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

AITCS

Vol. 7 No. 1 (2026) 1490-1509
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

GoBubble: Carwash Service Management Application

Mohd Nizwan Mohd Asri[1], Rozita Abdul Jalil[1]*

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

*Corresponding Author: rozita@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2026.07.01.080

Article Info

Received: 12 June 2025
Accepted: 18 June 2026
Available online: 30 June 2026

Keywords

Carwash management, Mobile application, Digital transformation, Service automation, Customer engagement

Abstract

The car wash industry faces significant operational challenges due to traditional manual booking and service management processes. This paper presents the development of a GoBubble: Carwash Services Management Application designed to modernize and streamline operations for service providers and customers. The system addresses key issues including, manual booking inefficiencies, payment processing complications, and customer engagement limitations. The Agile development methodology used to develop this application incorporates modules for user authentication, service management, booking and payments, notifications, feedback, analytics, and loyalty programs. The system has been developed using modern technologies, including Flutter for cross-platform compatibility and Firebase for backend services. Initial testing demonstrates improved operational efficiency and enhanced user satisfaction compared to traditional manual processes. The findings highlight the potential of digital transformation in revolutionizing traditional service industries through automated processes and improved customer engagement.

1. Introduction

The car wash sector has experienced significant growth due to increasing demand for convenient automotive maintenance services [1]. As consumer lifestyles become busier, there is a growing need for efficient, technology-driven solutions that can streamline service delivery while maintaining quality. Traditional car wash services, which often rely on manual processes for bookings and payments, struggle to meet modern customer expectations for convenience and digital accessibility. Developing a comprehensive digital solution like the GoBubble,can enhance customer satisfaction and streamline processes.

In recent years, the proliferation of mobile technology has fundamentally changed how consumers interact with service providers. The car wash industry, however, has been relatively slow to adapt to this digital transformation, with many businesses still relying on traditional methods of service delivery. This technological gap creates inefficiencies in service delivery and limits the potential for business growth and customer engagement. The proposed application aims to bridge this gap by providing a modern, user-friendly platform that meets the expectations of today's tech-savvy customers.

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 1490-1509

2. Related Work

2.1 Introduction

This section discusses the type of existing system to be compared with the developed system. Each system has its own advantages and disadvantages depending on the type of system that suits the user perfectly. It will discuss the literacy studies that have been analyzed and studied based on previous research topics. The literacy study aims to provide a clear picture of some questions to be studied through various methods to find requirements. In achieving analysis and meeting the requirements needed, reading methods, evaluation, and analysis are mainly related to the event management system topic needed in the development process of this project.

2.2 Case Study: Existing System

The research reviewed three carwash management systems to identify current strengths and gaps[4]. WashMe offers easy service management but lacks advanced analytics. Govicle supports online booking with location-based service finding, though its payment and CRM features are limited. Washify stands out with real-time availability, loyalty features, and secure payments but still falls short in analytics and automation capabilities.

2.3 Comparison with the Existing Application

Through the evaluation of existing systems, the proposed GoBubble adopts key strengths while addressing common shortcomings [5]. It features secure user authentication and efficient service management, along with a notification system that enhances communication throughout the service process.The application also introduces an advanced analytics module to provide business insights, a component often missing in current platforms. Additionally, its loyalty system improves customer retention by expanding on features found in other solutions, creating a more well-rounded and competitive platform for both users and operators. Table 1 shows system's comparison between 3 other system with GoBubble

Table 1: *System's Comparison*

Features/System	GoWash	Govicle	Washify	GoBubble
User Authentication	√	√	√	√
Service Management	X	X	√	√
Booking and Payments	√	√	√	√
Notification	√	X	X	√
Feedback and Support	√	X	√	√
Analytics Module	X	X	X	√
Loyalty Module	X	√	√	√

3. Methodology

The development of the Carwash Services Management Application follows the Agile methodology, emphasizing iterative development and continuous stakeholder feedback [6]. This approach enables close collaboration between the development team and stakeholders, ensuring that the final product aligns with user needs and business objectives. Table 2 shows the Software development activities and their task.

Table 2: *Software development activities and their task*

Phase	Task	Output
Requirement	The team conducts stakeholder meetings, identifies user needs through interviews and surveys, prioritizes features, creates project timelines, and allocates resources appropriately.	Requirements documentation, feature listings, sprint planning documents, project timeline, and resource allocation plans are produced.
Design	Activities include designing system architecture, creating UI mockups, planning database schema, designing workflows, and defining technology stack.	Deliverables encompass system architecture documentation, UI/UX designs, database schema, technical specifications, and wireframes.
Development	Development involves environment setup, feature implementation, module integration, code reviews, unit testing, and documentation.	Outputs include source code repository, working modules, integration reports, code documentation, and sprint progress reports.
Testing	Comprehensive testing includes creating test plans, conducting various levels of testing, and documenting issues and resolutions.	Test plans, cases, results, bug reports, coverage reports, and UAT sign-off documents are produced.
Deployment	The team prepares deployment plans, sets up production environment, deploys the application, and conducts training.	Deployment documentation, production setup, system documentation, performance reports are delivered.
Review	Post-deployment activities include collecting feedback, monitoring performance, implementing improvements, and planning updates.	Feedback reports, performance analytics, improvement plans, and maintenance schedules are maintained.

4. System Analysis and Design

The design and development of data in the system will be supported by models such as Unified Modelling Language (UML) and Class Diagram, which are essential for database production [6]. Following this chapter is crucial to ensure the implementation of a system that not only meets user demands but also aligns with the previously outlined goals.

4.1 Requirement Analysis

Functional requirements define the developed system's function as a specific behaviour that converts input to output. These requirements describe what the system should do, including tasks, services, and functionalities expected by the users.

4.2 Use Case Diagram

The use case diagram was created to illustrate the overall functionality and components of the application [7]. The description or module implemented in the suggested system will explain the use case diagram. Figure 1 shows the use case diagram that represents the overall activity of the Ordering System for Carwash Service Management Application.

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 1490-1509

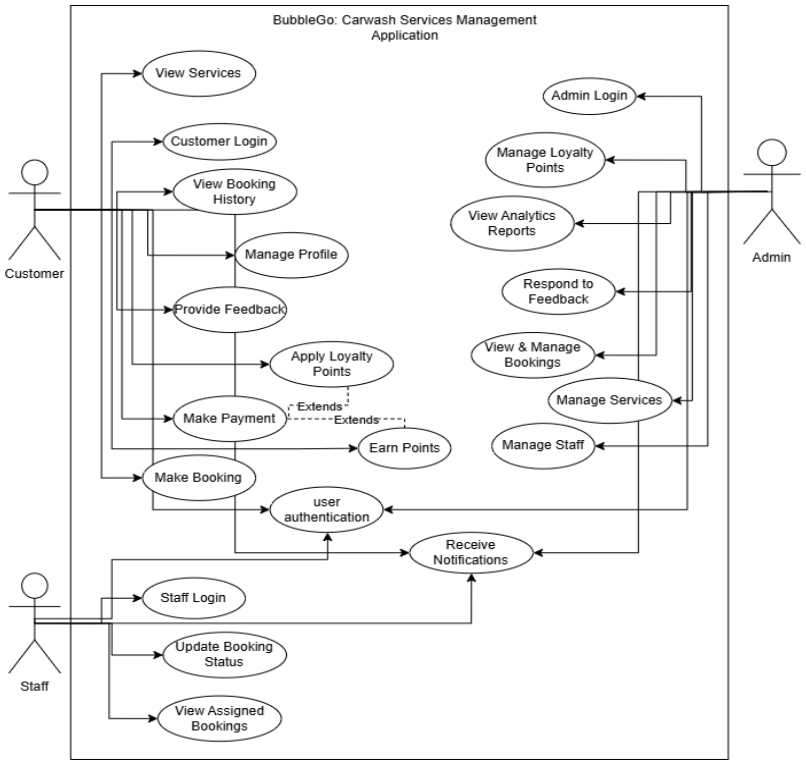


Fig. 1 *Use case diagram of the system*

4.3 Class Diagram

A class diagram is a type of UML diagram that shows the structure of a system by displaying its classes, attributes, methods, and relationships [8]. It's used in object-oriented design to model a system's static structure. Figure 2 shows the class diagram for the Carwash Service Management Application, illustrating its core components and how they are connected. The diagram highlights relationships between classes, like those between users, carts, products, and orders, helping to clarify how the system components interact and supporting efficient design and development.

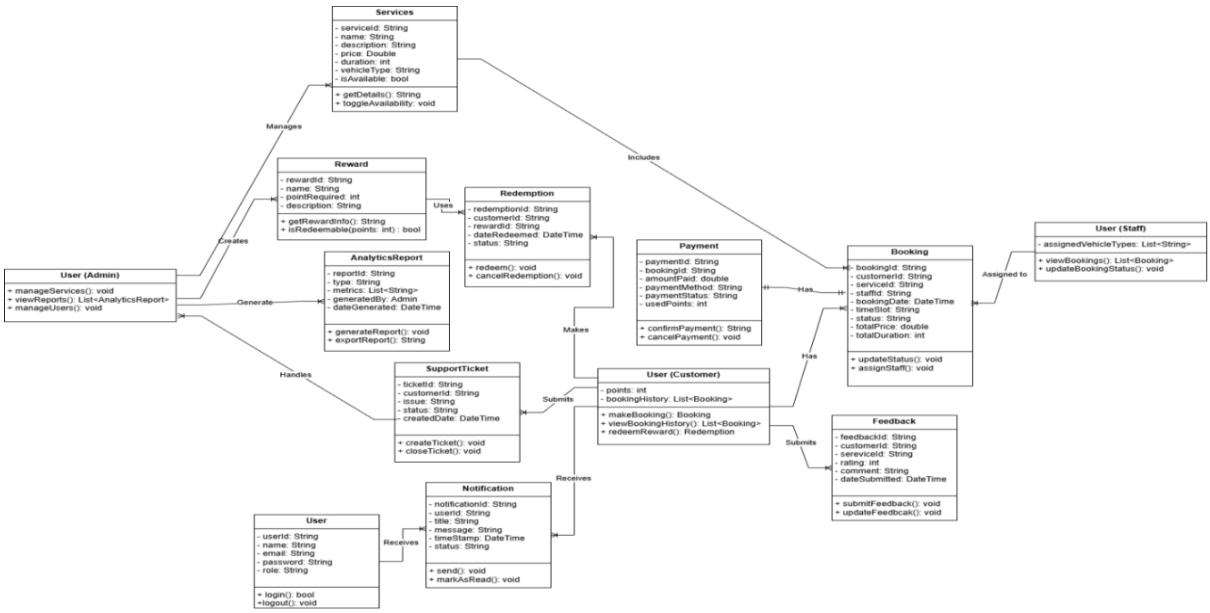


Fig. 2 *Class diagram of the system*

4.4 Database Design

Schema Table and Data Dictionary

The table 3 below are derived from the database design, which is based on the class diagram created during the system analysis phase. These tables represent the entities and their relationships, reflecting the core functionalities of the application.

Table 3*: Schema Table of GoBubble*

No	Collection	Attributes
1	users	assigmentId, createdAt, email, name, role, staffId, status, uid, vehicleType
2	services	AdditionalServices, createdAt, description, duration, isActive, pricing (vehicleTtpe), rating(average, count, totalRating), serviceName, updatedAt, vehicleTypes
3	bookings	assignedAt, assignedStaffEmail, assignedStaffId, assignedStaffName, bookingId, carType, completedAt, completedByStaff, createdAt, date, discountApplied, discountCouponId, discountValue, finalprice, name, paidAt, paymentMethod, paymentStatus, phone, plateNumber, price, quantity, respondedAt, respondAt, service, staffResponse, startedAt, startedByStaff, status, timeslot, totalDuration, uid
4	notifications	bookingId, createdAt, message, read, recipientId, title, type
5	feedback	adminReply, adminReplyAt, cleanlinessRating, createdAt, isPublic, overallRating, qualityRating, reviewComment, speedRating, staffRating, suggestion, userEmail, userId, userName, vehicleType
6	coupons	claimedAt, claimedBy, createdAt, description, expiredDate, isSelected, name, requiredPoints, rewardPercent, selectedAt, selectedBy, status
7	Support_tickets	adminResponse, createdAt, message, status, subject, uid, updatedAt, userEmail

4.5 Interface Design

The interface design focuses on creating an intuitive and user-friendly experience, incorporating modern design principles and responsive layouts [9]. The design ensures efficient navigation and task completion for both customers and administrators.

4.5.1 User Authentication

The user interface of login as shown in figure 3 and user interface of registration as shown in figure 4. The need to fill in Email and Password. The information will be validated. If the information is correct, it will be redirected to the homepage. If the information is not correct, the error message will pop up. If a user needs to register, they need to click the sign-up button.

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 1490-1509

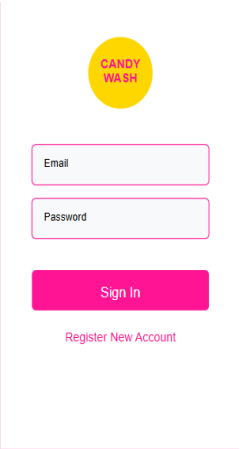


Fig. 3 *Sign In*

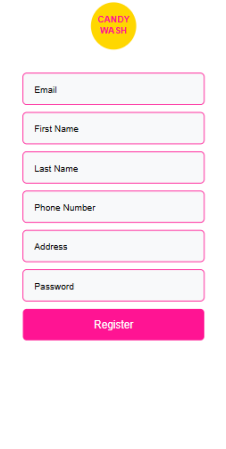


Fig. 4 *Sign Up*

4.5.2 Manage Service

Manage Interface will be on the admin side because the admin only can manage CRUD of services. The admin will fill in the details of the service and click the button created to add it. Then, it displays the list of admin and customer service. The user interface is shown in Figure 5.

4.5.3 Manage Booking

Manage Booking functionality is accessible to authorized users for overseeing and updating booking records. Users can view booking details, change booking statuses, and perform actions such as approval, rejection, or assignment based on their roles. This feature ensures efficient handling and tracking of service bookings. The user interface for managing bookings is shown in Figure 6.

4.5.4 Admin Dashboard

Admin Dashboard serves as the central control panel for administrators to oversee and manage system operations. It provides a summary of key metrics such as total bookings, user activity, service status, and revenue. The dashboard also includes quick access to important modules like bookings, users, feedback, and analytics. This centralized interface enables efficient decision-making and streamlined administration. The user interface of the admin dashboard is shown in Figure 7

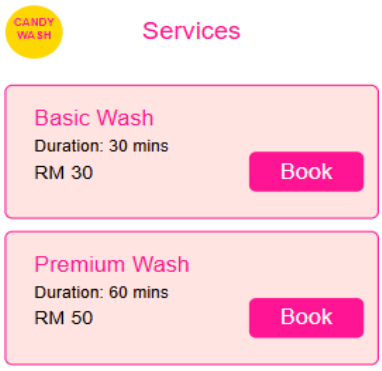


Fig. 5 *Manage Service*

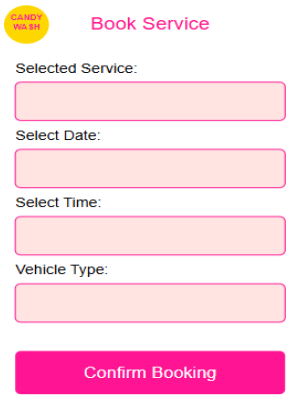


Fig. 6 *Manage Booking*

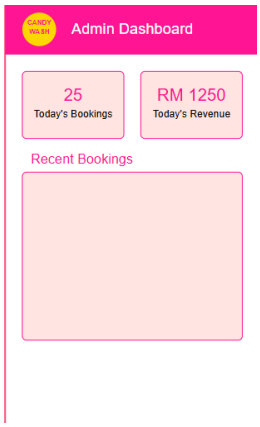


Fig. 7 *Admin Dashboard*

4.5.5 Feedback Module

Feedback Module allows administrators to view and manage customer reviews and suggestions regarding their service experience. This module displays user-submitted ratings based on various criteria such as cleanliness, speed, quality, and staff behavior. It also shows written comments and improvement suggestions. The feedback helps the admin monitor service quality, address customer concerns, and identify areas for improvement. The interface for managing feedback is shown in Figure 8.

4.5.6 Loyalty Module

Loyalty Module enables the system to reward customers for using the carwash services. Customers earn points for each successful transaction, which can be accumulated and redeemed for discounts or special rewards. The module allows users to view their current points balance, track points history, and browse available rewards. On the admin side, administrators can configure point rules, add or manage reward options, and oversee redemption activity. This encourages repeat usage and enhances customer engagement. The user interface for loyalty management is shown in Figure 9.

4.5.7 Notifications Module

Notifications Module is implemented to keep users informed about important updates related to their bookings, payments, promotions, and account activities. The system delivers real-time push notifications to users, enhancing communication and user engagement. Customers receive alerts about booking confirmations, status changes, and reward redemptions, while staff and admins are notified of new bookings and updates that require their action. The module supports both automated system notifications and manual messages from the admin. The user interface for notifications is shown in Figure 10.

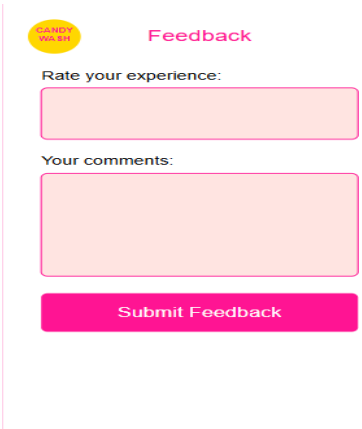


Fig. 8 *Feedback Interface*

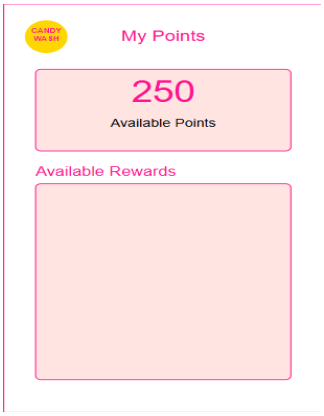


Fig. 9 *Loyalty Module*

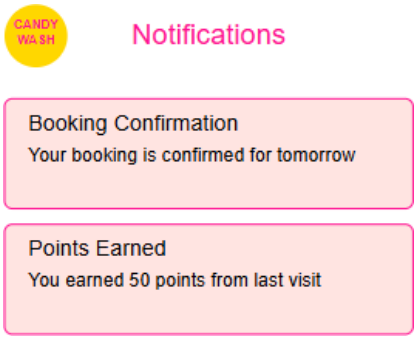


Fig. 10 *Notifications Module*

5. Result and Discussion

This section focuses on the system implementation and testing for the newly rebranded *GoBubble* application, which was previously referred to as *Candywash* during earlier development stages in collaboration with the stakeholder. The system now reflects an expanded scope and identity aligned with the stakeholder's vision.

It covers the implementation of key modules such as User Authentication, Service Management Module, Booking and Payments, Notifications, Feedback and Support, Analytics and Loyalty Module. The section also discusses the functional testing performed to evaluate the system's usability, reliability, and readiness for deployment.

5.1 Implementation

This section describes the development of functional modules in a system. Program code is provided to aid clarification.

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 1490-1509

5.1.1 User Authentication

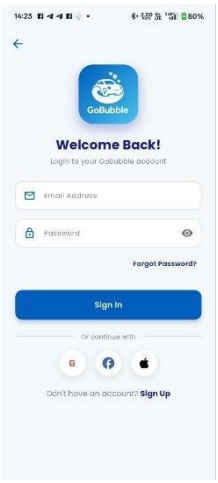


Fig. 11 *User Interface Sign In*

```dart
class Login extends StatefulWidget {
  const Login({super.key});

  @override
  State<Login> createState() => _LoginState();
}

class _LoginState extends State<Login> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  bool _isPasswordVisible = false;
  bool _isLoading = false;
  final _formKey = GlobalKey<FormState>();

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xFFF4F8FB),
      // Set this to false to allow the SingleChildScrollView to handle scrolling
      resizeToAvoidBottomInset: false,
      appBar: AppBar(
        backgroundColor: Colors.transparent,
        elevation: 0,
        leading: IconButton(
          icon: const Icon(Icons.arrow_back, color: Color(0xFF005FBE)),
          onPressed: () => Navigator.of(context).pop(),
        ), // IconButton
        toolbarHeight: 70, // Reduced height
        systemOverlayStyle: SystemUiOverlayStyle.dark,
```

Fig. 12 *Source Code Sign In*

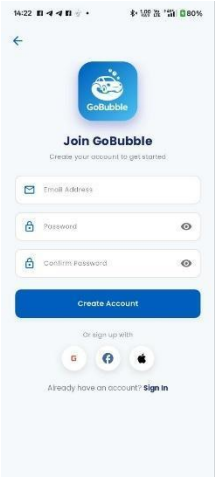


Fig. 13 *User Interface Sign Up*

```dart
class Signup extends StatefulWidget {
  const Signup({super.key});

  @override
  State<Signup> createState() => _SignUpState();
}

class _SignUpState extends State<Signup> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  final TextEditingController _confirmPasswordController = TextEditingController();
  bool _isPasswordVisible = false;
  bool _isConfirmPasswordVisible = false;
  final _formKey = GlobalKey<FormState>();
  bool _isLoading = false;

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    _confirmPasswordController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xFFF4F8FB),
      // Set this to false to allow the SingleChildScrollView to handle scrolling
      resizeToAvoidBottomInset: false,
      appBar: AppBar(
        backgroundColor: Colors.transparent,
        elevation: 0,
        leading: IconButton(
          icon: const Icon(Icons.arrow_back, color: Color(0xFF005FBE)),
          onPressed: () => Navigator.of(context).pop(),
```

Fig. 14 *Source Code Sign Up*

Figure 11 shows the user interface of the Sign In. Figure 13 shows the source code for Sign In where they have the verification of the user. The verification will use Firebase Authentication method to connect with the server side. The successful verification will redirect to the home page. The failed verification will return to login again with error pop up. Figure 12 shows the user interface of the Sign Up. Figure 14 shows the source code for Sign Up. The Sign Up starts with checking the email user if it already exists or not. If the user does not exist, then it will proceed to insert the data into Firestore. If the user exists, it will redirect the login page with a user exit message.

5.1.2 Service Management Module

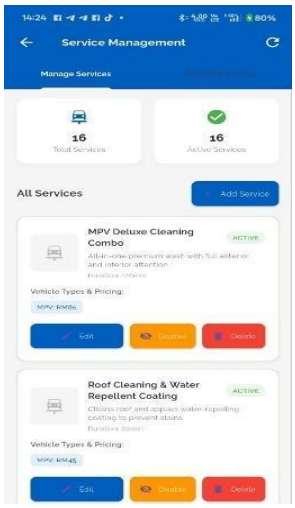


Fig. 15 *Service Management Module*

```
class ServicesScreen extends StatefulWidget {
  const ServicesScreen({Key? key}) : super(key: key);

  @override
  State<ServicesScreen> createState() => _ServicesScreenState();
}

class _ServicesScreenState extends State<ServicesScreen> {
  String selectedVehicle = 'Car';

  Widget getVehicleServicesWidget() {
    switch (selectedVehicle) {
      case 'MPV':
        return MPVServices();
      case 'Motor':
        return MotorServices();
      case 'Truck':
        return TruckServices();
      case 'Car':
      default:
        return SedanServices();
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xFFF5F5F5),
      appBar: AppBar(
        backgroundColor: const Color(0xFF005FBE),
        elevation: 0,
        title: Text(
```

Fig. 16 *Source Code Service Management*

Figure 15 shows the user interface for managing services, where the admin can add or update service details. Figure 16 presents the source code responsible for handling the creation and validation of service data. When the admin submits the form, the system checks for duplicates before saving the service information to Firestore. Successful operations update the service list dynamically, while errors display appropriate messages.

5.1.3 Booking and Payments

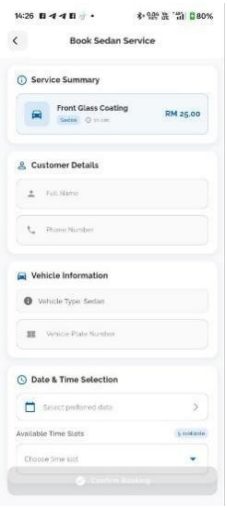


Fig. 17 *Booking Interface*

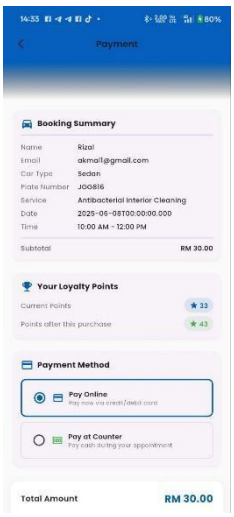


Fig. 18 *Payment Interface*

```dart
class BookingScreen extends StatefulWidget {
  final String service;
  final String carType;
  final double totalPrice;

  const BookingScreen({
    Key? key,
    required this.service,
    required this.carType,
    required this.totalPrice,
    required double price,
    required int quantity,
  }) : super(key: key);

  @override
  State<BookingScreen> createState() => _BookingScreenState();
}

class _BookingScreenState extends State<BookingScreen> {
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _phoneController = TextEditingController();
  DateTime? _selectedDate;
  String? _selectedTimeSlot;
  List<String> _timeSlots = [];
  bool _isLoading = false;

  @override
  void initState() {
    super.initState();
    _generateTimeSlotsBasedOnCarType();
  }
```

Fig. 19 *Source Code Booking*

```dart
class PaymentScreen extends StatefulWidget {
  final String bookingId;

  const PaymentScreen({super.key, required this.bookingId});

  @override
  State<PaymentScreen> createState() => _PaymentScreenState();
}

class _PaymentScreenState extends State<PaymentScreen> {
  Map<String, dynamic>? bookingData;
  bool isLoading = true;
  bool isPaying = false;
  double discount = 0;
  int userPoints = 0;

  // Add variables to store selected coupon info
  String? selectedCouponId;
  String? selectedCouponName;
  double selectedCouponReward = 0.0;

  // Add payment method selection
  String selectedPaymentMethod = 'online'; // Default to online payment

  List<Map<String, dynamic>> coupons = [];

  @override
  void initState() {
    super.initState();
    _loadData();
  }
```

Fig. 20 *Source Code Payment*

Figure 17 displays the booking interface, allowing users to select services, choose dates and time slots, and enter booking details. Figure 19 shows the source code that manages booking validation, storage, and status updates in the database. Figure 18 illustrates the payment interface, where users can select payment methods and review their final charges. Figure 20 presents the source code responsible for processing payments securely and updating booking payment statuses accordingly.

5.1.4 Notifications Module

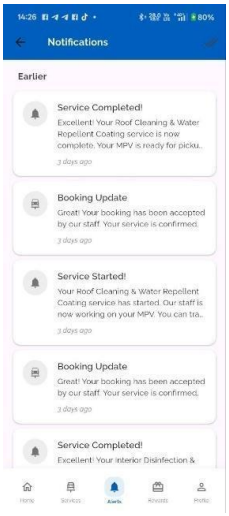


Fig. 21 *Notifications Module*

```dart
class NotificationsScreen extends StatefulWidget {
  const NotificationsScreen({Key? key}) : super(key: key);

  @override
  State<NotificationsScreen> createState() => _NotificationScreenState();
}

class _NotificationS  tends State<NotificationsScreen> {
  bool _isLoading = false;
  final currentUser = FirebaseAuth.instance.currentUser;

  @override
  Widget build(BuildContext context) {
    final userId = currentUser?.uid ?? '';

    return Scaffold(
      appBar: AppBar(
        title: Text(
          'Notifications',
          style: GoogleFonts.raleway(
            fontWeight: FontWeight.bold,
            fontSize: 18,
            color: const Color.fromARGB(255, 255, 255, 255),
          ),
        ), // Text
        backgroundColor: const Color(0xFF005FBE),
        actions: [
          // Mark all as read button
```

Type: bool

Fig. 22 *Source Code Notifications Module*

Figure 21 shows the notifications interface, which displays real-time alerts to users regarding booking status updates, promotional offers, and important announcements. Figure 22 presents the source code that handles the retrieval, display, and management of notifications, ensuring timely communication between the system and users.

5.1.5 Feedback and Support

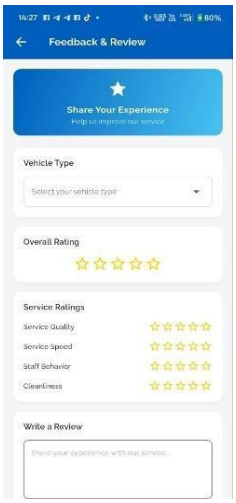


Fig. 23 *Feedback Interface*

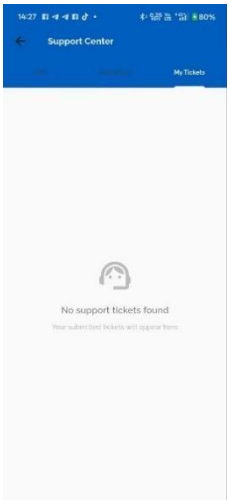


Fig. 24 *Support Interface*

```dart
class FeedbackPage extends StatefulWidget {
  const FeedbackPage({super.key});

  @override
  State<FeedbackPage> createState() => _FeedbackPageState();
}

class _FeedbackPageState extends State<FeedbackPage> {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  final FirebaseAuth _auth = FirebaseAuth.instance;

  // Form controllers
  final TextEditingController _reviewController = TextEditingController();
  final TextEditingController _suggestionController = TextEditingController();

  // Rating values
  double _overallRating = 0.0;
  double _qualityRating = 0.0;
  double _speedRating = 0.0;
  double _staffRating = 0.0;
  double _cleanlinessRating = 0.0;

  // Vehicle type selection
  String? _selectedVehicleType;
  final List<String> _vehicleTypes = ['Sedan', 'MPV', 'Motor', 'Truck'];

  bool _isSubmitting = false;

  @override
  void dispose() {
    _reviewController.dispose();
    _suggestionController.dispose();
    super.dispose();
  }
```

Fig. 25 *Source Code Feedback*

```dart
class SupportScreen extends StatefulWidget {
  const SupportScreen({super.key});

  @override
  State<SupportScreen> createState() => _SupportScreenState();
}

class _SupportScreenState extends State<SupportScreen> with SingleTickerProviderStateMixin {
  late TabController _tabController;
  final _messageController = TextEditingController();
  final _subjectController = TextEditingController();
  bool _isSubmitting = false;

  @override
  void initState() {
    super.initState();
    _tabController = TabController(length: 3, vsync: this);
  }

  @override
  void dispose() {
    _tabController.dispose();
    _messageController.dispose();
    _subjectController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[50],
      appBar: AppBar(
```

Fig. 26 *Source Code Support*

Figure 23 illustrates the feedback interface, allowing users to submit their ratings and comments regarding the services received. Figure 25 shows the source code responsible for capturing, validating, and storing user feedback in the database. Figure 24 displays the support interface, where users can raise support tickets for any issues or inquiries. Figure 26 presents the source code that manages ticket creation, tracking, and response handling to ensure efficient customer support.

5.1.6 Analytics Report

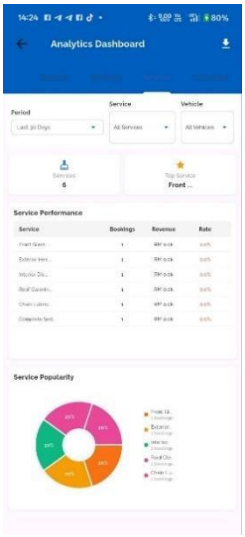


Fig. 27 Analytics Module

```
class AnalyticsDashboard extends StatefulWidget {
  const AnalyticsDashboard({Key? key}) : super(key: key);

  @override
  _AnalyticsDashboardState createState() => _AnalyticsDashboardState();
}

class _AnalyticsDashboardState extends State<AnalyticsDashboard> with SingleTickerProviderStateMixin {
  // Tab controller for different analytics sections
  late TabController _tabController;

  // Date range selection
  DateTime _startDate = DateTime.now().subtract(const Duration(days: 30));
  DateTime _endDate = DateTime.now();
  String _selectedPeriod = 'Last 30 Days';

  // Filter selections
  String _selectedServiceFilter = 'All Services';
  String _selectedVehicleFilter = 'All Vehicles';
  String _selectedStafffilter = 'All Staff';

  // Data containers
  Map<String, dynamic> _revenueData = {};
  Map<String, dynamic> _bookingsData = {};
  Map<String, dynamic> _serviceData = {};
  Map<String, dynamic> _customerData = {};
  Map<String, dynamic> _staffData = {};

  // Loading states
  bool _isLoadingRevenue = true;
  bool _isLoadingBookings = true;
  bool _isLoadingServices = true;
  bool _isLoadingCustomers = true;
```

Fig. 28 Source Code Analytics Module

Figure 27 shows the analytics interface, where the admin can select report types, set time periods, and view generated visualizations for business insights. Figure 28 presents the source code that handles data processing, report generation, and rendering of charts and graphs to support informed decision-making.

5.1.7 Loyalty Module

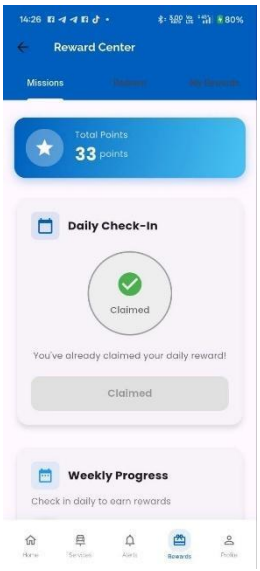


Fig. 29 Loyalty Module

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

  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
      length: 3,
      child: Scaffold(
        appBar: AppBar(
          title:Text(
          'Reward Center',
          style: GoogleFonts.raleway(
            fontWeight: FontWeight.bold,
            fontSize: 18,
            color: const Color.fromARGB(255, 255, 255, 255),
          ),
        ), // Text
          backgroundColor: const Color(0xFF005FBE),
          bottom: const TabBar(
            labelColor: Colors.white,
            indicatorColor: Colors.white,
            tabs: [
              Tab(text: "Missions"),
              Tab(text: "Redeem"),
              Tab(text: "My Rewards"),
            ],
          ), // TabBar
        ), // AppBar
        body: const TabBarView(
          children: [
            MissionPage(),          // missions.dart
```

Fig. 30 Source Code Loyalty Module

Figure 29 displays the loyalty module interface, allowing customers to view their points balance, browse available rewards, and redeem points. The interface also provides admins with options to configure point rules and manage rewards. Figure 30 shows the source code responsible for tracking points accumulation, handling redemptions, and updating customer loyalty data in the backend.

5.1 Testing Result

Functional testing was conducted to ensure each module of the system performs according to the specified requirements. Table 4 shows that includes verifying user registration, service management, booking flows, and admin operations. Each test case was evaluated based on expected outputs to confirm system reliability and correctness.

Table 4: Testing Results

Module	Description	Expected Result	Actual Results	Pass/Fail
User Authentication	To check whether customer, staff and admin can register an account	customer, staff and admin able create account	customer, staff and admin successfully create account	Pass
	To check whether customer, staff and admin can log in to system	customer, staff and admin able log in to system	customer, staff and admin successfully log in	
Service Management Module	Admins manage services; customers can view	Admin can add, update, delete services; customers can view available services	Admin CRUD operations successful; services displayed properly to customers	Pass
Booking and Payments	Booking with time slot and payment features	Customers can select services, choose time slots, and make payments securely	Booking system functions correctly; payment gateway integration works as expected	Pass
Notifications	Sends updates to users, staff and admins	Customers receive service updates, reminders, and promotions; admins and staff receive booking alerts	All notifications are delivered promptly via push and in-app alerts	Pass
Feedback and Support	Collects customer feedback and queries	Customers can submit feedback and issues; admins can read and respond in the dashboard	Feedback is submitted successfully; admin responses visible to users	Pass
Analytics Module	Provides performance insights	Admins can view user statistics, booking trends, and revenue reports in visual/chart format	Analytics dashboard displays accurate and up-to-date information	Pass
Loyalty Module	Rewards users with points and discounts	Customers earn and redeem points; admin can configure rewards and thresholds	Points are updated correctly; redemption and admin configuration are functioning	Pass

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 1490-1509

5.2 User Acceptance Testing

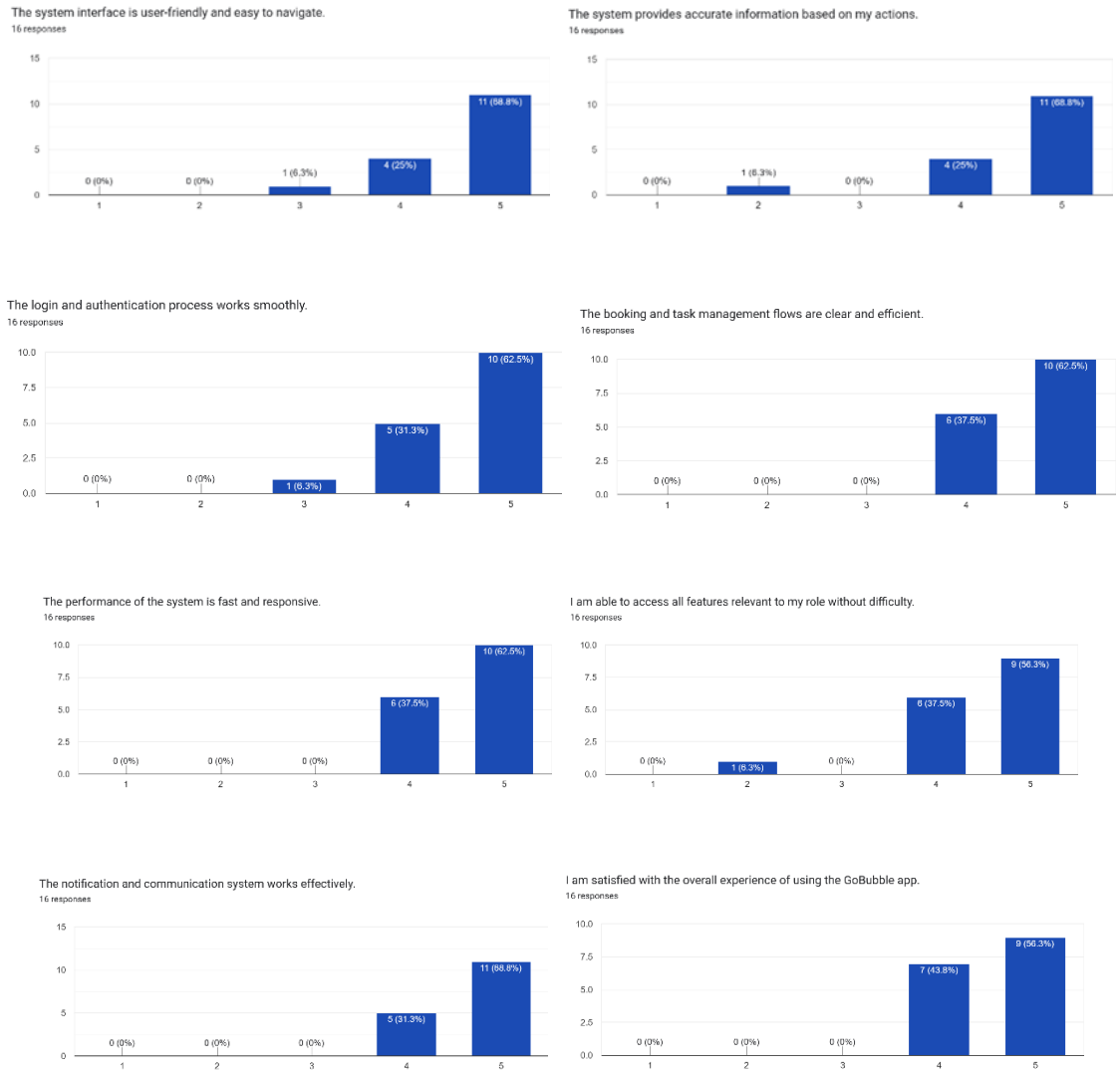


Fig. 31 *User Acceptance Testing*

Figure 31 present the User Acceptance Testing (UAT) result conducted for the GoBubble: Carwash Services Management Application. A total of 16 participants responded to the evaluation, and the feedback was strongly positive across all key areas. The system interface was rated as user-friendly and easy to navigate, with 68.8% of respondents rating it a 5, 25% rating it a 4 and 6.3% rating it a 3. The login and authentication process also received favourable ratings, with 62.5% selecting 5, 31.3% selecting 4 and 6.3% selecting 3, indicating a smooth and reliable sign-in experience. Additionally, users reported being able to access all features relevant to their role without difficulty, as shown by 6.3% giving rating 2, 37.5% giving a rating of 4 and 56.6% giving a 5.

Regarding system performance, 62.5% rated it as fast and responsive with a 5, while the remaining 37.5% gave it a 4. The booking and task management flows were considered precise and efficient, with 62.5% of users 5 and 37.5% giving a 4. Next, the notification and communication system was rated effective by 68.8% (rating 5) and 31.3% (rating 4), showing strong functionality in keeping users informed. When asked if the system provided accurate information based on user actions, 68.8% responded with a rating of 5, 25% with a 4 and 6.3% with a 2, highlighting the system's reliability in processing and reflecting user input. Lastly, overall satisfaction with the GoBubble app was high, with 56.3% rating their experience a 5 and 43.8% a 4.

Lastly, indicating a strong level of acceptance and satisfaction. Based on these positive outcomes, it is recommended to maintain the current system functionality, ensure continued performance reliability, and make iterative improvements where necessary to support evolving user expectations.

6. Conclusion and Future Work

In conclusion, the mobile-based booking system GoBubble: Carwash Services Management Application was developed to address common operational challenges such as overlapping service bookings, lack of centralized data, miscommunication between users and administrators, and non-streamlined workflows. The application provides a user-friendly platform that simplifies service access and enhances customer engagement. It supports secure authentication through sign-in and sign-up functionalities and includes features for managing user profiles, service listings, bookings, payments, notifications, feedback, analytics, and loyalty rewards. The implementation of GoBubble has improved operational efficiency by enabling robust data management within a secure Firebase backend. The system effectively handles real-time booking processes, service selections, and payment workflows while informing users through timely notifications. This comprehensive approach contributes to a better user experience, supports staff coordination, and streamlines administrative oversight, ultimately enhancing the overall management and sustainability of the carwash service operation. For future enhancements, GoBubble could be expanded to include real-time booking slot conflict resolution based on staff availability, more advanced analytics dashboards for business insights, and customizable promotional campaigns for customer retention. Further personalization options and intelligent recommendations based on booking history could also improve user engagement. Continuous improvements will ensure that GoBubble remains a reliable, modern, and efficient solution for carwash service management.

Acknowledgements

The author would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia for its support.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design****: Mohd Nizwan Mohd Asri, Rozita Abdul Jalil;* ***data collection****: Mohd Nizwan Mohd Asri;* ***analysis and interpretation of results****: Mohd Nizwan Mohd Asri, Rozita Abdul Jalil;* ***draft manuscript preparation****: Mohd Nizwan Mohd Asri. All authors reviewed the results and approved the final version of the manuscript.*

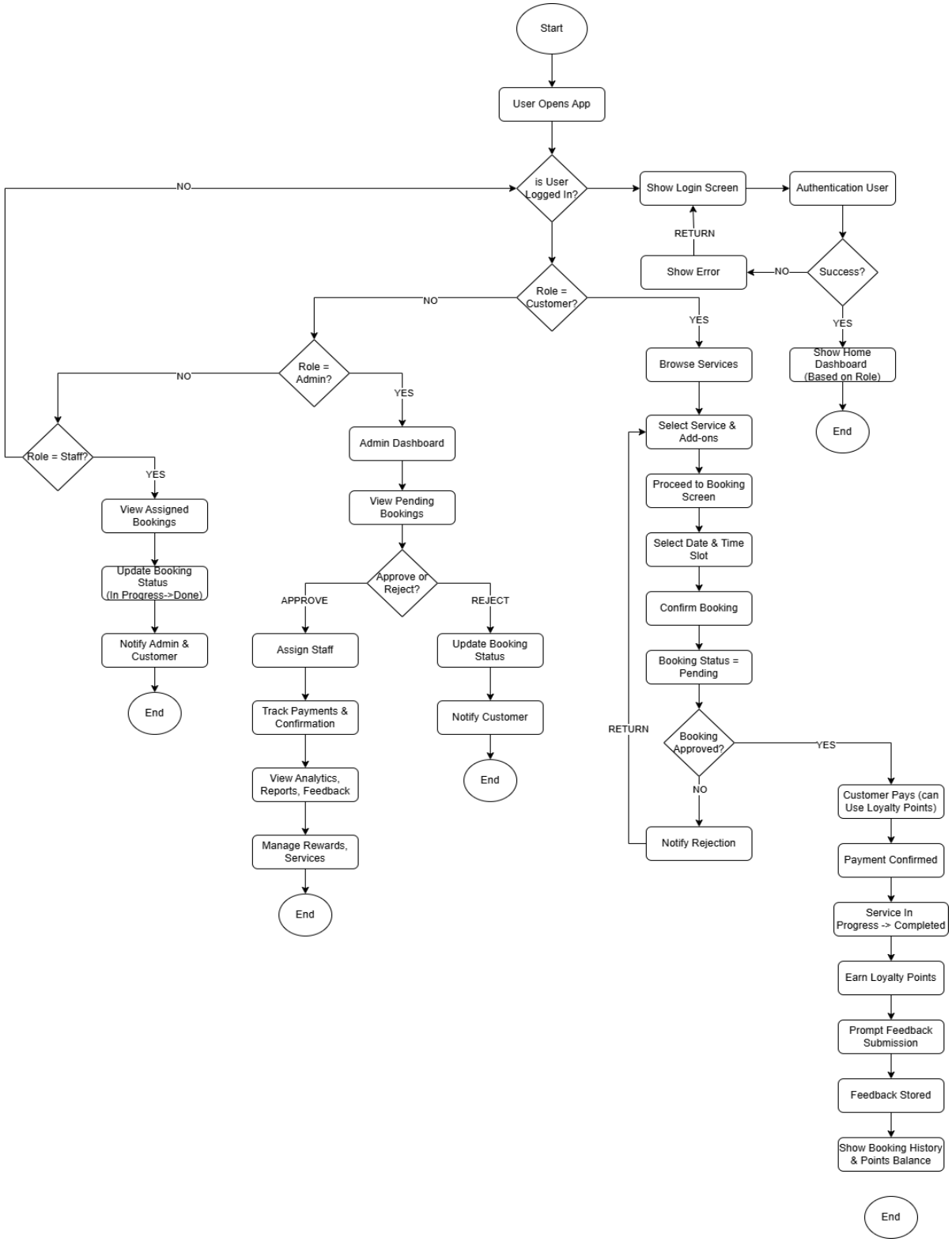
References

[1] Johnson, M., & Smith, K., "Digital Transformation in Service Industries: A Systematic Review," *International Journal of Service Management*, vol. 12, no. 4, pp. 78-95, 2023.
[2] Kumar, R., & Patel, D., "Mobile Application Development Frameworks: A Comparative Analysis," *Journal of Software Development*, vol. 8, no. 3, pp. 112-128, 2023. DOI: 10.1109/MSD.2023.1234567
[3] Williams, A., "User Interface Design Principles for Service Applications," *Human-Computer Interaction Journal*, vol. 18, no. 1, pp. 23-40, 2024.
[4] Rasheed, A., et al., "Requirement engineering challenges in agile software development," *Mathematical Problems in Engineering*, vol. 2021, no. 1, pp. 6696695, 2021. [Online]. Available: https://doi.org/10.1155/2021/6696695
[5] Arifin, M. N., & Siahaan, D., "Structural and semantic similarity measurement of UML use case diagram," *Lontar Komputer: Jurnal Ilmiah Teknologi Informasi*, vol. 11, no. 2, pp. 88, 2020.
[6] Bashir, N., et al., "Modeling class diagram using nlp in object-oriented designing," in *2021 National Computing Colleges Conference (NCCC)*, IEEE, 2021, pp. 1-6.
[7] Ghandour, A., "Ecommerce application value model for SMEs," *International Journal of E-Business Research*, vol. 11, no. 1, pp. 40-55, 2015.

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 1490-1509

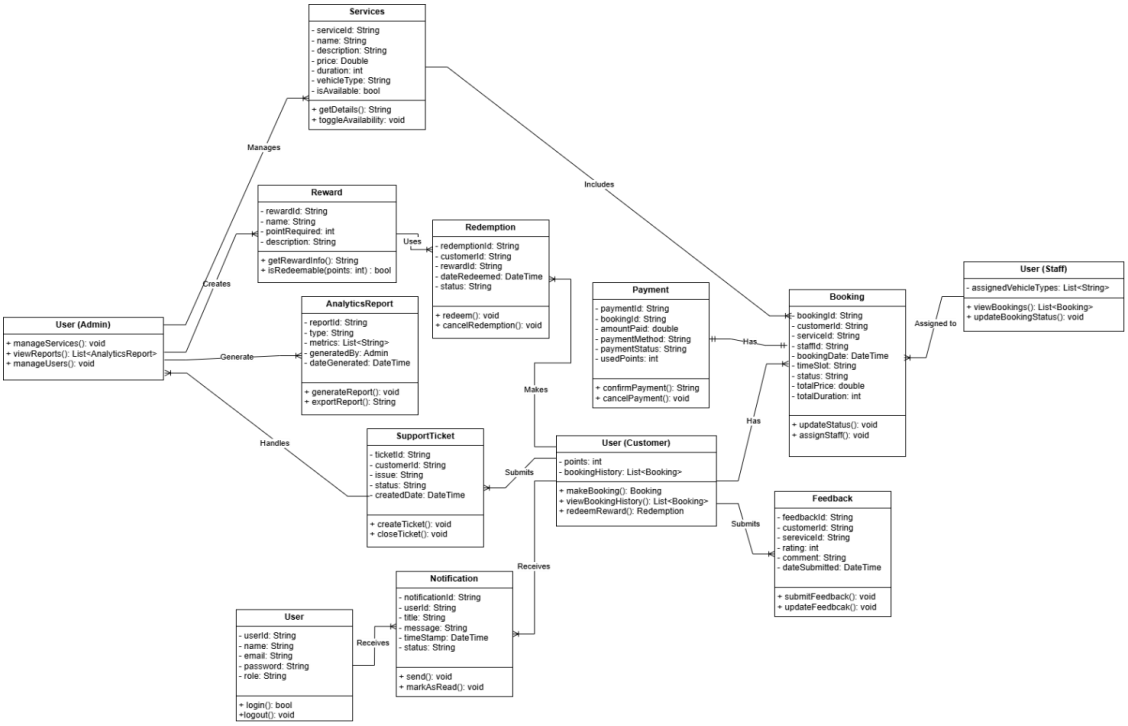
[8] Ullah, S. E., Alauddin, T., & Zaman, H. U., "Developing an E-commerce application," in *2016 International Conference on Microelectronics, Computing and Communications (MicroCom)*, IEEE, 2016, pp. 1-4. DOI: 10.1109/MicroCom.2016.7522426
[9] Windmill, E., *Flutter in Action*, Simon and Schuster, 2020. [Online]. Available: https://www.manning.com/books/flutter-in-action

Appendix A: Flow Chart



Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 1490-1509

Appendix B: Class Diagram



Appendix C: Example Activity Diagram

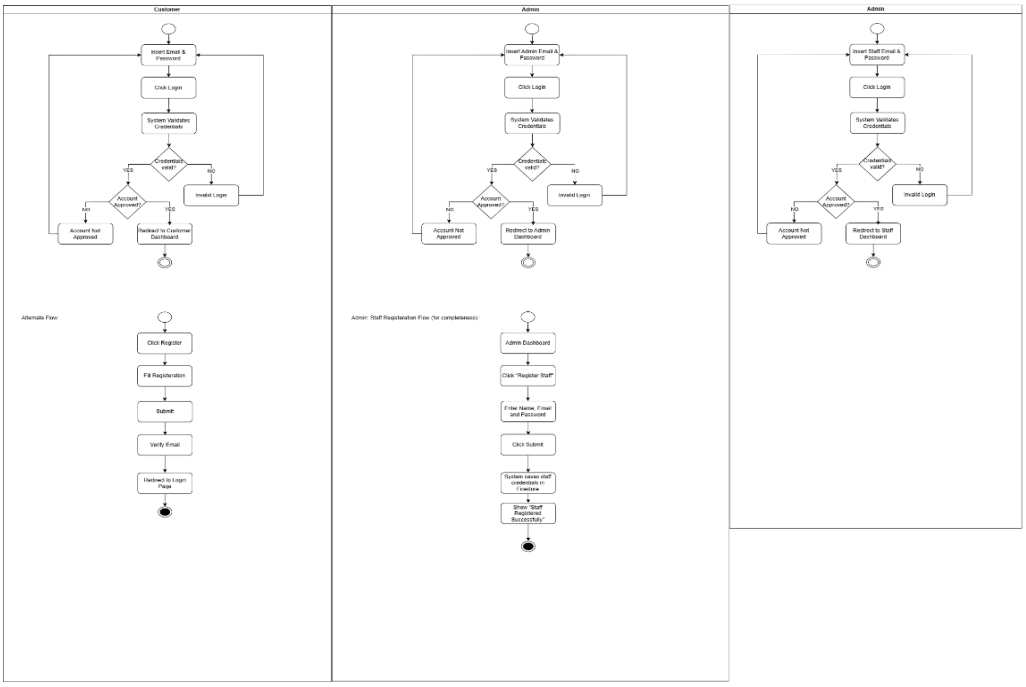


Fig. 32: Activity Diagram User Authentication

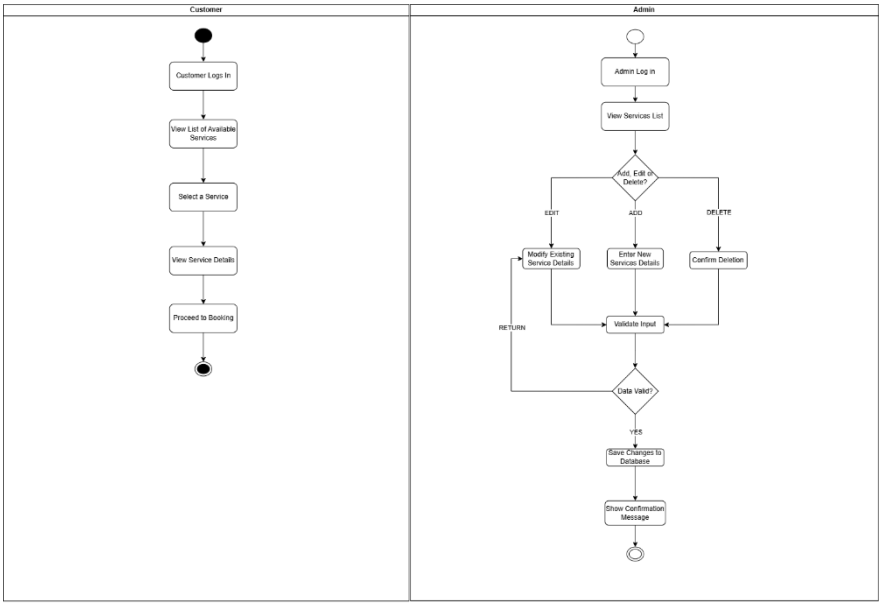


Fig. 33: Activity Diagram Service Management

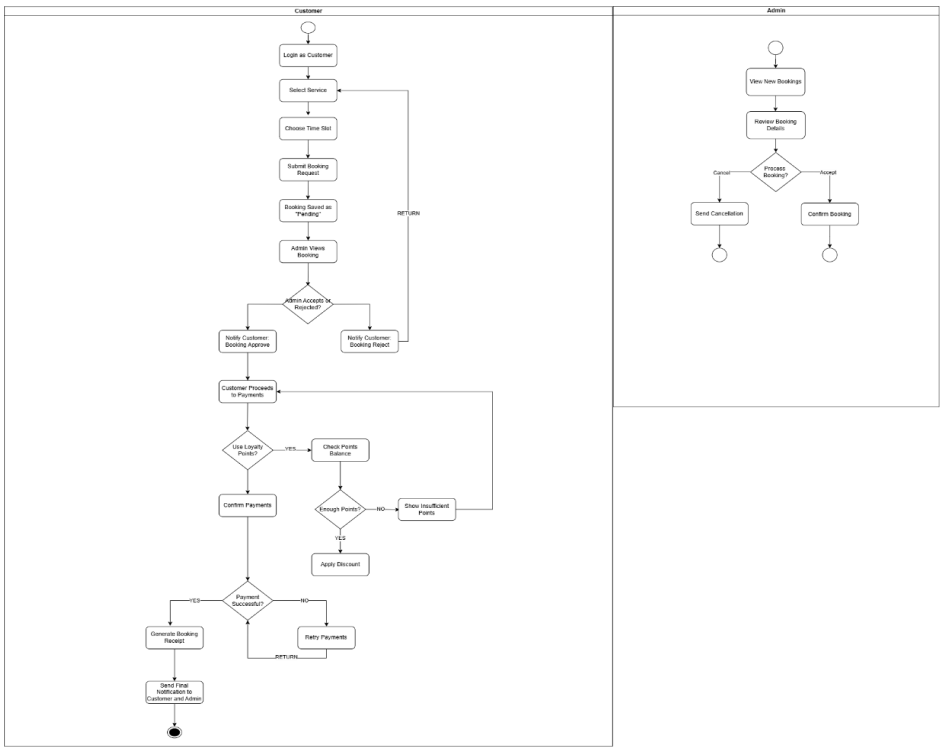


Fig. 34: Activity Diagram Booking and Payment

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 1490-1509

Appendix D: Sequence Diagram

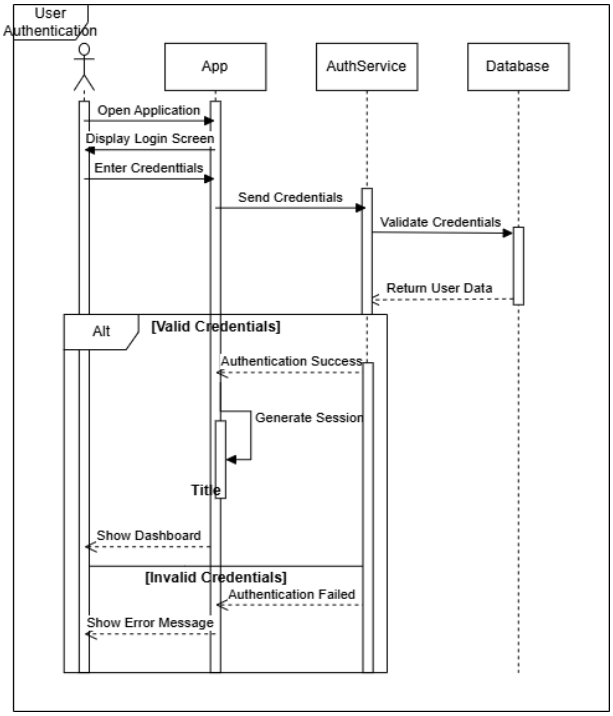


Fig. 35:: Sequence Diagram User Authentication

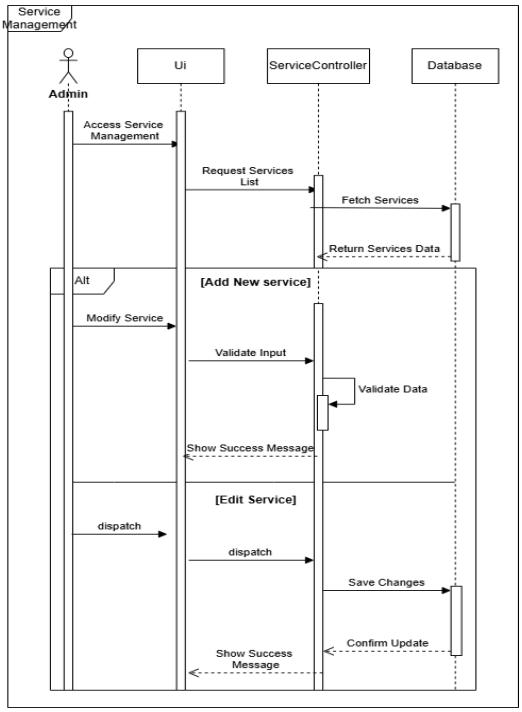


Fig. 36: Sequence Diagram Service Management

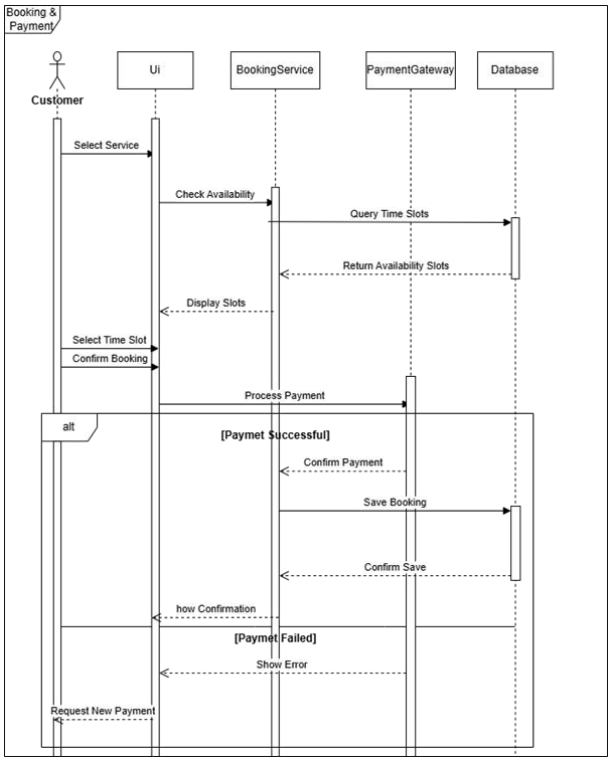


Fig. 39: Sequence Diagram Booking and Payment

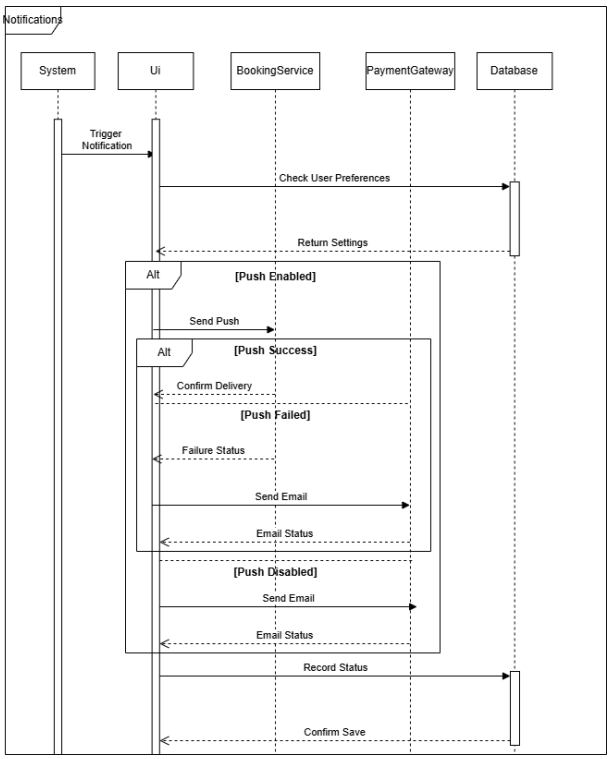


Fig. 40: Sequence Diagram Notification