LOW LEVEL DOCUMENT

FLIGHT FARE ESTIMATOR

Written By	Suraj Joshi
Document Version	1.0
Last Revised Data	20 Jan 2022

LOW LEVEL DESIGN

1 Document Control:

Version	Date	Author	Comments
1.0	20/01/2022	Suraj Joshi	

2 Reviews:

Version	Date	Reviewer	Comments
1.0	20/01/2022	Suraj Joshi	

3 Approval Status:

Version	Review Date	Reviewed By	Approved By	Comments
1.0	20/01/2022	Suraj Joshi	Suraj Joshi	

LOW LEVEL DESIGN

Table of Contents

1.Introduction

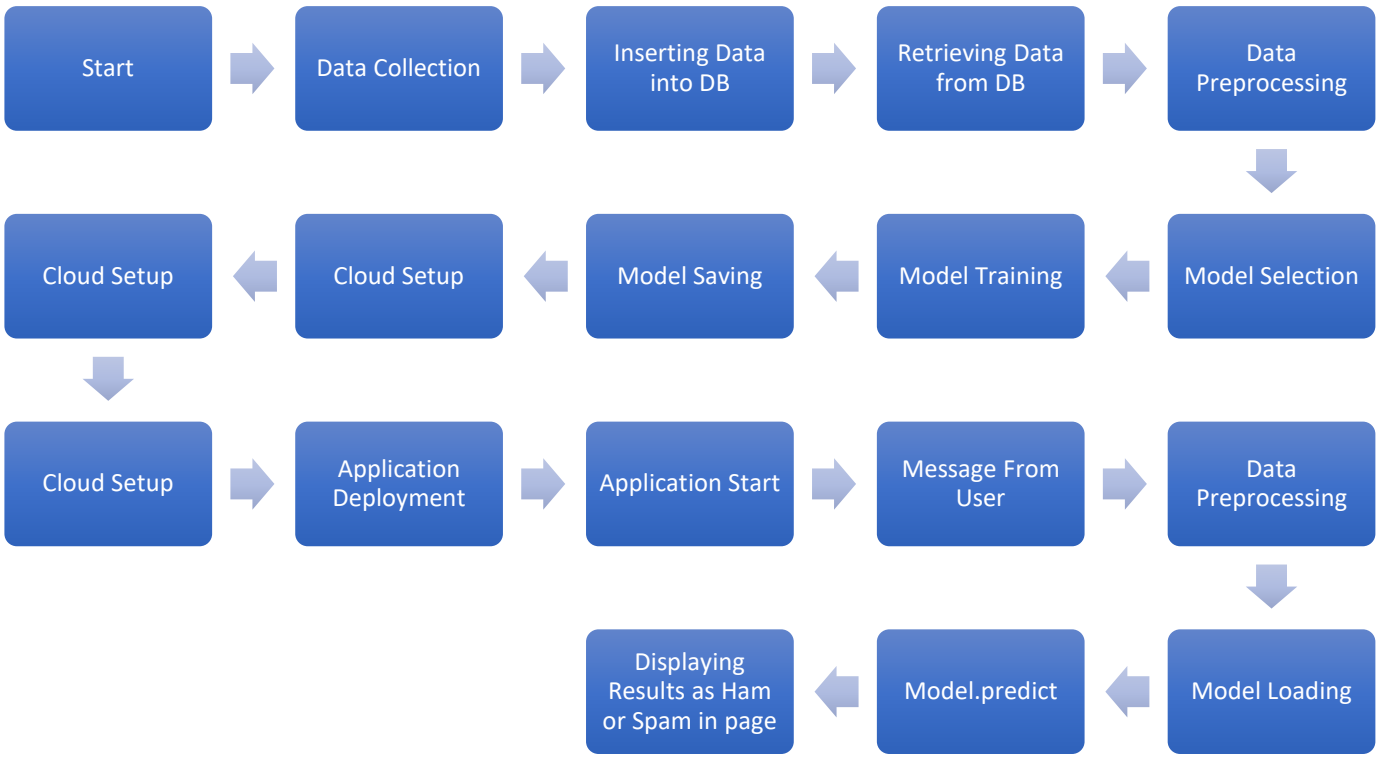
1.1 What is Low-Level design document?

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for flight fare estimation System. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

1.2. Scope

Low-level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

2. Architecture



3. Architecture Description

3.1 Data Collection

A collection of 425 SMS spam messages was manually extracted from the Grumbletext Web site A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. A list of 450 SMS ham messages collected from Caroline Tag's PhD Thesis Available. A list of 450 SMS ham messages collected from Caroline Tag's PhD Thesis available

These is given in the comma separated value format (.csv). These data is collected from the UCI which contains both the test data and train data.

3.2 Data Validation

In Data Validation part we are simply accepting the data provided from the UCI.

3.3 Inserting into DB

We are going to insert the data of all 4 columns into Database that is CassandraDB .For this we have used Cassandra driver Our Database name is Spam-Ham and keyspace is training.

3.4 Retrieving Data From DB

Retrieving the data and storing into a file called as Training_file.csv

3.5 Data Preprocessing

Data Preprocessing step is very necessary for model creation. All we have is 4 columns in which last two columns are useless so we removed them. First column is telling us about whether the message is spam or ham while the second is just the message.
There are lots of things we have to do in data preprocessing such as
1)Converting the message into lower case.
2)Avoiding punctuation character and stopwords which might be present in the messages.
3)Now the final part comes which is vectorization , we are achieving this using tf-idf vectorizer provided by sklearn.

3.6 Model Training

Model Selection is performed by file model_building.py which is inside Model_Building. For model Training we are just choosing Multinomial Naive Bayes because it is performing best than other models When calculating accuracy MNB always had a very high accuracy compared to others.

3.8 Model Saving

Model Saving is the final step in the training part. We are saving the model in folder as model.pickle and I am using module pickle for saving the ML model.

3.9 Data from User

On Application Starting user will be interacting with a UI which is designed using HTML/CSS. The webpage is actually very simple , User just needs to put message in the box and click predict button.

3.11 Data Preprocessing

In data preprocessing I will be handling categorical columns plus some other columns as well as like Duration,Arrival_Time column.

3.12 Model Loading

Model loading is actually very simple we will be calling the model which we saved as modle.pickle .For loading the model I am using pickle
library.pickle.load(open('modle.pickle')).

3.13 Model.predict

After loading the model everything is very simple you just have to predict the pre-processed data.

3.14 Displaying the results

The message type will be displayed in the webpage only. Like
No it is not a spam message
Or ,
Yes it is a spam message.

4 Test Cases

Test cases are given below

Test Case Description	Pre-Requisite	Expected Result
Verify whether the Application URL is accessible to the user	1. Application URL should be defined	Application URL should be accessible to the user
Verify whether the Application loads completely for the user when the URL is accessed	1. Application URL is accessible 2. Application is deployed	The Application should load completely for the user when the URL is accessed
Verify Response time of url from backend model.	1. Application is accessible	The latency and accessibility of application is very faster we got in heroku service.
Verify whether user is giving standard input.	1. Handeled test cases at backends.	User should be able to see successfully valid results.
Verify whether user is able to see input fields on logging in	1. Application is accessible 2. User is logged in to the application	User should be able to see input fields on logging in
Verify whether user is able to edit all input fields	1. Application is accessible 2. User is logged in to the application	User should be able to edit all input fields
Verify whether user gets Predict button to submit the message.	1. Application is accessible 2. User is logged in to the application	User should get predict button to submit the message.

LOW LEVEL DESIGN

Verify whether user is presented with recommended results on clicking predict button.	1. Application is accessible 2. User is logged in to the application	User should be presented with recommended results on clicking predict.
Verify whether the recommended results are in accordance to the selections user made	1. Application is accessible 2. User is logged in to the application and database	The recommended results should be in accordance to the selections user made