Day 18 – Asynchronous Programming in Node.js

Challenge 7: Callbacks

User Story:
As a developer, I want to read a file and then display a confirmation message once it's done – using callbacks.

Problem Statement:
Use the **fs module** with a callback function to:

- Read a file (data.txt).
- Log "Read operation completed" after reading.

Expected Outcome:
Should demonstrate asynchronous behavior.

Bonus:
Add an intentional delay using setTimeout() before confirmation.

Self-Evaluation Metrics:

Metric	Target
Used callback structure correctly	
No blocking behavior observed	

Challenge 8: Promises

User Story:
As a developer, I want to chain multiple async operations (read file → write to another file) using Promises.

Problem Statement:
Use Promises with **fs.promises** to:

1. Read content from input.txt

2. Write same content to output.txt

3. Log “File copied successfully!”

Expected Outcome:
Chained Promises working without callback nesting.

Bonus:
Handle errors using .catch().

Self-Evaluation Metrics:

Metric	Target
Used Promises correctly	
Handled .then() and .catch()	

Challenge 9: Async/Await

User Story:
As a developer, I want cleaner syntax for asynchronous operations using modern JavaScript.

Problem Statement:
Convert the previous challenge into an **async/await** version using try/catch.

Expected Outcome:
Same file copy operation, but uses async/await syntax.

Bonus:
Add an artificial delay (await new Promise(res => setTimeout(res, 1000))) to simulate a slow operation.

Self-Evaluation Metrics:

Metric	Target
Used async/await properly	
Wrapped logic in try/catch	
Graceful error handling implemented	