Problem Statement: Console-Based Usage-Based Invoicing System (C#/JavaScript/TypeScript)

Objective

Develop a clean, maintainable console application in C#, JavaScript (Node.js), or TypeScript (Node.js) that reads customer usage data from a JSON file, validates the data, computes usage-based charges, and prints formatted invoices to the console. Total time to build must not exceed 90 minutes.

Input: usage-data.json

The application must read from a JSON file containing an array of customer usage entries. Some entries will have incomplete or invalid data to test robustness.


```
[
 {
   "CustomerId": "CUST001",
   "API_Calls": 8500,
   "Storage_GB": 45.5,
   "Compute_Minutes": 150
 },
 {
   "CustomerId": "CUST002",
   "API_Calls": 12500,
   "Storage_GB": 120,
   "Compute_Minutes": 310
 },
 {
   "CustomerId": "CUST003",
   "API_Calls": "not_a_number",
   "Storage_GB": 10,
   "Compute_Minutes": 90
 },
 {
   "CustomerId": "CUST004",
```

```
    "API_Calls": 18000,
    "Storage_GB": null,
    "Compute_Minutes": 600
  },
  {
    "CustomerId": null,
    "API_Calls": 1000,
    "Storage_GB": 5,
    "Compute_Minutes": 50
  },
  {
    "CustomerId": "CUST006"
  }
]
```


Pricing Rules

- API Calls
  - First 10,000 calls -> $0.01 each
  - Above 10,000 calls -> $0.008 each
- Storage -> $0.25 per GB
- Compute Time -> $0.05 per minute

Functional Requirements

1. Load and Validate Input
   - Parse the JSON file.
   - Validate each entry:
     - All fields must be present and of correct type.
     - If a required field is missing or malformed, skip the entry and log a warning to the console.

2. Invoice Calculation
   - For valid entries:
     - Compute individual costs for API, storage, and compute.

- Calculate and format total.

3. Console Output

- For each valid customer, print a detailed invoice:

  Invoice for Customer: CUST001

  API Calls: 8500 calls -> $85.00

  Storage: 45.5 GB -> $11.38

  Compute Time: 150 minutes -> $7.50

  Total Due: $103.88

- For invalid entries, print:

  Skipped invalid entry: Missing or invalid fields for CustomerId: CUST003

Non-Functional Requirements

- Modular architecture:
  - InputLoader for parsing and validation.
  - InvoiceCalculator for applying pricing logic.
  - InvoicePrinter for console output.
  - Use constants for all pricing values.
- Maintainable, testable structure with single-responsibility per module/class.
- Avoid hardcoding logic in global scope.

Constraints

- No UI or external services.
- No persistent storage.
- Entire solution must be complete and runnable in under 90 minutes.

Bonus

- Accept JSON file path via command-line argument.
- Include light unit testing of invoice calculation logic.

Evaluation Criteria

- Correctness of calculations.

- Graceful handling of malformed data.

- Code readability, structure, and modularity.

- Output clarity.