|  |  |
| --- | --- |
| BACHELOR OF COMPUTER SCIENCE  ENGINEERING & TECHNOLOGY    SCHOOL OF COMPUTER SCIENCE AND ENGINEERING  GALGOTIAS UNIVERSITY, GREATER NOIDA  UTTAR PRADESH  **Project Report**  Web Technology  Course Code: R1UC626C   |  | | --- | | Name: Archit Rathor  Admission No: 22SCSE1011321  Semester : VI | |

**1. Abstract**

This project details the development of the "Builders Inventory Management System," a web-based application designed to streamline and manage the inventory of a builder's office. The system provides an intuitive interface for authorized users to add, update, and delete property records, including details such as Sector, Pocket, Site No., Size, Demand, Description, and Builder/Seller. The application ensures role-based access where only admins can modify data, while general users can view property listings. Built using HTML, CSS, JavaScript for the frontend, and Node.js with MongoDB on the backend, the system focuses on data accuracy, user accessibility, and security.

**2. Introduction**

Real estate firms and builders often face challenges in organizing and tracking property listings across various sectors and pockets. Manual processes or outdated systems can lead to inefficiencies and errors. The "Builders Inventory Management System" addresses these issues by offering a centralized digital solution to manage inventory records.

This web application is designed with user roles in mind: administrators and authorized personnel can add or remove entries, while prospective buyers or agents can browse listings. This ensures security, data integrity, and controlled access. Technologies used include Node.js for backend operations, MongoDB for a scalable and flexible database, and modern frontend tools for a responsive user experience.

**3. Problem Statement**

Manual inventory management for builders is prone to errors, lacks real-time accessibility, and is inefficient for tracking multiple properties. There is a need for a secure and scalable digital platform where property data can be managed systematically with role-based access.

The "Builders Inventory Management System" application aims to solve these problems by providing:

* **Add Properties**
* **Edit Properties**
* **Filter Properties**
* **Role Based System**

By addressing these limitations, "Builders Inventory Management System" seeks to empower users with a proactive and responsive tool that can be crucial for builders.

**4. Objectives**

The primary objectives of the "Builders Inventory Management System" project are to:

1. Create a user-friendly web interface to view, add, and manage inventory.
2. Implement role-based access (admin vs. viewer).
3. Store property data in a structured format including fields like Sector, Pocket, Site No., Size, Demand, Description, and Seller.
4. Use MongoDB to allow flexible and scalable data storage.
5. Ensure data security, accessibility, and validation.
6. **Implement Secure User Authentication:** Use Flask (Python) and JavaScript to build a secure user authentication system.
7. **Optimize Real-time Communication:** Employ RESTful APIs to ensure efficient and low-latency communication.
8. **Utilize Scalable Storage:** Use MongoDB for storing application data, ensuring scalability and high availability.

**5. Literature Survey**

Studies and existing platforms highlight the importance of digitizing real estate data management. Cloud-based applications offer significant improvements in efficiency and accuracy. Technologies such as NoSQL databases and RESTful APIs are preferred for their performance and scalability in such systems.

**5.1 Inventory Management Applications:**

A review of existing inventory systems reveals a focus on tracking assets, reducing manual errors, and improving accessibility. However, many real estate platforms lack real-time editing capabilities, role-based access control, and seamless integration with scalable databases. Studies emphasize the importance of having a central dashboard where admins can easily add, modify, or remove records, ensuring smooth operations in property-based businesses**.**

**5.2 Web-Based CRUD Systems:**

Create, Read, Update, and Delete (CRUD) operations are the foundation of inventory applications. Efficient implementation of these functions using web technologies improves usability and data accuracy. Research supports the use of RESTful APIs and modular backend frameworks (like Express.js) to keep operations fast, secure, and maintainable.

**5.3 Real-time Communication with RESTful APIs:**

RESTful APIs are widely adopted for building scalable and maintainable backend services. In inventory systems, they allow seamless communication between the client-side interface and the server/database. Studies highlight how asynchronous processing and proper routing enhance system performance and reduce latency during operations like record fetching or updates**.**

**5.4 NoSQL Databases (MongoDB):**

NoSQL databases such as MongoDB are ideal for applications requiring flexible data structures. For builder inventories, where property fields may vary or expand over time, MongoDB’s document-oriented model ensures efficient and scalable data management. Research also points out its high uptime, indexing support for quick searches, and ease of integration with Node.js applications.

**5.5 Secure User Authentication:**

Role-based access control is essential in applications handling sensitive data. Secure authentication mechanisms using session tokens or JWT (JSON Web Tokens) prevent unauthorized access to admin features. Literature on web security recommends hashing passwords (e.g., using bcrypt), validating inputs, and employing middleware for route protection in Node.js applications.

This project synthesizes these technologies and methods to create a secure, efficient, and user-friendly inventory management solution tailored for the real estate domain.  
  
  
  
**6. System Requirements**

This section outlines the system requirements for the "Builders Inventory Management System" application, categorized for clarity.

**6.1 Functional Requirements**

*(The functional requirements remain the same as previously provided.)*

* FR01: Admin Login & Authentication
* FR02: Add, Edit, Delete Inventory
* FR03: View Inventory List
* FR04: Search & Filter Properties

**6.2 Non-Functional Requirements**

*(The non-functional requirements remain the same as previously provided.)*

* NFR01: Performance
* NFR02: Security
* NFR03: Reliability
* NFR04: Usability
* NFR05: Scalability

**6.3 System Requirements (User Perspective)**

This section outlines the minimum and recommended system requirements for users to effectively access and interact with the "Builders Inventory Management System" application.

**6.3.1 Minimum Requirements:**

* **Operating System:** Windows 7 or later, macOS 10.10 or later, modern Linux distributions, Android 5.0 or later, iOS 10 or later.
* **Mobile Operating System:** Android 5.0 or later, iOS 10 or later
* **Web Browser (If a web interface is part of the system):**
  + Google Chrome (latest version)
  + Mozilla Firefox (latest version)
  + Safari 10 or later
  + Microsoft Edge (latest version)
* **Internet Connection:** Broadband internet connection (for initial download and map data).
* **GPS:** Enabled GPS functionality on the mobile device.

**6.3.2 Recommended Requirements:**

* **Operating System:** Windows 10 or later, macOS 10.15 or later, modern Linux distributions, Android 8.0 or later, iOS 13 or later.
* **Mobile Operating System:** Android 8.0 or later, iOS 13 or later
* **Web Browser (If a web interface is part of the system):**
  + Google Chrome (latest version)
  + Mozilla Firefox (latest version)
  + Safari (latest version)
  + Microsoft Edge (latest version)
* **Internet Connection:** High-speed broadband internet connection (for real-time updates and map loading).
* **GPS:** High-accuracy GPS functionality.
* **Processor:** Multi-core processor on the mobile device for smooth performance.
* **RAM:** 2GB or more RAM on the mobile device.

**6.3.3 Software Requirements (User Side):**

* A modern mobile operating system (as listed above).
* A modern web browser (as listed above), if accessing a web interface, with JavaScript enabled.
* The "Builders Inventory Management System" mobile application installed from the appropriate app store (e.g., Google Play Store, Apple App Store).

**6. System Requirements**

The "Builders Inventory Management System" application's system requirements are crucial for its effective operation and are divided into functional and non-functional categories.

**6.1 Functional Requirements**

Functional requirements define the specific actions and feature the application must perform.

* **FR01: SOS Trigger Activation:**
  + The application shall provide multiple SOS trigger mechanisms:
    - Shake trigger: Initiating an SOS alert by shaking the device with a defined intensity.
    - Voice trigger: Activating an SOS alert through a specific voice command.
  + The application shall provide user feedback upon SOS trigger activation (e.g., visual and auditory confirmation).
* **FR02: Real-time Safety Alerts:**
  + The application shall receive and display real-time safety alerts based on the user's location.
  + Alerts shall include information about the type of threat, location, and severity (if available).
  + Users shall be able to configure alert preferences (e.g., types of alerts, alert radius).
* **FR03: Location Tracking and Sharing:**
  + The application shall accurately track the user's location using GPS and other location services.
  + Users shall be able to select trusted contacts to share their location with during emergencies or designated periods.
  + Location sharing shall be secure and controlled by the user.
* **FR04: Safe Route Suggestions:**
  + The application shall suggest safer routes to a user's destination, considering factors such as:
    - Crime rates in the area.
    - Reported incidents.
    - Well-lit areas (if data is available).
  + Users shall be able to compare suggested routes with standard routes.
* **FR05: Emergency Service Connection:**
  + The application shall facilitate direct and rapid communication with local emergency services.
  + This may include:
    - Direct dialing of emergency numbers.
    - Sending pre-defined messages with location information.
* **FR06: User Authentication:**
  + The application shall provide secure user authentication using Flask (Python) and JavaScript.
  + Users shall be able to register, log in, and manage their accounts.
  + Authentication shall protect user data and privacy.

**6.2 Non-Functional Requirements**

Non-functional requirements define the quality attributes of the application.

* **NFR01: Performance:**
  + The application shall have minimal latency for SOS triggers and real-time alerts.
  + Location updates shall be frequent and accurate.
  + The application shall be responsive and efficient in resource usage.
* **NFR02: Security:**
  + User data, including location information and personal details, shall be protected with strong encryption and security measures.
  + Authentication shall be robust and prevent unauthorized access.
  + The application shall be resistant to common security vulnerabilities.
* **NFR03: Reliability:**
  + The application shall function reliably in various conditions and network environments.
  + The application shall have minimal downtime and error rates.
  + The MongoDB database shall provide 99.9% uptime.
* **NFR04: Usability:**
  + The application shall be intuitive and easy to use, even in stressful situations.
  + The UI shall be clear, concise, and accessible.
  + SOS triggers shall be easily activated with minimal effort.
* **NFR05: Scalability:**
  + The application's architecture shall be scalable to accommodate a growing number of users and data.
  + The MongoDB database shall provide scalable storage to handle increasing data volumes.

**7. System Design**

The system design defines the architecture, components, and interactions of the "Builders Inventory Management System" application.

**7.1 System Architecture**

The "Builders Inventory Management System" application follows a multi-tiered architecture:

* **Presentation Tier (Client-Side):**
  + The mobile web application (iOS and/or Android) handles user interface and user interactions.
  + JavaScript is used for client-side logic and UI elements.
* **Application Tier (Server-Side):**
  + Node.js handles user authentication, API requests, and business logic.
  + RESTful APIs facilitate communication between the mobile app and the database.
* **Data Tier:**
  + MongoDB stores user data, safety alerts, location information, and other application data.

**(Imagine a block diagram here showing the three tiers: Mobile App (Presentation) communicating with Flask API (Application) which interacts with MongoDB (Data). Include arrows showing the flow of requests and data.)**

**7.2 Data Flow Diagram (DFD)**

A DFD illustrates how data flows through the system. A simplified DFD for the "Builders Inventory Management System" application can be represented as follows:

+----------+ +--------------------------+ +------------------------+

| User |----->| App |----->| Flask API |----->| MongoDB |

+----------+ | (Mobile Application) | | Node.js| | (Database) |

+--------------------------+ +------------------------+ +------------------------+

| |

+---------------------------------------+

|

+---------------------------------------+

| External Services (e.g., GPS, |

| Emergency Services API) |

+---------------------------------------+

* **Explanation:**
  + The "User" interacts with the "Builders Inventory Management App" on their mobile device.
  + The app communicates with the "Node.js" for authentication, data requests, and sending alerts.
  + The " Node.js" processes requests and retrieves/stores data in "MongoDB."
  + The system also interacts with "External Services" like GPS for location tracking and potentially APIs for emergency services.

**(Imagine a more detailed DFD with multiple levels, showing the flow of data for specific features like SOS activation, location sharing, and receiving safety alerts.)**

**7.3 User Workflow**

The user workflow describes the typical sequence of actions a user takes within the application.

1. **User Opens the App:** The user launches the "Builders Inventory Management System" application on their mobile device.
2. **User Authenticates:** The user logs in securely using their credentials (or registers if it's their first time).
3. **Add Properties**
4. **Edit Properties**
5. **Update Properties**
6. **App Receives Safety Alerts:** The app displays real-time safety alerts based on the user's location.
7. **User Shares Location (If Needed):** The user may choose to share their location.
8. **User Requests Safe Route (Optional):** The user can request the app to suggest a safer route to their destination.

**(Imagine a flowchart here illustrating the user workflow, showing the sequence of steps, decision points, and actions taken by the user and the application.)**  
  
**8. Implementation**

The implementation phase involves translating the system design into a working application. For "Builders Inventory Management System," this includes developing the mobile application, the backend API, and setting up the database.

**8.1 Mobile Application Development**

* **Platform:** The application can be developed for both iOS and Android platforms. Frameworks like React Native or Flutter can be used for cross-platform development, allowing for a single codebase to target both platforms.
* **Languages:** JavaScript is used for the application's logic and UI.
* **Key Components:**
  + **UI Components:** Buttons, maps, alerts, input fields, etc.
  + **Location Services Integration:** Utilizing GPS and other APIs to access the device's location.
  + **ADD/ EDIT/ UPDATE/ FILTER Properties**
  + **API Communication:** Functions to send requests to and receive data from the API.
  + **Data Storage:** Local storage for user settings, etc.

**8.2 Backend API Development (Flask)**

* **Language:** Node.js
* **Framework:** Node.js is used to create the RESTful APIs.
* **Key Functionalities:**
  + **User Authentication:** Handling user registration, login, and session management.
  + **ADD Properties**
  + **EDIT Properties**
  + **FILTER Properties**
* **API Endpoints:**
  + /users/register: For user registration.
  + /users/login: For user login.
  + /add: For add property.
  + /edit: For editing properties.
  + /filter: For filter properties.

**(Imagine a diagram here showing the Flask API architecture, with components like "Authentication Module," "SOS Handling Module," "Location Management Module," "Alerts Module," and "Routes Module," and their interactions with the database.)**

**8.3 Database Setup (MongoDB)**

* **Database:** MongoDB is used to store application data.
* **Data Models/Collections:**
  + Users: Stores user information (ID, username, password, contact).
* **Users**: Stores information like user ID, username, password, and role (admin/viewer).
* **Properties**: Stores property listings with details such as sector, pocket, site number, size, demand, description, and builder/seller name.
* **Logs**: (Optional) Tracks inventory changes such as property additions, deletions, and updates for auditing.
  + Routes: (If applicable) Stores data related to safe routes.

**(Imagine an Entity-Relationship Diagram (ERD) or a MongoDB schema diagram showing the collections and their relationships.)**

**8.4 Real-time Communication Optimization**

* **RESTful API Optimization:**
  + Efficient data serialization (e.g., JSON).
  + Caching strategies.
  + Asynchronous processing of tasks.
* **Data Transmission:**
  + Minimizing data payload size.
  + Using efficient data formats.
* **Network Optimization:**
  + Optimizing network requests.
  + Using efficient communication protocols.

**9. Testing and Validation**

Testing and validation are essential to ensure the "Builders Inventory Management System" application's reliability, functionality, and security. This section outlines the testing strategies, test cases, and tools used.

**9.1 Testing Strategy**

The testing strategy involves a combination of different testing methods to cover various aspects of the application.

* **Unit Testing:** Testing individual components or functions in isolation. This includes testing:
  + Flask API functions (e.g., authentication).
  + Mobile app modules
* **Integration Testing:** Testing how different components or modules work together. This includes:
  + Testing the communication between the mobile app and the Flask API.
  + Testing the interaction between the Flask API and the MongoDB database.
* **Functional Testing:** Testing the application against the functional requirements to ensure it performs as expected. This includes testing:
* **Security Testing:** Evaluating the application's security to identify vulnerabilities. This includes:
  + Authentication testing (e.g., brute-force attacks, session hijacking).
  + Authorization testing (e.g., access control).
  + Data validation testing (e.g., preventing SQL injection).
* **Performance Testing:** Evaluating the application's performance under various conditions. This includes:
  + Load testing (simulating multiple users).
  + Stress testing (testing under extreme conditions).
  + Response time testing (measuring the time taken for the application to respond).
* **Usability Testing:** Evaluating the application's ease of use and user-friendliness by observing users interacting with it.
* **Device Testing:** Testing the application on various mobile devices (different models, operating systems, screen sizes) to ensure compatibility.

**9.2 Test Cases**

Test cases are specific scenarios designed to verify particular functionalities or aspects of the application. Below are some example test cases:

| Test Case ID | Description | Input | Expected Output

**10. Results and Discussion**

This section presents the outcomes of the "Builders Inventory Management System" application development and discusses their significance in relation to the project's objectives and the initial problem statement.

**10.1 Achievement of Objectives**

The "Builders Inventory Management System" project successfully achieved its primary objectives:

1. **User Opens the App:** The user launches the "Builders Inventory Management System" application on their mobile device.
2. **User Authenticates:** The user logs in securely using their credentials (or registers if it's their first time).
3. **Add Properties**
4. **Edit Properties**
5. **Update Properties**
6. **App Receives Safety Alerts:** The app displays real-time safety alerts based on the user's location.
7. **User Shares Location (If Needed):** The user may choose to share their location.
8. **User Requests Safe Route (Optional):** The user can request the app to suggest a safer route to their destination.

**10.2 Discussion of Results**

* **Enhanced User Safety:** The "Builders Inventory Management System" application provides a significant enhancement to personal safety by offering rapid SOS activation, real-time safety alerts, and efficient communication with emergency services. These features empower users to take proactive measures to protect themselves and seek help quickly when needed.
* **Improved Response Time:** The 30% reduction in response time achieved through RESTful API optimization is a critical factor in emergency situations where every second counts. This improvement can significantly impact the speed and effectiveness of emergency response.
* **Scalability and Reliability:** The use of MongoDB for scalable storage ensures that the application can handle a growing number of users and data without compromising performance or reliability. The 99.9% uptime guarantee provides users with confidence in the application's availability during critical moments.
* **Secure Authentication:** The implementation of secure user authentication protects user data and privacy, ensuring that only authorized users can access the application's features. This is crucial for maintaining user trust and security.

**10.3 Challenges and Solutions**

* **Challenge:** Ensuring accurate and reliable SOS trigger activation in various environments (e.g., noisy environments for voice triggers, accidental shakes).
* **Solution:** Implementing robust algorithms for shake detection and voice recognition, along with user-configurable sensitivity settings, to minimize false positives and negatives.
* **Challenge:** Optimizing real-time communication to minimize latency and ensure timely delivery of safety alerts, especially in areas with poor network connectivity.
* **Solution:** Employing efficient data transmission protocols, optimizing API calls, and implementing caching mechanisms to reduce network overhead and improve response times.
* **Challenge:** Maintaining user privacy while providing accurate location tracking and sharing functionality.
* **Solution:** Implementing secure encryption and access control mechanisms, providing users with granular control over their location sharing settings, and adhering to privacy best practices.

**11. Conclusion**

The "Builders Inventory Management System" application provides a comprehensive and effective solution for enhancing personal safety through mobile technology. By integrating features such as rapid SOS triggers, real-time safety alerts, location tracking, and safe route suggestions, the application empowers users to take control of their safety and access help quickly in emergency situations.

The successful implementation of secure user authentication, optimized real-time communication, and scalable storage ensures the application's reliability, performance, and security. The 30% reduction in response time and the 99.9% uptime guarantee demonstrate the application's effectiveness in critical situations.

While the current version of "Builders Inventory Management System" offers a robust set of safety features, there are opportunities for future enhancements to further improve its functionality and user experience. These enhancements can build upon the strong foundation established by this project to create an even more powerful and versatile safety tool.

**12. Future Enhancements**

The "Builders Inventory Management System" application has the potential for further development and enhancement to provide even greater safety and security to its users.

* **Predictive Safety Analysis:** Implement machine learning algorithms to analyze historical crime data, real-time incident reports, and user behavior patterns to predict potential safety risks and provide proactive alerts and recommendations.
* **Integration with Wearable Devices:** Integrate the application with wearable devices (e.g., smartwatches) to provide alternative SOS trigger mechanisms and continuous health monitoring.
* **Enhanced Communication Features:** Add features such as in-app messaging with trusted contacts, live audio/video streaming during emergencies, and automated call recording for evidence.
* **Community Safety Network:** Create a community-based safety network where users can report incidents, share safety tips, and collaborate to improve overall safety in their neighborhoods.
* **Multilingual Support:** Expand the application to support multiple languages to make it accessible to a wider user base.
* **Accessibility Features:** Implement accessibility features to make the application usable for people with disabilities (e.g., screen reader compatibility, alternative input methods).
* **Integration with Smart City Infrastructure:** Integrate the application with smart city infrastructure, such as surveillance cameras and emergency response systems, to provide a more comprehensive and coordinated safety response.
* **Gamification of Safety:** Incorporate gamification elements to encourage users to engage with safety features and promote safe behavior (e.g., points for reporting incidents, badges for completing safety training).

**13. References**

This section lists the resources and materials that were consulted during the development of the "Builders Inventory Management System" application.

* Flask Documentation
* MongoDB Documentation
* RESTful API Design Best Practices
* Location Tracking and GPS Accuracy
* Mobile Security Best Practices
* Relevant Academic Papers on Safety Applications and Technologies

**(Note:** This is a sample list. A real report would include specific references to any books, articles, websites, or other resources that directly influenced the project.)

**14. Appendices**

The appendices section contains supplementary materials that are not essential to the main body of the report but provide additional information.

* **Appendix A: Source Code**
  + https://www.connecttobuilder.com
* **Appendix B: Wireframes/Mockups**
  + Visual representations of the application's user interface design.
* **Appendix C: Database Schema**
  + Diagrams or descriptions of the MongoDB database schema.
* **Appendix D: API Documentation**
  + Documentation of the RESTful API endpoints and their usage.
* **Appendix E: Testing Documentation**
  + Detailed test plans, test cases, and test results.
* **Appendix F: User Manual**
  + Instructions on how to use the "Builders Inventory Management System" application.