This abstract outlines the development approach, architecture, and key learnings from building SkillSession, an online tutoring platform that connects learners with qualified teachers. The platform was designed for scalability, maintainability, and rapid deployment, leveraging a modern serverless architecture and a decoupled frontend-backend model. Below is a breakdown of the technical

implementation and the main insights gained during the project.

Concept and Technical Architecture

To address the need for a scalable and cost-efficient learning platform, we adopted a serverless, decoupled architecture. This allowed us to independently scale components, reduce infrastructure costs, and enable agile development workflows.

Frontend (Vue.js)

Built using Vue.js 3 with the Composition API

UI components implemented using Element Plus with responsive design

Pinia used for state management

Vue Router configured for navigation and route protection

AWS Amplify integrated for secure user authentication

Backend (AWS Serverless Stack)

AWS Lambda used to handle backend logic through granular function handlers

Amazon API Gateway exposed RESTful endpoints

DynamoDB selected for flexible NoSQL data persistence

S3 used for profile image uploads and retrieval

SNS (Simple Notification Service) enabled email-based notifications

AWS Cognito handled authentication and authorization

Development Process

We approached development using a modular, sprint-based methodology, breaking down the large goal into smaller deliverables:

Backend Development First: All Lambda functions were implemented and tested directly on the AWS portal.

Frontend Development Next: UI components were built independently, without initial API integration.

Infrastructure as Code: CloudFormation templates were written to define and deploy backend infrastructure.

Frontend-Backend Integration: APIs were then integrated into the frontend layer.

Deployment: Frontend was deployed on EC2 using an Elastic Beanstalk stack.

Key Lessons Learned

Architecture

Serverless Strengths & Challenges: Serverless enabled faster iteration and simplified scaling, but demanded careful configuration around cold starts and timeout limits.

Function Granularity: Initially, overly fine-grained Lambda functions increased complexity. Consolidating functions by business domain improved maintainability and performance.

Data Layer

DynamoDB Design: The flexible schema was powerful, but required us to shift from traditional relational thinking. Designing based on access patterns was essential.

Integration

API Design & Error Handling: A clean, consistent API contract and robust frontend error handling were vital to ensuring a smooth user experience.

Infrastructure

CloudFormation Complexity: Writing comprehensive infrastructure templates up front was difficult to test, but paid off in consistency and reproducibility.

Deployment Strategy: Defining infrastructure and deployment separately for backend and frontend clarified responsibilities and reduced coupling.

Process Insights

Incremental Delivery: Building the core features first and expanding iteratively allowed us to validate the platform early and prioritize effectively.

Testing Strategy: Unit testing Lambda functions and conducting end-to-end API integration tests ensured reliability throughout fast-paced development.

Documentation Discipline: Maintaining detailed documentation for APIs, data models, and deployment processes significantly improved team coordination and long-term maintainability.