

AIRLINE RESERVATION SYSTEM

A PROJECT REPORT

Submitted by

V.SURYA INDIRA

103678137

In partial fulfillment for the award of the degree of

B.Tech

In

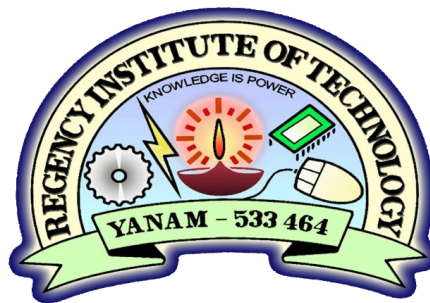
Mini Project Lab

COMPUTER SCIENCE

Under the esteemed Guidance of

Prof.Ch. Raja Ramesh

M.Tech, [PhD]



**REGENCY INSTITUTE OF TECHNOLOGY
ADAVIPOLAM, YANAM-533464**

**Department of Computer Science and Engineering
March-2012**

DECLARATION BY THE CANDIDATE

I here by declare that the project report entitled “**AIRLINE RESERVATION SYSTEM**” submitted by me to Regency Institute of Technology; Yanam in partial fulfillment of the requirement for the award of the degree of **B.TECH** in COMPUTER SCIENCE DEPARTMENT is a record of bonfide project work carried out by me under the guidance of Mr. CH. RAJA RAMESH. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Yanam

Signature of the Candidate

Date:

BONAFIDE CERTIFICATE

This is to certify that the project report entitled “**AIRLINE RESERVATION SYSTEM**” submitted by **T.S.S.N.SAILAJA (103678123), V.KRISHNA CHAITANYA (103678128), V.SURYA INDIRA (103678137), Y.SRI RAMYA (103678143)** to Regency Institute of Technology, Yanam in partial fulfillment of the requirement for the award of the degree of **B.TECH in COMPUTER SCIENCE DEPARTMENT** is a record of bonafide work carried out by him/her under my guidance. The project fulfills the requirements as per the regulations of this Institute and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

<Signature of the Coordinator>

<Signature of the Supervisor>

<Name>

COORDINATOR

Date:

Date:

<Signature>

External Examiner (s)

<Signature>

Internal Examiner (s)

ACKNOWLEDGEMENT

“Task successful” makes everyone happy. But the happiness will be gold without glitter if we didn’t state the persons who have supported us to make it a success. Success will be crowned to people who made it a reality but the people whose constant guidance and encouragement made it possible will be crowned first on the eve of success.

This acknowledgment transcends the reality of formality when we would like to express deep gratitude and respect to all those people behind the screen who guided, inspired and helped me for the completion of our project work. We consider ourselves lucky enough to get such a good project. This project would add as an asset to my academic profile.

We express our sincere gratitude to our respectful Principal **Dr.P.Kumar Babu** and dean **Dr. A. Ramakrishna Rao** for enabling us to make use of laboratory and library facilities liberally, that helped us a long way in carrying out our project work successfully.

We express our gratitude to the help of the Head of the Department of Computer Science and Engineering, **Mr. S. T. V. S. Kumar, M.Tech, (Ph. D)**, for his constant supervision, guidance and co-operation throughout the project and we would like to express our thankfulness to our project guide, **Mr.Ch. Raja Ramesh, M.Tech,(Ph.D)** for his constant motivation and valuable help through the project work.

We extend our sincere gratitude to our parents who have encouraged us with their blessings to do this project successfully. Finally we would like to thank to all our friends, all the teaching and non-teaching staff members of the CSE Department, for all the timely help, ideas and encouragement which helped throughout in the completion of project.

INDEX

Abstract	7
List of Tables	8
List of Figures	8
1. Introduction	9
2. Overview of Proposed system	10
2.1 Existing System	10
2.2 Proposed System	11
3. System Analysis	12
3.1 Front End	12
3.2 Back End	30
3.3 Modules	34
4. System Design	36
4.1 UML	37
4.1.1 Class Diagram	37
4.1.2 Activity Diagram	37
4.1.3 Sequence Diagram	37
4.1.4 Use Case Diagram	38
4.1.5 Collaboration Diagram	38
4.1.6 Dataflow	39
4.2 Data Dictionary	43
4.2.1 Introduction	43
4.2.2 Data Dictionary	43
5. Implementation	49
6. Experimental Results	60
7. Testing	67
7.1 Introduction	67
7.2 Type of Testing	67
7.2.1 Unit Testing	68
7.2.2 Integrated Testing	

7.2.3 Recovery Testing	68
7.2.4 Security Testing	68
7.2.5 Performance Testing	69
7.2.6 White-box Testing	69
8. Conclusion	70
9. References	71

ABSTRACT:

Airline reservation systems were first introduced in the late 1950s as relatively simple standalone systems to control flight inventory, maintain flight schedules, seat assignments and aircraft loading. The modern airline reservation system is comprehensive suite of products to provide a system that assists with a variety of airline management tasks and service customer needs from the time of initial reservation through completion of the flight.

One of the most common modes of travel is traveling by air. Customers who wish to travel by air nowadays have a wide variety of airlines and a range of timings to choose from. Nowadays competition is so fierce between airlines that there are lot of discounts and a lot of luxuries given to customers that will give an edge to that particular airline.

The World Wide Web has become tremendously popular over the last four years, and currently most of the airlines have made provision for online reservation of their flights. The Internet has become a major resource for people looking for making reservations online without the hassle of meeting travel agents. My Project intends to serve these purposes. It intends to check all the available airline databases and return a string of results, which can help them in their travel plans.

The objective of this project is to create an airline reservation system where a traveler can request all flight information as per their journey dates. They can get information regarding time, cost, etc all at the same time and place. When the customer calls the Counter Assistant for his/her travel needs, the counter assistant will enter the customer's details (flight requirements) in the system. The system displays all the available airlines, schedules and prices. This system would help the airline to better serve its customers by catering to their needs. The site would use a Database to hold this information as well as the latest pricing and availability information for the airlines.

LIST OF TABLES:

Cancellation table

Classes table

Flight days

Flight Details

Login

Mail

New User

Passenger

Payment

Reservation

LIST OF FIGURES:

Class Diagram

Activity Diagram

Sequence Diagram

Use Case Diagram

Collaboration Diagram

1. INTRODUCTION:

Airline reservation systems were first introduced in the late 1950s as relatively simple standalone systems to control flight inventory, maintain flight schedules, seat assignments and aircraft loading. The modern airline reservation system is comprehensive suite of products to provide a system that assists with a variety of airline management tasks and service customer needs from the time of initial reservation through completion of the flight.

One of the most common modes of travel is traveling by air. Customers who wish to travel by air nowadays have a wide variety of airlines and a range of timings to choose from. Nowadays competition is so fierce between airlines that there are lot of discounts and a lot of luxuries given to customers that will give an edge to that particular airline.

The World Wide Web has become tremendously popular over the last four years, and currently most of the airlines have made provision for online reservation of their flights. The Internet has become a major resource for people looking for making reservations online without the hassle of meeting travel agents. My Project intends to serve these purposes. It intends to check all the available airline databases and return a string of results, which can help them in their travel plans.

The objective of this project is to create an airline reservation system where a traveler can request all flight information as per their journey dates. They can get information regarding time, cost, etc all at the same time and place. When the customer calls the Counter Assistant for his/her travel needs, the counter assistant will enter the customer's details (flight requirements) in the system. The system displays all the available airlines, schedules and prices. This system would help the airline to better serve its customers by catering to their needs. The site would use a Database to hold this information as well as the latest pricing and availability information for the airlines.

2. OVERVIEW OF THE PROJECT:

The main purpose of this software is to reduce the manual errors involved in the airline reservation process and make it convenient for the customers to book the flights as when they require such that they can utilize this software to make reservation, modify reservations or cancel a particular reservation.

The name of the software is “AIRLINE RESERVATION SYSTEM”. This software provides options for viewing different flights available with different timings for a particular date and provides customers with the facility to book a ticket, modify or cancel a particular reservation but it does not provide the customers with details of cost of the ticket and it does not allow the customer to modify a particular part of his reservation and he/she can modify all details.

2.1 EXISTING SYSTEM:

The effectiveness of the system depends on the way in which the data is organized .In the existing system, much of the data is entered manually and it can be very time consuming. When records are accessed frequently, managing such records becomes difficult. Therefore organizing data becomes difficult. The major limitations are:

- Modifications are complicated
- Much time consuming
- Error prone
- Unauthorized access of data

2.2 PROPOSED SYSTEM:

The proposed system is better and more efficient than existing System by keeping in mind all the drawbacks of the present system to provide a permanent to them.

The primary aim of the new system is to speed up the transactions. User friendliness is another peculiarity of the proposed system. Messages are displayed in message boxes to make the system user friendly. The main Advantage of the proposed system is the reduction in labor as it will be possible so search the details of various

places. Every record is checked for completeness and accuracy and then it is entered into the database. The comments and valid messages are provided to get away redundant data. Another important feature of the proposed system is the data security provided by the system. The main objectives of the proposed system are:

- Complex functions are done automatically
- Processing time can be minimized
- Simple and easy to manage
- Chances of errors reduced
- Faster and more accurate than the existing system
- Easy for handling reports

The proposed system is complete software for Airline Reservation System, Which is more efficient, reliable, faster and accurate for processing.

3. SYSTEM ANALYSIS:

3.1 FRONT END (JAVA)

Overview of Java:

Java is a powerful but lean object oriented programming language. It has generated a lot of excitement because it makes it possible to program for Internet by creating applets, programs that can be embedded in web page. The context of an applet is limited only by ones imagination. Applets can be just little decorations to liven up web page, or they can be serious applications like word processors or spreadsheet.

But Java is more than programming language for writing applets. It is becoming so popular that many people believe it will become standard language for both general purposes and Internet programming.

Java from C++:

Java builds on the strength of C++. It has taken the best features of C++ and discarded the more problematic and error prone parts. To this, it has added garbage collection (automatic memory management), multi threading (the capacity for one program to do more than one thing at a time) and security capabilities. The result is that Java is simple, elegant, powerful and easy to use.

Java is actually a platform consisting of three components:

- Java Programming Language.
- Java library of classes and interfaces.
- Java virtual Machine.

But Java is more than programming language for writing applets. It is becoming so popular that many people believe it will become standard language for both general purpose and Internet programming.

Components of Java:

Java is actually a platform consisting of three components:

- Java Programming Language.
- Java Library of classes and interfaces.
- Java Virtual Machine.

Java is Object Oriented:

The Java programming language is object oriented, which makes program design focus on what you are dealing with rather than on how you are going to do something. This makes it more useful for programming in sophisticated projects because one can break the things down into understandable components. Reusability of these components is another big benefit.

Object oriented languages use the paradigm of classes. In simple terms, a class includes both the data and the functions to operate on that data. Object is an instance of the class forms the actual run time entity of the class. Encapsulation of code and data makes it possible to make the changes in code without breaking other programs that use that code.

Java includes inheritance, or the ability to derive new classes from existing class referred to as the parent class. A subclass can add new data members to those inherited from the parent class. As far as methods are concerned, the subclass can reuse the inherited methods as it is, change them, and its own new methods.

Java's exciting features are:

- Ease in code correction.
- Garbage collection.
- Absence of pointers.
- Java is extensible.

- Java is secure.
- Java is robust.
- Java is multithreading.
- Simplicity.

Library Classes:

The Java platform includes an extensive class library so that programmers can use already existing classes, as it is, create subclasses to modify existing classes or implement interfaces and augment the capabilities of classes.

Classes contain data members i.e. fields and functions i.e. methods. In classes fields may be either variable or constant, and methods are fully implemented.

Interfaces:

Interfaces is also merely like class. Interfaces also contain data members and functions. But the main difference is that in an interface, fields must be constants, and methods are just prototypes with no implementations. The prototype give the method signature (the return type, the function name and the number of parameters with the type for each parameter), but the programmer must supply implementation. To use an interface, a programmer defines a class, declares that implements the interfaces, and then implements all the methods in that interface as the class.

The methods are implemented in a way that is appropriate for the class in which the methods are being used. Interface let one add functionality to a class and give a great deal of flexibility in doing it. In other words interfaces provide most of the advantages of multiple inheritances without its disadvantages.

Packages:

A package is a collection of related java classes and interfaces. The following list, gives examples of some java packages and what they cover.

JAVA.IO- Classes those manage reading data from input streams and writing data to the output streams.

JAVA.AWT- Classes that manage user interface components such as windows, dialog boxes, buttons, check boxes, lists, menus, scrollbars, and text fields; the 'AWT' stands Abstract Window Toolkit.

JAVA.APPLET- The applet class, which provides the ability to write applets, this package also includes several interfaces that connect an applet to its document and to resources for playing audio.

JAVA.AWT.EVENT- GUIs are event driven; it means they generate events when the user of the program interacts with the GUI.

JAVAX.SWING- This package enables the user to create interfaces which performs the GUI operations.

JAVA.SQL- The JDBC API, classes and interfaces that access database and send SQL. In Java, packages serve as basis for building other package.

The Java Platform Builds in Security in Four Ways:

The way memory is allocated and laid out: In java an objects location in memory is not determined until the runtime, as opposed to C and C++, where the compiler makes memory layout decisions. As the result, a programmer cannot look at a class definition and figures out how it might be laid out in memory. Also since, java has no pointers, a programmer cannot forge pointer to memory.

The way incoming code is checked. The java virtual machine doesn't trust any incoming code and subjects it to what is called Byte Code verification. The byte code verifier, part of the virtual machine, checks that

- The format of incoming code is correct
- Incoming code doesn't forge pointers
- It doesn't violate access restrictions
- It access objects as what they are

The way classes are loaded. The java byte code loader, another Part of the virtual

machine, checks whether classes loaded during program execution are local or from across a network. Imported classes cannot be substituted for built in classes cannot accidentally reference classes brought in over a network.

The way access is restricted for entrusted code. The java security manager allows user to restrict entrusted java applets so that they cannot access the local network, local files and other resources.

What is JDBC?

- JDBC is a java TM API for executing SQL statements.
- It consists of a set of classes and interfaces written in the java programming language that makes it easy to send SQL statements to virtually any relational databases.
- JDBC (Java Database Connectivity) is a front end tool for connecting server to ODBC in that respect.
- JDBC is essentially a low-level application programming interface. It is called as low-level API since any data manipulation, storage and retrieval has to be done by the program itself. Some tools which provide a higher-level abstraction of, expected shortly.

The combination of java and JDBC lets a programmer write it once and run it anywhere.

Requirements to use JDBC:

- To use JDBC we need a basic knowledge of database and SQL.
- We need the jdk1.1 (Java Development Kit 1.1 available Java Soft's website) or a version of java since jdk1.1 and above come bundled with JDBC software.
- A back-end database engine for which a JDBC driver is available. When JDBC drivers area not available JDBC-ODBC bridge drivers are used to access the database through ODBC.
- Back-end is not needed when JDBC driver is capable of storing and retrieving the

data itself, or if JDBC-ODBC bridge and the ODBC driver can be used to store and retrieve the information.

What does JDBC do?

JDBC makes it possible to do three things.

- Establishes the connection to database
- Send SQL statements
- Process the results.

JDBC is a low-level API and a base for Higher-level API. JDBC is a low-level interface, which means that it is used to invoke SQL commands directly. It works very well in this capacity and is easier to use than other to build higher-level interfaces and tools. A higher level interface such as JDBC. There are two kinds of higher-level APIs.

- An embedded SQL for java and
- A direct mapping of relational database tables to java classes.

Java's Magig: The Byte Code

The key that allows java to solve both the security and the portability problems just described is that the output of a java compiler is not executable code. Rather, it is Byte code. Byte code is highly optimized set of instructions designed to be executed by the java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an interpreter for Byte code. This may come as a bit of a surprise.

Translating a java program into Byte code helps makes easier to run a program in a wide variety of environments. The reason is straight forward; only the JVM needs to be implemented for each platform. Once the run-time package exists for a given system, any java program can run on it. Remember, although the details of the JVM will differ from

platform to platform, all interrupt the same Java Byte Code.

JDBC Drivers:

The JDBC API found in `java.sql` package, consists only a few concrete classes. Much of the API is distributed as database-neutral interface classes that specify behavior without providing any implementation. The actual implementations are provided by third-party vendors.

An individual database system is accessed via a specific JDBC driver that implements the `java.sql.Driver` interface. Drivers exist for nearly all popular RDBMS systems, though few are available for free. Sun bundles a free JDBC-ODBC bridge driver with the JDK to allow access to standard ODBC data sources, such as a Microsoft Access Database.

However, Sun advises against using the bridge driver for anything other than development and very limited deployment. Servlet developers in particular should need this warning because any problem in the JDBC-ODBC bridge driver's native code section can crash the entire server, not just your servlets.

JDBC drivers are available for most database platforms, from a number of vendors and in a number of different flavors. There are four driver categories:

Type1-JDBC-ODBC Bridge Driver

Type1 drivers use a bridge technology to connect a java client to an ODBC database service. Sun's JDBC-ODBC bridge is the most common Type1 driver. These drivers are implemented using native code.

Type2- Native-API Partly-Java Driver

Type2 drivers wrap this layer of java around database-specific native code libraries. For Oracle databases, the native libraries might be based on OCI (Oracle Call

Interface) libraries, which were originally designed for C/C++ programmers. Because Type2 drivers are implementing using native code, in some cases they have better performance than their all-Java counterparts. They add an element of risk; however, because a defect in a driver's native code section can crash the entire server.

Type3- Net-Protocol All-Java Driver

Type3 drivers communicate via a generic network protocol to a piece of custom middleware. The middleware components might use any type of driver to provide the actual database access. WebLogic's Tengah product line is an example. These drivers are all java, which makes them useful for applet deployment and safe for servlet deployment.

Type4- Native-Protocol All-Java Driver

Type4 drivers are the most direct of the lot. Written entirely in java, Type4 drivers understand database-specific networking protocols and can access the database directly without any additional software.

A list of currently available JDBC drivers can be found at

Getting a Connection

The first step in using a JDBC driver to get a database connection involves loading the specific driver class into the application's JVM. This makes the driver available later, when we need it for opening the connection. An easy way to load the driver class is to use the `Class.forName ()` method:

`Class.forName ("sun.jdbc.odbc_JdbcOdbcDriver"):`

When the driver is loaded to memory, it registers itself with the `java.sql.DriverManager` class to open a connection to a given database, where the

database is specified by a specially formatted URL. The method used to open the connection is `DriverManager.getConnection()`. It returns a class that implements the `java.sql.Connection` interface:

A JDBC URL identifies an individual database in a driver-specific manner. Different drivers may need different information in the URL to specify the host database. JDBC URLs usually begin with `jdbc:sub_protocol:subname` during the call to `getConnection()`, the Driver Manager object asks each registered driver if it recognizes the URL. If a driver says yes, the driver manager uses that driver to create the connection object. Here is a snippet of code a servlet might use to load its database driver with the JDBC-ODBC Bridge and create an initial connection:

SERVLETS:

What are Java servlets?

Servlets are Java technology's answer to CGI programming. They are programs that run on a Web server and build Web pages. Building Web pages on the fly is useful (and commonly done) for a number of reasons:

The Web page is based on data submitted by the user.

For example the results pages from search engines are generated this way, and programs that process orders for E-commerce do this as well.

The data changes frequently.

For example, a weather-report or news headlines page might build the page dynamically, perhaps returning a previously built page if it is still up to date.

The Web pages uses information from corporate databases

or other such sources.

For example, you would use this for making a web page at On-line stores that lists current prices and number of items in stock.

What are the Advantages of Servlets Over “Traditional” CGI?

Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional CGI and than many alternative CGI-like technologies.

- **Efficient:**

With traditional CGI, a new process is started for each HTTP request. If the CGI process does a relatively fast operation, the overhead of starting the process can dominate the execution time.

With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java Thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are N simultaneous request to the same CGI program, then the code for the CGI program is loaded into memory n times. With servlets, however, there are N threads but only a single copy of the servlet class. Servlets also have more alternatives than do regular CGI programs for optimizations such as caching previous computations, keeping database connections open, and the like.

- **Convenient:**

Hey, you are already known Java. Why learn Perl too? Besides the convenience of being able to use a familiar language, servlets have an extension infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities.

- **Powerful:**

Java servlets let you easily do several things that are difficult or impossible with regular CGI. For one thing, servlets can talk directly to the Web server (regular CGI programs can't). This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations.

- **Portable:**

Servlets are written in java and follows a well-standardized API. Consequently, servlets return for, I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or Web Star. Servlets are supported directly or via plug-in on almost every major Web Server.

- **Inexpensive:**

There are a number of free or very inexpensive Web servers available that are good for “personal” use or low-level Web sites. However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive. Nevertheless, once you have a Web server, no matter the cost server, adding servlet support to it (if it doesn't come preconfigured to support servlets) is generally free or cheap.

What is JSP?

Java Serve Pages (JSP) is a technology that lets you mix regular, static HTML with dynamically-generated HTML. Many Web pages that are built by CGI programs are mostly static, with the dynamic part limited to a few small locations. But most CGI variations, including servlets, make you generate the entire page via your program, even

though most of it is always the same. JSP lets you create the two parts separately. Here's an example:

What are the Advantages of JSP?

- **Vs. Active Server Pages (ASP):**

ASP is a similar technology from Microsoft. The advantages of JSP are twofold. First, the dynamic part is written in Java not in Visual Basic or other MS-specific language. So it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft web servers.

- **Vs. Pure Servlets:**

JSP doesn't give you anything that you couldn't in principle do with a servlet. But it is more convenient to write (and to modify!) regular HTML than to have a zillion `println` statements that generate the HTML. Plus, by separating the look from the content you can put different people on different tasks: your Web page design experts can build the HTML, leaving places for your servlet programmers to insert the dynamic content.

- **Vs. Server-Side Includes (SSI):**

SSI is a widely-supported technology for including externally-defined pieces into a static Web page. JSP is better because it lets you use servlets instead of a separate program to generate that dynamic part. Besides, SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

- **Vs. JavaScript:**

JavaScript can generate HTML dynamically on the client. This is a useful

capability, but only handles situations where the dynamic information is based on the client's environment. With the exception of cookies, HTTP and form submission data is not available to JavaScript. And, since it runs on the client, JavaScript can't access server-side resources like databases, catalogs, pricing information, and the like.

- **Vs. Static HTML:**

Regular, HTML, of course, cannot contain dynamic information. JSP is so easy and convenient that it is quite feasible to argument HTML pages that only benefit marginally by the insertion of small amounts of dynamic data. Previously, the cost of using dynamic data would preclude its use in all but the most valuable instances.

Basic Servlet Structure:

Here's the outline of a basic servlet that handles GET requests. GET requests, for those unfamiliar with HTTP, are requests made by browsers when the types in a URL on the address line, follows a link from a web page, or makes an HTML form that does not specify a METHOD. Servlets can also very easily handle POST requests, which are generated when someone creates an HTML form that specifies METHOD="POST".

```
Import java.io.*;
Import javax.servlet.*;
Import javax.servlet.http.*;

Public class SomeServlets extends HttpServlet {
Public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException.IOException {

//Use "request" to read incoming HTTP headers (e.g. cookies)
//and HTML form data (e.g. data the user entered and submitted)
```



```
//Use “response” to specify the HTTP response line and headers
// (e.g. specifying the content type, setting cookies).

printWriter out=response.getWriter ();
//Use “out” to send content to browser
}
```

To be a servlet, a class should extend `HttpServlet` and override `doGet` or `doPost` (or both), depending on whether the data is being sent by GET or by POST. These methods take two arguments: an `HttpServletRequest` and an `HttpServletResponse`. The `HttpServletRequest` has methods that let you find out about incoming information such as FORM data, HTTP request headers, and the like. The `HttpServletResponse` has methods that lets you specify the HTTP response line (200,404, etc.), response headers (Content-Type, Set-Cookies, etc.), and, most importantly, lets you obtain a `PrintWriter` used to send output back to the client.

For simple servlets, most of the effort is `println` statements that generate the desired page. Note that `doGet` and `doPost` throw two exceptions, so you are required to include them in the declaration. Also note that you have to import classes in `java.io` (for `printWriter`, etc.), `javax.servlet` (for `HttpServlet`, etc.), and `javax.servlet.http` (for `HttpServletRequest` and `HttpServletResponse`). Finally, note that `doGet` and `doPost` are called by the service method, and sometimes you may want to override service directly, e.g. for a servlet that handles both GET and POST request.

Compiling and Installing the Servlet:

Note that the specific details for installing servlets vary from web server to web server. Please refer to your web server documentation for definitive directions. The on-line examples are running on Java Web Server (JWS) 2.0, where servlets are expected to be in a directory called `Servlets` in the JWS installation hierarchy. However, I placed this servlet in a separate package (hall) to avoid conflicts with other servlets on this server. You will want to be the same if you are using a web server that is used by other people

and doesn't have a good infrastructure for "virtual servers" to prevent these conflicts automatically.

If you've never used packages before, there are two ways to compile classes that are in packages.

One way is to set your CLASSPATH to point to the directory above the actually containing your servlets. You can then compile normally from within the directory. For example, if your base directory is C:\JavaWebServer\servlets and your package name (and thus subdirectory name) is hall, and were on windows, you'd do:

```
DOS> set CLASSPATH=C: \JavaWebServlets; %CLASSPATH%
DOS> cd C:\JavaWebServer\servlets\hall
DOS> javac yourServlet.java
```

The first path, setting the CLASSPATH, you probably want to do permanently, rather than each time you start a new DOS window. On Windows 95/98 you'd typically put the "set CLASSPATH=....." Statement in your autoexec.bat file somewhere after the line that set the CLASSPATH to point to servlet.hal and jsp.jar. On Windows NT, you'd go to the Start menu, select settings, select Control panel, select system, select Environment, then enter the variable and value. Note also that if your package were of the form name1.name2.name3 rather than simply name1 as here, you'd still have the CLASSPATH point to the top level directory of your package hierarchy (the one contain name1).

A Second way to compile classes that are in packages is to go to the directory above the one containing your servlets, and then do "javac directory\YourServlet.java" (UNIX; note the forward slash). For example, suppose again that your base directory is C:\JavaWebServer\Servlets and your package name (and thus subdirectory name) is hall, and you were on Windows. In that case, you'd do the following:

```
DOS> cd C:\JavaWeb Server\Servlets
DOS> javac hall\Yourservlet.java
```

Note that, on Windows, most JDK 1.1 versions of javac require a backslash, not a forward slash, after the directory name. This is fixed in JDK 1.1, many servlet authors stick with JDK 1.1 for portability.

Finally, another advanced option is to keep the source code in a location distinct from the .class files, and use javac's "-d" option to install them in the location the web server expects.

Running the Servlet:

With the Java web Server, servlets are placed in the servlets directory within the main JWS installation directory, and are invoked via <http://host/servlet/ServletName>. Note that the directory is servlets, plural, while the URL refers to servlet, singular. Since this example was placed in the hall package, it would be invoked via <http://host/servlet/hall.Helloworld>. Other Web servers may have slightly different conventions on where to install servlets and how to invoke them. Most servers also let you define aliases for servlets. So that a servlet can be invoked via <http://host/any-path/any-file.html>. The process for doing this is completely server specific: check your server's documentation for details.

A Servlet that generates HTML Most servlets generate HTML, not plain text as in the previous example. To do that, you need two additional steps: tell the browser that you are sending back HTML, and modify the println statements to build a legal Web page. The first step is done by setting the Content-Type response header. In general, headers can be set via the setHeader method of HttpServletResponse, but setting the content type is such a common task that there is also a special Setcontenttype method just for this purpose. Note that you need to set response headers before actually returning any of the content via the PrintWriter. Here's an example:

Maintenance Release of the Java Servlet 2.5 Specification:

Download the maintenance release of the Java Servlet Specification, version 2.5. This version of Java Servlet technology.

Is included in the EE5 platform: Web Tier to go with Java EE5:

A Look at Resource Injection Read about the support for annotations by Java Web tier technologies and how they can simplify access to resources, environment data, and life-cycle control.

Form Processing Servlet:

This section shows how to

- Process form data
- Manage persistent data
- Use init parameters

The next Servlet that we are going to write provides a user interface to a mailing list through HTML forms. A user should be able to enter an email address in a text field and press a button to subscribe to the list or another button to unsubscribe.

The Servlet consists of two major parts:

- Data management
- Client interaction.

Data management:

The data management is rather straight-forward for an experienced Java Programmer. We use a `java.lang.Vector` object which contains the email addresses as Strings. Since a Servlet can have data which persists between requests we load the address list only once, when the Servlet is initialized, and save it every time it has been changed by a request. An alternative approach would be keeping the list in memory while the Servlet is active and writing it to disk in the destroy method. This would avoid the

overhead of saving the address list after every change but is less fail-safe. If for some reason the address file can't be written to disk or the server crashes and cannot destroy the Servlet, all changes to the list will be lost even though the users who submitted the requests to change the list received positive responses.

In `init` we first call `super.init(config)` to leave the `ServletConfig` management to the super class (`HttpServlet`), then we get the name of the address file from an `init` parameter (which is set up in the Web Server configuration). If the parameter is not available the Servlet throws a `javax.servlet.UnavailableException` (a subclass of `javax.servlet.ServletException`) which indicates that a Servlet is temporarily (if a duration is specified) or permanently (as in this case) unavailable. Finally, the `init` method de-serializes the address file or creates an empty `Vector` if the address file does not exist yet. All exceptions that occur during the re-serialization are transformed into `UnavailableExceptions`.

The methods `subscribe` and `unsubscribe` are used to (un-)subscribe an address. They save the address list if it was modified by calling `save()` and return a Boolean success value. Note that these methods are both synchronized (on the Servlet object) to ensure the integrity of the address list, both, in memory and on disk. The `save` method serializes the address list to the address file on disk which can be read again by `init` when the Servlet is restarted.

Client interaction:

The client interaction is handled by two of the standard `HttpServlet` methods, `doGet` and `doPost`.

- The `doGet` method replies to GET requests by sending an HTML page which contains the list of the currently subscribe or unsubscribe an address:
- The response content type is again set to `text/html` and the response is marked as not cacheable to proxy servers and clients (because it is dynamically created) by setting an HTTP header "`pragma: no-cache`". The form asks the client to use the POST method for submitting form data.
- Here is a typical output by this method:
- The `doPost` method receives the submitted form data, updates the address list and

sends back a confirmation page:

Finally a confirmation page is sent with the usual method. `Req.getRequestURI ()` is used to get the URI of the Servlet for a link back to the main page (which is created by `doGet`).

As usual, the Servlet extends `javax.http.servlet.HttpServlet` and overrides `getServletInfo` to provide a short notice. At last, here is the full source code of the `ListManagerServlet`.

3.2 BACK END (ORACLE)

Executing SQL Queries:

To really use a database, we need to have some way to execute queries. The simplest way to execute a query is to use the `java.sql.Statement` class. Statement objects are never instantiated directly; instead, a program calls the `createStatement()` method of `Connection` to obtain a new Statement object:

```
Statement stmt=con.crea:Statement();
```

A query that returns data can be executed using the `executeQuery ()` method of `Statement` and returns a `java.sql.ResultSet` that encapsulates the retrieved data:

```
ResultSet rs=stmt.executeQuery(“SELECT * FROM CUSTOMERS”);
```

You can think of a `ResultSet` object as a representation of the query result returned one row at a time. You use the `next()` method of `ResultSet` to move from row to row. The `ResultSet` interface also boasts a multitude of methods designed for retrieving data from the current row. The `getString ()` and `getObject ()` methods are among the most frequently used for retrieving column values:

```
while(rs.next()) {
```

```
String event=rs.getString("even");
Object count=(Integer)rs.getObject("count");
}
```

You should know that the `ResultSet` is linked to its parent `Statement`. Therefore, if a `Statement` is closed or used to execute another query, any related `ResultSet` objects are closed automatically.

Handling SQL Exceptions:

`DBPhoneLookup` encloses most of its code in a try/catch block. This block catches two exceptions: `ClassNotFoundException` and `SQLException`. The former is thrown by the `Class.forName()` method when the JDBC driver class can not be loaded. The latter is thrown by any JDBC method that has a problem. `SQLException` objects are just like any other exception type, with the additional feature that they can chain. The `SQLException` class defines an extra method, `getNextException()`, that allows the exception to encapsulate additional `Exception` objects. We didn't bother with this feature in the previous example, but here's how to use it:

```
Catch(SQLException e) {
out.println(e.getMessage());
while((e=e.getNextException())!=null) {
out.println(e.getMessage());
}
}
```

This code displays the message from the first exception and then loops through all the remaining exceptions, outputting the error message associated with each one. In practice, the first exception will generally include the most relevant information.

Results in Detail:

Before we continue, we should take a closer look at the `ResultSet` interface and related `ResultSetMetaData` interface. In Example9-1, we knew what our query looked like, and we knew what we expected to get back, so we formatted the output appropriately. But, if we want to display the results of a query in an HTML table, it would nice to have some Java code that builds the table automatically from the `ResultSet` rather than having to write the same loop-and-display code over and over. As an added bonus, this kind of code makes it possible to change the contents of the table simply by changing the query.

The `ResultSetMetaData` interface provides a way for a program to learn about the underlying structure of a query result on the fly. We can use it to build an object that dynamically generates an HTML table from a `ResultSet`, as shown in Example9-2. Many Java HTML generation tools have a similar capability.

Handling Null Fields:

Handling null database values with JDBC can be a little tricky(A database field can be set to null to indicate that no value is present, in much the same way that a Java object can be set to null). A method that does not return an object, like `getInt()`, has no way of indicating whether a column is null or whether it contains actual information. (Some drivers return a string that contains the text “null” when `getString()` is called on a null column!) any special value like -1, might be a legitimate value. Therefore, JDBC includes the `wasNull()` method in `ResultSet`, which returns true or false depending on whether the last column read was a true database null. This means that you must read data from the `ResultSet` into a variable, call `wasNull()`, and proceed accordingly. It’s not pretty, but it works. Here’s an example:

```
int age=rs.getInt("age");
if(!rs.wasNull())
out.println("Age:"+age);
```


Another way to check for null values is to use the getObject() method. If a column is null, getObject() always returns null. Compare this to the getString() method that has been known, in some implementations, to return the empty string if a column is null. Using getObject() eliminates the need to call wasNull() and leads to simpler code.

Updating the Database:

Most database-enabled web sites need to do more than just perform queries. When a client submits an order or provides some kind of information, the data needs to be entered into the database. When you know you're executing a SQL UPDATE, INSERT, or DELETE statement and you know you don't expect a ResultSet, you can use the executeUpdate() method of statement. It returns a count that indicates the number of rows modified by the statement. It's used like this:

```
int count= stmt.executeUpdate("DELETE FROM CUSTOMERS WHERE  
CUSTOMER_ID=5")
```

If you are executing SQL, that may return either a ResultSet or a count (say, if you're handling user-submitted SQL or building generic data-handling classes), use the generic execute() method of statement. It returns a Boolean whose value is true if the SQL statement produced one or more ResultSet objects or false if it resulted in an update count:

```
boolean b=stmt.execute(sql);
```

The getResultSet() and getUpdateCount() method of statement provide access to the results of the execute() method.

Using Prepared Statements:

A prepared statement object is like a regular statement object, in that it can be used to execute SQL statements. The important difference is that the SQL in a PreparedStatement

is precompiled by the database for faster execution. Once a PreparedStatement has been compiled, it can still be customized by adjusting predefined parameters. Prepared statements are useful in applications that have to run the same general SQL command over and over.

Use the `prepareStatement (String)` method of connection to create `PreparedStatement` objects. Use the `?` Character as a placeholder for values to be substituted later. For example:

```
PreparedStatement pstmt = con.prepareStatement  
("INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID.TOTAL) VALUES (?,?,?)");  
INSERT INTO MUSKETEERS (NAME) VALUES ('John d'Artagan')
```

As you see, the string terminates twice. One solution is to manually replace the single quote `'` with two single quotes `''`, the Oracle escape sequence for one single quote. This solution, requires you to escape every character that your database treats as special—not an easy task and not consistent with writing platform-independent code. A far better solution is to use a `PreparedStatement` and pass the string using its `setString()` method, as shown below. The `PreparedStatement` automatically escapes the string as necessary for your database:

```
PreparedStatement pstmt = con.prepareStatement  
("INSERT INTO MUSKETEERS (NAME) VALUES (?)") ;  
Pstmt.setString (1,"John d'Artagan");  
Pstmt.executeUpdate ();
```

3.2 MODULES:

There are 5 modules in this project.

- Administrator Module.
- Reservation Agent Module.
- Passenger Module.
- Payment.
- Cancellation.

MODULES EXPLANATION:

- Administrator Module.

Enables the administrator to perform all administrative functions and manage inventory over LAN or the Internet. The administrator can define or modify routes, fares schedules and assign or deny access for qualified travel agents and other authorized users.

- Reservation Agent Module.

Allows the airlines reservation agents to make and modify reservation on the LAN or over the internet. The reservation agents could be stationed at any airline office location.

- Passenger Module.

This module enables online customers to make reservations, views their bookings, make special service requests and define their preferences over the web.

- Payment.

Provides the airline with the ability to set up various travel agents and give them reservations capabilities over the Internet. The travel agents are able to display and offer discounted fares to passengers.

- Cancellation.

The system should allow the user to cancel the existing booking. In this cancellation very helpful in all the travelers.

4. SYSTEM DESIGN:

4.1 UML DIAGRAMS:

The Unified Modeling Language prescribes a standard set of diagrams and notations for modeling object oriented systems, and describe the underlying semantics of what these diagrams and symbols mean. Whereas there has been to this point many notations and methods used for object-oriented design, now there is a single notation for modelers to learn.

UML can be used to model different kinds of systems: software systems, hardware systems, and real-world organizations. UML offers nine diagrams in which to model systems:

- **Use Case diagram** for modeling the business processes
- **Sequence diagram** for modeling message passing between objects
- **Collaboration diagram** for modeling object interactions
- **State diagram** for modeling the behavior of objects in the system
- **Activity diagram** for modeling the behavior of Use Cases, Objects, or Operations
- **Class diagram** for modeling the static structure of classes in the system
- **Object diagram** for modeling the static structure of objects in the system
- **Component diagram** for modeling components

- **Deployment diagram** for modeling distribution of the system.

UML is a consolidation of many of the most used object-oriented notations and concepts. It began as a consolidation of the work of Grady Booch, James Rumbaugh, and Ivar Jacobson, creators of three of the most popular object-oriented methodologies.

In 1996, the Object Management Group(OMG), a standards body for the object-oriented community, issued a request for proposal for a standard object-oriented analysis notation and semantic meta model. UML, version 1.0, was proposed as an answer to this submission in January of 1997. There were five other rival submissions. During the course of 1997, all six submitters united their work and presented to OMG a revised UML document, called UML version 1.1. This document was approved by the OMG in November 1997. The OMG calls this document OMG UML version 1.1. The OMG is currently in the process of performing a technical.

4.1.1 CLASS DIAGRAMS:

The class diagram is the main static analysis and design diagram for a system. In it, the class structure of the system is specified, with relationships between classes and inheritance structures. During analysis of the system, the diagram is developed with an eye for an ideal solution. During design, the same diagram is used, and modified to conform to implementation details.

4.1.2. ACTIVITY DIAGRAMS:

The Activity Diagram is a multi-purpose process flow diagram that is used to model behavior of the system. Activity Diagram can be used to model a Use Case, or a class, or a complicated method. An Activity Diagram can show parallel processing. This is important when using Activity Diagram to model business processes, some of which can be performed in parallel, and for modeling multiple threads in concurrent programs.

4.1.3 SEQUENCE DIAGRAM:

The Sequence diagram is one of the most effective diagrams to model object interactions in a system. A Sequence diagram is modeled for every Use Case. Whereas the Use Case diagram enables modeling of a business view of the scenario, the Sequence diagram contains implementation details of the scenario, including the objects and classes that are used to implement the scenario, and messages passed between the objects.

4.1.4 USE CASE DIAGRAM:

Use Case modeling is the simplest and most effective technique for modeling system requirements from a user's perspective. Use Cases are used to model how a system or business currently works, or how the users wish it to work. It is not really an object-oriented approach; it is really a form of process modeling. It is, however, an excellent way to lead into object-oriented analysis of systems. Use Cases are generally the starting point of object-oriented analysis with UML. The Use Case model consists of actors and Use Cases. Actors represent users and other systems that interact with the system. They are drawn as stick figures. They actually represent a type of user, not an instance of a user. Use Cases represent the behavior of the system, scenario that the system goes through in response to stimuli from an actor. They are drawn as Ellipses.

Each Use Case is documented by a description of the scenario. The description can be written in textual form or in a step-by-step format. Each Use Case can also be defined by other properties, such as the pre- and post conditions of the scenario – conditions that exist before the scenario begins, and conditions that exist after the scenario completes.

4.1.5 COLLABORATION DIAGRAM:

The Collaboration Diagram presents an alternate to the Sequence Diagram for modeling interactions between objects in the system. Whereas in the Sequence Diagram the focus is on the chronological sequence of the scenario being modeled, in the Collaboration Diagram the focus is on understanding all of the effects on a given object

during a scenario.

Objects are connected by links, each link representing an instance of an association between the respective classes involved. The link shows messages sent between the objects, the type of message passed, and the visibility of objects to each other.

4.1.6 DATAFLOW DIAGRAMS:

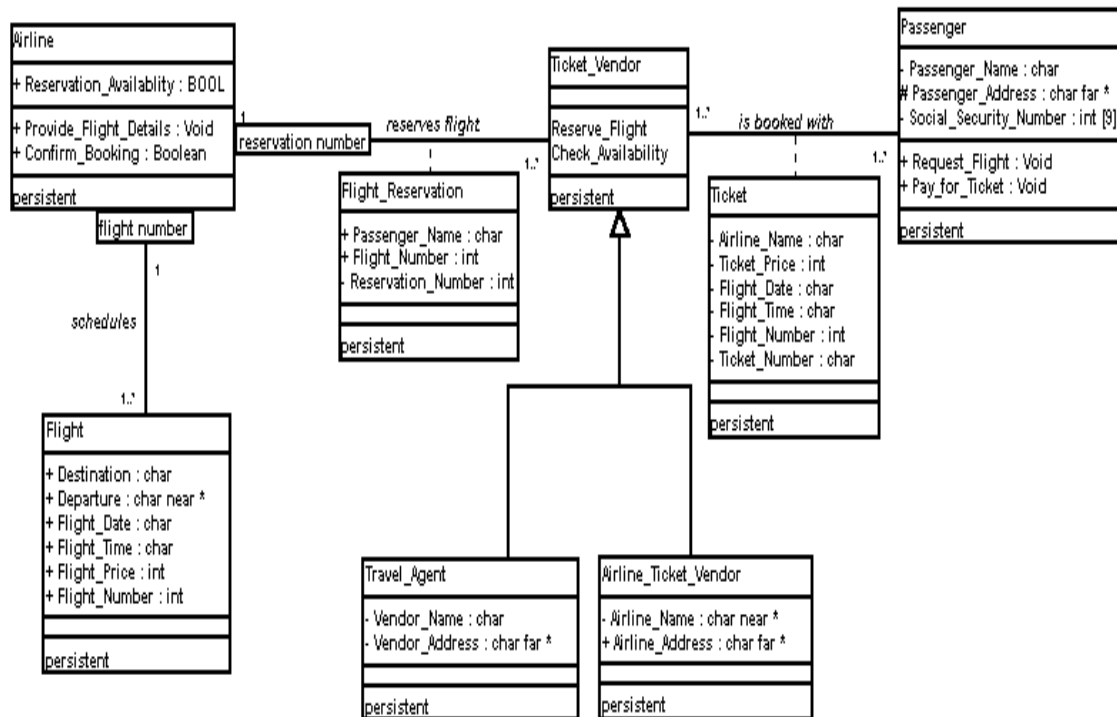


Figure 1 Class Diagram

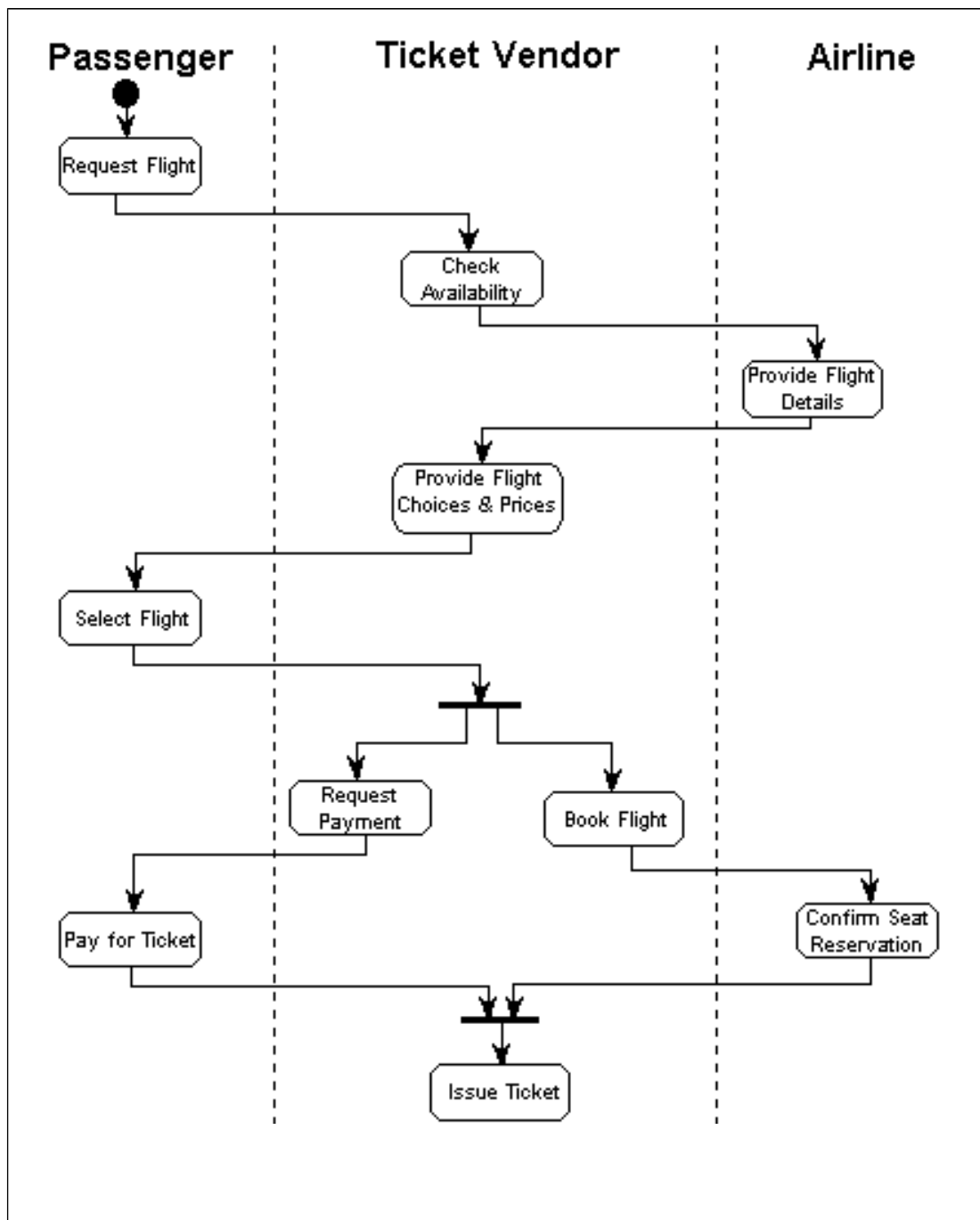


Figure 2 Activity Diagram

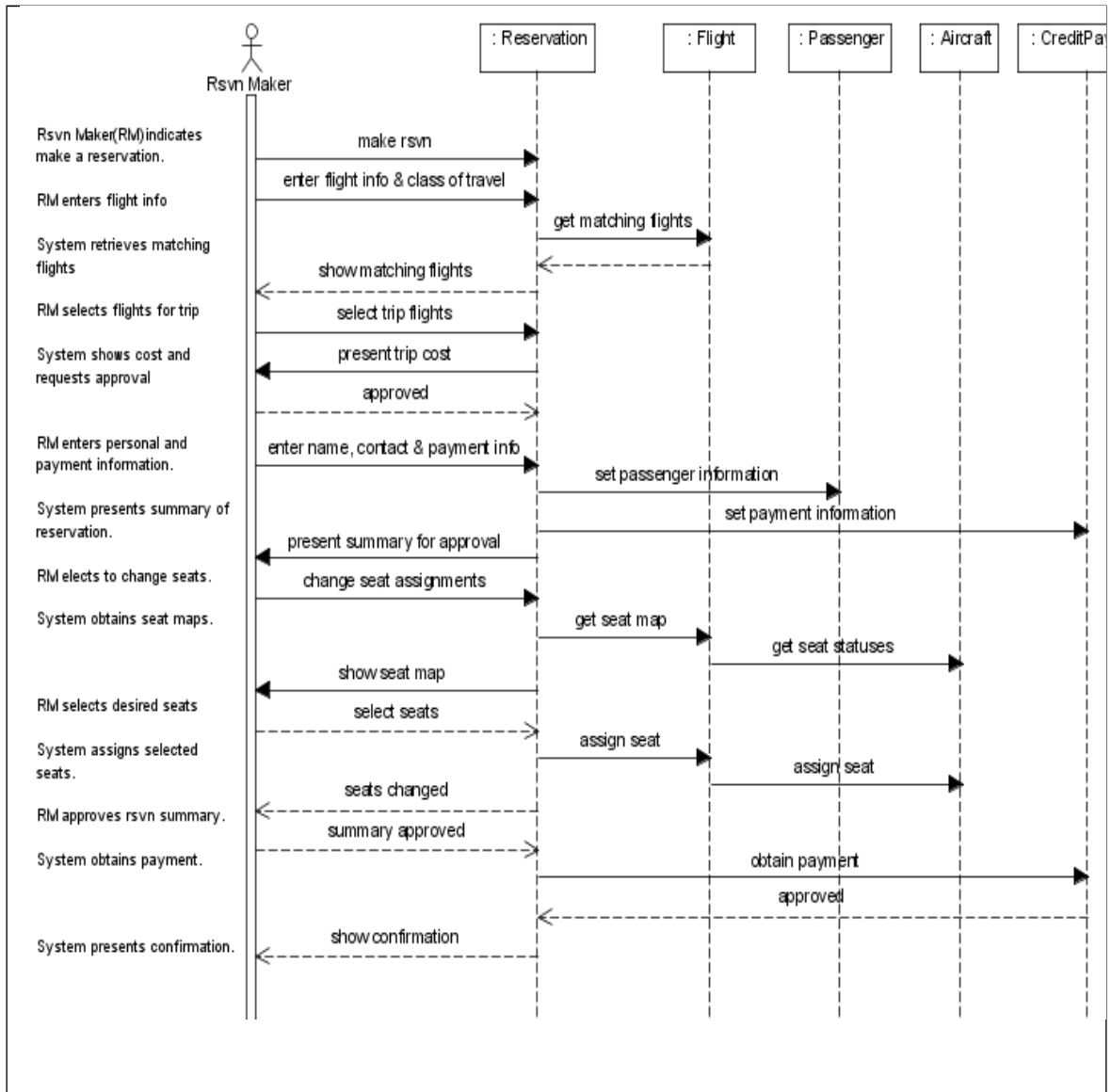


Figure 3 Sequence Diagram

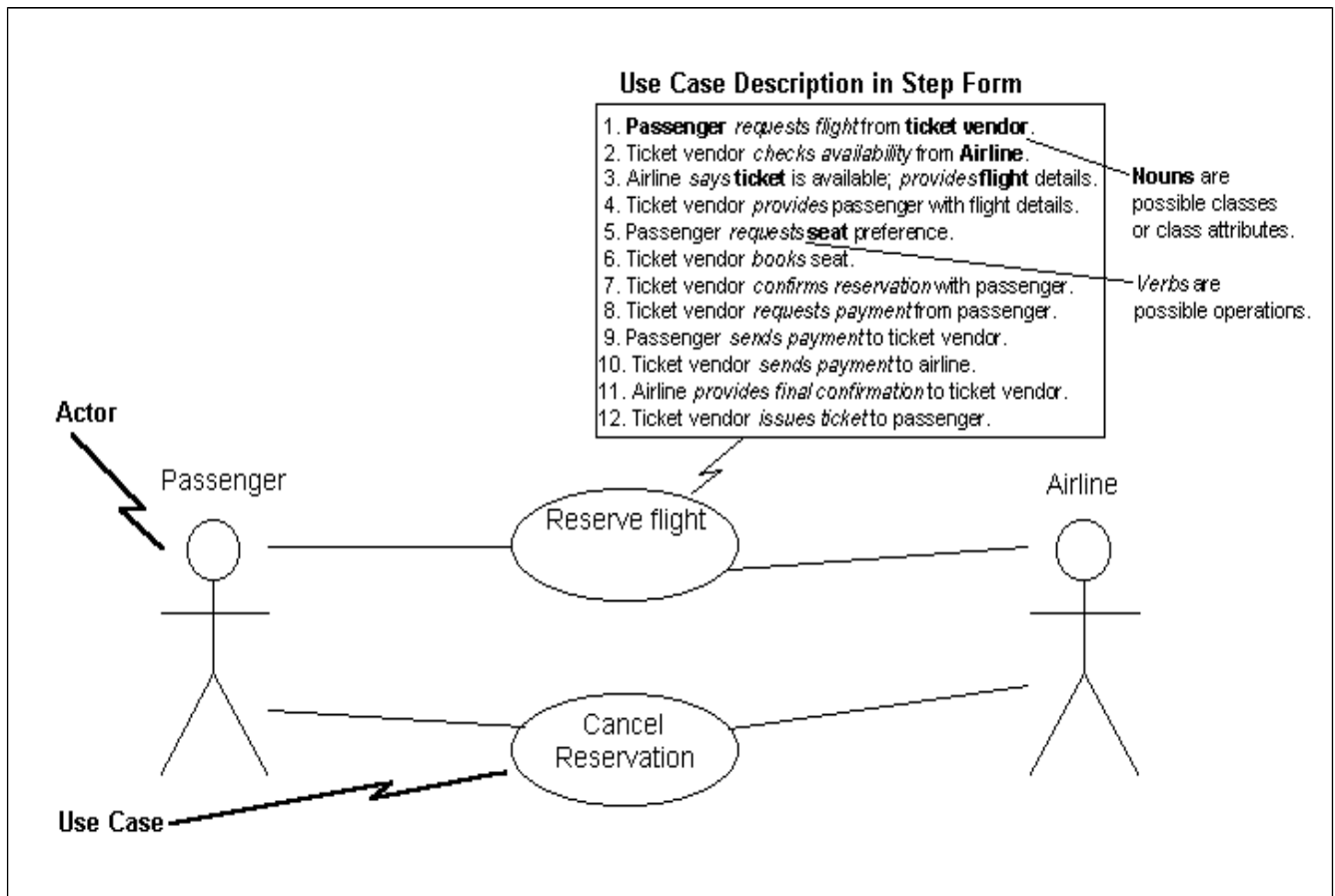


Figure 4 Use Case Diagrams

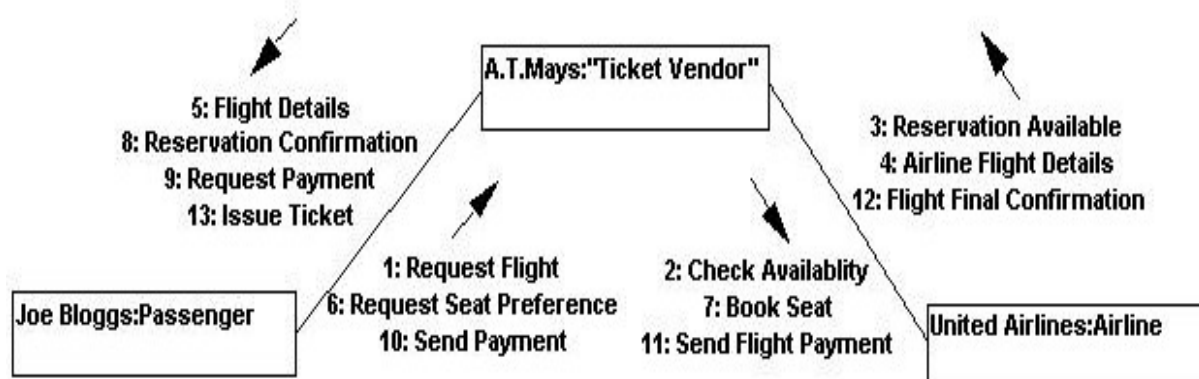


Figure 5 Collaboration Diagram

4.2 DATA DICTIONARY:

4.2.1 INTRODUCTION:

DICTIONARY The logical characteristics of current system data stores including Name, Address, Flight code, Source, Destination, Airline code, Flight code, Credit card number, Payment amount etc identifies process where the data are used and where immediate access to information required, Serves as the basis for identifying database requirements during system design.

Uses of Data Dictionary:

- To manage the details in large systems.
- To communicate a common meaning for all system elements.
- To document the features of the system.
- To facilities analysis of the details in order to evaluate characteristics and determine where system changes should be made.
- To locate errors and omissions in the system.

4.2.2 DATA DICTIONARY:

1. Cancellation.

This table is used to store the cancel details.

Field name	Description	Data type	Size	Constraints
cancelid	Cancellation id	int	10	PRIMARY KEY
reservationid	Reservation id	int	10	FOREIGN KEY
cancelationdate	Date of Cancellation	date		NOT NULL
refundmoney	Money to be refundable	decimal	10,0	NOT NULL

2. Classes

This table is used to store the class details.

Field name	Description	Data type	Size	constraints
Classid	Id of the class	int	10	PRIMARY KEY
flightcode	-	int	10	FOREIGN KEY
classcode	-	varchar	50	NOT NULL
classname	Name of the class	varchar	50	NOT NULL
Fare	-	decimal	10,0	NOT NULL
totalclasseat	Total seats in a class	int	10	NOT NULL

3. Flight days

This table is used to store the flight day's details

Field name	Description	Data type	Size	constraints
datecode	Code of the date flight departure	int	10	PRIMARY KEY
flightcode	Code of the flight	int	10	FOREIGN KE
Date	date	date		NOT NULL
departure	Departure time	time		NOT NULL

4. Flight details

This table is used to store the flight details.

Field name	Description	Data type	Size	constraints
flightcode	Code of the flight	int	10	PRIMARY KEY
airlinecode	Code of the airlines	varchar	100	NOT NULL
flightname	Name of the flight	varchar	100	NOT NULL
source	Starting place of the	varchar	100	NOT NULL

	flight			
destination	Destination of the flight	varchar	100	NOT NULL
totalcapacity	Total capacity of the flight	int	10	NOT NULL

5. Login

This table is used to store the login details

Field name	Description	Data type	Size	constraints
username	Name of the user	varchar	50	PRIMARY KEY
password	Password for the user login	varchar	50	NOT NULL

6. Mail

This table is used to store the mail details

Field name	Description	Data type	Size	constraints
mailid	Users mail id	int	10	PRIMARY KEY
Touser	Destination of the mail	varchar	250	FOREIGN KEY
fromuser	By whom the mail is sent	varchar	250	FOREIGN KEY
Subject	Subject of the mail	varchar	250	NOT NULL
message	Message to be sent	text		NOT NULL

7. New User

This table is used to store the new user details.

Field name	Description	Data type	Size	constraints
Userid	Users id	int	10	PRIMARY KEY
Firstname	First name of the user	varchar	250	NOT NULL
lastname	Last name of the user	varchar	250	NOT NULL

middlename	Middle name of the user	varchar	250	NOT NULL
mobilenno	Mobile number	int	10	NOT NULL
Dob	Date of birth	date		NOT NULL
address	-	varchar	250	NOT NULL
City	-	varchar	250	NOT NULL
State	-	varchar	250	NOT NULL
country	-	varchar	250	NOT NULL
emailid	Mail id	varchar	250	NOT NULL
password	-	varchar	250	NOT NULL
usertype	Type of the user	varchar	250	NOT NULL
username	Name of the user	varchar	250	UNIQUE KEY

8. Passenger

This table is used to store passenger details.

Field name	Description	Data type	Size	constraints
passengerno	Number of the passengers	Int	10	PRIMARY KEY
flightcode	Code of the flight	Int	10	FOREIGN KEY
Dob	Date of birth	Date		NOT NULL
address	-	varchar	250	NOT NULL
nationality	-	varchar	250	NOT NULL
Name	Name of the user	varchar	250	NOT NULL
Gender	-	varchar	250	NOT NULL
phoneno	Phone number	varchar	250	NOT NULL
emailid	Mail id	varchar	250	NOT NULL
passportno	Passport number	Int	10	NOT NULL
reservationid	Reservation id	Int	10	FOREIGN KEY

9. Payment

This table is used to store payment details

Field name	Description	Data type	Size	constraints
paymentid	-	Int	10	PRIMARY KEY
checkno	Checking number	Int	10	NOT NULL
creditcardno	-	Int	10	NOT NULL
paidamount	-	decimal	10,0	NOT NULL
paymentdate	-	date		NOT NULL
passengerno	-	Int	10	NOT NULL

10. Reservation

This table is used to store reservation details.

Field name	Description	Data type	Size	constraints
reservationid	-	Int	10	PRIMARY KEY
flightcode	-	Int	10	FOREIGN KEY
journeydate	-	Date		NOT NULL
Source	Starting place of the flight	varchar	50	NOT NULL
destination	Ending position of the flight	varchar	50	NOT NULL
Status	-	Int	10	NOT NULL
journeytime	Time the flight starts	time		NOT NULL

11. Reserve check

This table is used to store reserve check details

Field name	Description	Data type	Size	constraints
reservationid	Reservation id number	Int	10	PRIMARY KEY
username	Name of the user	Varchar	250	
Password	Password to login	Varchar	250	

5. IMPLEMENTATION:

modules of this project consist of

- 1)Main Menu
- 2)Reservation
- 3) Ticket
- 4)waiting
- 5)Warning
- 6)Confirmed
- 7)Create
- 8>Login
- 9)Message Box
- 10)Project

Main Menu

```
import java.awt.*;
import java.awt.event.*;
public class MainMenu extends Frame implements ActionListener {
    MenuBar mbar;
    Menu m1,m2,m3;
    MenuItem m1_1,m1_2,m2_1,m2_2,m2_3,m3_1;
    public MainMenu() {
        mbar = new MenuBar();
        setMenuBar(mbar);
        m1=new Menu("Bookings");
        mbar.add(m1);
        m1_1 = new MenuItem("Reservation");
        m1.add(m1_1);
        m1_2 = new MenuItem("Cancellation");
        m1.add(m1_2);
        m2=new Menu("Reports");
        mbar.add(m2);
        m2_1 = new MenuItem("Confirmed Passengers");
        m2.add(m2_1);
        m2_2 = new MenuItem("Waiting");
        m2.add(m2_2);
        m2_3 = new MenuItem("Daily Collection Report");
        m2.add(m2_3);
        m3=new Menu("Close");
```



```

mbar.add(m3);
m3_1 = new MenuItem("Close");
m3.add(m3_1);
m1_1.addActionListener(this);
m1_2.addActionListener(this);
m2_1.addActionListener(this);
m2_2.addActionListener(this);
m2_3.addActionListener(this);
m3_1.addActionListener(this);
addWindowListener(new M());
}
public void actionPerformed(ActionEvent ae) {
    if(ae.getSource()==m1_1) {
        Reservation r = new Reservation();
        r.setSize(400,400);
        r.setVisible(true);
        r.setTitle("Reservation Screen");
    } if(ae.getSource()==m1_2) {
        Cancellation c = new Cancellation();
        c.setSize(400,400);
        c.setVisible(true);
        c.setTitle("Cancellation Screen");
    } if(ae.getSource()==m2_1) {
        Confirmed cr = new Confirmed();
        cr.setSize(400,400);
        cr.setVisible(true);
        cr.setTitle("Confirmed Passengers List");
    }
    if(ae.getSource()==m2_2) {
        Waiting wr = new Waiting();
        wr.setSize(400,400);
        wr.setVisible(true);
        wr.setTitle("Waiting List");
    }
    if(ae.getSource()==m2_3) {
        Collection dcr = new Collection();
        dcr.setSize(400,400);
        dcr.setVisible(true);
        dcr.setTitle("Daily Collection Report");
    } if(ae.getSource()==m3_1) {
        System.exit(0);
    }
}
/*public static void main(String args[]) {
    MainMenu m = new MainMenu();
    m.setTitle("Main Menu");
    m.setSize(400,400);
}

```

```

m.setVisible(true);
}*/
class M extends WindowAdapter {
public void windowClosing(WindowEvent e) {
setVisible(false);
dispose();
}
}
}
}

```

Reservation Module

```

import java.awt.*;
import java.awt.event.*;
public class Reservation extends Frame implements ActionListener {
Button b1,b2,b3;
Label l1,l2;
GridBagLayout gbl;
GridBagConstraints gbc;
Font f;
Reservation() {
setBackground(Color.cyan);
f = new Font("TimesRoman",Font.BOLD,20);
gbl=new GridBagLayout();
gbc=new GridBagConstraints();
setLayout(gbl);
b1=new Button("Check Availability");
b1.setFont(f);
b2=new Button(" Create Passenger ");
b2.setFont(f);
// b3=new Button(" Fare Teller ");
// b3.setFont(f);
l1= new Label("");
l2= new Label("");
gbc.gridx=0;
gbc.gridy=0;
gbl.setConstraints(b1,gbc);
add(b1);
gbc.gridx=0;
gbc.gridy=4;
gbl.setConstraints(l1,gbc);
add(l1);
gbc.gridx=0;
gbc.gridy=8;
gbl.setConstraints(b2,gbc);
add(b2);
/* gbc.gridx=0; gbc.gridy=12;

```

```

    gbl.setConstraints(l2,gbc);
    add(l2);
    gbc.gridx=0;
    gbc.gridy=16;
    gbl.setConstraints(b3,gbc);
    add(b3);
    */
    b1.addActionListener(this);
    b2.addActionListener(this);
    // b3.addActionListener(this);
    addWindowListener(new W());
    }
    public void actionPerformed(ActionEvent ae) {
    if(ae.getSource()==b1) {
    Check m = new Check();
    //setVisible(false);
    m.setSize(400,400);
    m.setVisible(true);
    m.setTitle("Check Availability Screen");
    } if(ae.getSource()==b2) {
    Create v = new Create();
    //setVisible(false);
    v.setSize(400,500);
    v.setVisible(true);
    v.setTitle("Create Passenger Screen");
    } /* if(ae.getSource()==b3) { Fare f = new Fare();
    //setVisible(false);
    f.setSize(400,500);
    f.setVisible(true);
    f.setTitle("Fare Teller Screen"); }
    */
    } class W extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
    setVisible(false);
    dispose();
    }
    }
    }
    }

```

Ticket Module

```

import java.sql.*;
import java.awt.*;
import java.awt.event.*;
public class Ticket extends Frame implements ActionListener
{

```

```

TextField t1;
Label l1;
Button b1;
GridBagLayout gbl;
GridBagConstraints gbc;
Connection con;
PreparedStatement ps;
Statement stmt;
ResultSet rs;
int count;
Font f;
Ticket()
{
setBackground(Color.cyan);
t1 = new TextField(20);
l1 = new Label("PNR NO ");
l1.setFont(f);
gbc.gridx=0;
gbc.gridy=0;
gbl.setConstraints(l1,gbc);
add(l1);
gbc.gridx=0;
gbc.gridy=2;
gbl.setConstraints(t1,gbc);
add(t1);
addWindowListener(new W());
}
public void actionPerformed(ActionEvent ae)
{
}
class W extends WindowAdapter
{
public void windowClosing(WindowEvent e)
{
setVisible(false);
dispose();
}
}
/* public static void main(String args[])
{
Ticket t = new Ticket();
t.setSize(400,500);
t.setVisible(true);
t.setTitle("Ticket Screen");
}*/
}

```

Waiting Module

```
import java.awt.*;
import java.awt.event.*;
public class Waiting extends Frame {
    Waiting() {
        addWindowListener(new W());
    }
    class W extends WindowAdapter {
        public void windowClosing(WindowEvent e) {
            setVisible(false);
            //dispose();
            System.exit(0);
        }
    }
}
```

Warning Module

```
import java.awt.*;
import java.awt.event.*;
public class Warning extends Frame {
    GridLayout g;
    Button b1;
    Label l;
    Warning() {
        g = new GridLayout(2,1,10,40);
        setLayout(g);
        l = new Label("Incorrect username or password",Label.CENTER);
        b1 = new Button("Ok");
        add(l);
        add(b1);
        b1.addActionListener(new Y());
        addWindowListener(new X());
    } class Y implements ActionListener {
        public void actionPerformed(ActionEvent ae) {
            if(ae.getSource()==b1) {
                //dispose(); System.exit(0);
            }
        }
    }
    class X extends WindowAdapter { public void windowClosing(WindowEvent e) {
        setVisible(false);
        dispose();
    }
}
```

```

public Insets getInsets() {
return new Insets(40,40,40,40);
} /*public static void main(String args[]) {
Warning m = new Warning();
m.setTitle("Message Box");
m.setSize(300,200);
m.setVisible(true);
}*/
}

```

Confirmed Module

```

import java.awt.*;
import java.awt.event.*;
public class Confirmed extends Frame
{
Confirmed()
{
addWindowListener(new W());
}
class W extends WindowAdapter
{
public void windowClosing(WindowEvent e)
{
setVisible(false);
//dispose();
System.exit(0);
}
}
}

```

Creation Module

```

import java.awt.*;
import java.awt.event.*;
public class Confirmed extends Frame
{
Confirmed()
{
addWindowListener(new W());
}
class W extends WindowAdapter
{
public void windowClosing(WindowEvent e)
{
setVisible(false);
//dispose();
}
}
}

```

```

System.exit(0);
}
}
}
}

```

Login Module

```

import java.awt.*;
import java.awt.event.*;
public class Login extends Frame implements ActionListener { String username = "anu";
String password = "rag";
TextField t1,t2;
Label l1,l2,l3,l4,l5,l6;
Button b2,b3,b4;
GridBagLayout gbl;
GridBagConstraints gbc;
Font f1,f2;
public Login() { //setTitle("Login Screen");
//g = new GridLayout(4,2,0,60);
//setLayout(g);
setBackground(Color.cyan);
f1 = new Font("TimesRoman",Font.BOLD,20);
f2 = new Font("TimesRoman",Font.BOLD,15);
gbl=new GridBagLayout();
gbc=new GridBagConstraints();
setLayout(gbl); l1 = new Label("Username",Label.CENTER);
l1.setFont(f1);
l2 = new Label("Password",Label.CENTER);
l2.setFont(f1);
l3 = new Label("");
l4 = new Label("");
l5 = new Label("");
l6 = new Label("");
t1 = new TextField(15);
t2 = new TextField(15);
t2.setEchoChar('*');
//b1 = new Button("Change Login Details");
b2 = new Button("Reset");
b2.setFont(f2);
b3 = new Button("Submit");
b3.setFont(f2);
b4 = new Button("Close");
b4.setFont(f2);
gbc.gridx=0;
gbc.gridy=0;
gbl.setConstraints(l1,gbc);

```

```

add(l1); gbc.gridx=2;
gbc.gridy=0;
gbl.setConstraints(t1,gbc);
add(t1);
gbc.gridx=0;
gbc.gridy=2;
gbl.setConstraints(l2,gbc);
add(l2);
gbc.gridx=2;
gbc.gridy=2;
gbl.setConstraints(t2,gbc);
add(t2);
gbc.gridx=0;
gbc.gridy=4;
gbl.setConstraints(l3,gbc);
add(l3);
gbc.gridx=2;
gbc.gridy=4;
gbl.setConstraints(l4,gbc);
add(l4);
gbc.gridx=0;
gbc.gridy=6;
gbl.setConstraints(b2,gbc);
add(b2);
gbc.gridx=2;
gbc.gridy=6;
gbl.setConstraints(b3,gbc);
add(b3);
gbc.gridx=0;
gbc.gridy=8;
gbl.setConstraints(l4,gbc);
add(l4);
gbc.gridx=2;
gbc.gridy=8;
gbl.setConstraints(l5,gbc);
add(l5);
gbc.gridx=0;
gbc.gridy=10;
gbl.setConstraints(b4,gbc);
add(b4);
//add(l1);
//add(t1);
//add(l2);
//add(t2);
//add(b1);
//add(b2);

```



```

//add(b3);
//add(b4);
//b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this); }
/*public Insets getInsets() { return new Insets(40,40,40,40); }*/
public void actionPerformed(ActionEvent ae) {
if(ae.getSource()==b2) {
t1.setText("");
t2.setText(""); }
if(ae.getSource()==b4) {
System.exit(0); }
if(ae.getSource()==b3)
{ if((t1.getText().equals(username))&&(t2.getText().equals(password))) {
MainMenu m = new MainMenu();
setVisible(false);
m.setSize(400,400);
m.setVisible(true);
m.setTitle("Main Menu"); }
else { //Warning w = new Warning();
//w.setSize(300,200);
//w.setVisible(true);
//w.setTitle("Message Box");
MessageBox mb = new MessageBox(this);
mb.setLocation(200,200);
mb.setVisible(true); } }
/*if(ae.getSource() == b1) {
Change c = new Change();
c.setSize(400,400);
c.setVisible(true);
c.setTitle("Screen for Changing Login Details"); }*/

```

MessageBox

```

import java.awt.*;
import java.awt.event.*;
public class MessageBox extends Dialog implements ActionListener{
GridBagLayout gbl;
GridBagConstraints gbc;
FlowLayout F;
Button b1;
Label l;
Font f1,f2;
MessageBox(Frame fm) {
super(fm,true);

```

```

setBackground(Color.cyan);
f1 = new Font("Times Roman",Font.BOLD,20);
f2 = new Font("Times Roman",Font.BOLD,15);
gbl=new GridBagLayout();
gbc=new GridBagConstraints();
setLayout(gbl);
l=new Label("Incorrect username or password",Label.CENTER);
l.setFont(f1);
b1 = new Button(" OK ");
b1.setFont(f2);
gbc.gridx=0;
gbc.gridy=0;
gbl.setConstraints(l,gbc);
add(l);
gbc.gridx=0;
gbc.gridy=4;
gbl.setConstraints(b1,gbc);
add(b1);
setSize(350,200);
setTitle("Message Box");
b1.addActionListener(this);
addWindowListener(new X());
} public void actionPerformed(ActionEvent ae) {
if(ae.getSource()==b1) {
setVisible(false);
dispose();
}
}
class X extends WindowAdapter { public void windowClosing(WindowEvent e) {
setVisible(false);
dispose(); }
}
}

```

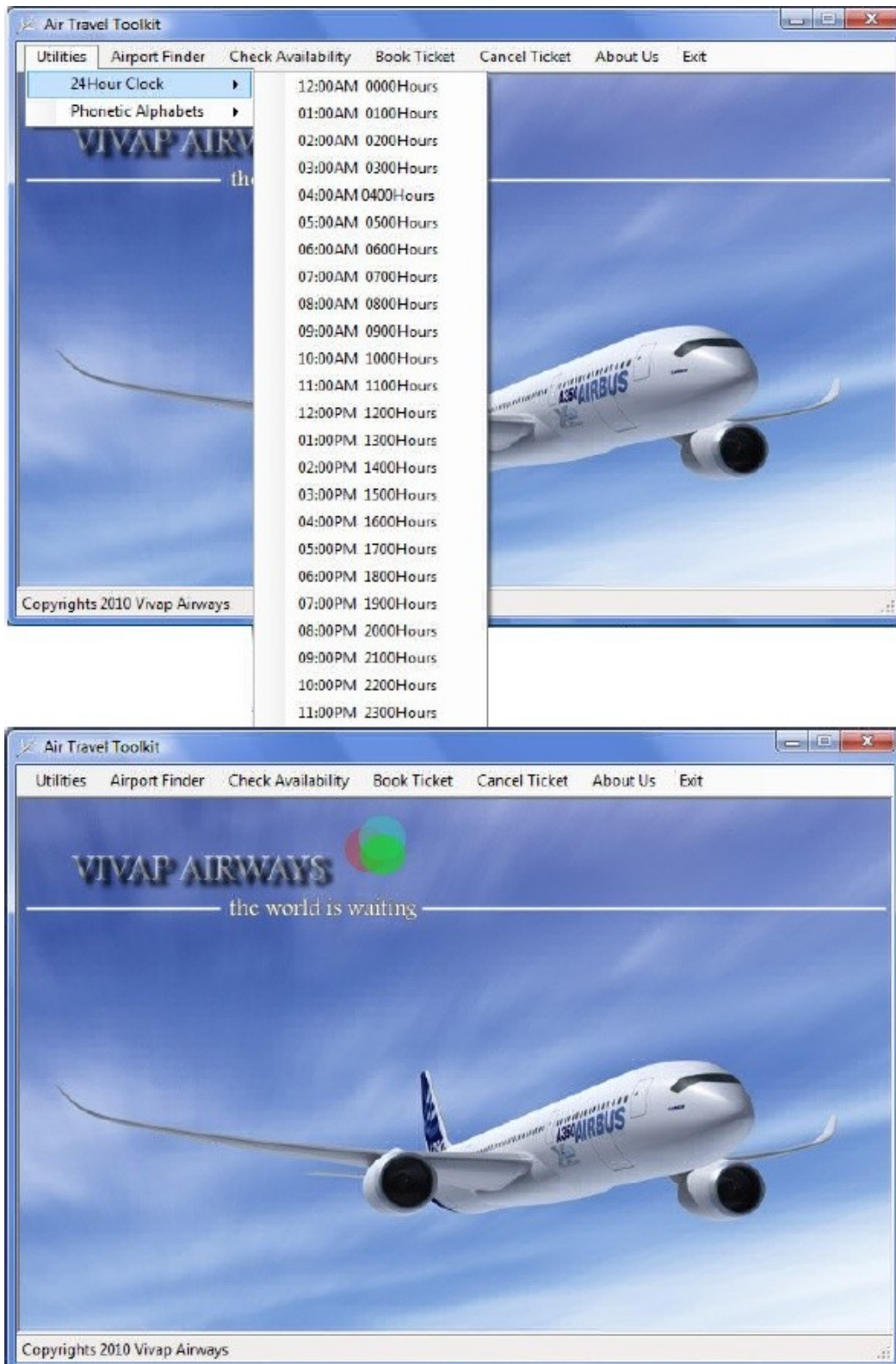
Project Module

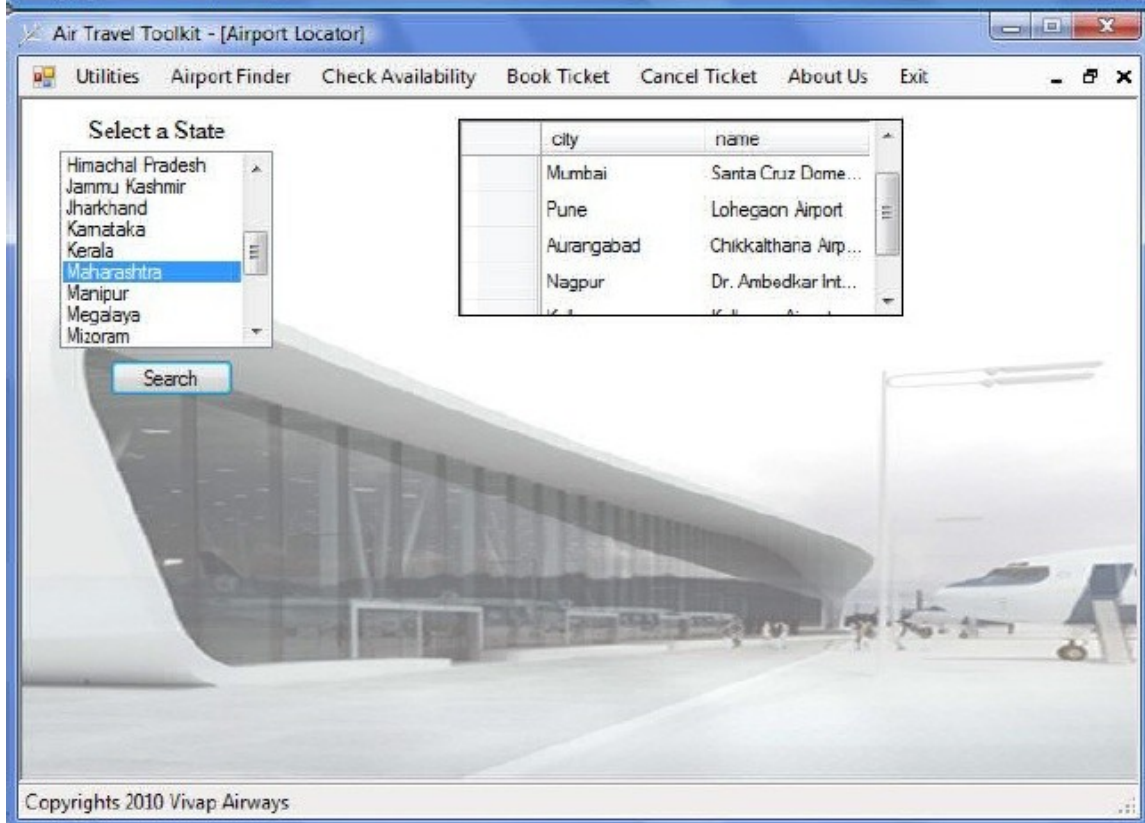
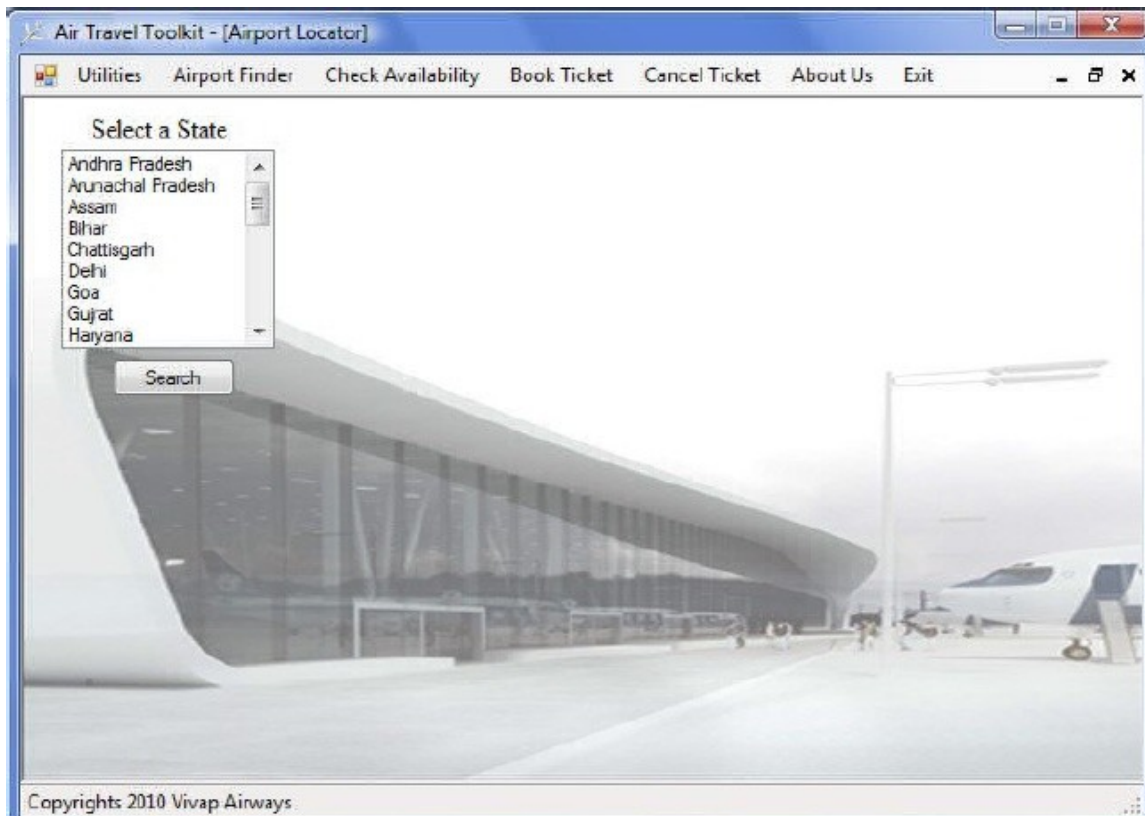
```

import java.awt.*;
import java.awt.event.*;
public class Project extends Frame {public static void main(String args[]){
Login L = new Login();
L.setLocation(200,100);
L.setSize(300,300);
L.setVisible(true);
L.setTitle("Login Screen");
}
}

```

6. EXPERIMENTAL RESULTS:





Air Travel Toolkit - [Check Availability]

Utilities Airport Finder Check Availability Book Ticket Cancel Ticket About Us Exit

VIVAP AIRWAYS

From To

Class

July, 2010

Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Today: 04-07-2010

Copyrights 2010 Vivap Airways

Air Travel Toolkit - [Form8]

Utilities Airport Finder Check Availability Book Ticket Cancel Ticket About Us Exit

VIVAP AIRWAYS

Click to Check	Flight_Name	Departure_Time	Arrival_Time	EconomyFare
<input type="button" value="▶"/>	6D-123	11:00	12:00	3200
<input type="button" value="▶"/>	6D-124	13:00	14:00	3029
<input type="button" value="▶"/>	6D-125	22:00	23:00	3849

53 Seats are Available

Copyrights 2010 Vivap Airways

Air Travel Toolkit - [Passenger's Information]

Utilities Airport Finder Check Availability Book Ticket Cancel Ticket About Us Exit

VIVAP AIRWAYS

Book Now

Airfare Rs 15145

	Name	Email ID	Contact Number	Age
▶				

Copyrights 2010 Vivap Airways

Air Travel Toolkit - [One Way Trip]

Utilities Airport Finder Check Availability Book Ticket Cancel Ticket About Us Exit

VIVAP AIRWAYS

From To

Date of Journey

Number of Passengers

Class

July, 2010

Mon	Tue	Wed	Thu	Fri	Sat	Sun
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8


Today: 04-07-2010

Search Flights

Copyrights 2010 Vivap Airways

Air Travel Toolkit - [Passenger's Information]

Utilities
Airport Finder
Check Availability
Book Ticket
Cancel Ticket
About Us
Exit



VIVAP AIRWAYS

Book Now

AirFare Rs 15145


	Name	Email ID	Contact Number	Age
	Vidya Niwas	holyanimal@in.com	9999662636	20
	Vatealya	happy_it@redff...	9873580954	20
	Prateek Bamel	prateekbamel@g...	9999789900	20
	Yashwardhan	yash_00998@gm...	9987622310	20
▶	Ankit Yadav	ankit_yadav009...	987354210	21

Form19

PAYMENT OPTION

AirFare Rs 15145

☐ Cash
☐ Card



Copyrights 2010 Vivap Airways

Air Travel Toolkit - [Cancel Ticket]

Utilities Airport Finder Check Availability Book Ticket Cancel Ticket About Us Exit

VIVAP AIRWAYS

Flight Name: 6C-123

Date of Journey:

PNR:

Class: Economy

GO

July, 2010

Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Today: 04-07-2010

Copyrights 2010 Vivap Airways

Air Travel Toolkit - [Form21]

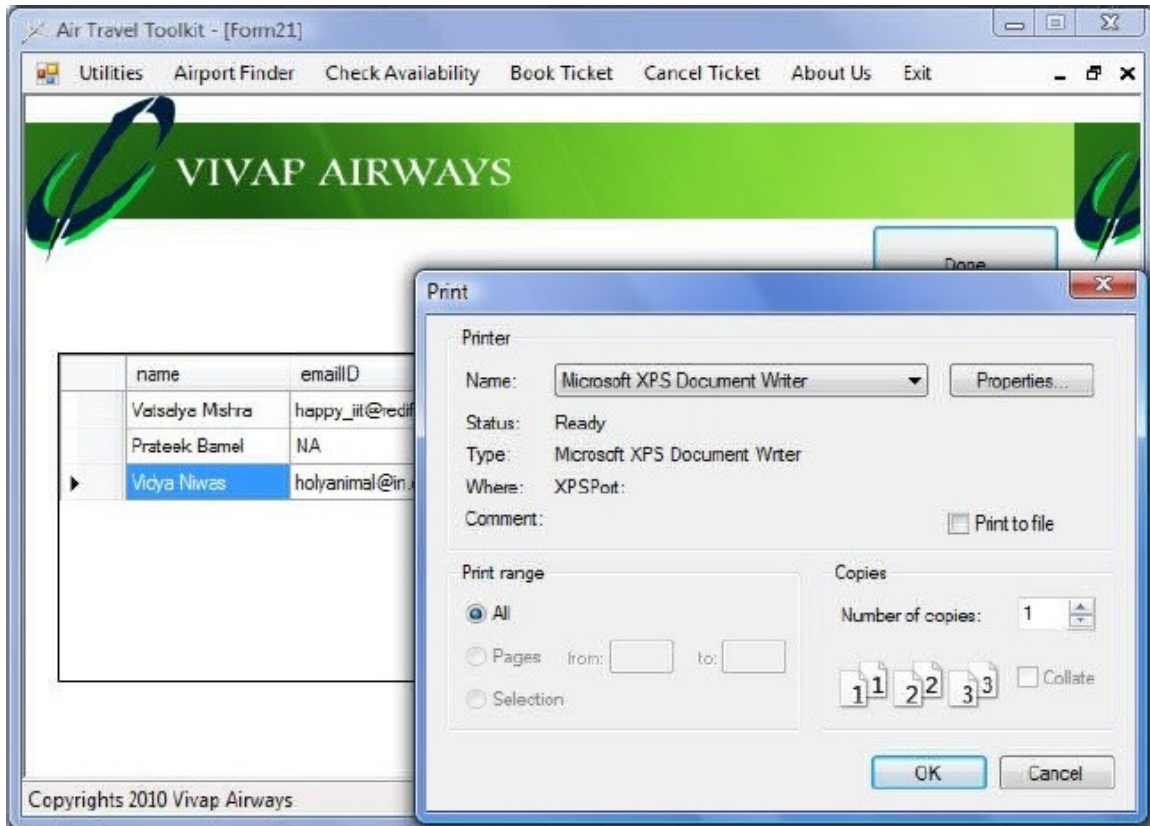
Utilities Airport Finder Check Availability Book Ticket Cancel Ticket About Us Exit

VIVAP AIRWAYS

Done

	name	emailID	contactnumber	age
▶	Vatsalya Mishra	happy_it@rediff...	9873580954	20
	Prateek Bamel	NA	NA	20
	Ankit Yadav	NA	NA	21
	Yash	NA	NA	20
	Vidya Niwas	holyanimal@in.com	9999662636	20

Copyrights 2010 Vivap Airways



7. TESTING:

7.1 Introduction:

System Development is a process of conceiving the specification specified in the designing stage into source code. Careful study and observation about system designing were made and accordingly the system was coded to convert from designing to source code, where visual Basic as the front end and OracleXE as the backend. The System was developed such that it should be used for the future enhancement.

All the module of the system is combined and is put to the operational use. This means that the new and old system are run in the parallel for sometimes, errors are identified and the corresponding errors are to be concerned to get the required output.

The set of working programs and initialized tables are also provided for the easy start of the user, in addition, system documentation is also provided, and all users have been trained to use the system.

This creates two problems,

- The time lag between the cause and appearance of the problem.
- The effect of system errors on files and records within the system.

7.2 Types of testing:

7.2.1 Unit Testing:

Unit test is designed to ensure that the purpose for which it was designed for which it was designed for is fulfilled. Each and every module was tested individually with the test data and error messages were displayed for incorrect and sufficient for entry works. All validation was tested to correctness. Test data were fed in and results were checked for the maintenance module, to ensure that all tables created contained nothing but valid data. Referential integrity constraints specified as part

of the table definition was also tested.

7.2.2 Integration Testing:

In integration testing a system consisting of different modules is tested for problems arising from component interaction. Integration testing should be developed from the system specification. Firstly, a minimum configuration must be integrated and tested. In my project I have done integration testing in a bottom up fashion i.e. in this project I have started construction and testing with atomic modules. After unit testing the modules are integrated one by one and then tested the system for problems arising from component interaction.

7.2.3 Recovery Testing:

Many computer based systems must recover from faults and resume processing within a pre-specified time. In some cases a system must be fault tolerant. ie processing faults must not cause overall system function to cease. In the casers a system failure must be corrected within a specified period of time or severe economic damage will occur.

7.2.4 Security Testing:

Any computer-based system the manages sensitive information or cause action that can improperly harm individual is a tablet for improper or illegal penetration Security testing attempts to verify that protection mechanism built into a system will, in fact protect it from improper penetration . During security testing, the tester plays the role of the individual who desire to penetrate the system. The tester may attempt to acquire passwords through external clerical means; may attack the system with custom software designed to break down any defenses that have been constructed; may overwhelm the System.

7.2.5 Performance Testing:

For real time and embedded system, software that provides required functions but not confirm to performance requirements is unacceptable. Performance testing is designed to test the run time performance of software within the context of an integrated system. Performance testing occurs throughout all steps in the testing process. Even at unit level, the performance of an individual module may be accessed as white box test recon ducted. However, it is not until all system elements are fully integrated that true performance of a system can be ascertained.

Performance Tests are sometimes coupled with stress testing and often required other hardware and software implementation. It is often necessary to measure Resource utilization .By incrementing a system the tester can uncover situations that lead to degradatation and possible system failure.

7.2.6 White Box Testing:

In white box testing knowing the internal working of the base, test can be conducted to ensure that internal operations are performed according to specification and all internal components have been adequately exercised. In white box testing logical path through the software are tested by providing test cases that exercise specific set of conditions and loops. Using white-box testing software developer can derive test case that

- Guarantee that all independent paths within a module have **been exercised at least once.**
- **Exercise all logical decisions on their true and false side.**
- **Exercise all loops at their boundaries and within their operational bound.**
- **Exercise internal data structure to ensure their validity.**

8. CONCLUSION:

The entire project has been developed and deployed as per the requirements stated by the user, it is found to be bug free as per the testing standards that are implemented. And by specification-untraced errors concentrated in the coming versions, which are planned to be developed in near future.

Finally, we like to conclude that we put all our efforts throughout the development of our project and tired to fulfill most of the **requirements of the user.**

9. REFERENCE:

Websites

- <http://www.google.com>
- <http://www.microsoft.com>
- <http://www.programmer2programmer.net>
- <http://www.codeproject.com>
- <http://www.slideshare.net>
- <http://www.1000projects.com>
- <http://www.firstload.com>

Books

- Introduction of Software Engineering, 8th Edition
- The complete reference of JavaTM 2 , 5th Edition
- SQL Bible, 2nd Edition (Paperback)
- Database Development