SE 3XA3: Test Report
SaveTheDate

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

April 12, 2022

Contents

List of Tables

Table 1: **Revision History**

Date	Version	Notes
Apr 12	1.0	Rev 1: All group members completed the test report.

This document contains the results of executing tests based on the revised test plan.

1 Functional Requirements Evaluation

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

   Result: Pass

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

   Result: Pass

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)

   Result: Pass

Empty notion table creation

1. FR-ST-5

   Type: Functional, Dynamic, Manual

   Input: Dimensions and column headings for the table

   Output: Notion table

   Result: Pass

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

   Result: Pass

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

Result: Pass

Course code

1. FR-ST-8

   Type: Functional, Dynamic, Manual

   Initial state: Notion page has a PDF uploaded

   Input: Course name/code field

   Output: Generated table title

   Result: Pass

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

   Result: Pass

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

   Result: Pass

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

   Result: Pass

2 Nonfunctional Requirements Evaluation

2.1 Look and Feel Requirements

Appearance Requirements

1. NFR-LF-1

   Type: Manual, Dynamic, Checklist

   Result: Pass

Style Requirements

1. NFR-LF-2

   Type: Manual, Usability Survey

   Result: Pass

2.2 Usability and Humanity Requirements

Ease of use Requirements

1. NFR-UH-1

   Type: Manual, Usability Survey
   Result: N/A

2. NFR-UH-2

   Type: Manual, Usability Survey
   Result: N/A

3. NFR-UH-3

   Type: Manual, Usability Survey
   Result: N/A

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
   Result: 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
   Result: 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

   Result: Pass

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

   Result: Pass

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

   Result: Pass

2.4 Operational and Environmental Requirements

Expected Physical Environment

1. NFR-OE-1

   Type: Manual, Dynamic
   Result: Pass

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.
   Result: Pass

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
   Result: Pass

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

Result: Pass

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

   Result: Pass

Adaptability Requirements

1. NFR-MA-4

   Type: Automated, Dynamic

   Initial state: Program is running on different Operating Systems

   Input: N/A

   Output: N/A

   Result: Pass

2. NFR-MA-5

   Type: Automated, Dynamic

   Initial state: Program is running on different Internet Browser

   Input: N/A

   Output: N/A

   Result: Pass

3 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.

4 Usability

These survey questions were 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)

5 Unit Testing

Unit testing was performed using only the Pytest Framework.

5.1 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.

5.2 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.

6 Changes Due to Testing

One major change to SaveTheDate was the UI of the Notion webpage, as it was previously unsatisfactory to meet the non-functional requirements. There were no other major changes to the code as a result of testing. This was because the majority of tests passed, or were found to be irrelevant after updating the requirements of our project.

7 Automated Testing

Automated testing was performed when possible, especially focusing on the backend with Python unit testing. However, the majority of testing was done manually, as it involved directly interacting with the webpage or look-and-feel requirements.

8 Trace to 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-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

Table 2: 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
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

Table 3: Traceability Matrix: Non-Functional Requirement

9 Trace to Modules

The trace to modules is covered in the MG.

10 Code Coverage Metrics

Code coverage was not formally tested due to the majority of automated testing covering the backend python, which comprises a relatively low percentage of the total code. Instead, a large amount of the React code was tested manually for coverage, by interacting with the webpage such that every module was tested.