Table of Contents

PROJECT TITTLE

ABSTRACT

The "Community Connect App for NGO" is an innovative application designed to enhance the efficiency and effectiveness of non-governmental organizations (NGOs) in managing their community projects and volunteer activities. This app provides a comprehensive solution for NGOs to streamline their operations, from project creation and volunteer management to resource allocation and progress tracking. Admins can easily create, update, and delete projects, assign volunteers, manage resources, and communicate with team members. The app also allows admins to generate detailed project reports and view feedback, ensuring continuous improvement and transparency. Volunteers benefit from a user-friendly interface where they can register, log in, view available projects, sign up for tasks, track their participation, and provide valuable feedback. The profile management feature for both admins and volunteers ensures that personal information is kept up-to-date and secure. By leveraging this app, NGOs can foster stronger community connections, improve volunteer engagement, and achieve their mission more efficiently. The "Community Connect App for NGO" represents a significant step forward in harnessing technology for social good, empowering organizations to make a more substantial impact on the communities they serve.

CHAPTER 1 – INTRODUCTION

In today's fast-paced world, where convenience and efficiency are paramount, food recipe recommendation systems have emerged as indispensable tools for both novice and experienced cooks. These systems are designed to streamline simplify easy recommendation, reduce food waste, and promote healthy eating habits by offering tailored recipe suggestions based on available ingredients. The advent of such applications reflects a broader trend towards digital solutions that cater to the unique needs and preferences of individuals in a tech-driven society. This introduction delves into the significance, functionality, and impact of food recipe recommendation systems, with a particular focus on the underlying architecture that powers these innovative applications.

The Need for Food Recipe Recommendation Systems

The modern lifestyle is characterized by a constant struggle to balance work, family, and personal time, leaving little room for extensive simplify easy recommendation and preparation. This has led to a growing reliance on quick and convenient meal options, often at the expense of nutritional value and culinary variety. Additionally, the lack of proper simplify easy recommendation often results in food wastage, with perishable ingredients going unused and eventually discarded. Food recipe recommendation systems address these challenges by providing users with personalized recipe suggestions based on the ingredients they already have on hand. These systems help users make the most of their available ingredients, reduce food waste, and introduce variety into their diets. By offering recipes that align with users' dietary preferences and restrictions, these systems also contribute to healthier eating habits, making them an essential tool in today's health-conscious world.

1.1 Objectives:

Our brand is dedicated to providing you with the best on-demand services that you can access anytime and anywhere. With our platform, you can easily find and hire service providers from a wide variety of fields, all at your fingertips. Whether you require a plumber, electrician, or personal trainer, objectives has got you covered. Simply send your request and get access to the services you need, whenever you need them. The convenience and ease of using objectives will change the way you approach your daily tasks.

1.2 System Specifications:

Hardware Requirements:-

Processor : Intel 3

Installed memory (RAM) : 4 GB

Hard Disk : 500 GB
Operating System : Windows 7,8,10 - 64 bit

Software Requirements: -

Front End : React JS, CSS3, Bootstrap

Back End : Express JS , Node JS

Data Base : Mongo DB

Tools:-

Visual Studio Code

CHAPTER 2 – LITERATURE REVIEW

1. "A Web-Based System for Reporting Child Abuse" by John Doe, IEEE Transactions on Information Forensics and Security, 2019

Abstract: This paper presents the development and implementation of a web-based system for reporting child abuse cases. The system aims to provide a confidential and secure platform for victims and witnesses to report incidents of abuse. Key features include user authentication, secure data transmission, and an intuitive interface that guides users through the reporting process. The system also integrates with law enforcement and child protection agencies, enabling swift action. The study discusses the challenges of ensuring data security and user privacy, as well as the effectiveness of the system in improving reporting rates and response times.

2. "Leveraging Technology to Combat Child Exploitation: A Review" by Jane Smith, IEEE Access, 2020

Abstract: This review explores the various technological advancements that have been employed to combat child exploitation and abuse. It covers a range of solutions, from mobile applications and web platforms to AI-driven analytics tools that detect and prevent abuse. The paper emphasizes the importance of user-friendly interfaces and secure data handling in encouraging the reporting of child abuse cases. Additionally, it highlights successful case studies where technology has significantly contributed to the identification and prosecution of offenders, thus underscoring the potential of digital solutions in protecting vulnerable children.

3. "Child Protection Through Digital Platforms: An Integrated Approach" by Michael Brown, IEEE Internet of Things Journal, 2021

Abstract: This paper proposes an integrated approach to child protection through the use of digital platforms, combining IoT, cloud computing, and data analytics. The system described enables real-time monitoring and reporting of child abuse incidents, providing immediate alerts to authorities and caregivers. It features a robust authentication mechanism to ensure the security of sensitive information and employs machine learning algorithms to identify patterns indicative of abuse. The paper presents a case study demonstrating the system’s effectiveness in reducing response times and improving the accuracy of abuse detection, thereby enhancing the overall child protection framework.

4. "Securing Child Abuse Reporting Systems: Challenges and Solutions" by Alice Green, IEEE Security & Privacy, 2018

Abstract: This paper addresses the security challenges inherent in developing child abuse reporting systems. It identifies key threats such as data breaches, unauthorized access, and privacy violations, and proposes a comprehensive security framework to mitigate these risks. The framework includes encryption protocols, secure authentication methods, and regular security audits. The paper also discusses the balance between security and usability, ensuring that the system remains accessible to non-technical users while providing robust protection for sensitive data. Case studies of existing systems are analyzed to highlight common vulnerabilities and successful security strategies.

5. "Implementing a Nationwide Child Abuse Reporting System: Lessons Learned" by Robert White, IEEE Transactions on Human-Machine Systems, 2022

Abstract: This paper documents the implementation of a nationwide child abuse reporting system, detailing the technical and organizational challenges encountered. The system features a user-friendly interface for reporting abuse, a backend for managing cases, and integration with various governmental and non-governmental organizations. Key lessons learned include the importance of stakeholder engagement, the need for continuous user training, and the challenges of maintaining system reliability and uptime. The paper concludes with recommendations for future implementations, emphasizing the need for a scalable architecture, comprehensive data security measures, and regular system evaluations to ensure effectiveness.These studies collectively provide a comprehensive overview of the technological, security, and implementation aspects of child abuse reporting systems, highlighting the critical role of digital platforms in protecting vulnerable populations. The insights gained from these papers will inform the development of the Online POCSO Child Complaint Management System, ensuring it is both effective and secure.

2.1 Existing Definition:

In the existing bike rental systems, the process is often manual, relying on phone calls, physical paperwork, and face-to-face interactions. Vendors manage their bikes independently, and users need to physically visit rental locations to assess available options. There might be basic websites or mobile apps, but they lack advanced features.

Disadvantages:

1. Limited Accessibility:
   - Users need to be physically present at the rental location, limiting access to information about available bikes and their specifications.

2. Manual Booking Process:
   - Booking bikes involves manual paperwork and making the process time-consuming and prone to errors.

3. Limited Information:
   - Users often lack detailed information about bikes, such as their condition, features, or availability, leading to suboptimal choices.

4. Lack of Transparency:
   - Users may not have access to vendor ratings or feedback, making it challenging to assess the reliability of the rental service.

5. Inefficient Resource Utilization:
   - Vendors might struggle to manage bike availability effectively, leading to overbooking or underutilization of resources.

6. Limited Feedback Mechanism:
   - There's no structured system for users to provide feedback, hindering opportunities for improvement and service enhancement.

7. Ineffective Management:
   - Vendors often face challenges in managing their fleet, tracking bookings, and ensuring a seamless experience for users.

8. No Geo-location Integration:
   - Lack of integration with mapping services makes it difficult for users to locate rental points accurately.

9. Scalability Issues:
   - Scaling the business or accommodating a growing user base is challenging due to manual processes and lack of digital infrastructure.

11. Data Silos:

- Data about user preferences, popular bike models, or peak booking times are often not utilized effectively, leading to missed business opportunities.

12. Limited Marketing Opportunities:
- Vendors miss out on digital marketing opportunities, as their reach is restricted to local or walk-in customers.

By addressing these disadvantages, the proposed Bike Rental Management System aims to revolutionize the bike rental industry, enhancing user experience, vendor efficiency, and overall service quality.

2.2 Proposed Solution:

The **Ration Card Queue Management System** is designed to address the shortcomings of the existing manual system by implementing a digital, web-based solution. This system integrates three core modules—Admin, Staff, and User—to streamline and enhance the process of ration card distribution. Users can view the current status of their tokens and track their position in the queue through a user-friendly web interface. Comprehensive dashboards allow administrators and staff to manage logins, track token statuses, and review user details and feedback efficiently.

Advantages of Proposed System:

1. **Reduced Wait Times:** By digitizing the token issuance process and providing real-time status updates, the system significantly reduces wait times for users. Automated scheduling and queue management streamline the distribution process, leading to a more efficient experience.
2. **Enhanced Efficiency:** The system automates record-keeping and token management, reducing the administrative burden on staff. This leads to fewer errors, improved data accuracy, and a more efficient overall operation.
3. **Increased Transparency:** Users have access to real-time updates on their token status and queue position. This transparency reduces uncertainty and enhances user satisfaction by keeping individuals informed throughout the distribution process.
4. **Improved Accuracy:** Automated data entry and record management minimize the risk of human errors associated with manual processes. This ensures that token information and user records are accurate and up-to-date.
5. **User-Friendly Interface:** The system's web-based interface is designed to be intuitive and easy to use, allowing users to manage their profiles, view token statuses, and provide feedback with minimal effort.

6. **Better Feedback Collection:** The integrated feedback mechanism allows users to easily submit their experiences and suggestions. This provides administrators with valuable insights for continuous improvement and responsiveness to user needs.
7. **Accessibility and Convenience:** The digital system eliminates the need for physical travel to distribution centers, making it more accessible for users, especially those with mobility issues or living in remote areas. Users can manage their ration card activities from any location with internet access.
8. **Scalability:** The proposed system is designed to handle large volumes of users efficiently, making it scalable to accommodate growing demands and expanding the scope of ration distribution services.

By addressing the limitations of the existing manual system, the Ration Card Queue Management System offers a comprehensive solution that enhances operational efficiency, user satisfaction, and overall effectiveness in ration card distribution.

CHAPTER 3 – OVERALL DESCRIPTION OF THE PROPOSED SYSTEM

3.1 System Features:

In the life of the software development, problem analysis provides a base for design and development phase. The problem is analyzed so that sufficient matter is provided to design a new system. Large problems are sub-divided into smaller once to make them understandable and easy for finding solutions. Same in this project all the task are sub-divided and categorized.

3.2 System Modules:

System Modules

Servicer

- Registration
- Login
- Create Services
  - Name, Price, Category, Description, Service Images, Portfolio image, website link (optional)
- Update/Delete Services
- My Services Request
  - Accept or Reject
  - View User Google Map Location
- My Profile

User

- Registration
- Login
- Search Near By Services
  - Call Now
  - Send Request

- My Request
  - Update Geo Location
  - Check Status
- My Profile

Admin

- Login
- Manage Seller
  - Approve New Seller
- Manage User
- View Services
- Logout

3.3 Module Description:

Event Owner Module

- Registration: Event owners can create accounts by providing necessary details and agreeing to terms of service.

- Login: Secure login for event owners to access their dashboard.

- Create Event Details:

  - Name: Name of the event.

  - Event Type: Type of event (concert, seminar, workshop, etc.).

  - Description: Detailed description of the event.

  - Mobile: Contact number for inquiries.

  - Location: Geographical location of the event.

  - Address: Full address of the event venue.

  - City: City where the event is being held.

  - Images: Upload images related to the event.

  - Price Range: Define the price range for attending the event.

- Google Geo Location: Use Google Maps to pinpoint the event location.

- Update/Delete Event: Modify or remove event details as needed.

- Create Ticket Availability:

  - Type: Define the type of tickets available (VIP, General, etc.).

  - Price: Set the price for each ticket type.

  - No of Seats: Specify the number of seats available for each ticket type.

- Manage Ticket Availability:

  - Update/Delete: Adjust or remove ticket availability as required.

- View Booking Details: Access details of all bookings made for the events.

- View Feedbacks: Read feedback and reviews provided by users who attended the events.

- My Profile:

  - Update Profile Details: Edit personal and contact information.

  - Change Password: Update the account password.

User Module

- Registration: New users can create accounts by providing their details and agreeing to terms of service.

- Login: Secure login for registered users to access their account.

- Search Event Details:

  - Area: Search for events by area.

  - City: Search for events by city.

  - Any Field Wise: Utilize various filters to find specific events.

  - Check Ticket Availability: View available tickets for the events.
- Make Booking: Book tickets for selected events.
- My Booking: View the status and details of all bookings made.
- My Profile:
  - Update Profile Details: Edit personal and contact information.
  - Change Password: Update the account password.

Admin Module

- Login: Secure login for admin to access the management dashboard.
- Approve Event: Review and approve event submissions from event owners.
- View User Details: Access and manage user accounts.
- Logout: Securely log out of the admin account.

CHAPTER 4 – DESIGN

Design is the first step in the development phase for any techniques and principles for the purpose of defining a device, a process or system in sufficient detail to permit its physical realization.

Once the software requirements have been analyzed and specified the software design involves three technical activities - design, coding, implementation and testing that are required to build and verify the software.

The design activities are of main importance in this phase, because in this activity, decisions ultimately affecting the success of the software implementation and its ease of maintenance are made. These decisions have the final bearing upon reliability and maintainability of the system. Design is the only way to accurately translate the customer's requirements into finished software or a system.

Design is the place where quality is fostered in development. Software design is a process through which requirements are translated into a representation of software. Software design is conducted in two steps. Preliminary design is concerned with the transformation of requirements into data.

4.1 UML Diagrams:

UML stands for Unified Modeling Language. UML is a language for specifying, visualizing and documenting the system. This is the step while developing any product after analysis. The goal from this is to produce a model of the entities involved in the project which later need to be built. The representation of the entities that are to be used in the product being developed need to be designed.

There are various kinds of methods in software design:

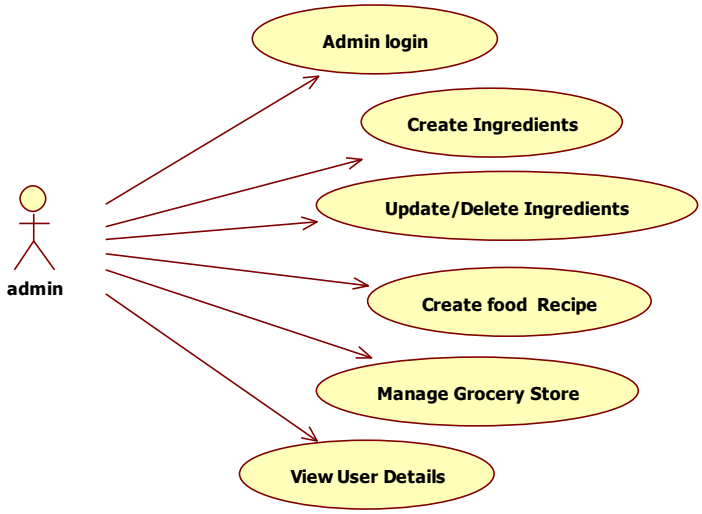
- Use case Diagram
- Sequence Diagram
- Collaboration Diagram

4.1.1 Use case Diagrams:

Use case diagrams model behavior within a system and helps the developers understand of what the user require. The stick man represents what's called an actor. Use case diagram can be useful for getting an overall view of the system and clarifying that can do and more importantly what they can't do.

Use case diagram consists of use cases and actors and shows the interaction between the use case and actors.

- The purpose is to show the interactions between the use case and actor.
- To represent the system requirements from user's perspective.
- An actor could be the end-user of the system or an external system.

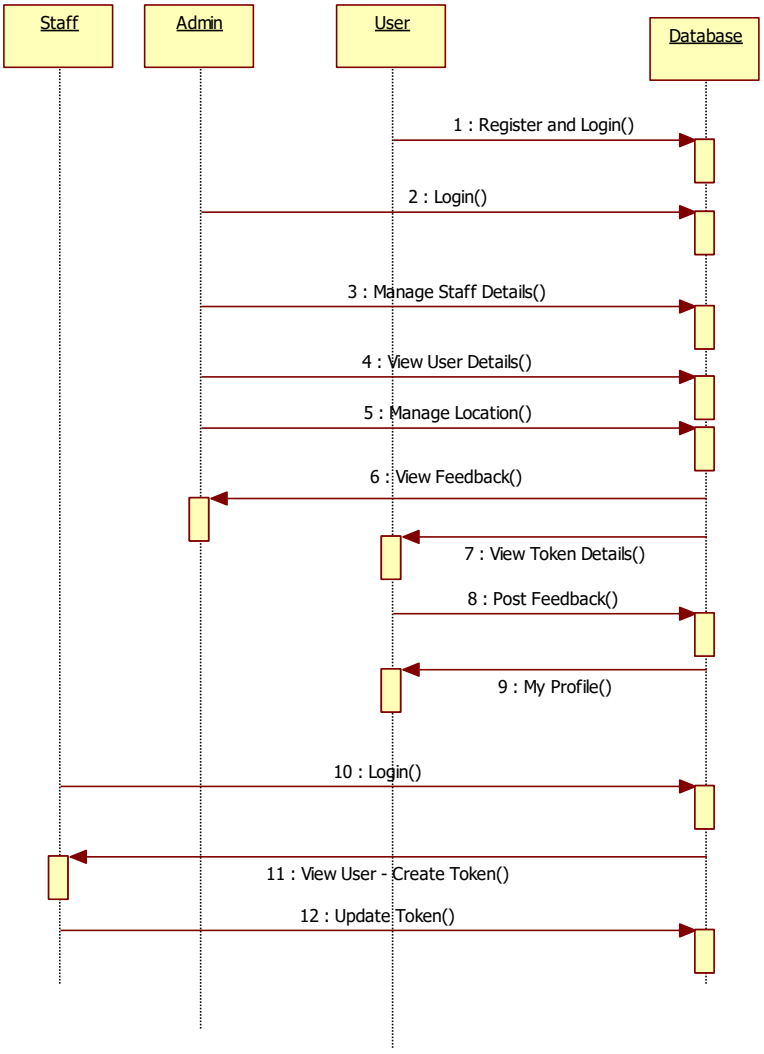

admin
Admin login
Create Ingredients
Update/Delete Ingredients
Create food Recipe
Manage Grocery Store
View User Details


User
Register and login
Choose Ingredients
Search All Cook Book
Create Recipe
Update/Delete Recipe
Give Review
Search Grocery store
My Profile

4.1.2 Sequence Diagram:

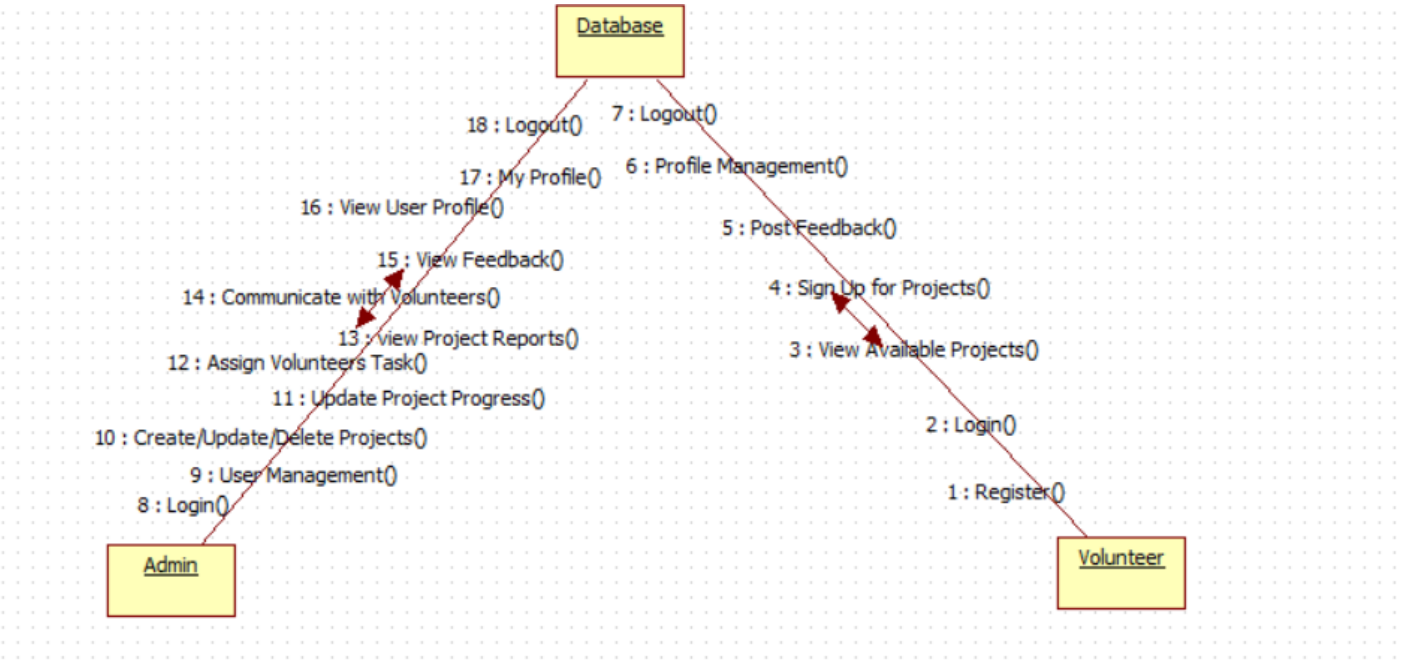
Sequence diagram and collaboration diagram are called INTERACTION DIAGRAMS. An interaction diagram shows an interaction, consisting of set of objects and their relationship including the messages that may be dispatched among them.

A sequence diagram is an introduction that empathizes the time ordering of messages. Graphically a sequence diagram is a table that shows objects arranged along the X-axis and messages ordered in increasing time along the Y-axis.

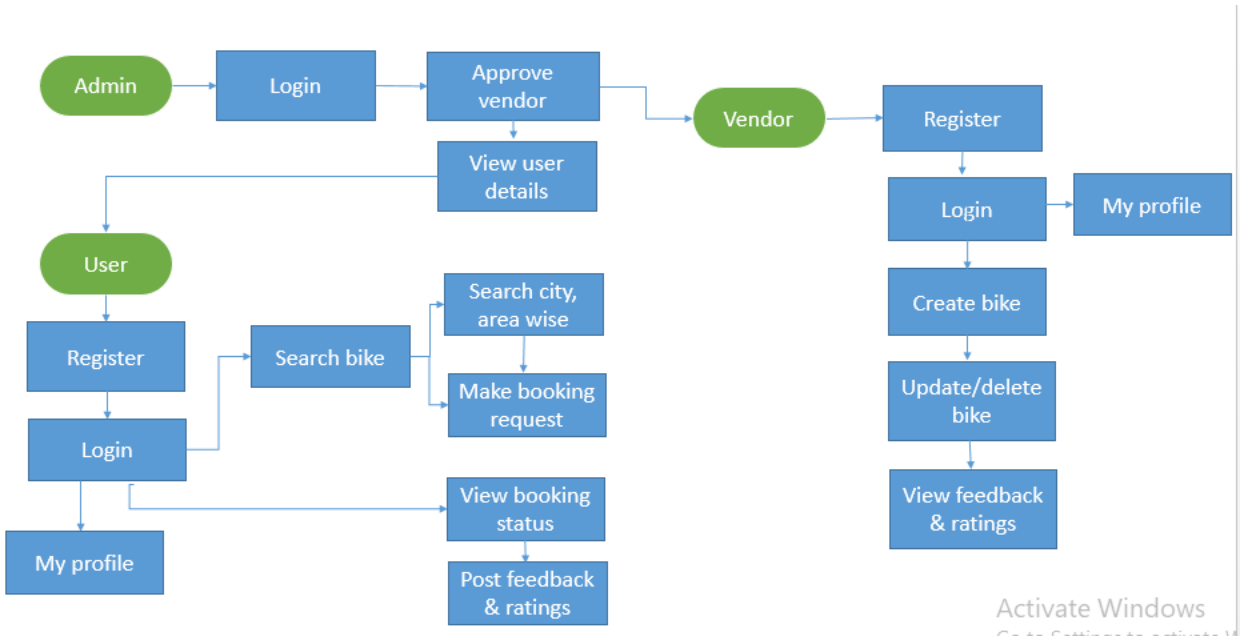


4.1.3 Collaboration Diagram:

A **collaboration diagram** is a type of visual presentation that shows how various software objects interact with each other within an overall IT architecture and how users can benefit from this **collaboration**. A **collaboration diagram** often comes in the form of a visual chart that resembles a flow chart.

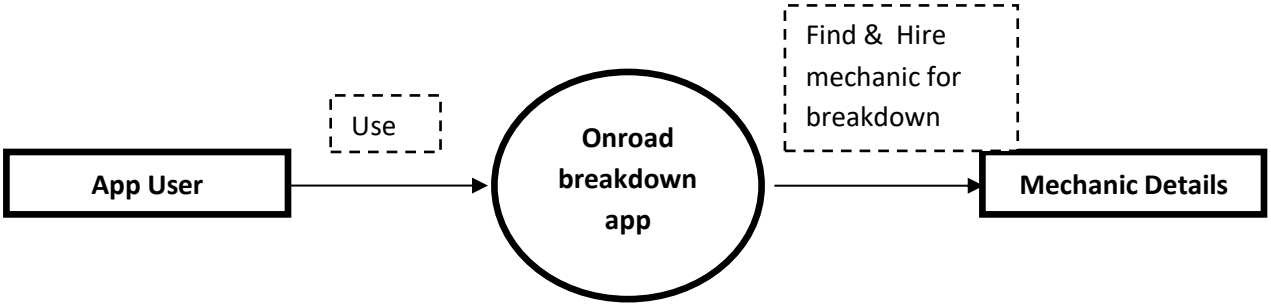


4.1.4 Architecture Design:

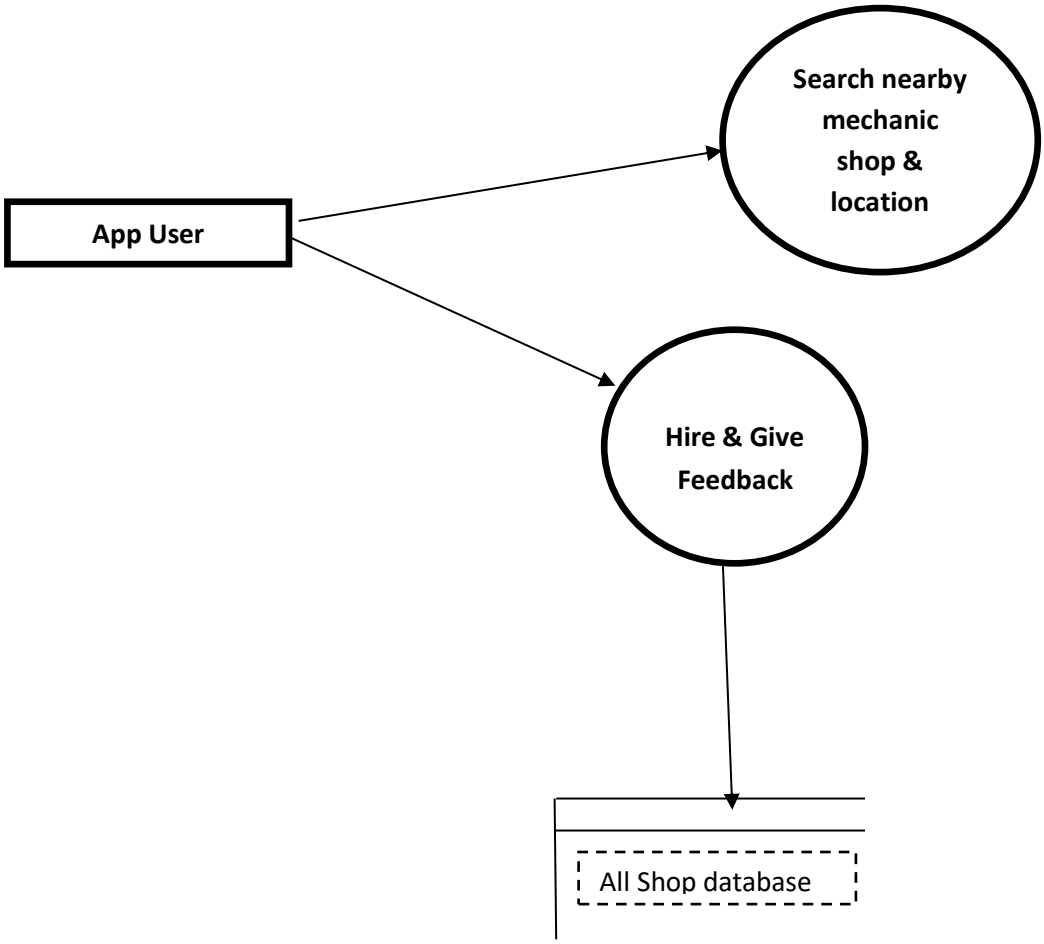


4.1.5 Data Flow Diagram:

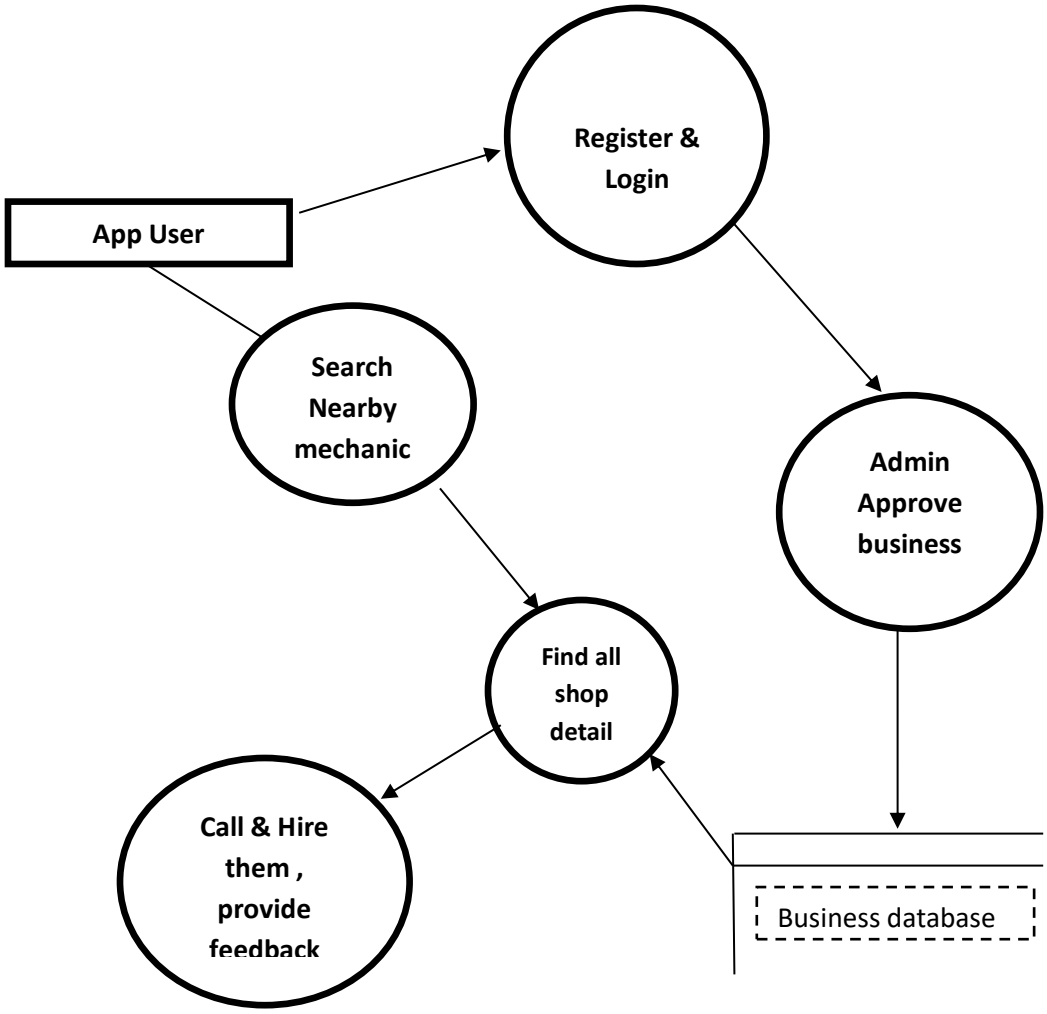
0-Level DFD



1-LEVEL DFD

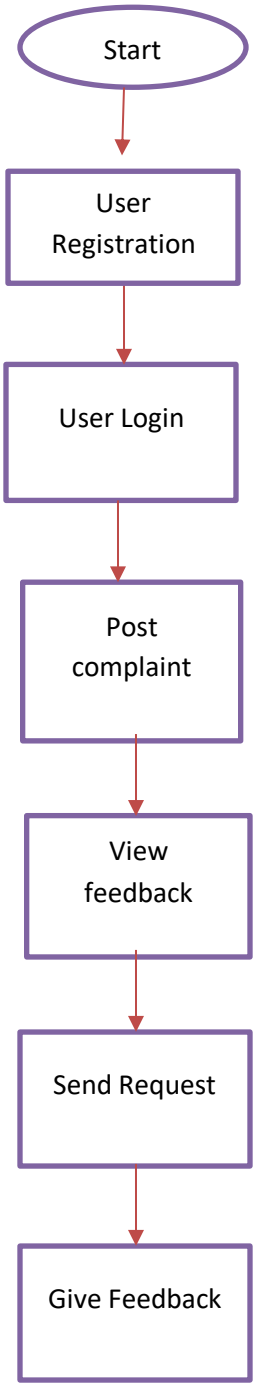

Search nearby mechanic shop & location
App User
Hire & Give Feedback
All Shop database

2-Level DFD

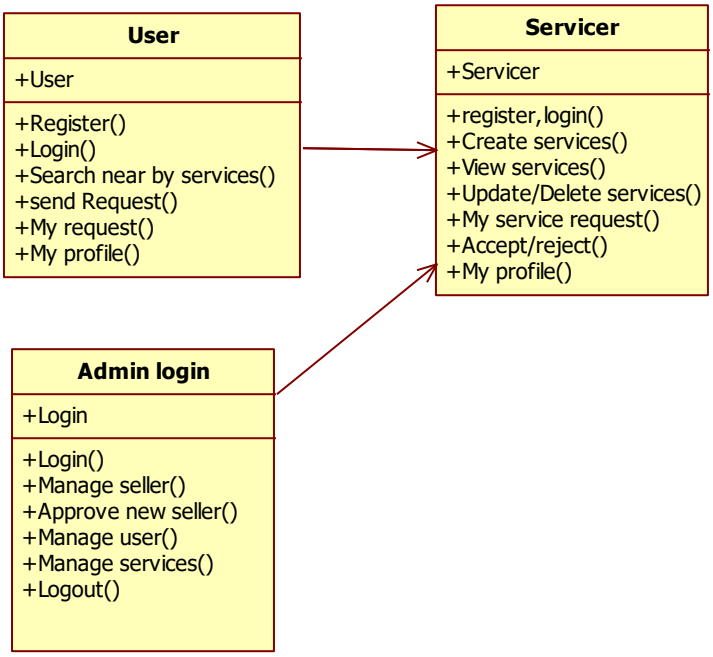

Register &
Login
App User
Search
Nearby
mechanic
Admin
Approve
business
Find all
shop
detail
Call & Hire
them ,
provide
feedback
Business database

3-Level DFD

User



4.1.6 Class Diagram:



4.1.7 Table Design:

Admin:

Field:	Email Id	Password
Type:	String	String
Required:	true	true

Owner Register & Login:

Field:	Object ID	Name	Email	Password	Mobile	isAdmin	Address	Question1	Question2
Type:	String	String	String	String	String	Boolean	String	String	String
Required:	true	true	true	true	true	false	true	true	true

User Register & Login:

Field:	Object ID	Name	Email Id	Password	Mobile	isAdmin	Address	Question1	Question2
Type:	String	String	String	String	String	Boolean	String	String	String
Required:	true	true	true	true	true	false	true	true	true

Event Details:

Field:	Object ID	Owner Email	name	description	mobile	location	Price Range	Google Geo Location
Type:	String	String	String	String	String	String	Number	Number
Required:	true	True	true	true	true	true	true	0

Field:	address	city	price	image	status	type
Type:	String	String	String	String	String	String
Required:	true	True	True		default	True

Ticket Details:

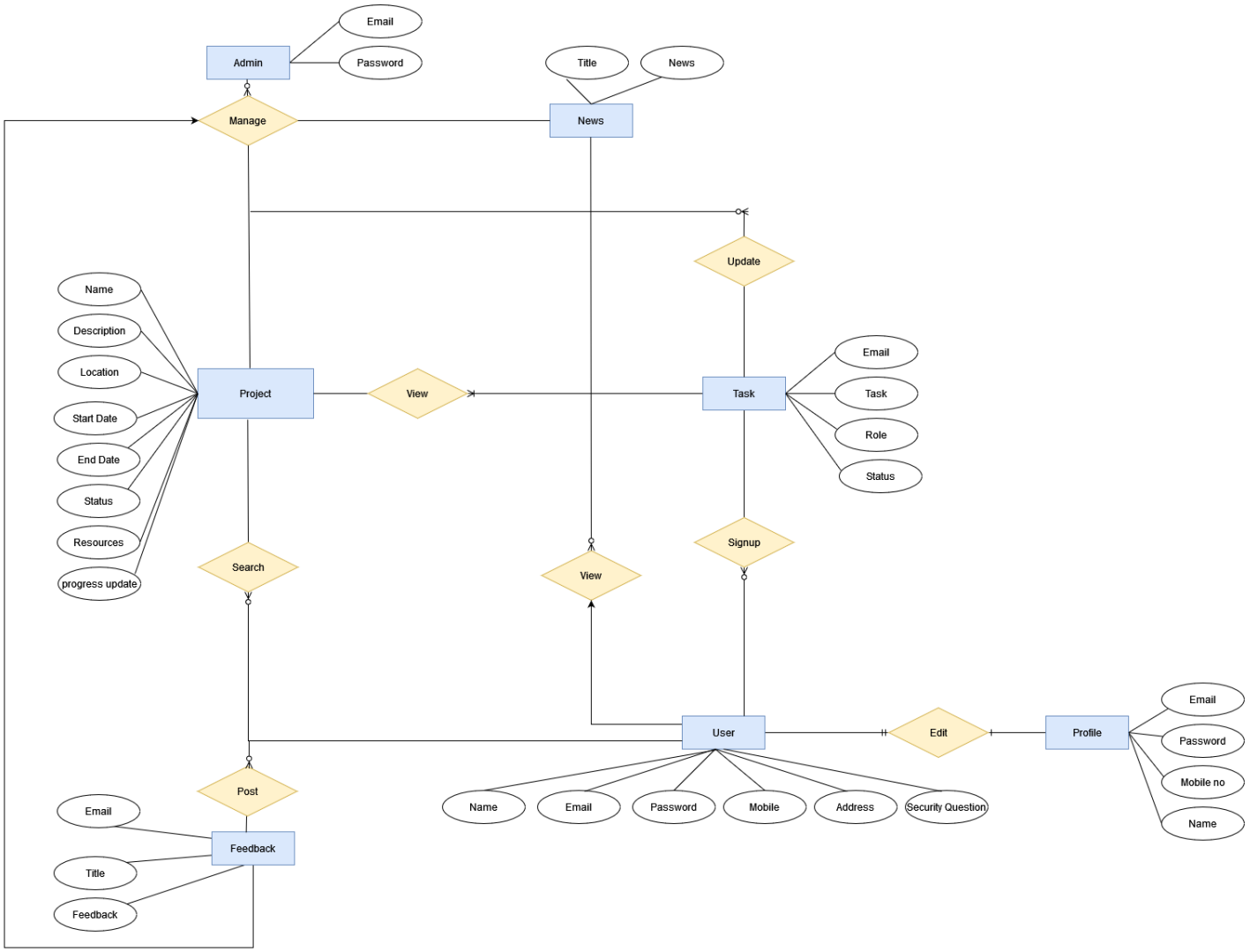
Field:	Object ID	name	type	Owner email	seats	price
Type:	String	String	String	String	String	String
Required:	true	true	true	true	true	true

Booking Details:

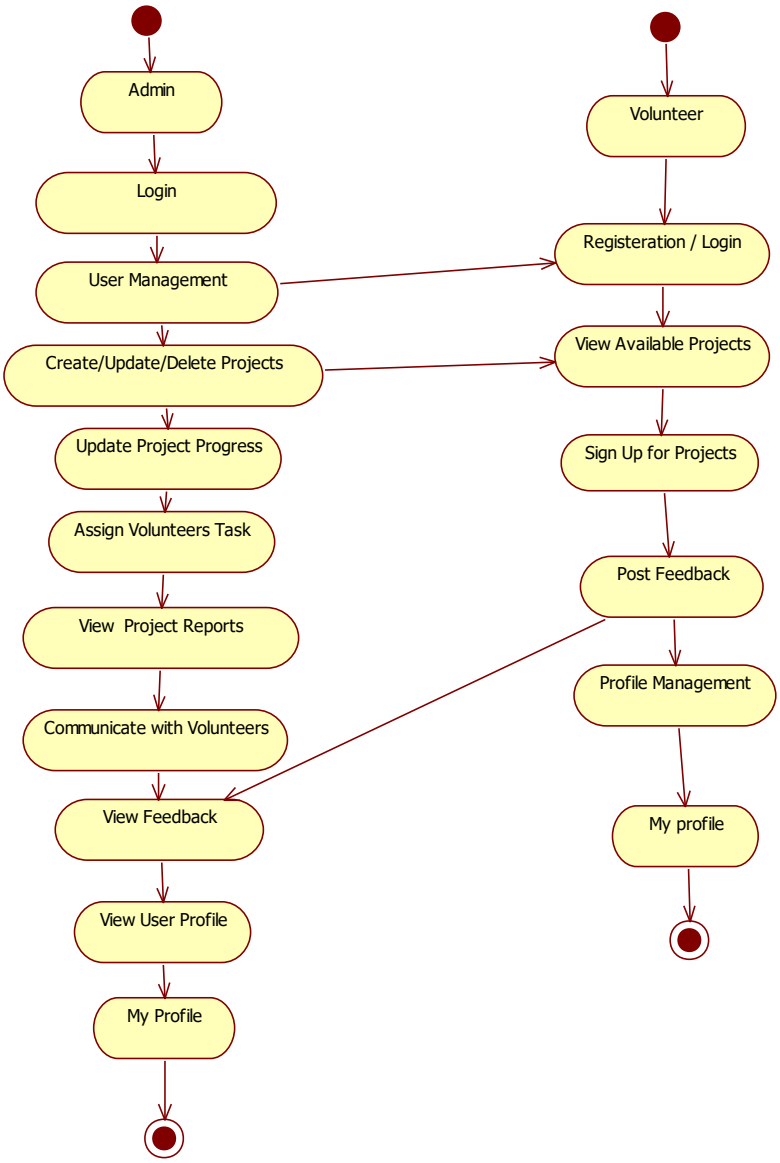
Field:	Object ID	Owner email	name	email	seats	type	price	Cus name	Person count
Type:	String	String	String	String	String	String	String	String	String
Required:	true	true	true	true	true	true	true	true	true

Field:	mobile	address	amount	ticketId	seats
Type:	String	String	String	String	String
Required:	true	true	true	true	true

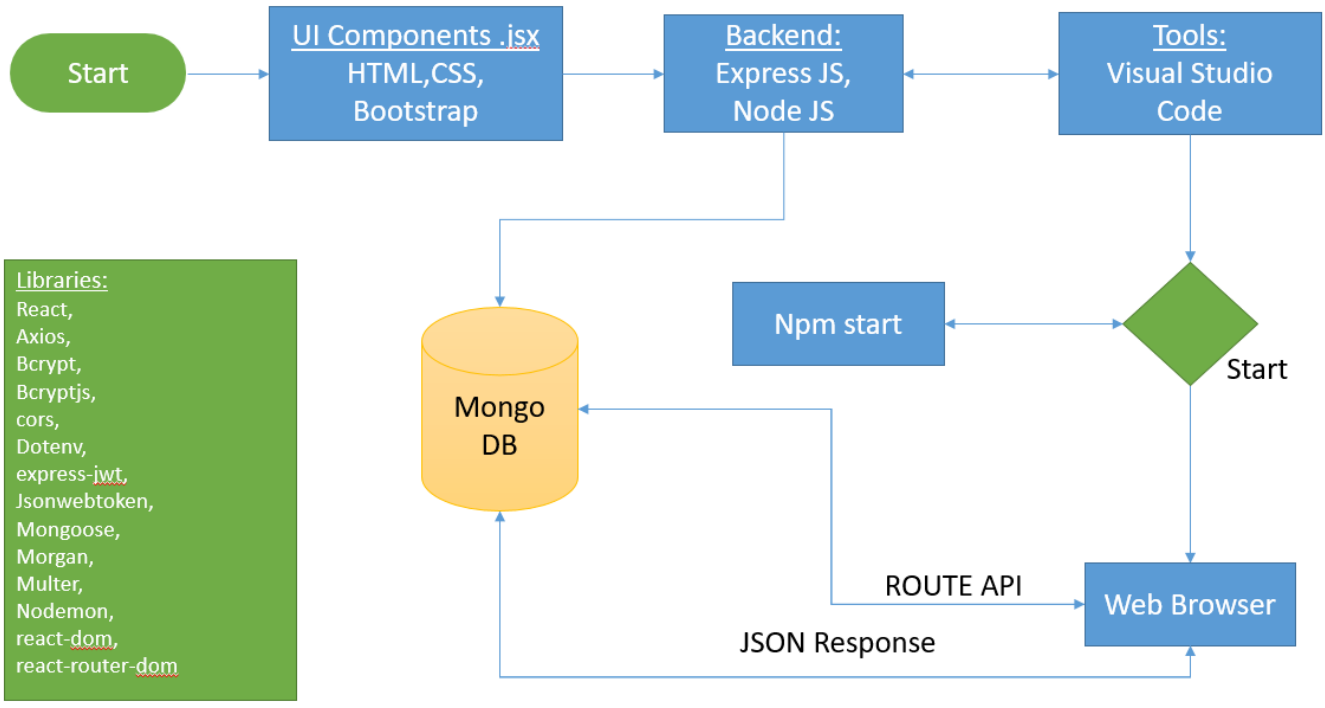
4.1.8 ER Diagram:



4.1.9 Activity Design:



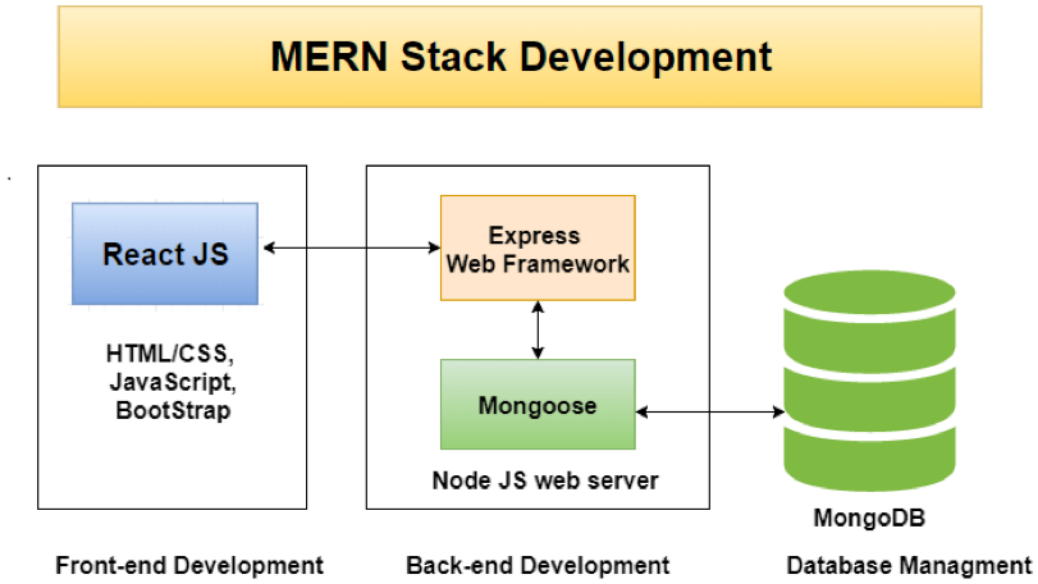
4.1.10 Work Flow Diagram:



CHAPTER 5 – OUTPUT SCREENSHOTS

CHAPTER 6 – IMPLEMENTATION DETAILS

6.1 MERN:



6.2 React.js:

Frontend web development using React.js involves creating user interfaces for web applications using the React JavaScript library. React.js, developed by Facebook, is a popular and powerful library for building interactive and dynamic user interfaces. Here's a detailed description of frontend web development with React.js:

1. Declarative UI:

React.js allows developers to build UI components using a declarative syntax, making it easier to understand and maintain the code. With React, you describe how

your UI should look based on the application state, and React takes care of updating the UI efficiently when the state changes.

2. Component-Based Architecture:

React encourages a component-based architecture where the UI is broken down into reusable and independent components. Each component represents a specific part of the UI, making it easier to manage and scale large applications. Components can be composed together to create complex UIs.

3.Virtual DOM:

React uses a virtual DOM (Document Object Model) to optimize rendering performance. Instead of directly manipulating the browser DOM, React creates a lightweight virtual representation of the DOM in memory. When the state of a component changes, React compares the virtual DOM with the previous version and only updates the necessary parts of the actual DOM, resulting in faster rendering.

4. JSX Syntax:

React uses JSX (JavaScript XML), an extension of JavaScript that allows developers to write HTML-like code within JavaScript. JSX makes it easier to create UI components by combining HTML structure with JavaScript logic. JSX code is transpiled to plain JavaScript before being executed in the browser.

5. Unidirectional Data Flow:

React follows a unidirectional data flow, also known as "props down, events up" architecture. Data flows from parent components to child components via props (properties), and child components can communicate with parent components by invoking callback functions passed down as props. This ensures predictable data flow and easier debugging.

6. Rich Ecosystem:

React.js has a vast ecosystem of libraries, tools, and community support, making it easier for developers to build and maintain web applications. Popular libraries and tools include React Router for routing, Redux for state management, Axios for HTTP requests, and Material-UI for UI components.

7. Server-Side Rendering (SSR) and Static Site Generation (SSG):

React can be used for server-side rendering (SSR) or static site generation (SSG) to improve performance and SEO (Search Engine Optimization). SSR renders React components on the server before sending HTML to the client, while SSG pre-renders pages at build time, reducing time-to-interactive and improving SEO.

8. Developer Tools:

React comes with developer tools that help developers inspect and debug React applications more efficiently. Tools like React DevTools provide real-time insights into component hierarchy, props, and state, allowing developers to identify performance bottlenecks and optimize their applications.

9. Cross-Platform Compatibility:

React can be used to build not only web applications but also mobile applications (React Native) and desktop applications (Electron). This allows developers to leverage their existing React skills and codebase to target multiple platforms, saving time and resources.

10. Continuous Improvement:

React.js is actively maintained and regularly updated by Facebook and the open-source community. New features, performance improvements, and bug fixes are regularly released, ensuring that React remains a cutting-edge library for frontend development.

11.Component-Based Architecture:

React.js follows a component-based architecture, where the UI is broken down into reusable components. This modular approach simplifies development, maintenance, and scalability of the application. With Bootstrap's predefined UI components (such as buttons, forms, navigation bars, and cards), developers can quickly create visually consistent and responsive components for their React.js applications.

12.Integration with Backend:

While React.js handles the frontend presentation and user interaction logic, it communicates with the backend (built using Node.js and Express.js) to fetch data, perform CRUD operations, and handle business logic. This communication typically occurs through HTTP requests (e.g., AJAX requests or fetch API) to RESTful APIs exposed by the backend server.

13. Development Server:

When running the frontend using the npm run dev method, React.js typically starts a development server that serves the frontend code to the client. This development server provides features such as hot module replacement (HMR), which allows for real-time updates to the application without the need for a full page refresh. It also includes tools for debugging, logging, and inspecting network requests, facilitating the development process.

Overall, frontend web development with React.js offers developers a powerful and efficient way to build modern, interactive, and scalable web applications. With its component-based architecture, virtual DOM, rich ecosystem, and continuous improvement, React.js continues to be a top choice for frontend development in the web development industry.

6.3 Cascading Style Sheets (CSS):

CSS is a style sheet language used for describing the presentation of a document written in a markup language. Although most often used to set the visual style of web pages and user interfaces written in HTML and XHTML, the language can be applied to any XML document, including plain XML, SVG andXUL, and is applicable to rendering in speech, or on other media. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites to create visually engaging webpages, user interfaces for web applications, and user interfaces for many mobile applications.

CSS is designed primarily to enable the separation of document content from document presentation, including aspects such as the layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content, such as semantically insignificant tables that were widely used to format pages before consistent CSS rendering was available in all major browsers. CSS makes it possible to separate presentation instructions from the HTML content in a separate file or style section of the HTML file. For each matching HTML element, it provides a list of formatting instructions.

For example, a CSS rule might specify that "all heading 1 elements should be bold", leaving pure semantic HTML markup that asserts "this text is a level 1 heading" without formatting code such as a<bold> tag indicating how such text should be displayed.

This separation of formatting and content makes it possible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (when read out by a speech-based browser orscreen reader) and on Braille-based, tactile devices. It can also be used to display the web page differently depending on the screen size or device on which it is being viewed. Although the author of a web page typically links to a CSS file within the markup file, readers can specify a different style sheet, such as a CSS file stored on their own computer, to override the one the author has specified. If the author or the reader did not link the document to a style sheet, the default style of the browser will be applied. Another advantage of CSS is that aesthetic changes to the graphic design of a document (or hundreds of documents) can be applied quickly and easily, by editing a few lines in one file, rather than by a laborious (and thus expensive) process of crawling over every document line by line, changing markup.

The CSS specification describes a priority scheme to determine which style rules apply if more than one rule matches against a particular element. In this so-called cascade, priorities (or weights) are calculated and assigned to rules, so that the results are predictable.

6.4 Bootstrap:

Bootstrap is a popular front-end framework that provides a comprehensive set of pre-styled components, utilities, and responsive layout grids to streamline the process of building web interfaces. When used in conjunction with React.js,

Bootstrap enhances the development experience by offering a wide range of UI components that can be easily integrated into React applications.

Bootstrap is used and integrated into React.js projects:

1. Responsive Grid System: Bootstrap provides a responsive grid system that allows developers to create flexible and mobile-friendly layouts. The grid system consists of rows and columns, enabling developers to organize content and adjust its layout based on different screen sizes. React developers can leverage Bootstrap's grid classes (`container`, `row`, `col`) to build responsive layouts that adapt to various devices and screen resolutions.

2. Pre-styled Components: Bootstrap offers a rich collection of pre-styled UI components, including buttons, forms, navigation bars, alerts, modals, and more. These components come with predefined styles and behaviors, allowing React developers to quickly integrate them into their applications without the need for extensive custom styling. By using Bootstrap components, developers can ensure consistency in design and enhance the overall user experience of their React applications.

3. Utility Classes: Bootstrap provides a set of utility classes that enable developers to apply common styles and effects to HTML elements easily. These utility classes allow for quick customization of typography, spacing, alignment, and other aspects of the UI. React developers can leverage Bootstrap's utility classes within their JSX code to achieve consistent styling and layout across different components and pages.

4. Integration with React Components: Bootstrap components can be seamlessly integrated into React.js applications by importing them as regular JavaScript modules or using third-party libraries like React-Bootstrap. React-Bootstrap is a popular library that provides React components that wrap around Bootstrap's JavaScript components, allowing for easy usage and integration within React applications. These React-specific components maintain the benefits of Bootstrap's styling and responsiveness while offering additional features tailored for React development.

5. Customization: While Bootstrap provides a comprehensive set of predefined styles and components, it also allows for customization to match specific design requirements. React developers can override Bootstrap's default styles or extend its components to create custom themes and UI elements that align with their project's visual identity and branding.

Bootstrap serves as a valuable toolkit for React.js developers, offering a wide range of pre-styled components, responsive layout grids, and utility classes to streamline the development of modern web applications. By integrating Bootstrap into React projects, developers can expedite the UI development process, ensure consistency in design, and deliver a visually appealing and user-friendly experience to their audience.

6.5 Node.js Server:

Node.js is a powerful JavaScript runtime built on Chrome's V8 JavaScript engine. It allows developers to run JavaScript code outside of a web browser, enabling server-side scripting for building scalable and high-performance web applications. Here's a detailed description of Node.js:

When you use `npm start` in a Node.js project, you're typically starting your application using a predefined script specified in your `package.json` file. Let's break down what happens in more detail:

1. npm (Node Package Manager):

- npm is the default package manager for Node.js. It's used to install, manage, and run dependencies for Node.js projects.

- `npm start` is a command provided by npm that is commonly used to start a Node.js application.

2. package.json:

- The `package.json` file is a metadata file for your Node.js project. It contains various project configurations, metadata, and scripts.

- One of the key sections in `package.json` is the `"scripts"` field, where you can define custom scripts to execute various tasks related to your project.

3. "start" script:

- In the `"scripts"` field of `package.json`, you can define a script named `"start"` that specifies the command to start your Node.js application.

- For example, `"start": "node server.js"` tells npm to execute `node server.js` when `npm start` is run.

4. Starting the Application:

- When you run `npm start`, npm looks for the `"start"` script in your `package.json` file.

- It then executes the command specified in the `"start"` script.

5. Starting a Backend Application:

- In the context of a backend application, the `"start"` script typically starts the server that handles incoming requests, processes data, and interacts with databases.

- The script might run a main file (e.g., `server.js`) that initializes and starts the backend server using a framework like Express.js or another HTTP server library.

- Depending on your project's setup, the script might also perform additional tasks like setting environment variables, initializing databases, or running other setup scripts.

6. Customizing the Start Script:

- You can customize the `"start"` script in your `package.json` file to fit the requirements of your Node.js application.

- For example, you might need to specify additional options or configurations when starting your application, such as setting the port number or specifying environment variables.

7. JavaScript Runtime:

- Node.js provides an environment for executing JavaScript code outside of a web browser. It uses the V8 JavaScript engine, which is the same engine that powers Google Chrome, to interpret and execute JavaScript code.

8. Asynchronous and Event-Driven:

- Node.js is designed with an event-driven, non-blocking I/O model. This means that it can handle many concurrent connections efficiently without getting blocked by I/O operations.

- Asynchronous programming is achieved through callback functions, promises, and async/await syntax, allowing developers to write code that can perform multiple tasks concurrently.

9. Single-Threaded, Event Loop Architecture:

- Node.js operates on a single-threaded event loop architecture, where all I/O operations are handled asynchronously.

- This architecture allows Node.js to handle thousands of concurrent connections efficiently, making it well-suited for building real-time applications, APIs, and microservices.

10. Package Management with npm:

- npm (Node Package Manager) is the default package manager for Node.js, used for installing, managing, and sharing JavaScript libraries and tools.

- The npm registry contains a vast ecosystem of open-source packages, allowing developers to easily integrate third-party libraries into their Node.js projects.

11. Built-in Modules:

- Node.js provides a rich set of built-in modules for performing various tasks, such as file system operations, networking, HTTP handling, and more.

- These core modules, such as `fs`, `http`, `util`, and `path`, enable developers to build robust server-side applications without relying heavily on external dependencies.

12. Scalability and Performance:

- Node.js is highly scalable and performant, thanks to its non-blocking I/O model and event loop architecture.

- It's well-suited for building real-time applications that require handling a large number of concurrent connections, such as chat applications, streaming services, and online gaming platforms.

13. Cross-Platform Compatibility:

- Node.js is cross-platform and runs on various operating systems, including Windows, macOS, and Linux.

- This allows developers to write and deploy Node.js applications across different environments without significant modifications.

14. Server-Side Development:

- Node.js is commonly used for server-side development, powering web servers, APIs, backend services, and microservices.

- It's often used in conjunction with web frameworks like Express.js to simplify the process of building web applications and APIs.

15. Community and Ecosystem:

- Node.js has a vibrant and active community of developers contributing to its ecosystem.

- The availability of open-source libraries, frameworks, and tools makes Node.js a popular choice for a wide range of applications, from simple web servers to complex enterprise systems.

Node.js revolutionized server-side JavaScript development by enabling developers to use the same language on both the client and server sides. Its asynchronous and event-driven nature, along with its performance and scalability, make it an excellent choice for building modern web applications and backend services.

In summary, `npm start` is a convenient way to start your Node.js application using a predefined script specified in your `package.json` file. It's commonly used in backend development to start the server and run the backend application. By customizing the `"start"` script, you can tailor the startup process to meet the specific needs of your project.

6.6 Express.js:

Express.js is a minimalistic and flexible web application framework for Node.js, designed to simplify the process of building robust and scalable web applications and APIs. Here's a detailed description of Express.js:

1. Web Application Framework:

- Express.js is a web application framework that runs on top of Node.js, providing a set of features and utilities for building web applications and APIs.

- It simplifies common web development tasks, such as routing, handling HTTP requests and responses, and serving static files.

2. Minimalistic and Unopinionated:

- Express.js is intentionally designed to be minimalistic and unopinionated, meaning it provides a basic structure and leaves many decisions up to the developer.

- It allows developers to choose from a wide range of middleware and plugins to customize their applications according to their specific requirements.

3. Middleware Architecture:

- Middleware functions are the heart of Express.js. They are functions that have access to the request and response objects (req and res) and can perform tasks such as modifying request and response objects, executing additional code, or terminating the request-response cycle.

- Middleware functions can be used for tasks such as logging, authentication, error handling, parsing request bodies, and more.

- Express.js allows developers to define middleware at the application level, router level, or route level, providing flexibility and granularity in handling requests.

4. Routing:

- Express.js provides a simple and intuitive way to define routes for handling different HTTP methods (GET, POST, PUT, DELETE, etc.) and URL patterns.

- Routes are defined using the `app.get()`, `app.post()`, `app.put()`, `app.delete()`, and similar methods, specifying the route path and the handler function to execute when the route is matched.

5. Request and Response Handling:

- Express.js provides a rich set of methods and properties on the request (`req`) and response (`res`) objects for handling HTTP requests and responses.

- Developers can access request parameters, query strings, headers, cookies, and request bodies, as well as send various types of responses (JSON, HTML, files, etc.) back to the client.

6. Template Engines:

- Although Express.js itself does not include a built-in template engine, it provides support for integrating with various template engines such as Pug (formerly Jade), EJS, Handlebars, and Mustache.

- Template engines allow developers to generate dynamic HTML content by combining static templates with dynamic data.

7. Error Handling:

- Express.js provides built-in middleware for error handling, allowing developers to define error-handling middleware functions to catch and handle errors that occur during the request-response cycle.

- Error-handling middleware functions typically have four parameters (err, req, res, next) and are called when an error is thrown or passed to the `next()` function.

8. Extensibility and Middleware Ecosystem:

- Express.js has a large and vibrant ecosystem of middleware and plugins developed by the community, which extends its functionality and provides solutions for common tasks and use cases.

- Developers can easily integrate third-party middleware and plugins into their Express.js applications using npm, the Node.js package manager.

Express.js is widely used for building various types of web applications and APIs, from simple websites and RESTful APIs to complex web services and

microservices. Its simplicity, flexibility, and robustness make it a popular choice among developers for Node.js web development.

6.7 Mongo DB:

MongoDB is a widely used NoSQL database management system known for its flexibility, scalability, and performance. Here's a detailed description of MongoDB:

1. NoSQL Database:

- MongoDB is a NoSQL (Not Only SQL) database, which means it does not use the traditional relational database model based on tables and rows. Instead, it stores data in a flexible, schema-less format, typically using JSON-like documents.

2. Document-Oriented:

- MongoDB is a document-oriented database, where data is stored in documents. Each document is a self-contained unit that represents a single entity or record and can contain nested structures, arrays, and other complex data types.

3. JSON-like Documents:

- Documents in MongoDB are stored in BSON (Binary JSON) format, which is a binary representation of JSON data. BSON allows for efficient storage and retrieval of data, as well as support for data types not available in JSON, such as binary data and date objects.

4. Collections and Documents:

- MongoDB organizes data into collections, which are analogous to tables in relational databases. Each collection contains multiple documents, and documents within a collection do not need to have the same structure or fields.

5. Schema Flexibility:

- MongoDB offers schema flexibility, allowing developers to store heterogeneous data in the same collection without a predefined schema. This flexibility is particularly useful in agile development environments where schema changes are frequent.

6. High Scalability:

- MongoDB is designed to scale horizontally across multiple servers, allowing for high availability and performance. It supports sharding, which involves distributing data across multiple servers (shards) to handle large volumes of data and high traffic loads.

7. Replication and High Availability:

- MongoDB supports replica sets, which are groups of MongoDB instances that maintain copies of the same data. Replica sets provide fault tolerance and high availability by automatically electing a primary node and failing over to secondary nodes in case of failures.

8. Rich Query Language:

- MongoDB provides a powerful and expressive query language for querying and manipulating data. It supports a wide range of query operators, aggregation pipelines, and indexing options to optimize query performance.

9. Aggregation Framework:

- MongoDB's Aggregation Framework allows developers to perform complex data aggregation and transformation operations, such as grouping, sorting, filtering, and joining, directly within the database.

10. Geospatial Indexing:

- MongoDB supports geospatial indexing and queries, allowing developers to store and query location-based data, such as points, lines, and polygons, with high precision and efficiency.

11. Full-Text Search:

- MongoDB provides full-text search capabilities through its text indexes and `$text` operator, allowing developers to perform efficient keyword searches on text data stored in MongoDB collections.

12. Community and Ecosystem:

- MongoDB has a large and active community of developers and contributors, as well as a rich ecosystem of tools, libraries, and resources to support development and operations with MongoDB.

MongoDB's combination of flexibility, scalability, performance, and rich feature set makes it well-suited for a wide range of use cases, including web applications, mobile apps, real-time analytics, content management systems, and more. It is often the preferred choice for modern, data-intensive applications that require fast iteration and scalability.

CHAPTER 7 – SYSTEM STUDY

7.1 FEASIBILITY STUDY

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

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

ECONOMICAL FEASIBILITY

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

TECHNICAL FEASIBILITY

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

SOCIAL FEASIBILITY

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

CHAPTER 8 – NON FUNCTIONAL REQUIREMENTS

8.1 Non Functional Requirements:

Non-functional requirements are the quality requirements that stipulate how well software does what it has to do. These are Quality attributes of any system; these can be seen at the execution of the system and they can also be the part of the system architecture.

8.2 Accuracy:

The system will be accurate and reliable based on the design architecture. If there is any problem in the accuracy then the system will provide alternative ways to solve the problem.

8.3 Usability:

The proposed system will be simple and easy to use by the users. The users will comfort in order to communicate with the system. The user will be provided with an easy interface of the system.

8.4 Accessibility:

The system will be accessible through internet and there should be no any known problem.

8.5 Performance:

The system performance will be at its best when performing the functionality of the system.

8.6 Reliability:

The proposed system will be reliable in all circumstances and if there is any problem that will be affectively handle in the design.

8.7 Security:

The proposed system will be highly secured; every user will be required registration and username/password to use the system. The system will do the proper authorization and authentication of the users based on their types and their requirements. The proposed system will be designed persistently to avoid any misuse of the application.

CHAPTER 9 – SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

Test Case:

Module	Scenario	Test Input	Expected Output	Actual Output
Event Owner	Registration	Valid details entered	Registration successful	Pass
Event Owner	Login	Valid email and password	Login successful	Pass
Event Owner	Create Event details	Valid Event details entered	Event details created successfully	Pass

Event Owner	Update/Delete Event	Event ID for update/delete provided	Event details updated/deleted successfully	Pass
Event Owner	Create Ticket Availability	Valid Ticket details entered	Ticket availability added successfully	Pass
Event Owner	Manage Ticket Availability	Ticket ID for update/delete provided	Ticket availability updated/deleted successfully	Pass
Event Owner	View Booking Details	Event ID for viewing booking details provided	Booking details displayed	Pass
Event Owner	My Profile	View and update profile information	Profile information updated successfully	Pass
User	Registration	Valid details entered	Registration successful	Pass
User	Login	Valid credentials entered	Login successful	Pass

User	Search Event Details	Area/City for searching provided	List of Events displayed	Pass
User	Check Ticket Availability	Event ID and date range provided	Available Tickets for specified date range displayed	Pass
User	Make Booking	Event ID, Ticket type, and date range provided	Booking confirmed successfully	Pass
User	My Booking	View list of bookings made	List of bookings displayed	Pass
User	My Profile	View and update profile information	Profile information updated successfully	Pass
Admin	Login	Valid credentials entered	Login successful	Pass
Admin	Approve Event	Event ID for approval provided	Event approved successfully	Pass
Admin	View User	View User Details	List of users displayed	Pass

Admin	Logout	Click logout button	Logout successful	

TYPES OF TESTS

Unit testing:

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing:

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test:

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing:

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing:

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

9.1 Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach:

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.

- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

9.3 Acceptance Testing:

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

CHAPTER 10 – CONCLUSION

The represents a significant advancement in the way ration card distribution is managed. By leveraging modern technology and integrating advanced features into its three core modules—Admin, Staff, and User—the system aims to address the challenges associated with traditional distribution methods and improve the overall efficiency and effectiveness of the process. With its focus on security, transparency, and user satisfaction, the Ration Card Queue Management System is poised to make a meaningful impact on the way ration cards are distributed. By providing administrators, staff, and users with the tools they need to manage and track the distribution process, the system not only enhances operational efficiency but also ensures that individuals receive the essential resources they need in a timely and organized manner.

CHAPTER 11 – FUTURE ENHANCEMENTS

1. **Enhanced Security Features**: Future iterations of online POCSO complaint systems should focus on advanced security protocols to protect sensitive data from breaches and unauthorized access. Implementing multi-factor authentication, end-to-end encryption, and regular security audits can bolster the system’s security.

2. **Integration of AI and Machine Learning**: Incorporating AI and machine learning algorithms can improve the system's ability to analyze and prioritize complaints based on urgency and severity. Predictive analytics could also assist in identifying patterns and potential threats, aiding in more proactive interventions.
3. **User Experience Improvements**: Continuous feedback from users (victims, guardians, and officials) should guide enhancements in the user interface and user experience design. Features such as multilingual support, accessibility options, and simplified navigation will make the system more inclusive and user-friendly.
4. **Real-Time Communication Channels**: Integrating real-time communication tools, such as live chat or video calls, can facilitate better interaction between complainants and support personnel. This will help address queries promptly and provide immediate assistance during the reporting process.
5. **Expanded Reporting Mechanisms**: Incorporating additional reporting channels, such as mobile apps and text message reporting, can further increase accessibility and ease of use. This will cater to different user preferences and ensure that help is available in various formats.
6. **Enhanced Data Analytics**: Developing advanced data analytics tools will allow for deeper insights into complaint trends, case outcomes, and system performance. This will help in identifying systemic issues and refining strategies for child protection.
7. **Cross-Agency Collaboration**: Future systems should focus on facilitating better collaboration between various agencies involved in child protection. Integration with national and international databases can enhance information sharing and coordination.

8. **Regular System Updates and Maintenance**: Establishing a routine for regular updates and maintenance will ensure that the system remains current with technological advancements and legal requirements. This will also help in addressing any emerging security threats.
9. **User Training and Awareness Programs**: Implementing training programs for users and administrators will ensure that they are well-versed in the system's features and best practices for handling complaints. Awareness campaigns can also educate the public about the availability and benefits of the online complaint system.
10. **Feedback Mechanisms**: Incorporating mechanisms for ongoing feedback from users can provide valuable insights into areas for improvement. Regularly evaluating user satisfaction and system effectiveness will help in making iterative enhancements.

By focusing on these enhancements, online POCSO complaint systems can continue to evolve and better serve their intended purpose of protecting children and ensuring justice.

CHAPTER 12 – REFERENCES

- A. Pal and S. Mitra, "Mobile-Based Volunteer Management System for Nonprofit Organizations," *IEEE Access*, vol. 8, pp. 45530-45545, 2020.
- P. Gupta, A. Singh, and S. Kumar, "Resource Management in Non-Governmental Organizations Using Cloud Computing," *IEEE Transactions on Cloud Computing*, vol. 9, no. 2, pp. 301-312, Apr. 2021.

- L. W. Barclay and C. J. Wright, "Designing Collaborative Platforms for Volunteer Engagement: A Case Study of NGO Digitalization," *IEEE Transactions on Professional Communication*, vol. 65, no. 1, pp. 18-28, Mar. 2022.

- J. Zhang, M. Zhang, and T. Wang, "Project Tracking and Volunteer Coordination Through Mobile Apps for NGOs," *IEEE Transactions on Mobile Computing*, vol. 15, no. 5, pp. 1158-1173, May 2020.

- R. Jones and A. Thomas, "Nonprofit Data Management Systems: Enhancing Efficiency through Technology," *IEEE Transactions on Human-Machine Systems*, vol. 50, no. 4, pp. 1251-1263, Aug. 2021.

- K. Tan and D. Tan, "The Role of Artificial Intelligence in NGO Operations: An Exploratory Study," *IEEE Transactions on Artificial Intelligence*, vol. 3, no. 2, pp. 176-185, Jun. 2022.

- S. Singh, "Integrating Communication Technologies in Nonprofit Volunteer Management Systems," *IEEE Communications Magazine*, vol. 58, no. 9, pp. 85-90, Sept. 2020.

- M. Chen, J. Wu, and L. Cao, "Automating Project Reporting and Feedback Collection in NGOs Using Mobile Apps," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6203-6211, Aug. 2019.

- E. Fernandes, R. A. Barreto, and A. Costa, "Impact of Technology Adoption in NGOs: A Comprehensive Review," *IEEE Transactions on Engineering Management*, vol. 67, no. 3, pp. 542-554, Sept. 2020.

- R. Brown, T. Edwards, and L. K. Carlson, "Enhancing Volunteer Engagement with Gamification Techniques," *IEEE Transactions on Games*, vol. 13, no. 4, pp. 384-396, Dec. 2021.

CHAPTER 13 – CODING

Backend:

Models

```
const mongoose = require('mongoose');


// name, email, password, phone, city, question1, question2

const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    email: {
        type: String,
        required: true,
    },
    passwordHash: {
        type: String,
        required: true,
    },
    phone: {
        type: String,
        required: true,
    },
    isAdmin: {
        type: Boolean,
        default: false,
    },
    city: {
        type: String,
        default: ''
    },
    question1: {
        type: String,
        required: true,
    },
    question2: {
        type: String,
        required: true,
    },
```

```
    dateCreated: {
        type: Date,
        default: Date.now
    },

});

userSchema.virtual('id').get(function () {
    return this._id.toHexString();
});

userSchema.set('toJSON', {
    virtuals: true,
});

exports.User = mongoose.model('User', userSchema);
exports.userSchema = userSchema;
```

routes

```
const {User} = require('../models/user');
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

router.get(`/`, async (req, res) =>{
    const userList = await User.find().select('-passwordHash');
    if(!userList) {
        res.status(500).json({success: false})
    }
    res.send(userList);
})



router.get(`/:id`, async (req, res) =>{
    const userList = await User.findById(req.params.id);
    if(!userList) {
        res.status(500).json({success: false})
    }
    res.send(userList);
})
```

```
router.post(`/`, async (req, res) =>{
    let user = new User({
        name: req.body.name,
        email: req.body.email,
        passwordHash: bcrypt.hashSync(req.body.passwordHash, 10),
        phone: req.body.phone,
        isAdmin: req.body.isAdmin,
        city: req.body.city,
        question1: req.body.question1,
        question2: req.body.question2,
    })
    user = await user.save();
    if(!user)
    return res.status(500).send('The product cannot be created')

    res.send(user);
})



router.post('/login', async (req,res) => {
    const user = await User.findOne({email: req.body.email})
    const secret = process.env.secret;
    if(!user) {
        return res.status(400).send('The user not found');
    }

    if(user && bcrypt.compareSync(req.body.password, user.passwordHash)) {
        const token = jwt.sign(
            {
                useremail: user.email,
                isAdmin: user.isAdmin
            },
            secret,
            {expiresIn : '1d'}
        )

        res.status(200).send({user: user.email , token: token})
    } else {
       res.status(400).send('password is wrong!');
    }


})
```

```
router.put('/:id',async (req, res)=> {

    const userExist = await User.findById(req.params.id);
    let newPassword
    if(req.body.password) {
        newPassword = bcrypt.hashSync(req.body.password, 10)
    } else {
        newPassword = userExist.passwordHash;
    }

    const user = await User.findByIdAndUpdate(
        req.params.id,
        {
            name: req.body.name,
            email: req.body.email,
            passwordHash: newPassword,
            phone: req.body.phone,
            city: req.body.city,


        },
        { new: true}
    )

    if(!user)
    return res.status(400).send('the user cannot be created!')

    res.send(user);
})


router.post('/reset_password', async (req, res) => {
    const { question1, question2, newPassword } = req.body;

    try {
      // Find the user by email
      const user = await User.findOne({ email: req.body.email });

      // Check if the user exists
      if (!user) {
        return res.status(404).json({ error: 'User not found' });
      }
```

```
      // Check if security questions match
      if (user.question1 !== question1 ||  user.question2 !== question2) {
        return res.status(400).json({ error: 'Security questions do not match'
});
      }
            // Hash the new password

            const hashedPassword = bcrypt.hashSync(newPassword, 10);

            // Update the user's password
            user.passwordHash = hashedPassword;

            // Save the updated user
            await user.save();
            res.status(200).json({ message: 'Password reset successful' });
            //res.send(user);


    } catch (error) {
      console.error('Error resetting password:', error);
      res.status(500).json({ error: 'Internal server error' });
    }
  });



module.exports =router;
```

Client:

```
import React, { useState } from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';
import { useCookies } from 'react-cookie';
import "./css/bootstrap.min.css";
import "./css/owl.carousel.min.css";
import "./css/font-awesome.min.css";
import "./css/animate.css";
import "./css/font-awesome.min.css";
import "./css/lineicons.min.css";
import "./css/magnific-popup.css";
import "./css/style.css";
```

```
import imgfolder from "./img/core-img/logo-white.png";

const UserLogin = () => {

  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [cookies, setCookie] = useCookies(['email']); // Use cookies to store the
email
  const [error, setError] = useState('');
  const token = localStorage.getItem('token');


  const handleEmailChange = (e) => {
    setEmail(e.target.value);
  };

  const handlePasswordChange = (e) => {
    setPassword(e.target.value);
  };

  const handleLogin = async (e) => {
    e.preventDefault();

    try {
      const response = await
axios.post('http://localhost:4000/api/v1/user/login', {
        email: email,
        password: password,
      });

      // Check if the login was successful
      if (response.status === 200) {
          // Store the JWT token in localStorage
          localStorage.setItem('token', response.data.token);

          // Include the token in the x-auth-token header for subsequent requests
          axios.defaults.headers.common['x-auth-token'] = response.data.token;


        // Redirect to the home page or perform other actions
        alert('Login Successful!');
        window.location.href = "user_home";
        console.log('Login successful!');

        setCookie('email', email, { path: '/' , sameSite: 'strict' });
```

```
        setError('');
        // You can handle the token and user details here, such as storing them
in state or cookies
      } else {
        setError('Login failed. Please check your credentials.');
        alert('Login Unsuccessful!');

      }
    } catch (error) {
      console.error('Error during login:', error.message);
      setError('Internal Server Error. Please try again later.');
      alert('Login Unsuccessful!');
    }
  };


  return (
    <div>
      <title>Complaint management app</title>

        <div className="login-wrapper d-flex align-items-center justify-content-
center text-center">
        <div className="background-shape"></div>
             <div className="container">
                <div className="row justify-content-center">
                <div className="col-12 col-sm-9 col-md-7 col-lg-6 col-xl-5">
                <img className="big-logo" src={imgfolder} alt="" ></img>

                <div className="row justify-content-center"><b>User
Login</b></div>

                        <div className="register-form mt-5 px-4">
                        <form  onSubmit={handleLogin}>

                        <div className="form-group text-start mb-
4"><span>Email</span>
                            <label htmlFor="username"><i className="lni lni-
user"></i></label>
                            <input className="form-control" name="email"
id="email" value={email}    onChange={handleEmailChange}  type="text"
placeholder="info@example.com"/>
                            </div>
```

```
                            <div className="form-group text-start mb-
4"><span>Password</span>
                            <label htmlFor="password"><i className="lni lni-
lock"></i></label>
                            <input className="form-control"  name="password"
id="password" value={password} onChange={handlePasswordChange}
type="password"   placeholder="password"/>
                            </div>
                            <button className="btn btn-warning btn-lg w-
100"type="submit">Log In</button>
                        </form>
                        </div>
                        <div className="login-meta-data"><a className="forgot-
password d-block mt-3 mb-1" href="reset_password">
                        Forgot Password?</a>
                        <p className="mb-0">Didn't have an account?<Link
to="/user_register" className="ms-1" >Register
Now</Link></p>
                        </div>

                </div>
                </div>
                </div>
            </div>
        </div>


  )
}

export default UserLogin
```