INSURANCE MANAGEMENT SYSTEM

Submitted in partial fulfillment of requirement for the award of the Degree

Bachelor of Computer Science

In the faculty of Computer Science of Bharathiar University, Coimbatore

Submitted by

S.JANANI

(Reg.No.2022K0130)

Under the guidance of

Mrs.N.Suganthi MCA., M. Phil., (Ph. D)

Lecturer , Department of Computer Science

Department of Computer Science

L.R.G GOVERNMENT ARTS COLLEGE FOR WOMEN

(Affiliated To Bharathiar University)

TIRUPUR-4

APRIL-2023

CERTIFICATE

CERTIFICATE

This is to certify that the project work entitled **“ INSURANCE MANAGEMENT SYSTEM '"** Submitted to Bharathiar University in partial fulfilled of the requirement for the award of the Degree Bachelor of computer science is a record of the original work done by **Ms.S.JANANI (Reg.No.2022K0130)** Under my supervisor and that project work has not formed the basis for the any Degree /Diploma /Association /Fellowship or similar title to any candidate of any university.

Internal Guide	**Head of the Department**
Mrs.N.Suganthi MCA., M. Phil., (Ph. D)	**Dr.R.PARIMALAMSc.,M.Phil.,Ph.D**

Viva-voce examination is held on ___________________L.R.G Government Arts College for Women, Tirupur-641604.

Internal Examiner	**External Examiner**

DECLARATION

DECLARATION

I hereby declare that the project work submitted to the **UG Department of the Computer Science, L.R.G. Government Arts College for Women, Tirupur** , affiliated to Bharathiar University, Coimbatore in the partial fulfillment of the required for the award of Bachelor of Computer Science is an original work done by me during the fourth semester.

Place: **Signature of the Candidate**

Date: **(S.JANANI)**

(Reg.No:2022K0130)

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

As first and foremost I would like to external my thankfulness to the almighty for blessing the work of my hands, I am grateful to my parents for continued encouragement that they had given to me.

I would like to external my profound gratitude and sincere thanks to **Dr.M.R.YEZHILI MA.,M.Phil.,Ph.D** Principal, L.R.G Government Arts College For Women , for the encouragement rendered to me during this project.

It's my privilege to thank **DR.R.PARIMALA MSc.,M.Phil.,Ph.D** Incharge Head of the department of computer science for her valuable guidance and support throughout the project development work.

I express my deep sense of gratitude and sincere thanks to my guide **Mrs.N.SUGANTHI MCA., M. Phil., (Ph. D).,**Lecturer, Department of computer science, for providing all sorts of facilities to complete my project work successfully.

I also extend my sincere thanks to all the other faculty member of the department and technical assistance for their co-operation and valuable guidance.

I thank our college library for providing me with many informative books that help me to enrich my knowledge to bring out the project successfully.

SYNOPSIS

SYNOPSIS

The project Insurance management system deals with insurance. This tool taking care of the policy in using tracking details of the customer. Customer can apply and view their insurance policy details. The objective of the application is to automate all the possible functionalities of insurance and provide the insurance service to the customers. Using this system agents and customers can know the detail about present policies, new policies, policy specifications, policy terms and conditions, etc. This system also calculates agent commission is based upon customer policy registration.

This system maintains profile management of all policy holders. This system providing interface to customer that helps to him to know his policy details. Payment process also like ecommerce transaction. Customers can pay their policies. The payment processes are hassle-free and customer can complete a transaction at a much lesser time.

Insurance is a concept which involves two parties; namely the insurer and the insured, also referred to as the policyholder. The insurer is the insurance company, whereas, the policyholder is the one who avails the service. In the insurance process, the policyholder has to pay certain amount of fees at prescribed time intervals to the insurance company. And in turn, the insurer agrees to bear the financial losses and expenses of the policyholder. Therefore, the risk of financial losses completely falls on the insurance company.

CONTENT

TABLE OF CONTENTS

INTRODUCTION

1. INTRODUCTION

1.1 ABOUT OF PROJECT

Insurance Management System is an application designed for a group of insurance agents, which supports the maintenance activities of Insurance Policies and their associated company details their procedures in maintaining various types of policies, with the corresponding details of policy holders, associated agents involved in the policy maintenance and etc. This web based system design maintains the details of the Companies offering various types of insurance policies, concerned agents and the detailed information of customers with their corresponding policies.

The design aims at providing the agents and the customers a facility to access the information of policies their descriptions, which include starting date, renewal date, premium payment, claims information, etc., under various categories like Life ,Motor, Health, Home insurance and the corresponding company details.

1.2. SYSTEM SPECIFICATION

System Requirements Specification also known as Software Requirements Specification, is a document or set of documentation that describes the features and behavior of a software application.

WINDOWS OS

Windows is a graphical operating system developed by Microsoft. It allows users to view and store files, run the software, play games, watch videos, and provides a way to connect to the internet. It was released for both home computing and professional works.

Microsoft introduced the first version as 1.0

It was released for both home computing and professional functions of Windows on 10 November 1983. Later, it was released on many versions of Windows as well as the current version, Windows 10.

In 1993, the first business-oriented version of Windows was released, which is known as Windows NT 3.1. Then it introduced the next versions, Windows 3.5, 4/0, and Windows 2000. When the XP Windows was released by Microsoft in 2001, the company designed its various versions for a personal and business environment. It was designed based on standard x86 hardware, like Intel and AMD processor. Accordingly, it can run on different brands of hardware, such as HP, Dell, and Sony computers, including home-built PCs.

Play Video

Editions of Windows

Microsoft has produced several editions of Windows, starting with Windows XP. These versions have the same core operating system, but some versions included advance features with an additional cost. There are two most common editions of Windows:

- Windows Home
- Windows Professional

Windows Home is basic edition of Windows. It offers all the fundamental functions of Windows, such as browsing the web, connecting to the Internet, playing video games, using office software, watching videos. Furthermore, it is less expensive and comes pre-installed with many new computers.

1.2.1 HARDWARE SPECIFICATION

➢ Processor	: P 4 700 GHz.
➢ RAM	: 4 GB RAM
➢ Hard Disk Drive	: 180 GB

1.2.2 SOFTWARE SPECIFICATION

➢ Operating System	: Windows 10
➢ Front End	: PYTHON
➢ Back End	: MYSQL

1.2.3 LANGUAGE DESCRIPTION

PYTHON

Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems. This versatility, along with its beginner-friendliness, has made it one of the most-used programming languages today. A survey conducted by industry analyst firm RedMonk found that it was the second-most popular programming language among developers in 2021.

Python is commonly used for developing websites and software, task automation, data analysis, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances.

"Writing programs is a very creative and rewarding activity," says University of Michigan and Courser instructor Charles R Severance in his book Python for Everybody. "You can write programs for many reasons, ranging from making your living to solving a difficult data analysis problem to having fun to helping someone else solve a problem."

MYSQL

MySQL is an Oracle-backed open source relational database management system (RDBMS) based on Structured Query Language (SQL). MySQL runs on virtually all platforms, including Linux, UNIX and Windows. Although it can be used in a wide range of applications, MySQL is most often associated with web applications and online publishing.

MySQL is an important component of an open source enterprise stack called LAMP. LAMP is a web development platform that uses Linux as the operating system, Apache as the web server, MySQL as the relational database management system and PHP as the object-oriented scripting language. (Sometimes Perl or Python is used instead of PHP.).

Originally conceived by the Swedish company MySQL AB, MySQL was acquired by Sun Microsystems in 2008 and then by Oracle when it bought Sun in 2010. Developers can use MySQL under the GNU General Public License (GPL), but enterprises must obtain a commercial license from Oracle.

Relational database management systems use structured query language (SQL) to store and manage data. The system stores multiple database tables that relate to each other. MS SQL Server, MySQL, or MS Access are examples of relational database management systems. The following are the components of such a system.

A SQL table is the basic element of a relational database. The SQL database table consists of rows and columns. Database engineers create relationships between multiple database tables to optimize data storage space.

SQL statements, or SQL queries, are valid instructions that relational database management systems understand. Software developers build SQL statements by using different SQL language elements. SQL language elements are components such as identifiers, variables, and search conditions that form a correct SQL statement.

SYSTEM STUDY

2. SYSTEM STUDY

2.1 EXISTING SYSTEM

The existing system is the manual system. It is time consuming. It is difficult to search for a data most of the insurance organizations are not having any existing fully computerized system. It is very difficult for a person to produce the report. There are chances for changing the scheme report and do malpractice. They are managing the information in the form of excel spread sheets. This system involves a lot of manual entries with the applications to perform the desired task. Every member organization has its own data structure. Usage of papers in the payment process leads to less efficiency, less accuracy and less productivity. Due to lack of centralized data structure, it is very difficult to merge the data to analyze the statistics.

2.1.1 DRAWBACKS

- ➢ Huge process of creating an insurance account.
- ➢ More number of papers are wasted for insurance.
- ➢ No records can be maintained.

2.2 PROPOSED SYSTEM

The proposed system is for making easier to manage policy holder details, agent details, policy details, claimant details and payment details. The proposed system is designed to eliminate the drawbacks of the existing system. It is designed by keeping to eliminate the drawbacks of the present system in order to provide a permanent solution to the problems.

The primary aim of the new system is to speedup transactions. The overall system is control through the main menu. The report is prepared for the schemes and implemented by the concerned officials.

2.2.1. FEATURES

- ➢ Can easily contact the agent.
- ➢ All the record are maintained by administrator.
- ➢ Easy to manage the software

SYSTEM DESIGN AND DEVELOPMENT

3. SYSTEM DESIGN AND DEVELOPMENT

3.1 FILE DESIGN

The selection of the file system design approach is done according to the needs of the developers what are the needed requirements and specifications for the new design. It allowed us to identify where our proposal fitted in with relation to current and past file system development. Our experience with file system development is limited so the research served to identify the different techniques that can be used. The variety of file systems encountered show what an active area of research file system development is. The file systems may be from one of the two fundamental categories. In one category, the file system is developed in user space and runs as a user process. Another file system may be developed in the kernel space and runs as a privileged process. Another one is the mixed approach in which we can take the advantages of both aforesaid approaches. Each development option has its own pros and cons. In this article, these design approaches are discussed.

A file system is the data structure designed to support the abstraction of the data blocks as an archive and collection of files. This data structure is unique because it is stored on secondary storage (usually the disk), which is a very slow device.

The file system structure is the most basic level of organization in an operating system. Almost all of the ways an operating system interacts with its users, applications, and security model are dependent upon the way it organizes files on storage devices.

File Design Information systems in business are file and database oriented. Data are accumulated into files that are processed or maintained by the system. The systems analyst is responsible for designing files, determining their contents and selecting a method for organizing the data.

The most important purpose of a file system is to manage user data. This includes storing, retrieving and updating data. Some file systems accept data for storage as a stream of bytes which are collected and stored in a manner efficient for the media.

3.2 INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

- Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
- It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
- When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user
- will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow.

3.3 OUTPUT DESIGN

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

External Outputs

Manufacturers create and design external outputs for printers. External outputs enable the system to leave the trigger actions on the part of their recipients or confirm actions to their recipient.

Some of the external outputs are designed as turnaround outputs, which are implemented as a form and re-enter the system as an input.

Internal outputs

Internal outputs are present inside the system, and used by end-users and managers. They support the management in decision making and reporting.

Output Integrity Controls

Output integrity controls include routing codes to identify the receiving system, and verification messages to confirm successful receipt of messages that are handled by network protocol.

Printed or screen-format reports should include a date/time for report printing and the data. Multipage reports contain report title or description, and pagination. Pre-printed forms usually include a version number and effective date.

3.4 DATABASE DESIGN

Today's businesses depend on their databases to provide information essential for day-to-day operations, especially in case of electronic commerce businesses who has a definite advantage with up-to-date database access. Good design forms the foundation of any database, and experienced hands are required in the automation process to design for optimum and stable performance.

Software Solutions have been constantly working on these platforms and have attained a level of expertise. We apply proven methodologies to design, develop, integrate and implement database systems to attain its optimum level of performance and maximize security to meet the client's business model.

Business needs addressed:

- Determine the basic objects about which the information is stored
- Determine the relationships between these groups of information and the objects
- Effectively manage data and create intelligent information
- Remote database administration or on site administrative support
- Database creation, management, and maintenance
- Information retrieval efficiency, remove data redundancy and ensure data security

The most important consideration in designing the database is how the information will be used. The main objective of designing a database is Data Integration, Data Integrity and Data Independence.

Data Integration

In a database, information from several files is coordinated, accessed and operated upon as through it is in a single file. Logically, the information is centralized, physically; the data may be located on different devices, connected through data communication facilities.

Data Integrity

Data integrity means storing all data in one place only and how each application accesses it. This approach results in more consistent information, one update being sufficient to achieve a new record status for all applications. This leads to less data redundancy that is data items need not be duplicated.

Data Independence

Data in dependence is the insulation of application programs from changing aspects of physical data organization. This objective seeks to allow changes in the content and organization of physical data without reprogramming of application and allow modifications to application programs without reorganizing the physical data.

3.5 SYSTEM DEVELOPMENT

Systems development is the process of defining, designing, testing, and implementing a new software application or program. It could include the internal development of customized systems, the creation of database systems, or the acquisition of third party developed software.

Systems development life cycle phases include planning, system analysis, system design, development, implementation, integration and testing, and operations and maintenance.

3.5.1 DESCRIPTION OF MODULES

Policy Registration Module

New policy holder gives their information like, name, age, sex, address, e-mail id. Customer can give their information to register the new policy account. The customer needs to give the details to apply their policy.

Customer account module

The customer account module is when the customer registers the form and can able to access login, here they can check the customer account details as well as policy renewal. Once the customer has been added then the agent can give the unique username and password to the customer. Customer can use the correct username, password and login the payment will be displayed customer can pay the amount.

Policy payment module

This module the customer can directly pay the insurance renewal for this module, once the date has crossed then the customer will get the alert for the policy details then can pay it. The customer can make policy payment while registration then the system generates the payment receipt after the payment confirmation.

Agent Registration module

Admin adds agent by verifying their profile manually. An agent can able to register by the administrator, once the agent has been added they will have unique username and password. Using this credential to login into the application. Once they register, they can able to create the customer account and policy registration. The agent work for insurance company who provide information regarding the policies and schemes and bring new customer to the insurance company.

Report module

Also, the agent has report to for them customer details as well as policy and renewal details. Once the process has been done, they will generate report to the respective insurance team. The all reports are to manage the administration. Admin can maintain the customer details, policy renewal details and policy payment details.

TESTING AND IMPLEMENTATION

4. TESTING AND IMPLEMENTATION

TESTING METHODOLOGIES

System testing is state of implementation, which is aimed at ensuring that the system works accurately and efficiently as expect before live operation commences. It certifies that the whole set of programs hang together.

System testing requires a test plan that consists of several key activities and step for run program, string, system and user acceptance testing. The implementation of newly designed package is important in adopting a successful new system

Testing is the important stage in software development. the system test in implementation stage in software development process. The system testing implementation should be confirmation that all is correct and an opportunity to show the users that the system works as expected. It accounts the largest percentage of technical effort in the software development process.

Testing phase in the development cycle validates the code against the functional specification testing is vital to achievement of the system goals. The objective of the testing is to discover errors to fulfill this objective a series of test step unit, integration. Validation and system tests were planned and executed the test steps are:

SYSTEM TESTING

Testing is an integral part of any system development life cycle. Insufficient and untested applications may tend to crash and the result is loss of economic and manpower investment besides user's dissatisfaction and downfall of reputation. Software testing can be looked upon as one among many processes, an organization performs, and that provides the lost opportunity to correct any flaws in the developed system. Software testing includes selecting test data that have more probability of giving errors.

The first step in system testing is to develop a plan that tests all aspects of the system. Completeness, correctness, reliability and maintainability of the software are to be tested for the best quality assurance that the system meets the specification and requirements for its intended use and performance. System testing is the most useful practical process of executing a program with the implicit intention of finding errors that make the program fails. System testing is done in three phases.

- Unit Testing
- Integration Testing
- Validation Testing

UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software the module. Using the detailed design and the process specification testing is done to registration by the user with in the boundary of the Login module. The login form receives the username and password details and validates the value with the database. If valid, the home page is displayed.

INTEGRATION TESTING

Integration Testing is the process of this activity can be considered as testing the design and hence module interaction. The primary objective of integration testing is to discover errors in the interfaces between the components. Login form and registration form are integrated and tested together. If the user is newly registered, the received details will be stored in the registration table. While logging in, the application will check for valid user name and password in the registration table and if valid the user is prompted for submitting complaints.

Data can be lost across an interface, one module can have adverse effect on another sub function when combined it may not produce the desired major functions. Integration testing is a systematic testing for constructing test to uncover errors associated within an interface.

The objectives taken from unit tested modules and a program structure is built for integrated testing. All the modules are combined and the test is made.

A correction made in this testing is difficult because the vast expenses of the entire program complicated the isolation of causes. In this integration testing step, all the errors are corrected for next testing process.

VALIDATION TESTING

Validation are independent procedures that are used together for checking that a product, service, or system meets requirements and specifications and that it fulfills its in purpose the actual result from the expected result for the complaint process. Select the complaint category of the complaint by user. The input given to various forms fields are validated effectively. Each module is tested independently. It is tested that the complaint module fields receive the correct input for the necessary details such as complaint category, complaint id, reference name, complaint description, and email for further process.

After the completion of the integrated testing, software is completely assembled as a package; interfacing error has been uncovered and corrected and a final series of software test validation begins.

Validation testing can be defined in many ways but a simple definition is that validation succeeds when the software function in a manner that can be reasonably expected by the customer. After validation test has been conducted, one of two possible conditions exists.

OUTPUT TESTING

The next process of validation testing, is output testing of the proposed system, since no system could be successful if it does not produce the required output in the specified format. Asking the user about the format required, list the output to be generated or displayed by the system under considerations.

Output testing is a different test whose primary purpose is to fully exercise the computer based system although each test has a different purpose all the work should verify that all system elements have been properly integrated and perform allocated functions.

The output format on the screen is found to be corrected as the format was designed in the system design phase according to the user needs for the hard copy also; the output testing has not resulted in any correction in the system.

SYSTEM IMPLEMENTATION

When the initial design was done for the system, the client was consulted for the acceptance of the design so that further proceedings of the system development can be carried on. After the development of the system a demonstration was given to them about the working of the system. The aim of the system illustration was to identify any malfunction of the system.
After the management of the system was approved the system implemented in the concern, initially the system was run parallel with existing manual system. The system has been tested with live data and has proved to be error free and user friendly.

Implementation is the process of converting a new or revised system design into an operational one when the initial design was done by the system; a demonstration was given to the end user about the working system.

This process is uses to verify and identify any logical mess working of the system by feeding various combinations of test data. After the approval of the system by both end user and management the system was implemented.

System implementation is made up of many activities. The six major activities are as follows.

CODING

Coding is the process of whereby the physical design specifications created by the analysis team turned into working computer code by the programming team. A design code may be a tool which helps ensure that the aspiration for quality and quantity for customers and their requirements, particularly for large scale projects, sought by the water agency Design pattern are documented tried and tested solutions for recurring problems in a given context. So basically you have a problem context and the proposed solution for the same.

INSTALLATION

Installation is the process during which the current system is replaced by the new system. This includes conversion of existing data, software, and documentation and work procedures to those consistent with the new system.

DOCUMENTATION

Documentation is descriptive information that describes the use and operation of the system. The user guide is provided to the end user as the student and administrator. The documentation part contains the details as follows,

User requirement and water agency details administration has been made online. Any customer can request their water requirement details through online and also use of documentation, they can view the purpose of each purpose, The admin could verify the authentication of the users, users requirements and need to take delivery process, thus the documentation is made of full view of project thus it gives the guideline to study the project and how to execute also.

USER TRAINING AND SUPPORT

The software is installed at the deployment environment, the developer will give training to the end user of the regional transport officer and police admin officer in that software. The goal of an end user training program is to produce a motivated user who has the skills needed to apply what has been to apply what has been learned to perform the job related task. The following are the instruction which is specified the handling and un-handling events in the application,

- The authenticated user of admin and office workers only login in the application with authorized username and password.
- Don’t make user waste their time to come straight to the water agency or make a phone call.
- It can easily track through online by the user.
- Very user friendliness software

IMPLEMENTATION PROCEDURES

Implementation includes all the activities that take place to convert the old system to the new one. Proper implementation is essential to provide a reliable system to meet the organization requirements. Implementation is the stage in the project where the theoretical design is turned into a working system. The most crucial stage is achieving a successful new system & giving the user confidence in that the new system will work efficiently & effectively in the implementation state.

IMPLEMENTATION PROCEDURES

PILOT RUNNING

Processing the current data by only one user at a time called the pilot running process. When one user is accessing the data at one system, the system is sets to be engaged and connected in network. This process is useful only in system where more than one user is restricted.

PARALLEL RUNNING:

Processing the current data by more than one user at a time simultaneously is said to be parallel running process. This same system can be viewed and accessed by more than one user at the time. Hence the implementation method used in the system is a pilot type of implementation.

Implementation is the stage in the project where the theoretical design is turned into a working system. The most crucial stage is achieving a successful new system & giving the user confidence in that the new system will work efficiently & effectively in the implementation state.

The stage consists of,

- Testing the developed program with sample data.
- Detection's and correction of error.
- Creating whether the system meets user requirements.
- Making necessary changes as desired by the user.

Training user personnel.

USER TRAINING

User Training is designed to prepare the user for testing &consenting the system. .

- User Manual.
- Help Screens.
- Training Demonstration.

USER MANUAL

The summary of important functions about the system and software can be provided as a document to the user.

HELP SCREENS

This features now available in every software package, especially when it is used with a menu. The user selects the "Help" option from the menu. The system accesses the necessary description or information for user reference.

TRAINING DEMONSTRATION:

Another User Training element is a Training Demonstration. Live demonstrations with personal contact are extremely effective for Training Users.

SYSTEM MAINTENANCE

Maintenance is actually the implementation of the review plan. As important as it is, many programmers and analysts are to perform or identify themselves with the maintenance effort. There are psychological, personality and professional reasons for this. Analysts and programmers spend far more time maintaining programs than they do writing them. Maintenance accounts for 50-80 percent of total system development

Maintenance is expensive. One way to reduce the maintenance costs are through maintenance management and software modification audits**.**

- Maintenance is not as rewarding as exciting as developing systems. It is perceived as requiring neither skill not experience.
- Users are not fully cognizant of the maintenance problem or its high cost.
- Few tools and techniques are available for maintenance.
- A good test plan is lacking.
- Standards, procedures, and guidelines are poorly defined and enforced.
- Programs are often maintained without care for structure and documentation.
- There are minimal standards for maintenance.
- Programmers expect that they will not be in their current commitment by time their programs go into the maintenance cycle.

Corrective Maintenance

It means repairing, processing or performance failure or making changes because of previously uncovered problems or false assumptions. Task performed to identify, isolate, and rectify a fault so that the failed equipment, machine, or system can be restored to an operational condition within the tolerances or limits established for in-service operations.

Corrective maintenance can be subdivided into "immediate corrective maintenance" (in which work starts immediately after a failure) and "deferred corrective maintenance" (in which work is delayed in conformance to a given set of maintenance rules).

Perfective Maintenance

It means changes made to a system to add new features or to improve performance. Preventive maintenance is predetermined work performed to a schedule with the aim of preventing the wear and tear or sudden failure of equipment components. process or control equipment failure can have adverse results in both human and economic terms. In addition to down time and the costs involved to repair and/or replace equipment parts or components, there is the risk of injury to operators, and of acute exposures to chemical and/or physical agents.

Time-based or run-based Periodically inspecting, servicing, cleaning, or replacing parts to prevent sudden failure .On-line monitoring of equipment in order to use important/expensive parts to the limit of their serviceable life. Preventive maintenance involves changes made to a system to reduce the chance of future system failure.

An example of preventive maintenance might be to increase the number of records that a system can process far beyond what is currently needed or to generalize how a system sends report information to a printer so that so that the system can adapt to changes in printer technology.

Preventive Maintenance

Changes made to a system to avoid possible future problems Perfective maintenance involves making enhancements to improve processing performance, interface usability, or to add desired, but not necessarily required, system features. The objective of perfective maintenance is to improve response time, system efficiency, reliability, or maintainability.

During system operation, changes in user activity or data pattern can cause a decline in efficiency, and perfective maintenance might be needed to restore performance. Usually, the perfective maintenance work is initiated by the IT department, while the corrective and adaptive maintenance work is normally requested by users.

CONCLUSION

5. CONCLUSION

Insurance Management Systems (IMS) are critical tools for insurance companies and agents to manage their operations, including policy administration, underwriting, claims processing, and customer management. These systems offer several benefits, including increased efficiency, enhanced accuracy, and improved customer satisfaction. IMS enable insurance companies to automate many of their processes, reducing the time and effort required for policy administration, underwriting, and claims processing. This leads to improved efficiency, reduced errors, and faster turnaround times. Moreover, IMS provide enhanced accuracy in policy administration, underwriting, and claims processing, reducing the risk of errors and fraudulent claims. This leads to improved profitability and reduced costs for insurance companies. IMS also provide improved customer satisfaction by providing customers with more convenient and efficient ways to manage their policies and claims. Customers can access policy information, submit claims, and communicate with insurance companies in real-time through web portals, mobile apps, and other digital channels. In conclusion, Insurance Management Systems are essential tools for insurance companies and agents looking to efficiently manage their operations and improve customer satisfaction. These systems offer several benefits, including increased efficiency, enhanced accuracy, and improved customer satisfaction. By leveraging these benefits, insurance companies can improve their profitability, reduce costs, and enhance their overall competitive advantage

BIBLIOGRAPHY

6. BIBLIOGRAPHY

Book Reference:

- Brownlee, Jason. Machine Learning Mastery with Python. Machine Learning Mastery, 2016.
- Lutz, Mark. Learning Python, 5th Edition. O'Reilly Media, 2013.
- McKinney, Wes. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media, 2017.
- Rossum, Guido van. "Python Programming Language." Python.org, https://www.python.org/.
- Ramalho, Luciano. Fluent Python: Clear, Concise, and Effective Programming. O'Reilly Media, 2015.
- Reitz, Kenneth, and Schlusser, Tanya. Flask Web Development: Developing Web Applications with Python. O'Reilly Media, 2018.

Website:

- https://www.python.org/doc/.
- https://www.tutorialspoint.com/python/index.htm.
- https://www.w3schools.com/python/.
- https://www.codecademy.com/learn/learn-python.
- https://pypi.org/.

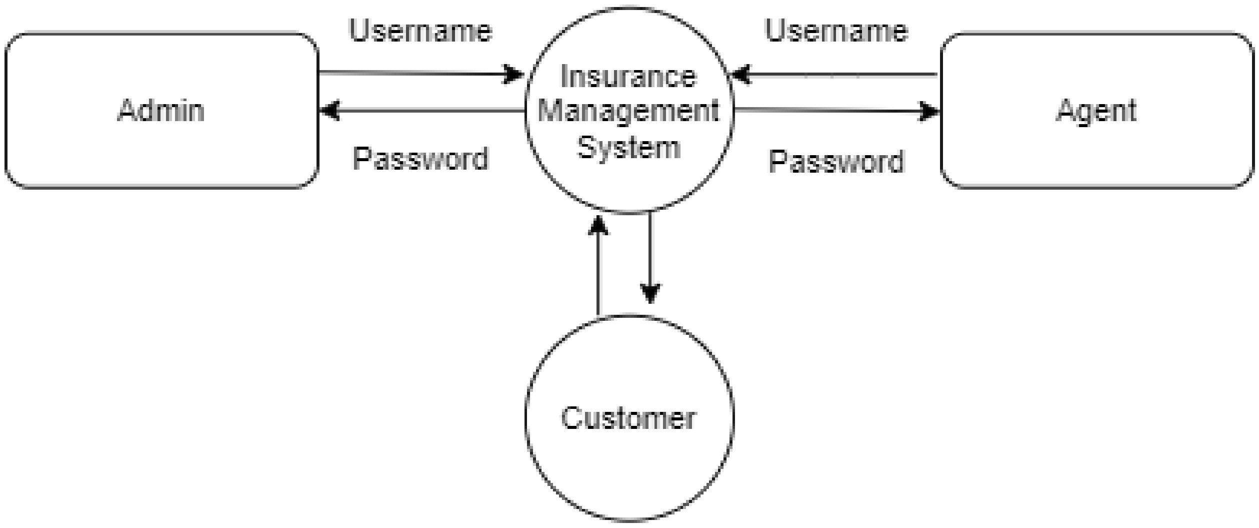
APPENDICES

7. APPENDICES

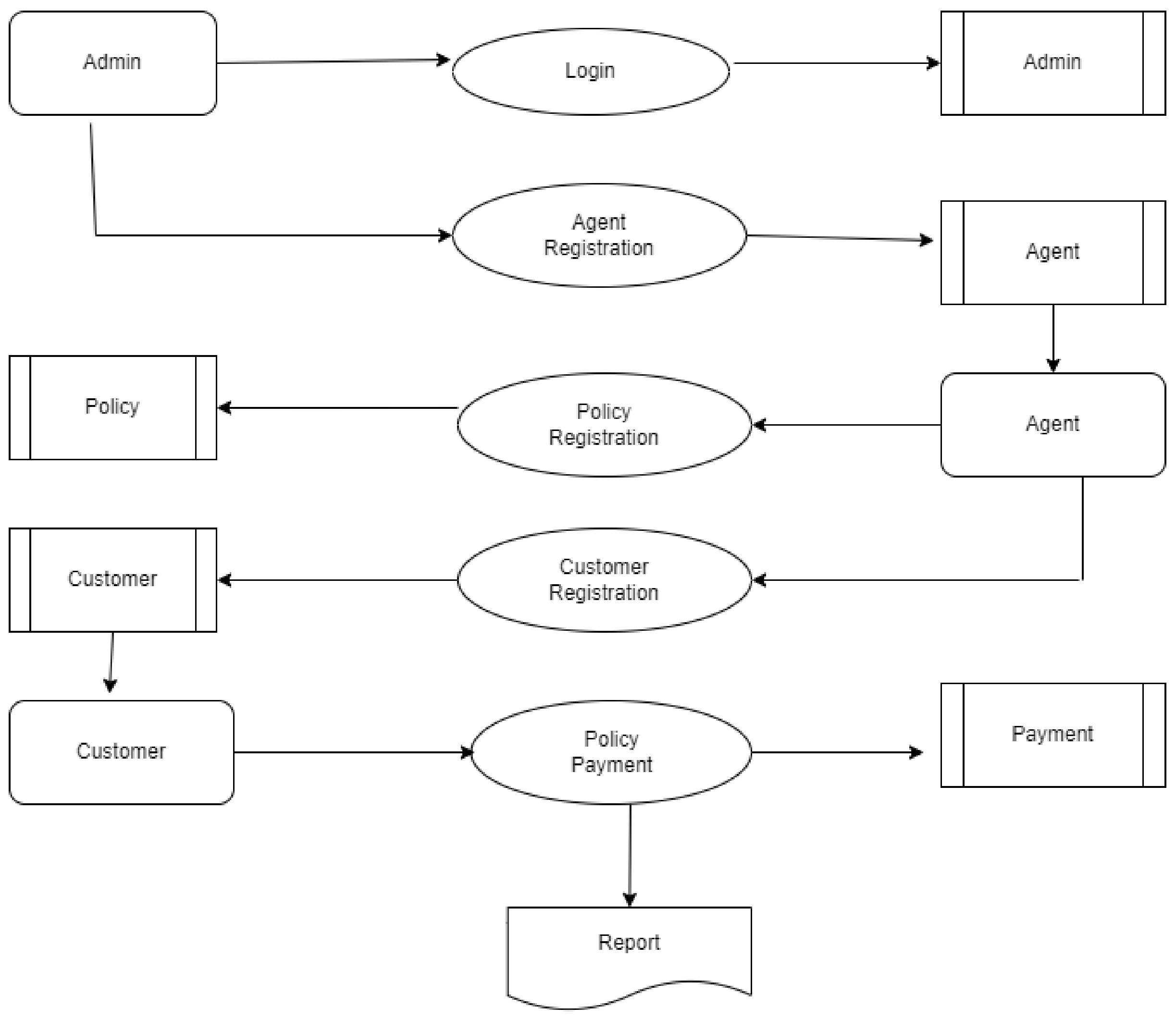
A. DATA FLOW DIAGRAM

A data-flow diagram (DFD)is a way of representing a flow of a data of a process or system. The DFD also provides information about the outputs and inputs of each entity and process itself. A data-flow diagram is a part of structured-analysis modeling tools.

LEVEL 0:



LEVEL 1:


Admin
Login
Admin
Agent Registration
Agent
Policy
Policy Registration
Agent
Customer
Customer Registration
Customer
Policy Payment
Payment
Report

B.TABLE STRUCTURE

The table needed for each module was designed and the specification of each and every column was given based on the records and details collected during record specification of the system study.

TABLE NAME: ADMIN

FIELD	DATA TYPE	SIZE	CONSTRAINT
Admin id	int	10	Primary key
Username	Varchar	20	Not null
password	Varchar	20	Not null

TABLE NAME: AGENT

FIELD	DATA TYPE	SIZE	CONSTRAINT
Agent id	Int	5	Primary key
First name	Varchar	10	Not null
Last name	Varchar	10	Not null
D.O.B	Date	8	Not null
Gender	Varchar	5	Not null
Mobile no	Int	10	Not null
Age	Int	2	Not null
State	Varchar	20	Not null
City	Varchar	20	Not null
Pincode	Int	7	Not null
Address	Varchar	30	Not null
Experience	Int	2	Not null
Password	Varchar	5	Not null

TABLE NAME: CUSTOMER

FIELD	DATA TYPE	SIZE	CONSTRAINT
Customer id	Int	5	Primary key
First name	Varchar	10	Not null
Last name	Varchar	10	Not null
D.O.B	Date	8	Not null
Gender	Varchar	7	Not null
Age	Int	2	Not null
Mobile.no	Int	10	Not null
State	Varchar	15	Not null
City	Varchar	20	Not null
Pincode	Int	7	Not null
Address	Varchar	20	Not null
Occupation	Varchar	10	Not null
Agent id	Int	5	Not null

TABLE NAME: POLICY

FIELD	DATA TYPE	SIZE	CONSTRAINT
Policy id	Int	5	Not null
Policy name	Varchar	20	Not null
Policy category	Varchar	20	Not null
Policy duration	Int	2	Not null
Policy status	Varchar	10	Not null
Policy date	Date	8	Not null
Policy expiry date	Date	8	Not null
Amount	Int	7	Not null

TABLE NAME: PAYMENT

FIELD	**DATA TYPE**	**SIZE**	**CONSTRAINT**
Payment id	Int	5	Primary key
Policy id	Int	5	Foreign key
Customer id	Int	5	Foreign key
Agent id	Int	5	Foreign key
Last payment	Int	7	Not null
Remaining amount	Int	7	Not null

C.SAMPLE CODING

```
#!/usr/bin/env python

# package: com.example.demo.controller

import python.util.List

import org.springframework.beans.factory.annotation.Autowired

import org.springframework.web.bind.annotation.GetMapping

import org.springframework.web.bind.annotation.PathVariable

import org.springframework.web.bind.annotation.PostMapping

import org.springframework.web.bind.annotation.RequestMapping

import org.springframework.web.bind.annotation.RestController

import com.example.demo.dao.ApiDao

import com.example.demo.service.ApiService

@RequestMapping(value="/api")

class ApiController(object):

    """ generated source for class ApiController """

    service = ApiService()

    dao = ApiDao()


 @GetMapping("/login/{username}/{password}")

    def login(self, username, password):

        """ generated source for method login """
        return self.service.login(username, password)



    @PostMapping("/agent_register/{firstname}/{lastname}/{dob}/{gender}/{emailid}/"                    +
"{mobile}/{age}/{state}/{city}/{pincode}/{address}/{profession}/{expierence}/{password}")

    def member_register(self, firstname, lastname, dob, gender, emailid, mobile, age, state, city,
pincode, address, profession, expierence, password):

        """ generated source for method member_register """

        self.dao.add_agent(firstname, lastname, dob, gender, emailid, mobile, age, state, city, pincode,
address, profession, expierence, password)

        return "Member Register Sucessfully"

@PostMapping("/policy_register/{name}/{category}/{period}/{premium}/{date}/"                           +
"{expierydate}/{amount}")

    def add_policy(self, name, category, period, premium, date, expierydate, amount):

        """ generated source for method add_policy """

        self.dao.add_policy(name, category, period, premium, date, expierydate, amount)

        return "Policy Register Sucessfully"

   @PostMapping("/add_customer/{fname}/{lname}/{dob}/{mobile}/{age}/"                                  +
"{gender}/{state}/{city}/{pincode}/{address}/{username}/{password}")

    def add_customer(self, fname, lname, dob, mobile, age, gender, state, city, pincode, address,
username, password):

        """ generated source for method add_customer ""”

  self.dao.add_customer(fname, lname, dob, mobile, age, gender, state, city, pincode, address,
username, password)

        return "Policy Register Sucessfully"

@overloaded

    @PostMapping("/add_payment/{policy_id}/{number}/{expdate}/{amount}")

    def add_payment(self, policy_id, number, expdate, amount):
        """ generated source for method add_payment """
        self.dao.add_payment(policy_id, number, expdate, amount)
        return "Policy Register Sucessfully"


    @PostMapping("/add_payment/{policy_id}/{customer_id}")
    @add_payment.register(object, int, int)
    def add_payment_0(self, policy_id, customer_id):
        """ generated source for method add_payment_0 """
        self.dao.add_payment(policy_id, customer_id)
        return "Payment Sucessfully Completed"
    @GetMapping("/get_report")
    def get_report(self):
        """ generated source for method get_report """
        return self.dao.get_report()


  #!/usr/bin/env python
# package: com.example.demo.dao
import python.text.DateFormat
import python.text.SimpleDateFormat
import python.util.Date
import python.util.List
import pythonx.transaction.Transactional
import org.hibernate.Session
import org.hibernate.SessionFactory
import org.hibernate.query.NativeQuery
import org.springframework.beans.factory.annotation.Autowired

import org.springframework.stereotype.Repository

class ApiDao(object):

    """ generated source for class ApiDao """

    sf = SessionFactory()



    def get_complaints(self, id):

        """ generated source for method get_complaints """

        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()

        sql = "SELECT * FROM `complaint` as c left join citizen as cz on (cz.id=c.citizen_id) where
c.citizen_id=" + id

        nq = session.createNativeQuery(sql)

        return nq.list_()

    def login(self, username, password):

        """ generated source for method login """

        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()

        sql = "select * from admin where username='" + username + "' and password='" + password + "'"

        nq = session.createNativeQuery(sql)

        if nq.list_().size() != 0:

return "admin"

        else:

            if nq1.list_().size() != 0:

                return "manager"
            else:
                if nq2.list_().size() != 0:
                    return "id=" + a.get(0)[0]
                    # return "customer";
                else:
                    return "Invalid"


    def add_agent(self, firstname, lastname, dob, gender, emailid, mobile, age, state, city, pincode, address, profession, expierence, password):
        """ generated source for method add_agent """
        #  TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "INSERT INTO `agent` (`id`, `fname`, `lname`, `dob`, `gender`, `email`, `mobile`, `age`, `state`, `city`, `pincode`, `address`, `profession`, `experience`, `password`) VALUES " + "(NULL, '" + firstname + "', '" + lastname + "', '" + dob + "', '" + gender + "', '" + emailid + "', '" + mobile + "'," + " '" + age + "', '" + state + "', '" + city + "', '" + pincode + "', '" + address + "', '" + profession + "', '" + expierence + "', '" + password + "');"
        session.createSQLQuery(sql).executeUpdate()


    def add_policy(self, name, category, period, premium, date, expierydate, amount):
        """ generated source for method add_policy """
        #  TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "INSERT INTO `policy` (`id`, `name`, `category`, `period`, `premium`, `date`, `expierydate`, `amount`) VALUES " + "(NULL, '" + name + "', '" + category + "', '" + period + "', '" + premium + "', '" + date + "', '" + expierydate + "', '" + amount + "');"
        session.createSQLQuery(sql).executeUpdate()
    def add_customer(self, fname, lname, dob, mobile, age, gender, state, city, pincode, address,
username, password):
        """ generated source for method add_customer """
        #  TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "INSERT INTO `customer` (`id`, `fname`, `lname`, `dob`, `mobile`, `age`, `gender`, `state`,
`city`, `pincode`, `address`, `username`, `password`) VALUES " + "(NULL, '" + fname + "', '" + lname + "',
'" + dob + "', '" + mobile + "', '" + age + "', '" + gender + "', " + "'" + state + "', '" + city + "', '" + pincode +
"', '" + address + "','" + username + "','" + password + "');"
        session.createSQLQuery(sql).executeUpdate()
    @overloaded
    def add_payment(self, policy_id, number, expdate, amount):
        """ generated source for method add_payment """
        #  TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "INSERT INTO `payment` (`id`, `policy_id`, `policy_number`, `exp_date`, `amount`) VALUES "
+ "(NULL," + policy_id + ", '" + number + "', '" + expdate + "', '" + amount + "');"
        session.createSQLQuery(sql).executeUpdate()


    @add_payment.register(object, int, int)
    def add_payment_0(self, policy_id, customer_id):
        """ generated source for method add_payment_0 """
        #  TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "INSERT INTO `payment` (`id`, `policy_id`, `customer_id`) VALUES " + "(NULL," + policy_id +
", '" + customer_id + "');"
```

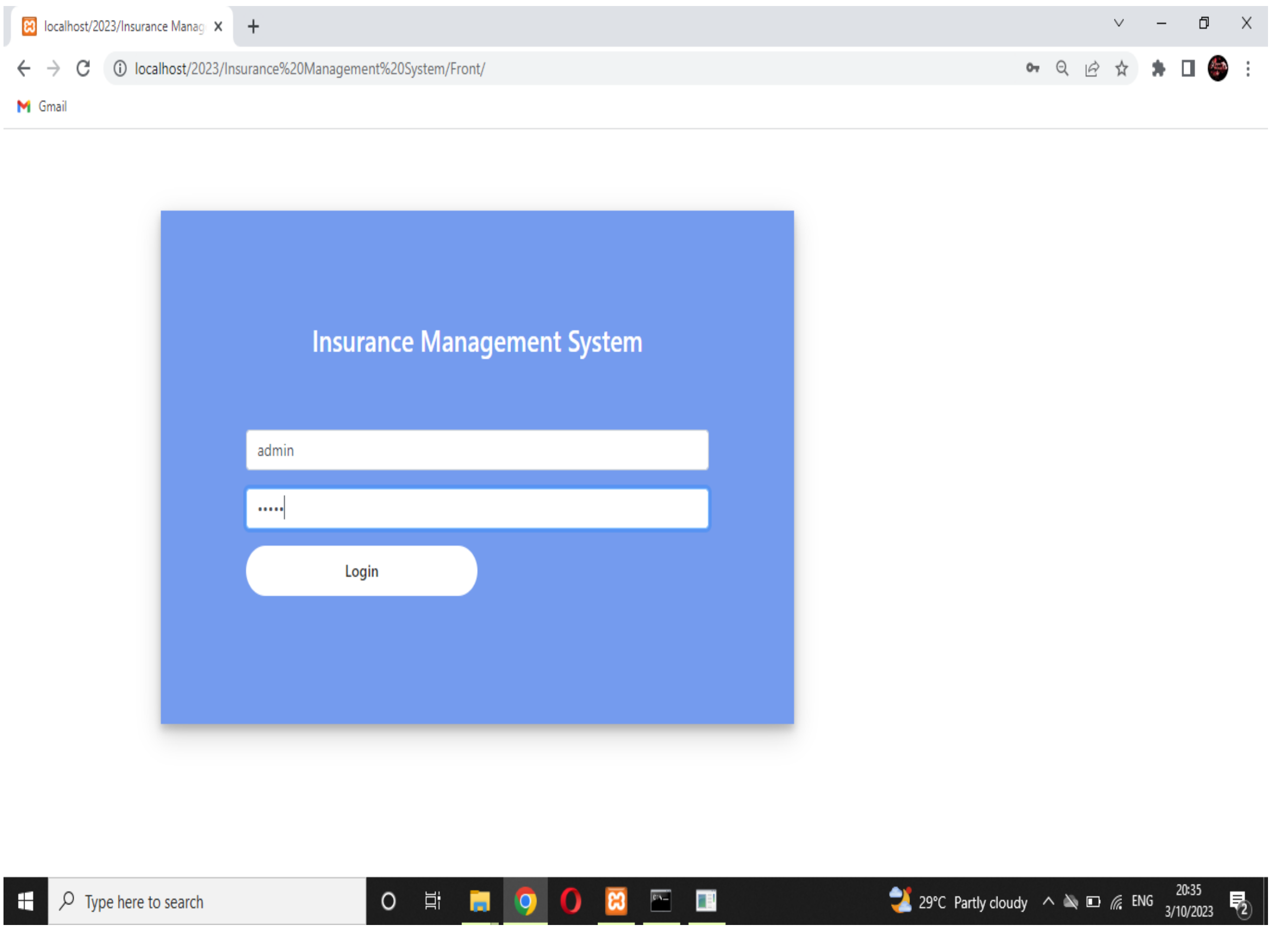
```
        session.createSQLQuery(sql).executeUpdate()


    def get_report(self):

        """ generated source for method get_report """

        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()

        sql = "SELECT policy.id, customer.fname, customer.lname, policy.name, policy.amount,
policy.expierydate  from payment LEFT JOIN customer ON (customer.id=payment.customer_id) LEFT
JOIN policy ON (policy.id=payment.policy_id);"

        nq = session.createNativeQuery(sql)

        return nq.list_()



        #!/usr/bin/env python

# package: com.example.demo.configuration

import python.util.Properties

import pythonx.sql.DataSource

import org.springframework.beans.factory.annotation.Value

import org.springframework.context.annotation.Bean

import org.springframework.context.annotation.Configuration

import org.springframework.jdbc.datasource.DriverManagerDataSource

import org.springframework.orm.hibernate5.HibernateTransactionManager

import org.springframework.orm.hibernate5.LocalSessionFactoryBean

import org.springframework.transaction.annotation.EnableTransactionManagement

@Value("${db.driver}")

@Value("${db.password}")

@Value("${db.url}")
```

```
@Value("${db.username}")

@Value("${hibernate.dialect}")

@Value("${hibernate.show_sql}")

@Value("${entitymanager.packagesToScan}")

class HibernateConfiguration(object):

    """ generated source for class HibernateConfiguration """

    DB_DRIVER = str()

    DB_PASSWORD = str()

    DB_URL = str()

    DB_USERNAME = str()

    HIBERNATE_DIALECT = str()

    HIBERNATE_SHOW_SQL = str()



    #     @Value("${hibernate.hbm2ddl.auto}")

    HIBERNATE_HBM2DDL_AUTO = str()

    ENTITYMANAGER_PACKAGES_TO_SCAN = str()



    def sessionFactory(self):

        """ generated source for method sessionFactory """

        sessionFactory = LocalSessionFactoryBean()

        sessionFactory.setDataSource(dataSource())

        sessionFactory.setPackagesToScan(self.ENTITYMANAGER_PACKAGES_TO_SCAN)

        hibernateProperties = Properties()

        hibernateProperties.put("hibernate.dialect", self.HIBERNATE_DIALECT)

        hibernateProperties.put("hibernate.show_sql", self.HIBERNATE_SHOW_SQL)
        #        hibernateProperties.put("hibernate.hbm2ddl.auto", HIBERNATE_HBM2DDL_AUTO);

        sessionFactory.setHibernateProperties(hibernateProperties)

        return sessionFactory


    def dataSource(self):

        """ generated source for method dataSource """

        dataSource = DriverManagerDataSource()

        dataSource.setDriverClassName(self.DB_DRIVER)

        dataSource.setUrl(self.DB_URL)

        dataSource.setUsername(self.DB_USERNAME)

        dataSource.setPassword(self.DB_PASSWORD)

        return dataSource


    def transactionManager(self):

        """ generated source for method transactionManager """

        txManager = HibernateTransactionManager()

        txManager.setSessionFactory(self.sessionFactory().getObject())

        return txManager


          #!/usr/bin/env python

# package: com.example.demo.dao

import python.text.DateFormat

import python.text.SimpleDateFormat

import python.util.Date

import python.util.List
import pythonx.transaction.Transactional

import org.hibernate.Session

import org.hibernate.SessionFactory

import org.hibernate.query.NativeQuery



import org.springframework.beans.factory.annotation.Autowired

import org.springframework.stereotype.Repository



class ApiDao(object):

    """ generated source for class ApiDao """

    sf = SessionFactory()



    def get_complaints(self, id):

        """ generated source for method get_complaints """

        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()

        sql = "SELECT * FROM `complaint` as c left join citizen as cz on (cz.id=c.citizen_id) where
c.citizen_id=" + id

        nq = session.createNativeQuery(sql)

        return nq.list_()



    def login(self, username, password):

        """ generated source for method login """

        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()
        sql = "select * from admin where username='" + username + "' and password='" + password + "'"

        nq = session.createNativeQuery(sql)

        if nq.list_().size() != 0:

            return "admin"

        else:

            if nq1.list_().size() != 0:

                return "manager"

            else:

                if nq2.list_().size() != 0:

                    return "id=" + a.get(0)[0]

                    # return "customer";

                else:

                    return "Invalid"

def add_agent(self, firstname, lastname, dob, gender, emailid, mobile, age, state, city, pincode, address, profession, expierence, password):

        """ generated source for method add_agent """

        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()

        sql = "INSERT INTO `agent` (`id`, `fname`, `lname`, `dob`, `gender`, `email`, `mobile`, `age`, `state`, `city`, `pincode`, `address`, `profession`, `experience`, `password`) VALUES " + "(NULL, '" + firstname + "', '" + lastname + "', '" + dob + "', '" + gender + "', '" + emailid + "', '" + mobile + "'," + " '" + age + "', '" + state + "', '" + city + "', '" + pincode + "', '" + address + "', '" + profession + "', '" + expierence + "', '" + password + "');"

        session.createSQLQuery(sql).executeUpdate()



    def add_policy(self, name, category, period, premium, date, expierydate, amount):

        """ generated source for method add_policy """
        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()

        sql = "INSERT INTO `policy` (`id`, `name`, `category`, `period`, `premium`, `date`, `expierydate`,
`amount`) VALUES " + "(NULL, '" + name + "', '" + category + "', '" + period + "', '" + premium + "', '" +
date + "', '" + expierydate + "', '" + amount + "');"

        session.createSQLQuery(sql).executeUpdate()


    def add_customer(self, fname, lname, dob, mobile, age, gender, state, city, pincode, address,
username, password):

        """ generated source for method add_customer """

        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()

        sql = "INSERT INTO `customer` (`id`, `fname`, `lname`, `dob`, `mobile`, `age`, `gender`, `state`,
`city`, `pincode`, `address`, `username`, `password`) VALUES " + "(NULL, '" + fname + "', '" + lname + "',
'" + dob + "', '" + mobile + "', '" + age + "', '" + gender + "', " + "'" + state + "', '" + city + "', '" + pincode +
"', '" + address + "','" + username + "','" + password + "');"

        session.createSQLQuery(sql).executeUpdate()


    @overloaded

    def add_payment(self, policy_id, number, expdate, amount):

        """ generated source for method add_payment """

        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()

        sql = "INSERT INTO `payment` (`id`, `policy_id`, `policy_number`, `exp_date`, `amount`) VALUES "
+ "(NULL," + policy_id + ", '" + number + "', '" + expdate + "', '" + amount + "');"

        session.createSQLQuery(sql).executeUpdate()
    @add_payment.register(object, int, int)

    def add_payment_0(self, policy_id, customer_id):

        """ generated source for method add_payment_0 """

        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()

        sql = "INSERT INTO `payment` (`id`, `policy_id`, `customer_id`) VALUES " + "(NULL," + policy_id +
", '" + customer_id + "');"

        session.createSQLQuery(sql).executeUpdate()


    def get_report(self):

        """ generated source for method get_report """

        #  TODO Auto-generated method stub

        session = self.sf.getCurrentSession()

        sql = "SELECT policy.id, customer.fname, customer.lname, policy.name, policy.amount,
policy.expierydate  from payment LEFT JOIN customer ON (customer.id=payment.customer_id) LEFT
JOIN policy ON (policy.id=payment.policy_id);"

        nq = session.createNativeQuery(sql)

        return nq.list_()


mport python.util.List

import org.springframework.beans.factory.annotation.Autowired

import org.springframework.web.bind.annotation.GetMapping

import org.springframework.web.bind.annotation.PathVariable

import org.springframework.web.bind.annotation.PostMapping

import org.springframework.web.bind.annotation.RequestMapping

import org.springframework.web.bind.annotation.RestController
import com.example.demo.dao.ApiDao

import com.example.demo.service.ApiService



@RequestMapping(value="/api")

class ApiController(object):

    """ generated source for class ApiController """

    service = ApiService()

    dao = ApiDao()



    @GetMapping("/login/{username}/{password}")

    def login(self, username, password):

        """ generated source for method login """

        return self.service.login(username, password)

    @PostMapping("/agent_register/{firstname}/{lastname}/{dob}/{gender}/{emailid}/"                  +
"{mobile}/{age}/{state}/{city}/{pincode}/{address}/{profession}/{expierence}/{password}")

    def member_register(self, firstname, lastname, dob, gender, emailid, mobile, age, state, city,
pincode, address, profession, expierence, password):

        """ generated source for method member_register """

        self.dao.add_agent(firstname, lastname, dob, gender, emailid, mobile, age, state, city, pincode,
address, profession, expierence, password)

        return "Member Register Sucessfully"



    @PostMapping("/policy_register/{name}/{category}/{period}/{premium}/{date}/"                  +
"{expierydate}/{amount}")

    def add_policy(self, name, category, period, premium, date, expierydate, amount):

        """ generated source for method add_policy """
        self.dao.add_policy(name, category, period, premium, date, expierydate, amount)
        return "Policy Register Sucessfully"


    @PostMapping("/add_customer/{fname}/{lname}/{dob}/{mobile}/{age}/" +
"{gender}/{state}/{city}/{pincode}/{address}/{username}/{password}")
    def add_customer(self, fname, lname, dob, mobile, age, gender, state, city, pincode, address,
username, password):
        """ generated source for method add_customer """
        self.dao.add_customer(fname, lname, dob, mobile, age, gender, state, city, pincode, address,
username, password)
        return "Policy Register Sucessfully"


    @overloaded
    @PostMapping("/add_payment/{policy_id}/{number}/{expdate}/{amount}")
    def add_payment(self, policy_id, number, expdate, amount):
        """ generated source for method add_payment """
        self.dao.add_payment(policy_id, number, expdate, amount)
        return "Policy Register Sucessfully"


    @PostMapping("/add_payment/{policy_id}/{customer_id}")
    @add_payment.register(object, int, int)
    def add_payment_0(self, policy_id, customer_id):
        """ generated source for method add_payment_0 """
        self.dao.add_payment(policy_id, customer_id)
        return "Payment Sucessfully Completed"
```

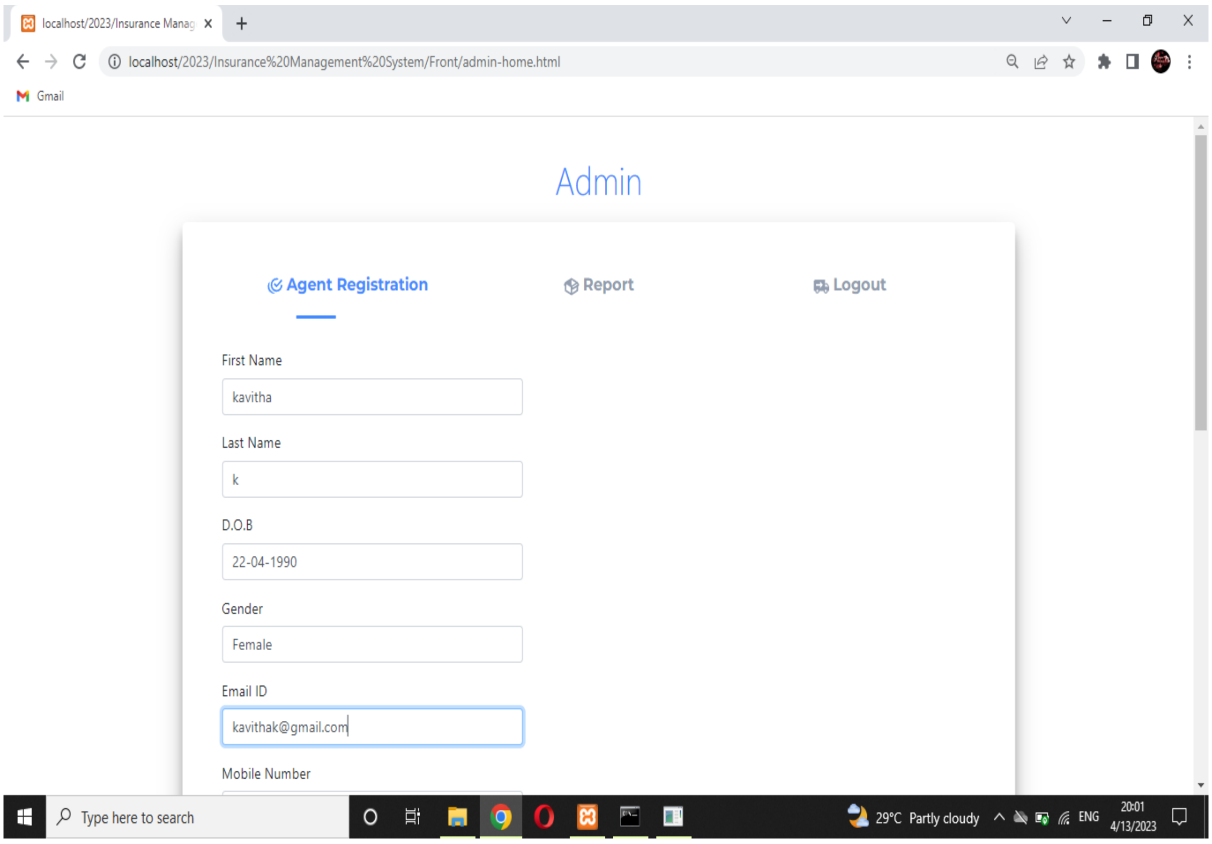
```
@GetMapping("/get_report")

def get_report(self):

    """ generated source for method get_report """

    return self.dao.get_report()
```

D.SAMPLE INPUT & OUTPUT DESIGN

ADMIN LOGIN PAGE:


localhost/2023/Insurance Manag
localhost/2023/Insurance%20Management%20System/Front/
Gmail
Insurance Management System
admin
Login
Type here to search
29°C Partly cloudy
ENG
20:35
3/10/2023

AGENT REGISTRATION PAGE:


localhost/2023/Insurance Manag
localhost/2023/Insurance%20Management%20System/Front/admin-home.html
Gmail
Admin
Agent Registration
Report
Logout
Type here to search
29°C Partly cloudy
ENG
20:01
4/13/2023

First Name

kavitha

Last Name

k

D.O.B

22-04-1990

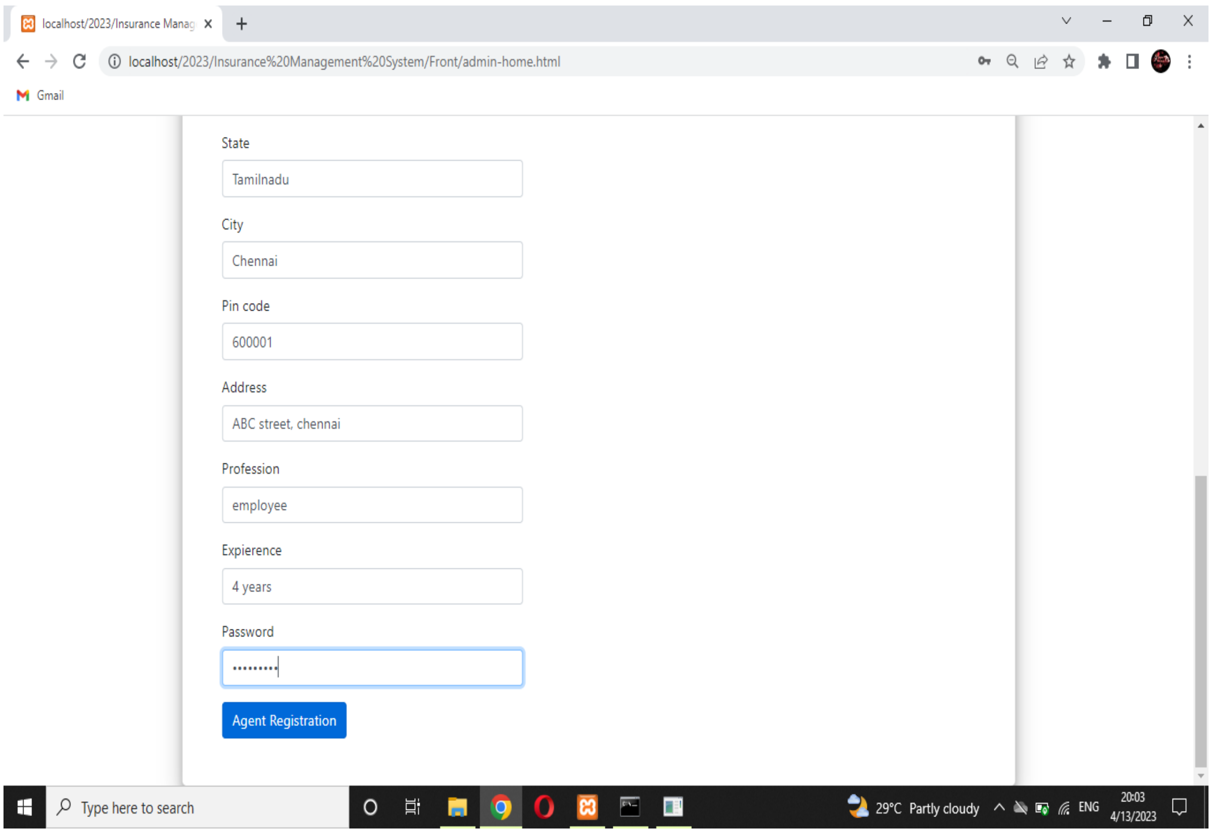
Gender

Female

Email ID

kavithak@gmail.com

Mobile Number



localhost/2023/Insurance%20Management%20System/Front/admin-home.html

State

Tamilnadu

City

Chennai

Pin code

600001

Address

ABC street, chennai

Profession

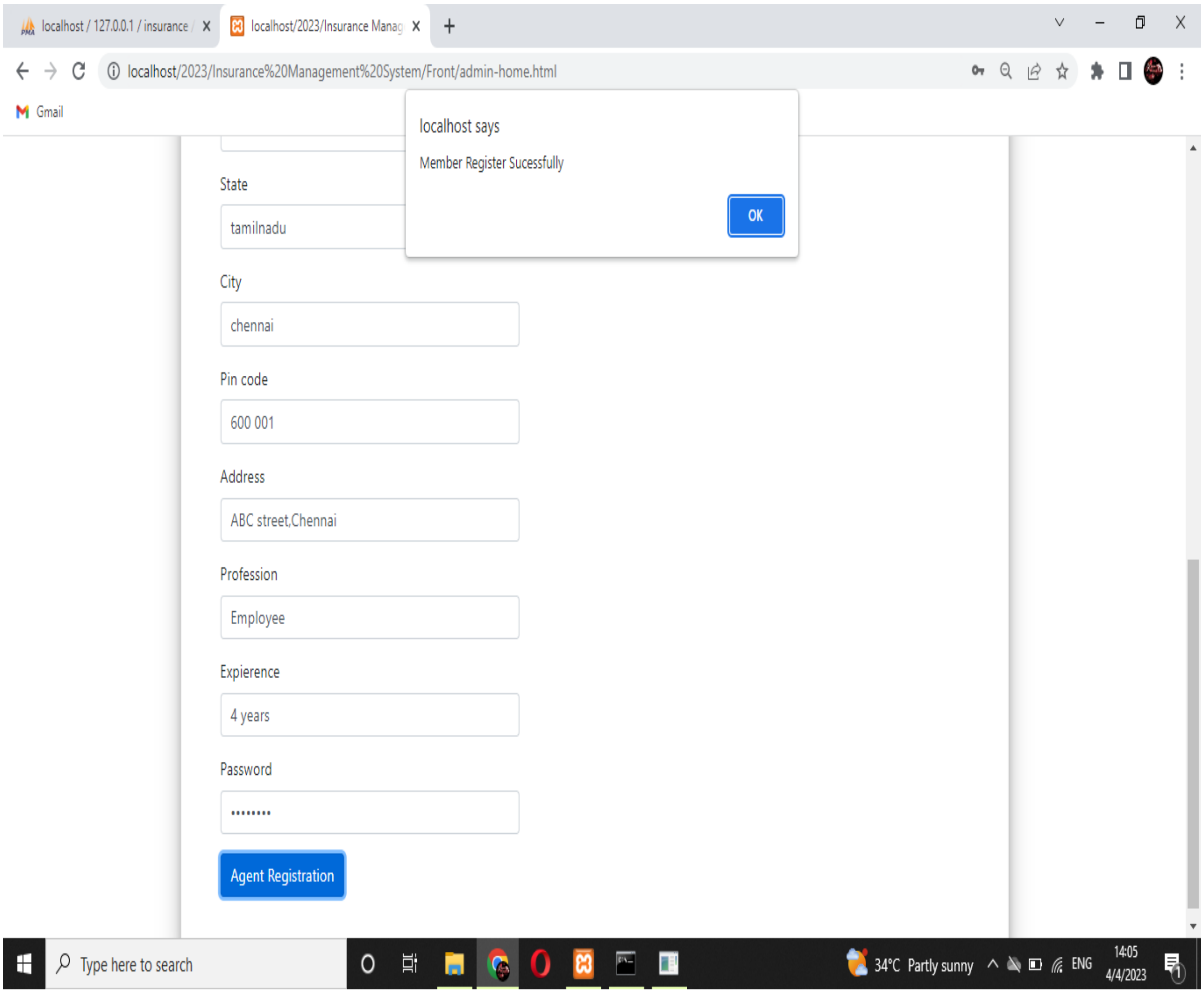
employee

Expierence

4 years

Password

•••••••••


localhost / 127.0.0.1 / insurance

localhost/2023/Insurance Manag

localhost/2023/Insurance%20Management%20System/Front/admin-home.html

Gmail

localhost says

Member Register Sucessfully

OK

State: tamilnadu

City: chennai

Pin code: 600 001

Address: ABC street,Chennai

Profession: Employee

Expierence: 4 years

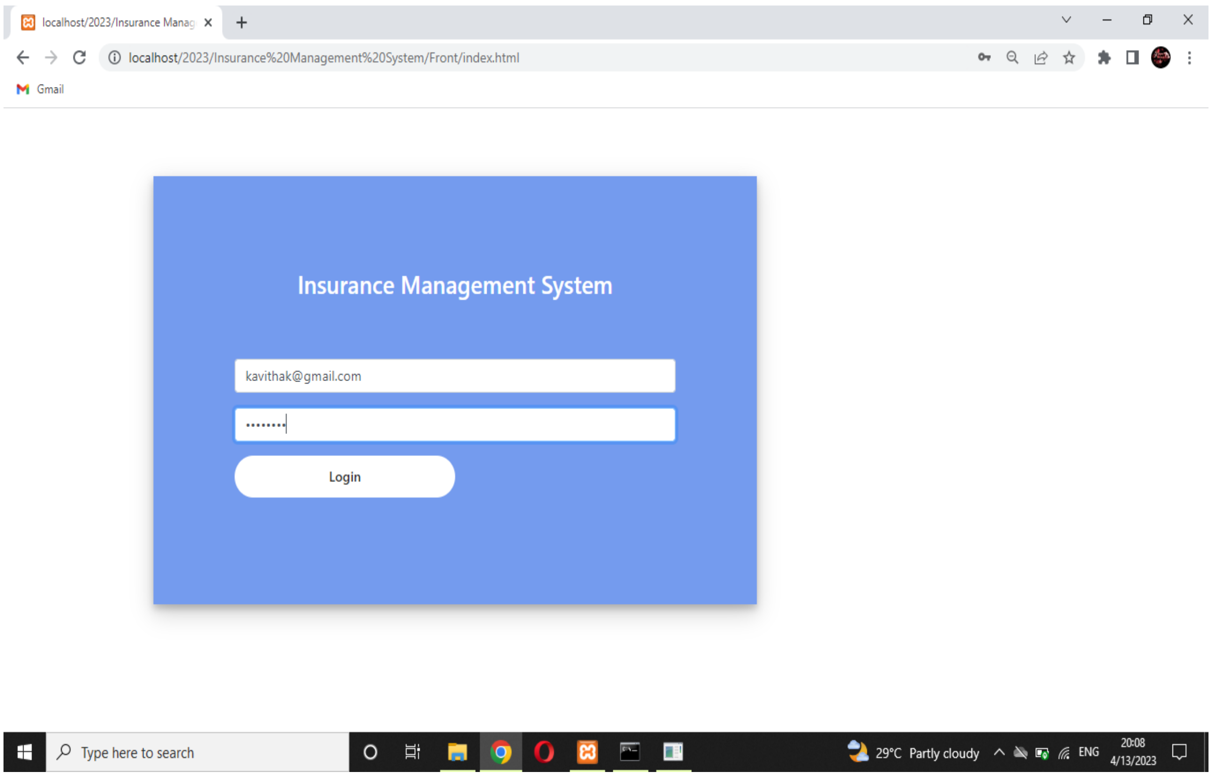
Password: ••••••••

Agent Registration

Type here to search

34°C Partly sunny ENG 14:05 4/4/2023

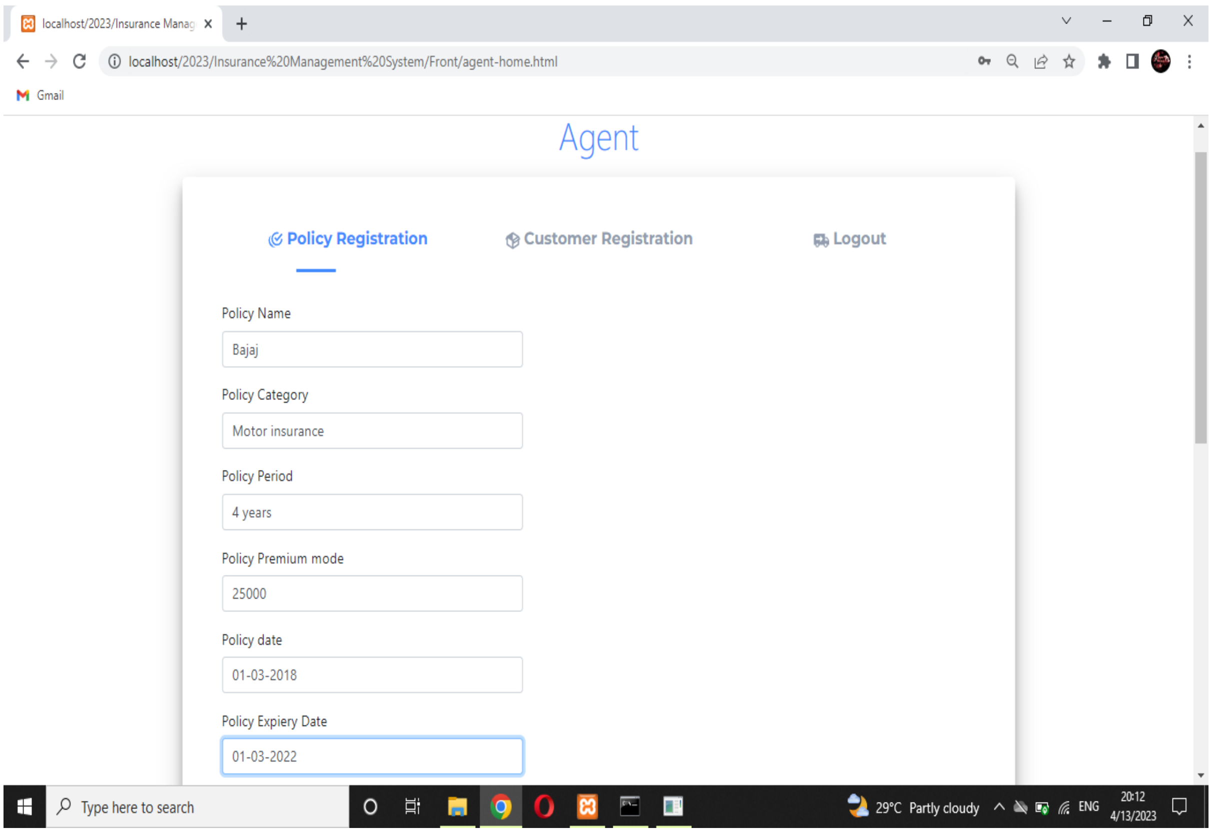
AGENT LOGIN PAGE:


localhost/2023/Insurance Manag
localhost/2023/Insurance%20Management%20System/Front/index.html
Gmail
Login
Type here to search
29°C Partly cloudy
ENG
20:08
4/13/2023

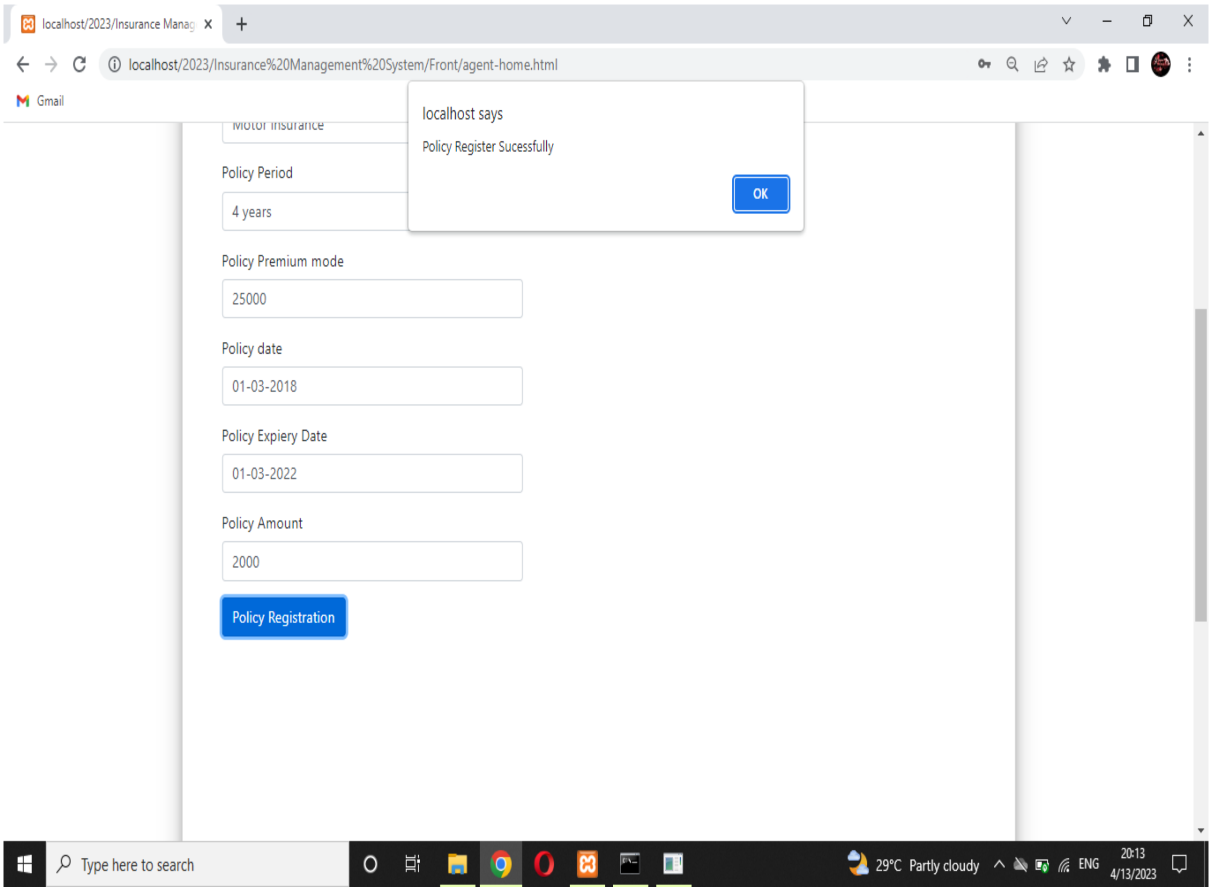
Insurance Management System

kavithak@gmail.com

POLICY REGISTRATION PAGE:



Policy Name	Bajaj
Policy Category	Motor insurance
Policy Period	4 years
Policy Premium mode	25000
Policy date	01-03-2018
Policy Expiery Date	01-03-2022


localhost/2023/Insurance Manag

localhost/2023/Insurance%20Management%20System/Front/agent-home.html

Gmail

Motor insurance

Policy Period

4 years

Policy Premium mode

25000

Policy date

01-03-2018

Policy Expiery Date

01-03-2022

Policy Amount

2000

Policy Registration

localhost says

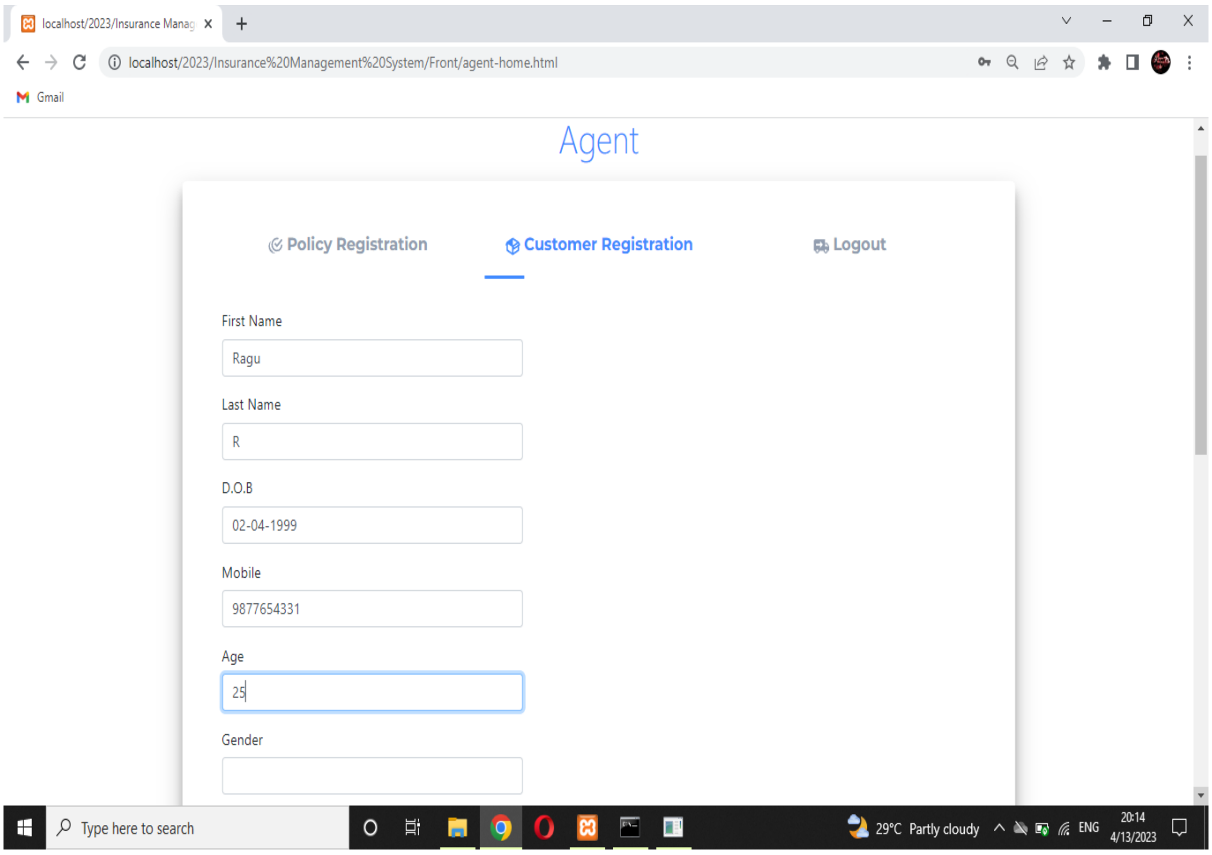
Policy Register Sucessfully

OK

Type here to search

29°C Partly cloudy ENG 20:13 4/13/2023

CUSTOMER REGISTRATION PAGE:



Agent

First Name

Ragu

Last Name

R

D.O.B

02-04-1999

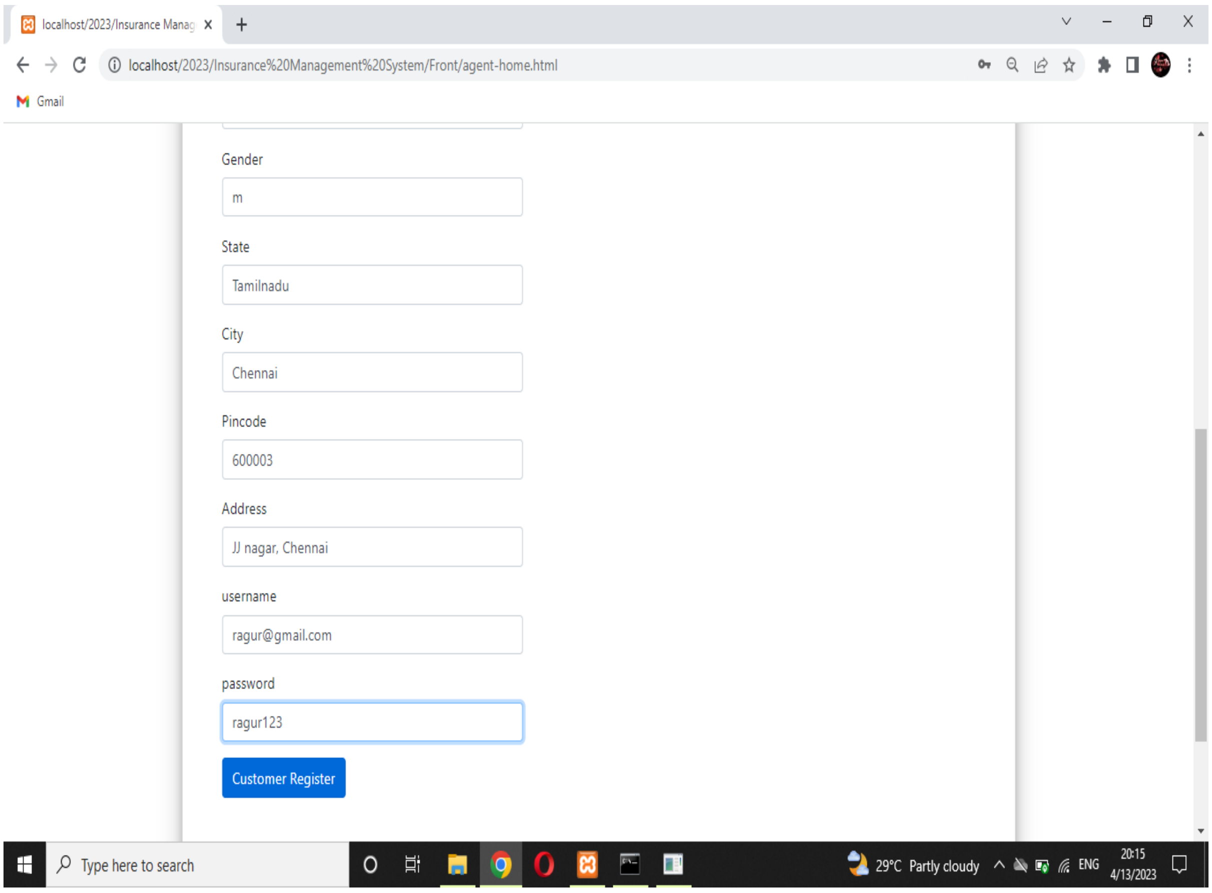
Mobile

9877654331

Age

25

Gender



localhost/2023/Insurance Manag
localhost/2023/Insurance%20Management%20System/Front/agent-home.html
Gmail

Gender

m

State

Tamilnadu

City

Chennai

Pincode

600003

Address

JJ nagar, Chennai

username

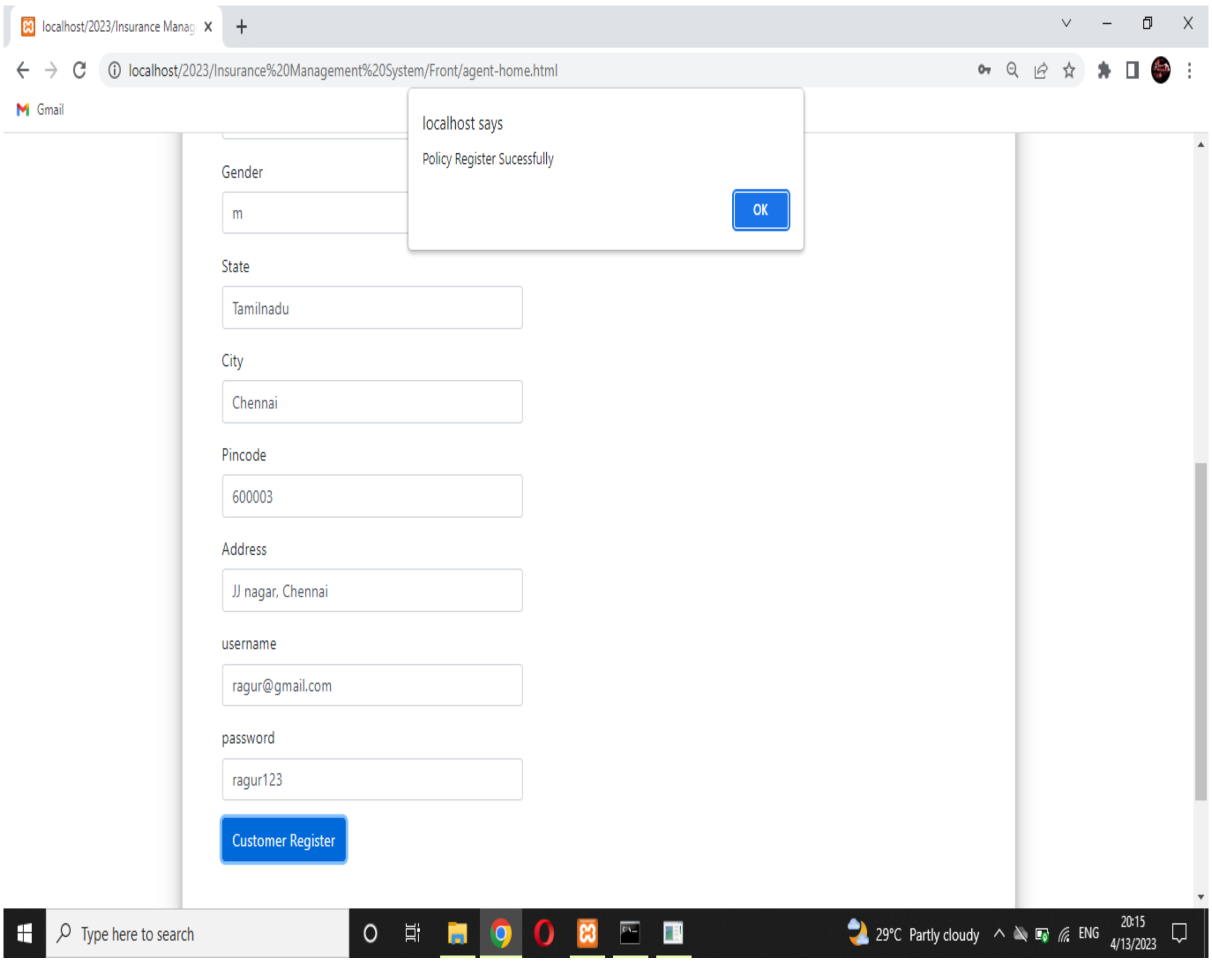
ragur@gmail.com

password

ragur123

Customer Register

Type here to search
29°C Partly cloudy
ENG
20:15
4/13/2023



localhost/2023/Insurance%20Management%20System/Front/agent-home.html

localhost says

Policy Register Sucessfully

Gender: m

State: Tamilnadu

City: Chennai

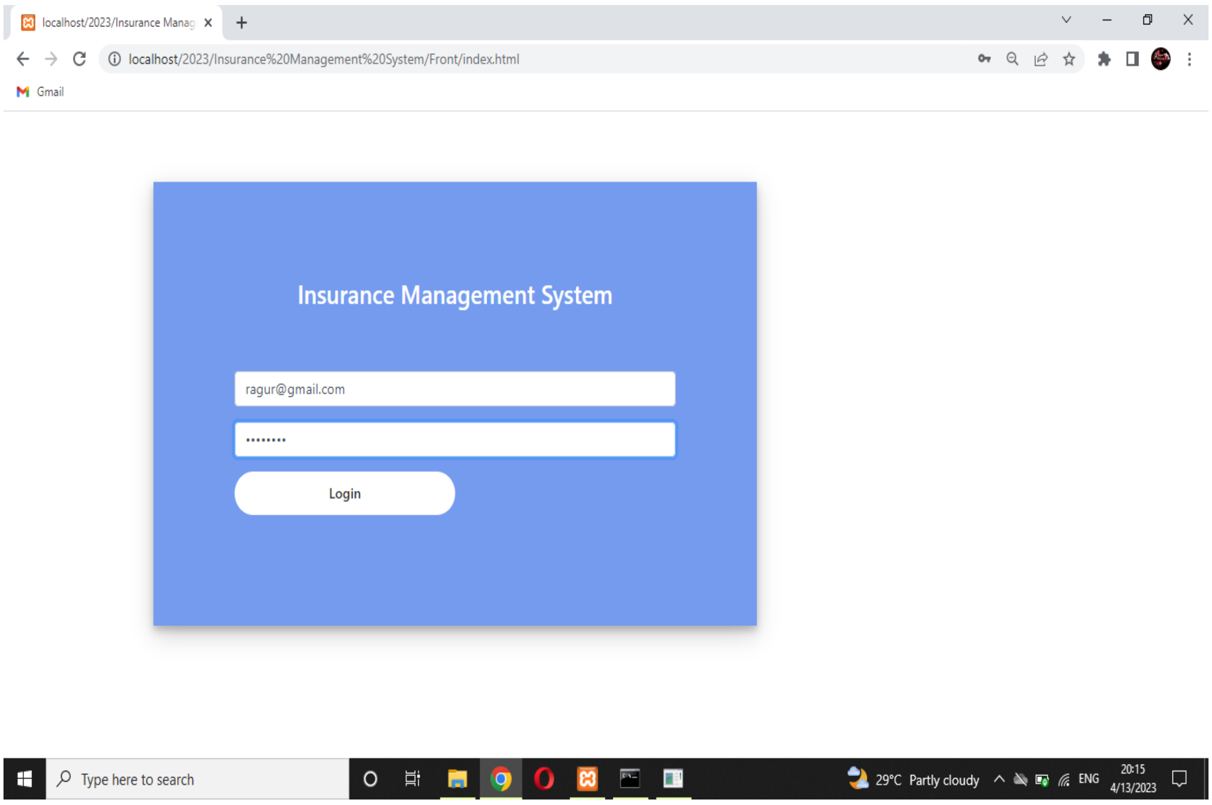
Pincode: 600003

Address: JJ nagar, Chennai

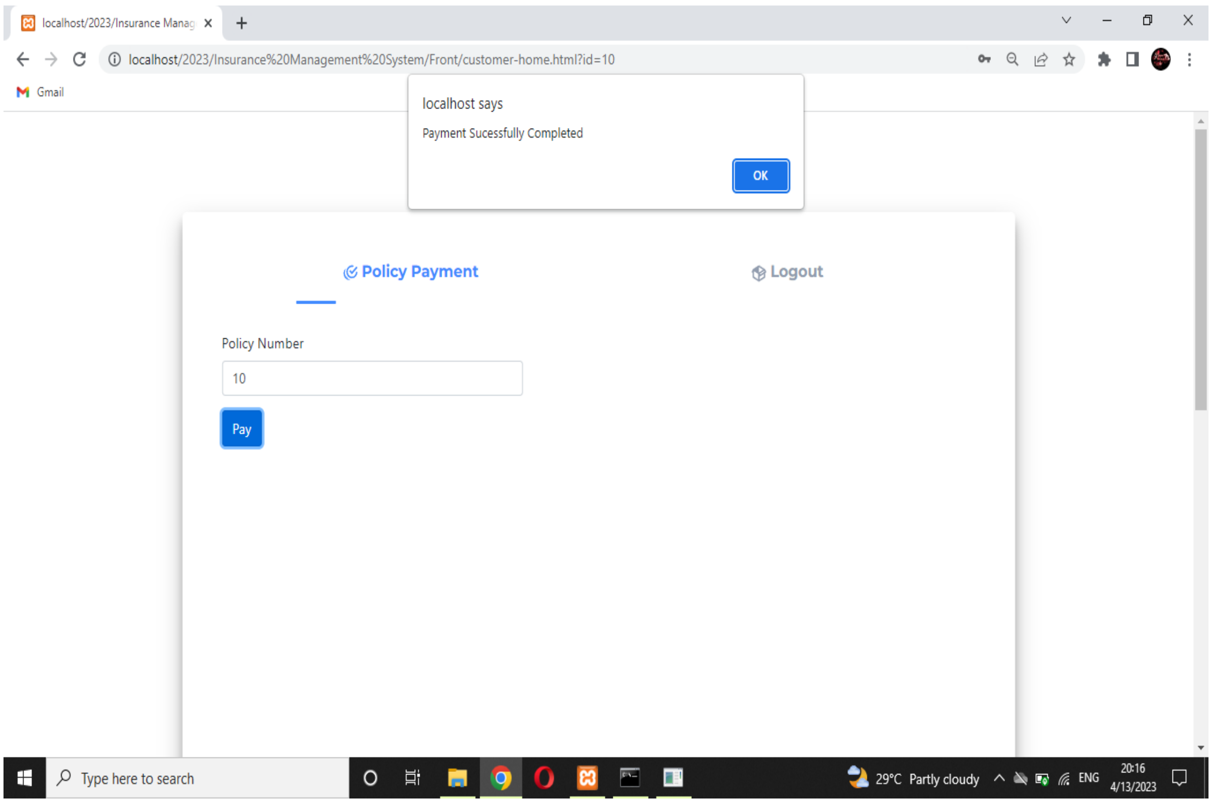
username: ragur@gmail.com

password: ragur123

CUSTOMER LOGIN PAGE:


localhost/2023/Insurance Manag
localhost/2023/Insurance%20Management%20System/Front/index.html
Gmail
Insurance Management System
ragur@gmail.com
••••••••
Login
Type here to search
29°C Partly cloudy
ENG
20:15
4/13/2023

PAYMENT PAGE:

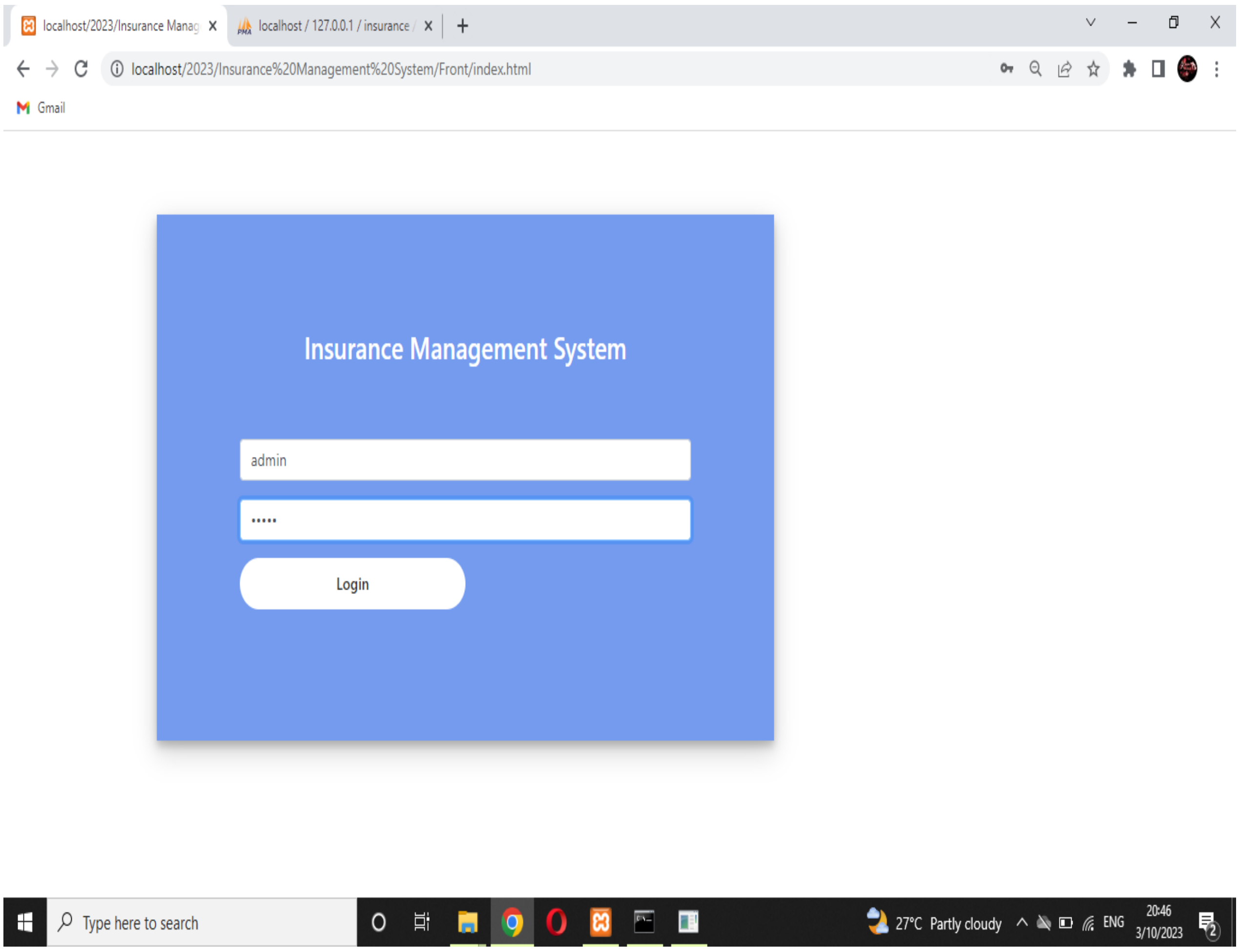

localhost/2023/Insurance Manag

localhost/2023/Insurance%20Management%20System/Front/customer-home.html?id=10

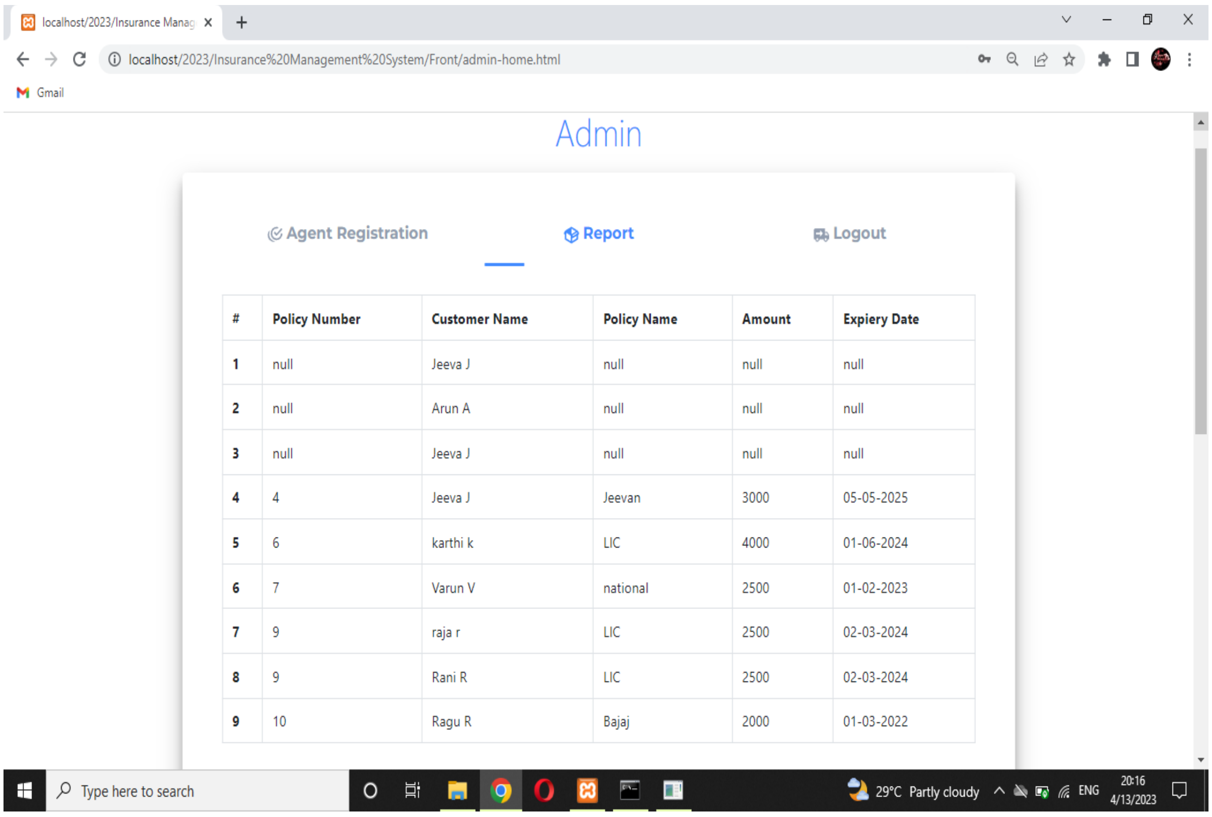
localhost says

Payment Sucessfully Completed

ADMIN LOGIN:


localhost/2023/Insurance Manag
localhost / 127.0.0.1 / insurance
localhost/2023/Insurance%20Management%20System/Front/index.html
Gmail
Insurance Management System
admin
•••••
Login
Type here to search
27°C Partly cloudy
ENG
20:46
3/10/2023

REPORT PAGE:


localhost/2023/Insurance Manag
localhost/2023/Insurance%20Management%20System/Front/admin-home.html
Gmail
Admin
Agent Registration
Report
Logout
Type here to search
29°C Partly cloudy
ENG
20:16
4/13/2023

#	Policy Number	Customer Name	Policy Name	Amount	Expiery Date
1	null	Jeeva J	null	null	null
2	null	Arun A	null	null	null
3	null	Jeeva J	null	null	null
4	4	Jeeva J	Jeevan	3000	05-05-2025
5	6	karthi k	LIC	4000	01-06-2024
6	7	Varun V	national	2500	01-02-2023
7	9	raja r	LIC	2500	02-03-2024
8	9	Rani R	LIC	2500	02-03-2024
9	10	Ragu R	Bajaj	2000	01-03-2022