

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Machhe, Belagavi, Karnataka 590018



FILE STRUCTURES LABORATORY WITH MINI-PROJECT REPORT

On

TRAVEL MANAGEMENT SYSTEM USING HASHING WITH PROGRESSIVE OVERFLOW

*Submitted in partial fulfillment of the requirement
for the award of the degree of*

Bachelor of Engineering
in
Information Science & Engineering
by
Mohammed Zaid M S (1BG19IS024)



Vidyayāmruthamashnuthe

B.N.M. Institute of Technology

An Autonomous Institution under VTU, Approved by AICTE

Department of Information Science and Engineering

2021 – 2022

B.N.M. Institute of Technology

An Autonomous Institution under VTU

**DEPARTMENT OF INFORMATION SCIENCE &
ENGINEERING**



Vidyayāmruthamashnuthe

CERTIFICATE

Certified that the Mini-project entitled **TRAVEL MANAGEMENT SYSTEM USING HASHING WITH PROGRESSIVE OVERFLOW** is carried out by **Mr. Mohammed Zaid M S** bearing USN (1BG19IS024) the bonafide student of **B.N.M Institute of Technology** in partial fulfillment for the award of **Bachelor of Engineering in Information Science & Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2021-2022. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The mini-project report has been approved as it satisfies the academic requirements in respect of mini-project prescribed for the said Degree.

Ms. Rashmi T. V
Asst. Professor, Dept. of ISE
BNMIT

Dr. Shashikala
Prof & Head, Dept. of ISE
BNMIT

Name & Signature of the Examiners with date:

- 1.
- 2.

Table of Contents

Chapter No.	Title	Page No.
1	Introduction	1
1.1	Objective	1
1.2	Scope of the project	1
1.3	Motivation	1
2	Methodology	2
2.1	Hashing with progressive overflow	2
2.2	Technology, Tools, Language used for project	3
3	System requirements specification	5
3.1	User requirements	5
3.2	Software requirements	5
3.3	Hardware requirements	5
3.4	Functional requirements	5
3.5	Non-functional requirements	6
4	System design and development	7
4.1	Architectural design	7
5	Implementation	10
5.1	List of Modules	10
5.2	Module Description	10
5.3	Algorithm /pseudo code	11
6	Results and discussion	18
6.1	Snapshots of the project and descriptions	18
6.2	Observation about the project	22
6.3	Advantages	22
6.4	Disadvantages	22
7	Conclusion	23
	References	24

List of Figures

Chapter No.	Figure No.	Description	Page No.
4	Fig 4.1	Hashing with progressive overflow	8
	Fig 4.2	Data flow diagram notations	9
6	Fig 6.1	Hashmap.txt	16
	Fig 6.2	Packagedetails.txt	16
	Fig 6.3	Receipt.txt	17
	Fig 6.4	Home Window	18
	Fig 6.5	Registration Window	18
	Fig 6.6	Admin Window	19
	Fig 6.7	User Window	19

CHAPTER 1

INTRODUCTION

File structures is an organization of data in Secondary Storage Device in such a way that it minimizes the access time and the storage space. It is a combination of representation for data in files and of operations for accessing the data. Early in the computing history, secondary storage was in the form of magnetic tape and punched cards. Storage was cheap but access was limited to sequential. In 1956, IBM introduced the RAMAC magnetic disk device.

Data could be accessed directly instead of sequentially. This was the dawn of the study of file structures. Advances in OS gave rise to more research on operating systems. The next analogy that was come up was the Direct Access which is the analogy to access to position in array. Indexes were invented and list of keys and pointers were stored in small files. This allowed direct access to a large primary file. But as the file grows the same problem arise as with the primary memory. Tree structures emerged for main memory in 1960's. This involved balanced and self-adjusting Binary Search. In 1979, a tree structure suitable for files was invented: B trees and B+ trees good for accessing millions of records with three or four disk accesses. Theory on Hashing tables were developed over 60's and 70's good for those files that do not change much over time. Expandable and dynamic Hashing were invented in late 70's and 80's which provided for one or two disk accesses even if file grow dramatically.

Disks are slow. They are also technological marvels: one can pack thousands of megabytes on disk that fits into a notebook computer. Only a few years ago, disks with that kind of capacity looked like small washing machines, However, relative to other parts of a computer disks are slow.

Disks provide enormous capacity at much less cost than memory. They also keep the information stored on it when it is turned off. The tension between a disk's relatively slow access time and its enormous, non-volatile capacity is the driving force behind file structure design.

CHAPTER 2

METHODOLOGY

2.1 Hashing with progressive overflow

Hashing is a technique or process of mapping keys, values into a hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

Hashing is also known as Hashing Algorithm or Message Digest Function. It is a technique to convert a range of key values into a range of indexes of an array. It is used to facilitate the next level searching method when compared with the linear search or binary search. Hashing allows to update and retrieve any data entry in a constant time $O(1)$ means the operation does not depend on the size of data. Hashing is used with database to enable items to be retrieved more quickly. It is used in the encryption and decryption of digital signatures. Hash table or hash map is a data structure used to store key-value pairs. It is a collection of items stored to make it easy to find it later. It uses a hash function to compute an index into an array of buckets or slots from which the desired value can be found. It is an array of list where each list is known as bucket. It contains value based on the key. Hash table is used to implement the map interface and extends Dictionary class. Hash table is synchronized and contains only unique elements.

Progressive Overflow:

When a record is to be stored or retrieved, its home bucket address is determined by hashing. the record overflow problem, but this occurs much less often than when each address can hold only one record. To deal with the records that are not fit into their home address progressive overflow technique is used. Progressive Overflow causes extra searches and thus extra disk accesses. If there are many collisions, then many records will be far from home.

2.2 Tools

NetBeans IDE

NetBeans is a software development platform written in Java. The NetBeans Platform allows application to be developed from a set of modular software components called modules. Applications based on the NetBeans Platform, including the NetBeans integrated development environment (IDE), can be extended by third party developers.

The NetBeans IDE is primarily intended for development in Java, but also supports other languages, in particular PHP, C/C++ and HTML, NetBeans is cross-platform and runs on Microsoft Windows, macOS, Linux, Solaris and other platforms supporting a compatible JVM

A new version was released 8.2 on October 3, 2016. NetBeans IDE is the official IDE for Java 8. With its editors, code analyzers, and converters, you can quickly and smoothly upgrade your applications to use new Java 8 language constructs, such as lambdas, functional operations, and method references. NetBeans IDE is an open-source integrated development environment.

Java Development Kit (JDK)

The Java Development Kit (JDK) is an implementation of either one of the Java Platform, Standard Edition, Java Platform, Enterprise Edition, or Java Platform, Micro Edition platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, mac OS or Windows. The JDK includes a private JVM and a few other resources to finish the development of a Java Application. Since the introduction of the Java platform, it has been by far the most widely used Software Development Kit (SDK). On 17 November 2006, Sun announced that they would release it under the GNU General Public License (GPL), thus making it free software. This happened in large part on 8 May 2007, when Sun contributed the source code to the Open JDK.

Language: Java

Object Oriented - In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

Platform Independent - Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

Simple - Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

Secure - With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

Architecture-neutral - Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

Portable - Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

Robust - Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Multithreaded - With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

Interpreted -Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

High Performance - With the use of Just-In-Time compiler, Java enables high performance.

Distributed - Java is designed for the distributed environment of the internet.

CHAPTER 3

SYSTEM REQUIREMENTS SPECIFICATION

3.1 User Requirements

A file system should be developed to store the names and details of all packages. The details must include the Package Id, Package Name, Duration and cost of the package. The record of all packages is displayed to the customer to choose. An admin must be able to register a new package and the necessary details. Admin must be able to view all the records in the file. A search option must be provided to search for any package details using the Id. If the record is present, it must be displayed. If it is not present, appropriate message should be displayed. Option to traverse a Product record must also be provided. The record is present, it must be traversed. If it is not, an appropriate message should be displayed.

3.2 Software Requirements

- Operating System type: Windows 10 pro-64-bit version
- NetBeans v12.0
- JDK version 14.0.2 Interfaces

3.3 Hardware Requirements

- Processor: Intel core i5-3470 clocked @3.20Ghz
- Ram:8gb ddr4
- Hard disk: 240Gb SSD

3.4 Functional Requirements

- Whenever the admin adds an item, a new record should be inserted into the file containing the required details of the product.
- New record should be inserted and the details are entered.
- The admin should be able to view the details of the new Package record inserted.
- In the display, the Product details like the Package Id, Package name, Availability and price are displayed.

- The search operation should help the admin to find the Package detailshe desires

3.5 Non - Functional Requirements

- The application should display appropriate messages to the admin ifsomething goes wrong.
- The application should not crash.
- The admin should be able to understand the facilities of the application.

CHAPTER4

SYSTEM DESIGN AND DEVELOPMENT

4.1 Architectural design

The reason to avoid overflow is, of course, that extra searches (hence, extra disk accesses) have to occur when a record is not found in its home address. If there are a lot of collisions, there are going to be a lot of overflow records taking up spaces where they ought not to be. Clusters of records can form, resulting in the placement of records a long way from home, so many disk accesses are required to retrieve those records. Consider the following set of keys and the corresponding addresses produced by some hash function.

Key	Home Address
Adams	20
Bates	21
Cole	21
Dean	22
Evans	20

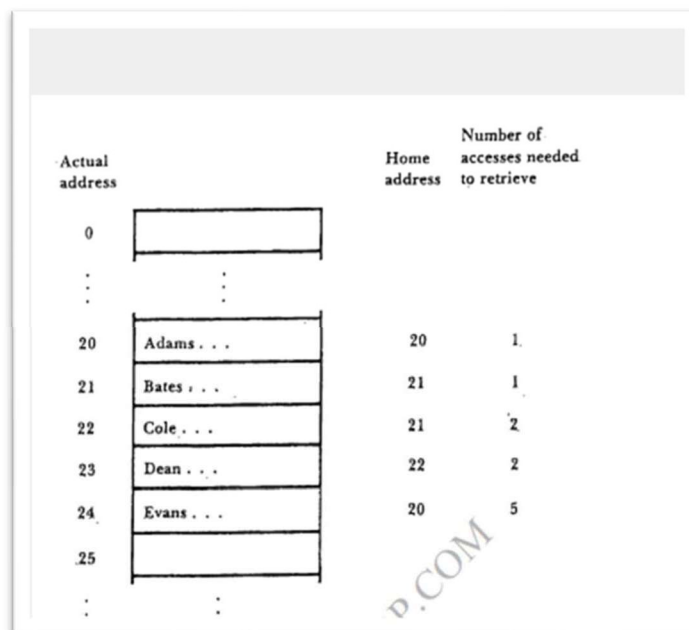


Figure 4.1 Illustration of the effects of clustering of records.

Refer Fig. 4.1. If these records are loaded into an empty file and progressive overflow is used to resolve collisions, only two of the records will be at their home addresses. All the others require extra accesses to retrieve. Fig 4.1 shows where each key is stored, together with information on how many - accesses are required to retrieve it. The term search length refers to the number of accesses required to retrieve a record from secondary memory. In the context of hashing, the search length for a record increases every time there is a collision. If a record is a long way from its home address, the search length may be run as search length.

The average search length is the average number of times you can expect to have to access the disk to retrieve a record. A rough estimate of average search length may be computed by finding the total search length (the sum of the search lengths of the individual records) and dividing this by the Average number search of records:

$$\text{length} = \text{total search length} / \text{total number of records}$$

CHAPTER5

IMPLEMENTATION

5.1 List of modules

Progressive overflow A collision resolution technique which places overflow records at the first empty address after the home address. With progressive overflow, a sequential search is performed beginning at the home address. The search is continued until the desired key or a blank record is found. The different types of operations supported by Hashing with chained progressive overflow are: - add, search, delete and update.

The different operations supported by Hashing with progressive overflow are listed below:

- 1) Add
- 2) Search
- 3) View
- 4) Delete

5.2 Description of module

Add module is used to add details of the Package to the file created by the admin. This module adds the information in an organized manner that is pre-defined and helps in proper management of all the information.

Search module is used to search for a specific package detail that the admin wishes to review. This module searches the file with the help of the package id entered by the admin and display's the information of the corresponding record.

View module is used to review all the information that is stored in a file at once. This module can be used to see all package details that is stored in a file.

Delete module is used to delete a particular record that is stored in the file by using theid of that Package.

5.3 Source code:

The source code used for implementation of all the file operations

```
package pm_sys;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

public class admin extends javax.swing.JFrame {
/**
 * Creates new form admin
 */
private HashNode[] buckets;
private int numOfBuckets=10;
//private int count=0;
private int size;
private int [] sz = new int[10];
public admin() {
    initComponents();
    this.buckets = new HashNode[numOfBuckets];
    this.size = 0;
    for(int t=0; t<10;t++)sz[t]=0;
    try {
        // TODO add your handling code here:
        BufferedReader reader = new BufferedReader(new FileReader("med_details.txt"));

        Object [] lines = reader.lines().toArray();
        for(int i=0;i<lines.length;i++)
            {System.out.println(lines[i]);
```

```
String[] row = lines[i].toString().split(" ");
int k=Integer.parseInt(row[0]);
String v=row[1];
int q=Integer.parseInt(row[2]);
int pi=Integer.parseInt(row[3]);
put(k,v,q,pi);
}
//for(int t=0; t<10;t++)System.out.println(sz[t]);
} catch (FileNotFoundException ex) {
    Logger.getLogger(user.class.getName()).log(Level.SEVERE, null, ex);
}
}
private class HashNode {
    private Integer key; // Can be generic type
    private String value; // Can be generic type
    private HashNode next; // reference to next HashNode
    private Integer Qy;
    private Integer Pz;
    public HashNode(Integer key, String value,Integer Qy,Integer Pz){
        this.key = key;
        this.value = value;
        this.Qy = Qy;
        this.Pz = Pz;
    }
}
public boolean isEmpty(){
    return size == 0;
}
public int put(Integer key, String value,Integer Qy,Integer Pz){
    if(key == null || value == null){
        throw new IllegalArgumentException("Key or Value is null !!!");
    }
    int bucketIndex = getBucketIndex(key);
    int w=bucketIndex;
```

```
do
{
    HashNode head = buckets[w];
    while(head != null){
        if(head.key.equals(key)){
            head.value = value;
            head.Qy=Qy;
            head.Pz=Pz;
            return 1;
        }
        head = head.next;
    }
    if(size == 10) {JOptionPane.showMessageDialog(rootPane,"Storage is FULL !");
    return 0;}
    if(sz[w]<10)
    {
        head = buckets[w];
        HashNode node = new HashNode(key, value,Qy,Pz);
        node.next = head;
        buckets[w] = node;
        ++sz[w];
        ++size;
        //System.out.println(size);
        return 1;
    }
    w= (w+1) % 10;
    }while(w!=bucketIndex);
    //for(int t=0; t<10;t++)System.out.println(sz[t]);
    return 2;
}

private int getBucketIndex(Integer key){
    return key % numOfBuckets; // buckets.length
}

public String get(Integer key){
    if(key == null){
```



```
        throw new IllegalArgumentException("Key is null !!!");
    }

    int bucketIndex = getBucketIndex(key);
    int r=bucketIndex;
    do
    {
        HashNode head = buckets[r];
        while(head != null)
        {
            if(head.key.equals(key)){
                mname.setText(head.value);
                mqty.setText(head.Qy.toString());
                mpz.setText(head.Pz.toString());
            }
            head = head.next;
        }
        r= (r+1) % 10;
    }while(r!=bucketIndex);
    return null;
}

public String remove(Integer key, String value,Integer Qy,Integer Pz){
    if(key == null){
        throw new IllegalArgumentException("Key is null !!!");
    }
    int bucketIndex = getBucketIndex(key);
    int y=bucketIndex;
    do
    {
        HashNode head = buckets[y];
        HashNode previous = null;
        while(head != null){
            if(head.key.equals(key)){
                break; }
            previous = head;
            head = head.next; }
    }
```

```
if(head == null){
    y=(y+1)%10;
}
else
{
    if(head.key.equals(key))
    {--size;
    --sz[y];
    if(previous != null )
    {
        previous.next = head.next;
    }
    else
    {

        buckets[y] = head.next;
    }
    }
    else
    {
        y=(y+1)%10;
    }
    }
}while(y!=bucketIndex);
return null;
}

public void printAll()
{
    try {
        BufferedWriter writer = new BufferedWriter(new FileWriter("hashmap.txt"));
        BufferedWriter writer1 = new BufferedWriter(new
        FileWriter("package_details.txt"));for(int i=0;i<10;i++)
        {
            HashNode head = buckets[i];
            writer.write("{}+i+");
```

```
while(head != null)
{
writer.write("\n"+"ID:"+head.key+"|"+"Name:"+head.value+"|"+"Availability:"+head.
Qy+"|"+"Price:"+head.Pz);
writer1.write(head.key+" "+head.value+" "+head.Qy+" "+head.Pz);
writer1.write("\n");
head = head.next;
}
writer.write("\n");
}
writer.close();
writer1.close();
} catch (IOException ex) {
    Logger.getLogger(admin.class.getName()).log(Level.SEVERE, null, ex);
}
}
```

Front End

JFrame form is used to create the front end of the mini project. A frame, implemented as an instance of the JFrame class, is a window that has decorations such as a border, a title, and supports button components that close or iconify the window. Applications with a GUI usually include at least one frame. Applets sometimes use frames, as well in the mini project. The page where admin can choose the operation that he/she wants to perform, corresponding to the option chosen the admin has to enter the details that is asked in that form to complete the operation. The admin also has the option to go back or logout of the system whenever the admin wishes to do so. If the admin chooses the add option, then the admin will be transferred to a same page where the admin has to enter all the details of the package. If he chooses search option, then he can search for a particular package detail by entering the package id.

Back End

Java class is used to write and execute the program in the back end. It is pretty easy to execute and call the program in java, than in any other language. The different operations used in the program are also written in java class which helps the program for internal operation of the function. Every function has a call to other function internally and java class helps in the proper execution of these functions.

CHAPTER6

RESULTS AND DISCUSSIONS

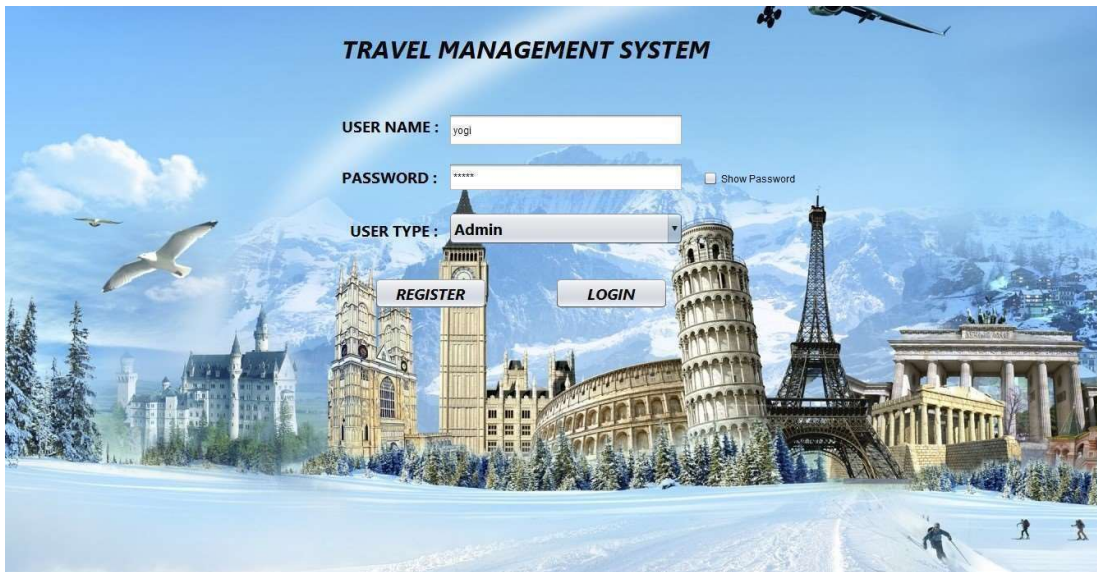


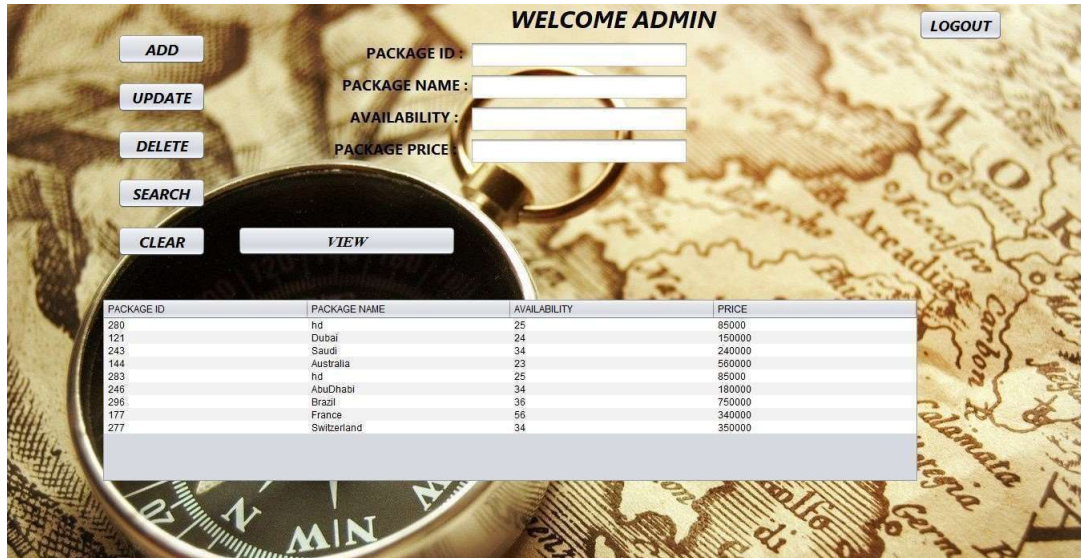
Fig 6.1: Home Window



Fig 6.2: Registration Window

Fig 6.1 is the home Window of the Travel Management System. The user as well as admin can login as well as register. The related page will be displayed upon input by user or admin.

Fig 6.2 shows the Registration window where the admin and user can register. The fields taken are username, password, email and phone number.

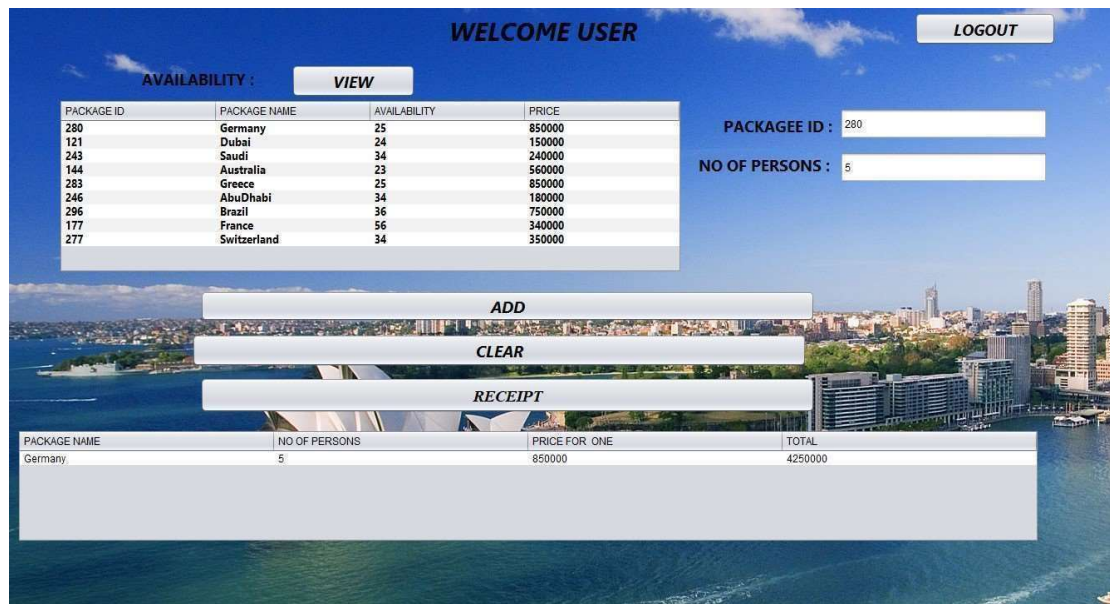


WELCOME ADMIN LOGOUT

PACKAGE ID :
 PACKAGE NAME :
 AVAILABILITY :
 PACKAGE PRICE :

PACKAGE ID	PACKAGE NAME	AVAILABILITY	PRICE
280	hd	25	85000
121	Dubai	24	150000
243	Saudi	34	240000
144	Australia	23	560000
283	hd	25	85000
246	AbuDhabi	34	180000
296	Brazil	36	750000
177	France	56	340000
277	Switzerland	34	350000

Fig 6.3: Admin Window



WELCOME USER LOGOUT

AVAILABILITY :

PACKAGE ID	PACKAGE NAME	AVAILABILITY	PRICE
280	Germany	25	850000
121	Dubai	24	150000
243	Saudi	34	240000
144	Australia	23	560000
283	Greece	25	850000
246	AbuDhabi	34	180000
296	Brazil	36	750000
177	France	56	340000
277	Switzerland	34	350000

PACKAGE ID :
 NO OF PERSONS :

PACKAGE NAME	NO OF PERSONS	PRICE FOR ONE	TOTAL
Germany	5	850000	4250000

Fig 6.4: User Window

Fig 6.3 shows the admin window where the admin can add, delete, update, search, view travel package details.

Fig 6.4 shows the User window where the user can view, select and book the package, then he can print the receipt for selected package.

6.2 Observation about the Project

The project is observed to correctly perform the operations of requirements like search, add, view, delete, update. The details when entered by the admin for addition is appended to the data file.

The admin can view whole file details by clicking view button. The whole file is displayed in the form of a table with field headers. The admin can delete the details by clicking on the delete.

It is observed that Hashing with progressive overflow has many advantages like it is simple to implement. It can efficiently resolve collision by looking for the next available address in the hash table.

On the other hand, the user can view the package details along with the available seats and receipt is generated based on the no. of seats selected by the user.

6.3 Advantages

- Easy to implement.
- Resolves collision by searching for next available space in the hash file.

6.4 Disadvantages

- Searching for the next available space is time consuming if there are a greater number of records in the hash file
- Insertion of record can not be done if the hash file is full

CHAPTER 7

CONCLUSION

In this mini project, a simple interface is created to make the add, search, view and deletion of the travel packages easy. The admin could enter all the details of the travel package such as the Package ID, Package name, Availability, price of the package. The admin can also search for a package by entering the ID. The admin can also delete a particular package by entering the ID. The second type is user, who is successfully able to view the updated available seats and select the package accordingly.

The above operations are efficiently implemented using hashing with progressive overflow.

REFERENCES

Textbooks:

1. Michael J. Folk, Bill Zoellick, Greg Riccardi: File Structures-An Object Oriented Approach with C++, 3rd Edition, Pearson Education, 1998.
2. Jim Keogh: J2EE-TheComplete Reference, McGraw Hill, 2007.

Websites:

1. <https://www.geeksforgeeks.org/introduction-of-hashing/>
2. <https://netbeans.org/features/java/swing.html>
3. <https://netbeans.org/kb/docs/java/quickstart-gui.html>