Cloud Service Provider: AWS

Justification (cost, scalability, services):

- **Scalability & reliability:** Easy horizontal scaling (Auto Scaling), managed databases (RDS), and multi-AZ design for high availability.
- **Service breadth:** Native building blocks for web apps (EC2/ALB), storage (S3), databases (RDS), networking (VPC), monitoring (CloudWatch).
- **Cost control:** Pay-as-you-go, ability to right-size instances and scale only when needed; free tier helpful for learning/prototyping.

Core AWS Services Used:

- **VPC** (network isolation, subnets, routing)
- **EC2** (Node/Express app servers)
- **Application Load Balancer (ALB)** (public entry point, distributes traffic)
- **RDS (MySQL)** (managed relational database)
- **S3** (static assets/file uploads/backups)
- **IAM** (least-privilege access control)
- **Security Groups + NACLs** (traffic control)
- **CloudWatch** (logs/metrics/alarms)

Application Design

Programming language: JavaScript
Runtime: Node.js
API style: RESTful JSON API
Middleware: Express.js
Frontend framework: React *or "None" if app is API-only — pick one*

High-level components:

- **Client (React SPA):** Runs in browser; calls backend REST API.
- **Backend (Node/Express):** Implements business logic, validation, authentication stubs (out-of-scope to implement), interacts with databases.
- **Database (MySQL on RDS):** Stores normalized transactional data.
- **Object storage (S3):** Stores uploaded files/images/attachments; optional for static website hosting.

Port configuration:

- **Public:** 443 (HTTPS) to ALB; redirect 80 → 443
- **App servers:** 3000 (Node/Express) internal from ALB only
- **Database:** 3306 (MySQL) from app servers only
- **Admin/troubleshooting:**
  - Linux admin: **22 (SSH)** from a restricted admin IP range only
  - Ping (ICMP): allow only from within VPC or admin subnet (for connectivity testing)

Operating System and Virtual Servers

Operating System: Linux (Amazon Linux 2023)
Why Linux: Lower cost, stable, widely supported for Node.js, strong security patching and automation.

Server configuration (baseline):

- **EC2 instance type:** t3.small
- **vCPU / RAM (t3.small):** 2 vCPU / 2 GB RAM
- **Storage:** 30 GB gp3 EBS (encrypted) per instance
- **Scaling:** Auto Scaling Group with min 2 instances across 2 AZs (for high availability)

Load Balancer:

- **ALB** in public subnets across 2 AZs.
- Health checks against /health endpoint.

Database Design (MySQL, 3NF)

DBMS: MySQL on Amazon RDS
Why MySQL/RDS: Managed backups, patching, encryption, monitoring, and multi-AZ failover options.

RDS configuration (baseline):

- **Instance type:** db.t3.micro *(class scale)* or db.t3.small
- **Storage:** 20–50 GB gp3, auto-scaling enabled if possible
- **High availability:** Multi-AZ enabled
- **Backups:** Automated daily backups (7–14 day retention)

4.1 ER Diagram + Schema (template you can adapt)

Since the prompt doesn't specify your app domain, here's a **clean 3NF "Project/Task Manager" schema** that fits most class apps. If your app is different (ex: inventory, booking, student app), keep the same normalization logic.

Entities (3NF):

- **Users** (user accounts)
- **Roles** (role definitions)

- **UserRoles** (many-to-many)
- **Projects**
- **Tasks**
- **TaskStatus** (lookup table)
- **Comments** (task comments)

Tables (example fields):

1. **Users**
   - user_id (PK)
   - email (UNIQUE)
   - display_name
   - created_at
2. **Roles**
   - role_id (PK)
   - role_name (UNIQUE) *(e.g., Student, Admin)*
3. **UserRoles**
   - user_id (FK → Users.user_id)
   - role_id (FK → Roles.role_id)
   - **PK (user_id, role_id)**
4. **Projects**
   - project_id (PK)
   - owner_user_id (FK → Users.user_id)
   - project_name
   - created_at
5. **TaskStatus** *(lookup)*
   - status_id (PK)
   - status_name (UNIQUE) *(ToDo, InProgress, Done)*

6. **Tasks**
    - task_id (PK)
    - project_id (FK → Projects.project_id)
    - assigned_user_id (FK → Users.user_id, nullable)
    - status_id (FK → TaskStatus.status_id)
    - title
    - description
    - created_at
    - due_date (nullable)
7. **Comments**
    - comment_id (PK)
    - task_id (FK → Tasks.task_id)
    - author_user_id (FK → Users.user_id)
    - body
    - created_at

3NF notes:

- Repeating/derived attributes are avoided (status names live in TaskStatus).
- Many-to-many handled via junction table (UserRoles).
- Non-key attributes depend only on the key.

(Draw the ER diagram from the above.)

Network Architecture and Security

Goal: high availability+least privilege.

5.1 VPC + Subnets (redundant)

- **One VPC** (e.g., 10.0.0.0/16)
- **Two Availability Zones** (AZ-a, AZ-b)
- **Public subnets (2):** ALB+(optional) NAT gateway / bastion
- **Private app subnets (2):** EC2 app instances
- **Private DB subnets (2):** RDS MySQL (subnet group spanning 2 AZs)

Routing:

- Public subnets route to **Internet Gateway**
- Private subnets route outbound internet via **NAT Gateway**
- DB subnets: no direct internet route

Firewalls:

- Security Groups (primary)
- Network ACLs (secondary)

5.2 Security Groups (ingress/egress rules)

SG-ALB (Load Balancer)

- Ingress:
  - TCP 443 from 0.0.0.0/0
  - TCP 80 from 0.0.0.0/0 (redirect to 443)
- Egress:
  - TCP 3000 to SG-App

SG-App (EC2 App Servers)

- Ingress:
  - TCP 3000 from SG-ALB

    - TCP 22 from Admin-IP (only)
    - ICMP from VPC CIDR
- Egress:
    - TCP 3306 to SG-DB
    - HTTPS (443) outbound for updates/log shipping if needed

SG-DB (RDS MySQL)

- Ingress:
    - TCP 3306 from SG-App only
- Egress:
    - Default outbound

Data Visualization Tool Standard

Selected Tool: Tableau

Justification:

- **Functionality & features:** Strong interactive dashboards (filters, parameters, drill-downs), excellent visual exploration, and polished stakeholder-ready reporting.
- **AI & data integration:** Connects well to relational databases (including MySQL) and supports calculated fields, forecasting, clustering, and "Explain Data" style insight features to speed analysis.
- **Team familiarity & ease of use:** Common in analytics courses and widely used in industry; quick to build dashboards once data connections are set.
- **Cost & scalability:** Tableau offers academic access options; supports scaling from a single workbook to published dashboards via Tableau Server/Cloud if needed.
- **Other criteria:** Works well for rapid prototyping—your team can stand up dashboards early and refine as sprints deliver more data.

How it connects:

- Primary: Tableau connects directly to **RDS MySQL** (live connection or extract).
- Optional: Curated extracts or flat files can be staged in **S3** for repeatable refresh workflows.

Testing & Quality Assurance During Sprints

Unit testing

- Backend: **Jest** + **supertest** (API route tests)
- Goal: validate functions, controllers, services independently.
- Gate: tests must pass in CI before merge.

Integration testing

- Validate modules working together:
  - API ↔ DB (use test database schema)
  - Example: create task → verify record inserted → verify retrieval.
- Run in CI using dockerized MySQL or a dedicated RDS test instance.

End-to-end (E2E) testing

- If React frontend: **Cypress** for user flows:
  - login (if stubbed, skip), create project, create task, update status.
- Goal: verify the full stack behaves correctly.

Quality gates

- Linting: ESLint
- Security scanning: npm audit (basic dependency scan)
- Code review: PR required, at least 1 approval

Authentication & Authorization

Access model: System is open to anyone for build/testing as required by assignment.
Roles: Define roles in app logic and DB (example: Student, Administrator), but **auth implementation is out-of-scope**

Team Responsibilities + Contribution Summary

Cait: Application Developer- Drafted core system structure concepts, handled majority of document drafting + GitHub coordination

Simon: Cloud Architect- Created Architecture Document, selected platforms/tools, provided system structure guidance

Muhammad: Database Architec- Developed ERD and structured data relationships

Saad: Project Manager- Final review, document consistency, formatting, submission verification

Collaboration challenges + resolution:

- Challenge: aligning on a single architecture while members worked in parallel.
- Resolution: created a shared outline + naming conventions early, then merged sections via PR review to avoid conflicting decisions.

GitHub Proof

- Add a screenshot showing the architecture document committed to your repo:
    - The file in the repository view **AND** the commit history showing it was added/updated.