Appendix 3: Educational Learning Outcomes — TastyTable Backend Project

ASP.NET Core 8 • EF Core (MySQL) • JWT • Docker • AWS Elastic Beanstalk + RDS

# 1. Overview

* This appendix documents educational learning outcomes demonstrated through the TastyTable project.
* TastyTable is a monolithic ASP.NET Core 8 Web API using Entity Framework Core (Pomelo.MySql) and JWT authentication.
* The project was developed and deployed from Ubuntu/VS Code, containerized with Docker, and hosted on AWS Elastic Beanstalk with an RDS MySQL database.
* This document connects real implementation steps, issues encountered, and resolutions to each Knowledge, Skills, and General Competence outcome.

# 2. Project Execution: From Start to Fixes

## 2.1 Initial Setup

* Created a .NET 8 solution with projects: TastyTable.Api, TastyTable.Core, TastyTable.Data, TastyTable.Services, TastyTable.Tests.
* Added EF Core with Pomelo MySQL and created migrations for Users, MenuItems, Orders, OrderItems.
* Implemented JWT auth (JwtBearer) and Swagger with an Authorize button; form-based login/register in Swagger (no raw JSON needed).
* Wired Controllers → Services → Repositories with Dependency Injection.
* Seeded demo data (Admin user + menu items) via ctx.Database.Migrate() + DemoDataSeeder.

## 2.2 Common Errors Faced and How I Solved Them

### 401 Unauthorized on protected endpoints

* Cause: Missing 'Authorization: Bearer <token>' header or incorrect Swagger security configuration.
* Fix: Added Swagger AddSecurityDefinition('Bearer', ...) and used OperationFilter to show locks only on [Authorize] endpoints.
* Fix: In cURL and Postman, included 'Authorization: Bearer <JWT>' (note the word 'Bearer' + space).

### JWT sent without 'Bearer' prefix

* Cause: Copying the raw token only.
* Fix: Updated Swagger description and examples; cURL examples include the full header with 'Bearer ' prefix.

### Object cycle JSON error when fetching Orders

* Cause: EF navigation properties caused cycles during serialization.
* Fix: In Program.cs, added AddJsonOptions with ReferenceHandler.IgnoreCycles to prevent infinite loops.

### Swagger UI did not show Authorize input

* Cause: Missing security definition or global requirement masking unsecured endpoints.
* Fix: Added AddSecurityDefinition('Bearer', ...) and OperationFilter<AuthorizeCheckOperationFilter>() only; avoided a global SecurityRequirement so only [Authorize] endpoints show lock icons.

### Elastic Beanstalk deploy failed: 'Both Dockerfile and Dockerrun.aws.json are missing'

* Cause: Dockerfile was inside TastyTable.Api; EB expects Dockerfile at the ZIP root.
* Fix: Moved Dockerfile to the repository root and zipped again (excluding bin/obj).

### 502 Bad Gateway on Elastic Beanstalk

* Cause: App listened on port 8080 but EB expects port 80 in container.
* Fix: In Dockerfile set EXPOSE 80 and ASPNETCORE\_URLS=http://+:80, then redeployed.

### EB health shows Warning + 404 on '/'

* Cause: No root endpoint and Swagger disabled in Production.
* Fix: Enabled Swagger in all environments, mapped '/' → redirect to /swagger, added '/healthz' and set Health Check URL in EB to /healthz.

### RDS connectivity issues (timeout)

* Cause: RDS is private; Security Group did not allow inbound 3306 from EB's instance SG.
* Fix: Allowed MySQL (3306) inbound on RDS SG from the EB instance SG; kept DB non-public.

# 3. Detailed Code Explanation

## Program.cs

What it does:

* Registers Controllers with JSON options: IgnoreCycles to avoid object loops; ignore nulls for clean responses.
* Adds Swagger with JWT Bearer security and custom filters (AuthorizeCheckOperationFilter, EmptyStringSchemaFilter).
* Configures DbContext with MySQL using ServerVersion.AutoDetect(connection).
* Registers JwtTokenService, UserService, MenuService, OrderService into DI container.
* Configures JwtBearer with issuer signing key, RoleClaimType, NameClaimType, and zero ClockSkew.
* Applies migrations and seeding inside a try/catch, so EB startup does not crash if RDS is slow.
* Maps '/' → Swagger redirect and '/healthz' endpoint returning 'OK'.
* UseAuthentication + UseAuthorization + MapControllers finalize the pipeline.

How to test:

* Open /swagger; verify the Authorize button (lock icon on protected endpoints only).
* Try GET /api/menu anonymously (should work).
* Try POST /api/menu with and without Admin token (should require Admin).
* Hit /healthz to confirm EB health.

## AuthController.cs

What it does:

* POST /api/auth/register: accepts username, email, password via form inputs; creates user with hashed password and default role 'User'.
* POST /api/auth/login: verifies credentials; returns JWT with claims (sub = userId, unique\_name = username, role = role).

How to test:

* Register a user in Swagger (form inputs, no JSON). Then login and copy the token.
* Use Authorize (paste 'Bearer <token>') and call protected endpoints.

## MenuController.cs

What it does:

* GET /api/menu: lists items; supports details GET /api/menu/{id}.
* POST /api/menu: Admin only; creates item.
* PATCH /api/menu/{id}/availability?isAvailable=true|false: Admin only; toggles availability.

How to test:

* Test GET anonymously; test POST/PATCH after Authorize as Admin.

## OrdersController.cs

What it does:

* POST /api/orders: creates order for current user; calculates totals from MenuItems.
* GET /api/orders: lists orders for current user (secured).
* GET /api/orders/{id}: shows order with its items.

How to test:

* After login, POST a new order with menuItemId and quantity.
* Verify totals and list your orders.

## Services

What it does:

* JwtTokenService: builds JWT with symmetric key from configuration; sets role and subject claims.
* UserService: registration, password hashing (BCrypt), user retrieval by username; validation in login.
* MenuService: CRUD + availability toggle; uses repository/DbContext.
* OrderService: calculates totals, persists Order + OrderItems; fetches orders for user.

How to test:

* Unit tests validate hashing, totals, and service logic.

## Data Layer

What it does:

* AppDbContext: DbSet<User>, DbSet<MenuItem>, DbSet<Order>, DbSet<OrderItem>.
* Migrations: InitialCreate sets up tables and unique constraints on Username and Email.
* Repository: wraps common CRUD operations (optional).

How to test:

* Ensure connection string is correct; run migrations via startup; confirm tables in RDS/MySQL.

# 4. API Testing Guide (Swagger, Postman, cURL)

* Swagger: open /swagger → click Authorize → paste 'Bearer <token>' → test endpoints.
* Postman: create a collection; add environment variable TOKEN; set Authorization: Bearer {{TOKEN}}.
* cURL examples:
* Register: curl -X POST http://<host>/api/auth/register -F username=u1 -F email=u1@x.com -F password=P@ssw0rd
* Login: curl -X POST http://<host>/api/auth/login -F username=u1 -F password=P@ssw0rd
* Create Menu (Admin): curl -X POST http://<host>/api/menu -H 'Authorization: Bearer <token>' -H 'Content-Type: application/json' -d '{"name":"Pizza","description":"Cheese","price":1200,"isAvailable":true}'
* Create Order: curl -X POST http://<host>/api/orders -H 'Authorization: Bearer <token>' -H 'Content-Type: application/json' -d '{"items":[{"menuItemId":1,"quantity":2}]}'

# 5. Deployment to AWS (Elastic Beanstalk + RDS)

* Create RDS MySQL (private). Note endpoint, username, password.
* Create EB environment (Docker). Ensure Dockerfile at ZIP root; app listens on port 80.
* Set environment variables: ConnectionStrings\_\_DefaultConnection, Jwt\_\_Secret, ASPNETCORE\_ENVIRONMENT=Production.
* Security Groups: allow inbound 3306 on RDS SG from EB EC2 SG only.
* Health: add '/healthz' endpoint and configure EB Health Check URL to '/healthz'.

# 6. Troubleshooting Journal

* 404 on '/': enable Swagger in prod and redirect '/' → '/swagger'.
* 502 on EB: listen on port 80; set ASPNETCORE\_URLS=http://+:80.
* Dockerfile missing: move Dockerfile to ZIP root while zipping the solution.
* 401: ensure 'Authorization: Bearer <token>' header and proper Swagger security definition.
* JSON cycle: add ReferenceHandler.IgnoreCycles in AddJsonOptions.
* RDS unreachable: fix SG to allow EB instance SG on 3306.

# 7. Educational Learning Outcomes Mapping

## 7.1 Knowledge

1. Principles & tools of backend programming

Used DI, REST, EF Core, JWT, Swagger, Docker, and AWS; applied clean layering and configuration.

1. Evaluate work vs quality standards

Unit tests, unique constraints, role-based authorization, EB health checks and logs.

1. Familiarity with industry contexts

Design aligns with common patterns in product companies and IT departments.

1. Update knowledge via trends/tools & professional contact

Resolved deployment/security issues by reading docs and analyzing logs; tuned Swagger UX.

1. History/tradition/role in society

Adopted modern practices (stateless auth, containers, cloud deployment).

1. Insight into development opportunities

Skills apply broadly to F&B, e-commerce, and enterprise integrations.

## 7.2 Skills

1. Explain tool/language choices

.NET 8 performance & tooling; EF Core Pomelo for MySQL; JWT for stateless auth; Docker for parity; AWS EB for managed deploy.

1. Reflect and adjust under guidance

Converted login/register to form-data; fixed Swagger lock display; aligned ports and health checks for EB.

1. Find and assess professional material

Used AWS/EF/ASP.NET docs and logs to target fixes efficiently.

1. Map status and identify change needs

Monitored EB logs, Nginx access/error logs, and container logs; iterated Program.cs/Dockerfile/SGs.

## 7.3 General Competence

1. Plan and carry out tasks within laws/ethics

Hashed passwords (BCrypt), enforced roles, private DB networking, plan for HTTPS/HSTS.

1. Perform work per sector needs

Menu & order flows ready for web/mobile clients; clean API surface.

1. Build relationships across roles/sectors

APIs and docs support collaboration with frontend, DevOps, DBAs.

1. Exchange views and participate in solution development

Documented decisions and trade-offs; aligned on best practices.

1. Contribute to organizational development

Added tests, automation, and security improvements; proposed CI/CD and observability.

# 8. Next Steps & Continuous Learning

* Add CI/CD: GitHub Actions → ECR → EB.
* Centralized logging/metrics (CloudWatch).
* Validation improvements (FluentValidation).
* Integration tests with Testcontainers for MySQL.
* HTTPS with ACM + ALB; enforce HSTS.

9. S**crum Collaboration — TastyTable Backend Project**

* Our team of four used a flexible Scrum-inspired workflow to collaborate effectively on the TastyTable backend project. We held **one in-person meeting per week** to plan, coordinate, and review progress. During this meeting, we discussed priorities, assigned tasks, and aligned on technical decisions such as JWT authentication, EF Core data modeling, and deployment strategies.
* For the rest of the week, each team member worked **independently** on their assigned tasks. We maintained communication through Git commit messages and documentation, ensuring transparency of work and ease of integration. Feature branches and pull requests were used to manage code changes, while peer reviews ensured quality and consistency across the project.
* **Roles and Responsibilities**
* **Product Owner:** Managed the backlog, prioritized features, and clarified requirements during weekly meetings.
* **Scrum Master:** Facilitated weekly sessions, tracked progress, and helped resolve blockers identified during individual work.
* **Development Team:** Each member focused on backend development tasks, including authentication, API endpoints, service logic, and database integration. Members independently implemented their tasks but collaborated during the weekly meeting to resolve dependencies.
* **Handling Issues & Adaptation**  
  Challenges such as JWT misconfigurations, Swagger security display issues, and AWS Elastic Beanstalk deployment errors were discussed during our weekly meetings. Solutions were planned collaboratively, then implemented individually, allowing each member to learn through hands-on problem-solving.
* **Outcome & Reflection**  
  This hybrid workflow—weekly meetings combined with independent daily work—allowed us to balance coordination with individual productivity. It promoted accountability, enabled efficient task completion, and enhanced our practical skills in backend development, deployment, and collaborative software engineering.