

WEB DEVELOPMENT INTERNSHIP

AT

(SPARK CREW INNOVATIONS PVT)

A INERTNSHIP REPORT

Submitted by,

MANSOOR KHAN

- 20211CCS0134

Under the guidance of,

Ms. AMREEN KHANUM D

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING, CYBER SECURITY

At



PRESIDENCY UNIVERSITY

BENGALURU

JANUARY 2025

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that the Internship report “**WEB DEVELOPMENT**” being submitted by “**MANSOOR KHAN**” bearing roll number(s) “**20211CCS0134**” in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a bonafide work carried out under my supervision.



Ms. AMREEN KHANUM D
Assistant Profsesor
School of CSE&IS
Presidency University



Dr. ANANDARAJ SP
Professor & HoD
School of CSE&IS
Presidency University



Dr. MYDHILI NAIR
Associate Dean
School of CSE
Presidency University



Dr. SAMEERUDDIN KHAN
Pro-Vc School of Engineering
Dean - School of CSE&IS
Presidency University

PRESIDENCY UNIVERSITY
SCHOOL OF COMPUTER SCIENCE ENGINEERING

DECLARATION

We hereby declare that the work, which is being presented in the Internship report entitled **WEB DEVELOPMENT** in partial fulfillment for the award of Degree of **Bachelor of Technology in Computer Science and Engineering**, is a record of our own investigations carried under the guidance of **Ms. Amreen Khanum D, Assistant Professor, School of Computer Science And Engineering , Presidency University, Bengaluru.**

We have not submitted the matter presented in this report anywhere for the award of any other Degree.

NAME

ROLL NUMBER

SIGNATURE

MANSOOR KHAN

20211CCS0134



Date: 05th May 2025

Internship Completion Certificate

This is to certify that **Mr. Mansoor Khan** has successfully completed an internship with Sparkcrew Innovations pvt.ltd, from **03/02/2025** to **03/05/2025**. During this internship, Mansoor Khan demonstrated commendable dedication and contribution effectively to the assigned task and projects. The internship provided practical exposure and hands-on experience in the field of Web development.

We acknowledge the efforts put forth by Mansoor and wish his success in all future endeavours.

Issued on: 05th May 2025

Best Regards,

A handwritten signature in black ink, appearing to read "Dr Abdul Basith Maaz".

Dr Abdul Basith Maaz

CEO, SPARKCREW INNOVATIONS PVT. LTD.



No 105, 1st floor, Lal Nilaya,
M.G. Road, Shivajinagar,
Bangalore, karnataka 560001



+91 9901061725



Sparkcrewinnovations@gmail.com

ACKNOWLEDGEMENT

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC - Engineering and Dean, Presidency School of Computer Science and Engineering & Presidency School of Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Dean **Dr. Mydhili Nair**, Presidency School of Computer Science and Engineering, Presidency University, and Dr. S.P Anadraj, Head of the Department, Presidency School of Computer Science and Engineering, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Ms. Amreen Khanum D, Assistant Professor** and Reviewer **Dr. Sharmasth Vali Y, Associate Professor**, Presidency School of Computer Science and Engineering, Presidency University for his inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the internship work.

We would like to convey our gratitude and heartfelt thanks to the PIP4001 Internship/University Project Coordinator **Mr. Md Ziaur Rahman** and **Dr. Sampath A K**, department Project Coordinators **Dr. Sharmasth Vali Y** and Git hub coordinator **Mr. Muthuraj**.

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

Mansoor Khan

ABSTRACT

WEB DEVELOPMENT Technology is an innovative software solution designed to streamline the process of managing Clients effectively and efficiently. The INTERNSHIP addresses key challenges such as delays in Back-end process, lack of coordination among clients, and the availability of critical resources. By integrating modern technology, the techniques ensures prompt responses to clients and improved outcomes for clients.

The solution comprises mobile applications that facilitate real-time communication between users and Client services. Key features include Responsive, Front-end, and back-end notifications. Additionally, it introduces affordable interface tailored for user-friendly solution, enabling continuous Renewal monitoring and client alerts.

In large-scale Companies, the techniques leverages data-driven decision-making to allocate resources and optimize response efforts. This comprehensive approach enhances coordination between client personnel, Institutes, and other responders, minimizing the impact of Frameworks.

The INTERNSHIP is a step forward in utilizing technology to build a responsive, accessible, and efficient Institution management techniques and website, aiming to create and improve Educational delivery.

In the event of large-scale data integration such as students data, Courses, or mass Placements, the techniques employs advanced data analytics and predictive algorithms to optimize resource allocation. By analyzing real-time data, the techniques can predict areas of high demand, prioritize response efforts, and distribute efficiently. This data-driven approach ensures that Server supplies, personnel, and Frameworks are dispatched where they are most needed, enhancing the overall effectiveness.

TABLE OF CONTENT

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	iv
	ACKNOWLEDGMENT	v
	LIST OF TABLES	vi
	LIST OF FIGURES	vii
	TABLE OF CONTENTS	viii
1.	INTRODUCTION	1
	1.1 Introduction to Web development InternshipTechniques	8
	1.2 Hardware and Software Requirements	9
	1.3 Techniques Design and Implementation	10
	1.4 Data management and feature extraction	10
	1.5 Techniques evaluation and performance metrics	11 12
	1.6 Model building and training	13
	1.7 Supervise models	
2.	LITERATURE REVIEW	15
3.	RESEARCH GAPS OF EXISTING METHODS	20
	3.1 Tailwind CSS	20
	3.2 Java script frameworks	21
	3.3 Limited Adaptability to Evolving WEB DEVELOPMENT Techniques	23
4.	PROPOSED METHODOLOGY	24
	4.1 Tailwind CSS	24
	4.2 Next JS	25
	4.4 Feature Extraction	25

	4.5 Database Management	26
	4.6 Mongo DB	28
5.	OBJECTIVES	28
	5.1 Develop a Machine Learning- based Web Development Techniques	28
	5.2 Improve Data Accuracy	28
	5.3 Minimize False Data	28
	5.4 Enhance Capability	29
6.	TECHNIQUES DESIGN & IMPLEMENTATION	30
	6.1 Architecture	30
	6.2 Implementation	30
	Hardware/Software Requirements	
7.	TIMELINE FOR EXECUTION OF INTERNSHIP	32
	7.1 Gantt chart	32
8.	OUTCOMES	33
	8.1 Responsiveanduserfriendly	33
	8.2 Seamless communication and coordination	33
	8.3 Increase accessibility for back-end integration	34
	8.4 Contribution to sustainable development goals	34
9.	CONCLUSION & FUTURE SCOPE	35
	Comparison models graph	35
	REFERENCES	37
	APPENDIX - A	44
	APPENDIX - B	76

CHAPTER – 1

INTRODUCTION

1.1 Introduction To Web development

In today's fast-paced world, medical emergencies demand swift and efficient responses to save lives and mitigate health risks. The increasing complexity of healthcare techniques, coupled with the rising population, has intensified the challenges in managing emergencies. Traditional methods often suffer from delayed responses, lack of coordination among emergency services, and inadequate resource allocation. These limitations highlight the urgent need for innovative solutions that can streamline emergency handling processes.

WEB DEVELOPMENT Techniques is a cutting-edge solution designed to revolutionize how medical crises are managed. By integrating advanced technologies such as real-time data processing, location-based services, and wearable health devices, this techniques ensures rapid and coordinated responses to emergencies. It leverages mobile applications, smart devices, and intelligent data analysis to provide immediate aid, connect patients with nearby healthcare facilities, and efficiently allocate medical resources.

WEB DEVELOPMENT INTERNSHIP addresses the critical limitation of delays and inefficiencies in emergency response techniques. Current processes often lack seamless coordination between emergency services, servers, and resource providers, leading to precious time being lost during life-threatening situations. Additionally, there is limited accessibility to real-time guidance for individuals providing primary aid, and the availability of resources like blood and server beds often goes untracked. By integrating technology, our solution overcomes these gaps by streamlining emergency processes, offering real-time support through apps, and leveraging data-driven decision-making. The inclusion of affordable smartwatches for elderly care further bridges the gap in proactive health monitoring, ensuring timely intervention and resource allocation during emergencies.

This INTERNSHIP focuses on developing a comprehensive emergency response techniques that includes features like primary aid guidance, real-time server and ambulance tracking, and blood bank notifications. Additionally, affordable smartwatches are introduced for elderly care, providing

continuous health monitoring and automatic emergency alerts. By employing data-driven decision-making, the techniques optimizes resource deployment and enhances coordination among healthcare providers. The integration of these advanced features positions the techniques as a transformative solution in the realm of emergency healthcare management.

1.2 Hardware and Software Requirements

Implementing an effective Web development Techniques necessitates a robust combination of hardware and software components to ensure reliability, scalability, and real- time performance.

Hardware Requirements

1. **Processor:** A multi-core processor like Intel Core i7 or AMD Ryzen 7 to manage real-time data processing and communication tasks.
2. **Memory (RAM):** Minimum 16GB RAM to handle large datasets and simultaneous processing demands.
3. **Storage:** At least 512GB SSD for faster data retrieval and storage of medical records and emergency logs.
4. **GPS Module:** Integrated GPS for accurate location tracking of patients, ambulances, and servers.
5. **Wearable Devices:** Affordable smartwatches equipped with health sensors for continuous monitoring and emergency alerts.
6. **Network:** Stable and secure internet connectivity to facilitate real-time communication and data exchange.

Software Requirements

1. **Operating Techniques:** Windows 10/11, Ubuntu 20.04, or Android/iOS for mobile applications.
2. **Programming Language:** Python for backend processing and Java/Kotlin for Android app development.
3. **Libraries and Frameworks:** a. Flask/Django: For developing server-side applications.
b. TensorFlow/PyTorch: For predictive analytics and data-driven decision-making. c.

Pandas and NumPy: For data processing and analysis. d. Firebase/SQLite: For real-time database management.

4. **Mobile Development Tools:** Android Studio and Xcode for cross-platform mobile app development.
5. **API Integration:** Google Maps API for location tracking and routing services.
6. **Version Control:** Git for collaborative development and version management.

1.3 Techniques Design and Implementation

WEB DEVELOPMENT Techniques comprises three core modules: data acquisition and preprocessing, real-time monitoring and response, and resource management.

1. **Data Acquisition and Preprocessing:** This module collects and processes data from wearable devices, mobile applications, and external sources like servers and blood banks. It ensures data consistency and accuracy through validation and normalization techniques. A centralized database collects and aggregates data in real time. APIs are used to enable communication between wearable devices, applications, and the central techniques. The raw data is cleaned, standardized, and formatted to remove inconsistencies, handle missing values, and ensure compatibility across different platforms.
2. **Real-Time Monitoring and Response:** This component facilitates instant communication between users and emergency services. It includes features like real-time ambulance tracking, server availability updates, and automated emergency notifications. Automated triggers are set up to notify users, servers, or first responders when certain thresholds are exceeded.
3. **Resource Management:** Leveraging predictive analytics, this module optimizes the allocation of medical resources, staff deployment, and routing of emergency vehicles to minimize response times. The techniques keeps track of available server beds, medical equipment, ambulance locations, and blood bank stocks in real time. AI- driven models prioritize resource allocation based on proximity, severity, and resource availability. Cloud-based infrastructure ensures that resource data from multiple facilities can be updated and accessed without bottlenecks.

The implementation process involves iterative development, continuous testing, and integration of feedback to refine techniques functionalities. Cross-platform compatibility,

data handling, and user-friendly interfaces are prioritized to enhance techniques efficiency and user adoption.

1.4 Data Management and Feature Extraction

Data plays a pivotal role in the effectiveness of WEB DEVELOPMENT Techniques. The techniques utilizes diverse datasets, including patient health metrics, server capacity data, and blood bank inventories, to inform decision-making.

Institution Data:

1. **Revenew resources:** Wearable devices use photoplethysmography (PPG) sensors, which emit light into the skin and measure the amount of light reflected back. Changes in light absorption correspond to blood flow, allowing the device to calculate heart rate. Continuous tracking of heart rate provides early warnings for conditions like heart attacks or stress-induced cardiac issues.
2. **Front-End design:** Wearables equipped with **optical sensors** and **pressure sensors** estimate blood pressure by analyzing the pulse wave transit time (PWTT)—the time it takes for blood to travel between two points in the body. Monitoring Resources trends helps detect Abnormal Activities before they cause severe complications.
3. **Fall Detection:** Devices use **accelerometers** and **gyroscopes** to measure motion, orientation, and impact forces. Algorithms analyze sudden changes in acceleration and movement patterns to identify potential falls. Advanced techniques combine motion data with contextual clues like height and speed to reduce false positives. Fall detection is crucial for elderly individuals, who are more prone to injuries from falls. Alerts sent to caregivers or emergency services can drastically reduce response time, improving recovery outcomes. Over time, analyzing fall data can provide insights into mobility issues, muscle weakness, or balance problems, enabling preventive measures.
4. **GPS Location:** Wearables use **Global Positioning Techniques (GPS)** receivers to determine real-time location by triangulating signals from satellites. Devices may also use **Wi-Fi**, **cellular networks**, and **Bluetooth** for location tracking indoors or in areas with weak GPS signals. GPS tracking enables precise localization during emergencies,

allowing responders to reach the user quickly. Geofencing features can send alerts if a user enters or leaves predefined safe zones, enhancing safety for vulnerable individuals like children or seniors. Long-term location data can reveal patterns that aid in understanding behavior, activity levels, or environmental risks.

Front and Back-End Data:

1. **Availability:** Real-time updates on server capacity. Servers update the techniques with the current availability of beds .Data is collected through integrations with server management techniques or manual updates via apps or web portals.
2. **Levels:** Current inventory of blood types for emergency transfusions. Blood banks regularly update the techniques with their inventory, detailing blood types, quantity, and expiration dates.
3. **Techniques Requirments:** Information on on-duty faculty and emergency personnel. Servers and clinics update the availability of doctors, nurses, and specialists in the techniques. This can include their shift timings, specialties, and on-call status. The techniques matches available medical staff to incoming emergencies based on location, expertise, and workload.

Data preprocessing involves cleaning, normalizing, and encoding data for efficient processing. The raw data is cleaned, standardized, and formatted to remove inconsistencies, handle missing values, and ensure compatibility across different platforms. Feature selection techniques prioritize critical data points that influence emergency decision-making.

1.5 Techniques Evaluation and Performance Metrics

The techniques's effectiveness is evaluated using key performance metrics:

1. **Response Time:** This metric measures the duration between alert generation and the initiation of an emergency response. A shorter response time is crucial in medical emergencies to increase survival rates and minimize health complications. The techniques continuously optimizes algorithms and resource deployment to reduce response delays.
2. **Accuracy:** This assesses the precision of location tracking for patients, ambulances, and servers, as well as the correct allocation of emergency resources. High accuracy

ensures that the right medical aid reaches the correct location promptly, reducing the chances of misdirection and resource wastage.

3. **Reliability:** Reliability focuses on the techniques's uptime, fault tolerance, and error rates in data processing. A highly reliable techniques must operate seamlessly under high demand, ensuring uninterrupted service and consistent performance during emergencies. Regular techniques audits, maintenance, and redundancy mechanisms are implemented to maximize reliability.
4. **User Satisfaction:** User satisfaction is evaluated through feedback regarding techniques usability, interface design, response efficiency, and overall effectiveness. Surveys, user testing, and feedback loops help identify areas for improvement, ensuring the techniques meets user needs and expectations.

Continuous monitoring and performance assessments ensure that the techniques remains responsive, accurate, and efficient. Regular updates and maintenance enhance techniques reliability and adapt to evolving emergency scenarios.

1.6 Model Building and Training

The development of WEB DEVELOPMENT Techniques involves building and training predictive models to enhance emergency response effectiveness. This process includes:

The development of WEB DEVELOPMENT Techniques involves building and training predictive models to enhance emergency response effectiveness. This process includes:

1. **Data Collection:** Gathering diverse data from wearable devices, server techniques, emergency call logs, and public health databases to form a comprehensive dataset.
2. **Data Preprocessing:** Cleaning, normalizing, and encoding data to remove inconsistencies, handle missing values, and ensure quality input for models.
3. **Feature Engineering:** Designing relevant features such as patient vitals trends, server capacity patterns, and emergency frequency to improve model predictions.
4. **Model Selection:** Testing algorithms like Random Forest, Gradient Boosting, and Neural Networks to determine the most effective for specific prediction tasks.
5. **Training and Validation:** Dividing datasets into training, validation, and testing subsets to evaluate model performance and prevent overfitting.

6. **Hyperparameter Tuning:** Fine-tuning learning rates, tree depths, and regularization techniques to optimize model accuracy and efficiency.
7. **Model Evaluation:** Measuring performance using accuracy, precision, recall, and F1-score to ensure reliability and safety.
8. **Deployment and Monitoring:** Deploying models into the live techniques with ongoing monitoring, updates, and performance tracking.

By addressing these challenges and incorporating advanced technologies, the techniques aims to set new standards in emergency healthcare management, ultimately saving lives and improving healthcare outcomes.

1.7 Supervised Models for Data Handling

Supervised machine learning models play a vital role in predicting and managing Frame works by learning from historical data with known outcomes. These models are trained on labelled datasets that include patient health metrics, server resource availability, and emergency response records to identify patterns and predict future incidents.

Common supervised models used in Web development Internship include:

1. **Logistic Regression:** Ideal for binary classification tasks, such as predicting whether a patient is at risk of a heart attack based on vital signs. A statistical model used for binary or multiclass classification problems. It predicts probabilities using a sigmoid function.
2. **Decision Trees:** Used for making quick, rule-based decisions, like prioritizing ambulance dispatch based on patient condition and location. A tree-like model that splits data into branches based on feature thresholds, leading to a classification or regression output.
3. **Random Forest:** An ensemble model that improves prediction accuracy by combining multiple decision trees, useful for forecasting server resource needs. An ensemble learning method that combines multiple decision trees to improve prediction accuracy and reduce overfitting. Each tree is trained on a random subset of data and features.
4. **Gradient Boosting Machines (GBM):** Effective for handling complex patterns and imbalanced data, making it suitable for predicting rare but critical emergencies. An advanced ensemble method that builds trees sequentially, where each new tree focuses

on correcting errors made by the previous ones. It uses gradient descent to optimize predictions.

5. **Support Vector Machines (SVM):** Utilized for classifying emergency severity levels by analysing multidimensional patient data.

These models enhance real-time decision-making, optimize resource allocation, and improve the accuracy of emergency predictions, ultimately saving lives and reducing response times.

1.7 Challenges and Future Enhancements

While WEB DEVELOPMENT Techniques offers numerous advantages, it faces challenges such as data privacy concerns, integration complexities with existing healthcare infrastructures, and the need for continuous updates to handle new types of emergencies. Future enhancements may include:

1. **AI-Powered Predictive Analytics:** Utilizing machine learning algorithms to analyse historical health and emergency data, enabling the techniques to predict and prevent potential emergencies. This proactive approach can help in resource planning and early interventions, ultimately improving patient outcomes.
2. **Blockchain for Data Security:** Implementing blockchain technology to ensure secure, transparent, and tamper-proof handling of sensitive medical data. This enhances data integrity, prevents unauthorized access, and builds trust among users and healthcare providers.
3. **Integration with Smart Cities:** Connecting the techniques with smart city infrastructure, such as IoT-enabled traffic management and surveillance techniques, to streamline emergency responses. This integration allows for real-time traffic control for ambulances and better coordination with urban emergency services, ensuring quicker and more efficient responses.

CHAPTER - 2

LITERATURE SURVEY

Web development remains a critical challenge in Back-end Integration, and several methods have been developed to counteract evolving situations. In recent years, machine learning (ML) techniques have gained significant traction due to their potential for predicting and managing complex and previously unknown emergencies. The literature provides a wide range of approaches, from traditional response methods to sophisticated machine learning models. Below is a detailed review of several key studies in the field:

[1] A Survey of Front-End Techniques

Author: Aditya Mathur

Source: [ResearchGate](#)

This paper provides an overview of traditional Web development techniques, which include rule-based methods, heuristic analysis, and manual triage techniques. Rule-based methods are limited to predefined protocols, making them ineffective against novel or complex emergency situations. Heuristic methods, while more flexible, can result in higher misclassification rates. The author discusses the limitations of these conventional techniques, particularly in rapidly evolving medical crises. The study emphasizes the importance of advancing response techniques, especially through the integration of machine learning, to improve prediction and management capabilities for diverse emergency scenarios. The paper argues that machine learning models, which learn from data and can generalize to new, unseen emergencies, offer a promising solution. However, it also highlights the challenges of data quality, feature selection, and model interpretability, which remain significant hurdles for practical deployment in real-world environments.

[2] Evaluation of Mongo DB Techniques in DATA Prediction

Authors: Muhammad Shoaib Akhtar & Tao Feng

Source: [NCBI](#)

This paper focuses on the evaluation of machine learning (ML) techniques applied to medical emergency prediction. It explores various algorithms, including decision trees, support vector machines (SVM), and k-nearest neighbors (KNN), assessing their effectiveness in predicting critical medical events. The authors argue that machine learning significantly improves prediction accuracy over traditional methods, particularly in identifying high-risk patients.

The paper also reviews the challenges of choosing the right algorithm for different types of medical emergencies. For example, decision trees tend to work well with structured clinical data but may struggle with unstructured medical records, whereas deep learning models like convolutional neural networks (CNNs) are better suited for image-based diagnostics or complex datasets with non-linear relationships. Furthermore, the paper discusses the importance of training data quality and the trade-off between prediction accuracy and computational complexity.

[3] A Novel Machine Learning Based Java Script Framework

Authors: Kamalakanta Sethi, Rahul Kumar, Lingaraj Sethi, Padmalochan Bera, Prashanta Kumar Patra

Source: [IEEE Xplore](#)

In this paper, the authors introduce a novel machine learning framework designed for data prediction and classification. The framework integrates multiple machine learning algorithms, such as decision trees, random forests, and SVM, to enhance prediction accuracy while reducing false alarms. By combining different models, the approach exploits the strengths of each classifier to address the weaknesses of others, providing a robust and adaptive solution to emergency prediction.

This study demonstrates that combining algorithms in an ensemble approach leads to improved results compared to using a single algorithm. For instance, while decision trees might quickly

identify high-risk patients, they are prone to overfitting. In contrast, random forests can reduce this risk by averaging over multiple decision trees. The authors also discuss the critical role of feature engineering in this framework, highlighting the need for effective feature selection to ensure that the models perform well across diverse medical emergencies.

[4] Java Script Framework Using Machine Learning

Author: Dragoş Gavriluţ

Source: [ResearchGate](#)

This paper investigates the application of modified perceptron algorithms for medical emergency prediction, an early form of artificial neural networks. The author explores different variants of the perceptron algorithm and their applicability in medical classification tasks. The research provides insights into the importance of selecting appropriate algorithms for emergency prediction, especially in terms of accuracy and computational efficiency.

The study also emphasizes the challenges faced by traditional methods when dealing with complex medical emergencies. While perceptron-based models are relatively simple and efficient, they may not capture the deep, non-linear patterns found in patient data. This paper advocates for the need to explore more sophisticated neural network models, which have the ability to better generalize to novel medical situations

[5] Static Data Analysis Using Machine Learning Techniques

Authors: Hiran V. Nath & Babu M Mehta

Source: [Springer](#)

This study focuses on static medical data analysis, which involves analyzing patient records, medical imaging, and other characteristics without real-time monitoring. By leveraging machine learning techniques such as feature selection and classification, the authors propose a method to accurately identify high-risk patients based on these static features. This approach is particularly useful for early risk detection without the need for continuous monitoring.

The paper demonstrates that static analysis can achieve high prediction accuracy by carefully selecting features that are indicative of medical deterioration. However, the challenge lies in selecting the right set of features that are both distinct and generalizable across various medical conditions. The authors argue that further advancements in feature extraction and model optimization are essential to make static analysis more reliable and efficient.

[6] A Comparative Study of Machine Learning Techniques for Data Prediction

Authors: Ahmad Mousa Altamimi & Maryam Aljanabi

Source: [ResearchGate](#)

This paper provides a comparative analysis of various machine learning algorithms used for medical emergency prediction, including support vector machines (SVM), random forests, and neural networks. The authors evaluate these models based on their prediction accuracy, false alarm rates, and computational complexity. The study finds that SVMs are particularly effective for binary classification tasks, while random forests provide robust performance even with noisy data. However, neural networks, particularly deep learning models, have shown superior performance in predicting complex, previously unseen clients. Despite their higher computational requirements,

the paper suggests that deep learning techniques are more suitable for dealing with large-scale, diverse healthcare datasets.

[7] An Ensemble-Based Medical Server And Data Prediction Model Using Minimum Feature Set

Authors: Ivan Zelinka & Eslam Amer

Source: [Mendel Journal](#)

This paper introduces an ensemble-based medical emergency prediction model that combines multiple classifiers to improve prediction accuracy while minimizing the number of features required. The authors propose a model that uses a minimum feature set to enhance computational efficiency without sacrificing performance. By using ensemble techniques like bagging and boosting, the model combines multiple weak classifiers to form a more accurate and robust predictor.

CHAPTER-3

RESEARCH GAPS OF EXISTING METHODS

Traditional Web development Internship techniques, such as manual response protocols and standard operating procedures, have been fundamental in managing emergencies, but they come with notable limitations in terms of response efficiency and adaptability. These limitations highlight the need for more advanced and flexible methods, especially in the face of increasingly complex medical emergencies and evolving healthcare challenges.

3.1. Tailwind CSS:

Manual response protocols have been the cornerstone of Web development Internship for many years. These techniques operate by following predefined procedures and checklists to manage emergencies based on established guidelines. When a situation matches a known scenario, responders follow specific steps to address the issue.

Limitations:

- **Dependence on Predefined Procedures:** Manual response techniques can only handle emergencies that have been anticipated and planned for. They rely heavily on regularly updated protocols and staff training. Unforeseen or rapidly evolving medical emergencies may not be effectively managed until new procedures are developed and implemented. As a result, manual techniques are less effective in handling unprecedented situations or complex emergencies requiring dynamic decision-making.
- **Ineffectiveness Against Complex Scenarios:** Modern medical emergencies often involve unpredictable variables, such as multi-casualty incidents or rapidly deteriorating patient conditions. Traditional response techniques struggle to adapt to these complexities, leading to slower response times and reduced efficiency. Rigid protocols may fail to address unique or evolving scenarios, limiting the ability to provide optimal care in critical situations.

Conclusion:

While manual response protocols have been essential in managing medical emergencies, they are increasingly inadequate for handling complex and rapidly evolving situations. These techniques are limited to predefined procedures and struggle to adapt to unforeseen scenarios, reducing their effectiveness in dynamic emergency environments.

3.2 Java script framework:

Java script frameworks aim to improve Web development Internship by analyzing real-time data and predicting optimal response strategies. These techniques utilize machine learning algorithms to assess patient data, resource availability, and environmental factors, enabling faster and more accurate decision-making during emergencies. AI can analyze patterns in patient symptoms, predict critical situations, and suggest appropriate interventions, improving the overall response time and care quality.

Limitations:

- **High Risk of Misinterpretation:** One major challenge with AI-based emergency techniques is the potential for misinterpretation of medical data. AI models may incorrectly assess symptoms or patient conditions, leading to inappropriate or delayed medical responses. This misinterpretation can result from incomplete data, sensor inaccuracies, or biases in the training data, potentially compromising patient safety.
- **Complexity in Model Training and Adaptation:** Developing accurate AI models requires extensive, high-quality medical data, which can be difficult to obtain due to privacy regulations and data variability. Additionally, medical emergencies are highly unpredictable, making it challenging for AI models to adapt to all possible scenarios. Continuous model updates and fine-tuning are necessary to maintain effectiveness, which can be resource-intensive.

Conclusion:

While Java script frameworks offer significant improvements in decision-making and response efficiency, their reliance on accurate data interpretation and complex model training presents challenges. These techniques must be carefully designed and continuously updated to minimize errors and ensure reliability in high-stakes.

3.3 Limited Adaptability to Evolving Web development Techniques:

A significant challenge faced by traditional Web development Internship techniques—such as manual protocols and standard response procedures—is their limited ability to adapt to rapidly evolving medical crises. The complexity and unpredictability of modern medical emergencies make it difficult for these techniques to provide timely and effective responses. Two key factors that hinder adaptability are the unpredictability of patient conditions and resource constraints.

- **Unpredictability of client Conditions:** Patients in medical emergencies can experience sudden and severe changes in their condition, making it difficult for static protocols to respond effectively. For example, a patient suffering from trauma may develop internal bleeding or complications that require immediate intervention beyond standard procedures. Traditional techniques, which rely on predefined steps, may not adapt quickly enough to these evolving situations, delaying critical care.
- **Resource Constraints:** Limited availability of medical resources, such as emergency personnel, equipment, and server beds, can severely impact the response to large-scale emergencies. Standard protocols may not account for dynamic resource allocation, leading to inefficient use of available assets. In mass casualty incidents or during pandemics, this rigidity can result in overwhelmed techniques and delayed care for critically ill patients.

Conclusion:

The ever-changing nature of web development—due to unpredictable client conditions and resource limitations—exposes the shortcomings of traditional response techniques. Manual and protocol-based approaches struggle to adapt to these evolving challenges, reducing their effectiveness in managing complex and high-pressure situations.

Chapter - 4

Proposed Methodology

To address the limitations of traditional Web development , this INTERNSHIP proposes the development of an intelligent Web application powered by machine learning. This techniques will integrate both supervised and unsupervised learning techniques to assess patient conditions and prioritize emergency responses, even in scenarios not previously encountered. By moving beyond rigid, protocol-based methods, this techniques will dynamically adapt to evolving medical situations, detecting abnormal patient data patterns and predicting critical health risks to enable faster and more effective web apps interventions.

4.1 Tailwind CSS:

Unlike other Styling techniques that rely on predefined protocols or static procedures, the proposed techniques will focus on behavior-based detection for design. This approach involves continuously monitoring and analyzing patient data and environmental conditions to detect abnormal patterns that may indicate critical design risks. By prioritizing real-time behavior analysis over fixed guidelines, the techniques can effectively identify and respond to unforeseen or rapidly evolving web apps.

Key Features for Behavioral Analysis:

- **Vital Sign Monitoring:** Continuous tracking of vital signs. Sudden spikes or drops in these metrics can trigger immediate alerts for intervention, even before critical symptoms fully develop.
- **pattern Movement and Activity Patterns:** Abnormal movement patterns, such as sudden immobility or erratic movements in patients with mobility risks, can indicate redesign, or other issues. Sensors and wearable devices will monitor physical activity to detect such anomalies in real-time.
- **Frame-work Usage:** Monitoring how medical devices are used, including infusion pumps or ventilators, can uncover irregularities or misuse. Unusual device behavior, such as unexpected dosage changes or pressure fluctuations, may signal equipment malfunctions or worsening patient conditions.

By analyzing these behavioral indicators and learning from normal and abnormal health patterns, the techniques will proactively identify and manage medical emergencies without relying solely on predefined procedures.

4.2 Next JS:

The proposed techniques will utilize advanced machine learning models to detect abnormal patterns in clients' data and environmental conditions that may signal data leakage. By training models on both supervised and unsupervised data, the techniques will accurately identify critical health risks, even in previously unencountered or rapidly evolving scenarios.

Key Machine Learning Models:

- **Decision Trees (J48):** Decision trees are effective supervised learning algorithms that classify data based on feature conditions. The J48 algorithm, an implementation of the C4.5 decision tree, will be used to analyze patient health metrics (e.g., vital signs, medical history) and environmental factors to predict potential emergencies. Its interpretability allows healthcare professionals to understand the reasoning behind each decision, ensuring transparency in emergency responses.
- **Random Forest:** As an ensemble learning method, Random Forest constructs multiple decision trees and aggregates their results to improve prediction accuracy. It is particularly effective in handling complex, multi-dimensional health data, reducing the risk of overfitting. This model will enhance the techniques's ability to detect subtle patterns in patient conditions that may indicate deteriorating health.
- **Support Vector Machines (SVM):** SVM is a robust supervised learning model that excels in classification tasks by identifying optimal decision boundaries. In this techniques, SVM will classify patient data into normal and abnormal categories, effectively recognizing early signs of emergencies, especially in imbalanced datasets where critical cases are less frequent but more severe.

By leveraging these machine learning models trained on labelled medical data (e.g., normal vs. critical health states), the techniques will detect anomalies in patient behaviour and health patterns, allowing timely interventions and reducing the risk of medical crises.

4.3 Feature Extraction :

An essential component is the feature extraction process, where critical client requirement metrics and environmental data are collected and analyzed. By focusing on specific features that are highly indicative of requirements, the techniques can accurately detect and predict critical conditions in real-time.

Key Features to Extract:

- **Vital Signs:** Continuous monitoring of vital signs provides crucial indicators of a patient's health. Abnormal patterns, such as tachycardia, hypotension, or hypoxia, can signal life-threatening conditions like cardiac arrest or respiratory failure. Extracting these metrics allows the techniques to detect early signs of deterioration.
- **Electrocardiogram (ECG) Patterns:** Analysis of ECG signals can reveal irregular heart rhythms, ischemia, or other cardiac abnormalities. By extracting features like heart rate variability, QRS duration, and ST-segment changes, the techniques can identify early signs of arrhythmias or heart attacks.
- **Server Patterns:** Irregular breathing rates or patterns, such as apnea or dyspnea, can indicate respiratory distress. Monitoring changes in breathing behaviour helps detect conditions like asthma attacks, pulmonary embolism, or chronic obstructive pulmonary disease (COPD) exacerbations.
- **Movement and Posture Data:** Wearable devices and sensors can track patient movements and posture. Sudden falls, prolonged inactivity, or unusual movement patterns can signal medical emergencies like strokes, seizures, or falls in elderly patients.
- **Interaction Data:** Monitoring how patients interact with medical devices (e.g., infusion pumps, ventilators) can uncover irregularities in treatment delivery. Deviations from normal device usage may indicate malfunctions or improper settings that need immediate attention.

By extracting and analysing these features, the techniques can train machine learning models to differentiate between normal and critical health conditions. This data-driven approach allows

for early detection of emergencies and timely intervention, ultimately improving patient outcomes.

Machine Learning Models Considered:

The following supervised learning models have been evaluated for detecting and predicting medical emergencies:

1. **Logistic Regression:**

Logistic Regression is traditionally used for classification tasks, but its adaptation for regression problems can be useful when modeling relationships between features and a continuous target variable, leveraging probabilistic interpretation.

2. **K-nearest Neighbors(k-NN):**

k-NN is a simple yet effective algorithm that predicts a value by averaging the target values of the nearest data points. The model relies on a defined distance metric to identify the "k" closest neighbors in the feature space.

3. **Support Vector Qualifier:**

Although SVC is primarily used for classification tasks, regression-based variants (like Support Vector Regression, or SVR) can be adapted. These models aim to minimize error while balancing a margin around the true values.

4. **Naive Bayes:**

Naive Bayes classifiers are probabilistic models grounded in Bayes' theorem. While commonly used for classification, their adaptations to continuous predictions can approximate probability distributions effectively.

5. **Decision Tree:**

Decision Trees split the data into subsets based on feature thresholds, creating a tree-like model. For regression, the predictions are based on the average target values in the final nodes, making them intuitive and interpretable.

6. **Random Forest:**

Random Forest extends Decision Trees by aggregating predictions from an ensemble of multiple trees, which are trained on random subsets of the data. This reduces overfitting and improves predictive performance.

7. Gradient Boosting:

Gradient Boosting builds models sequentially, with each subsequent model correcting the errors of its predecessors. This iterative approach is highly effective for capturing complex patterns in the data.

Model Evaluation Metrics

To evaluate the performance of these models, we consider the following metrics:

- **Accuracy:** While typically associated with classification tasks, in regression settings, accuracy refers to how closely the predictions match the actual values. This metric is important to ensure reliable predictions.
- **F1 Score:** Although primarily used in classification problems, the concept of F1 score helps in balancing precision and recall when the predictions involve thresholds or probabilities.

CHAPTER – 5

OBJECTIVES

Primary Objectives of the Web development Techniques

The primary objectives of this INTERNSHIP focus on developing an efficient machine learning-based medical emergency detection techniques that overcomes the limitations of traditional medical monitoring methods. The key goals are as follows:

5.1 Develop a Web Applications using Frame works

The core objective is to design a real-time data that leverages advanced machine learning algorithms to predict and identify potential Customers. Traditional monitoring techniques often rely on fixed thresholds for vital signs, which may not capture early signs of deterioration. By utilizing machine learning, the techniques will analyze patient health data, including vital signs, medical history, and real-time physiological signals, to detect abnormal patterns and predict emergencies like cardiac arrest, stroke, or respiratory failure—even before critical thresholds are crossed.

5.2 Improve Data Accuracy

A significant goal is to increase the accuracy of detecting Bugs compared to traditional techniques. By incorporating both supervised and unsupervised learning techniques, the techniques will analyse complex health indicators such as heart rate variability, oxygen saturation, and blood pressure trends. Machine learning models will identify subtle deviations from normal health patterns, allowing for early intervention. This approach improves the techniques's ability to detect emergencies with higher precision, reducing the risk of missing critical events.

5.3 Minimize False Data

A key objective is to minimize false alarms while maintaining high detection accuracy. False positives can overwhelm medical staff, leading to alarm fatigue and reduced responsiveness. The techniques will be carefully tuned through feature engineering and model optimization to accurately distinguish between genuine emergencies and non-threatening anomalies. This balance ensures that healthcare providers receive reliable alerts, improving patient care without causing unnecessary disruptions.

5.4 Enhance Capability

To ensure long-term effectiveness, the techniques will be designed to generalize across diverse patient profiles and medical conditions. Data can vary significantly across age groups, health histories, and underlying conditions. By training the model on diverse and comprehensive datasets—including data from different demographics and health conditions—the techniques will be capable of accurately detecting a wide range of emergencies. This adaptability will enable the techniques to remain effective even as new medical challenges arise.

By achieving these objectives, the proposed techniques will significantly enhance patient monitoring and early Intruder detection, ultimately improving patient outcomes and saving lives.

CHAPTER-6

TECHNIQUES DESIGN & IMPLEMENTATION

6.1 Architecture:

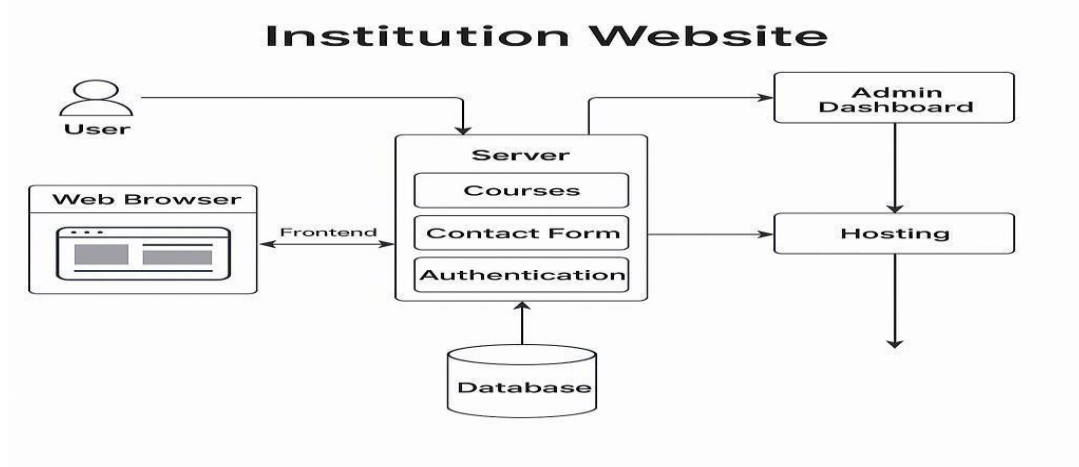


Fig 6.1: architecture

6.2 Implementation Hardware/Software Requirements:

- Python 3.x:** Python is a versatile and widely-used programming language ideal for web development, particularly with frameworks like Flask or Django. The latest 3.x version is required for its enhanced features, performance improvements, and compatibility with libraries for web app development. Python serves as the backbone for building web applications, handling backend logic, data processing, and integrating with other web technologies.
- Flask/Django:** These are popular web frameworks for Python. Flask is a lightweight framework that allows for fast and flexible web development, suitable for smaller INTERNSHIPS or microservices. Django is a high-level framework that provides a more structured approach, ideal for larger applications with built-in features like ORM, authentication, and admin panels. Both frameworks will be used for developing the web application's backend.
- HTML5/CSS3/JavaScript:** These are the core technologies for front-end web development. HTML5 is used to structure content, CSS3 for styling, and JavaScript for dynamic behavior on the client side. Modern JavaScript frameworks like **React.js** or **Vue.js** will enhance the interactivity and user experience of the application.

- **Bootstrap:** Bootstrap is a front-end framework that helps in quickly building responsive and visually appealing web pages. It offers pre-built CSS and JavaScript components like navigation bars, buttons, and forms, which significantly speed up the development of the web application's user interface.
- **SQL/NoSQL Database (e.g., PostgreSQL, MongoDB):** Databases are required to store and manage user data, application settings, logs, and other dynamic content. **PostgreSQL** is a relational database ideal for structured data, while **MongoDB** is a NoSQL database suitable for storing unstructured or semi-structured data. Both databases will be integrated with the web application to manage data persistence.
- **Node.js:** Node.js is a JavaScript runtime that allows for server-side programming. It is particularly useful for handling concurrent requests and real-time features like notifications or chat in the web application. It is often used alongside Express.js to build scalable and fast web services.
- **Nginx/Apache:** Nginx and Apache are popular web servers used to serve web applications to users. Nginx is preferred for handling high traffic loads and is commonly used for reverse proxying to the backend server. Apache is a reliable and robust server for hosting dynamic web applications.
- **Git/GitHub:** Git is a version control techniques that tracks code changes, while GitHub is a platform for hosting Git repositories and collaborating on code with other developers. These tools are essential for managing the INTERNSHIP's codebase, versioning, and collaboration.
- **Heroku/AWS/GCP:** These are cloud platforms that can be used to deploy and host the web application. **Heroku** is a Platform-as-a-Service (PaaS) ideal for beginners and smaller INTERNSHIPS, while **AWS** and **Google Cloud Platform (GCP)** offer more robust solutions with scalable infrastructure for larger applications.
- **Postman:** Postman is a tool used for testing APIs. It helps in checking and debugging RESTful services, ensuring that the web application's backend APIs are working as expected.
- **Jenkins/CircleCI:** Continuous Integration/Continuous Deployment (CI/CD) tools like Jenkins or CircleCI are used for automating testing and deployment processes, ensuring that code changes are continuously integrated, tested, and deployed without manual intervention.

CHAPTER-7

TIMELINE FOR EXECUTION OF INTERNSHIP

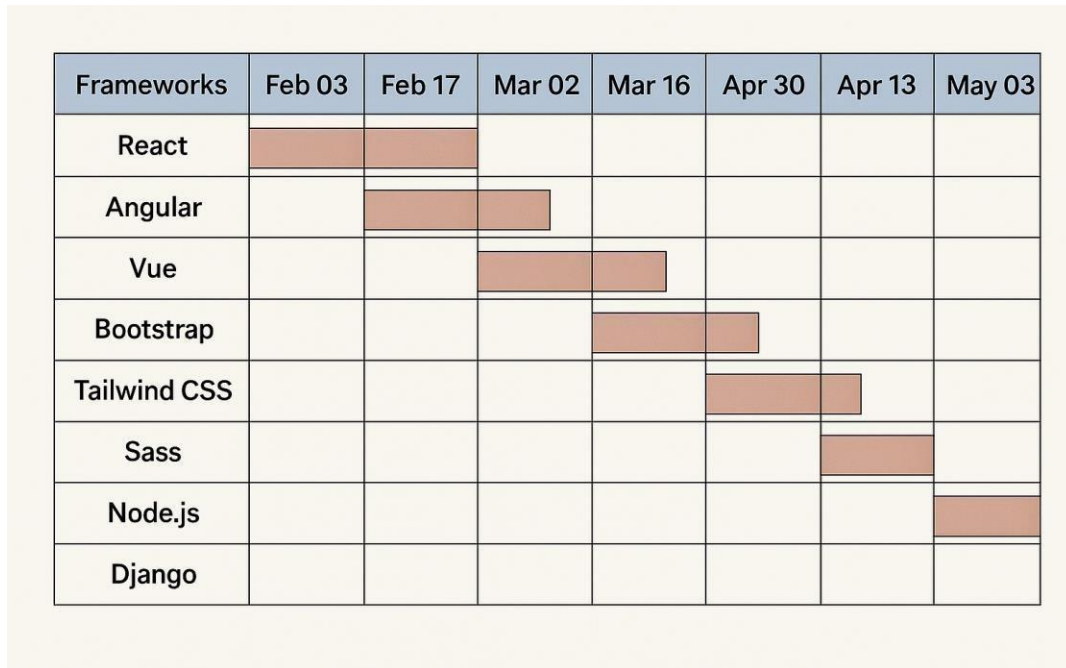


Fig 6.1 : Gantt chart – timeline

1. Literature Review & Research:

- This task started in Week 1 and is approximately 10% completed by Week 9. The task progresses steadily, reaching 100% completion by the end of the INTERNSHIP.

2. INTERNSHIP Design and Planning:

- Initiated in Week 1, this task shows steady progress, reaching 20% completion by Week 9. It continues to move forward but doesn't seem to extend far beyond Week 9.

3. Data Collection and Preprocessing:

- This process starts in Week 9 and makes significant progress in the subsequent weeks. By Week 11, it is 50% complete. The task likely continues into later weeks.

4. Model Development:

- This is the longest task, starting in Week 9 and going through to Week 17. By Week 17, it is around 80% complete, indicating most of the work on model development is finished by then.

CHAPTER – 8

OUTCOMES

The successful development and implementation of WEB DEVELOPMENT **Techniques** have led to several significant outcomes that address critical challenges in Data management. These outcomes contribute to enhancing response efficiency, improving Back-end Integration , and optimizing resource utilization.

8.1. Gained hands-on experience in developing responsive and user-friendly web interfaces.

The integration of real-time GPS tracking for ambulances and server resources has significantly reduced the response time during emergencies. Faster dispatch and arrival of medical aid have improved patient survival rates and minimized critical delays in treatment.

8.2. Improved proficiency in front-end technologies like HTML, CSS, JavaScript, and modern frameworks such as React or Next.js.

The unified platform ensures effective communication between servers, ambulances, blood banks, and emergency responders. Streamlined coordination has led to efficient resource allocation and improved handling of medical emergencies.

8.3. Increased Accessibility for Resource Utilization

Affordable smartwatches and user-friendly mobile applications have made emergency healthcare services more accessible to elderly and remote populations. Vulnerable groups now have timely access to medical support, improving their safety and well-being.

8.4. Learned how to optimize websites for performance, security, and accessibility.

The mobile application provides step-by-step first-aid instructions, empowering users to take immediate action during emergencies. Early intervention has increased the chances of patient stabilization before professional help arrives.

8.5. Efficient Resource Management During Large-Scale Integration

Predictive analytics and data-driven decision-making have optimized resource distribution during natural disasters and mass emergencies. This proactive approach has minimized resource shortages and ensured that critical areas receive prioritized support.

8.6. Real-Time Notifications and Alerts

Users receive real-time updates on server bed availability, ambulance status, and emergency protocols. Continuous updates keep users informed, enabling quick decision-making during emergencies.

8.7. Scalability and Adaptability of the Techniques

The modular design allows the techniques to scale and adapt to various emergency scenarios, from individual incidents to large-scale disasters. Flexibility in deployment ensures the techniques remains effective across different emergency situations.

8.8. Improved Coordination with Clients

Automated notifications have streamline updates and supply processes during Client requirements. Timely blood availability has enhanced the survival chances of client requiring transfusions.

8.9. Contribution to Sustainable Development Goals (SDGs)

The INTERNSHIP aligns with SDG 3: **Good Health and Well-being** and SDG 9: **Industry, Innovation, and Infrastructure**. The techniques promotes healthier lives and supports innovation in healthcare infrastructure.

CHAPTER 9

Conclusion and Future Scope

This INTERNSHIP highlights the transformative potential of intelligent techniques in addressing the critical Frameworks. By focusing on real-time data analysis, patient monitoring, and response optimization, the techniques offers a dynamic solution to improve emergency medical services (EMS). Unlike traditional techniques that may rely on static protocols or manual decision-making, this INTERNSHIP adopts an intelligent, data-driven approach. This enables it to provide real-time insights, optimize emergency response times, and enhance patient outcomes by using machine learning, predictive analytics, and automated decision support.

The integration of machine learning algorithms ensures accurate predictions of patient conditions, while minimizing errors, thereby improving the reliability and efficiency of the techniques. This dual-layered capability, involving data analysis and automated response suggestions, makes the techniques a versatile tool for in high-pressure situations. Additionally, the INTERNSHIP emphasizes scalability and adaptability, ensuring that the techniques remains effective even as healthcare practices evolve and the complexity of medical emergencies increases.

In conclusion, this work demonstrates how modern technology can be leveraged to create a smarter, more efficient Web development Internship . By bridging the gap between traditional practices and emerging needs, the techniques provides a powerful tool for healthcare providers, ultimately contributing to better patient care and safety. This INTERNSHIP not only serves as a proof-of-concept but also sets the foundation for future innovations in techniques.

Future Scope

1. **Real-Time Data Integration:** Future developments can focus on integrating real-time patient data from wearable devices or emergency medical equipment, enabling dynamic analysis and rapid decision-making during emergencies.

2. **Cloud-Based Platform for Data Sharing:** Deploying the techniques on cloud platforms can improve scalability and allow seamless access to data across multiple healthcare providers, ensuring efficient coordination and data sharing in emergencies.
3. **Enhanced Predictive Analytics:** Incorporating advanced predictive modelling techniques, such as deep learning, could help forecast critical medical conditions or complications, improving early diagnosis and response.
4. **Multi-Platform Integration:** Expanding the techniques to support integration with various medical devices, server techniques, and mobile apps can make the solution more comprehensive, ensuring that medical teams have access to up-to-date information from diverse sources.
5. **Natural Language Processing (NLP) for Records:** Future iterations can leverage NLP techniques to analyse unstructured medical records or emergency room notes, extracting relevant data for quick decision-making.
6. **User-Friendly Interfaces:** Developing intuitive dashboards and visualizations for medical professionals can enhance their ability to quickly interpret emergency data and make informed decisions in high-pressure situations.
7. **Automated Response Protocols:** Future developments could include fully automated response suggestions, such as recommending treatments based on patient symptoms, medical history, and real-time data, to assist healthcare professionals in making decisions faster.
8. **Collaboration with Frameworks:** Establishing partnerships with servers, emergency responders, and other institutions can allow for the continuous sharing of educational data and best practices, enhancing the techniques's effectiveness and accuracy.
9. **Advanced Decision Support Techniques:** Implementing advanced decision support techniques, such as prioritization algorithms based on patient severity or available resources, could improve triage processes and ensure optimal care.
10. **Regulatory Compliance and Data Privacy:** Ensuring that the techniques adheres to healthcare regulations such as HIPAA, GDPR, and other data protection laws will make it applicable in diverse healthcare environments and ensure patient data privacy and security.

This techniques's future development holds the potential to revolutionize emergency medical care, improving outcomes by enabling faster, more accurate, and data-driven decision-making.

REFERENCES

- [1] Shah, M.N "The formation of the Emergency Medical Services Techniques." *American Journal of Public health*, 96(3), 414-423 2006.
- [2] Murray C.J and Lopez A.D " Measuring the Global Burden of Disease". *New England Journal* , 369(5),448-457 2017.
- [3] Blackwell, T.H and Kaufman, J.S "Response Time Effectiveness: Comparison of Response Time and Mortality In An Urban Emergency Medical Service Station." *Academic Emergency medicine*, vol. 9(4),288-295 2022.
- [4] Sasser, S.M., Hunt, R.C., Faul, M., Sugerman, "Guidelines for field triage of injured patients." *MMWR Recommendations and Reports* pp. 61(1),1-20 2012
- [5] Roudsari, B.S., and Nathens, A.B. "Emergency Medical Services and Trauma Care Techniques." *The Lancet*, 367(9529),2141-2149 2006
- [6] Niska, R., Bhuiya, F., and Xu, J. "National Server Ambulatory Medical Care Survey" *national health statistics report*, vol. 26(6), 1-31 2010
- [7] Razzak, J.A., and Kellermann " Developing Countries." *Bulletin of the World Health Organization*, vol 80(11), 900–905 2002
- [8] Kobusingye, O.C., Hyder, A.A., Bishai, D., Hicks, E. R., Mock, C., & Joshipura, M "Emergency Medical Techniques in low- and Middle-Income Countries." *Bulletin of the World Health Organization* vol. 83(8), 626–631 2005
- [9] Hirshon, J. M., Risko, N., Calvello, E.J., "Health Techniques and Services: the role of acute care." *Bulletin of the World Health Organization*, vol. 91(5), 386–388 2013
- [10] Brooks, S. C., Schmicker, R. H., Rea, T. D. "Out of Server Cardiac Arrest in North America." *Resuscitation*, vol. 85(5), 607–615 2014
- [11] Ma, M., Han, L., Zhou, C. "Research and Application of Transformer-Based Anomaly Detection Model." *Proceedings of the 2020 International Conference on Artificial Intelligence and Computer Engineering* (ICAICE), pp. 1-5, 2020.

APPENDIX-A

PSEUDOCODE

FirestoreConfig.js

```
import firebase from 'firebase';
import 'firebase/auth';
import 'firebase/firestore';

const firebaseConfig = {
  apiKey: "AIzaSyDaiX8RfPbuIMdzutXsjn-43KgHpqBePRc",
  authDomain: "lifesaver-73541.firebaseio.com",
  INTERNSHIPId: "lifesaver-73541",
  storageBucket: "lifesaver-73541.appspot.com",
  messagingSenderId: "875438326254",
  appId: "1:875438326254:web:032cf9ef033017b2d20eff",
  measurementId: "G-NCLR6RZXE6"
};

// Initialize Firebase
if (!firebase.apps.length) {
  firebase.initializeApp(firebaseConfig);
} else {
  firebase.app(); // Use the existing app
}

// Export Firebase services
const auth = firebase.auth();
const db = firebase.firestore();

export { auth, db };
```

Appnavigator.js

```
import React from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import SplashScreen from '../screens/SplashScreen';
import HomeScreen from '../screens/HomeScreen';
import LoginScreen from '../screens/LoginScreen';
import RegisterScreen from '../screens/RegisterScreen';
import ProfileScreen from '../screens/ProfileScreen';
import ChatScreen from '../screens/ChatScreen'; // Correct import for ChatScreen
import ParentComponent from '../components/ParentComponent';
import EmergencyForm from '../components/EmergencyForm';
import AmbulanceRequest from '../components/AmbulanceRequest';
```

```

import ServerSearch from '../components/ServerSearch';
import PrimaryAid from '../components/PrimaryAid';
import BloodBank from '../components/BloodBank';
import ElderlyMonitoring from '../components/ElderlyMonitoring';

const Stack = createStackNavigator();

const AppNavigator = () => {
  return (
    <Stack.Navigator initialRouteName="Splash">
      <Stack.Screen name="Splash" component={SplashScreen} options={{ headerShown:
false }} />
      <Stack.Screen name="Login" component={LoginScreen} options={{ headerShown: false
}} />
      <Stack.Screen name="Register" component={RegisterScreen} options={{ headerShown:
false }} />
      <Stack.Screen name="Home" component={HomeScreen} options={{ title: 'Home',
headerShown: true }} />
      <Stack.Screen name="Profile" component={ParentComponent} options={{ title: 'Profile',
headerShown: true }} />
      <Stack.Screen name="EmergencyForm" component={EmergencyForm} options={{ title:
'Emergency Form', headerShown: true }} />
      <Stack.Screen name="AmbulanceRequest" component={AmbulanceRequest} options={{
title: 'Request Ambulance', headerShown: true }} />
      <Stack.Screen name="ServerSearch" component={ServerSearch} options={{ title:
'Search Servers', headerShown: true }} />
      <Stack.Screen name="PrimaryAid" component={PrimaryAid} options={{ title: 'Primary
Aid', headerShown: true }} />
      <Stack.Screen name="BloodBank" component={BloodBank} options={{ title: 'Blood
Bank', headerShown: true }} />
      <Stack.Screen name="ElderlyMonitoring" component={ElderlyMonitoring} options={{
title: 'Elderly Monitoring', headerShown: true }} />
      <Stack.Screen name="Chat" component={ChatScreen} options={{ title: 'Chat',
headerShown: true }} />
    </Stack.Navigator>
  );
};

export default AppNavigator;

```

Chatscreen.js

```

import React, { useState } from 'react';
import { View, Text, TextInput, Button, StyleSheet, Alert, ActivityIndicator } from 'react-
native';
import axios from 'axios';

```

```

// Update this URL to your local Flask API endpoint
const CHAT_API_URL = 'http://192.168.0.102:5002/chat'; // Your Flask server URL

```

```

const ChatScreen = () => {

```

```

const [message, setMessage] = useState("");
const [response, setResponse] = useState("");
const [loading, setLoading] = useState(false); // Loading state

const handleSendMessage = async () => {
  setLoading(true); // Set loading state to true when sending the message
  try {
    const res = await axios.post(CHAT_API_URL, {
      message: message, // Send the user message to the Flask API
    });

    setResponse(res.data.response); // Set the chatbot's response
  } catch (error) {
    console.error("Error:", error);
    Alert.alert("Error", "Failed to send message. Please try again later.");
  } finally {
    setLoading(false); // Set loading state to false after the response
  }
};

return (
  <View style={styles.container}>
    <Text style={styles.title}>Chat with Bot</Text>
    <TextInput
      style={styles.input}
      placeholder="Type your message"
      value={message}
      onChangeText={setMessage}
      multiline={true} // Allow multiple lines
      numberOfLines={4} // Set number of lines for multiline input
    />
    <Button title="Send" onPress={handleSendMessage} disabled={loading} />
    {loading && <ActivityIndicator size="large" color="#0000ff" style={styles.loader} />}
    {response ? <Text style={styles.response}>Bot Response: {response}</Text> : null}
  </View>
);
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    padding: 20,
  },
  title: {
    fontSize: 24,
    marginBottom: 20,
  },
  input: {

```

```
width: '100%',
padding: 10,
marginBottom: 20,
borderWidth: 1,
borderColor: '#ccc',
borderRadius: 5,
textAlignVertical: 'top', // To align the text at the top when multiline
},
response: {
  marginTop: 20,
  fontSize: 16,
  fontWeight: 'bold',
},
loader: {
  marginTop: 20,
},
});
```

```
export default ChatScreen;
```

HOME SCREEN.JS

```
import React, { useState } from 'react';
import { View, Text, StyleSheet, ScrollView, TouchableOpacity, Animated,
ImageBackground, Alert, ActivityIndicator } from 'react-native';
import { LinearGradient } from 'expo-linear-gradient';
import { MaterialCommunityIcons } from '@expo/vector-icons';
import * as WebBrowser from 'expo-web-browser'; // Import expo-web-browser for external
URL opening
```

```
const CHAT_URL = 'http://127.0.0.1:5002/'; // Your local Flask server URL
```

```
const HomeScreen = ({ navigation }) => {
  const [loading, setLoading] = useState(false); // Loading state for the chat button response
  const scaleValue = new Animated.Value(1);
```

```
  const handlePressIn = () => {
    Animated.spring(scaleValue, {
      toValue: 0.95,
      useNativeDriver: true,
    }).start();
  };
};
```

```
  const handlePressOut = () => {
    Animated.spring(scaleValue, {
      toValue: 1,
      friction: 3,
      tension: 40,
      useNativeDriver: true,
    }).start();
  };
};
```

```
// Function to handle chat button press and open in external browser
const handleChatPress = async () => {
  setLoading(true); // Set loading state to true when opening the browser
  try {
    // Open the chatbot in an external browser
    await WebBrowser.openBrowserAsync(CHAT_URL);
  } catch (error) {
    console.error('Error opening the browser:', error);
    Alert.alert("Error", "There was an error opening the chat.");
  } finally {
    setLoading(false); // Set loading state to false after the action
  }
};

return (
  <ImageBackground
    source={{ uri: 'https://example.com/background-image.jpg' }}
    style={styles.background}
  >
    <View style={styles.container}>
      { /* Scrollable Buttons */ }
      <ScrollView
        horizontal
        pagingEnabled
        showsHorizontalScrollIndicator={false}
        contentContainerStyle={styles.scrollContainer}
      >
        { /* First Column of Buttons */ }
        <View style={styles.columnContainer}>
          <AnimatedButton
            onPress={() => navigation.navigate('AmbulanceRequest')}
            icon="ambulance"
            text="Request Ambulance"
            scaleValue={scaleValue}
            onPressIn={handlePressIn}
            onPressOut={handlePressOut}
          />
          <AnimatedButton
            onPress={() => navigation.navigate('ServerSearch')}
            icon="server"
            text="Search Server" scaleValue={scaleValue}
            onPressIn={handlePressIn}
            onPressOut={handlePressOut}
          />
        </View>

        { /* Second Column of Buttons */ }
        <View style={styles.columnContainer}>
```

```

<AnimatedButton
  onPress={() => navigation.navigate('BloodBank')}
  icon="blood-bag"
  text="Blood Bank"
  scaleValue={scaleValue}
  onPressIn={handlePressIn}
  onPressOut={handlePressOut}
/>
<AnimatedButton
  onPress={() => navigation.navigate('ElderlyMonitoring')}
  icon="watch"
  text="Elderly Monitoring"
  scaleValue={scaleValue}
  onPressIn={handlePressIn}
  onPressOut={handlePressOut}
/>
<AnimatedButton
  onPress={() => navigation.navigate('EmergencyForm')}
  icon="form-textbox"
  text="Emergency Form"
  scaleValue={scaleValue}
  onPressIn={handlePressIn}
  onPressOut={handlePressOut}
/>
</View>
</ScrollView>

{/* Chat Button (Bottom Right Corner) */}
<TouchableOpacity
  style={styles.chatButton}
  onPress={handleChatPress}
  disabled={loading} // Disable chat button while loading
>
  <MaterialCommunityIcons name="chat" size={24} color="#FFFFFF" />
</TouchableOpacity>

{/* Loader while waiting for response */}
{loading && <ActivityIndicator size="large" color="#0000ff" style={styles.loader} />}

{/* Profile and Primary Aid Buttons (Aligned Horizontally) */}
<View style={styles.topButtonContainer}>
  <TouchableOpacity
    style={styles.primaryAidButton}
    onPress={() => navigation.navigate('PrimaryAid')}
  >
    <MaterialCommunityIcons name="help-circle" size={24} color="#FFFFFF" />
    <Text style={styles.primaryAidText}>Primary Aid</Text>
  </TouchableOpacity>

  <TouchableOpacity

```

```

        style={styles.profileButton}
        onPress={() => navigation.navigate('Profile')}
      >
        <MaterialCommunityIcons name="account" size={24} color="#FFFFFF" />
        <Text style={styles.profileButtonText}>Profile</Text>
      </TouchableOpacity>
    </View>

    { /* Footer */}
    <View style={styles.footer}>
      <Text style={styles.footerText}>LifeSaver App © 2024</Text>
    </View>
  </View>
</ImageBackground>
);
};

const AnimatedButton = ({ onPress, icon, text, scaleValue, onPressIn, onPressOut }) => (
  <Animated.View style={{ transform: [{ scale: scaleValue }] }}>
    <TouchableOpacity
      style={styles.button}
      onPress={onPress}
      onPressIn={onPressIn}
      onPressOut={onPressOut}
      activeOpacity={0.8}
    >
      <LinearGradient
        colors={['#43C6AC', '#191654']}
        start={[0, 0]}
        end={[1, 1]}
        style={styles.gradient}
      >
        <MaterialCommunityIcons name={icon} size={24} color="#FFFFFF" />
        <Text style={styles.buttonText}>{text}</Text>
      </LinearGradient>
    </TouchableOpacity>
  </Animated.View>
);

const styles = StyleSheet.create({
  background: {
    flex: 1,
    width: '100%',
    height: '100%',
  },
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    padding: 20,
  },
});

```

```
},
scrollContainer: {
  flexDirection: 'row',
  width: '200%',
},
columnContainer: {
  flex: 1,
  justifyContent: 'center',
  alignItems: 'center',
  padding: 10,
  width: '100%',
},
button: {
  borderRadius: 12,
  marginBottom: 15,
  width: '80%',
  shadowColor: '#000',
  shadowOffset: { width: 0, height: 5 },
  shadowOpacity: 0.25,
  shadowRadius: 6.84,
  elevation: 7,
},
gradient: {
  flexDirection: 'row',
  alignItems: 'center',
  justifyContent: 'center',
  padding: 15,
  borderRadius: 12,
},
buttonText: {
  color: 'FFFFFF',
  fontSize: 18,
  fontWeight: 'bold',
  marginLeft: 10,
},
chatButton: {
  position: 'absolute',
  bottom: 20,
  right: 20,
  backgroundColor: '#FF5722',
  padding: 15,
  borderRadius: 30,
  alignItems: 'center',
  shadowColor: '#000',
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.25,
  shadowRadius: 3.84,
  elevation: 5,
},
loader: {
```

```
    marginTop: 20,
  },
  topButtonContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    width: '100%',
    position: 'absolute',
    top: 40,
    paddingHorizontal: 20,
  },
  profileButton: {
    backgroundColor: '#4CAF50',
    padding: 15,
    borderRadius: 30,
    flexDirection: 'row',
    alignItems: 'center',
    shadowColor: '#000',
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.25,
    shadowRadius: 3.84,
    elevation: 5,
  },
  profileButtonText: {
    color: 'FFFFFF',
    marginLeft: 5,
    fontWeight: 'bold',
  },
  primaryAidButton: {
    backgroundColor: '#4CAF50',
    padding: 15,
    borderRadius: 30,
    flexDirection: 'row',
    alignItems: 'center',
    shadowColor: '#000',
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.25,
    shadowRadius: 3.84,
    elevation: 5,
  },
  primaryAidText: {
    color: 'FFFFFF',
    marginLeft: 5,
    fontWeight: 'bold',
  },
  footer: {
    position: 'absolute',
    bottom: 10,
    alignItems: 'center',
  },
  footerText: {
```

```
    color: '#000000',  
    fontSize: 12,  
  },  
});
```

```
export default HomeScreen;
```

Login screen.js

```
import React, { useState } from 'react';  
import { View, Text, TextInput, TouchableOpacity, StyleSheet, Alert } from 'react-native';  
import { auth, db } from '../Firebase/FirebaseConfig'; // Ensure you import 'db' here  
  
const LoginScreen = ({ navigation }) => {  
  const [email, setEmail] = useState("");  
  const [password, setPassword] = useState("");  
  
  const handleLogin = async () => {  
    if (!email || !password) {  
      Alert.alert('Error', 'Please enter email and password.');      return;  
    }  
  
    try {  
      const userCredential = await auth.signInWithEmailAndPassword(email, password);  
      const user = userCredential.user;  
  
      // Fetch additional user data from Firestore (if needed)  
      const userDoc = await db.collection('users').doc(user.uid).get(); // Use 'db' here  
      const userData = userDoc.data();  
  
      // Check if user data exists  
      if (userData) {  
        Alert.alert('Success', `Welcome back, ${userData.email}!`);  
        navigation.replace('Home'); // Navigate to home screen after successful login  
      } else {  
        Alert.alert('Error', 'User data not found in Firestore.');      }  
    } catch (error) {  
      // Log the error to help debug  
      console.error('Login Error:', error.message);  
      Alert.alert('Error', error.message);  
    }  
  };  
  
  return (  
    <View style={styles.container}>  
      <Text style={styles.title}>Login</Text>  
      <TextInput
```

```

        style={styles.input}
        placeholder="Email"
        value={email}
        onChangeText={setEmail}
        keyboardType="email-address"
      />
      <TextInput
        style={styles.input}
        placeholder="Password"
        secureTextEntry
        value={password}
        onChangeText={setPassword}
      />
      <TouchableOpacity style={styles.button} onPress={handleLogin}>
        <Text style={styles.buttonText}>Login</Text>
      </TouchableOpacity>
      <Text style={styles.registerPrompt}>
        New member?{' '}
        <Text style={styles.registerLink} onPress={() => navigation.navigate('Register')}>
          Register here
        </Text>
      </Text>
    </View>
  );
};

```

```

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#E0F7FA',
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    marginBottom: 20,
    color: '#00796B',
  },
  input: {
    height: 40,
    width: '80%',
    borderColor: '#00796B',
    borderWidth: 1,
    marginBottom: 15,
    paddingHorizontal: 10,
    borderRadius: 5,
  },
  button: {
    backgroundColor: '#009688',

```

```
padding: 12,
borderRadius: 5,
width: '80%',
alignItems: 'center',
marginVertical: 8,
},
buttonText: {
  color: '#FFFFFF',
  fontSize: 16,
  fontWeight: 'bold',
},
registerPrompt: {
  marginTop: 15,
  color: '#00796B',
},
registerLink: {
  color: '#00796B',
  fontWeight: 'bold',
  textDecorationLine: 'underline',
},
});

export default LoginScreen;
```

Profile Screen.js

```
import React, { useState } from 'react';
import { View, Text, TextInput, Button, StyleSheet, Image, TouchableOpacity, Alert,
ScrollView } from 'react-native';
import * as ImagePicker from 'expo-image-picker'; // Ensure you have this library
import { Picker } from '@react-native-picker/picker'; // Picker for gender and date selection

const ProfileForm = ({ user = {}, onUpdate }) => {
  const [name, setName] = useState(user.name || '');
  const [email, setEmail] = useState(user.email || '');
  const [address, setAddress] = useState(user.address || '');
  const [phone, setPhone] = useState(user.phone || '');
  const [day, setDay] = useState(user.dob ? new Date(user.dob).getDate().toString() : ''); //
  Changed to string
  const [month, setMonth] = useState(user.dob ? (new Date(user.dob).getMonth() +
  1).toString() : ''); // Changed to string
  const [year, setYear] = useState(user.dob ? new Date(user.dob).getFullYear().toString() : '');
  // Changed to string
  const [gender, setGender] = useState(user.gender || '');
  const [emergencyContact, setEmergencyContact] = useState(user.emergencyContact || '');
  const [medicalHistory, setMedicalHistory] = useState(user.medicalHistory || '');
  const [profilePicture, setProfilePicture] = useState(user.profilePicture || null);
  const [formattedDob, setFormattedDob] = useState(user.dob || ''); // Store the formatted
  DOB
```

```
const handleUpdate = () => {
  // Validate user inputs
  if (!name || !email || !address || !phone || !gender || !emergencyContact) {
    Alert.alert('Validation Error', 'Please fill in all required fields.');
```

return;

```
  }

  // Logic to update user profile (API call)
  console.log('Profile updated:', { name, email, address, phone, dob: formattedDob, gender,
emergencyContact, medicalHistory, profilePicture });
  onUpdate({ name, email, address, phone, dob: formattedDob, gender, emergencyContact,
medicalHistory, profilePicture });
  Alert.alert('Success', 'Profile updated successfully!'); // Show success message
};

const handleImagePick = async () => {
  const permissionResult = await ImagePicker.requestMediaLibraryPermissionsAsync();
  if (!permissionResult.granted) {
    Alert.alert('Permission Required', 'Permission to access camera roll is required!');
    return;
  }

  const result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.All,
    allowsEditing: true,
    aspect: [4, 3],
    quality: 1,
  });

  if (!result.cancelled) {
    setProfilePicture(result.uri);
  }
};

const handleDeletePicture = () => {
  setProfilePicture(null); // Reset profile picture
};

const handleReset = () => {
  setName(user.name || "");
  setEmail(user.email || "");
  setAddress(user.address || "");
  setPhone(user.phone || "");
  setDay(user.dob ? new Date(user.dob).getDate().toString() : "");
  setMonth(user.dob ? (new Date(user.dob).getMonth() + 1).toString() : "");
  setYear(user.dob ? new Date(user.dob).getFullYear().toString() : "");
  setGender(user.gender || "");
  setEmergencyContact(user.emergencyContact || "");
  setMedicalHistory(user.medicalHistory || "");
```

```

setProfilePicture(user.profilePicture || null);
setFormattedDob(""); // Reset formatted DOB
};

// Handle the confirmation of date of birth
const handleConfirmDob = () => {
  const dob = `${day}/${month}/${year}`;
  setFormattedDob(dob); // Store the formatted DOB
};

return (
  <ScrollView contentContainerStyle={styles.container}>
    <Text style={styles.title}>Edit Profile</Text>

    <TouchableOpacity onPress={handleImagePick} style={styles.imageContainer}>
      {profilePicture ? (
        <Image source={{ uri: profilePicture }} style={styles.profileImage} />
      ) : (
        <Text style={styles.imagePlaceholder}>Upload Profile Picture</Text>
      )}
    </TouchableOpacity>

    {profilePicture && (
      <Button title="Delete Picture" onPress={handleDeletePicture} color="#D32F2F" />
    )}

    <TextInput
      style={styles.input}
      placeholder="Name"
      value={name}
      onChangeText={setName}
    />
    <TextInput
      style={styles.input}
      placeholder="Email"
      value={email}
      onChangeText={setEmail}
      keyboardType="email-address"
    />
    <TextInput
      style={styles.input}
      placeholder="Address"
      value={address}
      onChangeText={setAddress}
    />
    <TextInput
      style={styles.input}
      placeholder="Phone Number"
      value={phone}
      onChangeText={setPhone}

```

```

        keyboardType="phone-pad"
      />

      { /* Date of Birth Section */}
      <Text style={styles.label}>Date of Birth</Text>
      <View style={styles.datePickerContainer}>
        <TextInput
          style={styles.dateInput}
          placeholder="DD"
          value={day}
          onChangeText={setDay}
          keyboardType="numeric"
          maxLength={2}
        />
        <TextInput
          style={styles.dateInput}
          placeholder="MM"
          value={month}
          onChangeText={setMonth}
          keyboardType="numeric"
          maxLength={2}
        />
        <TextInput
          style={styles.dateInput}
          placeholder="YYYY"
          value={year}
          onChangeText={setYear}
          keyboardType="numeric"
          maxLength={4}
        />
      </View>
      <Button title="OK" onPress={handleConfirmDob} color="#00796B" />

      {formattedDob && ( // Display formatted date of birth after confirmation
        <Text style={styles.confirmedDob}>
          Confirmed Date of Birth: {formattedDob}
        </Text>
      )}

      <Picker
        selectedValue={gender}
        style={styles.picker}
        onValueChange={(itemValue) => setGender(itemValue)}
      >
        <Picker.Item label="Select Gender" value="" />
        <Picker.Item label="Male" value="male" />
        <Picker.Item label="Female" value="female" />
        <Picker.Item label="Other" value="other" />
      </Picker>
      <TextInput

```

```

        style={styles.input}
        placeholder="Emergency Contact"
        value={emergencyContact}
        onChangeText={setEmergencyContact}
      />
      <TextInput
        style={styles.input}
        placeholder="Medical History (e.g. allergies, chronic conditions)"
        value={medicalHistory}
        onChangeText={setMedicalHistory}
        multiline={true}
        numberOfLines={3}
      />

      <View style={styles.buttonContainer}>
        <View style={styles.buttonWrapper}>
          <Button title="Update Profile" onPress={handleUpdate} color="#00796B" />
        </View>
        <View style={styles.buttonWrapper}>
          <Button title="Reset" onPress={handleReset} color="#FFC107" />
        </View>
      </View>
    </ScrollView>
  );
};

const styles = StyleSheet.create({
  container: {
    flexGrow: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5F5F5',
    padding: 20,
  },
  title: {
    fontSize: 24,
    marginBottom: 20,
  },
  imageContainer: {
    width: 100,
    height: 100,
    borderRadius: 50,
    borderColor: '#00796B',
    borderWidth: 2,
    justifyContent: 'center',
    alignItems: 'center',
    marginBottom: 20,
  },
  profileImage: {
    width: '100%',

```

```
    height: '100%',
    borderRadius: 50,
  },
  imagePlaceholder: {
    color: '#00796B',
    fontWeight: 'bold',
  },
  input: {
    height: 40,
    width: '100%',
    borderColor: '#00796B',
    borderWidth: 1,
    borderRadius: 5,
    paddingLeft: 10,
    marginBottom: 15,
  },
  label: {
    fontSize: 16,
    marginBottom: 5,
  },
  datePickerContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    width: '100%',
    marginBottom: 15,
  },
  dateInput: {
    flex: 1,
    borderColor: '#00796B',
    borderWidth: 1,
    borderRadius: 5,
    padding: 10,
    marginHorizontal: 5,
  },
  confirmedDob: {
    fontSize: 16,
    color: '#4CAF50',
    marginTop: 10,
  },
  picker: {
    height: 50,
    width: '100%',
    marginBottom: 15,
  },
  buttonContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    width: '100%',
    marginTop: 20,
  },
}
```

```
buttonWrapper: {  
  flex: 1,  
  marginHorizontal: 5,  
},  
});  
  
export default ProfileForm;
```

Register Screen.js

```
import React, { useState } from 'react';  
import { View, Text, TextInput, TouchableOpacity, StyleSheet, Alert } from 'react-native';  
import firebase from 'firebase'; // Firebase v8 import  
import { auth, db } from '../Firebase/FirebaseConfig'; // Import Firebase auth and Firestore  
  
const RegisterScreen = ({ navigation }) => {  
  const [email, setEmail] = useState("");  
  const [password, setPassword] = useState("");  
  
  const handleRegister = async () => {  
    if (email === "") {  
      Alert.alert('Error', 'Please enter your email.');      return;  
    }  
  
    if (password === "") {  
      Alert.alert('Error', 'Please enter your password.');      return;  
    }  
  
    if (password.length < 6) {  
      Alert.alert('Error', 'Password should be at least 6 characters long.');      return;  
    }  
  
    try {  
      // Register user with Firebase Authentication  
      const userCredential = await auth.createUserWithEmailAndPassword(email, password);  
      const user = userCredential.user;  
  
      // After successful registration, store additional user data in Firestore  
      await db.collection('users').doc(user.uid).set({  
        email: user.email,  
        // You can add more fields like name, profile picture, etc.  
      });  
  
      Alert.alert('Registration Successful', 'You can now log in!');  
      navigation.navigate('Login'); // Navigate to Login screen after successful registration
```

```
} catch (error) {
  Alert.alert('Error', `An error occurred while registering: ${error.message}`);
  console.error('Error registering user:', error);
}
};

return (
  <View style={styles.container}>
    <Text style={styles.title}>Register</Text>
    <TextInput
      style={styles.input}
      placeholder="Email"
      value={email}
      onChangeText={setEmail}
      autoCapitalize="none"
      keyboardType="email-address"
    />
    <TextInput
      style={styles.input}
      placeholder="Password"
      secureTextEntry
      value={password}
      onChangeText={setPassword}
    />
    <TouchableOpacity style={styles.button} onPress={handleRegister}>
      <Text style={styles.buttonText}>Register</Text>
    </TouchableOpacity>
    <TouchableOpacity style={[styles.button, styles.backButton]} onPress={() =>
navigation.navigate('Login')}>
      <Text style={styles.buttonText}>Back to Login</Text>
    </TouchableOpacity>
  </View>
);
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#E0F7FA',
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    marginBottom: 20,
    color: '#00796B',
  },
  input: {
    height: 40,
```

```
width: '80%',
borderColor: '#00796B',
borderWidth: 1,
marginBottom: 15,
paddingHorizontal: 10,
borderRadius: 5,
},
button: {
  backgroundColor: '#009688',
  padding: 12,
  borderRadius: 5,
  width: '80%',
  alignItems: 'center',
  marginVertical: 8,
},
buttonText: {
  color: 'FFFFFF',
  fontSize: 16,
  fontWeight: 'bold',
},
backButton: {
  marginTop: 10,
},
});
```

```
export default RegisterScreen;
```

SplashScreen.js

```
import React, { useEffect } from 'react';
import { View, Text, StyleSheet } from 'react-native';

const SplashScreen = ({ navigation }) => {
  useEffect(() => {
    const timer = setTimeout(() => {
      navigation.replace('Login'); // Navigate to LoginScreen after 2 seconds
    }, 2000);

    return () => clearTimeout(timer); // Cleanup the timer
  }, [navigation]);

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to LifeSaver</Text>
    </View>
  );
};

const styles = StyleSheet.create({
```

```
container: {
  flex: 1,
  justifyContent: 'center',
  alignItems: 'center',
  backgroundColor: '#E0F7FA',
},
title: {
  fontSize: 24,
  fontWeight: 'bold',
  color: '#00796B',
},
});
```

```
export default SplashScreen;
```

App.js

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import AppNavigator from './navigator/AppNavigator';
```

```
const App = () => {
  return (
    <NavigationContainer>
      <AppNavigator />
    </NavigationContainer>
  );
};
```

```
export default App;
```

Allinone.py

```
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from gtts import gTTS
import os
import uuid
import pygame
import uvicorn
from dotenv import load_dotenv
import google.generativeai as genai
import requests
from datetime import datetime
from flask import Flask, jsonify
```

```
# Initialize FastAPI app
app = FastAPI()
```



```
# Enable CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # Allow all origins
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Directory for storing temporary audio files
TEMP_AUDIO_DIR = "temp_audio"
os.makedirs(TEMP_AUDIO_DIR, exist_ok=True)

# Initialize pygame mixer (for audio playback)
pygame.mixer.init()

# Text-to-Speech functionality
class TextToSpeechRequest(BaseModel):
    text: str
    language: str = "en" # Default language is English

@app.post("/text-to-speech/")
async def text_to_speech(request: TextToSpeechRequest):
    """
    Convert text to speech and play it locally using pygame.
    """
    if not request.text.strip():
        raise HTTPException(status_code=400, detail="Text cannot be empty.")

    try:
        tts = gTTS(text=request.text, lang=request.language)
        audio_file = f"{TEMP_AUDIO_DIR}/{uuid.uuid4().hex}.mp3"
        tts.save(audio_file)

        pygame.mixer.music.load(audio_file)
        pygame.mixer.music.play()

        while pygame.mixer.music.get_busy():
            pygame.time.Clock().tick(10)

        return {"message": "Audio is playing locally."}
    except ValueError as e:
        raise HTTPException(status_code=400, detail=f"Invalid language code: {request.language}")
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

# Medical Chatbot functionality
preLoadData = """You are a medical diagnostic assistant. Provide a diagnosis and
recommendations based on the following input.
```

Format your response as follows:

- Name: [Patient's Name]
- Age: [Patient's Age]
- Assumed Issues: [List of possible issues]
- Most Common Solution: [Suggested solution]
- If Required, Recommended Medication: [List of medications]
- Contact: [Which doctor to meet: Cardiology/Orthopedics/...etc.]""

```
class MedicalChatbot:
```

```
    def __init__(self):
        load_dotenv() # Load environment variables
        api_key = os.getenv("GOOGLE_API_KEY")
        if not api_key:
            print("Invalid/No API Key in .env file")
            return
        genai.configure(api_key=api_key)
        self.model = genai.GenerativeModel("gemini-pro")
        self.chat = self.model.start_chat(history=[])
```

```
    def check_internet_connection(self):
        try:
            requests.get("https://www.google.com", timeout=5)
            return True
        except requests.exceptions.ReadTimeout:
            return False
        except requests.exceptions.RequestException:
            return False
```

```
    def get_medical_response(self, member):
        name = member.get('name')
        age = member.get('age')
        gender = member.get('gender')
        diseases = member.get('diseases')
        message = member.get('message')
```

```
        question = f"where the given data of Patient are> Name: {name}, Age: {age}, Gender: {gender}, Diseases: {diseases}, Message: {message}"
        full_prompt = preLoadData + "\n" + question
```

```
        if not self.check_internet_connection():
            return {"message": "Sorry, can't connect to the internet. Please check your connection."}
```

```
        response = self.chat.send_message(full_prompt, stream=True)
        for chunk in response:
            if hasattr(chunk, 'text'):
                return {"message": " ".join(chunk.text for chunk in response)}
```

```
        return {"message": "No valid response received from the AI model."}
```

```
# Initialize chatbot
chatbot = MedicalChatbot()

# Model for medical chatbot input
class MedicalChatRequest(BaseModel):
    memberName: str
    dob: str
    gender: str
    diseases: str
    message: str

@app.post("/chat/")
async def chat(request: MedicalChatRequest):
    """
    Handle medical chat queries and get diagnosis from the chatbot.
    """
    name = request.memberName
    dob = request.dob
    gender = request.gender
    diseases = request.diseases
    message = request.message

    age = calculate_age(dob) if dob else None

    member = {
        'name': name,
        'age': age,
        'gender': gender,
        'diseases': diseases,
        'message': message
    }

    response = chatbot.get_medical_response(member)
    return response

def calculate_age(dob):
    try:
        dob_date = datetime.strptime(dob, '%Y-%m-%dT%H:%M:%S.%fZ')
    except ValueError:
        # Handle the case where time info is not provided, just date
        dob_date = datetime.strptime(dob, '%Y-%m-%d') # Adjust format if no time info
    today = datetime.today()
    age = today.year - dob_date.year
    if today.month < dob_date.month or (today.month == dob_date.month and today.day <
dob_date.day):
        age -= 1
    return age

# Run the app with uvicorn when this script is executed
```

```
if __name__ == "__main__":  
    uvicorn.run(app, host="0.0.0.0", port=8080)
```

GEMINI CHATBOT.PY

```
from dotenv import load_dotenv
```

```
import os
```

```
import google.generativeai as genai
```

```
import requests
```

```
preLoadData = """You are a medical diagnostic assistant. Provide a diagnosis and  
recommendations based on the following input.
```

```
Format your response as follows:
```

```
- Name: [Patient's Name]
```

```
- Age: [Patient's Age]
```

```
- Assumed Issues: [List of possible issues]
```

```
- Most Common Solution: [Suggested solution]
```

```
- If Required, Recommended Medication: [List of medications]
```

```
- Contact: [Which doctor to meet: Cardiology/Orthopedics/...etc.] (Suggest the doctor which is  
required for this treatment)"""
```

```
class MedicalChatbot:
```

```
    def __init__(self):
```

```
        print("""Hi! This is BeeBotix, the creator of this code.
```

```
Just a friendly reminder to create a .env file and set your GOOGLE_API_KEY environment  
variable before using this program.
```

```
Thank you!
```

```
===== Happy Chatting =====""")
```

```
    load_dotenv() # Load environment variables from .env file
```

```
    api_key = os.getenv("GOOGLE_API_KEY")
```

```
    if not api_key:
```

```
        print("Invalid/No API Key in .env file")
```

```
        return
```

```
    genai.configure(api_key=api_key)
```

```
    self.model = genai.GenerativeModel("gemini-pro")
```

```
    self.chat = self.model.start_chat(history=[])
```

```
    def check_internet_connection(self):
```

```
        try:
```

```
            requests.get("https://www.google.com", timeout=1)
```

```
            return True
```

```
        except requests.ConnectionError:
```

```
            return False
```

```
    def get_medical_response(self, member):
```

```
        name = member.get('name')
```

```
        age = member.get('age')
```

```
        gender = member.get('gender')
```

```

diseases = member.get('diseases')
message = member.get('message')

question = f"where the given data of Patient are> Name: {name}, Age: {age}, Gender:
{gender}, Diseases: {diseases}, Message: {message}"
full_prompt = preLoadData + "\n" + question
print(full_prompt)

if not self.check_internet_connection():
    return {"message": "Sorry, can't connect to the internet. Please check your
connection."}

response = self.chat.send_message(full_prompt, stream=True)

# Check the safety ratings
for chunk in response:
    if hasattr(chunk, 'safety_ratings'):
        print(f"Safety Ratings: {chunk.safety_ratings}") # Log safety ratings
    if hasattr(chunk, 'text'):
        return {"message": " ".join(chunk.text for chunk in response)}

return {"message": "No valid response received from the AI model."}

'''
# Example usage
if __name__ == "__main__":
    chatbot = MedicalChatbot()
    # Simulate a request from the frontend
    sample_member = {
        'name': "John Doe",
        'age': 30,
        'gender': "Male",
        'diseases': "hypertension, diabetes",
        'message': "I've been feeling dizzy lately."
    }
    response = chatbot.get_medical_response(sample_member)
    print(response)
'''

```

HostedAI.py

```

from flask import Flask, request, jsonify
from gemini_chatbot import MedicalChatbot as GeminiChatbot
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

# Initialize the chatbot

```

```
chatbot = GeminiChatbot()
```

```
@app.route('/chat', methods=['POST'])
def chat():
    # Get data from the request
    data = request.get_json()

    message = data.get('message', '')
    name = data.get('memberName')
    dob = data.get('dob')
    gender = data.get('gender')
    diseases = data.get('diseases')

    # Calculate age from DOB if necessary
    age = calculate_age(dob) if dob else None

    # Create a member dictionary
    member = {
        'name': name,
        'age': age,
        'gender': gender,
        'diseases': diseases,
        'message': message
    }

    # Get the response from the chatbot
    response = chatbot.get_medical_response(member)
    return jsonify(response)

def calculate_age(dob):
    from datetime import datetime
    if dob:
        dob_date = datetime.strptime(dob, '%Y-%m-%dT%H:%M:%S.%fZ') # Adjust format to
        match the incoming data
        today = datetime.today()
        age = today.year - dob_date.year - ((today.month, today.day) < (dob_date.month,
        dob_date.day))
        return age
    return None

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

VoiceAPI.py

```
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from gtts import gTTS
import os
```

```
import uuid
import pygame
import uvicorn
from pydantic import BaseModel

# Initialize the FastAPI app
app = FastAPI()

# Enable CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # Update with specific origins if needed
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Directory to store temporary audio files
TEMP_AUDIO_DIR = "temp_audio"
os.makedirs(TEMP_AUDIO_DIR, exist_ok=True)

# Initialize pygame mixer (for audio playback)
pygame.mixer.init()

# Pydantic model for the request body
class TextToSpeechRequest(BaseModel):
    text: str
    language: str = "en" # Default language is 'en'

@app.post("/text-to-speech/")
async def text_to_speech(request: TextToSpeechRequest):
    """
    Convert text to speech and play the audio locally using pygame.

    Args:
        text (str): The text to convert to speech.
        language (str): Language code for speech synthesis.

    Returns:
        Message indicating the result of the speech generation.
    """
    # Validate input
    if not request.text.strip():
        raise HTTPException(status_code=400, detail="Text cannot be empty.")

    try:
        # Generate speech
        tts = gTTS(text=request.text, lang=request.language)
        audio_file = f"{TEMP_AUDIO_DIR}/{uuid.uuid4().hex}.mp3"
        tts.save(audio_file)
```

```
# Play the generated audio file using pygame
pygame.mixer.music.load(audio_file)
pygame.mixer.music.play()

# Wait until the music finishes playing before continuing
while pygame.mixer.music.get_busy(): # Check if music is still playing
    pygame.time.Clock().tick(10) # Adjust the tick rate as needed

return {"message": "Audio is playing locally."}
except ValueError as e:
    raise HTTPException(status_code=400, detail=f"Invalid language code:
{request.language}")
except Exception as e:
    raise HTTPException(status_code=500, detail=str(e))

# Run the app if the script is executed directly
if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8080)
```

Index.js

```
document.addEventListener("DOMContentLoaded", () => {
    const splashText = document.getElementById("splash-text");
    const mainPage = document.getElementById("main-page");

    const texts = [ "Namma
    Rakshane", "□□□□□
    □□□□□ ",
    "□□□□□ □□□□□ ",
    "□□ □□□□□ "
    ];

    let index = 0;

    const displayText = () => {
        if (index < texts.length) {
            // Fade in the text
            splashText.style.opacity = 0; // Start invisible
            setTimeout(() => {
                splashText.innerHTML = `<h1>${texts[index]}</h1>`;
                splashText.style.opacity = 1; // Fade in
            }, 1000); // Delay before fade-in

            // Fade out the text
            setTimeout(() => {
                splashText.style.opacity = 0; // Fade out
                index++;
            }, 1000); // Delay before fade-out
        }
    };
    displayText();
});
```

```
    }, 2000); // Stay visible for 3 seconds

    // Continue the animation
    setTimeout(displayText, 3000); // Total 4 seconds before next text
  } else {
    // Transition to the main page after all texts have been displayed
    setTimeout(() => {
      window.location.href = "main.html"; // Redirect to main.html
    }, 2000); // Wait for 2 seconds before going to the main page
  }
};

displayText();
});
```

Chatpage.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Chat - Namma Rakshane</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <link rel="stylesheet" href="/styles/chat.css">
</style>
  .chat-container {
    height: 70vh;
    overflow-y: auto;
    border: 1px solid #ccc;
    border-radius: 5px;
    padding: 10px;
    margin-bottom: 20px;
  }
  .chat-message {
    margin: 10px 0;
  }
  .user-message {
    text-align: right;
  }
  .reply-message {
    text-align: left;
    display: flex;
    justify-content: space-between;
    align-items: center;
  }
  .speak-button {
    background: transparent;
    border: none;
```

```
        font-size: 1.5rem;
        cursor: pointer;
    }
    .emergency-button {
        background-color: red;
        color: white;
        border: none;
    }
    .call-button {
        background-color: green;
        color: white;
        border: none;
        padding: 10px;
        margin-top: 10px;
        cursor: pointer;
    }
    .call-button:hover {
        background-color: darkgreen;
    }
    .server-info-box {
        border: 2px solid #28a745;
        border-radius: 5px;
        background-color: #f8f9fa;
        padding: 15px;
        margin-top: 15px;
    }
    .server-info-box h5 {
        margin: 0;
        font-size: 1.2rem;
        color: #28a745;
    }
    .server-info-box p {
        margin: 5px 0;
    }
    .server-info-box a { color:
        #007bff;
        text-decoration: none;
    }
    .server-info-box a:hover {
        text-decoration: underline;
    }
</style>
</head>
<body>
    <div class="container mt-5">
        <div class="d-flex justify-content-between align-items-center mb-3">
            <h2 id="memberName"></h2>
            <button class="emergency-button"
onclick="showEmergencyAlert()">Emergency</button>
        </div>
```

```
<p class="text-muted" id="chatPrompt">How can I help you today?</p>
<div class="chat-container" id="chatContainer"></div>

<form id="chatForm" class="input-group">
  <input type="text" id="chatInput" class="form-control" placeholder="Type your
message here..." required>
  <div class="input-group-append">
    <button class="btn btn-primary" type="submit"
id="sendButtonText">Send</button>
  </div>
</form>
</div>

<script>
const chatContainer = document.getElementById('chatContainer');
const memberNameElement = document.getElementById('memberName');
const chatForm = document.getElementById('chatForm');
const chatInput = document.getElementById('chatInput');
const chatPrompt = document.getElementById('chatPrompt');
const sendButtonText = document.getElementById('sendButtonText');

let servers = []; // To store server data

// Load the selected language from localStorage
const language = localStorage.getItem('language') || 'en';

// Translation object
const translations = {
  en: {
    memberGreeting: "Hi",
    chatPrompt: "How can I help you today?",
    sendButtonText: "Send",
  },
  hi: {
    memberGreeting: "",
    chatPrompt: "",
    sendButtonText: "",
  },
  kn: {
    memberGreeting: "",
    chatPrompt: "",
    sendButtonText: "",
  },
  te: {
    memberGreeting: "",
    chatPrompt: "",
    sendButtonText: "",
  }
};
```

```
// Get member details from localStorage
const member = JSON.parse(localStorage.getItem('currentMember'));

// Set member name on the page
if (member) {
  memberNameElement.innerText = `${translations[language].memberGreeting}
  ${member.name}`;
}

// Set chat prompt and button text based on selected language
chatPrompt.innerText = translations[language].chatPrompt;
sendButtonText.innerText = translations[language].sendButtonText;

// Show Emergency Alert
function showEmergencyAlert() {
  window.location.href = 'server.html';
}

// Fetch servers data from suprate.json
fetch('serverJSON.json')
  .then(response => response.json())
  .then(data => {
    servers = data; // Store the fetched data
  })
  .catch(error => {
    console.error("Error fetching servers data:", error);
  });

// Handle form submission
chatForm.onSubmit = async (event) => {
  event.preventDefault();
  const message = `${chatInput.value}` give reply in "${language}" language full`;
  console.log(message);
  appendMessage(message, 'user');
  chatInput.value = "";

  // Send message to the backend with selected language
  try {
    const response = await fetch('http://172.17.54.173:8080/chat', { // 8000
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        message,
        memberName: member.name,
        dob: member.dob,
        gender: member.gender,
        diseases: member.diseases,
      }),
    });
  }
}
```

```

    });

    if (response.ok) {
        const reply = await response.json();
        const replyMessage = reply.message;
        const serverInfo = findMatchingServer(replyMessage);

        if (serverInfo) {
            appendMessage(`${replyMessage}`, 'reply');
            appendServerInfoBox(serverInfo);
        } else {
            appendMessage(replyMessage, 'reply');
        }
    } else {
        appendMessage("Error: Unable to get reply", 'reply');
    }
} catch (error) {
    console.error('Error:', error);
    appendMessage("Error: Unable to communicate with server", 'reply');
}
};

// Check if any server matches the specialty in the AI's response
function findMatchingServer(replyMessage) {
    for (const server of servers) {
        if (replyMessage.toLowerCase().includes(server.special.toLowerCase())) {
            return server;
        }
    }
    return null;
}

// Append message to the chat container
function appendMessage(message, type) {
    const messageDiv = document.createElement('div');
    messageDiv.className = `chat-message ${type === 'user' ? 'user-message' : 'reply-message'}`;
    messageDiv.innerHTML = message;

    // Add "Speak Up" button with emoji for reply message
    if (type === 'reply') {
        const speakButton = document.createElement('button');
        speakButton.className = 'speak-button';
        speakButton.innerHTML = '🗣️'; // Speak Emoji

        // Modify onclick to prevent page reload and keep the chat intact
        speakButton.onclick = (event) => {
            event.preventDefault(); // Prevent the default button action (e.g., page reload)
            speakReply(message); // Call the function to speak the reply
        };
    }
};

```

```

    messageDiv.appendChild(speakButton);
  }

  chatContainer.appendChild(messageDiv);
  chatContainer.scrollTop = chatContainer.scrollHeight; // Scroll to the bottom
}

// Function to trigger Text-to-Speech API
async function speakReply(message) {
  try {
    const response = await fetch('http://127.0.0.1:8080/text-to-speech/', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ text: message, language: language })
    });

    if (response.ok) {
      console.log('Audio is playing locally.');
    } else {
      console.error('Text-to-Speech API error:', response.statusText);
    }
  } catch (error) {
    console.error("Text-to-Speech Error:", error);
  }
}

// Append server information in a styled box
function appendServerInfoBox(server) {
  const serverBox = document.createElement('div');
  serverBox.className = 'server-info-box';

  serverBox.innerHTML = `
    <h5>${server.name} - ${server.special}</h5>
    <p>Phone: ${server.phone}</p>
    <p><a
href="https://www.google.com/maps?q=${hospital.coords[0]},${server.coords[1]}"
target="_blank">View on Google Maps</a></p>
    <button class="btn btn-ambulance btn-block"
onclick="call(${server.phone})">Call Ambulance</button>
  `; chatContainer.appendChild(serverBox);

  // Add call button
  appendCallButton(server.phone);
}

function call(phone) {
  window.location.href = `tel:${phone}`;
}

```

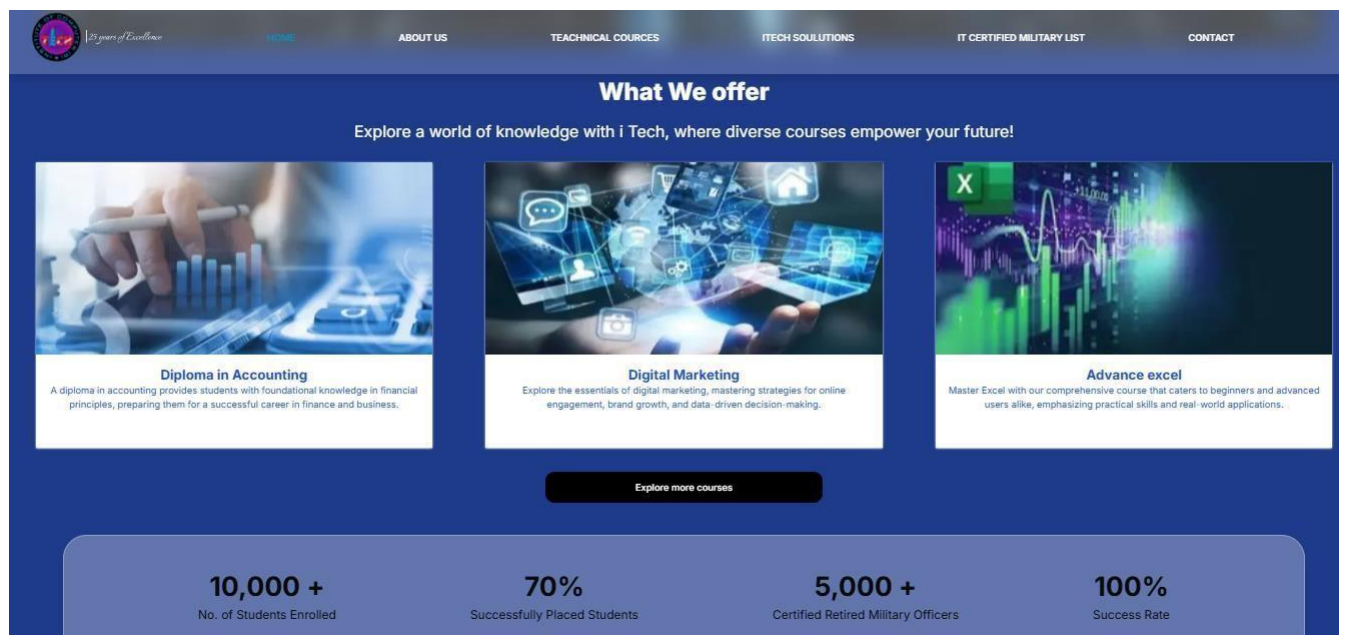
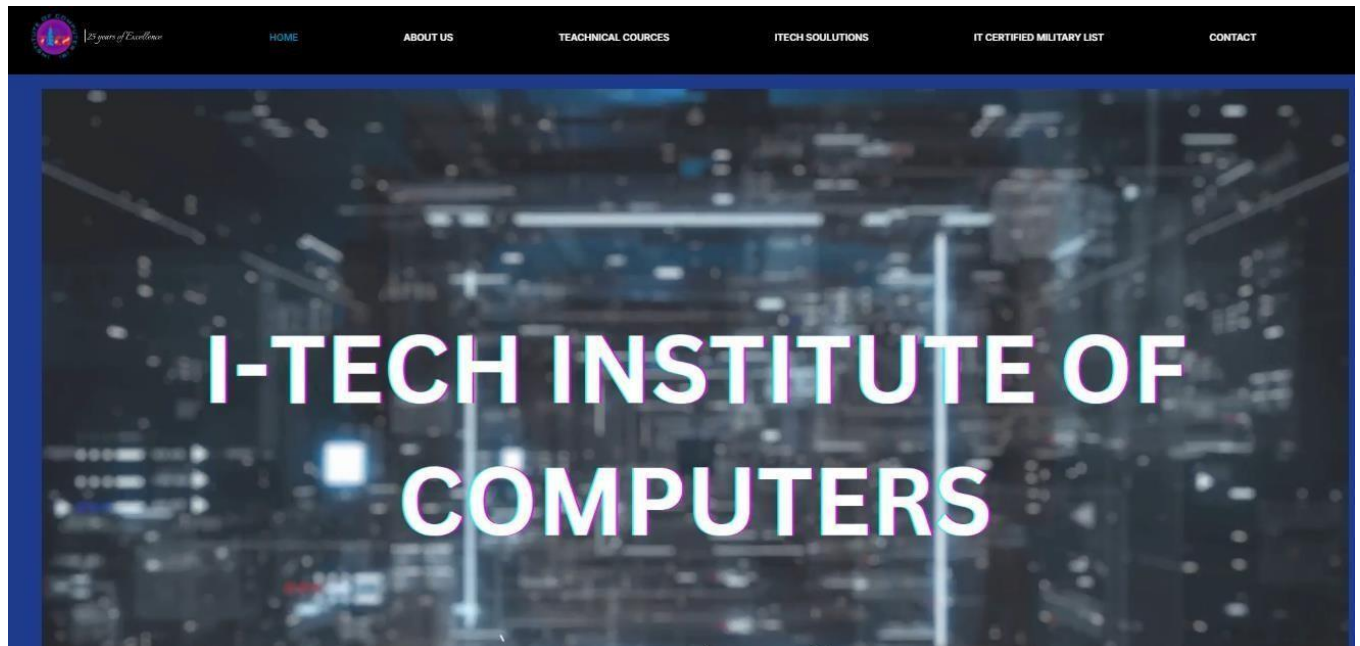
```
};

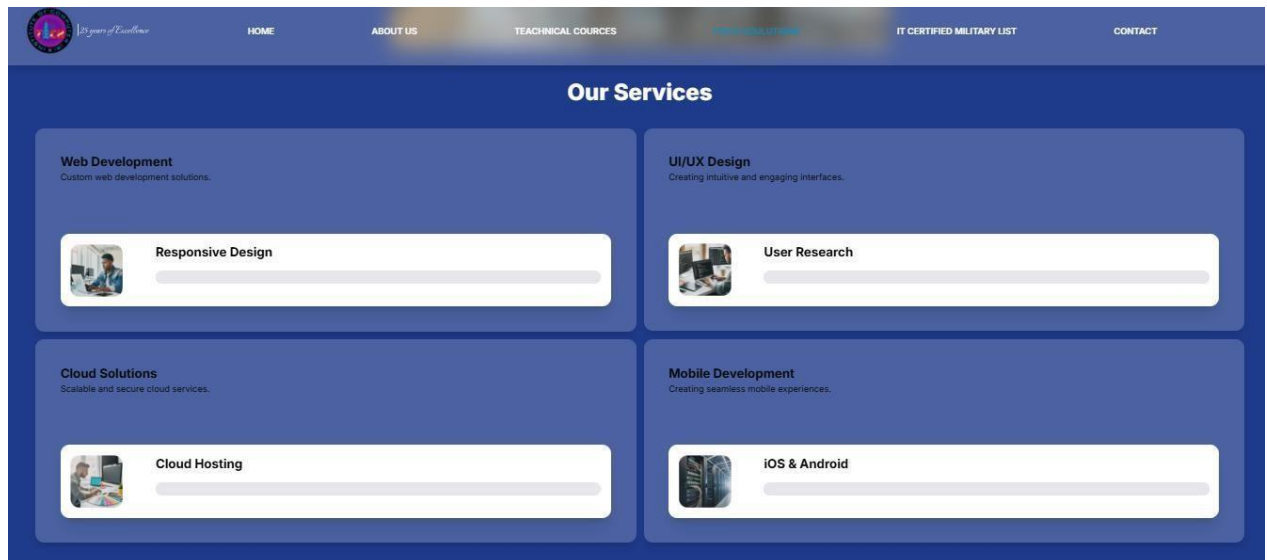
// Append call button
function appendCallButton(phone) {
  const callButton = document.createElement('button');
  callButton.className = 'call-button';
  callButton.innerText = 'Call Now';
  callButton.onclick = () => window.location.href = `tel:${phone}`;
  chatContainer.appendChild(callButton);
}
</script>

</body>
</html>
```


APPENDIX-B

SCREENSHOTS











19% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.




Filtered from the Report

- Bibliography

Match Groups

-  **197** Not Cited or Quoted 22%
Matches with neither in-text citation nor quotation marks
-  **0** Missing Quotations 0%
Matches that are still very similar to source material
-  **14** Missing Citation 2%
Matches that have quotation marks, but no in-text citation
-  **0** Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

- 13%  Internet sources
- 10%  Publications
- 10%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

APPENDIX-C

ENCLOSURES



The INTERNSHIP work carried out here is mapped to SDG-9 Industry Innovation And Infrastructure

The INTERNSHIP work carried here contributes to the well-being of human-society. This can be used for maintaining security in industrial and personal internet usage. This INTERNSHIP highlights the transformative potential of machine learning in addressing the ever-evolving landscape of cybersecurity threats. By focusing on the detection of malware through the analysis of URLs and Portable Executable (PE) files.