

LOAN MANAGEMENT SYSTEM

A Project Report

Submitted by:

**Ashutosh Pandey
20010001011**

In partial fulfillment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

**GATEWAY INSTITUTE OF
ENGINEERING & TECHNOLOGY**
DELHI - NCR, SONIPAT, INDIA

June, 2024

DECLARATION

I hereby, declare that the Project entitled “LOAN MANAGEMENT SYSTEM” submitted for the degree of Bachelor of Technology in Computer Science & Engineering is my original work. This Project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

Signature of the Student

Place:

Date:

CERTIFICATE

This is to certify that the Project entitled “LOAN MANAGEMENT SYSTEM” is the bonafide work carried out by “Ashutosh Pandey”, 8th semester student of Bachelor of Technology in Computer Science & Engineering of Gateway Institute of Engineering & Technology (GIET), Sonepat affiliated to DCRUST, Murthal during the academic year 2023-24, in partial fulfillment of the requirements for the award of the degree. This Project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

Signature of the Guide

Place:

Date:

ACKNOWLEDGEMENT

I express my sincere gratitude to my industry guide **Mr. Anil Arora** for her able guidance, continuous support, and cooperation throughout my project, without which the present work would not have been possible.

I would also like to give special thanks to my department HOD, **Mr. Anil Arora** for the guidance he provided throughout the project. I shall be failing in my works if I did not convey my thanks to my parents, who provided me all the thoughts and insights.

A special word of thanks to all the respondents for sharing their valuable time with me.

ABSTRACT

The Loan management system is important and helps to ensure success or failure of any credit institution. Mortgage loan problems have always been a key note on the risk of loan loss. The scope of this project is to provide good communication and communication between the customer and the manager. The current system can be a user-friendly system, which does not store data in the proper security and can easily track information and contains the operation of fast-recovery information, such as customer data, all loan details and includes many documents. The Financial Management System is designed to perform the functions of the back offices of a bank and a non-cash financial institution offers any sort of loan. The system can make daily operations more efficient and provide faster response. Including adding, editing, retrieving customer information, maintaining and issuing new loans, change the loan rate. The scope of this project is to use the loan in a very smart way.

The project includes a system analysis and style for obtaining a loan details process, settlement process, and approving the payment process. the existing system identifies issues arising from the functionality of the book. This project is designed to hit many issues such as data shortage, data inaccuracy, time, etc. The new computerized system will minimize errors while providing more control over the system and more robust management information in the form of implementation strategies. The new system was monitored to ensure that there was no error in the systems, so the program results met the export target financial target. to enhance the effective management of consumers, the system must also be developed to support other bidding loan details.

TABLE OF CONTENTS

Contents	Page No
Chapter 1: Introduction	10 - 11
1.1 About organization	10
1.2 Aims and objective of organization	11
Chapter 2: Introduction to Project.....	12
2.1 About project	12
Chapter 3: System study	13 - 15
3.1 Existing system	13
3.2 Proposed system along with objective	14
3.3 Feasibility study	15
Chapter 4: Project monitoring system	16 - 17
4.1 Gantt chart.....	17
Chapter 5: Requirement Specification	18 - 32
5.1 System Requirement.....	18
5.2 Software Requirement.....	20
5.3 Hardware Requirement.....	20
Chapter 6: System (Program) Design.....	33 - 48
6.1 Database design.....	33
6.2 DFD Design.....	36
6.3 E/R Design.....	37
6.4 Program Design.....	39
6.5 Screen Design.....	46

Chapter 7: System Testing.....	49 - 52
7.1 Test Data Preparation.....	49
7.2 Type of testing applied.....	51
7.3 Implementation of Testing.....	52
 Chapter 8: System Implementation.....	 53 - 54
 Chapter 9: Documentation.....	 55 - 56
9.1 Operational Document.....	55
9.2 User Manual	56
 Chapter 10: Conclusion & Further Scope.....	 57
 Chapter 11: Bibliography & References.....	 58

LIST OF FIGURES

Fig. No.	Description	Page No.
4.1	Gantt Chart	17
5.1	The first screen of VS Code	27
6.1	The notations and symbols used to construct the DFD	34
6.2	Context Level DFD for Recommendation Process	35
6.3	E-R model for Recommendation System	37

LIST OF TABLES

Table No	Description	Page No.
6.1.1	Table for Debtor	34
6.1.2	Table for Loan offers	34
6.1.3	Table for Loan Transaction	34
6.1.4	Table for Loan Information	35
6.1.5	Table for Payment Information	35
6.1.6	Table for Report	35

CHAPTER - 1

INTRODUCTION

1.1 Introduction of Organization

Gateway Institute of Engineering and Technology, located in Sonipat, Haryana, is a distinguished institution dedicated to providing quality education in the field of technology. The institute is affiliated with the Society Act and is registered with the Government of Haryana. It operates with the vision of offering a wide range of programs and commercial training, both recognized by the State Government and Central Government.

The institute boasts a well-equipped library housing an extensive collection of books, journals, and magazines authored by experts, both from India and abroad. This resource-rich environment benefits students pursuing IT and management courses.

Gateway Institute of Engineering and Technology is committed to ensuring that students have access to the latest hardware and software infrastructure. The institution aims to fulfill the computing needs of all its students and training requirements for programming, development, and design-related courses. The programming, development, and design labs are equipped with state-of-the-art hardware components, operating systems, and software to facilitate hands-on learning experiences.

The institution's vision extends beyond geographical boundaries, aiming to establish a strong tradition of high-quality, reliable, and cost-effective technology education. The goal is to ensure customer satisfaction and produce graduates capable of driving technological changes on a global scale.

1.2 Aim & Objective of Organization

Gateway Institute of Engineering and Technology (GIET) is committed to establishing a tradition of cost-effective, quality education and services through continuous improvement. The institution aspires to accomplish its mission by nurturing a dedicated team of professionals who work diligently to develop expert manpower. GIET's aim is to contribute to the nation by producing well-rounded IT professionals.

The faculty at GIET is the cornerstone of its success. These educators possess a deep understanding of real-world situations and impart their knowledge to students effectively. They play a crucial role in helping students tackle real-life challenges with confidence and enthusiasm. The faculty members design and update the study programs with the guidance of department heads, ensuring that the curriculum remains relevant and up-to-date.

The society's primary function is to provide higher technical education at affordable rates to all aspiring candidates. The institution offers a diverse range of courses, including C, C++, J2SE, J2EE, .NET Programming, and Web-site Development using PHP & MYSQL. In addition to these core courses, GIET also offers postgraduate and advanced diploma programs, as well as shorter certificate courses, to cater to the varied educational needs of its students.

Gateway Institute of Engineering and Technology is committed to nurturing a new generation of technologically adept individuals, and its dedicated faculty and comprehensive programs play a vital role in fulfilling this commitment.

Some other courses offered by the institute are listed below:

- ▶ Bachelor of Technology in Computer Science Engineering (4 Years)
- ▶ Bachelor of Computer Application (3 Year)
- ▶ Master of Computer Application (2 Year)
- ▶ Bachelor of Business Administration (3 Years)
- ▶ Master of Business Administration (2 Years)

CHAPTER - 2

INTRODUCTION TO PROJECT

Introduction

This program is called a loan management program. This method is created to keep records about consumers who have taken out bank loans. A registered user can sign in to the Loan system using their email id or user id and password. After logging in to this process there are decisions to add new customers, manage old customer account and check other details etc. Many new customers are visiting this bank, so adding new customer information and keeping records is very easy to use.

Description

A Loan Management System is a software or machine learning system designed to provide personalized method to keep records about the customer who have taken the bank loans. These loan management are typically based on the user's preferences, past behavior, or the characteristics of items or content available in each system.

Objective

The objective of loan management is to reduce the growth in utility peak demand in order to reduce the construction of new generation and transmission equipment and to more efficiently use existing equipment. To significantly reduce peak demand, substantial amounts of deferrable customer loads must exist. Hence, there is a huge scope of exploration in this field for improving scalability, accuracy and quality of Loan Management. Loan Management System is very powerful and important system.

CHAPTER - 3

SYSTEM STUDY

3.1 Existing System

Existing loan management system use a combination of data analysis, machine learning, and user interaction to provide personalized recommendations in a variety of domains, including e-commerce, streaming, social media, and more. These systems have evolved significantly over the years, with many relying on complex algorithms to improve the accuracy and relevance of their recommendations. Here are some common characteristics and components of existing loan management system.

3.2 Proposed System

Our project performs these tasks by using computer. It can rely on the properties of the items that a user likes, which are analyzed to determine what else the user may like; or, it can rely on the likes and dislikes of other users, which the loan management system then uses to compute a similarity index between users and recommend items to them accordingly. A loan management system is a behavioral tool that helps guide users into the best possible product experience based on their personal demographic characteristics and the tendencies they have exhibited in the past. So, based on the user's profile, these systems can predict whether a product will be preferable by a user or not. More broadly, recommender systems represent user preferences for the purpose of suggesting items to purchase or examine and are now an integral part.

A loan management system is a type of data filtering tool using machine learning algorithms to recommend the most relevant items to a particular user or customer. It operates on the principle of finding patterns in consumer behavior data, which can be

collected implicitly or explicitly. The goal of a recommender system is to generate meaningful recommendations to a collection of users for items or products that might interest them.

The main objectives of preliminary analysis are to identify the customer's needs, evaluate system concept for feasibility, perform economic and technical analysis, perform cost benefit analysis and create system definition that forms the foundation for all subsequent engineering works. There should be enough expertise available for hardware and software for doing analysis.

3.3 Feasibility Study

Feasibility study gives us an idea about the workability of the system. It is a measure of the impact on the organization ability to meet the user's needs and the effective use of the resources available. The objective of a feasibility study is a formal proposal. This is simply a report; a formal document detailing the nature and scope of the proposed system.

ECONOMIC FEASIBILITY:

In economic feasibility, which is also known as the Cost Benefit Analysis, costs are compared for systematic development and implementation over the outcomes and benefits of the system. Here the benefit expected from the system is compared with the cost. If the benefits outweigh costs, then decision is made to design and implement the system. **LOAN MANGEMENT SYSTEM** has many advantages over the manual system, and the most important of these advantages is that it will minimize the human efforts. The benefits derived from the system are expected to be much more than the cost, and so we accept it economically feasible.

TECHNICAL FEASIBILITY:

Technical feasibility centers on the existing computer system (hardware, software etc.) and to what it can support the proposed system. Under this consideration it is to be examined

as to whether the existing hardware and software is enough to carry out the system. The hardware as well as the software resources required developing and implementing the software for the **LOAN MANAGEMENT SYSTEM** is presently available. So, the proposed system can be regarded as technically beneficial.

BEHAVIORAL FEASIBILITY:

In general people are resistant to change and computers have been known to facilitate change. It is a common notion that computer installation APR led to transfers, turn-over, re-training and changes in the employee status. Therefore, it is understandable that the introduction of the **LOAN MANAGEMENT SYSTEM** would require special efforts to educate and train the staff on new ways of conducting Flightiness. Presence of a large number of computerized systems, people are very much familiar with computers and because of this the proposed system is considered to behaviorally feasible.

CHAPTER - 4

PROJECT MONITORING SYSTEM

We can monitor the progress of software project using Gantt chart

4.1 Gantt chart

Gantt Charts are project control technique that can be used for several purposes, including scheduling, budgeting and resource planning. A Gantt chart is a bar chart, with each bar representing an activity. The bars are drawn against a time line. The length of each bar is proportional to the length of time planned for the activity.

During the scheduling activity and also during implementation of the project, new activities APR be identified that were not envisioned during the initial planning. The developer must then go back and revise the breakdown structure and the schedules to deal with these new activities.

We estimated the number of weeks required for each of the seven tasks as follows:

Analysis	1 Weeks
Design	2 Weeks
Coding	3 Weeks
Testing	2 Days
Write Manual	2 Days
Implementation	1 Day
Maintenance	2 Days

Detail schedule of activities are listed in the Gantt chart shown in figure below.

Activities	20-FEB TO 27-FEB	28-FEB TO 11-MAR	12-MAR TO 12-APR	13-APR TO 31-APR	07-MAY TO 09-MAY	10-MAY	11-MAY TO 12-MAY
PROBLEM ANALYSIS							
DESINING							
CODING							
TESTING							
WRITE MANUAL							
IMPLEMENTATIO N							
MAINTAINANCE							

Figure 4.1: Gantt Chart

CHAPTER - 5

SOFTWARE REQUIREMENT SPECIFICATION

The Software Requirements Specification is produced at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by establishing a complete information description, a detailed functional description, a representation of system behavior, an indication of performance requirements and design constraints, appropriate validation criteria, and other information pertinent to requirements. The National Bureau of Standards, IEEE (Standard No. 830-1984), and the U.S. Department of Defense have all proposed candidate formats for software requirements specifications (as well as other software engineering documentation).

The Introduction of the software requirements specification states the goals and objectives of the software, describing it in the context of the computer-based system. Actually, the Introduction may be nothing more than the software scope of the planning document. The Information Description provides a detailed description of the problem that the software must solve. Information content, flow, and structure are documented. Hardware, software, and human interfaces are described for external system elements and internal software functions.

A description of each function required to solve the problem is presented in the Functional Description. A processing narrative is provided for each function, design constraints are stated and justified, performance characteristics are stated, and one or more diagrams are included to graphically represent the overall structure of the software and interplay among software functions and other system elements. The Behavioral Description section of the specification examines the operation of the software as a consequence of external events and internally generated control characteristics. Validation Criteria is probably the most important and, ironically, the most often neglected section of the Software Requirements Specification. How do we recognize a successful implementation? What classes of tests

must be conducted to validate function, performance, and constraints? We neglect this section because completing it demands a thorough understanding of software requirements—something that we often do not have at this stage. Yet, specification of validation criteria acts as an implicit review of all other requirements. It is essential that time and attention be given to this section.

In many cases the Software Requirements Specification may be accompanied by an executable prototype (which in some cases may replace the specification), a paper prototype or a Preliminary User's Manual. The Preliminary User's Manual presents the software as a black box. That is, heavy emphasis is placed on user input and the resultant output. The manual can serve as a valuable tool for uncovering problems at the human/machine interface.

Review: A review of the Software Requirements Specification (and/or prototype) is conducted by both the software developer and the customer. Because the specification forms the foundation of the development phase, extreme care should be taken in conducting the review.

The review is first conducted at a macroscopic level; that is, reviewers attempt to ensure that the specification is complete, consistent, and accurate when the overall information, functional, and behavioral domains are considered. However, to fully explore each of these domains, the review becomes more detailed, examining not only broad descriptions but the way in which requirements are worded. For example, when specifications contain “vague terms” (e.g., some, sometimes, often, usually, ordinarily, most, or mostly), the reviewer should flag the statements for further clarification.

Once the review is complete, the Software Requirements Specification is "signed off" by both the customer and the developer. The specification becomes a "contract" for software development. Requests for changes in requirements after the specification is finalized will not be eliminated. But the customer should note that each after the fact change is an extension of software scope and therefore can increase cost and/or protract the schedule.

Even with the best review procedures in place, a number of common specification problems persist.

The specification is difficult to "test" in any meaningful way, and therefore inconsistency or omissions may pass unnoticed. During the review, changes to the specification may be recommended. It can be extremely difficult to assess the global impact of a change; that is, how a change in one function affects requirements for other functions. Modern software engineering environments incorporate CASE tools that have been developed to help solve these problems.

5.2 Software Requirement

- **Operating System: Windows XP, Windows 11 or Later**

The system will be built on windows compatible environment. The application will be web based developed using Java technology.

- **GUI –based Software:** Visual Studio Code
- **Programming Language:** DJANGO, PYTHON
- **Data Base:** SQLite

5.3 Hardware Requirements:

- **Processor:** Intel Pentium V or higher
- **SSD:** 512GB or more
- **RAM:** 8GB or more
- **Keyboard, Mouse, Monitor**

GUI Tools and its Importance:

Graphical User Interface helps users to interact graphically with the computer i.e. a user can use different graphical capability such as text, mouse and picture to communicate with the computer.

The GUI divided into following types:

1. Front-End: The end is visible to the user and responsible for interactive with the user. The front end is responsible for receiving user's queries, passing it over to the backend. The front-end includes graphical user interface. Apart from the user interface, the front-end also contains database objects that form a layer between the user interface and the backend. Ex: Python etc.

2. Back-end: This end is not visible to the user but it performs all database-related tasks in the background. It receives user's queries and passes them to the database server. After processing the user's requests and queries, the server returns the results to the backend that passes it to the front-end. Ex: MONGODB, DJANGO.

Importance of GUI Application:

1. It is user friendly
2. It frees user from the internal working of the system.
3. It performs maximum of work, while requiring a minimum of information from users.

Contents of Front-end:

A front-end interface can display various graphical controls/objects. The graphical objects that facilitate interaction with users are also known as **User-Interface Objects** such as Text

Box, Label and Button the User-Interface objects have characteristics called properties that users can view or change, for example, a text box has a font property, which users can easily view or edit.

About Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Applications for Python

Python is used in many application domains. Here's a sampling.

The Python Package Index lists thousands of third party modules for Python.

Web and Internet Development

Python offers many choices for web development:

Frameworks such as Django and Pyramid.

Micro-frameworks such as Flask and Bottle.

Advanced content management systems such as Plone and django CMS.

Python's standard library supports many Internet protocols:

HTML and XML

JSON

Email processing.

Support for FTP, IMAP, and other Internet protocols.

Easy-to-use socket interface.

And the Package Index has yet more libraries:

Requests, a powerful HTTP client library.

Beautiful Soup, an HTML parser that can handle all sorts of oddball HTML.

Feedparser for parsing RSS/Atom feeds.

Paramiko, implementing the SSH2 protocol.

Twisted Python, a framework for asynchronous network programming.

Scientific and Numeric

Python is widely used in scientific and numeric computing:

SciPy is a collection of packages for mathematics, science, and engineering.

Pandas is a data analysis and modeling library.

IPython is a powerful interactive shell that features easy editing and recording of a work session, and supports visualizations and parallel computing.

The Software Carpentry Course teaches basic skills for scientific computing, running bootcamps and providing open-access teaching materials.

Education

Python is a superb language for teaching programming, both at the introductory level and in more advanced courses.

Books such as How to Think Like a Computer Scientist, Python Programming: An Introduction to Computer Science, and Practical Programming.

The Education Special Interest Group is a good place to discuss teaching issues.

Desktop GUIs

The Tk GUI library is included with most binary distributions of Python.

Some toolkits that are usable on several platforms are available separately:

wxWidgets

Kivy, for writing multitouch applications.

Qt via pyqt or pyside

Platform-specific toolkits are also available:

GTK+

Microsoft Foundation Classes through the win32 extensions

Software Development

Python is often used as a support language for software developers, for build control and management, testing, and in many other ways.

SCons for build control.

Buildbot and Apache Gump for automated continuous compilation and testing.

Roundup or Trac for bug tracking and project management.

Business Applications

Python is also used to build ERP and e-commerce systems:

Odoo is an all-in-one management software that offers a range of business applications that form a complete suite of enterprise management applications.

Tryton is a three-tier high-level general purpose application platform.

About Django

Django is a high-level Python web framework that promotes rapid development and clean, pragmatic design. It's built on top of Python's solid foundation, offering developers a powerful toolkit to create web applications quickly and efficiently. With its batteries-included philosophy, Django provides developers with everything they need to build robust web applications, from authentication and authorization to database management and templating.

One of Django's standout features is its adherence to the DRY (Don't Repeat Yourself) principle, which encourages code reuse and reduces redundancy. This not only speeds up development but also ensures easier maintenance and scalability of projects over time.

Django's architecture follows the MVC (Model-View-Controller) pattern, although it refers to it as MTV (Model-Template-View) due to its slight variation. Models define the data structure, views handle the logic and interact with the models, and templates render the HTML to present to the user. This clear separation of concerns allows for modular development and easier collaboration among team members.

The framework comes bundled with a robust ORM (Object-Relational Mapping) layer, allowing developers to interact with databases using Python objects instead of SQL queries directly. This abstraction simplifies database operations and reduces the risk of SQL injection attacks.

Django also includes a powerful admin interface out of the box, which can be easily customized to manage site content and user data. This saves developers time and effort in building administrative interfaces for their applications.

Moreover, Django emphasizes security, providing built-in protections against common web vulnerabilities such as CSRF (Cross-Site Request Forgery) and XSS (Cross-Site Scripting). Additionally, it encourages best practices like secure password hashing and HTTPS deployment.

The Django community is vibrant and supportive, with a wealth of documentation, tutorials, and third-party packages available to extend the framework's functionality. The Django Software Foundation oversees the development and maintenance of the framework, ensuring its continued growth and stability.

Overall, Django is an excellent choice for web developers looking to build scalable, maintainable web applications with Python. Its focus on simplicity, flexibility, and performance makes it a preferred framework for projects of all sizes, from small startups to large enterprise applications.

VS CODE

Visual Studio Code, often abbreviated as VS Code, is a free and open-source code editor developed by Microsoft. It has gained immense popularity in the software development community for its versatility, extensibility, and wide range of features. Here are some key points about Visual Studio Code:

- 1. Cross-Platform:** Visual Studio Code is available for Windows, macOS, and Linux, making it a versatile choice for developers on various platforms.
- 2. Lightweight:** VS Code is known for its speed and efficiency. It's a lightweight code editor that starts quickly and consumes minimal system resources.

3. Intuitive User Interface: It features a clean and user-friendly interface that allows for easy navigation and customization. The user interface is highly configurable to suit your preferences.

4. Integrated Development Environment (IDE) Features: Although VS Code is a code editor, it offers many IDE-like features through extensions. You can add extensions to enable features such as debugging, version control, linting, code formatting, and intelligent code completion.

5. Rich Language Support: Visual Studio Code supports a wide range of programming languages out of the box and offers extensions for even more. These extensions provide language-specific features like syntax highlighting, code navigation, and integrated documentation.

6. Git Integration: VS Code has built-in Git integration, making it easy to work with version control directly from the editor. You can commit, push, pull, and manage branches without leaving the editor.

7. Extensions and Marketplace: Visual Studio Code has a vibrant extension ecosystem. You can enhance the editor's functionality by installing extensions from the Visual Studio Code Marketplace. There are extensions available for virtually any development task or technology.

8. Integrated Terminal: It includes an integrated terminal that allows you to run command-line tools, scripts, and even start development servers right within the editor.

9. Debugging Tools: VS Code supports debugging for various programming languages. You can set breakpoints, inspect variables, and step through code with ease.

10. Customizable Themes and Color Schemes: You can customize the appearance of VS Code by installing themes and color schemes. This allows you to personalize the editor's look and feel.

11. Task Automation: Visual Studio Code allows you to define and run tasks, which can automate common development workflows, such as building, testing, and deploying your projects.

12. IntelliSense: VS Code provides intelligent code completion, code suggestions, and tooltips to help you write code more efficiently and with fewer errors.

13. Live Share: Visual Studio Code Live Share is an extension that enables real-time collaboration with other developers. You can share your coding session with team members, allowing them to edit, debug, and chat with you while you work.

14. Continuous Improvement: Microsoft actively maintains and updates Visual Studio Code, ensuring that it stays up to date with the latest technologies and best practices.

15. Community Support: The VS Code community is active and helpful. You can find a wealth of resources, extensions, and tutorials online.

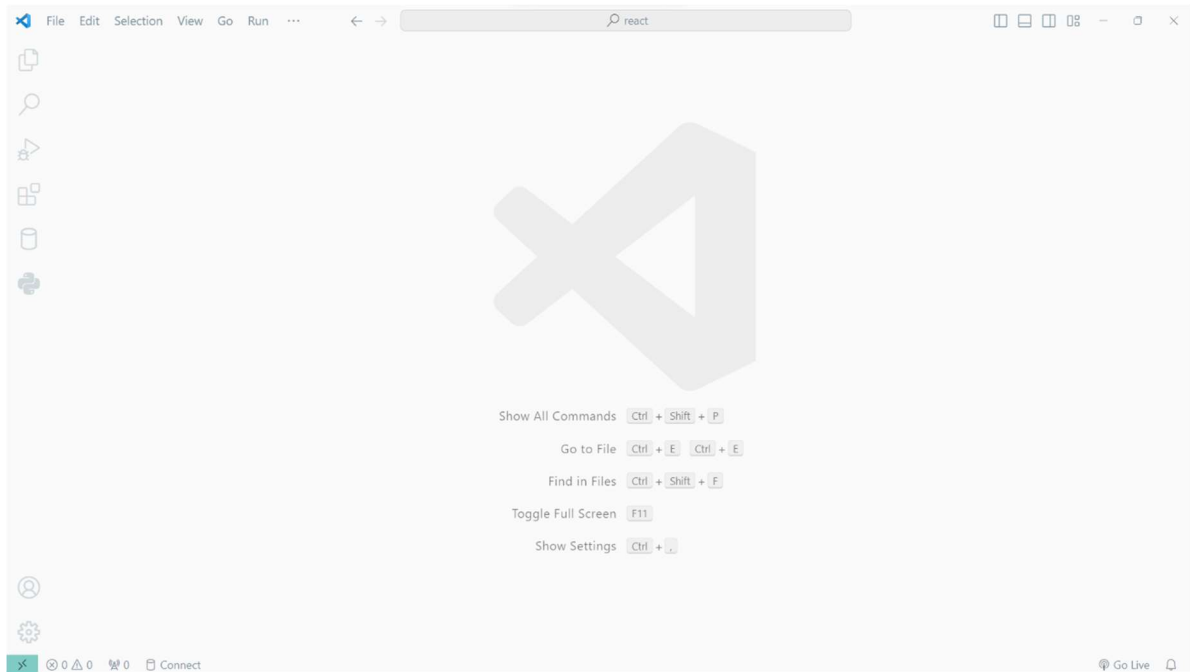


Figure 5.1: The first screen of Visual studio code

Components of Visual Studio Code:

Visual Studio Code (VS Code) is a versatile and extensible code editor with numerous components that make it a powerful tool for software development. Here are some key components of Visual Studio Code:

- 1. User Interface (UI):** The user interface is the main component that provides the visual environment for writing and managing code. It includes menus, toolbars, tabs, and the editor pane.
- 2. Editor:** The heart of VS Code is the code editor, where you write and edit your code. It supports syntax highlighting, code folding, line numbers, and various text editing features.
- 3. Integrated Terminal:** VS Code includes an integrated terminal that allows you to run command-line tools and scripts directly within the editor. This is convenient for tasks like running development servers, build scripts, and Git commands.

4. Side Bar: The Side Bar is a panel on the left side of the editor that provides quick access to various functionalities, such as file navigation, version control (Git), extensions, and more.

5. Extensions: Extensions are add-ons that enhance the functionality of VS Code. They can provide support for additional programming languages, debugging capabilities, code formatting, linters, themes, and various other features. The Extensions View in the Side Bar allows you to manage and install extensions.

6. Explorer: The Explorer is a file navigation tool within VS Code that helps you manage your project's directory structure. It allows you to browse and open files and folders, create new files, and perform file operations.

7. Source Control (Git): VS Code has built-in Git integration that enables you to manage version control tasks. You can stage, commit, push, pull, and resolve merge conflicts without leaving the editor.

8. Debugging Tools: VS Code provides debugging support for various programming languages and runtimes. You can set breakpoints, inspect variables, and interactively debug your code.

9. Activity Bar: The Activity Bar is a vertical panel on the side of the editor that provides quick access to different activities, such as source control, debugging, extensions, and more.

10. Status Bar: The Status Bar appears at the bottom of the editor and displays information like line and column numbers, file encoding, indentation settings, and extensions' status.

11. Settings and Preferences: VS Code allows you to configure a wide range of settings and preferences to tailor the editor to your needs. You can access these settings through the "Settings" panel.

12. Command Palette: The Command Palette is a powerful tool for executing various commands, such as opening files, running extensions, and configuring settings. You can access it by pressing Ctrl+Shift+P (or Command+Shift+P on macOS).

13. Search and Replace: VS Code provides a Find and Replace feature that allows you to search for specific text within your project and replace it as needed.

14. Themes and Color Schemes: You can customize the appearance of VS Code by choosing different themes and color schemes. These affect the overall look and feel of the editor.

15. Keyboard Shortcuts: VS Code offers a wide range of keyboard shortcuts to improve productivity. You can customize these shortcuts or use predefined ones.

16. Extension Marketplace: The Extension Marketplace is an online repository of VS Code extensions. You can search for, install, and manage extensions from this marketplace.

These components work together to create a feature-rich and flexible code editor that can be tailored to your specific development needs. Visual Studio Code's extensibility and its large library of extensions make it a popular choice among developers for a wide range of programming tasks.

BACK-END

Backend: For processing and responding to the front-end requests, the backend maintains a database. A database is an organized collection of related information where organized means that data is stored in the form of rows and columns (table). A system that manages

the database (i.e, to create the database, add records, delete records and modify the records) is called database management system (DBMS).

Data Base is a collection of tables and table is a collection of records in a tabular form i.e. in row and columns format.

Ex:

Table: Student

<i>Roll no</i>	<i>Name</i>	<i>Class</i>	<i>Fees</i>
17	ABC	BCA	750
18	XYZ	BSC	850
19	PQR	BBA	950

In the above table, a row represents relationship b/w different set of values, so a table is also called a Relation. A row in the table is called tuple or record & a column in the table is called attribute or field.

RDBMS (Relational Database Management System):- RDBMS helps us to create & manipulate different types of databases or relations in the computer i.e. a user can insert, delete and modify records. Different types of RDBMS software available in the market are: ORACLE, My SQL, SQL SERVER.

MS ACCESS: It is one of the leading RDBMS software created by Microsoft Corporation Ltd. It helps us to create & maintain different types of databases. Some common features of MS ACCESS are:

- i. It helps us to create any type & size of database.
- ii. It helps us to manipulate the table data i.e. we can insert, delete or modify any specific information from any specific table.
- iii. It helps us to query the table for retrieving (displaying) any specific information from any specific database.

CHAPTER - 6

PROGRAM DESIGN

The **Loan management system** is created to keep records about consumers who have taken out bank loans.

The Loan management system contain many loans id and their data. It also contains user login details such as userID, Email, Username, password. The user can search about their loan and get details of the loan id and also get their related loan application and similarly for EMI details.

The project contains total 4 Pages as

- | | |
|----------------------|--------------------|
| 1. Login | (Login Page) |
| 2. Home | (Home Page) |
| 3. New Asset | (Register Page) |
| 4. Client Management | (Loan Status Page) |

6.1 Database Design of the Project

These tables below provide the complete database table details such as **Field Name**, **Descriptions**, **data types**, and **character lengths**. Each of these tables represents the characteristics and the attributes of data storage. The **field** column presents the names of each database's attributes, the **description** column gives the complete thought of each attribute, the **type** column is their data type and the **length** are for their character lengths.

6.1.1 Table Name: Debtor

Field	Description	Type	Length
debtor_ID (PK)	Debtor's ID	Int	11
fname	Debtor's First Name	Varchar	255
lname	Debtor's Last Name	Varchar	255
contact_number	Debtor's Contact	Int	11
address	Debtor's Address	Text	
age	Debtor's Age	Int	11
gender	Debtor's Gender	Varchar	255

6.1.2 Table Name: Loan Offers

Field	Description	Type	Length
loan_ID (PK)	Loan ID	Int	11
loan_name	Loan Name	Varchar	255
amount	Loan Amount	Varchar	255
loan_range	Range of loan	Text	
interest	Loan Interest	Varchar	255

6.1.3 Table Name: Loan Transactions

Field	Description	Type	Length
transaction_ID (PK)	Employee ID	Int	11
debtor_ID (FK)	Debtor's ID	Varchar	30
date	Date of Loan Request	Date	
purpose	Loan Purpose	Text	
collateral	Collateral	Varchar	30

6.1.4 Table Name: Loan Information

Field	Description	Type	Length
info_ID (PK)	Information ID	Int	11
debtor_ID (FK)	Debtor ID	Int	11
loan_date	Date of Loan	Date	
loan_due	Due Date of Loan	Date	
payment_range	Range of the Payment	Int	11

6.1.5 Table Name: Payment Information

Field	Description	Type	Length
payment_ID (PK)	Payment ID	Int	11
info_ID (FK)	Information ID	Int	11
pay_amount	Payment Amount	Date	
payment_date	Date of Payment	Time	
debtor_ID	Debtor's ID	Int	11

6.1.6 Table Name: Reports

Field	Description	Type	Length
report_ID (PK)	Report ID	Int	11
debtor_ID (FK)	Debtor's Id	Int	11
loan_ID (FK)	Loan ID	Int	11
info_ID (FK)	Info_ID	Int	11
payment_ID (FK)	Payment ID	Int	11
date	Date of Report	Date	

6.2 DATA FLOW DIAGRAM

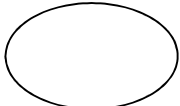
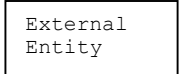


Data Flow Diagramming is a means of representing a system at any level of detail with a graphic network of symbols showing data flows, data stores, data processes, and data sources/destination. The data flow diagram is analogous to a road map. It is a network model of all possibilities with different detail shown on different hierarchical levels. This processes of representing different details level is called “leveling” or “partitioning” by some data flow diagram advocates. Like a road map, there is no starting point or stop point, any time or timing, or steps to get somewhere. We just know that the data path must exist

because at some point it will be needed. A road map shows all existing or planned roads because the road is needed.

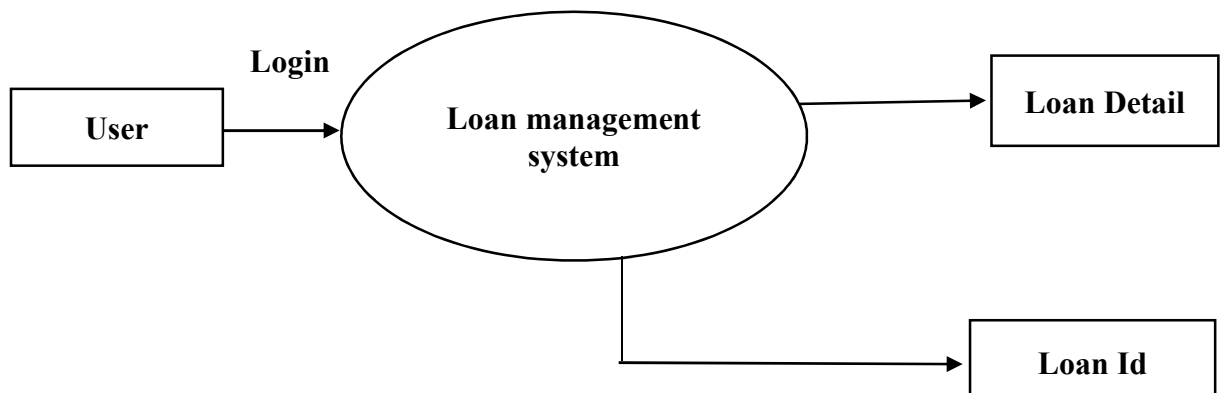
Details that is not shown on the different levels of the data flow diagram such as volumes, timing, frequency, etc. is shown on supplementary diagrams or in the data dictionary. For example, data store contents may be shown in the data dictionary.

Data Flow Diagram (DFD) uses a number of symbols to represent the systems. Data Flow Diagram also known as ‘Bubble Chart’ is used to clarify system requirements and identifying the major transformations that will become programs in system design. So it is the starting point of the design phase that functionally decomposes the requirements specifications down to the level of details.

The following diagram illustrates the notations and symbols used to construct the DFD:

	A circle or bubble represents a process transform Incoming Data flow(s) into outgoing flows.
	A producer or consumer of information that resides outside the bounds to be modeled.
	The arrowhead indicates the direction of flow.
	The table in which information will be stored ultimately.

Context Level DFD for Loan management system



6.3 E - R DIAGRAM

The Entity-Relationship (E/R) data model represents different entities, attributes and their relationship graphically. The E/R data model was introduced by P.P. Chen.

Components of E/R model:

The Entity relationship model contains following three components: Entity, Attribute and Relationship

1. Entity: An entity is a real-world data item having some characteristics and behaviour called properties. For Example, student is an entity having properties Rollno, Name and Marks. Similarly, Employee is also an entity having properties Empno, Ename & Salary.

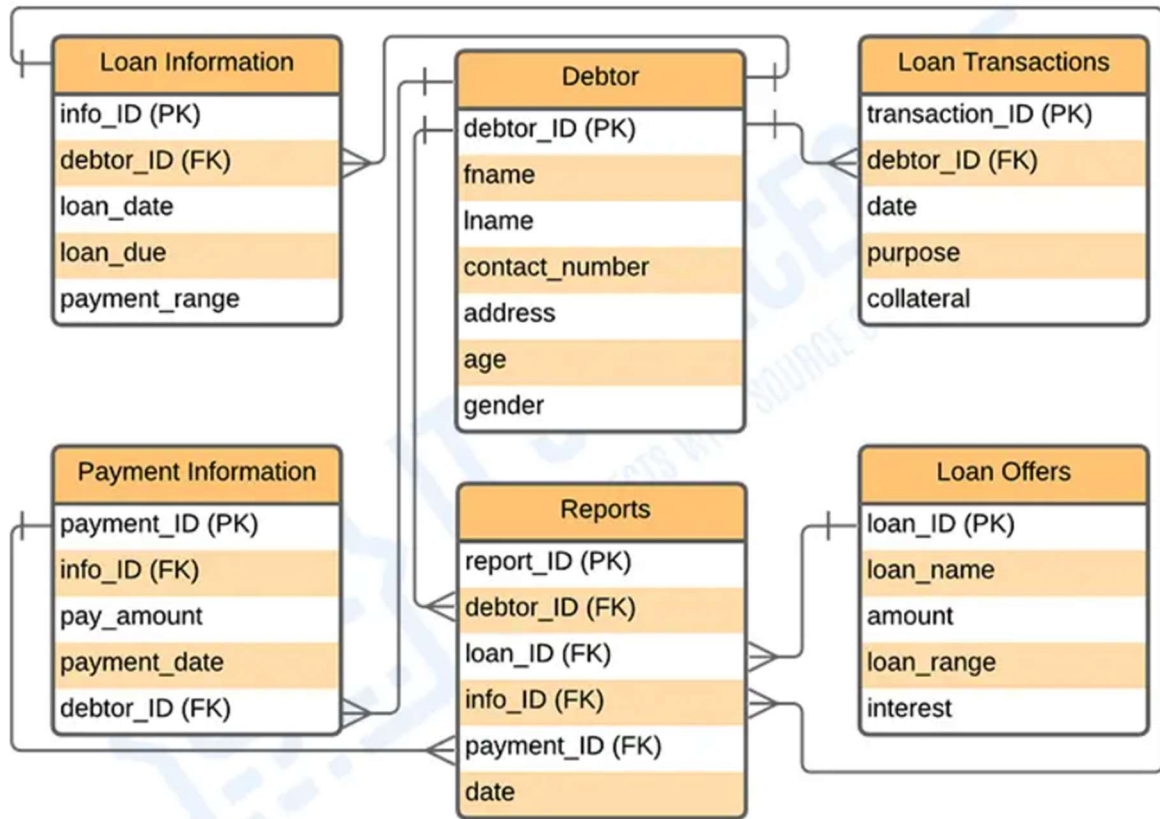
An **entity type (set)** is a set of entities having similar type i.e, entites having common properties are represented by an Entity Set. For example, group of students having properties Rollno, Name and Marks are represented by an Entity set Student. Similarly, the group of employees having properties Empno, Ename and Salary are represented by an entity set Employee.

An **entity instance** is an instance of entity set i.e, an entity instance is a specific individual, thing or object. For example, “Smith” is an instance of entity type Student, similarly 10000 is an instance of entity type Employee.

2. Attributes: An entity is represented by set of properties. These properties are called attributes. For example, student entity contains following properties Rollno, Name and Marks. Hence attributes of student entity set are Roll, Name and Marks. Similarly, Employee entity contains properties Empno, Ename and Salary, therefore employee entity has attributes Empno, ename and salary.

3. Relationship: A relationship is an association among several entities. For example, there is relationship between Vendor and Item as Vendor supply Item. Similarly, there is relationship between teacher and student as teacher teaches the student.

E/R Model for Loan management system



Detailed Description of Modules (Coding)

//Base.html

```
from django import forms
from .models import Client
from django.contrib.auth.models import User

# Form for user login
class LoginForm(forms.Form):
    username = forms.CharField(
        label='Username',
        help_text='Please enter your username',
        widget=forms.TextInput(attrs={'class': 'form-control lmsinput', 'required': True})
    )
    password = forms.CharField(
        label='Password',
        help_text='Please enter your password',
        widget=forms.PasswordInput(attrs={'class': 'form-control lmsinput', 'required': True})
    )

# Form for client data
class ClientForm(forms.ModelForm):
    class Meta:
        model = Client
        # Excluding some fields from the form
        exclude = ['user', 'created_by', 'updated_by', 'updated_on', 'company', 'branch']
        # Customizing widgets for form fields
        widgets = {
            'title': forms.Select(attrs={'class': 'form-control', 'required': True}),
            'fullname': forms.TextInput(attrs={'class': 'form-control', 'required': True}),
            'full_address': forms.TextInput(attrs={'class': 'form-control', 'required': True}),
```

```

# More fields and widgets...
}

# Form for user data
class UserForm(forms.ModelForm):
    class Meta:
        model = User
        fields = ('first_name', 'last_name', 'email', 'username', 'password')
# Customizing widgets for form fields
widgets = {
    'password': forms.PasswordInput(attrs={'class': 'form-control', 'required': True}),
    'username': forms.TextInput(attrs={'class': 'form-control', 'required': True}),
    'first_name': forms.TextInput(attrs={'class': 'form-control', 'required': True}),
    'last_name': forms.TextInput(attrs={'class': 'form-control', 'required': True}),
    'email': forms.EmailInput(attrs={'class': 'form-control', 'required': True}),
}

```

//models.py

```

from django.db import models
from django.contrib.auth.models import User
from dashboard.models import BaseModel # Assuming BaseModel is defined in
dashboard.models
from dashboard.choices import * # Importing choices from dashboard.choices
from django.db.models.signals import post_save
from django.dispatch import receiver

# Base model for common fields
class person(BaseModel):
    user = models.OneToOneField(User, on_delete=models.CASCADE,
related_name='person', default='1')

```



```

title = models.CharField(max_length=200, blank=True, choices=TITLE_CHOICES)
fullname = models.CharField(max_length=200, blank=True)
full_address = models.CharField(max_length=200, blank=True)
city = models.CharField(max_length=200, blank=True)
zip = models.CharField(max_length=200, blank=True)
state = models.CharField(max_length=200, blank=True)
phone = models.CharField(max_length=200, blank=True)
gender = models.CharField(max_length=200, choices=GENDER_CHOICES,
blank=True)
dob = models.DateField(blank=True, null=True)
state_of_origin = models.CharField(max_length=200, blank=True)
current_salary = models.CharField(max_length=200, blank=True)
employment_date = models.DateField(blank=True, null=True)
lga = models.CharField(max_length=200, blank=True)
job_description = models.TextField(blank=True)
marital_status = models.CharField(max_length=200,
choices=RELATIONSHIP_STATUS_CHOICES, blank=True)
picture = models.ImageField(max_length=200, blank=True, upload_to='employee/')
bio = models.TextField(max_length=200, blank=True)

```

Client model inheriting from person

```

class Client(person):
current_employer = models.CharField(max_length=200, blank=True)
years_in_workplace = models.TextField(max_length=100, blank=True)
vehicles_owned = models.CharField(max_length=200, blank=True)
years_at_residence = models.CharField(max_length=200, blank=True)
loan_officer = models.ForeignKey(User, on_delete=models.CASCADE,
help_text='Someone who manages the client among your staffs',
related_name='officer', null=True, blank=True)
residential_status = models.CharField(max_length=200,
choices=RESIDENTIAL_STATUS_CHOICES, blank=True)

```

```

educational_status = models.CharField(max_length=200,
choices=EDUCATIONAL_STATUS_CHOICES, blank=True)

# Employee model inheriting from person
class Employee(person):
years_in_workplace = models.TextField(max_length=100, blank=True)
manager = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='manager', blank=True, null=True)
educational_status = models.CharField(max_length=200,
choices=EDUCATIONAL_STATUS_CHOICES, blank=True)

# Signal to create related profile upon user creation
@receiver(post_save, sender=User)
def create_user_profile(sender, instance, created, **kwargs):
if created:
# Checking if the user is staff
if instance.is_staff:
Employee.objects.create(user=instance)
else:
Client.objects.create(user=instance)

```

//view.py

```

from django.shortcuts import render, get_object_or_404
from django.views.generic import ListView
from django.views.generic.edit import CreateView
from .models import Asset, assetDocument
from django.utils.decorators import method_decorator
from django.contrib.auth.decorators import login_required
from django.contrib.auth.mixins import LoginRequiredMixin
from .forms import assetForm, assetDocumentForm

```

```

from django.db import transaction
from django.template.loader import render_to_string
from django.http import JsonResponse
from django.forms.models import modelformset_factory
import pdb

# ListView for displaying a list of assets
class assetList(LoginRequiredMixin, ListView):
    login_url = '/account/login/'
    model = Asset
    template_name = 'assetmanager/asset_list.html'
    context_object_name = 'asset_list'

    def get_queryset(self):
        return Asset.objects.filter(company=self.request.user.person.company)

# Function-based view for saving an asset form and related documents
@login_required(login_url='/account/login/')
@transaction.atomic
def save_asset_form(request, form, template_name, formset):
    data = dict()
    if request.method == 'POST':
        if form.is_valid() and formset.is_valid():
            asset = form.save(commit=False)
            asset.audit(request)
            asset.save()
            assetDocument.objects.filter(proof_of=asset).delete()
            data['form_is_valid'] = True
            for document_form in formset:
                name = document_form.cleaned_data['document_name']
                file = document_form.cleaned_data['file']

```

```

document = assetDocument.objects.create(proof_of=asset, document_name=name,
file=file)
document.audit(request)
document.save()
assets = Asset.objects.all()
data['html_product_list'] =
render_to_string('assetmanager/includes/partial_asset_list.html', {'asset_list': assets})
else:
data['form_is_valid'] = False
context = {'form': form, 'formset': formset}
data['html_form'] = render_to_string(template_name, context, request=request)
return JsonResponse(data)

```

```

# Function-based view for creating a new asset

```

```

@login_required(login_url='/account/login/')

```

```

@transaction.atomic

```

```

def createAsset(request):

```

```

    modelformset = modelformset_factory(assetDocument, assetDocumentForm)

```

```

    if request.method == 'POST':

```

```

        form = assetForm(request, request.POST)

```

```

        documentformset = modelformset(data=request.POST, files=request.FILES)

```

```

    else:

```

```

        form = assetForm(request)

```

```

        documentformset = modelformset()

```

```

    return save_asset_form(request, form, 'assetmanager/includes/partial_asset_create.html',
documentformset)

```

```

# Function-based view for updating an existing asset

```

```

@login_required(login_url='/account/login/')

```

```

@transaction.atomic

```

```

def asset_update(request, id):

```

```

modelformset = modelformset_factory(assetDocument, assetDocumentForm, extra=1)
asset = get_object_or_404(Asset, id=id)
documentobjects = assetDocument.objects.filter(proof_of=asset)
documents = [{'name': document.document_name, 'file': document.file} for document in
documentobjects]
if request.method == 'POST':
    form = assetForm(request=request, data=request.POST, instance=asset)
    documentformset = modelformset(data=request.POST, files=request.FILES)
else:
    documentformset = modelformset(initial=documents)
    form = assetForm(request=request, instance=asset)
return save_asset_form(request, form, 'assetmanager/includes/partial_asset_update.html',
documentformset)

```

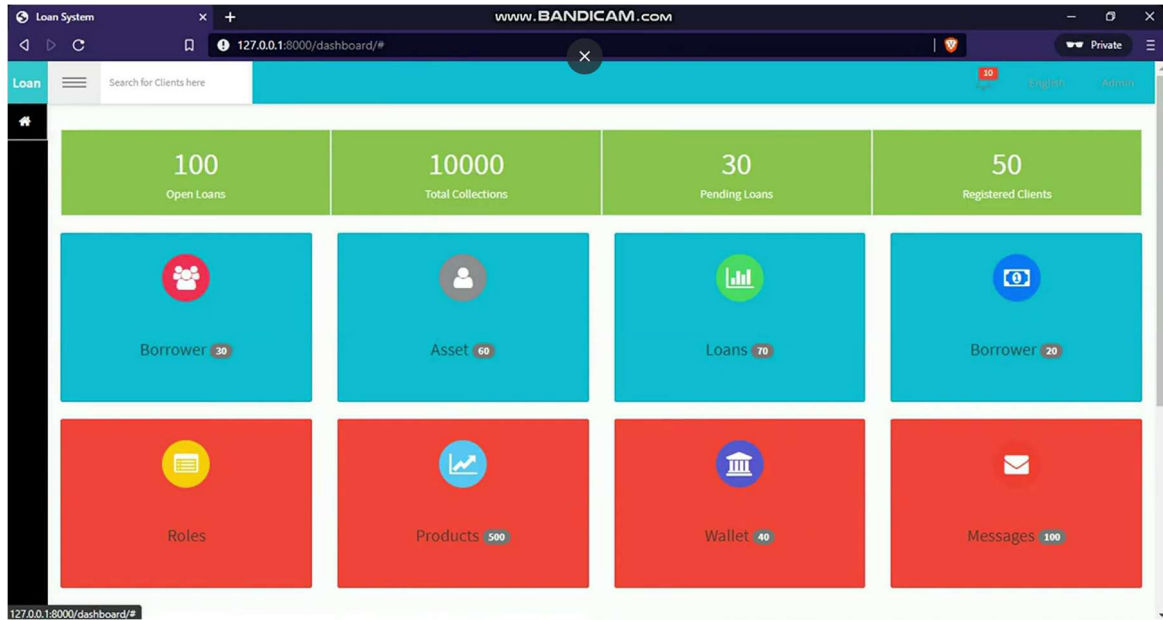
```

# Function-based view for deleting an asset
@login_required(login_url='/account/login/')
def asset_delete(request, id):
    asset = get_object_or_404(Asset, id=id)
    data = dict()
    if request.method == 'POST':
        assetDocument.objects.filter(proof_of=asset).delete()
        asset.delete()
        data['form_is_valid'] = True
        assets = Asset.objects.all()
        data['html_product_list'] =
render_to_string('assetmanager/includes/partial_asset_list.html', {'asset_list': assets})
    else:
        context = {'asset': asset}
        data['html_form'] = render_to_string('assetmanager/includes/partial_asset_delete.html',
context, request=request)
    return JsonResponse(data)

```

6.5 Screen Shots

Home Page



Loan Management List

Loan System

www.BANDICAM.COM

127.0.0.1:8000/dashboard/

Private

Roles

Products 500

Wallet 40

Messages 100

Recent Loans

Status	Clients	Orders#	Date
Paid Active	Adones Evangelista	7556588	Today 8:30
Paid Active	Jude Angel Suares	7556588	Today 16:30
Paid Inactive	Kimmy Matillano	7556588	Yesterday
Unpaid Inactive	Saxon Ong	7556588	23th May
Paid Active	Niko Curaza	7556588	Today 8:30
Paid Active	Adrian Mercurio	7556588	Today 16:30

© Copyright 2021. Loan Management System

Creating New Asset

The screenshot shows a web browser window with the URL `127.0.0.1:8000/asset/` and the title `www.BANDICAM.COM`. The browser window displays a web application titled "Loan Management System". A modal window titled "Create New Asset" is open, showing the following fields:

- Name: Myrel Flores
- Description: N/A
- Expiry date: 2021/12/10
- Inspection date: 2021/12
- Owner: (dropdown menu)

The background shows a sidebar with "Dashboard" and "Assets" (80 items), and a main area with "Add Assets +" and "Actions" (Edit, Delete).

The screenshot shows the same web browser window and modal window. The modal window now shows additional fields:

- Expiry date: 2021/12/10
- Inspection date: 2021/12/12
- Owner: user514
- Status: (dropdown menu)
- Value: (text input)
- Proof of Ownership: (text input) Choose File No file chosen
- enter document type: (text input) Choose File No file chosen
- Add More Remove Last

The background shows the same sidebar and main area.

CHAPTER - 7

TESTING

Software testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. Testing is a very important but an expensive activity. The purpose of testing can be quality assurance, verification and validation or reliability estimation.

Level of Testing:

Testing can be performed in following three levels.

- i. Unit testing
- ii. Integration testing
- iii. System testing

Unit Testing

Each program component is tested on its own, isolated from the other components in the system. Such testing, known as module testing, component testing or unit testing, verifies that the component functions properly with types of inputs expected from studying the component design.

Integrated Testing

When all collecting components have been unit tested, the next step is ensuring the interfaces among the components are defined and handled properly. Integration testing is the process of verifying that the system components work together as described in the system and program design specifications.

System Testing

System testing focuses on a complete, integrated system to evaluate compliance with specified requirements. Tests are made on characteristics that are only present when the entire system is run.

7.1 TEST CASE DESIGN

The design of tests for software and other engineered products can be as challenging as the initial design of the product itself. Yet, for reasons that we have already discussed, software engineers often treat testing as an afterthought, developing test cases that may "feel right" but have little assurance of being complete. Recalling the objectives of testing, we must design tests that have the highest likelihood of finding the most errors with a minimum amount of time and effort.

A rich variety of test case design methods have evolved for software. These methods provide the developer with a systematic approach to testing. More important, methods provide a mechanism that can help to ensure the completeness of tests and provide the highest likelihood for uncovering errors in software.

Any engineered product (and most other things) can be tested in one of two ways:

(1) Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function;

(2) knowing the internal workings of a product, tests can be conducted to ensure that "all gears mesh," that is, internal operations are performed according to specifications and all internal components have been adequately exercised.

7.2 Types of Testing

The first test approach is called black-box testing and the second, white-box testing. When computer software is considered, black-box testing alludes to tests that are conducted at the software interface. Although they are designed to uncover errors, black-box tests are used to demonstrate that software functions are operational, that input is properly accepted and output is correctly produced, and that the integrity of external information (e.g., a database) is maintained.

A black-box test examines some fundamental aspect of a system with little regard for the internal logical structure of the software.

White-box testing of software is predicated on close examination of procedural detail. Logical paths through the software are tested by providing test cases that exercise specific sets of conditions and/or loops. The "status of the program" may be examined at various points to determine if the expected or asserted status corresponds to the actual status.

WHITE-BOX TESTING: White-box testing, sometimes called glass-box testing, is a test case design method that uses the control structure of the procedural design to derive test cases. Using white-box testing methods, the software engineer can derive test cases that (1) guarantee that all independent paths within a module have been exercised at least once, (2) exercise all logical decisions on their true and false sides, (3) execute all loops at their boundaries and within their operational bounds, and (4) exercise internal data structures to ensure their validity.

BLACK-BOX TESTING: Black-box testing, also called behavioral testing, focuses on the functional requirements of the software. That is, black-box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program. Black-box testing is not an alternative to white-box techniques. Rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods.

Black-box testing attempts to find errors in the following categories: (1) incorrect or missing functions, (2) interface errors, (3) errors in data structures or external data base access, (4) behavior or performance errors, and (5) initialization and termination errors.

Unlike white-box testing, which is performed early in the testing process, black-box testing tends to be applied during later stages of testing. Because black-box testing purposely disregards control structure, attention is focused on the information domain.

CHAPTER - 8

SYSTEM IMPLEMENTATION

After having the user acceptance of the new system developed, the implementation phase begins. Implementation is the stage of a project during which theory is turned into practice. The major steps involved in this phase are:

- Acquisition and Installation of Hardware and Software
- Conversion
- User Training
- Documentation

The hardware and the relevant software required for running the system must be made fully operational before implementation. The conversion is also one of the most critical and expensive activities in the system development life cycle. The data from the old system needs to be converted to operate in the new format of the new system. The database needs to be setup with security and recovery procedures fully defined.

During this phase, all the programs of the system are loaded onto the user's computer. After loading the system, training of the user starts. Main topics of such type of training are:

- How to execute the package?
- How to enter the data?
- How to process the data (processing details)?
- How to take out the reports?

After the users are trained about the computerized system, working has to shift from manual to computerized working. The process is called **Changeover**. The following strategies are followed for changeover of the system.

1. **Direct Changeover:** This is the complete replacement of the old system by the new system. It is a risky approach and requires comprehensive system testing and training.
2. **Parallel run:** In parallel run both the systems, i.e., computerized and manual, are executed simultaneously for certain defined period. The same data is processed by both the systems. This strategy is less risky but more expensive because of the following facts:
 - Manual results can be compared with the results of the computerized system.
 - The operational work is doubled.
 - Failure of the computerized system at the early stage does not affect the working organization, because the manual system continues to work, as it used to do.
3. **Pilot run:** In this type of run, the new system is run with the data from one or more of the previous periods for the whole or part of the system. The results are compared with the old system results. It is less expensive and risky than parallel run approach. This strategy builds the confidence and the errors are traced easily without affecting the operations.

CHAPTER - 9

DOCUMENTATION

The documentation of the system is also one of the most important activities in the system development life cycle. This ensures the continuity of the system. Generally following two types of documentations are prepared for any system.

- User or Operator Documentation
- System Documentation

9.1 Operational Document

The system documentation contains the details of system design, programs, their coding, system flow, data dictionary, process description, etc. This helps to understand the system and permit changes to be made in the existing system to satisfy new user needs.

9.2 User Manual

The user documentation is a complete description of the system from the user's point of view detailing how to use or operate the system. It also includes the major error messages likely to be encountered by the user. User documentation is information created to help user understand and use a product. It is also known as end-user documentation. The purpose of user documentation is to guide users on how to properly install, use, and/or troubleshoot a product. User documentation is written in simple language, assuming that customer don't know a thing about the technicalities of your project. It can be delivered to customer through a variety of different mediums. User documentation also known as end-user documentation, is any form of documentation intended for the end user of a product or a service. The purpose of the document to guide the user completely about use, and/or troubleshoot a product.

User Manual System:

This module provides the functionality for the users to choose their favorite movies and music according to their needs. Users of the system, must be provided the following functionalities mentioned below:

- Create an account
- Manage their account
- Log into the System
- Navigate to the search menu
- Select the particular item
- Read the details loan type.
- Select the particular customerID and customer Detail
- Get the Loan Id on the basis of customerId
- Verify their Login Credentials

CHAPTER - 10

CONCLUSION

In conclusion, the development and implementation of the loan management system have significantly improved our organization's efficiency and accuracy in handling loan applications and processing. By digitizing and automating various aspects of the loan lifecycle, we have streamlined processes, reduced manual errors, and enhanced customer experience. The system has provided us with valuable insights into our lending operations, enabling better decision-making and risk management.

Throughout the project, we have overcome several challenges, including integration complexities, data security concerns, and user adoption issues. However, through collaborative efforts and effective communication, we have successfully addressed these challenges and delivered a robust and reliable loan management solution.

Moving forward, it's essential to continue monitoring and evaluating the performance of the system, gathering feedback from users and stakeholders, and making necessary enhancements to meet evolving business requirements and industry standards. Additionally, investing in ongoing maintenance, updates, and staff training will ensure the system remains effective and sustainable in the long term.

CHAPTER - 11

BIBLIOGRAPHY & REFERENCES

- 1) PYTHON: <https://www.python.org/>
- 3) REACT: <https://react.dev/>
- 4) GEEKSFORGEEKS: <https://www.geeksforgeeks.org/>
- 5) W3SCHOOLS: <https://www.w3schools.com/>
- 6) DJANGO: <https://www.djangoproject.com/>
- 7) SKLEARN: <https://scikit-learn.org/stable/>
- 8) TUTORIALSPPOINT: <https://www.tutorialspoint.com/>