Capstone Project Final Report - SLB

ABSTRACT

For the capstone project, we have developed a cross-platform application (iOS, Android, and web) that employs QR-code technology to streamline asset tracking, task assignment, and inventory management. The system enables rapid, error-resistant identification of parts and products. The backend, built on PostgreSQL with RESTful APIs secured by JWT-based authentication, supports comprehensive part and product management, role-based access control, and a full borrowing/return workflow. Automated unit, integration, and stress tests, along with a CI/CD pipeline, ensure reliability and facilitate safe deployments. The Flutter-based frontend delivers a responsive UI featuring real-time scanning, manual code entry, history tracking, and bulk QR-code generation from Excel inputs. Performance evaluations demonstrate high scanning accuracy and significant reductions in manual errors. This integrated solution enhances operational visibility, accelerates asset workflows, and lays the foundation for future data-driven optimizations and predictive inventory analyses.

1. Introduction

In response to the growing complexity of internal logistics management, SLB Company has commissioned the development of an application in the three end (iOS, android and web) to streamline inventory tracking and task allocation. The proposed system uses QR code technology to label and manage physical assets within the company. Each item will be tagged with a unique QR code, which encodes essential information such as item category, ID, tasker, and location. The app will greatly help employees to check the status of items and future data analysis.

2. Literature Review

2.1 QR codes

QR codes have emerged as an effective tool for asset identification and tracking due to their ease of use, cost-effectiveness, and ability to store substantial data [1]. Unlike barcodes, QR codes can store more information, including URLs, text, and metadata, making them suitable for inventory management applications.

Research indicates that QR code-based tracking systems improve accuracy in asset management by reducing human errors associated with manual data entry [2]. Additionally, QR codes facilitate quick scanning via mobile devices, enabling real-time updates and seamless integration with cloud databases [3].

In this case, we decide to use QR codes as identification for different items. The international standard will be ISO and input information will be restricted to text digits without bit limitation.

2.2 Three end solutions

The adoption of cross-platform applications (iOS, Android, and web) in logistics management has grown due to their accessibility and scalability [4]. Cloud-based systems enable centralized data storage, allowing multiple users to access and update information simultaneously [5].Furthermore, web-based dashboards facilitate data analysis, helping organizations derive insights for better decision-making.

Hence, portable and efficient three end application is what we strive for. We have realized the application deployment on iOS and Android systems. Web applications will be uploaded in the future.

2.3 Inventory management

Inventory management is a critical aspect of logistics, ensuring that resources are tracked, allocated, and utilized efficiently. Traditional manual systems are prone to errors, leading to inefficiencies in stock management [6]. Automated inventory systems, particularly those integrated with mobile technology, have been shown to significantly improve accuracy and reduce operational costs.

We are aimed at making inventory management automatic and intelligent, and reduce manual operations to meet more B2B demands, including task assignments and clear clients control.

3 Work completed

3.1 Backend

The backend architecture is designed to support a scalable and secure service ecosystem deployed on an Azure Virtual Machine. At the foundation, the system uses PostgreSQL as the primary relational database to store and manage all persistent data. This includes structured data related to employees, products, and user operation logs.

Above the database layer is a Service Layer running on the Azure VM, which encapsulates core backend functionalities. This layer is responsible for handling business logic and includes several major components, and detailed information will be introduced in following sections.

These backend services are accessed by multiple frontend clients, including iOS App and Android App. The system is modular, maintainable, and built to support further extensions as needed.

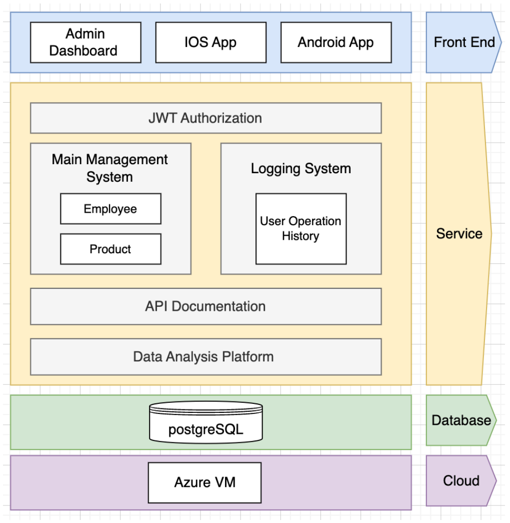


Figure1. Backend Structure

3.1.1 User Authentication and JWT-based Authorization

To secure the system and control access to various endpoints, we implemented a robust user authentication and authorization mechanism. Users register and log in through the /api/employee/ endpoint, and upon successful login, they receive a JWT. This token is required for subsequent API access, ensuring that only authenticated users can perform actions such as part creation, borrowing, or returning.

Each user is associated with a user_type, allowing us to define roles (e.g., admin vs regular employee) and apply role-based access control throughout the application. This system guarantees that only authorized users can perform sensitive operations like deleting or modifying component and category data.

3.1.2 Inventory Management and Part/Product Relationship

Our backend provides a comprehensive inventory management system using RESTful APIs under the part-api and product-api groups.

Part Management (/api/components)

The Part entity represents individual hardware components and is managed via the following endpoints:

- GET /api/components: Lists all components.
- GET /api/components/{id}: Fetches details of a specific component.
- POST /api/components: Adds a new part to the inventory.
- PUT /api/components/{id} and DELETE /api/components/{id}: Allows editing and removal of parts.

Additional endpoints such as /returned, /borrowed, and /status/{status} allow filtering by usage and availability.

Each Part is linked to attributes like part_number, borrowed_employee_id, status, and cost, which are critical for tracking availability and usage.

Product Management (/api/categories)

A Product in our system refers to a predefined bundle or category of parts. Through the following endpoints, users can manage product types and their associated parts:

- GET /api/categories and GET /api/categories/{id}: Retrieve all products or one specific product.
- POST /api/categories: Adds a new product definition.
- PUT /api/categories/{id} and DELETE /api/categories/{id}: Update or remove product categories.
- GET /api/categories/{id}/parts: Lists parts associated with a product.
- GET /api/categories/name/{name}: Enables search by product name.

Each Product is defined in the database with part_list, lead_time, and other attributes that support detailed tracking and planning for procurement or usage.

The tight coupling between parts and products ensures accurate inventory records, especially during borrow and return operations.

3.1.3 Borrowing and Returning Workflow

We implemented an activity-api module to handle component stock-in, stock-out, borrow, and return operations.

- POST /api/activities/stock-out: Marks parts as taken from the inventory.
- POST /api/activities/stock-in: Restocks returned or new parts.
- POST /api/activities/borrow: Logs when a user borrows a part, updating borrowed_employee_id in the Part table and writing an entry in Employee_activity.
- POST /api/activities/return: Reverses the borrowing status, updates the Returned_part table, and removes the borrower ID from the part.

All activities are recorded with timestamps (operate_time) and linked to employees and specific parts, making the system traceable and auditable. These operations are crucial for maintaining data consistency across tables such as Part, Returned_part, and Employee_activity.

3.1.4 Automated Unit and Integration Testing

We ensured the correctness and reliability of the backend through comprehensive unit and integration testing.

- Unit Tests: Targeted key service-layer logic, especially for activity logging and JWT token handling.
- API Tests: Automatically tested core activity and user-related endpoints using mock requests and responses.
- Multi-threaded Stress Testing: To simulate concurrent user activity, we implemented multi-threaded tests that perform simultaneous borrow/return requests. This revealed and helped mitigate potential race conditions, ensuring system robustness under load.

These tests were crucial in verifying that the business logic, such as preventing double borrowing or overstocking, functions as intended.

3.1.5 CI/CD Pipeline and Deployment Strategy

To streamline development and ensure safe deployments, we implemented a full CI/CD pipeline:

- Version Control: All code is managed via Git, with clear branching strategies for feature, test, and release versions.
- Automated Testing: Tests are triggered automatically upon each commit to ensure that changes do not break existing functionality.
- Deployment Automation: Using GitHub Actions (or a similar tool), code is automatically deployed to a staging server after passing all tests.
- Environment Isolation: We clearly separated development/testing environments from the production environment. This avoids any interference between test data and live user operations.

This workflow guarantees rapid iteration and minimizes the risk of introducing bugs to the production system.

3.1.6 Database Design and Entity Relationships

Our relational database schema models real-world interactions between employees, components, and products. Key aspects include:

- Employee Table: Stores credentials and roles.
  Part Table: Tracks each hardware unit's availability and borrowing status.
- Product Table: Defines part bundles or functional categories.
- Employee_activity Table: Logs all actions with timestamps, supporting audit trails.
- Returned_part Table: Specifically tracks part returns for reporting and validation purposes. Foreign keys and indexed attributes (such as employee_id and part_id) optimize query performance and ensure referential integrity.

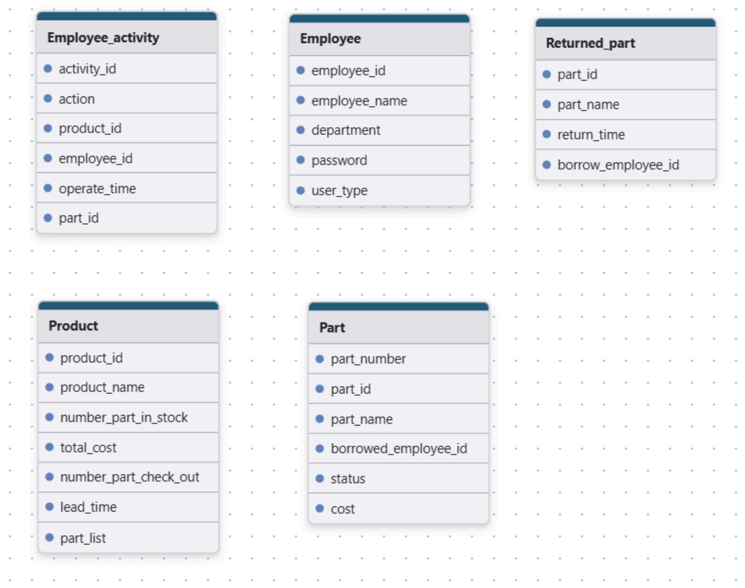


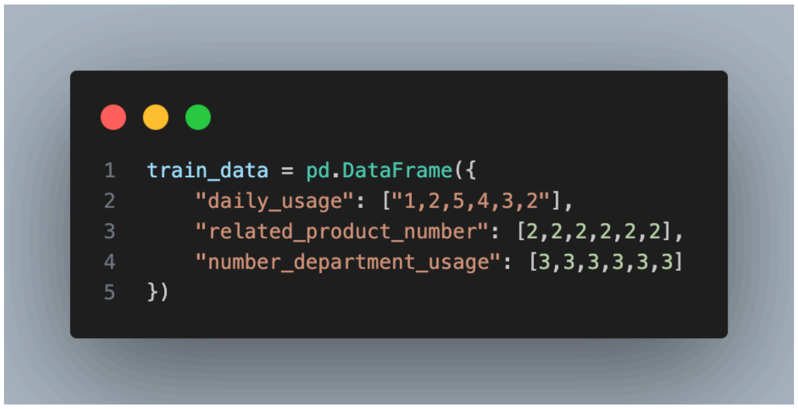
Figure2. Database table design

3.1.7 Data Analysis

For the data analysis part, since we currently do not have real data available for analysis, we provided SLB with several machine learning approaches for data analysis as a reference. This is to give them an idea of which machine learning models they could use in the future if they wish to incorporate data analysis into their projects.

Feature / Model	Linear Regression	Decision Tree	XGBoost
Support Non-Linear Patterns	No	Yes	Yes
Expect Accuracy	Low to Medium	Medium to High	High
Scales with Datasets	Simple datasets	Need more time to process	Designed for large-scale data
Model Complexity	Very Simple	Moderate	More Complex

Table 1. different machine learning approaches and their respective advantages and disadvantages.



```
train_data = pd.DataFrame({
    "daily_usage": ["1,2,5,4,3,2"],
    "related_product_number": [2,2,2,2,2,2],
    "number_department_usage": [3,3,3,3,3,3]
})
```

Figure3. Example of data format

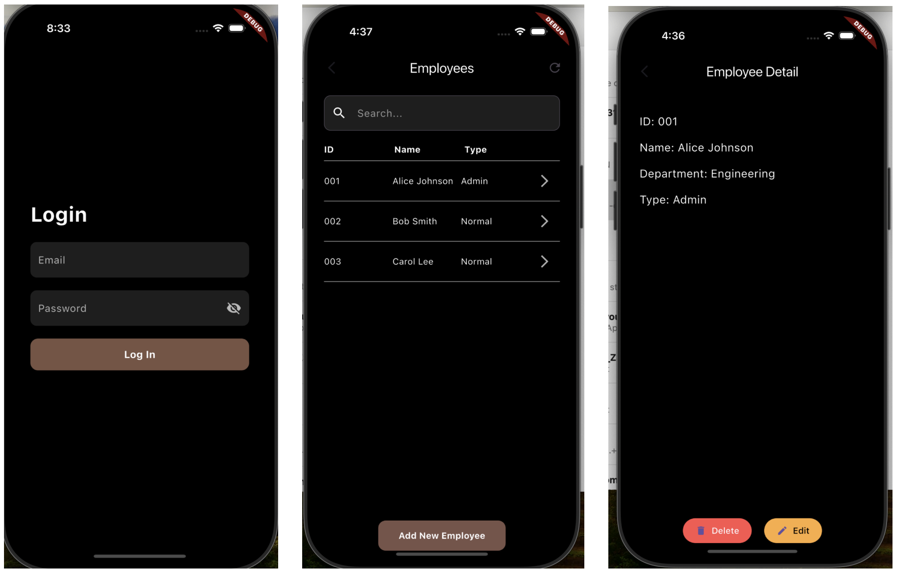
3.2 Frontend

3.2.1 Login and accounts

We designed the login system to be secure and user-friendly. When users open the app, they are greeted with a clean login screen where they can enter their account and password to access the system. To enhance security, we apply a hash function on the frontend before sending any password data to the server. This ensures that raw passwords are never exposed during transmission. By hashing the password client-side, we add an extra layer of protection alongside HTTPS encryption.

Once users successfully log in, we fetch their profile data and store session tokens locally to maintain their login state as they navigate through the app. Our system supports role-based access control, which allows us to assign different permissions based on user type. For example, Admins can add, edit, and delete employee records, while regular users have view-only access. In the Accounts section, we display a searchable list of all employees. Users can select an employee to view detailed information like ID, name, department, and role. This section helps us keep our team organized and ensures easy access to contact or administrative information.

To prevent accidental data loss, we implemented a pop-up confirmation message before any delete action. When an Admin taps the delete button on an employee record, a modal dialog appears asking the user to confirm the deletion. This extra step ensures that deletions are intentional and gives the user a chance to cancel the action if it was triggered by mistake.
All changes—including deletions—are immediately synced with the backend, ensuring our records remain accurate and up to date.



ID	Name	Type
001	Alice Johnson	Admin
002	Bob Smith	Normal
003	Carol Lee	Normal

ID: 001
Name: Alice Johnson
Department: Engineering
Type: Admin

Figure4. Login account and employee details

3.2.2 Homepage and inventory management

The homepage is designed as an expandable scrolling list and displays useful information like user activity and notifications. To streamline navigation, we integrated a sidebar menu. The sidebar includes essential links such as Inventory and Admin, allowing users to move quickly between core features. Its minimalist layout and icon-based design help maintain a clean interface while ensuring usability. The sidebar also adapts based on the user's role—Admin users will see additional links compared to regular users.

The inventory is implemented using multiple maps. SLB equipment is catalogued using a hierarchy of three levels: a product corresponds to multiple parts, and a part corresponds to multiple part numbers. As such maps are a natural way to associate products, parts, and part numbers as key-value pairs for quick look up.

Several api protocols are needed. When the user returns to the home page from any other page, a query to the backend for user history is immediately initiated and the user history is updated. Similarly, upon navigating to the inventory page, a query to the backend for all the user's currently borrowed parts is initiated and the user inventory is updated accordingly.

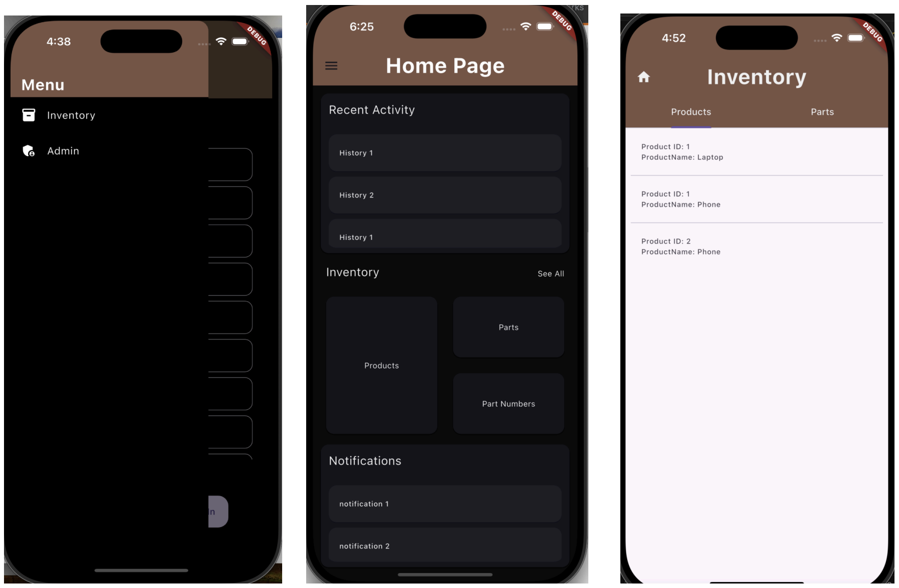


Figure5. Home page

3.2.3 Scanner

The QR code scanning feature was implemented using Flutter libraries such as *barcode_scan*, *barcode_scan2*, and *camera*. These libraries enable real-time image capture and processing. The scanner achieves high recognition accuracy. It can quickly and accurately detect QR codes as small as 0.5cm × 0.5cm, making it suitable for a wide range of use cases. Besides, in cases where scanning is not feasible, users can manually input QR codes for retrieval.

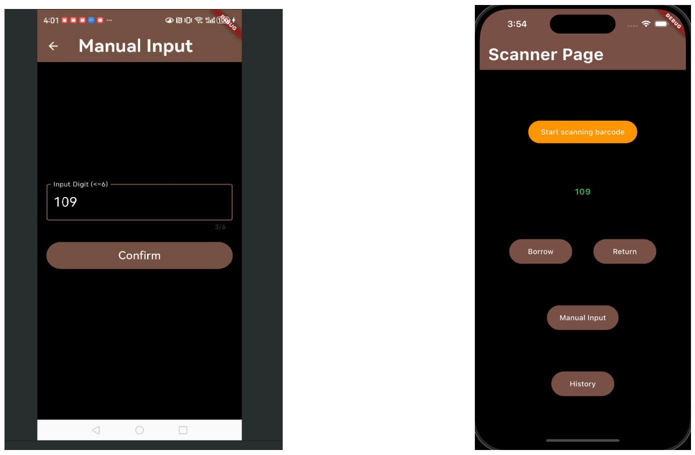
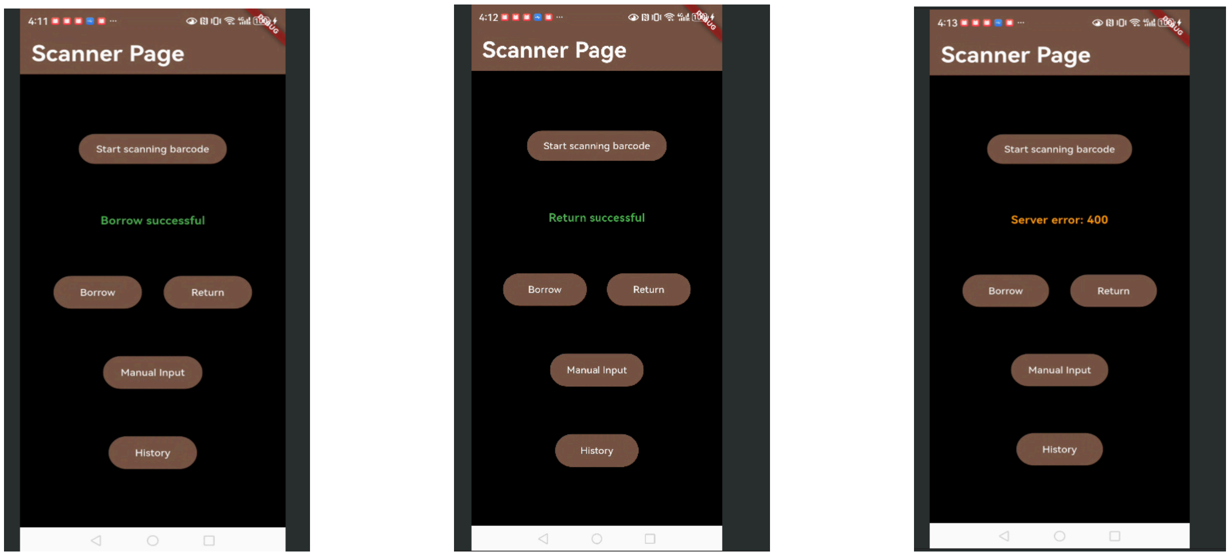


Figure6. Scanner page

For error handling, the app provides clear feedback for both successful scans and failed scans. For testing purposes, QR codes are encoded as no digit limitation numeric strings, where both the *category ID* and the *part ID* will be analysed and recorded.



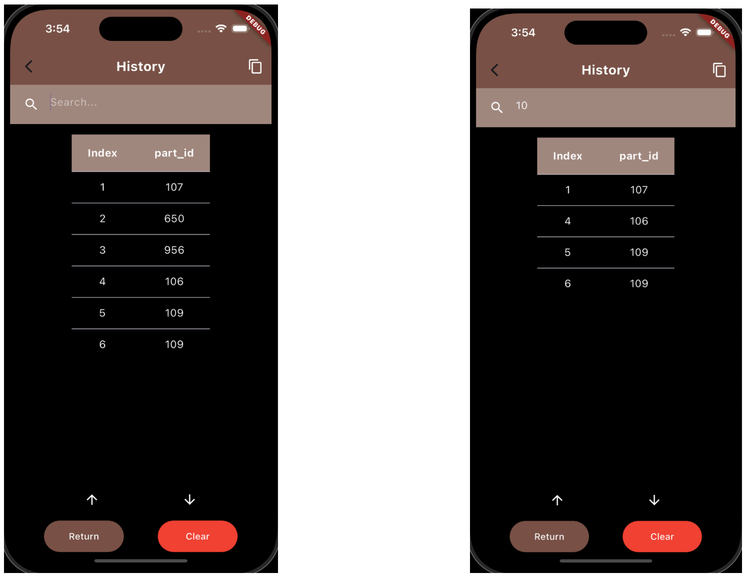
Borrow successful

Return successful

Server error: 400

Figure7. Error handling in Scanner Page

The scanner pages also show the clear navigation. Users can easily return to the homepage after scanning or manual input. As long as you click “borrow”, the item will be checked in. And after clicking “return”, the item will be checked out. Two statues can be easily viewed and transformed. Click “history” and all control records will be displayed.



Index	part_id
1	107
2	650
3	956
4	106
5	109
6	109

Index	part_id
1	107
4	106
5	109
6	109

Figure8. Scanner History

3.2.4 QR code generator

The application integrates a QR/barcode generator that allows users to create and print custom QR codes based on uploaded data (e.g., Excel files or CSV files). This feature streamlines asset labeling, inventory tracking, and task management by automating code generation and ensuring compatibility with different printing formats.

Users can upload *.xlsx*/*.csv* files containing barcode data (e.g., item ID, category, location, tasker). The system auto-generates QR codes by extracting data from specified columns. Also, the app tracks the local storage path of generated QR codes for easy retrieval. Users can export QR images in multiple print formats, including line spacing, the number of printed barcodes per row, column width, margins and other properties.

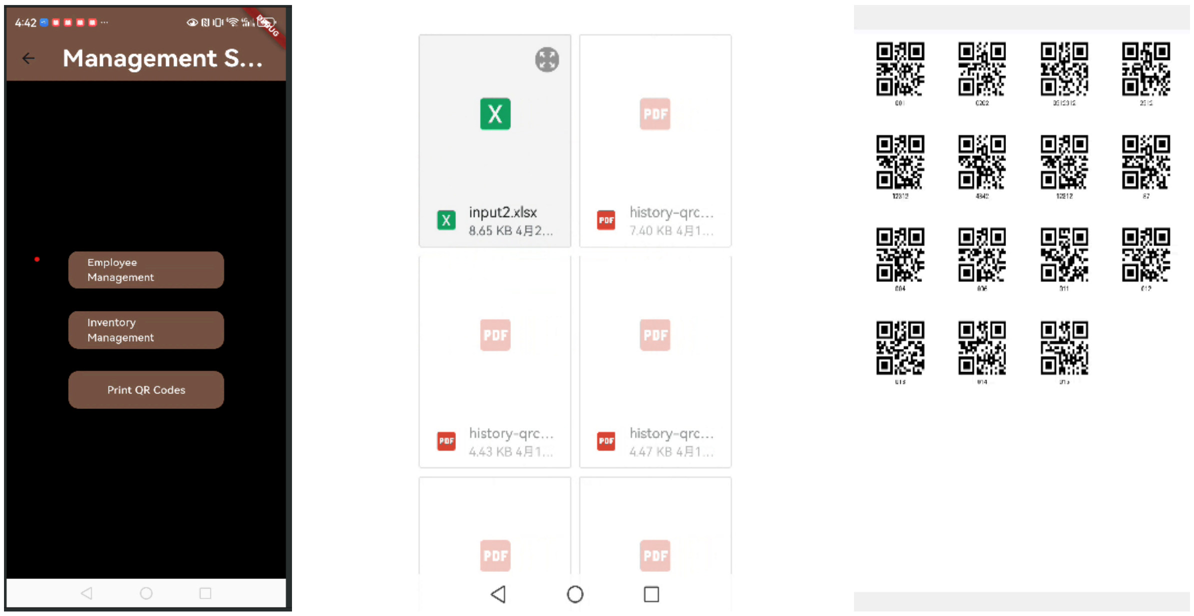


Figure9. QR code generator

4. Summary and Conclusions

In summary, we have implemented both the back end and front end of a mobile app for managing the engineering equipment that SLB uses. The front end is built through the flutter framework, which enables cross platform development for both iOS and Android. The backend is built through PostgreSQL.

To have front-end and back-end integration, a large amount of effort was dedicated to creating and standardizing the api protocols supported by the server that the mobile app would need. Some of these endpoints are detailed in the report. The front-end communicates with the back-end through the http protocol and obeys the characteristics of the RESTful protocols, where the client(front-end) initiates communication with the server and the server stores no information about the client state.

More specific features about the front-end and back-end are detailed in the report. Some of the front-end features include inventory management, qr-code/barcode scanner, employee and part management, and qr-code generator. Features of the backend include a full CI/CD pipeline for automated testing and automated deployment.

5. Suggestions for Future Work

While we have completed the necessary features for basic cataloguing and tracking of equipment, there are additional features especially in data analysis that would be useful to implement. Seeing as we had no prior history for how individual parts were used, we could not accurately use more advanced data analysis techniques such as machine learning. In the future, if we had access to how different parts were used over time, we would use the data to train more accurate models for predicting part usage.

6. Group Members

Frontend Team:

NetID	**Name**
allenp2	Allen Peng
yutaots2	Yu-Tao Sun
shiyul4	Shiyu Liu
chihyuh3	Sophie Huang
ch115	Chia-Chun Hsiao
yhchen6	Yuan-Hao Chen

Backend Team:

NetID	**Name**
mingyi5	MingYi Wei
yl191	Yujiang Liu
feng45	Jingting Feng
qijing3	Qi Jing
zixuan41	Zixuan Shen

7. Bibliography

[1] Huang W C, Tsai C C, Chen C L, et al. Glucosylceramide synthase inhibitor PDMP sensitizes chronic myeloid leukemia T315I mutant to Bcr‐Abl inhibitor and cooperatively induces glycogen synthase kinase‐3‐regulated apoptosis[J]. The FASEB Journal, 2011, 25(10): 3661-3673.

[2] Lin Y C, Cheung W F, Siao F C. Developing mobile 2D barcode/RFID-based maintenance management system[J]. Automation in construction, 2014, 37: 110-121.

[3] Want R. An introduction to RFID technology[J]. IEEE pervasive computing, 2006, 5(1): 25-33.

[4] Hevner A R, March S T, Park J, et al. Design science in information systems research[J]. MIS quarterly, 2004: 75-105.

[5] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing[J]. Communications of the ACM, 2010, 53(4): 50-58.

[6] Ngai E W T, Moon K K L, Riggins F J, et al. RFID research: An academic literature review (1995–2005) and future research directions[J]. International journal of production economics, 2008, 112(2): 510-520.