SE 3XA3: Test Plan
SaveTheDate

Group 17
Karuka Khurana (khurak1)
Utsharga Rozario (rozariou)
Samarth Kumar (kumars38)
Dhruv Cheemakurti (cheemakd)

April 12, 2022

Contents

1 General Information **1**
- 1.1 Purpose . . . 1
- 1.2 Scope . . . 1
- 1.3 Acronyms, Abbreviations, and Symbols . . . 2
- 1.4 Overview of Document . . . 4

2 Plan **4**
- 2.1 Software Description . . . 4
- 2.2 Test Team . . . 4
- 2.3 Automated Testing Approach . . . 5
- 2.4 Testing Tools . . . 5
- 2.5 Testing Schedule . . . 5

3 System Test Description **5**
- 3.1 Tests for Functional Requirements . . . 5
  - 3.1.1 Scraping PDF Tests . . . 5
  - 3.1.2 Uploading Image from local system to Notion clone . . 9
  - 3.1.3 All functionalities are accessiable . . . 10
- 3.2 Tests for Nonfunctional Requirements . . . 10
  - 3.2.1 Look and Feel Requirements . . . 10
  - 3.2.2 Usability and Humanity Requirements . . . 11
  - 3.2.3 Performance Requirements . . . 12
  - 3.2.4 Operational and Environmental Requirements . . . 13
  - 3.2.5 Release Requirements . . . 14
  - 3.2.6 Maintainability and Support Requirements . . . 14
  - 3.2.7 Cultural Requirements . . . 16
- 3.3 Traceability Between Test Cases and Requirements . . . 17

4 Tests for Proof of Concept **18**
- 4.1 Empty Table Generation . . . 18
- 4.2 PDF Scraping . . . 18

5 Comparison to Existing Implementation **19**

6 Unit Testing Plan **19**
- 6.1 Unit testing of internal functions . . . 19
- 6.2 Unit testing of output files . . . 20

7 Appendix **21**
7.1 Symbolic Parameters 21
7.2 Usability Survey Questions? 21

List of Tables

1 **Revision History** 1
2 **Table of Abbreviations** 2
3 **Table of Definitions** 3
4 Traceability Matrix: Functional Requirement 17
5 Traceability Matrix: Non-Functional Requirement 17

Table 1: **Revision History**

Date	**Version**	**Notes**
March 9	1.0	General Info and Plan by Karuka FR test by Samarth NFR tests by Utsharga
March 11	1.1	Appendix by Karuka Traceability Matrix By Utsharga Test for POC, Comparison and Unit Testing Plan by Dhruv
Apr 12	1.2	Rev 1: Updates of tests/traceability to match updated SRS by Samarth

1 General Information

1.1 Purpose

The purpose of this document is to outline the testing, validation, and verification process of the functional and non-functional requirements, for the SaveTheDate project. The test cases outlined in this document have been created prior to implementation and will therefore act as a guide for future the development and testing process. The test cases have been chosen to ensure that the modules created are functioning as expected.

1.2 Scope

The scope of the test plan outlined in this document is designed to test the functionality of the SaveTheDate application. To fully test SaveTheDate, we will test performance at a modular level by creating unit tests and at a higher level by using exploratory testing and specification-based testing.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
FR	functional requirements
NFR	non-functional requirements
POC	proof of concept
SRS	Software requirements specification
APP	application

Table 3: **Table of Definitions**

Term	**Definition**
Notion	An organization and project management tool
Repo	Gitlab Repository
Pytest	A unit testing framework for Python
Cypress	Used to test modern Javascript Frameworks
Mocha	Javascript test framework for Node.js programs
Unit Testing	A method of testing focused on testing individual methods and functions
Acceptance Testing	A method of testing which is conducted to determine if the requirements of the specification are met
Boundary Testing	A method of testing where values are chosen on semantically significant boundaries
Dynamic Testing	A method of testing where code is executed
Specification-Based Testing	A method of testing where test cases are built based on requirements specification
Exploratory Testing	A method of testing where the tester simultaneously learns the code while testing it. It approaches testing from a user's point of view.
Integration Testing	A method of testing where the individual software modules are combined and tested as a group
Partition Testing	A method of testing where the input domain is partitioned, and input values are selected from the partitions
Fuzz Testing	A method of testing where random inputs are given to attempt to violate assertions
Static Testing	A method of testing where code is not executed
System Testing	A method of testing where the tests are performed on the system as a whole

1.4 Overview of Document

This document outlines the test plan which includes all the requirements and tests needed to verify correctness and validation of the SaveTheDate application. It contains relevant information on the different types of tests that will be conducted and why. This includes testing tools, testing schedule, and test cases.

2 Plan

2.1 Software Description

Organization is a big part of a student's success and allows one to prioritize activities, set and achieve goals and reduce stress. Notion is a widely used organization and management tool with a variety of features. The open-source Notion project is an agile framework that relies heavily on customer involvement and satisfaction. With Notion encouraging organization and effective visualization, SaveTheDate provides a solution that eliminates the need for user's to manually update their due dates and deadlines and instead provides an automated process with beneficial visualization of what's to come in the user's calendar. SaveTheDate will add features including uploading a PDF, providing a page range the user would want the application to scrape, a scrape option and finally a dashboard with tables organized by due dates for the user to easily read and interact with. The user will be given the option to scrape multiple PDFs and add the output to the same dashboard to allow for information to be centralized and to promote organization.

2.2 Test Team

The team is split into 2 sub teams: backend and frontend. Each sub team will work towards testing their respective modules. It will be divided as such:

Karuka Khurana and Dhruv Cheemakurti: Front End

Samarth Kumar and Utsharga Rozario: Back End

2.3 Automated Testing Approach

Testing will mostly be automated. Each module created will have an associated test class which will be committed to the group's GitLab repository. These tests will run every time something is committed to the repo and can be updated when needed if more edge cases are found.

2.4 Testing Tools

The unit tests will be written using the Pytest Framework as well as Mocha and Cypress. Manual testing will also be done for tests that cannot be automated. This will be largely required for verifying visuals of the Notion page.

2.5 Testing Schedule

Attached is the Gantt chart that outlines the group's testing schedule.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Scraping PDF Tests

Begin scraping button

1. FR-ST-1

   Type: Functional, Dynamic, Manual

   Initial State: Notion page has a PDF document uploaded

   Input: A "Begin Scraping" button is pressed using the mouse

   Output: The scraping process should begin

   How test will be performed: Dynamic, manual testing is performed by pressing the scraping button and ensuring that the scraping process begins.

Page range

1. ~~FR-ST-2~~

   ~~Type: Functional, Dynamic, Automated, Unit~~

   ~~Initial State: Notion page has a PDF document uploaded~~

   ~~Input: The page range field is filled, ex. "1-10"~~

   ~~Output: The scraping process should only occur over the specified pages of the PDF document~~

   ~~Exception: An error message stating "Invalid page range" should be provided if the page range entered is invalid. For example, negative values, fractional/decimal values, or if the first page number is greater than the last number.~~

   ~~How test will be performed: Unit testing is performed by supplying various page range inputs over multiple PDFs that have known scraping outputs per page. Then, outputs are compared to the known outputs to ensure that scraping only occurred over the specified range.~~

   FR-ST-2 test case removed due to removal of page range requirement (SRS).

Identifying structured deadlines

1. FR-ST-3

   Type: Functional, Dynamic, Automated, Unit

   Input: The PDF document

   Output: All structured (tabular) deadlines are found in the form of a 2D list

   How test will be performed: Unit testing is performed by inputting multiple PDF cases into the python scraper, with varying dates and tasks. The expected results are found via manual inspection of the PDF for deadline tables. The scraper outputs are then compared to the expected results in each unit test to ensure all deadlines are found.

Structuring the output (Backend to frontend testing)

1. FR-ST-4

   Type: Functional, Dynamic, Automated, Unit

   Input: 2D list of deadlines and corresponding tasks from the python backend

   Output: Structured data in the frontend (React)

   How test will be performed: Unit testing is performed where the inputs are 2D lists of deadlines and tasks. Expected results in the frontend (React) are compared to the actual results.

Empty notion table creation

1. FR-ST-5

   Type: Functional, Dynamic, Manual

   Input: Dimensions and column headings for the table

   Output: Notion table

   How test will be performed: Manual testing will be used to verify that a table can be created in Notion, and that it is visually correct based on the dimensions and column headings provided.

Notion table creation with deadlines

1. FR-ST-6

   Type: Functional, Dynamic, Manual, ~~Automated~~

   Input: Structured data in the frontend (React)

   Output: Notion table with tasks and their deadlines provided

   How test will be performed: ~~Automated testing using Mocha.js is used to verify that table fields match their expected values based on the deadline/task data.~~ Manual testing will be used to verify that the tables are visually correct.

   Removal of automated testing in FR-ST-6 since we are no longer using Mocha.js.

Editing a table

1. FR-ST-7

   Type: Functional, Dynamic, Manual

   Initial state: Notion page has a table created

   Input: Editing table cells

   Output: Changed table

   How test will be performed: Manual testing is used to verify that making changes to a table cell reflect in the actual table (it is editable and not static).

Course code

1. FR-ST-8

   Type: Functional, Dynamic, ~~Automated, Unit,~~ Manual

   Initial state: Notion page has a PDF uploaded

   Input: Course name/code field

   Output: Generated table title

   Exception: ~~An error message will be provided if the course name/code has invalid characters, is inappropriate, or exceeds a certain length~~

   How test will be performed: ~~Unit testing is used to verify that the supplied course names and codes become the title for the generated table, and exceptions are raised appropriately~~. Manual testing is used to verify editing the course field on the Notion page itself.

   Automated testing and exceptions no longer tested. Manual testing still plans to be used.

Uploading PDF

1. FR-ST-9

   Type: Functional, Dynamic, Manual

Initial state: Notion page has no PDF uploaded

Input: An "Upload PDF" button

Output: File explorer window opens

How test will be performed: Manual testing is used to verify that the upload button is functional from the Notion page.

Selecting PDF from files

1. FR-ST-10

   Type: Functional, Dynamic, Manual, Automated

   Input: A file selected via a file explorer window

   Output: PDF file is uploaded to the Notion page

   ~~Exception: An error message is provided if the file does not end in .pdf.~~

   How test will be performed: Automated testing is first used to check whether various PDFs can be uploaded using files from the repository. Manual testing is further used to verify that various PDF files can be selected using the explorer, ~~and non-PDF files raise an exception~~.

   Not testing for invalid file extensions.

3.1.2 Uploading Image from local system to Notion clone

Uploading Image

1. FR-ST-11

   Type: Functional, Dynamic, Manual, Automated

   Initial state: Notion page has no image uploaded

   Input: An "Upload Image" button

   Output: File explorer window opens, image can be selected and uploaded

   ~~Exception: An error message is provided if the format of the file is not an image extension (.png, .jpg, etc.)~~

How test will be performed: Automated testing is used to check whether various images can first be uploaded to the page using a file from the repository (/img/...). Manual testing is then used to verify that the upload image button is functional from the Notion page.

3.1.3 All functionalities are accessable

Button with available functionalities

1. ~~FR-ST-12~~

   ~~Type: Functional, Dynamic, Manual~~

   ~~Input: Right-clicking on Notion line~~

   ~~Output: Menu with the various buttons (upload, scrape, etc.) should appear~~

   ~~How test will be performed: Manual testing is used to verify that menu appears and has all options present. Ensure that "Begin Scraping" button is unavailable until a PDF is uploaded.~~

   FR-ST-12 is no longer a relevant test case after changes in the SRS.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel Requirements

Appearance Requirements

1. NFR-LF-1

   Type: Manual, Dynamic, Checklist

   How test will be performed: The test team will run commands in the Notion SaveTheDate application and check if the PDFs, Table, and Image are returned in accordance with the Notion theme.

Style Requirements

1. NFR-LF-2

Type: Manual, Usability Survey

How test will be performed: The test team will run commands in the Notion SaveTheDate application and check if the PDFs, Table, and Image are returned in accordance with the Notion font and letter sizing.

3.2.2 Usability and Humanity Requirements

Ease of use Requirements

1. NFR-UH-1

   Type: Manual, Usability Survey

   How test will be performed: The test team will provide a questionnaire asking the user to input their age and then ask to follow up questions about their level of difficulty using the application. The following options will be provided to let the user decide their experience: intimidating, complicated, neutral, or excellent. The fit criterion is if 80% of the users who are 13+ years old voted neutral or excellent, then the test has passed and NFR has been satisfied.

2. NFR-UH-2

   Type: Manual, Usability Survey

   How test will be performed: The test team will provide a questionnaire asking the user to input the average time it takes for them to transition between tasks.

3. NFR-UH-3

   Type: Manual, Usability Survey

   How test will be performed: The test team will provide a questionnaire asking the user if the new components feel coherent to the application. The following options will be provided to let the user decide their experience: ‘Yes, all the components feel coherent’, ‘some components feel out of place’, and ‘no none of the components are coherent’, followed by the areas that do not seem coherent.

3.2.3 Performance Requirements

Speed Requirements

1. NFR-PR-1

   Type: Automated, Dynamic, UnitTest

   Initial State: PDF to scrape exists in test directory

   Input/Condition: set of PDFs

   Output/Result: Outputs the table data from the PDF in the command line

   How test will be performed: The test team will run automated tests using Python Unit test module where the system will keep track of the execution time ensure that it is less than 10 seconds to pass.

2. NFR-PR-2

   Type: Automated, Dynamic

   Initial State: Table to be used as input exits in test directory

   Input/Condition: set of table data

   Output/Result: Outputs the table data as a Notion Component

   How test will be performed: The test team will run automated tests using React test suite where the system will keep track of the execution time ensure that it is less than 10 seconds to pass.

Precision or Accuracy Requirements

1. NFR-PR-3

   Type: Automated, Dynamic

   Initial State: PDF to scrape exists in test directory

   Input/Condition: set of PDFs

   Output/Result: Outputs all the deadline tables and its data in the command line

How test will be performed: The test team will run automated tests using Python Unit test methods where the system will run assertions to check if all the existing deadline tables have been scraped with an 85% accuracy.

2. NFR-PR-4

   Type: Automated, Dynamic

   Initial State: Table to be used as input exits in test directory

   Input/Condition: set of table data

   Output/Result: Outputs all the deadline tables and its data as a notion table

   How test will be performed: The test team will run automated tests using React Unit test suite where the system will run assertions to check if all the table have been created with an 90% accuracy.

Longevity Requirements

1. NFR-PR-5

   Type: Automated, Dynamic

   Initial State: The React application is running

   Input/Condition: All dependencies are up to date

   Output/Result: All key functionality is working

   How test will be performed: The test team will run all automated tests using React and Python Unit test methods where the system will check if all the tests are passing with the specified requirements.

3.2.4 Operational and Environmental Requirements

Expected Physical Environment

1. NFR-OE-1

   Type: Manual, Dynamic

The test team will manually execute tasks on Notion SaveTheDate with the internet disabled to see if errors arise as expected. The team will then check if the program works once the internet is turned back on. If so, then test case passed.

3.2.5 Release Requirements

1. NFR-RR-1

   Type: Manual, static, code inspection

   Initial state: Program deployed to users in production

   Input: Users provide feedback and feature requests

   Output: Development team maintains feedback via a Gitlab issue tracker.

   How test will be performed: The testing team will keep track of feature requests via Gitlab monthly. Additionally, the testing team will conduct code inspections to identify the faults for bug-fixes.

3.2.6 Maintainability and Support Requirements

Maintenance Requirements

1. NFR-MA-1

   Type: Manual, Static

   Initial state: N/A

   Input: N/A

   Output: The code is documented accurately with comments

   How test will be performed: The code will be inspected by the team to determine if it has been accurately documented. The documentation will be manually inspected to ensure all aspects of the code have been documented.

2. NFR-MA-2

   Type: Manual, Static

Initial state: N/A

Input: N/A

Output: The code is commented sufficiently, and the formatting is consistent

How test will be performed: The code will be inspected by the team to determine if it has been commented sufficiently and the formatting is consistent. The source code will be manually inspected to ensure all aspects of the code have been commented and is consistent with standard styling.

Supportability Requirements

1. NFR-MA-3

   Type: Manual, Dynamic

   Initial state: Program deployed to users in GitLab and made public

   Input: Users provide feedback and raise issues

   Output: List of all the commands is displayed for the user

   How test will be performed: The testing team will keep track of raised issues via Gitlab monthly and attempt to resolve them.

Adaptability Requirements

1. NFR-MA-4

   Type: Automated, Dynamic

   Initial state: Program is running on different Operating Systems

   Input: N/A

   Output: N/A

   How test will be performed: The automated testing is run on different Operating Systems. If all the automated tests pass, if so, this test will have passed.

2. NFR-MA-5

   Type: Automated, Dynamic

   Initial state: Program is running on different Internet Browser

   Input: N/A

   Output: N/A

   How test will be performed: The automated testing is run on different Internet Browsers. If all the automated tests pass, if so, this test will have passed.

3.2.7 Cultural Requirements

1. ~~NFR-CR-1~~

   ~~Type: Automated, Dynamic~~

   ~~Initial state: The Notion SaveTheDate is running~~

   ~~Input: set of banned inappropriate words~~

   ~~Output: Prompt showing input is inappropriate~~

   ~~How test will be performed: The test team will run automated tests using React Unit test suite where the system will run assertions to check if all the banned words are not accepted as a course name.~~

2. ~~NFR-CR-2~~

   ~~Type: Automated, Dynamic~~

   ~~Initial state: The Notion SaveTheDate is running~~

   ~~Input: set of different languages that do not use English Language~~

   ~~Output: Prompt showing input is not in English~~

   ~~How test will be performed: The test team will run automated tests using React Unit test suite where the system will run assertions to check if different languages are not accepted as a course name.~~

   NFR-CR-1 and NFR-CR-2 are no longer relevant test cases due to changes in the SRS (not submitting course name).

3.3 Traceability Between Test Cases and Requirements

Test IDs	Requirement IDs																			
	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	FR13	FR14	FR15	FR16	FR17	FR18	FR19	FR20
FR-ST-1	X																			
~~FR-ST-2~~																				
FR-ST-3			X																	
FR-ST-4				X																
FR-ST-5					X															
FR-ST-6						X														
FR-ST-7							X					X	X	X	X					
FR-ST-8								X												
FR-ST-9		X							X									X	X	
FR-ST-10										X						X	X			
FR-ST-11											X									X
~~FR-ST-12~~																				

Table 4: Traceability Matrix: Functional Requirement

Test IDs	Requirement IDs																	
	LF1	LF2	UH1	UH2	UH3	PE1	PE2	PE3	PE4	PE5	OE1	RR1	MA1	MA2	MA3	MA4	MA5	~~CR1~~	~~CR2~~
NFR-LF-1	X																		
NFR-LF-2		X																	
NFR-UH-1			X																
NFR-UH-2				X															
NFR-UH-3					X														
NFR-PE-1						X													
NFR-PE-2							X												
NFR-PE-3								X											
NFR-PE-4									X										
NFR-PE-5										X									
NFR-OE-1											X								
NFR-RR-1												X							
NFR-MA-1													X						
NFR-MA-2														X					
NFR-MA-3															X				
NFR-MA-4																X			
NFR-MA-5																	X		
~~NFR-CR-1~~																			
~~NFR-CR-2~~																			

Table 5: Traceability Matrix: Non-Functional Requirement

4 Tests for Proof of Concept

4.1 Empty Table Generation

Creating an empty Notion Table

1. test-POC-1

   Type: Functional, Dynamic and Manual

   Initial State: PDF Scraping completed

   Input: Dimensions and column headings

   Output: Notion table

   How test will be performed: Based on the input of column headings and dimensions, we can assess whether the table is correctly generated from a visual standpoint. Manual testing can also be employed to determine whether a table can be made in Notion.

Creating a Notion table with the corresponding deadlines and tasks

1. test-POC-2

   Type: Functional, Dynamic, Manual and Automated

   Initial State: PDF Scarping completed with the 2D list ready to be inputted.

   Input: Structured data in the form of a 2D list in the front end (React)

   Output: Notion table with the tasks and deadlines outlined in it

   How the test will be performed: Automated testing will be employed using Mocha.js so that the table fields match their expected value. Manual testing will also be used to visually verify that the table creation is correct.

4.2 PDF Scraping

Identifying structured deadlines

1. test-POC-3

   Type: Functional. Dynamic, Automated, Unit

   Initial State: N/A

   Input: PDF

   Output: All structured deadlines in the form of a 2D list

   How test will be performed: Unit testing will be used to input multiple PDFs into the scraper. The output should be compared to the PDF by manual inspection to ensure that the PDF was scraped correctly.

5 Comparison to Existing Implementation

When comparing the current product (SaveTheDate) with the original product that Notion offers, there are a few differences. The layout and the visual user interface is identical to Notion. Both products use the feature that gives the user the option to write in a Heading, Paragraph etc. Since our application uses a Notion framework, the code structure and layout are fairly similar because it still has the option for tables, pages etc. The difference between our application and Notion's is that there will be a button for table generation. This button will run the backend processes to scrape a PDF of the user's choice. The scraped data will generate a table with all the relevant information from the PDF. There are multiple functions that are involved in order to scrape the PDF that will be tested for to ensure it output as intended.

6 Unit Testing Plan

Unit testing will be performed using the Pytest Framework including Mocha and Cypress

6.1 Unit testing of internal functions

Unit testing for internal functions will be done for each function to enforce reliability and robustness. The tests include boundary and partition testing to catch errors, this is to avoid a potential system crash. The purpose of unit testing is to ensure that our final version of the product will behave as it was originally intended and designed. The unit testing will use normal cases, boundary cases and abnormal ones that purposefully result in error states. Moreover, integration testing will also be implemented after unit testing to verify that the system's output and its communication between functions and modules match what was outlined in the SRS. As a team, we have collectively decided to aim for 85% for test coverage.

6.2 Unit testing of output files

The nature of our product does not produce any output files, as the PDF scraped table will only be added on to the original page of Notion. Therefore, unit testing for output files is not required.

7 Appendix

7.1 Symbolic Parameters

N/A

7.2 Usability Survey Questions?

These survey questions will be used to test usability requirements.

1. Would you use SaveTheDate as your main organization tool for due dates (Y/N)?
2. Are the SaveTheDate commands easy to use?
3. Is the process (from uploading the pdf to receiving the outputted table) intuitive? If not, why?
4. On average, how long does the process to receive a summary of your due dates take? (5 seconds, 10 seconds, etc.)
5. Rate the benefit of the application as “good”, “neutral”, or “bad”.
6. Rate your experience using the application. (Intimidating, complicated, neutral, or excellent)