

ANALYSIS OF USER BEHAVIOUR , DETECTION, TRACKING & ELIMINATION OF HACKER IN CLOUD ENVIRONMENT

A PROJECT REPORT

Submitted by

DEVI N

412919104003

LOGESH E

412919104006

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING



Vi INSTITUTE OF TECHNOLOGY, SIRUNKUNDRAM

ANNA UNIVERSITY : CHENNAI – 600025

JUNE & 2023

CERTIFICATE

This is to certify that the project report entitled “**Analysis Of User Behaviour , Detection , Tracking & Elimination Of Hacker In Cloud Environment**” being submitted by **Devi N (412919104003)** and **Logesh E (412919104006)** for the partial fulfilment of the requirements for the award of degree of Bachelor of Engineering Computer Science and Engineering of Vi Institute of Technology, Sirunkundram - 603 108 is a record of bonafide project work carried out by them under the supervision and guidance of the undersigned. In my opinion, the project report is worthy of consideration for the award of the degree of Bachelor of Engineering in Computer Science and Engineering in accordance with the regulations of the Anna University Chennai.

Approved and Forwarded by

Assistant Professor

Department of Computer Science and Engineering

Vi Institute of Technology,

Sirunkundam- 603 108

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this Report “**Analysis Of User Behaviour, Detection,Tracking & Elimination Of Hacker In Cloud Environment**” being submitted by **Devi N (412919104003)** and **Logesh E(412918104005)** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported herein doesnot form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Signature of the Supervisor

Signature of the HOD

Department of CSE

Department of CSE

Vi Institute of Technology

Vi Institute of Technology

Sirunkundram-603108

Sirunkundram-603108

Submitted for Viva voce Examination held on

Internal Examiner

External Examiner

Acknowledgements

It is our immense pleasure to acknowledge the people who have been behind us to bring out this project successfully. This is the right time to express our heartfelt thanks to all our well-wishers.

We express our deep sense of gratefulness to our project guide -
-----, Assistant Professor, Department of Computer Science and Engineering, Vi Institute of Technology for his immense support, motivation, inspiration and above all for his constant counsel that enabled me in bringing this thesis to its present form.

We would also like to place on record our deep sense of gratitude to -----
-----, Head of the department, Department of Computer Science and Engineering for all his immeasurable guidance and encouragement.

We are very much grateful to our beloved chairman- - - - -
- - - , who blessed us ever to complete the task successfully.

We express our thanks to our review committee, faculty members who supported us directly and indirectly to bring out this project completely. We would like to share our heartfelt thanks to our parents for their blessings. We also thank the god for giving us the enough strength to come out of the hurdles.

Devi N

Logesh E

ABSTRACT

Social networks Accounts are tracked & detected. If hacker attacks the Genuine user, then allows the attacker to proceed further until our system captures all the important information about the attacker. We generate Honey words based on the user info provided and the original password is converted into another format and stored along with the Honey words. We deploy Intermediate server, Shopping server for purchase and Cloud server for maintaining user account details. Attacker who knows the E mail account of original user can easily reset the password of the cloud server. Attacker is allowed to do attack in this project, so as to find him out very easily. Now attacker logs into the purchase portal, where he is been tracked unknowingly. Server identifies the attacker and sends the info to the original owner and also it blocks the attacker even doing transaction from his original account.

TABLE OF CONTENT

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	V
	LIST OF FIGURES	VIII
1.	INTRODUCTION	1
1.1	Domain Introduction	
1.2	Project Introduction	
2.	LITERATURE SURVEY	6
3.	SYSTEM ANALYSIS	12
3.1	Existing System	
3.2	Proposed System	
4.	ARCHITECTURE DIAGRAM	14
4.1	System Architecture	
4.2	Data Flow Diagram	
4.3	Feasibility Study	
5.	SYSTEM SPECIFICATIONS	23
5.1	Hardware Environment	
5.2	Software Environment	
5.3	N-Tier Architecture	

6.	FEATURES OF JAVASCRIPT	31
6.1	JavaScript Programming Language	
6.2	React JS	
6.3	Node JS	
6.4	Express JS	
7.	MODULES	43
7.1	Modules	
7.2	Modules Description	
8.	SYSTEM TESTING	48
9.	APPENDIX	53
9.1	Source Coding	
9.2	Screenshots	
10.	CONCLUSION	93
11.	REFERENCES	94

LIST OF FIGURES

FIGURE NO.	NAME	PAGE NO.
4.1	Architecture Diagram	14
4.2	Data Flow Diagrams	15
4.2.2.1	Use Case Diagram	17
4.2.3.1	Sequence Diagram	18
4.2.4.1	Activity Diagram	19
4.2.5.1	Collaboration diagram	20
5.3.1	N-Tier Architecture	30
6.1	JavaScript Architecture	32
6.2	JSX	36
6.3	Node JS Architecture	38
8.1	Taxonomy of Testing	47
9.2	Screenshots	81
9.2.1	Login Page	89
9.2.2	Register Page	89
9.2.3	Register Security Q/A Page	90
9.2.4	Admin Panel	90
9.2.5	Store Page	91
9.2.6	Cart Page	91
9.2.7	Shipping Address Page	92
9.2.8	Payment Options Page	92

CHAPTER - 1

INTRODUCTION

1.1 DOMAIN INTRODUCTION

Cloud computing is a fast-growing technology that has established itself in the next generation of IT industry and business. Cloud computing promises reliable software, hardware, and IaaS (Infrastructure as a Service) delivered over the Internet and remote data centers. Cloud services have become a powerful architecture to perform complex large-scale computing tasks and span a range of IT functions from storage and computation to database and application services. The need to store, process, and analyze large amounts of datasets has driven many organizations and individuals to adopt cloud computing. A large number of scientific applications for extensive experiments are currently deployed in the cloud and may continue to increase because of the lack of available computing facilities in local servers, reduced capital costs, and increasing volume of data produced and consumed by the experiments. In addition, cloud service providers have begun to integrate frameworks for parallel data processing in their services to help users access cloud resources and deploy their programs.

Cloud computing is a model for allowing ubiquitous, convenient, and on-demand network access to a number of configured computing resources (e.g., networks, server, storage, application, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. Cloud computing has a number of favorable aspects to address the

rapid growth of economies and technological barriers. Cloud computing provides total cost of ownership and allows organizations to focus on the core business without worrying about issues, such as infrastructure, flexibility, and availability of resources. Moreover, combining the cloud computing utility model and a rich set of computations, infrastructures, and storage cloud services offers a highly attractive environment where scientists can perform their experiments. Cloud service models typically consist of PaaS (Platform as a Service), SaaS (Software as a Service), and IaaS (Infrastructure as a Service). PaaS, such as Google's Apps Engine, Salesforce.com, Force platform, and Microsoft Azure, refers to different resources operating on a cloud to provide platform computing for end users.

SaaS, such as Google Docs, Gmail, Salesforce.com, and Online Payroll, refers to applications operating on a remote cloud infrastructure offered by the cloud provider as services that can be accessed through the Internet.

IaaS, such as Flexi scale and Amazon's EC2, refers to hardware equipment operating on a cloud provided by service providers and used by end users upon demand.

The increasing popularity of wireless networks and mobile devices has taken cloud computing to new heights because of the limited processing capability, storage capacity, and battery lifetime of each device. This condition has led to the emergence of a mobile cloud computing paradigm. Mobile cloud facilities allow users to outsource tasks to external service providers. For example, data can be processed and stored outside of a mobile device. Mobile cloud applications, such as Gmail, iCloud, and Dropbox, have become prevalent recently. Juniper research predicts that cloud-based mobile applications will increase to approximately 9.5\$ billion by 2014. Such applications improve mobile cloud performance and user experience. However, the limitations

associated with wireless networks and the intrinsic nature of mobile devices have imposed computational and data storage restrictions

1.1.1 Cloud Setup

Cloud Service Provider will contain the large amount of data in their Data Storage. Also the Cloud Service provider will maintain the all the User information to authenticate the User when are login into their account. The User information will be stored in the Database of the Cloud Service Provider. Also, the Cloud Server will redirect the User requested job to the Resource Assigning Module to process the User requested Job. The Request of all the Users will process by the Resource Assigning Module. To communicate with the Client and the with the other modules of the Cloud Network, the Cloud Server will establish connection between them. For this purpose, we are going to create an User Interface Frame. Also the Cloud Service Provider will send the User Job request to the Resource Assign Module in First in First out (FIFO) manner.

1.1.2 Private Cloud

Private clouds are dedicated to one organization and do not share physical resources. The resource can be provided in-house or externally. A typical underlying requirement of private cloud deployments are security requirements and regulations that need a strict separation of an organization's data storage and processing from accidental or malicious access through shared resources. Private cloud setups are challenging since the economical advantages of scale are usually not achievable within most projects and organizations despite the utilization of industry standards. The return of investment compared to public cloud offerings is rarely obtained and the operational overhead and risk of failure is significant.

1.1.3 Public Cloud

Public clouds share physical resources for data transfers, storage, and processing. However, customers have private visualized computing environments and isolated storage. Security concerns, which entice a few to adopt private clouds or custom deployments, are for the vast majority of customers and projects irrelevant. Visualization makes access to other customers' data extremely difficult.

1.1.4 Hybrid Cloud

The hybrid cloud architecture merges private and public cloud deployments. This is often an attempt to achieve security and elasticity, or provide cheaper base load and burst capabilities. Some organizations experience short periods of extremely high loads, e.g., as a result of seasonality like black Friday for retail, or marketing events like sponsoring a popular TV event. These events can have huge economic impact to organizations if they are serviced poorly.

The hybrid cloud provides the opportunity to serve the base load with in-house services and rent for a short period a multiple of the resources to service the extreme demand. This requires a great deal of operational ability in the organization to seamlessly scale between the private and public cloud. Tools for hybrid or private cloud deployments exist like Eucalyptus for Amazon Web Services. On the long-term the additional expense of the hybrid approach often is not justifiable since cloud providers offer major.

1.2 PROJECT INTRODUCTION:

Cloud storage services have rapidly become increasingly popular. Users can store their data on the cloud and access their data anywhere at any time. Because

of user privacy, the data stored on the cloud is typically encrypted and protected from access by other users. Considering the collaborative property of the cloud data, attribute-based encryption (ABE) is regarded as one of the most suitable encryption schemes for cloud storage. There are numerous ABE schemes that have been proposed, including [1], [2], [3], [4], [5], [6], [7].

Most of the proposed schemes assume cloud storage service providers or trusted third parties handling key management are trusted and cannot be hacked; however, in practice, some entities may intercept communications between users and cloud storage providers and then compel storage providers to release user secrets by using government power or other means. In this case, encrypted data are assumed to be known and storage providers are requested to release user secrets. As an example, in 2010, without notifying its users, Google released user documents to the FBI after receiving a search warrant [8]. In 2013, Edward Snowden disclosed the existence of global surveillance programs that collect such cloud data as emails, texts, and voice messages from some technology companies. Once cloud storage providers are compromised, all encryption schemes lose their effectiveness. Though we hope cloud storage providers can fight against such entities to maintain user privacy through legal avenues, it is seemingly more and more difficult.

CHAPTER - 2

LITERATURE SURVEY

2.1 Some Remarks on Honeyword Based Password-Cracking Detection

Imran Erguler TUBITAK BILGEM, Gebze, Kocaeli, Turkey,
imran.erguler@tubitak.gov.tr

2.1.1 Abstract:

Recently, Juels and Rivest proposed honeywords (decoy passwords) to detect attacks against hashed password databases. For each user account, the legitimate password is stored with several honey words in order to sense impersonation. If honeywords are selected properly, an adversary who steals a the of hashed passwords cannot be sure if it is the real password or a honeyword for any account. Moreover, entering with a honeyword to login will trigger an alarm notifying the administrator about a password the breach. At the expense of increasing storage requirement by 20 times, the authors introduce a simple and effective solution to detection of password file disclosure events. In this study, we scrutinize the honeyword system and present some remarks to highlight possible weak points. Also, we suggest an alternative approach that selects honeywords from existing user passwords in the system to provide realistic honeywords a perfectly at honeyword generation method and also to reduce storage cost of the honeyword scheme.

2.2 Prover and Verifier Based Password Protection: PVBPP

Priyanka Naik1*, Sugata Sanyal2 Computer Science Department, Manipal
Institute of Technology, Manipal, India ppnaik1890@gmail.com
Corporate Technology Office, Tata Consultancy Services, Mumbai, India
sugata.sanyal@tcs.com *Corresponding Author

2.2.1 Abstract:

In today's world password are mostly used for authentication. This makes them prone to various kinds of attacks like dictionary attacks. A dictionary attack is a method of breaking the password by systematically entering every word in a dictionary as a password. This attack leads to an overload on the server leading to denial-of-service attack. This paper presents a protocol to reduce the rate of dictionary attack by using a prover and a verifier system. This system makes it difficult for the attacker to prove it as a valid user by becoming computationally intensive. The rate of attempts is also reduced and thus restricting the Denial-of-Service attack.

2.3 The Dangers of Weak Hashes

GIAC GWEB Gold Certification Author: Kelly Brown,
kbrown@aboutweb.com Advisor: Robert Vandenbrink Accepted: November
15, 2013

2.3.1 Abstract:

There have been several high publicity password leaks over the past year including LinkedIn, Yahoo, and eHarmony. While you never want to have

vulnerabilities that allow hackers to get access to your password hashes, you also want to make sure that if the hashes are compromised it is not easy for hackers to generate passwords from the hashes. As these leaks have demonstrated, large companies are using weak hashing mechanisms that make it easy to crack user passwords. In this paper I will discuss the basics of password hashing, look at password cracking software and hardware, and discuss best practices for using hashes securely.

2.4 Improving Security using Deception

Mohammed H. Almeshekah, Eugene H. Spafford and Mikhail J. Atallah
November 11, 2013

2.4.1 Abstract:

As the convergence between our physical and digital worlds continues at a rapid pace, much of our information is becoming available online. In this paper we develop a novel taxonomy of methods and techniques that can be used to protect digital information. We discuss how information has been protected and show how we can structure our methods to achieve better results. We explore complex relationships among protection techniques ranging from denial and isolation, to degradation and obfuscation, through negative information and deception, ending with adversary attribution and counter-operations. We present analysis of these relationships and discuss how they can be applied at different scales within organizations. We also identify some of the areas that are worth further investigation. We map these protection techniques against the cyber kill-chain model and discuss some findings.

2.5 Kamouage: Loss-Resistant Password Management

Hristo Bojinov¹, Elie Bursztein¹, Xavier Boyen², and Dan Boneh¹ ¹ Stanford University ² Universite de Liege, Belgium

2.5.1 Abstract:

We introduce Kamouage: a new architecture for building theft-resistant password managers. An attacker who steals a laptop or cell phone with a Kamouage-based password manager is forced to carry out a considerable amount of online work before obtaining any user credentials. We implemented our proposal as a replacement for the built-in Firefox password manager, and provide performance measurements and the results from experiments with large real-world password sets to evaluate the feasibility and effectiveness of our approach. Kamouage is well-suited to become a standard architecture for password managers on mobile devices.

2.6 Honeywords: Making Password-Cracking Detectable

Ari Juels RSA Labs Cambridge, MA 02142 ajuels@rsa.com Ronald L. Rivest MIT CSAIL Cambridge, MA 02139 rivest@mit.edu May 2, 2013 Version 2.0

2.6.1 Abstract:

We suggest a simple method for improving the security of hashed passwords: the maintenance of additional honeywords" (false passwords) associated with each user's account. An adversary who steals a list of hashed passwords and inverts the hash function cannot tell if he has found the password or a honeyword. The attempted use of a honeyword for login sets off an alarm. An auxiliary server (the honey checker") can distinguish the user password from

honeywords for the login routine, and will set an alarm if a honeyword is submitted.

2.7 The science of guessing: analyzing an anonymized corpus of 70 million passwords

Joseph Bonneau Computer Laboratory University of Cambridge
jcb82@cl.cam.ac.uk

2.7.1 Abstract:

We report on the largest corpus of user-chosen passwords ever studied, consisting of anonymized password histograms representing almost 70 million Yahoo! users, mitigating Privacy concerns while enabling analysis of dozens of sub populations based on demographic factors and site usage characteristics. This large data set motivates a thorough statistical treatment of estimating guessing difficulty by sampling from a secret distribution. In place of previously used metrics such as Shannon entropy and guessing entropy, which cannot be estimated with any realistically sized sample, we develop partial guessing metrics including a new variant of guess work parameterized by an attacker's desired success rate. Our new metric is comparatively easy to approximate and directly relevant for security engineering. By comparing password distributions with a uniform distribution which would provide equivalent security against different forms of guessing attack, we estimate that passwords provide fewer than 10 bits of security against an online, trawling attack, and only about 20bits of security against an optimal offline dictionary attack. We find surprisingly little variation in guessing difficulty; every identifiable group of users generated a comparably weak password distribution. Security motivations such as their

registration of a payment card have no greater impact than demographic factors such as age and nationality. Even proactive efforts to nudge users towards better password choices with graphical feedback make little difference. More surprisingly, even seemingly distant language communities choose the same weak passwords and an attacker never gains more than a factor of 2 efficiencies gain by switching from the globally optimal dictionary to a population-specific lists.

CHAPTER – 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

Cloud storage services have become increasingly popular. Because of the importance of privacy, many cloud storage encryption schemes has not much of security to store the data safely.

3.1.1 DISADVANTAGES:

- Security is very low, so that the attacker can easily hack the passwords of the users and can do anything.
- Passwords can be hacked using guessing attacks

3.2 PROPOSED SYSTEM:

We include two processes. 1. As per the Paper Compromised Social Networks Accounts are tracked and detected. If hacker attacks the Genuine user, then our allows the attacker to proceed further until our system captures all the important information about the attacker. We generate Honeywords based on the user info provided and the original password is converted into another format and stored along with the Honeywords. We deploy Intermediate server, Shopping server for purchase and Cloud server for maintaining user account details. Attacker who knows the E mail account of original user can easily reset the password of the cloud server. Attacker is invited to do attack in this Project, so as to find him out very easily. Now attacker logs into the purchase portal, where he is been

tracked unknowingly & he is allowed to do purchase. Server identifies the attacker and sends the info to the original owner and also it blocks the attacker even doing transaction from his original account.

3.2.1 ADVANTAGES:

- The attackers are not able to guess and hack the passwords.
- It provides high security to data owner
- Hacker IP address and address sent original user via email

CHAPTER - 4

ARCHITECTURE DIAGRAM

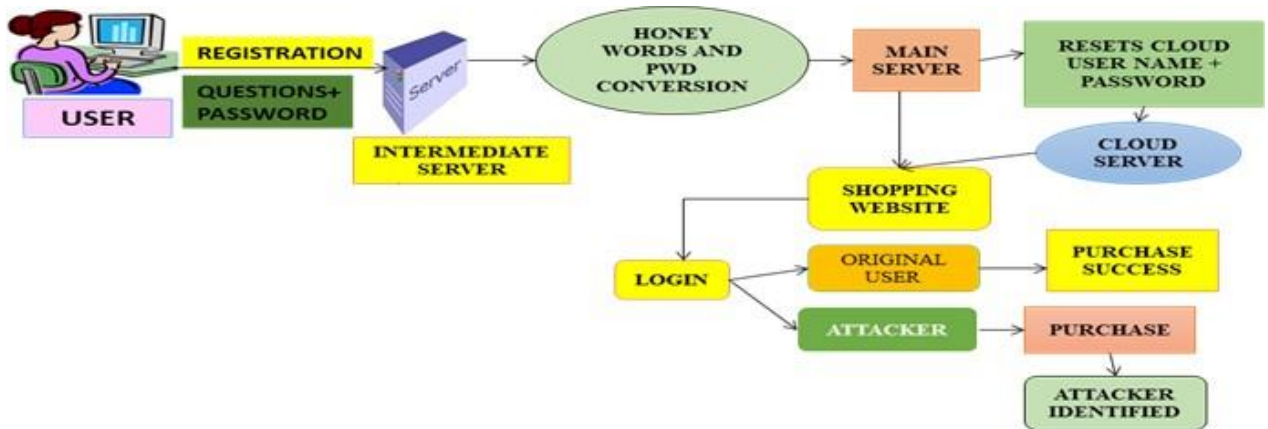


Fig 4.1 Architecture Diagram

4.1 SYSTEM ARCHITECTURE

The overall architecture represents the working procedure of the entire system. User will register their details on the portal with two email ids like primary and secondary account. while user login with correct user id password they can purchase the product with normal portal or else if any one tries to login user id with fake password by trying more than three system will automatically divert them into fake portal and fetch the fake user's delivery address and IP of the system.

4.2 DATA FLOW DIAGRAM:

DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment

and between components of a system. The visual representation makes it a good communication tool between User and System designer. A process receives input data and produces output with a different content or form. Processes can be as simple as collecting input data and saving in the database. A data store or data repository is used in a data-flow diagram to represent a situation when the system must retain data because one or more processes need to use the stored data in a later time.

LEVEL 0:

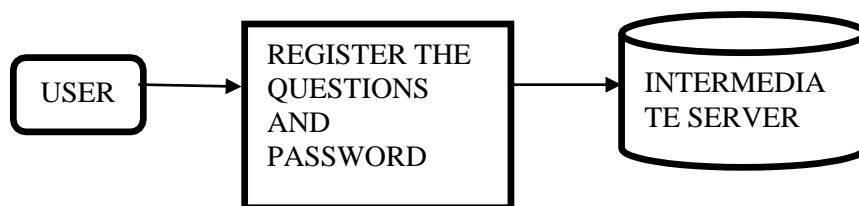


Fig 4.2.1 Level 0

LEVEL 1:

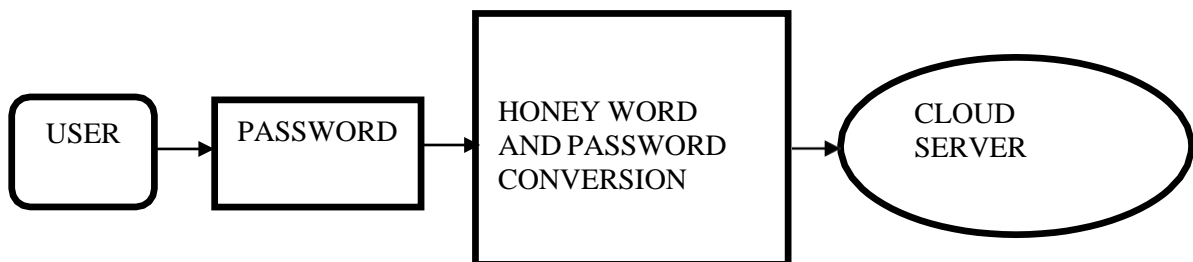


Fig 4.2.1 Level 1

LEVEL 2:



Fig 4.2.1 Level 2

LEVEL 3:



Fig 4.2.1 Level 3

4.2.1 UML DIAGRAMS:

UML is simply another graphical representation of a common semantic model. UML provides a comprehensive notation for the full lifecycle of object-oriented development.

4.2.1.1 ADVANTAGES

- To represent complete systems (instead of only the software portion) using object-oriented concepts
 - To establish an explicit coupling between concepts and executable code
 - To take into account the scaling factors that are inherent to complex and critical systems
 - To creating a modeling language usable by both humans and machines
- UML defines several models for representing systems
- The class model captures the static structure
 - The state model expresses the dynamic behavior of objects
 - The use case model describes the requirements of the user
 - The interaction model represents the scenarios and messages flows
 - The implementation model shows the work units

- The deployment model provides details that pertain to process allocation

4.2.2 USECASE DIAGRAM

Use case diagrams overview the usage requirement for system. they are useful for presentations to management and/or project stakeholders, but for actual development you will find that use cases provide significantly more value because they describe “the meant” of the actual requirements. A use case describes a sequence of action that provide something of measurable value to an action and is drawn as a horizontal ellipse.

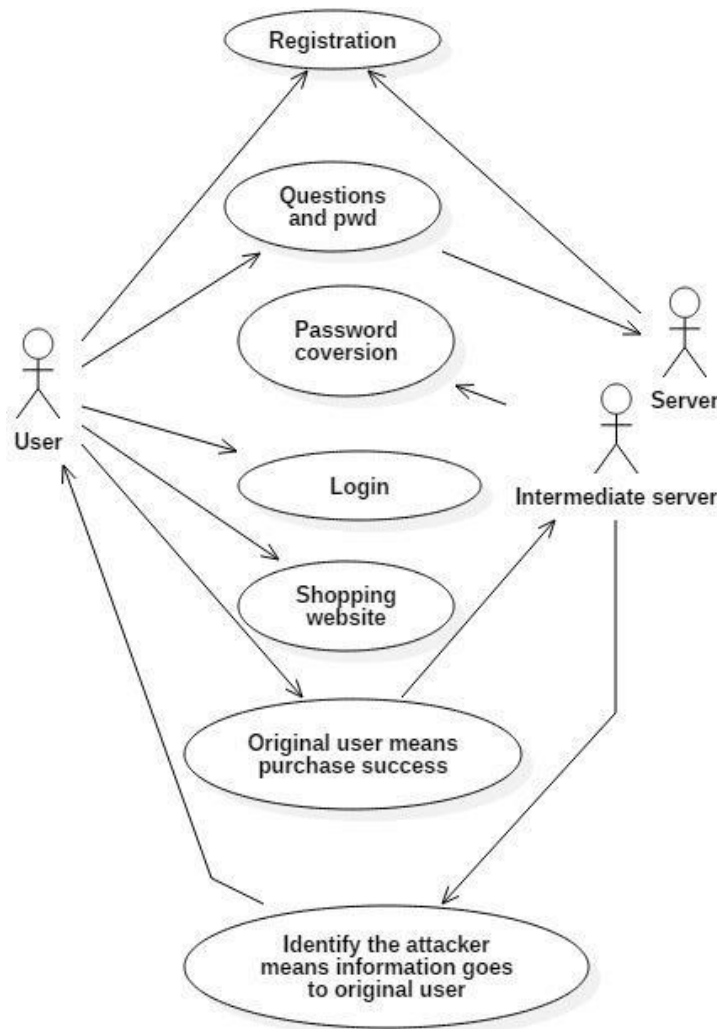


Fig 4.2.2.1 Use case diagram

4.2.3 SEQUENCE DIAGRAM

Sequence diagram model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and commonly used for both analysis and design purpose. Sequence diagram are the most popular UML artifact for dynamic modeling, which focuses on identifying the behavior within your system.

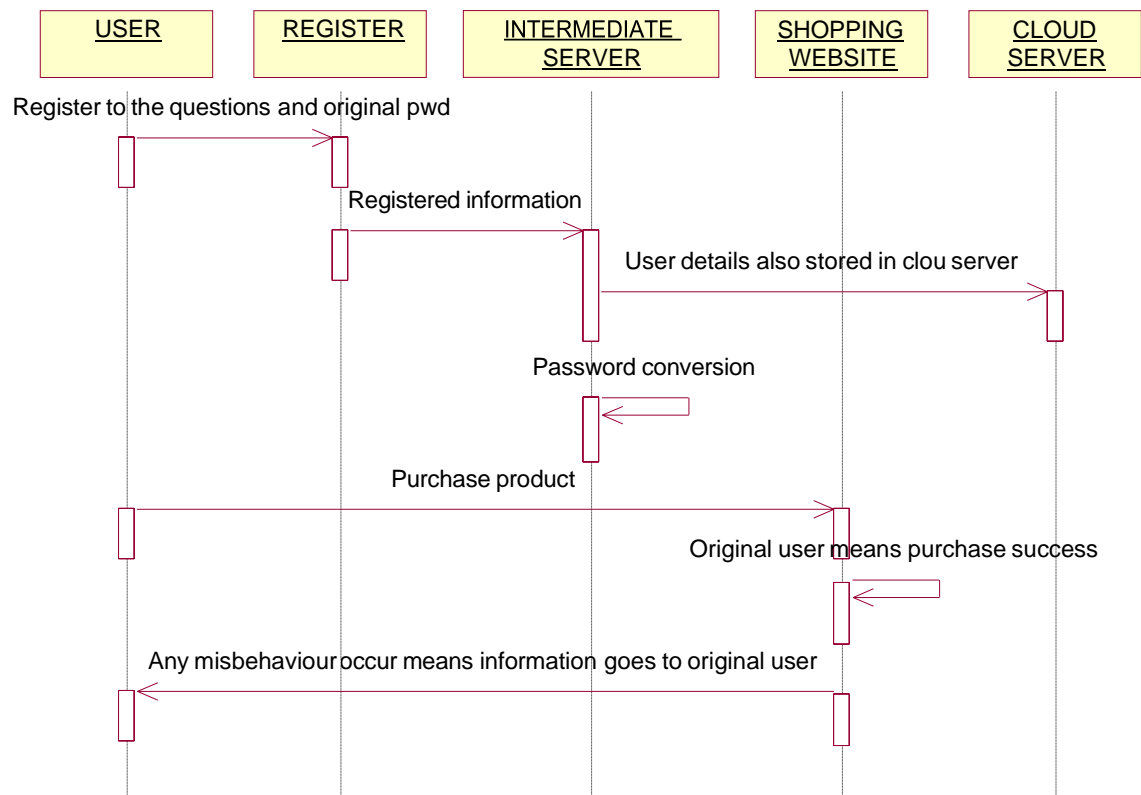


Fig 4.2.3.1 Sequence diagram

4.2.4 ACTIVITY DIAGRAM:

Activity diagram are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. The

activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. Activity diagram consist of Initial node, activity final node and activities in between.

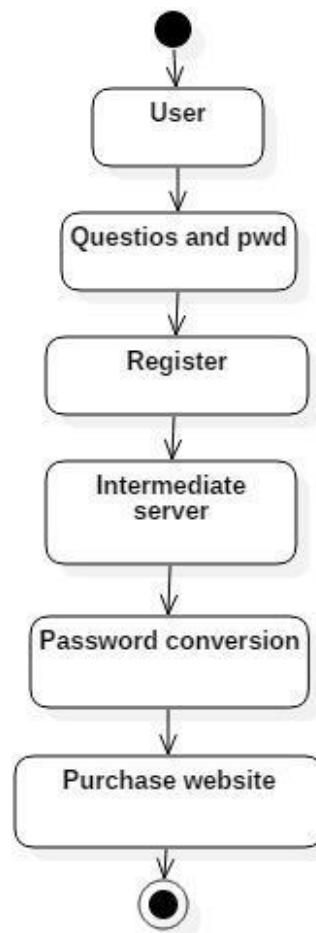


Fig 4.2.4.1 Activity diagram

4.2.5 COLLABORATION DIAGRAM

Another type of interaction diagram is the collaboration diagram. A collaboration diagram represents a collaboration, which is a set of objects related in a particular context, and interaction, which is a set of messages

exchange among the objects within the collaboration to achieve a desired outcome.

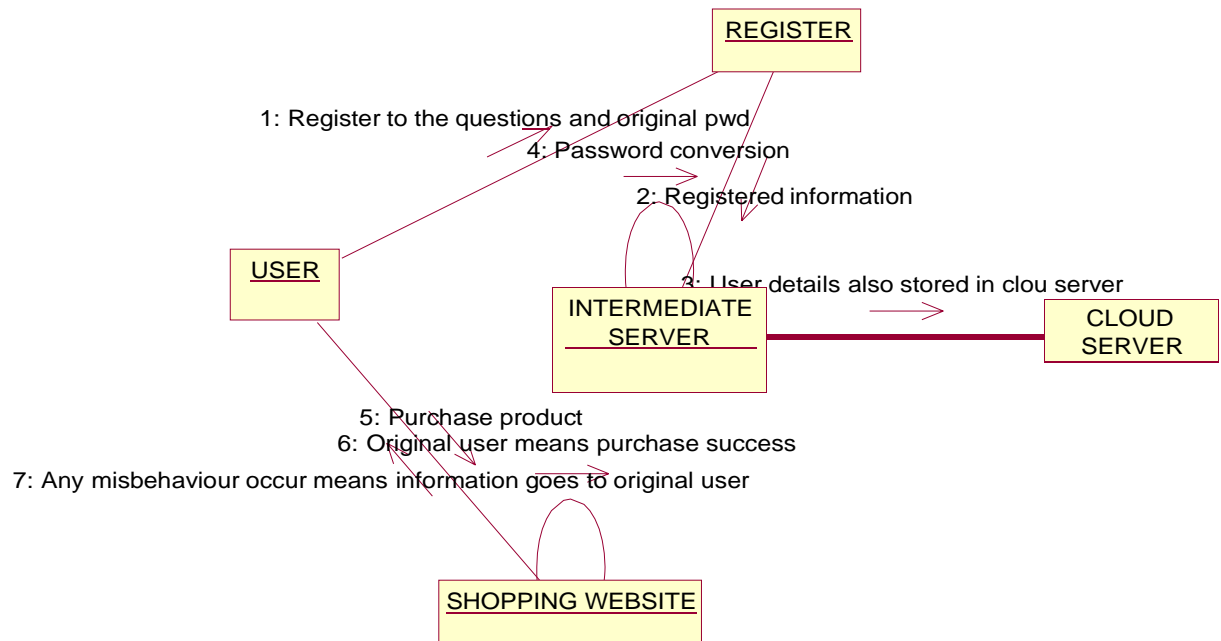


Fig 4.2.5.1 Collaboration diagram

4.3 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY

- **OPERATIONAL FEASIBILITY**

4.3.1 ECONOMICAL FEASIBILITY:

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

4.3.2 TECHNICAL FEASIBILITY:

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

4.3.3 OPERATIONAL FEASIBILITY:

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

4.4 PROBLEM DEFINITION AND METHODOLOGY:

This chapter deals with the problem definition of the project and methodology used in this project. Problem definition describes the detailed information about the project. Methodology that has been adapted in this project is discussed in methodology.

In previous system there is no identification of hacker, also does not detect the hacker ip address. Considering the modeling syntax method, one can conclude that the honeyword system loses its effectiveness against such passwords, i.e., the correct password has become noticeably recognized by an adversary. In fact, this problem seems an inherent weakness of randomly replacement-based honey word methods. Since character groups or individual characters are replaced by picked character/characters, the content integrity of such passwords would be broken and the correct password becomes quite salient.

CHAPTER – 5

SYSTEM SPECIFICATIONS

5.1 Hardware Environment

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shows what the systems do and not how it should be implemented.

- Hard disk : 120 GB
- Monitor : 15' color with vgi card support
- Ram : Minimum 256 MB
- Processor : Pentium iv and above (or) equivalent
- Processor speed : Minimum 500 MHZ

5.2 Software Environment

The software requirements are the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

- Operating system : Windows 10
- Frontend : React JS
- Backend : Node JS
- Database : MongoDB
- IDE : Visual Studio Code

5.2.1 OVERVIEW OF THE MONGODB MANAGEMENT SYSTEM

What is MongoDB?

MongoDB (link resides outside IBM) is an open source, non-relational database management system (DBMS) that uses flexible documents instead of tables and rows to process and store various forms of data. As a NoSQL solution, MongoDB does not require a relational database management system (RDBMS), so it provides an elastic data storage model that enables users to store and query multivariate data types with ease. This not only simplifies database management for developers but also creates a highly scalable environment for cross-platform applications and services.

MongoDB documents or collections of documents are the basic units of data. Formatted as Binary JSON (Java Script Object Notation), these documents can store various types of data and be distributed across multiple systems. Since MongoDB employs a dynamic schema design, users have unparalleled flexibility when creating data records, querying document collections through MongoDB aggregation and analyzing large amounts of information.

Comparing MongoDB to other databases

With so many database management solutions currently available, it can be hard to choose the right solution for your enterprise. Here are some common solution

comparisons and best use cases that can help you decide.

MongoDB vs. MySQL

MySQL (link resides outside IBM) uses a structured query language to access stored data. In this format, schemas are used to create database structures, utilizing tables as a way to standardize data types so that values are searchable and can be queried properly. A mature solution, MySQL is useful for a variety of situations including website databases, applications and commercial product management.

Because of its rigid nature, MySQL is preferable to MongoDB when data integrity and isolation are essential, such as when managing transactional data. But MongoDB's less-restrictive format and higher performance make it a better choice, particularly when availability and speed are primary concerns.

MongoDB vs. Cassandra

While Cassandra (link resides outside IBM) and MongoDB are both considered NoSQL databases, they have different strengths. Cassandra uses a traditional table structure with rows and columns, which enables users to maintain uniformity and durability when formatting data before it's compiled.

Cassandra can offer an easier transition for enterprises looking for a NoSQL solution because it has a syntax similar to SQL; it also reliably handles deployment and replication without a lot of configuration. However, it can't match MongoDB's flexibility for handling structured and unstructured data sets or its performance and reliability for mission-critical cloud applications.

MongoDB's JSON document model lets you store back-end application data wherever you need it, including in Apple iOS and Android devices as well as cloud-based storage solutions. This flexibility lets you aggregate data across

multiple environments with secondary and geospatial indexing, giving developers the ability to scale their mobile applications seamlessly.

Real-time analytics

As companies scale their operations, gaining access to key metrics and business insights from large pools of data is critical. MongoDB handles the conversion of JSON and JSON-like documents, such as BSON, into Java objects effortlessly, making the reading and writing of data in MongoDB fast and incredibly efficient when analyzing real-time information across multiple development environments. This has proved beneficial for several business sectors, including government, financial services and retail.

Content management systems

Content management systems (CMS) are powerful tools that play an important role in ensuring positive user experiences when accessing e-commerce sites, online publications, document management platforms and other applications and services. By using MongoDB, you can easily add new features and attributes to your online applications and websites using a single database and with high availability.

Enterprise Data Warehouse

The Apache Hadoop framework is a collection of open source modules, including Hadoop Distributed File System and Hadoop MapReduce, that work with MongoDB to store, process and analyze large amounts of data. Organizations can use MongoDB and Hadoop to perform risk modeling, predictive analytics and real-time data processing.

Benefits

Over the years, MongoDB has become a trusted solution for many businesses that are looking for a powerful and highly scalable NoSQL database. But MongoDB is much more than just a traditional document-based database and it boasts a few great capabilities that make it stand out from other DBMS.

Load balancing

As enterprises' cloud applications scale and resource demands increase, problems can arise in securing the availability and reliability of services. MongoDB's load balancing sharing process distributes large data sets across multiple virtual machines at once while still maintaining acceptable read and write throughputs. This horizontal scaling is called sharding and it helps organizations avoid the cost of vertical scaling of hardware while still expanding the capacity of cloud-based deployments.

Ad hoc database queries

One of MongoDB's biggest advantages over other databases is its ability to handle ad hoc queries that don't require predefined schemas. MongoDB databases use a query language that's similar to SQL databases and is extremely approachable for beginner and advanced developers alike. This accessibility makes it easy to push, query, sort, update and export your data with common help methods and simple shell commands.

Multilanguage support one of the great things about MongoDB is its multilanguage support. Several versions of MongoDB have been released and are in continuous development with driver support for popular programming languages, including Python, PHP, Ruby, Node.js, C++, Scala, JavaScript and many more.

MongoDB and IBM

For organizations seeking a better solution for managing their NoSQL databases while integrating into a multicloud environment, IBM Cloud® Databases for MongoDB provides a flexible and scalable solution for all their enterprise needs. By leveraging MongoDB's powerful indexing and querying capabilities with IBM's fully managed, secure cloud configurations, they get a highly sustainable and secure solution for enterprise database management.

Natively integrated and available in the IBM Cloud console, Databases for MongoDB provides seamless automation capabilities when maintaining, coordinating and monitoring your data structure across your entire infrastructure. With IBM's years of experience in enterprise development and database management at your disposal, you can let your team focus on creating better, more innovative solutions for your clients, knowing that your business's security, compliance, scalability and reliability are in the right hands.

To learn more about how easy it can be to deploy MongoDB in an enterprise setting and how you can maximize your team's efficiency, explore IBM Cloud Databases for MongoDB.

For a more in-depth look at MongoDB, check out Database Deep Dives: MongoDB.

IBM has also partnered with MongoDB to provide MongoDB Enterprise Advanced, a package that includes MongoDB Enterprise Server plus comprehensive support, security and advanced software tools. MongoDB Enterprise Advanced is available as an add-on for IBM Cloud Pak® for Data, a fully integrated, multicloud data and AI platform. Or you can integrate it into your existing data management solution for x86, IBM Power® and IBM Z® environments with IBM Data Management Platform for MongoDB Enterprise Advanced.

5.3 N-Tier Architecture

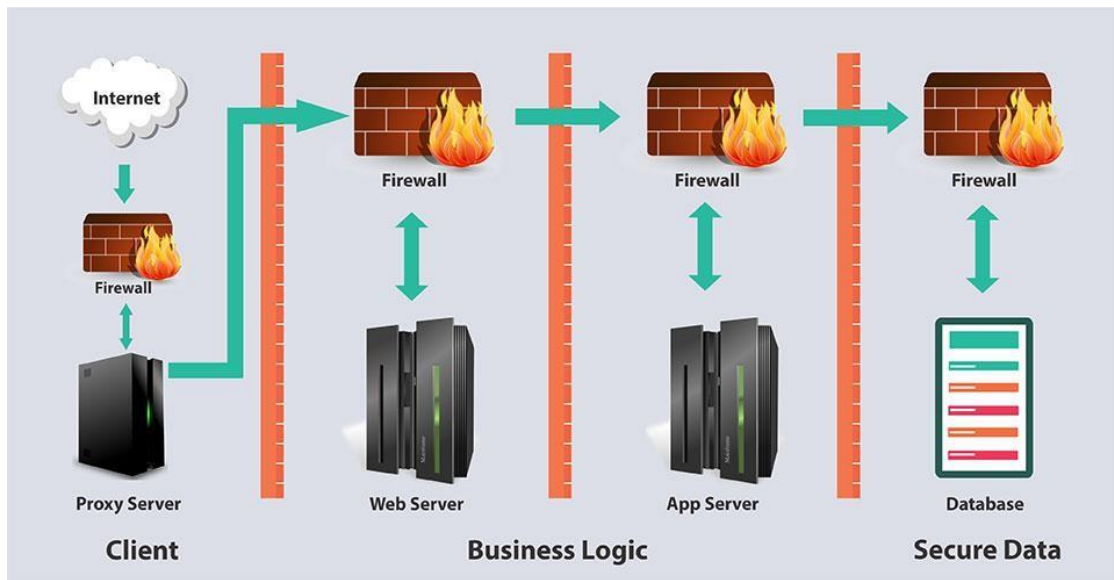


Fig 5.3.1 N-Tier Architecture

An N-tier architecture divides an application into **logical layers** and **physical tiers**.

Layers are a way to separate responsibilities and manage dependencies. Each layer has a specific responsibility. A higher layer can use services in a lower layer, but not the other way around.

Tiers are physically separated, running on separate machines. A tier can call to another tier directly, or use asynchronous messaging (message queue). Although each layer might be hosted in its own tier, that's not required. Several layers might be hosted on the same tier. Physically separating the tiers improves scalability and resiliency, but also adds latency from the additional network communication.

A traditional three-tier application has a presentation tier, a middle tier, and a database tier. The middle tier is optional. More complex applications can have

more than three tiers. The diagram above shows an application with two middle tiers, encapsulating different areas of functionality. An N-tier application can have a closed layer architecture or an open layer architecture:

In a closed layer architecture, a layer can only call the next layer immediately down.

In an open layer architecture, a layer can call any of the layers below it. A closed layer architecture limits the dependencies between layers. However, it might create unnecessary network traffic, if one layer simply passes requests along to the next layer.

CHAPTER – 6

FEATURES OF JAVASCRIPT

6.1 JavaScript

JavaScript is a lightweight, cross-platform, and interpreted compiled programming language which is also known as the scripting language for webpages. It is well-known for the development of web pages, many non-browser environments also use it. JavaScript can be used for Client-side developments as well as Server-side developments. JavaScript is both imperative and declarative type of language. JavaScript contains a standard library of objects, like Array, Date, and Math, and a core set of language elements like operators, control structures, and statements. Client-side: It supplies objects to control a browser and its Document Object Model (DOM). Like if client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation. Useful libraries for the client-side are Angular JS, React JS, Vue JS and so many others. Server-side: It supplies objects relevant to running JavaScript on a server. Like if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. The useful framework which is the most famous these days is node.js.

Imperative language – In this type of language we are mostly concern about how it is to be done. It simply control the flow of computation. The procedural programming approach, object, oriented approach comes under this like async await we are thinking what it is to be done further after async call.

Declarative programming – In this type of language we are concern about how it is to be done, basically here logical computation require. Here main goal is to describe the desired result without direct dictation on how to get it like arrow function do.

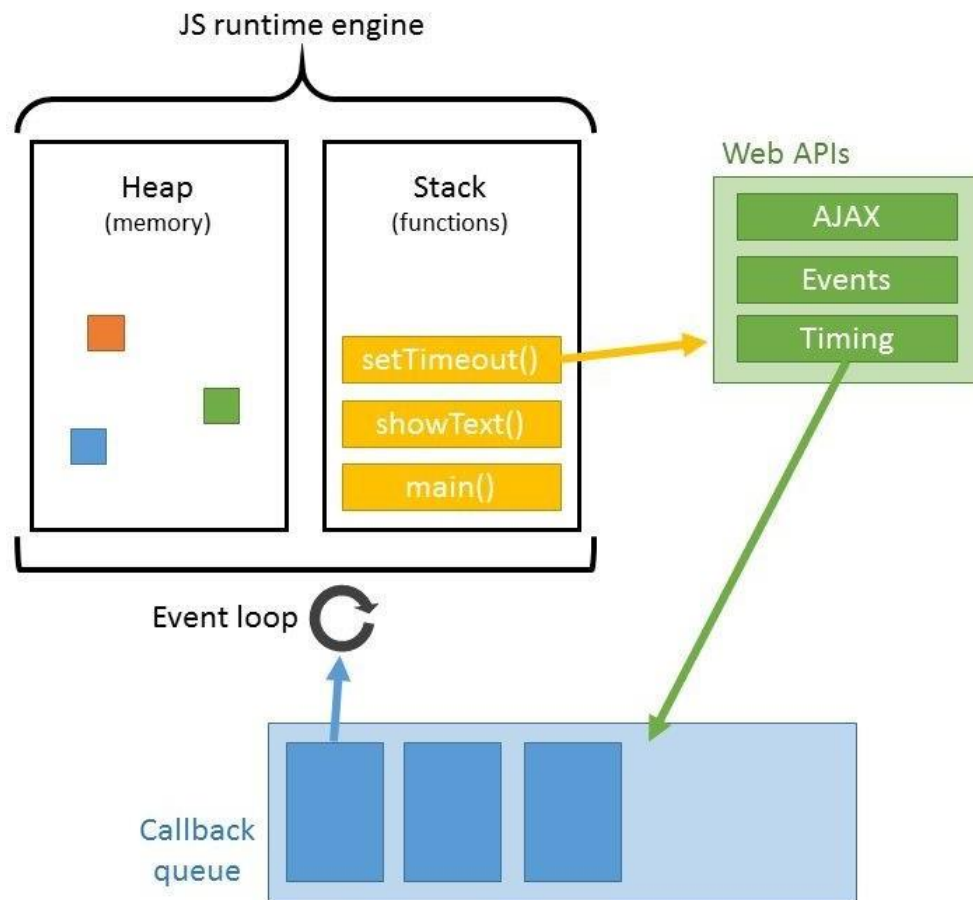


Fig 6.1 JavaScript Architecture

JavaScript can be added to your HTML file in two ways:

Internal JS: We can add JavaScript directly to our HTML file by writing the code inside the `<script>` tag. The `<script>` tag can either be placed inside the `<head>` or the `<body>` tag according to the requirement.

External JS: We can write JavaScript code in other file having an extension.js and then link this file inside the <head> tag of the HTML file in which we want to add this code.

The following are more characteristics of JavaScript:

1. Object-Centered Script Language
2. Client edge Technology
3. Validation of User's Input
4. Else and If Statement
5. Interpreter Centered
6. Ability to perform In Built Function
7. Case Sensitive format
8. Light Weight and delicate
9. Statements Looping
10. Handling Events

The Statement Control Syntax System of Java Script is very similar to Statement Control Syntax used in C Language. Java Script allows more abilities to perform as HTML is only capable of designing websites and incapable of performing logic operations like condition checking, statements looping (while & for), statement decision making (else and if) at client edge and adding two numbers.

JavaScript is incapable of performing these functions which results in our need for JavaScript to execute these operations over the client edge. This scripting language can help to build more interactive websites, developing inbuilt clocks, light client edge programs, building windows popup with different dialog boxes

like alerting dialog boxes, confirmation dialog boxes, and prompt dialog options.

Nowadays many web-based giants are using the technology of Java Script like Facebook, YouTube, Twitter, Gmail and Google Maps, etc. The Java Script features Enterprise level pop-down menus, Opening, and Closure of new window, manipulation of HTML Layers, Interface Enhancement of HTML, Adding animation essentials within the page, adding dynamic appearance in documents of HTML and Fixing of problems related to Browsers.

The Validation of data input within HTML Form data which is being sent to the server can be done using JavaScript. Users can also perform manipulation of data for layers of HTML like moving capability, hiding, change is the interface of HTML codes and writing them separately are some great features of Java Script.

6.2 REACT JS

React is a JavaScript-based UI development library. Facebook and an open-source developer community run it. Although React is a library rather than a language, it is widely used in web development. The library first appeared in May 2013 and is now one of the most commonly used frontend libraries for web development.

React offers various extensions for entire application architectural support, such as Flux and React Native, beyond mere UI.

Why React?

- Easy creation of dynamic applications: React makes it easier to create dynamic web applications because it requires less coding and offers more

functionality, as opposed to JavaScript, where coding often gets complex very quickly.

- Improved performance: React uses Virtual DOM, thereby creating web applications faster. Virtual DOM compares the components' previous states and updates only the items in the Real DOM that were changed, instead of updating all of the components again, as conventional web applications do.
- Reusable components: Components are the building blocks of any React application, and a single app usually consists of multiple components. These components have their logic and controls, and they can be reused throughout the application, which in turn dramatically reduces the application's development time.
- Unidirectional data flow: React follows a unidirectional data flow. This means that when designing a React app, developers often nest child components within parent components. Since the data flows in a single direction, it becomes easier to debug errors and know where a problem occurs in an application at the moment in question.
- Small learning curve: React is easy to learn, as it mostly combines basic HTML and JavaScript concepts with some beneficial additions. Still, as is the case with other tools and frameworks, you have to spend some time to get a proper understanding of React's library.
- It can be used for the development of both web and mobile apps: We already know that React is used for the development of web applications, but that's not all it can do. There is a framework called React Native, derived from React itself that is hugely popular and is used for creating

beautiful mobile applications. So, in reality, React can be used for making both web and mobile applications.

- Dedicated tools for easy debugging: Facebook has released a Chrome extension that can be used to debug React applications. This makes the process of debugging React web applications faster and easier.
- The above reasons more than justify the popularity of the React library and why it is being adopted by a large number of organizations and businesses. Now let's familiarize ourselves with React's features.

Features of React

React offers some outstanding features that make it the most widely adopted library for frontend app development. Here is the list of those salient features.



Fig 6.2 JSX

JSX is a JavaScript syntactic extension. It's a term used in React to describe how the user interface should seem. You can write HTML structures in the same file as JavaScript code by utilizing JSX.

React JS Prerequisites

Here Are Some Of The Concepts That You Should Be Familiar With, To One Degree Or Another:

- Programming Concepts Like Functions, Objects, Arrays, And To A Lesser Extent, Classes
- Basic Knowledge Of JavaScript
- Some Familiarity With Html

Now That You Know What Concepts You Should Already Be Familiar With Before Working On React, Let's Take A Look At The Industry Trends.

6.3 NODE JS

Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant. A Node.js app runs in a single process, without creating a new thread for every request.

ChaNNode.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm.

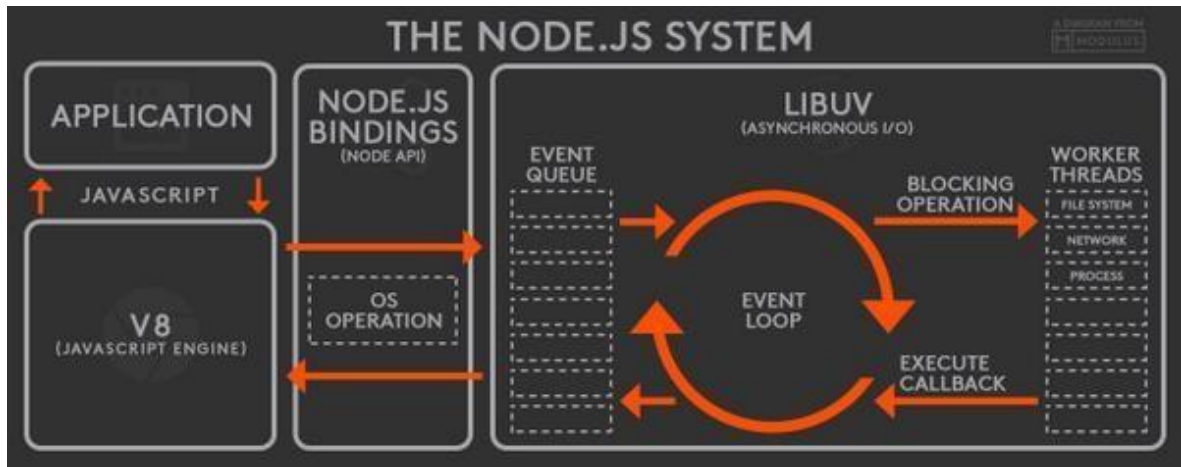


Fig 6.3 Node JS Architecture

When Node.js performs an I/O operation, like reading from the network, accessing a database or the file system, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.

This allows Node.js to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs.

Node.js has a unique advantage because millions of frontend developers that write JavaScript for the browser are now able to write the server-side code in addition to the client-side code without the need to learn a completely different language.

In Node.js the new ECMAScript standards can be used without problems, as you don't have to wait for all your users to update their browsers - you are in charge of deciding which ECMAScript version to use by changing the Node.js version, and you can also enable specific experimental features by running Node.js with flags.

- Adonis JS: A TypeScript-based fully featured framework highly focused on developer ergonomics, stability, and confidence. Adonis is one of the fastest Node.js web frameworks.
- Egg.js: A framework to build better enterprise frameworks and apps with Node.js & Koa.
- Express: It provides one of the simplest yet powerful ways to create a web server. Its minimalist approach, opinionated, focused on the core features of a server, is key to its success.
- Fastify: A web framework highly focused on providing the best developer experience with the least overhead and a powerful plugin architecture. Fastify is one of the fastest Node.js web frameworks.
- FeatherJS: Feathers is a lightweight web-framework for creating real-time applications and REST APIs using JavaScript or TypeScript. Build prototypes in minutes and production-ready apps in days.
- Gatsby: A React-based, GraphQL powered, static site generator with a very rich ecosystem of plugins and starters.
- hapi: A rich framework for building applications and services that enables developers to focus on writing reusable application logic instead of spending time building infrastructure.
- koa: It is built by the same team behind Express, aims to be even simpler and smaller, building on top of years of knowledge. The new project born out of the need to create incompatible changes without disrupting the existing community.
- Loopback.io: Makes it easy to build modern applications that require complex integrations.

- Meteor: An incredibly powerful full-stack framework, powering you with an isomorphic approach to building apps with JavaScript, sharing code on the client and the server. Once an off-the-shelf tool that provided everything, now integrates with frontend libs React, Vue, and Angular. Can be used to create mobile apps as well.
- Micro: It provides a very lightweight server to create asynchronous HTTP microservices.
- NestJS: A TypeScript based progressive Node.js framework for building enterprise-grade efficient, reliable and scalable server-side applications.
- Next.js: React framework that gives you the best developer experience with all the features you need for production: hybrid static & server rendering, TypeScript support, smart bundling, route pre-fetching, and more.
- Nx: A toolkit for full-stack monorepo development using NestJS, Express, React, Angular, and more! Nx helps scale your development from one team building one application to many teams collaborating on multiple applications!
- Remix: Remix is a fullstack web framework for building excellent user experiences for the web. It comes out of the box with everything you need to build modern web applications (both frontend and backend) and deploy them to any JavaScript-based runtime environment (including Node.js).
- Sapper: Sapper is a framework for building web applications of all sizes, with a beautiful development experience and flexible filesystem-based routing. Offers SSR and more!

- Socket.io: A real-time communication engine to build network applications.
- Strapi: Strapi is a flexible, open-source Headless CMS that gives developers the freedom to choose their favorite tools and frameworks while also allowing editors to easily manage and distribute their content. By making the admin panel and API extensible through a plugin system, Strapi enables the world's largest companies to accelerate content delivery while building beautiful digital experiences.

6.4 Express JS

Express is a node js web application framework that provides broad features for building web and mobile applications. It is used to build a single page, multipage, and hybrid web application.

It's a layer built on the top of the Node js that helps manage servers and routes.

Why Express JS?

- Express was created to make APIs and web applications with ease,
- It saves a lot of coding time almost by half and still makes web and mobile applications are efficient.
- Another reason for using express is that it is written in JavaScript as JavaScript is an easy language even if you don't have a previous knowledge of any language.
- Express lets so many new developers enter the field of web development.

The reason behind creating an express framework for node JS is:

- Time-efficient

- Fast
- Economical
- Easy to learn
- Asynchronous

Features of Express JS

- **Fast Server-Side Development**

The features of node js help express saving a lot of time.

- **Middleware**

Middleware is a request handler that has access to the application's request-response cycle.

- **Routing**

It refers to how an application's endpoint's URLs respond to client requests.

- **Templating**

It provides templating engines to build dynamic content on the web pages by creating HTML templates on the server.

- **Debugging**

Express makes it easier as it identifies the exact part where bugs are.

CHAPTER – 7

MODULES:

- A modular design reduces complexity, facilitates change (a critical aspect of software maintainability), and results in easier implementation by encouraging parallel development of different part of system. Software with effective modularity is easier to develop because function may be compartmentalized and interfaces are simplified. Software architecture embodies modularity that is software is divided into separately named and addressable components called modules that are integrated to satisfy problem requirements.
- Modularity is the single attribute of software that allows a program to be intellectually manageable. The five important criteria that enable us to evaluate a design method with respect to its ability to define an effective modular design are: Modular decomposability, Modular Comps ability, Modular Understand ability, Modular continuity, Modular Protection.
- The following are the modules of the project, which is planned in aid to complete the project with respect to the proposed system, while overcoming existing system and also providing the support for the future enhancement.

7.1 MODULES:

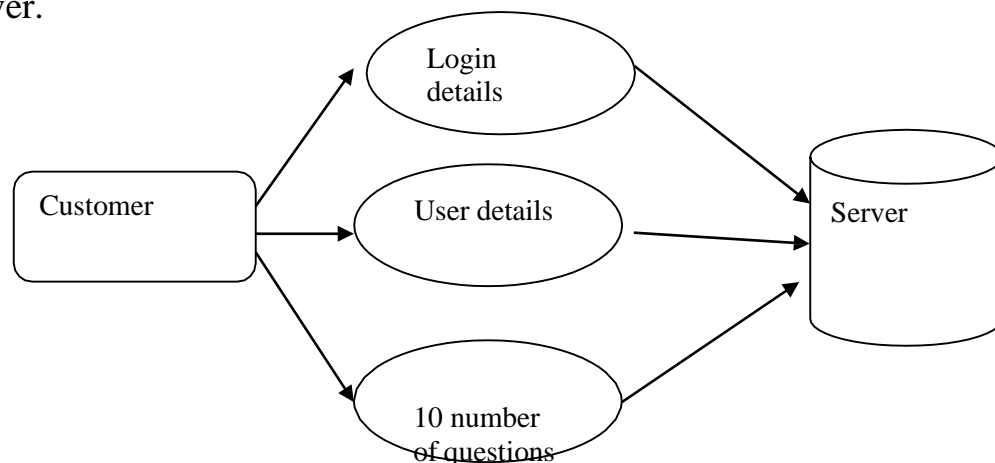
1. USER REGISTRATION
2. SERVER
3. HONEY WORDS GENERATION

4. INTERMEDIATE SERVER DEPLOYMENT & SHOPPING SERVER DEPLOYMENT
5. PASSWORD HACKING PROCESS
6. IDENTIFICATION OF ATTACKERS & AVOIDANCE OF DDOS ATTACK

7.2 MODULES DESCRIPTION:

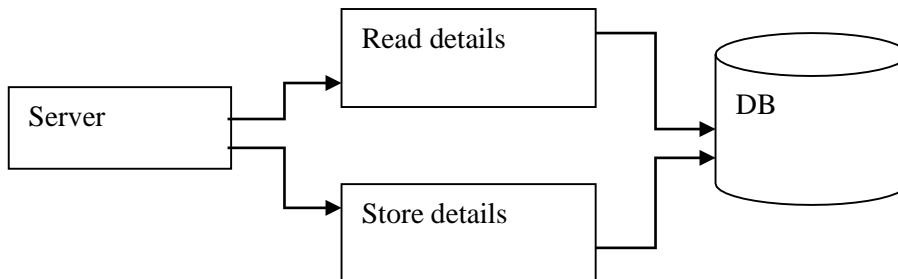
7.2.1. USER REGISTRATION

In this module we are going to create a User application by which the User is allowed to access the data from the Server. Here first the User wants to create an account and then only they are allowed to access the Network. Once the User creates an account, they are allowed to login into their account to access the application. Based on the User's request, the Server will respond to the User. All the User details will be stored in the Database of the Server. In this module bank user details are registered with the fields like username, password, and personal details with some set of questions and answers. These details are saved into the server. After proper registration only the user call allowed to login into the server.



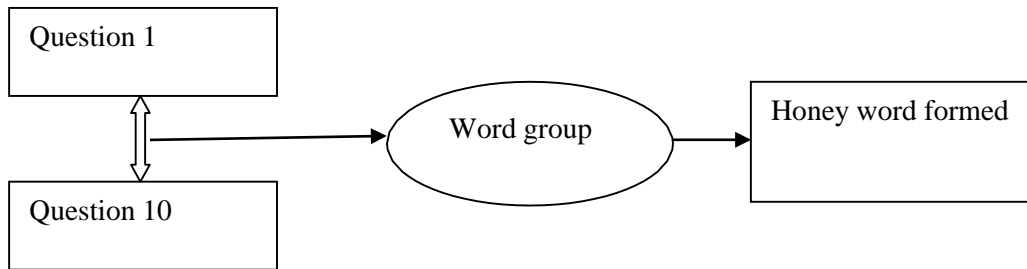
7.2.2. SERVER

In this Server module server will deployed to access the database and web-based application. Server will verify the users and generates honey word for save the users password. In case illegal actions happened, means server will generates alert and intimate it to user. The Server will monitor the entire User's information in their database and verify them if required. Also, the Server will store the entire User's information in their database. Also, the Server has to establish the connection to communicate with the Users. The Server will update each User's activities in its database. The Server will authenticate each user before they access the Application. So that the Server will prevent the Unauthorized User from accessing the Application.



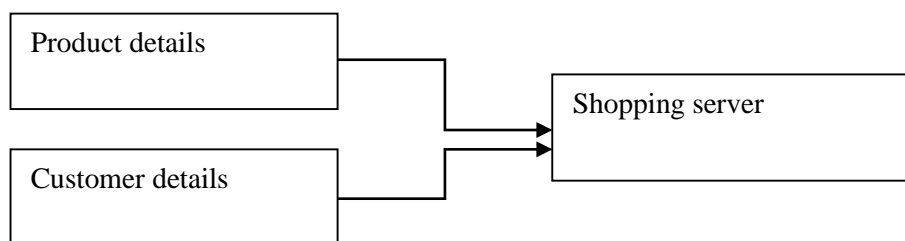
7.2.3. HONEY WORDS GENERATION

Password files have got a lot of security problem that has affected millions of users as well as many companies. Password file is generally stored in encrypted format, if a password file is stolen or theft by using the password cracking techniques and decryption technique it is easy to capture most of the plaintext and encrypt passwords. So, in this module we deployed honey word creations. That is the user's password and registered questions are combined and then it will generate a key as unknown Name.



7.2.4. INTERMEDIATE SERVER & SHOPPING SERVER DEPLOYMENT

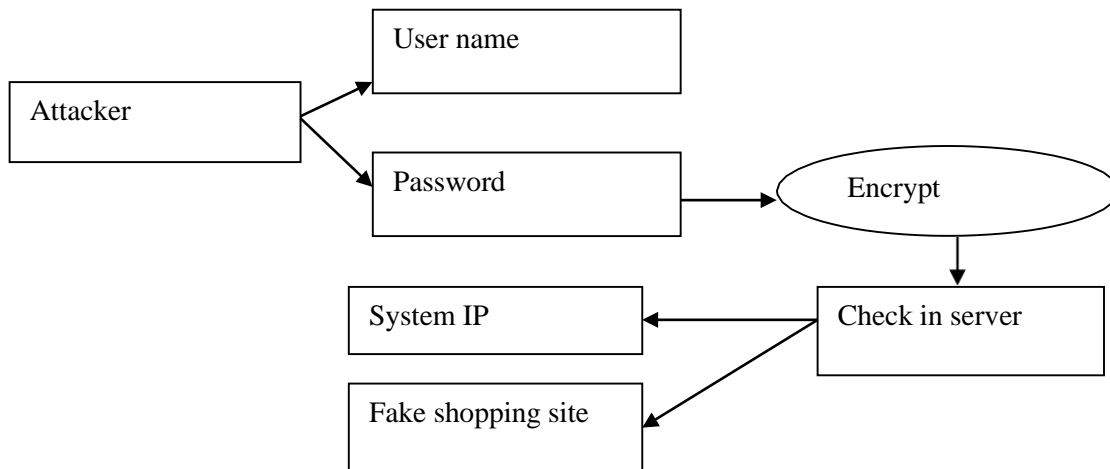
An intermediate server is a program that handles communications requests to a resource manager program on behalf of a user program. The user program can be referred to as a client of the intermediate server. Here we will generate the Intermediate server to make communication between user and Server. All requests come from the users are first sent to the intermediate server to verifies the password and user details. Shopping server is to collect the details from customer and sent to the details to the intermediate server for verification.



7.2.5. PASSWORD HACKING PROCESS

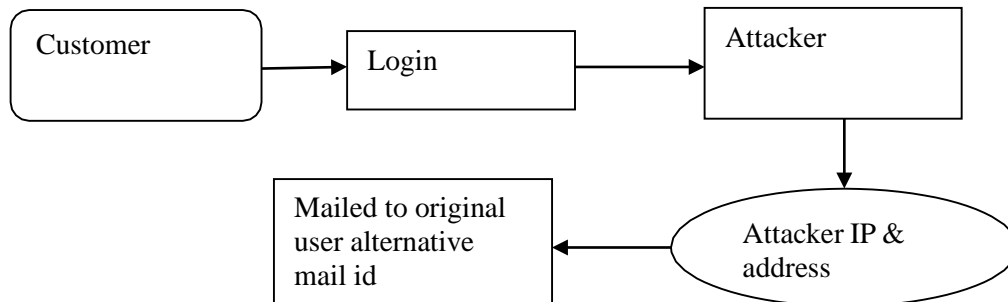
Hacking is the process of recovering passwords from data that has been stored in or transmitted by a computer system. A common approach is to repeatedly try guesses for the password. Another common approach is to say that you have

"forgotten" the password and then change it. Password Hacking is blocked in this Module. Because we modify the user's original passwords into unknown Name and saved into server.



7.2.6. IDENTIFICATION OF ATTACKERS & REMOVE DDOS ATTACKS

A distributed denial of service (DDOS) attack is an attempt to make an online service unavailable by overwhelming it with traffic from multiple sources. They target a wide variety of important resources, from banks to news websites and present a major challenge to making sure people can publish and access important information. If there is anybody trying with wrong password or any illegal action means server will block that action and intimate to the Specified Users. If the same request comes from same user or from different user's means server will blocks those actions also. This is done in DDOS attack.



CHAPTER – 8

SYSTEM TESTING

Generally testing process is carried out to ensure that the system has been developed according to the required specifications and the expected output is properly obtained. There are two main categories of testing namely, the White Box Testing and the Black Box Testing. Each of this testing in turn consists of many types of testing.

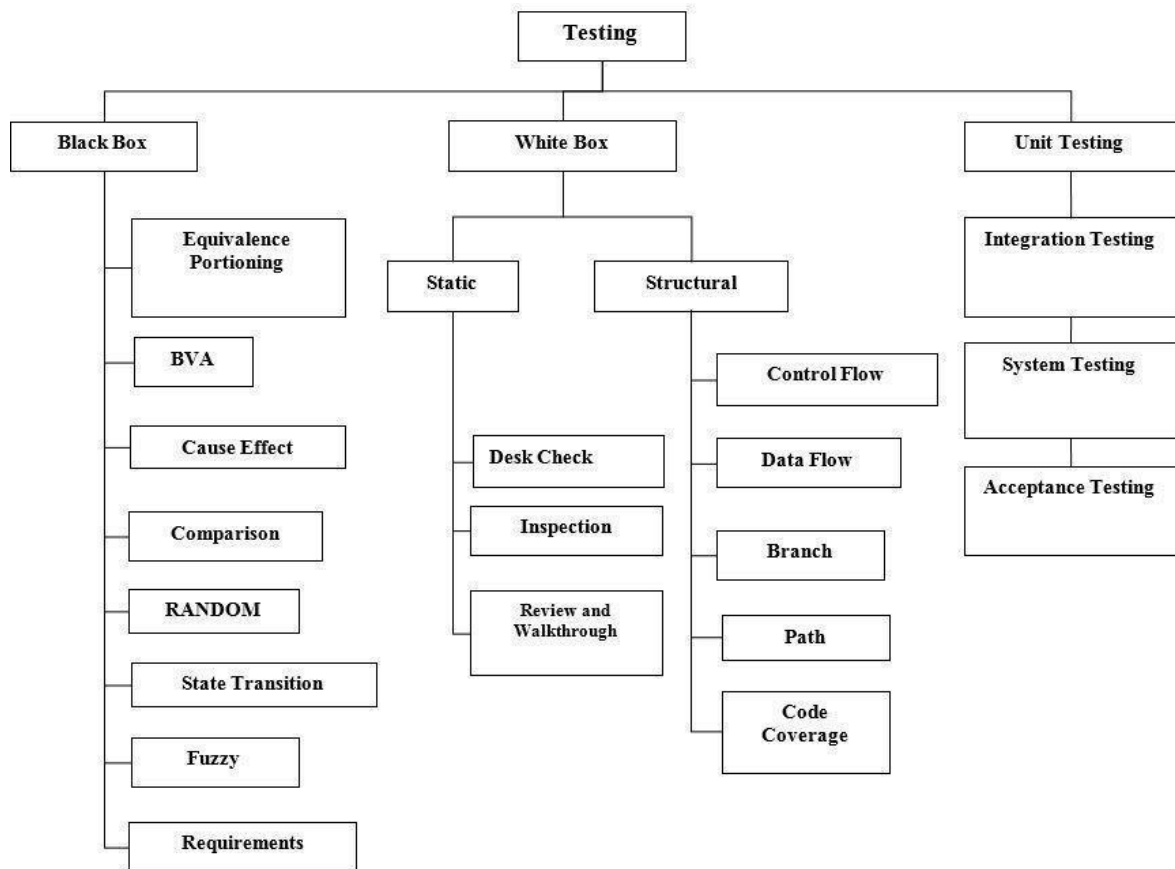


Fig 8.1 Taxonomy of Testing

8.2 WHITE BOX TESTING

White Box testing is also known as glass box testing. This type of testing, tests the internal structure of the program. This can be applied at the unit, integration and system levels of testing. Mostly, it is used in the unit level of the software testing process. Sometimes it may not reveal defects in areas which have not been implemented. It has its own advantages and its own disadvantages. The advantage is that knowing the programming language code and familiarizing with them may prove vital and help in identifying the errors quickly and at times may help in avoiding them at the earliest.

8.3 BLACK BOX TESTING

It is amongst the two methods of mostly used testing methods. This tests the main functionality of the program. It can be applied to every level of testing such as Unit, Integration, System and Acceptance levels of testing. Exhaustive input testing is required to find all errors. For doing this type of testing knowing the internal code and how works is not needed but what it is supposed to do is known by the person who is performing the test. The test cases are developed based on the specific requirements according to the goals. There are Boundary Value Analysis, Class Partitioning, and Cause Effect Graph etc.

8.4 UNIT TESTING:

Unit testing is also known as Module Testing which focuses on verification efforts on the module. The module is tested separately and this is carried out at the programming stage itself. Unit test comprises of the set of tests performed

by an individual programmer before integration of the unit into the system. This will help to test each and every single part or we can say as each and every module completely. These may even be small parts of the code and test cases will be developed which are independent of each other.

8.5 FUNCTIONAL TESTING:

Functional testing is mainly used as a Quality Assurance process. This is a very simple process where each function is provided with an appropriate input and is verified against an expected output and with boundary values. This would help in ensuring that the output is as acquired according to the expectations and would help assuring the quality. The various functions developed using Java are separately tested for their proper working by executing them as separate files.

8.6 INTEGRATION TESTING:

It is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with in the interface. It takes the unit tested modules and builds a program structure. Integration of all the components to form the entire system and an overall testing is executed. Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together as in a on the whole approach. Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed. It is mainly done

based on by taking into account of the number of modules used, how many number of interfaces maybe required to integrate them, which had to be combined and clustering process.

8.7 VALIDATION TESTING:

Validation test succeeds when the software functions in a manner that can be reasonably expected by the client. Software validation is achieved through a series of black box testing which confirms to the requirements. The software is 40 validated based on the series of tests that it passes through according to the condition posed by the customer. Mostly the customer main requirements would be to make every process as simple as possible and to reduce the complexity of the usage of the final product. Taking all these conditions into mind the validation testing is done and the various test cases are design.

8.8 SYSTEM TESTING:

System testing of software or hardware is testing conducted on complete, integrated system to evaluate the system's compliance with its specified requirements. Once all of the modules have been completely developed and both unit testing and integration testing is done on the various parts of the modules later the system testing is done so as to ensure that the requirements are fulfilled properly. All it basically does is it performs tests to find the discrepancies between the system and its original objective, current specifications and system documentation. If any discrepancy is to be found the

respective errors will be rectified and again system testing will be performed to make sure the rectification does not introduce a new error into the system.

8.9 STRUCTURE TESTING:

It is concerned with exercising the internal logic of a program and traversing certain execution paths. Structure testing takes into account of all process that works internally to make the entire system to work properly. The basic structure and the background codes and fragments that help in upholding the system are used.

This goes layer by layer as in till reaching the core process. The various layers or levels depends on the type of the project and the domain it comes under. This is considered to be a part of the White Box Testing Process. This is done so as to make sure all the internal processes work properly. If they don't 41 then the probability that the entire process may collapse is a possibility and this causes a grave danger to the project leading to failure.

8.10 TESTING IN PARTICULAR

8.10.1 UNIT TESTING

Unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use computer. And the result of the project is same in all system.

CHAPTER – 9

APPENDIX

9.1 SOURCE CODING:

9.1.1 FRONTEND (React JS)

Index.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";
import { BrowserRouter } from "react-router-dom";
import { AuthProvider } from "../Context/AuthProvider";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <AuthProvider>
        <App />
      </AuthProvider>
    </BrowserRouter>
  </React.StrictMode>
);
```

```
// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

App.jsx

```
import CartPage from "../Components/Cart/CartPage";
import LoginPage from "../Components/Login/LoginPage";
import RegisterPage from "../Components/Register/RegisterPage";
import StorePage from "../Components/Store/StorePage";
import { Route, Routes } from "react-router-dom";
import ProtectedRoutes from "../Components/ProtectedRoutes";
import AdminPage from "../Components/AdminPanel/AdminPage";
import Unauthorized from "../Components/Utils/Unauthorized";

function App() {
  // let role_data = localStorage.getItem("role");
  // role_data = JSON.parse(role_data);
  // console.log(role_data?.role);

  return (
    <div className="App">
      <Routes>
        <Route path="/" exact element={<LoginPage />} />
        <Route element={<ProtectedRoutes allowedRoles={["admin"]} />} />
        <Route path="/dashboard" element={<AdminPage />} />
      </Routes>
    </div>
  );
}
```

```

    </Route>
    <Route element={<ProtectedRoutes allowedRoles={["admin", "user"]}
/>}>
      <Route path="/register" element={<RegisterPage />} />
      <Route path="/store" element={<StorePage />} />
      <Route path="/cart" element={<CartPage />} />
    </Route>
    <Route path="/unauthorized" element={<Unauthorized />} />
    { /* <Route path="/store" element={<StorePage />} /> */ }
  </Routes>
</div>
);
}

```

```
export default App;
```

Login.jsx

```

import React, { useState } from "react";
import Box from "@mui/material/Box";
import Input from "@mui/material/Input";
import InputLabel from "@mui/material/InputLabel";
import InputAdornment from "@mui/material/InputAdornment";
import FormControl from "@mui/material/FormControl";
import TextField from "@mui/material/TextField";
import FormHelperText from "@mui/material/FormHelperText";
import Visibility from "@mui/icons-material/Visibility";
import VisibilityOff from "@mui/icons-material/VisibilityOff";

```



```

import IconButton from "@mui/material/IconButton";
import EmailIcon from "@mui/icons-material/Email";
import PasswordIcon from "@mui/icons-material/Password";
import Button from "@mui/material/Button";
import { Link } from "react-router-dom";
import Snackbar from "@mui/material/Snackbar";
import "./login.css";
import axios from "axios";
import MuiAlert from "@mui/material/Alert";
import { useNavigate } from "react-router-dom";
import useAuth from "../Hooks/useAuth";

const Alert = React.forwardRef(function Alert(props, ref) {
  return <MuiAlert elevation={6} ref={ref} variant="filled" {...props} />;
});

function LoginPage() {
  const navigate = useNavigate();

  const { setAuth } = useAuth();

  const [form, setForm] = useState({
    email_id: "",
    password: "",
    showPassword: false,
  });

```

```

const [alert, setAlert] = useState({ msg: "", type: "error" });
const [wrongPwdCount, setWrongPwdCount] = useState(0);
const incrementCount = () => {
  console.log("count");
  setWrongPwdCount(wrongPwdCount + 1);
};
const handleChange = (event) => {
  setForm({ ...form, [event.target.name]: event.target.value });
};

const openAlertShowPassword = () => {
  setForm({
    ...form,
    showPassword: !form.showPassword,
  });
};

const handleMouseDownPassword = (event) => {
  event.preventDefault();
};

const handleSubmit = async () => {

  if (form.email_id !== "" && form.password !== "") {
    console.log(wrongPwdCount);
    try {

```

```

const response = await axios
  .post(`http://localhost:5000/api/user/login/${form.email_id}`)
  .then((response) => {
    console.log(response.data);
    console.log(form);
    if (form.password === response.data[0].Password) {
      showAlert({ msg: "Login Success", type: "success" });
      console.log("login", { role: response.data[0].role });
      setAuth({ role: [response.data[0].role] });
      if (response.data[0].role === "admin") {
        navigate("/dashboard");
      } else {
        navigate("/store");
      }
    } else {
      showAlert({
        msg: "Email ID or Password is Wrong",
        type: "error",
      });
    }
  });
} catch (error) {
  console.log(error?.response?.data);
  incrementCount();
  showAlert({ msg: error?.response?.data[1].message, type: "error" });
  if (wrongPwdCount >= 2) {
    showAlert({ msg: "Login Success", type: "success" });
  }
}

```

```

    localStorage.setItem(
      "role",
      JSON.stringify({ role: "unauthorized" })
    );
    const response = await axios
      .post(`http://localhost:5000/api/user/email/`, {
        data: { emailID: form.email_id, name: "keerthana" },
      })
      .then((response) => {
        console.log(response);
      });
    navigate("/store");
  }
}
};

```

```

const [open, setOpen] = React.useState(false);

```

```

const openAlert = (data) => {
  setAlert(data);
  setOpen(true);
};

```

```

const handleClose = (event, reason) => {
  if (reason === "clickaway") {
    return;
  }
}

```

```

    }

    setOpen(false);
};

return (
    <div className="login-container">
        <div className="login-form">
            <div className="login-header">Login</div>
            <div className="login-body">
                <Box sx={{ display: "flex", alignItems: "flex-end" }}>
                    <EmailIcon sx={{ color: "action.active", mr: 1, my: 0.5 }} />
                    <TextField
                        id="input-with-sx"
                        label="Email ID"
                        variant="standard"
                        sx={{ width: "300px" }}
                        name="email_id"
                        value={form.email_id}
                        onChange={handleChange}
                        error={false}
                        helperText=""
                    />
                </Box>
                <Box sx={{ display: "flex", alignItems: "flex-end" }}>
                    <PasswordIcon sx={{ color: "action.active", mr: 1, my: 0.5 }} />
                    <FormControl variant="standard">

```

```

<InputLabel htmlFor="standard-adornment-password">
  Password
</InputLabel>
<Input
  error={false}
  sx={{ width: "300px" }}
  id="standard-adornment-password"
  type={form.showPassword ? "text" : "password"}
  name="password"
  value={form.password}
  onChange={handleChange}
  endAdornment={
    <InputAdornment position="end">
      <IconButton
        aria-label="toggle password visibility"
        onClick={openAlertShowPassword}
        onMouseDown={handleMouseDownPassword}
      >
        {form.showPassword ? <VisibilityOff /> : <Visibility />}
      </IconButton>
    </InputAdornment>
  }
/>
<FormHelperText                                id="standard-weight-helper-
text"></FormHelperText>
</FormControl>
</Box>

```

```

</div>
<div className="login-btn">
  <Button
    sx={{ width: "200px" }}
    variant="contained"
    onClick={handleSubmit}
  >
    Login
  </Button>
</div>
<div className="login-footer">
  <span>
    Not a Member ?
    <Link to={{ pathname: "/register", hash: "#faq-1" }}>
      Register Now!
    </Link>
  </span>
</div>
</div>
<Snackbar
  anchorOrigin={{ vertical: "top", horizontal: "center" }}
  open={open}
  autoHideDuration={6000}
  onClose={handleClose}
>
  <Alert
    onClose={handleClose}

```

```

        severity={alert.type}
        sx={{ width: "100%" }}
      >
        {alert.msg}
      </Alert>
    </Snackbar>
  </div>
);
}

```

export default LoginPage;

Register.jsx

```

import React, { useState } from "react";
import OutlinedInput from "@mui/material/OutlinedInput";
import Input from "@mui/material/Input";
import InputLabel from "@mui/material/InputLabel";
import InputAdornment from "@mui/material/InputAdornment";
import FormControl from "@mui/material/FormControl";
import TextField from "@mui/material/TextField";
import FormHelperText from "@mui/material/FormHelperText";
import Visibility from "@mui/icons-material/Visibility";
import VisibilityOff from "@mui/icons-material/VisibilityOff";
import IconButton from "@mui/material/IconButton";
import EmailIcon from "@mui/icons-material/Email";
import PasswordIcon from "@mui/icons-material/Password";
import Button from "@mui/material/Button";

```



```

import { AdapterDateFns } from "@mui/x-date-pickers/AdapterDateFns";
import { LocalizationProvider } from "@mui/x-date-pickers/LocalizationProvider";
import { DatePicker } from "@mui/x-date-pickers/DatePicker";
import "./register.css";
import axios from "axios";
import Swal from "sweetalert2";

const questions = [
  {
    question: "What is the name of your first pet?",
    answer: "",
  },
  {
    question: "What was your first car?",
    answer: "",
  },
  {
    question: "What elementary school did you attend?",
    answer: "",
  },
  {
    question: "What is the name of the town where you were born?",
    answer: "",
  },
];

```

```

function RegisterPage() {
  const init_form = {
    fName: "",
    lName: "",
    // username: "",
    email_id: "",
    dob: "",
    password: "",
    cpassword: "",
    showPassword: false,
    showCPassword: false,
  };
  const [form, setForm] = useState(init_form);

  const [date, setDate] = useState(null);

  const [questionForm, setQuestionForm] = useState(questions);
  const [isRegisterForm, setIsRegisterForm] = useState(true);

  const handleQuestion = (event, index) => {
    const cloneQuestion = [...questionForm];
    cloneQuestion[index].answer = event.target.value;
    setQuestionForm(cloneQuestion);
  };

  const handleClickShowPassword = () => {
    setForm({

```

```

    ...form,
    showPassword: !form.showPassword,
  });
};

const handleMouseDownPassword = (event) => {
  event.preventDefault();
};

const handleClickShowCPassword = () => {
  setForm({
    ...form,
    showCPassword: !form.showCPassword,
  });
};

const handleMouseDownCPassword = (event) => {
  event.preventDefault();
};

const handleChange = (event) => {
  setForm({ ...form, [event.target.name]: event.target.value });
};

const handleOpen = () => {
  setIsRegisterForm(false);
};

```

```
const handleClose = () => {  
  setIsRegisterForm(true);  
};
```

```
const handleSubmit = () => {  
  handleOpen();  
  if (!isRegisterForm) {  
    const data = {  
      firstName: form.fName,  
      lastName: form.lName,  
      emailID: form.email_id,  
      DOB: date,  
      Password: form.password,  
      Questions: questionForm,  
    };  
  }
```

```
  axios  
    .post("http://localhost:5000/api/user/", { data })  
    .then((res) => {  
      console.log("res", res.message);  
      Swal.fire("Register!", `New User is Added`, "success");  
      setForm(init_form);  
      setDate(null);  
      setQuestionForm(questions);  
      handleClose();  
    })
```

```

        .catch((err) => {
            console.log(err);
            Swal.fire("OOPS", "Register Failed!", "error");
        });

        console.log(data);
    }
};

return (
    <div className="register-container">
        <div className="register-form">
            <div className="register-header">Register</div>
            {isRegisterForm ? (
                <div className="register-body">
                    <div className="register-item">
                        <TextField
                            label="First Name"
                            variant="outlined"
                            sx={{ width: "250px" }}
                            name="fName"
                            value={form.fName}
                            onChange={handleChange}
                            error={false}
                            helperText=""
                            size="small"
                        />

```

```

name="email_id"
value={form.email_id}
onChange={handleChange}
error={false}
helperText=""
/>

```

```

<TextField

```

```

label="Confirm Password"
variant="outlined"
sx={{ width: "500px" }}
name="email_id"
value={form.email_id}
onChange={handleChange}
error={false}
helperText=""
/> */}

```

```

<FormControl variant="outlined">

```

```

  <InputLabel htmlFor="outlined-adornment-password" size="small">

```

```

    Password

```

```

  </InputLabel>

```

```

  <OutlinedInput

```

```

    type={form.showPassword ? "text" : "password"}

```

```

    name="password"

```

```

    value={form.password}

```

```

    onChange={handleChange}

```

```

    sx={{ width: "500px" }}

```

```

size="small"
endAdornment={
  <InputAdornment position="end">
    <IconButton
      aria-label="toggle password visibility"
      onClick={handleClickShowPassword}
      onMouseDown={handleMouseDownPassword}
      edge="end"
    />
    {form.showPassword ? <VisibilityOff /> : <Visibility />}
  </IconButton>
</InputAdornment>
}
label="Password"
/>
</FormControl>
<FormControl variant="outlined">
  <InputLabel htmlFor="outlined-adornment-password" size="small">
    Confirm Password
  </InputLabel>
  <OutlinedInput
    type={form.showCPassword ? "text" : "password"}
    name="cpassword"
    value={form.cpassword}
    onChange={handleChange}
    sx={{ width: "500px" }}
    size="small"

```

```

        variant="contained"
        sx={{ width: "200px" }}
        onClick={handleSubmit}
      >
        {isRegisterForm ? "continue" : "register"}
      </Button>
    </div>
  </div>
</div>
);
}

```

```
export default RegisterPage;
```

Store.jsx

```

import React, { useState } from "react";
import Card from "@mui/material/Card";
import CardActions from "@mui/material/CardActions";
import CardContent from "@mui/material/CardContent";
import CardMedia from "@mui/material/CardMedia";
import Button from "@mui/material/Button";
import Typography from "@mui/material/Typography";
import { InputLabel, MenuItem, TextField } from "@mui/material";
import img from "../../Images/iphone13.jfif";
import "./store.css";

```



```

import Navbar from "../Navbar/Navbar";
import { useNavigate } from "react-router-dom";
import { useEffect, useContext } from "react";
import AuthContext from "../Context/AuthProvider";

function StorePage() {
  let navigate = useNavigate();

  const { auth } = useContext(AuthContext);
  console.log("Authcontext: ", auth);

  const [cart, setCart] = useState([]);
  const [Local, setLocal] = useState();
  // useEffect(() => {
  //   localStorage.setItem("products", JSON.stringify([{}]));
  // }, []);

  useEffect(() => {
    const data = localStorage.getItem("products");
    if (data) {
      setLocal(JSON.parse(data));
    } else {
      localStorage.setItem("products", JSON.stringify([]));
    }
  }, []);
  const handleCart = (item) => {
    console.log(item);
  }

```

```

Local.push(item);
console.log(Local);
localStorage.setItem("products", JSON.stringify(Local));
// navigate("/cart", { state: { data: "" } });
};
return (
  <div>
    <Navbar />
    <div className="store-container">
      {products.map((item) => {
        return (
          <Card sx={{ width: "350px" }}>
            <CardMedia
              component="img"
              height="400"
              image={item.img}
              alt={item.productName}
            />
            <CardContent>
              <h2 variant="h5" component="div">
                {item.productName}
              </h2>
              <Typography variant="h5" component="div">
                <span> &#x20b9; {item.price}</span>
                <s> &#x20b9; 10,734</s>
                <span> -40%</span>
              </Typography>

```

```

</CardContent>
<CardActions sx={{ float: "right" }}>
  <TextField
    type="number"
    label="Qty"
    size="small"
    InputProps={{ inputProps: { min: 0, max: 10 } }}
    sx={{ width: "80px", marginLeft: "10px" }}
  />
  <Button
    variant="contained"
    size="small"
    sx={{ marginLeft: "10px" }}
    onClick={() => handleCart(item)}
  >
    Add to Cart
  </Button>
</CardActions>
</Card>
);
}}

```

```

{/* <Card sx={{ maxWidth: 345 }}>
  <CardMedia
    component="img"
    height="400"
    image={img}

```

```

    height="400"
    image={img}
    alt="green iguana"
  />
  <CardContent>
    <Typography variant="h5" component="div">
      iPhone 13
    </Typography>
  </CardContent>
  <CardActions sx={{ float: "right" }}>
    <TextField
      type="number"
      label="Qty"
      size="small"
      sx={{ width: "80px", marginLeft: "10px" }}
    />
    <Button
      variant="contained"
      size="small"
      sx={{ marginLeft: "10px" }}
    >
      Add to Cart
    </Button>
  </CardActions>
</Card>
<Card sx={{ maxWidth: 345 }}>
  <CardMedia

```

```

        component="img"
        height="400"
        image={img}
        alt="green iguana"
      />
    <CardContent>
      <Typography variant="h5" component="div">
        iPhone 13
      </Typography>
    </CardContent>
    <CardActions sx={{ float: "right" }}>
      <TextField
        type="number"
        label="Qty"
        size="small"
        sx={{ width: "80px", marginLeft: "10px" }}
      />
      <Button
        variant="contained"
        size="small"
        sx={{ marginLeft: "10px" }}
      >
        Add to Cart
      </Button>
    </CardActions>
  </Card> */}
</div>

```

```
    </div>  
  );  
}
```

```
export default StorePage;
```

9.1.1 BACKEND (Node JS):

Server.js

```
const express = require('express');  
const bodyparser = require('body-parser')  
const cors = require('cors')  
const connection = require("./dbcon")  
const user = require("./routes/user.route")
```

```
const app = express();  
connection()  
var PORT = process.env.port || 5000
```

```
app.use(bodyparser.urlencoded({ extended:false } ))  
app.use(bodyparser.json())
```

```
app.use(cors())
```

```
//routes  
app.use("/api/user", user)
```

```
var server = app.listen(PORT, (error)=> {  
  console.log("Server is listening on",PORT)  
})
```

dbcon.js

```
const { mongoose } = require("mongoose")  
  
const dbname = "admin";  
  
const DB_URL = `mongodb://localhost:27017/${dbname}`  
module.exports = () =>{  
  const connectionparams = {  
    useNewUrlParser:true,  
    useUnifiedTopology:true  
  };  
  try {  
    mongoose.connect(DB_URL, connectionparams);  
    console.log("DB is Connected")  
  } catch (error) {  
    console.log("DB Connection Error")  
  }  
}
```

User.route.js

```
const express = require("express");
const UserDB = require("../model/user.model");
const sendEmail = require("../utils/email")
const requestIp = require('request-ip');
const address = require('address');
const dropbox = require("../utils/dropbox")
const fs = require('fs')
const router = express.Router();
```

```
router.get("/", async (req, res) => {
  const email = req.params.email;
  const users = await UserDB.find();
  try {
    console.log(users);
    res.json(users);
  } catch (error) {
    console.log("err");
    res.json({});
  }
});
```

//Login User

```
router.post("/login/:email", async (req, res) => {
  const email = req.params.email;
  const [users] = await UserDB.find({ emailID: email });
  try {
```



```

    console.log({
      emailID: users.emailID,
      Password: users.Password,
      role: users.role,
    });
    res.json([
      { emailID: users.emailID, Password: users.Password, role: users.role },
      { message: "Login Success" },
    ]);

  } catch (error) {
    // console.log("err");
    res.status(400).json([{}], { message: "Email ID or Password is Wrong" });
  }
});

//Register User
router.post("/", async (req, res) => {
  const data = req.body.data;
  console.log(data);

  const user = new UserDB(data);

  try {
    const result = await user.save();
    res.status(200).json({ message: "User add successfully" });
  }
});

```

```

    // const fullName = `${users.firstName} ${users.lastName}`
    sendEmail("welcome", data.emailID, data.lastName)
  } catch (error) {
    console.log(error);
    res.status(400).json({ message: "failed" });
  }
});

router.get("/ip", (req, res)=>{
  // console.log(req.socket.remoteAddress);
  // console.log(req.ip);

  // var clientIp = requestIp.getClientIp(req);
  // console.log(clientIp);
  // res.send("your IP is: " + clientIp);

  // default interface 'eth' on linux, 'en' on osx.
  console.log(address.ip()) // '192.168.0.2'
  res.send("Your IP is: " + address.ip())
  // console.log(address.ipv6()) // 'fe80::7aca:39ff:feb0:e67d'
  // address.mac(function (err, addr) {
  // console.log(addr); // '78:ca:39:b0:e6:7d'
  // });

})

```

```

function generate_filename(length) {
  var result = "";
  var characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
  var charactersLength = characters.length;
  for ( var i = 0; i < length; i++ ) {
    result += characters.charAt(Math.floor(Math.random() *
charactersLength));
  }
  return result;
}

const fileName = generate_filename(8) + ".json"
console.log(fileName);

router.post("/drop", async (req, res) => {
  const data = req.body.data
  // console.log(data);
  const fileName = generate_filename(8) + ".json"

  try{
    fs.writeFile(`./data/${fileName}`, JSON.stringify(data), function (err) {
      if (err) {
        return console.log(err);
      }
      // console.log(data);

```

```

    res.send(`${fileName} is saved!`)
  });
}
catch(error){
  console.log(error)
}

```

```

dropbox(`./data/${fileName}`)
response.send("ok")
});

```

```

router.post("/email", (req, res)=>{
  const IP =address.ip()
  const data = req.body.data;
  sendEmail("alert", data.emailID, data.name, IP)
})

```

```

module.exports = router;

```

user.model.js

```

const mongoose = require("mongoose");

```

```

const UserSchema = new mongoose.Schema({

```

```
role:{
  type: String,
  default:"user"
},
firstName: {
  type: String,
  required: true,
},
lastName: {
  type: String,
  required: true,
},
emailID: {
  type: String,
  required: true,
},
DOB:{
  type: Date,
  required: true,
},
Password:{
  type: String,
  required: true,
},
Questions:{
  type: Object,
  required: true,
```

```
},  
});
```

```
const User = mongoose.model("User", UserSchema);
```

```
module.exports = User;
```

email.js

```
const nodemailer = require("nodemailer");
```

```
function sendEmail(mailType, to_mail, userName, IP="") {
```

```
    // const IP = "192.168.179.5";
```

```
    // const userName = "Gyana Madhavan";
```

```
    const link = "/";
```

```
    let MailTemplate="";
```

```
    let subject =""
```

```
    switch (mailType) {
```

```
        case "welcome":
```

```
            MailTemplate = Welcome_HTML
```

```
            subject = "Welcome Mail"
```

```
            break;
```

```
        case "alert":
```

```

    MailTemplate = Alert_HTML
    subject = "Alert Mail"
    break;
default:
    MailTemplate=""
    break;
}

if (MailTemplate === ""){
    console.log("Mail Template not found")
    return
}

// const { to, subject, message } = data;
const from = "gyanamadhavan@gmail.com";
console.log(to_mail, subject);

if(to_mail!="" & subject!=""){

    const transporter = nodemailer.createTransport({
        service: "gmail",
        auth: {
            user: env.MAIL,
            pass: env.PWD,
        },
    });

```

```

const mailOptions = {
  from: from,
  to: to_mail,
  subject: subject,
  html: MailTemplate,
};

transporter.sendMail(mailOptions, function (error, info) {
  if (error) {
    console.log(error);
    res.status(400).send("Failed to send Email!");
  } else {
    console.log("Email Sent" + info.response);
    res.status(200).send("Email Sent!");
  }
});
}
else{
  console.log("To Mail ID or Subject is missing")
}
}

```

```

module.exports = sendEmail;

```

dropbox.js

```

const fs = require('fs')
const https = require("https");

```



```
const data = "{\"limit\": 1000}";
```

```
const TOKEN = env.TOKEN
```

```
function UploadDropbox(filePath) {
  fs.readFile(filePath, 'utf8', function (err, data) {
    const req = https.request('https://content.dropboxapi.com/2/files/upload', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${TOKEN}`,
        'Dropbox-API-Arg': JSON.stringify({
          'path': '/Upload/user.json',
          'mode': 'overwrite',
          'autorename': true,
          'mute': false,
          'strict_conflict': false
        }),
        'Content-Type': 'application/octet-stream',
      }
    }, (res) => {
      res.on('data', function(d) {
        process.stdout.write(d);
      });
    });


    req.write(data);
    req.end(); }); module.exports = UploadDropbox;
```

9.3 SCREENSHOTS:



Login

Email ID

Password 

LOGIN

Not a Member ? [Register Now!](#)


Fig 9.2.1 Login Page





Register

First Name Last Name

Email ID

DOB 

Password 

Confirm Password 

CONTINUE

Fig 9.2.2 Register Page

SmartCart x +

localhost:3000/register#faq-1

Register

Please Fill below questions

What is the name of your first pet?

What was your first car?

What elementary school did you attend?

What is the name of the town where you were born?

BACK REGISTER

Fig 9.2.3 Register Security Questions Page

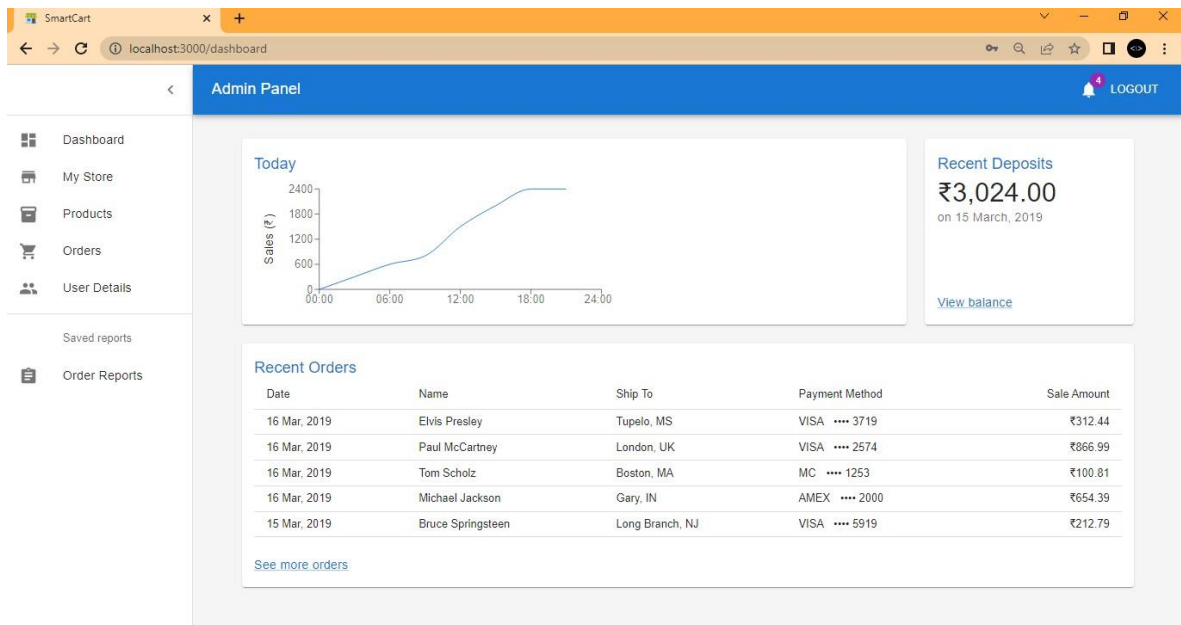


Fig 9.2.4 Admin Panel

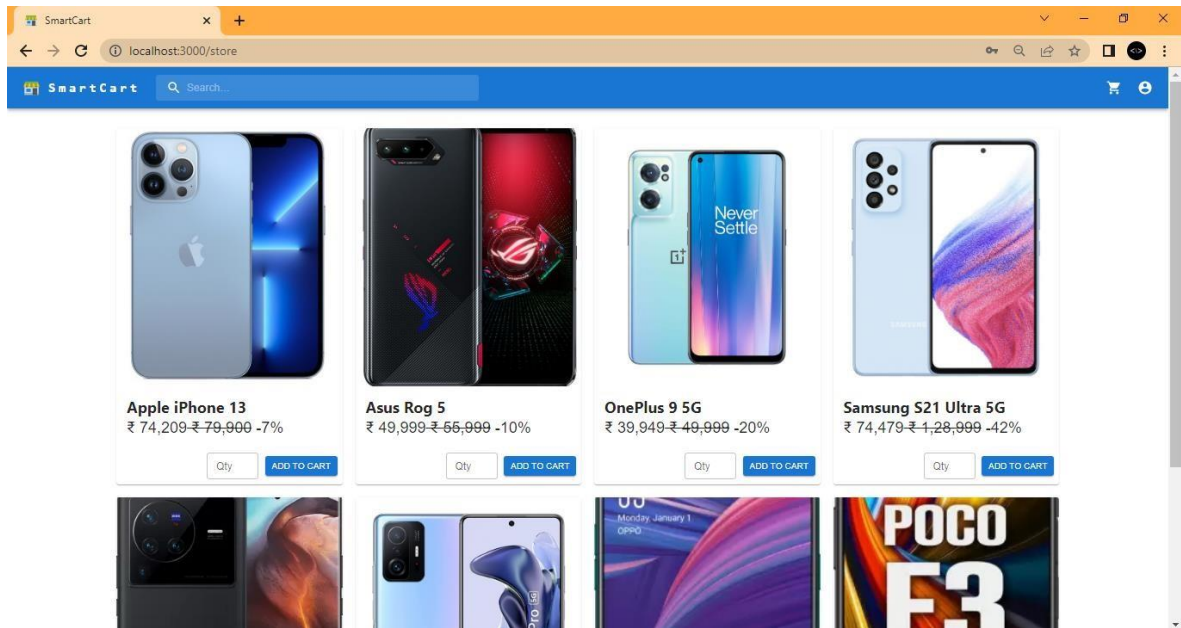


Fig 9.2.5 Store Page

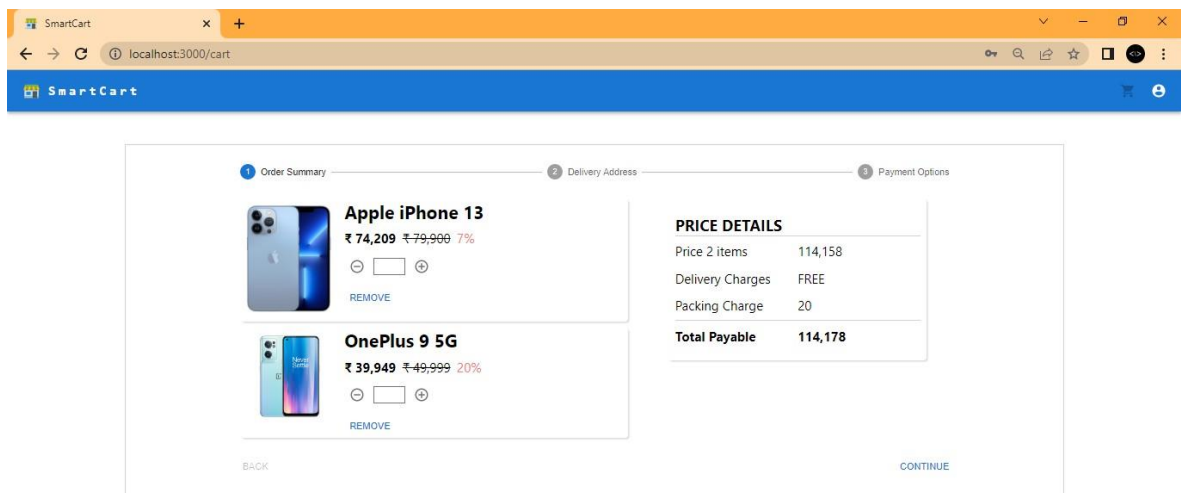


Fig 9.2.6 Cart Page

SmartCart

localhost:3000/cart

SmartCart

Order Summary ☒ Delivery Address ☒ Payment Options ☐

Name 10-digit mobile number

incorrect incorrect

Address

incorrect

Pincode City/District/Town

incorrect incorrect

State Landmark (Optional)

incorrect

Address Type

☒ Home (All Days delivery) ☐ Work (Delivery between 10 AM - 5 AM)

DELIVERY HERE CANCEL

BACK CONTINUE

Fig 9.2.7 Shipping Address Page

SmartCart

localhost:3000/cart

SmartCart

Order Summary ☒ Delivery Address ☒ Payment Options ☒

Your saved credit and debit cards

☒ Credit Card xxxx xxxx xxxx 4822

CVV EDIT CARD

☐ Debit Card xxxx xxxx xxxx 7832

☐ Add Credit/Debit/ATM Card

Another payment method

☐ UPI

☐ Pay on Delivery

BACK PAY

Fig 9.2.8 Payment Options Page

CHAPTER – 10

CONCLUSION

10.1 CONCLUSIONS:

In this project, we have to implement secured purchasing system in online. Honey words are generated based on the user info provided and the original password is converted into another format and stored along with the Honey words. If identify the hacker means, Hacker's IP Address, E mail ID, Phone number and postal Address are tracked and stored. Finally, these details are sent to original user alternative mail id.

CHAPTER – 11

11.1 REFERENCES:

- [1] D. Mirante and C. Justin, “Understanding password database compromises,” Dept. of Comput. Sci. Eng. Polytechnic Inst. of NYU, New York, NY, USA: Tech. Rep. TR-CSE-2013-02, 2013.
- [2] A. Vance, “If your password is 123456, just make it hackme,” New York Times, Jan. 2010.
- [3] K. Brown, “The dangers of weak hashes,” SANS Institute InfoSec Reading Room, Maryland US, pp. 1–22, Nov. 2013, [Online]. Available: <http://www.sans.org/reading-room/whitepapers/authentication/dangers-weak-hashes-34412>.
- [4] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek, “Password cracking using probabilistic context-free grammars,” in Proc. 30th IEEE Symp. Security Privacy, 2009, pp. 391–405.
- [5] F. Cohen, “The use of deception techniques: Honeypots and decoys,” Handbook Inform. Security, vol. 3, pp. 646–655, 2006.

[6] M. H. Almeshekah, E. H. Spafford, and M. J. Atallah, “Improving security using deception,” Center for Education and Research Information Assurance and Security, Purdue Univ., West Lafayette, IN, USA: Tech. Rep. CERIAS Tech. Rep. 2013-13, 2013.

[7] C. Herley and D. Florencio, “Protecting financial institutions from brute-force attacks,” in Proc. 23rd Int. Inform. Security Conf., 2008, pp. 681–685.

[8] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh, “Kamouflage: Loss-resistant password management,” in Proc. 15th Eur. Conf. Res. Comput. Security, 2010, pp. 286–302

[9] Wikipedia. (2014) Global surveillance disclosures (2013 present). [Online]. Available: [http://en.wikipedia.org/wiki/Global_surveillance_disclosures_\(2013-present\)](http://en.wikipedia.org/wiki/Global_surveillance_disclosures_(2013-present))

[10] (2014) Edward snowden. [Online]. Available: http://en.wikipedia.org/wiki/Edward_Snowden