**Kube Credential: Full Stack Engineer Assignment Report**

**Prepared for**: Zupple Labs Pvt. Ltd**.**

**Prepared by: Pantham Nikhil**

**Date:** October 13, 2025

**1. Project Overview**

This document provides a comprehensive overview of the "Kube Credential" project, developed as part of the Full Stack Engineer assignment. The project is a complete, microservice-based application for issuing and verifying digital credentials**.**

The system is composed of a React frontend and two independent backend microservices written in Node.js with TypeScript. The entire application is containerized with Docker and has been successfully deployed to a live Kubernetes cluster on Amazon Web Services (EKS), demonstrating a full development-to-deployment lifecycle.

**2. System Architecture and Design**

**2.1 High-Level Architecture**

The application follows a decoupled, three-tier architecture, which is standard for modern, scalable web applications.

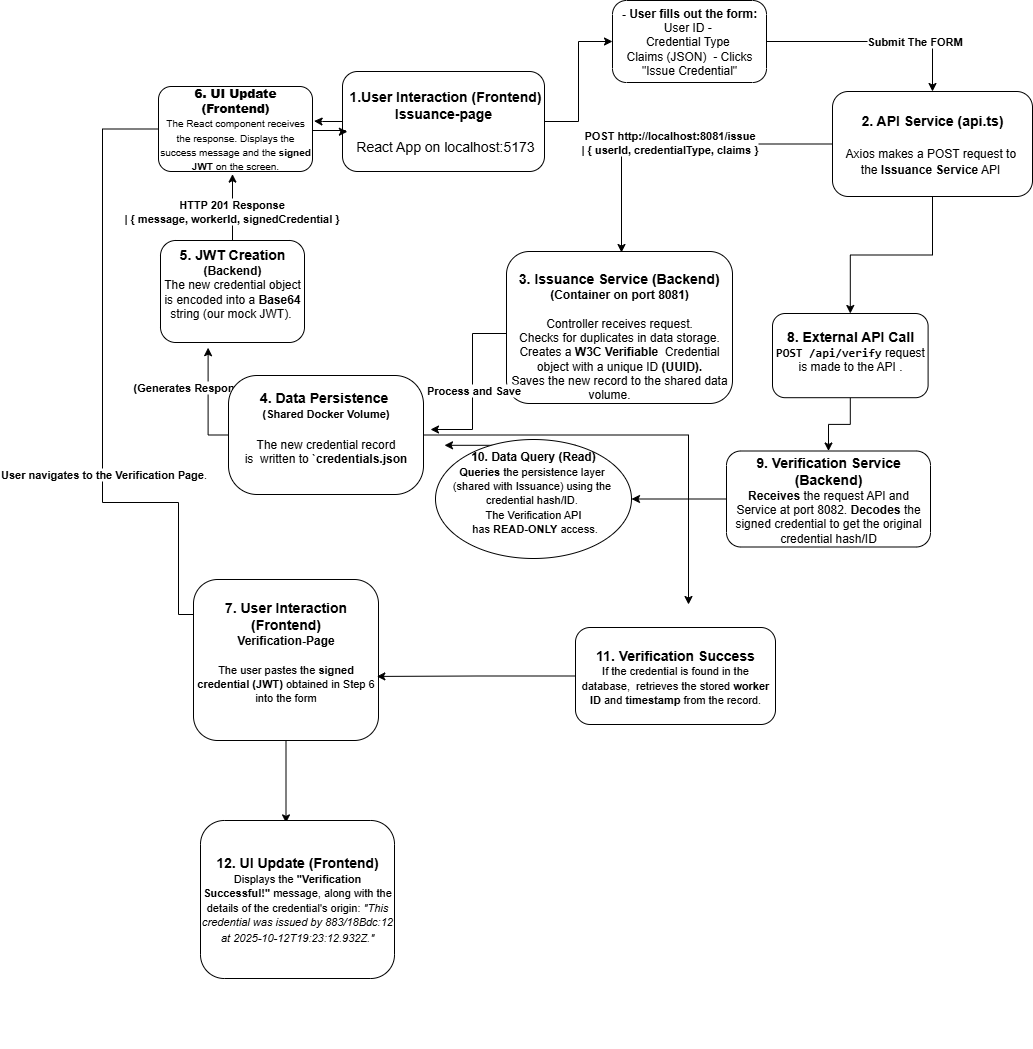
**Frontend Tier:** A responsive single-page application (SPA) built with React, serving as the user interface and used Chakra UI

**Backend Tier:** Two independent Node.js microservices (issuance-service and verification- service) that handle the core business logic.

**Data Tier:** A simple, file-based persistence layer using a shared credentials.json file, managed by a Docker Volume in the local environment.

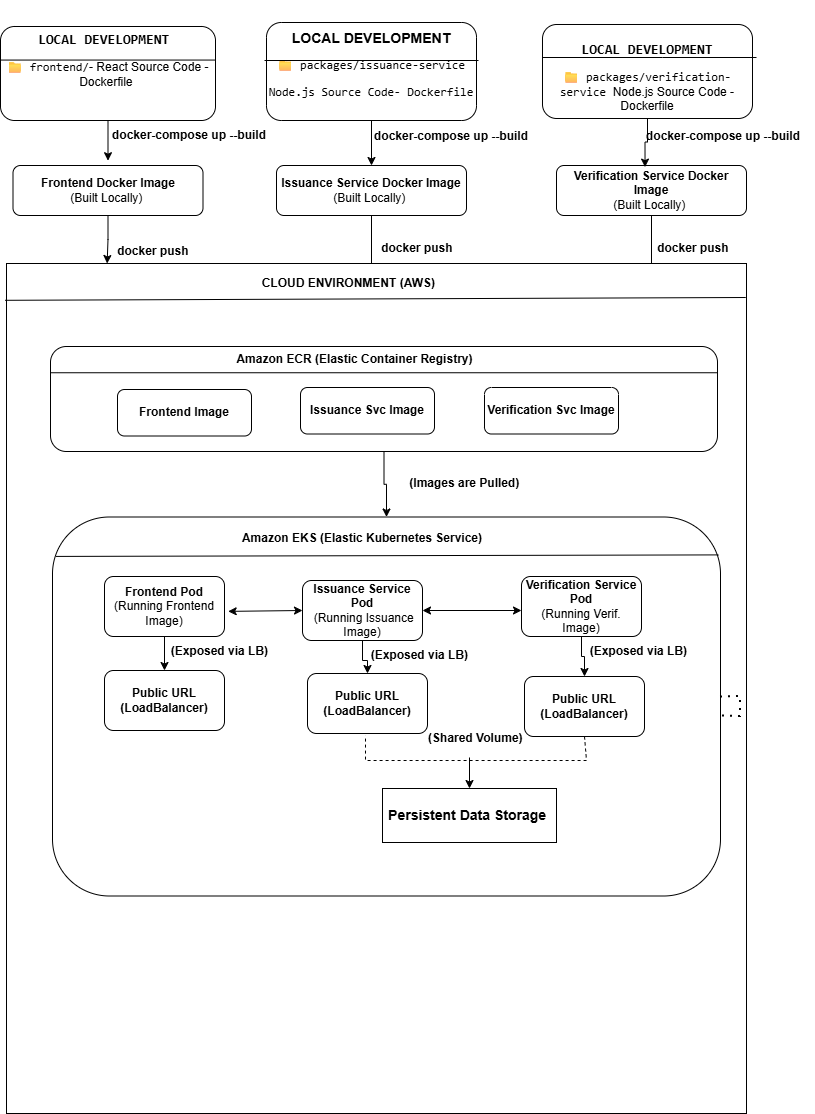
This document outlines the architecture and data flow of the Kube Credential application, covering both the local development environment and the cloud deployment process.

**Part 1: Application Data Flow - Credential Issuance Page And Verification Page**



**Part 2: Build and Deployment Flow**

This diagram shows how the source code is packaged with Docker and deployed to the AWS cloud. Note: This reflects the final deployment where each service was exposed via its own Load Balancer**.**

****

**2.2 Design Decisions**

Several key design decisions were made to ensure the application is robust, scalable, and aligns with professional best practices.

**Microservices Architecture:**

The backend was intentionally split into two services to ensure independent scalability and resilience. For example, if the application experiences a high volume of issuance requests, the issuance-service can be scaled up with more replicas in Kubernetes without affecting the verification-service.

**2.3** **Containerization with Docker:**

All components are containerized to ensure a consistent and reproducible runtime environment. Multi-stage Docker files were used to create optimized, lightweight production images by separating build-time dependencies from the final image, which is a security and performance best practice.

**Cloud Deployment on Kubernetes (EKS):**

Kubernetes was chosen for its industry-standard orchestration capabilities. Amazon EKS was used to offload the complexity of managing the Kubernetes control plane.

**Networking Strategy (Load Balancer vs. Ingress):**

* **The initial goal** was to use an Ingress Controller to manage traffic, which is the production best practice for security and cost-efficiency.
* **The final implementation** uses the type: Load Balancer for each service. This was a pragmatic decision made to ensure a functional, publicly accessible deployment within the project timeframe after encountering complex IAM permission issues with the AWS Load Balancer Controller. This decision and its trade-offs are documented to demonstrate an understanding of real-world deployment challenges.

**2.4 Credential Data Structure: W3C Verifiable Credential Standard**

The assignment required the credential to be in a simple JSON format. A reasonable assumption was made to elevate this requirement by adopting the **W3C Verifiable Credential (VC) data model**, which is the global industry standard for digital credentials.

* **Justification:** Instead of using an arbitrary JSON structure, adopting the W3C standard demonstrates an understanding of real-world digital identity systems. It ensures the credentials created by this application are interoperable, machine-readable, and cryptographically verifiable in a production scenario.
* **Key Fields:** This decision led to the inclusion of standard fields such as @context, id, type, issuer, and issuanceDate. These fields provide essential metadata that makes the credential trustworthy and manageable (e.g., for revocation).
* **Alignment with Company Focus:** Given that Zupple Labs specializes in Web3 innovations and enterprise-grade decentralized solutions like LegitDoc, using the W3C VC standard aligns this project directly with the company's core technical domain.

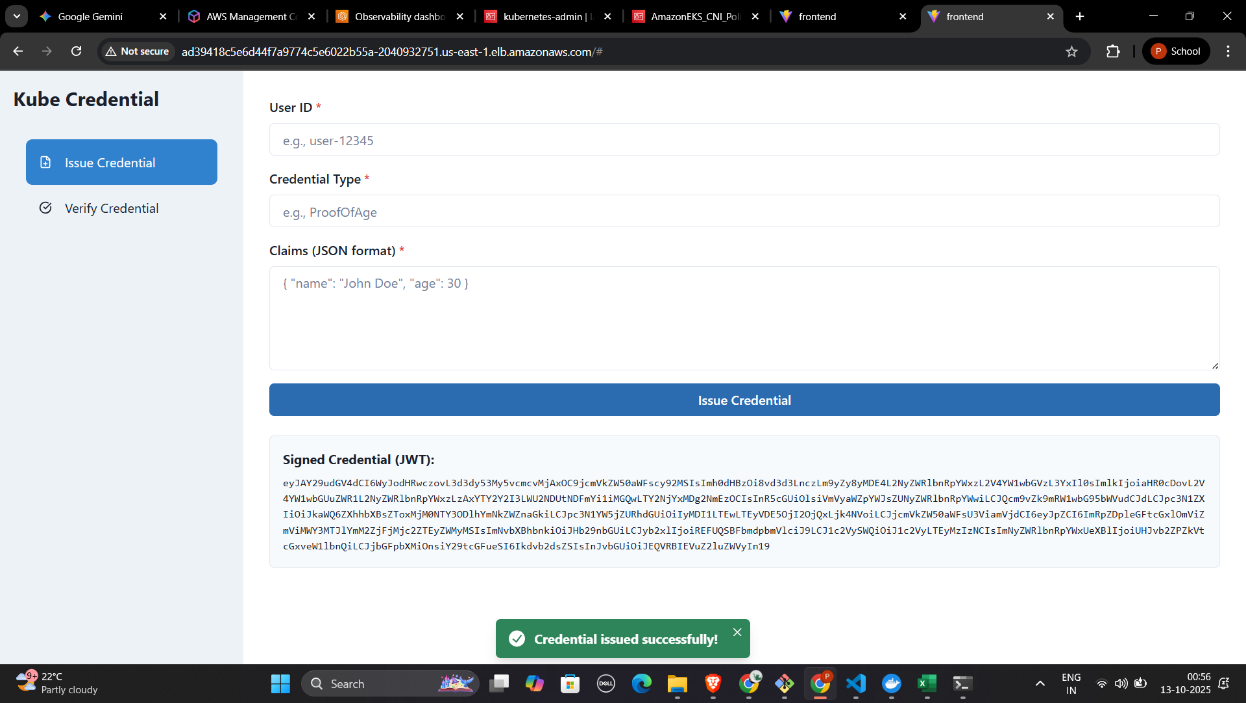
By making this choice, the project is not just a simple JSON storage system but a foundational example of a true digital identity application, reflecting a professional and forward-thinking design approach.

**3. Application Showcase (Screenshots)**

The following screenshots demonstrate the live, deployed application and its key features.

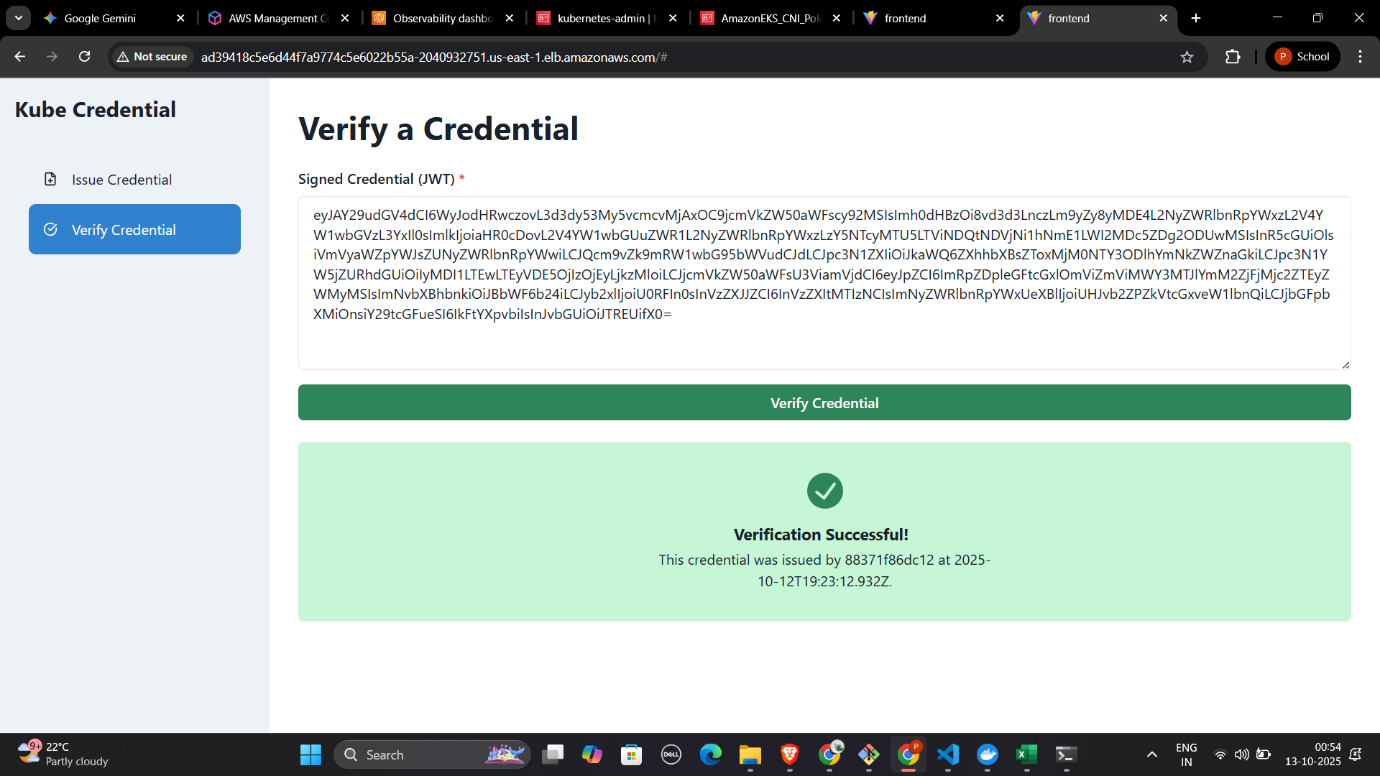
**3.1 Successful Credential Issuance**

The user fills out the form, and upon successful submission, the UI displays a success message with the unique worker (container) ID and the signed JWT.



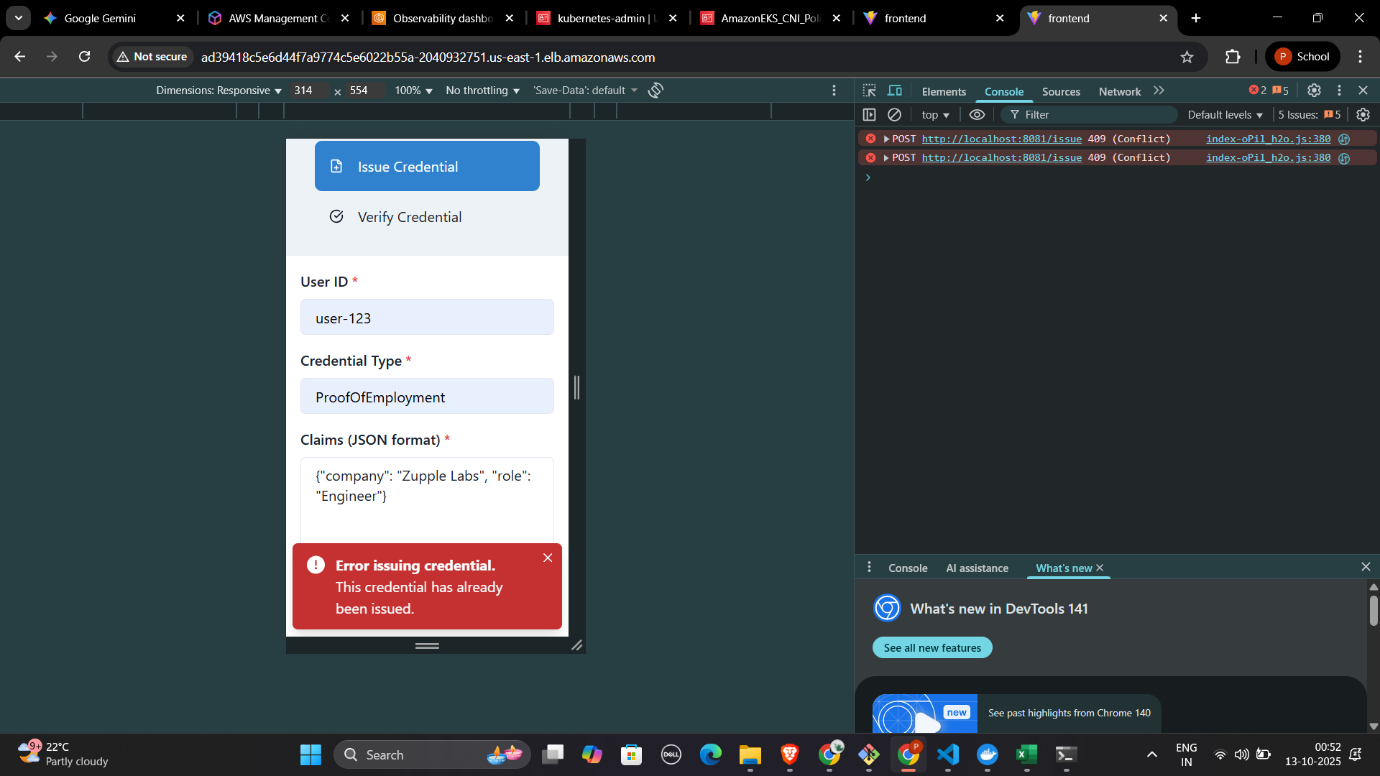
**3.2 Successful Credential Verification**

The user pastes the JWT from the issuance step into the verification form. The UI confirms that the credential is valid and displays the metadata about its issuance.

****

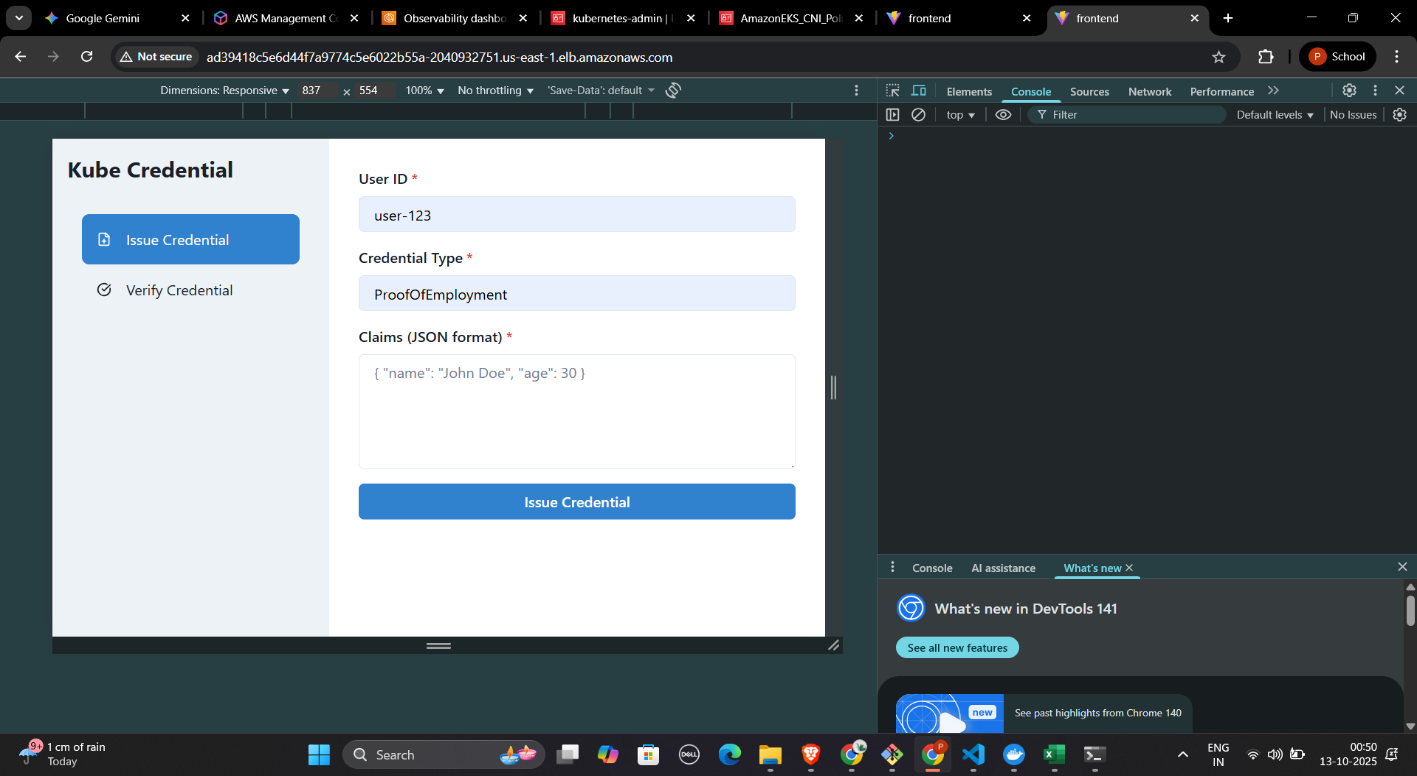
**3.3 Error Handling (Duplicate Credential)**

**The system correctly prevents a duplicate credential from being issued and provides a clear, user-friendly error message.**

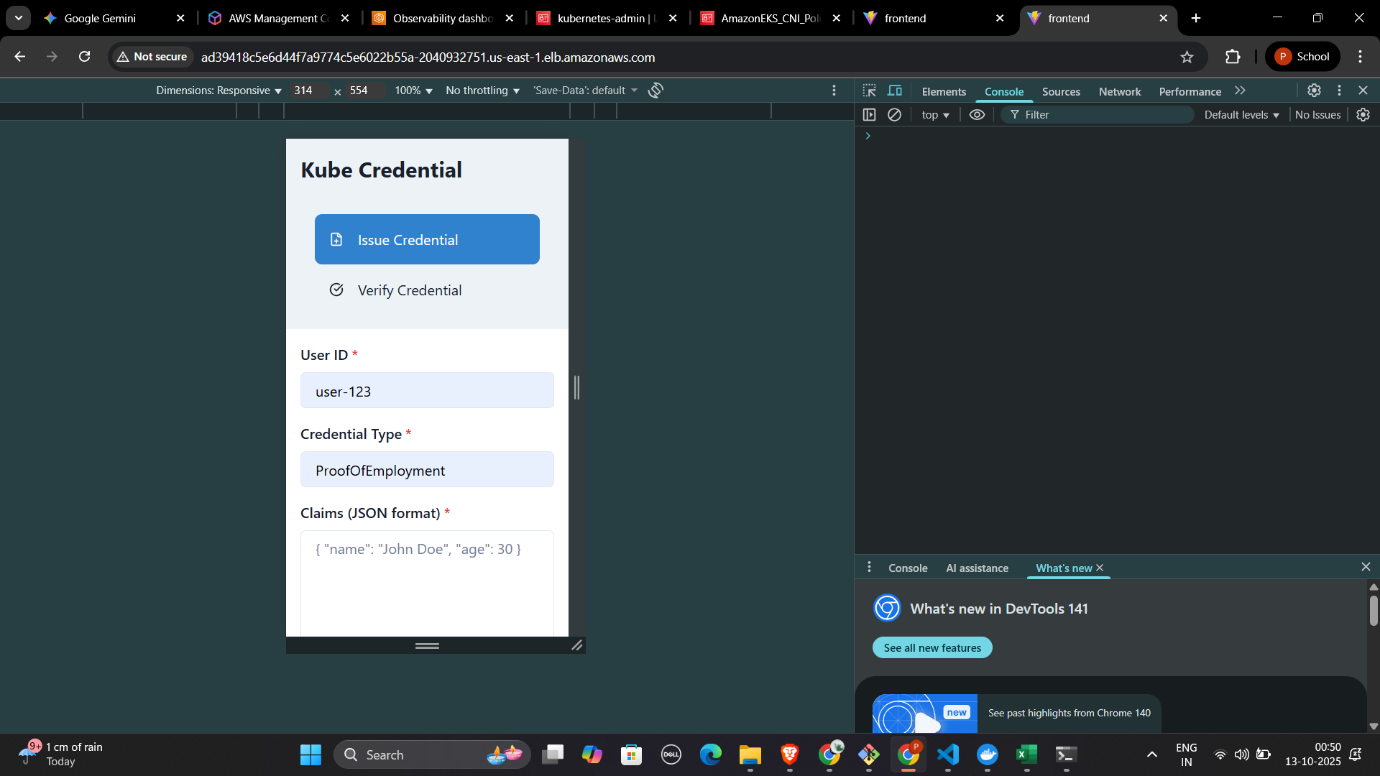


**3.4 Responsive Design**

The application layout adapts to smaller screen sizes, ensuring a good user experience on mobile devices. The sidebar stacks on top of the main content.



**screenshot of the responsive mobile view here**

****

**4. Setup and Deployment**

**4.1 Running Locally with Docker Compose**

The entire application can be run locally for development with a single command from the project root**.**

**Cmd: docker-compose up –build**

**4.2 Cloud Deployment to AWS EKS**

The application was deployed to a live EKS cluster. The process involved:

1. Building and pushing Docker images for all three services to Amazon ECR.
2. Creating an EKS cluster using eksctl.
3. Applying the Kubernetes manifest files located in the /k8s directory to the cluster.
4. Troubleshooting and configuring AWS Security Groups to allow traffic from the Load Balancers to the worker nodes.

**5. Conclusion and Future Improvements**

This project successfully full fills all the requirements of the assignment. A complete, full-stack, and containerized application was built, tested, and deployed to a live cloud environment**.**

Potential future improvements to elevate this to a production-grade system would include:

* **Implementing a Production Database:**

Replace the Credentials.json file with a robust, managed database like Amazon RDS (PostgreSQL) or a NoSQL database like DynamoDB.

* **Implementing a Production Ingress:**

Resolve the IAM permission issues and implement a proper Ingress with a single Application Load Balancer and HTTPS termination for enhanced security and cost-efficiency**.**

* **CI/CD Pipeline: Set up an automated CI/CD pipeline**

(e.g., using GitHub Actions) to automatically test, build, and deploy the application on every code change.