**FlyAway (An Airline Booking Portal)**

**Project objective:**

As a Full Stack Developer, design and develop an airline booking portal named as FlyAway. Use the GitHub repository to manage the project artifacts.

**Background of the problem statement:**

FlyAway is a ticket-booking portal that lets people book flights on their website.

**The website needs to have the following features:**

● A search form in the homepage to allow entry of travel details, like the date of travel, source, destination, and the number of persons.  
● Based on the travel details entered, it will show the available flights with their ticket prices.  
● Once a person selects a flight to book, they will be taken to a register page where they must fill in their personal details. In the next page, they are shown the flight details of the flight that they are booking, and the payment is done via a dummy payment gateway. On completion of the payment, they are shown a confirmation page with the details of the booking.   

For the above features to work, there will be an admin backend with the following features:

● An admin login page where the admin can change the password after login, if he wishes  
● A master list of places for source and destination  
● A master list of airlines  
● A list of flights where each flight has a source, destination, airline, and ticket price  
       
The goal of the company is to deliver a high-end quality product as early as possible.

**The flow and features of the application:**

● Plan more than two sprints to complete the application  
● Document the flow of the application and prepare a flow chart   
● List the core concepts and algorithms being used to complete this application  
● Implement the appropriate concepts, such as exceptions, collections, and sorting techniques for source code optimization and increased performance

**You must use the following:**

● Eclipse/IntelliJ: An IDE to code for the application   
● Java: A programming language to develop the web pages, databases, and others  
● SQL: To create tables for admin, airlines, and other specifics  
● Maven: To create a web-enabled Maven project  
● Git: To connect and push files from the local system to GitHub   
● GitHub: To store the application code and track its versions   
● Scrum: An efficient agile framework to deliver the product incrementally   
● Search and Sort techniques: Data structures used for the project   
● Specification document: Any open-source document or Google Docs

**Following requirements should be met:**

● The source code should be pushed to your GitHub repository. You need to document the steps and write the algorithms in it.  
● The submission of your GitHub repository link is mandatory. In order to track your task, you need to share the link of the repository. You can add a section in your document.   
● Document the step-by-step process starting from sprint planning to the product release.   
● The application should not close, exit, or throw an exception if the user specifies an invalid input.  
● You need to submit the final specification document which will include:   
● Project and developer details   
● Sprints planned, and the tasks achieved in them   
● Algorithms and flowcharts of the application   
● Core concepts used in the project   
● Links to the GitHub repository to verify the project completion

GitHub REPO:  
<https://github.com/catiafsantos/PhaseIProjectDelivery.git>

* All relevant information, such as this file, printscreens, the code, etc... is provided on this REPO.

Project Name: PhaseIIFlyAwayAirlineProject

Project Developer: Cátia Santos ([catia.santos3@vodafone.com](mailto:catia.santos3@vodafone.com))

Company Name: Vodafone

**Sprint Planning:**

The following information is not accurate, the project and the tasks did not take the time that is being mentioned bellow, the following is being provided as data to implement the Agile knowledge acquired during the course.

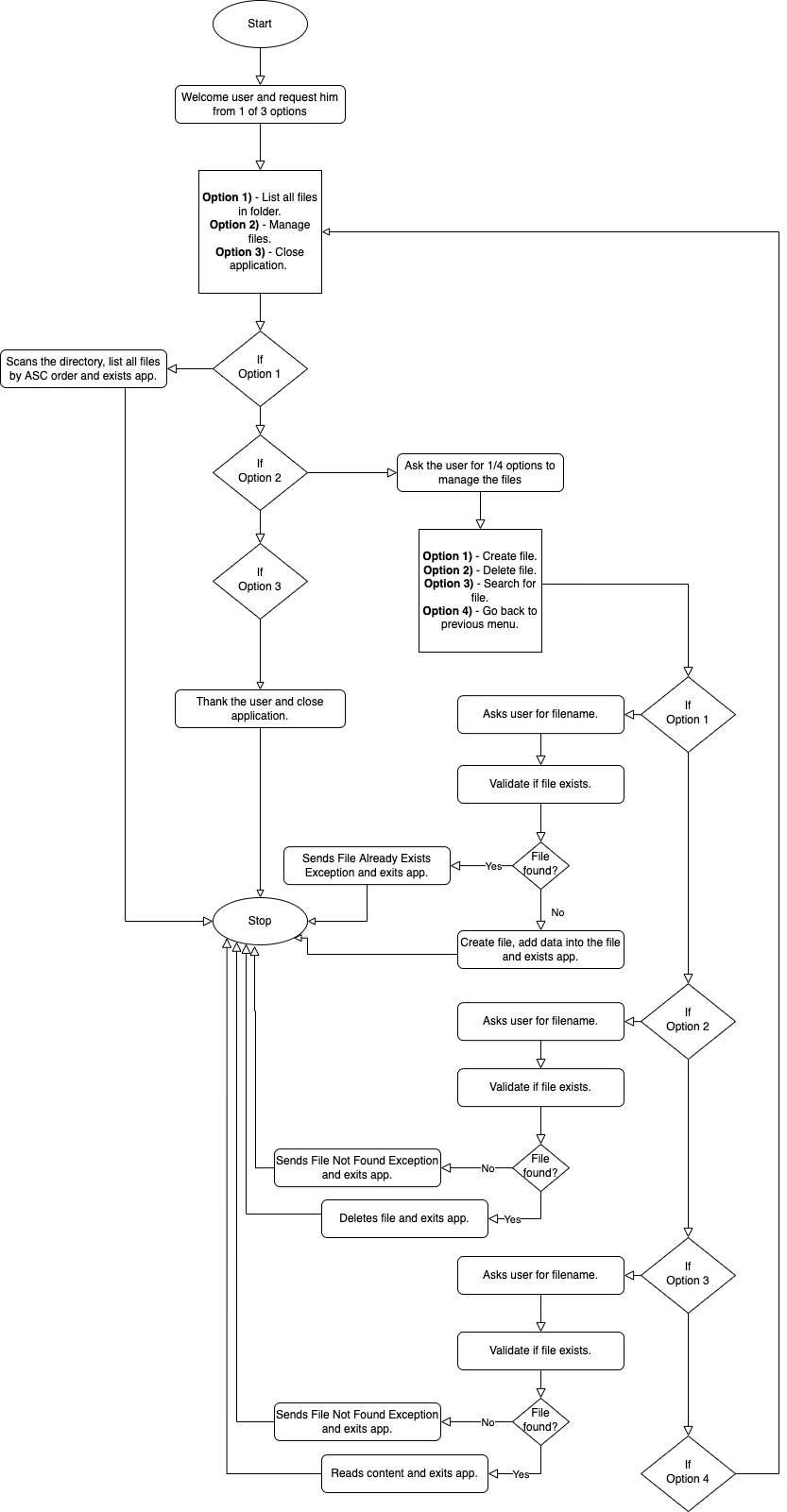
* Hereby the Sprint Tasks and estimates in story points, let’s assume that the average is 10 SPs per sprint:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprint** | **Task Code + Title** | **Simple Task Description** | **Estimation (SP)** |
| 1 | TASK-134 – Env Prep | Download the required tools, such as IDE, Java, JDK, Git, GitHub etc.  Install the software | 1 |
| 1 | TASK-156 – Project Design | Design the application flowcharts, algorithms and behaviors and document them. | 2 |
| 1 | TASK-904 – Menu creation | Create console Menu with user input for 3 options, one with 3 sub-options  Sub-options need to allow for user to input filename. | 2 |
| 1 | TASK-905 – Menu testing | Test the menu options with user input and console logging. | 1 |
| 1 | TASK-855 – Project Setup | Create the project in java  Create the directory of the files  Setup the Repository in the GitHub and link local with remote. | 1 |
| 1 | TASK-1440 – Create File | Create the code to add a file according with the filename provided in the user input.  Write data into the file previously created.  Return exception or successful message according with behavior. | 3 |
| 2 | TASK-1449 – Delete File | Create the code to delete a file according with the filename provided in the user input.  Return exception or successful message according with behavior. | 3 |
| 2 | TASK-1169 – Search File | Create the code to search for a file according with the filename provided in the user input.  Return exception or successful message according with behavior.  If file is found read its contents to the user. | 4 |
| 2 | TASK-1169 – List Files | Create the code to list all files in the directory by ASC order. | 3 |
| 3 | TASK-1181 – Exit Application | Create the code to allow user to exit application. | 2 |
| 3 | TASK-1187 – Create main code | Create the code to join the user options with the methods previously created. | 2 |
| 3 | TASK-1194 – Testing | Test the code functionalities - test all inputs and options. | 1 |
| 3 | TASK-1200 – Documentation | Document the test behaviors with examples.  Document the algorithms used. | 1 |

So, the conclusion is that it will take around 2 full sprints and 6 more SPs (story points) which is roughly half more sprint, therefore 2/5 sprints.

Flowchart:

* Flowchart was designed used diagram.io online.



Technologies/Tools used:

The following technologies were used on the creation of this project:

* Java 11 – OpenJDK Temurin 11 + SDK 11
* Git version 2.31.1
* GitHub
* Mac OS Monterey Version 12.1
* IDE – IntelliJ IDEA 2021.2 Community Edition
* Microsoft Word for Max OS X – Version 16.57
* Diagram.io Online - <https://app.diagrams.net/>

Algorithms used:

The following algorithms were written for this project:

I decided to divide the project into two classes, the fileHandling() one that contains:

* **createFile()** – This algorithm was created to add a file as per user request.
  + Uses the filename provided by the user on the mainMenu() class
    - Uses a if/else statement with file.exists() to identify if the file already exists and if so, fails saying that the file already exists and cannot be created, exiting the application.
    - If the file does not exist, a file is created using Files.write() with the filename provided by the user.
  + The filename is case insensitive as requested by the description of the project.
  + To get the file the Paths.get() method is being used.
  + Additionally, I added the functionality of writing something hardcoded onto the file using the FileWriter() class.

* **deleteFile()** – This algorithm was created to delete a file as per user request.
  + Uses the filename provided by the user on the mainMenu() class
    - If the file does not exist, an exception is thrown using a try()/catch() saying that the file requested does not exist therefore cannot be deleted, exiting the application.
    - If the file exists, the file gets deleted using Files.delete() and a success message is shown, exiting the application.
  + To get the file the Paths.get() method is being used.
* **searchFile()** – This algorithm was created to search for a file as per user request.
  + Uses the filename provided by the user on the mainMenu() class
    - If the file does not exist, an exception is thrown using a try()/catch() saying that the file request does not exist therefore cannot be retrieved, exiting the application.
    - If the file exists, the content of the file is shown using the readFile() method.
  + To get the file the Paths.get() method is being used.
* **readFile()** – This algorithm was created to read a file when the searchFile() method is successful.
  + Uses the filename provided by the user on the mainMenu() class.
  + No exceptions for file not found were added since this method is always being called by searchFile() one that already does that validation.
  + To get the file the Paths.get() method is being used.
* **orderFiles()** – This algorithm allows to get a list of all available files in the directory by ascending order.
  + Uses Files.list() with the .sorted() option to sort the files by ASC order, which is the default.
    - Uses the .forEach() to create a loop and print the file path and name.

And the mainMenu() class that contains:

* **main() –** The main class where the program goes when it gets triggered and that allows to access the other logic.
  + Gives the user some information about the application and the options available to choose from.
  + Scans the user chosen option and uses it on the switch() statement.
    - The switch() conditional statement goes through the first 3 options, and continues the program by calling the other methods at each case.
      * In case of 1 being selected, it shows a statement saying the option was chosen and triggers the orderFiles() method.
      * In case of 2 being selected, it shows a statement saying that the option was chosen and triggers the extraMenuOptions() method, allowing to access the remaining file handling options.
      * In case of 3 being selected, it shows a statement saying that the option was chosen, and it goes out of the program, exiting the application with a System.exit(0).
      * In case of any other option being chosen, it shows a statement saying that the option was invalid and exiting the application.
* **extraMenuOptions() –** This algorithm allows to access the remaining file handling options.
  + Gives the user more information about the additional file handling options available.
  + Scans the user chosen option and uses it on the switch() statement.
    - The switch() conditional statement goes through the first 3 options, and continues the program by calling the other methods at each case.
      * In case of 1 being selected, asks the user to input the filename and changes it to lower cases to be case insensitive, and uses that filename on the createFile() method.
      * In case of 2 being selected, asks the user to input the filename, and uses that filename on the deleteFile() method.
      * In case of 3 being selected, asks the user to input the filename, and uses that filename on the readFile() method.
      * In case of 4 being selected, it goes back into the main() method.
      * In case of any other option being chosen, it shows a statement saying that the option was invalid and exiting the application.

Additionally, there are common things between the two of them, such as:

* In all places where is necessary to send a message input to the console the System.out.println() method was used.
* In all places where is necessary to get user input from the console the Scanner() method was used.
* All file handling methods use try()/catch() and throws for IOException.
* All methods/classes that belong to Java existing collections were imported for use inside the written algorithms, such as Files(), FileWriter() and Paths().

Examples of the application running can be found into the GitHub Repo on the Documentation and Examples Folder.

END