Date: 18 June 2024 Day: Tuesday

Overview:

Day 10 of the internship was dedicated to mastering the async/await syntax in Node.js, a modern approach to writing asynchronous code that enhances readability and maintainability compared to traditional callback and promise patterns.

Learning Objectives:

- Understanding async/await Syntax:
    - Explored the async function declaration and await operator in JavaScript to manage asynchronous operations seamlessly.
    - Learned how async/await simplifies asynchronous code by allowing it to look more like synchronous code, improving code readability and reducing callback nesting.
- Comparison with Traditional Patterns:

Compared async/await with traditional callback and promise patterns:

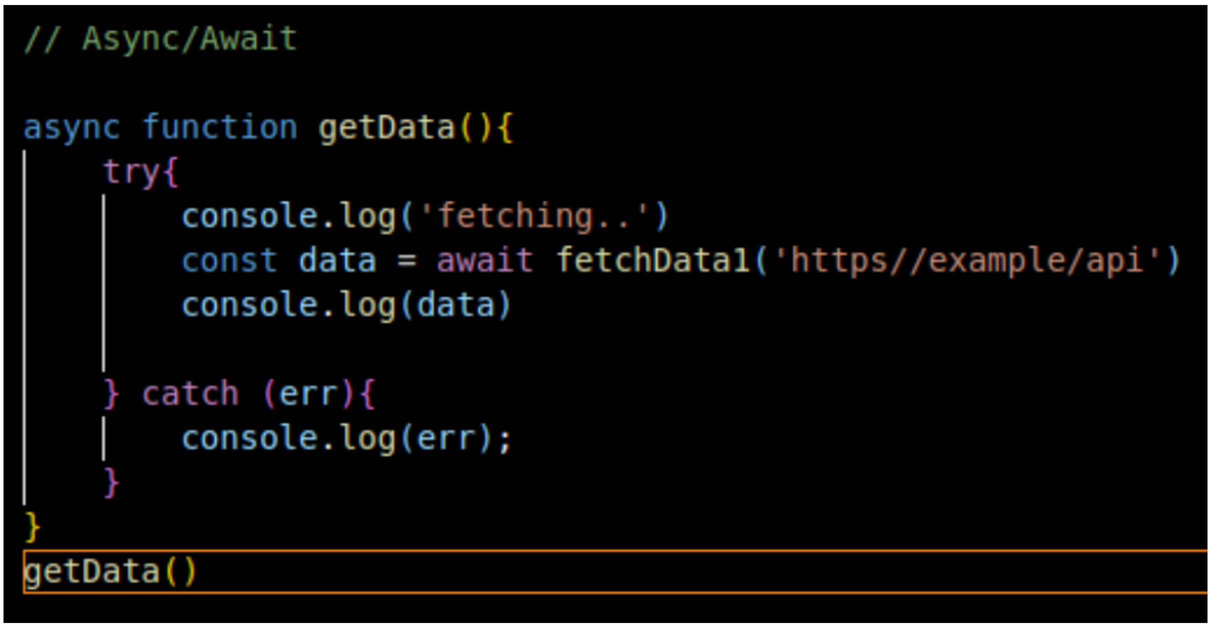
- **Callback Pattern**: Highlighted the drawbacks of callback hell and difficulty in error handling and sequential flow.
- **Promise Pattern:** Discussed the benefits of promises in managing asynchronous operations but noted potential verbosity and nesting issues.
- async/await Pattern: Emphasized the clarity and conciseness of async/await for handling asynchronous tasks, providing a more intuitive and structured approach.

- Refactoring to async/await:
    - Practiced refactoring existing callback-based and promise-based code to utilize async/await for improved readability and maintainability.
    - Implemented asynchronous functions using async/await to handle file I/O, database queries, and API calls, ensuring non-blocking execution and error handling.

Activities and Insights:

- Implementation of async/await Syntax:
    - Converted callback-based functions to async/await syntax to simplify code structure and improve error handling.
    - Utilized try-catch blocks with await for robust error management, ensuring comprehensive error handling in asynchronous workflows.
- Comparative Analysis:
    - Analyzed performance improvements and code readability enhancements achieved by transitioning from callback and promise patterns to async/await.
    - Documented best practices for integrating async/await into existing codebases, including migration strategies and impact on testing and debugging processes.
- Collaboration and Feedback:
    - Collaborated with team members to review and optimize asynchronous code refactored with async/await, gathering feedback on readability and maintainability improvements.

- Shared insights on using async/await effectively across different project modules and discussed potential scenarios for future optimizations.



```
// Async/Await

async function getData(){
    try{
        console.log('fetching..')
        const data = await fetchData1('https//example/api')
        console.log(data)

    } catch (err){
        console.log(err);
    }
}
getData()
```