LOAN DEFAULTER PREDICTION SYSTEM

APPLICATION DEVELOPMENT – II
SOFTWARE DESIGN SPECIFICATION

Submitted in partial fulfilment for the award of the degree
Of

Master of Technology
In
Information Technology

By

MAYANK KUMAR
(15MIN0018)

School of Information Technology and Engineering
March, 2018

Table of Contents

1 INTRODUCTION

Nowadays there are many risks related to bank loans, especially for the banks so as to reduce their capital loss. The analysis of risks and assessment of default becomes crucial thereafter. Banks hold huge volumes of customer behaviour related data from which they are unable to arrive at a judgement if an applicant can be defaulter or not. Data Mining is a promising area of data analysis which aims to extract useful knowledge from tremendous amount of complex data sets. This work aims to develop a model and construct a prototype for the same using a data set available in the repository. The model is a decision tree based classification application that uses the functions available in the IBM Watson Package. Prior to building the application, the dataset is pre-processed, reduced and made ready to provide efficient predictions. The final application is used for prediction with the test dataset and the experimental results prove the efficiency of the built application.

1.1 PURPOSE

The main purpose of **LOAN DEFAULTER PREDICTION SYESTEM** is to arrive at a judgement if a loan applicant can be defaulter or not. The final application is used for analysis with the test datasets and the experimental results prove the efficiency of the built application.

1.2 SCOPE

This project designs and implements LDPS to fulfill all the vision statements supported by a well-defined datasets, all available datasets is pre-processed and can be accessed easily through a single point. A friendly user interface is provided so that each customers can be easily analyzed by the bank representatives. LDPS provided banks, an interfaces that will be used for loan defaulter prediction in advanced. If time permits, LDPS will also support bulk customer defaulter prediction.

1.3 OVERVIEW

LDPS is an Internet-based application executing on IBM Bluemix platform and connected to an Enterprise database. LDPS accepts and processes requests from various banks representatives: financier and bank administrators. Besides the local server database (for storing/reservation records), LDPS also integrates databases from various banks. The system is expected to have a Web user interface for loan financier and for banks administrators. Its final release has merits of being efficient and precise in determining the defaulter customer.

1.4 REFERENCE MATERIAL

- IEEE STD 1058-1998, IEEE Standard for Software Project Management Plans
- IEEE STD 830-1998, IEEE Recommended Practice for Software Requirements Specifications
- IEEE STD 1016-1998, Recommended Practice for Software Design Descriptions
- Pressman, Roger S., Software Engineering "A Practitioner's Approach", Fifth Edition, McGraw-Hill, 2000

1.5 DEFINATIONS and ACRONYMS

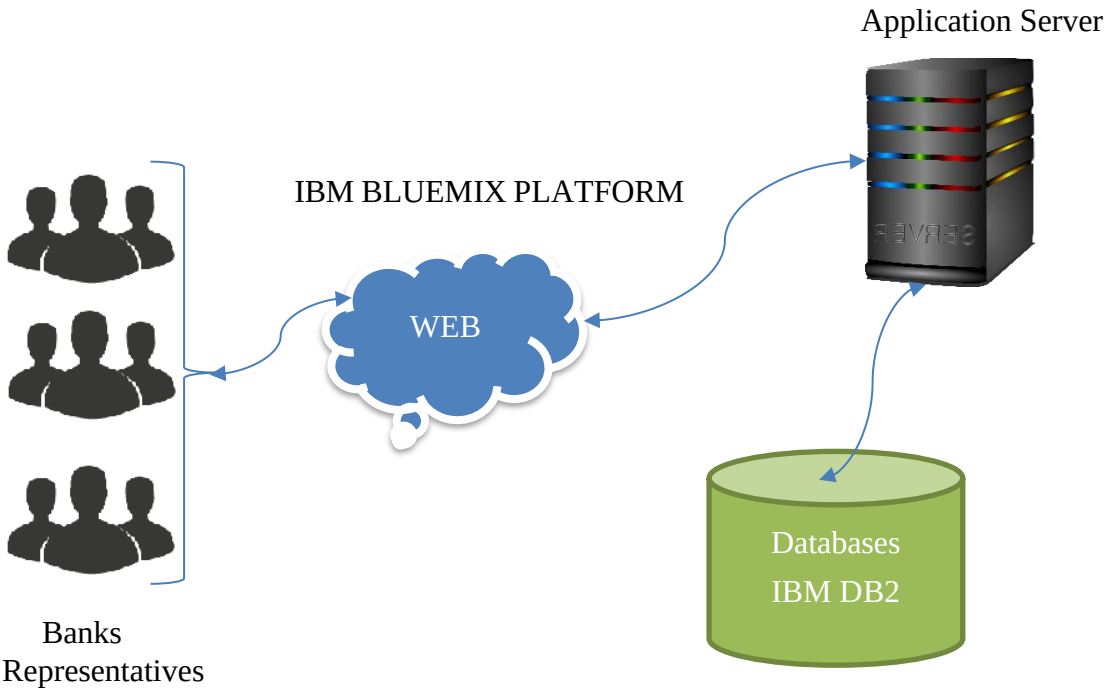
ACQUIRES	Specifies requirements for and accepts delivery of a new or modified software product and its documentation.
ADMINISTRATOR	The one who manages and maintains computer systems and software.
CALENDAR	It is a tool which user enters monthly program
LDPS	LOAN DEFAULTER PREDICTION SYSTEM
FAQ	Frequently Asked Questions.
IEEE	Institute of Electrics & Electronics Engineering
IS	Information Systems
IT	Information Technology

SRS	Software Requirements Specification
TEAM	The name of the developer group
USER	People who open the LDPS web site and the administrator of the LDPS web site.
WEB	the network of computers that forms the Internet

2 SYSTEM OVERVIEW

2.1 INTRODUCTION

LDPS is an Internet-based application executing on IBM Bluemix platform and connected to an Enterprise database. LDPS accepts and processes requests from various banks representatives: financier and bank administrators. Besides the local server database (for storing/reservation records), LDPS also integrates databases from various banks. The system is expected to have a Web user interface for loan financier and for banks administrators. Its final release has merits of being efficient and precise in determining the defaulter customer.



System Architecture Diagram

2.2 REQUIREMENT ANALYSIS

Requirements are prone to issues of the ambiguity, incompleteness and inconsistency techniques such as rigorous inspection have been shown to help deal with these issues. Ambiguity, incompleteness and inconsistencies that can be resolved in the requirement phase typically cost orders of the magnitude less to correct than when these same issues are found in later stages of product development. The purpose of developing the specified software is to describe the analysis involved in the prediction of the bank loan defaulters.

2.3 FUNCTIONAL ANALYSIS

Input: Collecting the information about the customer who is going to applied for loan.
Output: Predicting the chances of a given customer, being a defaulter.

2.4 PROCESS

- Get all the details from the customer.
- Ask his/her decision on how much loan he/she wants.
- Store all these details into the database.
- Check for customer, whether an existing customer or a new customer.
- Perform prediction analysis on the customer data.
- Get the analysis reports.
- Decide whether to sanction a loan to the customer or not, based on the prediction results.

2.5 EXISTING SYSTEM

In the existing system there is no provision to perform prediction analysis for bulk customers.

2.6 PROPOSED SYSTEM

- The client tier must not be changed, which means that the format of all the communication messages have to be preserved.
- Some functionality, like check digit validation, time, stamps etc. are supplied by already existing routines which we are obliged to use.
- The format of communication in modules are fixed and non-changeable.
- All the technical documentation formats are also fixed and have to be followed.
- Some customer implementation techniques have to be followed.
- A facility for viewing the single customer record is made available.
- We have made provision for bulk prediction analysis.

2.7 REPORT MODULE

The prediction results should have the details about the chances of the customer being defaulter or non-defaulter in terms of percentage. The customer should be informed about his/her credit score rating and the prediction analysis results .The names of the fields involved in the LOAN DEFAULTER PREDICTION system are

- Customer Id
- Gender
- Age
- Criminal Record
- Is Senior Citizen
- Marital Status
- Is Joint Loan
- Is Joint Applicant Earning
- Number of Dependants
- Number of years of relationship with bank
- Credit Score
- Profession
- Has user defaulted any time before?
- Existing Loan
- Loan Applying For

- Annual Income
- Is Insured

2.8 MODULES

2.8.1 Individual Processing

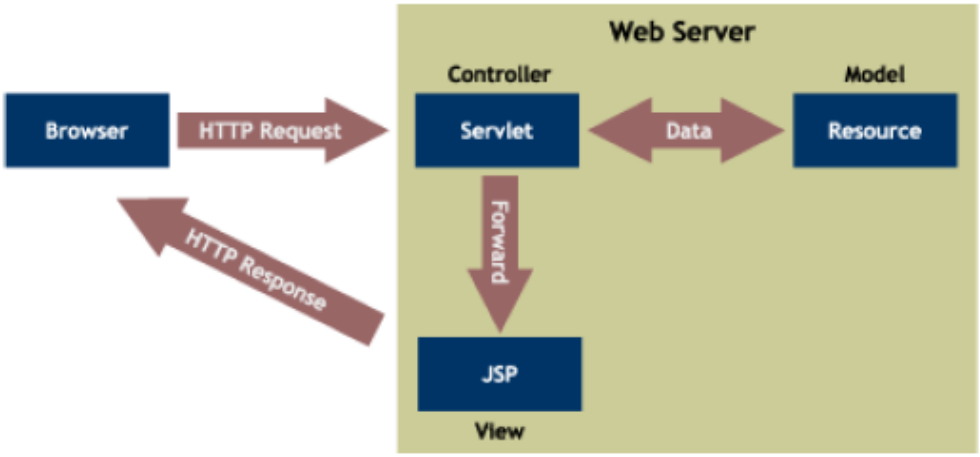
This function allows the bank representatives to perform prediction analysis for each single customer based on whether a customer is an existing customer or a new customer. In order to determining a probability of customer being a defaulter or non-defaulter, the bank representative has to provide required information such as Customer Id, Gender, Age, Criminal Record, Is Senior Citizen, Marital Status, Is Joint Loan, Is Joint Applicant Earning, Number of Dependents, Number of years of relationship with bank, Credit Score, Profession, Has user defaulted any time before?, Existing Loan, Loan Applying For, Annual Income, Is Insured during the analysis process.

2.8.2 Bulk Processing

This function allows the bank representatives to perform prediction analysis for all the existing customers in a single go and provide them the analysis data which they can used to determine what customers are eligible for loan request from the bank.

3 SYSTEM ARCHITECTURE

3.1 ARCHITECTURAL DESIGN



Architecture Design of LDPS

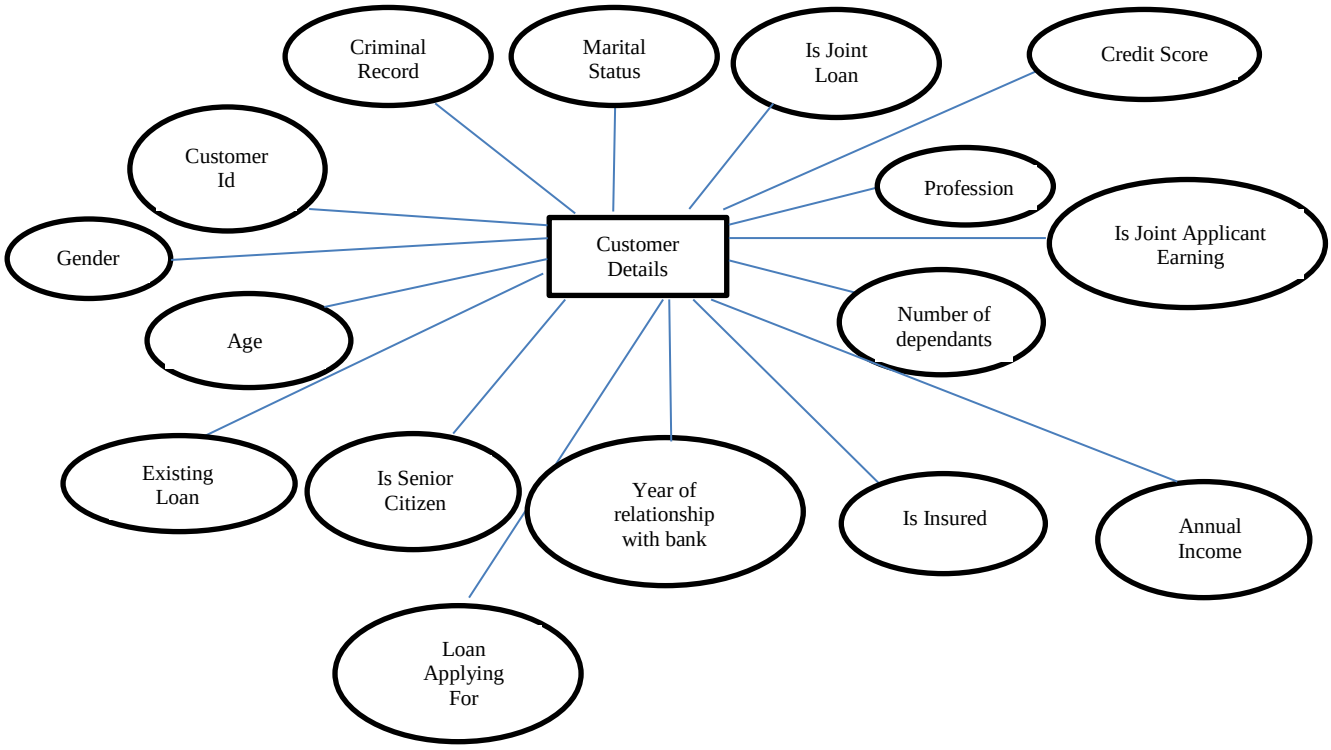
3.2 DECOMPOSITION DESIGN

3.2.1 Entity Relationship Diagram

The object relationship pair can be graphically represented by a diagram called Entity. Relationship Diagram. It is mainly used in database applications but now it is more commonly used in data design. The primary purpose of ERD is to represent the relationship between data objects.

Various components of ERD are:

1. Entity
2. Relationship
3. Attribute.



Entities:

- Customer Details

Attributes of entities:

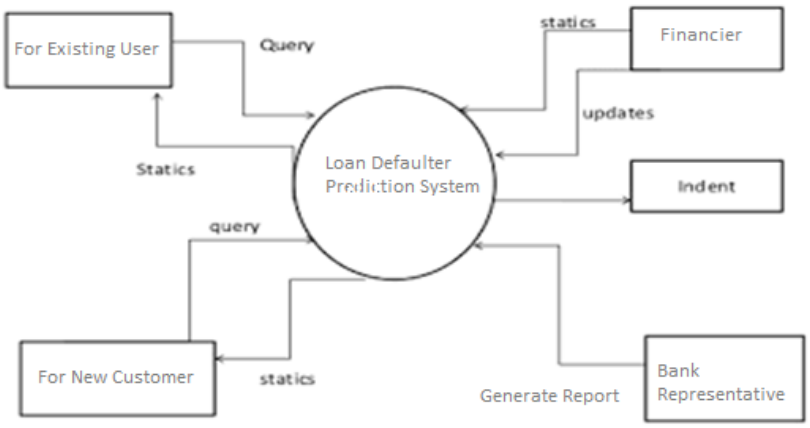
- Customer Details
- Customer Id
- Gender
- Age
- Criminal Record
- Is Senior Citizen
- Marital Status
- Is Joint Loan
- Is Joint Applicant Earning

- Number of Dependents
- Number of years of relationship with bank
- Credit Score
- Profession
- Has user defaulted any time before?
- Existing Loan
- Loan Applying For
- Annual Income
- Is Insured

3.2.2 Data Flow Diagram

Data Flow Diagram is one of the Functional Model which are used to represent the flow of information in any computer based system. The data flow diagram depicts the information flow and the transforms that are applied on the data as it moves from input to output.

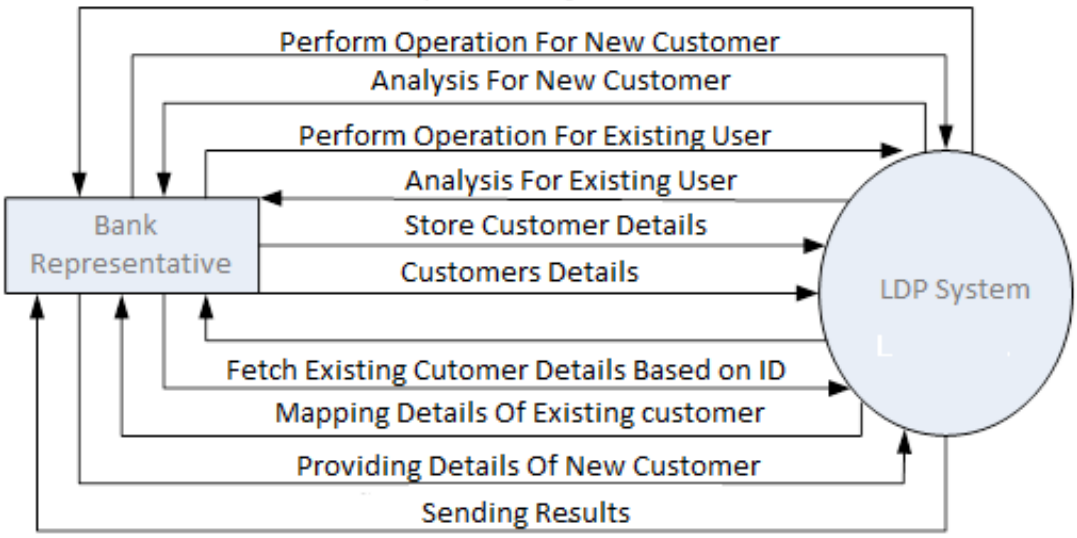
LEVEL 0: Initially in the first level of the Data flow the level 0 explains the basic outline of the system. The end-user sends the packets to the system to determine the source and destination address. The diagram marked as the 0 represents the complete Packet watching system which simply represents the basic operation that is being performed by it in the initial level.



Level 0 DFD

3.3 CONTEXT DIAGRAM

The Context Diagram shows the system under consideration as a single high-level process and then shows the relationship that the system has with other external entities (systems, organizational groups, external data stores, etc.). Another name for a Context Diagram is a Context-Level Data-Flow Diagram or a Level-0 Data Flow Diagram. Since a Context Diagram is a specialized version of Data-Flow Diagram, understanding a bit about Data-Flow Diagrams can be helpful.



3.4 ACTIVITY DIAGRAM

Activity diagram models the logic from workflow to use cases to methods. It borrows most of the notations from the flowchart but has added the concept of concurrency to support many modern applications. The arrow traces the flow from beginning to end through decision and loops, while identifying each logic steps in the process.

Activity diagrams are typically used for business process modelling, for modelling the logic captured by a single use case or usage scenario, or for modelling the detailed logic of a business rule. Although UML activity diagrams could potentially model the internal logic of a complex operation it would be far better to simply rewrite the operation so that it is simple enough that you don't require an activity diagram. In many ways UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structured development.

The easiest way to visualize an Activity diagram is to think of a flowchart of a code. The flowchart is used to depict the business logic flow and the events that cause decisions and actions in the code to take place.

Activity diagrams represent the business and operational workflows of a system. An Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state.

So, what is the importance of an Activity diagram, as opposed to a State diagram? A State diagram shows the different states an object is in during the lifecycle of its existence in the system, and the transitions in the states of the objects. These transitions depict the activities causing these transitions, shown by arrows.

An Activity diagram talks more about these transitions and activities causing the changes in the object states.

Activity diagrams are useful because:

1. They represent the logic required to implement system behaviours.
2. They are simple enough to learn quickly.
3. Represents the logic at any level the design needs, from system workflow to individual method implementation.

3.5 USE CASE DIAGRAMS

Use case diagrams are helpful in three areas.

- Determining features (requirements). New use cases often generate new requirements as the system is analysed and the design takes shape.
- Communicating with clients. Their notational simplicity makes use case diagrams a good way for developers to communicate with clients.
- Generating test cases. The collection of Diagrams for a use case may suggest a suite of test cases for those Diagrams.

The various use-case Diagrams for the system are as shown in below diagram.

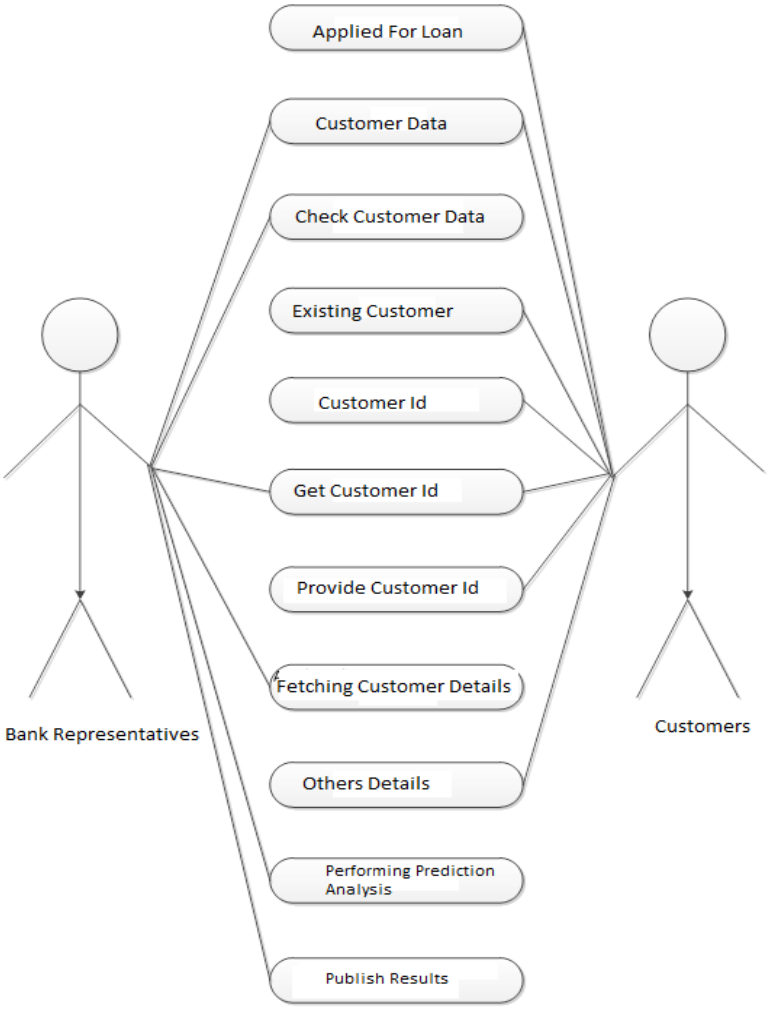
Actor in Use-Case Diagram:

- Bank Representatives - User in the system who perform the defaulter prediction analysis.
- Customer – For which system will perform the prediction analysis.

- LDP System – System that perform the prediction analysis.

Use-case in the system:

- Access -Allow bank representatives to access the prediction application.
- Fetching Existing customer Details – Bank representatives will fetch existing customer details based on customer Id.
- Perform Analysis For Existing User - Bank representatives will perform the prediction analysis for the existing user.
- Publish Results – Decide whether the customer is eligible for the loan.
- Enter New Customer Details – Bank representatives will provide the details for new customer to the LDP system.
- Perform Analysis For New User – Next the bank representatives will perform the prediction analysis for with new customer details.
- Publish Results – Decide whether the customer is eligible for the loan.



Use Case Diagram for Loan Defaulter Prediction System

Use Case Description

Actors:

1. Bank Representatives/Customer

Use cases:

1. Applied For Loan
2. BR will ask for customer data
3. Customer will provide the data
4. BR check the customer data
5. Customer is an existing customer
6. BR get customer id
7. BR fetch the customer details based on id
8. Fetch other details as well
9. Perform Prediction Analysis
10. Publish results.

3.6 UML Interaction Diagram (Sequence and Collaboration Diagram)

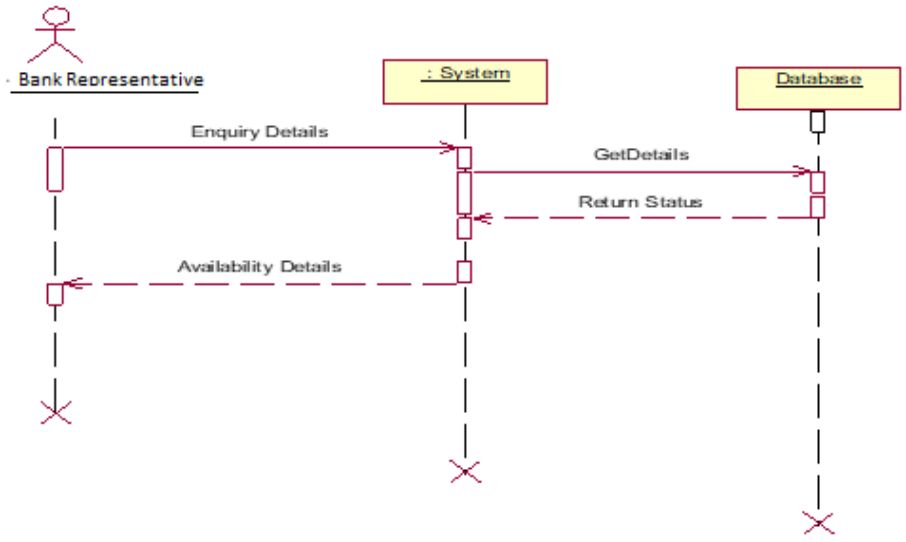
Interaction diagrams describe the communication between objects to accomplish some task such as placing an order. In UML the two types of interaction diagrams are sequence diagram and collaboration diagram. These diagrams model the dynamic aspects of the system.

3.6.1 SEQUENCE DIAGRAMS

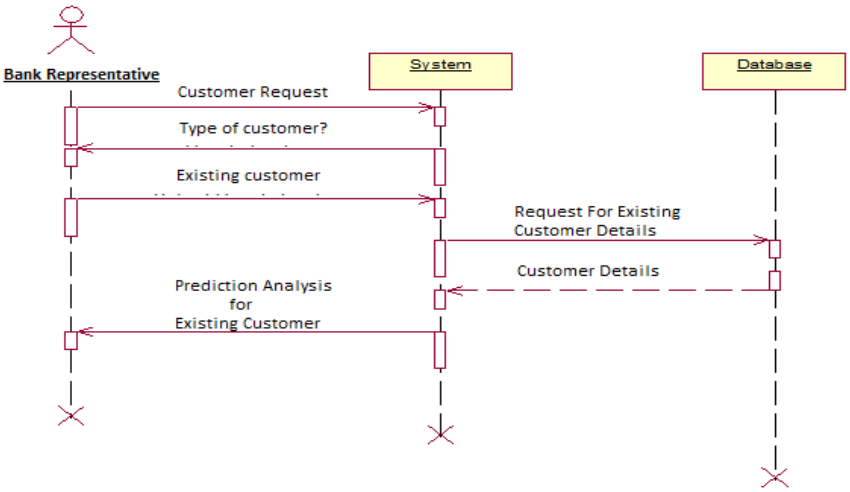
Sequence diagram is one kind of interaction diagrams, which shows an interaction among a set of objects and their relationships. The purpose of the Sequence diagram is to document the sequence of messages among objects in a time based view. The scope of a typical sequence diagram includes all the message interactions for a single use case.

Sequence Diagrams are valuable because:

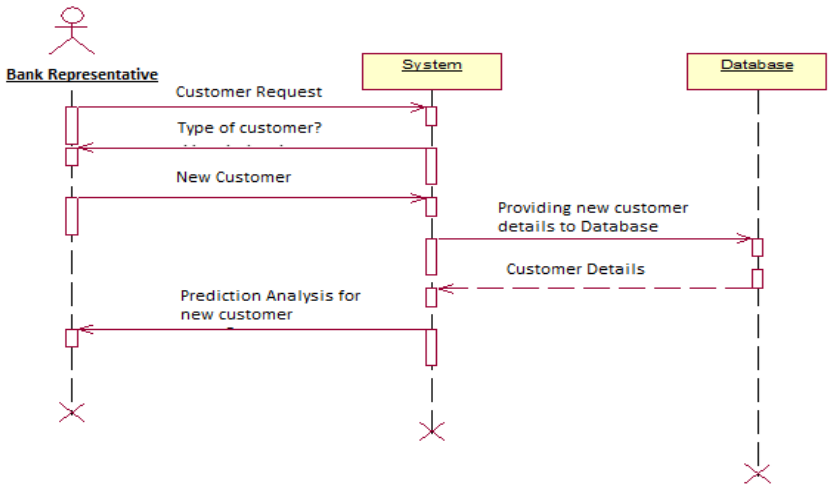
1. They have a narrow focus that helps us to see the specific questions, commands and data communicated during the execution of a specific task.
2. They explicitly identify the communication required to fulfil an interaction. They help us to identify the interfaces required by the classes.
3. They identify the objects that take part in the interaction. They help us to validate the features of a class.
4. They identify the data passed as part of the interaction.



Sequence Diagram to check Availability



Sequence Diagram for Existing Customer



Sequence Diagram for New Customer

3.6.2 COLLABORATION DIAGRAMS

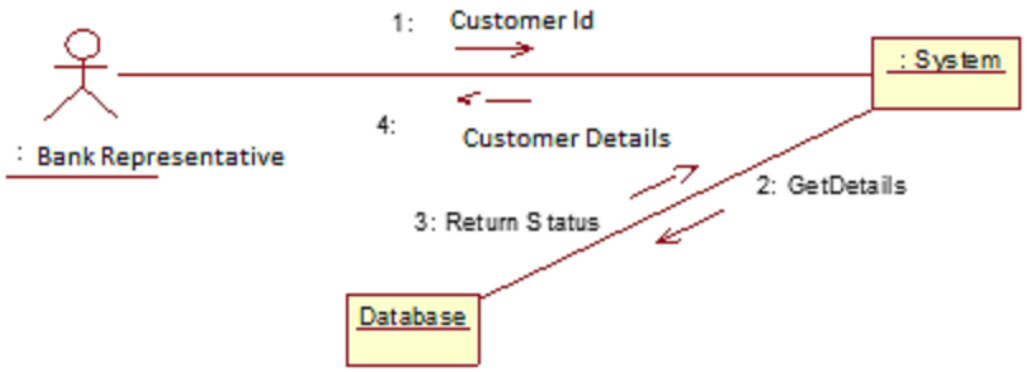
Collaboration diagram is very similar to a Sequence diagram in the purpose it achieves; in other words, it shows the dynamic interaction of the objects in a system. A distinguishing feature of a Collaboration diagram is that it shows the objects and

their association with other objects in the system apart from how they interact with each other. The association between objects is not represented in a Sequence diagram.

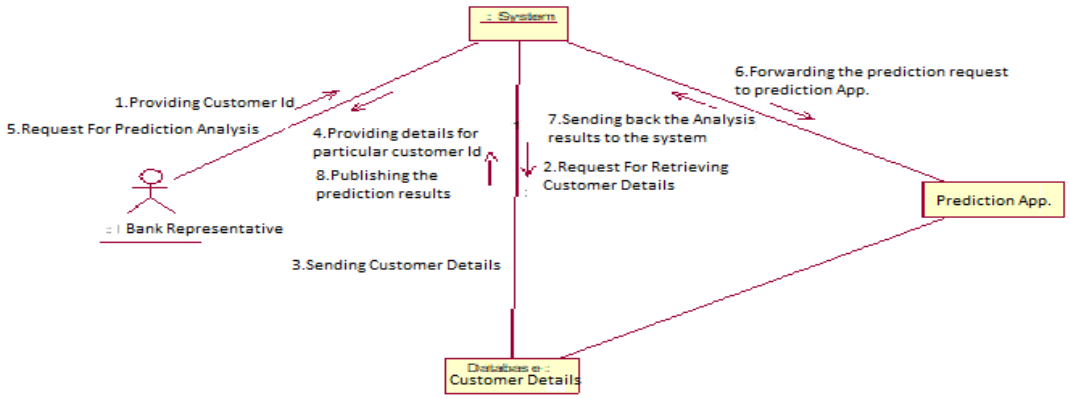
A Collaboration diagram is easily represented by modelling objects in a system and representing the associations between the objects as links. The interaction between the objects is denoted by arrows. To identify the sequence of invocation of these objects, a number is placed next to each of these arrows.

Collaboration diagrams are valuable because:

1. Shows the structural requirement for completing a task. They identify the objects that participate in an interaction.
2. Shows the interface requirement for a particular class.
3. Identify the data that is passed as a part of the interaction.



Collaboration Diagram to check Availability



Collaboration Diagram for Prediction Analysis

3.7 STATE TRANSITION DIAGRAM

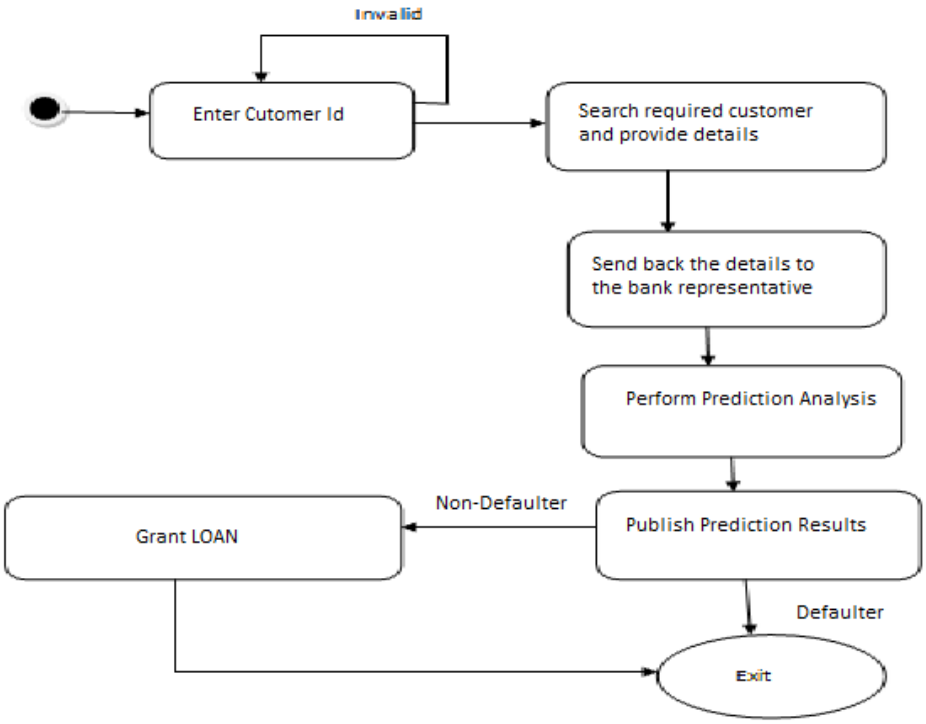
The state transition diagram represents a single object. It shows how external factors cause changes in the object over its lifetime. It captures the behaviour of a software system. They provide an excellent way of modelling communications that occurs with external entities via a protocol or event. State diagrams are used to give an abstract description of the behaviour of a system. This behaviour is analysed and represented in series of events that could occur in one or more possible states. Hereby "each diagram usually represents objects of a single class and tracks the different states of its objects through the system". The following are the basic notational elements that can be used to make up a diagram:

1. Filled circle, pointing to the initial state
2. Hollow circle containing a smaller filled circle, indicating the final state (if any)

3. Rounded rectangle, denoting a state. Top of the rectangle contains a name of the state. Can contain a horizontal line in the middle, below which the activities that are done in that state are indicated
4. Arrow, denoting transition.
5. Thick horizontal line with either x>1 lines entering and 1 line leaving or 1 line entering and x>1 lines leaving. These denote join/fork, respectively.

State Transition diagrams are valuable because:

1. Identify the specific responses of an object to everything that can happen to the object.
2. Identifies what events an object will and will not respond.
3. Validate the data needed to define the state of the object and the attributes affected by the change.
4. Helps in finding the internal effects of behaviour that cannot be seen using interaction based diagrams.



State transition diagram for Loan Defaulter Prediction System.

3.8 DESIGN RATIONALS

The algorithm is developed as flow chart and the data flow diagrams, to describe the step-wise procedure of the application. The basic requirements, which are got from the customer, should all be covered in this algorithm developed. Most components described in the system architecture section will require a more detailed discussion. Other lower-level components may need to be described as well. The kind of component, such as a sub system like delete, insert, module like student detail, class like library, package, function, file etc. The specific purpose and semantic meaning of the component describe this. This may need to refer back to the requirement specification.

4 DATA DESIGN

This section describes the category of data required by the system. Because there is no actual complete data set available for use we will produce the needed data synthetically. This data will be more formally represented in our entity relational design data model.

4.1 DATA DESCIPTION

- The information about several customers, more specifically:
    - Customer Id
    - Gender
    - Age
    - Criminal Record
    - Is Senior Citizen
    - Marital Status
    - Is Joint Loan
    - Is Joint Applicant Earning
    - Number of Dependents
    - Number of years of relationship with bank
    - Credit Score
    - Profession
    - Has user defaulted any time before?
    - Existing Loan
    - Loan Applying For
    - Annual Income
    - Is Insured

4.2 DATA DICTIONARY

A data dictionary is a catalogue – a repository – of the elements in a system. These elements centre on data the way they are structured to meet user requirements and organization needs. In a data dictionary, a list of all the elements composing the data flowing through a system is included. If a project team member wants to know the

definition of a data item name or the contents of a particular data flow, the information will be available in the data dictionary. Descriptions of all data used in the system are given in a data dictionary.

Analysts uses Data Dictionary for below important reasons

- To manage the detail in large systems.
- To communicate a common meaning for all system elements.
- To document the features of the system.
- To facilitate 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.

DATA DICTIONARY for LDPS:

1) Table Name: Customer Details
   Description: This table stores customer datasets
   Primary Key: customer Id

Sr. No.	Field Name	Data Type	Size	Constraint
1	Customer Id	Number	(5)	Not Null, PK
2	Gender	Varchar2	(7)	Not Null
3	Age	Number	(5)	
4	Criminal Record	Char	(1)	Not Null
5	Is Senior Citizen	Char	(1)	Not Null
6	Marital Status	Char	(1)	Not Null
7	Is Joint Loan	Char	(1)	Not Null
8	Is Joint Applicant Earning	Char	(1)	Not Null

9	Number Of Dependents	Number	(4)	
10	Years Of Relationship With Bank	Number	(4)	
11	Credit Score	Number	(10)	
12	Profession	VARCHAR	(40)	Not Null
13	Has User Defaulted Any Time Before	Char	(1)	
14	Existing Loan	Number	(40,8)	
15	Loan Applying For	Number	(40,8)	
16	Annual Income	Number	(40,8)	
17	Is Insured	Char	(1)	

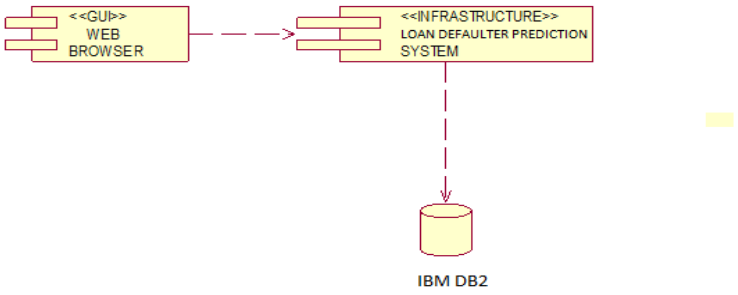
5 COMPONENT DESIGN

The component diagram represents pieces of software in the implementation environment. It models the implementation view of the software. We can use component to represent source code, XML or any piece of software. When using large software system it is common to break the software in to manageable subsystems. In

UML component classifier represents this. A component is a replaceable, executable piece of larger system whose implementation details are hidden. The functionality provided by a component is specified by a set of provided interfaces that the component realizes. In addition to providing interfaces, a component may require interfaces in order to perform. These are called required interfaces. Components are designed to be reused.

Component diagram are valuable because they:

1. Model the real software in the implementation environment.
2. They bring forward configuration issues through the dependency relationships.
3. They provide an accurate picture of existing systems prior to making changes or enhancements.



Component diagram for LOAN DEFAULTER PREDICTION System.

6 DEPLOYMENT DESIGN

The deployment diagram models the hardware of the implementing environment. Each node on a deployment diagram typically represents the type of hardware such as disk drive, a client PC, a server or a processor. A node may also represent a human being or organizational unit. Nodes are like classes. They represent a type of device, not a specific device, and the features of each device. Like classes they are related using association that explains how the nodes may be connected.

Deployment diagrams are valuable because:

1. They model the hardware platform for a system.
2. Identify hardware capabilities that affects performance planning and software configuration.

Deployment diagram for Loan Defaulter Prediction System.

1. The LDP system will be deployed in the IBM Blue mix Cloud Server.
2. The client computer will access the software deployed at the server using web browser.
3. The data will be accessed through the IBM DB2 database server.

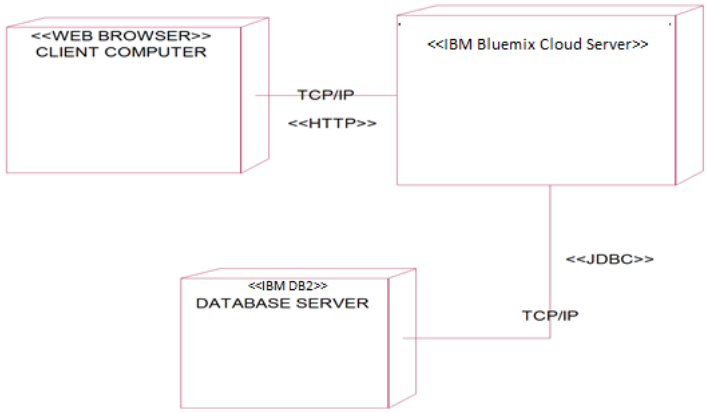


Fig 9. Deployment diagram for Loan Defaulter Prediction System.

7 HUMAN INTERFACE DESIGN

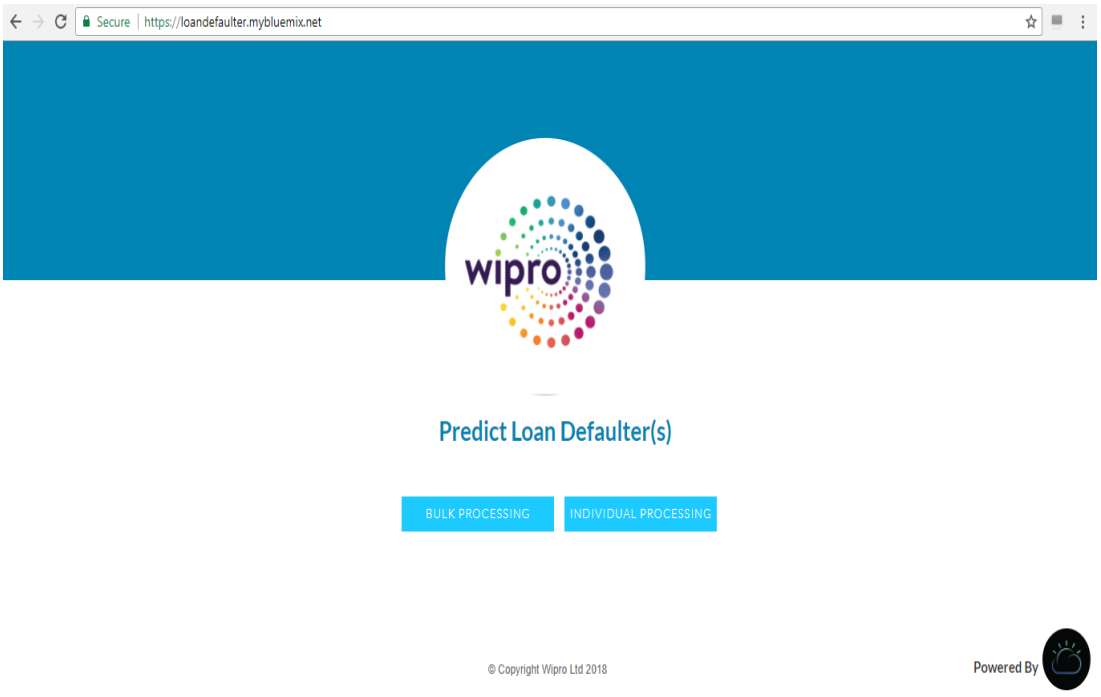
This section of the document will describe the end-user interface of LDPS. Further, the remainder of this document is divided into the overview of user interface and Screenshots.

7.1 OVERVIEW OF USER INTERFACE

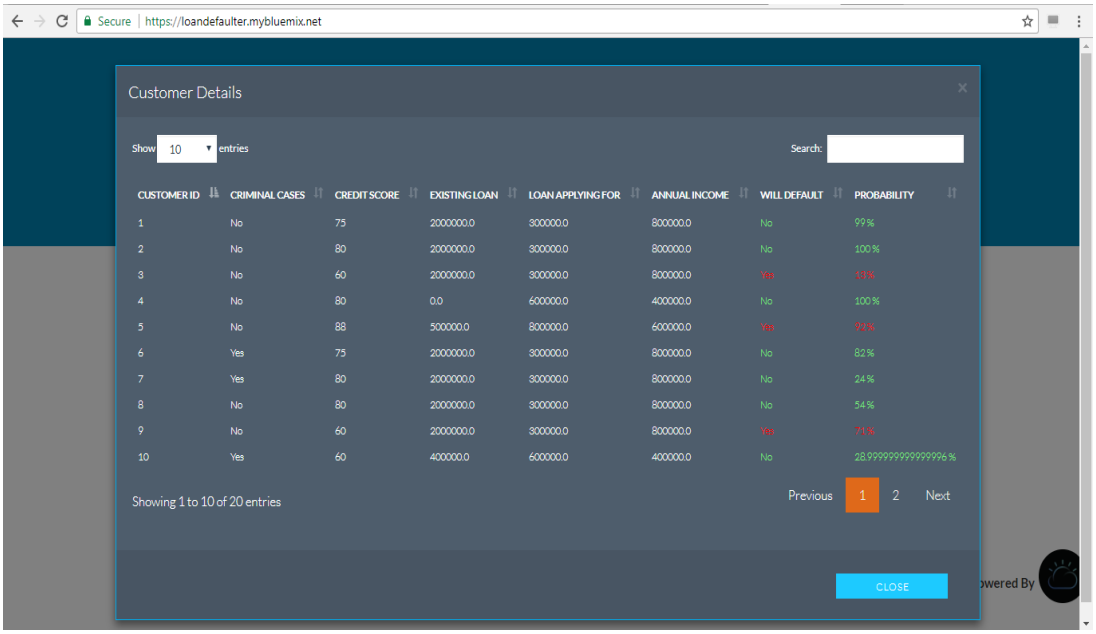
The User interface of LDPS will be served by JSP pages. When user open the system firstly, the home JSP page will appear. Further, the control will be directed as per the programming logics.

7.2 SCREEN IMAGES

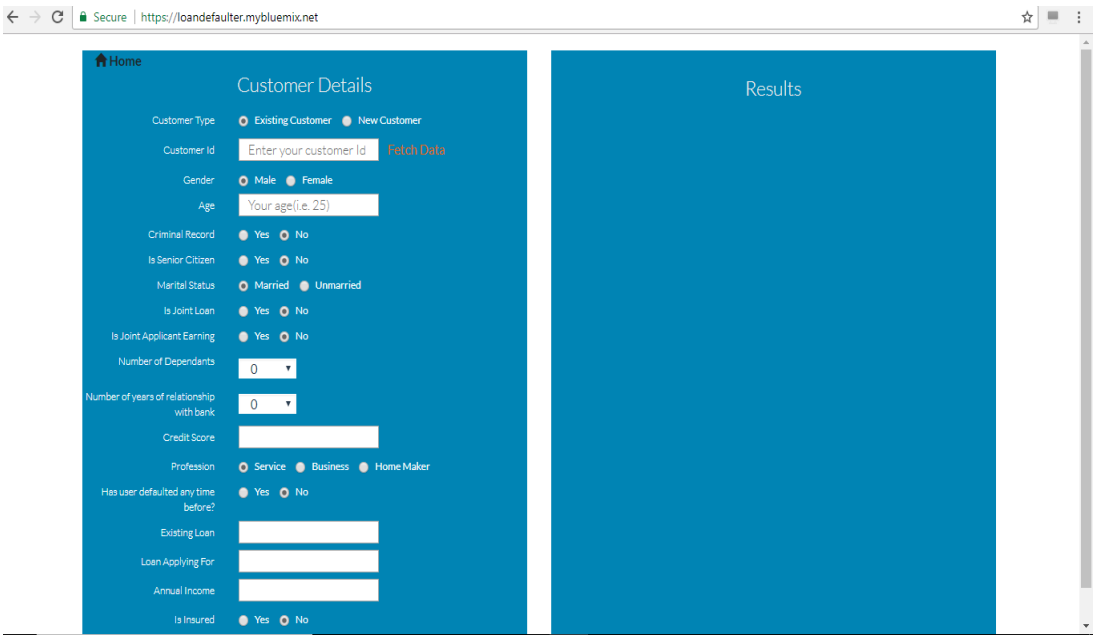
Home Page



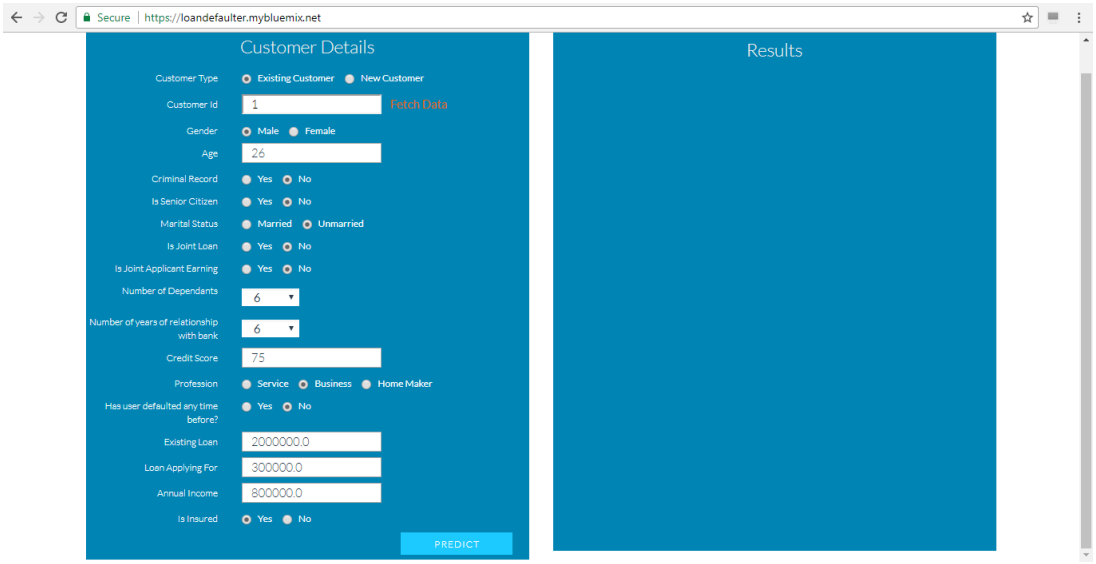
Bulk Processing in LDPS



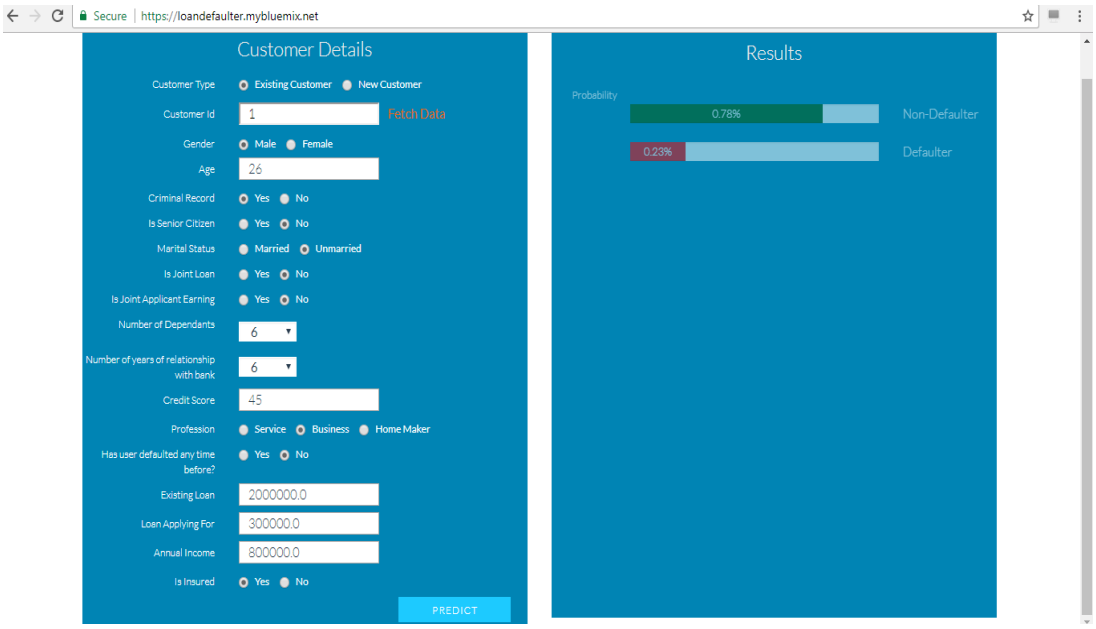
Fetch Details for Existing Customer



Fetch Details with Customer Id



Perform Loan Defaulter Prediction



8 EQUIREMENT MATRIX

REQ #	Description
REQ-SR1	The database backend system in use will be IBM DB2.
REQ-SR2	The Front-end and middle logic will be written using JAVA-J2EE.
REQ-SR3	Code will be stored on the IBM Blue mix server.
REQ-SR4	Our development environment will be the latest Eclipse Integrated Development Environment.
REQ-SR5	We will use IBM Bluemix as cloud server.
REQ-SR6	We may write scripts to create synthetic code in JavaScript/JQuery.
REQ-SR7	We may make changes to any of the above system requirement at any time and for any reason.