Development of SkillSession – An Online Tutoring Platform

This abstract outlines the development approach, architecture, and key lessons learned from building SkillSession, an online tutoring platform that connects learners with qualified teachers. The platform was designed with a focus on scalability, maintainability, and rapid deployment, leveraging a modern serverless architecture and a decoupled frontend-backend model. Below is a breakdown of the solution's concept, technical structure, development methodology, and lessons learned.

Concept and Technical Architecture

To meet the demands of a scalable and cost-effective solution, we adopted a serverless, decoupled architecture. This enabled independent scaling of services, reduced infrastructure costs, and supported agile development practices.

- **Frontend (React.js)**
  1. Built using React.js (Composition API)
  2. UI components implemented using Ant Design
  3. Utilized React Hooks for state management
  4. Configured React Router for navigation and route protection
  5. Integrated AWS Cognito for secure user authentication
- **Backend (AWS Serverless Stack)**
  1. Utilized AWS Lambda for backend logic via modular function handlers
  2. Amazon API Gateway exposed RESTful endpoints
  3. Chose DynamoDB for flexible NoSQL data storage
  4. Amazon S3 used for user profile image storage and retrieval
  5. Enabled email notifications using SNS (Simple Notification Service)
  6. Used AWS Cognito for authentication and authorization

Development Process

I followed a modular, sprint-based approach, breaking down major objectives into manageable tasks:

1. Backend First: Developed and tested Lambda functions using the AWS portal’s built-in tools.
2. Frontend Next: Built UI components independently before integrating them with backend APIs.
3. Infrastructure as Code: Wrote CloudFormation templates to define and deploy the backend infrastructure.

4. Frontend-Backend Integration: Connected frontend components to the backend services via API.
5. Deployment: Deployed the frontend using Elastic Beanstalk on EC2.
6. Testing: Each layer (backend, frontend, integration) was tested individually, culminating in end-to-end validation.

Key Lessons Learned

- **Technical Learnings**
  1. Serverless Architecture: This was my first time working with serverless infrastructure. I gained deep hands-on knowledge, including how to manage cold starts and execution timeouts.
  2. Function Granularity: Initially, the use of highly granular Lambda functions led to complexity. Grouping related functionality by business domain greatly improved maintainability and performance.
  3. DynamoDB Schema Design: Working with DynamoDB's flexible schema required a shift from traditional relational thinking. I learned to design based on query patterns rather than normalization.
  4. API Design & Error Handling: A well-structured API and robust frontend error handling were critical to delivering a reliable user experience.
  5. CloudFormation: Writing comprehensive CloudFormation stacks was initially challenging due to syntax unfamiliarity and limited testing methods. However, I gained significant knowledge over time through practice.
  6. Deployment Strategy: Separating the deployment strategies for backend and frontend clarified roles, reduced dependencies, and improved modularity.
- **Process Learnings**
  1. Breaking Down Tasks: Initially, I attempted to build multiple Lambda functions simultaneously, which led to delays. I learned to break large tasks into smaller, more manageable units for better focus and completion.
  2. Testing Strategy: This project marked my first in-depth experience with unit testing and integration testing, which I applied consistently to ensure code reliability.
  3. Documentation Importance: When integrating all Lambda functions with the frontend, I realized how crucial documentation is. Forgetting certain functionalities slowed me down—had I documented them earlier, it would have significantly improved my efficiency.