Given your goals (learn AI/ML + get solid AWS cloud infra experience with serverless + modern UI), here's a concrete project that ticks **all** your requirements and also aligns with your career:

Proposed Project

AI-Powered Test Analytics & Recommendation Hub (TARH)

A cloud-native web app that:

- Ingests **test cases, defects, and release data** (very close to your real work)
- Uses **ML models** to:
    - Predict **defect-prone modules** and **high-risk test cases**
    - Recommend **which tests to prioritize** in the next cycle
- Exposes everything through a **modern web dashboard** with rich visuals and chat-style AI assistant.

You can later talk about this as a real portfolio project in interviews ("Built a serverless AI test analytics platform on AWS").

How it fits your requirements

1️⃣ Modern UI

Tech:

- **React + TypeScript + Tailwind** (or Material UI)
- Host via **AWS Amplify Hosting** or **S3 + CloudFront**

Features:

- Login / signup (Cognito)
- Pages:
    - **Dashboard**: charts on defects, execution status, predicted risk scores
    - **Test Explorer**: list of test cases, filters, tags, ML "risk" badges
    - **Defect Insights**: heatmaps by module, release, severity
    - **AI Assistant** pane: "Ask questions like ChatGPT, but about your test data"

2️⃣ DB in the cloud

You can intentionally design **two types of storage** to learn both relational and NoSQL.

- **Relational DB** (structured relationships):
    - **Amazon Aurora Serverless v2 (PostgreSQL)**

    - Tables like:
        - projects, releases, test_cases, test_runs, defects
- **NoSQL DB (for quick access / session / caching)**:
    - **Amazon DynamoDB**
    - Use it for:
        - Storing AI assistant's conversation context
        - Storing feature-store-like denormalized records: { test_case_id, risk_score, last_run_status, tags }

If you want to keep it simple initially: start with **only Aurora Serverless**, add DynamoDB later as a "Phase 2".

3️⃣ Cloud Storage (files, datasets, ML artifacts)

Amazon S3:

- **Raw data**:
    - CSV/Excel exports of test cases, defects, logs
- **Processed / curated data**:
    - Cleaned training data: s3://tarh-ml/processed/train/…
- **ML artifacts**:
    - Trained model files (if you train using SageMaker and later export)
- **App static assets**:
    - Exported reports, downloadable PDFs, archived dashboards

You'll learn data layout patterns like:

```
s3://tarh-data/
  raw/
    charlie_exit/...
    alpha_uplift/...
  processed/
    classification/
    timeseries/
  models/
    test_risk_model/
    defect_volume_forecast/
```

4️⃣ Application logic in a serverless environment

Core runtime: **AWS Lambda**

Typical flows:

- **Ingestion Lambda**
  - Triggered by file upload to S3 (via Event Notifications)
  - Reads CSV/Excel with pandas, validates, and loads into Aurora tables
- **API Lambdas (behind API Gateway)**
  - GET /dashboard/summary → query Aurora + DynamoDB → return JSON to UI
  - GET /test-cases?project=Alpha&release=2025-04 → paginated list
  - POST /ai/ask → call Bedrock / SageMaker + your DB → return answer
- **Batch / ML Lambdas or SageMaker jobs**
  - Scheduled via **EventBridge** to recompute risk scores nightly or weekly

---

5️⃣ Use "full spectrum" of AWS services

Here's a **suggested AWS stack** for this project:

Identity & Access:

- **IAM** – roles for Lambda, SageMaker, RDS access
- **Amazon Cognito** – user pools for sign-in, JWT integration with API Gateway

Networking:

- **VPC** – private subnets for Aurora + SageMaker
- **VPC Endpoints** – secure access from Lambda to RDS/S3
- **API Gateway** – public HTTPS endpoints for your React app

Compute / Serverless:

- **AWS Lambda** – APIs, data processing
- **AWS Step Functions** (Phase 2) – orchestrate complex ML pipelines

Data & Storage:

- **Amazon S3** – datasets + artifacts
- **Amazon Aurora Serverless v2** – relational DB
- **Amazon DynamoDB** – fast lookups, conversation context, feature cache

AI / ML:
You have two good paths — both great for learning:

1. **Path A – SageMaker-based classic ML (great for exam prep)**
    - Use **Amazon SageMaker** to:
        - Train **XGBoost** model for:
            - *Test case failure prediction* → classification
            - *Defect volume forecast per release* → regression / time-series
        - Host a **SageMaker endpoint** for real-time inference
    - Lambda calls the SageMaker endpoint when:
        - Dashboard needs fresh risk scores
        - User asks: "What are the top 10 riskiest test cases in Alpha Uplift?"
2. **Path B – Bedrock for Generative AI (RAG style)**
    - Use **Amazon Bedrock** (Titan / Claude) for:
        - Natural language Q&A over your **test cases and defects**
        - "Generate test scenarios for module X, given these requirements"
    - Store embeddings / text chunks in:
        - DynamoDB or OpenSearch (if you want to go deeper)

You can **combine both**:

- SageMaker → predictive analytics
- Bedrock → conversational insights & test-case suggestion

Monitoring & Observability:

- **CloudWatch Logs & Metrics** – Lambda, API Gateway, scheduled jobs
- **CloudWatch Dashboards** – infra monitoring
- Optional: **X-Ray** for tracing

6️⃣ AI/ML Learning Opportunities (Concrete)

Here's what you'll learn ML-wise:

1. **Data Engineering**
    - Import, clean, and join test cases + defects + releases
    - Create labeled dataset: e.g.,
        - label = 1 if test case failed in last N runs / caused P1 defects

        - label = 0 otherwise

2. **Feature Engineering**
    - Features like:
        - Number of steps in test case
        - Past failure rate
        - Module complexity (approximate)
        - Time since last edit
        - Number of linked defects
3. **Modeling**
    - Start with **XGBoostClassifier / RandomForest** in SageMaker or locally
    - Evaluate with:
        - AUC, Precision/Recall, F1
        - Confusion matrix
4. **Deployment & Inference**
    - Host the model in **SageMaker endpoint**
    - Or package a small model as a .pkl and serve via Lambda for small-scale use
5. **Generative AI (optional, but highly relevant)**
    - Use Bedrock to:
        - Generate test cases from a requirement description
        - Summarize defect clusters: "What are the main themes in P1 defects this quarter?"
        - Implement a chat UI in the frontend → "AI QA Co-pilot"

7️⃣ How we can proceed (no need to answer, I'll assume this plan)

I'll assume we're going with **this TARH project**. In the next steps, I can:

1. **Draw the high-level architecture** (text + simple block diagram description)
2. **Define phases**:
    - Phase 0 – Minimal local prototype with fake CSVs and a notebook
    - Phase 1 – AWS infra + serverless APIs + basic dashboard
    - Phase 2 – ML model with SageMaker
    - Phase 3 – GenAI assistant with Bedrock

3. Start with:
    - **AWS resource list + naming conventions**
    - **Initial data model (Aurora schema)**
    - **Skeleton of the frontend and backend APIs**

If this direction feels good, I'll immediately start with **Phase 0 + architecture** in the next message and we'll build it step by step like we did for your earlier projects—but with a clean slate and a clear ML + cloud story.