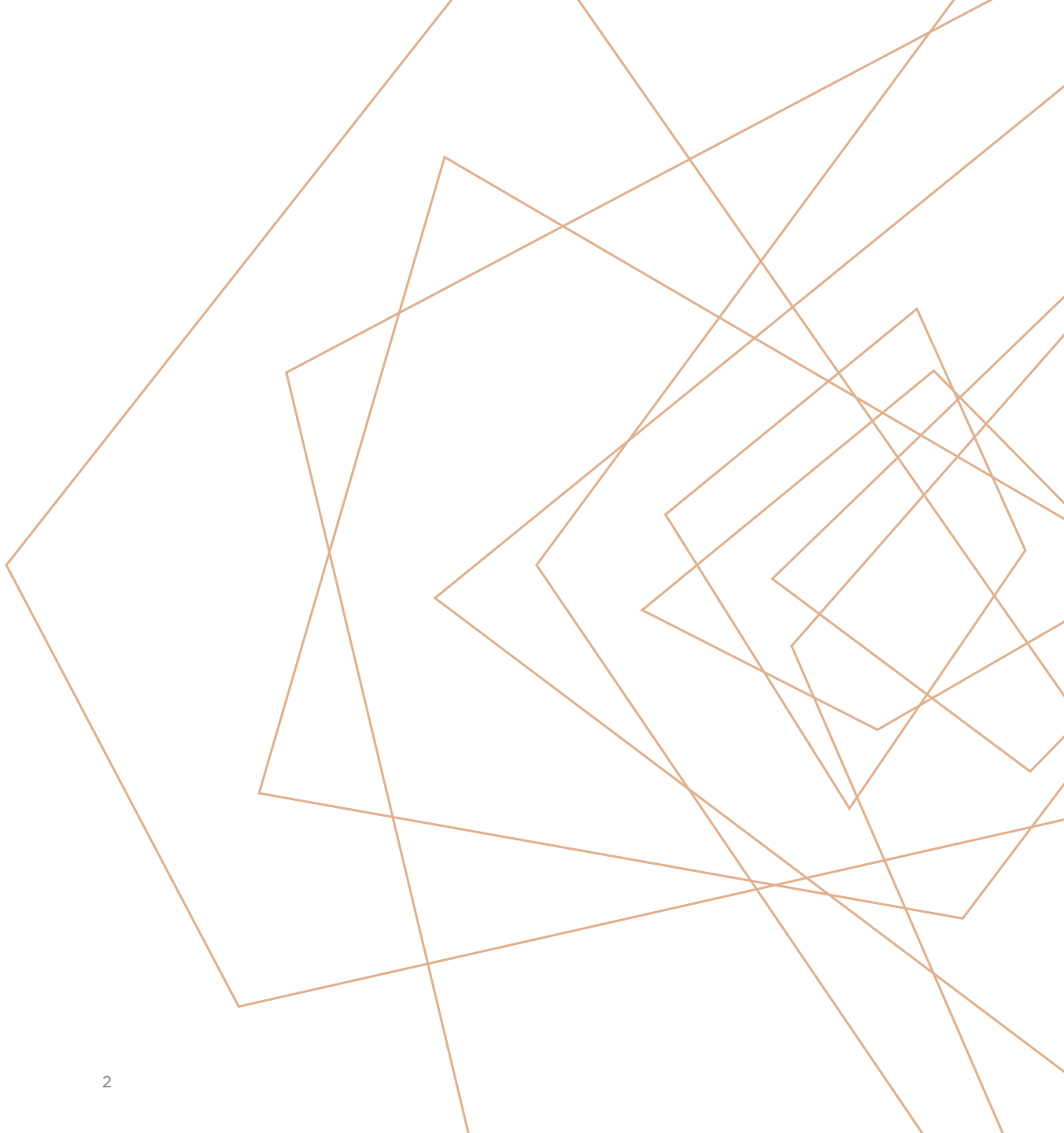
ROSTERBOARD

My Pham (Ellen) | T1A3 Terminal Application

APPLICATION OVERVIEW

NKG Corp. is a warehousing and distribution company. They have large number of staff working across rotating shifts. **RosterBoard** is the company internal application that allows company staff to provide their availability and unavailability, which are used for roster building and workforce planning.



MAIN FEATURES

CREATE ROSTER

Users are asked to provide their availability for the following week – the requirement to form a roster is to be available for at least THREE days and only ONE shift per day.

ADD UNAVAILABILITY

Users are asked to provide their unavailability for the week after the following week – there is no limit for adding unavailable days and shifts.

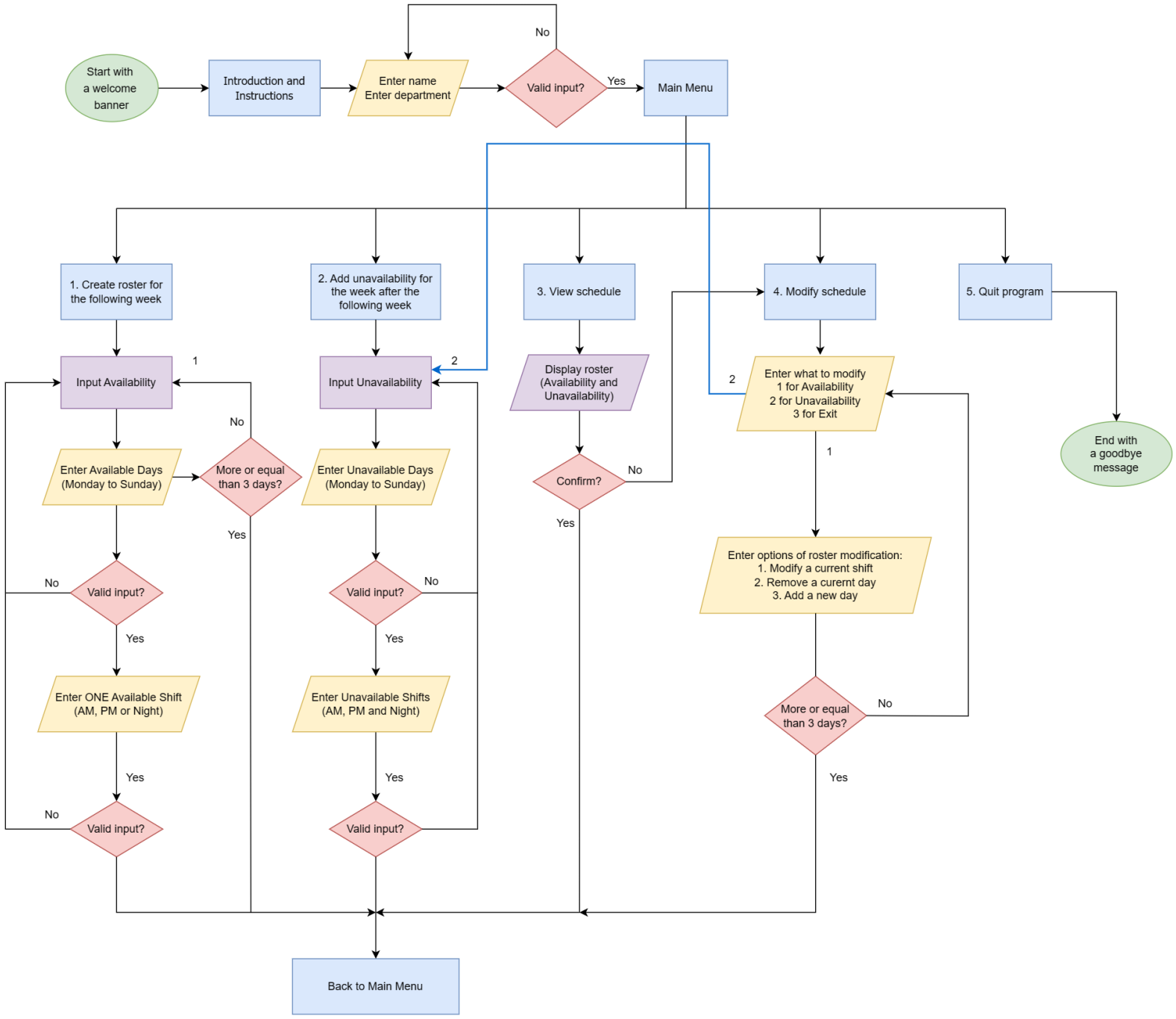
VIEW SCHEDULE

Users can view their final work schedule (including both availability and unavailability) and are required to confirm the schedule. If users do not wish to confirm, they have option to modify their current work schedule.

MODIFY SCHEDULE

Options for modifying roster include changing a current shift, removing a current day or adding a new day. For modifying unavailability, users will have to redo the unavailability from the scratch if they do not wish to confirm their current one.

OVERALL STRUCTURE



SOURCE CODE FILES ARRANGEMENT

Files	Content
main.py	input_name(), department_choice(), main_menu()
Roster.py Unavailability.py	load_from_file(), save_to_csv(), display_roster()
Item.py	from_str()
common_functions.py	functions for text styling and a function for checking valid shift input.
create_roster_function.py	create_roster() → imported in main.py as Prompt 1
add_unavailability_function.py	add_unavailability() → imported in main.py as Prompt 2
view_roster_function.py	view_schedule() → imported in main.py as Prompt 3.
modify_roster_function.py	modify_schedule() → imported in main.py as Prompt 4

EXPLAINING SOURCE CODE

Roster class

```python
# Define Roster class that represents the roster
class Roster:
    def __init__(self):
        self.roster = []

    # Load data from file to a list
    def load_from_file(self, file_name):
        try:
            with open(file_name,"r") as file:
                csv_reader = csv.reader(file)
                data = list(csv_reader)
                for i in range(1, len(data)):
                    item = Item.from_str(data[i])
                    self.roster.append(item)
        except FileNotFoundError:
            print(stylize("Error found! There is no file to be read.", warning_color()))
        except Exception:
            print(stylize("It seems like something went wrong!", warning_color()))
```

```python
# Write to csv file
def save_to_csv(self, file_name):
    try:
        with open(file_name, "w") as file:
            csv_writer = csv.writer(file)
            csv_writer.writerow(["Rostered Day", "Shift", "Action"])
            for item in self.roster:
                full_date = item.day.strftime("%a %d/%m/%Y")
                csv_writer.writerow([full_date, item.shift, item.action])
    except Exception:
        print(stylize("Something happens! Data can not be saved to file.", warning_color()))
```

Item class

```python
# Define the Item class that represents an item object
class Item:

    # initialize an Item object from input parameters
    def __init__(self, day, shift, action):
        self.day = day
        self.shift = shift
        self.action = action

    # Initialize an Item object from an input string
    @classmethod
    def from_str(self, str_data):
        try:
            str = str_data[0]
            str = str[4:]
            str = str.split("/")
            date = int(str[0])
            month = int(str[1])
            year = int(str[2])
            day = datetime.date(year,month,date)
            shift = str_data[1]
            action = str_data[2]
            item = Item(day, shift, action)
            return item
        except ValueError:
            invalid_input_message()
```

Sun 07/05/2023, AM, Added

2023-05-07, AM, Added

create_roster() function

Screenshot 1

```python
# Initialize roster object
users_roster = Roster()

# Start looping users for selection until they hit Q to quit
while user_day_selection:
    available_day = input("Please select your available day: ")

    """
    user_day_selection = False in two cases:
    1. Three or more days (no duplication) were chosen before users hit Q
        --> Successfully created roster --> Back to Home Menu
    2. users choose two available days but do not want to continue to add more days
        --> Any previous data are cleared, no roster --> Back to Home Menu

    """

    if (available_day == "Q"):
        # When users hit Q, count items in the roster list
        # If more than 3 days then users have completed creating roster

        if len(users_roster.roster) >= 3:
            user_day_selection = False
            users_roster.save_to_csv(file_name)
            print("-" * 130)
            print("\nTHANK YOU! You have completed your roster for the following week.")
            print("You have the option to view your roster again in the Home Menu.\n")
            print("-" * 130)
            print("\n")
            break

        # If less than 3 days, notify users of criteria for creating roster
        else:
            print(stylize("--> You are required to be available for at least THREE days. Do you want to CONTINUE to select more days?", warning_color()))
```

Screenshot 2

```python
        else:
            print(stylize("--> You are required to be available for at least THREE days. Do you want to CONTINUE to select more days?", warning_color()))

            # Keep looping until users enter invalid answer
            # If yes, users are prompted back to continue add more days
            # If no, break out of main loop, no roster recorded
            while True:
                continue_or_not = input(stylize(f"--> Enter 'Yes' to continue or 'No' to quit: ", warning_color()))
                if continue_or_not == "No":
                    user_day_selection = False
                    print("-" * 110)
                    print("You have no roster for the following week.")
                    print("If you need further discussion, please contact our HR department on 1300 123 456.")
                    print("-" * 110)
                    break
                elif continue_or_not == "Yes":
                    break
                else:
                    invalid_input_message()

    # If a chosen day is in the days dictionary
    elif available_day in days_dict:
        # Check if the day is already chosen to prevent duplication
        if available_day in selected_day:
            print(stylize("--> Sorry you have selected this day! You can only select ONE shift per day.", warning_color()))
            continue
        else:
            selected_day.append(available_day)

        # Shifts selection comes after a day is successfully chosen
        while True:
            available_shift = input("Enter your available shift (AM, PM or Night): ")
            # If not valid shift input, keep looping
            if not check_valid_shift(available_shift):
                invalid_input_message()
            # If valid shift input, the item is good to add to the roster list
            else:
                new_item = Item(days_dict[available_day], available_shift, action = "Added")
                users_roster.roster.append(new_item)
                day_str = days_dict[available_day].strftime("%a %d/%m/%Y")
                print(stylize(f'--> {day_str} - {available_shift} is added to your roster.', notice_color()))
                break

    # Invalid input if users' days selection is not listed or is not "Q"
    else:
        invalid_input_message()
```

modify() function

```python
# Function to change a current rostered day's shift
def modify(file_name):
    modified_rostered_day = str()

    # Initialize roster object
    users_roster = Roster()
    users_roster.load_from_file(file_name)
    users_roster.display_roster()

    # Create a dictionary for storing day item and its index
    currentitem_dict = dict()
    i = 1
    for item in users_roster.roster:
        currentitem_dict[str(i)] = item
        i += 1

    # Keep looping users to select until they hit Q
    while modified_rostered_day != "Q":
        print("Select the corresponding number in [] to choose the day you want to modify or enter Q to finish: ")
        modified_rostered_day = input("Enter your selection: ")

        # Break the loop if users hit Q
        if modified_rostered_day == "Q":
            break

        # If not Q, checking if the index selection is in dictionary's keys
        else:
            while True:
                # If yes, users can start change the current shift
                if modified_rostered_day in currentitem_dict.keys():
                    modified_shift = input("Enter the shift you want to change to (AM, PM or Night): ")
                    # If shift input is valid, form a new day item then save new record to csv file
                    if check_valid_shift(modified_shift):
                        currentitem_dict[modified_rostered_day].shift = modified_shift
                        currentitem_dict[modified_rostered_day].action = "Modified"
                        users_roster.save_to_csv(file_name)
                        # Convert the date format to display a notice
                        str_day = currentitem_dict[modified_rostered_day].day.strftime("%a %d/%m/%Y")
                        print(stylize(f'--> Your shift on {str_day} has been changed to {modified_shift}.', notice_color()))
                        break
                    # Invalid shift input
                    else:
                        invalid_input_message()
```

CHALLENGES

My normal algorithm thinking is still procedural programming. Not until almost all app features were coded, then I realized there were a few repetitive part of code. I redesigned the code structure and apply OOP, which made data manipulation becomes easier and code is reusable.

I still find it hard to make the code “dry”. My abstractive thinking is yet to develop since this is my first time to approach a programming language. More exposure to various code challenges and practice is crucial.

It is still a challenge to find a relevant test case. It depends on how we design our code to facilitate testing process, especially unit test.

Overall, I really embraced the whole development process. Many concepts have come to realization and things that I’ve learnt from the course are gradually making more sense to me.

THANK YOU.

My Pham (Ellen)

Coder Academy STD-FEB-23