Enterprise Programming for Distributed Application

INDIVIDUAL WORK

Subject : CT027-3-3-EPDA

Intakes : APD3F2411CS(AI)

Date Assigned : Week 2

Date Due : 14 March 2025

Lecturer : Mr. Kau Guan Kiat

Student Name : LUAI MOHAMMED ABDULLAH AL-HARAZI TP070855

Table of Contents

Table of Figures

Assumptions

The APU Hostel Visitor Verification System assumes three user roles (Managing Staff, Security Staff, and Residents), each with specific permissions. The system relies on a stable MS SQL Server connection via JDBC, with secure SHA-256 hashed passwords for authentication. Each visitor request follows a structured workflow (submission → approval → verification), with a unique verification code generated to prevent duplication. Sessions store user details and expire after inactivity, ensuring security. Profile pictures are uploaded or defaulted, and navigation dynamically updates the user's name and image. SQL queries use prepared statements to prevent SQL injection, and reports track visitor requests and verifications. The Use Case and Class Diagrams accurately reflect system interactions and backend structure. Future enhancements may include email notifications, multi-factor authentication (MFA), and additional roles for improved security and scalability.

1.0 Brief Overview of Distributed Computing and Various Enterprise Application Architectures

1.1 Introduction to Distributed Computing

Definition: What is Distributed Computing?

Distributed computing is a model of computation wherein the computational tasks are distributed in more than one computer node that communicates with each other in a network. Even though the system components are scattered at different locations, they will work as an integrated system for a common objective and enhance efficiency, performance, and scalability (Kinza Yasar & Gillis, 2024).

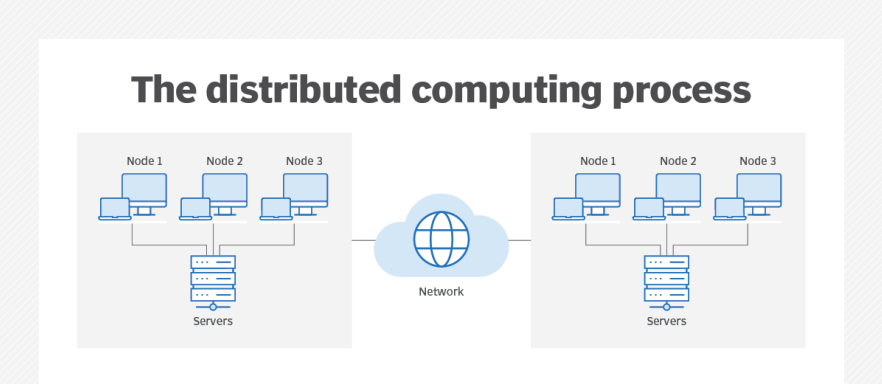


Figure 1: Overview of a Distributed Computing Process (Kinza Yasar & Gillis, 2024)

The essence of distributed computing is to distribute the tasks among nodes, perform parallel execution, and share resources. A central algorithm divides a big task into smaller subtasks and distributes them to different nodes. Further, all these nodes execute their subtasks simultaneously, enhancing the speed of complex computation manifold compared to sequential processing (Kinza Yasar & Gillis, 2024).

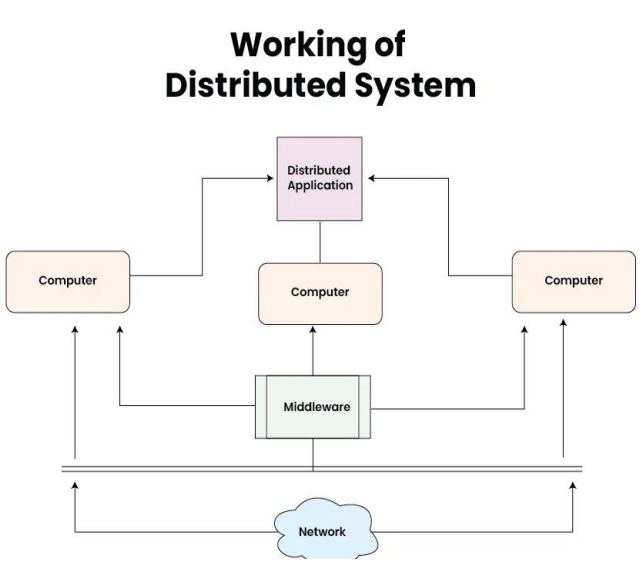


Figure 2: Working of Distributed System (GeeksforGeeks, 2022).

Key Characteristics of Distributed Computing

1. **Fault tolerance**: means that distributed systems are resilient to failures, such as failures at one node or other components; the system should be able to operate through redundancy and replication (GeeksforGeeks, 2022).

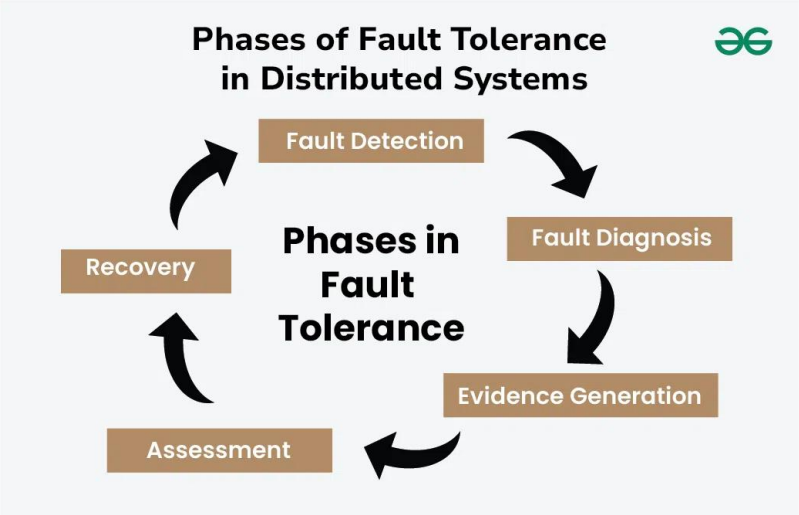


Figure 3:Phases of Fault Tolerance in Distributed Systems (GeeksforGeeks, 2023).

2. **Parallelism**: Many nodes in distributed systems perform in parallel, reducing the overall computation time due to the division of efforts among multiple machines (GeeksforGeeks, 2022).

3. **Scalability:** Since distributed computing can scale horizontally, adding nodes to handle a more significant workload, it is easily adaptable to an ever-growing demand (GeeksforGeeks, 2022).

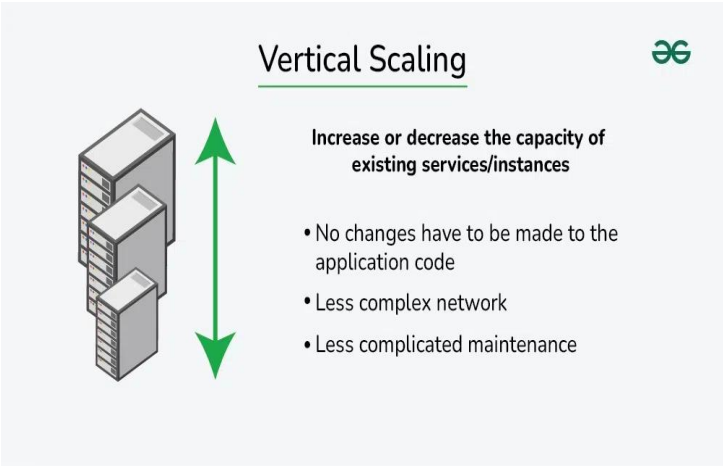


Figure 5:Vertical Scaling in Distributed Systems

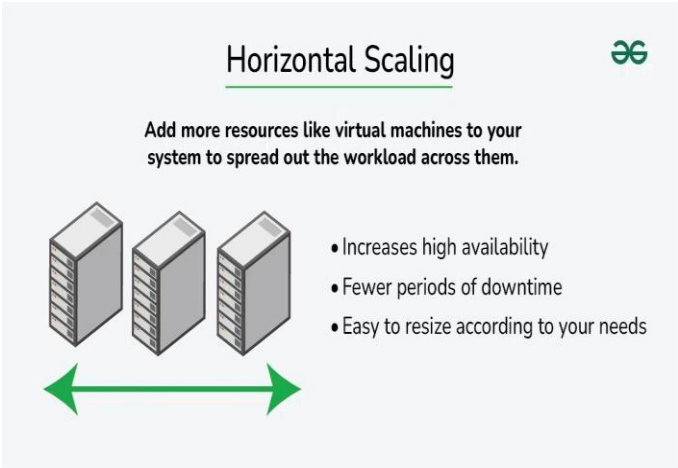


Figure 4:Horizontal Scaling in Distributed Systems

4. **Transparency:** A distributed system hides the structural complexities from the user and applications, allowing them to interact with the system as if all resources were centrally located, whereas they are dispersed over several machines (GeeksforGeeks, 2022).

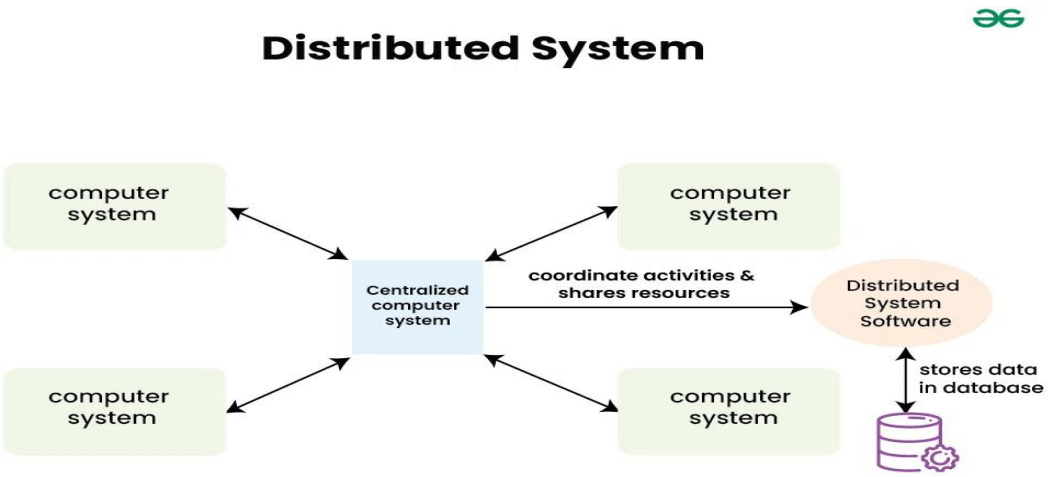


Figure 6: Distributed System(GeeksforGeeks, 2022).

5. **Resource Sharing:** Distributed systems allow the sharing of resources like data, processing, and storage across different machines, enhancing efficiency by reducing operational costs (GeeksforGeeks, 2022).

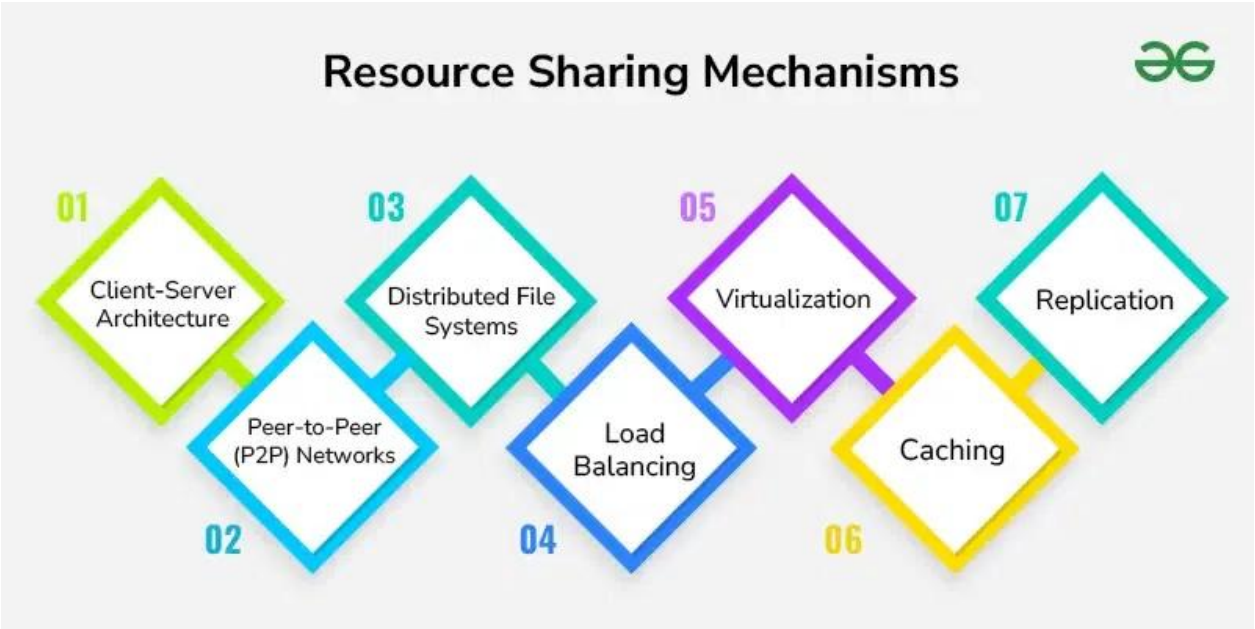


Figure 7: Resource Sharing Mechanisms in Distributed Systems

Benefits of Distributed Computing

Distributed computing has several advantages that make it suitable for the needs of contemporary computing:

1. **Enhanced Performance:** By breaking down tasks into smaller subtasks and distributing them among multiple machines, the overall performance improves due to parallel execution (Kinza Yasar & Gillis, 2024).

2. **Scalability:** Additional resources can always be added whenever a system's demands increase without causing disturbances in operations. Inherent in their design, distributed systems can scale well for large numbers of users and improve data (GeeksforGeeks, 2022).

3. **Resilience and Redundancy:** Many machines perform similar tasks, so they are unaffected if one node crashes down. Their fault tolerance is an added advantage for nonstop processing and avoidance of disturbance in services (GeeksforGeeks, 2022).

4. **Cost-effectiveness:** The hardware used in distributed computing is generally off-the-shelf and thus much cheaper than high-end, expensive servers that centralised systems require (Kinza Yasar & Gillis, 2024).

5. **Accessibility:** Because distributed systems span a large area or different geographical locations, they also ensure that resources are globally accessible. Such systems, therefore, enable organisations to service users around the world, reducing latency (GeeksforGeeks, 2022).

6. **Efficiency:** In this respect, dividing and distributing complex tasks among several systems allows faster processing. This is extremely useful in broad applications, such as scientific research, cloud computing, and enterprise resource management (Kinza Yasar & Gillis, 2024).

1.2 Types of Distributed Computing Models

Client-Server Model

In system design, the client-server architecture is a predominant model where the network has been divided into servers and clients. Clients are generally desktop computers or mobile devices that request resources or services from the server, a powerful machine, or a cluster of machines that provide the same. Due to its capability for data organisation and resource sharing, this model remains fundamental in controlling network-based systems, such as web applications, databases, and email systems. The architecture enables high scalability as more servers could be added while existing ones are upgraded to handle more prominent clients. Thus, security and management are realised with much control and ease since centralised maintenance enhances systems' reliability and performance (GeeksforGeeks, 2024).

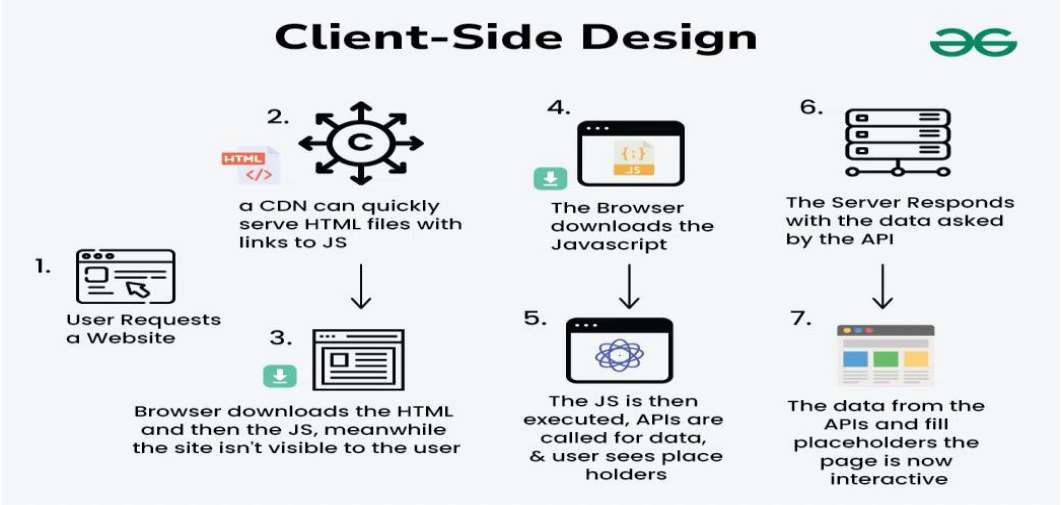


Figure 8: Client-side design illustrating the interaction between a user's browser and the server in a client-server architecture.

Description: This diagram shows the interaction between the client and server in a web application. The user requests a webpage, which, through the CDN, delivers the required HTML and JavaScript files. The browser downloads and executes JavaScript, making API calls to the server and sending the data needed to make the page interactive. This process highlights how efficiently and responsive the client-server model handles web application requests.

Peer-to-Peer (P2P) Model

In the P2P model, every network node is a client and a server. Thus, the P2P model is reasonably fit for implementing applications built on decentralised networks. Unlike nodes working in any traditional model of the client-server paradigm, resources are directly shared between peer nodes without resorting to the presence of any centralised server. This ensures, in return, that load/pressure is not concentrated on a single node to bring down the entire structure quickly and increase resiliency in case of a network failure. P2P architectures are widely utilised in file-sharing systems and blockchain technologies, where data transparency and integrity are essential (GeeksforGeeks, 2019).

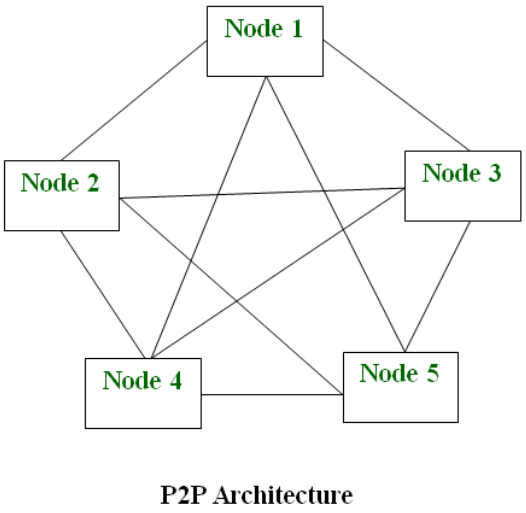


Figure 9: Diagram illustrating the Peer-to-Peer (P2P) architecture, showing a network of five interconnected nodes.

Description: The figure above illustrates a typical topology of a P2P network, where each node communicates with several others. The significant difference compared to the classic client-server architecture is that resources are supplied and consumed by all nodes in the P2P network. Resources could include data, bandwidth, and/or processing cycles. This gives full interconnectivity to distribute shared resources redundantly. Such a connection contributes to the hardening of sharing in ways that strengthen the network's overall resiliency through there being no point of failure. Such architectures find extensive use in file-sharing and blockchain-related applications.

Grid Computing

Grid computing employs computers distributed in various forms over a network, hence aggregating computer resources for a specified purpose. Many jobs require ultimate computational power, and its effectiveness in operations such as the simulation of materials, data analytics, and a significant amount of digital content manipulation is very apparent in this context of grid computing. Accordingly, it acquires and mobilises those vastly underutilised resources of other computers distributed along with every computer, effectively executing compute-intensive jobs that call for scalability and high levels of resource utilisation. This distributed computing is instrumental in research and development projects across different geographical areas by guaranteeing solutions to challenging problems (GeeksforGeeks, 2019).

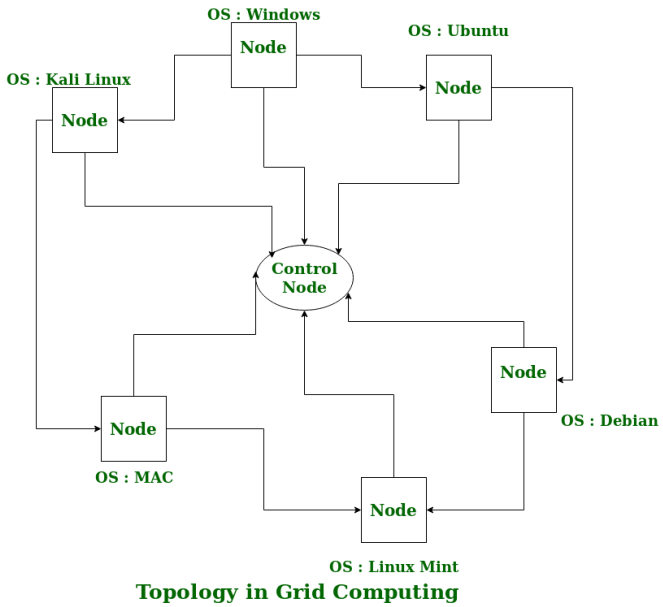


Figure 10: Topology of a Grid Computing Network showcasing the integration of nodes with diverse operating systems coordinated by a central control node

Description: This diagram depicts the typical topology of a grid computing network: different operating system nodes on a central control node. Each node contributes its processing power and memory to one shared resource pool managed by the control node. This way, the network can run applications that demand vast amounts of computation and data storage. It illustrates how grid computing utilises the hitherto unused resources of computers within a widely spread network to enable higher capabilities related to data processing and complex problem-solving in different geographic locations.

Cloud Computing

Cloud computing, the significant evolution of distributed computing, includes delivering servers, storage, and applications over the internet by cloud providers, including AWS, Google Cloud, and Microsoft Azure. This helps businesses run scalable infrastructure without investing money in hardware acquisition. It saves the enterprise from higher investments and expands its freedom in operational areas. Cloud services are characterised by resource availability on demand, which can be scaled up or down depending on immediate needs, business agility, and efficiency. It supports various deployment modes, including public, private, and hybrid clouds, each offering different levels of control, flexibility, and management (GeeksforGeeks, 2017).

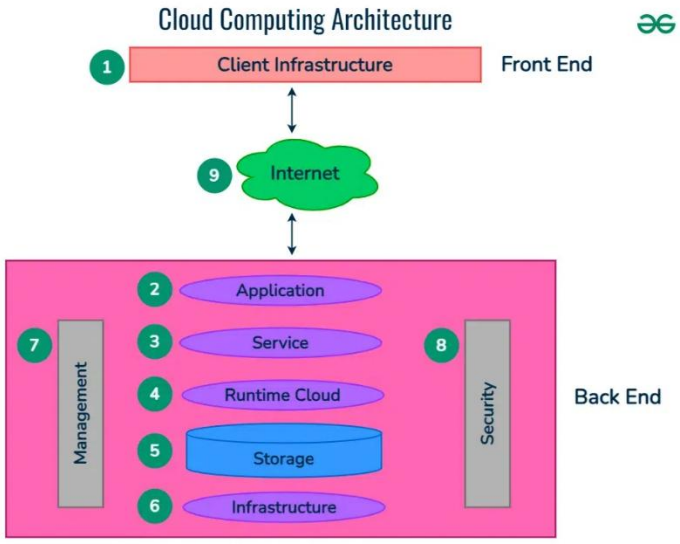


Figure 11: Cloud Computing Architecture

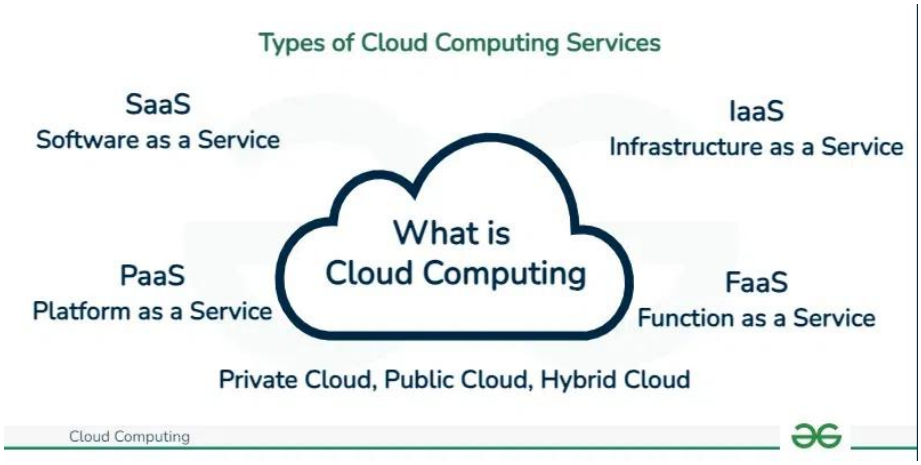


Figure 12: Types of Cloud Computing Services

1.3 Challenges of Distributed Computing

Distributed computing systems are pivotal in handling large-scale data processing across multiple locations. However, these systems encounter several challenges that can impact their efficiency and reliability:

- **Latency:**

  The Latency in distributed systems is the time gap between sending and receiving a response or message. Over the years, it has been among the most dominant influencing factors that affect the performance of a system and users at large. These result from propagation, transmission, processing, and queuing delays. Latency-controlling techniques mainly emanate from the optimisation of the network path, strategies in the way data is being processed, and effective protocols in communication aimed at minimising general time delays that characterise the operation of any system (GeeksforGeeks, 2024).

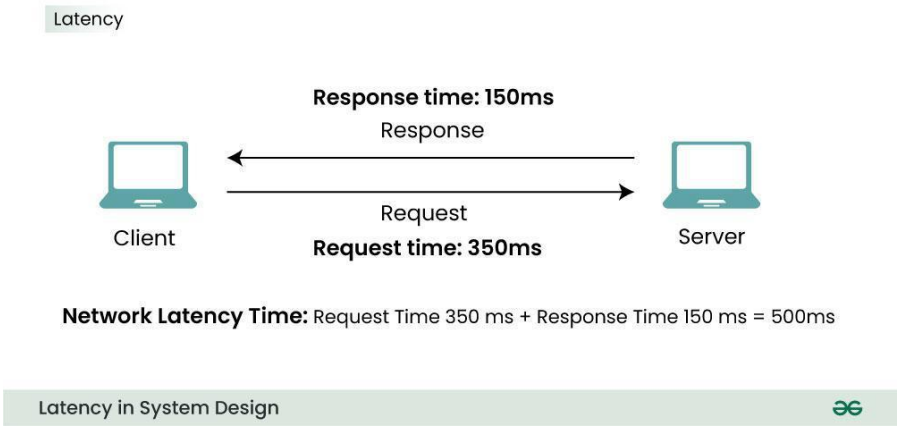
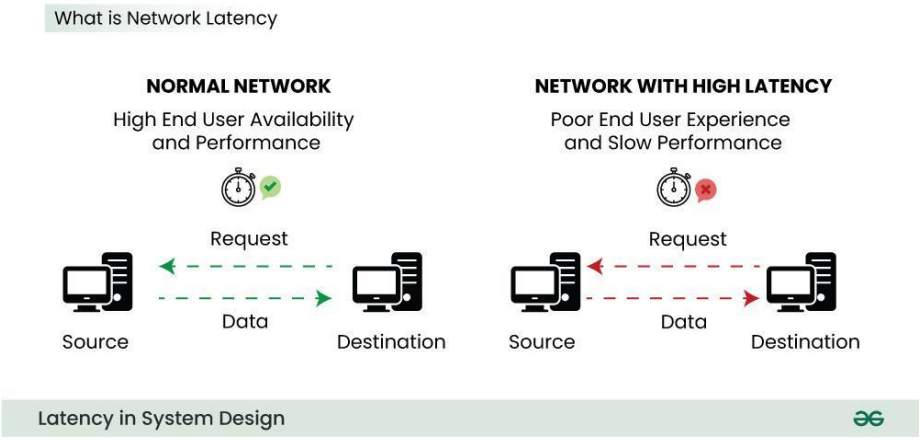


Figure 13: Detailed Breakdown of Network Latency Time

- **Network Issues:**

  Some problems related to distributed systems include network problems such as bandwidth management-issue latency and throughput, congestion in networks, scalability, and bottlenecks within data transfer. Efficient bandwidth management is essential regarding distributed systems' performance, reliability, and scalability issues. Techniques including traffic shaping, bandwidth allocation, and quality of service have been employed to achieve that effect (GeeksforGeeks, 2024).



- **Data Consistency:**

  Data consistency in a distributed system necessitates node synchrony. The occurrence of latency and issues regarding network latencies make it quite challenging to avoid data being out-of-sync. Approaches to ensure consistency involve strong consistency models, robust synchronisation protocols, and distributed databases designed to handle such complexities (Sankar, 2023).

- **Fault Tolerance:**

  Fault tolerance refers to the ability of a distributed system to handle several failures or other types of malfunctions occurring in single or various constituent components using redundancy, error detection, replication, and failover mechanisms. In this respect, the system will retain reliability, availability, and consistency properties if any component fails (GeeksforGeeks, 2023).

1.4. Enterprise Application Architecture Models

Monolithic Architecture

Monolithic Architecture is a traditional approach to building applications in which all UI, business logic, and database functionalities are integrated into a single deliverable unit. Such architecture simplifies development, debugging, testing, deployment, and application evolution since all components are colocated and tightly coupled. This simplicity and low cost initially make monolithic architectures appealing for small-scale applications and rapid prototyping (Oleksii Dushenin, 2021).

Advantages:

- Easy to develop, debug, and test since all components are in one place.
- Low initial costs and easy deployment.

Disadvantages:

- Not scalable and maintainable when the system grows.
- The possible problems brought about by high coupling include slow development speed, painful CI/CD, and scaling services and databases that are hard to manage (Oleksii Dushenin, 2021).

Layered (N-tier) Architecture

In a layered architecture, the concerns get separated through multiple layers, including presentation, business logic, and data access. This enhances maintainability and scalability. Such architecture would be suitable for applications like Google's Gmail, which processes and then presents the content in different available languages by employing different layers for data interaction, business logic, and user interface (*Layered Architecture*, 2024).

Advantages:

- Layer independence allows for easy updating and scaling.
- It enhances customisation without affecting other layers.

Disadvantages:

- Performance overhead due to interactions between layers.

- Increased complexity in maintenance, thus possibly affecting application performance.

Microservices Architecture

Microservices are an architecture comprising small units and independently deployable services confined to specific parts of business matters. They are connected through lightweight protocols to enable communication by different applications to accomplish certain aspects of the requestor. Due to its flexibility and manageability in scaling any distributed system, every microservice stands independently deployable (Vitaly Kuprenko, 2019).

Advantages:

- Services are independently deployable and scalable, which makes the system more flexible.
- Isolation of services increases the system's resilience and enables continuous delivery/deployment.

Disadvantages:

- Complexity in managing multiple services and their interactions.
- Potential for increased resource consumption due to multiple service instances.

Service-oriented architecture (SOA)

SOA breaks down business logic into individual services; each may perform a different function that could be reused and assembled to build complex applications. Services communicate over the network using standard protocols such as SOAP or REST, facilitating integration and reuse (GeeksforGeeks, 2018).

Advantages:

- High reusability of services, thus speeding up application development.
- Platform independence and flexibility in service integration.

Disadvantages:

- Managing service interactions can be awkward, hence leading to overhead.
- High initial investment and possible performance issues because of service validation

Event-driven architecture (EDA)

EDA reacts to events or changes instead of abiding by some procedure. Hence, it will be apt for all those applications that demand real-time processing of data, an IoT environment being a prime example. Events are handled asynchronously, supporting highly scalable and responsive systems (RobBagby, 2024).

Advantages:

- High scalability and real-time processing capability.
- Decouples the producers and consumers of events to make a system more resilient.

Disadvantages:

- Complexity in handling event orders and ensuring delivery.
- It might not be easy to manage distributed transactions.

2.0 Detailed Architecture Review of a CRM (Customer Relationship Management) System

2.1 Introduction to CRM

Definition and Importance of CRM

CRM is the system businesses use to interact with their existing and potential customers. CRM software provides a unified platform for storing customer information, enabling a business to enhance its sales and marketing processes and customer service. This, in turn, brings better customer integration, increases operational efficiency, and ensures business growth (GeeksforGeeks, 2022).

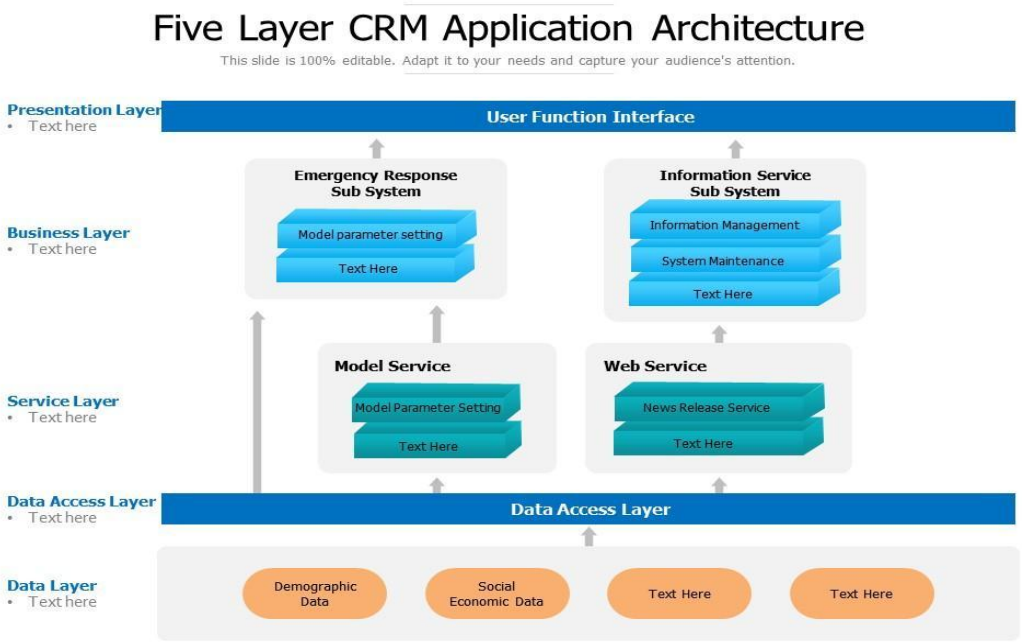


Figure 15: A five-layer CRM architecture that illustrates how different components interact to manage customer data and processes.

Why CRM is Crucial for Businesses

CRM systems are important in businesses as they help a business gain insight into the trends and preferences of the customers. In doing so, it provides businesses with a platform to personalise their communication, enhances service delivery, and offers the possibility of newer opportunities. Ultimately, it allows companies to enrich customer satisfaction and improve retention, thus driving growth in sales (GeeksforGeeks, 2022).

MARKETING	SALES	CUSTOMER SERVICE
• Customer segmentation	• Lead management	• Request management
• Campaign development	• Account management	• Service tracking
• Campaign execution	• Pipeline management	• Escalation/ prioritization
• Project/ event management	• Cross-selling and up-selling	• SLA agreements
	• Activity management	• Account inquiries

Figure 16:Key CRM functions across Marketing, Sales, and Customer Service (Five Layer Crm Application Architecture, 2020).

Key Functions of CRM

1- Sales Automation: Automates important sales functions that include lead management, forecasting of sales, and pipeline tracking, which frees the forces to work on the deal closure (GeeksforGeeks,2022).

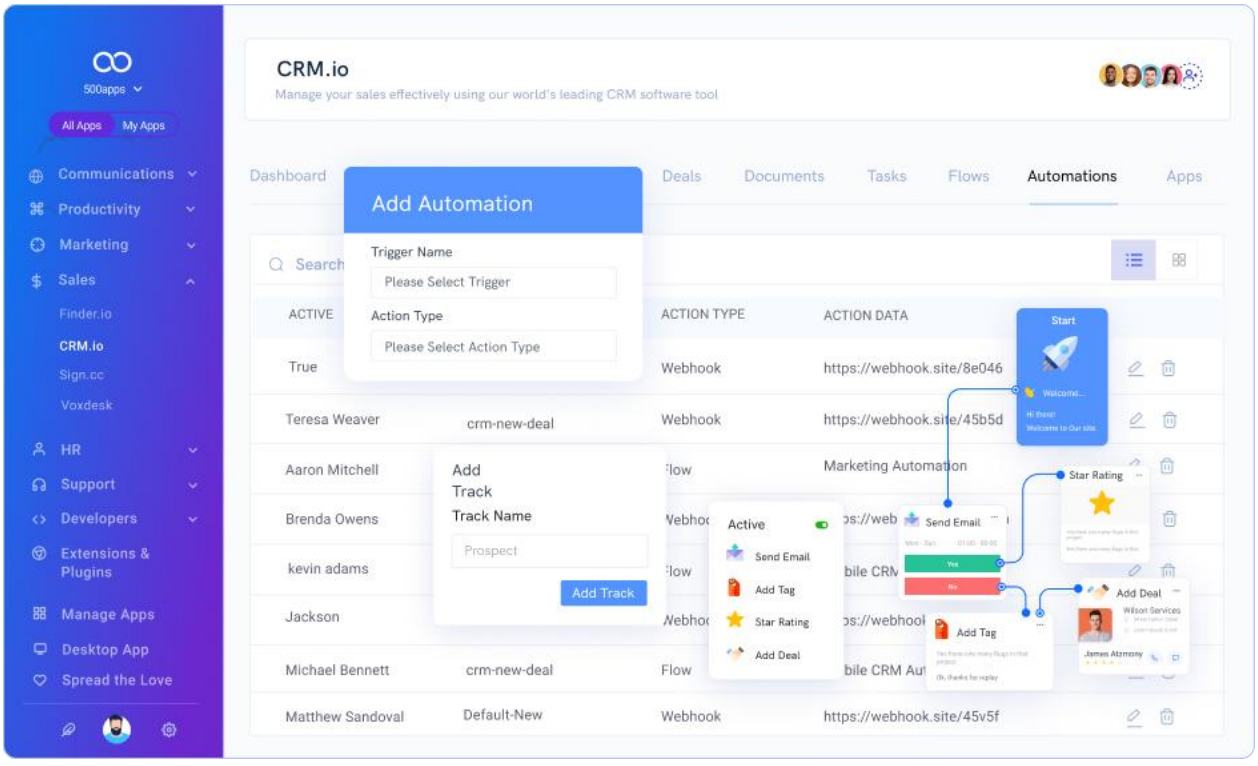


Figure 17: Example of CRM sales automation dashboard.

2-Marketing Automation: Automates marketing processes such as email campaigns, customer segmentation, and lead nurturing for greater efficiency and better targeting (GeeksforGeeks, 2022).

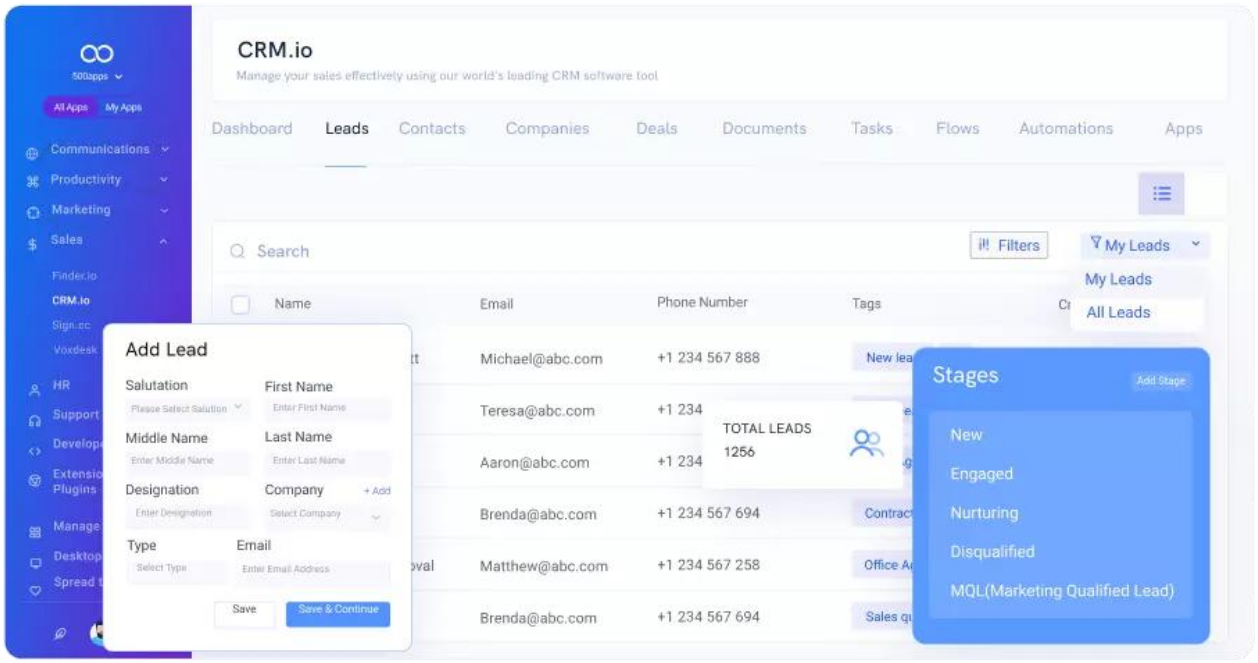


Figure 18: Example of lead scoring and segmentation in CRM systems.

3-Customer Support: Improves customer service by providing tools to track and resolve customer inquiries; this often includes ticketing systems and knowledge bases.

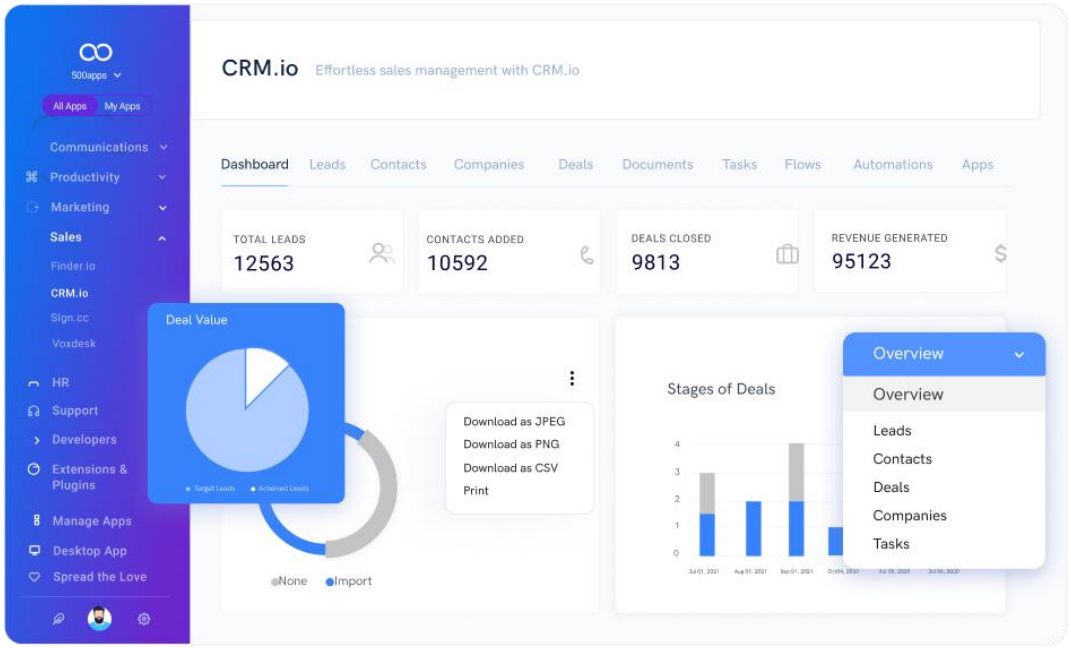


Figure 19: CRM analytics dashboard showing customer service performance insights.

4-Analytics and Reporting: Offers in-depth reporting and visualisation that enables insight into business performance, customer trends, and general effectiveness in using CRM.

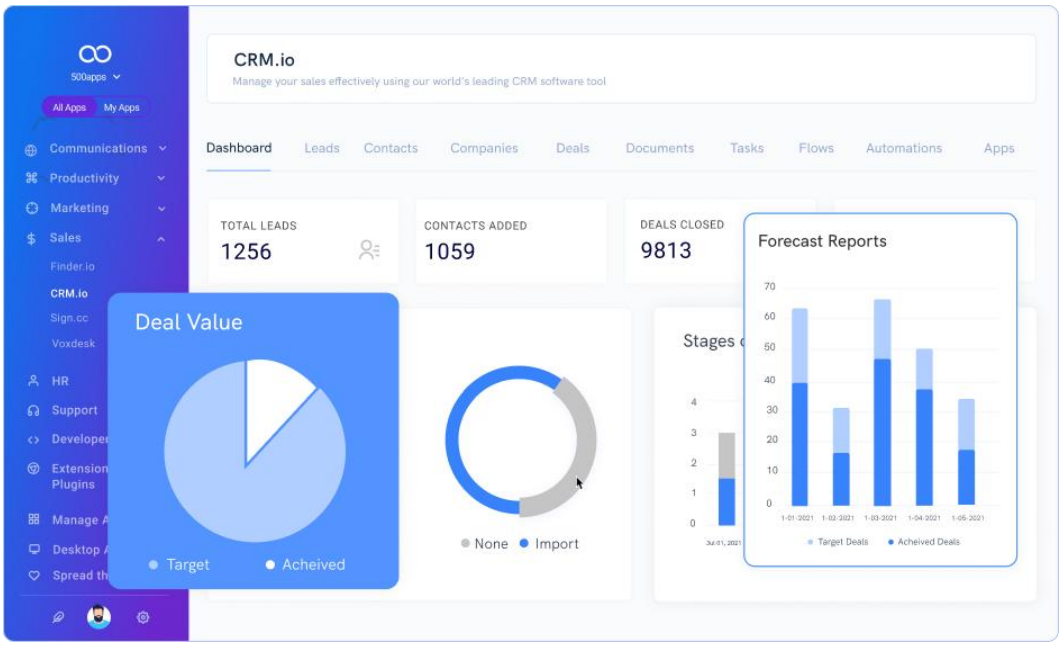


Figure 20: CRM forecasting tool showing sales and performance projections.

2.2. Common CRM Architectures

CRM systems are implemented based on different architectures, depending on the needs, resources, and specific business requirements. Three major CRM architectures are **Cloud-Based, On-Premises, and Hybrid**. Each model has advantages and disadvantages; therefore, every business must choose an appropriate solution.

➢ Cloud-Based CRM

The Cloud-Based CRM architecture is hosted on remote servers, which one can access via the Internet. Such an architecture allows for managing customers' data and interactions through cloud software solutions like Salesforce or HubSpot. A Cloud-Based CRM scales with ease, and provided one has a computer and an access point to the internet, this kind of solution can be accessible from just about anywhere. Also, Cloud-based CRMs eliminate or reduce the need for businesses to invest in physical hardware or IT infrastructure (Zur, 2024).

Pros:

1- Accessibility: Cloud CRMs can be accessed anywhere in the world with an internet connection, allowing employees to work from any location and collaborate (Zur, 2024).

2-Scalability: In such systems, scaling up can be quickly done in growth, and the business incurs the cost for only the utilised resources.

3-Minimum infrastructure costs: The company will not have to spend on hardware since servers and maintenance are a headache for the cloud provider.

Cons:

1- **Dependence on Internet Connectivity:** Businesses cannot access the CRM system without a stable Internet connection.

2- Data Privacy Concerns: Storing sensitive customer data in the cloud raises concerns about security and compliance with data protection regulations.

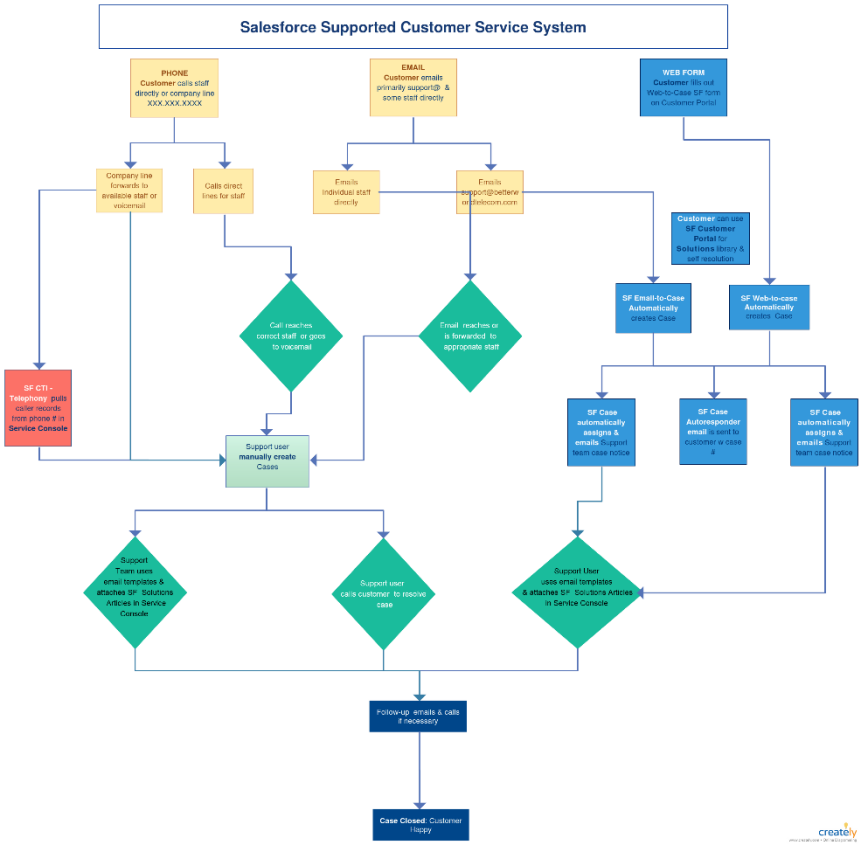


Figure 21:Salesforce Supported Customer Service System Flow (Roob, 2024).

This diagram illustrates how customer service cases are created and processed on **Salesforce cloud-based CRM** that supports multi-channel communication with automated case management.

Code Example:

```
import java.io.*;
import java.util.*;

import com.sforce.async.*;
import com.sforce.soap.partner.PartnerConnection;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;

public static void main(String[] args)
      throws AsyncApiException, ConnectionException, IOException {
        BulkExample example = new BulkExample();
        example.runSample("Account", "myUser@myOrg.com", "myPassword",
"mySampleData.csv");
    }

public void runSample(String sobjectType, String userName, String password,
String sampleFileName)
            throws AsyncApiException, ConnectionException, IOException {
        BulkConnection connection = getBulkConnection(userName, password);
        JobInfo job = createJob(sobjectType, connection);
        List<BatchInfo> batchInfoList = createBatchesFromCSVFile(connection
, job, sampleFileName);
        closeJob(connection, job.getId());
        awaitCompletion(connection, job, batchInfoList);
        checkResults(connection, job, batchInfoList);
    }

private BulkConnection getBulkConnection(String userName, String password)
          throws ConnectionException, AsyncApiException {
        ConnectorConfig partnerConfig = new ConnectorConfig();
        partnerConfig.setUsername(userName);
        partnerConfig.setPassword(password);
        partnerConfig.setAuthEndpoint("https://login.salesforce.com
/services/Soap/u/63.0");
        new PartnerConnection(partnerConfig);
        ConnectorConfig config = new ConnectorConfig();
        config.setSessionId(partnerConfig.getSessionId());
        String soapEndpoint = partnerConfig.getServiceEndpoint();
        String apiVersion = "63.0";
        String restEndpoint = soapEndpoint.substring(0, soapEndpoint
.indexOf("Soap/"))
            + "async/" + apiVersion;
        config.setRestEndpoint(restEndpoint);
        config.setCompression(true);
        config.setTraceMessage(false);
        BulkConnection connection = new BulkConnection(config);
        return connection;
    }
```

Figure 22: Salesforce Bulk API for Data Loading

The above example shows the use of the Bulk API in automating the upload and processing of large volumes of data within Salesforce. Scalability and resource management, powered by cloud CRM solutions like Salesforce, enable companies to grow without fear regarding infrastructure. This code shows how, without much effort, API can easily automate the given data tasks and extend cloud CRM management to massive datasets.

➢ On-Premises CRM

The On-premises system is where the CRM system sits on the company's private servers, allowing the company to have complete sovereignty over the information, system modifications, and system security. An example of an on-prem CRM system is **Microsoft Dynamics**, in which the firm has internal control of both software and hardware.

Pros:

1- Total Control over Data and Security: Total control of security procedures and sensitive information is at the enterprise's disposal with greater ease and efficiency.

2-System Modifications: The system is customisable for specific business needs without resorting to vendors.

Cons:

3- Substantial Infrastructure Expenses: Significant amounts are spent upfront on hardware, software, and IT services

4- Maintenance Costs: The ever-growing maintenance and repair work and system upgrades need personnel and resources from an internal IT department.

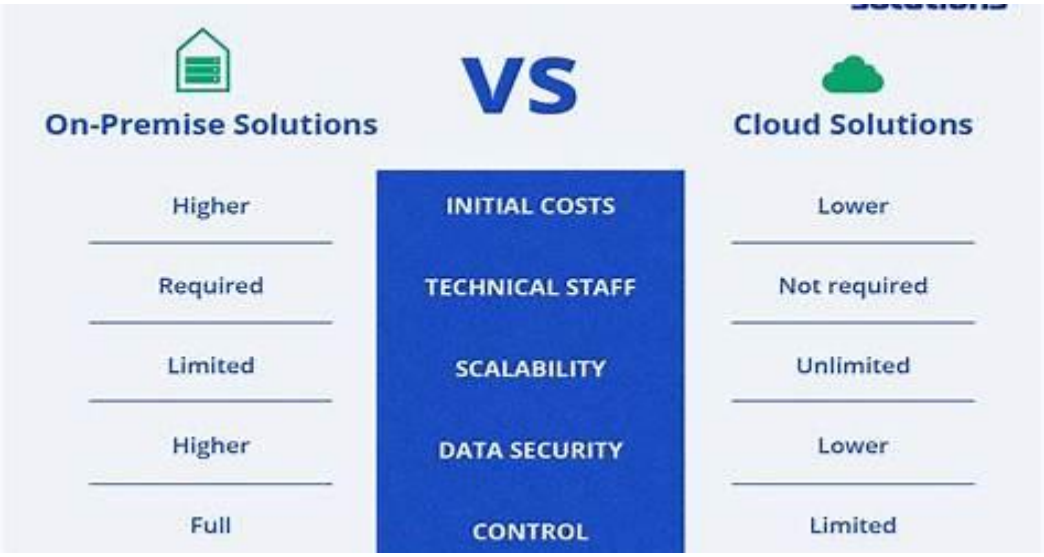


Figure 23: On-Premises Solutions vs. Cloud Solutions

Code Example:

```java
import com.microsoft.aad.adal4j.AuthenticationContext;
import com.microsoft.aad.adal4j.AuthenticationResult;
import com.microsoft.aad.adal4j.ClientCredential;
import okhttp3.*;

import java.io.IOException;
import java.net.MalformedURLException;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Main {
    public static void main(String[] args) {
        String authority = "https://login.microsoftonline.com/";
        String resource = "https://msott.crm.dynamics.com";
        String clientId = "your-client-id";
        String clientSecret = "your-client-secret";
        String tenantID = "your-tenant-id";
        ExecutorService service = Executors.newFixedThreadPool(1);
        AuthenticationResult result;

        try {
            AuthenticationContext context = new AuthenticationContext(authority + tenantID, true, service);
            Future<AuthenticationResult> future = context.acquireToken(resource, new ClientCredential(clientId,
clientSecret), null);
            result = future.get();
            String accessToken = result.getAccessToken();

            createWithDataReturned(accessToken);
        } catch (MalformedURLException | InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }
    }

    // Retrieving customized responses on POST method
    public static void createWithDataReturned(String accessToken) {
        try {
            OkHttpClient client = new OkHttpClient();

            MediaType mediaType = MediaType.parse("application/json; charset=utf-8");
            RequestBody body = RequestBody.create(mediaType, "{" +
                    "\"name\": \"Sample Postman Account\"," +
                    "\"creditonhold\": false," +
                    "\"address1_latitude\": 47.639583," +
                    "\"description\": \"This is the description of the sample account\"," +
                    "\"revenue\": 5000000," +
                    "\"accountcategorycode\": 1" +
                    "}");
            Request request = new Request.Builder()
                    .url("https://msott.api.crm.dynamics.com/api/data/v9.0/accounts")
                    .post(body)
                    .addHeader("OData-MaxVersion", "4.0")
                    .addHeader("OData-Version", "4.0")
                    .addHeader("Accept", "application/json")
                    .addHeader("Content-Type", "application/json; charset=utf-8")
                    .addHeader("Prefer", "return=representation")
                    .addHeader("Authorization", "Bearer " + accessToken)
                    .build();

            Response response = client.newCall(request).execute();
            String dataReturnedFromCreate = response.body().string();
            System.out.println("Response: " + dataReturnedFromCreate);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Figure 24: Java Code Example for Integrating with Microsoft Dynamics CRM (On-Premises) Using OAuth and REST API

➢ Hybrid CRM

The hybrid version combines cloud-based and on-premise CRM models to provide options for businesses regarding the location of sensitive data storage while reaping scalability and accessibility benefits from the cloud. This architecture fits well for organisations that require high security and flexibility in customer data management.

Pros:

1- Flexibility: Sensitive information can be kept on-premises while utilising cloud services for scaling.

2. Cost-Efficiency: Hybrid models balance on-premises infrastructure costs and scalability through cloud services. Scalability: Hybrid CRMs can scale up and grow with businesses since cloud solutions cover variable needs.

Cons:

3- Complexity: This could be a more complex integration of the on-prem and cloud, where systems would require deeper integration, thus challenging an IT team.

4-Security Risks: Hybrid systems have to be integrated with much care in order to maintain the security of data across platforms.

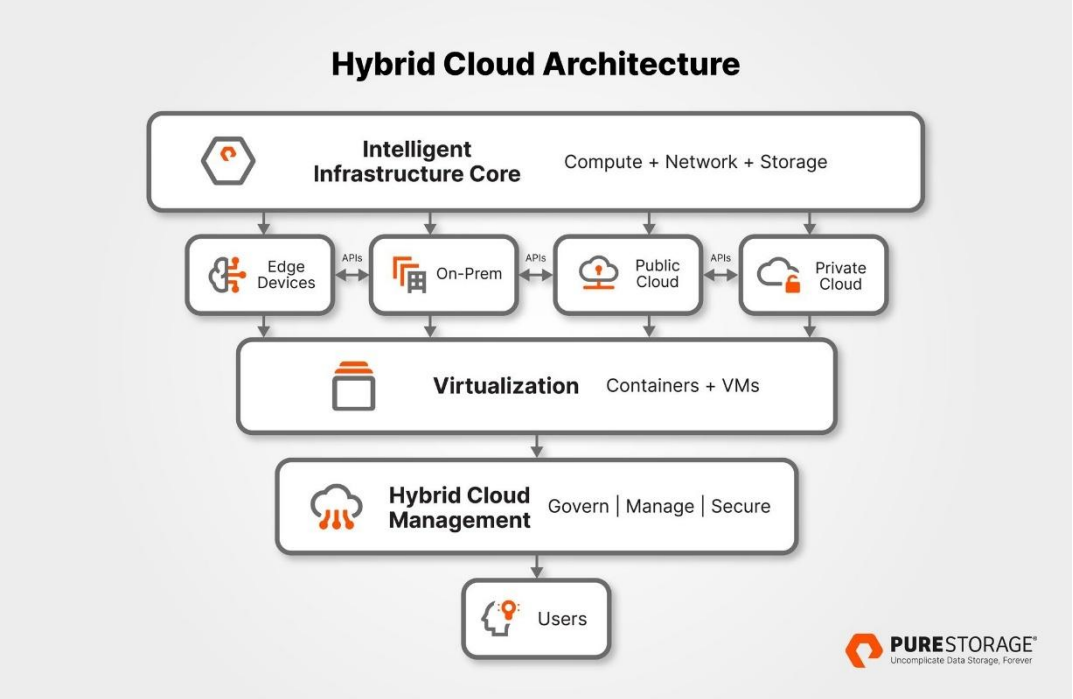


Figure 25: Hybrid Cloud Architecture

Code Example:

```java
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import java.io.IOException;

public class HybridCRMIntegration {

    private static final String ON_PREM_URL = "https://onprem.example.com/api/customers";
    private static final String CLOUD_URL = "https://cloudcrm.example.com/api/customers";
    private static final String ON_PREM_USER = "onprem_user";
    private static final String ON_PREM_PASSWORD = "onprem_password";
    private static final String CLOUD_USER = "cloud_user";
    private static final String CLOUD_PASSWORD = "cloud_password";

    public static void main(String[] args) throws IOException {
        String customersData = fetchOnPremCustomers();
        pushToCloud(customersData);
    }

    private static String fetchOnPremCustomers() throws IOException {
        try (CloseableHttpClient client = HttpClients.createDefault()) {
            HttpGet request = new HttpGet(ON_PREM_URL);
            request.setHeader("Authorization", "Basic " + encodeCredentials(ON_PREM_USER, ON_PREM_PASSWORD));

            HttpResponse response = client.execute(request);
            return EntityUtils.toString(response.getEntity());
        }
    }

    private static void pushToCloud(String customersData) throws IOException {
        try (CloseableHttpClient client = HttpClients.createDefault()) {
            HttpPost request = new HttpPost(CLOUD_URL);
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Authorization", "Basic " + encodeCredentials(CLOUD_USER, CLOUD_PASSWORD));

            StringEntity entity = new StringEntity(customersData);
            request.setEntity(entity);

            HttpResponse response = client.execute(request);
            System.out.println("Response Status: " + response.getStatusLine().getStatusCode());
        }
    }

    private static String encodeCredentials(String user, String password) {
        return java.util.Base64.getEncoder().encodeToString((user + ":" + password).getBytes());
    }
}
```

Figure 26: Java Code Example for Hybrid CRM Integration (On-Prem to Cloud)

2.3. Detailed Architecture of a CRM System

Components of a CRM:

1. **User Interface (UI):** The user interface, web application, or mobile application is an access platform for and use of the CRM system through which users access and utilize it. The user interface is an essential feature in improving user activity and usability.

2. **Business Logic Layer:** It processes information and executes business rules, controlling workflows such as lead scoring or segmentation of a customer base (Business Rules Engine, 2024).

3. **Database Layer:** Stores information about contacts, sales, and communications with a customer and forms the basis for storing and retrieval in CRM software (Rajpal, 2024).

4. **API Layer:** The 4th layer, namely, the API layer, helps integrate with external systems (e.g., payment gateways and emailing platforms), allowing CRM information to be shared with several tools in a business (Salesforce, 2020).

5. **Reporting/Analytics:** It realizes customer behavior and business performance through reporting and visualization tools and aids decision-making (Rajpal, 2024).

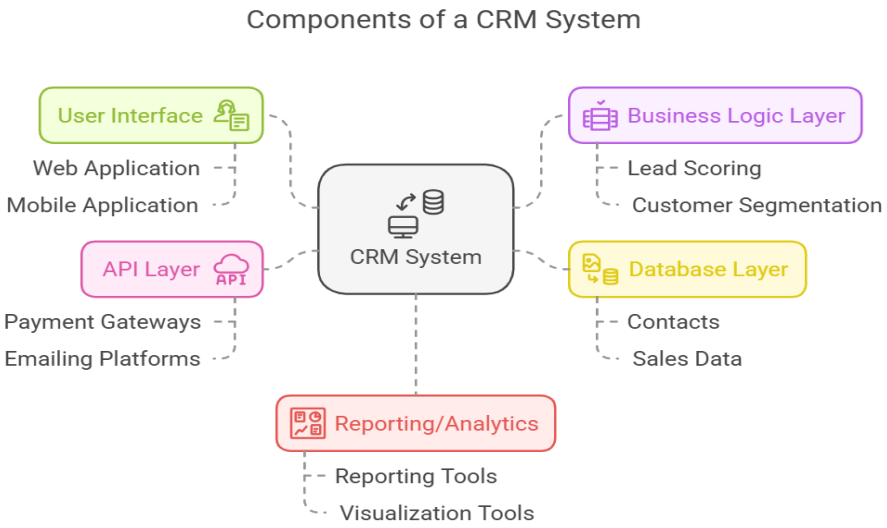


Figure 27: Diagram illustrating the key components of a CRM system, highlighting the main layers and their connections

CRM Architecture Design Example:

Salesforce Architecture: Salesforce utilizes a cloud, multi-tenant infrastructure to offer scalability, efficient information management, and transparent integration via APIs. It utilizes a metadata model with custom settings for individual organizations about their specific requirements (Salesforce Architects, 2022).

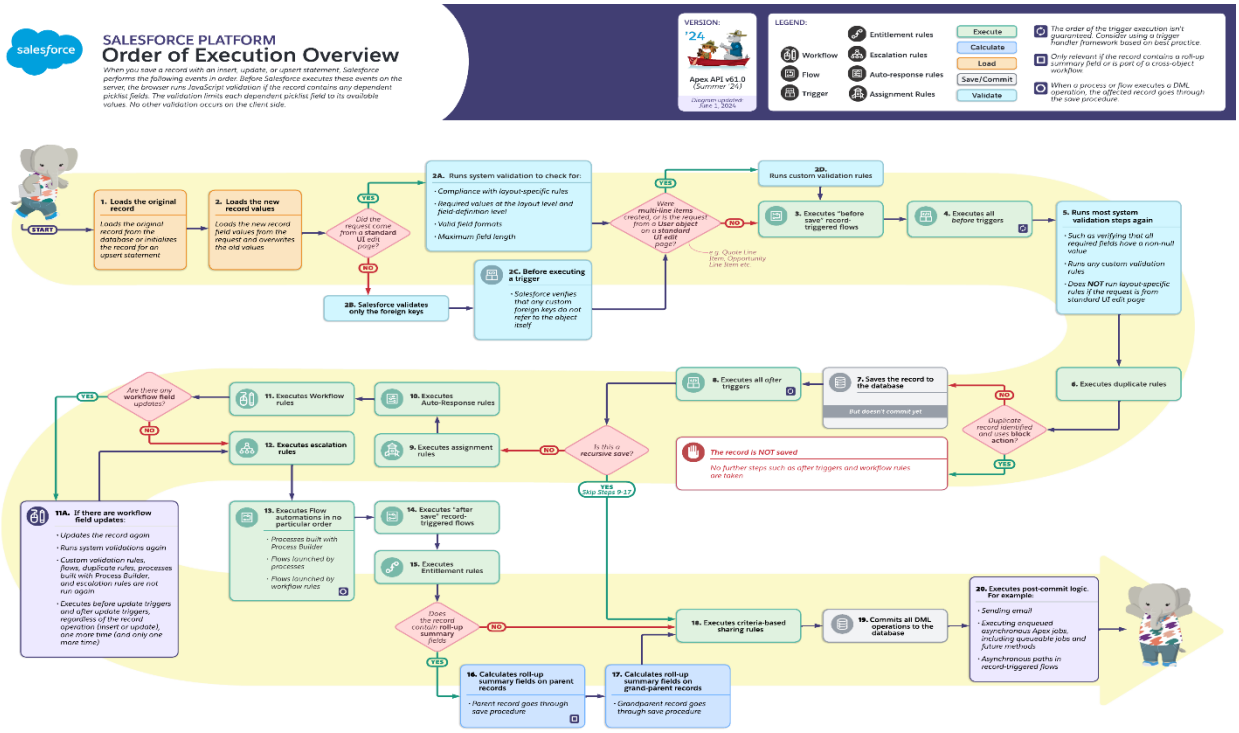


Figure 28: Salesforce's Order of Execution Diagram, showing the sequence of operations performed when records are saved, highlighting validation, workflow, trigger execution, and data handling (Salesforce, 2020).

Scalability and Performance:

CRM software is designed to scale well with growing data, users, and capabilities. Techniques such as load balancing allow for the distribution of loads between users, and caching operations store infrequently updated information in RAM for rapid access and performance improvement (Salesforce, 2020)

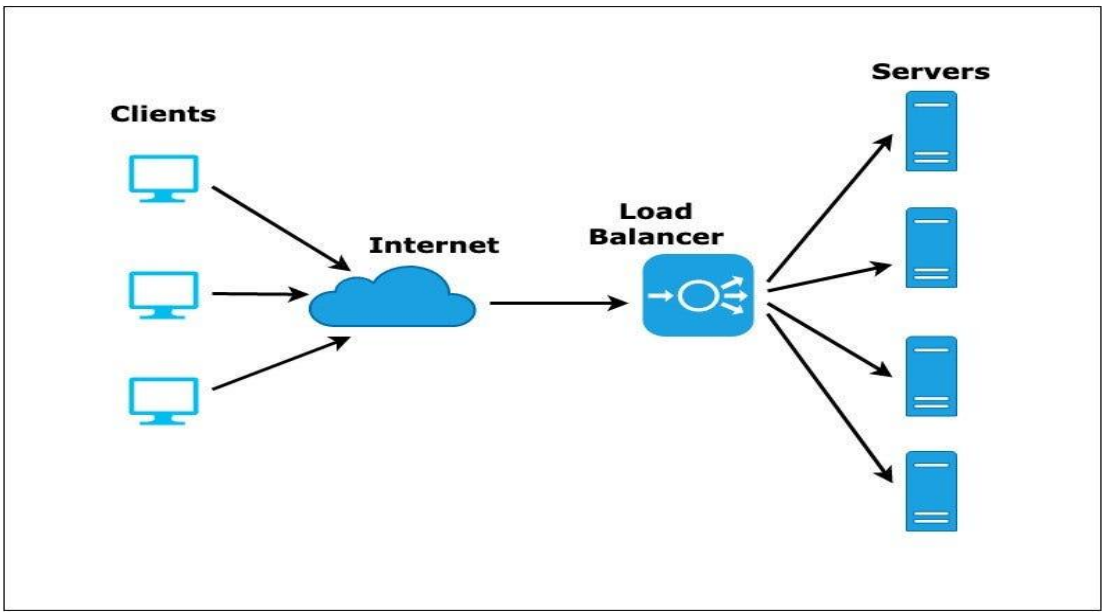


Figure 29: Load balancing diagram illustrating how client requests are distributed across multiple servers, ensuring efficient handling of increased traffic and improving CRM system performance (Avinashsoni, 2022).

Code Example:

```java
public class LoadBalancer {
    private static final String[] servers = {"Server1", "Server2", "Server3", "Server4"};
    private static int currentServerIndex = 0;

    // Get the next server in round-robin fashion
    public static String getNextServer() {
        String server = servers[currentServerIndex];
        currentServerIndex = (currentServerIndex + 1) % servers.length; // Round-robin logic
        return server;
    }

    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println("Client request handled by: " + getNextServer());
        }
    }
}
```

Figure 30: This code shows how a basic load balancing strategy works by routing client requests to different servers in a round-robin manner

```java
import java.util.HashMap;

public class CacheExample {
    private static final HashMap<String, String> cache = new HashMap<>();

    // Simulate fetching data from the database
    public static String fetchDataFromDatabase(String key) {
        // In a real-world scenario, this would be a database call.
        return "Data for " + key;
    }

    // Simulate fetching data with caching mechanism
    public static String getData(String key) {
        // Check if data is in cache
        if (cache.containsKey(key)) {
            System.out.println("Fetching from cache: " + key);
            return cache.get(key);
        } else {
            System.out.println("Fetching from database: " + key);
            String data = fetchDataFromDatabase(key);
            cache.put(key, data); // Cache the data
            return data;
        }
    }

    public static void main(String[] args) {
        System.out.println(getData("user1"));
        System.out.println(getData("user2"));
        System.out.println(getData("user1")); // This should be fetched from cache
    }
}
```

Figure 31: this code simulates a caching mechanism by storing frequently accessed data in a HashMap.

Reliability:

High availability and disaster recovery are assured using fault-tolerant parts and data replication. All these processes make it such that CRM system operations will not stop, even in case of system and network failure (Salesforce, 2020).

Security:

CRM software utilizes strong in-transit and at-rest security techniques to protect sensitive customer information. Role-based access controls (RBAC) protect access to information for a specific individual, and secure APIs protect information shared with a third-party system (Salesforce Architects, 2022).

```java
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class EncryptionExample {

    // Method to encrypt data using AES
    public static String encrypt(String data, String key) throws Exception {
        SecretKeySpec secretKey = new SecretKeySpec(key.getBytes(), "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedData = cipher.doFinal(data.getBytes());
        return Base64.getEncoder().encodeToString(encryptedData);
    }

    // Method to decrypt data using AES
    public static String decrypt(String encryptedData, String key) throws Exception
{
        SecretKeySpec secretKey = new SecretKeySpec(key.getBytes(), "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decodedData = Base64.getDecoder().decode(encryptedData);
        byte[] decryptedData = cipher.doFinal(decodedData);
        return new String(decryptedData);
    }

    public static void main(String[] args) throws Exception {
        String key = "1234567890123456"; // 16-byte key for AES-128
        String data = "Sensitive Customer Information";

        // Encrypt the data
        String encryptedData = encrypt(data, key);
        System.out.println("Encrypted Data: " + encryptedData);

        // Decrypt the data
        String decryptedData = decrypt(encryptedData, key);
        System.out.println("Decrypted Data: " + decryptedData);
    }
}
```

Figure 32: This code shows how to encrypt and decrypt sensitive data using the AES encryption algorithm, which is commonly used to protect data in CRM systems.

```java
import java.util.HashMap;

public class RBACExample {
    // Enum for defining roles
    public enum Role {
        ADMIN,
        SALES,
        CUSTOMER_SUPPORT
    }

    // Class to define a user with a role
    static class User {
        String username;
        Role role;

        User(String username, Role role) {
            this.username = username;
            this.role = role;
        }
    }

    // Method to check if a user has access to a particular resource
    public static boolean checkAccess(User user, String resource) {
        switch (user.role) {
            case ADMIN:
                return true; // Admin has access to all resources
            case SALES:
                return resource.equals("Sales Data"); // Sales role has access to
Sales Data
            case CUSTOMER_SUPPORT:
                return resource.equals("Support Requests"); // Customer Support
role has access to Support Requests
            default:
                return false;
        }
    }

    public static void main(String[] args) {
        User admin = new User("adminUser", Role.ADMIN);
        User salesUser = new User("salesUser", Role.SALES);

        System.out.println("Admin has access to Sales Data: " + checkAccess(admin,
"Sales Data"));
        System.out.println("Sales User has access to Sales Data: " +
checkAccess(salesUser, "Sales Data"));
        System.out.println("Sales User has access to Support Requests: " +
checkAccess(salesUser, "Support Requests"));
    }
}
```

Figure 33: This code checks access based on the role of the user. It's useful for showing how CRM systems restrict access to certain data or functionality based on user roles.

3.0 Discussion on the Architecture Implementation of a Selected APU System

3.1 Overview of APU's IT Infrastructure

Asia Pacific University has designed a sophisticated IT infrastructure to support its academic objectives and ordinary administration operations. It is a mix of many interrelated systems, each designed to support a specific function in university administration, including student information systems and academic management platforms. These work together to form a harmonized environment for enhancing students' and workers' experiences and supporting fact-based decision-making.

Key Components of APU's IT Ecosystem include:

1- Student Information System (SIS): Keeps student information from admissions through graduation, including personal information, grades, and course enrollments.

2- Academic Management Platforms: Programs like APSpace, which houses course content tools (Moodle), class schedules, attendance, and analysis of academic performance. This is the core platform for both face-to-face and distance learning environments.

3- Financial Management Systems: About payment of fees, grants, and financial transactions, such as in APSpace's transaction history feature.

4-Campus Management Solutions: Used in room bookings, events scheduling, and maintenance orders to efficiently use campus assets.

They are built over a platform of robust network infrastructure supporting high-speed connectivity and secure data warehousing, both of which are critical in maintaining the integrity and availability of important academic and individual information.

3.2 Selection of a CRM System for APU

Needs Assessment:

APU needs a CRM system that deepens student engagement through enhanced communications, automates routine administration, and maintains strong alumni ties. The CRM must have in-depth tracking of student contacts from admission to alum, have marketing automation for one-to-one communications, and integrate seamlessly with existing academic and financial systems.

System Selection:

Considering APU's needs and requirements for a strong, scalable, cost-effective solution, a fitting choice is Salesforce Education Cloud. Specifically designed for educational institutions, with recruitment, student success, and alumni relations capabilities, some of its most important advantages **include:**

1- Scalability: Managing changing loads is important for fluctuations in academic calendars.

2- Integration: Provides strong API capabilities, allowing integration with current systems such as APSpace and SIS.

3- Cost-effectiveness: Cloud-based offerings reduce On-premise infrastructure requirements, minimizing IT expenses.

4-Customization: Highly flexible and adaptable according to individual institution processes and requirements.

3.3 How CRM Architecture Can Improve APU's System

Performance:

Salesforce Education Cloud infrastructure can scale dynamically, which is essential during periods of high usage, like enrollments and exams. By being able to sustain performance despite the weight of APU's large student base,

Scalability:

The microservices model in Salesforce allows each service to scale individually. That is when demand is high; for example, during admissions season, high demand can be handled in the CRM without impacting other capabilities in the system.

Reliability:

Salesforce's platform is engineered with high availability via redundancy and disaster recovery capabilities. With its fault-tolerant configuration, even when part of the system fails, CRM will work critically during times of registration and critical grading timelines.

3.4 Architectural Modifications for APU

Data Handling Adjustments

To accommodate specific information for APU, CRM in Salesforce will be developed to manage specific groups of information, such as student academic information, attendance, and faculty information, in compliance with Malaysia's laws regarding PDPA.

Real-time Data Sync:

A real-time mechanism for data synchronization will ensure that any change in CRM is immediately synced with APU's portals, providing real-time information through faculty and student interfaces.

3.5 Potential Improvements

Better Customer Insights

The CRM will use data analysis to better understand student behavior and preference, allowing for sounder decision-making at both academic and administration levels.

User Experience Enhancement

By integrating CRM with accessible interfaces, students and faculty will enjoy a more responsive system with less cumbersome scheduling, communications, and enrollment processes.

Optimized Data Processing:

Using load balancing and data replication features, the CRM can maximize processing efficiency and speed, delivering rapid and accurate student information processing.

Part 2: Application

1.0 Introduction

APU Hostel Visitor Verification System is an enterprise-level web-based solution for automating visitor management in hostel environments. It enables safe and effective verification of visitors by providing role-based rights to the management personnel, security officers, and hostel residents. It streamlines visitor requests, approvals, and verifications in such a manner that it inhibits unwanted entry and enhances security.

The architecture is tiered with Java EE technologies used with JSP, Servlets, JDBC, and MS SQL Server for the backend database. GlassFish 7.0.9 is employed for deployment, and NetBeans 24 with JDK 23 for development. JDBC is employed for database interaction, and security is achieved by password hashing and role-based permission.

1.1 Overview of the Application

The APU Hostel Visitor Verification System consists of three main user roles:

- **Managing Staff**

- Oversees user management (residents, security staff).
- Approves or rejects visitor requests.
- Generates reports for visitor logs and activity tracking.

- **Residents**

- Submit visitor requests by providing visitor details.
- View the status of their visitor requests.
- Edit their profile details.

- **Security Staff**

- Verify visitors upon arrival using a unique verification code.
- Manage activity logs of verified visitors.
- Maintain records of visitor verifications.

Key Features of the System:

- **User Authentication & Role-Based Access** – Secure login system with different functionalities for managing staff, residents, and security staff. Moreover, if the user enters the email address and password wrong 3 times, it will have a time count of 1 minute, and if the user tries it again wrong, the page will be blocked for security purposes.
- **Visitor Request Management** – Residents can submit visitor requests, which must be approved by managing staff.
- **Approval & Rejection System** – Managing staff can approve or reject visitor requests.
- **Visitor Verification** – Security staff verifies visitors using their provided verification code.
- **Activity Logging** – Security personnel maintain verified visitor logs for future reference.
- **Report Generation** – Managing staff can generate reports based on visitor logs and security verification activities.
- **Profile Management** – Users can update their details, including profile pictures and contact information.

This system ensures security, efficiency, and accuracy in managing hostel visitor records, making it an essential tool for hostel administration.

2.0 System Architecture

The APU Hostel Visitor Verification System is developed using a multi-tier architecture, ensuring modularity, scalability, and security. The system follows a three-tier architecture consisting of the Presentation Tier (Frontend), Business Tier (Application Logic), and Database Tier (Backend). Each tier is responsible for a specific function, clearly separating concerns and improving maintainability.

2.1 Description of the Multi-tier Architecture

1. Presentation Tier (Frontend) – User Interface Layer

This tier comprises **JSP (JavaServer Pages), HTML, CSS, and JavaScript**, forming the **frontend interface** for different user roles. It handles user interactions, displays data dynamically, and forwards user inputs to the business tier.

Key Components in the Presentation Tier:

- **JSP Pages** – Dynamic web pages handle different functionalities for residents, managing staff, and security personnel.
- **CSS & JavaScript** – Frontend styling and client-side functionality.
- **AJAX & jQuery** – Used for interactive features like form validation and live data updates.

Example Code from the Presentation Tier:

This example from **Login.jsp** handles login inputs and sends them to LoginServlet.java for processing:

```
<div class="form-container" id="login-form" style="display:none;">
    <h2>Login</h2>
    <form action="LoginServlet" method="POST">
        <input type="email" name="email" placeholder="Email" required>
        <div class="password-container">
            <input type="password" id="login-password" name="password" placeholder="Password" required>
            <span onclick="togglePasswordVisibility('login-password', 'login-icon')" class="toggle-password">
                <i class="fas fa-eye-slash" id="login-icon"></i>
            </span>
        </div>
        <button type="submit" class="button submit">Login</button>
    </form>
</div>
```

2. Business Tier (Application Logic) – Middleware Layer

This tier consists of Java Servlets that handle business logic, process user requests, and interact with the database tier via JDBC. Servlets process HTTP requests from the Presentation Tier, retrieve data from the Database Tier, and send responses to the user interface.

Key Components in the Business Tier:

- **Servlets (Java EE Servlets)**
  - **LoginServlet.java –** Handles authentication and role-based redirection.
  - **SignupServlet.java –** Manages user registration with hashed passwords.
  - **ApproveRequestServlet.java –** Handles visitor request approval or rejection.
  - **UpdateProfileServlet.java –** Allows users to update their profile details.

Example Code from the Business Tier:

This snippet from LoginServlet.java retrieves user credentials, validates them against the database, and redirects based on user roles:

```
try (Connection conn = DBConnection.getConnection()) {
    // Query user by email
    String query = "SELECT * FROM Users WHERE LOWER(email) = LOWER(?)";
    PreparedStatement stmt = conn.prepareStatement(query);
    stmt.setString(1, email);
    ResultSet rs = stmt.executeQuery();

    if (rs.next()) {
        String storedPassword = rs.getString("password"); // Password from DB
        boolean isAuthenticated = false;

        // Check if stored password matches the entered password (plain-text case)
        if (storedPassword.equals(password)) {
            isAuthenticated = true;
        }
        // If plain-text check fails, try hashing and comparing
        else {
            String hashedPassword = hashPassword(password);
            if (storedPassword.equals(hashedPassword)) {
                isAuthenticated = true;
            }
        }
```

This business logic ensures that only authenticated users can access the system and are redirected based on their roles.

3. Database Tier (Data Storage Layer)

The database tier consists of Microsoft SQL Server and stores all user, visitor, and activity log data. The JDBC API enables communication between Java Servlets and the database.

Key Components in the Database Tier:

- **Tables:**
    - Users – Stores user information.
    - Visitor_Requests – Stores visitor requests submitted by residents.
    - Requests_Log – Keeps track of approvals and rejections.
    - Activity_Logs – Records visitor verification activities.

Example Code from the Database Tier:

The DBConnection.java class handles database connectivity using JDBC:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {
    private static final String URL = "jdbc:sqlserver://localhost:1433;databaseName=APUHostelSystem;integratedSecurity=true;encrypt=true;trustSer

    public static Connection getConnection() throws SQLException {
        try {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return DriverManager.getConnection(URL);
    }
}
```

2.2 Interconnection Among the Tiers

***The three tiers of the system interact in the following way*:**

1. **User Requests (Presentation Tier → Business Tier)**
    - The user submits a login request via Login.jsp.
    - The request is sent to LoginServlet.java, which processes authentication.
    - If successful, the user is redirected to the respective dashboard**.**
2. **Data Processing (Business Tier → Database Tier)**
    - The business tier sends SQL queries using JDBC to retrieve or update records.
    - For example, LoginServlet.java queries the Users table for authentication.
    - ApproveRequestServlet.java updates the Visitor_Requests table when a request is approved**.**
3. **Response Back to the User (Database Tier → Business Tier → Presentation Tier)**
    - The database sends the retrieved data back to the servlet.
    - The servlet processes the data and updates the JSP pages dynamically.
    - Users see real-time updates, such as visitor approval statuses changing from "Pending" to "Approved."

3.0 Presentation Tier

The Presentation Tier of the APU Hostel Visitor Verification System is responsible for managing user interactions and rendering dynamic content. It consists of JSPs (JavaServer Pages), Servlets, CSS, JavaScript, and AJAX to ensure a responsive and intuitive web experience. The system supports multiple user roles (Managing Staff, Security Staff, and Residents) with specific dashboards and functionalities tailored to each role.

3.1 Use of Web Component Technologies (JSPs, JSFs, Servlets)

Technologies Used in the Presentation Tier:

Technology	Purpose
JSP (JavaServer Pages)	Generates dynamic web pages using Java code embedded within HTML. Used for pages like dashboard.jsp, approve_request.jsp, Residents.jsp.
Java Servlets	Handles requests and responses between the frontend and backend, ensuring secure data processing. Examples: LoginServlet.java, SignupServlet.java.
CSS & JavaScript	Enhances the UI/UX with styles and interactive client-side scripts. Example: dashboard.css, Security.css.
AJAX & jQuery	Used for asynchronous data updates, such as live form submission in EditProfile.jsp and approve_request.jsp.

Examples of Web Component Technologies in Use

1. JSP Example (Login Page)

```
        <form action="LoginServlet" method="POST">
            <input type="email" name="email" placeholder="Email" required>
            <div class="password-container">
                <input type="password" id="login-password" name="password" placeholder="Password" required>
                <span onclick="togglePasswordVisibility('login-password', 'login-icon')" class="toggle-passv
                    <i class="fas fa-eye-slash" id="login-icon"></i>
                </span>
            </div>
            <button type="submit" class="button submit">Login</button>
        </form>
    </div>

    <!-- Signup Messages -->
    <% if (request.getParameter("signupSuccess") != null) { %>
    <p style="color: green; text-align: center;">✅ Account created successfully! You can now log in.</p>
    <% } else if (request.getParameter("signupFailed") != null) {
        String reason = request.getParameter("signupFailed");
        if (reason.equals("emailExists")) { %>
    <p style="color: red; text-align: center;">⊘ Account already exists, please login.</p>
    <% } else { %>
```

2. Servlet Example (Login Processing)

```java
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String email = request.getParameter("email");
        String password = request.getParameter("password");
        HttpSession session = request.getSession();

        try (Connection conn = DBConnection.getConnection()) {
            // Query user by email
            String query = "SELECT * FROM Users WHERE LOWER(email) = LOWER(?)";
            PreparedStatement stmt = conn.prepareStatement(query);
            stmt.setString(1, email);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                String storedPassword = rs.getString("password"); // Password from DB
                boolean isAuthenticated = false;

                // Check if stored password matches the entered password (plain-text case)
                if (storedPassword.equals(password)) {
                    isAuthenticated = true;
```

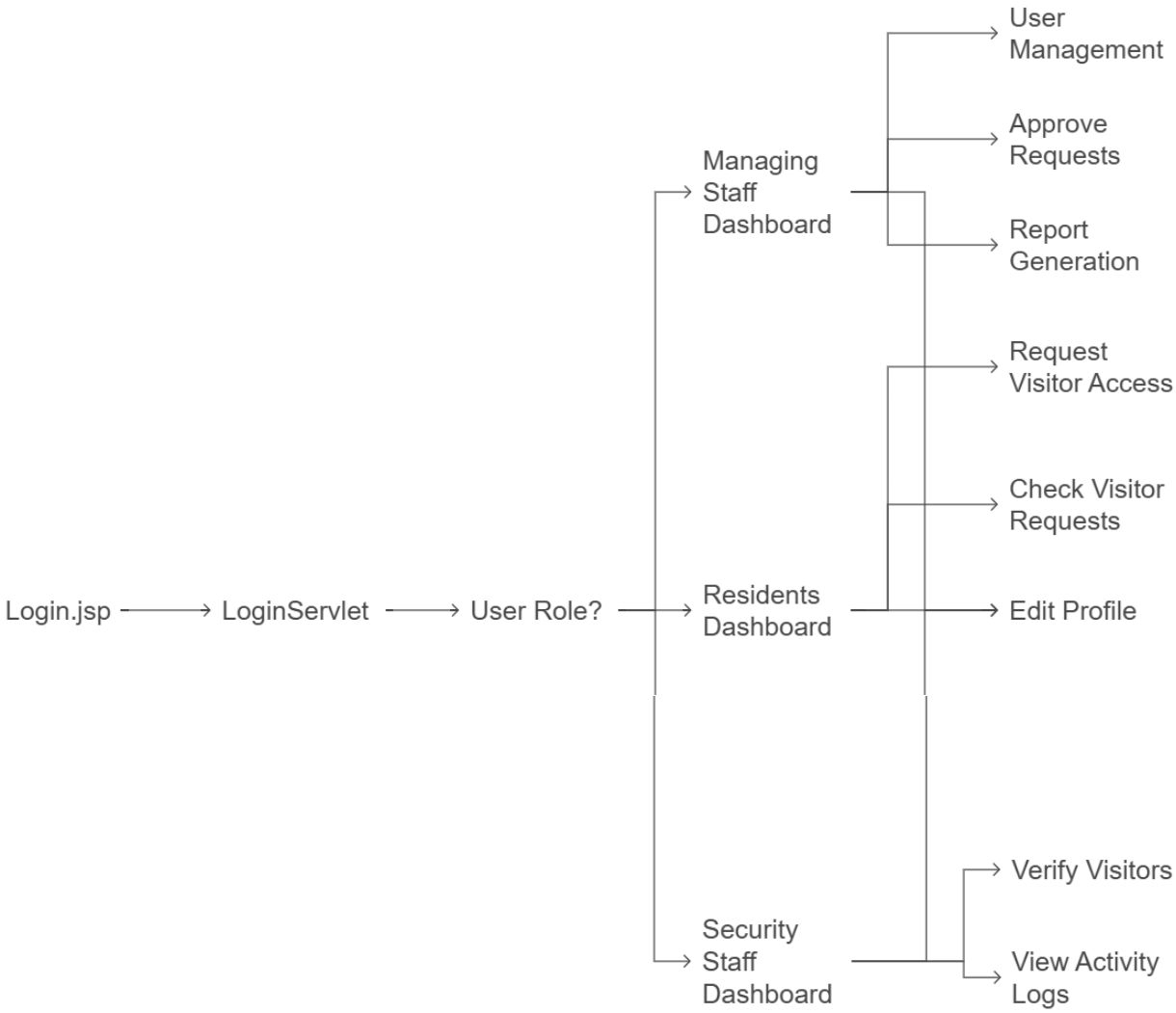
3. AJAX Example (Profile Update without Page Refresh)

```javascript
<script>
    $(document).ready(function () {
        $("#editProfileForm").submit(function (event) {
            event.preventDefault();
            var formData = new FormData(this);

            $.ajax({
                url: "UpdateProfileServlet",
                type: "POST",
                data: formData,
                processData: false,
                contentType: false,
                success: function (response) {
                    $("#messageBox").html(response).show();
                    setTimeout(() => {
                        $("#messageBox").fadeOut();
                        location.reload(); // Reload page to reflect updates
                    }, 2000);
                }
            });
        });
```

3.2 Web Page Design and Navigation Chart

Navigation Flowchart



Web Page Design

Each user role has a unique dashboard with a sidebar navigation menu, content area, and interactive elements. The following explains how the dashboard looks for every role:

Managing Staff Dashboard (dashboard.jsp)

- Displays the **number of active residents** and **pending visitor requests**.
- Uses a **pie chart** to visualize system activity.
- Links to **user management, request approval, and report generation**.

Residents Pages (Residents.jsp, CheckRequest.jsp)

- Residents **submit visitor requests** and track approvals.

- Requests are listed in a **table with real-time status updates**.

Security Pages (VisitorVerification.jsp, ActivityLogs.jsp)

- Security **verifies visitors** using a search-based verification system.
- **Activity logs** display verified visitor entries.

4.0 Database Tier

The Database Tier of the APU Hostel Visitor Verification System is responsible for data storage, retrieval, and management. It ensures that all user information, visitor requests, and activity logs are securely stored in MS SQL Server.

4.1 Database Design and Schemas

The system uses a relational database model with multiple tables to store different types of data, including user details, visitor requests, approvals, and activity logs.

Database: APUHostelSystem

The database consists of four primary tables:

1. **Users** – Stores user details and roles.
2. **Visitor_Requests – Stores visitor request information.**
3. **Requests_Log** – Stores request approval and rejection records.
4. **Activity_Logs** – Stores visitor verification records.

4.2 Description of Each Table

1. Users Table

Purpose: Stores user details (Residents, Managing Staff, Security Staff).

Columns:

Column Name	Data Type	Description
user_id	INT (PK, AUTO_INCREMENT)	Unique ID for each user
name	VARCHAR(100)	Full name of the user
email	VARCHAR(100) UNIQUE	User's email (used for login)
phone_number	VARCHAR(20)	Contact number
dob	DATE	Date of birth
gender	VARCHAR(10)	Gender (Male/Female)
password	VARCHAR(256)	Hashed password
role	VARCHAR(50)	User role (Managing Staff, Resident, Security Staff)
profile_picture	VARCHAR(255)	Profile image filename

Purpose: Stores visitor requests submitted by residents.

2. Visitor_Requests Table

Column Name	Data Type	Description
request_id	INT (PK, AUTO_INCREMENT)	Unique ID for each request
user_id	INT (FK → Users.user_id)`	Resident who created the request
visitor_name	VARCHAR(100)	Visitor's name
visitor_email	VARCHAR(100)	Visitor's email
visitor_phone	VARCHAR(20)	Visitor's phone number
visitor_id	VARCHAR(50)	Visitor’s ID/Passport
visitor_gender	VARCHAR(10)	Visitor’s gender
purpose	VARCHAR(255)	Reason for visit
visit_date	DATE	Date of visit
Column Name	**Data Type**	**Description**
request_id	INT (PK, AUTO_INCREMENT)	Unique ID for each request
user_id	INT (FK → Users.user_id)`	Resident who created the request

Purpose: Stores records of approved or rejected visitor requests.
Columns:

3. Requests_Log Tabl

Column Name	Data Type	Description
log_id	INT (PK, AUTO_INCREMENT)	Unique log entry ID
request_id	INT (FK → Visitor_Requests.request_id)`	Associated visitor request
approved_by	INT (FK → Users.user_id)`	Managing staff who approved/rejected request
approval_time	DATETIME	Time of approval/rejection
status	VARCHAR(20)	Approval status (Approved/Rejected)
comments	TEXT	Additional comments from managing staff

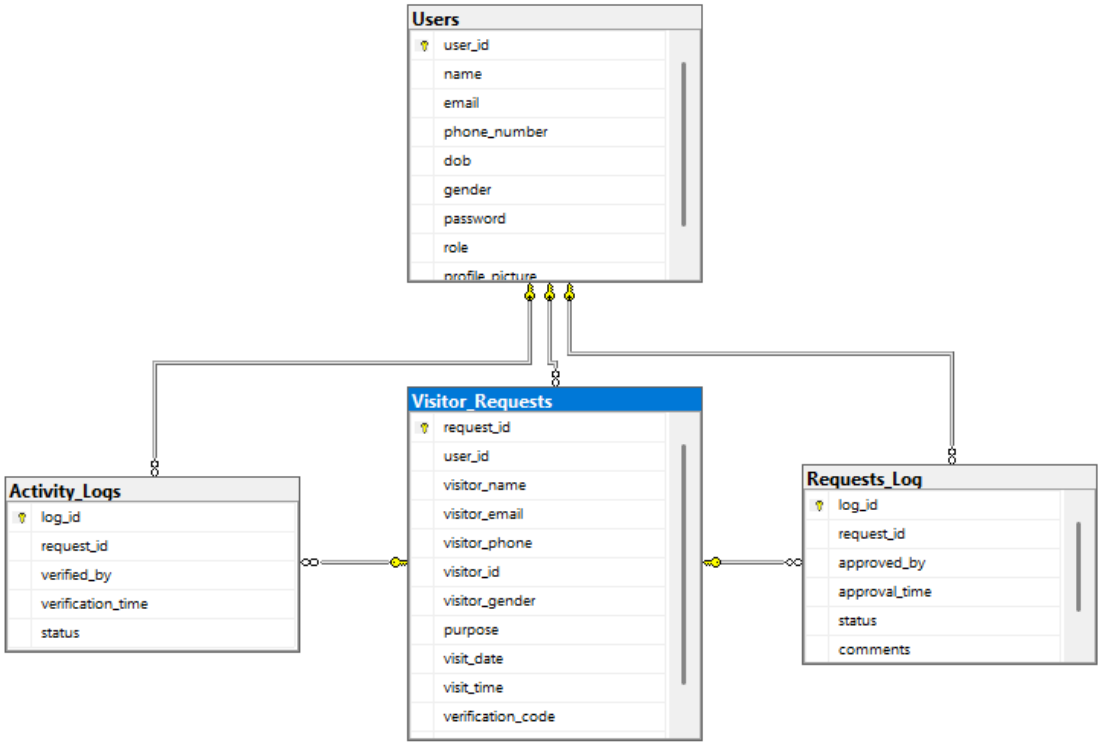
4. Activity_Logs Table

Purpose: Stores logs for visitor verification at security checkpoints.

Column Name	Data Type	Description
log_id	INT (PK, AUTO_INCREMENT)	Unique log entry ID
request_id	INT (FK → Visitor_Requests.request_id)`	Associated visitor request
verified_by	INT (FK → Users.user_id)`	Security staff who verified visitor
verification_time	DATETIME	Time of verification
status	VARCHAR(20)	Verification status (Verified/Failed)

4.3 Entity-relationship diagram or Domain Diagram

The ERD diagram illustrates the relationships between the primary tables in the APU Hostel Visitor Verification System. It defines the data structure and how different entities interact within the system.



1. Users Table (Primary Entity)

- Stores all user data, including residents, security staff, and managing staff.
- Each user is identified by user_id.
- Users can have different roles: Resident, Security Staff, and Managing Staff.

2. Visitor_Requests Table (Core Entity)

- Stores all visitor requests submitted by residents.
- user_id is a foreign key linking each request to the resident who submitted it.
- Each request contains visitor details, purpose, date, and a unique verification code.
- This table is directly linked to Requests_Log and Activity_Logs.

3. Requests_Log Table

- Maintains a record of approvals and rejections for visitor requests.
- request_id is a foreign key linking to Visitor_Requests.
- approved_by references Users (Managing Staff) who approved or rejected the request.
- Stores approval time, status, and comments for tracking purposes.

4. Activity_Logs Table

- Stores visitor verification records.
- request_id is a foreign key linking to Visitor_Requests.
- verified_by references Users (Security Staff) who verified the visitor.
- Stores verification time and status (e.g., Verified, Failed).

Key Relationships:

1. **Users** → Visitor_Requests (One-to-Many)
    - A resident can submit multiple visitor requests.
2. **Visitor_Requests** → Requests_Log (One-to-One)
    - Each visitor request can be approved or rejected once.
3. **Visitor_Requests** → Activity_Logs (One-to-One)
    - Each visitor request can be verified once by security staff.
4. **Users** → Requests_Log (One-to-Many)
    - A managing staff member can approve multiple requests.
5. **Users** → Activity_Logs (One-to-Many)
    - A security staff member can verify multiple visitors.

4.4 Design of Database Access APIs

1. Database Connection API (DBConnection.java)

This class establishes connections to MS SQL Server.

```
public class DBConnection {
    private static final String URL = "jdbc:sqlserver://localhost:1433;databaseName=APUHostelSystem;integratedSecurity=true;encrypt=true;trust

    public static Connection getConnection() throws SQLException {
        try {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return DriverManager.getConnection(URL);
    }
}
```

2. Query Execution (Using JDBC in Servlets)

Each servlet interacts with the database using JDBC.

```
try {
    String DB_URL = "jdbc:sqlserver://localhost:1433;databaseName=APUHostelSystem;integratedSecurity=true;encrypt=true;trustServerCertificate
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
    Connection conn = DriverManager.getConnection(DB_URL);

    String sql = "SELECT name, email, phone_number, dob, gender, password, profile_picture FROM Users WHERE user_id=?";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, userId);
    ResultSet rs = pstmt.executeQuery();

    if (rs.next()) {
        name = rs.getString("name");
        email = rs.getString("email");
        phoneNumber = rs.getString("phone_number");
        dob = rs.getString("dob");
        gender = rs.getString("gender");
        password = rs.getString("password");

        String dbImage = rs.getString("profile_picture");
        if (dbImage != null && !dbImage.isEmpty()) {
            profileImage = "uploads/" + dbImage; // Ensure the correct file path
        }
    }
```

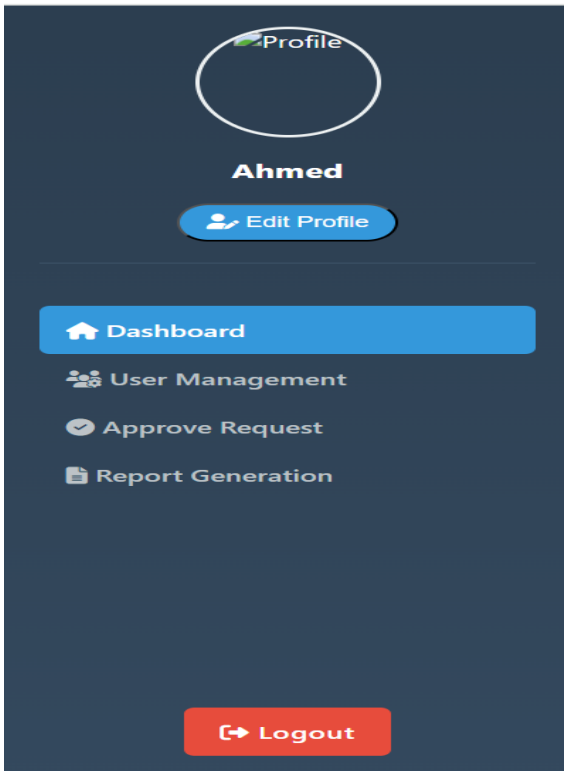
3. Insert, Update, and Delete APIs

```
String requestId = request.getParameter("request_id");
if (requestId != null) {
    String updateSQL = "UPDATE Visitor_Requests SET status=? WHERE request_id=?";
    PreparedStatement updateStmt = conn.prepareStatement(updateSQL);
    updateStmt.setString(1, action.equals("approve") ? "Approved" : "Rejected");
    updateStmt.setInt(2, Integer.parseInt(requestId));
    updateStmt.executeUpdate();
    updateStmt.close();
    out.print("Request " + (action.equals("approve") ? "Approved" : "Rejected") + " Successfully!");
    return;
```

5.0 User Interface Design

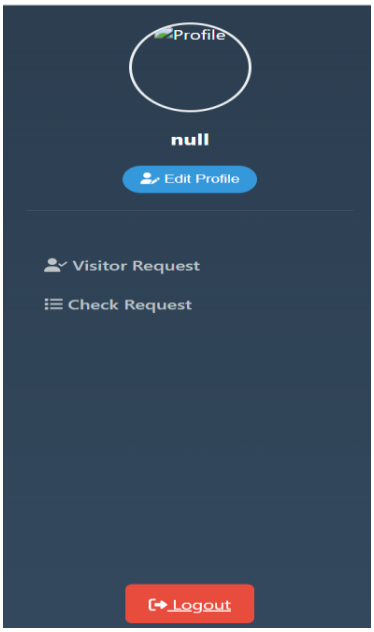
5.1 General Navigation

➢ Managing Staff Interface



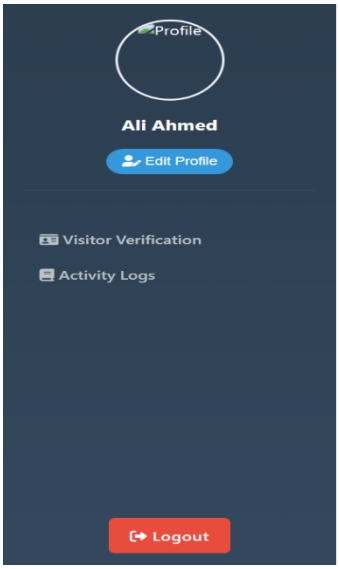
- The managing staff dashboard includes options for User Management, Approve Requests, and Report Generation.
- The user profile is displayed with an Edit Profile option

➢ Resident Interface



- Residents can manage their Visitor Requests and Check Requests.
- The profile name appears as "null," indicating a potential issue with session handling.

➢ Security Staff Interface

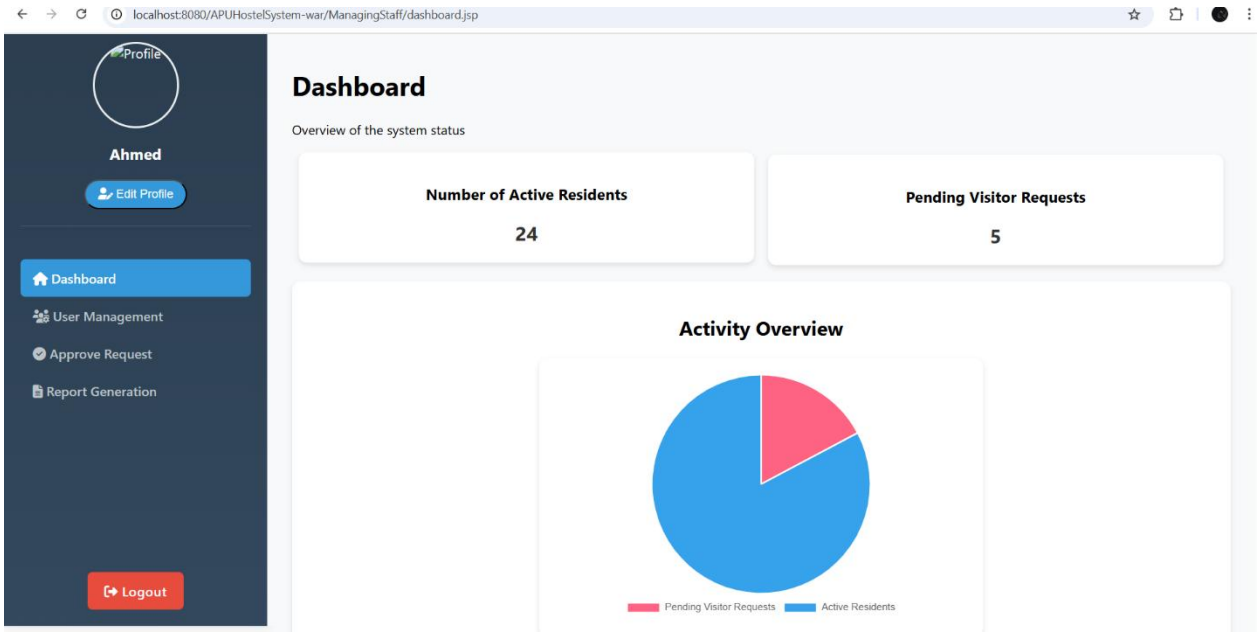


- Security staff can access Visitor Verification and Activity Logs.
- The interface provides a profile section with an Edit Profile button.

5.2 Screenshots and Descriptions of Each User Interface:

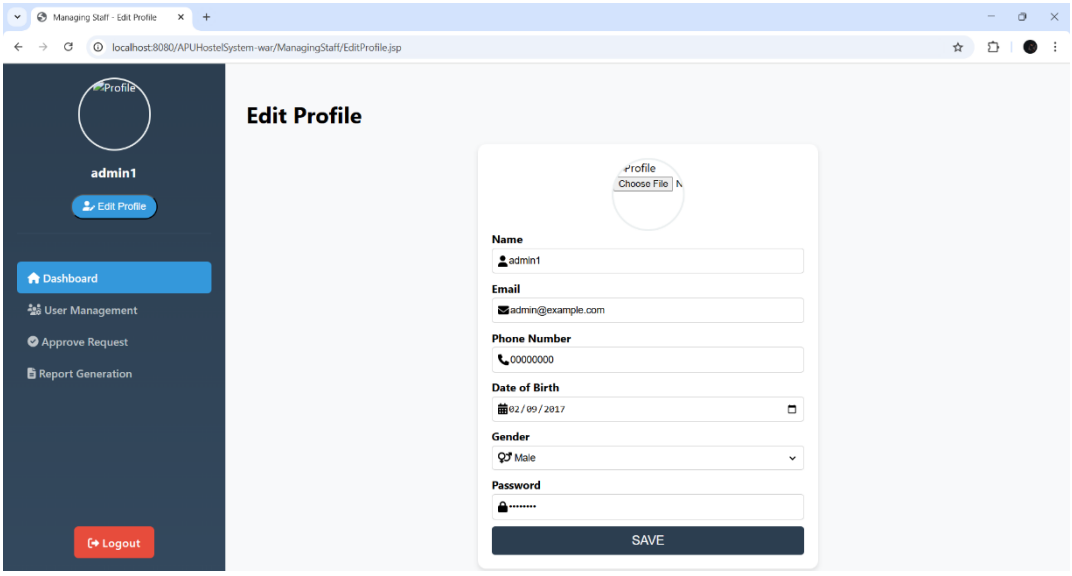
- Managing Staff Interface

Dashboard



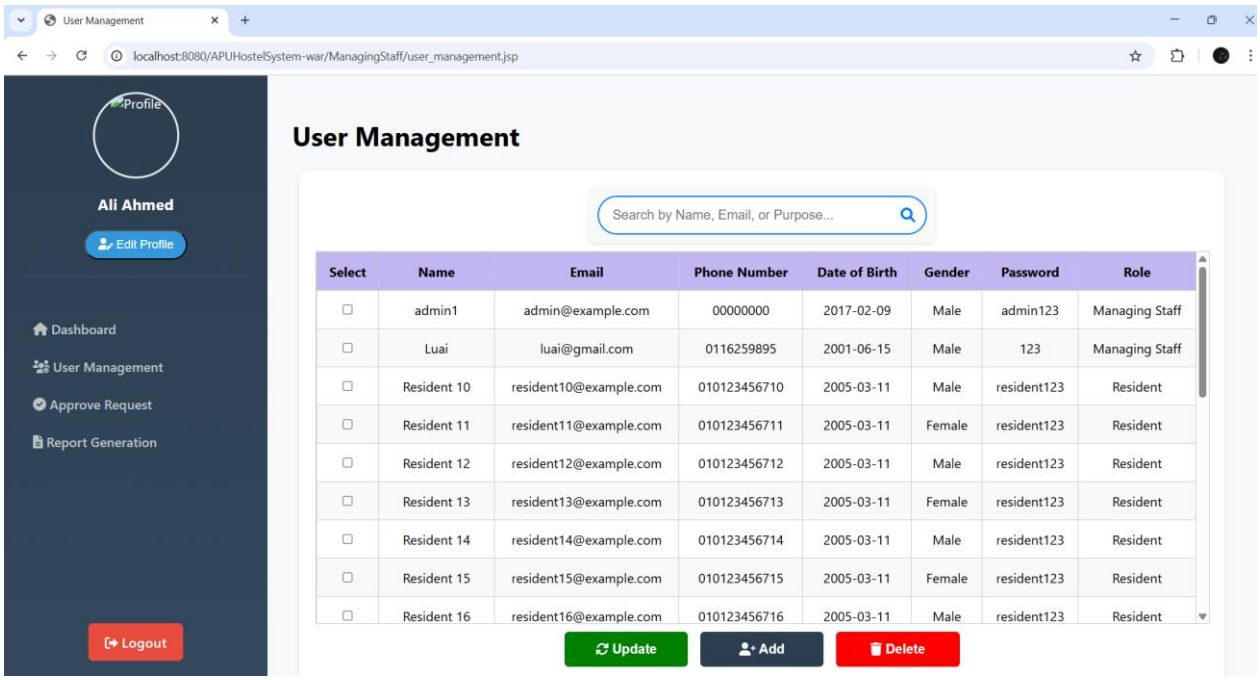
The Managing Staff Dashboard provides an overview of the system, displaying the number of active residents and pending visitor requests, along with a visual activity overview using a pie chart. The left sidebar allows navigation to User Management, Approve Requests, and Report Generation, enabling staff to manage users, approve visitor requests, and generate reports efficiently. The Logout button ensures secure sign-out.

- **Edit Profile**



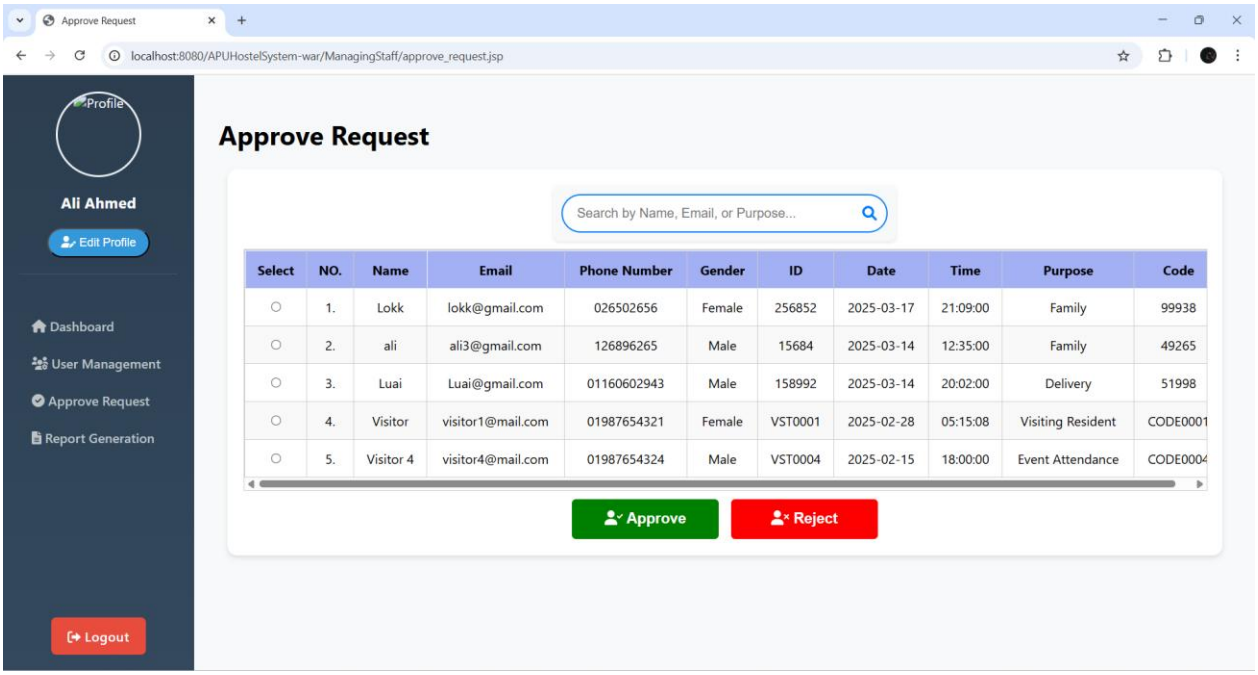
The Edit Profile Page allows users to view and update their personal information, including name, email, phone number, date of birth, gender, and password. Users can also upload a profile picture. After making changes, clicking the Save button updates the Users database and displays a success message, confirming the profile update.

- **User Management**



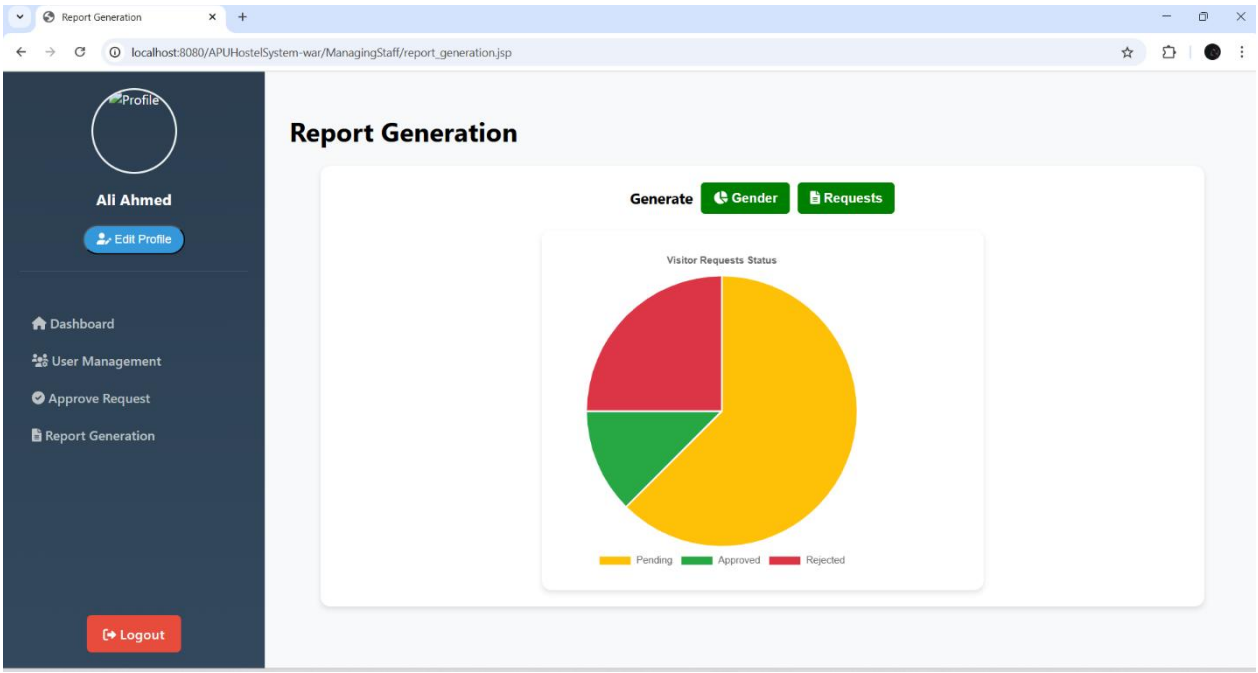
The User Management Page enables managing staff to view, search, add, update, and delete user records in the system. The table displays user details such as name, email, phone number, date of birth, gender, password, and role. Staff can search for users using the search bar and perform actions like updating user information, adding new users, or deleting existing ones, ensuring efficient user administration.

- **Approve Request**



The **Approve Request Page** allows managing staff to review all visitor requests, including details such as name, email, phone number, gender, ID, visit date, time, purpose, and verification code. Staff can search for specific requests and take action by either approving or rejecting them, ensuring proper visitor management and security compliance.

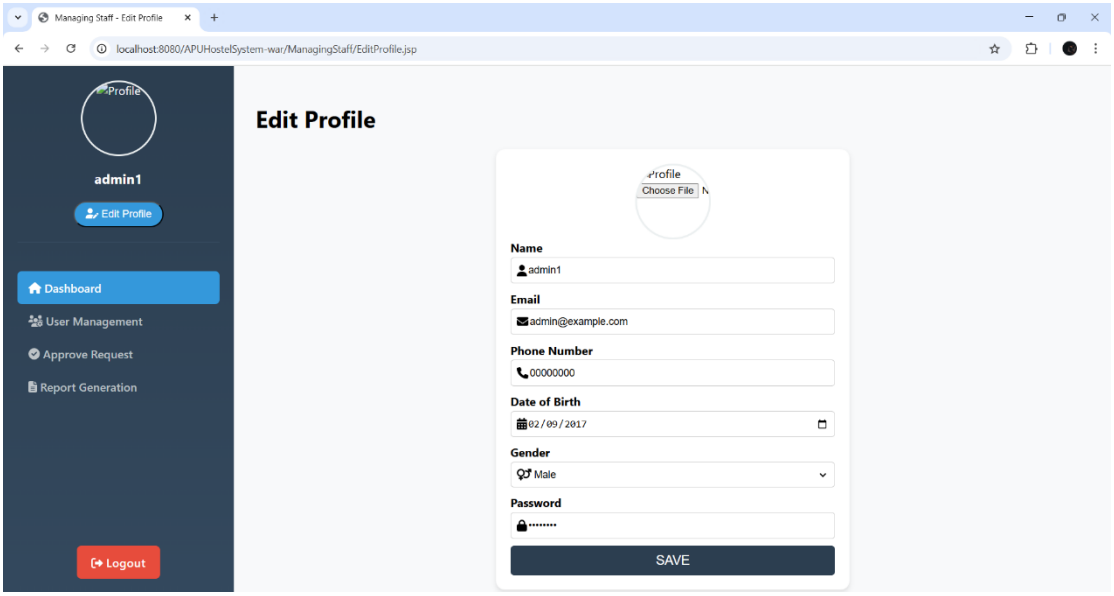
- **Report Generation**



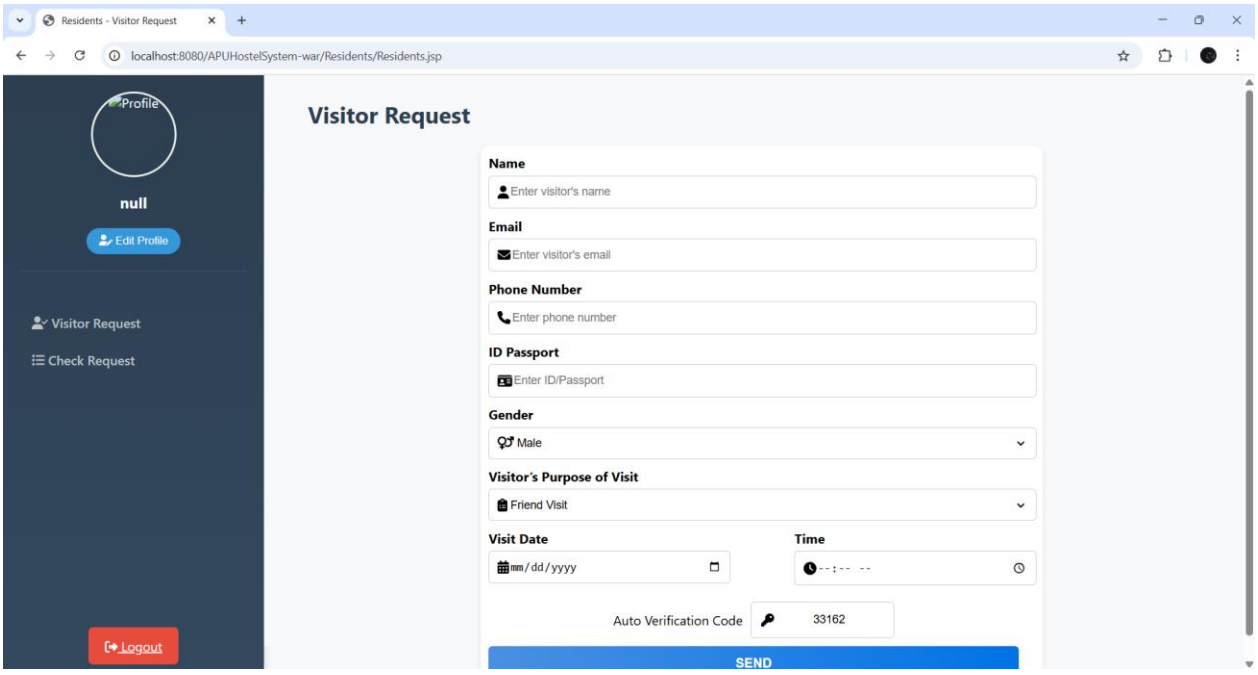
The **Report Generation Page** provides managing staff with a visual representation of visitor request statuses through pie charts. Users can generate reports based on visitor gender or request status, aiding in data-driven decision-making and efficient system management.

- Resident Interface

Edit Profile

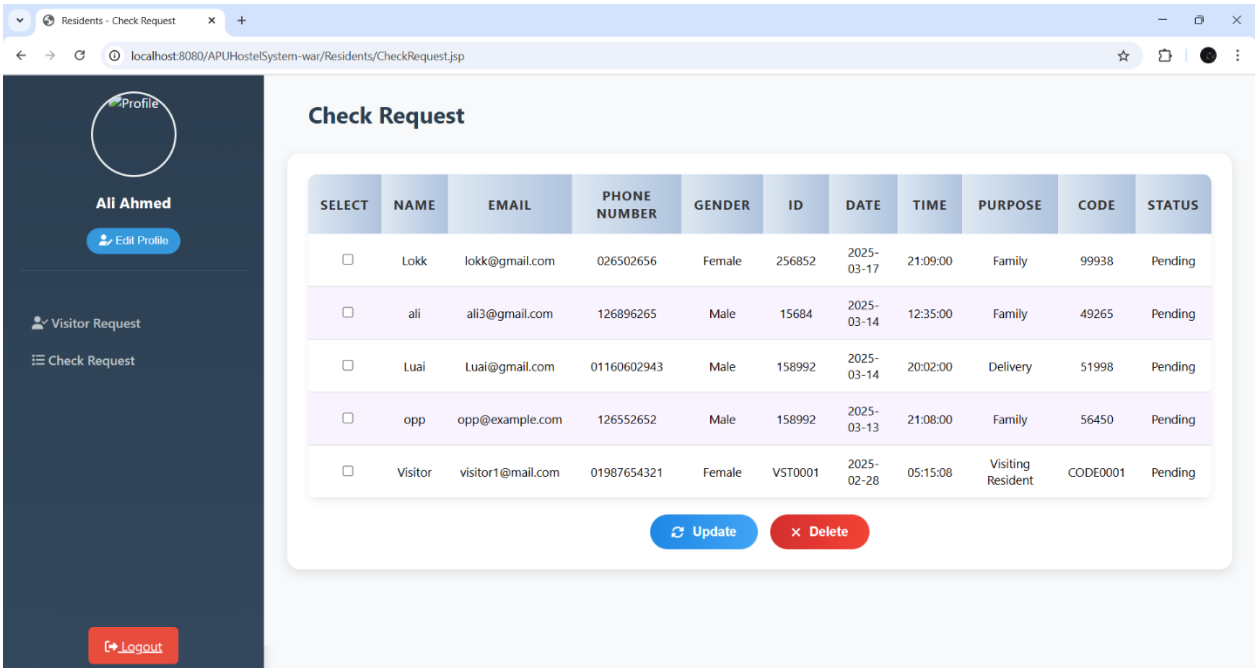


Visitor Request



The Visitor Request Page allows residents to submit visitor requests by providing visitor details such as name, email, phone number, and purpose of visit. The system automatically generates a unique verification code for each request to ensure secure tracking and management.

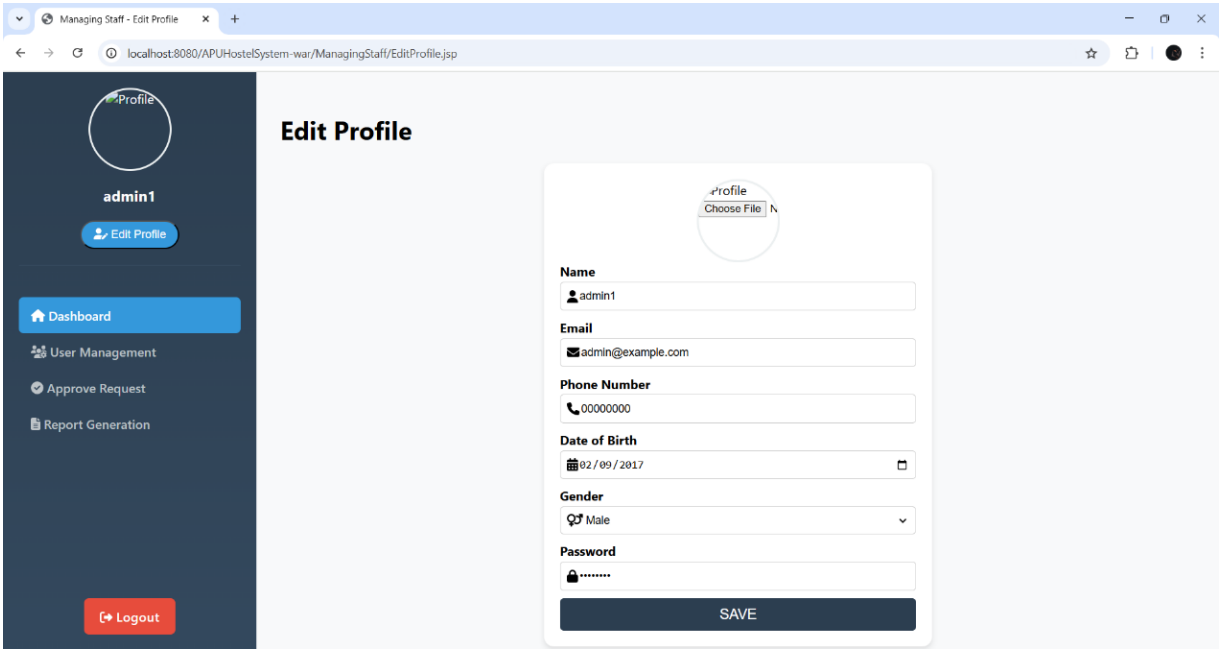
Check Request



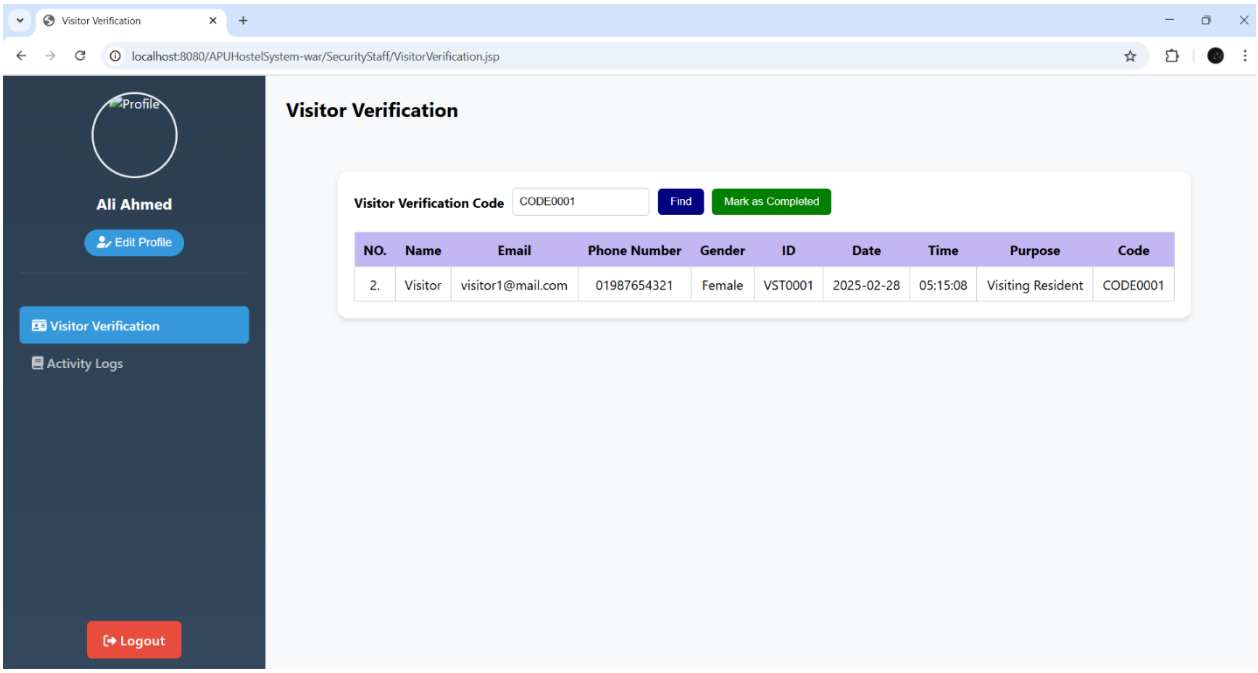
The **Check Request Page** allows residents to track the status of their submitted visitor requests. They can view details such as visitor name, email, phone number, visit date, and purpose. Additionally, residents can update or delete pending requests before the staff processes them.

- security Staff Interface

Edit profile

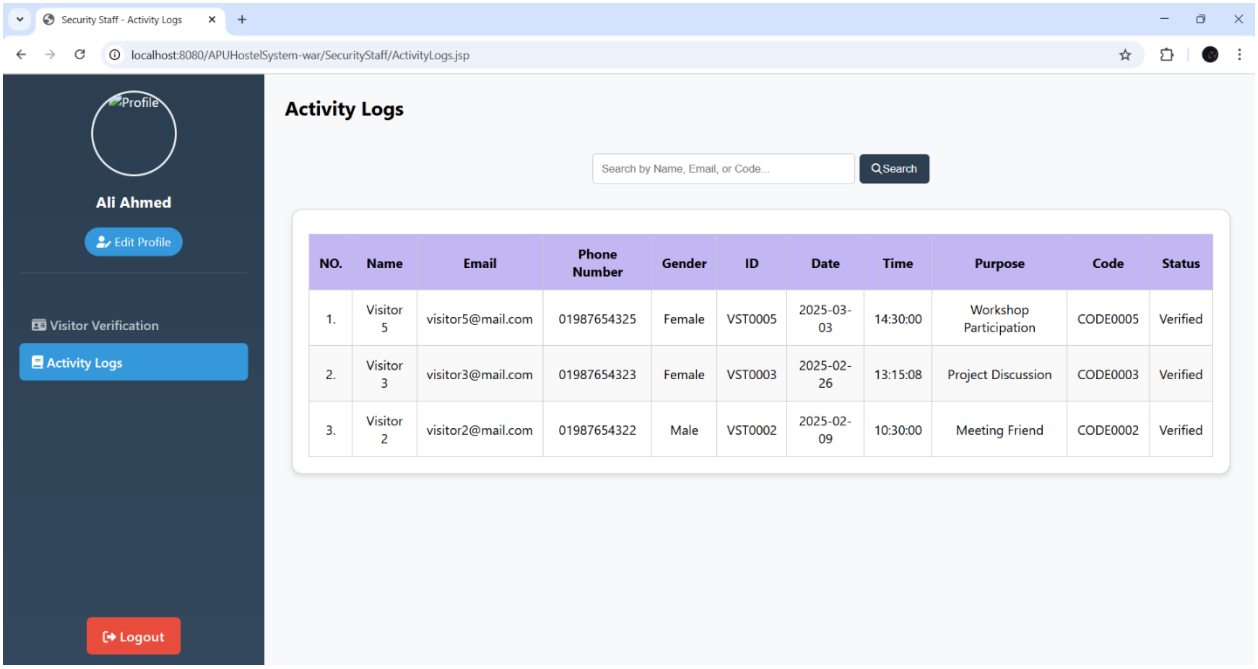


Visitor Verification



The **Visitor Verification Page** allows security staff to verify visitor entries by entering a unique verification code. Visitor details are displayed if the code is valid, confirming whether they have an approved visit. Security staff can then mark the visit as completed for logging purposes.

Activity Logs



The **Activity Logs Page** displays a record of all verified visitor entries. It allows security staff to track completed visits, showing visitor information, visit time, purpose, verification code, and status. This ensures transparency and accountability in visitor management.

6.0 Additional Features

Automated Verification Code Generation

Each visitor request is assigned a unique 5-digit verification code at the time of request submission. This code helps security staff verify visitors upon arrival.

```
if (session.getAttribute("verification_code") == null) {
    Random random = new Random();
    int verificationCode;
    boolean codeExists;
    int attempts = 0;
    int maxAttempts = 10;

    do {
        verificationCode = 10000 + random.nextInt(90000); // Generate 5-digit number
        attempts++;

        try (Connection conn = DriverManager.getConnection("jdbc:sqlserver://localhost:1433;databaseName=APUHostelSystem;integratedSecurity=
             PreparedStatement pstmt = conn.prepareStatement("SELECT COUNT(*) FROM Visitor_Requests WHERE verification_code = ?")) {

            pstmt.setInt(1, verificationCode);
            ResultSet rs = pstmt.executeQuery();
            rs.next();
            codeExists = rs.getInt(1) > 0;
        } catch (Exception e) {
            codeExists = false; // Assume unique if error occurs
        }
    } while (codeExists && attempts < maxAttempts);
```

Role-Based Access Control (RBAC)

```
// Redirect based on role
String redirectUrl = request.getContextPath();
switch (rs.getString("role")) {
    case "Managing Staff":
        redirectUrl += "/ManagingStaff/dashboard.jsp";
        break;
    case "Resident":
        redirectUrl += "/Residents/Residents.jsp";
        break;
    case "Security Staff":
        redirectUrl += "/SecurityStaff/Security.jsp";
        break;
    default:
        request.setAttribute("error", "role");
        request.getRequestDispatcher("Login/Login.jsp").forward(request, response);
        return;
}
response.sendRedirect(redirectUrl);
return;
```

Different user roles (Managing Staff, Residents, Security Staff) have specific permissions and functionalities.

Graphical Report Generation

```
let chartInstance;
const ctx = document.getElementById('reportChart').getContext('2d');

function generateGenderChart() {
    const data = {
        labels: ["Male", "Female"],
        datasets: [{
            data: [<%= maleCount %>, <%= femaleCount %>],
            backgroundColor: ['#007bff', '#ff6384'],
            hoverOffset: 4
        }]
    };
    updateChart(data, "Users Gender Distribution");
}

function generateRequestChart() {
    const data = {
        labels: ["Pending", "Approved", "Rejected"],
        datasets: [{
            data: [<%= pendingCount %>, <%= approvedCount %>, <%= rejectedCount %>],
            backgroundColor: ['#ffc107', '#28a745', '#dc3545'],
            hoverOffset: 4
```

Managing staff can generate reports based on visitor request statistics.

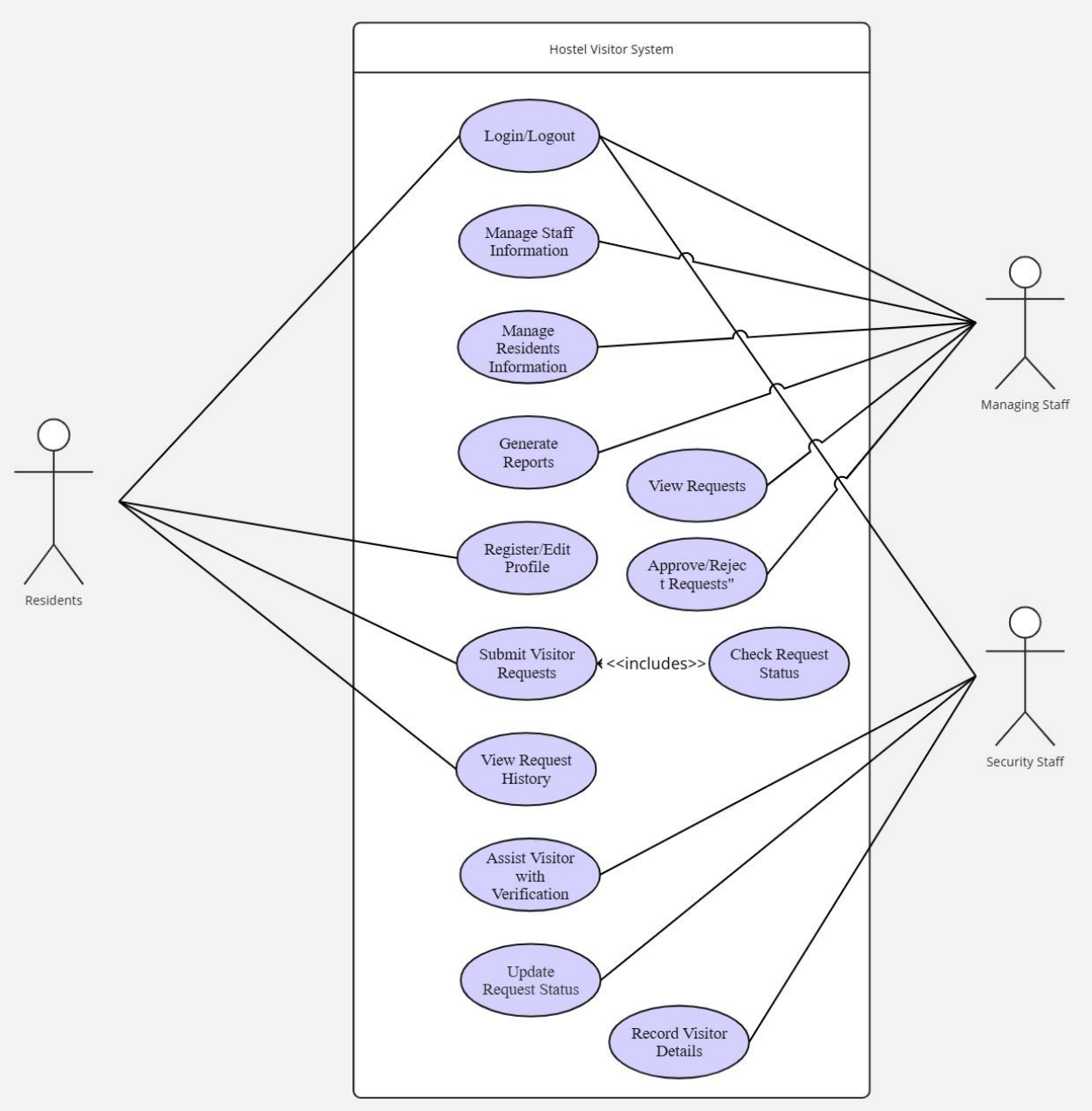
Secure Password Hashing

User passwords are stored securely using SHA-256 hashing.

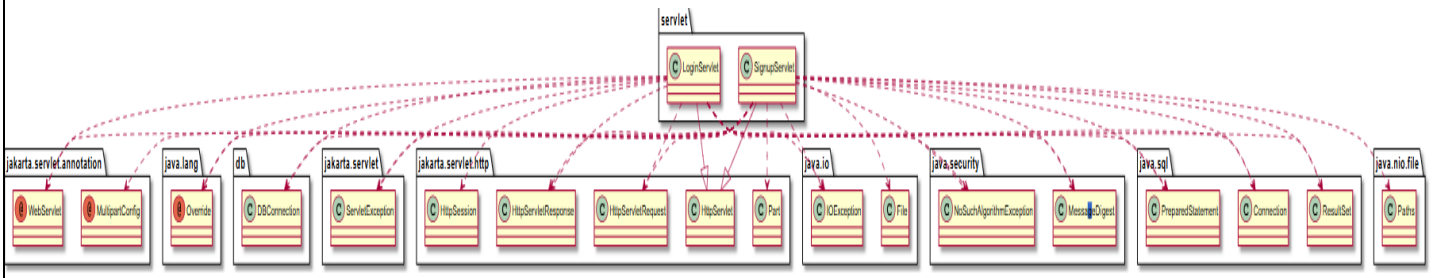
```
    // Hash Password Using SHA-256
    private String hashPassword(String password) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hashedBytes = md.digest(password.getBytes());
            StringBuilder hexString = new StringBuilder();
            for (byte b : hashedBytes) {
                hexString.append(String.format("%02x", b));
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

7.0 UML Diagrams

7.1 Use Case Diagrams



7.2 Class Diagrams



8.0 References

1. GeeksforGeeks. (2022, May 23). *What is a Distributed System?* GeeksforGeeks. https://www.geeksforgeeks.org/what-is-a-distributed-system/#characteristics-of-distributed-system
2. Kinza Yasar, & Gillis, A. S. (2024). *What is distributed computing?* WhatIs; TechTarget. https://www.techtarget.com/whatis/definition/distributed-computing
3. GeeksforGeeks. (2024, June 28). *ClientServer Architecture System Design*. GeeksforGeeks. https://www.geeksforgeeks.org/client-server-architecture-system-design/
4. GeeksforGeeks. (2020, April 22). *What is P2P (PeertoPeer Process)?* GeeksforGeeks. https://www.geeksforgeeks.org/what-is-p2p-peer-to-peer-process/
5. GeeksforGeeks. (2019, October 23). *ClientServer Model*. GeeksforGeeks. https://www.geeksforgeeks.org/client-server-model/
6. GeeksforGeeks. (2019, October 10). *Grid Computing*. GeeksforGeeks. https://www.geeksforgeeks.org/grid-computing/
7. Sankar, H. (2023, June 19). *Challenges in Distributed Systems - Scaler Topics*. Scaler Topics. https://www.scaler.com/topics/challenges-of-distributed-system/
8. Oleksii Dushenin. (2021, October 22). *Monolithic Architecture. Advantages and Disadvantages*. Medium. https://datamify.medium.com/monolithic-architecture-advantages-and-disadvantages-e71a603eec89
9. *Layered Architecture*. (2024). OpenClassrooms. https://openclassrooms.com/en/courses/6397806-design-your-software-architecture-using-industry-standard-patterns/6896176-layered-architecture
10. Vitaly Kuprenko. (2019, November 14). *Stackify*. Stackify. https://stackify.com/6-key-benefits-of-microservices-architecture/
11. GeeksforGeeks. (2018, August 2). *ServiceOriented Architecture*. GeeksforGeeks. https://www.geeksforgeeks.org/service-oriented-architecture/
12. RobBagby. (2024). *Event-driven architecture style - Azure Architecture Center*. Microsoft.com. https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven
13. *Five-layer CRM application architecture*. (2020, November). Slideteam.net. https://www.slideteam.net/five-layer-crm-application-architecture.html

14. *What is CRM & Best CRM Software Suitable for your Business - Agile CRM Blog*. (2022, March 2). Agile CRM Blog. https://www.agilecrm.com/blog/what-is-crm-best-crm-software-suitable-for-your-business/
15. *What is Hybrid CRM and its Importance? - CrmOne*. (2024, June 28). Crmone. https://www.crmone.com/glossary/hybrid-crm
16. Team Atlan. (2023, May 20). *Cloud vs On-Premise vs Hybrid: Which One is Best for You?* Atlan.com; Atlan. https://atlan.com/cloud-vs-on-premise-vs-hybrid/
17. *What is an On-Premise CRM? | Coldlytics*. (2023). Coldlytics.com. https://www.coldlytics.com/glossary/on-premise-crm
18. Zur, A. (2024, November 20). Creatio.com; Creatio. https://www.creatio.com/glossary/cloud-crm#:~:text=A%20cloud-based%20CRM%20is%20a%20CRM%20system%20hosted,unit%29%20can%20work%20with%20the%20same%20data%20simultaneously.
19. *Hybrid Cloud*. (2021). Inspiredpencil.com. https://ar.inspiredpencil.com/pictures-2023/hybrid-cloud
20. java. (2017, March 16). *java to connect Microsoft Dynamics 365*. Stack Overflow. https://stackoverflow.com/questions/42826395/java-to-connect-microsoft-dynamics-365
21. Salesforce. (2025). *Salesforce Developers*. Salesforce.com. https://developer.salesforce.com/docs/atlas.en-us.api_asynch.meta/api_asynch/asynch_api_code_walkthrough.htm

22. GHL Automation. (2024, November 30). *CRM Data Security Best Practices- GHL Automation*. GHL Automation. https://ghlautomation.com/protecting-customer-data-security-best-practices-for-your-crm-system/#:~:text=CRM%20Data%20Security%20Best%20Practices%201%201.%20Regular,Awareness%20...%206%206.%20Limit%20Data%20Access%20

23. Salesforce. (2020, June 22). *Caching In The Salesforce Platform*. Salesforce.com. https://developer.salesforce.com/blogs/2020/06/caching-in-the-salesforce-platform

24. HubSpot, & HubSpot, I. (2025). *HubSpot Developer Documentation*. HubSpot Developer Documentation. https://developers.hubspot.com/docs/guides/api/crm/understanding-the-crm

25. *Business Rules Engine*. (2024, October 22). Sparkling Logic. https://www.sparklinglogic.com/business-rules/business-rules-engine/

26.