A MINI PROJECT REPORT ON

**CHESS | THE GAME OF INDIA**

BY

**SAURABH (24763)**

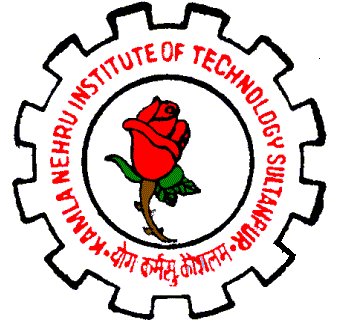
SUBMITTED TO

**PROFESSOR GARIMA YADAV**

**PROFESSOR ATUL LAL SRIVASTAVA**

A PROJECT REPORT SUBMITTED ON PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF DEGREE

**MASTER OF COMPUTER APPLICATIONS (MCA)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**KAMLA NEHRU INSTITUTE OF TECHNOLOGY**

**SULTANPUR (U.P) - 228118**

(An Autonomous Government Engineering Institute)

*AFFILIATED TO*

**DR A. P. J. ABDUL KALAM TECHNICAL UNIVERSITY LUCKNOW (U.P.) INDIA**

**2025-26**

# CERTIFICATE

**This is to certify that Saurabh (Roll No: 24763) has successfully carried out the project work entitled “Chess | The Game of India” for the award of the Master of Computer Applications (MCA) degree at Kamla Nehru Institute of Technology, Sultanpur, affiliated to Dr. A. P. J. Abdul Kalam Technical University, Lucknow.**

This report represents the candidate’s own work completed under our supervision and guidance. The project forms a part of the Master of Computer Applications curriculum. His performance during the project work was excellent, and we wish him continued success in all future endeavours.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Professor Garima Yadav**  **Prof. Arvind Kumar Tiwari Professor Atul Lal Srivastava**

(Professor, CSED, KNIT Sultanpur) (HOD, CSED, KNIT Sultanpur) (Professor, CSED, KNIT Sultanpur)

# DECLARATION

I hereby declare that this project work entitled **“Chess | The Game of India”** is the record of my own work carried out by me under the supervision and guidance of the faculty of the Department of Computer Applications, Kamla Nehru Institute of Technology, Sultanpur, affiliated to Dr. A. P. J. Abdul Kalam Technical University, Lucknow.

To the best of my knowledge and belief, this work contains no material previously published or written by another person, nor material which has been accepted for the award of any other degree or diploma of the university or any other institution, except where due acknowledgement has been made in the text.

**Name:** Saurabh  
**Roll:** 24763  
**Branch:** MCA, FINAL YEAR, KNIT SULTANPUR  
**Date:** 15.11.2025

# ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Prof. Garima Yadav, Department of Computer Applications, Kamla Nehru Institute of Technology, Sultanpur, for her invaluable guidance, constant encouragement, and insightful suggestions throughout the completion of this project titled “Chess | The Game of India.” Her continuous support and motivation have been instrumental in helping me overcome challenges and successfully carry out this work and for their constructive feedback, valuable insights, and encouragement during the course of this project.

Lastly, I wish to acknowledge all the faculty members and staff of the Department of Computer Applications for their continuous support and motivation. Their encouragement and guidance have played a vital role in the successful completion of this project.

# ABSTRACT

In today’s digital era, interactive applications play an important role in learning, entertainment, and problem-solving. This project, titled **“Chess | The Game of India”**, is a browser-based application developed to provide a smooth and engaging platform for playing chess. The system allows two players to interact with the board, move pieces, and experience classic chess mechanics through an intuitive and responsive interface. The game logic differentiates between White and Black turns, enabling structured gameplay and ensuring that movement rules and piece interactions are followed accurately.

Built using **HTML, CSS, and JavaScript**, the application focuses on delivering a clean, modern user experience with smooth transitions, responsive design, and visually appealing piece rendering. Local processing is used to handle all game logic, ensuring fast performance without requiring any backend system. The project emphasizes simplicity, usability, and clarity, making it suitable for learners and enthusiasts who want to understand board logic, DOM manipulation, and front-end game development. This work demonstrates the integration of well-structured logic, aesthetic UI design, and essential web development practices to produce a lightweight yet functional chess application.

Table of Contents

[CERTIFICATE 1](#_Toc213670892)

[DECLARATION 2](#_Toc213670893)

[ACKNOWLEDGEMENT 3](#_Toc213670894)

[ABSTRACT 4](#_Toc213670895)

[CHAPTER 1: Introduction 7](#_Toc213670896)

[1.1. Overview Of The Project 8](#_Toc213670897)

[1.2 Motivation 9](#_Toc213670898)

[1.3 Objectives 10](#_Toc213670899)

[1.4 Features 11](#_Toc213670900)

[CHAPTER 2: Literature Survey 13](#_Toc213670901)

[2.1 BACKGROUND INFORMATION 13](#_Toc213670902)

[2.2 COMPARISON BETWEEN EXISTING SYSTEMS 15](#_Toc213670903)

[CHAPTER 3: System Analysis 16](#_Toc213670904)

[3.1 Existing Model 16](#_Toc213670905)

[3.2 Proposed Model 16](#_Toc213670906)

[CHAPTER 4: Design & Implementation 18](#_Toc213670907)

[4.1 System Requirements 18](#_Toc213670908)

[4.2 Procedure 19](#_Toc213670909)

[4.3 Project Modules 21](#_Toc213670910)

[4.4 Data Flow Diagram 23](#_Toc213670911)

[4.5 Sequence Diagram 25](#_Toc213670912)

[4.6 Use Case Diagram 28](#_Toc213670913)

[4.7 UI Interfaces 31](#_Toc213670914)

[CHAPTER 5: Technologies Used 34](#_Toc213670915)

[5.1 JavaScript 34](#_Toc213670916)

[5.2 React.js (Frontend Logic & State Management) 34](#_Toc213670917)

[5.3 Tailwind CSS 35](#_Toc213670918)

[5.4 Visual Studio Code 35](#_Toc213670919)

[5.5 Git Version Control 36](#_Toc213670920)

[CHAPTER 6: Testing & Validation 37](#_Toc213670921)

[6.1 Component and Functional Testing 37](#_Toc213670922)

[6.2 Browser Compatibility Testing 37](#_Toc213670923)

[6.3 Operating System Compatibility Testing 37](#_Toc213670924)

[6.4 NPM Package Integrity Validation 37](#_Toc213670925)

[6.5 Network and Performance Testing 37](#_Toc213670926)

[6.6 Mobile Responsiveness Testing 37](#_Toc213670927)

[CHAPTER 7: Conclusion 38](#_Toc213670928)

[CHAPTER 8: Future Scope 39](#_Toc213670929)

[CHAPTER 9: PERT/CPM Chart 40](#_Toc213670930)

[CHAPTER 10: References 41](#_Toc213670931)

**Table of Figures**

[Figure 1.1. Employee Management System Logo 8](#_Toc213671215)

[Figure 2.1 Comparison Between Existing Technologies 16](#_Toc213671216)

[Figure 4.1 DFD Employee Management System 24](#_Toc213671217)

[Figure 4.2 Sequence Diagram Employee Management System 26](#_Toc213671218)

[Figure 4.3 Sequence Diagram Employee Management System 29](#_Toc213671219)

[Figure 4.4 Login Page 32](#_Toc213671220)

[Figure 4.5 Admin Dashboard - Create User/Admin Page 33](#_Toc213671221)

[Figure 4.6 Assign Task Page 33](#_Toc213671222)

[Figure 4.7 Task Overview Page 34](#_Toc213671223)

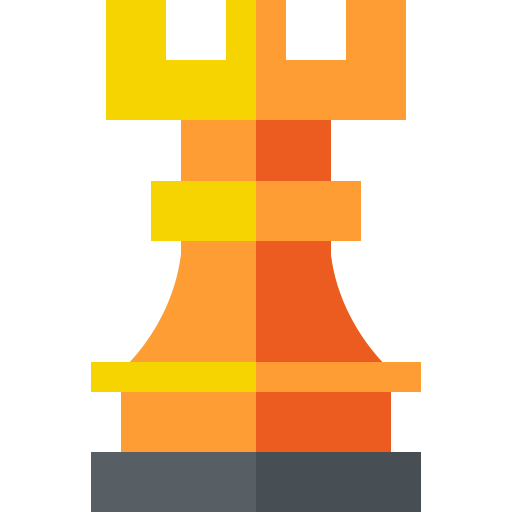
[Figure 4.8 User Dashboard 34](#_Toc213671224)

[Figure 9.1 Program Evaluation and Review Technique & Critical Path Method 41](#_Toc213671225)

# CHAPTER 1: Introduction

In today’s fast-paced and technology-driven world, interactive digital applications play a key role in enhancing user engagement, learning, and entertainment. Traditional ways of playing chess, such as physical boards or basic offline software, often lack modern visual appeal, ease of access, and smooth gameplay interactions. To overcome these limitations, **“Chess | The Game of India”** has been developed as a browser-based solution that provides players with a clean, responsive, and intuitive platform to experience classic chess with improved clarity and convenience.

This system is designed to simulate the fundamental mechanics of chess by offering a structured two-player turn-based environment where each move is validated and executed according to standard chess rules. The game highlights valid movements, manages turns, and ensures an immersive and error-free playing experience. The project integrates **modern front-end technologies** to deliver a lightweight, user-friendly, and visually aesthetic interface. Its focus on smooth gameplay, clear interactions, and minimalistic design ensures an engaging user experience while demonstrating strong fundamentals of web-based game development.



### Figure 1.1. Chess | The Game of India Logo

## 1.1. Overview Of The Project

**“Chess | The Game of India”** is a fully interactive web-based application that brings the traditional game of chess into a clean, modern, and accessible digital environment. The application enables two players to engage in strategic gameplay, move pieces according to standard chess rules, and experience smooth turn-based interaction directly within the browser. Each piece’s movement is validated, squares are highlighted for clarity, and the game flow is maintained in real-time to ensure a seamless playing experience.

The project provides an intuitive and visually appealing interface built using **HTML, CSS, and JavaScript**, featuring smooth transitions, responsive design, and clearly rendered piece icons. All gameplay logic is processed locally in the browser, ensuring fast interaction without requiring backend support or external dependencies. The primary goal of this project is to deliver an engaging, accurate, and user-friendly digital chess experience while showcasing fundamental concepts of front-end development, DOM manipulation, and interactive UI design.

## 1.2 Motivation

Every individual interested in strategy, mental discipline, or traditional games often faces limitations when relying on physical boards or outdated digital versions of chess. Issues such as lack of mobility, poor interfaces, and inconsistent rule handling can disrupt the overall experience. Recognizing this gap, the motivation behind developing **“Chess | The Game of India”** was to create a lightweight yet powerful digital platform that brings the classic game to users in a clean, modern, and accessible form.

This project aims to provide players with an intuitive and reliable environment to practice, enjoy, and understand chess more effectively. By highlighting valid moves, managing turns automatically, and offering a smooth interactive board, the system reduces confusion and enhances clarity. Ultimately, it promotes strategic thinking, improves user engagement, and makes the timeless game of chess more enjoyable and convenient in a digital-first world.

## 1.3 Objectives

The **“Chess | The Game of India”** project has been developed with the primary goal of providing a clean, interactive, and engaging digital environment for playing chess. The system is designed to simplify the traditional gameplay experience, ensure clarity in piece movement, and enhance user interaction by replacing physical boards or outdated interfaces with a smooth and intuitive web-based platform. This project emphasizes accuracy, usability, and user engagement through well-structured chess logic and modern front-end design.

The detailed objectives of this project are as follows:

**1. To develop a clean and accessible web-based platform for playing chess:**

The system aims to provide a simple, unified interface where users can play chess directly in their browser. The platform handles all chess elements—piece rendering, movement validation, turn switching, and board updates—ensuring a seamless gameplay experience.

**2. To implement a structured and rule-based movement system:**

The game enforces standard chess rules for every piece.  
Each move is validated according to the piece’s movement logic, and turn-based flow is maintained between White and Black.  
This structured mechanism ensures fairness, accuracy, and a realistic chess experience.

**3. To automate move highlighting and interaction handling:**

Whenever a player selects a piece, its valid moves are automatically highlighted on the board.  
This reduces confusion, improves clarity, and helps players understand possible moves instantly.  
Captures, promotions, and special moves (such as castling) are handled visually and logically within the system.

**4. To enhance transparency and user interaction:**

The interface clearly shows whose turn it is, which piece is selected, and which moves are allowed.  
This transparency improves user understanding and reduces the chances of illegal or accidental moves.  
The visual design and highlights contribute to a smooth and engaging playing experience.

## 1.4 Features

The **“Chess | The Game of India”** project is designed to provide a smooth, structured, and interactive digital environment for playing chess. The system replicates the complete mechanics of traditional chess and ensures seamless interaction between players through an intuitive interface, efficient move handling, and visually refined gameplay.  
It integrates modern frontend technologies to deliver a responsive and aesthetic experience. Key features of the system are categorized below.

**1. Turn-Based Gameplay Control**

The application implements a strict turn-based system, ensuring alternate chances for White and Black players.

* **White’s Turn:** White begins the game and moves first.
* **Black’s Turn:** Automatically activated after White completes a valid move.

This structured turn flow maintains fairness, rule accuracy, and proper sequencing throughout the match.

**2. Interactive Board Rendering**

The chessboard is visually represented through a grid-based structure where each square is clickable and responsive.

* Squares automatically highlight valid moves when a piece is selected.
* Illegal moves are prevented through built-in logic.
* Captures, promotions, and special moves are handled visually and instantly.

This interactivity eliminates confusion and helps players visualize possible actions clearly.

**3. Complete Piece Movement Logic**

Each chess piece follows its traditional rules:

* Pawns move forward and capture diagonally, with promotion on the last rank.
* Rooks, Knights, Bishops, Queen, and King follow their natural movement patterns.
* The system validates every move before execution, ensuring full rule compliance.

This guarantees authentic gameplay consistent with official chess mechanics.

**4. Real-Time Move Updates**

As soon as a player selects and moves a piece:

* The board updates dynamically.
* Highlight colors change instantly.
* Turn indicator switches automatically.

This real-time feedback improves responsiveness and enhances the overall playing experience.

**5. Minimal, Responsive, and Aesthetic User Interface**

Designed using **HTML, CSS, and JavaScript**, the interface provides:

* Smooth transitions and hover animations
* Modern dark-theme styling
* Clean board layout with rounded edges
* Scalable visuals for mobile and desktop screens

The UI ensures clarity, accessibility, and consistent gameplay across devices.

**6. Logic-Based State Handling Without Backend Dependency**

All game logic—piece positions, movement validation, turn management—is processed directly in the browser using JavaScript.

* No server or database is required
* Instant responses with zero delay
* Fully offline functionality

This makes the game lightweight, fast, and easy to run on any system.

**7. Move Validation and Error Prevention**

The system ensures that:

* Only legal moves are permitted
* Invalid or blocked moves are ignored
* Each selection is visually highlighted
* The King cannot be moved illegally into blocked squares (basic safety)

These safeguards maintain accuracy and prevent gameplay mistakes.

**8. Special Move Handling (Castling & Promotion)**

The system supports essential advanced chess mechanics:

* **Castling:** Automatically detects if castling is allowed and highlights eligible squares.
* **Promotion:** Automatically upgrades a pawn to a queen upon reaching the last rank.

These additions make the digital game closer to real-life chess rules.

**9. Player-Friendly Interaction Feedback**

The system enhances usability through:

* Color-coded highlights (selected piece, possible moves, attack squares)
* Smooth tile hover animations
* High-clarity piece icons
* Clear text indicator for current turn

This improves user engagement and reduces confusion.

# CHAPTER 2: Literature Survey

The literature survey explores the evolution of digital chess systems, from traditional over-the-board gameplay to modern computer-based, web-based, and AI-powered chess platforms. Over the years, advancements in computing, visualization technologies, and artificial intelligence have significantly transformed how chess is played, analyzed, and experienced.  
By tracing historical developments, this survey highlights the transition from manual gameplay to automated, interactive systems that support move validation, gameplay visualization, and strategic analysis. A comparative study of existing platforms reveals variations in functionality, usability, and computational intelligence, offering insights into areas that can be enhanced in modern digital chess applications.

## 2.1 BACKGROUND INFORMATION

The evolution of digital chess systems has been shaped by technological progress, the rise of personal computing, and increasing interest in computer-assisted strategy games.  
The following stages outline this evolution, supported by research and historical developments:

**2.1.1 Traditional Board-Based Chess**

In its earliest form, chess was played exclusively on physical boards with manually tracked moves. Gameplay depended entirely on human memory and rule knowledge. While this fostered face-to-face interaction and strategic depth, it lacked automated rule enforcement, move recording, and replay analysis.

**Reference:** Murray, H.J.R. (1913). *A History of Chess*. Oxford University Press.

**1.2 Computer-Based Chess Programs (Early Digital Era)**

With the introduction of personal computers in the 1980s–1990s, early chess software began to emerge. These applications executed legal moves, detected illegal actions, and allowed users to play against simple rule-based engines. However, graphical interfaces were basic, and computational strength was limited.

**Reference:** Levy, D., & Newborn, M. (1991). *How Computers Play Chess*. Computer Science Press.

**1.3 Standalone Chess Engines and Desktop Applications**

During the late 1990s and early 2000s, sophisticated engines such as Fritz, Rybka, and Crafty revolutionized digital chess. These engines introduced powerful evaluation functions, opening books, and advanced search algorithms. However, they were platform-dependent, lacked mobility, and required manual installation and configuration.

**Reference:** Shannon, C. E. (1950). *Programming a Computer for Playing Chess*. Philosophical Magazine, 41(314), 256–275.

**1.4 Web-Based Chess Platforms**

The rise of the internet enabled online chess platforms such as Chess.com and Lichess. These web-based systems allowed multiplayer games, real-time interaction, move highlighting, and browser-based analysis tools. They eliminated platform dependency and significantly increased global accessibility.

**Reference:** Richards, D. (2006). *Internet-Based Chess: A New Era for the Game*. Journal of Online Gaming, 4(2), 45–52.

**1.5 Cloud-Integrated Chess Systems**

Modern cloud-driven platforms introduced scalability, cloud storage of games, synchronized accounts, and access across multiple devices. Cloud-based engines support deeper analysis, seamless matchmaking, and integration with learning modules, enhancing user engagement and reducing computational load on local devices.

**Reference:** Gupta, A., & Singh, R. (2019). *Cloud Technologies in Modern Chess Platforms*. International Journal of Computer Applications, 178(32).

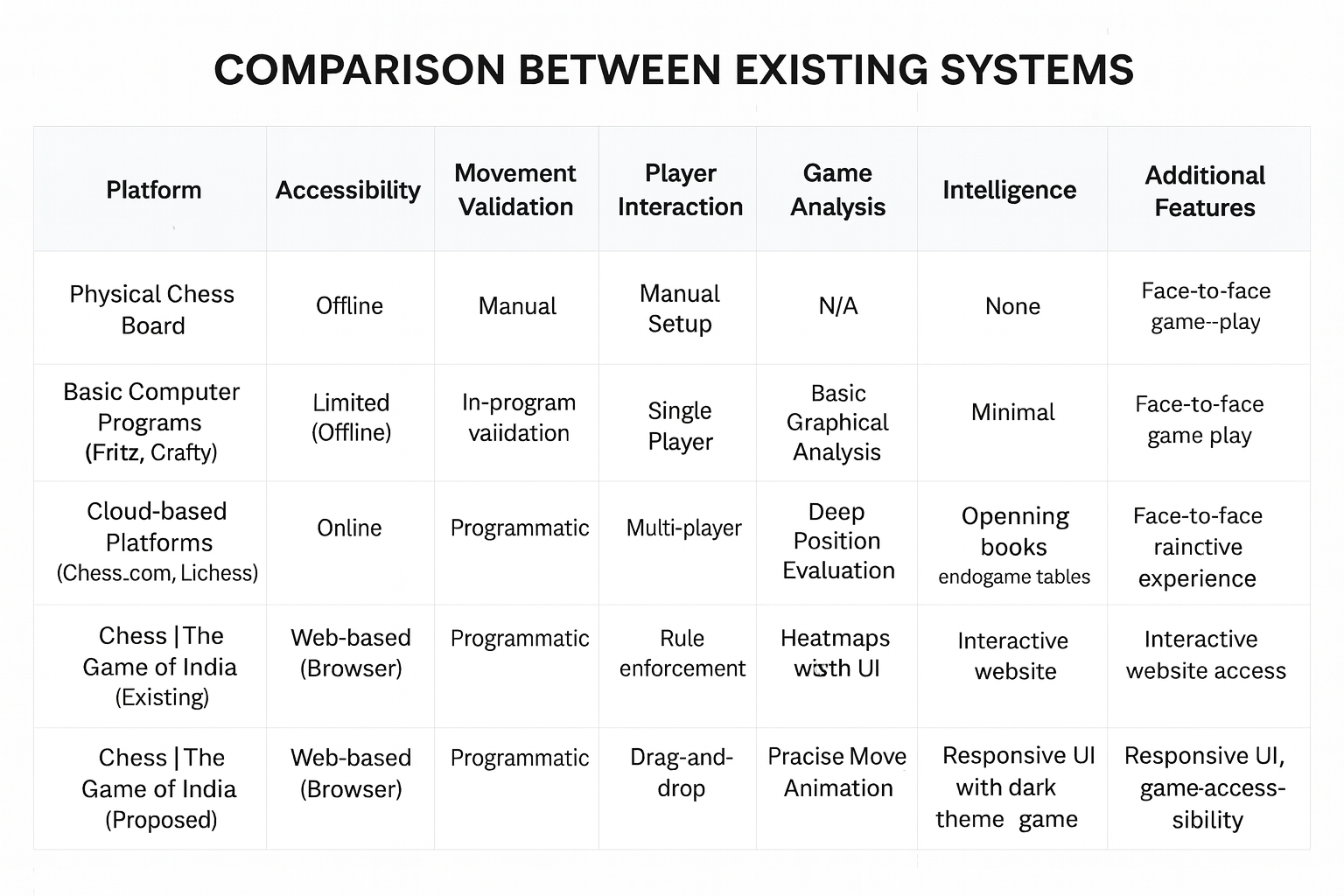
**1.6 Automation, Game Analysis, and Visualization Tools**

Recent developments emphasize automated move validation, PGN handling, real-time evaluation graphs, and interactive user interfaces. Features like heatmaps, arrows, and board highlights improve gameplay clarity and learning efficiency.

**Reference:** Silver, D. (2017). *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. DeepMind Research Report.

## 2.2 COMPARISON BETWEEN EXISTING SYSTEMS

**“Chess | The Game of India”** is a web-based platform designed to provide an interactive and streamlined digital chess experience. It ensures accurate move handling, turn management, and real-time board interaction through a clean, responsive interface built with HTML, CSS, and JavaScript, offering modern aesthetics and smooth usability for players.



### Figure 2.1 Comparison Between Existing Technologies

# CHAPTER 3: System Analysis

## 3.1 Existing Model

The current experience of playing chess varies widely depending on the platform, ranging from traditional over-the-board setups to advanced AI-powered engines. However, these systems each come with their own limitations in terms of accessibility, interactivity, usability, and learning support.

**Physical Chess Boards:**

Traditional chess boards require manual piece movement and human oversight for rule enforcement. This approach lacks automated move validation, digital highlights, or assistance — making it slow for beginners and completely dependent on player expertise. Additionally, no move history or digital visualization is available.

**Basic Digital Chess Applications:**

Early computer-based chess programs (offline software) offer basic board rendering and movement but typically lack modern UI, animations, mobile responsiveness, or interactive move guidance. These applications often feel outdated and are not easily accessible across devices.

**Legacy Desktop Chess Engines:**

Older engines like Fritz or Crafty focus mainly on computational strength rather than user experience. While they offer strong analysis, they require installation, are not web-based, and do not provide modern visuals or smooth interaction — making them less engaging for casual players.

**Cloud-Based Chess Platforms:**

Modern platforms like Chess.com or Lichess provide online multiplayer features, powerful engines, and deep analysis tools. However, these systems can be overwhelming for beginners, require user accounts, and often include advanced features not needed for simple gameplay. They also rely heavily on servers, internet availability, and complex backend systems.

**3.2 Proposed Model**

**“Chess | The Game of India”** aims to provide a clean, lightweight, and interactive browser-based chess application focused on smooth gameplay, clarity, and accessibility. It removes unnecessary complexity and delivers the essential chess experience in a visually appealing and responsive way.

**Interactive Turn-Based System:**

The game follows a structured turn-based system where White and Black move alternately. This ensures rule consistency, clarity, and fairness throughout the game.

**Move Validation and Tracking:**

The system validates each move based on traditional chess rules.  
Key features include:

* Highlighting valid moves
* Preventing illegal movement
* Allowing captures and pawn promotion
* Supporting castling logic  
  This improves accuracy, usability, and player understanding.

**Modern Minimal UI:**

The interface is developed using **HTML, CSS, and JavaScript**, focusing on:

* Smooth hover animations
* Modern dark-mode styling
* Clean tile highlights
* Mobile-responsive board design  
  The minimal aesthetic and fluid feedback make the game visually engaging.

**Local Logic & No Backend Dependency:**

All gameplay logic — board state, movement rules, selection, and turn updates — is handled directly in the browser.  
This ensures:

* Zero installation
* Offline functionality
* Instant performance
* Easy testing and deployment

**Scalability and Future Enhancements:**

The codebase is structured to support future upgrades, such as:

* Saving and loading games
* Online multiplayer using WebSockets or Firebase
* Move history and PGN export
* Built-in AI opponent using Minimax or NNUE models

This architecture ensures long-term maintainability and extendibility.

# CHAPTER 4: Design & Implementation

## 4.1 System Requirements

***Hardware Requirements***

**a**. Modern multi-core processor (Intel Core i3 or higher) with sufficient processing power to handle multiple user interactions and smooth dashboard rendering.

**b.** Minimum 8 GB of RAM recommended for efficient performance during task management and UI transitions.

**c.** Minimum 256 GB of storage space to store project files, user data, and application cache.

**d.** Stable internet connection with at least 10 Mbps download and upload speeds recommended for seamless navigation and data synchronization.

**e.** High-resolution display (minimum 1280×720) with adequate screen size for clear viewing of admin and user dashboards.

**f.** Keyboard and mouse or touchpad for interacting with the application, entering data, and navigating between modules.

**g.** Support for various input/output ports (USB, HDMI, DisplayPort, etc.) for connecting peripherals and external devices if required.

***Software Requirements***

**a.** Operating System compatible with Windows 10/11, macOS, or Linux.

**b.** Latest versions of popular web browsers such as Google Chrome, Mozilla Firefox, Microsoft Edge, or Safari.

**c.** Node.js runtime environment (version 14.x or later) for managing dependencies and running development servers.

**d.** Local Storage or Firebase database for storing user credentials, task details, and progress data.

**e.** Any modern text editor or IDE (such as Visual Studio Code) for writing, editing, and maintaining the project codebase.

**f.** Version control system (like Git and GitHub) for tracking code changes and collaborating with team members.

**g.** Extensions for Visual Studio Code (such as Prettier, Tailwind IntelliSense, and ESLint) to enhance productivity and streamline frontend development.

**h.** If integrating authentication features in the future, a secure user authentication system (like Firebase Auth or JWT) can be used for login and role verification.

## 4.2 Procedure

***1. Defined Project Structure***

● Begin by organizing the project folder structure to ensure clarity, maintainability, and smooth development.

● Create distinct directories for assets (chess piece images), scripts (JavaScript logic), and styles (CSS files).

● Maintain a consistent naming convention for all files such as app.js, style.css, images/, and index.html to ensure readability and scalability.

***2. Frontend Development***

● The frontend is developed using HTML, CSS, and JavaScript to create an interactive and lightweight chess interface.

● Design and implement a fully responsive chessboard using 8×8 grid-based layout and clickable squares for movement.

● Use modern CSS styling for a minimal, aesthetic, and visually engaging user interface.

● Implement smooth transitions and hover effects to enhance the gameplay experience.

● Render chess pieces using PNG images and dynamically update them based on moves and captures.

● Highlight valid moves, captures, and special actions using real-time color-coded square feedback.

● Ensure a fully responsive layout so the game adapts well to desktops, tablets, and mobile screens.

***3. Game Logic (JavaScript-Based Simulation)***

● Although the project has no backend, logical computations are executed entirely in JavaScript.

● The board state, piece positions, and moves are simulated using arrays and DOM manipulation.

● Implement core chess rules:

– Piece movements (rook, bishop, knight, queen, king, pawn)

– Pawn promotion

– Captures

– Turn switching

– Castling logic

● Functions handle piece movement, validation, square highlighting, and state updating.

● This simulation ensures the game behaves like a standard chess system without requiring a server or database.

***4. Integration and State Management***

● Establish smooth interaction between board squares and game logic using event listeners.

● Ensure that every move updates the board state immediately and reflects visually on the screen.

● Maintain turn-based flow using simple game state variables to alternate between White and Black.

● Manage UI consistency by re-rendering pieces and colors after every move to maintain gameplay clarity.

● Prepare the architecture for future scalability (e.g., adding AI opponent or online multiplayer).

## 4.3 Project Modules

Our project, **“Chess | The Game of India”**, consists of multiple interrelated modules that collectively create a complete, interactive chess-playing environment. Each module is designed to deliver a smooth, intuitive, and realistic gameplay experience while maintaining accurate move validation, turn management, and user interaction. The modules include:

**1. Board Rendering and Setup Module**

● This module initializes the chessboard layout using an 8×8 grid structure.  
● All pieces (King, Queen, Rook, Bishop, Knight, Pawn) are placed in their standard starting positions.  
● Each square is represented with a unique ID to support move calculation.  
● Chess piece images are dynamically rendered using JavaScript for a clean visual setup.

**2. Piece Selection and Highlighting Module**

● When the user clicks a piece, this module highlights all legal moves based on its movement rules.  
● The system prevents selecting the opponent’s piece during the wrong turn.  
● Valid squares are visually differentiated using color-coded highlights (e.g., green for moves, pink for selected piece).  
● This module ensures players fully understand their possible moves and interactions.

**3. Movement Validation Module**

● This module enforces the rules of chess for each piece (pawn advances, knight jumps, diagonal bishop moves, etc.).  
● Legal moves are calculated using positional math based on the coordinate-like ID system.  
● Illegal or blocked moves are restricted automatically.  
● Captures and special conditions (promotion, castling) are validated before execution.

**4. Turn Management Module**

● Maintains alternating turns between White and Black.  
● Updates the visual turn indicator (“White’s Turn” or “Black’s Turn”) after each valid move.  
● Ensures that a player cannot move during the opponent’s turn.  
● Prevents simultaneous moves and maintains game order.

**5. Special Moves Module**

● Handles advanced chess mechanics including:  
– **Pawn Promotion:** Automatic conversion to Queen upon reaching final rank.  
– **Castling:** Validates castling conditions and executes rook–king rearrangements.  
– **Basic Capture Logic:** Manages piece removal when landing on an opponent’s square.  
● Ensures these moves follow standard chess rules and updates the board accordingly.

**6. Real-Time Board Update Module**

● After every move, the board is re-rendered to reflect new positions and remove captured pieces.  
● Colors are reset and refreshed for the next move to maintain clarity.  
● Piece images reload dynamically to ensure visual accuracy.  
● Continuous DOM updates ensure smooth and responsive gameplay.

**7. Game Progress and End Condition Module**

● Tracks vital pieces (especially kings) to detect win conditions.  
● Ends the game when one king is captured and displays a win notification.  
● Resets the board upon game completion to allow replay.  
● Ensures accurate detection of game termination.

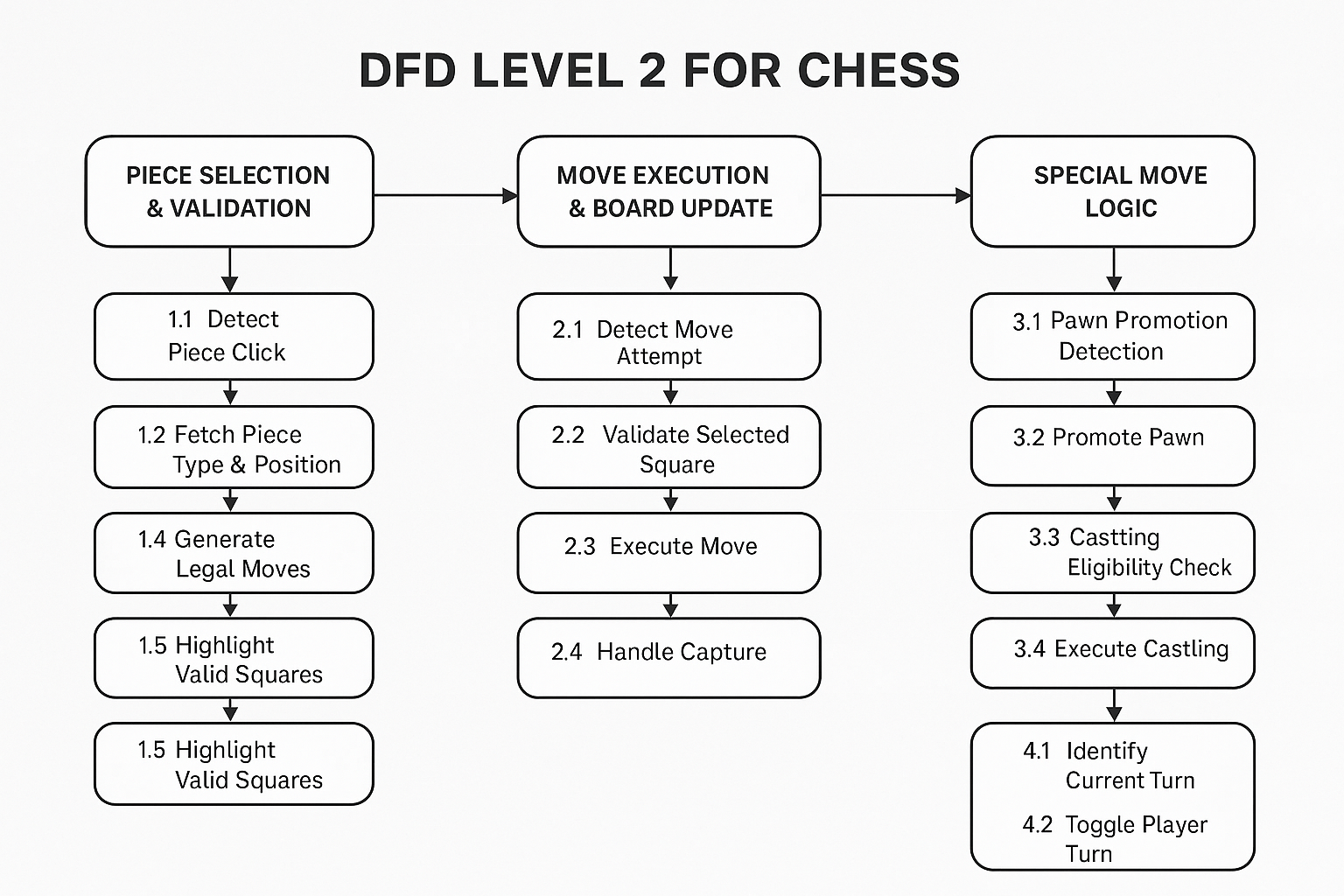
**8. User Interface and Experience Module**

● Implements smooth transitions, hover animations, and modern styling using CSS.  
● Ensures responsive board scaling for desktops, tablets, and mobile devices.  
● Maintains consistent visual feedback during piece selection, movement, and capture.  
● Offers a minimal aesthetic to ensure focus on gameplay rather than visual clutter.

**9. Reset and Session Handling Module**

● Allows players to reset the game to its original state.  
● Clears all highlighted squares, refreshed moves, and piece positions.  
● Ensures clean session handling where each new game starts from a fresh board setup.  
● Maintains professional flow and prevents leftover state from previous games.

## 4.4 Data Flow Diagram



### Figure 4.1 DFD Chess | The Game of India

This diagram illustrates the flow of data between the Player (White), Player (Black), the UI Board Interface, and various system modules. It demonstrates how “Chess | The Game of India” manages piece movement, turn switching, special rules, and board updates efficiently.

***User Interaction***

● Players (White and Black) interact with the system by clicking on chess pieces on the board.

● After selecting a piece, the system highlights valid moves based on chess rules.

● Once a move is chosen, the game validates and executes it, then switches the turn.

***UI Board Interface***

● Board Rendering: Displays the 8×8 chessboard with modern styling and responsive layout.

● Piece Display Module: Renders each chess piece using images placed on corresponding squares.

● Move Highlighting: Shows valid moves (green), selected piece (pink), and special move options (aqua).

● Turn Indicator: Updates text (“White’s Turn” / “Black’s Turn”) after every legal move.

● Reset Module: Allows restarting the game from the initial setup.

***White Player & Black Player Modules***

● Piece Selection Module: Detects which piece the player has clicked and validates whether the move is allowed in the current turn.

● Move Execution Module: Moves the piece to the selected square if the move is legal.

● Capture Module: Handles removing opponent pieces when landing on an occupied valid square.

● Special Move Module:

– Castling logic for both kings

– Pawn promotion to queen

– Basic en-passant prevention logic (if needed later)

***Core System Modules***

1. Move Validation Module

● Calculates legal moves for each piece based on its movement pattern.

● Ensures moves follow the rules of chess and are within board boundaries.

● Blocks moves obstructed by other pieces.

***2. Turn Management Module***

● Switches turns after each valid move.

● Restricts interaction to the correct player’s pieces.

● Ensures smooth alternating gameplay between White and Black.

***3. Board State Management Module***

● Keeps track of all piece positions using IDs and DOM elements.

● Updates board colors and layout after every move.

● Handles piece rendering, disappearance after capture, and square color reset.

***4. Special Move Handling Module***

● Processes advanced mechanics such as:

– Castling validations (king and rook movement)

– Pawn reaching final rank and auto-promoting

● Ensures special moves follow chess rules and update board accurately.

***Data Structure / State Handling***

Since the project is browser-based and runs entirely on the client:

● Board State (JavaScript): Stores piece positions and valid move logic.

● DOM Elements (HTML/CSS): Act as the visual representation of the board.

● No backend dependency: All computations, validations, updates are handled in real-time through JavaScript logic.

***System Communication Flow***

● The user click event triggers piece selection → move validation → highlight display.

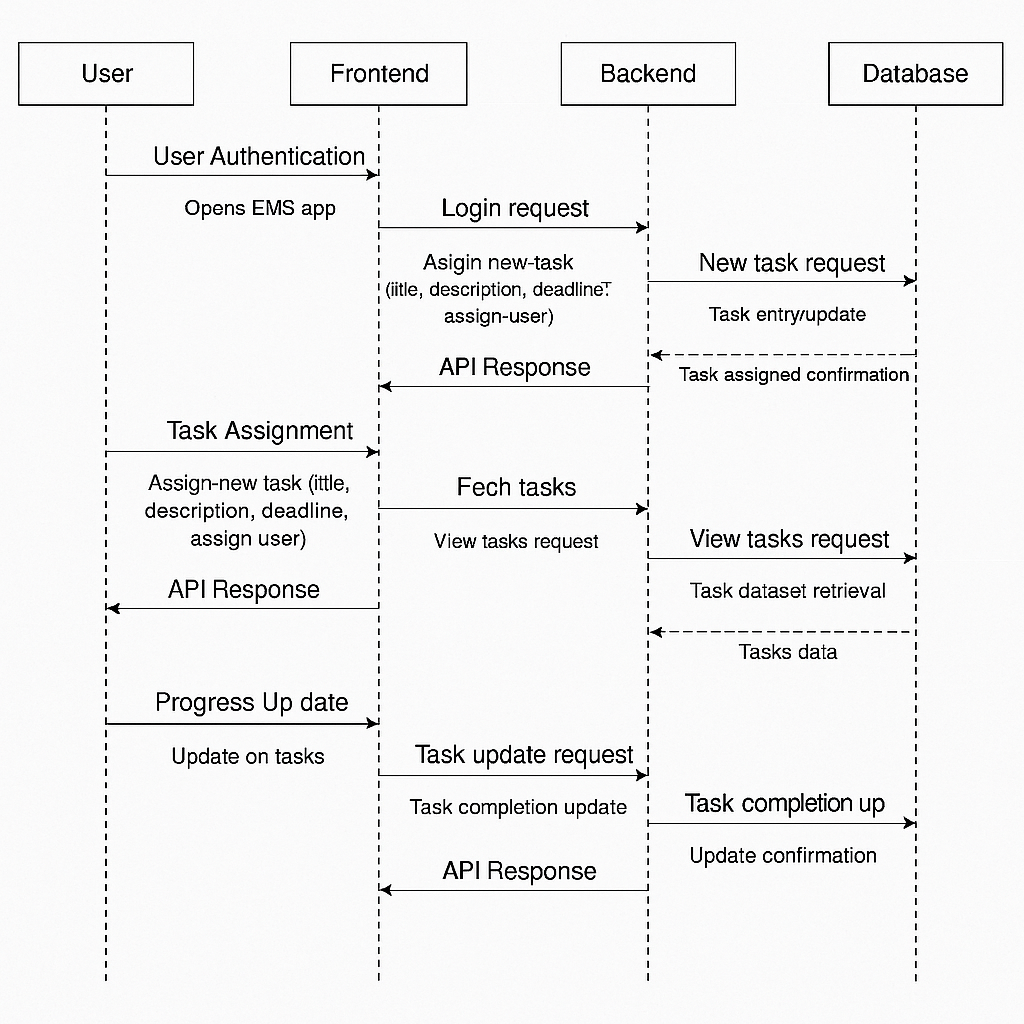
● When a valid square is clicked, the system updates the piece position, resets highlights, and switches the turn.

● Special rules (castling/promotion) are triggered automatically when conditions are met.

● The board and UI update instantly, ensuring real-time feedback for both players.

● The system maintains a smooth, responsive, and rule-driven gameplay flow between White and Black throughout the match.

## 4.5 Sequence Diagram



### Figure 4.2 Sequence Diagram Employee Management System

This sequence diagram illustrates the interaction between different components of the Employee Management System (EMS) — the Admin/User, Frontend, Backend, and Database Services.

It is divided into four major interactions: *User Authentication, Task Assignment, Task Overview, and Progress Update.*

***1. User Authentication***

* *User Interaction:*

The user (either Admin or Employee) opens the EMS application and enters login credentials.

* *Frontend:*

The frontend sends the login data to the backend through an API request for authentication.

* *Backend:*

The backend verifies credentials against user records stored in the database.

* *Database Service:*

The database returns the user data or authentication failure message.

* *Frontend:*

Displays the dashboard (Admin/User) after successful login or an error message if authentication fails.

***2. Task Assignment (Admin Panel)***

* *Admin Interaction:*

Admins create and assign new tasks to specific employees.

* *Frontend:*

The frontend sends task details (title, description, deadline, assigned user) to the backend API.

* *Backend:*

The backend validates the input, updates the respective user’s record in the database, and confirms successful task assignment.

* *Database Service:*

Stores new task data linked to the user account.

* *Frontend:*

Displays a confirmation message to the admin that the task has been successfully assigned.

***3. Task Overview (Admin Dashboard)***

* *Admin Interaction:*

The admin requests a consolidated view of all assigned tasks.

* *Frontend:*

Sends a “fetch task” request to the backend API.

* *Backend:*

Retrieves all stored tasks and their statuses from the database.

* *Database Service:*

Return task dataset to backend.

* *Frontend:*

Displays a structured task overview table with details like employee name, task status, and progress.

***4. Progress Update (User Dashboard)***

* *User Interaction:*

The employee logs into their dashboard and updates task progress or marks a task as completed.

* *Frontend:*

Sends a “task update” request to the backend API.

* *Backend:*

Update task completion status in the database.

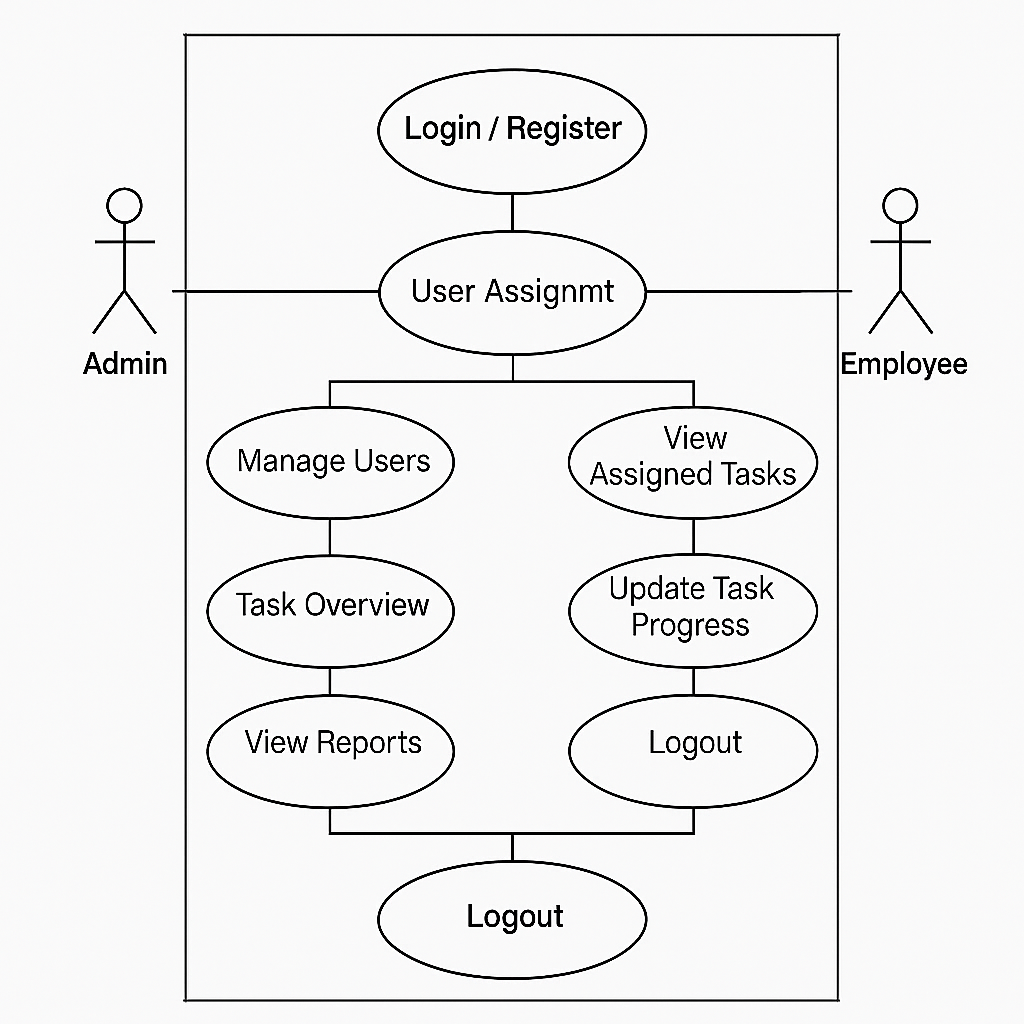
* *Database Service:*

Saves the new task status and confirms the update.

* *Frontend:*

Reflects the updated task status instantly on both the User Dashboard and Admin Dashboard.

## 4.6 Use Case Diagram



### Figure 4.3 Sequence Diagram Employee Management System

This Use Case Diagram illustrates the interactions between the Admin and Employee within the Employee Management System (EMS).

It highlights how both roles interact with various modules to manage users, assign tasks, track progress, and view task details.

***1. Login / Register***

* ***Description:***

Allows both Admin and Employees to access the system through secure authentication.

* ***Actors***: Admin, Employee
* ***Flow***:
  + The user accesses the login or registration page.
  + If registering, the user enters details such as name, email, and role (Admin/Employee).
  + If logging in, the credentials are verified and the respective dashboard (Admin or Employee) is displayed.

***2. User Management (Admin Only)***

* ***Description:***

Enables the Admin to create, edit, or remove employee accounts.

* ***Actors:*** Admin
* ***Flow***:
  + The Admin logs into the system.
  + Navigate to the “Manage Users” module.
  + Adds new users, assigns roles, or removes inactive employees.
  + The system updates the database and confirms the operation.

***3. Task Assignment (Admin Only)***

* ***Description:***

Allows the Admin to assign specific tasks to employees.

* ***Actors:*** Admin.
* ***Flow:***
  + The Admin selects an employee from the user list.
  + Creates a new task by entering details such as title, description, and deadline.
  + The system stores the task in the database and notifies the assigned employee.

***4. Task Overview (Admin)***

* ***Description:***

Provides the Admin with a comprehensive overview of all assigned tasks.

* ***Actors***: Admin
* ***Flow:***
  + The Admin opens the Task Overview section.
  + The system fetches all employee tasks with completion statuses.
  + Admin reviews progress and manages workload accordingly.

***5. View Assigned Tasks (Employee)***

* ***Description:***

Allows employees to view tasks assigned by the Admin.

* ***Actors:*** Employee
* ***Flow:***
  + The Employee logs into the system.
  + Access “My Tasks” dashboard.
  + The system displays a list of all assigned tasks with deadlines and status.

***6. Update Task Progress (Employee)***

* ***Description:***

Enables employees to update task status (e.g., In Progress, Completed).

* ***Actors:*** Employee
* ***Flow:***
  + The Employee selects a specific task.
  + Update progress or mark it as completed.
  + The system saves the changes and notifies the Admin of the update.

***7. View Reports (Admin)***

* ***Description:***

Displays detailed reports on employee performance and task completion.

* ***Actors:*** Admin
* ***Flow:***
  + Admin selects the “Reports” option.
  + The system generates visual summaries of task completion rates and deadlines.
  + Reports assist the Admin in tracking efficiency and productivity.

***8. Logout***

* ***Description:***

Ends the user session securely for both Admin and Employees.

* ***Actors:*** Admin, Employee
* ***Flow:***
  + The user clicks the Logout button.
  + The system clears session data and redirects to the login page.

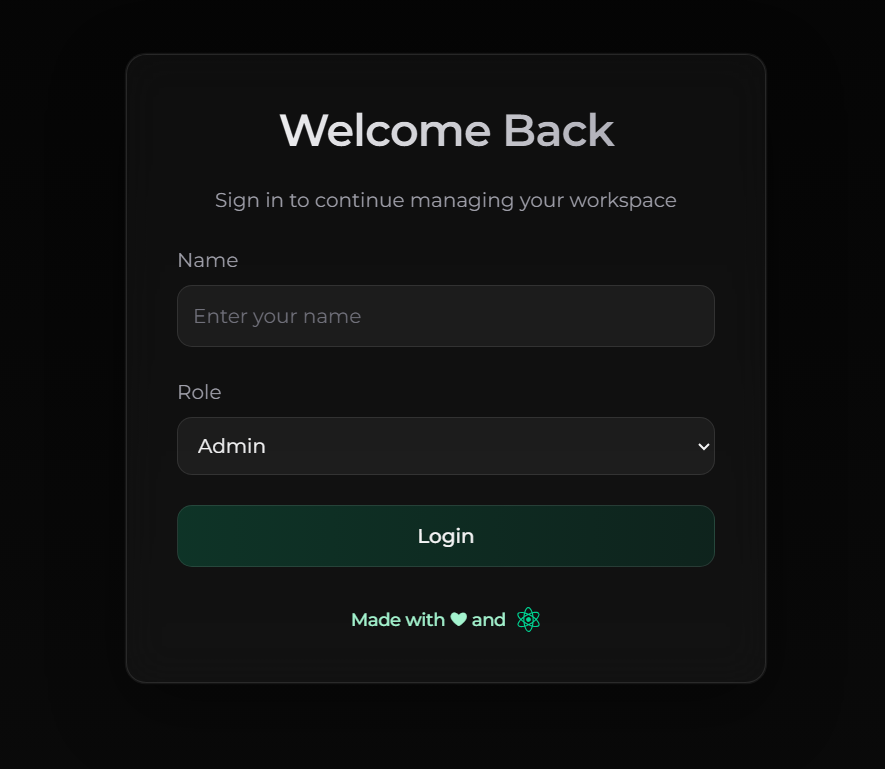
## 4.7 UI Interfaces

1. ***Login Page***

The login/register page enables users to securely access the platform,

facilitating account creation or authentication for personalized interaction

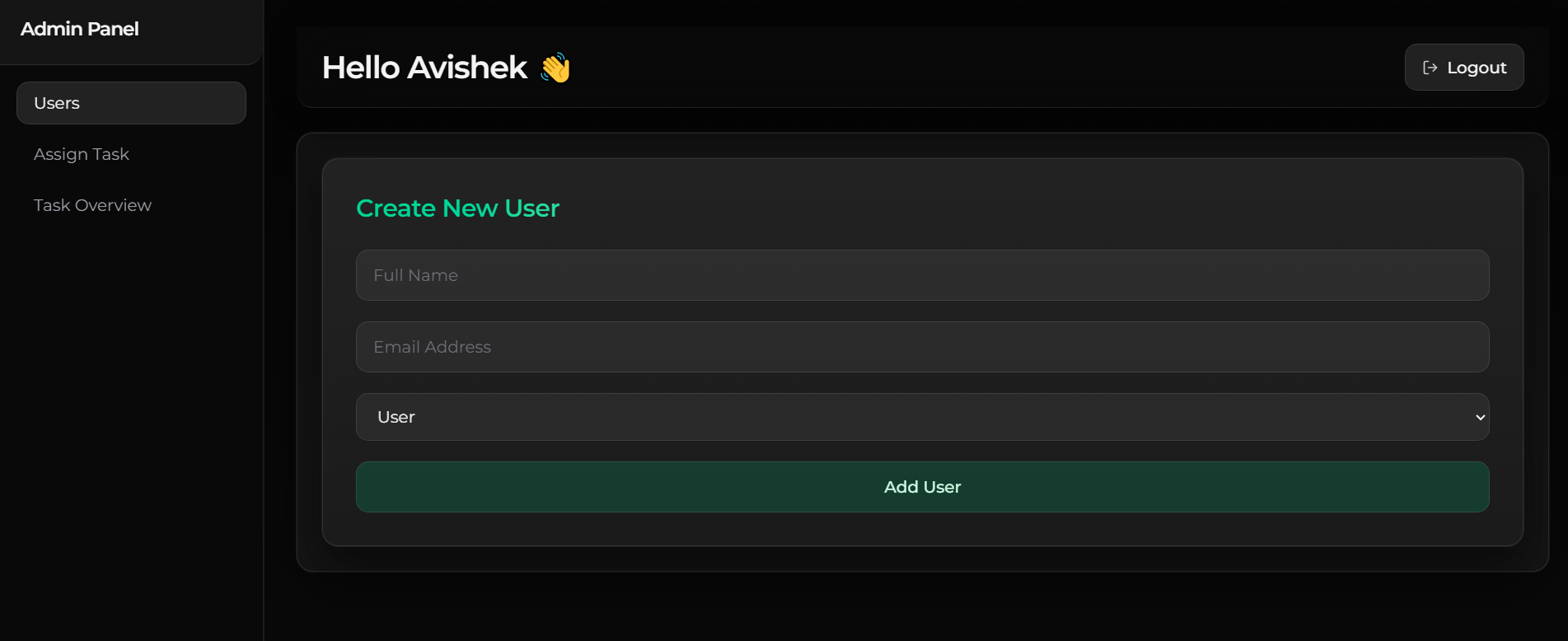
with the system.



### Figure 4.4 Login Page

1. ***Admin Dashboard***

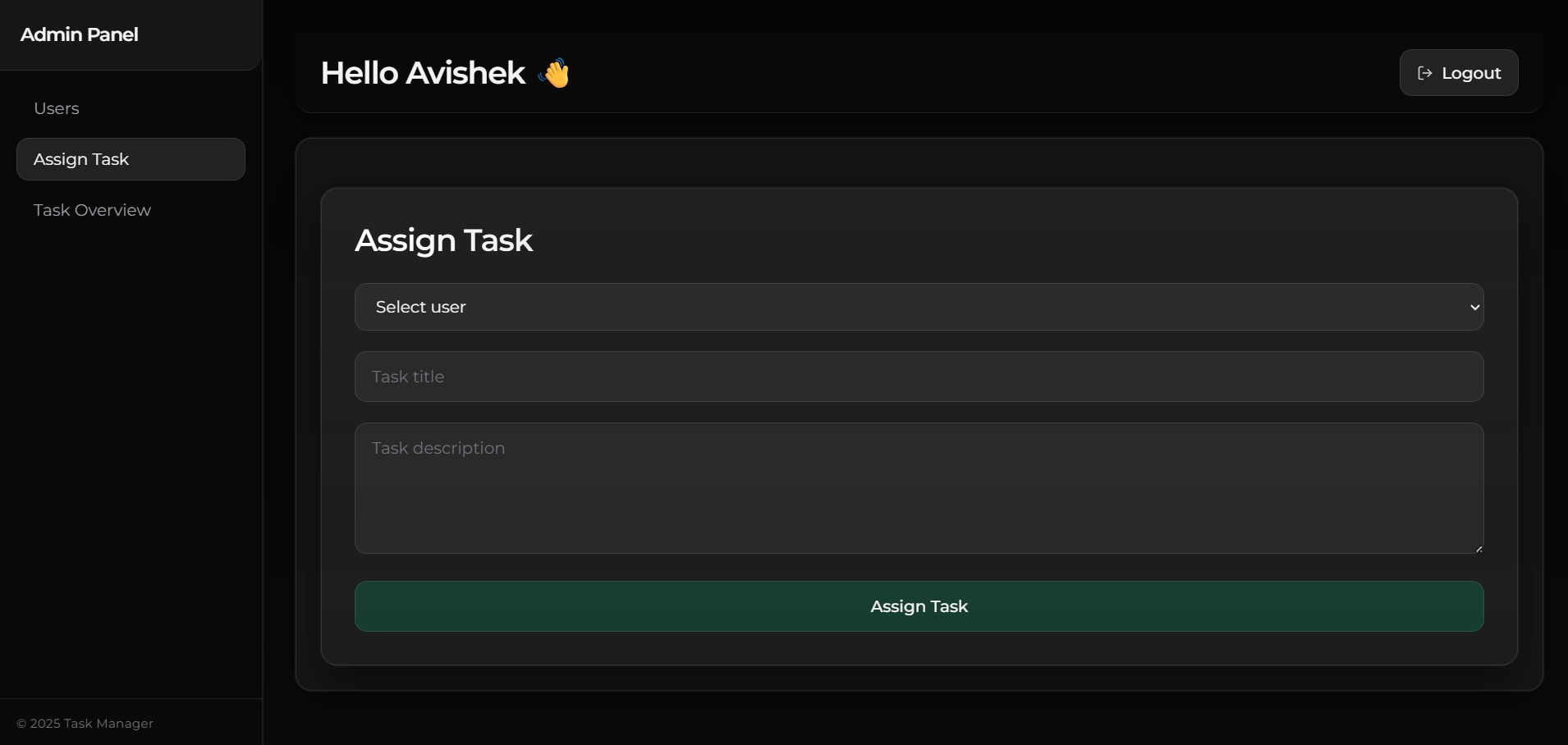
The admin dashboard pages several sections including, creating user or admin, assigning and managing tasks. The default view after admin login is creating an user or admin with name, email and role.



### Figure 4.5 Admin Dashboard - Create User/Admin Page

1. ***Assign Task Page***

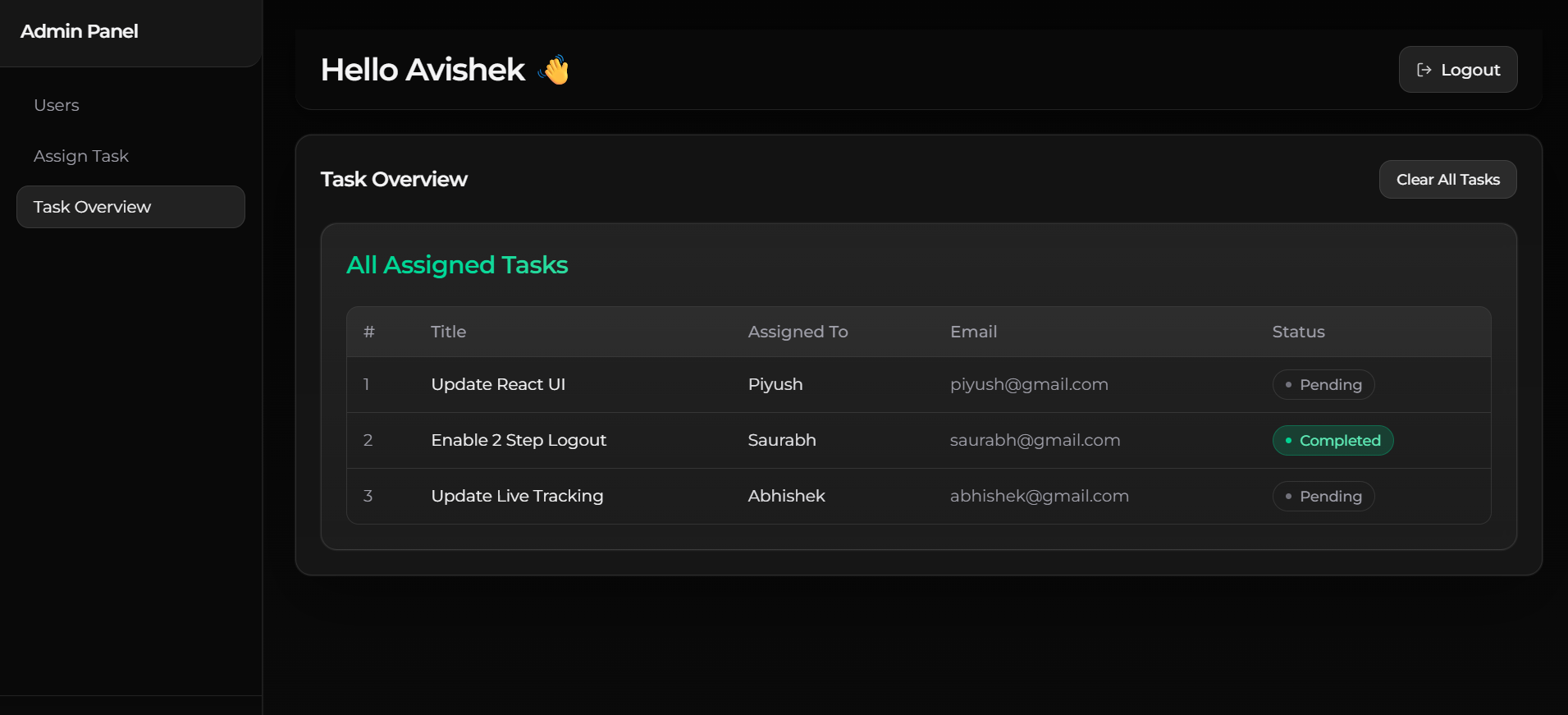
Now, the admin can assign tasks to the user or any other admin, by selecting a user/admin, and thereby entering the title of the task and details regarding the tasks in respective fields, after that upon successfully clicking Add User button, the task will be updated to local storage.



### Figure 4.6 Assign Task Page

1. ***Task Overview for Admin***

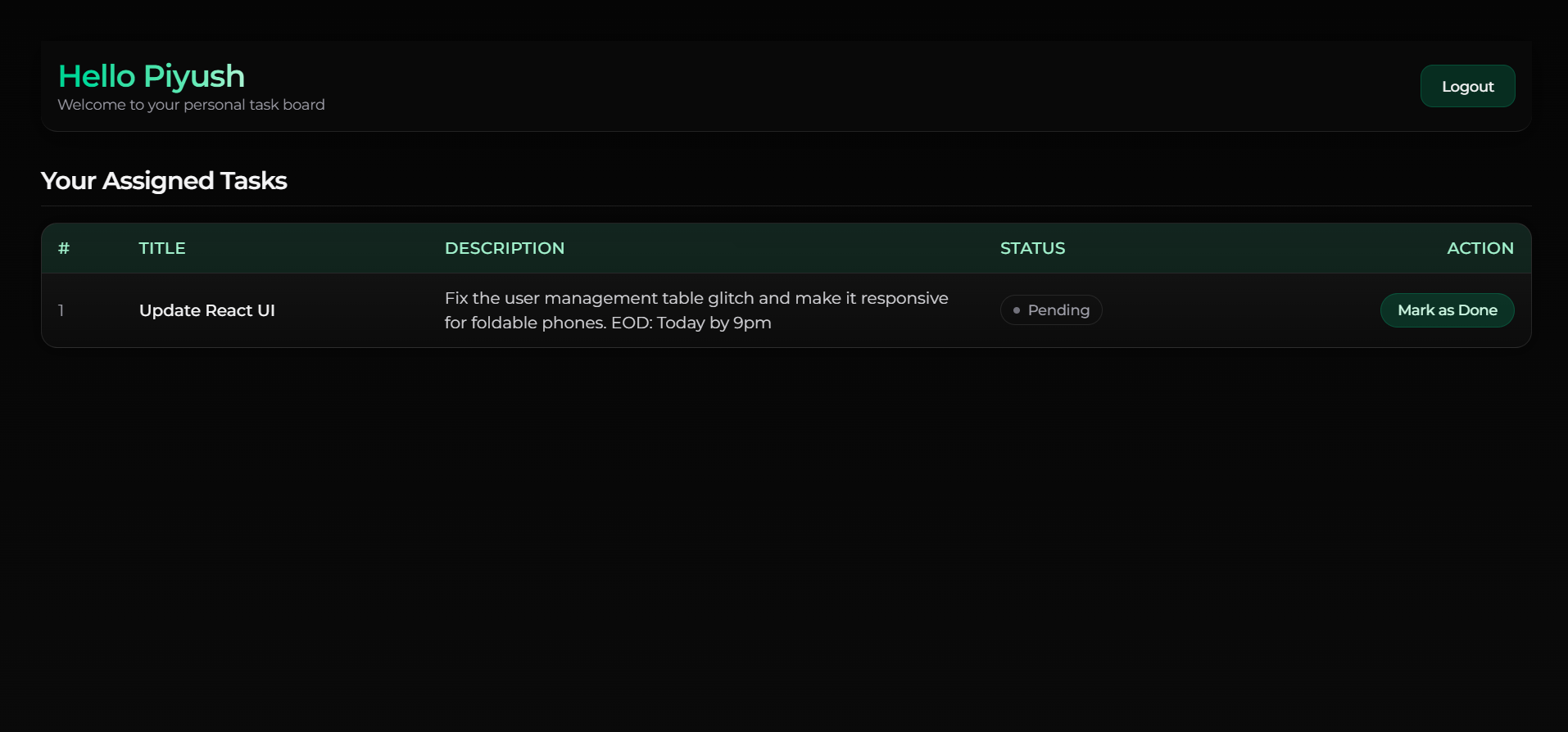
After assigning the task user can view all the assigned tasks to respective persons, with a tracker of the task being completed or not. The task completion is updated by the user and shown real time here. Admin can clear all tasks based on priority.



### Figure 4.7 Task Overview Page

1. ***User Dashboard***

The user dashboard all the pending and completed tasks as assigned by the admin. The user can change the status of the task as completed whenever the task is done and will be shown real time in the user dashboard.



### Figure 4.8 User Dashboard

# CHAPTER 5: Technologies Used

A description about the various tools, libraries, and programming languages used during the development of the Employee Management System project are as follows:

## 5.1 JavaScript

JavaScript (JS) plays a crucial role in the development of the Employee Management System, particularly in the frontend. It brings interactivity, responsiveness, and logic to the application, making the user interface intuitive and efficient. Using JavaScript and its ecosystem allows the system to handle real-time updates such as task assignment, progress tracking, and dynamic user dashboards seamlessly. Frameworks like React enhance component-based UI development, while Node.js on the backend ensures fast and scalable performance.

***Features:***

1. *Asynchronous Programming:* JS supports asynchronous operations, enabling smooth handling of multiple processes like task updates and user requests simultaneously without page reloads.
2. *ES6 Modules:* With modular JavaScript, the system maintains reusable and well-structured code, simplifying debugging and future updates.
3. *Component-Based Development (React):* React enables building reusable UI components like task cards, user tables, and admin dashboards efficiently.
4. *Event-Driven Backend (Node.js):* Node.js handles multiple concurrent API requests efficiently, ensuring real-time responsiveness between users and the admin dashboard.
5. *Real-Time Updates (Socket.IO):* Enables live updates of task progress and status changes across users and admin dashboards without manual refreshing.

## 5.2 React.js (Frontend Logic & State Management)

The Employee Management System is a frontend-only web application built with React.js, handling both user interface and data management. Instead of a dedicated backend server like Node.js or Express, the system uses browser localStorage for persisting user, task, and progress data. This approach simplifies deployment and allows full functionality directly from the client side.

***Features:***

1. *Component-Based Design*: Each major feature — user management, task assignment, and task overview — is modularized into reusable React components.
2. *LocalStorage Integration:* Data persistence (users, tasks, roles) is managed entirely in the browser, enabling seamless offline use.
3. *Real-Time Updates:* State updates instantly reflect on the interface, mimicking real backend behavior.
4. *Routing:* React Router manages navigation between different dashboard sections without page reloads.

Scalable Structure: The app architecture easily supports future migration to a real backend (Node + Express + MongoDB).

## 5.3 Tailwind CSS

Tailwind CSS powers the visual layer of the Employee Management System. Its utility-first design system allows for rapid UI development, resulting in a clean, modern, and responsive interface that aligns with the dark theme aesthetic of the application.

***Features:***

1. *Utility-First Framework*: Enables direct styling via class names (e.g., bg-emerald-500, text-zinc-100), promoting fast prototyping.
2. *Responsive Design*: Built-in breakpoints ensure the system adapts to screens of all sizes—from desktops to mobile devices.
3. *Customizable Themes*: Allows easy color scheme control (emerald green and black) for consistent branding.
4. *Low Specificity:* Prevents CSS conflicts, simplifying updates and maintenance.
5. *Ecosystem & Plugins:* Offers plugins for forms, animations, and typography to enhance UI aesthetics.

## 5.4 Visual Studio Code

Visual Studio Code (VS Code) was the primary IDE used for developing the Employee Management System. It provides an efficient and customizable environment for writing, debugging, and testing code with full JavaScript and React support.

***Features:***

1. *IntelliSense:* Smart suggestions for React components, Node APIs, and MongoDB queries.
2. *Debugger:* Built-in debugging with breakpoints and real-time variable inspection.
3. *Extensions Marketplace:* Enhances productivity through extensions like Prettier, ESLint, and Tailwind IntelliSense.
4. *Git Integration:* Streamlined version control with direct commit, branch, and merge support.
5. *Cross-Platform:* Consistent experience across Windows, macOS, and Linux for collaborative development.

## 5.5 Git Version Control

Git was used for version management throughout the project lifecycle. It allowed tracking code changes, branching for new features, and merging updates efficiently among team members.

***Features:***

1. *Distributed Version Control:* Enables each developer to maintain a full local repository copy for offline work.
2. *Branching & Merging:* Helps manage separate feature branches for modules like Admin Dashboard and Task Overview.
3. *Commit History:* Maintains detailed logs of all changes and development progress.
4. *Collaboration*: Supports concurrent development while minimizing merge conflicts.
5. *Version Rollback*: Easily revert to previous stable versions during testing or deployment.

# CHAPTER 6: Testing & Validation

## 6.1 Component and Functional Testing

Each component of the Employee Management System (such as Login, Dashboard, Sidebar, Task Overview, and Assign Task) was tested individually to verify that all modules perform as expected. Functionality such as user creation, task assignment, progress tracking, and logout were validated using mock data in localStorage. Edge cases (like empty inputs or duplicate users) were also tested to ensure stability.

## 6.2 Browser Compatibility Testing

The application was tested across multiple web browsers — Google Chrome, Mozilla Firefox, Microsoft Edge, and Brave — to ensure consistent performance and visual integrity. Special attention was given to layout rendering, responsiveness, and smooth animations powered by Framer Motion and Tailwind CSS.

## 6.3 Operating System Compatibility Testing

The frontend was executed and verified on various operating systems including Windows, macOS, and Linux (Ubuntu). The tests confirmed that the application performs uniformly with no UI distortion, logic break, or dependency-related errors across environments.

## 6.4 NPM Package Integrity Validation

All installed npm packages — including React, Framer Motion, Lucide Icons, and Tailwind CSS — were validated for compatibility using Git Bash and npm audit commands. The production build was generated using npm run build to ensure that no dependency conflicts or runtime issues were present.

## 6.5 Network and Performance Testing

Although the Employee Management System operates locally through client-side storage, network responsiveness was evaluated for deployment readiness. The application was tested under different network conditions — broadband, LAN, and mobile hotspots — to ensure smooth loading and minimal lag when hosted on Firebase or Vercel.

## 6.6 Mobile Responsiveness Testing

Using Chrome DevTools and live devices (smartphones and tablets), the application was tested on multiple screen sizes to confirm responsive design behavior. Layouts were verified for fluid scaling, proper sidebar behavior, and adaptive text. The UI remained intuitive and accessible on smaller screens due to Tailwind’s responsive classes (sm:, md:, lg: breakpoints).

# CHAPTER 7: Conclusion

In the ever-evolving landscape of digital transformation, the Employee Management System emerges as a robust and efficient solution for modern organizational needs. This project embodies the principles of automation, simplicity, and precision — aiming to replace outdated manual methods of managing employee records and tasks. With its sleek, intuitive interface and modular React-based architecture, the system ensures seamless interaction between administrators and employees.

At its core, the Employee Management System is not just a tool for task assignment and progress tracking but a comprehensive framework that enhances communication, accountability, and productivity within the workplace. Through efficient use of local storage, dynamic dashboards, and real-time task status visualization, it streamlines daily operations without the complexity of backend dependencies.

Ultimately, this project demonstrates how thoughtful design and modern web technologies can redefine efficiency in workforce management — delivering a scalable, responsive, and user-friendly solution adaptable to diverse organizational environments.

# CHAPTER 8: Future Scope

While the current version of the Employee Management System provides a strong and functional framework for task assignment, user management, and progress tracking, there remains significant potential for future improvement and scalability. The following advancements can be implemented in subsequent iterations:

***Backend Integration with Database:***

Introduce a robust backend using Node.js, Express.js, and MongoDB to replace localStorage. This will ensure secure, centralized, and scalable data management suitable for enterprise-level use.

***User Authentication and Role Management:***

Implement a complete authentication system with password hashing, session handling, and role-based access control to enhance security and prevent unauthorized access.

***Advanced Analytics Dashboard:***

Integrate analytical tools and data visualization libraries to provide insights into employee performance, task completion rates, and productivity trends through interactive charts and reports.

***Automated Notifications and Email Integration:***

Enable email or in-app notifications for task updates, deadlines, and new assignments to keep employees and administrators informed in real-time.

***Cloud Deployment and Accessibility:***

Deploy the system on cloud platforms such as Firebase, Vercel, or AWS, ensuring global accessibility, better performance, and uptime reliability.

***Mobile Application Development:***

Develop a dedicated mobile version using React Native or Flutter to provide on-the-go access to dashboards and task management tools for employees and administrators.

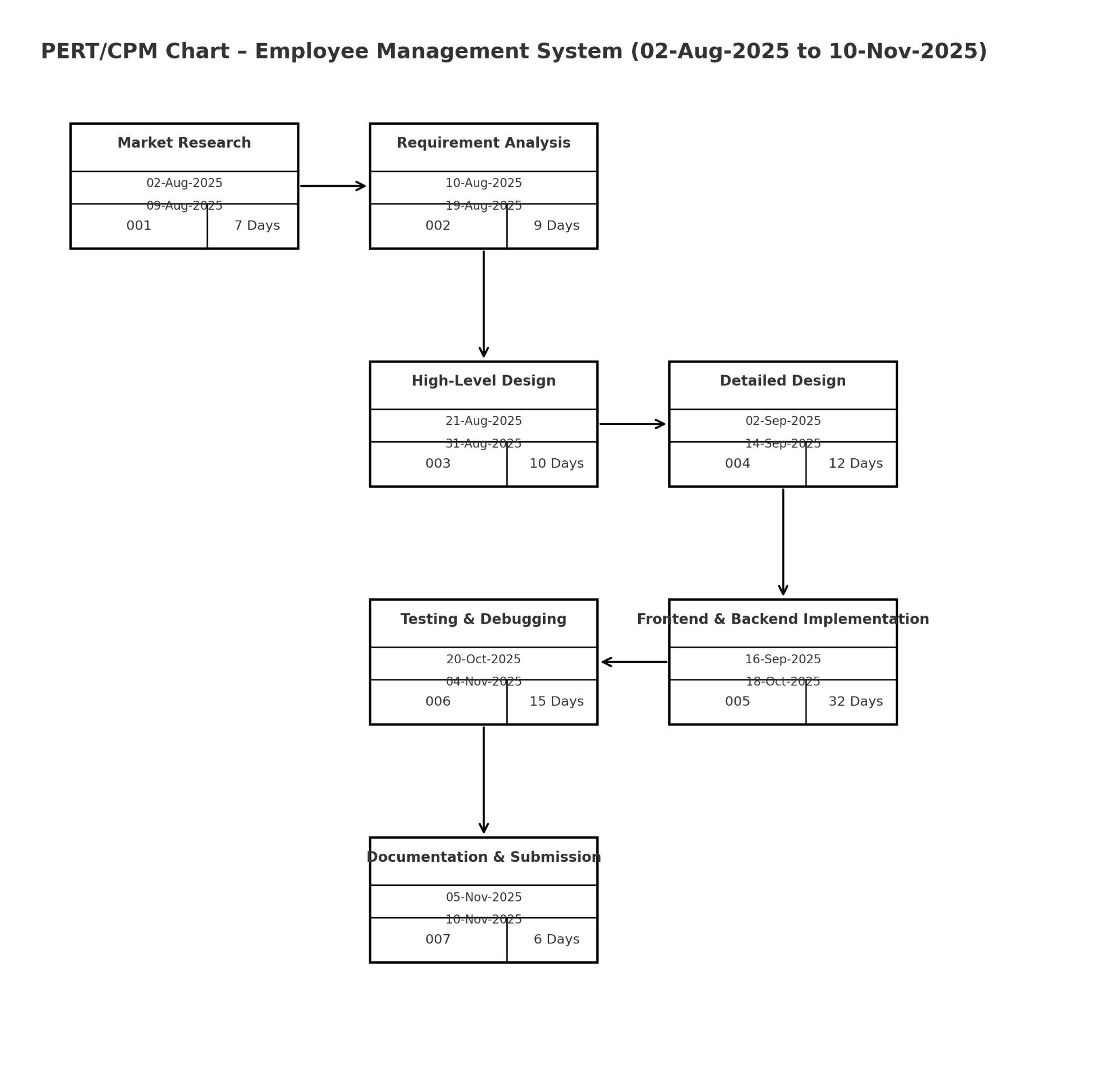
***Integration with Payroll and HR Modules:***

Extend the system to include payroll, attendance tracking, and leave management features, transforming it into a complete HR automation suite.

***Enhanced UI/UX and Accessibility:***

Continue refining the design for improved responsiveness, accessibility, and user experience across all devices and platforms.

# CHAPTER 9: PERT/CPM Chart



# Figure 9.1 Program Evaluation and Review Technique & Critical Path MethodReferences

1. React.js Team (Meta). (2025). *React – A JavaScript Library for Building User Interfaces.* Retrieved from<https://react.dev/>
2. Tailwind Labs Inc. (2025). *Tailwind CSS Documentation.* Retrieved from<https://tailwindcss.com/docs>
3. Google Firebase. (2025). *Firebase Documentation – App Development Platform.* Retrieved from<https://firebase.google.com/docs>
4. Postman Inc. (2025). *Postman API Platform – API Development and Testing Tool.* Retrieved from<https://www.postman.com/>
5. Git SCM. (2025). *Git Documentation – Distributed Version Control System.* Retrieved from<https://git-scm.com/docs>
6. Johnson, R. E., & Foote, B. (1988). *Designing Reusable Classes*. Journal of Object-Oriented Programming, 1(2), 22–35.
7. Lee, S., Park, J., & Kim, H. (2020). *Cloud-Based Application Development and Deployment Frameworks*. Proceedings of the 2020 International Conference on Cloud Computing, ACM.
8. Brown, T., Mann, B., Ryder, N., et al. (2020). *Language Models are Few-Shot Learners*. Advances in Neural Information Processing Systems (NeurIPS), 33, 1877–1901.
9. Express.js Documentation. (2025). *Fast, unopinionated, minimalist web framework for Node.js.* Retrieved from<https://expressjs.com/>
10. MongoDB, Inc. (2025). *MongoDB: The Developer Data Platform.* Retrieved from<https://www.mongodb.com/docs/>
11. React.js Team (Meta). (2025). *React – A JavaScript Library for Building User Interfaces.* Retrieved from<https://react.dev/>
12. Tailwind Labs Inc. (2025). *Tailwind CSS Documentation.* Retrieved from<https://tailwindcss.com/docs>
13. Google Firebase. (2025). *Firebase Documentation – App Development Platform.* Retrieved from<https://firebase.google.com/docs>
14. Postman Inc. (2025). *Postman API Platform – API Development and Testing Tool.* Retrieved from<https://www.postman.com/>
15. Git SCM. (2025). *Git Documentation – Distributed Version Control System.* Retrieved from<https://git-scm.com/docs>
16. Johnson, R. E., & Foote, B. (1988). *Designing Reusable Classes*. Journal of Object-Oriented Programming, 1(2), 22–35.
17. Lee, S., Park, J., & Kim, H. (2020). *Cloud-Based Application Development and Deployment Frameworks*. Proceedings of the 2020 International Conference on Cloud Computing, ACM.
18. Brown, T., Mann, B., Ryder, N., et al. (2020). *Language Models are Few-Shot Learners*. Advances in Neural Information Processing Systems (NeurIPS), 33, 1877–1901.
19. Express.js Documentation. (2025). *Fast, unopinionated, minimalist web framework for Node.js.* Retrieved from<https://expressjs.com/>
20. MongoDB, Inc. (2025). *MongoDB: The Developer Data Platform.* Retrieved from<https://www.mongodb.com/docs/>