Complete Guide to Automated Testing in Node.js with Jest, ESLint, and GitHub Actions

Overview

This tutorial will guide you through setting up automated testing for your Node.js backend project. You'll learn how to:

- Structure your Node.js project for easy testing
- Use Jest for running tests and coverage
- Use ESLint for code linting and Prettier for formatting
- Use TypeScript for type checking (optional but recommended)
- Test across multiple Node.js versions
- Set up GitHub Actions for continuous integration

By the end, you'll have a Node.js project that automatically runs tests across multiple Node.js versions every time you push code to GitHub.

Prerequisites

- Basic JavaScript/Node.js knowledge
- Git and GitHub account
- Node.js 14+ installed
- npm or yarn package manager

Step 1: Initialize Your Node.js Project

1.1 Create Project Structure

Start by organizing your project with a clear separation between source code and tests:

```
your-node-project/
├── src/
│   ├── controllers/
│   ├── middleware/
│   ├── models/
│   ├── routes/
│   ├── services/
│   ├── utils/
│   └── app.js
├── tests/
│   ├── unit/
```

```
│   ├── integration/
│   └── fixtures/
├── package.json
└── (configuration files we'll add)
```

1.2 Initialize npm Project

```
mkdir your-node-project
cd your-node-project
npm init -y
```

1.3 Install Dependencies

```
# Production dependencies
npm install express cors helmet dotenv

# Development dependencies
npm install --save-dev jest supertest eslint prettier nodemon ts-node typescript @types/node
@types/jest @types/express
```

Step 2: Configure package.json

2.1 Update package.json Scripts

```
{
  "name": "your-node-project",
  "version": "1.0.0",
  "description": "Your Node.js backend project",
  "main": "src/app.js",
  "scripts": {
    "start": "node src/app.js",
    "dev": "nodemon src/app.js",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:coverage": "jest --coverage",
    "test:ci": "jest --coverage --watchAll=false",
    "lint": "eslint src/ tests/",
    "lint:fix": "eslint src/ tests/ --fix",
    "format": "prettier --write \"src/**/*.js\" \"tests/**/*.js\"",
    "format:check": "prettier --check \"src/**/*.js\" \"tests/**/*.js\""
  },
  "keywords": ["node", "express", "api"],
  "author": "Your Name",
  "license": "MIT",
  "engines": {
    "node": ">=14.0.0"
```

```
  }
}
```

Step 3: Set Up Testing with Jest

3.1 Create jest.config.js

```
module.exports = {
  testEnvironment: 'node',
  testMatch: ['**/tests/**/*.test.js'],
  collectCoverageFrom: [
    'src/**/*.js',
    '!src/app.js',
    '!**/node_modules/**'
  ],
  coverageDirectory: 'coverage',
  coverageReporters: ['text', 'lcov', 'html'],
  coverageThreshold: {
    global: {
      branches: 70,
      functions: 70,
      lines: 70,
      statements: 70
    }
  },
  setupFilesAfterEnv: ['<rootDir>/tests/setup.js'],
  verbose: true
};
```

3.2 Create Test Setup File

Create `tests/setup.js`:

```
// Global test setup
process.env.NODE_ENV = 'test';

// Mock console.log for cleaner test output
global.console = {
  ...console,
  log: jest.fn(),
  debug: jest.fn(),
  info: jest.fn(),
  warn: jest.fn(),
  error: jest.fn(),
};

// Common test utilities
global.testUtils = {
```

```
  delay: (ms) => new Promise(resolve => setTimeout(resolve, ms)),
  createMockReq: (overrides = {}) => ({
    body: {},
    params: {},
    query: {},
    headers: {},
     ...overrides
  }),
  createMockRes: () => {
    const res = {};
    res.status = jest.fn().mockReturnValue(res);
    res.json = jest.fn().mockReturnValue(res);
    res.send = jest.fn().mockReturnValue(res);
    res.cookie = jest.fn().mockReturnValue(res);
    return res;
  }
};
```

3.3 Write Your First Tests

Create `tests/unit/utils/math.test.js`:

```
const { add, multiply, divide } = require('../../../src/utils/math');

describe('Math Utils', () => {
  describe('add', () => {
    test('should add two positive numbers', () => {
      expect(add(2, 3)).toBe(5);
    });

    test('should handle negative numbers', () => {
      expect(add(-1, 1)).toBe(0);
    });

    test('should handle zero', () => {
      expect(add(0, 5)).toBe(5);
    });
  });

  describe('multiply', () => {
    test.each([
      [2, 3, 6],
      [0, 5, 0],
      [-2, 3, -6],
      [2.5, 4, 10]
    ])('multiply(%p, %p) should return %p', (a, b, expected) => {
      expect(multiply(a, b)).toBe(expected);
    });
  });
```

```
  describe('divide', () => {
    test('should divide two numbers', () => {
      expect(divide(10, 2)).toBe(5);
    });

    test('should throw error when dividing by zero', () => {
      expect(() => divide(10, 0)).toThrow('Division by zero');
    });

    test('should handle decimal results', () => {
      expect(divide(10, 3)).toBeCloseTo(3.333, 3);
    });
  });
});
```

Create `tests/integration/routes/health.test.js`:

```
const request = require('supertest');
const app = require('../../../src/app');

describe('Health Check Routes', () => {
  describe('GET /health', () => {
    test('should return 200 and health status', async () => {
      const response = await request(app)
        .get('/health')
        .expect(200);

      expect(response.body).toEqual({
        status: 'OK',
        timestamp: expect.any(String),
        uptime: expect.any(Number)
      });
    });
  });

  describe('GET /health/detailed', () => {
    test('should return detailed health information', async () => {
      const response = await request(app)
        .get('/health/detailed')
        .expect(200);

      expect(response.body).toHaveProperty('status');
      expect(response.body).toHaveProperty('checks');
      expect(response.body.checks).toHaveProperty('database');
      expect(response.body.checks).toHaveProperty('memory');
    });
```

```
  });
});
```

3.4 Advanced Testing Patterns

Create `tests/unit/services/userService.test.js`:

```
const UserService = require('../../../src/services/userService');
const User = require('../../../src/models/User');

// Mock the User model
jest.mock('../../../src/models/User');

describe('UserService', () => {
  let userService;

  beforeEach(() => {
    userService = new UserService();
    jest.clearAllMocks();
  });

  afterEach(() => {
    jest.resetAllMocks();
  });

  describe('createUser', () => {
    test('should create a new user successfully', async () => {
      const userData = { email: 'test@example.com', name: 'Test User' };
      const mockUser = { id: 1, ...userData };

      User.create.mockResolvedValue(mockUser);

      const result = await userService.createUser(userData);

      expect(User.create).toHaveBeenCalledWith(userData);
      expect(result).toEqual(mockUser);
    });

    test('should throw error if email already exists', async () => {
      const userData = { email: 'existing@example.com', name: 'Test User' };

      User.create.mockRejectedValue(new Error('Email already exists'));

      await expect(userService.createUser(userData))
        .rejects
        .toThrow('Email already exists');
    });

    test('should validate user data before creation', async () => {
```

```
      const invalidData = { email: 'invalid-email' };

      await expect(userService.createUser(invalidData))
        .rejects
        .toThrow('Invalid user data');

      expect(User.create).not.toHaveBeenCalled();
    });
  });

  describe('getUserById', () => {
    test('should return user if found', async () => {
      const userId = 1;
      const mockUser = { id: userId, email: 'test@example.com' };

      User.findById.mockResolvedValue(mockUser);

      const result = await userService.getUserById(userId);

      expect(User.findById).toHaveBeenCalledWith(userId);
      expect(result).toEqual(mockUser);
    });

    test('should return null if user not found', async () => {
      const userId = 999;

      User.findById.mockResolvedValue(null);

      const result = await userService.getUserById(userId);

      expect(result).toBeNull();
    });
  });
});
```

3.5 Testing with Async/Await and Promises

Create `tests/unit/services/emailService.test.js`:

```
const EmailService = require('../../../src/services/emailService');

describe('EmailService', () => {
  let emailService;

  beforeEach(() => {
    emailService = new EmailService();
  });

  describe('sendEmail', () => {
```

```
    test('should send email successfully', async () => {
      const emailData = {
        to: 'test@example.com',
        subject: 'Test',
        body: 'Test body'
      };

      // Mock external API call
      jest.spyOn(emailService, 'sendToProvider')
        .mockResolvedValue({ messageId: 'abc123' });

      const result = await emailService.sendEmail(emailData);

      expect(result).toHaveProperty('messageId');
      expect(emailService.sendToProvider)
        .toHaveBeenCalledWith(emailData);
    });

    test('should handle email sending failure', async () => {
      const emailData = { to: 'test@example.com' };

      jest.spyOn(emailService, 'sendToProvider')
        .mockRejectedValue(new Error('Provider error'));

      await expect(emailService.sendEmail(emailData))
        .rejects
        .toThrow('Failed to send email');
    });

    test('should retry failed emails', async () => {
      const emailData = { to: 'test@example.com' };

      jest.spyOn(emailService, 'sendToProvider')
        .mockRejectedValueOnce(new Error('Temporary error'))
        .mockResolvedValue({ messageId: 'abc123' });

      const result = await emailService.sendEmail(emailData);

      expect(result).toHaveProperty('messageId');
      expect(emailService.sendToProvider).toHaveBeenCalledTimes(2);
    });
  });
});
```

Step 4: Set Up ESLint and Prettier

4.1 Create .eslintrc.js

```
module.exports = {
  env: {
    node: true,
    es2021: true,
    jest: true
  },
  extends: [
    'eslint:recommended'
  ],
  parserOptions: {
    ecmaVersion: 12,
    sourceType: 'module'
  },
  rules: {
    'indent': ['error', 2],
    'linebreak-style': ['error', 'unix'],
    'quotes': ['error', 'single'],
    'semi': ['error', 'always'],
    'no-unused-vars': ['error', { 'argsIgnorePattern': '^_' }],
    'no-console': 'warn',
    'no-debugger': 'error',
    'no-trailing-spaces': 'error',
    'comma-dangle': ['error', 'never'],
    'object-curly-spacing': ['error', 'always'],
    'array-bracket-spacing': ['error', 'never'],
    'max-len': ['error', { 'code': 100 }],
    'prefer-const': 'error',
    'no-var': 'error'
  }
};
```

4.2 Create .prettierrc

```
{
  "singleQuote": true,
  "trailingComma": "none",
  "tabWidth": 2,
  "semi": true,
  "printWidth": 100,
  "bracketSpacing": true,
  "arrowParens": "avoid"
}
```

4.3 Create .eslintignore and .prettierignore

`.eslintignore`:

```
node_modules/
coverage/
dist/
build/
*.min.js
```

`.prettierignore`:

```
node_modules/
coverage/
dist/
build/
package-lock.json
*.min.js
```

Step 5: Optional TypeScript Setup

5.1 Create tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "lib": ["ES2020"],
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "resolveJsonModule": true,
    "declaration": true,
    "declarationMap": true,
    "sourceMap": true,
    "incremental": true,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "dist", "tests"]
}
```

5.2 Update package.json for TypeScript

```
{
  "scripts": {
    "build": "tsc",
    "build:watch": "tsc --watch",
    "start": "node dist/app.js",
    "start:ts": "ts-node src/app.ts",
    "dev": "nodemon --exec ts-node src/app.ts",
    "test": "jest --preset ts-jest",
    "type-check": "tsc --noEmit"
  }
}
```

5.3 Update jest.config.js for TypeScript

```
module.exports = {
  preset: 'ts-jest',
  testEnvironment: 'node',
  testMatch: ['**/tests/**/*.test.ts'],
  collectCoverageFrom: [
    'src/**/*.ts',
    '!src/**/*.d.ts',
    '!src/app.ts'
  ],
  transform: {
    '^.+\\.ts$': 'ts-jest'
  },
  moduleFileExtensions: ['ts', 'js', 'json'],
  coverageDirectory: 'coverage',
  coverageReporters: ['text', 'lcov', 'html']
};
```

Step 6: Create Sample Application Code

6.1 Create src/app.js

```
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
require('dotenv').config();

const app = express();

// Middleware
app.use(helmet());
app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

```
// Routes
app.get('/health', (req, res) => {
  res.json({
    status: 'OK',
    timestamp: new Date().toISOString(),
    uptime: process.uptime()
  });
});

app.get('/health/detailed', (req, res) => {
  res.json({
    status: 'OK',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    checks: {
      database: 'OK',
      memory: {
        used: process.memoryUsage().heapUsed,
        total: process.memoryUsage().heapTotal
      }
    }
  });
});

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Something went wrong!' });
});

// 404 handler
app.use((req, res) => {
  res.status(404).json({ error: 'Route not found' });
});

const PORT = process.env.PORT || 3000;

if (process.env.NODE_ENV !== 'test') {
  app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
  });
}

module.exports = app;
```

6.2 Create src/utils/math.js

```
const add = (a, b) => a + b;

const multiply = (a, b) => a * b;

const divide = (a, b) => {
  if (b === 0) {
    throw new Error('Division by zero');
  }
  return a / b;
};

module.exports = { add, multiply, divide };
```

Step 7: Set Up GitHub Actions

7.1 Create .github/workflows/tests.yml

```
name: Tests

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ${{ matrix.os }}

    strategy:
      matrix:
        node-version: [14.x, 16.x, 18.x, 20.x]
        os: [ubuntu-latest, windows-latest, macos-latest]

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v4
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Run linter
```

```
      run: npm run lint

    - name: Check formatting
      run: npm run format:check

    - name: Run tests
      run: npm run test:ci

    - name: Upload coverage to Codecov
      uses: codecov/codecov-action@v3
      with:
        file: ./coverage/lcov.info
        flags: unittests
        name: codecov-umbrella
        fail_ci_if_error: false

  build:
    runs-on: ubuntu-latest
    needs: test

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Use Node.js 18.x
      uses: actions/setup-node@v4
      with:
        node-version: 18.x
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Build application
      run: npm run build

    - name: Upload build artifacts
      uses: actions/upload-artifact@v3
      with:
        name: build-files
        path: dist/
```

7.2 Create .github/workflows/security.yml

```
name: Security

on:
  push:
    branches: [ main ]
```

```yaml
  pull_request:
    branches: [ main ]
  schedule:
    - cron: '0 0 * * 1'  # Weekly on Monday

jobs:
  security:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Use Node.js 18.x
      uses: actions/setup-node@v4
      with:
        node-version: 18.x
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Run security audit
      run: npm audit --audit-level moderate

    - name: Run npm audit fix
      run: npm audit fix

    - name: Check for outdated packages
      run: npm outdated || true
```

Step 8: Environment Configuration

8.1 Create .env.example

```
# Server Configuration
NODE_ENV=development
PORT=3000

# Database Configuration
DB_HOST=localhost
DB_PORT=5432
DB_NAME=your_database
DB_USER=your_user
DB_PASSWORD=your_password

# JWT Configuration
JWT_SECRET=your_super_secret_jwt_key
JWT_EXPIRES_IN=24h
```

```
# Email Configuration
EMAIL_SERVICE=gmail
EMAIL_USER=your_email@gmail.com
EMAIL_PASSWORD=your_app_password

# Rate Limiting
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100

# Logging
LOG_LEVEL=info
LOG_FILE=logs/app.log

# External APIs
THIRD_PARTY_API_KEY=your_api_key
THIRD_PARTY_API_URL=https://api.example.com
```

8.2 Create .gitignore

```
# Dependencies
node_modules/
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# Production builds
dist/
build/

# Environment variables
.env
.env.local
.env.production

# Testing
coverage/
*.lcov

# Logs
logs/
*.log

# Runtime data
pids/
*.pid
*.seed
*.pid.lock
```

```
# Coverage directory used by tools like istanbul
coverage/
*.lcov

# nyc test coverage
.nyc_output

# IDEs
.vscode/
.idea/
*.swp
*.swo

# OS generated files
.DS_Store
.DS_Store?
._*
.Spotlight-V100
.Trashes
ehthumbs.db
Thumbs.db
```

Step 9: Advanced Testing Strategies

9.1 Integration Tests with Database

Create `tests/integration/database.test.js`:

```
const { Pool } = require('pg');
const app = require('../../src/app');
const request = require('supertest');

describe('Database Integration Tests', () => {
  let pool;

  beforeAll(async () => {
    pool = new Pool({
      host: process.env.TEST_DB_HOST || 'localhost',
      port: process.env.TEST_DB_PORT || 5432,
      database: process.env.TEST_DB_NAME || 'test_db',
      user: process.env.TEST_DB_USER || 'test_user',
      password: process.env.TEST_DB_PASSWORD || 'test_password'
    });

    // Run migrations or seed test data
    await pool.query('CREATE TABLE IF NOT EXISTS users (id SERIAL PRIMARY KEY, email
VARCHAR(255))');
  });
```

```
  afterAll(async () => {
    // Clean up test data
    await pool.query('DROP TABLE IF EXISTS users');
    await pool.end();
  });

  beforeEach(async () => {
    // Clear test data before each test
    await pool.query('DELETE FROM users');
  });

  test('should create and retrieve user', async () => {
    const userData = { email: 'test@example.com' };

    // Create user via API
    const createResponse = await request(app)
      .post('/api/users')
      .send(userData)
      .expect(201);

    expect(createResponse.body).toHaveProperty('id');

    // Verify user exists in database
    const dbResult = await pool.query('SELECT * FROM users WHERE id = $1',
[createResponse.body.id]);
    expect(dbResult.rows).toHaveLength(1);
    expect(dbResult.rows[0].email).toBe(userData.email);
  });
});
```

9.2 Load Testing

Create `tests/load/basic.test.js`:

```
const request = require('supertest');
const app = require('../../src/app');

describe('Load Tests', () => {
  test('should handle multiple concurrent requests', async () => {
    const concurrentRequests = 50;
    const requests = Array(concurrentRequests).fill(null).map(() =>
      request(app).get('/health').expect(200)
    );

    const startTime = Date.now();
    await Promise.all(requests);
    const endTime = Date.now();

    const totalTime = endTime - startTime;
```

```
    const averageResponseTime = totalTime / concurrentRequests;

    expect(averageResponseTime).toBeLessThan(1000); // Should respond within 1 second on
average
  }, 30000); // 30 second timeout

  test('should maintain performance under sustained load', async () => {
    const requestsPerSecond = 10;
    const durationSeconds = 5;
    const totalRequests = requestsPerSecond * durationSeconds;

    const results = [];

    for (let i = 0; i < totalRequests; i++) {
      const startTime = Date.now();
      await request(app).get('/health').expect(200);
      const endTime = Date.now();

      results.push(endTime - startTime);

      // Maintain rate of requests per second
      await new Promise(resolve => setTimeout(resolve, 1000 / requestsPerSecond));
    }

    const averageResponseTime = results.reduce((a, b) => a + b, 0) / results.length;
    const maxResponseTime = Math.max( ... results);

    expect(averageResponseTime).toBeLessThan(500);
    expect(maxResponseTime).toBeLessThan(2000);
  }, 30000);
});
```

Step 10: Documentation and Best Practices

10.1 Create README.md

```
# Your Node.js Project

[![Tests](https://github.com/yourusername/your-node-
project/actions/workflows/tests.yml/badge.svg)](https://github.com/yourusername/your-node-
project/actions/workflows/tests.yml)
[![Coverage](https://codecov.io/gh/yourusername/your-node-
project/branch/main/graph/badge.svg)](https://codecov.io/gh/yourusername/your-node-project)
[![Security](https://github.com/yourusername/your-node-
project/actions/workflows/security.yml/badge.svg)](https://github.com/yourusername/your-node-
project/actions/workflows/security.yml)

A robust Node.js backend application with comprehensive testing and CI/CD.
```

```
## Features

- Express.js REST API
- Comprehensive testing with Jest
- Code linting with ESLint
- Code formatting with Prettier
- TypeScript support (optional)
- GitHub Actions CI/CD
- Security auditing
- Coverage reporting

## Getting Started

### Prerequisites

- Node.js 14+
- npm or yarn

### Installation

```bash
git clone https://github.com/yourusername/your-node-project.git
cd your-node-project
npm install
```
```

Environment Setup

```
cp .env.example .env
# Edit .env with your configuration
```

Running the Application

```
# Development
npm run dev

# Production
npm start

# With TypeScript
npm run start:ts
```

Testing

```
# Run all tests
npm test

# Run tests in watch mode
```

```
npm run test:watch

# Run tests with coverage
npm run test:coverage

# Run tests for CI
npm run test:ci
```

Code Quality

```
# Lint code
npm run lint

# Fix linting issues
npm run lint:fix

# Format code
npm run format

# Check formatting
npm run format:check

# Type checking (TypeScript)
npm run type-check
```

Project Structure

```
your-node-project/
├── src/
│   ├── controllers/     # Route controllers
│   ├── middleware/      # Express middleware
│   ├── models/          # Data models
│   ├── routes/          # Route definitions
│   ├── services/        # Business logic
│   ├── utils/           # Utility functions
│   └── app.js           # Application entry point
├── tests/
│   ├── unit/            # Unit tests
│   ├── integration/     # Integration tests
│   ├── load/            # Load tests
│   └── fixtures/        # Test data
├── .github/
│   └── workflows/       # GitHub Actions
├── coverage/            # Coverage reports
└── dist/                # Compiled TypeScript (if using TS)
```

API Documentation

Health Check

- `GET /health` - Basic health check
- `GET /health/detailed` - Detailed health information

Contributing

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/amazing-feature`)
3. Commit your changes (`git commit -m 'Add some amazing feature'`)
4. Push to the branch (`git push origin feature/amazing-feature`)
5. Open a Pull Request

Development Workflow

1. Write tests for new features
2. Implement the feature
3. Ensure all tests pass (`npm test`)
4. Check code quality (`npm run lint && npm run format:check`)
5. Create pull request

License

This project is licensed under the MIT License - see the LICENSE file for details.

```
### 10.2 Testing Best Practices

1. **Test Structure**: Follow the AAA pattern (Arrange, Act, Assert)
2. **Test Naming**: Use descriptive names that explain what is being tested
3. **Test Isolation**: Each test should be independent and not rely on others
4. **Mocking**: Mock external dependencies to keep tests fast and reliable
5. **Coverage**: Aim for high coverage but focus on testing critical paths
6. **Integration Tests**: Test the interaction between components
7. **Error Cases**: Test both success and failure scenarios

### 10.3 Development Workflow

1. **Daily Development**:
   - Run `npm run test:watch` for continuous testing
   - Use `npm run lint:fix` to maintain code quality
   - Check `npm run format` before committing
```

2. **Before Committing**:
   - Run `npm test` to ensure all tests pass
   - Run `npm run lint` to check for linting errors
   - Run `npm run format:check` to verify formatting

3. **Before Pushing**:
   - Ensure all tests pass with coverage
   - Review changed files
   - Write meaningful commit messages

Conclusion

You now have a complete Node.js testing setup that includes:

- Jest for comprehensive testing
- ESLint for code quality
- Prettier for consistent formatting
- GitHub Actions for CI/CD
- TypeScript support (optional)
- Security auditing
- Coverage reporting

This setup provides a solid foundation for building robust, maintainable Node.js applications with confidence in code quality and reliability.