# Bernice Chio Hui Yin - Project Portfolio

## PROJECT: MORTAGO

# 1. Introduction

This document serves to note down my contributions to Mortago, our team's product for our Software Engineering project.

Our team consists of five computer science undergraduates undertaking CS2103T Software Engineering module. We were tasked with enhancing a basic Command Line Interface (CLI) desktop address book application for our Software Engineering project. We chose to morph it into a mortuary management system called Mortago.

Mortago is a desktop application used for managing mortuaries efficiently. It is primarily used for keeping records of entries into the mortuary.

The user keys in commands using a CLI, and views the command results on a Graphical User Interface (GUI) created with JavaFX. Mortago was built upon a given base code provided by the SE-EDU team. It is written in Java, and has about 20kLoC.

Here are the main features of Mortago:

1. Generate Report
2. View Alerts
3. View Dashboard
4. View Statistics
5. Undo and Redo

# 2. Summary of contributions

This section provides the overall view of my contributions to Mortago. All my pull requests (PRs) can be found here.

### 2.1. Major Enhancements

I have contributed 3 supportive enhancements for Mortago's main feature Generate PDF. Generate PDF automatically generates reports with the specified command.

- **2.1.1. Major enhancement 1:** added the ability to create a PDF document which contains reports for all bodies in the mortuary

    - **What it does:** This enhancement allows the manager to automatically generate all full reports for all bodies in a PDF document.
    - **Justification:** This enhancement provides convenience for the manager to generate reports for all bodies using a single command.
    - **Highlights:** This enhancement standardises the format of every report in the document for consistency and prints timestamp when the report is generated. The manager's signature can also be added to every report automatically. The implementation required a good understanding of the iText Library.
    - **Credits:** The iText Library is heavily used in this feature to facilitate generating a PDF file in Java.
- **2.1.2. Major enhancement 2:** added the ability to create a PDF document which serves as the report for a specific body
    - **What it does:** This enhancement allows the manager to automatically generate a single full report containing details of a specific body in a PDF document.
    - **Justification:** In the event that the details of a specific body has been updated, this enhancement provides flexibility for the manager to generate an updated report for the specific body.
    - **Highlights:** This enhancement utilises tables to organise the details of the specific body to improve readability for a comprehensive report and prints timestamp when the report is generated. The implementation required a good understanding of the iText Library.
    - **Credits:** The iText Library is heavily used in this feature to facilitate generating a PDF file in Java.
- **2.1.3. Major enhancement 3:** added the ability to create a PDF document which contains a tabular view for all bodies in the mortuary
    - **What it does:** This enhancement allows the manager to have a brief overview of all the bodies in a PDF document.
    - **Justification:** This enhancement highlights key information of each body to the manager so that the manager need not go through every full body report.
    - **Highlights:** This enhancement organises the crucial information for every body in a table and includes statistics such as the total number of bodies in the mortuary. The implementation required a good understanding of the iText Library.
    - **Credits:** The iText Library is heavily used in this feature to facilitate generating a PDF file in Java.

**2.2. Minor enhancement**: morphed `list` command to allow the manager to list all bodies, workers and fridges in the application.

**2.3. Code contributed**: [via RepoSense]

**2.4. Other contributions**

- Project management: Remove non-existing developers photos from `docs/images` folder (Pull request #130)

- Documentation
  - made cosmetic tweaks to existing contents of the Developer Guide (Commits: 1, 2, 3)
  - updated existing diagrams and color scheme for diagrams of the Developer Guide (Pull request #137)
  - updated Document Generation section in the User Guide (Pull request )
  - updated Section 4.4 Generate PDF feature in the Developer Guide (Pull request )
  - updated Appendix B User Stories in the Developer Guide (Commit 1)
- Community
  - PRs reviewed: #9, #31, #47, #63, #102
  - Contributed to forum discussions (Issue: #89)
  - Reported bugs and suggestions for other teams (examples: 1, 2, 3)
- Tools: Integrated a third party library (iText) to the project (#42)

# Contributions to the User Guide

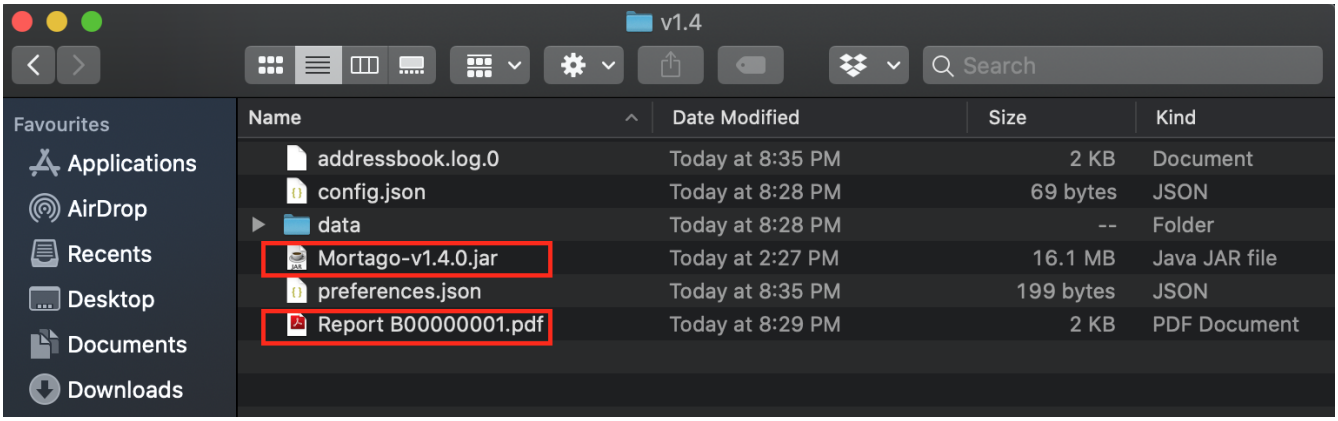
*Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.*

## Document Generation

This section showcases three commands you can use to generate different kinds of reports.

### Generate report for a specific body: `genReport`

This command allows you to generate an individual report in a PDF filename `Report <body_ID>.pdf` located in the folder containing the jar file.




You can also add your signature name to the report by following the command format below.

Format: `genReport bodyId (yourSignatureName)`

Example:

- `genReport 123 John Doe`
  Outputs the report PDF for body ID B00000123 in the folder containing the jar file. The report contains signature of John Doe.

### Generate reports for all bodies: `genReports`

This command allows you to generate all individual reports in a PDF filename `Report (ALL BODIES).pdf` located in the folder containing the jar file.

You can also add your signature name to the report by following the command format below.

Format: `genReports (yourSignatureName)`

Example:

- `genReports John Doe`
  Outputs the PDF containing reports for all bodies in the folder containing the jar file. The report contains signature of John Doe.

### Generate summary report for all bodies: `genReportSummary`

This command allows you to generate a tabular report overview containing key information for all bodies in a PDF filename `Report Summary.pdf` located in the folder containing the jar file.

You can also add your signature name to the report by following the command format below.

Format: `genReportSummary (yourSignatureName)`

# Contributions to the Developer Guide

> *Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.*

## Generate PDF Feature

This feature allows manager to automatically generate different kinds of reports with three commands: `genReport`, `genReports` and `genReportSummary`.

### Implementation

The generate PDF feature is facilitated by `ReportGenerator`. It extends `Mortago` with the ability to create a report, supported by iText external library. Additionally, it implements the following operations:

- `ReportGenerator#generate(body, sign)` — Creates report containing sign name of manager in a PDF file for the specific body.
- `ReportGenerator#generateAll(sign)` — Creates reports containing sign name of manager in a PDF file for all bodies registered in Mortago.

- `ReportGenerator#generateSummary(sign)` — Creates a tabular summary report containing sign name of manager in a PDF file for all bodies registered in Mortago.

The following sequence diagram (Figure 4.4.1.a) shows how the generate operation works:

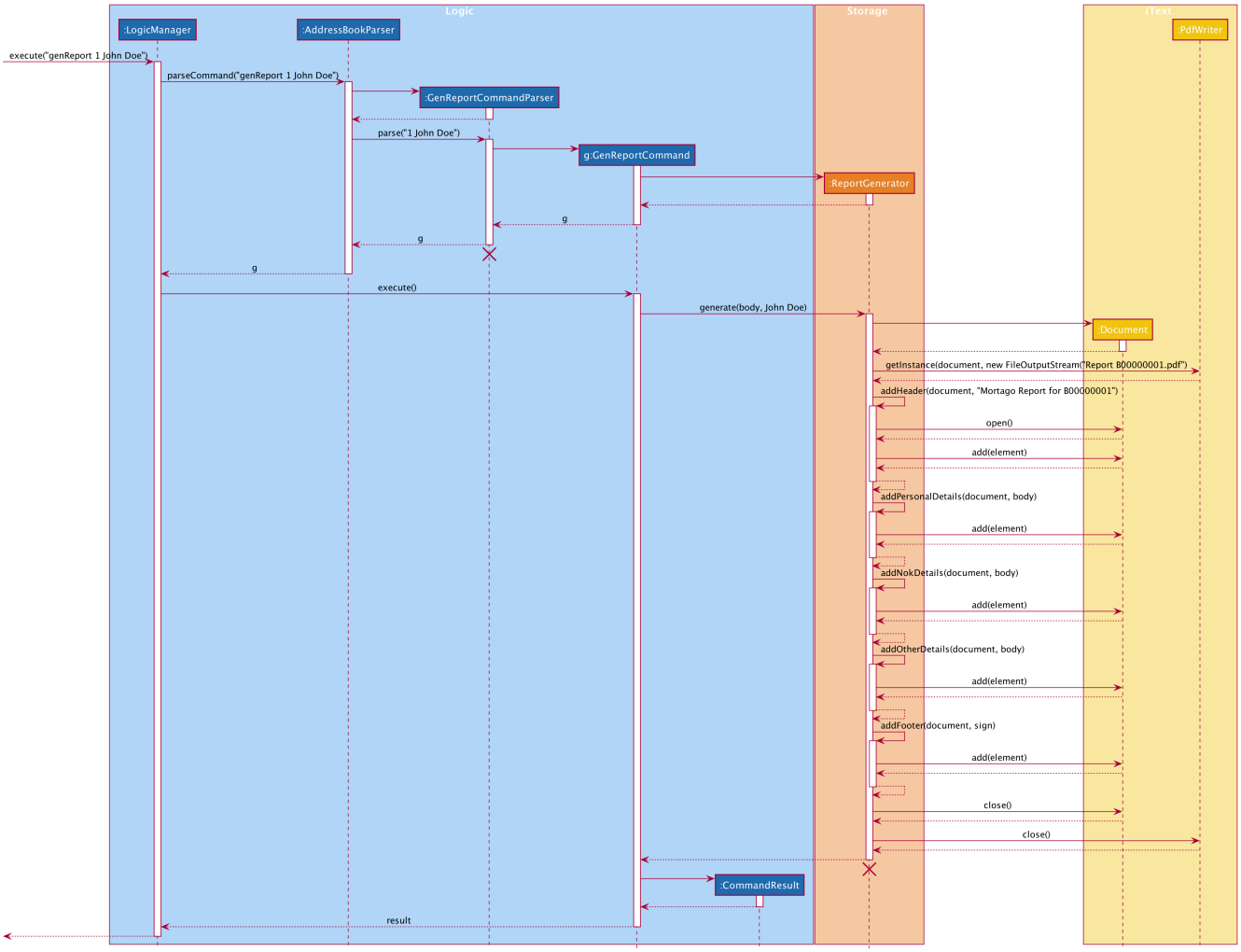



Figure 4.4.1.a. Sequence diagram when `genReport 1 John Doe` is executed by manager.

| | |
|---|---|
| **NOTE** | The lifeline for `GenReportCommandParser` and `ReportGenerator` should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram. |

The `genReport <BODY_ID> (sign)` command calls `ReportGenerator#generate(body, sign)`, which creates the document.

The following activity diagram (Figure 4.4.1.b) summarizes what happens when manager executes a `genReport <BODY_ID> (sign)` command:

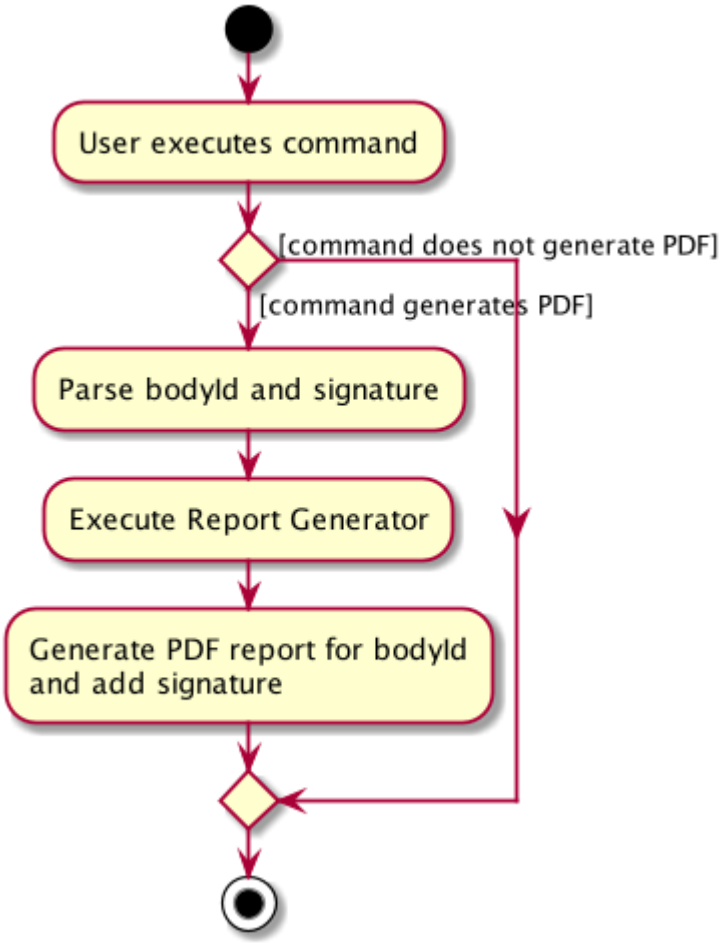



Figure 4.4.1.b. Activity diagram when `genReport 1 John Doe` is executed by manager.

The following code snippet from `GenReportCommand.java` demonstrates how an error message is displayed when manager inputs an invalid command and when report is not successfully generated:

```
if (bodyToGenReport == null) {
    throw new CommandException(MESSAGE_INVALID_ENTITY_DISPLAYED_INDEX);
}
boolean generated = reportGenerator.generate(bodyToGenReport, sign);
if (!generated) {
    throw new CommandException(MESSAGE_REPORT_NOT_GENERATED);
}
```

The following class diagram (Figure 4.4.1.c) models the relationships and dependencies among classes in this feature:

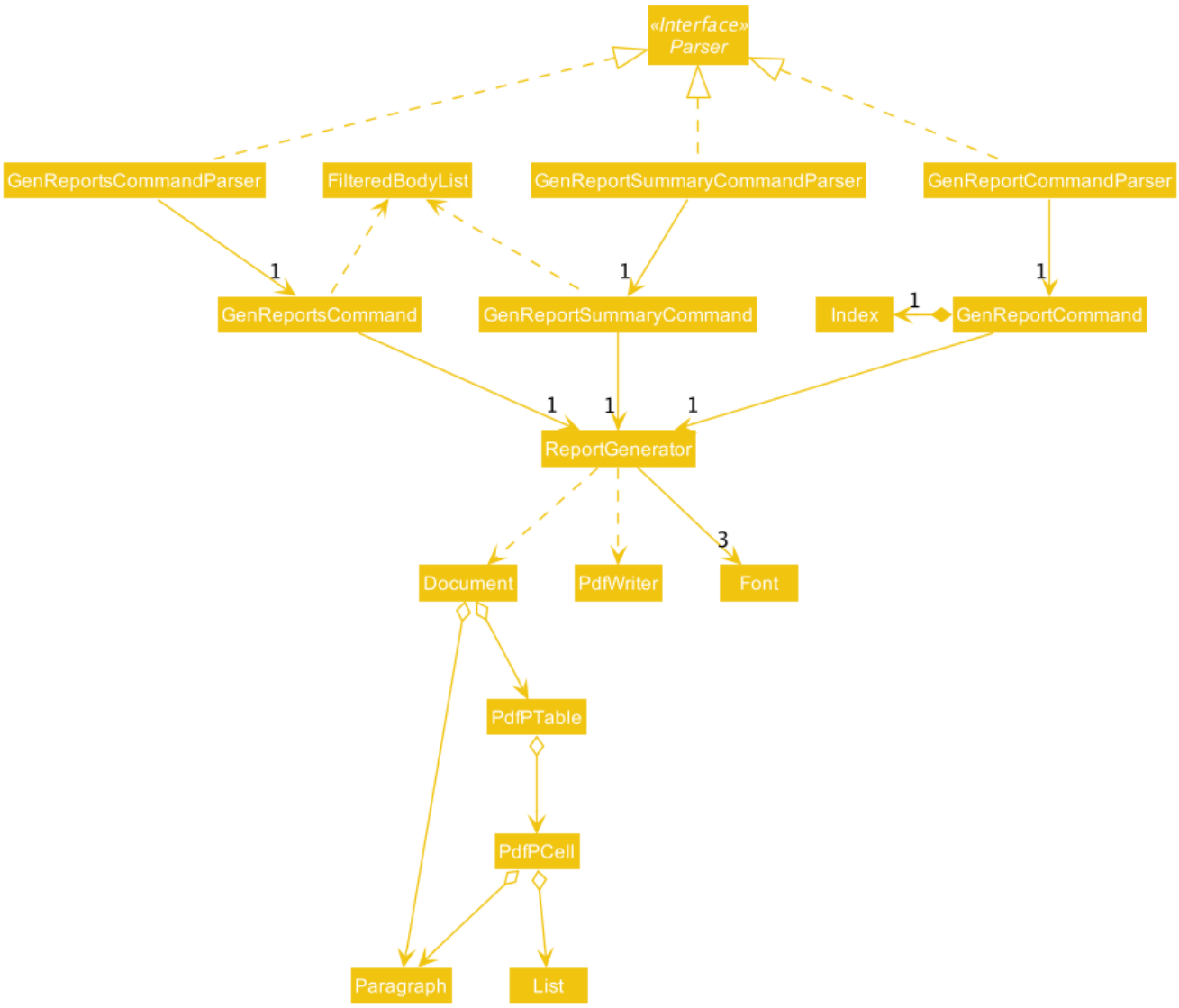



Figure 4.4.1.c. Class diagram for generate PDF feature.

## Design Considerations

**Aspect: How generate report executes**

- **Alternative 1 (current choice):** Create a PDF file.
  - Pros: Implementation is easy.
  - Cons: Implementation must ensure that each individual body attribute is correct.
- **Alternative 2:** Create a Word Document file.
  - Pros: Implementation allows manager to edit the contents of the report.
  - Cons: Implementation defeats the purpose of being automated.

Alternative 1 is the current choice because this will prevent manager from making accidental changes to the report when report is formatted in PDF.

**Aspect: What library to utilise for generating PDF in Java**

- **Alternative 1 (current choice):** Use iText to implement this feature.
  - Pros: Implementation is simple because using iText would allow an API-driven approach.
  - Cons: Implementation is unable to use the latest version (iText 7) because it is not compatible, only version 5.5.13 is compatible.
- **Alternative 2:** Use Apache PDFBox to implement this feature.
  - Pros: Implementation is easy because PDFBox is widely used and help is more accessible.
  - Cons: Implementation is limited because PDFBox can only create simple PDFs based on text files, supports few of the features iText does.

Alternative 1 is the current choice because this implementation does not require the enhancements provided by iText 7 but still requires more advanced library to create tables in a PDF document.

**Aspect: How report is formatted**

- **Alternative 1 (current choice):** Use tables to organise related details in the report.
  - Pros: Implementation allows report to be organised, increases readability for manager.
  - Cons: Implementation is tedious.
- **Alternative 2:** List all attributes in the report without any formatting.
  - Pros: Implementation is easy.
  - Cons: Implementation decreases readability for manager.

Alternative 1 is the current choice because manager will be able to save time and reduce work-related stress when manager is able to view an organised report.

---