

Cura: A Serverless Cloud-Based Medical Consultation System with Role-Based Access Control and Machine Learning-Driven Disease Prediction

Ronald Aaron¹, Sam Manoj², Sharon Sam³, Ms. Varsha Nagpurkar⁴

¹²³Department of Computer Engineering, St. Francis Institute of Technology, Mumbai, India, Email:

⁴Project Guide, Department of Computer Engineering, St. Francis Institute of Technology, Mumbai, India, Email:

Abstract - Primary health care in resource-constrained environments often faces challenges in terms of manual records and absence of specialists. This delays patient diagnosis by weeks, putting lives at risk. In this paper, we present Cura, a secure, serverless cloud-based consultation system, which digitizes the clinical processes and connects on-site staff to remote doctors. The architecture of Cura uses React.js front-end on Amazon S3 along with an AWS serverless back-end, coordinated by AWS Amplify SDK. The Role-Based Access Control (RBAC) is done using Amazon Cognito User Pools with the help of custom attributes to differentiate administrators, nurses, doctors, and patients. The clinical vitals (blood pressure, SpO2, pulse) collected by on-site nurses using the system are saved in Amazon DynamoDB by AWS Lambda. The remote doctors use the logs, issue signed digital prescriptions, and initiate disease prediction based on Amazon SageMaker scikit-learn Random Forest classifier. Tests reveal high performance, less than 15 ms database queries, and usability.

Key Words: Serverless Computing, Telemedicine, Electronic Health Records (EHR), NoSQL Database, Amazon Web Services (AWS), Machine Learning, Random Forest, Amazon SageMaker.

1. INTRODUCTION

Telemedicine platforms increase healthcare accessibility, efficiency, and effectiveness [5], [8] by enabling remote consultations despite geographical boundaries [7].

Nevertheless, most rural healthcare centers still use manual record systems, creating three major problems. First, the lack of real-time access to vital signs and fragmented patient data causes delayed diagnosis [2]. Second, without encryption or role-based access controls, records face significant security risks [6], [9]. Third, the expensive infrastructure needed for dedicated IoT servers makes these systems unaffordable for resource-constrained clinics [3], [10]. Cura addresses all three challenges..

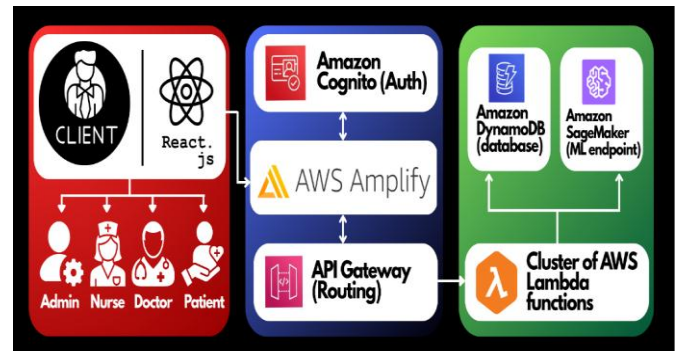


Fig -1: Cura platform's AWS Serverless Cloud Architecture overall view, illustrating S3 static website hosting, API Gateway routing, Lambda business logic, Cognito authentication, DynamoDB NoSQL data storage, and SageMaker machine learning endpoints.

This issue can be addressed using Cura, a hardware agnostic serverless platform for medical consultations. Implemented using React.js (Amazon S3 and CloudFront CDN) and an AWS serverless backend (API Gateway and Lambda), Cura allows for high availability and scalability. Access to the service is done through user authentication by means of Amazon Cognito User Pools. Role-Based Access Control is used to grant permissions to users in one of the four roles – Administrators (user management and configuration of the UI), Nurses (vitals input), Doctors (reviewing the records, disease predictions by SageMaker ML, and issuing digital prescriptions) and Patients (reviewing historical vitals and active prescriptions).

The service makes use of an intelligent diagnostic component in the form of a scikit-learn Random Forest classifier running on Amazon SageMaker to make disease predictions based on symptoms in real time. Without the need for any hardware or virtual machines, Cura becomes a cost-effective solution that scales to zero.

Key Contributions:

- A serverless clinical workflow system that scales itself without maintaining any servers.
- Granular RBAC via Cognito custom attributes (custom:role) securing sensitive medical data.
- A completely decoupled architecture including React.js, Amplify, API Gateway, Lambda, and DynamoDB.

- Random Forest Real-Time Integration on SageMaker for Diagnosis During Consultation.
- Usability testing and benchmarking for latency, availability, and effectiveness.

2. LITERATURE SURVEY

Predictive health analysis is achieved with cloud computing and machine learning technologies. Sundas et al. [1] introduced SPMR with deep learning techniques for chronic conditions classification. Ramadhan et al. [2] discussed the methods of ML preprocessing for disease prediction. Mai et al. [4] created an adaptive ML approach to regulate remote blood pressure. These solutions perform well offline but require no interactive consulting and prescriptions.

For remote clinics, Gupta et al. [3] introduced SAHARA with cloud telemetry features. However, SAHARA requires special biosensors which raises its cost. Cura gets rid of hardware with responsive web-form data acquisition.

Privacy in the healthcare sector is essential. The Privacy-by-Design reference architecture was developed by Tertulino et al. [6], whereas Mishra et al. [9] developed GDSPS following the HIPAA/GDPR regulations. Nankya et al. [8] provided a review on the paradigms of e-health security. SHACS was developed by Sangeetha et al. [11], which includes both RBAC and ABAC along with anomaly detection mechanisms. Fugkeaw et al. [10] implemented blockchain with CP-ABE in fog EHR systems but with significant computation costs. Zhang et al. [7] proved that structured consultation.

Table -1: Comparative Analysis of Proposed Cura System with Existing Literature

Reference	Focus Area	Core Methodology	Key Advantages	Research Gap / Limitations
Sundas [1]	Cloud Analytics	Deep Learning & Cloud	Highly accurate predictions of chronic diseases	Lack of consultation and prescription management.
Gupta [3]	Remote Monitoring	Cloud Telemetry & Sensors	Physiological status monitoring	Expensive hardware requirements; not suitable for resource-poor hospitals.
Tertulino [6]	EHR Architecture	Privacy by Design SRA	Systematic security segregation	Abstract reference model; lacks practical implementation.
Fugkeaw [10]	Fine-Grained Access	CP-ABE & Blockchain	Fine-grained encryption & Revocation	Computationally heavy on resource-constrained systems.
Sangeetha [11]	Cloud Security	RBAC-ABAC & ML Anomaly	Low authentication delay	Administering dynamic rules raises complexity.
Zhang [7]	OHC Interfaces	Wizard-of-Oz User Study	Power imbalance in web interface reduced	Is focused solely on front-end conversations.
Cura (Proposed)	Serverless Consultation & ML Diagnosis	React.js, AWS Amplify, API Gateway, Lambda, DynamoDB, SageMaker	Fully serverless (zero maintenance), scale-to-zero cost, integrated ML-driven disease prediction	Lacks automated sensor integration and real-time video consults.

3. PROPOSED SYSTEM

Cura consolidates the information in Amazon DynamoDB using AWS Lambda and Amazon API Gateway.

3.1 Design Objectives

Some of the key design objectives that will be achieved include: (1) Real-Time Ingestion using the AWS Amplify SDK to log vitals; (2) Granular RBAC using Amazon Cognito User Pools; (3) NoSQL Persistence for flexible clinical records; (4) Decision support through SageMaker real-time ML endpoints; and (5) Cost efficiency by means of a scale-to-zero serverless deployment approach.

3.2 User Roles and Functional Partitioning

The features are segregated through custom attributes defined in Cognito (custom:role): **Administrator** (manages user accounts and UI layout settings without accessing patient information), **Nurse** (consumes BP, SpO₂, and pulse on examination), **Doctor** (analyzes patient medical record, makes predictions for ML diseases (and prescriptions issued), and **Patient** (personal history and current medications accessed).

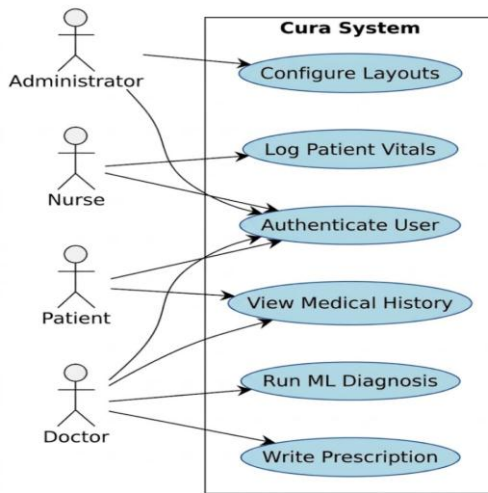


Fig -2: Use case diagram showing interaction of actors with system through Administrator, Nurse, Doctor, and Patient classes in Cura System.

3.3 NoSQL Data Structure

DynamoDB stores two tables - **Cura_Medical_Records** (Partition Key: patientId, Sort Key: entryTimestamp) that holds the data related to vitals, doctor's notes, and prescriptions; and **Cura_Config** containing information about the UI layout (NAV_CONFIG, PANEL_LABELS) loaded at client startup.

4. METHODOLOGY

All clinical transactions involve validation and role-based verification by Cura.

4.1 Operational Workflow

- **Authentication:** User authenticates via React; Amplify retrieves a JWT from Cognito.
- **Role Verification:** The frontend takes custom:role out of the JWT for role-specific dashboards.
- **Vitals Ingestion:** Nurse inputs vitals; API Gateway sends the payload to Write Lambda, which stores the information in DynamoDB.
- **Clinical Consultation:** Doctor gets medical history from Read Lambda and may use symptoms to query SageMaker.
- **Prescription Issuance:** Doctor makes entry in patient's notes; signed notes are put in DynamoDB database.

- **Patient Access:** Authentic patient access to their records.

Algorithm 1: Serverless Ingestion and Role-based Filtering

```

1: Input: Credentials C = {email, pass}, Pid, vitals V = {bp, spo2, pulse}
2: Output: Redirect dashboard or commit to DynamoDB.
3: JWT ← CognitoAuth(C)
4: if JWT is invalid then return AccessDenied end if
5: Role ← ExtractCustomRole(JWT); Redirect to Role dashboard
6: if Role = Nurse then
7:   Validate: 40 ≤ bpsys ≤ 220, 30 ≤ bpdia ≤ 130, 40 ≤ pulse ≤ 200, 50 ≤ spo2 ≤ 100
8:   if valid then
9:     Item ← {Pid, bp, spo2, pulse, timestamp}
10:    DynamoDB_PutItem(Cura_Medical_Records, Item); return Success
11:   else return ValidationError: Out of bound metrics
12:   end if
13: else return PermissionError: Nurse role required
14: end if

```

Algorithm 2: Consultation, SageMaker ML Prediction, and Prescription

```

1: Input: Doctor JWTdoc, Pid, symptoms S, notes L, prescription Rx = {DrugName, Dosage, Frequency, Duration}
2: Output: Prediction, Prescription, and Updating of Database.
3: Role ← ExtractCustomRole(JWTdoc)
4: if Role ≠ Doctor then return PermissionError end if
5: History ← DynamoDB_Query(patientId = Pid); Display on dashboard
6: if S ≠ ∅ then
7:   Prediction ← SageMaker.InvokeEndpoint(S); Display Prediction
8: end if
9: if Rx = ∅ or L = ∅ then return ValidationError end if
10: Sig ← Hash(JWTdoc + timestamp)
11: Item ← {Pid, L, Rx, Prediction, Sig, timestamp}
12: DynamoDB_PutItem(Cura_Medical_Records, Item); return Success

```

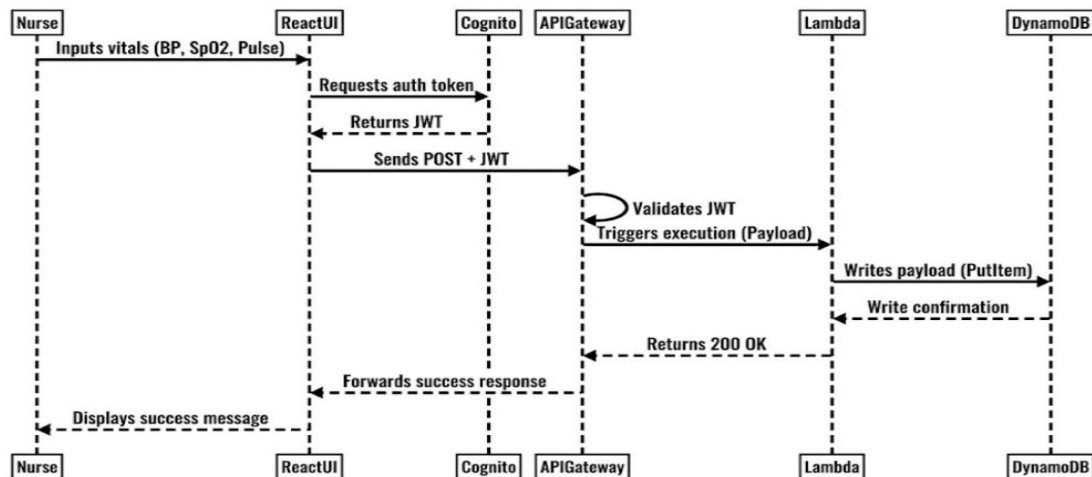


Fig -3: Sequence diagram depicting the timelines of messages passed among Nurse, React Frontend, AWS Amplify SDK, Cognito User Pools, API Gateway, Lambda Write Function, and DynamoDB while ingesting vitals.

4.2 Algorithmic Formalization

Algorithm 1 gives details about nurse vitals validation, and Algorithm 2 provides a formal structure for doctor consultation, SageMaker ML inference, and prescription authorization.

5. SYSTEM ARCHITECTURE

The Cura design uses a three-tier serverless microservice architecture that is highly available, scalable, and requires minimal management effort.

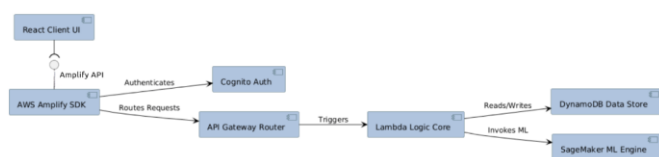


Fig -4: Component Diagram depicting modular coupling between React Client UI, AWS Amplify SDK, Cognito Auth, API Gateway, Lambda Logic Core, DynamoDB Data Store, and SageMaker ML Engine.

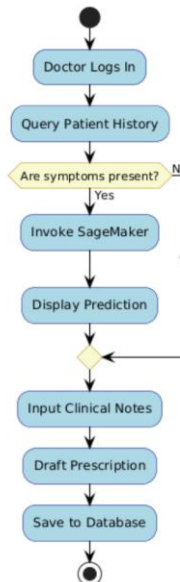


Fig -5: Activity Diagram showcasing the paths of decision-making, fork, and join for remote patient consultation, machine learning prediction, and prescription authorization.

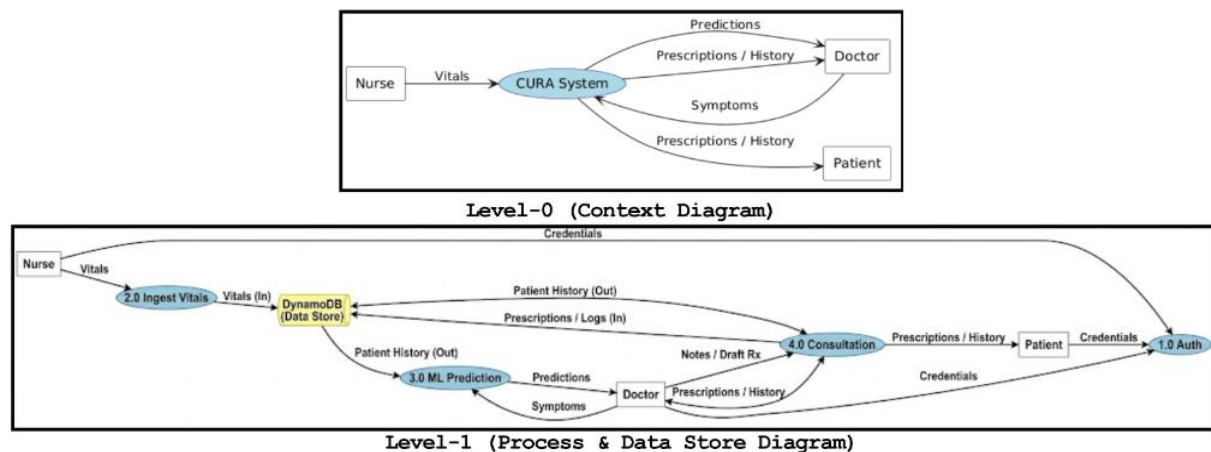


Fig -6: Data Flow Diagram showing data flow at Level-0 (Context Diagram) and Level-1 (Process & Data Store Diagram) starting from Nurse Intake till Prescription Delivery using SageMaker Predictions.

5.1 Architectural Layering

1. User Interface Layer: React.js SPA running on Amazon S3 with CloudFront CDN; session management and routing are handled via Amplify SDK.
2. Business Logic Layer: Amazon API Gateway validates Cognito JWT and directs requests to isolated AWS Lambda functions.
3. Database and ML Layer: Amazon DynamoDB for NoSQL data store, Amazon Cognito for user directories and Amazon SageMaker for Random Forest models.

5.2 Architectural Modeling

The structure and behavior of the system are captured using unified diagrams: System Architecture (Fig. 1), Use Case Diagram (Fig. 2), Sequence Diagram (Fig. 3), Component Diagram (Fig. 4), Activity Diagram (Fig. 5), Data Flow Diagram (Fig. 6), Deployment Architecture (Fig. 7), and Patient Care Workflow (Fig. 8).

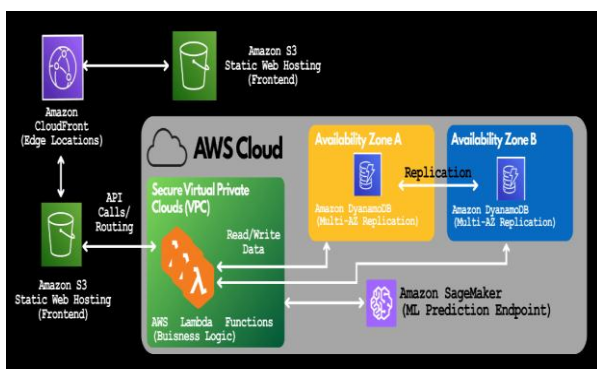


Fig -7: Deployment Architecture consisting of CloudFront Edge Caching, S3 Frontend Hosting, Secure VPC Lambda Runtimes, DynamoDB Replication in Multi-AZ across Availability Zones A & B, and SageMaker Inference Endpoints.

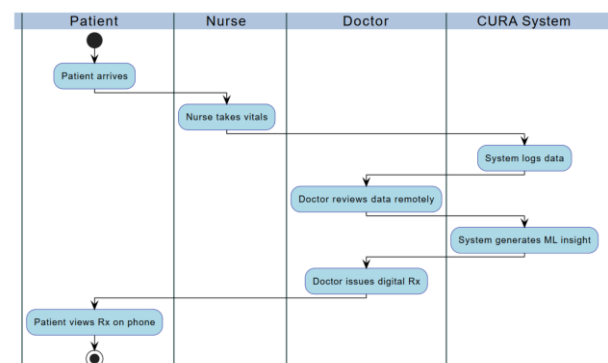


Fig -8: Patient Care Workflow Diagram outlining logical transitions from patient arrival and nurse vitals logging to remote doctor examination, SageMaker ML insight generation, and prescription retrieval.

6. IMPLEMENTATION

6.1 Technology Stack

The technology stack chosen is described in Table 2

Table -2: Cura System Serverless Technology Stack

Component Layer	Technology / Service Selection
Client Frontend	React.js, HTML5, CSS3, JavaScript (ES6)
Client SDK	AWS Amplify Library (Auth & API Modules)
API Router	Amazon API Gateway REST API
Compute Layer	AWS Lambda (Node.js / Python Functions)
Database Layer	Amazon DynamoDB (NoSQL Key-Value Store)
Authentication	Amazon Cognito User Pools
Model Hosting	Amazon SageMaker AI Real-time

Component Layer	Technology / Service Selection
	Endpoints
ML Classifier	scikit-learn Random Forest (cura_model.py)
Static Web Hosting	Amazon Simple Storage Service (S3) & CloudFront
Development Tools	VS Code, Git, GitHub, Python, Jupyter Notebook

6.2 Frontend Client and Amplify Integration

Route guarding in the React.js SPA is accomplished using react-router-dom. Amplify Auth fetches the user's role from the JWT token (custom:role), while Amplify API sends HTTPS requests with bearer tokens. The following modules are among those in use: Vitals Ingestion Form, Consultation Panel, Prescription Generator, and Patient Portal.

6.3 Serverless Business Logic Layer (AWS Lambda)

Stateless Lambda functions perform backend processing: Write Lambda checks for valid input ranges and performs DynamoDB PutItem operations, meanwhile, Read Lambda authenticates the user and executes DynamoDB query through Query on patientId.

6.4 Machine Learning Integration (Amazon SageMaker)

Random Forest Classifier trained on the symptom-disease data set is used as a SageMaker endpoint for prediction purposes. A Lambda function takes the symptom entered by the doctor and invokes the SageMaker endpoint using the AWS SDK.

6.5 Security Infrastructure

Security measures will be taken at all levels, which include: (1) Amazon Cognito User Pools, (2) Custom role JWT claims for protecting client routes, (3) API Gateway JWT authorizers, and (4) IAM roles.

7. EXPERIMENTAL RESULTS

7.1 System Specifications

Deployments were verified for environments as described in Table 3.

Table -3: Cura Platform System Resource Specifications

Requirement Type	Specification / Environment
Client CPU	Intel Core i3 or equivalent (2.0 GHz or

Requirement Type	Specification / Environment
	above)
Client RAM	Minimum 4 GB (Recommended 8 GB)
Client OS	Windows 10/11, Linux (Ubuntu/Debian), macOS
Client Browser	Google Chrome, Mozilla Firefox, Microsoft Edge
Cloud Serverless Host	Amazon API Gateway & AWS Lambda
Database Engine	Amazon DynamoDB (Multi-AZ NoSQL Store)
Inference Engine	Amazon SageMaker AI Real-time Endpoint
Connectivity	Standard Internet Connection (Broadband/4G/5G)

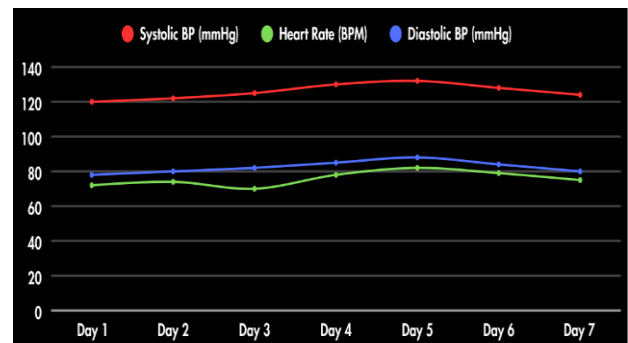


Fig -9: Vital Signs Monitoring for Patient (7-day Trend) – Systolic Blood Pressure (120–132 mm Hg), Diastolic Blood Pressure (78–88 mm Hg), and Heart Rate (70–82 BPM) tracked across a 7-day observation window.

7.2 Performance and Latency Audits

The following performance benchmarks were established through the evaluation: DynamoDB Multi-AZ Replication has 99.99% availability; queries against the primary key, fetch patient's data within <15 ms; Lambda cold start takes 150–220 ms (as opposed to 12–18 ms taken by warm Lambda function); and Cognito token authentication prevents unauthorized API requests.

7.3 Physiological Vitals Trend Monitoring

Longitudinal vitals for 7 days are displayed in Fig. 9. The systolic blood pressure values vary between 120–132 mmHg, while the diastolic blood pressure varied between 78–88 mmHg, and Heart rate was 70–82 BPM. Correlation of multiple metrics helps doctors quickly recognize patient trends.

8. DISCUSSION

8.1 Serverless, Scale-to-Zero Architecture

Different from IoT solutions that require specific hardware to be purchased [3], [10], Cura is entirely software-based. Pay per request billing models prevent expenses on idle resources, whereas managed services reduce server management overhead.

8.2 Flexible Schemas in NoSQL Environments

The flexible schema of DynamoDB does not require any migrations to accommodate new attributes for clinical information, whereas the user interface metadata (NAV_CONFIG) is stored directly with the record.

8.3 Intelligent Decision Support via SageMaker

Unlike offline models [1], [2], the SageMaker endpoints dynamically adapt to symptoms, helping the remote physicians during consultation sessions.

9. LIMITATIONS & FUTURE WORK

Limitations: Needs continuous connectivity on internet; manual entries prone to errors by humans; no video/audio calling facility; no synchronization of data in offline mode.

Future Work: Video consultation using WebRTC; React Native mobile application development; Bluetooth-based IoT sensors; Clinical alerts from AWS SNS; and deep learning-based models [2] for multi-classification of diseases.

10. CONCLUSION

Cura was introduced in this paper, which is a cloud-based platform for medical consultations and diagnosis using serverless technology. With the transition from the old system of maintaining paper-based files to an integrated dynamo DB, Cura offers real-time tracking of vital signs and even virtual consultations. The technology used to build Cura includes React.js, AWS Amplify, API Gateway, Lambda, Cognito, and SageMaker.

REFERENCES

- 1) A. Sundas, S. Badotra, G. S. Shahi, A. Verma, S. Bharany, A. W. Abulfaraj, and F. Binzagr, "Smart Patient Monitoring and Recommendation (SPMR) Using Cloud Analytics and Deep Learning," IEEE Access, vol. 12, pp. 51834–51848, 2024.
- 2) N. G. Ramadhan, Adiwijaya, W. Maharani, and A. A. Gozali, "Chronic Diseases Prediction Using Machine Learning With Data Preprocessing Handling: A Critical Review," IEEE Access, vol. 12, pp. 84261–84279, 2024.
- 3) A. K. Gupta, J. Nagaraja, C. R. Padma, M. Kumar N, and R. M. S, "SAHARA: A Cloud-based Remote Health-Vitals Monitoring System," in Proc. 2024 2nd Int. Conf. Networking, Embedded and Wireless Systems (ICNEWS), Bengaluru, India, 2024.
- 4) V. T. Mai, K. A. Alattas, Y. Bouteraa, E. Ghaderpour, and A. Mohammadzadeh, "Personalized Blood Pressure Control by Machine Learning for Remote Patient Monitoring," IEEE Access, vol. 12, pp. 88350–88365, 2024.
- 5) E. Gaeta, M. S. Haleem, L. Lopez-Perez, M. Manea, M. F. C. Umpierrez, L. Pecchia, and G. Fico, "GATEKEEPER Platform: Secure Processing Environment for European Health Data Space," IEEE Access, vol. 13, pp. 34627–34638, 2025.
- 6) R. Tertulino, N. Ivaki, and H. Morais, "Design a Software Reference Architecture to Enhance Privacy and Security in Electronic Health Records," IEEE Access, vol. 12, pp. 112157–112179, 2024.
- 7) L. Zhang, J. Zhan, V. K. W. Wan, and Y. Wang, "Designing and Evaluating Online Health Consultation Interfaces: A Perspective of Physician-Patient Power Asymmetry," IEEE Access, vol. 12, pp. 124111–124127, 2024.
- 8) M. Nankya, A. Mugisa, Y. Usman, A. Upadhyay, and R. Chataut, "Security and Privacy in E-Health Systems: A Review of AI and Machine Learning Techniques," IEEE Access, vol. 12, pp. 148796–148816, 2024.
- 9) V. Mishra, K. Gupta, D. Saxena, and A. K. Singh, "A Global Medical Data Security and Privacy Preserving Standards Identification Framework for Electronic Healthcare Consumers," IEEE Trans. Consum. Electron., vol. 70, no. 1, pp. 1–9, Feb. 2024.
- 10) S. Fugkeaw, R. P. Gupta, and K. Worapaluk, "Secure and Fine-Grained Access Control With Optimized Revocation for Outsourced IoT EHRs With Adaptive Load-Sharing in Fog-Assisted Cloud Environment," IEEE Access, vol. 12, pp. 82753–82768, 2024.
- 11) S. K. B. Sangeetha, C. Selvarathi, S. K. Mathivanan, J. Cho, and S. V. Easwaramoorthy, "Secure

Healthcare Access Control System (SHACS) for Anomaly Detection and Enhanced Security in Cloud-Based Healthcare Applications," IEEE Access, vol. 12, pp. 164543–164559, 2024.

- 12) S. Kamble, K. P. Karunakaran, and J. Shah, "Machine Minds: A Psychotherapeutic Evaluation of LLMs," Journal of Technology, vol. 13, no. 7, pp. 583–591, 2025.
- 13) S. B. Soumma, F. Shahriar, U. N. Mahi, M. H. Abrar, M. A. R. Fahad, and A. S. L. Hoque, "Design and Implementation of a Scalable Clinical Data Warehouse for Resource-Constrained Healthcare Systems," arXiv:2502.16674v2, May 2025.