A picture containing logo

Description automatically generated

**Software Test Plan**

University of Maryland Global Campus

SWEN 670 – Team A

Fall Semester

Version 1.1

October 29, 2022

Document Control

Document Information

|  |  |
| --- | --- |
| © | Information |
| Document Id | USPS-IDE-STP |
| Document Owner | UMGC SWEN 670 TEAM A |
| Issue Date | September 17, 2022 |
| Last Saved Date | October 29, 2022 |
| File Name | USPS\_Informed\_Delivery\_App\_Enhancements\_STP\_Team\_A.docx |

Document History

|  |  |  |
| --- | --- | --- |
| Version | Issue Date | Changes |
| 1.0 | 9/17/2022 | Initial Version |
| 1.1 | 10/29/2022 | Post-development updates |
|  |  |  |
|  |  |  |

Contents

[1 Introduction 1](#_Toc118544869)

[1.1 Purpose 1](#_Toc118544870)

[1.2 Scope 1](#_Toc118544871)

[1.3 Test Items 1](#_Toc118544872)

[1.3.1 To be Tested 1](#_Toc118544873)

[1.3.2 Not to be Tested 2](#_Toc118544874)

[1.4 Project Documents 2](#_Toc118544875)

[2 Approach 3](#_Toc118544876)

[2.1 Strategy 3](#_Toc118544877)

[2.2 Types of Testing Used 4](#_Toc118544878)

[2.3 Test Verification 5](#_Toc118544879)

[3 Environment 6](#_Toc118544880)

[4 Roles and Responsibilities 6](#_Toc118544881)

[5 Schedule 7](#_Toc118544882)

[6 Defect Management 8](#_Toc118544883)

[6.1 Defect Severity Levels 8](#_Toc118544884)

[6.2 Risks and Contingencies 8](#_Toc118544885)

[6.3 Suspension and Resumption Criteria 9](#_Toc118544886)

[7 Proposed Test Cases 9](#_Toc118544887)

[7.1.1 Test Case 01: QR Code Reading from Mail Piece 9](#_Toc118544888)

[7.1.2 Test Case 02: Embedded Links Navigate User to the Browser 10](#_Toc118544889)

[7.1.3 Test Case 03: Google Assistant Search for an Mailpiece 10](#_Toc118544890)

[7.1.4 Test Case 04: Google Assistant Add a new Sender/Keyword for Notifications 10](#_Toc118544899)

[7.1.5 Test Case 05: Google Assistant Add an existing Sender/Keyword for Notifications 11](#_Toc118544900)

[7.1.6 Test Case 06: Google Assistant Opens the most recent Digest email 11](#_Toc118544917)

[7.1.7 Test Case 07: Google Assistant Opens the notifications page 11](#_Toc118544918)

[7.1.8 Test Case 08: Google Assistant Opens Application 12](#_Toc118544919)

[7.1.9 Test case 09: Add Sender/Keyword Notification list visually/manually 12](#_Toc118544941)

[7.1.10 Test case 10: Delete Sender/Keyword Notification list visually/manually 12](#_Toc118544942)

[7.1.11 Test case 11: Receive Notification 13](#_Toc118544943)

[7.1.12 Test case 12: View Notification visually/manually 13](#_Toc118544944)

[7.1.13 Test case 13: Clear Notification 14](#_Toc118544945)

[7.1.14 Test case 14: Clear All Notifications 14](#_Toc118544946)

[7.1.15 Test case 15: Display QR code and click Mail View able link 15](#_Toc118544947)

[7.1.16 Test case 16: Mail View Display URL and a clickable link 15](#_Toc118544948)

[7.1.17 Test case 17: Mail View Display text from mail 16](#_Toc118544949)

[7.1.18 Test case 18: Mail View Display verify contact information 16](#_Toc118544950)

[7.1.19 Test case 19: Analytics Dashboard confirms screen time 17](#_Toc118544951)

[7.1.20 Test case 20: Analytics Dashboard displays link metrics 17](#_Toc118544952)

[7.1.21 Test case 21: Analytics Dashboard displays searches performed 17](#_Toc118544953)

[7.1.22 Test case 22: Mail View Call Phone Number 18](#_Toc118544978)

[7.1.23 Test case 23: Mail View Text Phone Number 18](#_Toc118544979)

[7.1.24 Test case 24: Mail View Email provided email address. 19](#_Toc118544980)

[8 Assumptions and Constraints 19](#_Toc118544981)

[8.1 Assumptions 19](#_Toc118544982)

[8.2 Constraints 20](#_Toc118544983)

[9 Deliverables 20](#_Toc118544984)

[10 Appendices 21](#_Toc118544985)

[10.1 Definitions, Acronyms, Abbreviations 21](#_Toc118544986)

[10.2 References 21](#_Toc118544987)

# Introduction

## Purpose

The software test plan (STP) aims to describe the approach Team A will use to test the United States Postal Service (USPS) Informed Delivery mobile application enhancements to be developed during this development cycle. The goal will be to utilize an approach that provides complete coverage of all code to ensure that all source code is processed correctly during the execution of the application's various features. In addition, when issues are identified during compilation time or execution, the issues will be logged as bugs for the development team to address during future development sprints to ensure the final deliverable is error-free.

## Scope

The scope described in this document will encompass the types of testing, provide detailed test cases, and the acceptance criteria for each test case. Each testing strategy has been selected to verify the expected functionality of the system. The strategies include unit tests, integration testing, and user acceptance testing.

Risk is inherent in all development initiatives, but perhaps even more in this project because this development cycle operates with two parallel teams working semi-independently on separate backlogs. As a result, there are some project dependencies between the groups, so it is critical to perform integration testing throughout the project before merging the code into the main branch.

Finally, all development initiatives rely on user feedback to ensure the requirements are satisfied as expected. Therefore, user acceptance testing is critical in obtaining the end-user's acceptance. If issues in the system are found during the testing process, the issue is prioritized by severity and added as a defect in the project GitHub repository.

Testing all accessibility features, including voice input and screen reading, is not within the scope of Team A's requirements. However, team B's documentation will cover all accessibility features.

## Test Items

### To be Tested

All application functionalities will be tested on both the iOS and Android platforms.

1. The application shall connect to any email service using user-provided email address and password combination.
2. The application shall connect to the Google Vision application programming interface (API).
3. The application shall retrieve text from an image.
4. The application shall retrieve information from logos.
5. The application shall create a daily digest of the user's Informed Delivery.
6. The application shall read a quick response (QR) code.
7. The application shall read a barcode.
8. The application shall notify for items in the notification list
9. The application shall allow the user to add or remove keywords to the notification list.
10. The application shall allow the user to search for mail pieces/emails using Google Assistant and Siri
11. The application shall allow the user to add and delete keywords that are being notified for using Google Assistant and Siri.
12. The application shall allow the user to visit the daily digest using Google Assistant and Siri.
13. The application shall cache retrieved data locally using a SQLite local database
14. The application shall track specific user activities and report them to a Google Firebase analytics dashboard.
15. The application shall allow users to select and navigate links in the mail screen.
16. The application shall allow users to select barcodes or QR codes and navigate to any stored hyperlinks in the mail screen.
17. The application shall allow the user to open the contacts of a sender from the mail screen.
18. The application shall allow the user to call or text a phone number from the mail screen.
19. The application shall allow the user to reply to an email address from the mail screen.

### Not to be Tested

1. Accessibility functionality will not be tested under Team A; this will be covered by Team B.

## Project Documents

This Technical Design Document is part of a set of documents created to aid in developing the USPS Informed Delivery Application and to provide artifacts with vital information for the application's ongoing support and operation throughout its life cycle.

**The following documents are included in the entire documentation package:**

* + –

|  |  |  |  |
| --- | --- | --- | --- |
|  | Document | Version | Date |
| 1 | Project Management Plan (PMP) | 1.2 | 9-17-2022 |
| 2 | Software Requirements Specification (SRS) | 1.2 | 10-29-2022 |
| 3 | Technical Design Document (TDD) | 1.2 | 10-29-2022 |
| 4 | Software Test Plan (STP) | 1.1 | 10-29-2022 |
| 5 | Programmers Guide (PG) | 1.1 | 11-05-2022 |
| 6 | Deployment and Operations (DevOps) | 1.1 | 11-05-2022 |
| 7 | User Guide (UG) | 1.0 | 11-05-2022 |
| 8 | Test Report (TR) | 1.0 | 11-05-2022 |

# Approach

## Strategy

When it comes to testing, each organization has its own standards to maintain to have a successful strategy/plan for testing. We have to set up automation testing, create a test schedule, and plan resources. As testing progresses later in the project, we would be able to create test plans and use sample test data. The testing strategy shows what tests, when in the SDLC, and how it progresses to have a high-quality product.

**What**: Unit testing, integration testing, and system testing. A Performance test would also be an ideal choice of test to use for such software since it has a high probability of being used by thousands or millions of USPS-informed delivery users. As a result of this project's life span, the possibility of the product's exposure to the mass user is low, which makes it hard to do the performance test in such a short life span. The team recommends the following group working on this App consider performance testing.

**When:** Testing is throughout the iteration. Since the project is under Agile methodology, testing needs to happen early, often, and continuously as features change. The team is working in collaboration with team ‘one’ using the GitHub open-source platform. With the knowledge of the project manager and DevOps teams, testing takes place throughout the iteration.

**How:** Since features can evolve in response to changing customer requirements, the testing will be incremental and interactive. All the test strategies are performed with the collaboration of the dev team, QA, and PM to achieve the objective of a high-quality product.

**Step 1: Scope and Overview**

**Step 2: Test Approach (Type of Testing)**

**Step 3: Test Environment**

**Step 4: Analyze Risks**

**Step 5: Review and Approve**

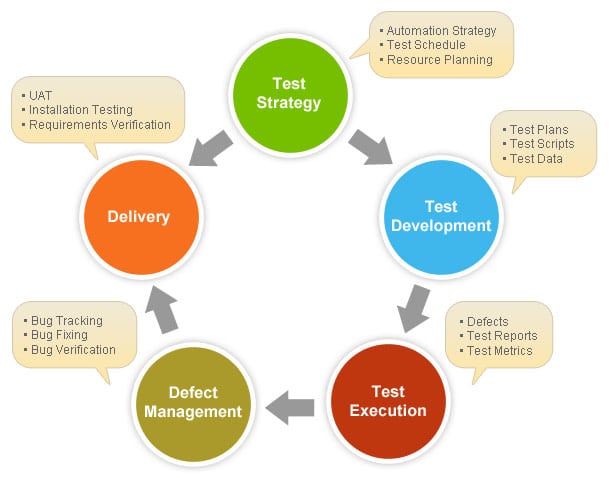


Figure 1Testing Process

[[1]](#footnote-2)

## Types of Testing Used

As testing progresses and bugs are identified, issues will be logged to the GitHub backlog. In addition, they will be logged with the information relating to their tests and the specific requirements that are associated with them.

* Unit Testing - Unit tests will provide the bulk of the test that will be performed throughout the development of this project. These tests will represent most of the code coverage. Unit tests are run automatically when submitting code changes into the build pipeline and can also be run manually when making local changes.
* Integration/Regression Testing - As components of the application are completed, it will be vital for the teams to perform integration tests regularly. The teams must ensure that we have a robust testing schedule to minimize the amount of re-work and defects that could be present. Integration testing will be performed using Flutter's package called integration test. This package functions similarly to how other self-driving tests function.
* User Acceptance Testing - User Acceptance Tests will verify that the application meets the objectives of a requirement. This will allow the team to ensure that they can meet the needs of the end-users and the client. Once a requirement is completed, team members will get together to perform User Acceptance Testing due to the unique nature of our relationship with the customer. This means users will not be able to test the product initially as would be completed in traditional acceptance testing. Developers will also not be permitted to do final acceptance testing on their own features to limit bias and to add an extra set of eyes. To confirm that a requirement is completed and has passed all final acceptance tests, the product owner must sign off and give his approval.
* Performance Testing – Performance testing evaluates how a system performs under a certain workload. This technique also determines the stability, speed, and responsiveness of the software application. The goal of this technique is simply to increase workload efficiency. This testing will be informal and will not use any particular tools; however, while testers are completing their test cases, they will keep in mind that all interactions not involving network latency should take no more than 1 second. A Performance test would have been best to use for such software that has a high probability of being used by a significant number of USPS-informed delivery users. As a result of this project's time being short, the product's exposure to a big number of users is low, which is hard to do the performance test in such a short time.

## Test Verification

Testing Verification is the process of verifying the product matches specification and requirements. This process is meant to find bugs early on in development and ensure we are continuously building the correct product. To accomplish this, developers will constantly be referring to design documents to fully understand the requirements, as well as reaching out to the stakeholders to clarify any uncertainty. Furthermore, developers will have code reviews and walk through, which other developers and team members can ask questions. Having several eyes on some code is a great way for team members to agree that the design and purpose of the feature was completed. These code reviews should be conducted before checking into GitHub. If the peer-reviewed code is successful, the code can be checked-in, and the next steps can be taken. If the peer-reviewed code fails, then the developer must revisit the code to ensure the requirements are met.

For manually-performed integration testing, testers must validate that any test cases they are testing match the requirements for the project. Testers will write and maintain test cases for each functional requirement. Testers will execute them with physical iOS and Android devices to confirm verification of product function according to the test case.

# Environment

This project includes three different environments; development, testing, and production. The developers will perform unit testing in the development environment. The local development environment will be running on the simulator device belonging to each individual developer. Finally, unit tests would be performed in Flutter's test environment.

Prior to putting the product into production, all of the testing will be carried out on the intermediate layer. The TestFlight testing environment for iOS and Google Play Console for android will be used for formal integration testing directly on devices.

In the production environment, user-ready software is deployed, executed, and made available for use by users. The product's production environment would be the final product available for download from Google Play or the Apple Store.

# Roles and Responsibilities

|  |  |
| --- | --- |
| Role | Responsibilities |
| Project Manager | Acts as a primary contact for development and the testing team.  Planning and organization of test process for the release.  Coordinate with Software Engineers performing Quality Assurance (QA) for any issues/problems encountered during testing.  Report progress on work assignments to the customer |
| Project Team | Understand requirements  Writing and executing Test cases  Reviewing Test cases, RTM  Defect reporting and tracking  Retesting and regression testing  Bug Review (conducted during weekly planning meeting)  Preparation of Test Data  Coordinate with Project Manager for any issues or problems encountered during test preparation/execution/defect handling. |
| Product Owner | Prepare Requirements Traceability Matrix and ensure proper tracing for all requirements. |

Table 1 Roles and Responsibilities

# Schedule

|  |  |  |  |
| --- | --- | --- | --- |
| Test Id | Test Name | Planned | Completed |
| 1 | Google Vision API | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 2 | Text from Image | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 3 | Info from Logos | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 4 | Daily Digest | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 5 | Read QR code | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 6 | Read Barcode | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 7 | Image Contains No Gatherable Info | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 8 | Validate clarity of returned info | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 9 | Open Camera | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 10 | Save photos to the device | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 11 | Cache data locally | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 12 | Send and display user metrics in Firebase Analytics dashboard | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |
| 13 | Test Google Assistant Functionality | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | COULDN’T COMPLETE |
| 14 | Test Mail Piece page functionality | M3: 10/6 - 10/19  M4: 11/1 - 11/4 | YES |

Table 2 Testing Schedule

# Defect Management

## Defect Severity Levels

There are three levels of defect severity with this project: S0, S1, and S2.

**S0**: This is a launch-blocking issue. This is an issue that causes catastrophic system failures, resulting in the inability for the user to use the application. Some indicators of S0 severity are:

* Inability to launch the application
* Application crashes
* An unresponsive user interface (UI)

In the presence of a S0 bug, all development should focus on immediately fixing this issue to allow for the application development to continue.

**S1**: This is a moderate issue, where user experience is degraded but not totally impeded. These tend to result in either poor user experience, unmet user expectations, or even poor backend performance that can degrade future user interactions. Some indicators of S1 severity are:

* Feature does not align with requirements
* Database is not persisted properly
* UI is not properly displayed on login
* Backend queries take a long time to return

Bugs with S1 severity will be prioritized accordingly but do not warrant the freezing of application development to proceed. These bugs will need to be fixed to release a polished product.

**S2**: These are small issues where there are noticeable defects that do not affect the user's experience in the application. Some indicators of S2 severity are:

* Misspelling of words in the UI
* Missing animations
* General UI element alignment

Bugs with S0 severity will be addressed after all S0 and S1 bugs have been addressed. The existence of S2 bugs is to be expected and will likely be discovered throughout the lifetime and maintenance of the application.

## Risks and Contingencies

Risk management focuses on identifying current and upcoming risks to the project and mitigations for it. Contingency planning is how to overcome possible risks.

Risk: Having a large team could possibly bring confusion across the team on what to work on.

Contingency: Having the GitHub backlog and a great task management tool can easily overcome this risk.

Risk: Team members not having enough experience.

Contingency: There will be a team analysis/discussion on strengths and weaknesses so everyone will have a better understanding of what they are comfortable with.

## Suspension and Resumption Criteria

There are several situations in which testing activities must be suspended due to an inability to properly vet the associated software application. Once these issues have been resolved, testing may resume as originally planned.

1. **Backend services unavailable**: In cases where Google Cloud Platform (GCP) or the testing email server is unavailable, testing must be suspended until they come back online. It is not possible to exercise the features of the application without an email server to draw emails from, and it is not possible to test the Optical Character Recognition (OCR) and voice integrations without access to GCP.
2. **The presence of S0 bugs that prevent use of the application**: In the case where a show-stopping bug is found that prevents the tester from using the features of the application (e.g., the application crashes on start-up), testing must be suspended until the issue is resolved. In this case, the tester will need to reevaluate all testing criteria prior to the discovery of the S0 bug, since a new release of the software will be needed containing the fix.

# Proposed Test Cases

### Test Case 01: QR Code Reading from Mail Piece

**Description:** In this test, the user will open a digest with a single mail piece that has a QR code with a URL embedded in it and the application will navigate to the correct page.

**Requirements:** This test case validates that the application can read a QR code with a URL in it and make it selectable.

**Prerequisites:** A mail digest with mail containing a QR code with a valid link is received.

**Steps:**

1. The customer opens the email digest message with the previous mail piece in it.
2. The user clicks on the link from the QR code.

**Expected output:** The application opens the browser and navigates the user to that page.

### Test Case 02: Embedded Links Navigate User to the Browser

**Description:** In this test, the user takes an email that has links in the body, and will be able to navigate to those links.

**Requirements:** This test case validates that all links will be linkifed in the body of the message and the user will be able to open the browser for the link.

**Prerequisites:** A email with a valid link is received.

**Steps:**

1. The customer opens a piece of email that contains a link.
2. The user clicks on the link

**Expected output:** The application opens the browser and navigates the user to that link.

### Test Case 03: Google Assistant Search for an Mailpiece

**Description:** In this test, the user will search for a mail piece using Google Assistant and will not provide a date range.

**Requirements:** This test case validates that the user can search for an email or mailpiece without a date range via Google Assistant

**Prerequisites:** The application is running on an Android device and the device has Google Assistant installed.

**Steps:**

1. The customer says "Ok Google Search for Test Mailpiece On Mailspeak"

**Expected output:** The application opens the search page with Test Mailpiece filled in.

### Test Case 04: Google Assistant Add a new Sender/Keyword for Notifications

**Description:** In this test, the user will add a Sender/Keyword to receive notifications for it via Google Assistant.

**Requirements:** This test case validates that the user can add a Sender/Keyword to the notifications list via Google Assistant

**Prerequisites:** The application is running on an Android device and the device has Google Assistant installed.

**Steps:**

1. The customer says, "Ok, Google Add Plumbing to MailSpeak."

**Expected output:** The application opens the notifications page and adds the Plumbing keyword to the Sender/Keyword list

### Test Case 05: Google Assistant Add an existing Sender/Keyword for Notifications

**Description:** In this test, the user will add a Sender/Keyword that already exists to the notifications list via Google Assistant.

**Requirements:** This test case validates that a duplicate item will not be set if a sender/keyword is attempted to be added where it already exists.

**Prerequisites:** The application is running on an Android device, the device has Google Assistant installed, and the keyword "Plumbing" is already added to the notifications list.

**Steps:**

1. The customer says, "Ok, Google Add Plumbing to MailSpeak."

**Expected output:** The application does not add plumping a second time

### Test Case 06: Google Assistant Opens the most recent Digest email

**Description:** In this test, the user will open the most recent digest mail using a voice command on an Android device.

**Requirements:** This test case validates that the user will be able to open the most recent digest email using Google Assistant.

**Prerequisites:** The application is running on an Android device and the device has Google Assistant Installed.

**Steps:**

1. The customer says, "Ok, Google Get Digest on MailSpeak.

**Expected output:** The application opens the mail digest page and shows the most recent digest message.

### Test Case 07: Google Assistant Opens the notifications page

**Description:** In this test, the user will open the notifications page

**Requirements:** This test case validates that the user will be able to open the notifications page using Google Assistant.

**Prerequisites:** The application is running on an Android device and the device has Google Assistant Installed.

**Steps:**

1. The customer says, "Ok, Google Open Notifications On MailSpeak.

**Expected output:** The application opens the manage notifications page

### Test Case 08: Google Assistant Opens Application

**Description:** In this test, the user will open the app using Google Assistant

**Requirements:** This test case validates that the user will be able to open the application via Google Assistant

**Prerequisites:** The application is running on an Android device and the device has Google Assistant Installed.

**Steps:**

1. The customer says, "Ok, Google Open MailSpeak

**Expected output:** Google Assistant opens the application.

### Test case 09: Add Sender/Keyword Notification list visually/manually

**Description:** In this test, the user will visually/manually add a Specific sender/Keyword to receive a notification.

**Requirements:** This test case validates that the user can add a Sender/Keyword to the notifications list manually/visually.

**Prerequisites:** Application running on Android device and app is logged in with customer's email address.

**Steps:**

1. The Customer selects the Notification button from the home page
2. The Customer selects the 'Manage' button under 'Notifications Center.'
3. The Customer then selects the 'Add' button under the 'Manage' section
4. The Customer types in the specific sender/keyword they want to be notified for
5. The Customer clicks on the save button next to the text box specified sender information

**Expected output:** The application adds the typed and saved sender/keyword into the notification list.

### Test case 10: Delete Sender/Keyword Notification list visually/manually

**Description:** In this test, the user will visually/manually Delete a Specific Sender/Keyword from the existing notification lists.

**Requirements:** This test case validates that the user can delete a previously existing Sender/Keyword from the notifications list manually/visually.

**Prerequisites:** Application running on Android device and app is logged in with customer's email address.

**Steps:**

1. The Customer selects the Notification icon from the home page
2. The Customer selects the Manage button
3. The Customer then selects the clear button next to the item to be deleted.

**Expected output:** The application deletes the sender/keyword next to the pushed 'clear' button. It is no longer on the notification list.

### Test case 11: Receive Notification

**Description:** In this test, a sample email will be sent to the user's phone with a sender id from within the notification list.

**Requirements:** This test case validates the application notifying the user as any of the senders/keywords in the notification list to arrive.

**Prerequisites:** Application running, user logged in to the app with their email address, and there is a minimum of one notification list set.

**Steps:**

1. A test email is sent to the Customer from a sender's name within the notification list.

**Expected Output:** The app sends notifications as the user receives an email from a sender/keyword form within the notification list.

### Test case 12: View Notification visually/manually

**Description:** In this test, the user will be viewing an email for which the user received a notification.

**Requirements:** This test case validates the notification properly, opens the email notified as the user clicks on the notification.

**Prerequisites:** Application running, user login to the app with their email address and an email received from a sender that is in the notification list.

**Steps:**

1. A test email is sent to the Customer from a sender's name within the notification list.
2. A notification alert from the application
3. The Customer clicks on the popped-up notification

**Expected Output:** A click on the popped-up notification opens the email, and the customer can go through the email.

### Test case 13: Clear Notification

**Description:** In this test, the user will be deleting a specific notification from the existing notification lists.

**Requirements:** This test case validates that the user can delete a previously existing notifications from the notifications list manually/visually.

**Prerequisites:** Application running, user login to the app with their email address and an email received from a sender that is in the notification list.

**Steps:**

1. The Customer selects the Notification icon from the home page
2. The Customer selects the Notifications button
3. The Customer then selects the clear button next to the item to be deleted.

**Expected output:** The application deletes the sender/keyword next to the pushed 'clear' button. It is no longer on the notification list.

### Test case 14: Clear All Notifications

**Description:** In this test, the user will be deleting all the notifications from the existing notification lists.

**Requirements:** This test case validates that the user can delete all notifications from the notifications list manually/visually.

**Prerequisites:** Application running, user login to the app with their email address and an email received from a sender that is in the notification list.

**Steps:**

1. The Customer selects the Notification icon from the home page
2. The Customer selects the Notifications button
3. The Customer then selects the clear all button at the top of the notification list

**Expected output:** The application deletes all notifications, and the notification list will be empty

### Test case 15: Display QR code and click Mail View able link

**Description:** In this test, the user will view their mail and see a generated URL link from a QR code that will redirect them to the appropriate page.

**Requirements:** This test case validates that the OCR is able to properly extract a URL from the image of the mail, as well as properly generate an actionable link that the user can click on.

**Prerequisites:** Application running, user login to the app with their email address, and the OCR has cached all the potential data for the user to work with.

**Steps:**

1. Test mail is sent with a QR code embedded in an image.
2. The user opens the application. (Back-end should trigger OCR and caching process)
3. The user opens the mail and sees their mail and our transcription with an actionable link.
4. The user clicks on the link, and we confirm this is the correct location.

**Expected Output:** When the user opens the mail view page, there is a generated link from a QR code in the text we insert. Upon clicking the URL, it sends the user to the appropriate webpage.

### Test case 16: Mail View Display URL and a clickable link

**Description:** In this test, the user will be viewing their mail and see a generated URL link from a URL that will redirect them to the appropriate page.

**Requirements:** This test case validates that the OCR is able to properly extract a URL from the image of the mail, as well as properly generate an actionable link that the user can click on.

**Prerequisites:** Application running, user login to the app with their email address, and the OCR has cached all the potential data for the user to work with.

**Steps:**

1. Test mail is sent with a URL embedded in an image
2. The user opens the application. (Back-end should trigger OCR and caching process)
3. The user opens the mail and sees their mail and our transcription with an actionable link.
4. The user clicks on the link, and we confirm this is the correct location.

**Expected Output:** When the user opens the mail view page, there is a generated link from a URL in the text we insert. Upon clicking the URL, it sends the user to the appropriate webpage.

### Test case 17: Mail View Display text from mail

**Description:** In this test, the user will be viewing their mail and see the content of their mail transcribed.

**Requirements:** This test case validates that the OCR can properly extract the text content of the mail.

**Prerequisites:** Application running, user login to the app with their email address, and the OCR has cached all the potential data for the user to work with.

**Steps:**

1. Test mail is sent with text content.
2. The user opens the application. (Back-end should trigger OCR and caching process)
3. The user opens the mail and sees their mail and our transcription of the text content.

**Expected Output:** When the user opens the mail view page, there is a generated transcription of the text content. The text matches exactly what is in the mail.

### Test case 18: Mail View Display verify contact information

**Description:** In this test, the user will be viewing their mail and see the content of their mail transcribed. Specifically contact information (email, phone number).

**Requirements:** This test case validates that the OCR is able to properly extract the text content of the mail.

**Prerequisites:** Application running, user login to the app with their email address, and the OCR has cached all the potential data for the user to work with.

**Steps:**

1. Test mail is sent with text content.
2. The user opens the application. (Back-end should trigger OCR and caching process)
3. The user opens the mail and sees their mail and our transcription of the text content.
4. The user checks for contact information.

**Expected Output:** When the user opens the mail view page, there is a generated transcription of the text content. The text displays contact information.

### Test case 19: Analytics Dashboard confirms screen time

**Description:** In this test, the user will start a new session, view the Mail View screen for 3 minutes, and then view the analytics dashboard.

**Requirements:** This test case validates that session times are being sent to an analytics aggregator.

**Prerequisites:** The analytics server is running.

**Steps:**

1. The user logs into the application, making a note of the current date and time.
2. After three minutes, the user logs out of the application.
3. There is a delay in analytics reporting, so the user will need to wait at least an hour.
4. The user logs into the analytics dashboard and confirms that the three minutes of session time is displayed.

**Expected Output:** The user's session time is reported to the analytics server.

### Test case 20: Analytics Dashboard displays link metrics

**Description:** In this test, the user will open a link and confirm that data was transmitted to the analytics server.

**Requirements:** This test case validates that link data is being sent to an analytics server.

**Prerequisites:** The analytics server is running.

**Steps:**

1. The user logs into the application.
2. The user opens the Mail View screen and opens an email containing a link.
3. The user clicks on the link, and the URL is opened in a web browser.
4. There is a delay in analytics reporting, so the user will need to wait at least an hour.
5. The user logs into the analytics dashboard and confirms that the opened link data is present.

**Expected Output:** The user's selected links are reported to the analytics server.

### Test case 21: Analytics Dashboard displays searches performed

**Description:** In this test, the user will perform a search and confirm that data was transmitted to the analytics server.

**Requirements:** This test case validates that search data is being sent to an analytics server.

**Prerequisites:** The analytics server is running.

**Steps:**

1. The user logs into the application.
2. The user opens the Search screen and performs a keyword search.
3. There is a delay in analytics reporting, so the user will need to wait at least an hour.
4. The user logs into the analytics dashboard and confirms that the keyword search data is present.

**Expected Output:** The user's keyword search is reported to the analytics server.

### Test case 22: Mail View Call Phone Number

**Description:** In this test, the user will view their mail and can open the phone for a given mail piece with a number listed.

**Requirements:** This test case validates that the application can call a phone number from the mail screen.

**Prerequisites:** Application running, user login to the app with their email address, and the OCR has cached all the potential data for the user to work with. The given phone number is in the mail view

**Steps:**

1. Test mail is sent with text content and includes contact information.
2. The user opens the application.
3. The user opens the mail and sees their mail and our transcription of the text content.
4. The phone number is available, and the user selects it.
5. The application asks if the user would like to call or text, select call

**Expected output:** The application opens the device-specific dialer for the given phone number.

### Test case 23: Mail View Text Phone Number

**Description:** In this test, the user will be viewing their mail and can open the text application for a given mail piece with a number listed.

**Requirements:** This test case validates that the application can text a phone number from the mail screen.

**Prerequisites:** Application running, user login to the app with their email address, and the OCR has cached all the potential data for the user to work with. The given phone number is in the mail view

**Steps:**

1. Test mail is sent with text content and includes contact information.
2. The user opens the application.
3. The user opens the mail and sees their mail and our transcription of the text content.
4. The phone number is available, and the user selects it.
5. The application asks if the user would like to call or text, select text

**Expected output:** The application opens the device-specific text application for the given phone number.

### Test case 24: Mail View Email provided email address.

**Description:** In this test, the user will be viewing their mail and will be able to open their default email application.

**Requirements:** This test case validates the application can email and email address from the mail screen.

**Prerequisites:** Application running, user login to the app with their email address, and the OCR has cached all the potential data for the user to work with. The given email address is in the mail view.

**Steps:**

1. Test mail is sent with text content and includes contact information.
2. The user opens the application.
3. The user opens the mail and sees their mail and our transcription of the text content.
4. The email address is available, and the user selects it.

**Expected output:** The application opens the device-specific email application for the given email address.

# Assumptions and Constraints

## Assumptions

The following is a set of assumptions, along with their associated owners, that the project will respect for testing the application:

|  |  |
| --- | --- |
| **Assumptions** | **Owner** |
| The team will have access to an email account that contains representative emails for USPS Informed Delivery emails. | Project Manager and Project Owner |
| The team will have funding to exercise the use of paid services through the Google Cloud Platform and the Apple App Store. | Project Manager and Project Owner |
| The team will have access to both Android and iOS devices to test the software on both operating systems. | Development Team |
| The users will only interact with the system in the English language. | Project Manager and Project Owner |

Table 3 Project Assumptions

## Constraints

Similarly, the following is a set of constraints that must be acknowledged when assessing the delivered software with respect to testing:

|  |  |
| --- | --- |
| **Constraints** | **How to Address** |
| The testing timeline may be compressed due to class-level schedule constraints. This could result in a more compressed User Acceptance testing timeline and may result in fewer bugs being found. | Ensure the team finishes all features on time and allow for an acceptable amount of testing before the software delivery. |
| The ability for a single team to re-architecture the application is impossible due to the coordination needed between both teams. As such, the team must not make significant structural changes that will interfere with the other team's ability to complete their assignment. | All cross-cutting changes should be clearly documented and confirmed with the other team. |

Table 4 Project Constraints

# Deliverables

Deliverables for test concerns are delivered before, during, and after testing. Specific deliverables are intended for customer consumption, while others are delivered internally to the team to support project completion.

**Deliverables before testing**

* Test Plan Documentation: Delivered to customer
* Test Design Documentation: Delivered to customer

**Deliverables during testing**

* Test script documentation: Delivered to software engineers and customer
* Simulators and/or emulators: Delivered to software engineer
* Test data: Delivered to software engineer performing tests
* Error and execution logs: Delivered to software engineer

**Deliverables after testing**

* Test results: Delivered to customer
* Defect reports: Delivered internally to the team
* Release notes: Delivered to App Stores after release

# Appendices

## Definitions, Acronyms, Abbreviations

|  |  |
| --- | --- |
| Definitions, Acronyms, Abbreviations | Definition |
| API | Application Programming Interface |
| GCP | Google Cloud Platform |
| QA | Quality Assurance |
| QR | Quick Response |
| RTM | Requirement Traceability Matrix |
| STP | Software Test Plan |
| UI | User Interface |
| USPS | United States Postal Service |

Table 5 Definitions, Acronyms, Abbreviations

## References

"How to Write Test Strategy Document (With Sample Test Strategy Template)". (n.d). *softwaretestinghelp.com.* Retrieved from <https://www.softwaretestinghelp.com/wp-content/qa/uploads/2012/12/STLC.jpg>

TeamArch-TTD for Reference: <https://umgcdev361.sharepoint.com/:w:/r/sites/SWEN670Fall2022/Shared%20Documents/General/Previous%20Class%20Docs%20(For%20Reference)/TDD/TeamArch-Technical-Design-Document.docx?d=wdac92747283044cc882b98e477d3abe3&csf=1&web=1&e=aThrxL>

1. From "How to Write Test Strategy Document” (With Sample Test Strategy Template) SoftwareTestingHelp.com, n.d. (https://www.softwaretestinghelp.com/writing-test-strategy-document-template/) [↑](#footnote-ref-2)