APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 6 No. 1 (2025) 1085-1104
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

WHISKIT.CO Mobile Ordering Application

Sutanth Kunsuraman[1], Mohd Hamdi Irwan Hamzah[1]*

[1] *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: hamdi@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2025.06.01.060

Article Info

Received: 7 January 2025
Accepted: 18 June 2025
Available online: 30 June 2025

Keywords

Mobile Ordering Application, object-oriented approach , React Native, Firebase, Prototyping methodology.

Abstract

Whiskit.co, a made-to-order tart and cupcake shop in Gombak, ensures product freshness by preparing items upon order. However, the business faces significant challenges with long customer wait times and inaccurate tracking due to manual processes and more. To address these issues, a mobile ordering application is developed using the Prototyping methodology. This iterative approach enabled continuous user feedback and refinement. There are eigth modules that will be discussed in this proposed project, which is login and signup, manage profile, manage product, manage staff, manage order, make a payment, generate report and manage feeback. This proposed project developed using a object-oriented approach. The application was iteratively developed with React Native for cross-platform compatibility and Firebase for real-time operations, the app streamlines ordering and facilitates real-time communication. Hopefully, this application will help Whiskit.co shop reduce these problems.

1. Introduction

The WHISKIT.CO Mobile Ordering Application aims to address the operational inefficiencies faced by the bakery business in Gombak. Established in June 2018, WHISKIT.CO specializes in made-to-order tarts and cupcakes, ensuring freshness but resulting in significant wait times for customers. This project focuses on developing a mobile application. According to study, mobile food ordering apps can significantly improve customer satisfaction and intention to reuse by providing convenient and efficient service [1]. In addition, evaluating customers' dining attitudes, e-satisfaction, and continuance intention towards mobile food ordering apps has shown positive impacts, further highlighting the potential benefits for WHISKIT.CO [2].

The primary problem WHISKIT.CO faces is long customer wait times due to manual order processing, leading to inefficiencies and customer dissatisfaction. Additionally, the business struggles with inaccurate product tracking, which impacts order management and sales analysis. The lack of real-time updates and communication further exacerbates these issues, resulting in a suboptimal customer experience. The objectives of this project are first to design a WHISKIT.CO Mobile Ordering Application using an object-oriented approach, second is to develop the application using a mobile-based approach, and the third is to test the application through user acceptance testing. The scope of the project includes the design, development, and implementation of the mobile application, with eight key modules for login and signup, manage profile, manage product, manage staff, manage order, manage payment, manage report, and manage feedback to enhance both customer and internal operations.

The expected results of the project include significantly reduced customer wait times, improved product tracking accuracy, and enhanced overall customer satisfaction. The significance of this project lies in its potential to transform WHISKIT.CO's operations. By introducing a mobile ordering application, the business can achieve higher customer satisfaction, better resource management, and improved profitability. The system will be used by three types of users: admin, staff, and customers. The admin is the owner of the shop, responsible for

overseeing all operations, managing products, staff, and generating reports. The staff are the employees who work at the shop, handling daily operations such as managing orders, updating order statuses, and interacting with customers. Customers are individuals who place orders for tarts and cupcakes through the application, manage their profiles, make payments, and provide feedback on their purchases. This comprehensive user structure ensures that the application meets the needs of all stakeholders involved in the WHISKIT.CO business.
Section 2 covers related work, Section 3 details the software development model. Next the user interface will be included in Section 4 and Section 5 will be the conclusion of the paper.

2. Related Work

This section primarily explains technology behind the Mobile Ordering Application, its decision-making capabilities, and how it compares to the existing system.

2.1 Mobile Application Framework Techonology

The development of the Whiskit.co mobile ordering application leverages modern mobile application frameworks to ensure cross-platform compatibility and efficiency. Mobile applications have transformed how we interact with technology, integrating critical functionalities into daily life and providing essential services across diverse sectors. Cross-platform frameworks have become crucial for developing these applications, enabling a single product to operate seamlessly across various platforms. This approach not only accelerates development but also simplifies code maintenance and enhances software testing procedures, addressing key challenges faced by developers in the fast-paced tech environment [3]. The rise of mobile devices as primary tools for accessing services has driven the need for robust applications that cater to different hardware configurations, leading to the growth of platforms like Flutter and Xamarin. These frameworks support the development of native-like applications from a single codebase, expediting the development process and reducing time to market [4][5]. Ionic offers a hybrid development framework that uses web technologies to run applications within a native container on each platform, balancing native performance and web flexibility [6]. Similarly, React Native, introduced by Facebook, allows for the development of native-like apps using JavaScript, streamlining the development process without sacrificing app quality [7]. In a comparative analysis, differences in performance, developer productivity, and community support between Flutter and React Native are highlighted, providing insights into choosing the right framework based on specific project needs [8]. Furthermore, evidence of these platforms' effectiveness shows Flutter's superior UI capabilities and React Native's flexibility, significantly influencing the mobile app development landscape [9]. The choice between these frameworks often depends on project requirements, such as performance optimization, UI customization, and developer expertise, each offering distinct advantages that can significantly influence the efficiency and effectiveness of the development process.

2.2 Database Mobile Application Technology

In the realm of mobile application development, databases play a crucial role in managing the storage, retrieval, and manipulation of data. For the WHISKIT.CO Mobile Ordering Application, where real-time data processing and high data availability are paramount, the choice of database technology significantly impacts overall performance. Firebase, initially developed by Firebase Inc. and acquired by Google in 2014, is a key platform utilized for building high-quality mobile and web applications efficiently. Firebase offers a comprehensive suite of services, including analytics, authentication, databases, file storage, and push messaging, designed to accelerate development and enhance app functionality [3]. A core feature of Firebase is its Realtime Database, a NoSQL cloud database that stores and synchronizes data in JSON format across all clients instantaneously, making it ideal for applications requiring live data updates and robust offline capabilities [4]. Firebase provides two primary database solutions: the Firebase Realtime Database and Cloud Firestore. The Realtime Database excels in scenarios requiring immediate data synchronization, such as chat applications and collaborative platforms, due to its efficient JSON-based data storage and low-latency updates [5]. However, it may present challenges in managing large or complex datasets due to its hierarchical data structure, which can limit the scope of querying capabilities [4]. On the other hand, Cloud Firestore enhances Firebase's capabilities by offering a document-model architecture where data is organized into documents within collections. This structure simplifies data management and supports more sophisticated querying, making it more suitable for larger datasets and higher traffic applications. Cloud Firestore retains the real-time synchronization and offline capabilities of the Realtime Database while enhancing performance and scalability [5]. While Firebase databases are praised for real-time data updates, offline access, and seamless integration with other Firebase services, they also have limitations. The Realtime Database is highly effective for real-time data flows but can be cumbersome for managing large datasets. Cloud Firestore, although advantageous for complex applications due to its advanced querying and scalability, may introduce a steeper learning curve for developers accustomed to relational databases and could lead to higher costs with extensive data operations [6]. Therefore, selecting the appropriate database technology is critical for optimizing

performance and ensuring the scalability of the WHISKIT.CO Mobile Ordering Application. This consideration is crucial for maintaining a smooth and responsive user experience, which is essential for customer satisfaction and operational efficiency.

2.3 Study of existing system

In this section, the introduction and background study of three existing related systems are discussed. The comparison between the existing system, and the proposed system is shown in **Table 1.**

Table 1 *Comparisons of Features between Proposed System and Existing System*

Features	Eat Cake Today	Love A Loaf	CakeTella	WHISKIT.CO Application
Sign Up and Login	Use email and password		Use email or name and password	Use email and password
Manage Profile	User able to edit their profile details			Able to edit personal details
Manage Product	Cannot be accessed			Admin can add, edit and delete the product
Manage Staff	Cannot be accessed			Admin can add, and delete the Staff
Manage Order	Staff page cannot be accessed Customer can view items and can add to cart			Staff can manage incoming orders and update their status Customer can view items and can add to cart
Make a Payment	Users can make online payments.			Customer can make online payment
Scheduling pick-up timing	Customers can schedule their pick-up time.	Customers cannot schedule their pick-up time.		Customers able schedule their pick-up time.
Generate Report	Cannot be accessed			Admins can generate and view sales and product reports
Manage Feedback	Admin page cannot be accessed Customers can rate products and provide feedback.			Customers can rate products and provide feedback. Admins can view or hide feedback but cannot access the customer page.
System Domain	Web Based/Application	Web Based/Application	Web Based	Application

The proposed WHISKIT.CO Mobile Ordering Application stands out by offering more features compared to existing systems like Eat Cake Today, Love A Loaf, and CakeTella. While all these systems have basic features such as user sign-up and login, profile management, product management, order management, and payment processing, the WHISKIT.CO app includes additional benefits. It allows scheduling pick-up times, generating detailed sales reports, and collecting customer feedback, which the other systems do not offer. Also, the WHISKIT.CO application is a mobile app, making it easy for customers and staff to use it anytime and anywhere, unlike the other systems that are mainly web-based. These extra features help reduce customer wait time and provide better overall service.

3. Methodology

This section is mainly discussing the System Development Life Cycle (SDLC) model chosen to guide the development of the Whiskit.Co Mobile Ordering System.

3.1 Prototyping Methodology

The Prototyping Model is chosen for this project because it allows for iterative development and continuous feedback from end users, which is crucial for refining the system to meet the specific needs of Whiskit.Co Mobile Ordering Application. This model supports the creation of an initial prototype to test design possibilities, illustrate concepts, and gain insights into user requirements and potential solutions. By engaging users early and throughout the development process, the Prototyping Model helps ensure that the final product is user-friendly and functionally robust. Table 2 depicts the software development actives associated with the task during the software development.

Table 2 *Software Development Activities*

Phase	Task	Output
Planning	☐ Define the background of the case study ☐ Define the problem statement ☐ Prepare the proposal ☐ Gathering information about the proposal from Google Scholar	☐ Project proposal ☐ Gantt chart
Iteration 1 : Prototype (Interface)		
Analysis	☐ Conduct interviews ☐ Analyze the requirements ☐ Create UML diagrams	☐ Use Case Diagram, Activity Diagram, Class Diagram, Requirement Definition
Design	☐ Design the user interfaces ☐ Design the system architecture ☐ Design the data dictionary	☐ Architecture Diagram, interfaces, Data Dictionary ☐ Prototype 1 (with interface)
Implementation	☐ Implement Prototype 1	☐ Prototype 1
Iteration 2 : Prototype (Database)		
Analysis	☐ Validate the requirement	☐ Prototype with database
Design	☐ Design database	☐ Schema table
Implementation	☐ Implementation prototype	☐ Prototype 2
Implementation system	☐ Implement system ☐ Test the system	☐ Test case ☐ Requirement traceability matrix ☐ Completed system

3.2 Analysis

The list of requirement for each use case are listed in Appendix A.

3.3 Use Case and Class Diagram

Fig. 1 diagram for the WHISKIT.CO Mobile Ordering Application illustrates the interactions between different user types Admin, Staff, Customer and the system's functions. Admins can manage the overall system, including adding, updating, and removing products, manage staff, generating reports, and feedback. Staff members are responsible for managing customer orders. Customers can sign up, log in, manage their profiles, place orders, makea a

payments, and provide feedback. The diagram highlights the main actions each user can perform, ensuring a clear understanding of the system's functionalities and user roles.

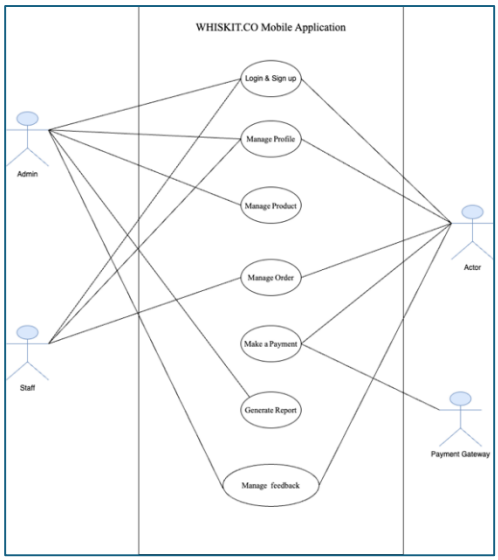


Fig . 1 *Use Case for Diagram WHISKIT.CO Mobile Ordering Application*

A class diagram has been prepared for WHISKIT.CO Mobile Ordering Application shown in **Fig 2**. In general admin, staff, customer, product, order, payment, receipt, report, and feedback. Each class has its own attributes and relationship.

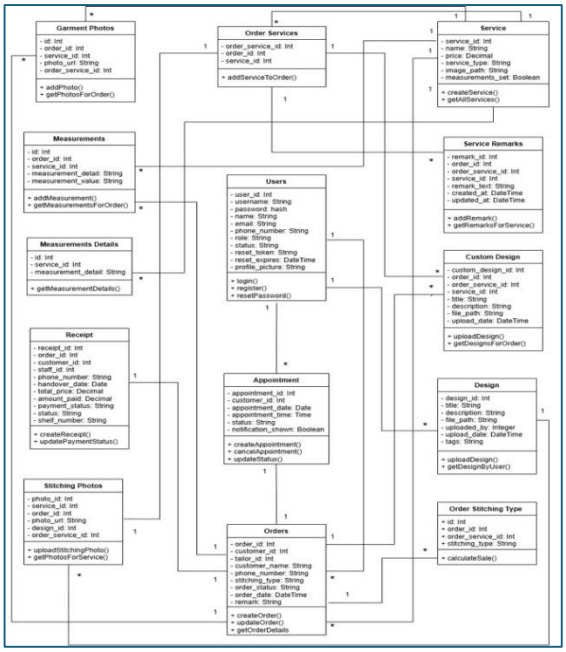


Fig. 2 *Class Diagram for WHISKIT.CO Mobile Ordering Application*

3.4 Schema table

The schema table represents the organization and structure of the database in the WHISKIT.CO Mobile Ordering Application. Each table within the database has attributes defined by the schema. The following tables show all schema tables in the WHISKIT.CO Mobile Ordering Application database. They were created using Firebase database.

i. Admin (`admin_id(PK), name, email, phone_number, password, image_url)`.

ii. Staff (`staff_id(PK), name, email, phone_number, password, status, image_url)`.

iii. Customer (`customer_id(PK), name, email, phone_number, password(hashed), image_url, favorites(FK))`.

iv. Cart (`cart_id(PK), customer_id(FK), total_amount, name, price, quantity, image_url, stock)`.

v. Product (`product_id(PK), name, description, price, category_id(FK),stock, status, image_url)`.

vi. Category (`category_id (PK), name)`.

vii. Order (`order_id(PK), customer_id(FK), staff_id(FK),product_id(FK), total_amount, pickup_option, pickup_time, status, creates_at,updated_at,name, quantity, image_url)`.

viii. Payment (`payment_id(PK), order_id(FK), customer_id(FK), amount, payment_status, payment_method, transaction_data`.

ix. Promotion (`promotion_id(PK), start_data, end_data, image_url`).

x. Flash Sale (`flash_sale_id(PK, title, description, discount_type, start_date, end_date, status(Boolean)`.

xi. Feedback (`feeback_id(PK),customer_id(FK),order_is(FK), product_is(Fk), rating, comments, categories, created_at)`

4. Result and Dicussion

This section is separated into two parts, the implementation part that shows all webpage created and code segment used in the system while testing part that summarizes the test results of the system.

4.1 Implementation

The login process begins when the user interacts with the Login Interface, as shown in Figure 4(a). The user provides their email and password, and clicking the "Login" button triggers the handleAuthentication function, depicted in Figure 4(b). This function validates the inputs, ensuring the email is properly formatted and the password is not empty. If the inputs pass validation, the system checks for authentication errors like incorrect credentials. Upon successful validation, the user is logged in, while any errors are displayed to guide the user. Similarly, the sign-up process is initiated through the interface in Figure 4(c). Users input their name, email, phone number, and password. Clicking "Sign Up" triggers the same handleAuthentication function, as shown in Figure 4(d), which validates the inputs against criteria like password strength. If all validations pass, the system creates the account, saves the user's details, and registers them successfully. The forgot password process is handled through, where users provide their registered email and click "Send Reset Link." This triggers the sendPasswordResetEmail function, depicted , which validates the email format and checks its existence in the system. If valid, a password reset link is sent to the user's email. For administrative functionality, developers provide administrators with login credentials, allowing them to access the system via Figure 4(a) and manage staff accounts using the same handleAuthentication process shown in Figure 4(b). Administrators can create staff accounts, and credentials are sent to staff via email. Customers, on the other hand, can independently register through the Sign Up Interface, as shown in Figure 4(c) and Figure 4(d). This setup ensures seamless management for users, staff, and administrators.

Applied Information Technology and Computer Science Vol. 6 No. 1 (2025) p 1085-1104

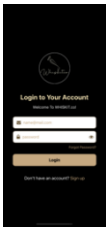


Fig. 4(a) *Login Interface;*

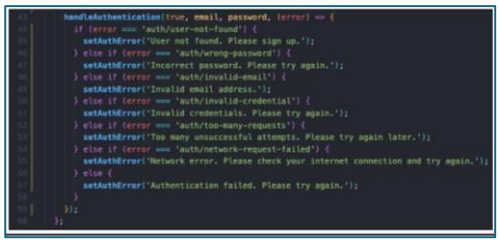


Fig. 4(b) *Login Code;*

Fig. 4*(c) Signup Interface*

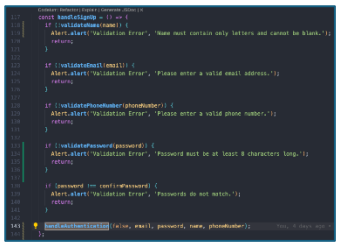


Fig. 4(d) *Signup Code*

The Admin Staff View interface, depicted in Figure 5(a), allows administrators to view a list of staff members with their respective details, such as ID, name, and phone number. When the admin interacts with this interface, the staff data is fetched from the database using the code shown in Figure 5(b). The fetchStaff function retrieves the staff list from Firebase, and each staff item is displayed in the form of a card using the StaffItem component. The admin can also navigate to a detailed view of individual staff members or use the "Create Staff" button to add a new staff member.The Admin Add Staff interface, shown in Figure 5(c), provides a form for administrators to input staff details, such as name, email, phone number, and generate a password. The password is created using the generatePassword function, and the join date is automatically populated using the current date and time. When the Create Staff button is clicked, the system validates the inputs and sends a request to create a staff account using the code illustrated in Figure 5(d). Additionally, the system sends the generated credentials to the staff's email using the sendEmailToStaff function. This workflow ensures administrators can efficiently manage staff accounts while keeping them informed via email .

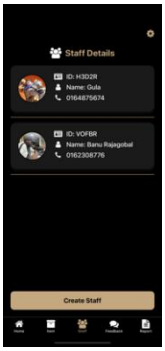


Fig. 5*(a) Admin Staff View*

Fig. 5*(b) Admin Staff View Code*

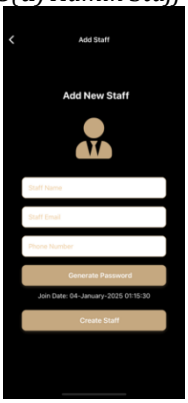


Fig.5 *(c) Admin Add Staff*

Fig.5 *(d) Admin Add Staff Code*

The admin product management system is supported by intuitive interfaces and efficient functionalities. Figure 6(a) illustrates the Product List View Interface, where administrators can view, search, edit, or delete products, toggle availability, and access product details such as name, category, price, and images. This interface dynamically fetches data from the database using getDocs.Figure 6(c) depicts the Add Product Interface, where administrators input product details including name, description, price, category, and optional images. The corresponding Add Product Code in Figure 4(d) validates inputs, uploads images using uploadBytesResumable, and saves data with unique IDs using setDoc. Key functions like handleSubmit and handleImagePick simplify these processes. The interface also includes category management functionality, allowing administrators to create, edit, or delete categories, ensuring seamless product handling.

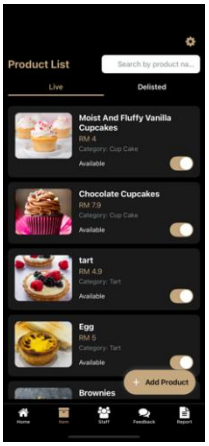


Fig. 6 *(a) Product View*

```
Codeium: Refactor | Explain | X
const fetchItems = async () => {
  try {
    const querySnapshot = await getDocs(collection(db, 'products'));
    const itemsList = querySnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
    setItems(itemsList);
    // Fade-in animation for the list
    Animated.timing(fadeAnim, {
      toValue: 1,
      duration: 1000,
      useNativeDriver: true,
    }).start();
  } catch (error) {
    console.error("Error fetching items: ", error);
    Alert.alert('Error', 'Failed to fetch items. Please try again.');
  } finally {
    setLoading(false);
  }
};
```

Fig. 6 *(b) Product View Code*

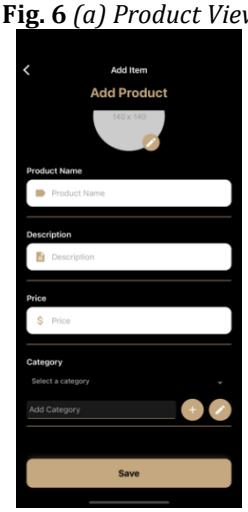


Fig. 6*(c) Product View Code*

```
const handleSubmit = async () => {      You, 2 months ago • Save progress before upgrad
  // Parse the price to extract the numerical value
  const parsedPrice = parseFloat(newProduct.price.replace('RM ', '').replace(',', ''));

  // Optional: Log the parsed price for debugging
  console.log('Parsed Price:', parsedPrice);

  // Validate the name and the parsed numerical price
  if (!validateName(newProduct.name)) {
    Alert.alert('Invalid Name', 'Please enter a valid product name.');
    return;
  }

  if (!validatePrice(parsedPrice)) {
    Alert.alert('Invalid Price', 'Please enter a valid price.');
    return;
  }

  if (!newProduct.category) {
    Alert.alert('Invalid Category', 'Please select a category.');
    return;
  }

  try {
    // Generate a new DocumentReference with a unique id
    const docRef = doc(collection(db, 'products'));
    const productId = docRef.id;

    // Prepare product data with id and default stock of 30
    const productToAdd = {
      ...newProduct,
      price: parsedPrice,
      id: productId,
      stock: 30, // Set default stock here
```

Fig.6 *(d) Product View Code*

The admin order management system features an interface Figure (j) for staff to view, search, and filter orders by status, date, or pickup option. Orders include details like customer name, total amount, and status with color-coded indicators. Staff can update statuses using a predefined flow. The code handles data fetching with fetchAllOrders and updates using updateOrderStatus, while features like debounced search and reset filters ensure efficient performance. The customer order management system includes interfaces for viewing and managing orders. Figure 4(o) shows the Orders Screen, displaying order details like ID, status, pickup type, time, and items. The connected code uses Firebase's onSnapshot for real-time updates and navigation. The Order Detail interface provides item images, quantities, total amounts, and an invoice view, with dynamic data fetching for a seamless experience.

Applied Information Technology and Computer Science Vol. 6 No. 1 (2025) p 1085-1104

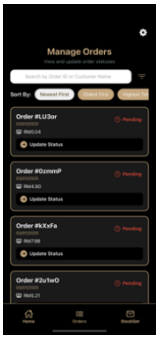


Fig. 7*(a) Staff Mange Order*

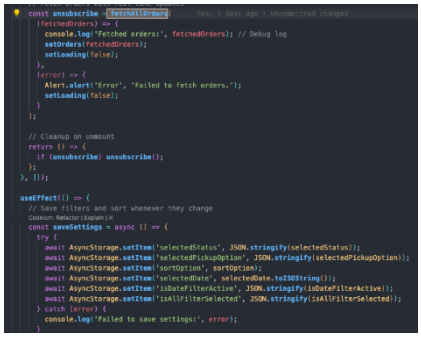


Fig. 7*(b) Staff Mange Order Code*

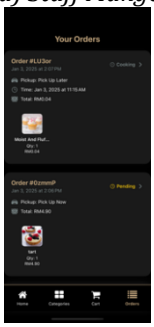


Fig. 7 *(c) Staff Mange Order*

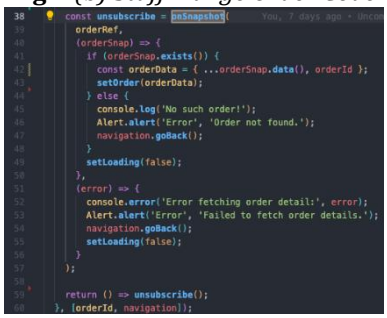


Fig. 7 *(d) Staff Mange Order*

Figure 8(a) showcases the openPaymentSheet function, which manages the payment process using Stripe. It disables user interactions during payment, displays the payment sheet via presentPaymentSheet(), and handles errors gracefully, such as canceled payments or specific issues. On success, it stores order details, clears the cart, and navigates to the order confirmation screen. The function ensures a smooth and secure payment experience while managing UI updates and error handling efficiently.

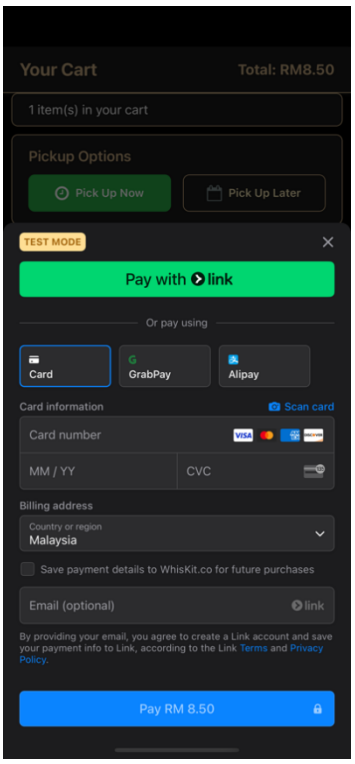


Fig. 8*(a) Payment sheet*

Fig 8(a).PaymentSheet function code

The Feedback Management Code Figure 9(a) underpins the system by handling key operations such as real-time feedback data fetching with Firebase's onSnapshot, filtering and searching feedback based on user input or selected ratings, and updating feedback statuses or visibility using updateDoc. Functions like handleSearch,

applyFilters, handleMarkAsResolved, and handleToggleHide ensure seamless and efficient management. By combining user-friendly interfaces with robust backend functionality, the system enables administrators to efficiently manage feedback, providing actionable insights to enhance customer satisfaction and service quality. The Feedback Submission Code Figure.9.(c) handles critical operations like collecting feedback data from custoemers, validating inputs, and submitting feedback to Firebase using functions such as saveOrderFeedback. It includes features like real-time character count updates, emoji-based rating selection, and predefined tags for categorizing feedback. The code ensures input validation through functions like validateFeedback, preventing incomplete submissions. Additionally, tactile feedback via Haptics enhances user interaction, while features like dynamic theming based on appearance settings provide a visually engaging experience. This robust system ensures customers can easily submit feedback, empowering businesses to gather actionable insights for improvement.

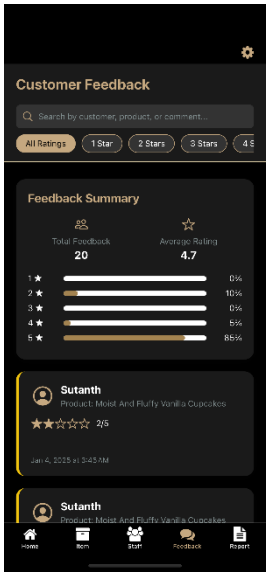


Fig. 9*(a) Admin View Feedback*

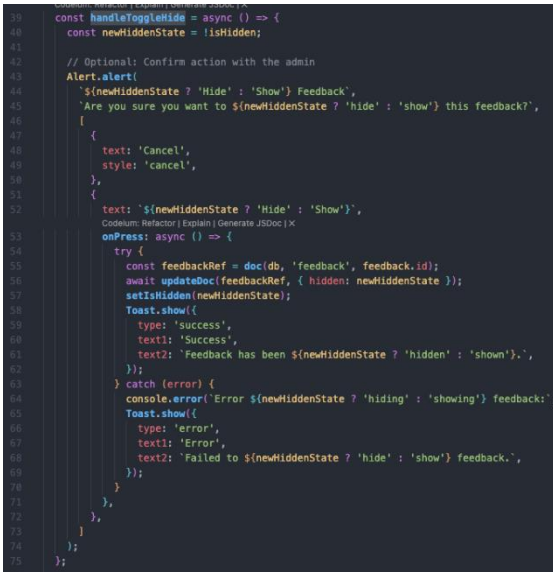


Fig. 9*(b) Admin Feedback hide function*

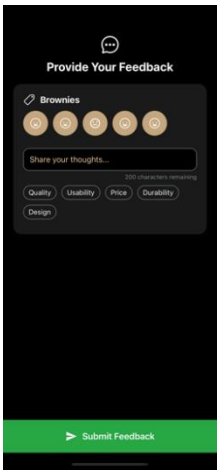


Fig. 9*(c) Feedback Submission*

Fig. 9*(d) Feedback Submission Feedback*

The StockSetScreen fetches categories using fetchCategories, dynamically updating the interface as new categories are added or removed. Staff can navigate to a specific category to view its products. Within the ProductListScreen, functions such as fetchProductsByCategory load products based on the selected category, and increaseStock or decreaseStock allow staff to adjust stock levels for each product. Editing involves real-time updates through the editedProductId and editedQuantity states, with a confirmation modal ensuring accidental changes are avoided. Features like LayoutAnimation enhance user experience by providing smooth transitions. The system also includes validation, preventing negative stock values or incomplete updates. This robust, user-friendly interface enables businesses to track and adjust inventory efficiently.

Applied Information Technology and Computer Science Vol. 6 No. 1 (2025) p 1085-1104

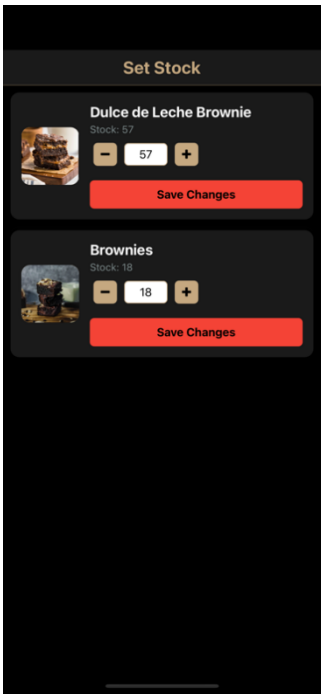


Fig. 10*(a)* Stock SetS creen

Fig. 10*(b)* Stock SetS creen Code

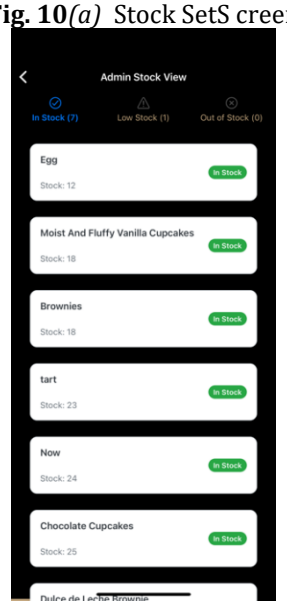
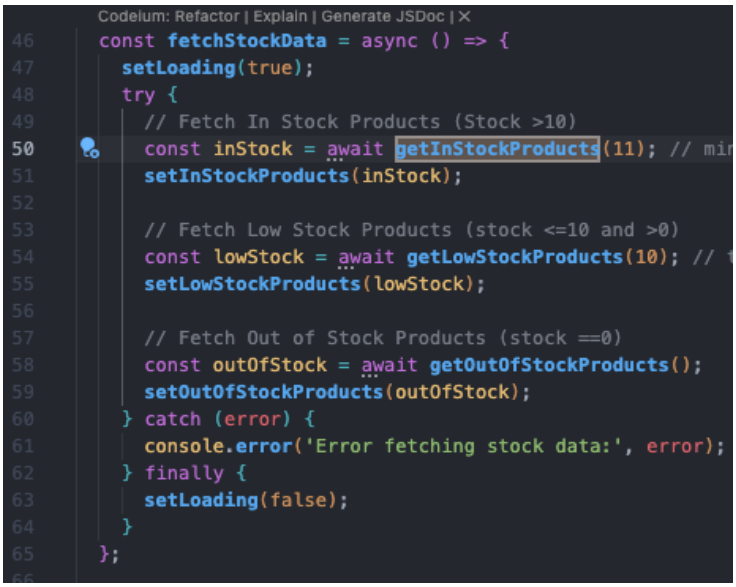


Fig. 10*(c) Admin Stock Check Interface*



```
Codelum: Refactor | Explain | Generate JSDoc | X
const fetchStockData = async () => {
  setLoading(true);
  try {
    // Fetch In Stock Products (Stock >10)
    const inStock = await getInStockProducts(11); // mi
    setInStockProducts(inStock);

    // Fetch Low Stock Products (stock <=10 and >0)
    const lowStock = await getLowStockProducts(10); //
    setLowStockProducts(lowStock);

    // Fetch Out of Stock Products (stock ==0)
    const outOfStock = await getOutOfStockProducts();
    setOutOfStockProducts(outOfStock);
  } catch (error) {
    console.error('Error fetching stock data:', error);
  } finally {
    setLoading(false);
  }
};
```

Fig. 10*(d) Admin Stock Check Interface*

4.2 Testing

After the system prototypes is finished, testing begins. System testing and acceptability testing are the two kinds of testing that are necessary. System testing assesses the produced system's usability and functionality. In order to perform acceptance testing, user input is gathered using questionnaires that highlight the characteristics of the system. This procedure guarantees that the system functions as intended and conforms to the objectives and needs of the users. The total testing findings are displayed in Table 3, and a bar chart summarizing them is displayed in Figure 9. In conclusion, all of the test case modules' functions are working as intended, and none of the test cases in any module have failed. As a result, every test case module produces a 100% outcome, which sums up to a 100% result. Appendix B displays details of the tets cases.

Table 3 *List of Test Cases*

Test Case	Software Requirement	Description	Status
TC_100_01	4	4	0
TC_100_02	5	5	0
TC_100_03	6	6	0
TC_100_04	3	3	0
TC_100_05	5	5	0
TC_100_06	2	2	0
TC_100_07	2	2	0
Total	4	4	0

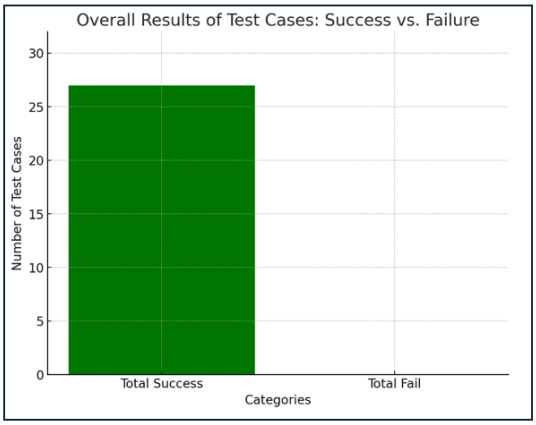


Fig. 11 *Bar Chart of Overall Result of Test Cases*

The details of the test cases is shown in the Appedix B

4.3 User Acceptance Testing

Out of a total of 10 respondents, each was asked to indicate their level of acceptance for the given statements on a five-point rating scale. The questionnaire responses and evaluations for each statement are summarized in Table 8. In conclusion, the majority of respondents rated the system's functionality and user interface with a score of 5. This indicates a high likelihood of the system being accepted by its users. he outcome after collecting feedback from 10 respondents for the interface and module evaluation is represented as an average displayed in a bar graph show in Figure.12. Appendix C contains the details for the system module and user interface evaluations.

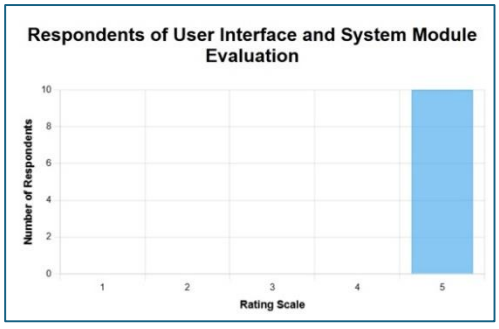


Fig. 12 *Bar Chart of Overall Respondenets*

5. Conclusion

WHISKIT.CO is a user-friendly mobile ordering application that enables customers to place orders efficiently and track their order status in real-time. It is not only beneficial for customers but also for the staff, management, and admin of WHISKIT.CO. By having this kind of application, the staff can manage orders more effectively, ensuring accurate order processing, while admin can oversee operations and make informed decisions based on real-time data. However, as with any system, there are limitations. Currently, the application focuses on streamlining the

Applied Information Technology and Computer Science Vol. 6 No. 1 (2025) p 1085-1104

ordering process. Future enhancements may include additional features to better match evolving business needs and customer preferences.

References

[1] Al Amin, M., Arefin, M. S., Sultana, N., Islam, M. R., Jahan, I., & Akhtar, A. (2020). Evaluating the customers' dining attitudes, e-satisfaction and continuance intention toward mobile food ordering apps (MFOAs): Evidence from Bangladesh. *European Journal of Management and Business Economics, 30*(2), 211–229.

https://science.howstuffworks.com/science-vs-myth/strange-creatures/sasquatch-bigfoot-difference.htm

[2] Zohud, A., & Zein, S. (2021). Cross-platform mobile app development in industry: A multiple case-study. *International Journal of Computing, 20*(1), 46–54. Retrieved from https://www.researchgate.net/profile/Samer-Zein-2/publication/350497565_CrossPlatform_Mobile_App_Development_in_Industry_A_Multiple_Case-Study/links/6069f91292851c91b1a2be07/Cross-Platform-Mobile-App-Development-in-Industry-A-Multiple-Case-Study.pdf

[3] Flutter. (n.d.). *Build apps for any screen.* Retrieved from https://flutter.dev/

[4] Microsoft. (n.d.-a). *Mobile app development & app creation software.* Retrieved from https://dotnet.microsoft.com/apps/xamarin

[5] Ionic Framework. (n.d.). *Ionic Framework – The cross-platform app development leader.* Retrieved from https://ionicframework.com/

[6] React Native. (2024, May 22). *Get started without a framework.* Retrieved from https://reactnative.dev/docs/getting-started-without-a-framework

[7] Khan, A. U., Nabi, & Bhanbhro, T. H. (2022). Comparative analysis between Flutter and React Native. *International Journal of Artificial Intelligence and Mathematical Sciences, 1*(1), 15–28. https://doi.org/10.58921/ijaims.v1i1.19

Appendix A:

Applied Information Technology and Computer Science Vol. 6 No. 1 (2025) p 1085-1104

Table A .1 ***Requirement***

Software Requirement Specification	Description
SRS_REQ_100	**Login & Sign-Up**
SRS_REQ_101	The system shall allow the user to login with a correct email and password.
SRS_REQ_102	The system shall allow the user to sign up with user details and valid email, password and confirm password.
SRS_REQ_103	The system shall validate the email during sign-up.
SRS_REQ_104	The system shall redirect the user to the home page after a successful login.
SRS_REQ_105	The system shall display a message indicating successful login or sign-up.
SRS_REQ_106	The system shall display an error message if there is a blank text field or invalid login attempt.
SRS_REQ_107	The system shall handle exceptions, such as invalid credentials or used email addresses
SRS_REQ_108	The system shall update the new password in the database if the user forgets their password.
SRS_REQ_200	**Manage Profile**
SRS_REQ_201	The system shall allow the user to access profile management options from the settings page.
SRS_REQ_202	The system shall display personal details and log out options on the settings page.
SRS_REQ_203	The system shall allow the user to update personal details such as name, phone number, and email address.
SRS_REQ_204	The system shall validate and save the updated personal details in the database.
SRS_REQ_205	The system shall display a confirmation message after successfully updating personal details.
SRS_REQ_206	The system shall allow the user to change their password from the settings page.
SRS_REQ_207	The system shall validate the new password and confirmation password.
SRS_REQ_208	The system shall update the new password in the database if the passwords match.
SRS_REQ_209	The system shall display a confirmation message after successfully changing the password.
SRS_REQ_210	The system shall allow the user to log out from the settings page.
SRS_REQ_211	The system shall display a pop-up message asking for log out confirmation.
SRS_REQ_212	The system shall log out the user and redirect them to the login page after confirmation.
SRS_REQ_213	The system shall be able to handle exceptions.
SRS_REQ_214	The system shall allow the user to access profile management options from the settings page.
SRS_REQ_215	The system shall display personal details and log out options on the settings page.
SRS-REQ_216	The system shall allow the user to update personal details such as name, phone number, and email address.
SRS_REQ_300	**Manage Product**
SRS_REQ_301	The system shall allow the admin to access product management options from the admin dashboard.
SRS_REQ_302	The system shall display options to add, update, delete or change the status of product.
SRS_REQ_303	The system shall allow the admin to add new product by entering item details.
SRS_REQ_304	The system shall validate and save the new product details in the database.

SRS_REQ_305	The system shall display a confirmation message after successfully adding a new product.
SRS_REQ_306	The system shall allow the admin to update existing product.
SRS_REQ_307	The system shall validate and save the updated product details in the database.
SRS_REQ_308	The system shall display a confirmation message after successfully updating a product.
SRS_REQ_309	The system shall allow the admin to delete product.
SRS_REQ_310	The system shall ask for confirmation before deleting a product.
SRS_REQ_311	The system shall delete the product from the database upon confirmation.
SRS_REQ_312	The system shall display a confirmation message after successfully deleting a product item.
SRS_REQ_313	The system shall allow the admin to update the status of product available, out of stock.
SRS_REQ_314	The system shall display a confirmation message after successfully updating the status of a product.
SRS_REQ_315	The system shall display alert messages for blank text fields during product management.
SRS_REQ_316	The system shall display alert messages for duplicate item names during product management.
SRS_REQ_400	**Manage Staff**
SRS_REQ_401	The system shall display the staff management page to the admin.
SRS_REQ_402	The system shall display the list of all staff members.
SRS_REQ_403	The system shall allow the admin to view detailed information of any staff member.
SRS_REQ_404	The system shall provide an option for the admin to create new accounts for staff members.
SRS_REQ_500	**Manage Order**
SRS_REQ_501	The system shall allow staff to log in and navigate to the “Orders” page.
SRS_REQ_502	The system shall display a list of incoming orders to the staff.
SRS_REQ_503	The system shall send notifications to the staff about new orders.
SRS_REQ_504	The system shall allow staff to update the order status.
SRS_REQ_505	The system shall update the order status in the database.
SRS_REQ_506	The system shall allow customers to log in and navigate to the Home page.
SRS_REQ_507	The system shall display a list of product items on the Home page.
SRS_REQ_508	The system shall allow customers to add items to their cart.
SRS_REQ_509	The system shall allow customers to edit the quantity, special requests, add, and remove items from the cart.
SRS_REQ_510	The system shall redirect customers to the payment page after they click the checkout button.
SRS_REQ_511	The system shall display to the customer about ready-to-pick-up items after payment.
SRS_REQ_512	The system shall allow customers to view the ordered list and order status on the My Order page.
SRS_REQ_600	**Make a Payment**
SRS_REQ_601	The system shall allow customers to initiate payments through a payment gateway.
SRS_REQ_602	The system shall process and record payment details after a successful transaction.
SRS_REQ_603	The system shall generate an invoice for the completed order.
SRS_REQ_604	The system shall allow customers to view their invoices on the receipt page.

Applied Information Technology and Computer Science Vol. 6 No. 1 (2025) p 1085-1104

SRS_REQ_700	**Generate Report**
SRS_REQ_701,	The system shall display the report page
SRS_REQ_702	The system shall allow the admin to generate reports.
SRS_REQ_703	The system shall allow the admin to export.
SRS_REQ_800	**Manage Feedback**
SRS_REQ_801	The system shall allow customers to rate products on a scale of 1 to 5 stars after purchasing them.
SRS_REQ_802	The system shall allow customers to provide text feedback for purchased products.
SRS_REQ_803	The system shall save customer ratings and feedback in the database.
SRS_REQ_804	The system shall allow admins to view customer feedback.

Appendix B:

Table A .2 ***List of Test Cases***

Test Cases	Software Requirement	Description	**Status**
TC 100	**REQ 100**	**Login & Sign Up**	**Fail/Pass**
TC_100_01	FR01-01, FR0104, FR01-05	The user logs in with a correct email and password. Upon success, a message is shown, and the user is redirected to the home page.	Pass
TC_100_02	FR01-02, FR01-03, FR01-05	The user signs up with valid name, email, phone number, password and confirm password; the system validates the email and displays a success message.	Pass
TC_100_03	FR01-06, FR01-07	The user submits empty or invalid credentials. The system displays an error message and handles the exception	Pass
TC_100_04	FR01-08	The user resets the password using a registered email the system updates the new password in the database upon confirmation.	Pass
TC 200	**REQ 200**	**Manage User Profile**	**Fail/Pass**
TC_200_01	FR02-01, FR02-02, FR02-10, FR02-11, FR02-12	The user navigates to profile management in settings, sees personal details and logout option, logs out, and is redirected to login after confirmation.	Pass
TC_200_02	FR02-03, FR02-04, FR02-05	The user updates name, phone the system validates inputs, saves changes, and displays a confirmation message.	Pass
TC_200_03	FR02-06, FR02-07, FR02-08, FR02-09	The user enters a new password and confirm password; the system validates both matches, updates the Firebase, and shows success.	Pass
TC_200_04	FR02-13	Test how the system handles invalid updates like duplicate email or blank fields and ensures the exception is handled gracefully.	Pass
TC_200_05	FR02-14, FR02-15, FR02-16	Verifies repeated or additional calls to load settings page, display personal details, and update them covering FR02-14,15,16 if they differ from FR02-01,02,03.	Pass
TC 300	**REQ 300**	**Manage Product**	**Fail/Pass**
TC_300_01	FR03-01, FR03-02	The admin logs in, navigates to the admin dashboard, and sees the options to add, update, delete, change product status.	Pass
TC_300_02	FR03-03, FR03-04 , FR03-05	The admin enters product details the system validates, saves the product in the Firebase and displays a success confirmation.	Pass
TC_300_03	FR03-06, FR03-07, FR03-08	The admin edits a product's information; the system validates, saves updates and displays a success confirmation.	Pass
TC_300_04	FR03-09, F R03-10, FR03-11, FR03-12	The admin chooses to delete an existing product the system asks for confirmation, deletes from the Firebase, and shows a confirmation message.	Pass
TC_300_05	FR03-13, FR03-14	The admin marks a product as "Live" or "Delisted" system updates Firebase.	Pass

TC_300_06	FR03-15, FR03-16	Admin tries to add update product with blank fields, or a duplicate item name system shows the appropriate error.	Pass
TC 400	**REQ 400**	**Manage Staff**	**Fail/Pass**
TC_400_01	FR04-01, FR04-02	The admin navigates to staff management and sees the list of all staff members.	Pass
TC_400_02	FR04-03	The admin selects a staff member to view detailed info.	Pass
TC_400_03	FR04-04	The admin creates a new staff account system saves it and confirms the creation.	Pass
TC 500	**REQ 500**	**Manage Order**	**Fail/Pass**
TC_500_01	FR05-01, FR05-02, FR05-03	Staff logs in, navigates to Order page, sees a list of incoming orders, and is view of new orders.	Pass
TC_500_02	FR05-04, FR05-05	Staff updates an order's status, and the system updates the Firebase accordingly.	Pass
TC_500_03	FR05-06, FR05-07, FR05-08	Customer logs in, sees product list, and adds items to the cart.	Pass
TC_500_04	FR05-09	Customer modifies item quantity or removes items.	Pass
TC_500_05	FR05-10, FR05-11, FR05-12	Customer proceeds to payment page, completes payment, they can view about ready to pick-up items and sees order status.	Pass
TC 600	**REQ 600**	**Manage a Payment**	**Fail/Pass**
TC_600_01	FR06-01, FR06-02	Customer pays via a gateway; system processes and records payment and updates Firebase.	Pass
TC_600_02	FR06-03, FR06-04	After payment success, system generates an invoice and allows customer to view it.	Pass
TC 700	**REQ 700**	**Generate Report**	**Fail/Pass**
TC_700_01	FR07-01, FR07-02	Admin navigates to the report page, selects criteria, and the system generates a report.	Pass
TC_700_02	FR07-03	Admin exports the generated report in a chosen format in PDF, CSV, and Excel.	Pass
TC 800	**REQ 800**	**Manage Feedback**	**Fail/Pass**
TC_800_01	FR08-01, FR08-02, FR08-03	Customer rates a product 1 to 5 stars and adds text feedback, which is saved to Firebase.	Pass
TC_800_02	FR08-04	Admin views the customer feedback on a dedicated page.	Pass

Appendix C:

Table A.3 *User Interface Evaluation*

No	Statement	Scale					Total
		1	**2**	**3**	**4**	**5**	
1	Is it easy for you to learn how to use the WHISKIT.CO ordering features?	0	0	0	0	10	10
2	Can you do what you want in the app (like adding items to your cart) without much trouble?	0	0	0	0	10	10
3	Do you find the app's interface clear and easy to understand?	0	0	0	0	10	10
4	Do you feel the app is flexible enough for tasks like changing quantities or adding special requests?	0	0	0	0	10	10
5	Overall, do you find the WHISKIT.CO Mobile Ordering app simple and user-friendly?	0	0	0	0	10	10

Table A.4 *Application Module Evaluation*

No	Statement	Scale					Total
		1	**2**	**3**	**4**	**5**	
1	Does the WHISKIT.CO Mobile Ordering app help you place orders more quickly?	0	0	0	0	10	10
2	Does the app increase your satisfaction by showing order status or product details?	0	0	0	0	10	10
3	Does the app make you more effective in tracking your orders?	0	0	0	0	10	10
4	Does the app make your ordering process easier like payments?	0	0	0	0	10	10
5	Overall, do you find the WHISKIT.CO Mobile Ordering app useful for your needs?	0	0	0	0	10	10