Project Portfolio

Lau Wei Tang

1. Introduction

Welcome to my Project Portfolio. This document provides you with an overview to my contributions to the project I have worked on, PDF++

2. About PDF++

Most university students have one thing in common, messy desktop. As a student, I have to manage multiple new documents weekly. Furthermore, each file is categorised differently or has different unique purpose. For instance, I would look for files from my programming modules on certain algorithm concepts or look from my language module for files with writing tips. However, current File Management System (FMS), such as File Explorer in Windows, has its limitations in the categorisation of files. Even for organised users, it is ineffective to make a constant effort maintaining the documents in an orderly manner, and inefficient switching between folders to look for the files of their interest. My team and I recognised this limitation and decided to work on building an elaborated FMS, PDF++.

2.1. Project Scope

For the team project, we were provided with an existing AddressBook application. We decided to morph the application into PDF++, a sophisticated FMS targeting students who prefer working with command line interface (CLI). However, given the limitation of the project, our application currently supports PDF files only, and thus the name.

2.2. Main role and contributions

I am one of the developers for PDF++ that in charge of implementing **File Protection** feature and most of the test cases for the Logic component.

2.3. Legend

Please refer to the table below for the icons and formatting used in this document:

icon	description
💡	Tip
⚠️	Warning

3. Summary of contributions

This section highlights my key contributions to the documentation, coding and technical aspects for the PDF++ project.

3.1. Robust File Protection System

PDF++ not only supports several essential features, such as add and open features, it also has a built-in sophisticated File Protection System (FPS).

For FPS, I have incorporated both `encrypt` and `decrypt` features and other improvements that enforces the security of the application.

This allows the users to protect sensitive data such that only authorised personnel have access to. Similarly, users can remove the security of the document with the password that was previously set. During this process of file protection, we have disabled some features, such as `history` and pressing up on the command box that will potentially reveal the password of the documents.

Though this system, users can safely rely on PDF++ for the protection of their files, without the fear of exposing the passwords which compromises their privacy.

Credit to Java Mitra for the tutorial guide in implementing the encrypt feature

3.2. Test Cases for Logic Component

PDF++ offers many advanced features that has complex implementation. Due to the complexity of the code, it is ideally that we have the comprehensive test cases for each command before the actual implementation of the features. This requires an in-depth knowledge of our code base and discussion with my team on how the feature should be implemented.

- **Code contributed**: Project Code Dashboard

3.3. Technical Leadership

For this project, we decided to make a complete overhaul from the existing code that works with imaginary data to support for actual file manipulation. This would require much initial adjustments in enabling our application to accept PDF documents. I have made significant contributions to the refactoring of **Person** class to **Pdf** class. Furthermore, for future expansion of features, I have implemented the add command. These initial ground work allows my team to test their code using the GUI with the pdf files they have added.

4. Contributions to the User Guide

This section includes my contributions to the documentation of the User Guide.

4.1. File Protection: `encrypt` & `decrypt`

PDF++ offers native support for file protection. The command `encrypt` allows you to protect your files with a password such that they cannot be accessed without the password. Similarly, the command `decrypt` allows you to remove the password that you had set for the file.

```
The terms "protect" and "encrypt" will be used interchangeably.
```

The **Encrypt** and **Decrypt** feature has the following syntax:

Format: `encrypt INDEX password/PASSWORD` [For Encryption]
Format: `decrypt INDEX password/PASSWORD` [For Decryption]

- `INDEX` refers to the index of the file on the list that you wish to encrypt/decrypt.
- `PASSWORD` refers to the password which you wish to assign to the particular file.

Examples:

- `encrypt 2 password/ThisIsNotASecurePassword`
- `decrypt 2 password/ThisIsNotASecurePassword`

Please refer to Encryption Guide or Decryption Guide for help in using these features.

4.1.1. Encryption Guide

```
"encrypt" feature will not work on files that are already encrypted.
```

Illustrated below is a sample usage scenario that provides a clear view to the inner workings of the Encrypt feature.

Step 1: Launch the application by double clicking the `pdfplusplus.jar`.

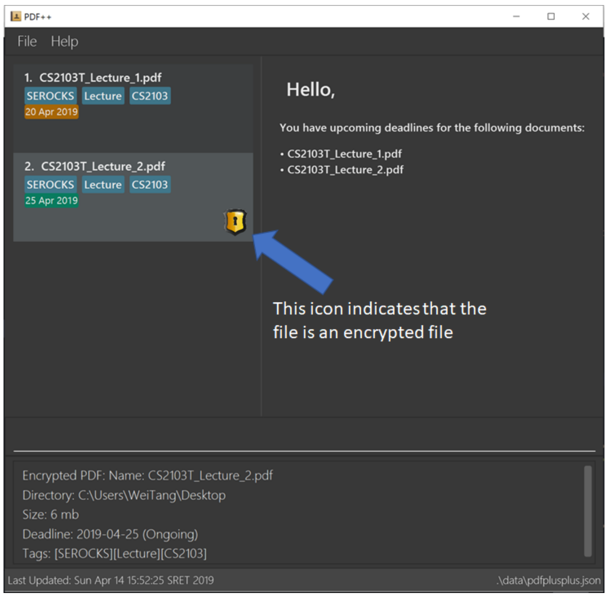
Step 2: Select the file that you wish to encrypt via the `INDEX` on the list.

Step 3: Enter the `encrypt` command into the text box, following the syntax as illustrated below.

- `encrypt 2 password/ThisIsNotASecurePassword`

Step 4: Upon hitting enter to execute the command, your inputs are verified and if successful, your selected file is encrypted with the given password.

```
Upon successful encryption, there will be a lock icon at the bottom right of the card
to indicate that the file is an encrypted file.
```



1. CS2103T_Lecture_1.pdf
SEROCKS Lecture CS2103
20 Apr 2019

2. CS2103T_Lecture_2.pdf
SEROCKS Lecture CS2103
25 Apr 2019

Hello,

You have upcoming deadlines for the following documents:

- CS2103T_Lecture_1.pdf
- CS2103T_Lecture_2.pdf

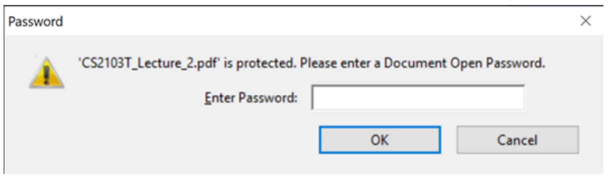
This icon indicates that the file is an encrypted file

```
Encrypted PDF: Name: CS2103T_Lecture_2.pdf
Directory: C:\Users\WeiTang\Desktop
Size: 6 mb
Deadline: 2019-04-25 (Ongoing)
Tags: [SEROCKS][Lecture][CS2103]
```

Last Updated: Sun Apr 14 15:52:25 SRET 2019 .\data\pdfplusplus.json

Figure 1. Lock icon on Encrypted Files

Step 5: If the command passes the validity check, the file you have selected is encrypted. You can open your file to see the result. Please refer to open guide for the `open` feature.



'CS2103T_Lecture_2.pdf' is protected. Please enter a Document Open Password.

Enter Password:

Figure 2. File has been encrypted

4.1.2. Decryption Guide

Illustrate below is a sample usage scenario that provides a clear view to the inner workings of the Decrypt feature.

```
"decrypt" feature is very similar to "encrypt" feature.
```

```
"decrypt" feature will not work on files that are not encrypted.
```

Step 1: Launch the application, similar to the Step-By-Step Encrypt guide.

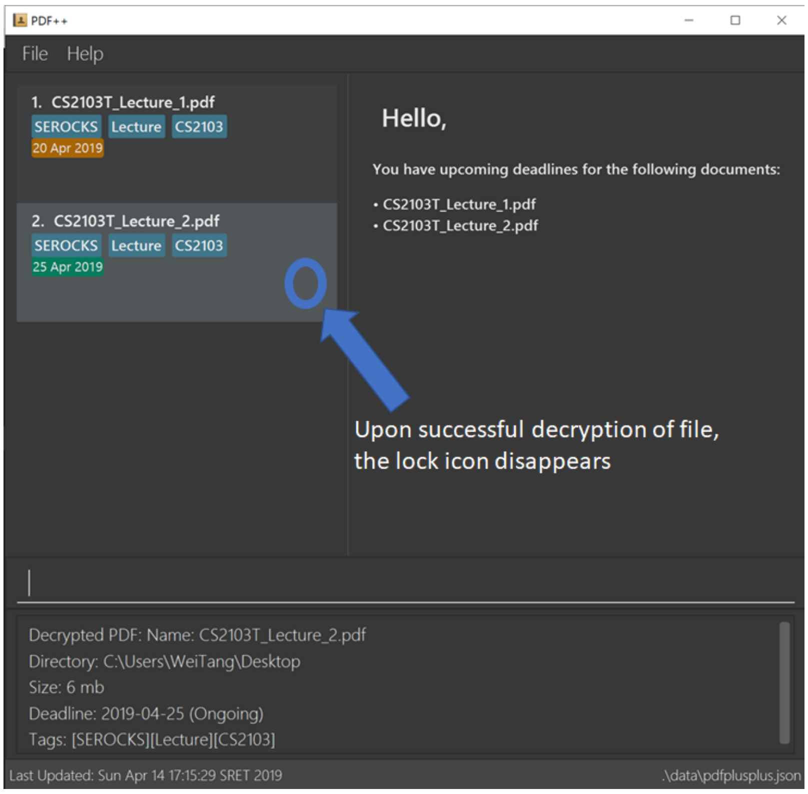
Step 2: You select the file that you wish to decrypt via the INDEX on the list.

Step 3: Enter the `decrypt` command into the text box, following the syntax as illustrated below.

- `decrypt 2 password/ThisIsNotASecurePassword`

```
Please enter the password of the encrypted file. You will not be able to decrypt the
file without the password.
```

Step 4: Upon hitting enter, the application verifies that you have entered the correct password and decrypts your file as shown below.



1. CS2103T_Lecture_1.pdf
SEROCKS Lecture CS2103
20 Apr 2019

2. CS2103T_Lecture_2.pdf
SEROCKS Lecture CS2103
25 Apr 2019

Hello,

You have upcoming deadlines for the following documents:

- CS2103T_Lecture_1.pdf
- CS2103T_Lecture_2.pdf

Upon successful decryption of file, the lock icon disappears

```
Decrypted PDF: Name: CS2103T_Lecture_2.pdf
Directory: C:\Users\WeiTang\Desktop
Size: 6 mb
Deadline: 2019-04-25 (Ongoing)
Tags: [SEROCKS][Lecture][CS2103]
```

Last Updated: Sun Apr 14 17:15:29 SRET 2019 .\data\pdfplusplus.json

Figure 3. Decrypt Command Step 4

4.2. Proposed feature in v2.0

- Login Page:
  We plan to include a Login Page to help authenticate you to the application so that you can access the app more securely.

Furthermore, together with the connection to external servers feature, this feature will allow you to access your documents anywhere you are.

5. Contributions to the Developer Guide

This section includes my contributions to the documentation of the Developer Guide.

5.1. File Protection System

PDF++ has a robust in-built file protection system which allows you to encrypt or decrypt any PDF files you want. These features utilises the *Apache PDFBox® library*, specifically the *PDDocument*, *AccessPermission*, and *StandardProtectionPolicy*.

```
An encrypted file is a file that is protected with a password. The terms "protect" and
"encrypt" will be used interchangeably.
```

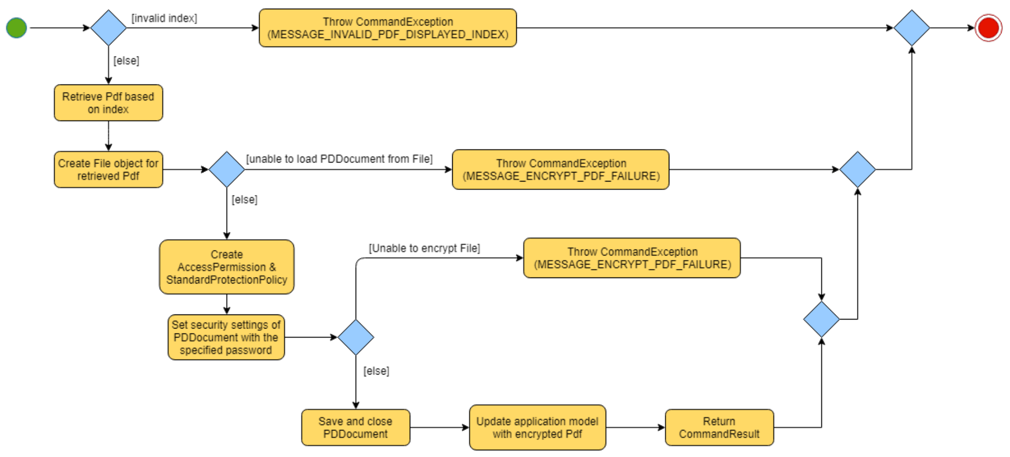
You can visit Encryption feature and Decryption feature to understand more about the respective feature.

5.1.1. Encryption feature

Current Implementation

The `encrypt` feature is facilitated by both **EncryptCommand** and **EncryptCommandParser**.

The implementation of the **EncryptCommand** is summarised in the following activity diagram:



[invalid index]
Throw CommandException (MESSAGE_INVALID_PDF_DISPLAYED_INDEX)
[else]
Retrieve Pdf based on index
Create File object for retrieved Pdf
[unable to load PDDocument from File]
Throw CommandException (MESSAGE_ENCRYPT_PDF_FAILURE)
[else]
Create AccessPermission & StandardProtectionPolicy
[Unable to encrypt File]
Throw CommandException (MESSAGE_ENCRYPT_PDF_FAILURE)
Set security settings of PDDocument with the specified password
[else]
Save and close PDDocument
Update application model with encrypted Pdf
Return CommandResult

1. The provided index is checked for validity i.e. referring to a specific Pdf in PdfBook.
   a. If the index is invalid, a **CommandException** will be thrown and the execution ends.
2. The Pdf specified via the index is retrieved from the PdfBook.
3. A `File` object is created for the Pdf.
4. The `File` will be loaded as *PDDocument*, which is an indicator that the `File` is a **PDF** document that is uncorrupted and not protected with a password.
   a. Error in loading Pdf as **PDDocument** would throw an **IOException**. Common reasons of

error are:

i. File not found in location

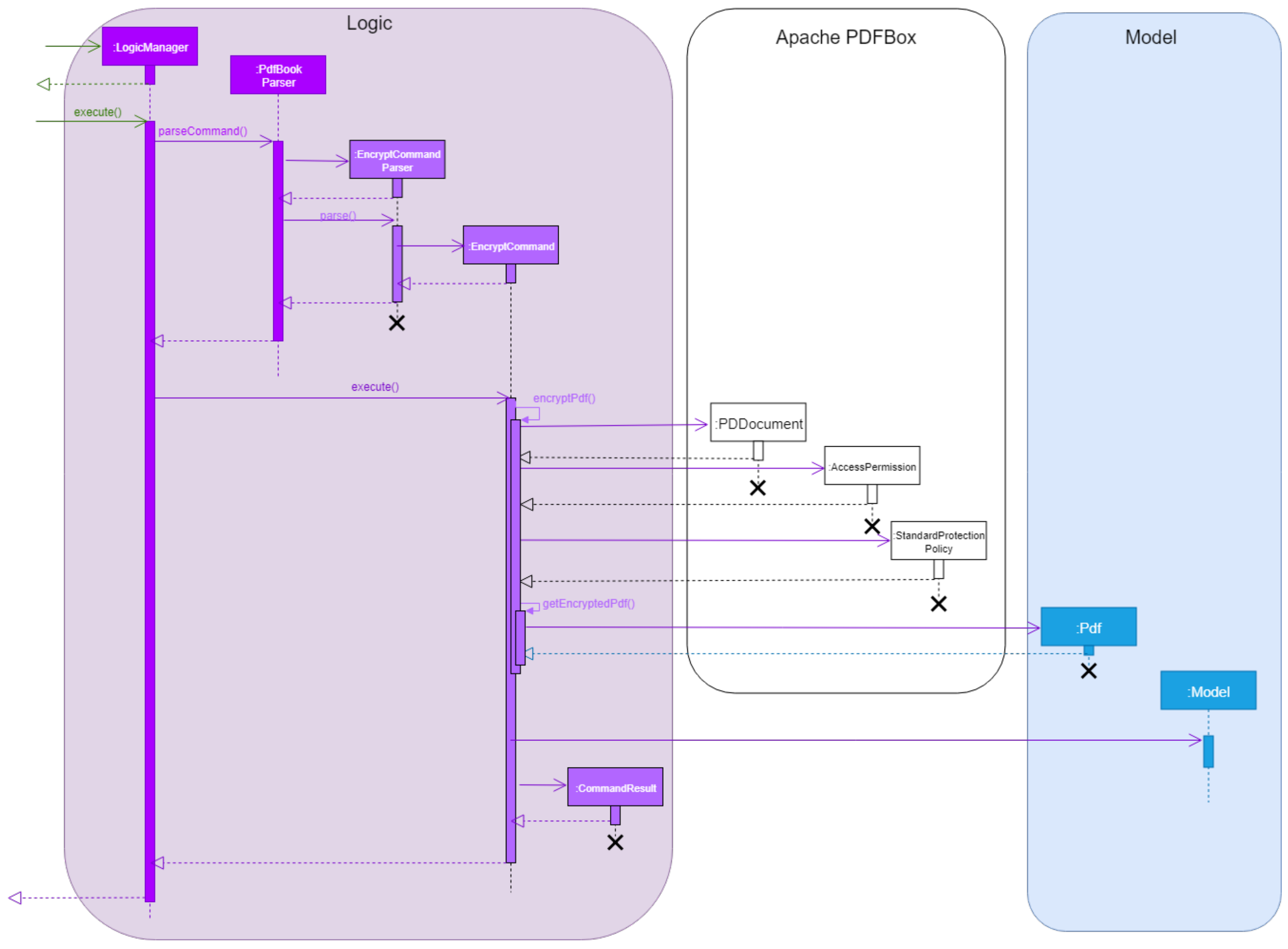
ii. Lack of user permissions to open File

iii. Protected, corrupted File

b. Thrown IOException is intercepted, a **CommandException** will be thrown and the execution ends.

5. *AccessPermission* and *StandardProtectionPolicy* are created. The password specified will be passed to `StandardProtectionPolicy` for the purpose of setting security settings for the `PDDocument`.
6. A protected Pdf will be saved and closed.
   a. Error in encrypting the file will throw an **IOException**. Common reasons of error are:
      i. Excessive long password
      ii. Empty password
   b. Thrown IOException is intercepted, a **CommandException** will be thrown and the execution ends.
7. The Pdf is recorded in the Model component and the changes are saved.
8. A **CommandResult** is returned upon successful exception of **EncryptCommand**.

This sequence diagram demonstrates the Main Success Scenario from the **LogicManager** to the end of **EncryptCommand** execution:



Design Considerations

Edit password of an encrypted file

- Alternative 1 (current choice): Execute **DecryptCommand** then **EncryptCommand**
  - Due to security reasons, it was decided to focus on encrypting an unprotected Pdf. You will need to use Decryption feature before encrypting it with a new password. This is to ensure your intent in changing the password, as the current version **PDF++** does not support `Forget Password` feature.
  - However, this process is inefficient as you will need to enter 2 commands instead of 1.
- Alternative 2: Change password of an encrypted file
  - This minimised the number of commands to be executed, but there are several security concerns as mentioned above.

A sophisticated protection system

- Multiple adjustments to protect your interest
  1. History feature will not show the executed `EncryptCommand` which includes the password of the file.
  2. The Undo/Redo feature is temporarily disabled until a solution that will not comprise your privacy has been found.
  3. Pressing up in the command box will not show the `EncryptCommand` that was previously executed.

Future Implementation

There are concerns of accidental encryption of a file with a wrong password. In **PDF++** v2.0, the **EncryptCommand** will prompt you to re-enter the password as a form of confirmation message. If there is a mismatch of the two passwords entered, the command will not be executed.

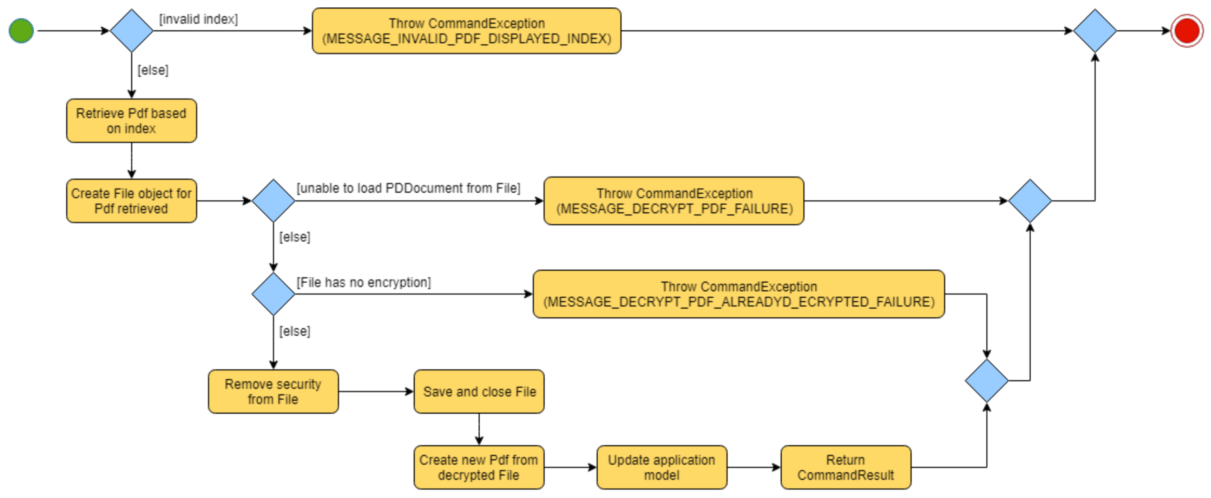
5.1.2. Decryption feature

Current Implementation

```
The current Implementation of *DecryptCommand* is very similar to Encryption feature.
The part where it is implemented differently will be specifically marked with a `*`
for your convenience.
```

The `decrypt` feature is facilitated by both **DecryptCommand** and **DecryptCommandParser**.

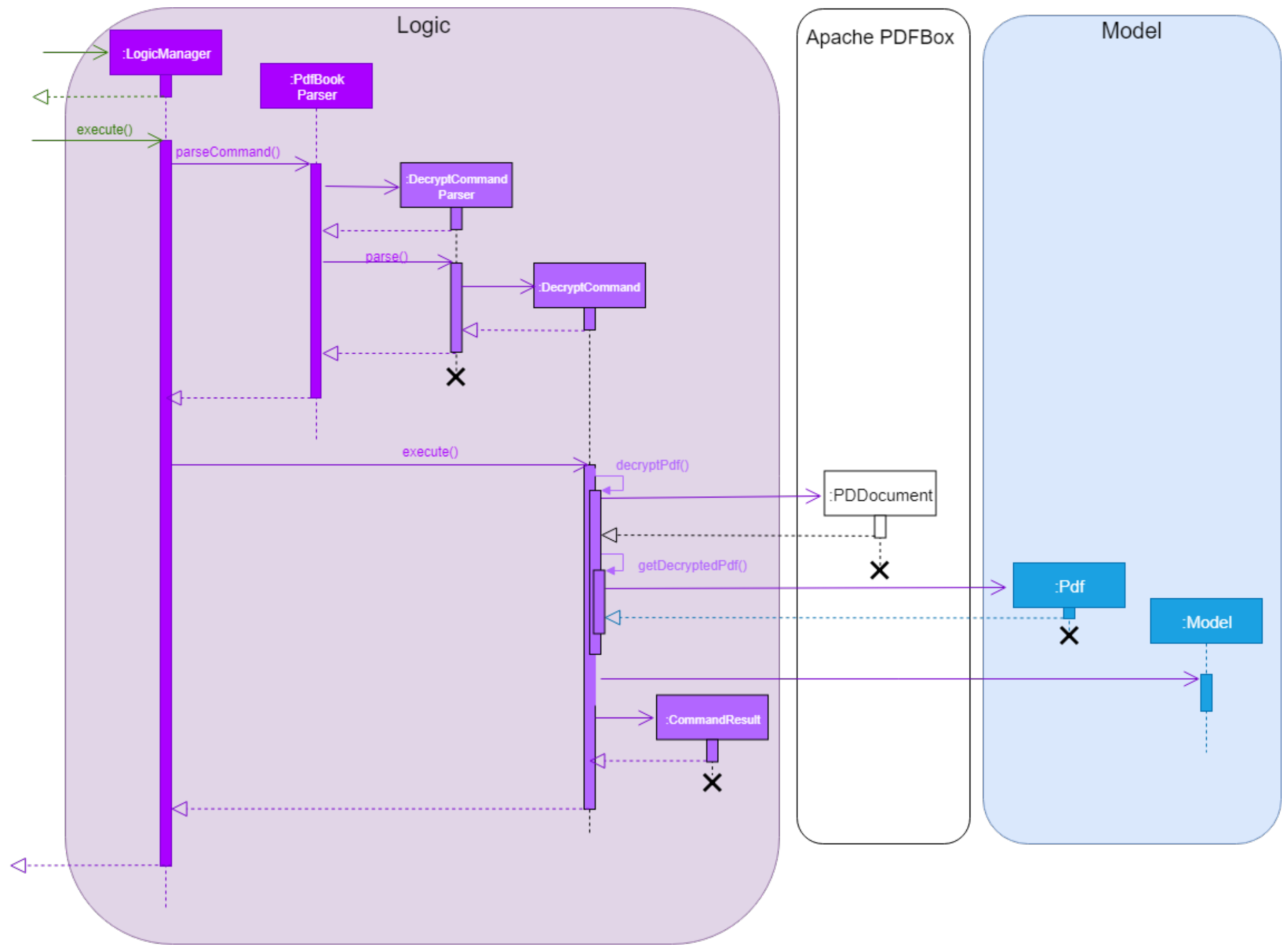
The implementation of the `DecryptCommand` execution is summarised in the following activity diagram.



[invalid index]
Throw CommandException (MESSAGE_INVALID_PDF_DISPLAYED_INDEX)
[else]
Retrieve Pdf based on index
Create File object for Pdf retrieved
[unable to load PDDocument from File]
Throw CommandException (MESSAGE_DECRYPT_PDF_FAILURE)
[else]
[File has no encryption]
Throw CommandException (MESSAGE_DECRYPT_PDF_ALREADYD_ECRYPTED_FAILURE)
[else]
Remove security from File
Save and close File
Create new Pdf from decrypted File
Update application model
Return CommandResult

1. The provided index is checked for validity i.e. referring to a specific Pdf in PdfBook.
   a. If the index is invalid, a **CommandException** will be thrown and the execution ends.
2. The Pdf specified via the index is retrieved from the PdfBook.
3. A `File` object is created for the Pdf.
4. The `File` will be loaded as *PDDocument* with the specified password, which is an indicator that the `File` is a **PDF** document that is uncorrupted, protected and the password provided is valid *.
   a. Error in loading Pdf as **PDDocument** would throw an **IOException** and invalid password would throw an **CommandException**. Common reasons of error are:
      i. File not found in location
      ii. Lack of user permissions to open File
      iii. Unprotected File *
      iv. Corrupted File
      v. Wrong password *
   b. Thrown IOException is intercepted, a **CommandException** will be thrown and the execution ends.
5. Upon success loading of the PDDocument, the security will be removed. *
6. An unprotected * Pdf will be saved and closed.
7. The Pdf is recorded in the Model component and the changes are saved.
8. A **CommandResult** is returned upon successful exception of **DecryptCommand**.

This sequence diagram demonstrates the Main Success Scenario from the **LogicManager** to the end of **DecryptCommand** execution:



Design Considerations

A sophisticated protection system

- Multiple adjustments to protect your interest

  1. History will not show the executed `EncryptCommand` which includes the password of the file.
  2. The Undo/Redo feature is temporarily disabled until a solution that will not comprise your privacy has been found.
  3. Pressing up in the command box will not show the `EncryptCommand` that was previously executed.

Future Implementation

If an unauthorised personnel obtained the password of your files through illegal means, they can potentially set the file with a new password. This will hinder your access to your files.

However, With 2-Factor Authentication, there is an additional layer of protection that prevents these personnel from changing the passwords of your files. This ensures that only you/any authorised personnel can decrypt your files.

In **PDF++** v2.0, the **DecryptCommand** will support for the 2FA feature as mentioned above. If this feature is highly demanded, this feature will be implemented to **EncryptCommand** too.