American International University-Bangladesh (AIUB)

Department of Computer Science

Faculty of Science & Technology (FST)

Summer 22-23

Section: C

Software Quality Assurance and Testing

Easy Health App

A Report submitted by **Team Happify**

SN	Student Name	Student ID
1	RUBYEA ZANNAT	20-44229-3
2	MD TANZIM BIN IDRIS	20-44252-3
3	KHALED HOSSAIN	20-44256-3
4	SAMIHA LABIBA IBNAT	20-44338-3

Under the supervision of

MD. MOMAN UL HAQUE KHAN

Lecturer, Faculty of Computer Science and Technology, American International University-Bangladesh

Checked By Industry Personnel

Name:

Designation:

Company:

Sign:

Date:

Software Test Plan

for

<Easy Health App>

Version 1.0 approved

Prepared by

Rubyea Zannat

Md Tanzim Bin Idris

Khaled Hossain

Samiha Labiba Ibnat

American International University-Bangladesh

Date: 23-8-2023

Table of Contents

Revision History

Revision	Date	Updated by	Update Comments
0.1	2023.07.29	Khaled Hossain	First Draft
0.2	2023.08.02	Md Tanzim Bin Idris	Second Draft
0.3	2023.08.07	Md Tanzim Bin Idris	Third Draft
0.4	2023.08.12	Rubyea Zannat	Fourth Draft
0.5	2023.08.14	Rubyea Zannat	Fifth Draft
0.6	2023.08.15	Samiha Labiba Ibnat	Sixth Draft
0.7	2023.08.18	Samiha Labiba Ibnat	Seventh Draft
0.8	2023.08.23	Khaled Hossain	Eighth Draft
0.9	2023.08.25	Khaled Hossain	Final Draft

1. TEST PLAN IDENTIFIER: EH-EasyHealthApp1.3

2. REFERENCES

[1] Software Quality and Testing Lecture Slides
[2] Software Quality Engineering: Testing, Quality Assurance and Quantifiable Improvement – Jeff Tian.

3. INTRODUCTION

Background to the Problem

The delivery of public services, including healthcare, must be done so safely and effectively in the modern world. The ambulance service industry in Bangladesh still needs to develop in these contemporary times, as technology is used primarily in many facets of life. In Bangladesh, it can be challenging for people to contact an ambulance. Although there are more ambulance services in cities than in rural regions, both urban and rural residents struggle with a lack of emergency ambulance services. In Bangladesh, there are two types of ambulance services: those provided directly from the hospital and those provided by private companies. Due to high demand, hospital-based services are frequently unavailable, and they are more expensive than private services. Private services can be accessed quickly, but they quite often lack proper medical personnel. Ambulances are booked through mobile phone calls in this country, and often the contact numbers are unreachable or engaged, wasting valuable time, and endangering the patient's life. People nowadays can contact ambulance services via 999, but because it requires several call transfers before booking an ambulance, this service isn't very reliable in an urgent situation. As a result, I believe Bangladesh must expand this sector to provide people with easy access to emergency ambulance services. Although there is a mobile application in Bangladesh that provides ambulance service, it has limited features and is only available in one area. We plan to implement a mobile-based application that will provide emergency ambulance services in Bangladesh. We can use it to improve emergency ambulance service, allowing people to access emergency ambulance services more conveniently.

Solution to the Problem

In Bangladesh, the emergency medical services frequently lack qualified medical personnel; thus, they provide only basic first aid and transportation assistance. Other countries provide competent emergency medical workers who can provide the emergency department doctor with a general overview of the patient's condition. People who work in emergency medicine should take a legitimate medical course where they learn how to stabilize patients' conditions and assess emergency patients. Our application will help people to access ambulance services easily without facing any hassles. The main goals of our application are:

- To help people access emergency ambulance services more conveniently.
- To improve healthcare system and help monetize medical services.

- To help emergency cases/accidents/natural disasters.

Our application is called "Easy Health". Two different ambulance call types will be available on this app, which will function similar to Uber. First is a significant condition (heart disease, diabetes, respiratory problems, trauma attacks, etc.) that requires an emergency call, while second is less critical (patient transportation from hospital to home, transferring a disabled patient to hospital for checkup etc.) ambulance call. The application will have an additional interface for diseases, where consumers will see disease symptoms, prevention, and treatments. This application will also offer a first aid course for people free of cost. We plan to implement it to lessen the crowd and pressure in hospitals because often ambulances are misused due to the lack of first aid education.

4. REQUIREMENT SPECIFICATION

4.1 System Features

Functional Requirements:

1) System Registration

i) The software must allow users to register with the necessary information.

ii) If the username is not unique, the system will prompt the user to try registration with a different username again.

Priority Level: High

Precondition: Not applicable.

2) System Login

i) Users must be able to log in using their assigned username and password.

ii) If the username and/or password have been entered incorrectly more than three times, the random verification code will be generated by the system to retry login.

Priority Level: High

Precondition: The user must have a valid user ID and password.

3) Request for ambulance

i) Users will be able to request an ambulance with a location and description using the software.

ii) If no location is specified, the system should reject the request and instruct the user to try again with a more precise location.

iii) Adding the condition of the patient (Critical/ Less critical)

Priority Level: High Precondition: User must have valid account.

4) Calling/Messaging for approximate time

i) The software allows the user to call and send messages to the ambulance driver for details about the approximate arrival time of the ambulance.

Priority Level: High

Precondition: User must have valid account

5) Request for payment

i) User can payment their charges through the software. Three ways to get payment 1. Bkash/ Nagad 2. Banking 3. Cash on service.

Priority Level: Medium

Precondition: User must have valid account

6) Applying for First-aid course

i) User can apply for the first-aid course through the software. They must register to take the course.

Priority level : Medium

Precondition: User must have valid account

7) User Account

i) Users can see their rating points.

ii) System shall allow the user to track their booked ambulance.

iii) System also allows the user to add or remove a coupon for a discount.

iv) The system shall allow the user to update their payment options.

v) System shall allow the user to access available account security options.

Priority Level: Medium.

Precondition: The user must be logged in to the system.

8) Settings & Privacy

i) The system shall allow the user to update their location settings.

ii) The system also allows the user to choose their preferred languages.

iii) System shall allow the user to delete or deactivate their profile/account.

Priority Level: High.

Precondition: The user must be logged in to the system.

9) Account Security

i) The system shall allow the user to change or update their password.

Priority Level: High

Precondition: The user must be logged in to the system

4.2 System Quality Attributes

Non-Functional Requirements:

1) **Usability:** The system must be user-friendly. The system should be intuitive and simple to navigate.
2) **Efficiency:** The system should maximize the capacity and memory of the processor. Any task should be completed with optimal efficiency.
3) **Security:** System security should be sufficient to prevent unauthorized access to system functions, to prevent information loss, protect data privacy, and safeguard the system against viruses.
4) **Modularity:** The system's every block of code must be under separate and acceptable modules.
5) **Testability:** The system should be simple to test and identify flaws.
6) **Flexibility:** The system should be flexible enough to be modified.

7) **Reusability:** Code library classes should be general enough to be utilized on multiple versions of an application or new projects.

4.3 System Interface

4.3.1 Use-Case Diagram:

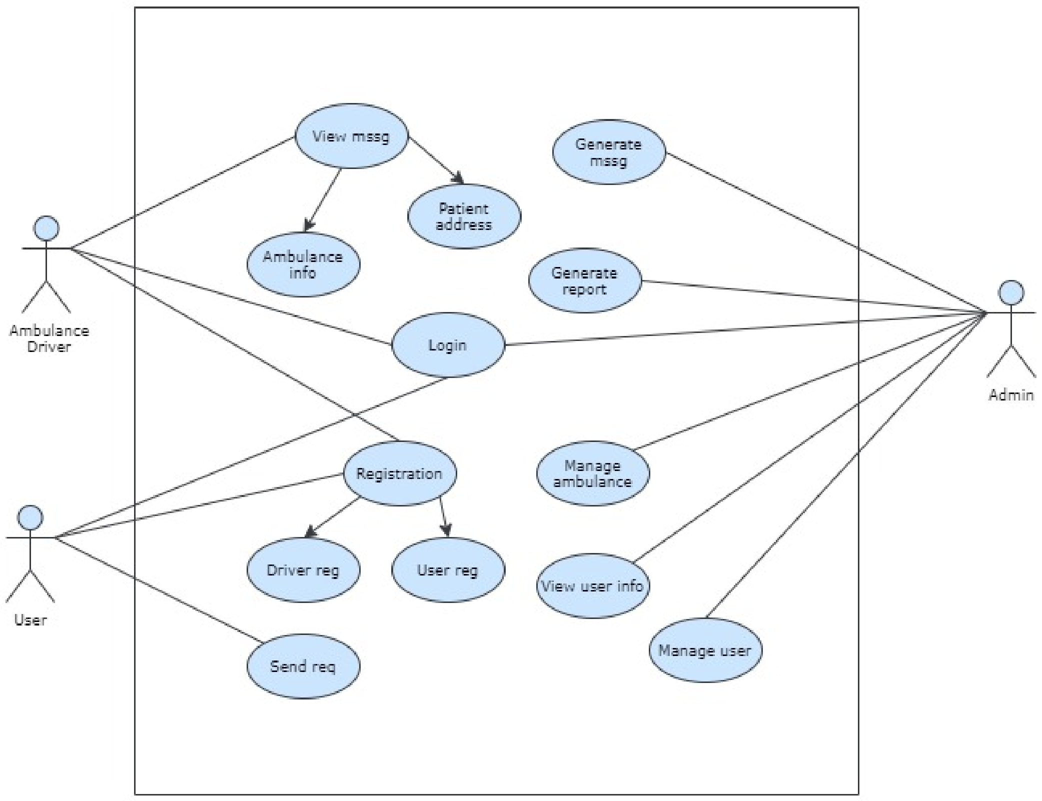


Fig. 1.1: Use-Case Diagram

User and Ambulance driver is a primary actor who interacts with various functionalities of the Ambulance Service System such as register, log in, request an ambulance, communicate with ambulance drivers, apply for a first-aid course. Users can register in the system by providing their necessary information. Ambulance drivers can register in the system by submitting their details and qualifications. Registered users and drivers can log into the system. Users can send ambulance service requests when they require urgent medical assistance. The request includes the user's location and relevant information. Ambulance drivers can view messages containing important information, including details about the assigned ambulance and the patient's address. The Ambulance Service System involves an Admin who plays a secondary role. The admin’s functionalities include logging in, managing ambulances, viewing user information, generating reports, and managing users. Admins can log into the system using their credentials to access administrative functionalities. Admin can manage ambulance-related information, including adding new ambulance details, updating existing records, and removing ambulances from the system. Admin can view user information stored in the system. This allows the admin to monitor user activity and respond to user-related queries. Admin can generate various types of reports based on system data, such as the number of ambulance requests, response times, user statistics, etc. Admin can manage user accounts, including approving new user registrations, updating user profiles, and deactivating user accounts if necessary.

4.3.2 Class Diagram:

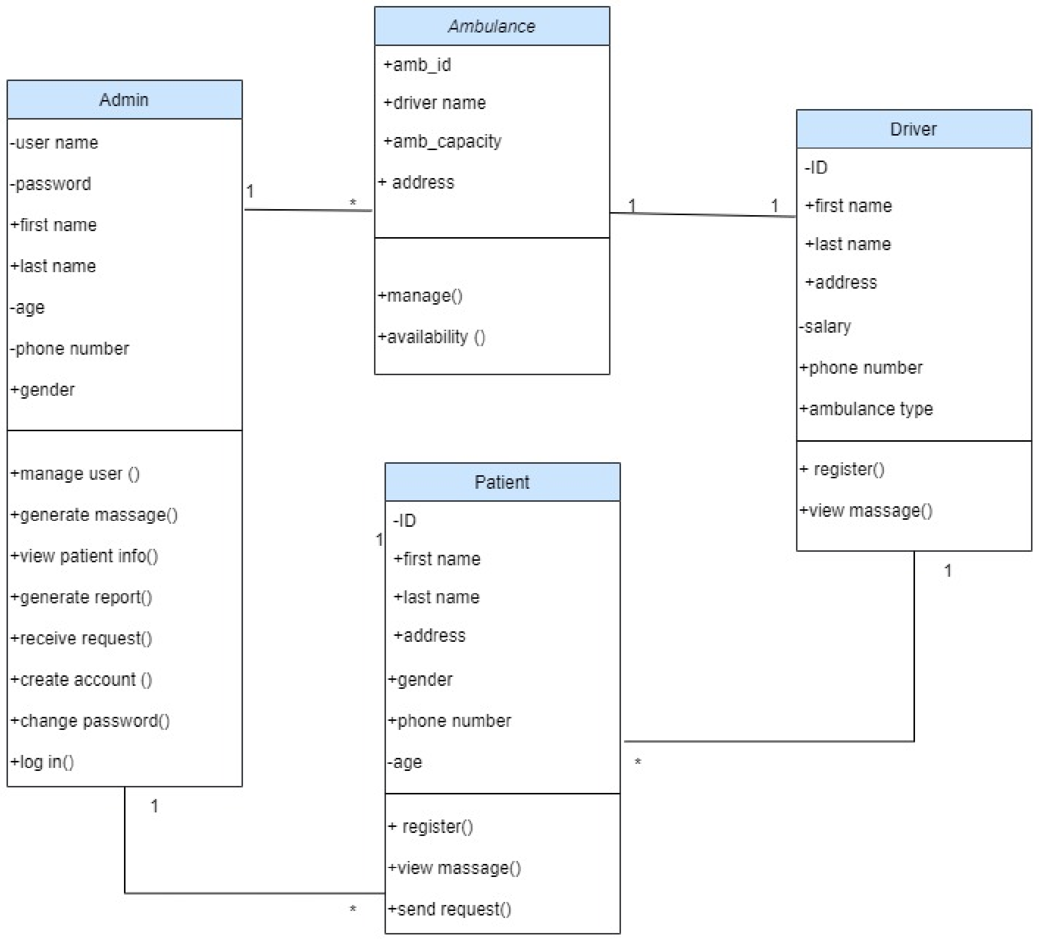


Fig 1.2: Class Diagram

- Admin class: The Admin class represents a manager within the ambulance service system. Admins have the authority to oversee various aspects of the system, manage users, generate reports, and more. Attributes are Username (private), Password (private), First name (public), Last name (public), Age (public), Phone number (private), Gender (public). Methods are manageUser(), generateMessage(), viewPatientInfo(), generateReport(), receiveRequest(), createAccount(), changePassword(), login().

- Ambulance class: The Ambulance class represents an ambulance within the ambulance service system. Ambulances are tracked for their locations and statuses. Attributes are Ambulance id(public), driver name(public), ambulance capacity(public), address(public). Methods are Manage(), Availability().
- Driver class: The Driver class represents a driver associated with an ambulance. Drivers operate ambulances and respond to emergency requests. Attributes are id(private), First name (public), Last name (public), Address(public), Salary(private), Phone number(public), ambulance type(public). Methods are register(), view message()
- Patient: The Patient class represents a patient who can request ambulance services. Attributes are id(private), First name (public), Last name (public), Address(public), Gender(public), Phone number(public), age(public). Methods are register(), view message(), send request().

Relationships:

- Admin to Ambulance: One Admin manages multiple Ambulances. The admin class can interact with the Ambulance class to manage ambulance-related activities, such as updating their locations and statuses.
- Ambulance to Driver: One Driver is associated with one Ambulance. Each driver operates a single ambulance. This is a "one-to-one" relationship from Driver to Ambulance.
- Driver to Patient: One Driver can be associated with many Patients. A driver can respond to multiple patient requests for assistance. This is a "one-to-many" relationship from Driver to Patient.
- Admin to patient: One Admin manages multiple Patients. Admins have the capability to manage multiple patients, but each patient is associated with only one Admin.

4.3.3 Activity Diagram:

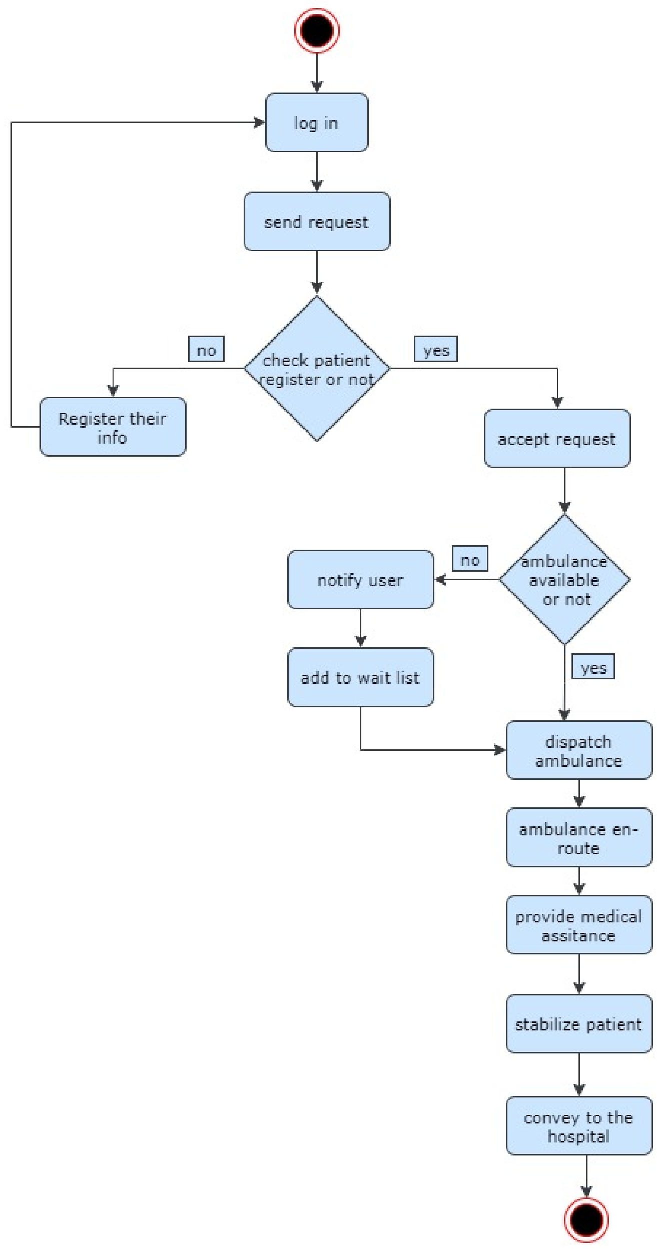


Fig. 1.3: Activity Diagram

The diagram begins with the initiation of the software. The system verifies the login information and grants access to authorized users. Authorized users access the "Send Request" functionality. They input emergency details, including the nature of the emergency and the location. The system checks whether the patient's information is registered in the system. If the patient is not registered: The system guides the user back to the login page, indicating that the patient needs to be registered before requesting assistance. If the patient is registered-The system proceeds to check ambulance availability. The system checks the availability of ambulances to respond to emergency requests. If an ambulance is available: The process proceeds to dispatch the ambulance. The software assigns the nearest available ambulance to the emergency If no ambulance is available: The system notifies the caller about the unavailability of ambulances. The caller might be informed about the waitlist option. The process advances to adding the request to the waitlist. The system monitors the ambulance's real-time location using GPS. It provides estimated time of arrival updates to both the dispatcher and the caller. The software records the medical procedures performed on the scene. It might involve logging vital signs, administered treatments, and patient information. If needed, the software assists the medical team in recording stabilization efforts. This could involve decisions about further treatment or transport. The system records the ambulance's arrival at the hospital and alerts the hospital staff.

4.4 Project Requirements

Project Constraints:

Resource Constraints:

- Human Resources: Developers, QA and Testers, Project Manager and UI/UX Designer.
- Technical Resources: QA Tools, Development Tools Licenses and Cloud Servers.

Environment Constraints:

- Development Environment: Agile methodology with 2-week sprints.
- Deployment Environment: Cloud based hosting with end-to-end encryption for data security.

Budget Estimation:

Constructive Cost Model (COCOMO)

Software project type: Organic; [p=1.05]
Coefficient<Effort Factor> = 2.4

Effort = PM
So, P = 1.05 and T = 0.38
SLOC = 25000 Lines

Persons-months, PM = Coefficient<Effort Factor> * (SLOC / 1000) P = 2.4 * (15000/1000)^1.05
= 41.22

Development time, DM = 2.50 * (PM) ^T
= 2.50 * (41.22) ^0.38
= 10.27 = 11 months
= 1760 Working hours in total (Per week 40 hours)

Required number of people, ST = PM/DM
= 41.22/11
= 3.74 = 4 persons (developers)

Budgeting

Developer/Tester salary of 11 months:
Per employee salary per month = 40000 Taka = 400 Taka per hour
Total salary = 400 * 1760 = **7,04,000 Taka**

Requirement analysis:
Required time = 1 month = 25 working days = 200 working hour
Requirement analysis persons per hour salary = 250 Taka
Total requirement analysis salary = 250 * 200 = **50,000 Taka**
Transportation cost: **15,000 Taka** (Approximate)
Hardware expense: **1,20,000 Taka** (Approximate)
Rent expenses:
Total in 11 months **= 1,65,000 Taka** [Per month = 15,000 Taka]
Total utilities in 11 months: **15,000 Taka** (Approximate)
Maintenance (Till 4 months after delivery):
Cost per hour = 1,200 Taka
Total estimated time needed for maintenance = 40 hours
Total estimated maintenance cost = 1,200 * 40 = 48,000 Taka

Project manager's salary of 11 months:
Per month salary = 40,000 Taka
Total salary = 40,000 * 11 = 4,40,000 Taka
Accountant's salary of 11 months:
Per month salary = 12,000 Taka
Total salary = 12,000 * 11 = 1,32,000 Taka
Total expense: 7,04 000 + 50,000 + 15,000 + 1,20,000 + 1,65,000 + 15,000 + 48,000 + 4,40,000 + 1,32,000 = 1,689,000 Taka
Profit: 25% of total expense = 1,689,000 * 25% = 4,22,250 Taka

Total budget: 1,689,000 + 4,22,250 = 2, 411, 250 Taka ≈ 2,500,000 Taka

5. FEATURES NOT TO BE TESTED

The following is a list of the areas that will not be specifically addressed. All testing in these areas will be indirect as a result of other testing efforts.

1) **Third-party Services Integrations:**
The integration of third-party services, such as payment gateways or authentication providers, will not be tested in detail. However, integration points will be validated indirectly during the testing of related functionalities that rely on these services.

2) **Operating System Compatibility:**
The platform's compatibility with various operating systems (OS) and devices will not be explicitly tested for each OS version. Instead, comprehensive testing will be performed on a representative set of OS versions to ensure a consistent user experience.

3) **Cosmetic Changes:**
Minor cosmetic changes, such as color scheme updates or font adjustments, will not be subject to dedicated testing efforts. These changes are generally unlikely to impact the platform's core functionality.

4) **Network Infrastructure:**
The underlying network infrastructure and its components, such as routers and switches, will not undergo specific testing. However, network stability and performance will indirectly impact the overall user experience and will be monitored during testing.

5) **Data Privacy Compliance:**
While data privacy is crucial, the specific compliance of the platform with various data privacy regulations will not be tested in detail. Instead, the platform's general data protection measures will be validated.

6) **External Hardware Interaction:**
Direct interaction with external hardware devices (e.g., medical devices) will not be tested. Instead, the platform's ability to integrate with such devices will be validated through end-to-end scenarios.

7) **Database Transactions:**
Low-level database transactions and specific database management operations will not be tested individually. Instead, these interactions will be indirectly tested as part of broader functionality tests.

6. TESTING APPROACH

6.1 Testing Levels

The testing for the “Easy Health” project will consist of Unit, Integration, Acceptance, Performance, Compatibility and Regression testing levels. It is hoped that there will be at least one full time independent test person for system/integration testing. However, with the budget constraints and timeline established, most testing will be done by the test manager with the development teams’ participation.

1) **Unit Testing:**

Unit testing assures that individual components, methods, and functions inside the "Easy Health App" are valid. The testing team may detect issues early by isolating these components and confirming their behavior, resulting in more robust and dependable software.

Based on Requirement Specifications:

a) **System Login:**

- **Test Case Creation:**
  Design test cases to cover different scenarios, including valid and invalid usernames/passwords, account status (active/inactive), and authentication methods.

- **Mock Dependencies:**
  Isolate the System Login module from external dependencies such as user database or authentication services using mock objects.

- **Execute Tests:**
  Implement unit tests to call the login function/method with various inputs. Verify that the function handles different scenarios, such as correct login, incorrect password, and locked accounts.

- **Assertion and Validation:**
  Use assertion libraries to validate that the function's return values match expected outcomes. Confirm that appropriate error messages are returned for failed login attempts.

b) **System Registration:**

- **Test Case Design:**
  Develop test cases to cover valid and invalid registration inputs, duplicate usernames, and missing fields.

- **Mock Dependencies:**
  Isolate the registration process from the database or external services by using mock objects.

- **Execute Tests:**
  Write unit tests to simulate user registration with different inputs. Validate that the registration function/method handles various scenarios, such as successful registration and duplicate usernames.

- **Assertion and Validation:**
  Check that the function returns expected results for successful and unsuccessful registration attempts. Verify that appropriate error messages are provided for validation failures.

c) **User Profile Management:**

- **Test Case Preparation:**
  Design test cases to cover profile updates, password changes, and user-specific configurations.

- **Mock Dependencies:**
  Isolate the profile management module from the database or external services using mock objects.

- **Execute Tests:**
  Implement unit tests to call functions/methods for updating profile details, changing passwords, and modifying configurations.

- **Validation:**
  Validate that the profile management functions handle input changes correctly. Check if changes are correctly reflected in the user's profile data.

d) **Ambulance Request:**

- **Test Case Creation:**
  Develop test cases to simulate ambulance requests with various conditions such as critical or non-critical situations, location data, and patient information.

- **Mock Dependencies:**
  Isolate the ambulance request module from external services using mock objects, especially for communication with ambulance drivers.

- **Execute Tests:**
  Write unit tests to call functions/methods that initiate ambulance requests. Verify that the requests are formatted correctly and include relevant information.

- **Validation:**
  Validate that the ambulance request functions handle different scenarios appropriately. Check if the correct notifications are generated and sent.

e) **Communication:**

- **Test Case Design:**
  Create test cases for different communication methods like messaging drivers and calling drivers.

- **Mock Dependencies:**
  Isolate the communication module from actual communication channels using mock objects.

- **Execute Tests:**
  Implement unit tests to simulate communication functions/methods, such as sending messages and making calls.

- **Assertion and Validation:**
  Validate that the communication functions work as expected and initiate the desired communication channels. Verify that appropriate messages or calls are generated.

f) Payment Processing:

- **Test Case Preparation:**
  Develop test cases for different payment methods (Bkash, banking, cash), payment amounts, and success/failure scenarios.

- **Mock Dependencies:**
  Isolate the payment processing module from actual payment gateways using mock objects.

- **Execute Tests:**
  Write unit tests to call functions/methods that handle payment processing. Simulate payment requests with different inputs.

- **Validation:**
  Validate that the payment functions handle different payment methods correctly. Verify that the payment results are processed accurately based on test inputs.

Tools: JUnit, NUnit, pytest.

Entry Criteria for Unit Testing:

1. **Code Availability:** The code for the module under testing should be developed and ready for testing.

2. **Code Review:** The code should have undergone a code review process to ensure quality and adherence to coding standards.

3. **Unit Test Plan:** The unit test plan should be prepared, outlining the test cases, scenarios, and expected outcomes.

4. **Mock Dependencies:** Mock objects or simulators for external dependencies should be prepared to isolate the module from external services.

5. **Unit Test Environment:** The testing environment, including required tools and frameworks, should be set up and ready.

6. **Test Data Preparation:** Any necessary test data or input parameters should be prepared to execute the test cases effectively.

7. **Unit Test Cases:** The unit test cases should be developed, covering various scenarios and functionalities of the module.

8. **Code Compilation:** The code should be successfully compiled and built, ensuring that there are no syntax errors.

Exit criteria for Unit Testing:

1. **Test Execution:** All planned unit test cases have been executed for the module under testing.

2. **Defect Reporting:** Any defects or issues identified during unit testing have been documented and reported in the defect tracking system.

3. **Test Coverage:** Adequate test coverage has been achieved, ensuring that critical paths and functionalities have been thoroughly tested.

4. **Mock Dependency Validation:** The interactions between the module and its mock dependencies have been validated, ensuring the correct simulation of external services.

5. **Validation of Expected Outcomes:** The actual outcomes of the unit tests have been compared with the expected outcomes defined in the test plan.

6. **Passing Criteria:** The module passes the unit testing phase if the test cases execute successfully without critical failures and meet the predefined acceptance criteria.

7. **Defect Fix Verification:** Defects reported during unit testing have been fixed, and a verification of the fixes has been performed.

8. **Code Quality:** The code has maintained or improved its quality standards after addressing any issues identified during testing.

9. **Documentation:** The unit testing results, including test cases, test execution logs, and defect reports, are properly documented.

10. **Approval:** The unit testing phase is considered complete and approved by the QA team, signaling readiness for integration and security testing.

2) Integration Testing:

Integration testing combines different software parts to ensure they work together seamlessly. It detects issues caused by their interactions, aiming to validate accurate integration and smooth data flow for a stable, functional application.

a) **Test Environment Setup:**

Set up a controlled integration testing environment that mirrors the production environment as closely as possible. Ensure all components and systems are properly configured and connected.

b) **Integration Test Plan:**

Prepare a detailed integration test plan that outlines the integration scenarios, test cases, and expected outcomes.

c) **Test Data Preparation:**

Generate or gather test data that represents real-world scenarios and covers different integration paths.

d) **Component Integration Testing:**

Test the integration of individual components (modules, services, APIs) to ensure they work together as intended. Validate data flow, communication, and interactions between components.

e) **Top-Down Integration Testing:**

Start with testing the higher-level components and gradually integrate lower-level components. Verify that the higher-level components can interact with their subcomponents seamlessly.

f) **Bottom-Up Integration Testing:**

Begin with testing the lower-level components and integrate higher-level components gradually. Verify that lower-level components can provide correct data and interactions to higher-level components.

g) **Middleware and Interface Testing:**

Test the integration points where different systems or modules connect via middleware or interfaces. Validate data exchange, message formats, and compatibility.

h) **Data Flow Testing:**

Test the flow of data across different components, systems, and databases. Verify that data is transferred accurately and consistently.

i) **Concurrency and Parallelism Testing:**

Test scenarios where multiple components or processes run concurrently or in parallel. Validate that the integration can handle simultaneous operations effectively.

j) **API Integration Testing:**

Test the integration of different application programming interfaces (APIs) to ensure they can communicate and exchange data. Verify API endpoints, data formats, and authentication mechanisms.

k) **Database Integration Testing:**

Test interactions between different components and the database. Verify data retrieval, storage, updates, and consistency.

l) **Security Integration Testing:**

Test the integration points for security vulnerabilities and ensure that security measures are maintained across components. Verify that user access and data privacy are not compromised.

m) **Defect Resolution and Verification:**

Address and fix any integration-related defects or issues identified during testing. Verify that the fixes have been implemented correctly and do not introduce new problems.

Tools: JUnit, TestNG, Postman

Entry Criteria for Unit Testing:

1. **Component Availability:**

   All individual components/modules that are part of the integration must be fully developed and tested.

2. **Test Environment Readiness:**

   The integration testing environment should be set up with the necessary databases, servers, middleware, and interfaces.

3. **Test Data Preparation:**

   Relevant and representative test data must be prepared to simulate various integration scenarios.

4. **Integration Test Plan:**

   A detailed integration test plan should be prepared, outlining the integration scenarios, test cases, and expected outcomes.

5. **Unit Testing Completed:**

   Individual components/modules must have passed their respective unit tests successfully.

6. **Component Interfaces Defined:**

   The interfaces and communication protocols between different components/modules should be clearly defined.

7. **Code Review and Static Analysis:**

   Code reviews and static analysis of the individual components/modules should be completed to identify and fix any potential issues.

8. **Integration Test Environment Validation:**

   Validate that the integration test environment is properly configured and can mimic the production environment.

Exit criteria for Unit Testing:

1. **Successful Component Integration:**

   All planned integration scenarios have been tested, and the integration of components/modules has been successful.

2. **Defect Resolution:**

   All identified integration defects have been addressed, fixed, and retested.

3. **Consistent Data Flow:**

   Data flows correctly and consistently between different components/modules without loss or corruption.

4. **Component Interfaces Validated:**

   The interfaces between components/modules have been thoroughly validated for data exchange and compatibility.

5. **Concurrency and Parallelism Tested:**

   Concurrency and parallelism scenarios have been tested to ensure smooth operation under simultaneous interactions.

6. **Security Measures Maintained:**

   Security measures have been verified to remain intact across integrated components/modules.

7. **Documentation Updated:**

   Integration testing results, including observations, outcomes, and defect resolutions, are documented.

8. **Integration Test Report Prepared:**

   An integration test report summarizing the testing process, outcomes, and any remaining risks is prepared.

9. **Approval and Sign-off:**

   Stakeholders, including development and testing teams, review the integration testing results and provide approval for moving to the next testing phase or deployment.

3) Acceptance Testing:

Purpose: Confirm that the application meets specified requirements and is acceptable to end-users.
Scope: End-to-end scenarios and user acceptance criteria.
Test Focus: Validating the app's functionality from the user's perspective.
Participants: Testers, QA Team, End-users.
Tools: Cucumber, Selenium, TestNG

4) Performance Testing:

Purpose: Evaluate the application's responsiveness, stability, and scalability.
Scope: Load testing, stress testing, and measuring response times.
Test Focus: Ensuring the app performs well under different conditions.
Participants: Testers, QA Team.
Tools: JMeter, Gatling, Apache Benchmark

5) Compatibility Testing:

Purpose: Verify the application's compatibility across various platforms, devices, and browsers.
Scope: Different operating systems, browsers, and screen sizes.
Test Focus: Ensuring a consistent experience for all users.
Participants: Testers, QA Team.
Tools: BrowserStack, CrossBrowserTesting, Sauce Labs

6) Regression Testing:

Purpose: Ensure that new changes do not adversely affect existing functionalities.
Scope: Testing areas affected by recent changes and related functionalities.
Test Focus: Detecting regressions and maintaining application stability.
Participants: Testers, QA Team.
Tools: Selenium, JUnit, TestNG

6.2 Test Tools

1) **Test Management Tools:**
   - Tool: Asana
   - Purpose: Asana will serve as the central hub for managing test cases, tracking defects, and monitoring testing progress. It streamlines communication among the testing team and facilitates task assignment, issue tracking, and reporting.

2) **Automated Testing Frameworks:**
   - Tools: Selenium, Appium
   - Purpose: Selenium and Appium will enable automated testing of the app's user interface (UI) and mobile functionalities across different platforms and devices. Automation improves efficiency by executing repetitive tests, ensuring consistent outcomes, and identifying UI glitches.

3) **Performance Testing Tools:**
   - Tools: JMeter, LoadRunner
   - Purpose: JMeter and LoadRunner will be employed for performance and load testing. These tools simulate user activity, measure response times, and assess the app's scalability, ensuring it can handle expected user loads.

4) **Security Testing Tools:**
   - Tools: OWASP ZAP (Zed Attack Proxy), Burp Suite
   - Purpose: OWASP ZAP and Burp Suite will aid in identifying potential security vulnerabilities and threats. These tools help ensure that the app's data and users are safeguarded against cyber risks.

5) **API Testing Tools:**
   - Tools: Postman, SoapUI
   - Purpose: Postman and SoapUI will facilitate comprehensive testing of APIs and backend services. They validate data exchange, API functionality, and integration integrity.

6) **Monitoring and Logging Tools:**
   - Tools: New Relic, Splunk
   - Purpose: New Relic and Splunk assist in monitoring app performance, capturing logs, and identifying issues for timely resolution.

7) **Continuous Integration Tools:**
   - Tools: Jenkins, GitLab CI/CD
   - Purpose: Jenkins and GitLab CI/CD enable continuous integration and deployment, automating the build and testing process whenever new code is integrated.

6.3 Meetings

Week	Meeting Type	Agenda	Participants
1	Test Team Meeting	Assess current progress and discover mistake patterns and prospective difficulties.	Test Team Members
2	Test Team Lead, Devs & Project Manager Meeting	Review testing advances, challenges encountered, and testing-development cooperation.	Test Team Leader, Devs' Team Lead, Project Manager
3	Test Team Meeting	Examine current progress and identify mistake trends and prospective difficulties.	
4	Test Team Lead, Devs & Project Manager Meeting	Analyze testing progress, challenges encountered, and testing-development cooperation.	Test Team Leader, Devs' Team Lead, Project Manager
…	…	…	…
N	Emergency Meeting (as required)	Respond to emergency scenarios or serious concerns discovered during testing.	Relevant participants depending on the nature of the emergency

7. TEST CASES/TEST ITEMS

Project Name: Easy Health: Healthcare App		Test designed by: Khaled		
Test Case ID: EH_01 Test Priority (Low, Medium, High): High		Test designed date: 01/07/2023 Test executed by: Khaled		
Module Name: System login		Test execution date: 07/07/2023		
Test Title: Verify the home page feature				
Description: Test to view logged in user's home page				
Precondition (if any): User must have valid username and password				
Test Steps	Test Data	Expected Results	Actual Results	Status (Pass/Fail)
1. Go to application login page 2. Enter username 3. Enter password 4. Click submit	Username: khaledtester@gmail.com Password: khaled_256	User can login to the application.	User successfully logged in.	Pass
Post Condition: User must contain a valid username with a valid password with database to successfully login to their account. The account session details are logged in the database.				

Project Name: Easy Health: Healthcare App			Test designed by: Khaled	
Test Case ID: EH_02 Test Priority (Low, Medium, High): High			Test designed date: 01/07/2023 Test executed by: Khaled	
Module Name: Create new account			Test execution date: 07/07/2023	
Test Title: New user with new username and password				
Description: Test application’s create new account feature				
Precondition (if any): User must input valid email or phone number as username				
Test Steps	Test Data	Expected Results	Actual Results	Status (Pass/Fail)
1. Open the application’s registration page 2. Click on register account 3. Enter email or phone number 4. Enter OTP code sent to email or phone 5. Confirm new password 6. Click on submit	Phone number: 01572348796 OTP: 451246 Password: khaled_572	User should login to the application with the newly created account.	New user created successfully.	Pass
Post Condition: User can successfully login to their account.				

Project Name: Easy Health: Healthcare App			Test designed by: Labiba	
Test Case ID: EH_03 Test Priority (Low, Medium, High): Medium			Test designed date: 03/07/2023 Test executed by: Labiba	
Module Name: Forgot password			Test execution date: 09/07/2023	
Test Title: Provide user with new password				
Description: Test application’s forgot password feature				
Precondition (if any): User must have valid email or phone number				
Test Steps	Test Data	Expected Results	Actual Results	Status (Pass/Fail)
1. Open the application’s login page 2. Click on forgot password 3. Enter email or phone number 4. Enter OTP 5. Enter new password 6. Click on submit	Email: labiba338@gmail.com OTP: 792402 New password: labiba_338	User should login to the system with new password	User can successfully login to the system	Pass
Post Condition: User can successfully login to their account using their new password.				

Project Name: Easy Health: Healthcare App			Test designed by: Labiba	
Test Case ID: EH_04 Test Priority (Low, Medium, High): Medium			Test designed date: 04/07/2023 Test executed by: Labiba	
Module Name: Menu option			Test execution date: 09/07/2023	
Test Title: Ambulance call request				
Description: User requesting for ambulance				
Precondition (if any): User must be logged in to the system to request for an ambulance				
Test Steps	Test Data	Expected Results	Actual Results	Status (Pass/Fail)
1. Login to the application 2. Click on menu 3. Click on request for ambulance 4. Choose the type of ambulance call	Menu option: Ambulance call Types of ambulance call: Critical/Less critical	User should be able to see vehicle options to request	User being able to see vehicle options to request	Pass
Post Condition: User can successfully request for ambulance.				

Project Name: Easy Health: Healthcare App			Test designed by: Rubyea	
Test Case ID: EH_05 Test Priority (Low, Medium, High): High			Test designed date: 07/07/2023 Test executed by: Rubyea	
Module Name: Availability of ambulance			Test execution date: 15/07/2023	
Test Title: Checking the availability of nearby ambulance				
Description: User can book an ambulance by checking the availability				
Precondition (if any): User must request for ambulance and choose the type of ambulance call				
Test Steps	Test Data	Expected Results	Actual Results	Status (Pass/Fail)
1. Open the application 2. Login and click on menu 3. Request for ambulance 4. Select nearby ambulance (according to preference) 5. Click done	Select an ambulance based on the available nearby ambulance(s)	User will be able to book their preferable ambulance	User can book their preferred ambulance	Pass
Post Condition: User successfully booked the ambulance.				

Project Name: Easy Health: Healthcare App			Test designed by: Rubyea	
Test Case ID: EH_06 Test Priority (Low, Medium, High): High			Test designed date: 11/07/2023 Test executed by: Rubyea	
Module Name: Check payment methods			Test execution date: 15/07/2023	
Test Title: Checking user’s payments				
Description: Verifying user’s payment feature				
Precondition (if any): User must confirm booking an ambulance				
Test Steps	Test Data	Expected Results	Actual Results	Status (Pass/Fail)
1. Login to the application 2. Book an ambulance 3. Click on payment method 4. Choose a payment method 5. Select one from – VISA Card/Bkash/Nagad/Rocket 6. Enter the amount 7. Enter your pin and click on agree 8. Click on done	Payment details according to the selected ambulance service	User will be able to see the payment details	User being able to see the payment receipt	Pass
Post Condition: User can successfully see the payment receipt after completing their payment.				

Project Name: Easy Health: Healthcare App			Test designed by: Tanzim	
Test Case ID: EH_07 Test Priority (Low, Medium, High): High			Test designed date: 16/07/2023 Test executed by: Tanzim	
Module Name: Contact driver			Test execution date: 22/07/203	
Test Title: Contact driver using direct cellular call/system chatting				
Description: User can contact the ambulance driver for any required information				
Precondition (if any): User must book an ambulance and pay for it				
Test Steps	Test Data	Expected Results	Actual Results	Status (Pass/Fail)
1. Login to the system and click on menu 2. Book an ambulance 3. Choose payment method and pay 4. Click on contact driver and choose a medium of contact	Contact medium: Cellular call or System chatting	User will be able to communicate with the driver	User can communicate with the driver	Pass
Post Condition: User can successfully contact the ambulance driver.				

Project Name: Easy Health: Healthcare App			Test designed by: Tanzim	
Test Case ID: EH_08 Test Priority (Low, Medium, High): Medium			Test designed date: 16/07/2023 Test executed by: Tanzim	
Module Name: User information update			Test execution date: 24/07/2023	
Test Title: Updating user information				
Description: User being able to update their profile information				
Precondition (if any): User must be logged in using valid username and password				
Test Steps	Test Data	Expected Results	Actual Results	Status (Pass/Fail)
1. Login to the system 2. Go to menu 3. Click on update information 4. Enter the update information 5. Select submit 6. Verify using email or phone number to save the updates	Fill out the information form that will update the profile details	User being able to update their information and view the updated details	User can update their information and view it	Pass
Post Condition: User can successfully view their updated profile information.				

8. ITEM PASS/FAIL CRITERIA

1) **Test item for System login module, EH_01:**
   Pass: User can login successfully with valid credentials
   Fail: User cannot login with valid credentials

2) **Test item for Creating new account module, EH_02:**
   Pass: User can successfully create a new account with valid email or phone number
   Fail: User cannot create a new account even with valid email or phone number

3) **Test item for Forgot password module, EH_03:**
   Pass: User can change the previous password and login using the new password
   Fail: User cannot login to the application using the new password

4) **Test item for Ambulance request module, EH_04:**
   Pass: User can choose an ambulance from the vehicle request options
   Fail: User cannot choose an ambulance from the vehicle request options

5) **Test item for Availability of ambulance module, EH_05:**
Pass: User can book their preferred ambulance service
Fail: User cannot book their preferred ambulance service

6) **Test item for Payment methods module, EH_06:**
Pass: User can see their payment details and can pay using any method
Fail: User cannot continue with payment due to bad gateways

7) **Test item for Contact driver module, EH_07:**
Pass: User can contact the ambulance driver
Fail: User cannot contact the ambulance driver

8) **Test item for User information update module, EH_08:**
Pass: User can update their information and view it
Fail: User cannot update their information nor view any changes

9. TEST DELIVERABLES

- Test Design Specifications
- Test Plans
- Test Cases and Test Scripts
- Traceability Matrix
- Defect Reports and Lag
- Test Execution Report
- Performance Test Results
- Security Test Results
- Usability Test Feedback

10. STAFFING AND TRAINING NEEDS

1) **Staffing Needs:**

- **Development Team:**
Frontend Developers: Responsible for creating user interfaces and user experience.
Backend Developers: In charge of building the backend logic and server-side components.
UI/UX Designer: Designs intuitive and user-friendly interfaces.

- **Quality Assurance (QA) Team:**
Plan and execute testing activities, identify defects, and ensure quality.

- **Test Manager:**
To oversee the testing strategy, lead the test team, and ensure alignment with project goals. Should have experience in healthcare software testing and project management.

- **Test Engineers:**
  Individuals responsible for writing and executing test cases. Should have a mix of junior and senior testers, with some possessing domain knowledge in healthcare.

- **Project Management**:
  Project Manager: Oversees the project, manages resources, timelines, and communication.

- **Training Coordination:**
  Training Coordinator: Organizes and conducts training sessions for team members.

- **Security Specialist:**
  Ensures the application's security and data protection.

2) **Training Needs:**

- **Technical Training:**
  Provide training on relevant programming languages, frameworks, and testing tools.
  Ensure team members are proficient in using development and testing environments.

- **Testing Methodologies:**
  Train QA team members in various testing methodologies such as unit testing, integration testing, and user acceptance testing.
  Ensure they understand how to design effective test cases and validate software quality.

- **Agile and Project Management:**
  Offer training on Agile methodologies to ensure the team understands iterative development, regular feedback, and continuous improvement.
  Train project managers in project management principles, risk management, and resource allocation.

- **UI/UX Design:**
  Provide UI/UX designers with training on design principles, user-centered design, and tools for creating visually appealing and user-friendly interfaces.

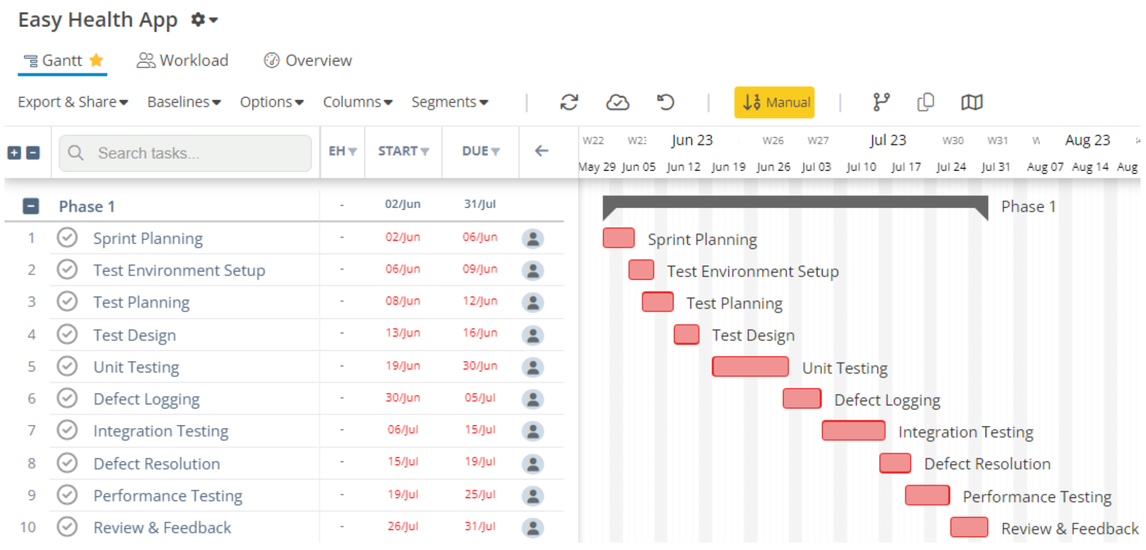
- **Security Awareness:**
  Offer security training to raise awareness about secure coding practices, data privacy, and potential vulnerabilities.

- **Continuous Learning:**
  Given the rapid evolution of both healthcare and IT, regular workshops on emerging trends, technologies, and best practices will keep the team updated and competitive.

11. RESPONSIBILITIES

Responsibilities	TM (Test Manager)	PM (Project Manager)	Dev Team	Test Team	Client
Define test objectives and scope	X	X		X	
Identify testing requirements	X	X			
Create test strategy and approach	X				
Determine testing environment and data	X		X		
Design test cases and test scenarios				X	
Review and approve test cases	X			X	
Execute test cases				X	
Report and track defects				X	
Perform regression testing			X	X	
Conduct integration testing			X	X	
Perform performance testing				X	
Document test results and generate reports	X			X	
Obtain client/user acceptance					X
Finalize test documentation and deliverables	X	X			
Review and approve final test deliverables	X	X			

12. TESTING SCHEDULE



		EH	START	DUE
	Phase 1	-	02/Jun	31/Jul
1	Sprint Planning	-	02/Jun	06/Jun
2	Test Environment Setup	-	06/Jun	09/Jun
3	Test Planning	-	08/Jun	12/Jun
4	Test Design	-	13/Jun	16/Jun
5	Unit Testing	-	19/Jun	30/Jun
6	Defect Logging	-	30/Jun	05/Jul
7	Integration Testing	-	06/Jul	15/Jul
8	Defect Resolution	-	15/Jul	19/Jul
9	Performance Testing	-	19/Jul	25/Jul
10	Review & Feedback	-	26/Jul	31/Jul

Fig. 2.1: Gantt Chart

13. PLANNING RISKS AND CONTINGENCIES

1) **Risk: Evolving regulatory requirements:**

   Contingency: Establish a regulatory monitoring team to track changes. Plan for regular compliance assessments and updates to ensure the software remains compliant with the latest regulations.

2) **Risk: Lack of integration with existing systems**

   Contingency: Conduct thorough integration testing during development. Implement a modular architecture to facilitate future integrations. Have backup plans in case integrations encounter issues.

3) **Risk: Technical compatibility issues**

   Contingency: Conduct compatibility testing with various devices and platforms. Plan for adaptation or alternative solutions in case technical issues arise. Maintain a close relationship with technology vendors.

4) **Risk: Scope expansion due to user requests**

   Contingency: Implement a strict change management process. All additional features must be evaluated for impact on the project timeline and budget. Obtain approvals and prioritize requests based on project goals.

5) **Risk: Key team members leaving**

   Contingency: Cross-train team members and maintain documentation to ensure knowledge sharing. Develop contingency plans to mitigate the impact of key members' departure.

6) **Risk: Inadequate user training**

   Contingency: Develop comprehensive user training materials and resources. Provide training sessions and user support to ensure effective system adoption. Offer additional training if needed.

7) **Risk: Unforeseen technical challenges**

   Contingency: Establish research and troubleshooting team to address technical issues. Maintain regular communication with developers and experts to find quick solutions.

8) **Risk: Data security breaches**

   Contingency: Implement robust security measures, encryption protocols, and access controls. Regularly monitor and audit security practices. Develop incident response plans to handle breaches effectively.

9) **Risk: Third-party service disruptions**

   Contingency: Have alternative service providers identified for critical functions. Maintain a service level agreement with third-party vendors to ensure swift issue resolution.

10) **Risk: Scope misalignment with client expectations**

   Contingency: Maintain continuous communication with the client. Provide regular updates and demos to align expectations with project progress. Document all discussions and agreements.

11) **Risk: Delays in equipment or technology procurement**

   Contingency: Identify critical equipment and technologies early. Develop relationships with suppliers and maintain open communication. Have backup plans for alternative sources or technologies.

14. APPROVALS

Project Sponsor – Khaled	Approved
Development Management – Rubyea	Approved
EDI Project Manager – Khaled	Approved
RS Test Manager – Tanzim	Approved
RS Development Team Manager – Labiba	Approved
Reassigned Sales – Rubyea	Approved
Order Entry EDI Team Manager – Labiba	Approved