

# Choong Jin Yao - Project Portfolio

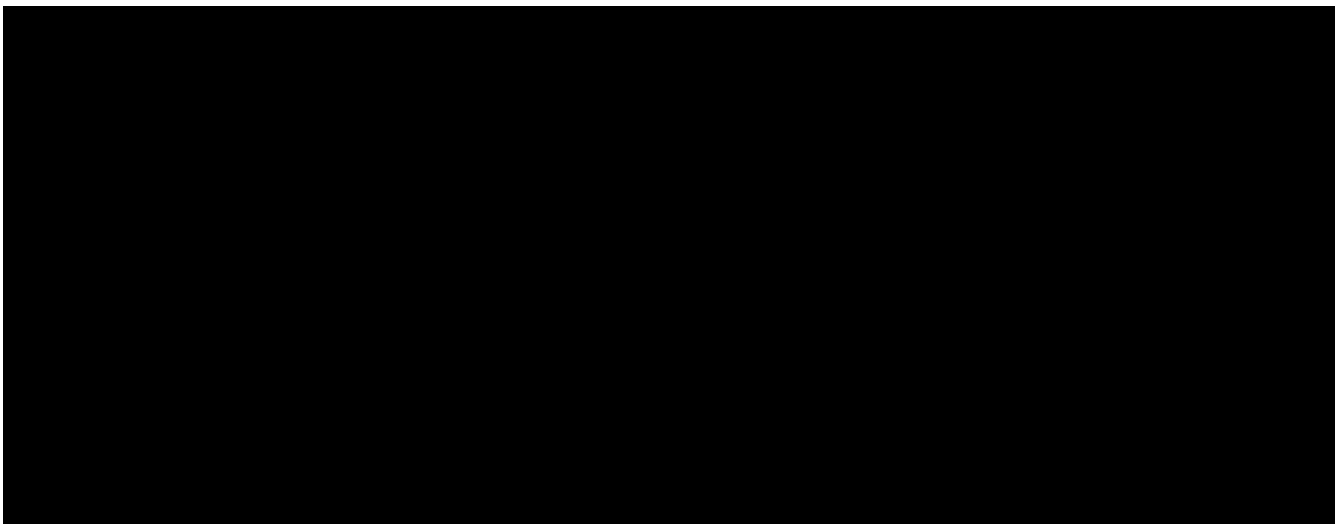
## PROJECT: DeliveryMANS

---

### Introduction

DeliveryMANS is a food delivery management system primarily for administrators in food delivery centres to effectively manage their delivery services. This necessitates the need to manage 4 different aspects which are food orders, deliverymen, restaurants as well as customers. To accommodate more efficient management of these aspects, DeliveryMANS has 4 different in-built interfaces(also known formally as contexts): universal/order, deliverymen, restaurant, and customer. The user primarily interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 20 kLoC.

This is what our project looks like in the case of the deliverymen context:



My role was to design and write the codes for the context for the deliverymen side. This comprises all the features to manage deliverymen status as well as computing crucial statistics regarding their status.

The following sections illustrate these enhancements in more detail, as well as the relevant documentation I have added to the user and developer guides in relation to these enhancements.

### Summary of contributions

- ¥ Major feature: implemented the ability to analyse and compute relevant statistics regarding deliverymen status

- ! What it does: allows the user to have a quick summary of the current state of his deliverymen

- ! Justification: This feature allows the user to have a quick overview of the state of his manpower with a simple command so that he can be aware of any potential shortage of deliverymen.
- ! Highlights: This enhancement has high development potential in future when the app is further enhanced as the statistics can begin to employ predictive capabilities with regards to demand over the next few weeks.

#### ¥ Major feature: implemented the management of deliverymen and their status

- ! What it does: allows the user to easily manipulate the list of deliverymen as well as updating their respective status
- ! Justification: This feature allows for efficient management of deliverymen and convenient navigation of the lists of deliverymen through the status buttons as well as unique colors to denote different status.
- ! Highlights: This enhancement required an in-depth analysis of design alternatives for the deliverymen interface. The implementation for the color change and buttons was challenging as it required an excellent knowledge in JavaFX.

#### ¥ Minor enhancement: integrated the updating of deliverymen status into other contexts

- ! What it does: allows all the relevant commands in all contexts to make necessary updates to the deliverymen status
- ! Justification: This enhancement ensures the updating of deliverymen status are well-integrated into all the commands from the four contexts.
- ! Highlights: This enhancement will always be necessary as long as there are new commands or features that are added in future that are related to deliverymen status.

#### ¥ Code contributed: [[RepoSense](#)][[Test code](#)]

#### ¥ Other contributions:

- ! Project management:
  - " Managed releases [v1.2](#) - [v1.4](#) (3 releases) on GitHub
- ! Enhancements to existing features:
  - " Wrote additional test for existing features ([#260](#))
  - " Improved the GUI appearance for easier navigation ([#99](#))
- ! Documentation:
  - " Did cosmetic tweaks to existing contents of the User Guide: ([#93](#), [#108](#), [#231](#), [#252](#))
- ! Community:
  - " PRs reviewed (with non-trivial review comments): [#241](#)
- ! Tools:
  - " Integrated a new Github plugin (TravisCI) to the team repo ([#2](#))

# Contributions to the User Guide

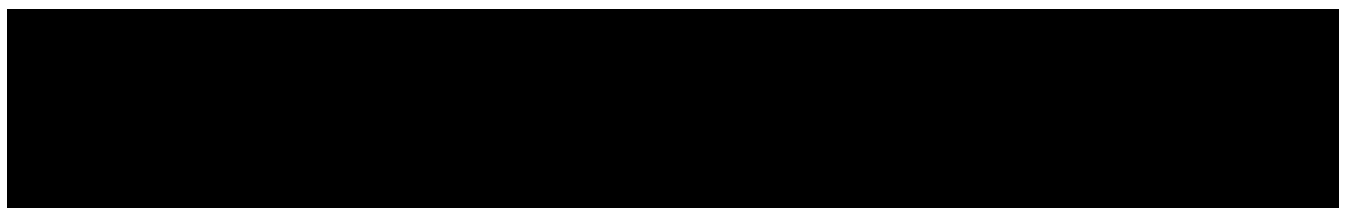
Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

Start of extract ↵

## Deliverymen commands

These are the commands you can use after entering the deliverymen context (via the command `-del i verymen`). The screenshot below shows how the deliverymen context looks like.

For each deliveryman, his information will be displayed in the following vertical order: name, tag(s), current status and phone number. This is shown in another screenshot below.



↵

NOTE

*Note:*

A deliveryman can have any of the 3 statuses: AVAILABLE, UNAVAILABLE, DELIVERING.

## Adding a deliveryman: **add**

This command allows you to add a new deliveryman to the deliverymen database. The name and phone number of the deliveryman must be provided.

Format: **add** n/NAME p/PHONE

Example: **add** n/John Doe p/91234567

NOTE	<i>Requirements</i> ⌘ The phone number must be at least 3 digits and not exceed 10 digits.
NOTE	<i>Note:</i> ⌘ A newly-added deliveryman will be assigned with the default status UNAVAILABLE. ⌘ Tags cannot be added through this command. Refer to the <b>edit</b> command in <a href="#">Section 3.4.2</a> to add tags to a deliveryman.

Ê

## Editing a deliveryman: **edit**

This command allows you to edit an existing deliveryman in the deliverymen database. The index of the deliveryman to be edited must be provided. You can edit only the field that you wish to change, but at least one field must be edited (eg. name, phone number, tag(s)). You can also add more than one tag, as illustrated in Example 2.

Format: **edit** INDEX [n/NAME] [p/PHONE] [t/TAG]\*

Example 1: **edit** 1 n/John Hoe p/97654321

Example 2: **edit** 1 t/active t/buff t/bestDeliveryman2019

NOTE	<i>Requirements</i> ⌘ The index INDEX provided must be within the deliverymen list size and be at least 1. ⌘ Tags must not contain any whitespace.
------	--

Ê

## Deleting a deliveryman: **delete**

This command allows you to delete an existing deliveryman from the deliverymen database. The index of the deliveryman to be deleted must be provided.

Format: **delete** INDEX

Example: **delete** 1

NOTE

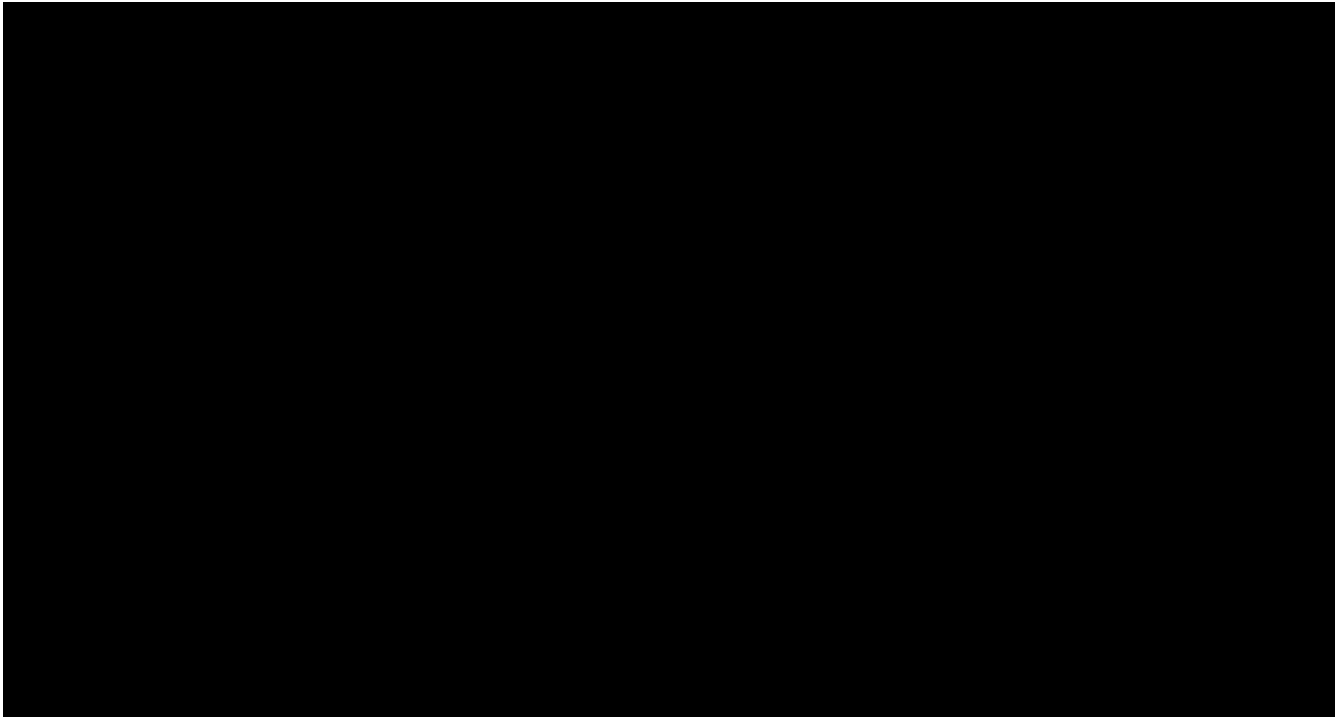
*Requirements*

¥ The index provided must be within the deliverymen list size and be at least 1.

Ê

## Listing status lists of deliverymen: **lists**

This command allows you to view all the deliverymen sorted according to their statuses. For easy navigation, you can click on any of the three buttons, as shown in the red box of the screenshot below, to display the status list that you wish to view.



Format: **lists**

Example: **lists**

Ê

## Changing status of a deliveryman: **status**

This command allows you to switch the status of a deliveryman between AVAILABLE and UNAVAILABLE. Using this command will prompt the status lists as well (that can also be done through **lists** command explained in [Section 3.4.4](#)).

Format: **status INDEX**

Example: **status 3**

NOTE

*Requirements*

¥ The index provided must be within the deliverymen list size and be at least 1.

¥ The status of the deliveryman must be either AVAILABLE or UNAVAILABLE.

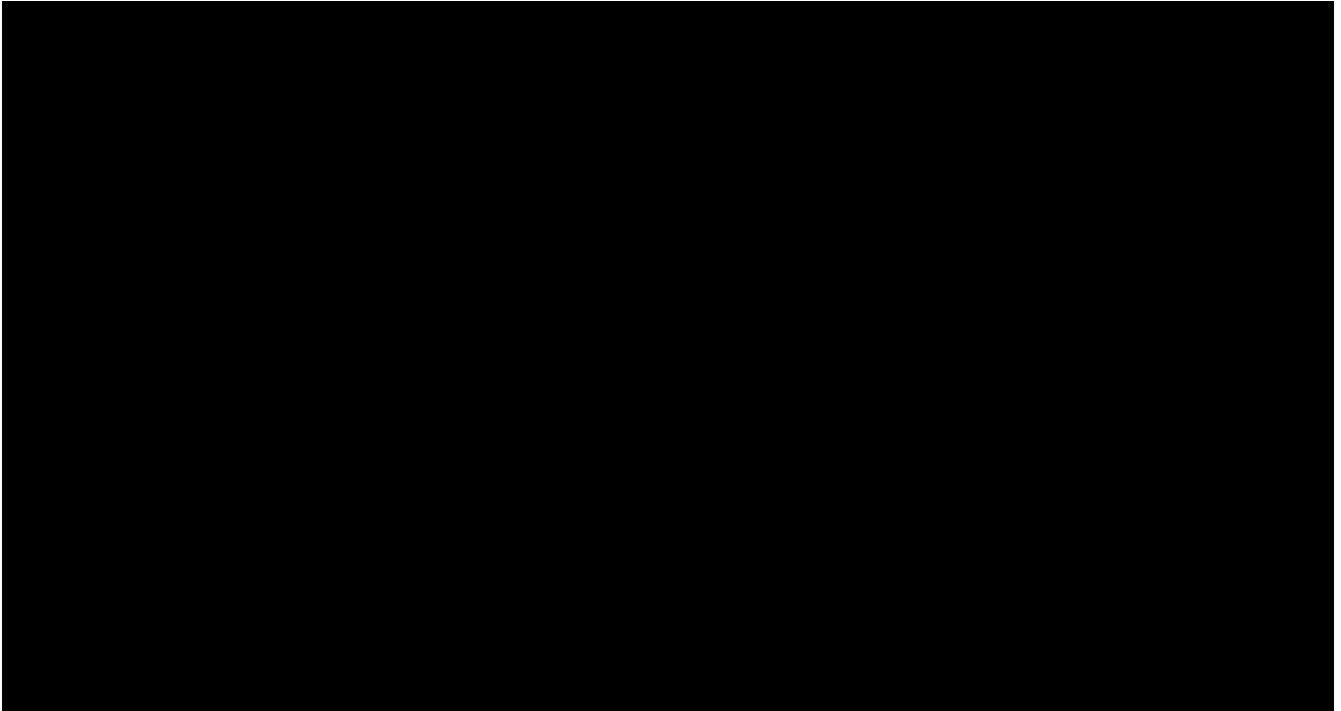
*Note:*

NOTE

You are not allowed to change the status of a deliveryman who has the status DELIVERING. It can only be changed through changes to orders assigned to him.

*Example use case*

1. Your deliveryman, Donald Trump, was on sick leave and his status is currently UNAVAILABLE. However, he just informed you that he has recovered and is ready to work again! You want to set his status back to AVAILABLE.
2. You search for his name and identify his index as **2**, as shown in the red border. Then you type in the command **status 2**.



3. The result box will display that the status of Donald Trump has been changed. And indeed, it has been switched to AVAILABLE! Also, you notice that the status lists have been prompted, as shown in the blue border.



Ê

## Viewing the statistics of the deliverymen statuses: `stats`

This command allows you to view the statistics of the current statuses of deliverymen. Relevant statistics such as the utilisation level will also be computed and displayed for your reference. A brief explanation of each statistic is provided as well.

Format: `stats`

Example: `stats`



In the screenshot above, the information inside the blue border is the statistics. The text shown inside the red border is a summary of the analysis.

#### TIP

If you just want a brief update on the current state of your deliverymen, the textual analysis displayed inside the red border summarizes it all for you!

*End of extract*

## 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.*

*Start of extract*

### Deliverymen status statistics feature

Tracking the amount of available manpower you have at any point in time is very crucial to deal with sudden changes in food orders. This is a feature which allows the user to effectively manage the deliverymen by providing relevant statistics in regards to their status. The user will be able to foresee and hence prepare for any sudden shortage or excess in manpower. Relevant statistics include current utilization level and activity level of the deliverymen.

#### NOTE

A deliveryman can have any of the 3 status: AVAILABLE, UNAVAILABLE, DELIVERING. To effectively manage their status, 3 different lists are used to categorise all the deliverymen based on their status. This way of implementation is efficient when it comes to sorting or searching for a deliveryman.

### Implementation

The statistics feature is primarily managed by the `StatisticsManager`, which has access to the status of every deliveryman in the database through the 3 status lists - lists of available deliverymen, unavailable deliverymen and delivering deliverymen respectively. The `StatisticsManager` obtains these 3 status lists from the `StatusManager` and passes it to the `Analyzer` to compute the relevant data, which in turn returns a filled `StatisticsRecordCard` containing the computed statistics.

From the status lists, the Analyzer computes two data:

#### ¥ Utilization Level

! This signals to the user the amount of available manpower that is not utilized. It is the percentage of the number of deliverymen who are working (ie. delivering an order) over the number of active deliverymen (ie. available or delivering an order) .

! It is computed as follows:

"  $\text{UtilizationLevel} = \text{DeliveringMen} / (\text{AvailableMen} + \text{DeliveringMen}) \times 100$



## ¥ Activity Level

! This signals to the user the amount of active manpower. It is the percentage of the number of deliverymen who are active deliverymen over the total number of deliverymen.

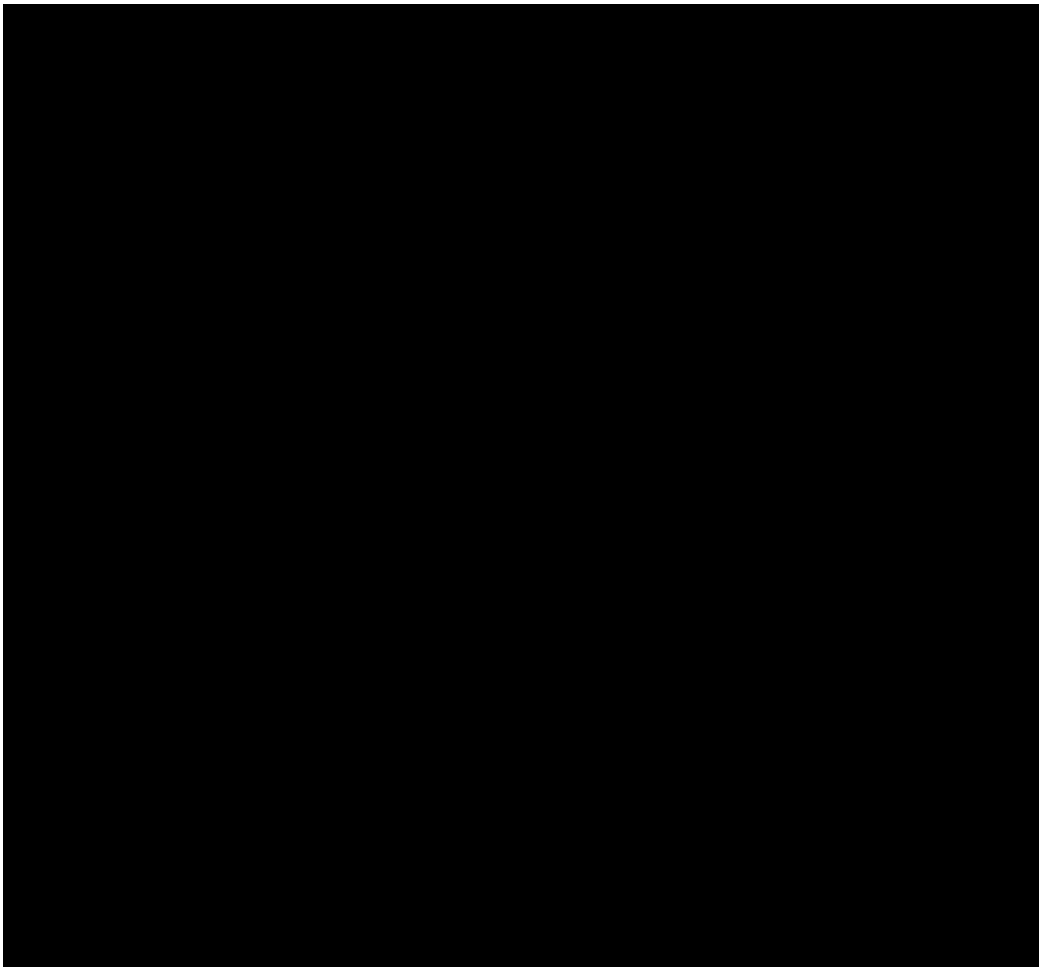
! It is computed as follows:

$$\text{ActivityLevel} = (\text{AvailableMen} + \text{DeliveringMen}) / (\text{TotalMen}) \times 100$$

With this structure, it implements the following operation:

¥ `DeliverymenDatabase#analyzeDeliverymenStatus()` #N#Retrieves a filled record card containing all the relevant statistics regarding deliverymen statuses

This feature requires a high degree of interaction between the `Status` component and `Statistics` component of `DeliverymenDatabase`. This interaction of the relevant classes are illustrated in the class diagram below.



*Figure 1. Partial Structure of Deliverymen Database*

The `DeliverymenDatabase` acts as an agent for both components to parse information around. This helps to maintain a clear separation of concerns to carry out different functions.

The following sequence diagram shows how the `stats` operation works:

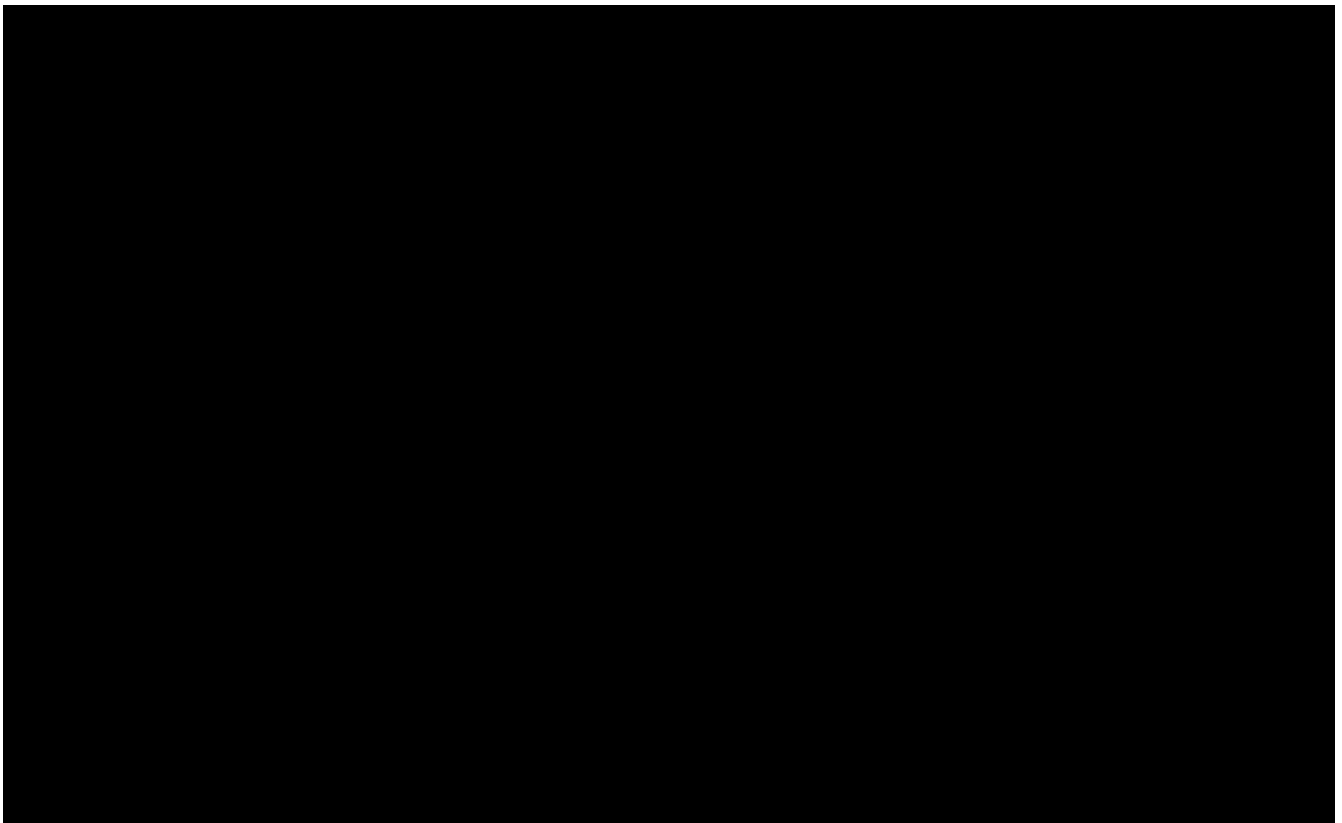


Figure 2. Sequence Diagram of Retrieving Deliverymen Statistics

As seen in Figure 10, for every use of the `stats` command, a `RecordCard` object is created. The `Analyzer` will take in the status lists and the `RecordCard` to compute and record the results onto the `RecordCard`. Subsequently, it will return the edited `RecordCard` to the `StatisticsManager`.

#### NOTE

The above explanation and sequence diagram (Figure 10) only illustrates what happens inside the `Model` component of the architecture when the `stats` command is used. The higher-level components such as `LogicManager` are left out, even though they are also involved in the execution of this command.

## Design Considerations

Aspect: Analysing the status lists

This feature necessitates the analysis of the status lists. A consideration is the design of the architecture in which the statistics feature should be implemented. It may be subtle in the current version of DeliveryMANS, but will have significant implications when the app is further developed.

¥ Alternative 1(current choice): Create another class (`StatisticsManager`) to deal with all the statistics-related methods.

! Pros: Allows for a more structured and object-oriented way of executing out the `stats` command. This makes it easier to extend additional statistics-related enhancements in future, such as adding predictive capabilities to predict the utilisation level over the next few weeks.

! Cons: More inefficient as the status lists from `StatusManager` have to be passed through different classes before reaching `StatisticsManager`. This can affect the runtime of the

command when the enhancements get more complicated.

¥ Alternative 2: Modify the existing `StatusManager` to handle statistics computation as well.

! Pros: More convenient as the status lists can be directly obtained from the same class instead of having to pass it around.

! Cons: Harder to manage in future as the app is improved due to coupling between status management and statistics management. Changes to status-related issues may cause regressions to the statistics feature.

*End of extract*

Ê

Also, I also wrote a test case for one of my implemented commands in the Developer Guide.

*Start of extract*

## Switching the status of a deliveryman

1. Switching the deliveryman status from AVAILABLE to UNAVAILABLE, or vice versa.
  - a. Prerequisites: Currently in deliverymen context. The customers in the database are default from the Sample Util data. The current status of the deliveryman is either AVAILABLE or UNAVAILABLE; it cannot be DELIVERING.
  - b. Test case: `swi tch 1`  
(NOTE: For default data, the first deliveryman is 'Damith' with the status AVAILABLE)  
Expected: The status of the first deliveryman, 'Damith', has been switched to UNAVAILABLE.
  - c. Test case: `swi tch 2`  
(NOTE: For default data, the second deliveryman is 'Donald Trump' with the status DELIVERING)  
Expected: The status of the second deliveryman, 'Donald Trump', remains unchanged. Error details shown in the status message.
  - d. Test case: `swi tch 0`  
Expected: No change in any deliveryman status. Error details shown in the status message.

*End of extract*