

# ArchDuke - Developer Guide

## Table of Contents

Preface .....	1
About ArchDuke .....	1
Document Change History .....	2
1. Setting up .....	2
1.1. Prerequisites .....	2
1.2. Setting up the project environment on your local machine. ....	3
1.3. Verifying the setup .....	3
2. Design .....	3
2.1. Architecture Design .....	3
2.2. View Layer .....	5
2.3. Controller Layer .....	5
2.4. Repository layer .....	6
2.5. Data Layer .....	6
3. Implementation .....	6
3.1. CRUD functionality for Projects .....	6
3.2. CRUD functionality for Tasks .....	8
3.2. CRUD functionality for Reminder [Plan to do] .....	9
3.3. CRUD functionality for Task Assignments .....	10
3.2. Logging .....	12
4. Documentation .....	12
5. Testing .....	12
6. Dev Ops .....	12
Appendix A: Product Scope .....	12
Appendix B: User Stories .....	12
Appendix C: Use Cases .....	15
Appendix D: Non Functional Requirements .....	19

By: **CS2113-T13-1** Since: **Aug 2019** Licence: **MIT**

## Preface

### About ArchDuke

Team leaders often face problems managing multiple projects, especially software development projects where the usage of a Kanban board is common.

Having to manage multiple Kanban boards for each individual project is a huge hassle for team leaders. As such, ArchDuke solves this issue by allowing team leaders to manage

#### 1. Team members

- ArchDuke allows users to assign tasks to team members, and track their workload, progress on tasks, and contributions.

#### 2. Tasks in a group project

- ArchDuke allows users to track multiple tasks within a project. Tasks can be created and assigned to members.
- ArchDuke tracks the deadlines and priority levels of tasks, to help the user organise which tasks should be assigned and completed.

#### 3. Multiple group projects

- ArchDuke allows users to track multiple group projects. New group projects can be created and a default Kanban board will be created for users to start adding tasks.

## Document Change History

Revision Date	Summary of Changes
23 October 2019	Version 1.0: Added implementation design for Assignment Controller and Project functions
22 October 2019	Version 0.9: Updated Architecture Design
15 October 2019	Version 0.8: Updated the name of project from Duke to ArchDuke
2 October 2019	Version 0.7: Updated Prerequisites and removed unwanted lines
30 September 2019	Version 0.6: Added table of contents, preface and document history
26 September 2019	Version 0.5: Updated use cases for a cleaner look
24 September 2019	Version 0.4: Added Non-functional Requirements to DeveloperGuide
23 September 2019	Version 0.3: Added user stories to DeveloperGuide
22 September 2019	Version 0.2: Added use cases to DeveloperGuide
18 September 2019	Version 0.1: Creation of DeveloperGuide based on template

## 1. Setting up

### 1.1. Prerequisites

1. **JDK 11** or above is required
2. **IntelliJ** IDE is recommended

## 1.2. Setting up the project environment on your local machine.

1. Fork and clone the forked remote on your local machine.
2. Launch IntelliJ (If you're not on the IntelliJ Welcome screen please close your existing project by going to **File > Close Project.**)
3. Set up the correct JDK version for Gradle
  - a. Click **Configure > Structure for New Projects**
  - b. Under **Project Settings** Click on **Project**
  - c. Under **Project SDK** Click **New** and point it to JDK 11 path.
  - d. Click **OK** to save the configuration
4. Click **Import Project**
5. Locate the **build.gradle** file and select it. Click **Open.**
6. Open the IntelliJ console/terminal and run the gradle command **gradlew processResource** on **Windows** or **./gradlew processResource** on **Mac/Linux** (If you are encountered a permission error: **./gradlew: Permission denied** add the executable permission to the the shell script by running **chmod 744 gradlew** in your terminal) It should finish with the **BUILD SUCCESSFUL** message. This will generate the resources required by the application and tests.

## 1.3. Verifying the setup

1. Run Duke to verify and try a few commands. (Refer here {insert link to the command page} the commands)
2. Run the **JUNIT Test/gradlew test** command to ensure that all the test case passes.

# 2. Design

## 2.1. Architecture Design

ArchDuke Architecture Diagram

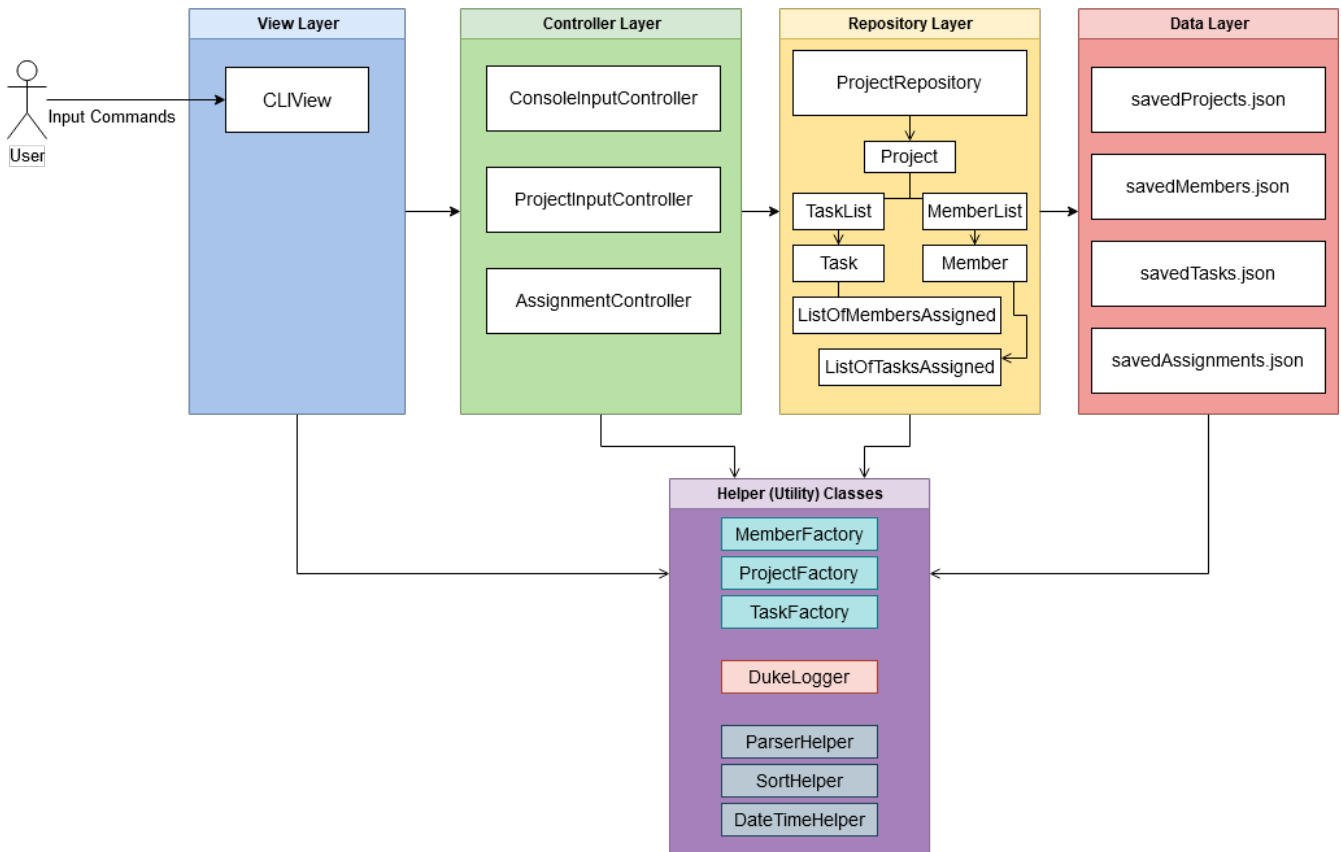


Figure 1. Architecture Diagram

The Architecture Diagram given above explains the high-level design of ArchDuke. ArchDuke is based on the n-tier architecture design. Below is a quick overview of each component based on our Architecture.

Not included in the diagram is the **Main** program in the folder **launcher**. The Main program is responsible for initializing the **View** layer.

- In our case, as ArchDuke is a command line program, **Main** will initialize **CLView**.

**Utility** represents a collection of classes used by multiple other components and can be accessed by all layers.

- **Factory** classes : Used mainly by **Repositories** and **Controllers** to create objects based on user input.
- **Logger** classes: Used by all classes to write log messages to ArchDuke's log file.
- **ParserHelper** class: Used by all classes for user user input parsing
- **SortHelper** class: Used primarily by **Repositories** and **Controllers** for sorting objects based on description before a Response model is generated for the **View** layer.
- **DateTimeHelper** class: Used for handling anything that is related to Date objects or parsing inputs for Date objects

The rest of the App consists of four main layers.

- **View** layer: The UI of ArchDuke. Responsible for printing everything that the user will see and

reading inputs from the user.

- **Controller** layer: Responsible for handling user inputs and sending them to the respective classes for parsing, cleaning, or object creation.
- **Repository** layer: Responsible for holding data in-memory
- **Data** layer: Responsible for saving and loading persistent data from hard disk.

Each layer will be discussed in detail below.

## 2.2. View Layer

Our main UI Component is a class called CLIView. It is mainly responsible for reading the user input and displaying formatted messages to the user.

When ArchDuke is running, CLIView repeatedly reads the user input line by line, and sends it to the parsing components (Controllers) to make sense of the input. Eventually, the controllers will retrieve the relevant messages and information for CLIView to display. Currently, the main CLIView is being called directly by the controllers to execute the commands by coordinating all other components.

## 2.3. Controller Layer

### 2.3.1 ConsoleInputController

### 2.3.2 ProjectInputController

### 2.3.3 AssignmentController

AssignmentController is a class which acts as a parser for commands regarding the assignment of tasks to group members.

#### Rationale for implementation

Before the implementation of AssignmentController, the parsing of assign commands was planned to be done within the ProjectInputController class. However, we realised that parsing for assignment commands would be extremely complex due to the potentially high number of arguments.

- Example command: `assign task -i 1 2 -to 1 2 3 -rm 4 5`

The task index numbers (1, 2) , assignee indexes (1, 2, 3) and unassigned indexes (4, 5) must be parsed. They should also be validated to ensure that the index numbers exist, and do not cause errors/exceptions such as `IndexOutOfBoundsException`. From the parsed input, task assignments can then be managed. Therefore, the AssignmentController was created for the following reasons:

1. To ensure that these 3 parts of the input can be easily managed
2. To isolate the assignment commands to avoid making the code in `projectInputController` too long

## 2.4. Repository layer

## 2.5. Data Layer

# 3. Implementation

This section describes in detail on how certain features of ArchDuke are implemented. Most features are based on Create, Read, Update, Delete, also known as **CRUD** functions

## 3.1. CRUD functionality for Projects

### Implementation

CRUD functions are facilitated by `ConsoleInputController`, `ProjectRepository` and `ProjectFactory`. It allows ArchDuke to be able to do some basic CRUD functions for a Project, namely only Creation, Reading and Deletion. `ConsoleInputController` will read the relevant commands from the `View` layer and call the relevant methods in `ProjectRepository`.

It implements the following commands:

- `create PROJECT_NAME` — Creation of a new Project
- `list` — Viewing all Projects that have been created
- `delete PROJECT_INDEX` — Delete a Project that has been created previously

These operations are exposed in the `IRepository` interface as `addToRepo()`, `getAll()` and `deleteItem()`.

#### NOTE

However, in order to create a object, inputs sent to the `Repository` layer must be sent to a `Factory` class as the `Repository` layer is not responsible for the creation of Objects.

The example usage scenario below will explain in detail the data flow and how the program behaves at each step of CRUD functions with regards to a Project object.

Step 1) ArchDuke is launched for the first time by the user. A new `CLIView()` and `ConsoleInputController` is created upon initialization. Immediately after initialization, `CLIView.start()` will be called which prints a welcome message to the user and awaits for user input.

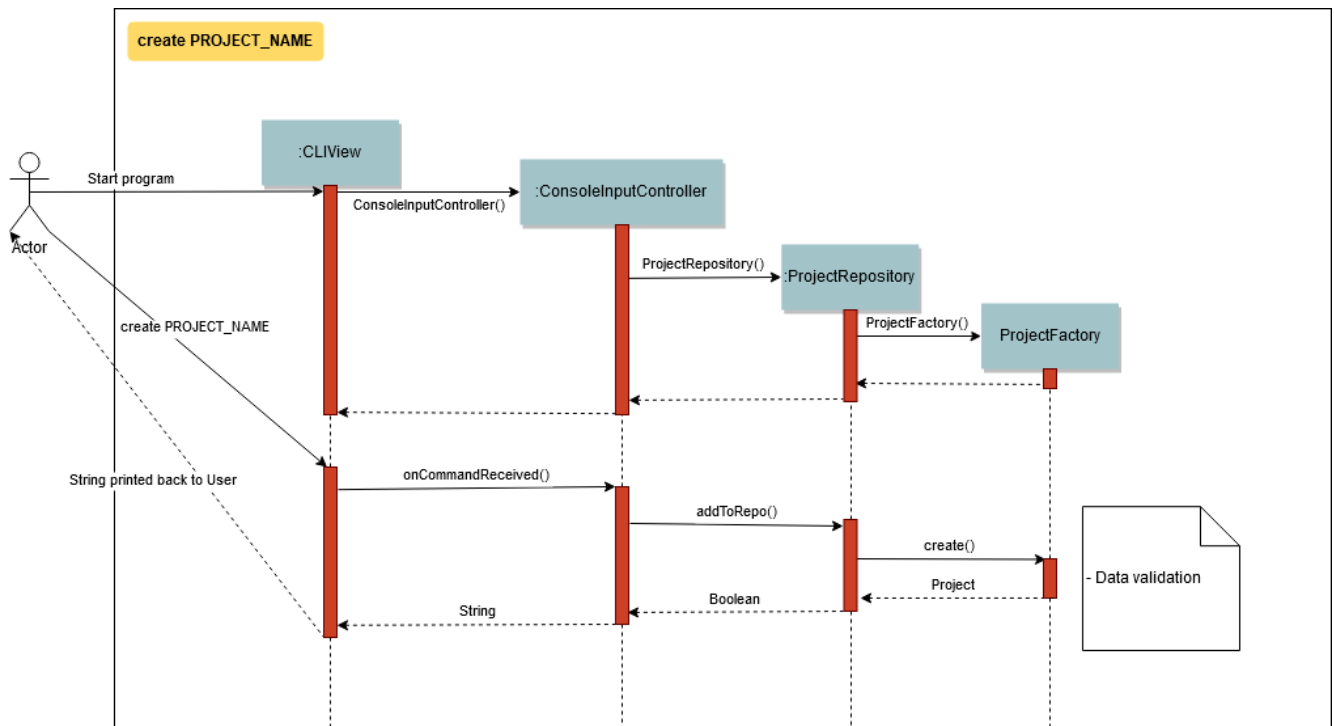
Step 2) The user executes the command `create Avengers Assemble!` to create a new Project with the description "Avengers Assemble!". User input is fed from `CLIView` to `ConsoleInputController`, where simple parsing will be done to determine the type of command that the user has executed.

Step 3) User input will be understood as a command to create a new project and thus sent to `ProjectRepository` where it will call on `ProjectFactory` for the creation of a new Project object.

Step 4) **ProjectRepository** will check if **ProjectFactory** managed to create an object successfully. Any unsuccessful creation will be due to wrong user commands or a bug during data validation in **ProjectFactory**.

Step 5) Assuming Project creation was a success, **ProjectRepository** will store it in an ArrayList and return **True** back to **ConsoleInputController** to signify the successful creation of a new Project object. **ConsoleInputController** will call **CLIView** to print appropriate messages to the user based on whether a new Project object was created successfully or not.

The following sequence diagram shows how the **create PROJECT\_NAME** operation works.

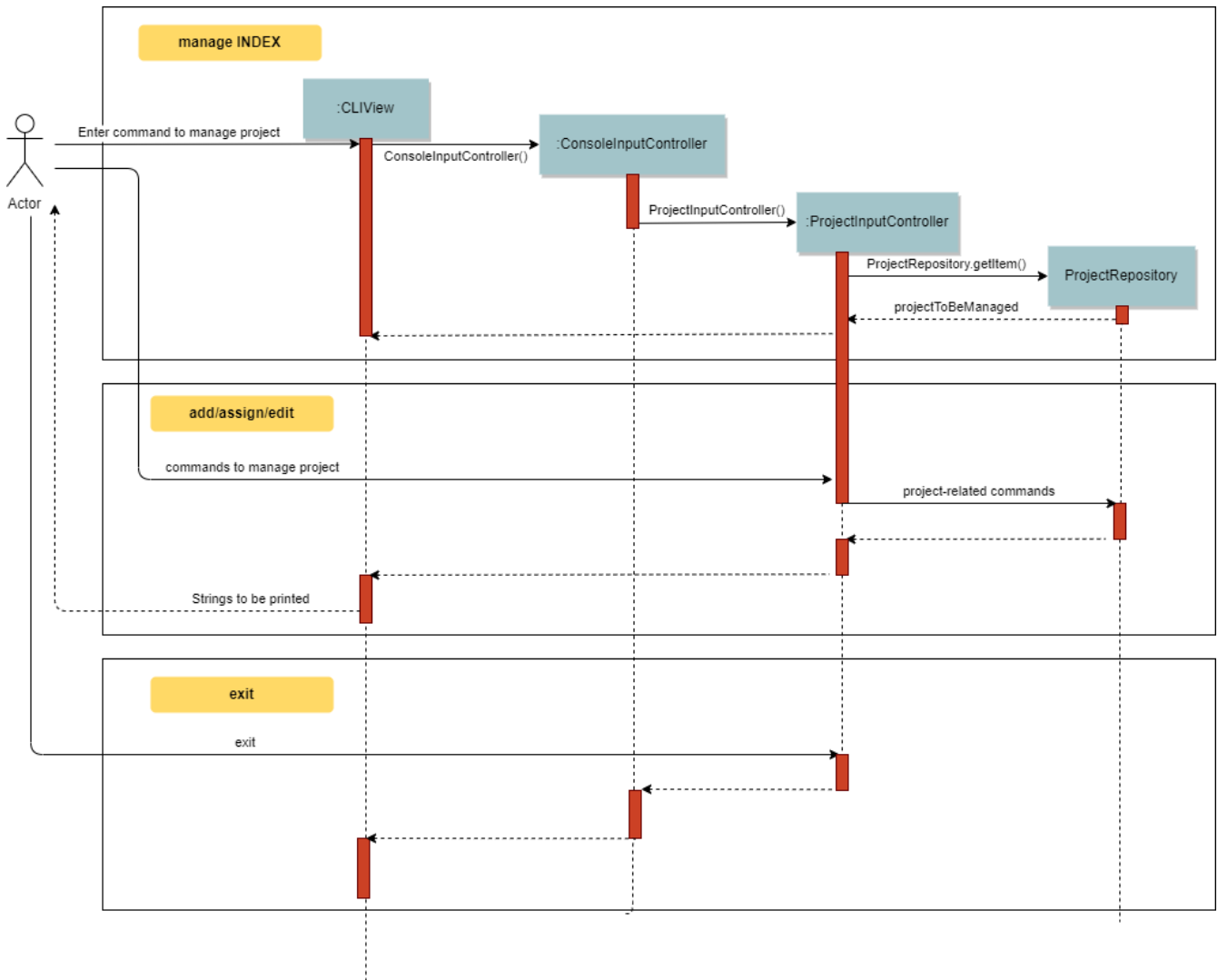


The **delete PROJECT\_INDEX** command works similarly to **create PROJECT\_NAME**. Both commands will result in a **Boolean** of either **True** or **False** to indicate whether command was executed successfully. There are minor differences, listed below:

- Instead of creating a new Project object, the **delete PROJECT\_INDEX** command will call **deleteItem()** in **ProjectRepository** instead of **addToRepo()**.
- Deletion of Project works by Project Index instead of Project Name.

## Manage project

ArchDuke allows users to manage each individual project in the **ProjectRepository**.



## 3.2. CRUD functionality for Tasks

### Implementation

CRUD Task functions are handled by `Task`, `TaskList` and `TaskFactory`, `Project` and `ProjectInputController`. It allows ArchDuke to perform simple CRUD function for Task in the Project, these simple functions include Create, Read, Update and Delete. `TaskFactory` will create the relevant task with the apporiated input from the user which will then be added into the `TaskList` managed by the `Project`. `ProjectInputController` will read the relevant command related to task function and call the relevant methods in `TaskFactory` And `ParserHelper`.

It implements the following commands:

- `add task -t TASK_NAME -p TASK_PRIORITY -c TASK_CREDIT -d TASK_DUEDATE -s TASK_STATE -r TASK_REQUIREMENT1`—Creation of a new Task with the task name, priority, credit, due date (optional), state(optional) and additional requirements (if any)
- `edit task TASK_INDEX -t TASK_NAME -p TASK_PRIORITY -c TASK_CREDIT -s TASK_STATE`—Edits existing task with the new input values
- `view tasks`—Viewing of all tasks in current project
- `view task requirements TASK_INDEX`—Viewing of all additional requirements of a specified task



- `edit task requirements TASK_INDEX rm/TASK_INDEXES r/TASK_REQUIREMENT1` — Edits task requirements of specified task by removing unwanted requirements and adding new ones
- `delete task TASK_INDEX` — Deletion of task with stated index

The example usage scenario below will explain in detail the data flow and how the program behaves at each step of CRUD functions with regards to a Task object.

Step 1) Assuming Project have been created and the user is currently managing a specific project.

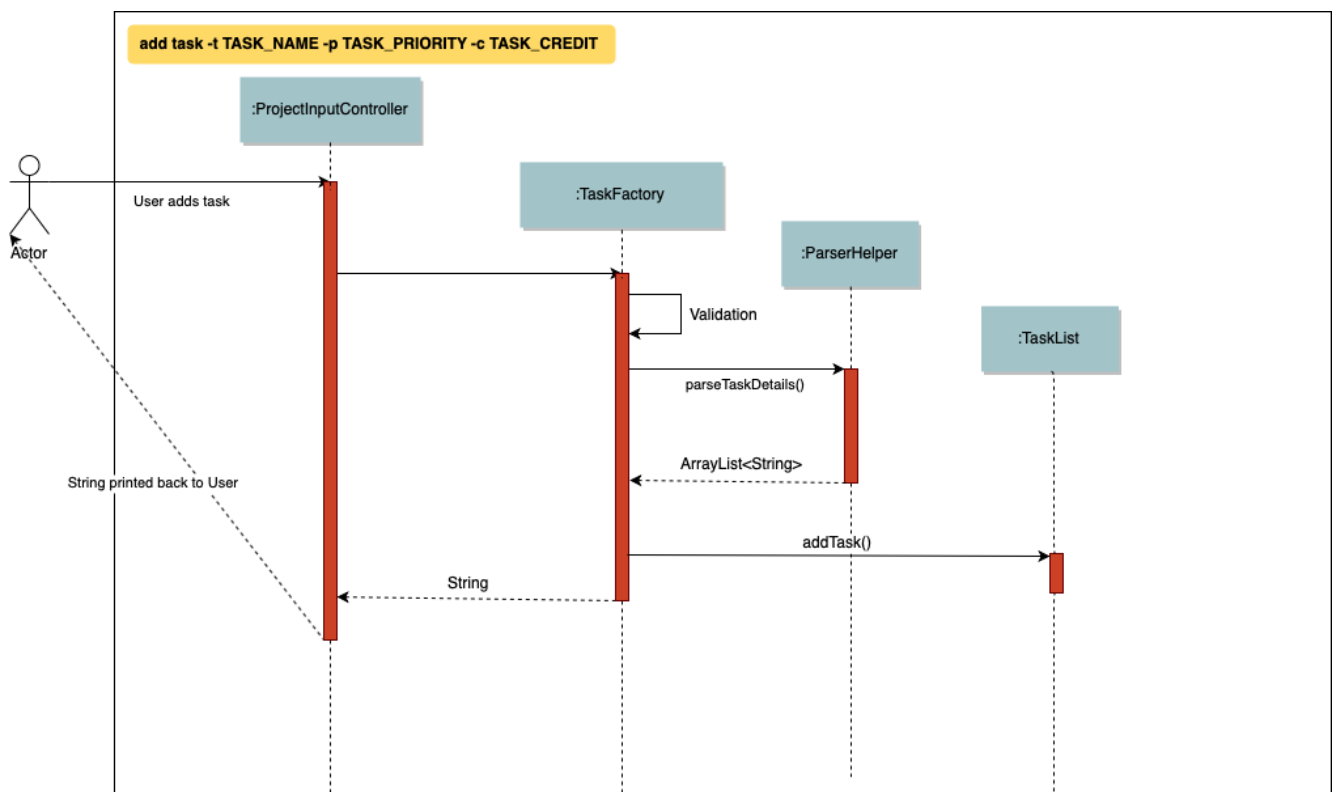
Step 2) The user execute the command `add task -t kill thanos! -p 100 -c 100` to create a new task with the task name “kill thanos!”, priority value “100” and a credit of “100”. These input will be consumed by `ProjectInputController.manageProject()`

Step 3) The `ProjectInputController.manageProject()` will trigger the `TaskFactory` which will do a validation to ensure the required input are given.

Step 4) `TaskFactory` will then call `parserHelper.parseTaskDetails()` to do a simple parsing which will clean up the flags and return `ArrayList<String>` for `TaskFactory` to create the task.

Step 5) `TaskFactory` will create the task based on the information given by the user. The created task will subsequently be added into `taskList` managed by the project and successfully or unsuccessfully a message in String will be returned.

The following sequence diagram show how `create task` operation works.



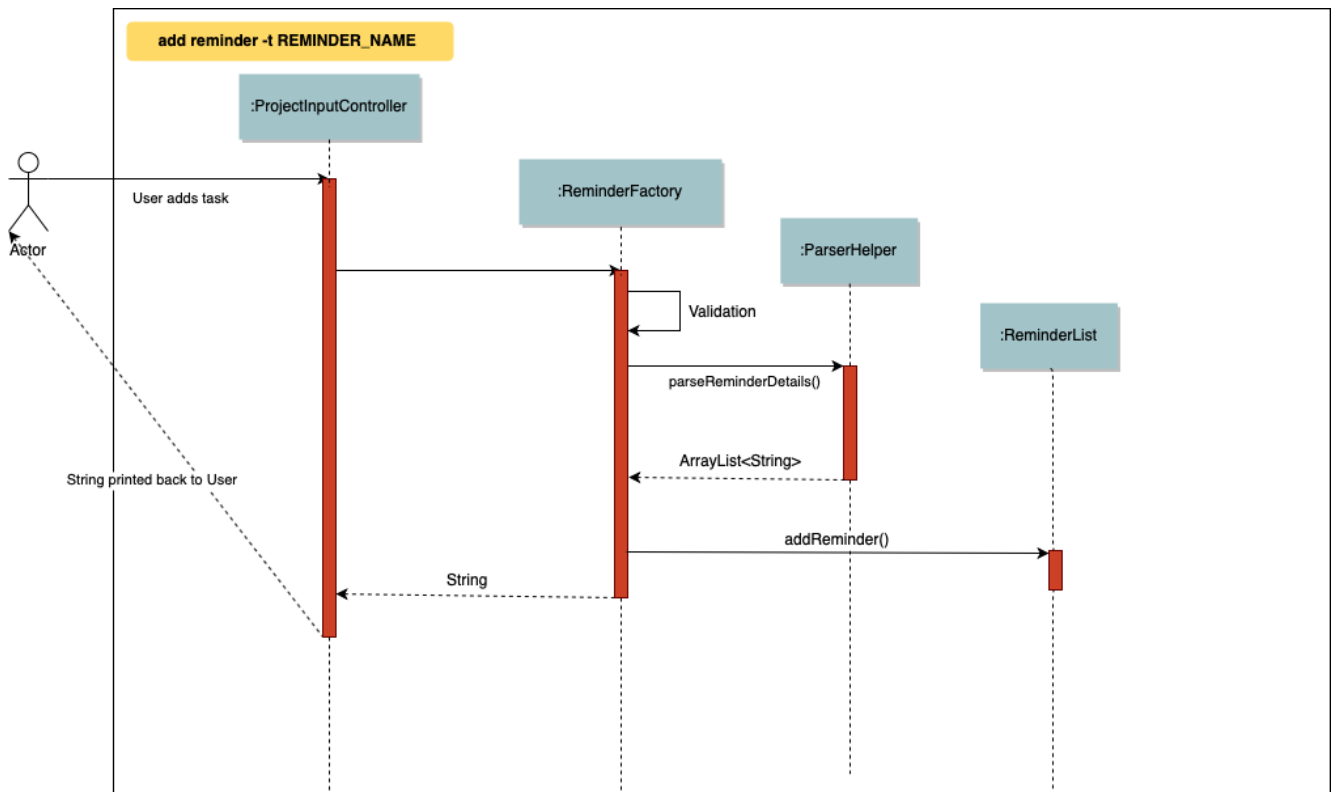
## 3.2. CRUD functionality for Reminder [Plan to do]

## Implementation

Implementation CRUD Reminder functions are handled by `Reminder`, `ReminderList` and `ReminderFactory`, `Project` and `ProjectInputController`. It allows ArchDuke to perform simple CRUD function for Reminder in the Project, these simple functions include Create, Read, Update and Delete. `TaskFactory` will create the relevant task with the apporiated input from the user which will then be added into the `TaskList` managed by the `Project`. `ProjectInputController` will read the relevant command related to task function and call the relevant methods in `TaskFactory` And `ParserHelper`.

Reminder function implements the following commands:

- `add reminder -n REMINDER_NAME --d REMINDER_DUEDATE -Tag REMINDER_TAG` — Creation of a new Reminder with the reminder name and due date (optional)
- `edit reminder TASK_INDEX -n REMINDER_NAME -d REMINDER_DUEDATE` — Edits existing task with the new input values
- `view reminder` — Viewing of all reminders in current project



## 3.3. CRUD functionality for Task Assignments

### Implementation

CRUD Assignment functions are handled by `ProjectInputController`, `ParserHelper`, `AssignmentController` and `Project`.

ArchDuke allows users to track tasks and their assignments to members in a project. Assignments establish a relationship between a task and a member. When a member is assigned a task, they are expected to complete it, and will be given the stipulated credit upon completion. The degree of each

member's contributions are measured by task credit.

Assignments are tracked in the `Project` class using 2 Java HashMaps.

- `taskAndListOfMembersAssigned`
  - Key: `Task`
  - Value: ArrayList of assigned `Member` objects (List of members assigned to task)
- `memberAndIndividualListOfTasks`
  - Key: `Member`
  - Value: ArrayList of `Task` objects (List of each member's individual tasks)

Using a HashMap for storage allows for fast and easy writing and retrieval of data. It also prevents cyclic dependencies as `Task` and `Member` objects would not have to be aware of each other (such as each class keeping an ArrayList of the assignment) in order to establish the assignment.

The following steps show how task assignments are made in ArchDuke.

Step 1) Assume Project has been created and the user is currently managing a specific project.

Step 2) User enters the command `assign task -i 1 2 -to 1 2 3 -rm 4`. This indicates the user would like to assign tasks with index number 1 and 2 to members 1, 2 and 3, and also unassign/remove the task from member 4.

Step 3) The `ProjectInputController.manageProject()` will call `AssignmentController.assignAndUnassign` to manage the assignment.

Step 4) Within the method call in `AssignmentController`, the command by the user will be parsed by the `ParserHelper` to split the input into 3 parts: the task index numbers, the assignee index numbers, and the unassignees index numbers. The index numbers will be checked to ensure that they are valid (non-negative integers, and exist within the project) with the help of `ParserHelper.parseMembersIndexes()` and `ParserHelper.parseTasksIndexes()` [yet to refactor and implement].

`AssignmentController.checkForSameMemberIndexes()` checks if the list of assignees and unassignees contain any identical index numbers, and remove them to avoid redundant work.

Step 5) All valid index numbers are stored in `AssignmentController`. Using these index numbers `AssignmentController.assignAndUnassign` manages the assigning of tasks, followed by the unassigning of tasks. A for loop is used to iterate through the tasks, and 2 separate nested for loops iterate through the assignees and unassignees.

Step 6) In each loop, `Project.containsAssignment()` is used to check if an assignment between a task and member already exists to avoid any errors (for example, duplicating assignments or trying to remove an assignment which does not exist). The errors are noted down by adding error messages to the ArrayList `errorMessages` which will be displayed to the user later.

Step 7) If the input is valid, the assignment is created by calling `Project.createAssignment()` or removed by calling `Project.removeAssignment()`. The HashMaps are manipulated accordingly to note down the assignment between the specified task and member. Success messages are stored in

`successMessages` and are also displayed to the user later.

## 3.2. Logging

We are using `org.apache.logging.log4j` package for logging. The `DukeLogger` class under `Utility` layer is used for logging every step that ArchDuke takes so that debugging will be easier.

## 4. Documentation

## 5. Testing

## 6. Dev Ops

# Appendix A: Product Scope

**Target user profile:**

- Team leaders of group projects
- Group project team leaders who monitor contributions of team members
- Project managers who track progress for multiple projects
- Teachers who evaluate and act upon their students' progress
- Technical professionals who keep track of task deadlines
- Project planners who plan out priority and assignment of roles

# Appendix B: User Stories

Priorities: High (must have) - \* \* \*, Medium (nice to have) - \* \*, Low (unlikely to have) - \*

Priority	As a ...	I want to ...	So that I can...
* * *	project leader	be able to track deadlines for each project	prioritize which project to be completed earlier
* * *	project leader	be able to manage multiple projects and view all the task delegations of my team members in them	distribute my resources appropriately
* * *	project leader	be able to create projects	keep track of all my projects using a command line application

Priority	As a ...	I want to ...	So that I can...
* * *	project leader	be able delete projects that are completed	have a cleaner working environment
* * *	project leader	view a progress bar for each project	have a clearer view of the total progress for each project
* * *	project leader	manage my team members	assign different roles to team members so that they are clear of their relevant roles
* * *	project leader	manage my team members	assign different tasks to team members so that they will be clear of the work that they are supposed to do
* * *	project leader	add team members to a specific group project	assign different roles and tasks to them based on the project they belong to
* * *	project leader	update the details of my team members	have up to date information about them whenever needed
* * *	project leader	remove members from a particular project	remove unwanted or old members from a project that they are no longer contributing
* * *	project leader	keep track of each member's progress and contributions	ensure all students contribute to their respective tasks sufficiently
* * *	project leader	find a person by name	locate details of persons without having to go through the entire list
* * *	project leader	generate a report of the contributions of the members	credit can be rightfully assigned to the respective members
* * *	project leader	indicate the credit of each task	track the level of contribution by each member
* * *	project leader	track the status of every task	track the progress of each project
* * *	project leader	indicate the priority of the tasks	ensure members know what order to be done

Priority	As a ...	I want to ...	So that I can...
* * *	project leader	input the requirements of the tasks I have been assigned	ensure needs members are clear about what needs to be done
* * *	project leader	be able to track deadlines for each task	I will know if a group member is slacking.
* * *	project leader	assign one task to multiple students	more than one student can contribute to the task
* * *	project leader	delete erroneous tasks from the project I am managing	prevent any confusion and achieve a cleaner work space
* * *	project leader	view tasks sorted by name, index, date, priority, credit, assigned member names or Kanban board style	I can view all the tasks in a customised manner according to the required scenario.
* * *	project leader	import files from other sources	track all my projects from different workstations or work environments
* * *	project leader	be able to edit and read the exported file	have other people can add in details as well into the file and send back
* * *	project leader	have a good overview of all the projects I am managing	I can distribute my resources appropriately
* * *	project leader	define a clear end goal/target for the project	that we stay on the right track
* * *	project leader	I can view the tasks and roles that I have assigned	ensure members can complete them
* * *	project leader	keep track of the contributions of members	ensure everyone does his/her fair share
* * *	project leader	create task dependencies	members can complete tasks in a certain order
* * *	project leader	calculate the total weightage of tasks done by each member	keep track of the amount of contributions done by each member
* *	project leader	schedule project meet-ups	group members can meet at a stipulated date and time

Priority	As a ...	I want to ...	So that I can...
* *	project leader	be able to export the relevant details for each project	send it to other people for viewing
* *	project leader	be able to export the details for each project in different formats	avoid compatibility issues with a specific file format
* *	project leader	view a calendar with all tasks, milestones and deadlines	easily visualise the progress of the project
* *	project leader	have a more intuitive way to view the current task and role assigned to a particular team member	better manage their well being
* *	project leader	be able to track the technical and non-technical roles assigned to my team members	I can keep track of the overall progress of the project
* *	project leader	define milestones to track the progress of the entire project	have users work towards each milestone sequentially
*	project leader	save time managing my team members from the manual way of tracking my group progress	make this group can be as efficient as possible
*	project leader	change the assignment of tasks halfway through the project	cater to different needs and schedules of team members

*{To be edited}*

## Appendix C: Use Cases

(For all use cases below, the **System** is the ArchDuke and the **Actor** is the user, unless specified otherwise)

### Use case: Create project (UC01)

MSS

1. User requests to create project with desired project name and number of members

2. ArchDuke creates a project named after desired project name and number of members

Use case ends.

#### **Extensions**

- 1a. The given input is wrong.

- 1a1. ArchDuke shows an error message.

Use case ends.

## **Use case: View all projects (UC02)**

#### **MSS**

1. User requests to view all projects.
2. ArchDuke shows a list of all projects with their respective details.

Use case ends.

#### **Extensions**

- 1a. The given input is wrong.

- 1a1. ArchDuke shows an error message.

Use case ends.

## **Use case: Manage a project (UC03)**

#### **MSS**

1. User requests to view all projects.
2. ArchDuke shows a list of all projects with their respective details.
3. User requests to manage a project specified in the list.
4. ArchDuke opens up the specified project.

Use case ends.

#### **Extensions**

- 2a. The list is empty.

Use case ends.

- 3a. The given index is invalid.

- 3a1. ArchDuke shows an error message.



Use case resumes at step 2.

## Use case: Add members to a specific project (UC04)

### MSS

1. User selects a specific project to manage (UC03).
2. User requests to add member specifying name, phone number and email address.
3. ArchDuke adds specified member into current project.

Use case ends.

### Extensions

- 2a. The given information is invalid.
  - 2a1. ArchDuke shows an error message.

Use case resumes at step 1.

## Use case: Edit members in a specific project (UC05)

### MSS

1. User selects a specific project to manage (UC03).
2. User requests to edit member specifying member index and fields that require editing.
3. ArchDuke edits specified fields of specified member in current project.

Use case ends.

### Extensions

- 2a. The given index is invalid.
  - 2a1. ArchDuke shows an error message prompting user to check again and enter the correct index.

Use case resumes at step 1.

## Use case: Add task in a specific project (UC06)

### MSS

1. User selects a specific project to manage (UC03).
2. User requests to add task.
3. ArchDuke adds task to current project.

Use case ends.

## Use case: Edit task in a specific project (UC07)

### MSS

1. User selects a specific project to manage (UC03).
2. User requests to edit task specifying task index and fields that require editing
3. ArchDuke edits specified fields of specified task in current project.

Use case ends.

### Extensions

- 2a. The given index is invalid.
  - 2a1. ArchDuke shows an error message.

Use case resumes at step 1.

## Use case: Assign tasks to members (UC08)

### MSS

1. User selects a specific project to manage (UC03).
2. User requests to assign a specific task to one or several members.
3. ArchDuke assigns specified members to specified task in current project.

Use case ends.

## Use case: Complete tasks in a specific project (UC09)

### MSS

1. User selects a specific project to manage (UC03).
2. User requests to mark a specific task as completed.
3. ArchDuke marks specified task in current project as completed.

Use case ends.

### Extensions

- 2a. The given index is invalid.
  - 2a1. ArchDuke shows an error message.

Use case resumes at step 1.

## Use case: Generate report for a specific project (UC10)

### MSS

1. User selects a specific project to manage (UC03).
2. User requests to generate a report of the project and members' contributions.
3. ArchDuke gathers information from tasks, and presents it in a report.
4. ArchDuke saves a copy of the report in a readable format to the hard disk.

## Appendix D: Non Functional Requirements

1. ArchDuke should be able to run on any machine with Java Development Kit (JDK 11) installed.
2. ArchDuke should be able to handle up to a thousand tasks and projects.
3. ArchDuke should be secure, to prevent unauthorised modification.
4. ArchDuke should not save passwords in plain text.
5. ArchDuke should be smooth and fast to view and edit.
6. ArchDuke output should be organised clearly with proper tabbing.