**Lesson 07: Generative AI in Testing**

**Overview:**

In this project, we explore how to use generative AI, specifically Amazon Q to speed up testing for a mobile e-commerce app. We'll use this technology to quickly create synthetic data and generate test cases that match our project goals. Participants will also create files with test data and test cases focusing on customer interactions, transactions, and products related to a mobile e-commerce site. Through practical exercises, participants will learn how to use GenAI for efficient test planning and management, which are crucial for the success of software development projects.

Instructions:

1. Go to Amazon Q and start the testing process by entering specific prompts to generate synthetic data and test cases
2. Use the provided code to incorporate the synthetic data and test cases into your Python application
3. Connect the new synthetic data and test cases to your application with the help of prompts
4. If needed, update your application by creating a new script via Amazon Q to fix any missing code or steps
5. Execute the test cases using the node js jest framework package in Node JS

**Tasks:**

1. Generate synthetic data for customer, product, and transaction for mobile

e-commerce website

* 1. Connect Amazon Q to Visual Studio Code
  2. Generate synthetic customer data using Amazon Q
  3. Generate synthetic product data using Amazon Q
  4. Generate synthetic transaction data using Amazon Q
  5. Link the application with all combined data

1. Generate test cases for mobile e-commerce website with Customer Products and Transactions data
   1. Create a file and add the test cases
2. Run and execute those test cases using JEST
   1. Install node and execute the test cases

**Tools Required:**

1. Visual Studio Code
2. Amazon Q

**Guided Practice Solution**

**Task 1: Generate synthetic data for customer, product, and transaction for mobile e-commerce website**

**Step 1: Connect Amazon Q to Visual Studio Code**

* 1. Navigate to Visual Studio Code, search for the **Amazon Q** extension, and install it

A screenshot of a computer

Description automatically generated

* 1. Once installed, click on **Use For Free**

A screenshot of a computer

Description automatically generated

* 1. Click on **Open**

A screenshot of a computer

Description automatically generated

* 1. Enter your details and click on **Next**A screenshot of a computer

     Description automatically generated
  2. Click on **Allow access**  
       
     A screenshot of a web page

     Description automatically generated  
      The screen will appear as below:

A screenshot of a computer

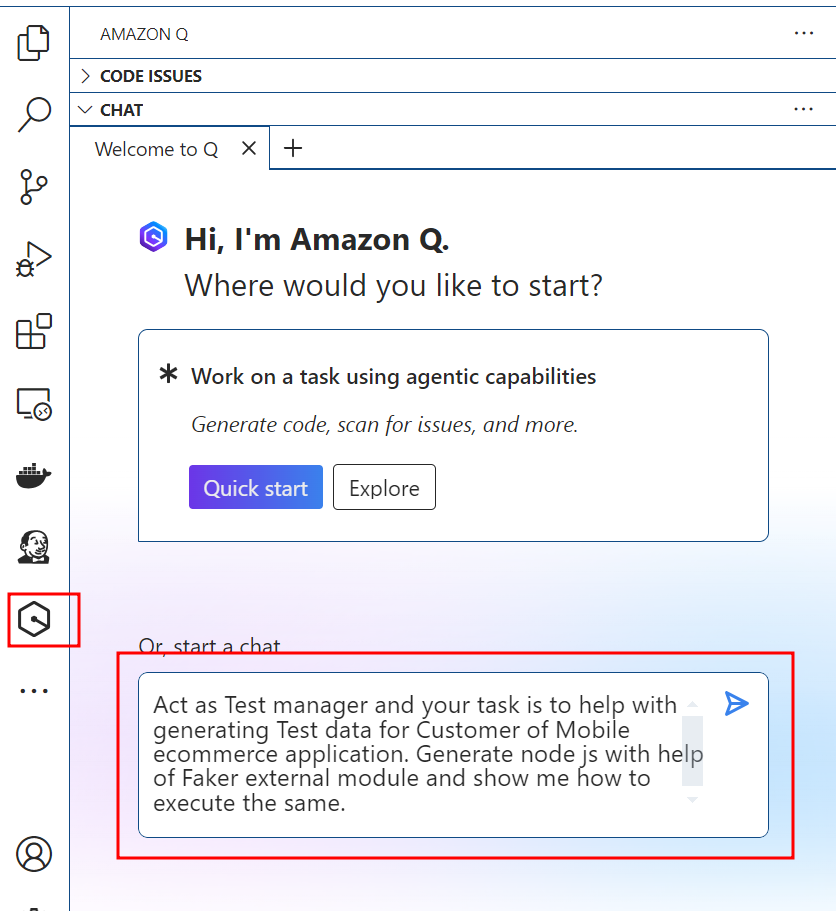
Description automatically generated

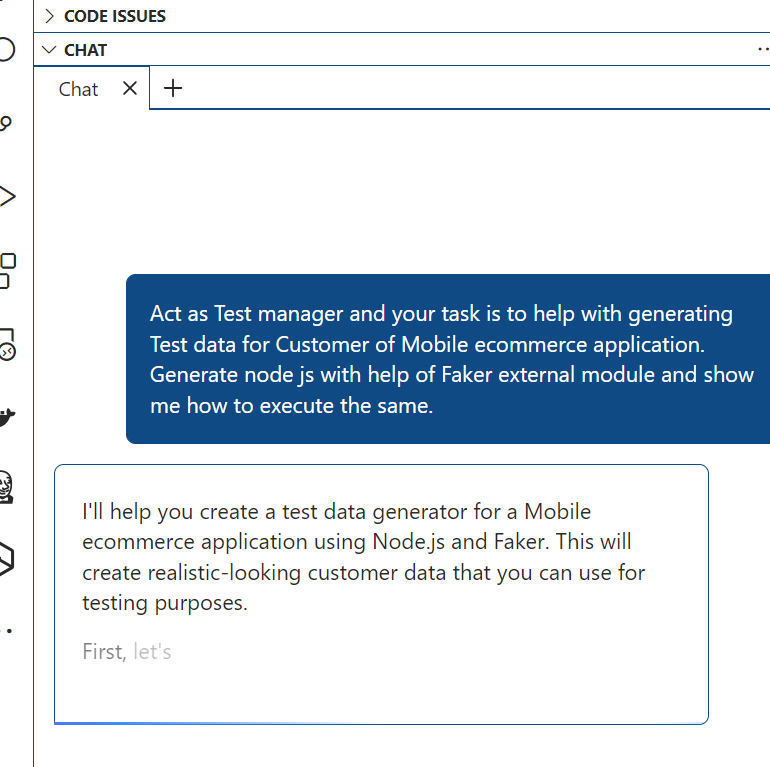
**Step 2: Generate synthetic customer data using Amazon Q**

* 1. Create a folder **GenAI\_testing**

Use the following prompt in Amazon Q to generate synthetic customer data:

**Act as Test manager and your task is to help with generating Test data for Customer of Mobile ecommerce application. Generate node js with help of Faker external module and show me how to execute the same.**





**Run the following commands:**

npm init -y

**npm install @faker-js/faker**

create the file with name as **generateTestData.js** with below code

const { faker } = require('@faker-js/faker');

const fs = require('fs');

// Function to generate a single customer record

const generateCustomer = () => {

return {

id: faker.string.uuid(),

firstName: faker.person.firstName(),

lastName: faker.person.lastName(),

email: faker.internet.email(),

phone: faker.phone.number(),

address: {

street: faker.location.streetAddress(),

city: faker.location.city(),

state: faker.location.state(),

zipCode: faker.location.zipCode(),

country: faker.location.country()

},

preferences: {

preferredPaymentMethod: faker.helpers.arrayElement(['Credit Card', 'PayPal', 'Apple Pay', 'Google Pay']),

notifications: faker.helpers.arrayElement(['SMS', 'Email', 'Both', 'None']),

language: faker.helpers.arrayElement(['English', 'Spanish', 'French', 'German'])

},

orderHistory: Array.from({ length: faker.number.int({ min: 0, max: 5 }) }, () => ({

orderId: faker.string.uuid(),

orderDate: faker.date.past(),

orderTotal: faker.commerce.price({ min: 10, max: 500 }),

items: Array.from({ length: faker.number.int({ min: 1, max: 4 }) }, () => ({

productId: faker.string.uuid(),

productName: faker.commerce.productName(),

price: faker.commerce.price(),

quantity: faker.number.int({ min: 1, max: 5 })

}))

})),

accountCreated: faker.date.past(),

lastLogin: faker.date.recent()

};

};

// Function to generate multiple customers

const generateCustomers = (count) => {

const customers = [];

for (let i = 0; i < count; i++) {

customers.push(generateCustomer());

}

return customers;

};

// Generate 100 customers

const customerData = generateCustomers(100);

// Save to JSON file

fs.writeFileSync(

'customer\_test\_data.json',

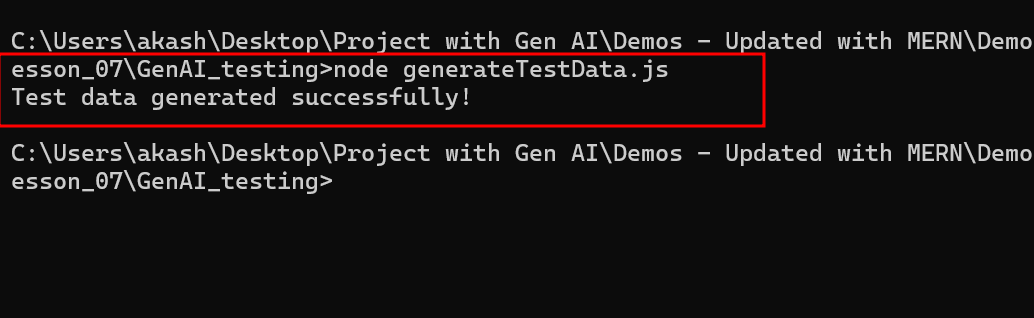
JSON.stringify(customerData, null, 2),

'utf-8'

);

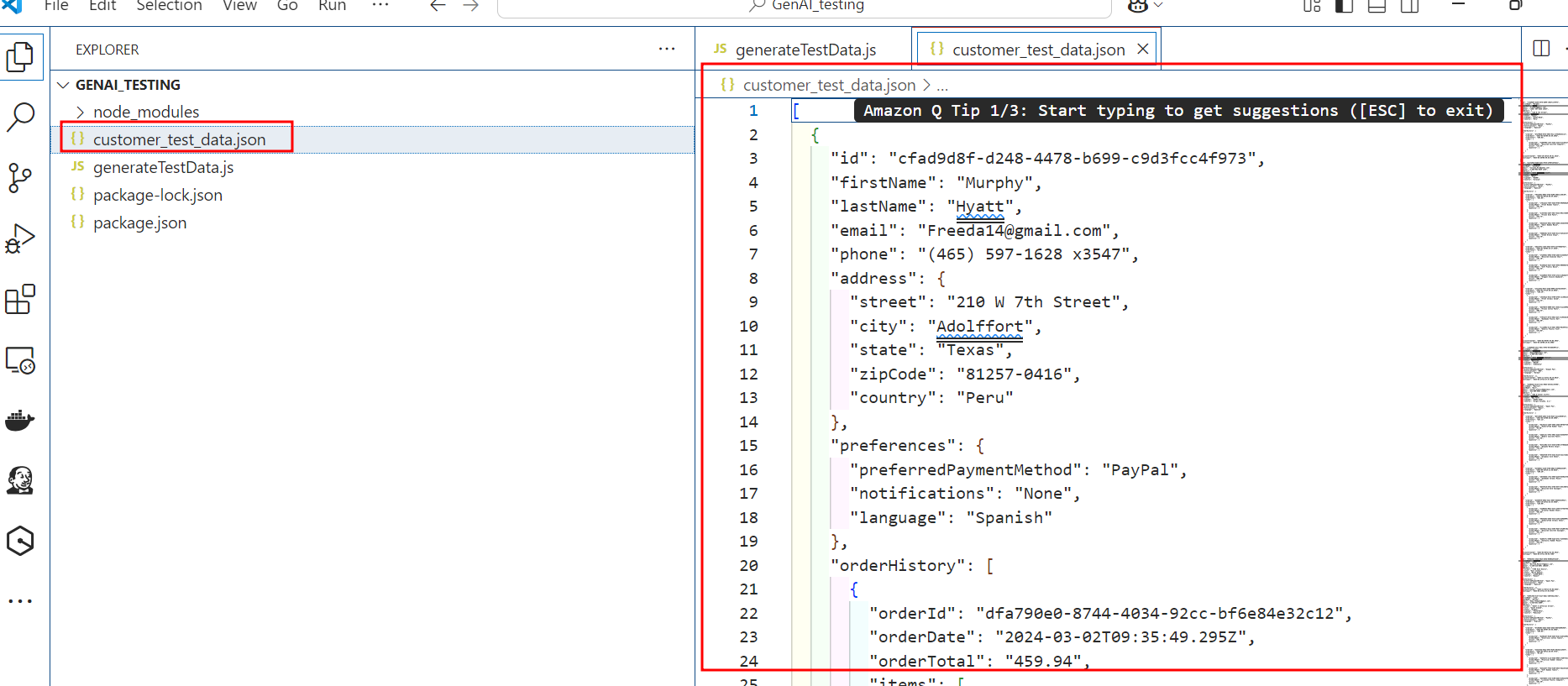
console.log('Test data generated successfully!');

To run this application

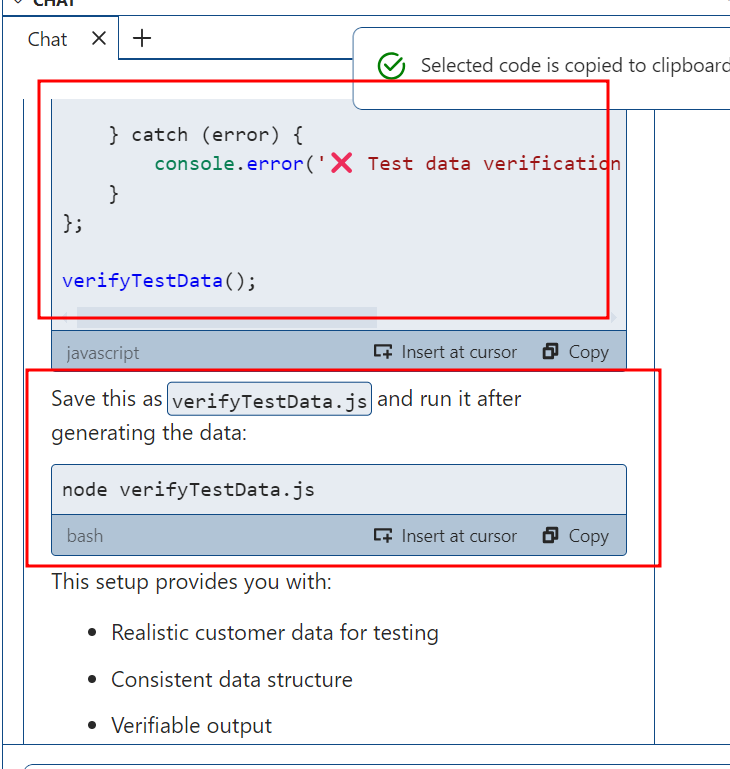


You can see the test data generated in json file.

**node testDataGenerator.js**



**Event Amazon Q provide help to test this code using assert module**



**Create the file as verifyTestData.js and copy/paste the code as**

const assert = require('assert');

const fs = require('fs');

// Test verification script

const verifyTestData = () => {

try {

// Read generated data

const data = JSON.parse(fs.readFileSync('customer\_test\_data.json', 'utf-8'));

// Basic verification

assert(Array.isArray(data), 'Data should be an array');

assert(data.length === 100, 'Should have 100 records');

// Verify first record structure

const firstRecord = data[0];

assert(firstRecord.id, 'Record should have ID');

assert(firstRecord.firstName, 'Record should have firstName');

assert(firstRecord.email, 'Record should have email');

assert(firstRecord.address, 'Record should have address');

console.log('✅ Test data verification passed!');

console.log(`Total records generated: ${data.length}`);

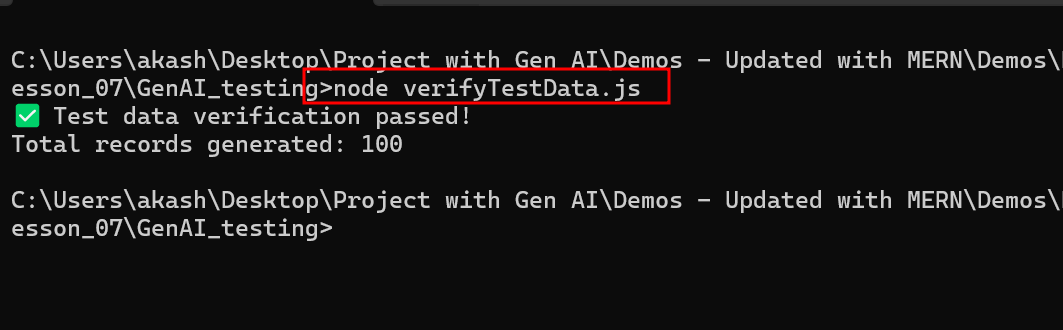
} catch (error) {

console.error('❌ Test data verification failed:', error.message);

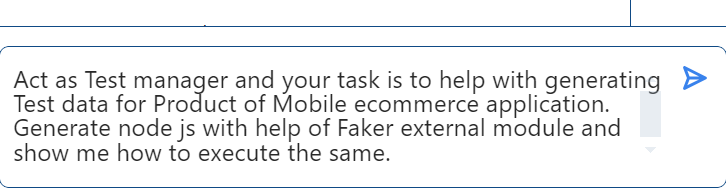
}

};

verifyTestData();



**Act as Test manager and your task is to help with generating Test data for Product of Mobile ecommerce application. Generate node js with help of Faker external module and show me how to execute the same.**



Create two files:

generateProductData.js (first script)

const { faker } = require('@faker-js/faker');

const fs = require('fs');

// Function to generate product categories

const generateCategories = () => {

return [

'Smartphones',

'Tablets',

'Accessories',

'Wearables',

'Audio Devices',

'Mobile Cases',

'Screen Protectors',

'Chargers',

'Power Banks',

'Storage Devices'

];

};

// Function to generate a single product

const generateProduct = (categories) => {

const category = faker.helpers.arrayElement(categories);

const brand = faker.company.name();

return {

productId: faker.string.uuid(),

sku: faker.string.alphanumeric(8).toUpperCase(),

name: `${brand} ${faker.commerce.productName()}`,

brand: brand,

category: category,

description: faker.commerce.productDescription(),

price: {

regular: parseFloat(faker.commerce.price({ min: 50, max: 2000 })),

discount: faker.number.int({ min: 0, max: 30 }),

currency: 'USD'

},

specifications: {

color: faker.helpers.arrayElement(['Black', 'White', 'Silver', 'Gold', 'Blue']),

weight: `${faker.number.int({ min: 100, max: 500 })}g`,

dimensions: {

length: `${faker.number.int({ min: 10, max: 30 })}cm`,

width: `${faker.number.int({ min: 5, max: 15 })}cm`,

height: `${faker.number.int({ min: 1, max: 5 })}cm`

}

},

inventory: {

inStock: faker.number.int({ min: 0, max: 100 }),

warehouse: faker.location.city(),

leadTime: `${faker.number.int({ min: 1, max: 10 })} days`

},

ratings: {

average: faker.number.float({ min: 1, max: 5, precision: 0.1 }),

count: faker.number.int({ min: 0, max: 1000 })

},

images: Array.from({ length: 4 }, () => ({

url: faker.image.url(),

alt: faker.commerce.productName(),

isPrimary: false

})),

features: Array.from(

{ length: faker.number.int({ min: 3, max: 6 }) },

() => faker.commerce.productAdjective()

),

tags: Array.from(

{ length: faker.number.int({ min: 2, max: 5 }) },

() => faker.commerce.productMaterial()

),

createdAt: faker.date.past(),

updatedAt: faker.date.recent(),

isActive: faker.datatype.boolean()

};

};

// Function to generate multiple products

const generateProducts = (count) => {

const categories = generateCategories();

const products = [];

for (let i = 0; i < count; i++) {

products.push(generateProduct(categories));

}

return products;

};

// Generate 50 products

const productData = generateProducts(50);

// Save to JSON file

fs.writeFileSync(

'product\_test\_data.json',

JSON.stringify(productData, null, 2),

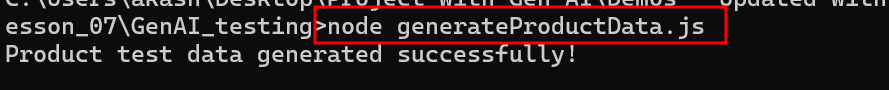
'utf-8'

);

console.log('Product test data generated successfully!');

**run the application as**

node generateProductData.js



Here product\_test\_data.json file created

After run this application it will create

verifyProductData.js (second script)

const assert = require('assert');

const fs = require('fs');

const verifyProductData = () => {

    try {

        // Read generated data

        const data = JSON.parse(fs.readFileSync('product\_test\_data.json', 'utf-8'));

        // Basic verification

        assert(Array.isArray(data), 'Data should be an array');

        assert(data.length === 50, 'Should have 50 records');

        // Verify first record structure

        const firstProduct = data[0];

        assert(firstProduct.productId, 'Product should have ID');

        assert(firstProduct.name, 'Product should have name');

        assert(firstProduct.price, 'Product should have price');

        assert(firstProduct.category, 'Product should have category');

        assert(firstProduct.inventory, 'Product should have inventory');

        // Verify price structure

        assert(typeof firstProduct.price.regular === 'number', 'Price should be a number');

        assert(firstProduct.price.currency === 'USD', 'Currency should be USD');

        console.log('✅ Product test data verification passed!');

        console.log(`Total products generated: ${data.length}`);

        // Display sample product

        console.log('\nSample Product:');

        console.log(JSON.stringify(firstProduct, null, 2));

    } catch (error) {

        console.error('❌ Product test data verification failed:', error.message);

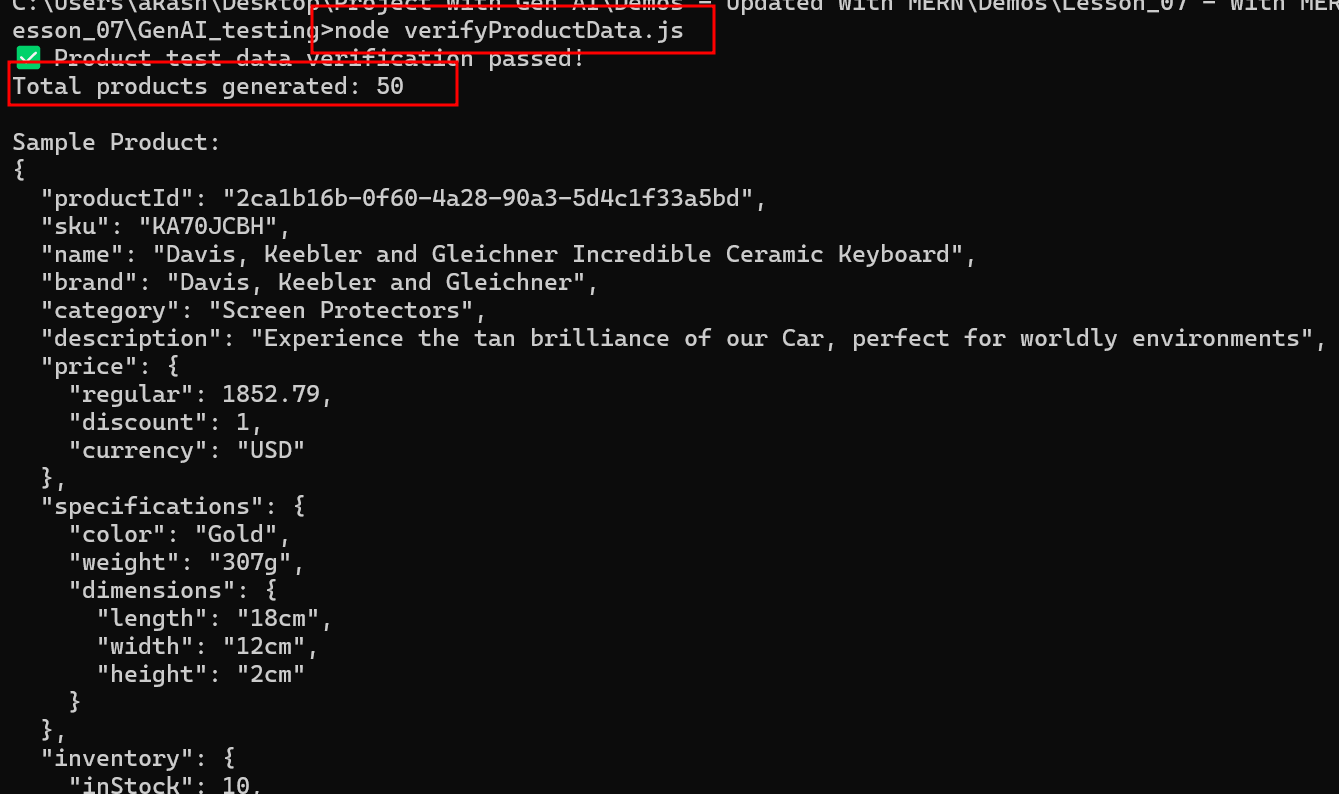
    }

};

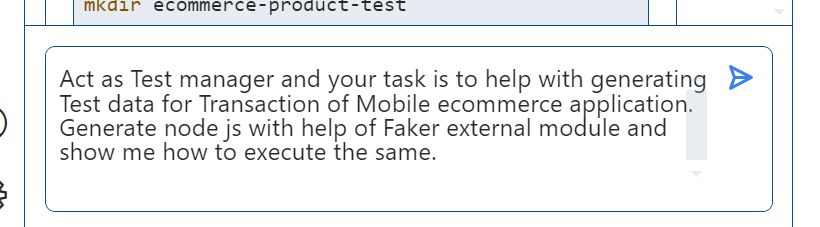
verifyProductData();

**test this application using node js assert module**

**node verifyProductData.js**



**Act as Test manager and your task is to help with generating Test data for Transaction of Mobile ecommerce application. Generate node js with help of Faker external module and show me how to execute the same.**



**Here create the two file as**

**Create two files:**

**generateTransactionData.js (first script)**

const { faker } = require('@faker-js/faker');

const fs = require('fs');

// Function to generate payment methods

const getPaymentMethods = () => ({

    CREDIT\_CARD: {

        type: 'CREDIT\_CARD',

        cardNumber: faker.finance.creditCardNumber(),

        expiryDate: faker.date.future().toISOString().slice(0, 7),

        cardHolder: faker.person.fullName()

    },

    DEBIT\_CARD: {

        type: 'DEBIT\_CARD',

        cardNumber: faker.finance.creditCardNumber(),

        bankName: faker.company.name()

    },

    UPI: {

        type: 'UPI',

        upiId: `${faker.internet.username()}@upi`

    },

    WALLET: {

        type: 'WALLET',

        walletProvider: faker.helpers.arrayElement(['PayPal', 'Google Pay', 'Apple Pay'])

    }

});

// Function to generate a single transaction

const generateTransaction = () => {

    const paymentMethods = getPaymentMethods();

    const paymentMethod = faker.helpers.arrayElement(Object.values(paymentMethods));

    const orderStatus = faker.helpers.arrayElement([

        'PENDING', 'PROCESSING', 'SHIPPED', 'DELIVERED', 'CANCELLED', 'REFUNDED'

    ]);

    const itemCount = faker.number.int({ min: 1, max: 5 });

    const items = Array.from({ length: itemCount }, () => ({

        productId: faker.string.uuid(),

        productName: faker.commerce.productName(),

        quantity: faker.number.int({ min: 1, max: 5 }),

        unitPrice: parseFloat(faker.commerce.price()),

        category: faker.commerce.department()

    }));

    // Calculate totals

    const subtotal = items.reduce((sum, item) => sum + (item.unitPrice \* item.quantity), 0);

    const taxRate = 0.08;

    const shippingCost = parseFloat(faker.commerce.price(10, 50));

    const tax = subtotal \* taxRate;

    const total = subtotal + tax + shippingCost;

    return {

        transactionId: faker.string.uuid(),

        orderId: `ORD-${faker.string.alphanumeric(8).toUpperCase()}`,

        customerId: faker.string.uuid(),

        customerDetails: {

            name: faker.person.fullName(),

            email: faker.internet.email(),

            phone: faker.phone.number()

        },

        orderDate: faker.date.recent(),

        items: items,

        billing: {

            subtotal: parseFloat(subtotal.toFixed(2)),

            tax: parseFloat(tax.toFixed(2)),

            shippingCost: shippingCost,

            total: parseFloat(total.toFixed(2)),

            currency: 'USD'

        },

        payment: {

            method: paymentMethod.type,

            details: paymentMethod,

            status: faker.helpers.arrayElement(['SUCCESS', 'PENDING', 'FAILED']),

            paidAt: faker.date.recent()

        },

        shipping: {

            address: {

                street: faker.location.streetAddress(),

                city: faker.location.city(),

                state: faker.location.state(),

                zipCode: faker.location.zipCode(),

                country: faker.location.country()

            },

            method: faker.helpers.arrayElement(['Standard', 'Express', 'Next Day']),

            trackingNumber: faker.string.alphanumeric(12).toUpperCase(),

            estimatedDelivery: faker.date.future()

        },

        status: orderStatus,

        statusHistory: [

            {

                status: 'PENDING',

                timestamp: faker.date.recent(),

                comment: 'Order placed successfully'

            },

            {

                status: orderStatus,

                timestamp: faker.date.recent(),

                comment: `Order ${orderStatus.toLowerCase()}`

            }

        ],

        metadata: {

            deviceInfo: {

                platform: faker.helpers.arrayElement(['iOS', 'Android']),

                deviceModel: faker.helpers.arrayElement(['iPhone 13', 'Samsung S21', 'Google Pixel']),

                ipAddress: faker.internet.ip()

            },

            userAgent: faker.internet.userAgent(),

            sessionId: faker.string.uuid()

        }

    };

};

// Function to generate multiple transactions

const generateTransactions = (count) => {

    const transactions = [];

    for (let i = 0; i < count; i++) {

        transactions.push(generateTransaction());

    }

    return transactions;

};

// Generate 50 transactions

const transactionData = generateTransactions(50);

// Save to JSON file

fs.writeFileSync(

    'transaction\_test\_data.json',

    JSON.stringify(transactionData, null, 2),

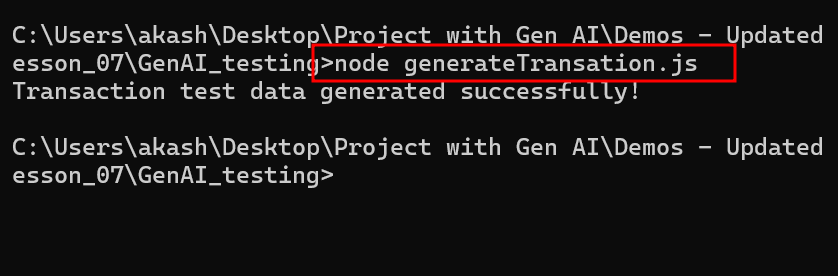
    'utf-8'

);

console.log('Transaction test data generated successfully!');

run the application as

**node generateTransaction.js**



**verifyTransactionData.js (second script)**

const assert = require('assert');

const fs = require('fs');

const verifyTransactionData = () => {

    try {

        // Read generated data

        const data = JSON.parse(fs.readFileSync('transaction\_test\_data.json', 'utf-8'));

        // Basic verification

        assert(Array.isArray(data), 'Data should be an array');

        assert(data.length === 50, 'Should have 50 records');

        // Verify first record structure

        const firstTransaction = data[0];

        assert(firstTransaction.transactionId, 'Transaction should have ID');

        assert(firstTransaction.orderId, 'Transaction should have order ID');

        assert(firstTransaction.customerDetails, 'Transaction should have customer details');

        assert(firstTransaction.items, 'Transaction should have items');

        assert(firstTransaction.billing, 'Transaction should have billing info');

        assert(firstTransaction.payment, 'Transaction should have payment info');

        // Verify billing calculations

        const billing = firstTransaction.billing;

        assert(typeof billing.total === 'number', 'Total should be a number');

        assert(billing.total === billing.subtotal + billing.tax + billing.shippingCost,

            'Total should equal subtotal + tax + shipping');

        console.log('✅ Transaction test data verification passed!');

        console.log(`Total transactions generated: ${data.length}`);

        // Display sample transaction

        console.log('\nSample Transaction:');

        console.log(JSON.stringify(firstTransaction, null, 2));

    } catch (error) {

        console.error('❌ Transaction test data verification failed:', error.message);

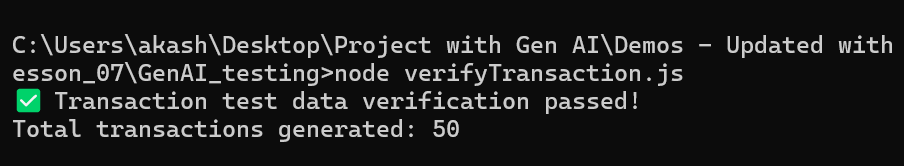
    }

};

verifyTransactionData();

run the test file

**node verifyTransaction.js**



**Act as Test manager and your task is to help with combine all these code product, customer and transaction for ecommerce application using node js.**

It generate the output

Now create the generateEcommerceData.js file and copy/paste the generated code.

const { faker } = require('@faker-js/faker');

const fs = require('fs');

// Product Generation Functions

const generateCategories = () => {

return [

'Smartphones',

'Tablets',

'Accessories',

'Wearables',

'Audio Devices',

'Mobile Cases',

'Screen Protectors',

'Chargers',

'Power Banks',

'Storage Devices'

];

};

const generateProduct = (categories) => {

const category = faker.helpers.arrayElement(categories);

const brand = faker.company.name();

const regularPrice = parseFloat(faker.commerce.price({ min: 50, max: 2000 }));

const discount = faker.number.int({ min: 0, max: 30 });

const discountedPrice = regularPrice \* (1 - discount/100);

return {

productId: faker.string.uuid(),

sku: faker.string.alphanumeric(8).toUpperCase(),

name: `${brand} ${faker.commerce.productName()}`,

brand: brand,

category: category,

description: faker.commerce.productDescription(),

price: {

regular: regularPrice,

discounted: parseFloat(discountedPrice.toFixed(2)),

discount: discount,

currency: 'USD'

},

specifications: {

color: faker.helpers.arrayElement(['Black', 'White', 'Silver', 'Gold', 'Blue']),

weight: `${faker.number.int({ min: 100, max: 500 })}g`,

dimensions: {

length: `${faker.number.int({ min: 10, max: 30 })}cm`,

width: `${faker.number.int({ min: 5, max: 15 })}cm`,

height: `${faker.number.int({ min: 1, max: 5 })}cm`

}

},

inventory: {

inStock: faker.number.int({ min: 0, max: 100 }),

warehouse: faker.location.city(),

leadTime: `${faker.number.int({ min: 1, max: 10 })} days`

},

ratings: {

average: faker.number.float({ min: 1, max: 5, precision: 0.1 }),

count: faker.number.int({ min: 0, max: 1000 })

},

isActive: faker.datatype.boolean()

};

};

// Customer Generation Functions

const generateCustomer = () => {

const firstName = faker.person.firstName();

const lastName = faker.person.lastName();

return {

customerId: faker.string.uuid(),

firstName: firstName,

lastName: lastName,

email: faker.internet.email({ firstName, lastName }),

phone: faker.phone.number(),

address: {

street: faker.location.streetAddress(),

city: faker.location.city(),

state: faker.location.state(),

zipCode: faker.location.zipCode(),

country: faker.location.country()

},

preferences: {

preferredPaymentMethod: faker.helpers.arrayElement(['Credit Card', 'PayPal', 'Apple Pay', 'Google Pay']),

notifications: faker.helpers.arrayElement(['SMS', 'Email', 'Both', 'None'])

},

accountCreated: faker.date.past(),

lastLogin: faker.date.recent()

};

};

// Transaction Generation Functions

const generateTransaction = (customers, products) => {

const customer = faker.helpers.arrayElement(customers);

const itemCount = faker.number.int({ min: 1, max: 5 });

const selectedProducts = faker.helpers.arrayElements(products, itemCount);

const items = selectedProducts.map(product => ({

productId: product.productId,

productName: product.name,

quantity: faker.number.int({ min: 1, max: 5 }),

unitPrice: product.price.discounted,

category: product.category

}));

const subtotal = items.reduce((sum, item) => sum + (item.unitPrice \* item.quantity), 0);

const taxRate = 0.08;

const shippingCost = parseFloat(faker.commerce.price(10, 50));

const tax = subtotal \* taxRate;

const total = subtotal + tax + shippingCost;

return {

transactionId: faker.string.uuid(),

orderId: `ORD-${faker.string.alphanumeric(8).toUpperCase()}`,

customerId: customer.customerId,

customerDetails: {

name: `${customer.firstName} ${customer.lastName}`,

email: customer.email,

phone: customer.phone

},

orderDate: faker.date.recent(),

items: items,

billing: {

subtotal: parseFloat(subtotal.toFixed(2)),

tax: parseFloat(tax.toFixed(2)),

shippingCost: shippingCost,

total: parseFloat(total.toFixed(2)),

currency: 'USD'

},

shipping: {

address: customer.address,

method: faker.helpers.arrayElement(['Standard', 'Express', 'Next Day']),

trackingNumber: faker.string.alphanumeric(12).toUpperCase(),

estimatedDelivery: faker.date.future()

},

status: faker.helpers.arrayElement(['PENDING', 'PROCESSING', 'SHIPPED', 'DELIVERED']),

paymentStatus: faker.helpers.arrayElement(['PAID', 'PENDING', 'FAILED'])

};

};

// Main Generation Function

const generateEcommerceData = (productCount = 50, customerCount = 100, transactionCount = 200) => {

console.log('Generating ecommerce test data...');

// Generate Products

const categories = generateCategories();

const products = Array.from({ length: productCount }, () => generateProduct(categories));

// Generate Customers

const customers = Array.from({ length: customerCount }, generateCustomer);

// Generate Transactions

const transactions = Array.from({ length: transactionCount }, () =>

generateTransaction(customers, products)

);

// Save all data to separate files

const dataDir = './test\_data';

if (!fs.existsSync(dataDir)){

fs.mkdirSync(dataDir);

}

fs.writeFileSync(

`${dataDir}/products.json`,

JSON.stringify(products, null, 2)

);

fs.writeFileSync(

`${dataDir}/customers.json`,

JSON.stringify(customers, null, 2)

);

fs.writeFileSync(

`${dataDir}/transactions.json`,

JSON.stringify(transactions, null, 2)

);

// Generate summary

const summary = {

productsGenerated: products.length,

customersGenerated: customers.length,

transactionsGenerated: transactions.length,

timestamp: new Date().toISOString()

};

fs.writeFileSync(

`${dataDir}/summary.json`,

JSON.stringify(summary, null, 2)

);

return summary;

};

// Verification Function

const verifyGeneratedData = () => {

try {

const dataDir = './test\_data';

// Read all data files

const products = JSON.parse(fs.readFileSync(`${dataDir}/products.json`));

const customers = JSON.parse(fs.readFileSync(`${dataDir}/customers.json`));

const transactions = JSON.parse(fs.readFileSync(`${dataDir}/transactions.json`));

// Verify products

console.log('\nVerifying products...');

console.log(`✓ ${products.length} products generated`);

console.log(`✓ Sample product: ${products[0].name}`);

// Verify customers

console.log('\nVerifying customers...');

console.log(`✓ ${customers.length} customers generated`);

console.log(`✓ Sample customer: ${customers[0].firstName} ${customers[0].lastName}`);

// Verify transactions

console.log('\nVerifying transactions...');

console.log(`✓ ${transactions.length} transactions generated`);

console.log(`✓ Sample transaction ID: ${transactions[0].orderId}`);

// Verify relationships

const sampleTransaction = transactions[0];

const customerExists = customers.some(c => c.customerId === sampleTransaction.customerId);

const productExists = products.some(p => p.productId === sampleTransaction.items[0].productId);

console.log('\nVerifying relationships...');

console.log(`✓ Customer reference valid: ${customerExists}`);

console.log(`✓ Product reference valid: ${productExists}`);

console.log('\n✅ All verifications passed successfully!');

} catch (error) {

console.error('❌ Verification failed:', error.message);

}

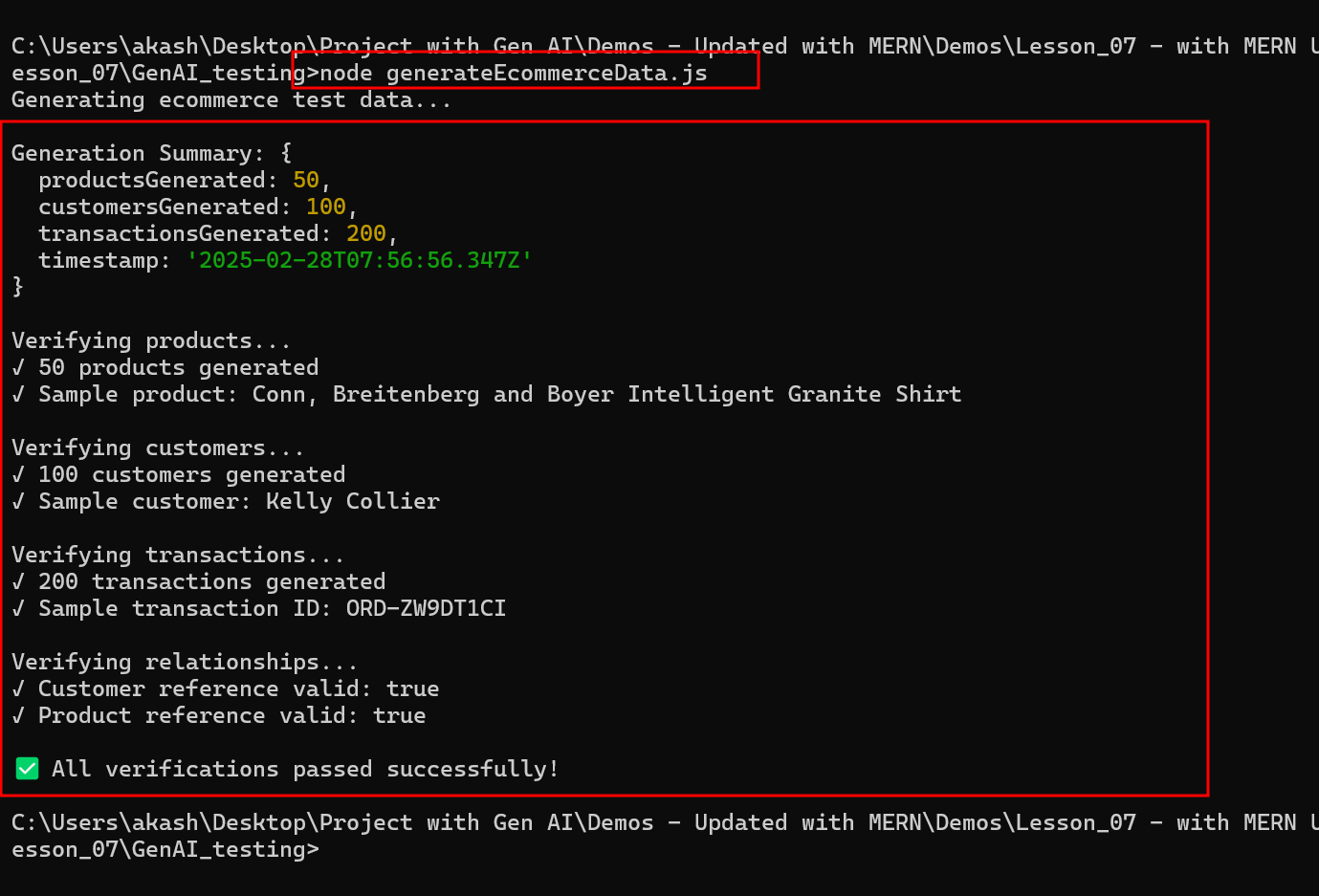
};

// Execute the generation and verification

const summary = generateEcommerceData();

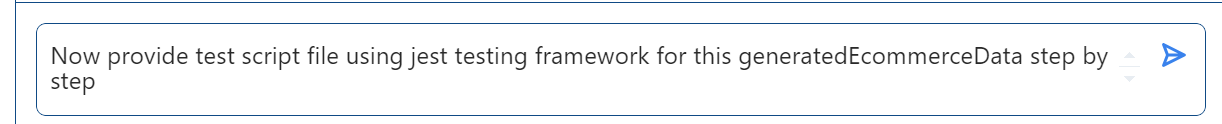
console.log('\nGeneration Summary:', summary);

verifyGeneratedData();



**Now you ask the AI tool to generate testing file using jest testing framework**

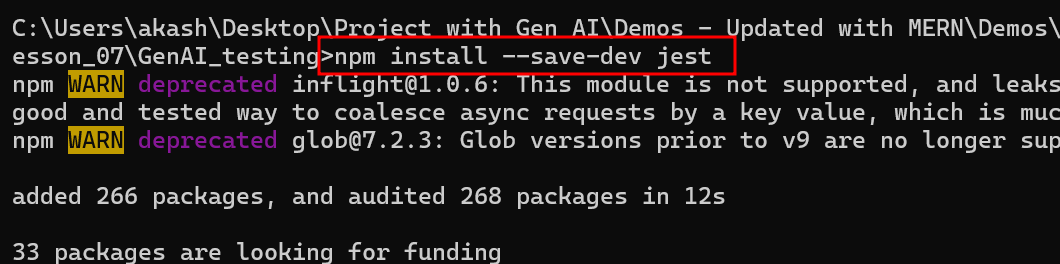
**Now provide test script file using jest testing framework for this generatedEcommerceData step by step**



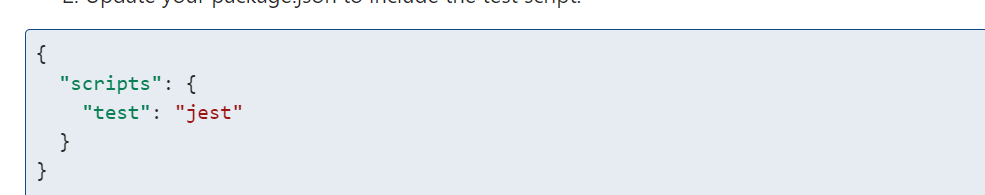
It generate the output as

First, install Jest:

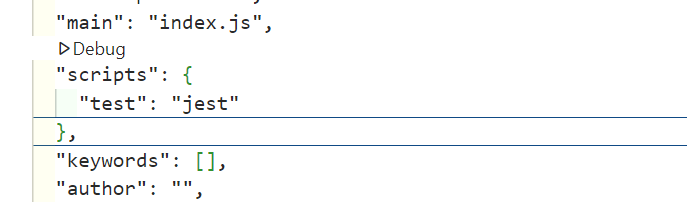
**npm install --save-dev jest**



Update your package.json to include the test script:



After updated in package.json file



Now you need to create the file as

**generateEcommerceData.test.js**

const fs = require('fs');

const path = require('path');

const { faker } = require('@faker-js/faker');

// Import the functions from your main file

const {

    generateCategories,

    generateProduct,

    generateCustomer,

    generateTransaction,

    generateEcommerceData,

    verifyGeneratedData

} = require('./generateEcommerceData'); // Ensure you export these functions in your main file

describe('E-commerce Data Generator Tests', () => {

    // Test data directory

    const TEST\_DATA\_DIR = './test\_data';

    // Clean up test data before and after tests

    beforeEach(() => {

        if (fs.existsSync(TEST\_DATA\_DIR)) {

            const files = fs.readdirSync(TEST\_DATA\_DIR);

            files.forEach(file => {

                fs.unlinkSync(path.join(TEST\_DATA\_DIR, file));

            });

            fs.rmdirSync(TEST\_DATA\_DIR);

        }

    });

    afterAll(() => {

        if (fs.existsSync(TEST\_DATA\_DIR)) {

            const files = fs.readdirSync(TEST\_DATA\_DIR);

            files.forEach(file => {

                fs.unlinkSync(path.join(TEST\_DATA\_DIR, file));

            });

            fs.rmdirSync(TEST\_DATA\_DIR);

        }

    });

    // Test Product Generation

    describe('Product Generation', () => {

        test('generateCategories should return an array of categories', () => {

            const categories = generateCategories();

            expect(Array.isArray(categories)).toBe(true);

            expect(categories.length).toBeGreaterThan(0);

            expect(categories).toContain('Smartphones');

        });

        test('generateProduct should create valid product object', () => {

            const categories = generateCategories();

            const product = generateProduct(categories);

            expect(product).toHaveProperty('productId');

            expect(product).toHaveProperty('name');

            expect(product).toHaveProperty('price');

            expect(product.price).toHaveProperty('regular');

            expect(product.price).toHaveProperty('discounted');

            expect(product.price.discounted).toBeLessThanOrEqual(product.price.regular);

            expect(categories).toContain(product.category);

        });

    });

    // Test Customer Generation

    describe('Customer Generation', () => {

        test('generateCustomer should create valid customer object', () => {

            const customer = generateCustomer();

            expect(customer).toHaveProperty('customerId');

            expect(customer).toHaveProperty('firstName');

            expect(customer).toHaveProperty('lastName');

            expect(customer).toHaveProperty('email');

            expect(customer.email).toContain('@');

            expect(customer).toHaveProperty('address');

            expect(customer.address).toHaveProperty('street');

            expect(customer.address).toHaveProperty('city');

        });

        test('customer email should be properly formatted', () => {

            const customer = generateCustomer();

            const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

            expect(emailRegex.test(customer.email)).toBe(true);

        });

    });

    // Test Transaction Generation

    describe('Transaction Generation', () => {

        test('generateTransaction should create valid transaction object', () => {

            const categories = generateCategories();

            const products = [generateProduct(categories)];

            const customers = [generateCustomer()];

            const transaction = generateTransaction(customers, products);

            expect(transaction).toHaveProperty('transactionId');

            expect(transaction).toHaveProperty('orderId');

            expect(transaction).toHaveProperty('customerId');

            expect(transaction).toHaveProperty('items');

            expect(Array.isArray(transaction.items)).toBe(true);

        });

        test('transaction totals should be calculated correctly', () => {

            const categories = generateCategories();

            const products = [generateProduct(categories)];

            const customers = [generateCustomer()];

            const transaction = generateTransaction(customers, products);

            const calculatedSubtotal = transaction.items.reduce(

                (sum, item) => sum + (item.unitPrice \* item.quantity),

                0

            );

            expect(transaction.billing.subtotal).toBeCloseTo(calculatedSubtotal, 2);

            expect(transaction.billing.total).toBeCloseTo(

                transaction.billing.subtotal +

                transaction.billing.tax +

                transaction.billing.shippingCost,

                2

            );

        });

    });

    // Test Complete Data Generation

    describe('Complete E-commerce Data Generation', () => {

        test('generateEcommerceData should create all required files', () => {

            const summary = generateEcommerceData(5, 10, 15);

            expect(fs.existsSync(TEST\_DATA\_DIR)).toBe(true);

            expect(fs.existsSync(`${TEST\_DATA\_DIR}/products.json`)).toBe(true);

            expect(fs.existsSync(`${TEST\_DATA\_DIR}/customers.json`)).toBe(true);

            expect(fs.existsSync(`${TEST\_DATA\_DIR}/transactions.json`)).toBe(true);

            expect(fs.existsSync(`${TEST\_DATA\_DIR}/summary.json`)).toBe(true);

        });

        test('generated data should have correct counts', () => {

            const summary = generateEcommerceData(5, 10, 15);

            const products = JSON.parse(

                fs.readFileSync(`${TEST\_DATA\_DIR}/products.json`)

            );

            const customers = JSON.parse(

                fs.readFileSync(`${TEST\_DATA\_DIR}/customers.json`)

            );

            const transactions = JSON.parse(

                fs.readFileSync(`${TEST\_DATA\_DIR}/transactions.json`)

            );

            expect(products.length).toBe(5);

            expect(customers.length).toBe(10);

            expect(transactions.length).toBe(15);

        });

        test('generated data should maintain referential integrity', () => {

            generateEcommerceData(5, 10, 15);

            const products = JSON.parse(

                fs.readFileSync(`${TEST\_DATA\_DIR}/products.json`)

            );

            const customers = JSON.parse(

                fs.readFileSync(`${TEST\_DATA\_DIR}/customers.json`)

            );

            const transactions = JSON.parse(

                fs.readFileSync(`${TEST\_DATA\_DIR}/transactions.json`)

            );

            const sampleTransaction = transactions[0];

            // Verify customer reference

            const customerExists = customers.some(

                c => c.customerId === sampleTransaction.customerId

            );

            expect(customerExists).toBe(true);

            // Verify product reference

            const productExists = products.some(

                p => p.productId === sampleTransaction.items[0].productId

            );

            expect(productExists).toBe(true);

        });

    });

    // Test Data Verification

    describe('Data Verification', () => {

        test('verifyGeneratedData should run without errors', () => {

            generateEcommerceData(5, 10, 15);

            expect(() => verifyGeneratedData()).not.toThrow();

        });

    });

});

**Modify your main generateEcommerceData.js file to export the necessary functions:**

// Add at the end of your file

module.exports = {

generateCategories,

generateProduct,

generateCustomer,

generateTransaction,

generateEcommerceData,

verifyGeneratedData

};

The modified file code for the **generateEcommerceData.js** is

const { faker } = require('@faker-js/faker');

const fs = require('fs');

// Product Generation Functions

const generateCategories = () => {

    return [

        'Smartphones',

        'Tablets',

        'Accessories',

        'Wearables',

        'Audio Devices',

        'Mobile Cases',

        'Screen Protectors',

        'Chargers',

        'Power Banks',

        'Storage Devices'

    ];

};

const generateProduct = (categories) => {

    const category = faker.helpers.arrayElement(categories);

    const brand = faker.company.name();

    const regularPrice = parseFloat(faker.commerce.price({ min: 50, max: 2000 }));

    const discount = faker.number.int({ min: 0, max: 30 });

    const discountedPrice = regularPrice \* (1 - discount/100);

    return {

        productId: faker.string.uuid(),

        sku: faker.string.alphanumeric(8).toUpperCase(),

        name: `${brand} ${faker.commerce.productName()}`,

        brand: brand,

        category: category,

        description: faker.commerce.productDescription(),

        price: {

            regular: regularPrice,

            discounted: parseFloat(discountedPrice.toFixed(2)),

            discount: discount,

            currency: 'USD'

        },

        specifications: {

            color: faker.helpers.arrayElement(['Black', 'White', 'Silver', 'Gold', 'Blue']),

            weight: `${faker.number.int({ min: 100, max: 500 })}g`,

            dimensions: {

                length: `${faker.number.int({ min: 10, max: 30 })}cm`,

                width: `${faker.number.int({ min: 5, max: 15 })}cm`,

                height: `${faker.number.int({ min: 1, max: 5 })}cm`

            }

        },

        inventory: {

            inStock: faker.number.int({ min: 0, max: 100 }),

            warehouse: faker.location.city(),

            leadTime: `${faker.number.int({ min: 1, max: 10 })} days`

        },

        ratings: {

            average: faker.number.float({ min: 1, max: 5, precision: 0.1 }),

            count: faker.number.int({ min: 0, max: 1000 })

        },

        isActive: faker.datatype.boolean()

    };

};

// Customer Generation Functions

const generateCustomer = () => {

    const firstName = faker.person.firstName();

    const lastName = faker.person.lastName();

    return {

        customerId: faker.string.uuid(),

        firstName: firstName,

        lastName: lastName,

        email: faker.internet.email({ firstName, lastName }),

        phone: faker.phone.number(),

        address: {

            street: faker.location.streetAddress(),

            city: faker.location.city(),

            state: faker.location.state(),

            zipCode: faker.location.zipCode(),

            country: faker.location.country()

        },

        preferences: {

            preferredPaymentMethod: faker.helpers.arrayElement(['Credit Card', 'PayPal', 'Apple Pay', 'Google Pay']),

            notifications: faker.helpers.arrayElement(['SMS', 'Email', 'Both', 'None'])

        },

        accountCreated: faker.date.past(),

        lastLogin: faker.date.recent()

    };

};

// Transaction Generation Functions

const generateTransaction = (customers, products) => {

    const customer = faker.helpers.arrayElement(customers);

    const itemCount = faker.number.int({ min: 1, max: 5 });

    const selectedProducts = faker.helpers.arrayElements(products, itemCount);

    const items = selectedProducts.map(product => ({

        productId: product.productId,

        productName: product.name,

        quantity: faker.number.int({ min: 1, max: 5 }),

        unitPrice: product.price.discounted,

        category: product.category

    }));

    const subtotal = items.reduce((sum, item) => sum + (item.unitPrice \* item.quantity), 0);

    const taxRate = 0.08;

    const shippingCost = parseFloat(faker.commerce.price(10, 50));

    const tax = subtotal \* taxRate;

    const total = subtotal + tax + shippingCost;

    return {

        transactionId: faker.string.uuid(),

        orderId: `ORD-${faker.string.alphanumeric(8).toUpperCase()}`,

        customerId: customer.customerId,

        customerDetails: {

            name: `${customer.firstName} ${customer.lastName}`,

            email: customer.email,

            phone: customer.phone

        },

        orderDate: faker.date.recent(),

        items: items,

        billing: {

            subtotal: parseFloat(subtotal.toFixed(2)),

            tax: parseFloat(tax.toFixed(2)),

            shippingCost: shippingCost,

            total: parseFloat(total.toFixed(2)),

            currency: 'USD'

        },

        shipping: {

            address: customer.address,

            method: faker.helpers.arrayElement(['Standard', 'Express', 'Next Day']),

            trackingNumber: faker.string.alphanumeric(12).toUpperCase(),

            estimatedDelivery: faker.date.future()

        },

        status: faker.helpers.arrayElement(['PENDING', 'PROCESSING', 'SHIPPED', 'DELIVERED']),

        paymentStatus: faker.helpers.arrayElement(['PAID', 'PENDING', 'FAILED'])

    };

};

// Main Generation Function

const generateEcommerceData = (productCount = 50, customerCount = 100, transactionCount = 200) => {

    console.log('Generating ecommerce test data...');

    // Generate Products

    const categories = generateCategories();

    const products = Array.from({ length: productCount }, () => generateProduct(categories));

    // Generate Customers

    const customers = Array.from({ length: customerCount }, generateCustomer);

    // Generate Transactions

    const transactions = Array.from({ length: transactionCount }, () =>

        generateTransaction(customers, products)

    );

    // Save all data to separate files

    const dataDir = './test\_data';

    if (!fs.existsSync(dataDir)){

        fs.mkdirSync(dataDir);

    }

    fs.writeFileSync(

        `${dataDir}/products.json`,

        JSON.stringify(products, null, 2)

    );

    fs.writeFileSync(

        `${dataDir}/customers.json`,

        JSON.stringify(customers, null, 2)

    );

    fs.writeFileSync(

        `${dataDir}/transactions.json`,

        JSON.stringify(transactions, null, 2)

    );

    // Generate summary

    const summary = {

        productsGenerated: products.length,

        customersGenerated: customers.length,

        transactionsGenerated: transactions.length,

        timestamp: new Date().toISOString()

    };

    fs.writeFileSync(

        `${dataDir}/summary.json`,

        JSON.stringify(summary, null, 2)

    );

    return summary;

};

// Verification Function

const verifyGeneratedData = () => {

    try {

        const dataDir = './test\_data';

        // Read all data files

        const products = JSON.parse(fs.readFileSync(`${dataDir}/products.json`));

        const customers = JSON.parse(fs.readFileSync(`${dataDir}/customers.json`));

        const transactions = JSON.parse(fs.readFileSync(`${dataDir}/transactions.json`));

        // Verify products

        console.log('\nVerifying products...');

        console.log(`✓ ${products.length} products generated`);

        console.log(`✓ Sample product: ${products[0].name}`);

        // Verify customers

        console.log('\nVerifying customers...');

        console.log(`✓ ${customers.length} customers generated`);

        console.log(`✓ Sample customer: ${customers[0].firstName} ${customers[0].lastName}`);

        // Verify transactions

        console.log('\nVerifying transactions...');

        console.log(`✓ ${transactions.length} transactions generated`);

        console.log(`✓ Sample transaction ID: ${transactions[0].orderId}`);

        // Verify relationships

        const sampleTransaction = transactions[0];

        const customerExists = customers.some(c => c.customerId === sampleTransaction.customerId);

        const productExists = products.some(p => p.productId === sampleTransaction.items[0].productId);

        console.log('\nVerifying relationships...');

        console.log(`✓ Customer reference valid: ${customerExists}`);

        console.log(`✓ Product reference valid: ${productExists}`);

        console.log('\n✅ All verifications passed successfully!');

    } catch (error) {

        console.error('❌ Verification failed:', error.message);

    }

};

// Execute the generation and verification

const summary = generateEcommerceData();

console.log('\nGeneration Summary:', summary);

verifyGeneratedData();

// Add at the end of your file

module.exports = {

    generateCategories,

    generateProduct,

    generateCustomer,

    generateTransaction,

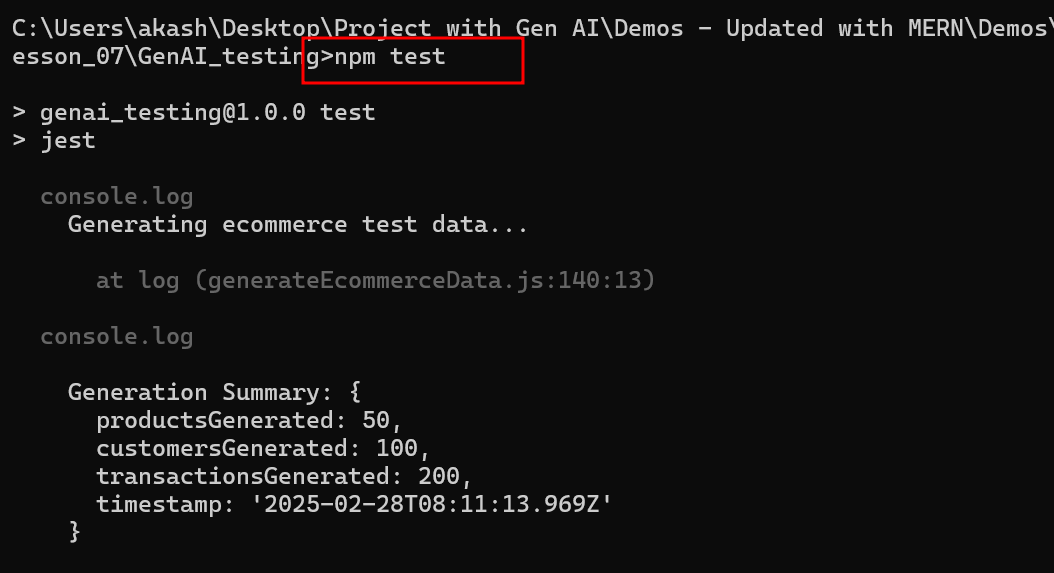
    generateEcommerceData,

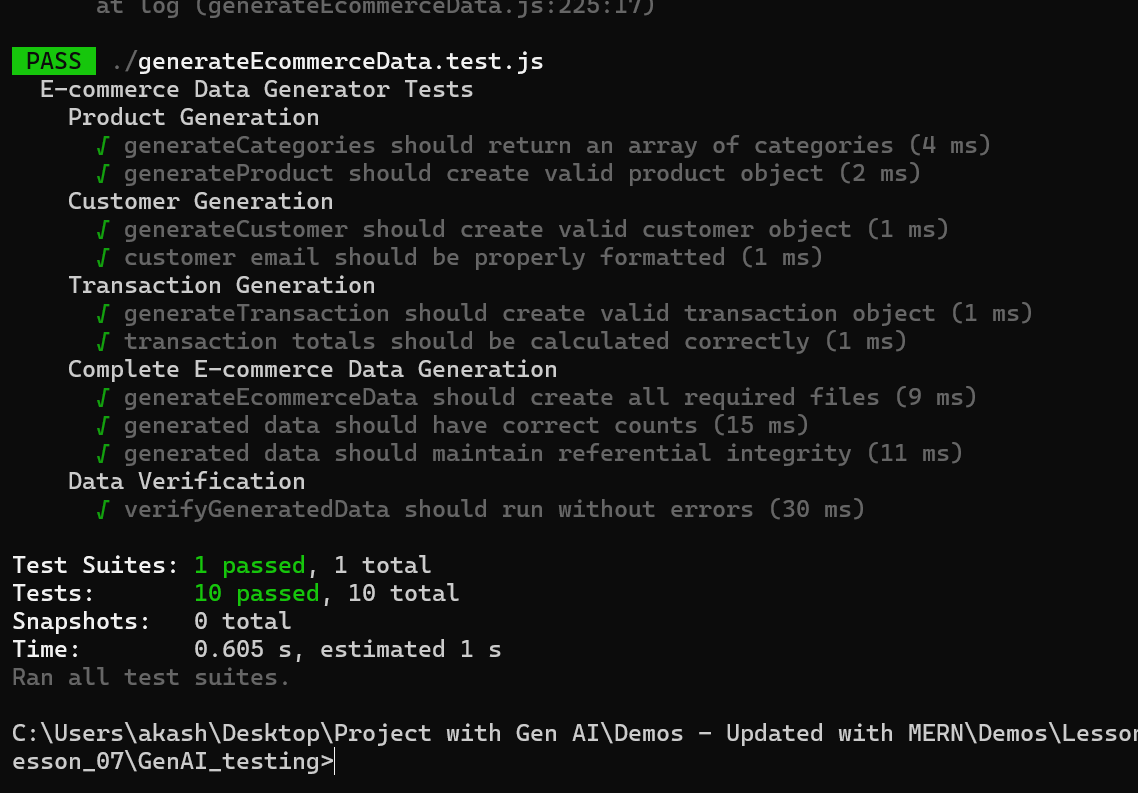
    verifyGeneratedData

};

Now runt the test case

**npm test**





**Here we done the testing using JEST testing framework with help of Amazon Q AI tool**