|  |
| --- |
|  |
| Fly Away Airline Ticket Reservation System |
| [Specification Document] |
| |  |  |  | | --- | --- | --- | | Kariyavula Gopika Manvitha | 1/24/21 | SimpliLearn – Phase 2 Project | |

**INTRODUCTION**

The Airline Ticket Booking System (ATBS) was one of the earliest changes to improve efficiency. ATBS eventually evolved into the Computer Reservations System (CRS), and then into Global Distribution System (GDS). The airline industry created the first GDS in the 1960s as a way to keep track of flight schedules, availability, and prices. Although accused of being “dinosaurs” due to their use of legacy system technology, GDSs were actually among the first e-commerce companies in the world facilitating B-2-B electronic commerce as early as the mid-1970s, when SABRE (owned by American Airline) and Apollo (United) began installing their propriety internal reservations systems in travel agencies. Prior to this, travel agents spent an inordinate amount of time manually entering reservations. The airlines realized that by automating the reservation process for travel agents, they could make the travel agents more productive and essentially turn into an extension of the airline’s sales force. It is these original, legacy GDSs that today provide the backbone to the Internet travel distribution system.

There are currently four major GDS systems:

1. Amadeus
2. Sabre
3. Galileo
4. World span

**PROBLEM DEFINITION**

In 21st century the world has become a global village where everything is available in a single click of mouse button. Aviation sector is one of fastest mode of travel available with us, both at domestic and international level. To maintain such a large system is a hectic job. The present system is very time consuming and inefficient.

The definition of our problem lies in manual system and **a** fully automated system.

**Manual system:** The system is more prone to errors and sometimes it encounters various problems which are unstructured.

**Technical system**: With the advent of latest technology if we do not update our system then our business will suffer massive losses financially. The technical system (we have proposed) contains the tools of latest trend i.e. computers printers, fax etc. The systems with this technology are very fast, accurate, user-friendly and reliable.

**Need of Airlines system**

1. A few factors that direct us to develop a new system are given below -:

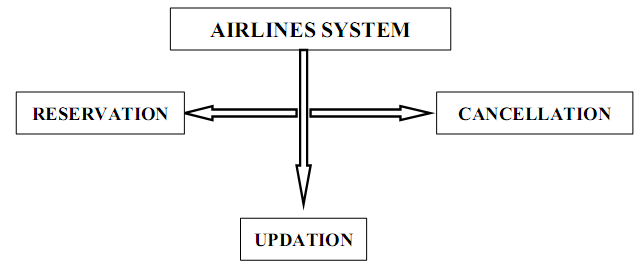
1) Faster System

2) Accuracy

3) Reliability

4) Informative

5) Reservations.



**Project Objectives**

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

* To develop a system to management of airlines, this will perform all the functions with a click of mouse button’s
* To develop a system that has good management of data along with integrity and minimizing redundancy.
* To develop a system that will be user friendly in all possible ways.
* To provide better customer support for passengers.

**TOOLS & PLATFORM USED**

We have a wide range of options of languages. From these options we can choose appropriate platform/ tools and languages for development of the project. Some of these are as follows: -

**Programming Languages:**  In programming language we have Java, JDBC, JSP, Servlet, SQL etc.

**Relational Database:** MYSQL, SQL Server, etc.

**SOFTWARE REQUIREMENTS:**

**Operating system** : Windows Family

# Front End : CSS, JSP, HTML

**Back End** : JAVA, Servlet, JDBC.

**HARDWARE SPECIFICATIONS**

**Processor** : Intel Pentium or more

# Ram : 128 MB or more

**Cache**  : 512 KB

**Hard** **disk** : 16 GB hard disk recommended

**ANALYSIS**

**DFD for Airline Reservation System**

**First Level of Data Flow Diagram for**

**System Login**

**ENTER USERNAME**

**PASSWORD**

Password Verification

User Name

Log In

**EXIT FROM**

**SYSTEM**

**VERIFIY USERNAME PASSWORD**

**SYSTEM**

**ADMINSTRATOR**

**Airline Reservation**

**System**

Verification

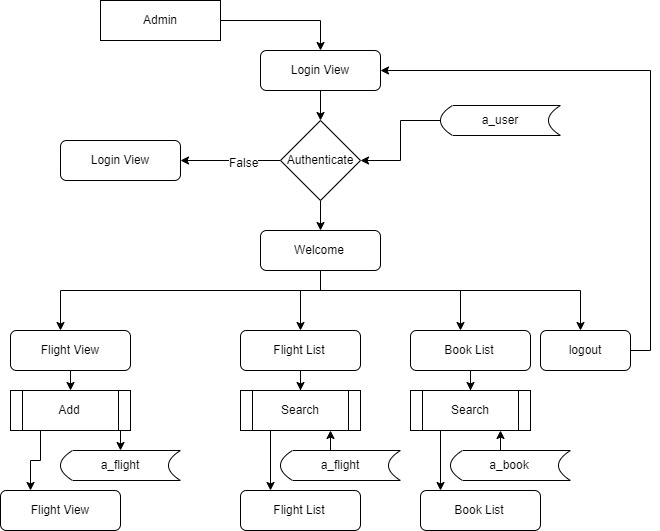
Fail

Verification

Success

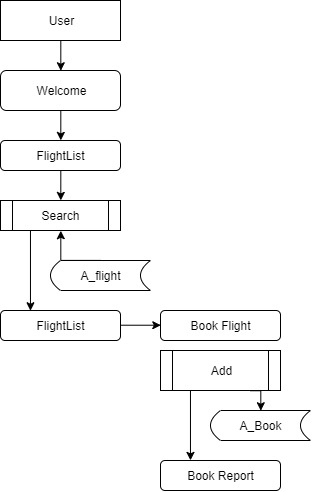
**Second Level of Data Flow Diagram for**

**General Inquiry System**

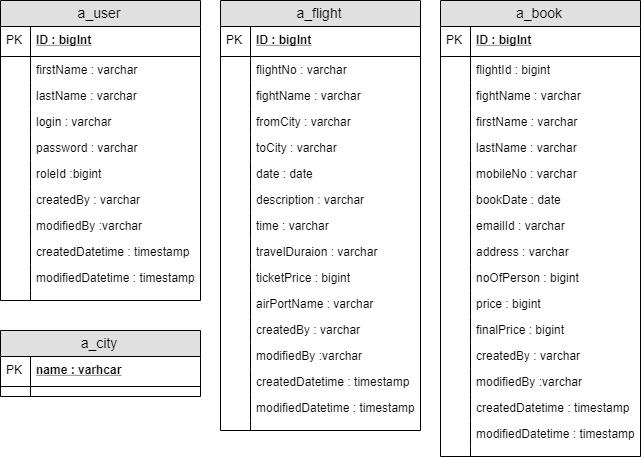
****

**Third Level DATA FLOW DIAGRAM**

**OF BOOKING SECTION**



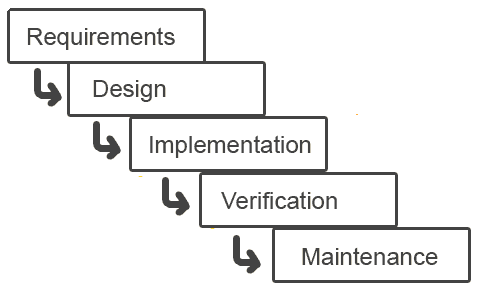
**E-R Diagram**

****

**Software Development & Management**

The field of software engineering is related to the development software in systematic manner unlike simple programs which can be developed in isolation and there may not be any systematic approach being followed. As there is large difference between programming and software engineering. As it provides models that lead to the production of well documented software in a manner that is predictable. For a mature process, it should be possible to determine in advance how much time and effort will be required to produce the final product. To develop successful software, I have to follow some models, which act as guidelines.

The model I have used is **Agile Waterfall Model or Classic Life Cycle**. In this model first of all the existed system is observed. Then customer requirements are taken in consideration then planning, modeling, construction and finally deployment.

****

**Fig.1. Waterfall Model**

**SYSTEM DESIGN**

**Introduction**

Analysis collects a great deal of unstructured data through interviews, questionnaires, on-site observations, and procedural manuals and like. It is required to organize and convert the data through system flowcharts, data flow diagrams, structured English, decision tables and the like which support future development of the system.

The Data flow diagrams and various processing logic techniques show how, where, and when data are used or changed in an information system, but these techniques do not show the definition, structure and relationships within the data.

It is a way to focus on functions rather than the physical implementation. This is analogous to the architect’s blueprint as a starting point for system design. The design is a solution, a “how to” approach, compared to analysis, a “what is” orientation.

System design is a highly creative process. This system design process is also referred as data modeling. The most common formatted used the E-R notation explains the characteristics and structure of data independent of how the data may be stored in computer memories.

The process of system design can be divided into three stages. They are:

* Structure design (already discussed)
* Database design
* Interface design

As we know that system design is a solution to “How to approach to the creation of new system”. It provides the understudying and procedural details necessary for implementing the system. The steps involved during system design were as follow: -

**LOGICAL AND PHYSICAL DESIGN**

The current physical system was thoroughly reviewed from point of view how the data flow, what are file contents, its volumes and frequency etc.

After this input, output specifications security & control specification were prepared. It was also decided that how physical information will flow through the system and a physical design walkthrough.

**OUTPUT DESIGN**

The format of outputs is designed in such a way that it is simple to read and interpret in the present output we have clearly labeled title it contains date and time and all the fields are clearly mentioned (labeled).

**INPUT DESIGN**

. Input should be as simple as possible. It is design to reduce possibility of incorrect data being enter and the need of system user are considered with this view of mind several human factors is evaluated.

**SCREEN DESIGN**

The screen design for inputting the inputs were also panned as the format of inputs.

**Sprint Planning**

The Sprint planning for the project management was done using JIRA project management tool and is been planned using Scrum project management

The Sprints have been planned of how the developer can develop with the User interface development and also with the time span of the project was managed within 3 weeks of completion

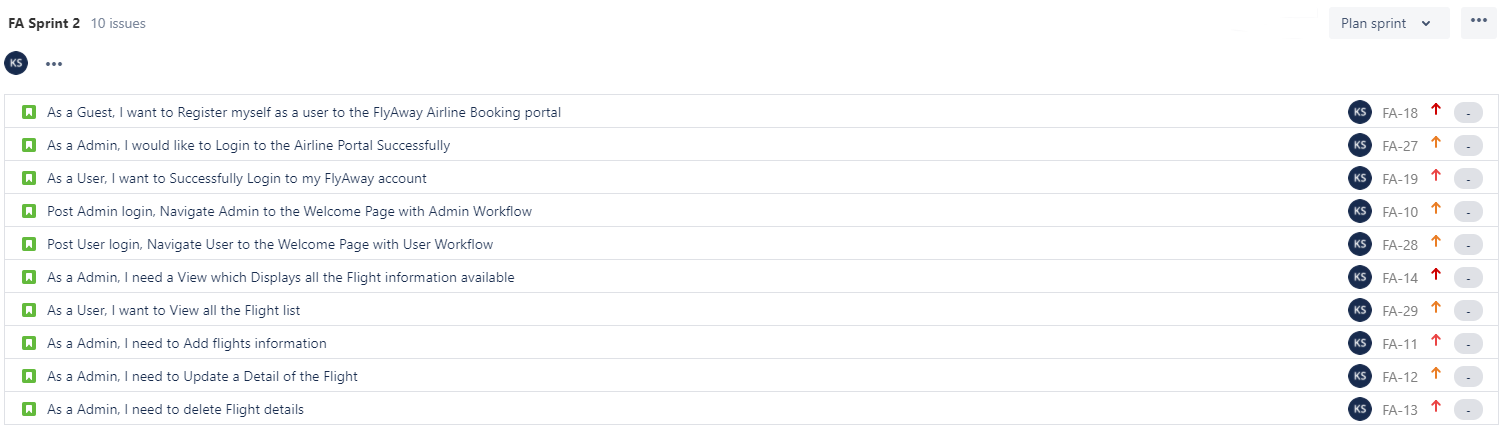
So, each sprint was planned as per the schedule of 4 days for development and one day for testing and the other day for the bug fixes

The Sprint planning of FlyAway Project images planned in JIRA is attached below

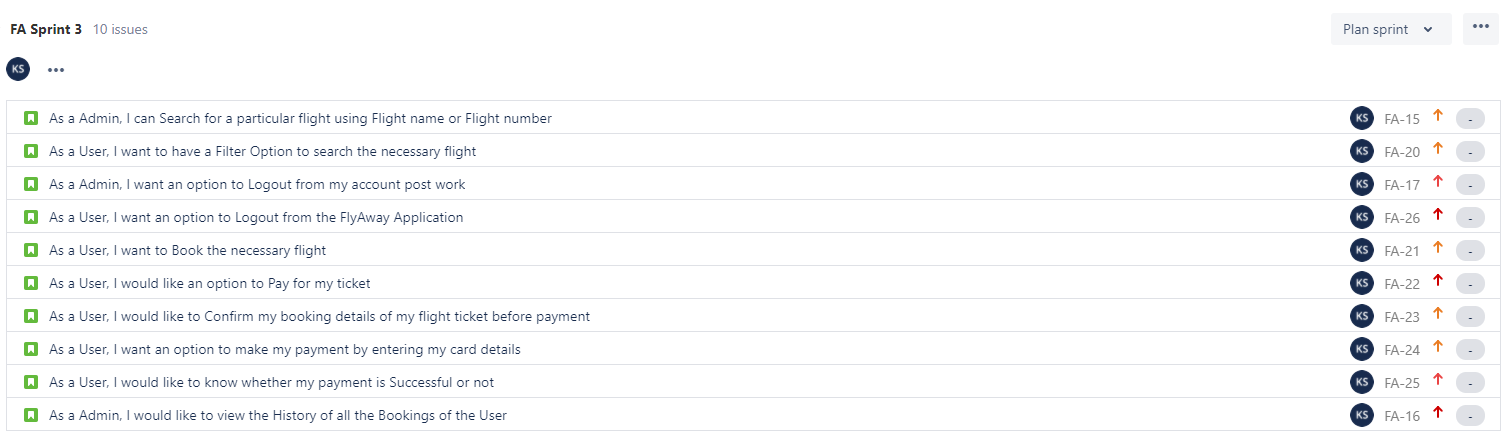
**Sprint 1**

****

**Sprint 2**

****

**Sprint 3**

****

**Sprint Planning Description**

The project has been planned in such a way that there is time to test and re-do the validations of the project workflow

Below are the following which was taken into account while planning the sprints of the project respectively:

* Sprint 1 is solely planned in the development of the UI of the workflow so that the integration with the database can be done and can be started with the workflow as well simultaneously.
* Sprint 2 is planned to work on the UI and with the backend of the project creating the user and admin portals common grounds work, so simultaneously both the user and the admin login development comes to an end at the right phase of time for project delivery.
* Sprint 3 is planned in such a way that the validation of the workflow along with the important criteria matched the workflow and the important events of the workflow are covered with the development of the project

**TESTING**

**Software Testing**

Software testing is a process of *verifying* and *validating* that a software application or program. Software testing

**1.** Meets the business and technical requirements that guided its design and development, and

**2.** Works as expected.

Software testing also identifies important *defects*, flaws, or errors in the application code that must be fixed. The modifier “important” in the previous sentence is, well, important because defects must be categorized by severity.

During test planning we decide what an important defect is by reviewing the requirements and design documents with an eye towards answering the question “Important to whom?” Generally speaking, an important defect is one that from the customer’s perspective affects the usability or functionality of the application. Using colors for a traffic lighting scheme in a desktop dashboard may be a no-brainer during requirements definition and easily implemented during development but in fact may not be entirely workable if during testing we discover that the primary business sponsor is color blind. Suddenly, it becomes an important defect. (About 8% of men and .4% of women have some form of color blindness.)

The quality assurance aspect of software development—documenting the degree to which the developers followed corporate standard processes or best practices—is not addressed in this paper because assuring quality is not a responsibility of the testing team. The testing team cannot improve quality; they can only measure it, although it can be argued that doing things like designing tests before coding begins will improve quality because the coders can then use that information while thinking about their designs and during coding and debugging.

Software testing has three main purposes: verification, validation, and defect finding.

* The *verification* process confirms that the software meets its technical specifications. A “specification” is a description of a function in terms of a measurable output value given a specific input value under specific preconditions. A simple specification may be along the line of “a SQL query retrieving data for a single account against the multi-month account-summary table must return these eight fields <list> ordered by month within 3 seconds of submission.”
* The *validation* process confirms that the software meets the business requirements. A simple example of a business requirement is “After choosing a branch office name, information about the branch’s customer account managers will appear in a new window. The window will present manager identification and summary information about each manager’s customer base: <list of data elements>.” Other requirements provide details on how the data will be summarized, formatted and displayed.
* A *defect* is a variance between the expected and actual result. The defect’s ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

## Testing methods

Software testing methods are traditionally divided into black box testing and white box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

#### Black box testing

Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

**Specification-based testing**: Specification-based testing aims to test the functionality of software according to the applicable requirements. Thus, the tester inputs data into, and only sees the output from, the test object. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case.

Specification-based testing is necessary, but it is insufficient to guard against certain risks.

**Advantages and disadvantages**: The black box tester has no "bonds" with the code, and a tester's perception is very simple: a code *must* have bugs. Using the principle, "Ask and you shall receive," black box testers find bugs where programmers do not. *But,* on the other hand, black box testing has been said to be "like a walk in a dark labyrinth without a flashlight," because the tester doesn't know how the software being tested was actually constructed. As a result, there are situations when (1) a tester writes many test cases to check something that could have been tested by only one test case, and/or (2) some parts of the back-end are not tested at all.

Therefore, black box testing has the advantage of "an unaffiliated opinion," on the one hand, and the disadvantage of "blind exploring," on the other.

#### White box testing

White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

Types of white box testing

* API testing (application programming interface) - Testing of the application using Public and Private APIs
* Code coverage - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
* Fault injection methods - improving the coverage of a test by introducing faults to test code paths
* Mutation testing methods
* Static testing - White box testing includes all static testing

### FlyAway’s testing cycle

Although variations exist between organizations, there is a typical cycle for testing:

* **Requirements analysis**: Testing should begin in the requirements phase of the software development life cycle. During the design phase, testers work with developers in determining what aspects of a design are testable and with what parameters those tests work.
* **Test planning**: Test strategy, test plan, tested creation. Since many activities will be carried out during testing, a plan is needed.
* **Test development**: Test procedures, test scenarios, test cases, test datasets, test scripts to use in testing software.
* **Test execution**: Testers execute the software based on the plans and tests and report any errors found to the development team.
* **Test reporting**: Once testing is completed; testers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.
* **Test result analysis**: Or Defect Analysis, is done by the development team usually along with the client, in order to decide what defects should be treated, fixed, rejected (i.e. found software working properly) or deferred to be dealt with later.
* **Defect Retesting**: Once a defect has been dealt with by the development team, it is retested by the testing team.
* **Regression testing**: It is common to have a small test program built of a subset of tests, for each integration of new, modified, or fixed software, in order to ensure that the latest delivery has not ruined anything, and that the software product as a whole is still working correctly.
* **Test Closure**: Once the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects.

**\*\*\*\*\* Thank you \*\*\*\*\***