Version 30 Jul

### **Enhanced Product Requirements Document: P³ Situational Learning Sandbox - Landing Webpage/Dashboard**

1. Introduction

\* App Name: P³ Situational Learning Sandbox - Landing Webpage/Dashboard

\* Purpose: This is the central hub and entry point for users after sign-in. It provides an overview of their learning journey, quick access to different modules (Prepare, Practice, Perform), and essential account management options.

\* Target Audience: All registered users of the P³ Situational Learning Sandbox platform.

2. Goals & Objectives

\* To provide a clear and intuitive navigation experience to all core modules.

\* To offer a personalized overview of the user's learning progress and latest activities.

\* To ensure easy access to account settings and support resources.

\* To serve as the primary dashboard for user engagement and retention.

3. Key Features

\* 1. User Sign-in/Sign-up & Authentication:

\* Description: Secure entry point for new and returning users, directing them to the dashboard upon successful authentication.

\* Requirements: Integration with Amazon Cognito for user registration, login, and session management.

\* 2. Dashboard Overview:

\* Description: A central display showing key information such as:

\* Selection of Modules/Skills (e.g., Interview, Sales Conversation, Customer Service, Technical Assessment).

\* Progress Chart of Preparation and Practice modules completed (by % or hours).

\* A summary of past performance (e.g., last simulation score, overall average).

\* A prompt to start or continue their interview preparation journey.

\* Requirements: UI elements for displaying module cards, progress charts, and performance summaries. Backend logic to fetch aggregated user data from DynamoDB.

\* 3. Quick Navigation Buttons/Menu:

\* Description: Prominent buttons or menu options for quick access to the "Ready to Prepare" (Interview - Preparation Phase), "Ready to Practice" (Interview - Practice Phase), and "Performance Summary" (Interview - Perform Phase) sections.

\* Requirements: Clear UI navigation elements. Links to respective module pages.

\* 4. Knowledge Bank Search Bar (Library):

\* Description: An integrated search bar for quick access to a curated library of best practices (e.g., STAR Interview method) and other relevant information. This library could be organized by role, company, or industry, similar to how SimuInterviewer defines its scenarios.

\* Requirements: Search input field. Backend search functionality (DynamoDB/S3). Display area for search results.

\* 5. AI Coach Widget (Chatbot):

\* Description: A persistent chatbot icon (AI coach) at the bottom/side. Tapping it opens the AI coaching interface for real-time questions or drafting.

\* Requirements: Persistent UI widget. Integration with AI conversational model (via AWS Bedrock).

\* 6. Account Profile & Settings:

\* Description: A cog symbol or menu option for users to access preference settings, billing information, usage data, and logout options.

\* Requirements: UI for profile settings. Backend integration with Cognito for user attributes and potentially other billing/usage services.

4. User Flow (High-Level)

1. User lands on the sign-in/sign-up page.

2. Upon successful sign-in, the user is directed to the Dashboard.

3. From the Dashboard, the user can:

\* View their overall progress and summaries.

\* Click quick buttons to navigate to Prepare, Practice, or Perform modules.

\* Use the Knowledge Bank search bar.

\* Interact with the AI Coach chatbot.

\* Access their Account Profile and settings.

5. Technical Considerations (Updated with SimuInterviewer insights)

\* Frontend (Replit/Next.js/React): The primary web application served from Replit, handling all UI rendering, user interactions, and routing between modules. Adopting a component-based structure with a custom design system using a specific primary/background/accent color palette and typography (Inter font family for body and headlines, Source Code Pro for code examples) can enhance UI/UX and maintainability.

\* Backend (AWS Lambda via API Gateway):

\* Authentication calls to Amazon Cognito.

\* Data retrieval for dashboard summaries from Amazon DynamoDB.

\* Calls to Amazon Bedrock for AI Coach interactions.

\* Data retrieval from Amazon S3/DynamoDB for Knowledge Bank content.

\* Consider server-side AI flows for secure API key management and complex AI workflow orchestration, similar to how Genkit is used in the reference.

\* Database (AWS DynamoDB): Stores user profiles, progress data, and metadata for Knowledge Bank.

\* Storage (AWS S3): Stores static assets for the frontend and Knowledge Bank content.

\* Authentication (AWS Cognito): Manages user identities and access.

\* CDN (AWS CloudFront): (Optional for POC, but recommended for production) To serve static assets from S3 with low latency.

### **Enhanced Product Requirements Document: P³ Situational Learning Sandbox - Phase 1 Web App (Prepare Module)**

1. Introduction

\* App Name: P³ Situational Learning Sandbox - Prepare Module

\* Purpose: This module, accessed from the Landing Webpage, is designed to equip learners with foundational knowledge and strategic preparation for interview scenarios. It focuses on understanding interview criteria, accessing key information, and setting personalized learning goals.

\* Target Audience: Learners preparing for interviews (job, academic, specific role-based) who need structured guidance and foundational knowledge.

2. Goals & Objectives

\* To provide clear understanding of interview evaluation criteria.

\* To offer AI-generated ideal response examples for preparation.

\* To facilitate access to a comprehensive knowledge base related to interviews.

\* To enable personalized learning objectives and track initial progress.

3. Key Features

\* 1. Interview Scenario Setup:

\* Description: Allows the user to input details to create their interview scenario: Target Role, Hiring Company, Industry, Interview Stage (supporting multiple stages from HR screening to executive rounds, up to 5 stages as per SimuInterviewer), and Language (with multilingual support focusing on Asian languages).

\* Requirements: UI forms for input fields. Backend logic to store these details (DynamoDB) and prepare for AI interactions.

\* 2. Job Description (JD) Upload:

\* Description: Provides an option for the user to upload a Job Description document, which can be used to tailor AI-generated content and feedback.

\* Requirements: File upload functionality. Backend to store JD (S3) and potentially extract text for AI processing.

\* 3. Interview Criteria Display:

\* Description: Displays a detailed breakdown of evaluation criteria for interviewee responses (Relevant, Structured, Specific, Aligned with the Role, Outcome-Oriented), typically rated out of 5 stars (max 25 total). This forms a competency scoring system.

\* Requirements: UI to display criteria. Backend to fetch criteria (DynamoDB or hardcoded for POC).

\* 4. "What Good Looks Like" (WGLL) Generation:

\* Description: An LLM-powered feature that generates and guides learners through example scenarios, ideal responses, and effective communication approaches, serving as a benchmark.

\* Requirements: Integration with Amazon Bedrock to generate example content. UI to present scenarios and ideal responses.

\* 5. AI Coach Widget (Chatbot):

\* Description: A persistent chatbot icon (AI coach) accessible within the module. Tapping it opens an AI coaching interface for real-time questions, drafting responses, or seeking clarification.

\* Requirements: Chatbot UI. Backend connection to an AI conversational model (via AWS Bedrock).

\* 6. Instant Feedback on Answers:

\* Description: As the user inputs answers to questions in PREPARE mode, the system provides immediate feedback on their response, potentially leveraging contextual suggestions for improvement.

\* Requirements: Backend logic to analyze user input (using Amazon Comprehend or Bedrock). UI to display feedback. "Next question" button to proceed.

\* 7. Review Feedback and STAR Evaluation Results:

\* Description: Upon finishing preparation, the user can review consolidated feedback and STAR evaluation results for their practice answers. This aligns with a STAR-based assessment methodology.

\* Requirements: UI to display aggregated feedback. Backend to store and retrieve evaluation results.

\* 8. Knowledge Bank Search Bar (Library):

\* Description: Integrated search within the module for quick access to best practices and curated information.

\* Requirements: Search bar UI. Backend search logic (DynamoDB/S3).

4. User Flow (High-Level)

1. User navigates to "Prepare Mode" from the Dashboard.

2. User inputs interview scenario details.

3. User has the option to upload a JD.

4. User starts PREPARE mode by inputting answers to questions.

5. User reads instant feedback on answers and clicks "Next question."

6. User can chat with the AI Coach.

7. User can access the Knowledge Bank search bar.

8. User finishes preparation and reviews feedback/STAR Evaluation results.

5. Technical Considerations (Updated with SimuInterviewer insights)

\* Frontend (Replit/Next.js/React): Handles UI for forms, text input, displaying AI-generated content and feedback. Utilize Radix UI components for accessible primitives and Tailwind CSS for styling, adhering to the defined design system.

\* Backend (AWS Lambda via API Gateway):

\* Processes user input and JD uploads.

\* Orchestrates calls to Amazon Bedrock for WGLL generation and AI Coach responses.

\* Stores scenario details, user answers, and feedback in Amazon DynamoDB.

\* Manages Knowledge Bank search queries.

\* Consider server-side AI flows for robustness and secure API key management, mirroring the ai/flows structure for specific AI tasks.

\* AI/ML (AWS Bedrock): Core for WGLL generation and AI Coach.

\* Database (AWS DynamoDB): Stores all user-specific preparation data, criteria, and progress.

\* Storage (AWS S3): Stores uploaded JDs and Knowledge Bank content.

### **Enhanced Product Requirements Document: P³ Situational Learning Sandbox - Phase 2 Web App (Practice Module)**

1. Introduction

\* App Name: P³ Situational Learning Sandbox - Practice Module

\* Purpose: This module, accessed from the Landing Webpage, provides an immersive and interactive environment for users to practice their interview skills through dynamic, AI-powered role-play simulations. It focuses on real-time interaction and immediate feedback.

\* Target Audience: Learners who have completed the preparation phase and are ready for hands-on, simulated interview practice.

2. Goals & Objectives

\* To simulate realistic interview experiences with an adaptive AI interviewer.

\* To enable natural voice-based communication during practice sessions.

\* To provide instant, constructive feedback for immediate learning.

\* To offer a diverse range of interview scenarios across different contexts.

\* To support practice in multiple languages for cultural relevance.

3. Key Features

\* 1. Practice Scenario Selection:

\* Description: User chooses their practice scenario from a curated list, allowing them to focus on specific interview types or skill areas. This could include varied interviewer personas.

\* Requirements: UI for displaying and selecting scenarios. Backend to manage scenario data (DynamoDB).

\* 2. Pre-Simulation Setup & Briefing:

\* Description: User sets up pre-simulation details (Target Role, Hiring Company, Interview Stage, Language) and reads a briefing screen containing instructions, persona details, and learning objectives for the chosen scenario.

\* Requirements: UI forms for setup. Display area for briefing content.

\* 3. AI Interviewer & Dynamic Role Play:

\* Description: The central component that conducts dynamic, real-time interviews. The AI interviewer (with an AI voice avatar) adapts its questions and follow-ups based on the learner's responses, simulating a realistic interview flow. It supports dynamic question progression based on the interview stage.

\* Requirements: Integration with Amazon Bedrock for dynamic conversational AI, context management, and adaptive questioning.

\* 4. Voice Interaction (Dual Input Modes):

\* Description: Enables seamless verbal communication with the AI interviewer using voice input (voice recording via Web Audio API/MediaRecorder API), and receives voice responses back. This feature facilitates realistic practice of verbal communication skills. Dual input modes (voice recording and text input) should be supported.

\* Requirements: Integration with Amazon Transcribe for speech-to-text. Integration with Amazon Polly for text-to-speech (custom TTS/STT integration via server actions as per SimuInterviewer concept). Real-time audio processing and streaming capabilities.

\* 5. Real-Time Feedback:

\* Description: Delivers instant, constructive feedback during the interview session, covering aspects like clarity, confidence, relevance of answers, and communication style. This helps learners adjust their approach on the fly. It should leverage AI-powered analysis using competency frameworks and provide contextual suggestions for improvement.

\* Requirements: Integration with Amazon Comprehend for sentiment/keyphrase analysis. Logic to compare user responses against ideal criteria and generate feedback. UI to display real-time feedback subtly (e.g., subtle progress indicators and response highlighting).

\* 6. Role Play Completion/Time Out:

\* Description: The role play concludes either when the user completes it or after a set time limit (e.g., 10 minutes).

\* Requirements: Timer functionality. Backend logic to manage session end.

\* 7. Practice Assessment & Scores Review:

\* Description: After the role play, the user can review a detailed assessment of their performance, including scores and suggested improvements.

\* Requirements: UI for displaying assessment results. Backend to aggregate and present scores from simulation data.

\* 8. Simulation History & Review:

\* Description: Allows users to select 'History' to review past simulations, including transcripts (text or voice) with an option to download. This aligns with interview transcript storage and retrieval.

\* Requirements: UI for history Browse. Backend to store transcripts (DynamoDB/S3). Playback functionality for voice (if stored).

\* 9. Option to Restart Practice:

\* Description: User has the option to restart the same practice scenario immediately.

\* Requirements: UI button for restart. Backend logic to reset simulation state.

4. User Flow (High-Level)

1. User navigates to "Practice Mode" from the Dashboard.

2. User selects a practice scenario.

3. User sets up pre-simulation details.

4. User reads the pre-simulation briefing screen.

5. User clicks "I am ready" to start the simulation.

6. User interacts with the AI Interviewer via voice/text.

7. Real-time feedback is provided.

8. Role play completes or times out.

9. User reads practice assessment and scores.

10. User can select 'History' to review past simulations or 'START again' of the same practice.

11. User can select 'Home' to get back to the Landing Page.

5. Technical Considerations (Updated with SimuInterviewer insights)

\* Frontend (Replit/Next.js/React): Highly interactive web interface for real-time chat/voice interaction, feedback display, and scenario management. Emphasis on responsive design for desktop and mobile compatibility. Use React hooks for state management and localStorage for interview data persistence.

\* Backend (AWS Lambda via API Gateway):

\* Manages simulation turns and state.

\* Orchestrates calls to Amazon Bedrock for AI Interviewer.

\* Handles real-time audio processing with Amazon Transcribe and Amazon Polly.

\* Processes feedback with Amazon Comprehend.

\* Stores simulation data and transcripts in Amazon DynamoDB/S3.

\* Implement server-side AI flows (analyze-interview-response, generate-interviewer-persona, provide-real-time-feedback, speech-to-text, text-to-speech) to manage complex AI interactions and secure API keys.

\* AI/ML (AWS Bedrock, Transcribe, Polly, Comprehend): Core for all AI-driven interactions and analysis.

\* Database (AWS DynamoDB): Stores simulation state, conversation history, and performance metrics.

\* Storage (AWS S3): Stores scenario content and optionally raw audio/transcripts.

### **Enhanced Product Requirements Document: P³ Situational Learning Sandbox - Phase 3 Web App (Perform Module)**

1. Introduction

\* App Name: P³ Situational Learning Sandbox - Perform Module

\* Purpose: This module, accessed from the Landing Webpage, is dedicated to assessing and validating a learner's mastery of interview skills acquired during the practice phase. It provides comprehensive performance analysis, benchmark comparisons, and pathways to certification and continued growth.

\* Target Audience: Learners who have engaged in practice simulations and are looking to evaluate their proficiency, gain recognition, and identify areas for ongoing development.

2. Goals & Objectives

\* To provide a clear and detailed overview of a learner's performance.

\* To offer AI-generated benchmarks for self-assessment and improvement.

\* To motivate learners through gamification and verifiable certifications.

\* To provide personalized guidance for continued skill development.

\* To enable optional sharing of achievements within a community.

3. Key Features

\* 1. Performance Score Card:

\* Description: A comprehensive dashboard displaying an overall performance score and detailed scores across various pre-defined criteria (e.g., communication, problem-solving, cultural alignment) derived from practice sessions. This should specifically include competency scoring (1-5 scale) across multiple dimensions and a detailed performance breakdown with specific improvement suggestions, potentially using a STAR-based assessment methodology.

\* Requirements: UI for a dashboard-like summary (e.g., using a charting library like Recharts). Backend logic to fetch and aggregate scores from DynamoDB.

\* 2. Qualitative Feedback:

\* Description: Provides AI-generated qualitative observations about the learner's strengths and areas for improvement, using natural language to offer actionable insights.

\* Requirements: Integration with AI models (e.g., Bedrock) to generate insightful textual feedback. UI to display narrative feedback.

\* 3. "What Good Looks Like" Examples (Recommendation):

\* Description: Recommends examples of "What Good Looks Like" (e.g., video with prompts) to further guide the user's learning based on their performance gaps. This is a crucial element for improvement.

\* Requirements: Integration with Amazon Bedrock for generating/selecting relevant WGLL examples. UI to display examples.

\* 4. Personalized Learning Path:

\* Description: Based on the learner's performance data, the system intelligently suggests custom learning modules, drills, or resources designed to address identified weaknesses and reinforce strengths, guiding future learning.

\* Requirements: Recommendation engine logic based on performance data. UI for displaying personalized suggestions. Integration with content from the Prepare phase or new drills.

\* 5. Gamified Learning Journey & Pop-up Screen:

\* Description: Incorporates points, badges, leaderboards, and other gamification elements to motivate learners, track progress towards skill mastery, and encourage continuous engagement. A pop-up screen will reward users with incentives.

\* Requirements: Gamification system backend (score tracking, badge assignments). UI elements for displaying points, badges, and leaderboard rankings.

\* 6. Certifiable Skills/Badges:

\* Description: Upon achieving specific performance milestones and demonstrating proficiency, learners can earn verifiable digital certificates or badges, recognizing their acquired skills.

\* Requirements: Define certification criteria. Backend for issuing and managing digital certificates/badges. Potential integration with a blockchain or verifiable credential service for authenticity.

\* 7. Shareable Progress:

\* Description: An optional feature allowing learners to share anonymized performance data or achievements with peers or a community, fostering collaborative learning and recognition.

\* Requirements: UI for sharing option. Anonymization logic. Integration with a community platform or social media (if applicable).

4. User Flow (High-Level)

1. User navigates to "Perform Mode" from the Dashboard.

2. User reviews their Performance Summary.

3. User inputs Self Reflection to interact with the AI Coaching Chatbot.

4. User can share progress (optional).

5. User sees the Gamification pop-up screen.

6. User sees recommendations to move to the next stage.

7. User sees recommendations for "What Good Looks Like" examples.

8. User can select 'Home' to get back to the Landing Page.

5. Technical Considerations (Updated with SimuInterviewer insights)

\* Frontend (Replit/Next.js/React): Rich dashboard UI with data visualizations, interactive elements for self-reflection, and display of gamification/badges. Ensure responsive design for desktop and mobile compatibility.

\* Backend (AWS Lambda via API Gateway):

\* Aggregates and analyzes performance data from DynamoDB.

\* Manages gamification logic and badge assignment.

\* Orchestrates calls to Amazon Bedrock for self-reflection prompts and WGLL recommendations.

\* Generates recommendations for next steps.

\* Focus on server-side AI flows for assessment and feedback generation.

\* AI/ML (AWS Bedrock): Powers the self-reflection chatbot and WGLL recommendations, and contributes to performance analysis.

\* Database (AWS DynamoDB): Stores all historical performance data, gamification data, and self-reflection notes.

\* Data Analytics: Potentially AWS QuickSight for richer dashboarding and reporting if desired beyond basic UI.

Building your P³ Situational Learning Sandbox MVP on Replit in stages, while leveraging AWS for backend services, is a smart approach for rapid prototyping and iteration. This plan breaks down the development process for each phase, focusing on getting core functionality working before adding complexity.

Here's a recommended staged build process on Replit:

### **Overall Strategy**

* **Replit as Frontend & Orchestrator:** Your Replit application will serve the web interface (HTML, CSS, JavaScript) and act as the central orchestrator, making calls to various AWS backend services.1
* **AWS as Backend Services:** All heavy lifting, AI processing, database operations, and user management will reside on AWS.
* **Iterative Development:** Build one feature or a small set of features at a time, test thoroughly, and then move to the next.

### **Stage 0: Foundation & Core Setup (Days 1-3)**

This stage is about setting up your development environment and ensuring basic connectivity.

1. **Replit Project Initialization:**
   * Create a new Repl: Choose **Python** (e.g., Flask or FastAPI template) as it's excellent for AWS SDK integration and web serving.
   * Name your Repl (e.g., P3-Situational-Learning-Sandbox-MVP).
2. **Install Core Dependencies:**
   * Open pyproject.toml (or requirements.txt) and add:
     + boto3 (AWS SDK for Python)2
     + Flask (or FastAPI and uvicorn) for your web server.3
   * Replit will automatically install these.
3. **Secure AWS Credentials:**
   * In Replit's "Secrets" tab (lock icon), add environment variables:
     + AWS\_ACCESS\_KEY\_ID: Your AWS Access Key ID.
     + AWS\_SECRET\_ACCESS\_KEY: Your AWS Secret Access Key.
     + AWS\_REGION: Your preferred AWS region (e.g., ap-southeast-1).
   * **Never hardcode credentials in your code!**
4. **Basic AWS Cognito Setup:**
   * In the AWS Management Console, create a new **Cognito User Pool**. Configure basic user attributes (email, password).
   * Note down the User Pool ID and Client ID.
5. **Basic Web Server on Replit:**
   * Create a simple app.py (or similar) that serves a "Hello World" HTML page or a basic Flask route.
   * Ensure it runs correctly on Replit's preview window.

### **Stage 1: Landing Webpage & User Flow (Week 1)**

Focus on getting users authenticated and navigating the main hub.

1. **User Sign-in/Sign-up:**
   * **Frontend (Replit):** Create basic HTML forms for sign-up and sign-in. Use JavaScript to capture user input.
   * **Backend (Python in Replit):**
     + Use boto3.client('cognito-idp') to interact with your Cognito User Pool for sign\_up, confirm\_sign\_up, initiate\_auth, and respond\_to\_auth\_challenge (for MFA if enabled).4
     + Implement session management (e.g., using Flask sessions to store user tokens).
     + Redirect authenticated users to the dashboard.
2. **Dashboard/Landing Page Layout:**
   * **Frontend (Replit):** Design the main dashboard.html (or equivalent) with placeholders for:
     + Module selection (Prepare, Practice, Perform cards/buttons).
     + Progress charts/summaries.
     + Knowledge Bank search bar.
     + AI Coach widget.
     + Account Profile link.
   * **Backend (Python in Replit):** Create a route (/dashboard) that renders this HTML after successful login.
3. **Navigation Links:**
   * **Frontend (Replit):** Make the "Ready to Prepare," "Ready to Practice," and "Performance Summary" buttons/links navigate to simple placeholder pages (e.g., /prepare, /practice, /perform) served by your Replit app. These will be built out in later stages.

### **Stage 2: Phase 1 (Prepare Module) - Core Features (Weeks 2-3)**

Build out the initial preparation functionality and essential AI integrations.

1. **Interview Scenario Setup & JD Upload:**
   * **Frontend (Replit):** Create an HTML form with input fields for Target Role, Company, Industry, Stage, Language, Input Mode. Add a file input for JD upload.
   * **Backend (Python in Replit):**
     + Create a route (e.g., /prepare/setup) to handle form submission.5
     + Store scenario details in **DynamoDB** (create a user\_scenarios table).
     + Use boto3.client('s3') to upload the JD file to an **S3 bucket**.6 Store the S3 object key in DynamoDB.
2. **Knowledge Bank Search Bar:**
   * **AWS Setup:** Create a **DynamoDB table** (knowledge\_bank\_articles) for article metadata (title, snippet, keywords, S3 key). Upload sample article content to **S3**.
   * **Frontend (Replit):** Integrate the search bar from the Dashboard/Prepare page. Display results as a list of titles/snippets.
   * **Backend (Python in Replit):** Create a route (e.g., /knowledge-bank/search) that queries DynamoDB based on keywords and fetches full content from S3 if a user clicks an article.
3. **AI Coach Widget (Text-Only):**
   * **Frontend (Replit):** Implement the chat UI within the Prepare module.
   * **Backend (Python in Replit):**
     + Create a route (e.g., /ai-coach/chat) that accepts user messages.
     + Use boto3.client('bedrock-runtime') to invoke an LLM (e.g., Amazon Titan Text or Anthropic Claude) with the user's prompt.7
     + Return the AI's text response to the frontend.
     + **(Optional for POC):** Implement simple session history management (e.g., a list in memory per user session).
4. **Interview Criteria Display & "What Good Looks Like" (WGLL):**
   * **Frontend (Replit):** Display the 5 evaluation criteria (Relevant, Structured, etc.). Add a section for WGLL examples with a "Generate Example" button.
   * **Backend (Python in Replit):**
     + **WGLL:** Create a route (e.g., /prepare/generate-wgll) that calls **Amazon Bedrock** with a prompt to generate an ideal response for a given scenario. Return the text.
     + Store the generated WGLL and user answers/feedback in **DynamoDB** for future review.
5. **Progress Tracking & Custom Learning Path:**
   * **Frontend (Replit):** Basic display of completed items for progress. Simple form for selecting learning path topics.
   * **Backend (Python in Replit):** Update DynamoDB records as users complete steps or save learning path preferences.

### **Stage 3: Phase 2 (Practice Module) - Core Features (Weeks 4-5)**

This stage involves significant AI integration, particularly for voice.

1. **Practice Scenario Selection & Pre-Simulation Briefing:**
   * **Frontend (Replit):** UI to display a list of practice scenarios (from DynamoDB). Display briefing text on selection.
   * **Backend (Python in Replit):** Routes to fetch scenario data from DynamoDB.
2. **AI Interviewer (Text-First, then Voice):**
   * **Frontend (Replit):**
     + Implement the chat interface for the interview.
     + Add "Record Voice" and "Stop Recording" buttons.
     + Add an audio player for AI voice responses.
   * **Backend (Python in Replit):**
     + **Text-only Interview Flow:** Create a route (e.g., /practice/interview-turn) that takes user text, calls **Amazon Bedrock** for the AI Interviewer's response, and returns the AI's text.
     + **Voice Integration (Major Step):**
       - **User Voice Input:** When the user records, capture the audio (e.g., as webm or wav). Send this audio as a stream/file to a backend route.
       - **Amazon Transcribe:** In the backend, use boto3.client('transcribe') to send the audio to Transcribe. For real-time feedback, consider the streaming API or short audio chunks.
       - **AI Voice Output:** Once Bedrock generates the AI's text response, send this text to boto3.client('polly') to convert it to audio (e.g., MP3).8 Stream or send this audio back to the frontend.
3. **Real-Time Feedback:**
   * **Backend (Python in Replit):** After Transcribe converts user's voice/text, send the text to boto3.client('comprehend') for sentiment analysis or key phrase extraction.9
   * **Frontend (Replit):** Display this feedback in the UI alongside the conversation.
4. **Session Management & History:**
   * **Backend (Python in Replit):** Store each turn of the interview (user input, AI response, feedback) in **DynamoDB** to reconstruct the session history.
   * **Frontend (Replit):** Create a "History" view that fetches past sessions from DynamoDB and displays the transcript.

### **Stage 4: Phase 3 (Perform Module) & Enhancements (Week 6)**

Focus on analytics, gamification, and advanced recommendations.

1. **Performance Summary & Scorecard:**
   * **Frontend (Replit):** Design a dashboard to display aggregated performance data (scores, criteria breakdown).
   * **Backend (Python in Replit):** Retrieve and aggregate simulation data from **DynamoDB**. Perform calculations to generate overall scores and specific metrics.
2. **Qualitative Feedback & Self-Reflection:**
   * **Frontend (Replit):** Display AI-generated qualitative feedback. Implement a chat UI for self-reflection prompts.
   * **Backend (Python in Replit):**
     + Use **Amazon Bedrock** to generate qualitative feedback based on analysis of past simulation transcripts.
     + Reuse AI Coach logic for the self-reflection chatbot, storing user's reflection notes in DynamoDB.
3. **Gamification (Basic):**
   * **Frontend (Replit):** Display earned badges, simple point totals. Implement a "Gamification Pop-up" when criteria are met.
   * **Backend (Python in Replit):** Simple logic to assign badges/points based on completion of simulations or achieving certain scores. Store in **DynamoDB**.
4. **Recommendations & Next Stage:**
   * **Frontend (Replit):** Display text-based recommendations (e.g., "You're ready for Advanced Interviews!").
   * **Backend (Python in Replit):** Logic to analyze performance data (from DynamoDB) and recommend next steps or "What Good Looks Like" examples using **Amazon Bedrock**.
5. **Share Progress (Optional):**
   * **Frontend (Replit):** Add "Share" buttons (e.g., copy link, mock LinkedIn share).
   * **Backend (Python in Replit):** (For POC) Simply generate a static link or prepare text for sharing. Actual integration with social platforms is more complex and likely out of scope for a quick POC.

### **Stage 5: Refinement, Testing & Monitoring (Continuous)**

This stage runs concurrently throughout development, but gets dedicated focus towards the end.

1. **Error Handling & Robustness:** Implement try-except blocks for all AWS API calls and user inputs. Provide meaningful error messages to the user.
2. **Performance Optimization:** Monitor API response times. For critical paths, look for ways to reduce latency (e.g., batching requests, optimizing DynamoDB queries).
3. **UI/UX Polish:** Refine the look and feel, ensure responsiveness across different screen sizes. Improve user experience.
4. **Comprehensive Testing:**
   * **Unit Tests:** Test individual functions.
   * **Integration Tests:** Test interactions between your Replit app and AWS services.
   * **User Acceptance Testing (UAT):** Have a small group of users test the full flow and provide feedback.
5. **Logging & Monitoring:**
   * Ensure your Python application on Replit logs relevant events and errors. These logs will be visible in Replit's console.
   * Monitor your AWS services using **Amazon CloudWatch** for metrics (Lambda invocations, DynamoDB throughput, Bedrock token usage) and logs.10 This is crucial for managing your POC credits.
   * Use **AWS CloudTrail** for auditing API calls.

This staged approach allows you to build a functional MVP on Replit, proving out the core value proposition with AWS services, within your 4-6 week POC timeframe. Remember to leverage Replit's rapid iteration capabilities!