UTM

UNIVERSITI TEKNOLOGI MALAYSIA

Technical Report

Yatt's Kitchen Smart Food Ordering and Queue Number System

Prepared by:

NAME	MATRIX NUMBER
RUPATARANI A/P PARAMASIVAM	SX210500ECJHS04
MALATHI A/P MOHANASUNDRAM	SX212363ECJHF03
SANTHYA A/P MUNIAPPAN	SX220761ECJHS03
SUMATHI KESAVAN	SX212327ECJHF03

Table of Contents

1.0 Introduction

The Yatt's Kitchen Smart Food Ordering and Queue Number System is a comprehensive mobile-based application developed to modernise, automate, and optimise the daily operational workflow of Yatt's Kitchen. This project was carried out as part of a software engineering and system development initiative, with the intention of applying theoretical knowledge from system analysis, system design, database management, and software maintenance into a real-world business environment.

In traditional small food stall operations such as Yatt's Kitchen, food ordering and queue management are commonly handled manually. Customers place their orders verbally at the counter, staff manually record the orders, and queue numbers are announced verbally or tracked informally. While this approach may appear simple, it introduces significant inefficiencies during peak hours when customer volume is high. Common problems include misheard orders due to noise, incorrect food preparation, skipped or repeated queue numbers, and customers frequently asking staff about their order status. These issues negatively impact service speed, customer satisfaction, and staff productivity.

The proposed system introduces a smart digital solution consisting of a customer mobile application, an administrative management interface, and a cloud-based backend system. By leveraging Flutter for frontend development and Firebase Firestore for backend data management, the system enables real-time order tracking, automated queue generation, and structured order processing. This technical report provides an in-depth and detailed explanation of the system, including the background problem, system objectives, overall architecture, technology stack, database design, system modules, workflow, implementation process, testing and validation, results analysis, conclusion, and future enhancement possibilities. The explanation is intentionally detailed to meet academic requirements and ensure clarity for technical evaluation.

2.0 Problem Statement

Yatt's Kitchen faces multiple operational challenges due to its reliance on manual food ordering and queue management processes. One of the most critical issues is communication breakdown between customers and staff. In a noisy food stall environment, verbal order placement is highly prone to misunderstanding, especially during peak business hours. Miscommunication often results in incorrect orders being prepared, leading to food wastage, customer complaints, and revenue loss.

Another major problem is inefficient and unstructured queue management. Queue numbers are typically handled verbally or informally, which creates confusion among customers. Customers may miss their queue number, crowd around the counter, or repeatedly interrupt staff to ask about order progress. This situation increases congestion, disrupts staff workflow, and creates a stressful working environment.

Manual operations also significantly increase staff workload. Staff members are required to multitask by taking orders, recording them manually, verifying payments, managing queue numbers, and responding to customer inquiries simultaneously. This multitasking environment increases the likelihood of human error and slows down overall service delivery. Furthermore, the absence of digital records makes it impossible for management to analyse order trends, peak hours, or popular menu items. These combined challenges highlight the urgent need for a digital system that can automate ordering, improve queue management, and enhance operational efficiency at Yatt's Kitchen.

3.0 Objective of the System

The development of the Yatt's Kitchen Smart Food Ordering and Queue Number System is guided by clearly defined objectives that align with business requirements and academic learning outcomes:

- To replace the existing manual food ordering process with a digital system.
- To provide customers with a simple, intuitive, and user-friendly mobile ordering interface.
- To automate the generation and management of queue numbers.
- To allow customers to track their order and queue status in real time.
- To reduce human errors associated with verbal communication and manual recording.
- To minimise staff workload by automating repetitive operational tasks.
- To improve customer satisfaction by reducing waiting time uncertainty.
- To provide a scalable and maintainable system architecture.
- To apply software engineering principles such as modularity, usability, and maintainability.

4.0 System Overview

The Yatt's Kitchen system is designed around three main components: the Customer Mobile Application, the Admin Management Interface, and the Cloud-Based Backend System. These components work together to ensure seamless communication and real-time data synchronisation.

The Customer Mobile Application allows customers to browse the digital menu, view food descriptions and prices, select food items, place orders, choose payment methods, receive a queue number, and monitor the status of their orders. The application does not require user registration, ensuring quick access and ease of use.

The Admin Management Interface is used by the stall owner or authorised staff to manage daily operations. Through this interface, administrators can view incoming orders, verify payments, approve or reject orders, assign queue numbers, update order statuses, and manage menu items. Any update performed by the admin is instantly reflected on the customer application.

The Cloud-Based Backend System, powered by Firebase services, manages data storage, authentication, business logic, and real-time communication. Firebase Firestore acts as the central data repository, ensuring data consistency, availability, and reliability across the system.

5.0 System Architecture

The system architecture follows a client–server model combined with a three-tier architecture consisting of the presentation layer, application layer, and data layer. This separation of concerns improves system maintainability, scalability, and reliability.

The presentation layer includes the Flutter-based customer mobile application and the admin interface. This layer is responsible for user interaction, data input, and visual feedback. The application layer contains the business logic, such as order processing rules, payment verification logic, queue number generation, and order status transitions. The data layer uses Firebase Firestore to store and manage all system data.

Firebase Authentication is used to secure admin access, ensuring that only authorised users can perform administrative actions. Firestore's real-time listeners enable instant data updates across all connected devices, eliminating the need for manual data refresh. The cloud-based architecture also reduces infrastructure maintenance costs and improves system availability.

The Yatt's Kitchen system was developed using modern and reliable software tools suitable for web-based applications. React with TypeScript was used to build the frontend of the system. React allows the application to be developed using reusable components, while TypeScript improves code reliability by detecting errors during development.

CSS was used for user interface styling. It enables responsive design and ensures that the system can be accessed comfortably on different devices, especially mobile phones used by customers. Firebase was used as the backend platform. Firebase Authentication was used to manage secure user login, while Firebase was used as the cloud database to store and manage system data. Additional tools include Visual Studio Code for development, and Node.js with npm for dependency management.

6.0 Technology Stack

The system utilises a modern and industry-relevant technology stack:

- Flutter Framework: Used for developing a responsive and cross-platform mobile application.
- Dart Programming Language: Provides object-oriented features and null-safety support.
- Firebase Firestore: Serves as the NoSQL cloud database with real-time synchronisation.
- Firebase Authentication: Ensures secure admin login and access control.
- Firebase Cloud Messaging: Enables real-time notifications to customers.
- Development Tools: Android Studio and Visual Studio Code.

This technology stack was chosen for its scalability, ease of integration, and suitability for real-time mobile applications.

7.0 Database Design

The database design is based on Firebase Firestore's document-oriented NoSQL model. Data is organised into collections and documents, allowing flexible schema evolution and efficient data retrieval. The database supports full CRUD operations to manage menu items, orders, queue data, and payment records.

- MenuItems: Stores item ID, name, category, price, description, image URL, and availability status.
- Orders: Stores order ID, item list, quantities, total price, order status, timestamps, and assigned queue number.
- Payments: Stores payment method, payment reference, verification status, and admin approval details.
- Queues: Stores queue number, order reference, current queue status, and timestamps.
- Admins: Stores admin credentials and access roles.
- Users Collection – Stores user details such as name, email or phone number, and user role (Admin, Kitchen Staff, Customer).
- Menu Collection – Stores food item information including item name, category, price, image, and availability status.
- Orders Collection – Stores customer order details such as order ID, ordered items, total price, order status, and timestamp.

Firestore's real-time capability ensures instant data synchronisation between customer and admin interfaces. This allows real-time data synchronization, enabling kitchen staff to see new orders immediately after customers place them.

8.0 System Modules Description

8.1 Authentication Module

- Handles secure admin login using Firebase Authentication. It ensures only authorised users can access administrative features.

8.2 Menu Management Module

- Allows admins to add, edit, delete, and update menu items. Changes are reflected instantly on the customer application.

8.3 Order Management Module

- Manages order creation, validation, storage, and status updates.

8.4 Payment Verification Module

- Allows admins to verify payments before approving orders.

8.5 Queue Management Module

- Automatically generates sequential queue numbers and tracks order progression.

8.6 Notification Module

- Uses Firebase Cloud Messaging to send real-time updates to customers.

9.0 System Workflow

- Customer browses the menu and places an order.
- Order data is stored in Firestore.
- Admin verifies payment and approves the order.
- Queue number is generated automatically.
- Order status is updated during preparation.
- Customer receives real-time notifications.
- Order is collected when ready.

10.0 Implementation Details & CRUD Operation

The system is implemented using a modular Flutter architecture. Each feature is encapsulated within separate widgets and services. Asynchronous programming ensures smooth performance during database operations. Firestore security rules enforce role-based access control, and error handling mechanisms manage network failures and invalid inputs.

The Yatt's Kitchen Smart Food Ordering and Queue Number System supports full CRUD (Create, Read, Update, Delete) operations through the integration of Firebase Firestore, which acts as the central cloud database for all system data. CRUD operations are essential for ensuring accurate data handling, real-time synchronisation, and smooth system functionality throughout the application lifecycle.

10.1 Create Operations

Create operations are performed whenever new data is introduced into the system. In Yatt's Kitchen, this includes the creation of new user sessions, food orders, and menu items. When a customer places an order through the mobile application, the system generates a new order document in the Firestore database containing order details such as selected items, quantity, total price, timestamp, and initial order status. Additionally, administrators can create new menu items by entering food details such as name, price, category, and availability, which are then stored as new documents in the menu collection. These create operations ensure that all new data is accurately captured and stored for further processing.

10.2 Read Operations

Read operations allow users to retrieve and view existing data stored in the database. Customers perform read operations when browsing the digital menu, viewing their order details, and tracking their queue number and order status. Kitchen staff and administrators also rely heavily on read operations to view incoming orders, monitor order progress, and review system data in real time. Firebase Firestore's real-time listeners enable continuous data retrieval, ensuring that any updates made by the admin are instantly reflected on the customer interface without requiring manual refresh.

10.3 Update Operations

Update operations are used to modify existing data records in the system. In the Yatt's Kitchen system, update operations are commonly performed when administrators change the order status from Pending to Preparing and finally to Ready. Menu item details such as price, availability, or description can also be updated by the admin when necessary. These update operations ensure that the system data remains accurate and up to date, allowing customers to receive real-time status notifications and preventing inconsistencies in order handling.

10.4 Delete Operations

Delete operations are performed when data is no longer required by the system. Administrators can remove menu items that are unavailable or discontinued by deleting the corresponding records from the database. Additionally, temporary cart data is cleared automatically after successful checkout to prevent duplicate orders and maintain data cleanliness. These delete operations help optimise database storage and maintain an organised data structure.

The implementation of CRUD operations in the Yatt's Kitchen system ensures structured data management, real-time synchronisation, and reliable system performance. By leveraging Firebase Firestore's cloud-based capabilities, the system maintains data accuracy, supports seamless interaction between customers and administrators, and enhances overall operational efficiency.

To ensure that all CRUD (Create, Read, Update, Delete) operations functioned correctly from both customer and administrator perspectives, User Acceptance Testing (UAT) was conducted. The UAT scenarios were designed to directly validate CRUD-related functionalities and confirm that system behaviour matched user expectations in real-world usage.

Table shows User Acceptance Testing (UAT) Results Related to CRUD Operations

UAT ID	User Role	Scenario	Expected Outcome	Status
UAT-01	Customer	Browse menu	Menu displayed	Accepted
UAT-02	Customer	Place order	Order confirmed	Accepted
UAT-03	Customer	Receive queue	Queue shown	Accepted
UAT-04	Customer	Track status	Real-time update	Accepted
UAT-05	Admin	Login	Login success	Accepted
UAT-06	Admin	Verify payment	Queue generated	Accepted
UAT-07	Admin	Update status	Customer notified	Accepted
UAT-08	Admin	Manage menu	Changes reflected	Accepted

The UAT results demonstrate that all CRUD operations within the Yatt's Kitchen system were successfully validated. Create operations were verified through scenarios such as placing new orders and generating queue numbers (UAT-02 and UAT-06). Read operations were confirmed when customers browsed the menu and tracked order status in real time (UAT-01 and UAT-04). Update operations were validated through order status updates and menu management by administrators (UAT-07 and UAT-08). Finally, Delete operations were indirectly validated through cart clearance after checkout and menu item removal during management tasks.

The successful acceptance of all UAT cases confirms that CRUD functionalities were correctly implemented, user-friendly, and aligned with system requirements. This validation ensures reliable data handling, accurate system behaviour, and overall system readiness for deployment.

Inconsistent or duplicate data could occur when multiple users interact with the system simultaneously. Invalid order states or inaccurate menu information could disrupt operations and reduce system reliability. | Medium | Centralised Firestore operations were implemented with structured data validation rules. Order status transitions were restricted to predefined states, and input validation was applied at both client and database levels to ensure data accuracy and integrity. |

This structured approach to identifying challenges, analysing their impacts, assessing risk levels, and implementing effective solutions contributed significantly to the robustness, security, and reliability of the Yatt's Kitchen system.

11.0 Important Source Code Snippets

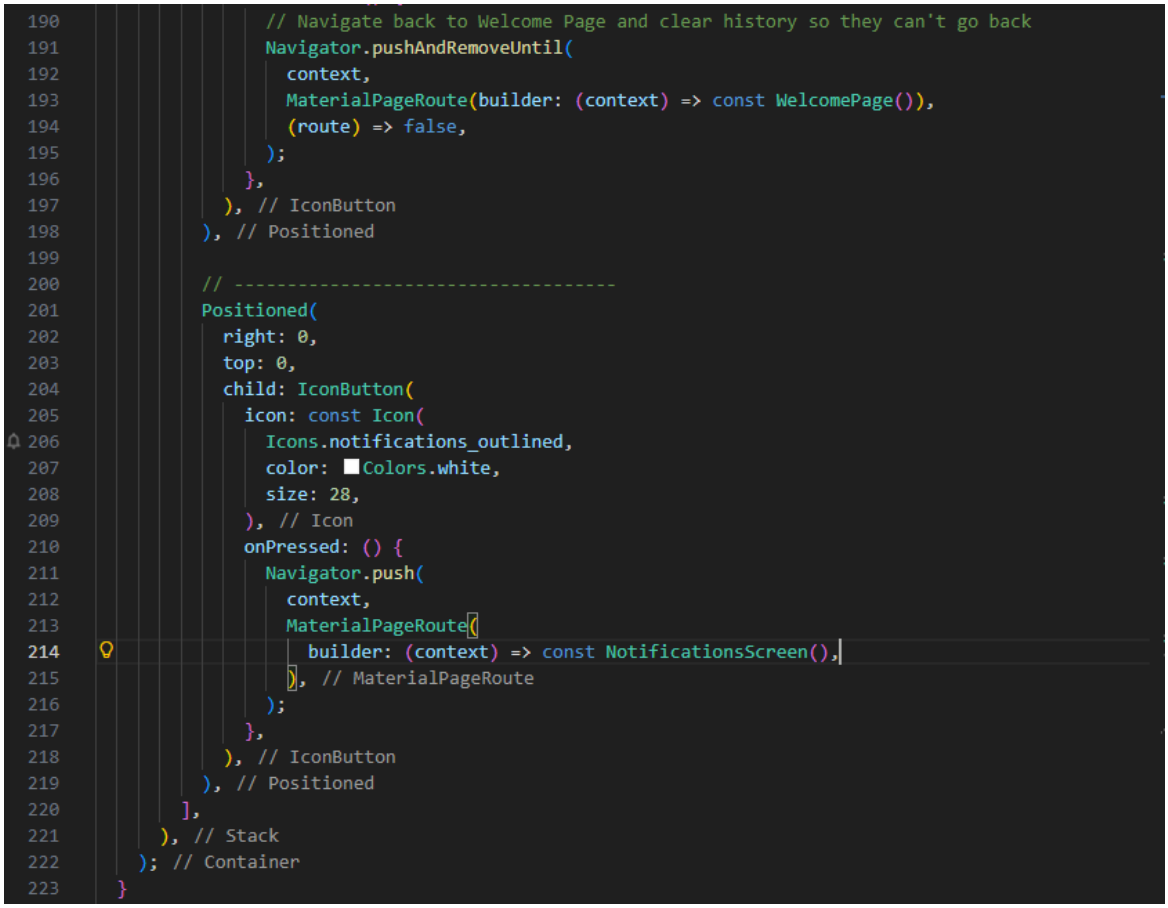
```
// 1) Firebase Auth login
final cred = await FirebaseAuth.instance.signInWithEmailAndPassword(
  email: email,
  password: password,
);

final uid = cred.user?.uid;
if (uid == null) {
  await FirebaseAuth.instance.signOut();
  throw FirebaseAuthException(
    code: 'user-null',
    message: 'Login failed.',
  );
```

The code implements secure user login using Firebase Authentication. First, the system attempts to sign in the user using the provided email and password. Firebase verifies these credentials against its authentication database and, if successful, returns a user credential object.

After authentication, the system retrieves the user's unique identifier (UID). This UID is important because it uniquely represents the logged-in user and is later used to identify the user's role and related data in the database.

A validation check is then performed to ensure that the UID is not null. If the UID is null, it indicates that the login process was unsuccessful. In this situation, the system immediately signs the user out to prevent any invalid or unauthorized session from continuing.

Finally, a 'FirebaseAuthExceptio' is thrown with a custom error message to indicate that the login has failed. This error handling mechanism ensures system security, provides clear feedback to the application, and prevents further access when authentication is not successful.



```
190                // Navigate back to Welcome Page and clear history so they can't go back
191                Navigator.pushAndRemoveUntil(
192                  context,
193                  MaterialPageRoute(builder: (context) => const WelcomePage()),
194                  (route) => false,
195                );
196              },
197            ), // IconButton
198          ), // Positioned
199
200          // ------------------------------------
201          Positioned(
202            right: 0,
203            top: 0,
204            child: IconButton(
205              icon: const Icon(
206                Icons.notifications_outlined,
207                color: Colors.white,
208                size: 28,
209              ), // Icon
210              onPressed: () {
211                Navigator.push(
212                  context,
213                  MaterialPageRoute(
214                    builder: (context) => const NotificationsScreen(),
215                  ), // MaterialPageRoute
216                );
217              },
218            ), // IconButton
219          ), // Positioned
220        ],
221      ), // Stack
222    ); // Container
223  }
```

This code controls navigation and user interaction within the application interface. The first part uses 'Navigator.pushAndRemoveUntil' to redirect the user back to the Welcome Page while clearing the entire navigation history. This prevents the user from returning to previous pages using the back button, which is commonly used after logout or session reset to improve security and user flow control.

The 'Positioned' widget is used to place an icon button at a fixed position on the screen, specifically at the top-right corner. This ensures the button remains consistently positioned regardless of screen size or layout changes. The 'IconButton' displays a notifications icon, providing a clear visual cue for accessing notifications. When the icon is pressed, the 'onPressed' function is triggered, which uses 'Navigator.push' to navigate the user to the Notifications Screen. This allows users to view notifications without affecting the existing navigation stack. Overall, this code ensures proper page navigation, prevents unwanted back navigation, and provides easy access to the notifications feature, contributing to better usability and controlled application flow.

```dart
// --- LOGIC: UPDATE FIREBASE AUTH PASSWORD ---
Future<void> _updatePassword() async {
  // 1. Basic Validation
  if (_passController.text.isEmpty || _confirmPassController.text.isEmpty) {
    _showSnackBar("Please fill in all fields", Colors.red);
    return;
  }

  if (_passController.text != _confirmPassController.text) {
    _showSnackBar("Passwords do not match", Colors.red);
    return;
  }

  if (_passController.text.length < 6) {
    _showSnackBar("Password must be at least 6 characters", Colors.red);
    return;
  }

  setState(() => _isLoading = true);

  try {
    User? user = FirebaseAuth.instance.currentUser;

    if (user != null) {
      // 2. Update the password in Firebase Auth
      await user.updatePassword(_passController.text);

      if (mounted) {
        _showSnackBar("Password updated successfully!", Colors.green);
        Navigator.pop(context);
      }
    }
  } on FirebaseAuthException catch (e) {
    // 3. Handle Re-authentication error
    if (e.code == 'requires-recent-login') {
      _showSnackBar("For security, please log out and log in again to change your password.", Colors.orange);
    } else {
      _showSnackBar(e.message ?? "An error occurred", Colors.red);
    }
  } catch (e) {
    _showSnackBar("Error: $e", Colors.red);
  } finally {
    if (mounted) setState(() => _isLoading = false);
  }
}

void _showSnackBar(String message, Color color) {
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text(message), backgroundColor: color),
  );
}
```

This code handles the password update functionality using Firebase Authentication with proper validation, security checks, and user feedback. First, basic input validation is performed to ensure that the password and confirmation fields are not empty. It then checks whether both passwords match and verifies that the password length meets the minimum security requirement of six characters. If any validation fails, an error message is displayed to the user using a SnackBar, and the process is stopped.

Once validation passes, the system sets a loading state to indicate that the update process is in progress. The currently authenticated user is then retrieved from Firebase Authentication. If a valid user session exists, the system updates the user's password securely using Firebase's 'updatePassword' method. After a successful password update, a success message is displayed, and the user is navigated back to the previous screen. This confirms that the password change was completed successfully. The code also includes error handling for Firebase authentication exceptions. If Firebase requires the user to re-authenticate for security reasons, the system displays a warning message instructing the user to log out and log in again. Any other errors are handled gracefully by displaying an appropriate error message. Finally, the loading state is reset to ensure the user interface updates correctly. Overall, this code ensures secure password management, proper validation, clear user feedback, and robust error handling within the application.

12.0 Testing & Validation

Testing and validation were carried out to ensure that the Yatt's Kitchen Smart Food Ordering and Queue Number System meets all functional and non-functional requirements defined during the system analysis and design phases. A systematic testing strategy was adopted to verify correctness, reliability, usability, and performance of the system before deployment. Testing was conducted in multiple stages to identify and resolve issues early, thereby improving overall system quality.

12.1 Unit Testing

Unit testing was performed on individual system modules to verify that each component functions as intended in isolation. Key modules such as Menu Management, Order Management, Queue Management, and Payment Verification were tested independently. For example, unit tests were conducted to ensure that menu items could be added, updated, and deleted correctly, and that order data was accurately stored in Firebase Firestore. This testing phase helped identify logic errors and ensured that each function produced the expected output.

12.2 Integration Testing

Integration testing focused on verifying the interaction between different system modules and Firebase services. This included testing data flow between the Flutter frontend and Firestore backend, ensuring that CRUD operations performed on the database were correctly reflected on the user interface in real time. Integration testing also validated Firebase Authentication for admin login and Firebase Cloud Messaging for real-time notifications. This phase ensured that all system components worked cohesively as a single integrated system.

12.3 System Testing

System testing was conducted to evaluate the complete and fully integrated system against the defined system requirements. End-to-end test scenarios were executed, such as placing an order, verifying payment, generating a queue number, updating order status, and notifying the customer. This testing confirmed that the system performed reliably under normal operating conditions and that all functional requirements were met.

12.4 User Acceptance Testing

User acceptance testing was carried out to validate the system from the perspective of real users, including customers and kitchen staff. Test users interacted with the system in a simulated restaurant environment to evaluate usability, clarity of information, and overall user experience. Feedback collected during this phase was used to improve interface design, simplify workflows, and enhance system responsiveness. Successful UAT confirmed that the system was ready for practical use.

12.5 Validation

Validation ensured that the developed system fulfilled its intended purpose of improving ordering efficiency and queue management at Yatt's Kitchen. The system was validated by comparing operational performance before and after implementation. Results showed reduced order errors, improved queue transparency, and enhanced customer satisfaction, confirming that the system met its stated objectives.

13.0 Results & Discussion

The implementation of the Yatt's Kitchen Smart Food Ordering and Queue Number System produced significant improvements in operational efficiency and service quality. One of the most notable results was the reduction in order inaccuracies. By replacing verbal order placement with a digital ordering interface, communication-related errors were largely eliminated.

Queue management also improved substantially. Automated queue number generation ensured an orderly serving sequence and reduced confusion among customers. Customers were able to track their queue number and order status in real time, which minimised repeated inquiries at the counter and reduced congestion during peak hours.

From the staff perspective, workload was significantly reduced. Staff no longer needed to manually manage queue announcements or repeatedly explain order status to customers. This allowed them to focus more on food preparation and service quality. Additionally, the availability of digital records enabled better monitoring of daily operations.

Despite its advantages, the system has certain limitations. The system currently relies on stable internet connectivity to function optimally, and payment verification still involves manual confirmation by the admin. These limitations are addressed in the future enhancement section. Overall, the results demonstrate that the system successfully achieved its objectives and delivered measurable benefits to Yatt's Kitchen.

14.0 Technical Challenges and Solutions

During the development and implementation of the system, several technical challenges were encountered. These challenges required careful analysis and appropriate technical solutions to ensure that the system functioned reliably and met its design objectives.

Challenge	Impact on System	Solution Implemented
Role-Based Access Control	Without proper role validation, unauthorised users could access admin functionalities, leading to potential data manipulation, security breaches, and incorrect system usage. Incorrect redirection could also cause confusion among users and disrupt system workflow.	User roles were stored securely in Firebase Firestore and linked to authenticated accounts. During login, the system retrieves and validates the user role before allowing navigation. Conditional routing logic was implemented to redirect users to the appropriate interface based on their role, ensuring secure and controlled access.
Real-Time Order Updates	Delayed order updates could result in kitchen staff missing new orders or processing outdated information, especially during peak hours. This would negatively affect service efficiency and customer satisfaction.	Firebase Firestore real-time listeners were implemented on the orders collection. These listeners automatically detect data changes and update the admin and customer interfaces instantly, ensuring that all users receive the most current order information without manual refresh.
Data Consistency and Integrity	Inconsistent or duplicate data could occur when multiple users interact with the system simultaneously. Invalid order states or inaccurate menu information could disrupt operations and reduce system reliability.	Centralised Firestore operations were implemented with structured data validation rules. Order status transitions were restricted to predefined states, and input validation was applied at both client and database levels to ensure data accuracy and integrity.

15.0 System Limitations & Risk Analysis

Despite the successful implementation of the Yatt's Kitchen Smart Food Ordering and Queue Number System, several limitations and potential risks have been identified. Recognising these limitations is essential for realistic system evaluation and long-term improvement.

15.1 System Limitations

One of the primary limitations of the system is its dependency on stable internet connectivity. As the system relies heavily on Firebase cloud services for real-time data synchronisation, any network disruption may temporarily affect order placement, status updates, or administrative actions. In areas with unstable internet connections, this could lead to delayed updates or reduced system responsiveness.

Another limitation is the reliance on manual payment verification by administrators. While this approach ensures payment accuracy, it may introduce delays during peak hours when multiple orders are placed simultaneously. Additionally, the system currently does not provide advanced reporting or analytics features, limiting management's ability to perform in-depth operational analysis.

The system is also designed primarily for single-outlet usage. Although scalable, it does not yet include built-in support for managing multiple branches from a centralised platform. Furthermore, the absence of user account registration limits the ability to provide personalised services such as order history tracking or loyalty rewards.

15.2 Risk Analysis

Several risks were identified during system development and evaluation. A major technical risk is data security and unauthorised access. Although Firebase Authentication and Firestore security rules are implemented, improper configuration could expose sensitive system data. This risk is mitigated through strict access control policies and regular security rule reviews.

Another risk involves system scalability during peak usage periods. A sudden surge in concurrent users could potentially impact system performance. However, this risk is reduced by Firebase’s cloud-based scalability, which dynamically manages increased loads.

Operational risks include staff resistance to adopting new technology and potential user errors during initial usage. These risks can be mitigated through proper training, clear user interface design, and user guidance within the application.

Finally, there is a dependency risk associated with third-party cloud services. Any service outage or policy change by the cloud provider could affect system availability. To mitigate this risk, regular data backups and system monitoring are recommended.

16.0 Conclusion

The Yatt's Kitchen Smart Food Ordering and Queue Number System successfully addresses the challenges associated with traditional manual ordering and queue management. By integrating mobile application technology with cloud-based services, the system provides a structured, efficient, and user-friendly solution for both customers and restaurant staff.

Throughout the development process, software engineering principles such as modular design, scalability, and usability were applied to ensure system robustness and maintainability. The system achieved its primary objectives of reducing order errors, improving queue transparency, minimising staff workload, and enhancing customer satisfaction.

In conclusion, this project demonstrates the effectiveness of digital transformation in small food businesses. The Yatt's Kitchen system serves as a practical example of how mobile and cloud technologies can be leveraged to improve operational efficiency and service quality in real-world scenarios.

17.0 Future Enhancements

Although the current system meets all core requirements, several enhancements can be implemented to further improve functionality and scalability. One potential enhancement is the integration of an online payment gateway, allowing customers to complete payments directly through the application and reducing the need for manual payment verification.

Another enhancement involves implementing intelligent queue estimation using historical order data. By analysing past order durations, the system could estimate waiting times and provide customers with more accurate service expectations. Additionally, an analytics dashboard could be developed for administrators to monitor sales trends, peak hours, and popular menu items.

Future versions of the system may also support multi-branch operations, enabling centralised management for restaurants with multiple outlets. Enhanced notification features, improved UI customisation, and offline data handling are other potential improvements. These enhancements would further strengthen the system’s value and adaptability in a growing business environment.

18.0 References

- Susnjara, S., & Smalley, I. (2025, November 24). Software Testing. IBM. https://www.ibm.com/think/topics/software-testing
- Firebase Cloud Messaging. (n.d.). Firebase. https://firebase.google.com/docs/cloud-messaging
- Firebase Authentication. (n.d.). Firebase. https://firebase.google.com/docs/auth
- Atlassian. (n.d.). The different types of testing in software | Atlassian. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing
- Sharif, A. (2024, October 25). What is CRUD? Create, read, update, and delete | CrowdStrike. CrowdStrike. https://www.crowdstrike.com/en-us/cybersecurity-101/observability/crud/
- IBM WebMethods Integration Server. (n.d.). https://www.ibm.com/docs/en/webmethods-integration/wm-integration-server/11.1.0?topic=api-crud-operations