

Project Report: Student Course Registration System

Varsha SP

22 September, 2025

1. Abstract

The Student Course Registration System (SCRS) is a Java-based application designed to streamline the process of course enrollment for students. It leverages DynamoDB Local as the NoSQL backend, supports both console interaction and a demo web interface built using Spark Java, and is deployed using Docker and Jenkins CI/CD pipelines. The system enforces strict business rules such as maximum enrollments, waitlisting, deadlines, and audit logging. The project follows a layered architecture (DAO → Service → Application) and adheres to SOLID principles and well-known design patterns, ensuring scalability, maintainability, and extensibility. A lightweight Spark Java web UI was added as a demonstration layer; the application remains console-first and the structured design of the project allowed seamless addition of a WebApp UI class without altering the existing core architecture.

2. Introduction

The Student Course Registration System addresses the challenge of managing student course enrolments in a lightweight, scalable manner using NoSQL. This system is built with DynamoDB Local to simulate real-world AWS DynamoDB behaviour while remaining fully offline and containerized for testing and deployment.

The system provides:

- Student registration and login (with hashed passwords).
- Enrolment and dropping of courses.
- Waitlist management with FIFO and auto-promotion.
- Profile and course viewing with deadlines enforced.
- Audit logging of all key actions.

The project is primarily console-based, but a minimal Spark Java web UI was introduced later for demonstration purposes. This addition validated the robustness of the design, as the WebApp UI directly reused service classes without changes to the core project structure.

3. Technology Stack

- Language: Java 17
 - Build & Dependency: Maven
 - Testing: JUnit 5, Mockito (for unit testing where needed)
 - Data Store: DynamoDB Local (AWS SDK v2)
 - Password hashing: bcrypt (at.favre.lib.crypto.bcrypt)
 - Web: Spark Java (demo UI)
 - Containerization: Docker, Docker Compose
 - CI/CD: Jenkins (Declarative Pipeline)
 - Infrastructure tooling: Docker Desktop, DynamoDB Workbench (for visual validation)
-

4. System Design

4.1 Architecture

The architecture follows a layered design:

1. **Model Layer** → Core entities (Student, Course, StudentLog).
2. **DAO Layer** → Interfaces for data access (StudentDao, CourseDao, LogDao).
3. **Repository Layer** → DynamoDB-based DAO implementations.
4. **Service Layer** → Business logic (StudentService, EnrollmentService, CourseService).
5. **Application Layer** → Console App (App.java) and Spark WebApp.java.

This separation enables testing (in-memory DAOs for unit tests) and easy substitution of persistence backends.

4.2 Design Principles and Patterns

SOLID Principles

- Single Responsibility: Each service and DAO have one well-defined responsibility.
- Open/Closed: Easily extensible (WebApp UI added without modifying existing classes).
- Liskov Substitution: Interfaces (DAOs) ensure substitutability (e.g., in-memory DAOs for testing).
- Interface Segregation: Fine-grained DAO interfaces for Course, Student, and Logs.
- Dependency Inversion: Services depend on DAO interfaces, not implementations; actual implementations are injected at composition time.

Design Patterns

- Strategy Pattern: Enrolment strategies (direct seat allocation vs. waitlist handling).
- Singleton Pattern: DynamoDBClientUtil ensures a single shared DynamoDB client.
- Factory Pattern: DAO instantiation provides flexibility (switch between DynamoDB and in-memory DAOs).

4.3 Database Design

- **Students Table** → Stores studentId, email, hashed password, enrolled and waitlisted courses.
- **Courses Table** → Stores courseId, maxSeats, currentEnrolledCount, enrolledIds, waitlistIds.
- **StudentLogs Table** → Stores logId, studentId, action, courseId, timestamp.

NoSQL design ensures scalability and supports atomic updates using DynamoDB ConditionExpression.

5. Implementation

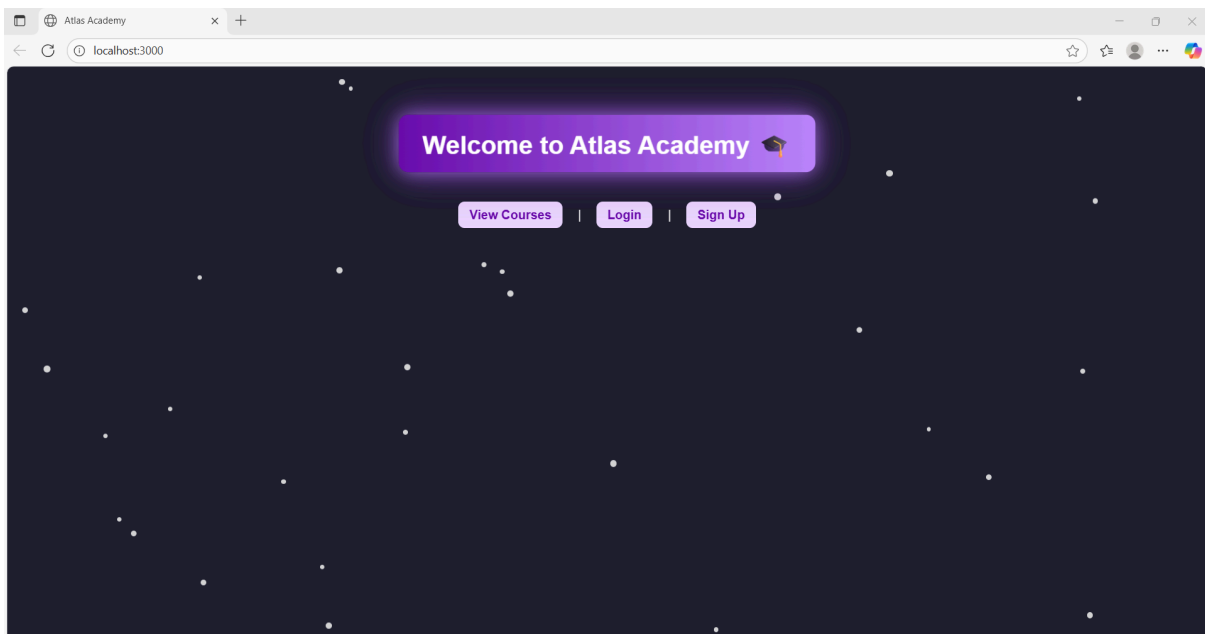
5.1 Console App

- Simple menu-driven flow implemented in App.java (main).
- All protected operations check SessionStore.
- Provides a text-driven menu for signup, login, enrollment, dropping courses, and viewing profiles.
- Uses services for business logic; minimal console I/O logic in app class.

5.2 Spark Java Web UI (Demo)

- WebApp.java provides a lightweight HTML UI for presentation and demonstration.
- Uses same service instances (StudentService, CourseService, EnrollmentService) via constructor injection; no changes required to the service layer when adding the UI.
- Inline CSS provides a polished look while preserving console-first behaviour.

Why this is important: The ability to add a controller layer (Spark routes) that reuses service classes shows that the design strictly follows separation of concerns and adheres to SOLID.



5.3 Code Highlights

- Password Security: Implemented using BCrypt hashing.
 - Session Management: In-memory session store with UUID tokens and expiry.
 - Waitlist Management: FIFO-based, with auto-promotion when seats free up.
 - Logging: Every action recorded in StudentLogs.
-

6. Testing

6.1 Unit Tests

- Implemented using JUnit 5 with in-memory DAOs used to isolate business logic.
- Coverage keyed on core flows: signup/login, enroll when seat available, enroll → waitlist when full, drop → promotion from waitlist, duplicate prevention & limits.

6.2 Integration Tests

- Run against DynamoDB Local using test utility DynamoDBClientUtil pointing to local endpoint.
- Integration test class EnrollmentFlowsIntegrationTest seeds isolated courses per test and cleans them up after execution.
- Verified atomic seat allocation, waitlist promotions, and deadlines.

6.3 Test Evidence

- Jenkins console and local terminal executions show passing unit and integration tests.
- All tests passed successfully, ensuring system correctness.

Pipeline Execution Screenshots of successful test execution in Jenkins:



```
Jenkins / Atlas-AMZ-Capstone-SCRS-Docker-Pipeline / #27
[INFO] --- resources:3.3.1:testResources (default-testResources) @ atlas-capstone ---
[INFO] skip non existing resourceDirectory /var/jenkins_home/workspace/Atlas-AMZ-Capstone-SCRS-Docker-Pipeline/src/test/resources
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ atlas-capstone ---
[INFO] Changes detected - recompiling the module! :dependency
[INFO] Compiling 5 source files with javac [debug target 17] to target/test-classes
[WARNING] system modules path not set in conjunction with -source 17
[INFO] --- surefire:3.1.2:test (default-test) @ atlas-capstone ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] T E S T S
[INFO]
[INFO] Running com.atlas.tests.EnrollmentFlowsUnitTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.598 s -- in com.atlas.tests.EnrollmentFlowsUnitTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 21.337 s
[INFO] Finished at: 2025-09-21T07:52:09Z
[INFO]
Post stage
[Pipeline] junit
```

Console Test Execution screenshots Maven Verify Results:

```
INFO] -----
INFO] T E S T S
INFO] -----
INFO] Running com.atlas.tests.EnrollmentFlowsUnitTest
INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.783 s -- in com.atlas.tests.EnrollmentFlowsUnitTest
INFO] Results:
INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
INFO]
INFO] --- jar:3.3.0:jar (default-jar) @ atlas-capstone ---
INFO] Building jar: C:\Users\spvar\OneDrive\Desktop\CapstoneAtlas\StudentCourseRegistrationSystem\target\atlas-capstone-1.0.0.jar
INFO]
INFO] --- assembly:3.7.1:single (make-assembly) @ atlas-capstone ---
INFO] Building jar: C:\Users\spvar\OneDrive\Desktop\CapstoneAtlas\StudentCourseRegistrationSystem\target\atlas-capstone-1.0.0-jar-with-dependencies.jar
INFO]
INFO] --- failsafe:3.1.2:integration-test (default) @ atlas-capstone ---
INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
INFO]
INFO] T E S T S
INFO] -----
INFO] Running com.atlas.tests.EnrollmentFlowsIntegrationTest
INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.475 s -- in com.atlas.tests.EnrollmentFlowsIntegrationTest
INFO] Results:
INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
INFO]
INFO] --- failsafe:3.1.2:verify (default) @ atlas-capstone ---
INFO] BUILD SUCCESS
INFO] -----
INFO] Total time: 24.849 s
INFO] Finished at: 2025-09-21T12:27:04+05:30
INFO] -----
C:\Users\spvar\OneDrive\Desktop\CapstoneAtlas\StudentCourseRegistrationSystem>
```

7. Deployment

7.1 Docker

- Dockerfile builds a minimal runtime image containing the packaged JAR.
- Docker Compose orchestrates services (Jenkins + DynamoDB Local + App).

7.2 Jenkins CI/CD

- Declarative pipeline performs: Checkout → Unit Tests (surefire) → Integration Tests (failsafe) → Package (assembly) → Archive.
- Integrated with Docker for automated builds.

Jenkins

Atlas-AMZ-Capstone-SCRS-Docker-Pipeline

Status

Changes

Build Now

Configure

Delete Pipeline

Full Stage View

Open Blue Ocean

Rename

Pipeline Syntax

Credentials

Atlas-AMZ-Capstone-SCRS-Docker-Pipeline

Last Successful Artifacts

atlas-capstone-1.0.0-jar-with-dependencies.jar

12.81 MiB

view

Test Result Trend

Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	Checkout	Unit Tests	Integration Tests	Package	Declarative: Post Actions
Average stage times: (full run time: ~38s)	1s	214ms	1s	12s	6s	11s	124ms
<div>#27</div> <div>Sep 21 13:21</div> <div>7 commits</div>	2s	245ms	1s	25s	7s	14s	161ms
<div>#26</div> <div>Sep 21 13:13</div> <div>No Changes</div>	1s	180ms	1s	10s	7s	14s	129ms
<div>#25</div> <div>Sep 21</div> <div>No Changes</div>	1s	364ms	1s	11s	5s	11s	143ms

Builds

Filter

Today

#27 7:51 AM

#26 7:43 AM




#25 7:42 AM

Sep 20, 2025

#24 6:41 PM


Jenkins

/ Atlas-AMZ-Capstone-SCRS-Docker-Pipeline
/ #27






```

[WARNING] system modules path not set in conjunction with -source 17
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ atlas-capstone ---
[INFO] skip non existing resourceDirectory /var/jenkins_home/workspace/Atlas-AMZ-Capstone-SCRS-Docker-Pipeline/src/test/resources
[INFO]
[INFO] --- compiler:3.11.0:testCompile (default-testCompile) @ atlas-capstone ---
[INFO] Changes detected - recompiling the module! :dependency
[INFO] Compiling 5 source files with javac [debug target 17] to target/test-classes
[WARNING] system modules path not set in conjunction with -source 17
[INFO]
[INFO] --- surefire:3.1.2:test (default-test) @ atlas-capstone ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running com.atlas.tests.EnrollmentFlowsUnitTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.598 s -- in com.atlas.tests.EnrollmentFlowsUnitTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 21.337 s
[INFO] Finished at: 2025-09-21T07:52:09Z
[INFO] -----

Post stage
[Discipline] junit4

```


unlabeled-2000

🔍 Search

Ctrl+K

🔔

📧

⚙️

🔧

V

—

🗖

✕

🏠 Ask Gordon BETA

📦 Containers

🖼️ Images

💾 Volumes

🌐 Kubernetes

🔗 Builds

🔗 Models

🔗 MCP Toolkit BETA

🌐 Docker Hub

🔗 Docker Scout

🔗 Extensions

Builds

Give feedback

Selected builder

desktop-linux

Import builds

Builder settings

Build large and multi-platform Docker images faster in Docker Build Cloud.

To improve build speeds for you, your team, and even your CI, try [Docker Build Cloud now](#).

✕

Build history

Active builds

🔍 Search

📄

☰

🔵 Show only my builds

<input type="checkbox"/>	Name	ID	Builder	Duration	Created	Author	
<input type="checkbox"/>	✓ StudentCourseRegistrationSystem	g21f9s	📦 desktop-linux	3.0s	6 minutes ago	N/A	🗑️
<input type="checkbox"/>	✓ StudentCourseRegistrationSystem	69ppvw	📦 desktop-linux	2m 17s	20 minutes ago	N/A	🗑️
<input type="checkbox"/>	✓ StudentCourseRegistrationSystem	hkpetx	📦 desktop-linux	3.4s	13 hours ago	N/A	🗑️
<input type="checkbox"/>	✓ StudentCourseRegistrationSystem	wzta2x	📦 desktop-linux	2m 21s	14 hours ago	N/A	🗑️
<input type="checkbox"/>	○ spvar	srqao6	📦 desktop-linux	0.1s	14 hours ago	N/A	🗑️
<input type="checkbox"/>	○ target	rr0zdk	📦 desktop-linux	0.1s	2 days ago	N/A	🗑️
<input type="checkbox"/>	○ StudentCourseRegistrationSystem	24uey3	📦 desktop-linux	-2.1s	2 days ago	N/A	🗑️

Rows per page: 10 1-7 of 7

🔌 Engine running | RAM 3.37 GB CPU 63.19% Disk: 6.19 GB used (limit 1006.85 GB) | 📄 Terminal v4.46.0

Windows PowerShell

Windows PowerShell

+

⌵

PS C:\Users\spvar> cd "C:\Users\spvar\OneDrive\Desktop\CapstoneAtlas\StudentCourseRegistrationSystem"

PS C:\Users\spvar\OneDrive\Desktop\CapstoneAtlas\StudentCourseRegistrationSystem> docker build -t atlas-capstone:latest .

[+] Building 3.2s (16/16) FINISHED

=> [internal] load build definition from Dockerfile

=> => transferring dockerfile: 722B

=> [internal] load metadata for docker.io/library/eclipse-temurin:17-jdk-alpine

=> [internal] load metadata for docker.io/library/maven:3.9.9-eclipse-temurin-17

=> [auth] library/eclipse-temurin:pull token for registry-1.docker.io

=> [auth] library/maven:pull token for registry-1.docker.io

=> [internal] load .dockerignore

=> => transferring context: 2B

=> [build 1/5] FROM docker.io/library/maven:3.9.9-eclipse-temurin-17@sha256:f58d59b6273e785ac0a4477f6e9b5bald7731c75b906c0f7b34076f1851318cc

=> => resolve docker.io/library/maven:3.9.9-eclipse-temurin-17@sha256:f58d59b6273e785ac0a4477f6e9b5bald7731c75b906c0f7b34076f1851318cc

=> [internal] load build context

=> => transferring context: 2.87kB

=> [stage-1 1/3] FROM docker.io/library/eclipse-temurin:17-jdk-alpine@sha256:c0dfec8fb4aec4adad2d00d267c5cb5348465d77f087e455d2905fb180ce27d2

=> => resolve docker.io/library/eclipse-temurin:17-jdk-alpine@sha256:c0dfec8fb4aec4adad2d00d267c5cb5348465d77f087e455d2905fb180ce27d2

=> CACHED [stage-1 2/3] WORKDIR /app

=> CACHED [build 2/5] WORKDIR /app

=> CACHED [build 3/5] COPY pom.xml ./

=> CACHED [build 4/5] COPY src ./src

=> CACHED [build 5/5] RUN mvn -B -DskipTests package

=> CACHED [stage-1 3/3] COPY --from=build /app/target/*-jar-with-dependencies.jar app.jar

=> exporting to image

=> => exporting layers

=> => exporting manifest sha256:d0055ae0ba5aa2cf1dc3c7b2796b4c29720b940ec22e98bf90a1970c3faa3228

=> => exporting config sha256:c25411e7876385e9bdeaaa5d370827f7f5a8926e6473506ec1dc8ca208e9712

=> => exporting attestation manifest sha256:7800f29bd8f9921296a8c17f52ca95e691aead66fa420f54e36dd9940c31554e

=> => exporting manifest list sha256:a3c36bb1b1250c66040f27df63af09e5a2a9d7fbcf08d67b3a3c029e32d9f1505

=> => naming to docker.io/library/atlas-capstone:latest

=> => unpacking to docker.io/library/atlas-capstone:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/g21f9shwr086q4p0fn0ap630p

PS C:\Users\spvar\OneDrive\Desktop\CapstoneAtlas\StudentCourseRegistrationSystem> docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

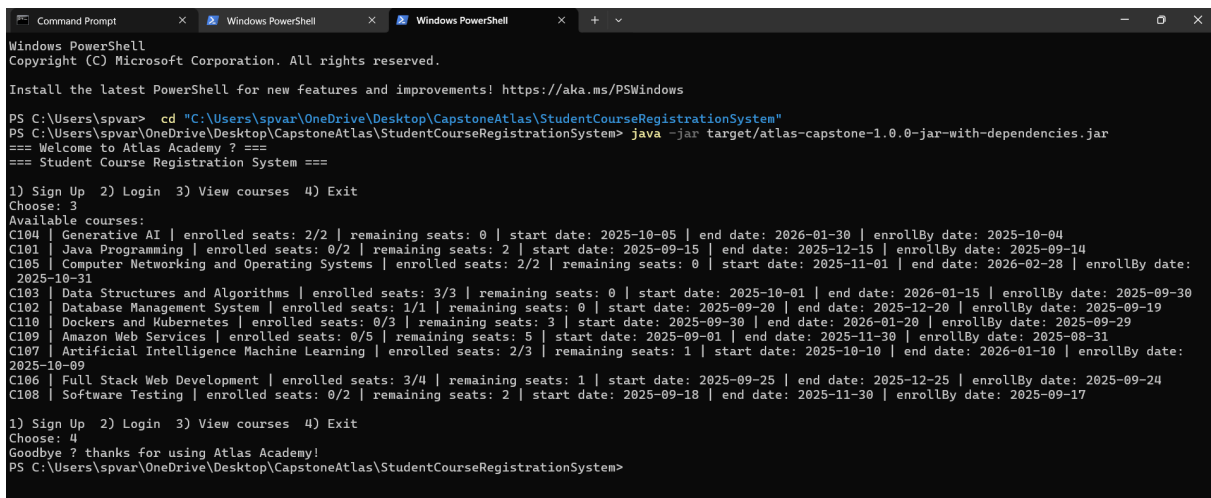
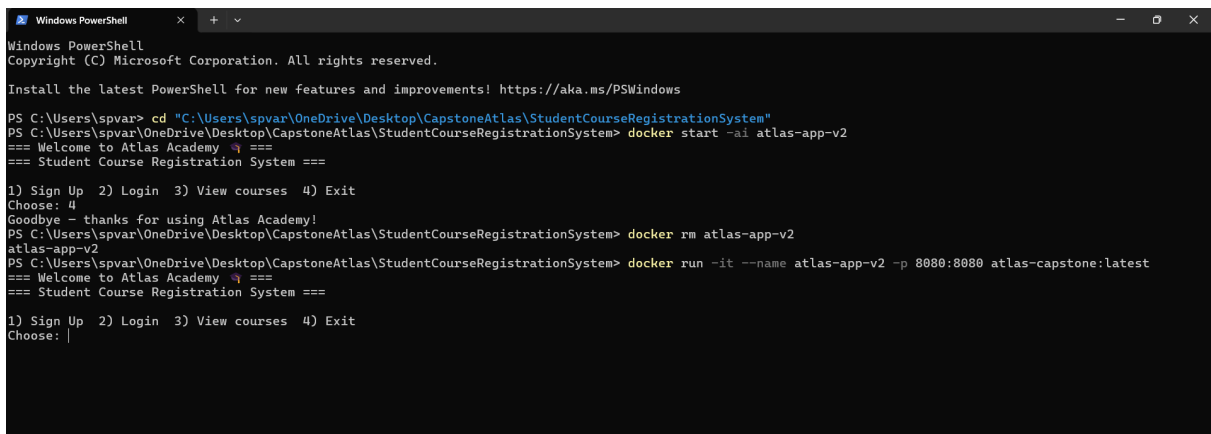
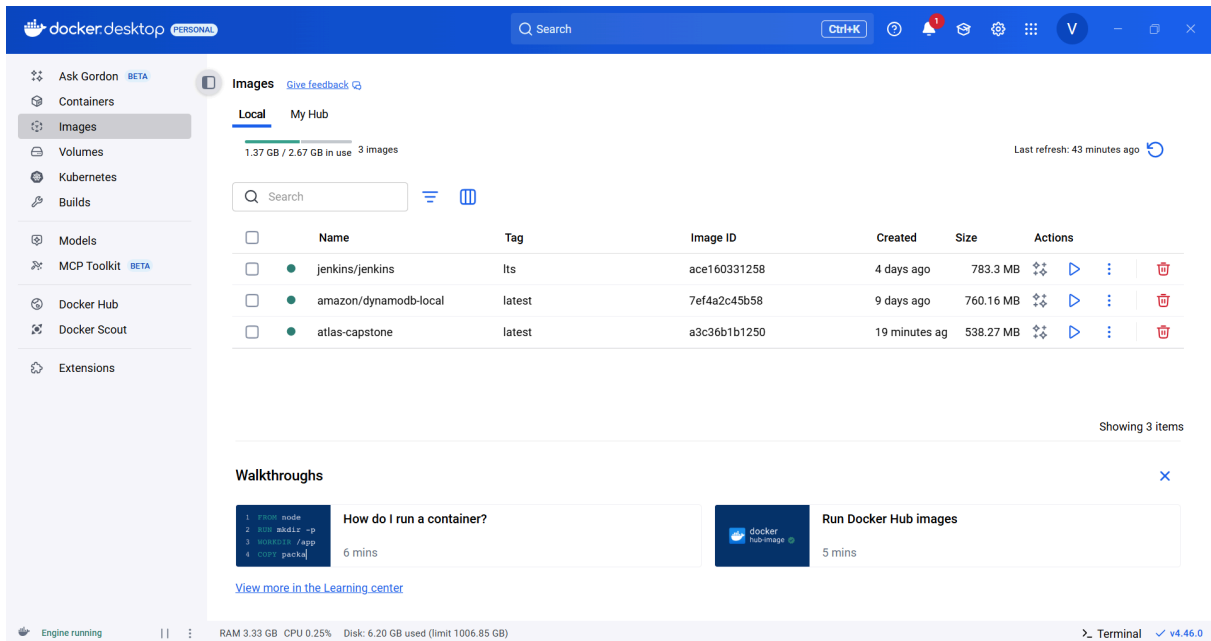
PS C:\Users\spvar\OneDrive\Desktop\CapstoneAtlas\StudentCourseRegistrationSystem> docker start -ai atlas-app-v2

=== Welcome to Atlas Academy ===

=== Student Course Registration System ===

1) Sign Up 2) Login 3) View courses 4) Exit

Choose: |



8. DynamoDB Local

The system uses three primary tables (DynamoDB Local):

Students Table

- **Primary Key:** id (S)
- **Attributes:**
 - id → Student ID
 - name (S)
 - email (S) → normalized (trim + lowercase)
 - passwordHash (S)
 - enrolledCourseIds (L) → list of courseId strings
 - waitlistedCourseIds (L)

Courses Table

- **Primary Key:** courseId (S)
- **Attributes:**
 - courseId (S)
 - courseName (S)
 - maxSeats (N)
 - currentEnrolledCount (N)
 - enrolledIds (L) → list of student IDs
 - waitlistIds (L) → list of student IDs (FIFO ordering)
 - startDate (S)
 - endDate (S)
 - latestEnrollmentBy (S)

StudentLogs Table

- **Primary Key:** logId (S)
- **Attributes:**
 - logId (S)
 - studentId (S)
 - action (S) → e.g., SIGNUP, LOGIN, ENROLL, DROP, WAITLIST_JOIN, AUTO_ENROLL
 - courseId (S, nullable)
 - timestamp (S) → ISO-8601 string

Important DynamoDB Details

- **Atomic seat reservation** is implemented via UpdateItem with ConditionExpression (e.g., currentEnrolledCount < :max).
 - This ensures no overbooking under concurrent attempts.
- **List management** → Implemented using list_append, ensuring attributes update gracefully.

Table Creation & Seeding

The system uses DynamoDB Local for persistence. Tables were created using AWS CLI commands:

```
aws dynamodb create-table \  
  --table-name (tablename) \  
  --attribute-definitions AttributeName=id,AttributeType=S \  
  --key-schema AttributeName=id,KeyType=HASH \  
  --billing-mode PAY_PER_REQUEST \  
  --endpoint-url http://localhost:8000 \  
  --region ap-south-1
```

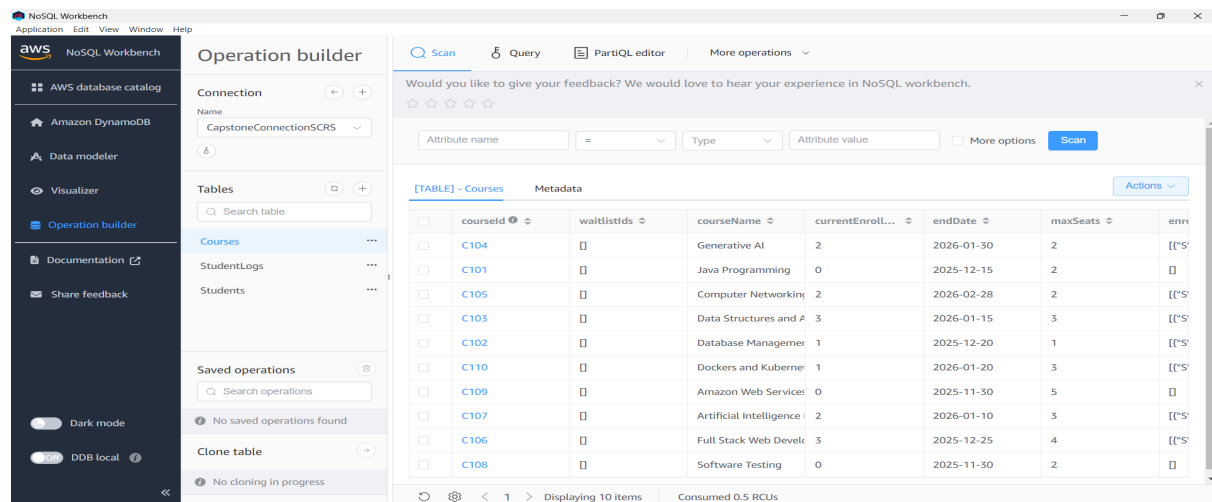
Sample Data Seeding

Courses were seeded into the Courses table using put-item commands:

```
aws dynamodb put-item \  
  --table-name Courses \  
  --item '{  
    "courseId": {"S": "C101"},  
    "courseName": {"S": "Java Programming"},  
    "maxSeats": {"N": "2"},  
    "currentEnrolledCount": {"N": "0"},  
    "startDate": {"S": "2025-09-15"},  
    "endDate": {"S": "2025-12-15"},  
    "latestEnrollmentBy": {"S": "2025-09-14"},  
    "enrolledIds": {"L": []},  
    "waitlistIds": {"L": []}  
  }' \  
  --endpoint-url http://localhost:8000 \  
  --region ap-south-1
```

This process was repeated for multiple courses (C102 – C110). (seed-courses.json and setup-dynamodb.ps1 have been included under resources, and can be used to pre-seed course data.)

The **DynamoDB Workbench** tool was also used for schema visualization and validation.



9. Component Descriptions (Classes & Services)

Below is a concise description of key classes and responsibilities:

Model Classes

- **Student** → Holds identity, email, password hash, and lists of enrolled/waitlisted course IDs.
- **Course** → Course attributes and lists for enrolled/waitlisted students.
- **StudentLog** → Audit representation.

DAO Interfaces

- **StudentDao** → save, getById, findByEmail, listAll, enrollStudentAtomic
- **CourseDao** → getById, listAll, addToWaitlist, replaceWaitlist, replaceEnrolled
- **LogDao** → append method to write logs.

Repository Implementations

- **DynamoStudentDao, DynamoCourseDao, DynamoLogDao** → Implement the above interfaces using AWS SDK v2 DynamoDB models and attribute conversions.

Test Utils

- **InMemoryStudentDao, InMemoryCourseDao, InMemoryLogDao** → Used for unit tests.

Service Layer

- **StudentService** → Registration, login, profile retrieval. Handles password hashing and session creation.
- **EnrollmentService** → Core business logic for enroll, drop, and waitlist promotion. Handles rule enforcement (max enrollments, waitlist caps, deadlines).
- **CourseService** → Thin wrapper to fetch/list courses.

Session Management

- **SessionStore** → In-memory token store with TTL (30 minutes). Stores expiry and student ID.
 - **SessionInfo** → Session details structure.
-

10. Use Cases & Sequence Flows

10.1 Sign Up

- User supplies (id, name, email, password) to service.
- Service validates uniqueness via:
 - StudentDao.getById
 - StudentDao.findByEmail
- Password is hashed, student object saved, SIGNUP logged.

10.2 Login

- User submits (email, password).
- Service finds student by email.
- Verifies password against stored bcrypt hash.
- Creates session token.
- Logs LOGIN.

10.3 Enroll

- Validate student and course existence.
- Check latestEnrollmentBy and student limits.
- Attempt: CourseDao.enrollStudentAtomic(courseId, studentId, maxSeats)
 - If true → Add to student.enrolledCourseIds, log ENROLL.
 - If false → Add to waitlist (if under limit), log WAITLIST_JOIN.

10.4 Drop

- Validate dropping window (endDate).
- If enrolled:
 - Remove from course.enrolledIds.
 - Decrement count via CourseDao.replaceEnrolled.

- o Save student.
 - o Log DROP.
 - o Call promoteFromWaitlist.
- If waitlisted:
 - o Remove from waitlist via CourseDao.replaceWaitlist.
 - o Log WAITLIST_OPT_OUT.

10.5 Waitlist Promotion

- After a seat opens, iterate waitlist FIFO.
- For each candidate:
 - o Skip if they already have 5 active enrollments (keep them in the list).
 - o Otherwise, attempt a conditional enroll.
- On success:
 - o Remove candidate from waitlist.
 - o Update student record.
 - o Log AUTO_ENROLL.
 - o Break (only one promotion per seat).

11. Results

- **Full implementation** of all Functional Requirements (FR-01 → FR-12) and Non-Functional Requirements (NFR-01 → NFR-07).
- **Robust atomic seat allocation** achieved using DynamoDB conditional updates to prevent race conditions.
- **Complete test suite** implemented:
 - o Unit tests (with in-memory DAOs).
 - o Integration tests (with DynamoDB Local).
 - o All tests passed successfully in both local runs and Jenkins pipeline.
- **CI/CD Pipeline:**
 - o Achieved seamless pipeline configuration in Jenkins.
 - o Integrated Dockerized build and test steps.
 - o Artifacts archived for reproducibility.
- **Web UI Extension:**
 - o Successfully added a Spark-based web UI.
 - o Achieved without modifying the core service layer, demonstrating modular and maintainable architecture.
- **DynamoDB Validation:**
 - o Validated workflows using DynamoDB Workbench and Local integration.
- **Evidence:**
 - o Logs and screenshots provided as proof of working pipeline, containers, and UI functionality.

12. Conclusion

12.1 Achievements

- Delivered a scalable, modular system following SOLID principles.
- Implemented session-based authentication with password hashing.
- Achieved full CI/CD pipeline integration with Jenkins and Docker.
- Extended the system with Spark Java UI without modifying core services.
- Successfully validated with unit and integration tests.

12.2 Future Enhancements

Below are recommended improvements should this project progress to a production-grade system:

- Add Admin Module for managing courses dynamically.
 - Extend WebApp UI with better styling and JavaScript interactivity.
 - Integrate email/SMS notifications for enrollment and waitlist changes.
 - Provide analytics dashboards (course popularity, enrollment stats).
 - Implement role-based access control (RBAC).
-

13. References

- AWS DynamoDB Documentation – <https://docs.aws.amazon.com/dynamodb>
 - Spark Java Framework – <http://sparkjava.com/>
 - JUnit 5 User Guide – <https://junit.org/junit5/docs/current/user-guide/>
 - BCrypt Library – <https://github.com/patrickfav/bcrypt>
 - Docker Documentation – <https://docs.docker.com/>
 - Jenkins Documentation – <https://www.jenkins.io/doc/>
 - SOLID Principles Overview – <https://en.wikipedia.org/wiki/SOLID>
-

Appendix

Appendix A — Functional Requirements (FR) and Acceptance Criteria

(Condensed from original Requirements Analysis document.)

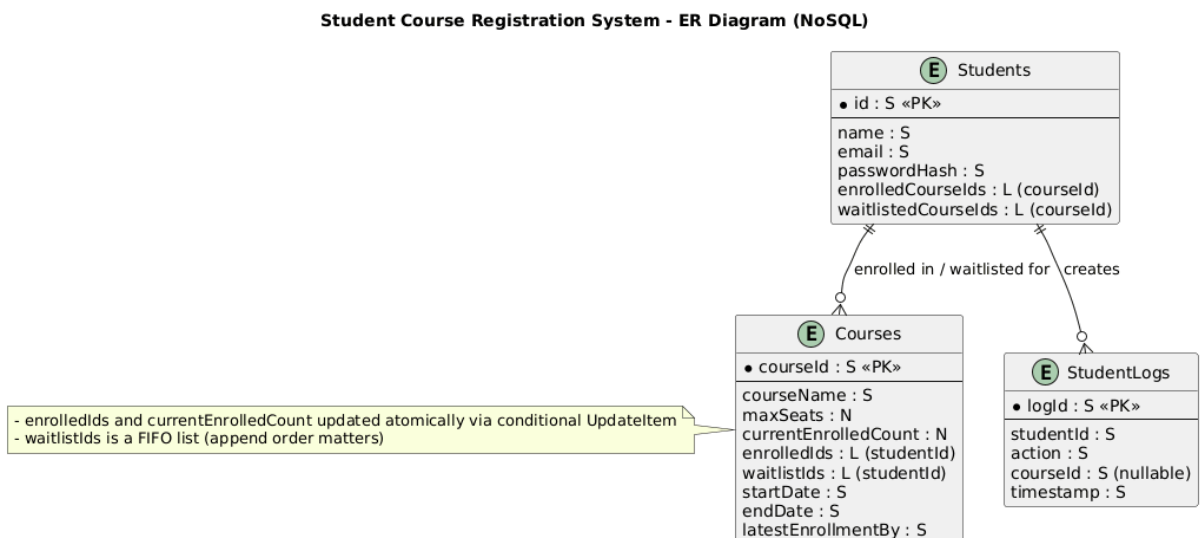
- **FR-01 — Student Sign Up:** Unique ID & email, hashed password.
 - *Acceptance:* Student row stored with hashed password.
- **FR-02 — Student Login:** Return session token and write LOGIN log.
 - *Acceptance:* Login returns token and log entry exists.
- **FR-03 — View Profile:** Show enrollment statuses.
 - *Acceptance:* Correct statuses and waitlist positions.
- **FR-04 — Preloaded Courses:** Seeded via JSON.
 - *Acceptance:* CourseService.list() returns seeded data.
- **FR-05 — View Courses:** Show remaining seats.
 - *Acceptance:* Correct remaining seats calculation.
- **FR-06 — Enroll:** Atomic enroll or waitlist.
 - *Acceptance:* Atomic update on success; waitlist append on full.
- **FR-07 — Drop:** Remove enrollment or waitlist; promote candidate if seat opens.
 - *Acceptance:* Seat freed and AUTO_ENROLL recorded.
- **FR-08 — Waitlist Management:** FIFO with skipped candidates preserved.
 - *Acceptance:* Correct promotion logic.
- **FR-09 — Audit Logging:** All actions write StudentLogs entries.
- **FR-10 — Prevent Duplicates & Limits:** No duplicates; max 5 enrollments and 3 waitlists.
- **FR-11 — Session & Minimal Auth:** In-memory token TTL of 30 minutes.
- **FR-12 — Tests:** Unit and integration tests for core flows.

Appendix B — Non-Functional Requirements (NFR)

(Condensed from original Requirements Analysis document.)

- **Usability:** Clear console messages and minimal web UI.
- **Maintainability:** Layered architecture and dependency injection via constructors.
- **Security:** BCrypt hashed passwords and session TTLs.
- **Logging:** Append-only StudentLogs in DB.
- **Testability:** Unit and integration tests with in-memory DAOs and DynamoDB Local.
- **Performance:** Conditional updates prevent overbooking with near-constant-time seat checks.
- **Offline Operation:** Fully functional with local DynamoDB.

Class diagram:



Use-case diagram:

