page is shown.

8.1.2 QR CODE MANAGEMENT

```dart
class _ScanPageState extends State<ScanPage> {
  final GlobalKey qrKey = GlobalKey(debugLabel: 'QR');
  QRViewController? controller;
  String qrText = '';
  final TextEditingController _valueController = TextEditingController();
  int currentPageIndex = 0;

  @override
  void reassemble() {
    super.reassemble();
    if (Platform.isAndroid) {
      controller!.pauseCamera();
    }
    controller!.resumeCamera();
  }
```

Figure 10 "Scan page declaration"

The core functionality of QR Code management resides within the *ScanPage* class and its state, *_ScanPageState*. It utilizes the *QRViewController* from the *qr_code_scanner* package, a crucial element that enables the scanning and decoding of QR codes. When a QR code is scanned, the *QRViewController* captures the data encoded within the QR code and updates the *qrText* variable in *_ScanPageState*. This variable, which holds the decoded information from the QR code, typically contains essential details such as the verification stamp (“ClariusDP”), ID of the post (*iPostID*), the parameter ID (*iParamID*), and the site ID (*iSiteID)*.

```dart
if (qrText.isNotEmpty)
  Padding(
    padding: const EdgeInsets.all(10.0),
    child: Row(
      children: [
        Expanded(
          //entrée de la valeur
          child: TextField(
            controller: _valueController,
            decoration: const InputDecoration(
              labelText: 'Value',
              border: OutlineInputBorder(),
            ),
          ),
        ),
        const SizedBox(width: 10),
        ElevatedButton(
          onPressed: _submitData,
          child: const Text('Validate'),
        ),
      ],
    ),
  ),
```

Figure 11 "Validation text field"

The ScanPage class also includes a TextEditingController, _valueController, which allows users to input additional data related to the scanned QR code, such as measurement values. This user input, coupled with the information decoded from the QR code, forms the complete data set that is eventually processed and stored in the database.

8.1.3 DATA VALIDATION

```dart
//Soumet les données dans la table TBL_DATAINBOX
void _submitData() async {
  try {
    // Extract parameter ID and measurement value
    List<String> dataParts = qrText.split(';');
    int paramID;
    int postID;

    try {
      postID = int.parse(dataParts[1]);
      paramID = int.parse(dataParts[2]);
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Invalid parameter ID')),
      );
      return;
    }
    double rValue = double.tryParse(_valueController.text) ?? 0.0;

    // Fetch parameter constraints
    var paramConstraints = await fetchParameterConstraints(paramID);
    if (paramConstraints != null) {
      double rMin = (paramConstraints['rMin'] as num?)?.toDouble() ??
          double.negativeInfinity;
      double rMax =
          (paramConstraints['rMax'] as num?)?.toDouble() ?? double.infinity;

      // Check if the measurement is within constraints
      if (rValue < rMin || rValue > rMax) {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text('Measurement out of bounds')),
        );
        return;
      }
    }
```

Figure 12 "_submitData function for data validation"

The data validation process is primarily handled in the _submitData method of _ScanPageState. This method is invoked when a user submits a measurement after scanning a QR code. The first step in the data validation process involves parsing the scanned data to extract relevant IDs, such as paramID and postID, which are crucial for linking the measurement to the correct parameter and post in the database.

```dart
//Recupère les contraintes du paramètre
Future<Map<String, dynamic>?> fetchParameterConstraints(int paramID) async {
  try {
    var snapshot = await FirebaseFirestore.instance
        .collection('TBL_PARAMETER')
        .where('iParamID', isEqualTo: paramID)
        .limit(1)
        .get();

    if (snapshot.docs.isNotEmpty) {
      return snapshot.docs.first.data();
    } else {
      throw new Exception('Parameter not found');
    }
  } catch (e) {
    print('Error fetching parameter constraints: $e');
    return null; // Error occurred
  }
}
```

Figure 13 "fetchParameterConstraints function"

Once the relevant IDs are extracted, the _submitData method calls fetchParameterConstraints, a function that retrieves the parameter constraints from the TBL_PARAMETER table in the database. This table stores key information about each parameter, including minimum (rMin) and maximum (rMax) allowable values.

After fetching the constraints, the _submitData method then compares the user-entered measurement value (rValue) against these constraints. If the value falls outside the defined rMin and rMax range, the method prompts the user with an error message, indicating that the measurement is out of bounds. This check ensures that only valid data that meets the predefined criteria is accepted, thereby maintaining the reliability and quality of the data stored in the database.

```
    // Proceed with data submission
    await FirebaseFirestore.instance.collection('TBL_DATAINBOX').add({
      'dTimeStamp': Timestamp.now(),
      'iPostID': postID,
      'iParamID': paramID,
      'iSiteID': 0,
      'iUserID': 0,
      'jsValue': '',
      'rValue': rValue,
      'sValue': '',
    });

    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Data submitted successfully')),
    );
    setState(() {
      qrText = '';
      _valueController.clear();
    });
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Error submitting data: $e')),
    );
  }
}
```

Figure 14 "Data submission in _submitData"

In the event that the measurement passes the validation checks, it is then saved to the TBL_DATAINBOX table, along with other relevant details like the timestamp, user ID, and the post and parameter IDs. This approach to data validation not only enforces data integrity but also enhances the overall usability of the application by providing immediate feedback to the user on the validity of their input.

8.1.4 CLOUD DATA COLLECTION & DATABASE MANAGEMENT

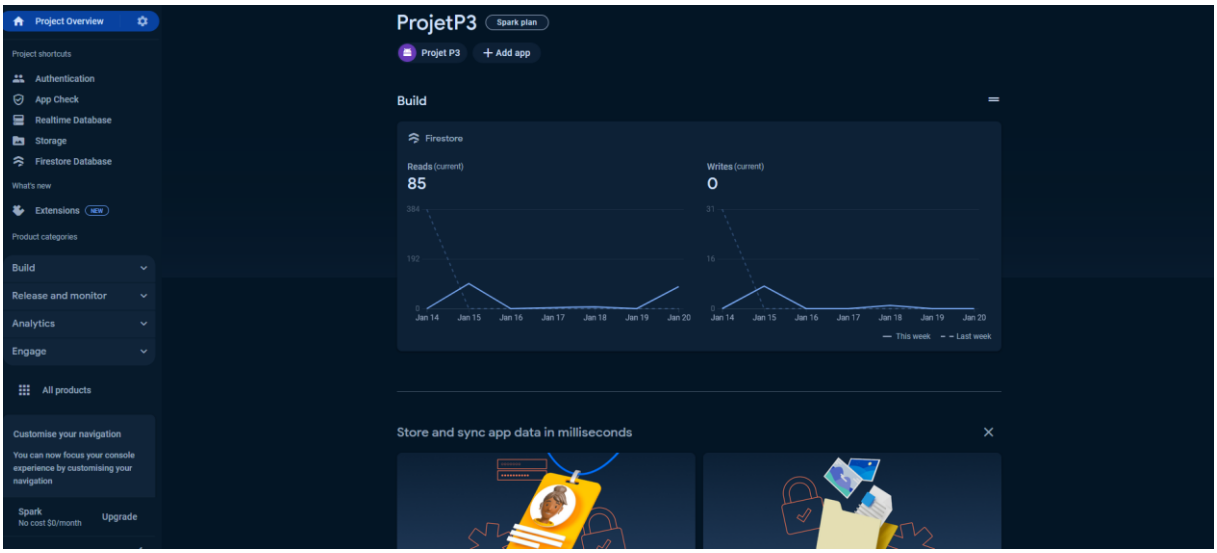

Figure 15 "Firebase console interface"

Before delving into code, establishing a link to Google's Firebase DB starts on the Firebase console. This initial step involves navigating to the Firebase website, logging in with a Google account, and embarking on the project creation process. The procedure requires providing essential details such as the project's name and agreeing to Firebase's terms and conditions. As this is an academic project, we've chosen to go with the free "Spark" plan, which is adequate for a small testing application, but would have to be upgraded once the application is deployed.

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MaterialApp(home: const MainPage(), theme: darkTheme));
}
```

Figure 16 "Firebase init in main()"

The connection to Firebase in the Flutter application is then established in the *main.dart* file, which serves as the entry point of the application. This connection sets up Firebase services that the application will use for various functionalities like authentication, database operations, and storage.

Aside from the necessary imports, the main function has to be set to "async", or asynchronous. The Flutter bindings are then made with *WidgetsFlutterBinding.ensureInitialized(),* and Firebase initialized with *await Firebase.initializeApp();.*

The database operations are distributed across various classes, each handling specific aspects of data interaction. Key classes/widgets like *ScanPage* (*_ScanPageState*), *LogsCard*, *LogGraph*, and *MeasurementsCard* play vital roles in this process, as previously mentioned in the "Data Validation" section.

8.1.5 AUTHENTICATION & USER MANAGEMENT

```dart
class _LoginPageState extends State<LoginPage> {
  final _formKey = GlobalKey<FormState>();
  late String _email;
  late String _password;

  Future<void> _login(BuildContext context) async {
    try {
      UserCredential userCredential =
          await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: _email,
        password: _password,
      );
      print('User signed in: ${userCredential.user}');
    } catch (e) {
      // error dialog when failing to sign in
      // ignore: use_build_context_synchronously
      showDialog(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
            title: const Text('Error Signing In'),
            content: Text('Error signing in: $e'),
            actions: <Widget>[
              TextButton(
                child: const Text('OK'),
                onPressed: () {
                  Navigator.of(context).pop(); //enleve la boite de dialogue
                },
              ),
            ],
          );
        },
      );
    }
  }
```

Figure 17 "Login page declaration"

The *LoginPage* class is pivotal in managing user login. It houses a form with fields for email and password input, facilitated by the use of *TextFormField* widgets. This form utilizes a *GlobalKey<FormState>* for form validation, ensuring the entered credentials are valid before submission.

The crucial part of the login process occurs in the *_login* method within the *_LoginPageState* class. This method interacts directly with Firebase's authentication services. When a user submits their credentials, _login invokes *FirebaseAuth.instance.signInWithEmailAndPassword*, passing the user's email and password. This method is responsible for authenticating the user against the credentials stored in Firebase. In case of a successful login, the user is granted access to the application; otherwise, an error message is displayed.

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

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<User?>(
      stream: FirebaseAuth.instance.authStateChanges(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.active) {
          // Get the user
          User? user = snapshot.data;

          // If the user is null, they are not logged in
          if (user == null) {
            return const LoginPage();
          } else {
            // If the user is not null, they are logged in
            return const MyHomePage(title: 'Projet P3');
          }
        }
        // If the connection to the stream is still loading, show a loading indicator
        return const Center(
          child: CircularProgressIndicator(),
        );
      },
    );
  }
}
```

Figure 18 "AuthWrapper class"

The AuthWrapper class acts as a gatekeeper for the application. It uses Firebase's authStateChanges() stream to listen for changes in the user's authentication state. This class essentially decides whether to present the user with the LoginPage (when no user is authenticated) or to redirect them to the MyHomePage (when a user is authenticated).

One problem with this current implementation is the lack of a “timeout”, which means users stay indefinitely logged in after the first time they’ve done so. Indeed, with Firebase’s User Sessions system, even if the application is uninstalled and reinstalled, the user would still be logged on as that data is kept on the device.

8.2 SECONDARY OBJECTIVES

8.2.1 DAILY ROUND STATISTICS & GRAPH DISPLAY

```
const Row(
  mainAxisAlignment: MainAxisAlignment.spaceAround,
  children: [
    Text('Start Time', style: TextStyle(fontSize: 12)),
    Spacer(flex: 1),
    Text('Stop Time', style: TextStyle(fontSize: 12)),
    Spacer(flex: 1),
    Text('Average Delta', style: TextStyle(fontSize: 12)),
    //texte de la colonne
    //Text('Percentage Completed',
    //    style: TextStyle(fontSize: 12)),
  ],
),
ListView.builder(
  shrinkWrap: true,
  physics: const NeverScrollableScrollPhysics(),
  itemCount: documents.length,
  itemBuilder: (context, index) {
    var documentData =
        documents[index].data() as Map<String, dynamic>;
    DateTime dStartDT =
        (documentData['dStartDT'] as Timestamp).toDate();
    DateTime dEndDT =
        (documentData['dEndDT'] as Timestamp).toDate();
    double rAverageDelta =
        documentData['rAverageDelta'].toDouble();
    double rPerCompleted =
        documentData['rPerCompleted'].toDouble();

    String formattedStartDT =
        DateFormat('MM.dd HH:mm').format(dStartDT);
    String formattedEndDT =
        DateFormat('MM.dd HH:mm').format(dEndDT);

    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceAround,
      children: [
        Text(formattedStartDT),
        Spacer(flex: 1),
        Text(formattedEndDT),
        Spacer(flex: 1),
        Text('$rAverageDelta'),
        //texte du body
        //Text('$rPerCompleted%'),
      ],
```

Figure 19 "LogsCard formatting"

The *LogsCard* class is designed to present a concise summary of the data collection rounds. It utilizes data fetched from the TBL_LOGS table in Firebase Firestore. Each record in this table contains information about individual data collection rounds, including the start and end times (*dStartDT* and *dEndDT*), the average delta of measurements (*rAverageDelta*), and the percentage completion of the round (*rPerCompleted*). The *LogsCard* class processes this data and displays it in a card format, offering a quick overview of each round's key metrics. This format is particularly useful for users who need to review and compare data across multiple rounds at a glance.

It is important to note that the "*rPerCompleted*" functionality was not implemented in this version of the application, and is therefore kept as a future improvement.

```dart
class LogGraph extends StatelessWidget {
  final List<DocumentSnapshot> documents;

  const LogGraph({Key? key, required this.documents}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    double maxY = 0.0;

    double y = 0.0;

    // date de demarrage dynamique
    // On calcule la date de depart en fonction de la date la plus ancienne dans la table
    DateTime getStartDate(List<DocumentSnapshot> documents) {
      DateTime earliest = DateTime.now();
      for (var doc in documents) {
        var data = doc.data() as Map<String, dynamic>;
        Timestamp? timestamp = data['dTimeStamp'];
        y = data['rAverageDelta']?.toDouble() ?? 0.0;
        if (timestamp != null) {
          DateTime date = timestamp.toDate();
          if (date.isBefore(earliest)) {
            earliest = date;
          }
        }
        if (y > maxY) {
          maxY = y; //definit le maximum pour l'echelle du graph
        }
      }
      return earliest;
    }

    DateTime startDate = getStartDate(documents);
```

Figure 20 "LogGraph class and getStartDate function"

In parallel, the *LogGraph* class serves a complementary role by providing a graphical representation of the data. It visualizes the same data from the *TBL_LOGS* table, particularly focusing on the average delta of measurements over time. This class leverages the *fl_chart* package to create line charts, which plot the average delta against the time, represented by the start date of each round.

The *getStartDate* function within *LogGraph* determines the starting point of the graph. It iteratively examines each document in the documents list, which contains data from the TBL_LOGS table, to find the earliest date. This is done by comparing the timestamp (dTimeStamp) of each record with the current earliest date. In this process, the function also calculates the maximum value (maxY) for the y-axis of the graph, which is used to scale the graph appropriately. The maxY is determined by finding the highest rAverageDelta value among all documents, ensuring the graph can accommodate all data points.

```dart
List<FlSpot> spots = documents
    .map((doc) {
      var data = doc.data() as Map<String, dynamic>;
      double y =
          data['rAverageDelta']?.toDouble() ?? 0.0; //redeclaration de y
      Timestamp? timestamp = data['dStartDT'];
      if (timestamp == null) {
        return null; // Skip if no timestamp
      }
      DateTime x = timestamp.toDate();
      int daysSinceStart = x.difference(startDate).inDays;
      return FlSpot(daysSinceStart.toDouble(), y);
    })
    .where((spot) => spot != null)
    .cast<FlSpot>()
    .toList(); //cast pour transformer en liste
```

Figure 21 "LogGraph spots list"

The core of the graph's data representation is encapsulated in the spots list. This list is constructed by mapping each document to a *FlSpot* object, which represents a point on the graph. For each document, the function extracts the *rAverageDelta* value and the start date timestamp (*dStartDT*). The date is then used to calculate the number of days since the start date (determined by *getStartDate*). This number forms the x-coordinate, while the *rAverageDelta* value forms the y-coordinate of each point.

```dart
LineChartData(
  minY: 0,
  maxY: maxY,
  gridData: FlGridData(show: true),
  titlesData: FlTitlesData(
    rightTitles: const AxisTitles(
      sideTitles: SideTitles(showTitles: false),
    ),
    topTitles: const AxisTitles(
      sideTitles: SideTitles(showTitles: false),
    ),
    leftTitles: const AxisTitles(
      sideTitles: SideTitles(showTitles: false),
    ),
    bottomTitles: AxisTitles(
      sideTitles: SideTitles(
        showTitles: true,
        getTitlesWidget: (double value, TitleMeta meta) {
          DateTime date =
              startDate.add(Duration(days: value.toInt()));
          return SideTitleWidget(
            axisSide: meta.axisSide,
            space: 8.0,
            child: Text(DateFormat('MM/dd').format(date)),
          );
        },
      ),
    ),
  ),
  borderData: FlBorderData(show: true),
  lineBarsData: [
    LineChartBarData(
      spots: spots,
      isCurved: true,
      barWidth: 2,
    ),
  ],
),
),
```

Figure 22 "LineChartData declaration"

In rendering the graph, *LogGraph* uses the *LineChart* widget from the *fl_chart* package. The data for the chart is provided through *LineChartData*, which includes the spots list. The chart is configured to display grid lines (*FlGridData*), and custom titles for each axis. Particularly, the bottom titles use a custom widget to format the x-axis labels as dates.

8.2.2 MEASUREMENT AUTO-COMPLETION & MINI-GRID DISPLAY

The implementation of measurement data history is adeptly handled by the *MeasurementsCard* class. This class is specifically designed to fetch and display historical measurement data, enhancing the user's understanding of past measurements associated with specific parameters and posts. However, it's important to note that, based on the provided class declarations, the feature of auto-completion for measurement data entry is not implemented in the current version of the application.

```dart
class MeasurementsCard extends StatelessWidget {
  final int postID;
  final int paramID;

  const MeasurementsCard({
    Key? key,
    required this.postID,
    required this.paramID,
  }) : super(key: key);

  Future<List<Map<String, dynamic>>> fetchLastThreeMeasurements() async {
    var querySnapshot = await FirebaseFirestore.instance
        .collection('TBL_DATAINBOX')
        .where('iPostID', isEqualTo: postID)
        .where('iParamID', isEqualTo: paramID)
        .orderBy('dTimeStamp', descending: true)
        .limit(3)
        .get();

    //trie les données par date

    return querySnapshot.docs.map((doc) => doc.data()).toList();
  }
```

Figure 23 "MeasurementsCard and fetchLastThreeMeasurements"

The function *fetchLastThreeMeasurements* within this class is responsible for retrieving the last three measurements for a given post and parameter ID. It uses Firestore's query mechanism to filter data based on these IDs and orders the results by the timestamp, ensuring that the most recent measurements are fetched.

Each measurement entry fetched includes key details such as the measurement value (*rValue*) and the timestamp of when the measurement was taken. This data is then rendered in a card format on the application's UI, presenting the information in an easily digestible manner. The card displays each measurement along with its corresponding date, allowing users to quickly grasp the recent measurement history.

9 PLANNING

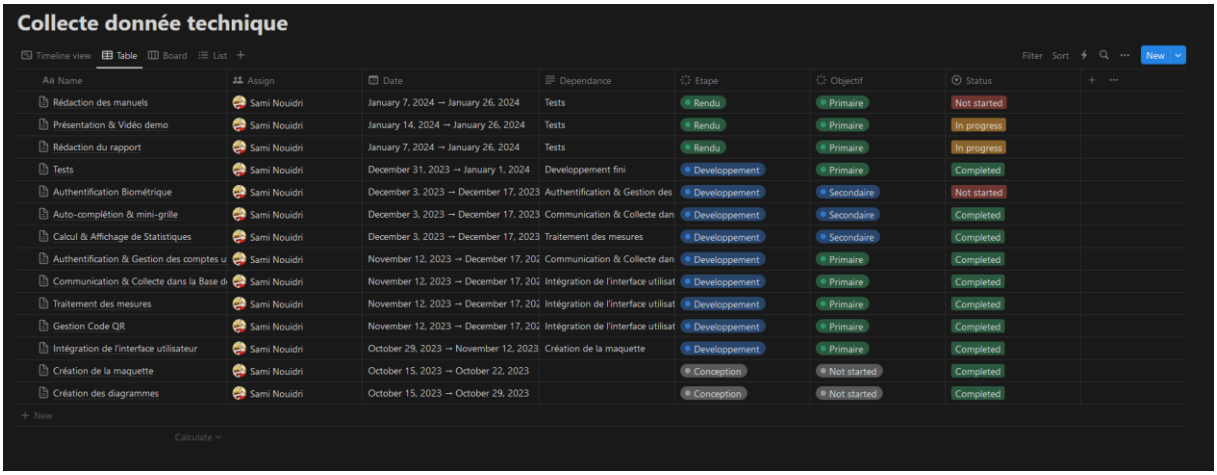

Figure 24 "Project Planning"

Above is the project's planning table, with the various tasks laid out during the conception phase. Each task is assigned labels, to denote whether it is a primary, secondary, or final objective, as well as its project phase.

Throughout the development phase, the deadlines were relatively well met, especially at the start. However, a lot of the secondary tasks (notably daily round statistics and data history) were relegated and done after the winter break, due to the end of year workload.

That being said, they were done in-time to start the final objectives on-schedule (during the month of January), which means the overall project scheduling was maintained.

10 CONCLUSION

In conclusion, the development of our application marks a significant advancement in the process of conducting daily rounds for machine measurements in industrial facilities. This project, through its innovative approach and strategic use of technology, addresses the core challenges inherent in traditional manual methods, primarily those related to accuracy, efficiency, and real-time data processing.

The application successfully integrates QR code technology, providing a swift and error-free mechanism for identifying machines and recording measurements. This not only minimizes the risk of human errors but also enhances the reliability of the data collected, a crucial aspect for maintaining compliance with insurance requirements. The digitization of the data collection process has significantly improved operational efficiency, reducing the time spent on manual entries and allowing employees to allocate more time to other critical maintenance tasks.

Furthermore, the application's ability to provide real-time data analysis and its capacity to store historical data have significantly improved the way maintenance data is processed and interpreted. These features enable quicker decision-making, trend analysis, and proactive maintenance strategies, all of which contribute to the overall health and longevity of the facility's machinery.

However, it is important to note areas for future development, such as the implementation of an auto-completion feature for measurement entries and the exploration of more advanced QR code formats. Security, with Biometric authentication for example, is also an area to be improved. These enhancements could further streamline the data entry process and expand the application's capabilities.

11 BIBLIOGRAPHY

QR Codes for manufacturing, qr-code-generator, https://www.qr-code-generator.com/blog/qr-code-for-manufacturing/

Advantages of cloud-based asset/data management, AssetPanda, https://www.assetpanda.com/resource-center/blog/what-are-the-advantages-of-cloud-based-asset-management/

QR Code formats, qrcode.com, https://www.qrcode.com/en/codes/

12 DATE AND SIGNATURE

Neuchâtel, the 26th of January 2024, _____

Nouidri Sami