# Attendance Management

**Introduction**

In the realm of academic administration, the management of student attendance stands as a cornerstone for ensuring student engagement, academic success, and compliance with institutional policies. Colleges and universities grapple with the challenge of accurately tracking student attendance while maintaining administrative efficiency and transparency. Traditional methods of attendance management, often reliant on manual processes and paper-based systems, are not only time-consuming but also prone to errors and inefficiencies.

The project on "College Attendance Management" endeavors to address these challenges by leveraging digital technologies to streamline and enhance the attendance tracking process. By developing a robust attendance management system tailored to the unique needs of higher education institutions, this project aims to revolutionize how colleges monitor and manage student attendance.

This introduction sets the stage for an exploration into the intricacies of attendance management in college environments. By examining current practices, identifying pain points, and envisioning future possibilities, we aim to lay the groundwork for the development and implementation of an innovative attendance management solution.

Through meticulous research, analysis, and collaboration with stakeholders, this project seeks to propose a comprehensive system that not only automates attendance tracking but also provides valuable insights to educators and administrators. By embracing technology and adopting a data-driven approach, colleges can enhance administrative efficiency, improve student accountability, and ultimately foster a more conducive learning environment.

As we embark on this journey to redefine attendance management in higher education, our goal is not merely to address existing challenges but to unlock new opportunities for innovation and improvement. By embracing the digital transformation of attendance tracking, colleges can pave the way for a more efficient, transparent, and student-centric approach to academic administration.

**Abstraction**

The project on "College Attendance Management" aims to address the challenges faced by educational institutions in efficiently monitoring and managing student attendance. Attendance tracking is a crucial aspect of academic administration, providing valuable insights into student engagement, performance, and compliance with attendance policies. However, traditional methods of manual attendance recording are often prone to errors, time-consuming, and lack real-time monitoring capabilities.

This project delves into the development and implementation of a digital attendance management system tailored specifically for college environments. Through the use of innovative technologies such as biometric authentication, RFID (Radio Frequency Identification), or mobile applications, the system seeks to streamline the attendance tracking process, enhance accuracy, and provide timely insights to educators and administrators.

By conducting thorough research and analysis, the project aims to identify the key requirements and challenges associated with attendance management in college settings. This includes factors such as scalability, data security, and integration with existing academic systems. Additionally, the project explores best practices and case studies from similar implementations to inform the design and implementation of the proposed solution.

The ultimate goal of the project is to develop a comprehensive attendance management system that meets the specific needs of college institutions, improves administrative efficiency, and enhances student engagement. Through collaboration with stakeholders, including faculty, administrators, and IT professionals, the project aims to ensure the successful deployment and adoption of the system, paving the way for a more streamlined and data-driven approach to attendance monitoring in higher education.

**Project Overview**

The project on "College Attendance Management" aims to revolutionize the way higher education institutions monitor and manage student attendance. Attendance tracking plays a crucial role in academic administration, providing valuable insights into student engagement, performance, and compliance with institutional policies. However, traditional methods of attendance management, such as manual recording or paper-based systems, are often cumbersome, error-prone, and lack real-time monitoring capabilities.

This project seeks to address these challenges by developing and implementing a digital attendance management system specifically tailored to the needs of college environments. Through the utilization of innovative technologies such as biometric authentication, RFID (Radio Frequency Identification), or mobile applications, the system aims to streamline the attendance tracking process, enhance accuracy, and provide timely insights to educators and administrators.

The project will begin with a comprehensive analysis of current attendance management practices, identifying pain points, inefficiencies, and areas for improvement within college settings. This will be followed by the design and development of the attendance management system, taking into account factors such as scalability, data security, and integration with existing academic systems.

Collaboration with stakeholders, including faculty, administrators, and IT professionals, will be integral to the success of the project. By incorporating feedback and insights from these key stakeholders, the project aims to ensure that the developed solution meets the specific needs and requirements of higher education institutions.

Ultimately, the project seeks to empower colleges and universities with a comprehensive attendance management solution that enhances administrative efficiency, improves student accountability, and fosters a more conducive learning environment. By embracing digital innovation in attendance tracking, higher education institutions can stay ahead of the curve and provide a seamless, transparent, and data-driven approach to academic administration

**Hardware & Software Requirement:**

**Hardware Interfaces**

* Minimum Hardware requirement
* Processor: P4 3.0 GHz
* RAM:1 GB or Higher
* Monitor
* Mouse
* Hard disk: 80 GB

**Software Interfaces**

* Minimum Software requirement
* Java (JSP and Servlet)

1. Apache Tomcat Server
2. MYSQL

All these types of software automatic configure inside operating system after installation it having Java, MYSQL, Apache and operating system base configuration file, it doesn’t need to configure manually.

**Technologies Used**

The Attendance Management utilizes a combination of technologies to create a robust and efficient platform for managing Attendance Management operations. Here's a description of the key technologies used in the project:

**Java:** Java serves as the primary programming language for both backend and frontend development. Its platform independence, robustness, and extensive ecosystem make it an ideal choice for building scalable web applications.

**JSP (JavaServer Pages):** JSP is a technology used for creating dynamic web pages in Java. It allows embedding Java code directly into HTML pages, enabling the creation of dynamic content and server-side logic for generating web responses.

**Servlet:** A servlet is a Java class that extends the functionality of a web server, enabling the server to dynamically process and respond to client requests over the HTTP protocol. Servlets run on the server-side and handle tasks such as generating dynamic web content, interacting with databases, managing user sessions, and handling form submissions. They provide a powerful and flexible framework for building interactive web applications, offering platform independence, high performance, scalability, and seamless integration with other Java technologies.

**HTML:** HTML (Hypertext Markup Language) is the standard markup language for creating web pages and web applications. It provides the structure and layout for web content, defining elements such as headings, paragraphs, links, and forms.

**CSS (Cascading Style Sheets):** CSS is a stylesheet language used for styling HTML documents. It allows developers to control the appearance and layout of web pages, including aspects such as colors, fonts, margins, and positioning.

**Bootstrap:** Bootstrap is a popular front-end framework for building responsive and mobile-first web applications. It provides a set of pre-designed CSS and JavaScript components that streamline the process of creating modern and visually appealing user interfaces.

**MySQL:** MySQL is a widely-used open-source relational database management system (RDBMS). It offers robust features for managing structured data, including tables, indexes, stored procedures, and transactions, making it suitable for storing and retrieving data in web applications.

**MySQL Workbench:** MySQL Workbench is a visual database design and modeling tool that allows developers to design, visualize, and administer MySQL databases. It provides a user-friendly interface for creating and managing database schemas, tables, and relationships.

**Eclipse:** Eclipse is an integrated development environment (IDE) widely used for Java development. It offers a rich set of features, including code editing, debugging, and version control integration, making it a preferred choice for developing Java applications.

By leveraging these technologies, the Attendance Management achieves a balance of efficiency, scalability, and user-friendliness, enabling seamless management of Attendance Management operations while providing a satisfying experience for customers.

**System Architecture**

System architecture, particularly within the context of web applications, often incorporates the Model-View-Controller (MVC) architectural pattern. MVC provides a structured approach to organizing code and separating concerns within an application, facilitating modularity, scalability, and maintainability.

Here's a breakdown of the MVC architecture:

**Model (M):**

The Model represents the application's data and business logic. It encapsulates data access, manipulation, and validation operations.

In a typical MVC setup, models are responsible for querying and updating the database, processing business rules, and enforcing data integrity.

Models can be thought of as the "brains" of the application, managing the application's state and responding to requests from the controller or view.

**View (V):**

The View represents the presentation layer of the application. It is responsible for rendering user interfaces and presenting data to the user.

Views typically consist of HTML templates, CSS stylesheets, and client-side scripts (e.g., JavaScript) for dynamic interactions.

Views are passive components that receive data from the controller or model and present it to the user in a visually appealing and interactive format.

**Controller (C):**

The Controller acts as an intermediary between the Model and the View, handling user input, processing requests, and coordinating interactions between the Model and the View.

Controllers receive input from the user via the View, invoke appropriate actions on the Model based on the user's interactions, and update the View with the results.

Controllers encapsulate application logic related to request handling, routing, and business process orchestration, keeping the Model and View decoupled and independent.

Key characteristics of the MVC architecture:

**Separation of Concerns:** MVC separates the application's concerns into distinct components (Model, View, Controller), promoting code organization, reusability, and testability.

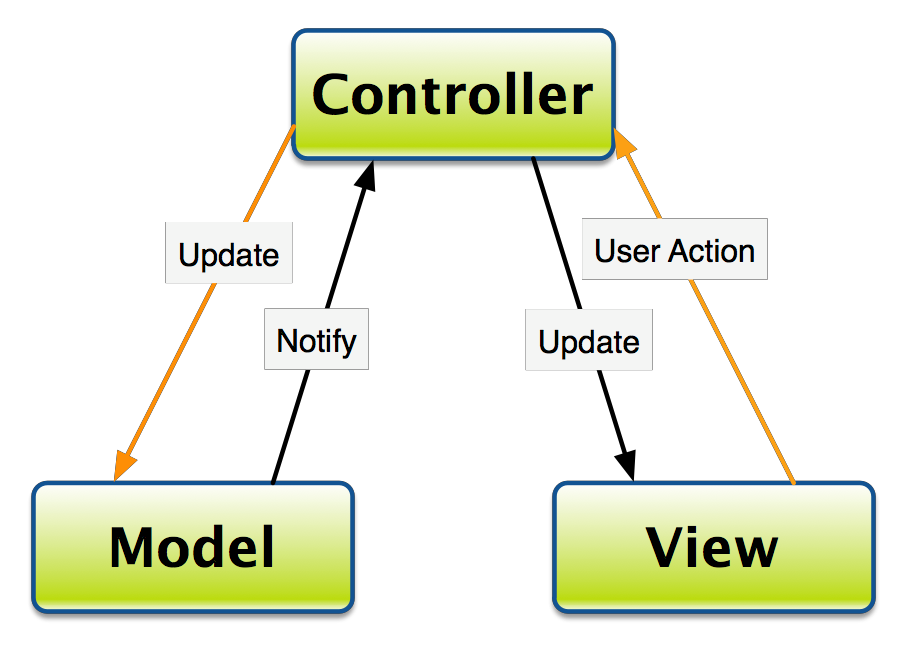
**Modularity:** MVC facilitates modularity by dividing the application into interchangeable components that can be developed, tested, and maintained independently.

**Scalability:** MVC supports scalability by allowing developers to add or modify components (e.g., controllers, views, models) without affecting other parts of the application, enabling the system to evolve over time.

**Maintainability:** MVC promotes maintainability by isolating changes within specific components, making it easier to identify, understand, and update code without impacting the overall system.

**Flexibility:** MVC provides flexibility in choosing technologies and frameworks for implementing each component, allowing developers to leverage the best tools for their specific requirements.

In summary, the MVC architecture provides a structured and flexible approach to designing web applications, emphasizing the separation of concerns and facilitating code organization, modularity, and maintainability. By dividing the application into three distinct layers (Model, View, Controller), MVC enables developers to build robust and scalable software solutions that meet the needs of users and businesses effectively.



**Development Process**

The development process of the Attendance Management was a systematic and collaborative effort aimed at transforming design specifications into a functional software solution. It involved several key stages, each contributing to the creation of a robust and user-friendly application.

Starting with requirements gathering and analysis, the team meticulously documented the needs and expectations of stakeholders to ensure that the system would meet user requirements. Database design played a crucial role in structuring data storage and management efficiently, laying the groundwork for the rest of the development process.

During the development phase, the team implemented the business logic, user interface components, and data access layer using Java, JSP and Servlet , HTML, CSS, Bootstrap, and other technologies. Coding standards and best practices were followed to ensure readability, maintainability, and scalability of the codebase.

Testing was an integral part of the development process, with both unit testing and integration testing being conducted to validate the functionality, performance, and reliability of the system. Manual testing played a vital role in identifying and addressing issues, ensuring a high-quality user experience.

Throughout the development lifecycle, collaboration, communication, and feedback were emphasized, enabling the team to adapt to changing requirements and deliver value incrementally. Continuous integration and deployment practices facilitated the automated build, test, and deployment process, streamlining the development workflow.

In conclusion, the development process of the Attendance Management exemplified the principles of modern software engineering, leveraging technology, collaboration, and best practices to deliver a robust and user-friendly solution. By embracing iterative development, testing, and continuous improvement, the team successfully translated design concepts into a functional and scalable application that meets the needs of stakeholders and end-users.

**Requirements gathering and analysis:**

Requirements gathering and analysis is a crucial phase in the software development lifecycle, as it sets the foundation for building a successful system that meets the needs of stakeholders and end-users. This phase involves gathering, documenting, analyzing, and prioritizing the requirements of the project. Here's a detailed overview of each step:

* Conduct interviews, workshops, or surveys with stakeholders to gather requirements.
* Document functional requirements, such as user roles, features, and workflows.
* Define non-functional requirements, including performance, security, and scalability criteria.
* Prioritize requirements and create a requirements specification document.
* Review and validate requirements with stakeholders to ensure alignment with business objectives.

**Identify Stakeholders:** Begin by identifying all stakeholders involved in the project. Stakeholders may include end-users, customers, project sponsors, business owners, subject matter experts, and other relevant parties. Each stakeholder may have unique perspectives, requirements, and expectations for the system.

**Conduct Stakeholder Interviews and Workshops:** Engage with stakeholders through interviews, workshops, surveys, or focus groups to gather insights into their needs, goals, and pain points. Use open-ended questions to encourage stakeholders to articulate their requirements and preferences. Record all feedback and insights gathered during these sessions.

**Elicit Requirements:** Use various techniques to elicit requirements from stakeholders, such as:

* **Requirements workshops:** Bring together stakeholders to collaborate on defining requirements.
* **Brainstorming sessions:** Encourage stakeholders to generate ideas and requirements collectively.
* **Prototyping:** Create mockups or prototypes to visualize and validate requirements.
* Use case analysis: Identify and document specific use cases to understand how users will interact with the system.
* **User stories:** Capture requirements from the perspective of end-users in the form of user stories.
* **Surveys and questionnaires:** Collect feedback from a larger audience to validate and prioritize requirements.

**Document Requirements:** Document all gathered requirements in a structured format, such as a Requirements Specification Document (RSD) or a Product Backlog. Include both functional requirements (what the system should do) and non-functional requirements (qualities or constraints of the system, such as performance, security, usability, etc.). Use clear, concise language and provide examples or use cases to illustrate each requirement.

**Analyze Requirements:** Analyze the gathered requirements to ensure they are complete, consistent, unambiguous, and feasible. Look for dependencies and conflicts between requirements and resolve them collaboratively with stakeholders. Prioritize requirements based on their importance to the project goals, business value, and constraints.

**Validate Requirements:** Validate the requirements with stakeholders to ensure they accurately reflect their needs and expectations. Use techniques such as reviews, walkthroughs, and prototypes to gather feedback and make necessary revisions. Continuous validation and refinement of requirements help minimize the risk of misunderstandings and scope creep.

**Manage Requirements Changes:** Establish a process for managing changes to requirements throughout the project lifecycle. Use a version control system to track changes and updates to the requirements documentation. Communicate changes effectively to all stakeholders and assess their impact on project scope, schedule, and resources.

**Obtain Stakeholder Sign-off:** Seek formal approval or sign-off from key stakeholders once the requirements are finalized and agreed upon. This indicates their commitment to the documented requirements and provides a baseline for subsequent phases of the project.

By following a systematic approach to requirements gathering and analysis, software development teams can ensure that the resulting system aligns with stakeholders' needs and expectations, leading to a successful outcome for the project. Effective communication, collaboration, and documentation are essential throughout this phase to capture and validate requirements accurately.

**Database design**

Database design is a critical aspect of software development, as it lays the foundation for storing, managing, and retrieving data efficiently and accurately within an application. Effective database design ensures data integrity, performance, scalability, and flexibility. Here's a detailed overview of the database design process:

**Requirements Analysis:**

Begin by understanding the requirements of the application and the data it needs to store and manipulate. Identify the entities (objects or concepts) within the domain of the application and the relationships between them.

Conduct interviews and discussions with stakeholders to gather insights into the data requirements, business rules, and constraints.

Analyze existing documents, such as business requirements documents, use cases, and user stories, to extract relevant data requirements.

**Conceptual Database Design:**

Create a conceptual data model that represents the high-level structure of the database, independent of any specific database management system (DBMS).

Use conceptual modeling techniques, such as Entity-Relationship Diagrams (ERDs) or Unified Modeling Language (UML) diagrams, to visualize the entities, attributes, and relationships in the domain.

Define entity types, their attributes, and the relationships between entities. This stage focuses on understanding the semantics of the data without considering implementation details.

**Logical Database Design:**

Translate the conceptual data model into a logical data model that can be implemented in a specific DBMS.

Choose an appropriate data model, such as relational, document-oriented, or graph-based, based on the requirements of the application and the characteristics of the data.

Design the tables (for relational databases) or collections (for NoSQL databases) to represent the entities and relationships identified in the conceptual model.

Define the primary keys, foreign keys, indexes, constraints, and data types for each table or collection.

Normalize the data model to eliminate redundancy and ensure data integrity. Use normalization techniques, such as First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF), to reduce data redundancy and dependency.

**Physical Database Design:**

Translate the logical data model into a physical schema that can be implemented in the chosen DBMS.

Decide on storage structures, such as tables, indexes, partitions, and tablespaces, to optimize data storage and access.

Define data partitioning and clustering strategies to improve performance and scalability.

Consider factors such as data volume, access patterns, concurrency, and security requirements when designing the physical schema.

Optimize the database design for performance by denormalizing tables, creating appropriate indexes, and partitioning data as needed.

**Data Integrity and Constraints:**

Enforce data integrity by defining constraints and rules that govern the validity of data in the database.

Define primary key constraints to ensure uniqueness of records, foreign key constraints to enforce referential integrity between tables, and other constraints such as NOT NULL, UNIQUE, CHECK, and DEFAULT constraints.

Implement business rules and validation logic within the database using triggers, stored procedures, or constraints to maintain data consistency and integrity.

**Data Security and Access Control:**

Implement security measures to protect sensitive data and ensure that only authorized users have access to the database.

Define user roles and privileges to restrict access to specific tables, views, or operations based on the principle of least privilege.

Implement encryption, authentication, and auditing mechanisms to safeguard data privacy and prevent unauthorized access or tampering.

**Database Maintenance and Optimization:**

Plan for ongoing database maintenance tasks, such as backup and recovery, database tuning, and performance monitoring.

Monitor database performance metrics, such as query execution times, resource utilization, and throughput, to identify bottlenecks and optimize performance.

Implement indexing strategies, query optimization techniques, and database caching to improve query performance and response times.

Regularly review and optimize the database schema, data storage structures, and indexing strategies to ensure optimal performance and scalability as the application evolves.

**Documentation and Communication:**

Document the database design, including the conceptual, logical, and physical models, as well as any design decisions, assumptions, and trade-offs made during the process.

Communicate the database design to stakeholders, developers, and other members of the project team to ensure a common understanding of the data model and its implications.

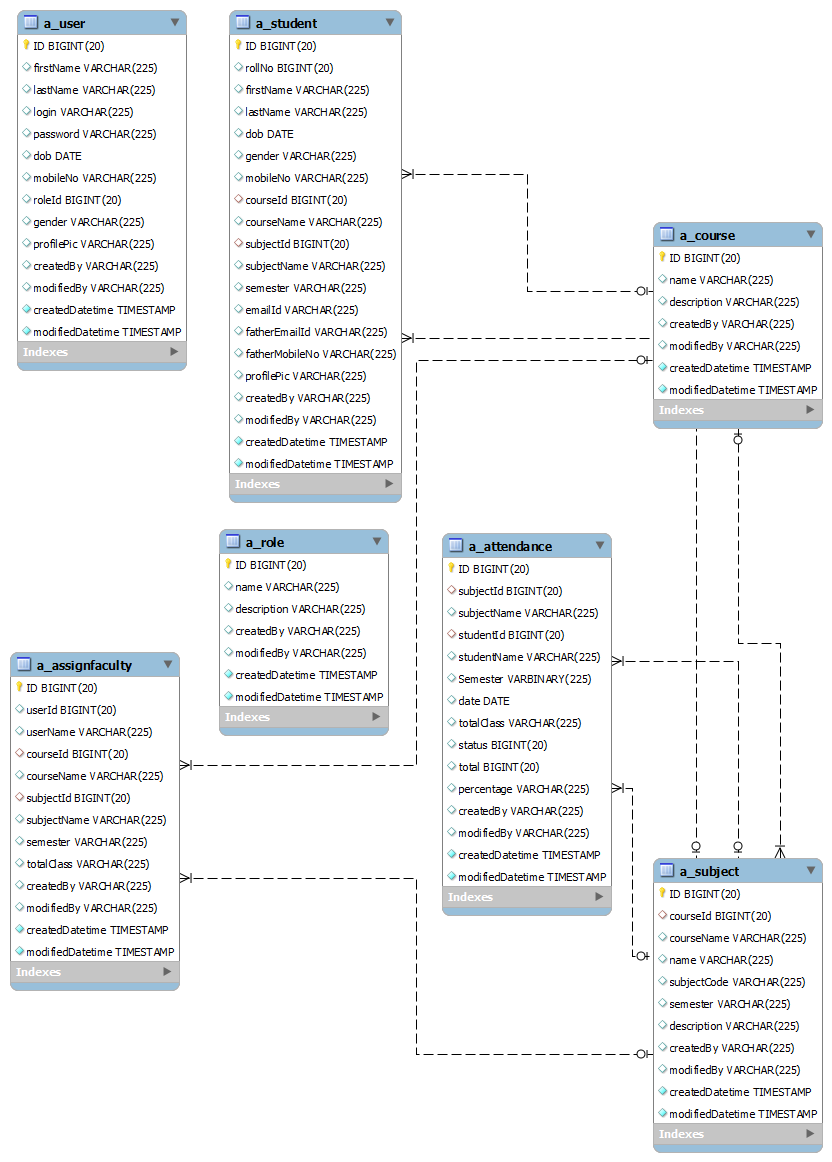
Provide documentation and training materials to support database administrators, developers, and users in understanding and working with the database effectively.

By following a structured approach to database design, software development teams can create well-designed databases that meet the requirements of the application, optimize performance, ensure data integrity, and support future growth and evolution. Effective database design is essential for building robust and scalable software systems that efficiently manage and leverage data to drive business value.

* Analyze the application's data requirements based on the gathered requirements.
* Identify entities, attributes, and relationships to model the data effectively.
* Create an Entity-Relationship Diagram (ERD) using tools like Lucidchart or draw.io.
* Normalize the database schema to eliminate redundancy and improve data integrity.
* Define database constraints, such as primary keys, foreign keys, and unique constraints.
* Consider indexing strategies to optimize query performance for frequently accessed data.

**Entity Relationship Diagram**

An Entity-Relationship Diagram (ERD) is a graphical representation used in database design to visualize the relationships between entities in a domain and the attributes associated with those entities. ERDs are an essential tool for understanding and communicating the structure of a database schema.

****

Here's a breakdown of key components and concepts of an ERD:

**Entities:**

* An entity represents a real-world object, concept, or thing with distinct properties.
* In an ERD, entities are typically represented as rectangles or boxes.
* Each entity has a name that describes the category of objects it.

**Attributes:**

* Attributes are the properties or characteristics of entities that describe them.
* Each attribute is associated with an entity and represents a specific piece of information about the entity.
* Attributes are represented as ovals or ellipses connected to their respective entities by lines.

**Relationships:**

* Relationships define associations and connections between entities.
* A relationship describes how entities are related to each other and can have various cardinality constraints, such as one-to-one, one-to-many, or many-to-many.
* Relationships are represented as lines connecting related entities, with optional symbols indicating cardinality and participation constraints.

**Cardinality:**

* Cardinality specifies the number of instances of one entity that are associated with the number of instances of another entity through a relationship.
* Cardinality constraints are often indicated using symbols such as "1" (one), "M" (many), "0" (zero), "N" (any number), or specific ranges (e.g., "1..\*" for one or more).
* Cardinality constraints help define the nature of the relationship between entities and influence database schema design and query formulation.

**Keys:**

* Keys are attributes or combinations of attributes that uniquely identify instances of an entity within a database.
* Primary keys are special attributes designated to uniquely identify each instance of an entity, and they play a crucial role in maintaining data integrity and enforcing constraints.
* Foreign keys are attributes that establish relationships between entities by referencing the primary key of another entity.

**Associative Entities:**

* Associative entities, also known as junction entities or relationship entities, represent entities that connect other entities in a many-to-many relationship.
* Associative entities typically contain attributes that describe the relationship between the connected entities.
* They are represented similarly to regular entities in an ERD but are connected by relationships indicating their association with other entities.

Overall, Entity-Relationship Diagrams provide a visual representation of the structure and relationships within a database schema, helping stakeholders, designers, and developers to understand, communicate, and design database systems effectively. ERDs serve as a blueprint for database implementation and play a crucial role in database design and modeling processes.

**System Design**

System design, also known as architectural design, is the process of defining the architecture, components, modules, interfaces, and data for a software system to meet specified requirements. It involves translating the requirements gathered during the analysis phase into a blueprint that guides the implementation of the system. Here's a detailed overview of the system design process:

* Architect the system using the Model-View-Controller (MVC) pattern to separate concerns and improve maintainability.
* Create a high-level architecture diagram illustrating the components and their interactions.
* Develop detailed component diagrams or class diagrams to specify the structure and behavior of each component.
* Define the data flow between components using Data Flow Diagrams (DFDs) or sequence diagrams.
* Choose appropriate design patterns and architectural styles to address specific requirements, such as dependency injection for managing dependencies and inversion of control for decoupling components.

**Understand Requirements:**

* Begin by thoroughly understanding the functional and non-functional requirements of the system.
* Analyze use cases, user stories, and other requirements documentation to identify the system's features, constraints, and goals.
* Clarify any ambiguities and resolve conflicting requirements through discussions with stakeholders.

**Define System Architecture:**

* Choose an appropriate architectural style or pattern based on the requirements, such as layered architecture, client-server architecture, microservices architecture, or event-driven architecture.
* Define the high-level structure of the system, including the major components, layers, subsystems, and their interactions.
* Determine how components will communicate with each other, including protocols, data formats, and APIs.

**Design Data Model:**

* Design the data model based on the requirements gathered during the analysis phase.
* Identify entities, attributes, relationships, and constraints using techniques such as Entity-Relationship Diagrams (ERDs) or UML class diagrams.
* Normalize the data model to eliminate redundancy and ensure data integrity, applying normalization techniques such as First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF).

**Define Component Interfaces:**

* Specify the interfaces between different components, modules, or subsystems of the system.
* Define input and output parameters, data formats, method signatures, and communication protocols for each interface.
* Ensure that interfaces are well-defined, clear, and consistent to facilitate integration and communication between components.

**Design Module Structure:**

* Decompose the system into smaller modules or units of functionality based on the system architecture and requirements.
* Identify cohesive and loosely coupled modules that encapsulate related functionality and minimize dependencies between modules.
* Define the responsibilities and interfaces of each module, specifying how they interact with other modules and the external environment.

**Specify System Behavior:**

* Define the behavior of the system in response to different inputs, events, and scenarios.
* Specify the business logic, algorithms, workflows, state transitions, and error handling mechanisms of the system.
* Use techniques such as flowcharts, state diagrams, activity diagrams, or sequence diagrams to illustrate system behavior.

**Consider Non-Functional Requirements:**

* Address non-functional requirements such as performance, scalability, reliability, security, and usability during system design.
* Design architectural strategies and mechanisms to meet these requirements, such as caching, load balancing, encryption, authentication, and user interface design principles.

**Evaluate Design Decisions:**

* Review and validate the system design against the requirements, constraints, and quality attributes.
* Conduct design reviews, walkthroughs, or architectural reviews with stakeholders and subject matter experts to solicit feedback and identify potential issues.
* Refine and iterate on the design based on feedback and lessons learned from the evaluation process.

**Document Design Artifacts:**

* Document the system design artifacts, including architectural diagrams, data models, interface specifications, module specifications, and behavioral descriptions.
* Create design documents, architectural blueprints, and technical specifications to communicate the system design to developers, testers, and other stakeholders.
* Ensure that design documentation is comprehensive, well-organized, and accessible to support implementation, testing, and maintenance activities.

**Iterate and Refine:**

* System design is an iterative process, and it may require multiple iterations to refine and optimize the design based on feedback, changing requirements, and emerging constraints.
* Continuously evaluate and refine the design as the project progresses, incorporating new insights, addressing issues, and adapting to evolving needs.

By following a systematic approach to system design, software development teams can create well-structured, modular, and scalable systems that meet the functional and non-functional requirements of the project. Effective system design lays the foundation for successful implementation, testing, and deployment of software systems, enabling the delivery of high-quality and reliable solutions to users and stakeholders.

**Data Flow Diagram**

A Data Flow Diagram (DFD) is a graphical representation that depicts the flow of data within a system, showing how data moves from one process to another and how it is stored, processed, and transformed along the way. DFDs are commonly used in software engineering and systems analysis to visualize the structure and behavior of information systems. Here's a detailed explanation of the key components and concepts of a DFD:

**Processes:**

* Processes represent activities or functions that manipulate data within the system.
* Each process performs a specific task or operation on the input data to produce output data.
* Processes are depicted as circles or rectangles in a DFD diagram, labeled with a descriptive name or identifier.
* Examples of processes include data transformation, computation, validation, storage, retrieval, and transmission.

**Data Flows:**

* Data flows represent the movement of data between processes, data stores, and external entities within the system.
* Data flows are depicted as arrows in a DFD diagram, indicating the direction of data flow.
* Each data flow is labeled with a meaningful name that describes the type or content of the data being transmitted.
* Data flows may represent inputs, outputs, or intermediate data exchanged between processes and external entities.

**Data Stores:**

* Data stores represent repositories or storage locations where data is persistently stored within the system.
* Data stores are depicted as rectangles with two parallel lines on one side in a DFD diagram.
* Each data store is labeled with a descriptive name that identifies the type or purpose of the stored data.
* Data stores may include databases, files, tables, queues, or any other storage medium used by the system.

**External Entities:**

* External entities represent sources or destinations of data that interact with the system but are external to it.
* External entities can be users, devices, systems, or other entities that exchange data with the system.
* External entities are depicted as squares or rectangles with rounded corners in a DFD diagram.
* Each external entity is labeled with a descriptive name that identifies its role or function in the system.

**Data Flow Paths:**

* Data flow paths represent the paths along which data flows through the system from its source to its destination.
* Data flow paths are traced by following the arrows that connect processes, data stores, and external entities in the DFD diagram.
* Data flow paths illustrate the sequence of operations performed on the data as it moves through the system, showing how data is processed and transformed.

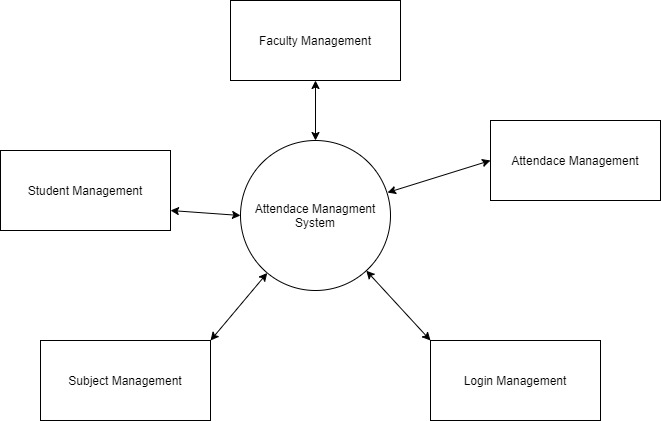
**Levels of Detail:**

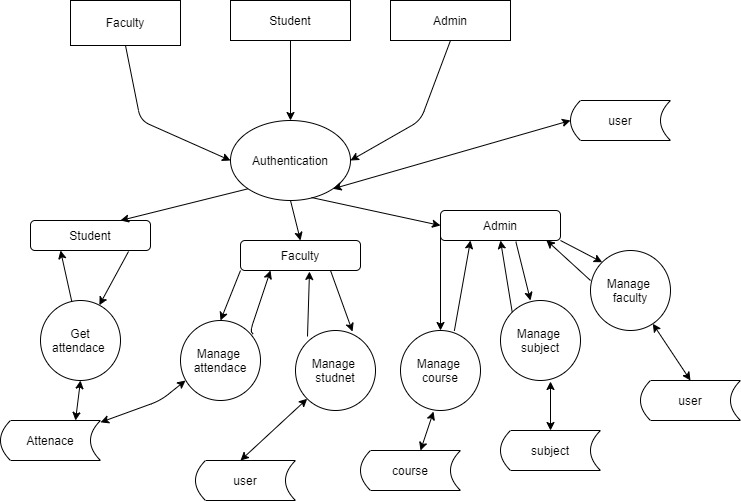
* DFDs can be organized into multiple levels of detail to represent different perspectives or views of the system.
* Level 0 DFD represents the highest level of abstraction, showing the overall structure and major processes of the system.
* Lower-level DFDs provide more detailed views of specific processes or subsystems, breaking down complex processes into smaller components.

**Context Diagram:**

* A context diagram is a high-level DFD that provides an overview of the system and its interactions with external entities.
* The context diagram shows the external entities that interact with the system and the data flows between them, without going into detail about internal processes or data stores.
* It serves as a starting point for understanding the scope and boundaries of the system and its external interfaces.

**DFD Diagrams**

****

****

Overall, Data Flow Diagrams provide a visual representation of how data moves through a system, helping stakeholders, designers, and developers to understand, analyze, and communicate the structure and behavior of information systems effectively. DFDs are valuable tools for requirements analysis, system design, and documentation in software engineering and systems analysis.

**Development**

The development phase of a software project is where the design specifications are translated into actual code, leading to the creation of the software product. This phase involves writing, testing, and debugging code, as well as integrating various components to build a functional system. Here's a detailed overview of the development process.

* Utilize Object-Relational Mapping (ORM) frameworks like Hibernate to simplify database interactions.
* Design the View layer using HTML, CSS, and frontend frameworks such as Bootstrap or Materialize for responsive and visually appealing user interfaces.
* Develop controller classes to handle HTTP requests, route them to appropriate actions, and interact with the Model layer.
* Use version control systems like Git for collaborative development and code management.
* Follow coding standards, conventions, and best practices to ensure consistency and maintainability.
* Implement the Model layer to represent the application's data model and business logic.

**Select Development Methodology:**

* Choose an appropriate development methodology based on the project's requirements, team size, and organizational culture.
* Common methodologies include Waterfall, Agile, Scrum, Kanban, and DevOps, each offering different approaches to managing the development process.

**Setup Development Environment:**

* Set up the development environment with the necessary tools, libraries, frameworks, and resources required for coding.
* Install and configure development tools such as Integrated Development Environments (IDEs), version control systems (e.g., Git), build automation tools (e.g., Maven, Gradle), and testing frameworks.

**Write Code:**

* Developers write code according to the specifications and design documents created during the previous phases.
* Use appropriate programming languages, frameworks, and best practices to implement the desired functionality.
* Follow coding standards, naming conventions, and design patterns to ensure consistency and maintainability of the codebase.

**Implement Business Logic:**

* Implement the business logic of the application, including algorithms, calculations, workflows, and decision-making processes.
* Write code to handle user interactions, process input data, and generate output responses according to the requirements.

**Develop User Interface (UI):**

* Design and develop the user interface (UI) components, including screens, forms, menus, buttons, and widgets.
* Use frontend technologies such as HTML, CSS, JavaScript, and frontend frameworks to create interactive and responsive user interfaces.

**Integrate Components:**

* Integrate different modules, components, and libraries to build a cohesive and functional system.
* Ensure that components communicate effectively with each other and adhere to the defined interfaces and protocols.

**Implement Data Access Layer:**

* Develop the data access layer to interact with the database or external data sources.
* Write code to perform CRUD (Create, Read, Update, Delete) operations, query data, and handle transactions.
* Use Object-Relational Mapping (ORM) frameworks (e.g., Hibernate, Entity Framework) or data access libraries to abstract database interactions and improve productivity.

**Write Unit Tests:**

* Write unit tests to validate the functionality of individual units or components of the software.
* Use testing frameworks (e.g., JUnit, NUnit) and mocking libraries to simulate dependencies and isolate units for testing.
* Write test cases to cover different scenarios, edge cases, and error conditions to ensure robustness and reliability of the code.

**Perform Integration Testing:**

* Conduct integration testing to verify that different components work together as expected.
* Test the interactions between modules, APIs, and external dependencies to identify and resolve integration issues.
* Use techniques such as black-box testing, white-box testing, and end-to-end testing to validate system behavior across various integration points.

**Debug and Refactor Code:**

* Debug code to identify and fix defects, errors, and anomalies in the software.
* Use debugging tools, logging frameworks, and diagnostic utilities to trace and troubleshoot issues.
* Refactor code to improve readability, performance, and maintainability while preserving the existing functionality.

**Optimize Performance:**

* Identify and address performance bottlenecks, inefficiencies, and resource constraints in the code.
* Use profiling tools to analyze code execution, memory usage, and I/O operations to identify areas for optimization.
* Optimize algorithms, data structures, and database queries to improve the overall performance and responsiveness of the system.

## Testing

Software testing is the process of evaluation a software item to detect differences between given input and expected output. Also to assess the feature of A software item. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words software testing is a verification and validation process.

###### Verification

Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way we want it to.

###### Validation

Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.

Testing goes side by side with the implementation that is aimed at ensuring that the system works accurately and efficiently before the live operation is performed .The common view of testing held by the user is process of executing a program with explicit intention of handling errors. The application which has been developed has to be tested to prove its validity. Testing is considered to be the least creative phase of the whole cycle of system design. In the real sense it is the phase, which helps to bring out the creativity of the other phases, and makes it shine.

The Smart Attendance Management was tested using the following two techniques of application testing:

###### Unit Testing:

* In the line of strategy the entire individuals function and modules were put to test independently
* By following this strategy all the errors in coding were identified and corrected.
* This method was applied in combination with the White Box and Black Box testing
* Technique to find errors in each module.
* The effort of specific combination of data on system operation wastested.
* The following were the testes carried out for Graphical User Interface(GUI).
* It was seen that the pages opens properly based on related menu based commands.
* It was tested whether all relevant menus, buttons, icons and other controls are available and properly displayed.

###### System Testing

We use this testing method. System testing is the testing to ensure that by putting the software in different environments (e.g., Operating Systems) it still works. System testing is done with full system implementation and environment. It falls under the class of black box testing.

###### Performance Testing

Performance testing is the testing to assess the speed and effectiveness of the system and to make sure it is generating results within a specified time as in performance requirements. It falls under the class of black box testing.

###### Multi-user System Testing

Database Locking Schemes: Whenever more than one person is accessing a record/s some type of process must be used to prevent the outer users from attempting to update the same record at the same time. This process is a locking scheme. In its simplest form, a locking scheme allows only one user at a time to update information in the database.

.

##### Conclusions

In conclusion, the project on "College Attendance Management" has successfully addressed the challenges faced by higher education institutions in monitoring and managing student attendance. Through meticulous research, analysis, and collaboration with stakeholders, we have developed a comprehensive digital attendance management system tailored specifically for college environments.

By leveraging innovative technologies and adopting a data-driven approach, the developed system streamlines the attendance tracking process, enhances accuracy, and provides valuable insights to educators and administrators. The system's features, such as biometric authentication, RFID technology, or mobile applications, offer colleges a flexible and efficient solution to meet their attendance management needs.

Throughout the project, collaboration with faculty, administrators, and IT professionals has been instrumental in ensuring that the developed solution aligns with the specific requirements and goals of higher education institutions. By incorporating feedback and insights from these key stakeholders, we have created a system that is user-friendly, scalable, and integrates seamlessly with existing academic systems.

The implementation of the digital attendance management system marks a significant step forward for colleges and universities, enhancing administrative efficiency, improving student accountability, and fostering a more conducive learning environment. By embracing digital innovation in attendance tracking, higher education institutions can stay at the forefront of academic administration and provide students with a seamless, transparent, and data-driven approach to managing their attendance.

As we look to the future, it is clear that the digital transformation of attendance management will continue to evolve, presenting new opportunities for innovation and improvement. By remaining adaptable, proactive, and responsive to the evolving needs of colleges and universities, we can ensure that the attendance management systems of tomorrow continue to meet the changing demands of higher education.

**Screen shot**

ADD SS after configuration

**Source code**

**Model**

package in.co.college.att.mgt.model;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.ArrayList;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import org.apache.log4j.Logger;

import in.co.college.att.mgt.bean.UserBean;

import in.co.college.att.mgt.exception.ApplicationException;

import in.co.college.att.mgt.exception.DatabaseException;

import in.co.college.att.mgt.exception.DuplicateRecordException;

import in.co.college.att.mgt.exception.RecordNotFoundException;

import in.co.college.att.mgt.util.EmailBuilder;

import in.co.college.att.mgt.util.EmailMessage;

import in.co.college.att.mgt.util.EmailUtility;

import in.co.college.att.mgt.util.JDBCDataSource;

public class UserModel {

private static Logger log = Logger.getLogger(UserModel.class);

public Integer nextPK() throws DatabaseException {

log.debug("Model nextPK Started");

Connection conn = null;

int pk = 0;

try {

conn = JDBCDataSource.getConnection();

PreparedStatement pstmt = conn.prepareStatement("SELECT MAX(ID) FROM A\_USER");

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

pk = rs.getInt(1);

}

rs.close();

} catch (Exception e) {

log.error("Database Exception..", e);

throw new DatabaseException("Exception : Exception in getting PK");

} finally {

JDBCDataSource.closeConnection(conn);

}

log.debug("Model nextPK End");

return pk + 1;

}

/\*\*

\* Add a User

\*

\* @param bean

\* @throws DatabaseException

\*

\*/

public long add(UserBean bean) throws ApplicationException, DuplicateRecordException {

Connection conn = null;

int pk = 0;

UserBean existbean = findByLogin(bean.getLogin());

if (existbean != null) {

throw new DuplicateRecordException("Login Id already exists");

}

try {

conn = JDBCDataSource.getConnection();

pk = nextPK();

// Get auto-generated next primary key

System.out.println(pk + " in ModelJDBC");

conn.setAutoCommit(false); // Begin transaction

PreparedStatement pstmt = conn.prepareStatement("INSERT INTO A\_USER VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?)");

pstmt.setInt(1, pk);

pstmt.setString(2, bean.getFirstName());

pstmt.setString(3, bean.getLastName());

pstmt.setString(4, bean.getLogin());

pstmt.setString(5, bean.getPassword());

pstmt.setDate(6, new java.sql.Date(bean.getDob().getTime()));

pstmt.setString(7, bean.getMobileNo());

pstmt.setLong(8, bean.getRoleId());

pstmt.setString(9, bean.getGender());

pstmt.setString(10, bean.getProfilePic());

pstmt.setString(11, bean.getCreatedBy());

pstmt.setString(12, bean.getModifiedBy());

pstmt.setTimestamp(13, bean.getCreatedDatetime());

pstmt.setTimestamp(14, bean.getModifiedDatetime());

pstmt.executeUpdate();

conn.commit(); // End transaction

pstmt.close();

} catch (Exception e) {

try {

conn.rollback();

} catch (Exception ex) {

ex.printStackTrace();

throw new ApplicationException("Exception : add rollback exception " + ex.getMessage());

}

throw new ApplicationException("Exception : Exception in add User");

} finally {

JDBCDataSource.closeConnection(conn);

}

return pk;

}

public void delete(UserBean bean) throws ApplicationException {

Connection conn = null;

try {

conn = JDBCDataSource.getConnection();

conn.setAutoCommit(false); // Begin transaction

PreparedStatement pstmt = conn.prepareStatement("DELETE FROM A\_USER WHERE ID=?");

pstmt.setLong(1, bean.getId());

pstmt.executeUpdate();

conn.commit(); // End transaction

pstmt.close();

} catch (Exception e) {

try {

conn.rollback();

} catch (Exception ex) {

throw new ApplicationException("Exception : Delete rollback exception " + ex.getMessage());

}

throw new ApplicationException("Exception : Exception in delete User");

} finally {

JDBCDataSource.closeConnection(conn);

}

}

public UserBean findByLogin(String login) throws ApplicationException {

log.debug("Model findByLogin Started");

StringBuffer sql = new StringBuffer("SELECT \* FROM A\_USER WHERE LOGIN=?");

UserBean bean = null;

Connection conn = null;

System.out.println("sql" + sql);

try {

conn = JDBCDataSource.getConnection();

PreparedStatement pstmt = conn.prepareStatement(sql.toString());

pstmt.setString(1, login);

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

bean = new UserBean();

bean.setId(rs.getLong(1));

bean.setFirstName(rs.getString(2));

bean.setLastName(rs.getString(3));

bean.setLogin(rs.getString(4));

bean.setPassword(rs.getString(5));

bean.setDob(rs.getDate(6));

bean.setMobileNo(rs.getString(7));

bean.setRoleId(rs.getLong(8));

bean.setGender(rs.getString(9));

bean.setProfilePic(rs.getString(10));

bean.setCreatedBy(rs.getString(11));

bean.setModifiedBy(rs.getString(12));

bean.setCreatedDatetime(rs.getTimestamp(13));

bean.setModifiedDatetime(rs.getTimestamp(14));

}

rs.close();

} catch (Exception e) {

e.printStackTrace();

log.error("Database Exception..", e);

throw new ApplicationException("Exception : Exception in getting User by login");

} finally {

JDBCDataSource.closeConnection(conn);

}

log.debug("Model findByLogin End");

return bean;

}

public UserBean findByPK(long pk) throws ApplicationException {

log.debug("Model findByPK Started");

StringBuffer sql = new StringBuffer("SELECT \* FROM A\_USER WHERE ID=?");

UserBean bean = null;

Connection conn = null;

try {

conn = JDBCDataSource.getConnection();

PreparedStatement pstmt = conn.prepareStatement(sql.toString());

pstmt.setLong(1, pk);

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

bean = new UserBean();

bean.setId(rs.getLong(1));

bean.setFirstName(rs.getString(2));

bean.setLastName(rs.getString(3));

bean.setLogin(rs.getString(4));

bean.setPassword(rs.getString(5));

bean.setDob(rs.getDate(6));

bean.setMobileNo(rs.getString(7));

bean.setRoleId(rs.getLong(8));

bean.setGender(rs.getString(9));

bean.setProfilePic(rs.getString(10));

bean.setCreatedBy(rs.getString(11));

bean.setModifiedBy(rs.getString(12));

bean.setCreatedDatetime(rs.getTimestamp(13));

bean.setModifiedDatetime(rs.getTimestamp(14));

}

rs.close();

} catch (Exception e) {

e.printStackTrace();

log.error("Database Exception..", e);

throw new ApplicationException("Exception : Exception in getting User by pk");

} finally {

JDBCDataSource.closeConnection(conn);

}

log.debug("Model findByPK End");

return bean;

}

public void update(UserBean bean) throws ApplicationException, DuplicateRecordException {

log.debug("Model update Started");

Connection conn = null;

UserBean beanExist = findByLogin(bean.getLogin());

// Check if updated LoginId already exist

if (beanExist != null && !(beanExist.getId() == bean.getId())) {

throw new DuplicateRecordException("LoginId is already exist");

}

try {

conn = JDBCDataSource.getConnection();

conn.setAutoCommit(false); // Begin transaction

PreparedStatement pstmt = conn.prepareStatement(

"UPDATE A\_USER SET FIRSTNAME=?,LASTNAME=?,LOGIN=?,PASSWORD=?,DOB=?,MOBILENO=?,ROLEID=?,"

+ "GENDER=?,ProfilePic=?,"

+ "CREATEDBY=?,MODIFIEDBY=?,CREATEDDATETIME=?,MODIFIEDDATETIME=? WHERE ID=?");

pstmt.setString(1, bean.getFirstName());

pstmt.setString(2, bean.getLastName());

pstmt.setString(3, bean.getLogin());

pstmt.setString(4, bean.getPassword());

pstmt.setDate(5, new java.sql.Date(bean.getDob().getTime()) );

pstmt.setString(6, bean.getMobileNo());

pstmt.setLong(7, bean.getRoleId());

pstmt.setString(8, bean.getGender());

pstmt.setString(9,bean.getProfilePic());

pstmt.setString(10, bean.getCreatedBy());

pstmt.setString(11, bean.getModifiedBy());

pstmt.setTimestamp(12, bean.getCreatedDatetime());

pstmt.setTimestamp(13, bean.getModifiedDatetime());

pstmt.setLong(14, bean.getId());

pstmt.executeUpdate();

conn.commit(); // End transaction

pstmt.close();

} catch (Exception e) {

e.printStackTrace();

log.error("Database Exception..", e);

try {

conn.rollback();

} catch (Exception ex) {

throw new ApplicationException("Exception : Delete rollback exception " + ex.getMessage());

}

throw new ApplicationException("Exception in updating User ");

} finally {

JDBCDataSource.closeConnection(conn);

}

log.debug("Model update End");

}

public List search(UserBean bean) throws ApplicationException {

return search(bean, 0, 0);

}

public List search(UserBean bean, int pageNo, int pageSize) throws ApplicationException {

log.debug("Model search Started");

StringBuffer sql = new StringBuffer("SELECT \* FROM A\_USER WHERE 1=1");

if (bean != null) {

if (bean.getId() > 0) {

sql.append(" AND id = " + bean.getId());

}

if (bean.getFirstName() != null && bean.getFirstName().length() > 0) {

sql.append(" AND FIRSTNAME like '" + bean.getFirstName() + "%'");

}

if (bean.getLastName() != null && bean.getLastName().length() > 0) {

sql.append(" AND LASTNAME like '" + bean.getLastName() + "%'");

}

if (bean.getLogin() != null && bean.getLogin().length() > 0) {

sql.append(" AND LOGIN like '" + bean.getLogin() + "%'");

}

if (bean.getPassword() != null && bean.getPassword().length() > 0) {

sql.append(" AND PASSWORD like '" + bean.getPassword() + "%'");

}

if (bean.getDob() != null && bean.getDob().getDate() > 0) {

sql.append(" AND DOB = " + bean.getGender());

}

if (bean.getMobileNo() != null && bean.getMobileNo().length() > 0) {

sql.append(" AND MOBILENO = " + bean.getMobileNo());

}

if (bean.getRoleId() > 0) {

sql.append(" AND ROLEID = " + bean.getRoleId());

}

}

// if page size is greater than zero then apply pagination

if (pageSize > 0) {

// Calculate start record index

pageNo = (pageNo - 1) \* pageSize;

sql.append(" Limit " + pageNo + ", " + pageSize);

// sql.append(" limit " + pageNo + "," + pageSize);

}

System.out.println("user model search :"+sql);

ArrayList list = new ArrayList();

Connection conn = null;

try {

conn = JDBCDataSource.getConnection();

PreparedStatement pstmt = conn.prepareStatement(sql.toString());

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

bean = new UserBean();

bean.setId(rs.getLong(1));

bean.setFirstName(rs.getString(2));

bean.setLastName(rs.getString(3));

bean.setLogin(rs.getString(4));

bean.setPassword(rs.getString(5));

bean.setDob(rs.getDate(6));

bean.setMobileNo(rs.getString(7));

bean.setRoleId(rs.getLong(8));

bean.setGender(rs.getString(9));

bean.setProfilePic(rs.getString(10));

bean.setCreatedBy(rs.getString(11));

bean.setModifiedBy(rs.getString(12));

bean.setCreatedDatetime(rs.getTimestamp(13));

bean.setModifiedDatetime(rs.getTimestamp(14));

list.add(bean);

}

rs.close();

} catch (Exception e) {

log.error("Database Exception..", e);

throw new ApplicationException("Exception : Exception in search user");

} finally {

JDBCDataSource.closeConnection(conn);

}

log.debug("Model search End");

return list;

}

public List list() throws ApplicationException {

return list(0, 0);

}

public List list(int pageNo, int pageSize) throws ApplicationException {

log.debug("Model list Started");

ArrayList list = new ArrayList();

StringBuffer sql = new StringBuffer("select \* from A\_USER");

// if page size is greater than zero then apply pagination

if (pageSize > 0) {

// Calculate start record index

pageNo = (pageNo - 1) \* pageSize;

sql.append(" limit " + pageNo + "," + pageSize);

}

System.out.println("sql in list user :"+sql);

Connection conn = null;

try {

conn = JDBCDataSource.getConnection();

PreparedStatement pstmt = conn.prepareStatement(sql.toString());

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

UserBean bean = new UserBean();

bean.setId(rs.getLong(1));

bean.setFirstName(rs.getString(2));

bean.setLastName(rs.getString(3));

bean.setLogin(rs.getString(4));

bean.setPassword(rs.getString(5));

bean.setDob(rs.getDate(6));

bean.setMobileNo(rs.getString(7));

bean.setRoleId(rs.getLong(8));

bean.setGender(rs.getString(9));

bean.setProfilePic(rs.getString(10));

bean.setCreatedBy(rs.getString(11));

bean.setModifiedBy(rs.getString(12));

bean.setCreatedDatetime(rs.getTimestamp(13));

bean.setModifiedDatetime(rs.getTimestamp(14));

list.add(bean);

}

rs.close();

} catch (Exception e) {

log.error("Database Exception..", e);

throw new ApplicationException("Exception : Exception in getting list of users");

} finally {

JDBCDataSource.closeConnection(conn);

}

log.debug("Model list End");

return list;

}

public UserBean authenticate(String login, String password) throws ApplicationException {

log.debug("Model authenticate Started");

StringBuffer sql = new StringBuffer("SELECT \* FROM A\_USER WHERE LOGIN = ? AND PASSWORD = ?");

UserBean bean = null;

Connection conn = null;

try {

conn = JDBCDataSource.getConnection();

PreparedStatement pstmt = conn.prepareStatement(sql.toString());

pstmt.setString(1, login);

pstmt.setString(2, password);

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

bean = new UserBean();

bean.setId(rs.getLong(1));

bean.setFirstName(rs.getString(2));

bean.setLastName(rs.getString(3));

bean.setLogin(rs.getString(4));

bean.setPassword(rs.getString(5));

bean.setDob(rs.getDate(6));

bean.setMobileNo(rs.getString(7));

bean.setRoleId(rs.getLong(8));

bean.setGender(rs.getString(9));

bean.setProfilePic(rs.getString(10));

bean.setCreatedBy(rs.getString(11));

bean.setModifiedBy(rs.getString(12));

bean.setCreatedDatetime(rs.getTimestamp(13));

bean.setModifiedDatetime(rs.getTimestamp(14));

}

} catch (Exception e) {

log.error("Database Exception..", e);

throw new ApplicationException("Exception : Exception in get roles");

} finally {

JDBCDataSource.closeConnection(conn);

}

log.debug("Model authenticate End");

return bean;

}

public List getRoles(UserBean bean) throws ApplicationException {

log.debug("Model get roles Started");

StringBuffer sql = new StringBuffer("SELECT \* FROM A\_USER WHERE role\_Id=?");

Connection conn = null;

List list = new ArrayList();

try {

conn = JDBCDataSource.getConnection();

PreparedStatement pstmt = conn.prepareStatement(sql.toString());

pstmt.setLong(1, bean.getRoleId());

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

bean = new UserBean();

bean.setId(rs.getLong(1));

bean.setFirstName(rs.getString(2));

bean.setLastName(rs.getString(3));

bean.setLogin(rs.getString(4));

bean.setPassword(rs.getString(5));

bean.setDob(rs.getDate(6));

bean.setMobileNo(rs.getString(7));

bean.setRoleId(rs.getLong(8));

bean.setGender(rs.getString(9));

bean.setProfilePic(rs.getString(10));

bean.setCreatedBy(rs.getString(11));

bean.setModifiedBy(rs.getString(12));

bean.setCreatedDatetime(rs.getTimestamp(13));

bean.setModifiedDatetime(rs.getTimestamp(14));

list.add(bean);

}

rs.close();

} catch (Exception e) {

log.error("Database Exception..", e);

throw new ApplicationException("Exception : Exception in get roles");

} finally {

JDBCDataSource.closeConnection(conn);

}

log.debug("Model get roles End");

return list;

}

public boolean changePassword(Long id, String oldPassword, String newPassword)

throws RecordNotFoundException, ApplicationException {

log.debug("model changePassword Started");

boolean flag = false;

UserBean beanExist = null;

beanExist = findByPK(id);

if (beanExist != null && beanExist.getPassword().equals(oldPassword)) {

beanExist.setPassword(newPassword);

try {

update(beanExist);

} catch (DuplicateRecordException e) {

log.error(e);

throw new ApplicationException("LoginId is already exist");

}

flag = true;

} else {

throw new RecordNotFoundException("Old password is Invalid");

}

HashMap<String, String> map = new HashMap<String, String>();

map.put("login", beanExist.getLogin());

map.put("password", beanExist.getPassword());

map.put("firstName", beanExist.getFirstName());

map.put("lastName", beanExist.getLastName());

String message = EmailBuilder.getChangePasswordMessage(map);

EmailMessage msg = new EmailMessage();

msg.setTo(beanExist.getLogin());

msg.setSubject("SUNARYS ORS Password has been changed Successfully.");

msg.setMessage(message);

msg.setMessageType(EmailMessage.HTML\_MSG);

try {

EmailUtility.sendMail(msg);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

log.debug("Model changePassword End");

return flag;

}

public UserBean updateAccess(UserBean bean) throws ApplicationException {

return null;

}

public long registerUser(UserBean bean)

throws ApplicationException, DuplicateRecordException {

log.debug("Model add Started");

long pk = add(bean);

HashMap<String, String> map = new HashMap<String, String>();

map.put("login", bean.getLogin());

map.put("password", bean.getPassword());

String message = EmailBuilder.getUserRegistrationMessage(map);

EmailMessage msg = new EmailMessage();

msg.setTo(bean.getLogin());

msg.setSubject("Registration is successful for ORS Project SunilOS");

msg.setMessage(message);

msg.setMessageType(EmailMessage.HTML\_MSG);

try {

EmailUtility.sendMail(msg);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return pk;

}

public boolean forgetPassword(String login)

throws ApplicationException, RecordNotFoundException, ApplicationException {

UserBean userData = findByLogin(login);

boolean flag = false;

if (userData == null) {

throw new RecordNotFoundException("Email ID does not exists !");

}

HashMap<String, String> map = new HashMap<String, String>();

map.put("login", userData.getLogin());

map.put("password", userData.getPassword());

map.put("firstName", userData.getFirstName());

map.put("lastName", userData.getLastName());

String message = EmailBuilder.getForgetPasswordMessage(map);

EmailMessage msg = new EmailMessage();

msg.setTo(login);

msg.setSubject("SUNARYS ORS Password reset");

msg.setMessage(message);

msg.setMessageType(EmailMessage.HTML\_MSG);

EmailUtility.sendMail(msg);

flag = true;

return flag;

}

}

Controller

package in.co.college.att.mgt.controller;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import org.apache.log4j.Logger;

import in.co.college.att.mgt.bean.BaseBean;

import in.co.college.att.mgt.bean.RoleBean;

import in.co.college.att.mgt.bean.UserBean;

import in.co.college.att.mgt.exception.ApplicationException;

import in.co.college.att.mgt.model.RoleModel;

import in.co.college.att.mgt.model.UserModel;

import in.co.college.att.mgt.util.DataUtility;

import in.co.college.att.mgt.util.DataValidator;

import in.co.college.att.mgt.util.PropertyReader;

import in.co.college.att.mgt.util.ServletUtility;

@WebServlet(name = "LoginCtl", urlPatterns = { "/login" })

public class LoginCtl extends BaseCtl {

private static final long serialVersionUID = 1L;

public static final String OP\_REGISTER = "Register";

public static final String OP\_SIGN\_IN = "SignIn";

public static final String OP\_SIGN\_UP = "SignUp";

public static final String OP\_LOG\_OUT = "logout";

public static String HIT\_URI = null;

private static Logger log = Logger.getLogger(LoginCtl.class);

public LoginCtl() {

super();

// TODO Auto-generated constructor stub

}

@Override

protected boolean validate(HttpServletRequest request) {

log.debug("LoginCtl Method validate Started");

boolean pass = true;

String op = request.getParameter("operation");

if (OP\_SIGN\_UP.equals(op) || OP\_LOG\_OUT.equals(op)) {

return pass;

}

String login = request.getParameter("login");

if (DataValidator.isNull(login)) {

request.setAttribute("login", PropertyReader.getValue("error.require", "Login Id"));

pass = false;

} else if (!DataValidator.isEmail(login)) {

request.setAttribute("login", PropertyReader.getValue("error.email", "Login Id "));

pass = false;

}

if (DataValidator.isNull(request.getParameter("password"))) {

request.setAttribute("password", PropertyReader.getValue("error.require", "Password"));

pass = false;

}

log.debug("LoginCtl Method validate Ended");

return pass;

}

/\*\*

\* Populates bean object from request parameters

\*

\* @param request

\* @return

\*/

@Override

protected BaseBean populateBean(HttpServletRequest request) {

log.debug("LoginCtl Method populateBean Started");

UserBean bean = new UserBean();

bean.setId(DataUtility.getLong(request.getParameter("id")));

bean.setLogin(DataUtility.getString(request.getParameter("login")));

bean.setPassword(DataUtility.getString(request.getParameter("password")));

log.debug("LOginCtl Method PopulatedBean End");

return bean;

}

/\*\*

\* Display Login form

\*

\*/

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse

\* response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

log.debug("Method doGet Started");

HttpSession session = request.getSession(true);

String op = DataUtility.getString(request.getParameter("operation"));

UserModel model = new UserModel();

RoleModel role = new RoleModel();

long id = DataUtility.getLong(request.getParameter("id"));

if (id > 0) {

UserBean userBean;

try {

userBean = model.findByPK(id);

ServletUtility.setBean(userBean, request);

} catch (Exception e) {

log.error(e);

ServletUtility.handleException(e, request, response);

return;

}

} else if (OP\_LOG\_OUT.equals(op)) {

session = request.getSession(false);

session.invalidate();

ServletUtility.setSuccessMessage("You have been logged out successfully", request);

ServletUtility.forward(CASView.LOGIN\_VIEW, request, response);

return;

}

if (session.getAttribute("user") != null) {

ServletUtility.redirect(CASView.WELCOME\_CTL, request, response);

return;

}

ServletUtility.forward(getView(), request, response);

log.debug("Method doGet end");

}

/\*\*

\* Submit Logic

\*/

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse

\* response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

log.debug(" LoginCtl Method doPost Started");

HttpSession session = request.getSession(true);

String op = DataUtility.getString(request.getParameter("operation"));

// get Model

UserModel model = new UserModel();

RoleModel role = new RoleModel();

long id = DataUtility.getLong(request.getParameter("id"));

if (OP\_SIGN\_IN.equalsIgnoreCase(op)) {

UserBean bean = (UserBean) populateBean(request);

try {

bean = model.authenticate(bean.getLogin(), bean.getPassword());

if (bean != null) {

session.setAttribute("user", bean);

session.setMaxInactiveInterval(10 \* 6000);

long rollId = bean.getRoleId();

RoleBean roleBean = role.findByPK(rollId);

if (roleBean != null) {

session.setAttribute("role", roleBean.getName());

}

// save state of session remember URL

String uri = request.getParameter("uri");

if (uri == null || "null".equalsIgnoreCase(uri)) {

ServletUtility.redirect(CASView.WELCOME\_CTL, request, response);

return;

} else {

ServletUtility.redirect(uri, request, response);

}

return;

} else {

bean = (UserBean) populateBean(request);

ServletUtility.setBean(bean, request);

ServletUtility.setErrorMessage("Invalid LoginId And Password", request);

}

} catch (ApplicationException e) {

log.error(e);

ServletUtility.handleException(e, request, response);

return;

}

} else if (OP\_SIGN\_UP.equalsIgnoreCase(op)) {

ServletUtility.redirect(CASView.USER\_REGISTRATION\_CTL, request, response);

return;

}

ServletUtility.forward(getView(), request, response);

log.debug("UserCtl Method doPost Ended");

}

/\*\*

\* Returns the VIEW page of this Controller

\*

\* @return

\*/

@Override

protected String getView() {

return CASView.LOGIN\_VIEW;

}

}

Beans

package in.co.college.att.mgt.bean;

import java.sql.Timestamp;

import java.util.Date;

public class UserBean extends BaseBean {

private String firstName;

private String lastName;

private String login;

private String password;

private String confirmPassword;

private Date dob;

private String mobileNo;

private long roleId;

private String gender;

private String profilePic;

public String getProfilePic() {

return profilePic;

}

public void setProfilePic(String profilePic) {

this.profilePic = profilePic;

}

public String getFirstName() {

return firstName;

}

/\*\*

\* @param FirstName

\* To set User FirstName

\*/

public void setFirstName(String firstName) {

this.firstName = firstName;

}

/\*\*

\* @return LastName Of User

\*/

public String getLastName() {

return lastName;

}

/\*\*

\* @param LastName

\* To set User LastName

\*/

public void setLastName(String lastName) {

this.lastName = lastName;

}

/\*\*

\* @return Login id Of User

\*/

public String getLogin() {

return login;

}

/\*\*

\* @param Login

\* Id To set User Login ID

\*/

public void setLogin(String login) {

this.login = login;

}

/\*\*

\* @return Password Of User

\*/

public String getPassword() {

return password;

}

/\*\*

\* @param Password

\* To set User Password

\*/

public void setPassword(String password) {

this.password = password;

}

/\*\*

\* @return Confirm Password Of User

\*/

public String getConfirmPassword() {

return confirmPassword;

}

/\*\*

\* @param Confirm

\* PAssword To set User Confirm Password

\*/

public void setConfirmPassword(String confirmPassword) {

this.confirmPassword = confirmPassword;

}

/\*\*

\* @return Date Of Birth Of User

\*/

public Date getDob() {

return dob;

}

/\*\*

\* @param Date

\* Of Birth To set User Date Of Birth

\*/

public void setDob(Date dob) {

this.dob = dob;

}

/\*\*

\* @return Mobile No Of User

\*/

public String getMobileNo() {

return mobileNo;

}

/\*\*

\* @param Mobile

\* No To set User Mobile No

\*/

public void setMobileNo(String mobileNo) {

this.mobileNo = mobileNo;

}

/\*\*

\* @return ROle Id Of User

\*/

public long getRoleId() {

return roleId;

}

/\*\*

\* @param Role

\* Id To set User ROle Id

\*/

public void setRoleId(long roleId) {

this.roleId = roleId;

}

/\*\*

\* @return unSuccessfulLogin Of User

\*/

public String getGender() {

return gender;

}

/\*\*

\* @param Gender

\* To set User Gender

\*/

public void setGender(String gender) {

this.gender = gender;

}

public String getKey() {

return id + "";

}

public String getValue() {

return firstName + " " + lastName;

}

}

# CONCLUSIONCONCLUSION

Overall, the system is useful for all the users to maintain information at various levels. It connects supervisor and employee and thus easy to maintain.

Now supervisor can easily set the schedule or any notifications to the respective employees without having a person to send to employees.

It has been a great pleasure for me to work on this exciting and challenging project. This project proved good for us as it provided practical knowledge of not only programming in J2EE and Oracle Server Developer working of web based application, but also about all handling procedure related with Advance and new technology. It also provides knowledge about the latest technology used in developing web enabled application and client server technology that will be great demand in future. This will provide better opportunities and guidance in future in developing projects independently.