Intro:

Hi, I'm Keon Abbott. This is my code review for CS 499, where I'll be walking through three projects—each tied to one of the major computer science categories: software engineering and design, algorithms and data structures, and databases. I'll show you what the code does, what I think could be better, and how I plan to level it up for the final ePortfolio. And hey—maybe you'll learn a sign or two along the way.

Part 1

This C++ project is all about exception handling. I kept it simple to focus on how to catch errors without crashing the program.

(Point out CustomException class) Here's a custom exception class I wrote. It lets me define my own error messages instead of using the default ones. This is super useful when you want more control over how your program communicates errors.

(Point out do_even_more_custom_application_logic()) This function throws a runtime error to simulate what happens when something goes wrong deeper in the logic. I built this part to see how the app handles unexpected situations.

(Point out do_custom_application_logic()) Here, I use a try-catch block to catch the error and keep the app from crashing. It logs the error message and keeps going, which is the kind of stability I want in anything user-facing.

Right now, everything is kind of crammed in there in one spot. The logic, error handling, and output are all together. It works, but it could be cleaner and easier to manage. I plan to refactor this by breaking it into smaller parts. I'll move logging into its own class and clean up the structure using better separation of concerns. I'll also write some unit tests to check how the app handles different kinds of exceptions. These changes show how I apply secure, professional design to make code more modular, and maintainable. It aligns with the course outcomes like creating reliable and real-world-ready systems.

Part 2

This project is a Q-learning agent that tries to find treasure in a grid maze. It trains the agent using reinforcement learning. I wrote this function to handle the training loop which includes decision-making, memory, and model updates.

(Point out random < epsilon block) This is where the agent decides to either explore or exploit. I used epsilon to control that balance. The lower the epilson, the more the agent relies on what it has learned at first, it explores a lot—but that eventually changes over time.

(Point out decay epsilon block with win rate) I added a basic form of epsilon decay here. If the agent wins consistently, it switches to exploitation by reducing the exploration rate. It is a way for the agent to improve decision-making as learning improves.

(Point out win with history tracking) Here, I also track win rate and print out progress during training. This helps me monitor how the agent is improving and whether I should continue or stop early.

Right now, the function already includes exploration/exploitation logic, basic performance tracking, and memory through experience relay. I wanted to take it further by visualizing the learning process. Right now, the win rate is printed to the console, I planned to create plots using matplotlib to show reward trends and episode lengths over time. This will make the results easier to explain and debug. Let me scroll down to show you the result of this maze when the agent is trained. As you can see in the end, the agent found its treasure—pretty cool right? This shows that I can build and improve a learning algorithm using state management, reward tracking, and neural networks. It aligns with course outcomes around evaluating algorithmic performance and using tools to solve problems.

Part 3

(Point out app.js models/travlr and apiRouter) This project is a full-stack travel app. I built the backend with Node.js and MongoDB using Mongoose. It connects to an API and lets users search for trips by code or destination.

(Point out Trip.find()) Right now, the search query is case-sensitive, so if the user types its differently, it might not return results. There's also no indexing, so as the database grows, performance could take a hit.

Right now, it works, but it's not user-friendly or optimized. There's no input validation or sanitization, so that's a risk. The search behavior could be much more flexible. I plan to use RegEx with the 'i' flag for case-insensitive words/searches. I'll also add indexes to speed things up and middleware to clean and validate input before it hits the database. This work aligns with outcomes around security, performance, and real-world backend development. It shows how I can improve functionality and being consistent.

Ending:

These three projects show how I approach building smarter, safer, and more thoughtful software. I’m not just checking off boxes—I'm applying everything I’ve learned to improve real projects. Thanks for watching and again, maybe you have picked up a sign or two along the way.