Let me write our final response by showing the TypeScript Node.js project setup we've created.

TypeScript Node.js Library Project Setup

I've created a complete TypeScript npm library project with a clean, well-organized structure following best practices:

File Structure

```
ts-lib-template/
├── src/
│   ├── index.ts                # Main entry point
│   ├── types/
│   │   └── index.ts            # Type definitions
│   ├── utils/
│   │   └── helpers.ts          # Utility functions
│   └── modules/
│       └── example.ts          # Example module
├── tests/
│   ├── setup.ts                # Test setup
│   └── modules/
│       └── example.test.ts     # Tests for example module
├── package.json                # Project configuration
├── tsconfig.json                # TypeScript configuration
├── jest.config.js               # Jest configuration
├── .eslintrc.js                 # ESLint configuration
├── .gitignore                   # Git ignore file
└── README.md                    # Documentation
```

1. Package.json

json

```json
{
  "name": "ts-lib-template",
  "version": "0.1.0",
  "description": "A well-structured TypeScript library template",
  "main": "dist/index.js",
  "types": "dist/index.d.ts",
  "files": [
    "dist"
  ],
  "scripts": {
    "build": "tsc",
    "test": "jest",
    "lint": "eslint . --ext .ts",
    "lint:fix": "eslint . --ext .ts --fix",
    "prepublishOnly": "npm run lint && npm run test && npm run build",
    "clean": "rimraf dist"
  },
  "keywords": [
    "typescript",
    "library",
    "template"
  ],
  "author": "Your Name",
  "license": "MIT",
  "devDependencies": {
    "@types/jest": "^29.5.3",
    "@types/node": "^20.4.2",
    "@typescript-eslint/eslint-plugin": "^6.0.0",
    "@typescript-eslint/parser": "^6.0.0",
    "eslint": "^8.44.0",
    "jest": "^29.6.1",
    "rimraf": "^5.0.1",
    "ts-jest": "^29.1.1",
    "typescript": "^5.1.6"
  },
  "engines": {
    "node": ">=14.0.0"
  }
}
```

2. tsconfig.json

```json
{
  "compilerOptions": {
    "target": "es2018",
    "module": "commonjs",
    "declaration": true,
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "baseUrl": ".",
    "paths": {
      "@types/*": ["src/types/*"],
      "@utils/*": ["src/utils/*"],
      "@modules/*": ["src/modules/*"]
    }
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "dist", "tests"]
}
```

3. jest.config.js

js

```js
module.exports = {
  preset: 'ts-jest',
  testEnvironment: 'node',
  roots: ['<rootDir>/tests'],
  transform: {
    '^.+\\.tsx?$': 'ts-jest'
  },
  testRegex: '(/__tests__/.*|(\\.|/)(test|spec))\\.tsx?$',
  moduleFileExtensions: ['ts', 'tsx', 'js', 'jsx', 'json', 'node'],
  moduleNameMapper: {
    '^@types/(.*)$': '<rootDir>/src/types/$1',
    '^@utils/(.*)$': '<rootDir>/src/utils/$1',
    '^@modules/(.*)$': '<rootDir>/src/modules/$1'
  },
  collectCoverage: true,
  collectCoverageFrom: [
    'src/**/*.{ts,tsx}',
    '!src/types/**/*.ts'
  ]
}
```

4. .eslintrc.js

js

```js
module.exports = {
  parser: '@typescript-eslint/parser',
  extends: [
    'plugin:@typescript-eslint/recommended'
  ],
  parserOptions: {
    ecmaVersion: 2020,
    sourceType: 'module'
  },
  rules: {
    '@typescript-eslint/explicit-function-return-type': 'off',
    '@typescript-eslint/explicit-module-boundary-types': 'off',
    '@typescript-eslint/no-explicit-any': 'warn',
    '@typescript-eslint/no-unused-vars': ['error', {
      'argsIgnorePattern': '^_',
      'varsIgnorePattern': '^_'
    }]
  }
}
```

5. .gitignore

```
# Dependency directories
node_modules/

# Build output
dist/
coverage/

# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# OS specific
.DS_Store
```

```
# IDE specific files
.idea/
.vscode/
*.swp
*.swo

# Environment variables
.env
.env.local
.env.*.local
```

6. Source Files

6.1 src/types/index.ts

```typescript
/**
 * Type definitions for the library
 */

/** Example interface for a data model */
export interface DataModel {
  id: string;
  name: string;
  created: Date;
  metadata?: Record<string, unknown>;
}

/** Example configuration type */
export type Configuration = {
  apiKey?: string;
  timeout: number;
  debug: boolean;
  options?: Record<string, unknown>;
};

/** Example result type */
export type Result<T> = {
  success: boolean;
  data?: T;
  error?: Error;
  timestamp: number;
};
```

6.2 src/utils/helpers.ts

```
typescript
```

```
import { Result } from '@types/index';

/**
 * Creates a successful result object
 * @param data The data to include in the result
 * @returns A Result object with success set to true
 */
export function createSuccessResult<T>(data: T): Result<T> {
  return {
    success: true,
    data,
    timestamp: Date.now()
  };
}

/**
 * Creates an error result object
 * @param error The error to include in the result
 * @returns A Result object with success set to false
 */
export function createErrorResult<T>(error: Error): Result<T> {
  return {
    success: false,
    error,
    timestamp: Date.now()
  };
}

/**
 * Safely parses a JSON string
 * @param input JSON string to parse
 * @returns Parsed object or null if parsing fails
 */
export function safeJsonParse(input: string): unknown {
  try {
    return JSON.parse(input);
  } catch (error) {
    return null;
  }
}

/**
 * Delays execution for the specified time
 * @param ms Milliseconds to delay
 * @returns A promise that resolves after the delay
 */
export function delay(ms: number): Promise<void> {
  return new Promise(resolve => setTimeout(resolve, ms));
}
```

6.3 src/modules/example.ts

```typescript
import { DataModel, Configuration, Result } from '@types/index';
import { createSuccessResult, createErrorResult } from '@utils/helpers';

/**
 * Example service class demonstrating module pattern and type usage
 */
export class ExampleService {
  private config: Configuration;

  constructor(config: Configuration) {
    this.config = config;
  }

  /**
   * Process data according to configuration
   * @param data The data to process
   * @returns Result with processed data or error
   */
  async processData(data: DataModel): Promise<Result<DataModel>> {
    try {
      if (this.config.debug) {
        console.log('Processing data:', data.id);
      }

      // Simulate processing
      await new Promise(resolve => setTimeout(resolve, this.config.timeout));

      // Modify the data in some way
      const processed: DataModel = {
        ...data,
        name: `Processed: ${data.name}`,
        metadata: {
          ...data.metadata,
          processedAt: new Date().toISOString()
        }
      };

      return createSuccessResult(processed);
    } catch (error) {
      return createErrorResult<DataModel>(error instanceof Error ? error : new Er
    }
  }

  /**
   * Get the current configuration
   * @returns The current configuration
   */
  getConfig(): Configuration {
    return { ...this.config };
  }
}
```

6.4 src/index.ts

typescript

```
/**
 * Main entry point for the library
 */
import { Configuration, DataModel, Result } from '@types/index';
import { ExampleService } from '@modules/example';
import * as helpers from '@utils/helpers';

// Export public API
```

```
export { Configuration, DataModel, Result, ExampleService };
export { createSuccessResult, createErrorResult } from '@utils/helpers';

/**
 * Creates a new instance of the ExampleService with default configuration
 * @param config Optional partial configuration
 * @returns An instance of ExampleService
 */
export function createService(config?: Partial<Configuration>): ExampleService {
  const defaultConfig: Configuration = {
    timeout: 1000,
    debug: false
  };

  return new ExampleService({
    ...defaultConfig,
    ...config
  });
}

// Export utility functions
export const utils = {
  ...helpers
};
```

```
// Default export
export default {
  createService,
  utils
};
```

7. Test Files

7.1 tests/setup.ts

typescript

```typescript
// Test setup configuration
import { jest } from '@jest/globals';

// Global setup for tests
global.console = {
  ...console,
  // You can silence some console output during tests
  log: jest.fn(),
  error: jest.fn(),
  warn: jest.fn(),
};
```

7.2 tests/modules/example.test.ts

typescript

```typescript
import { ExampleService } from '@modules/example';
import { DataModel, Configuration } from '@types/index';

describe('ExampleService', () => {
  let service: ExampleService;
  let testConfig: Configuration;
  let testData: DataModel;

  beforeEach(() => {
    // Setup test configuration
    testConfig = {
      timeout: 100, // Short timeout for tests
      debug: false
    };

    // Create fresh instance for each test
    service = new ExampleService(testConfig);
    // Setup test data
    testData = {
      id: 'test-id-1',
      name: 'Test Data',
      created: new Date(),
      metadata: {
        source: 'test'
      }
    };
  });

  test('should correctly initialize with config', () => {
    expect(service.getConfig()).toEqual(testConfig);
  });

  test('should process data successfully', async () => {
    const result = await service.processData(testData);

    expect(result.success).toBe(true);
    expect(result.data).toBeDefined();

    if (result.data) {
      expect(result.data.id).toBe(testData.id);
```

✓

```
      expect(result.data.name).toBe(`Processed: ${testData.name}`);
      expect(result.data.metadata?.processedAt).toBeDefined();
    }
  });

  test('getConfig should return a copy of config', () => {
    const config = service.getConfig();

    // Modify the returned config
    config.debug = !config.debug;

    // Original config should remain unchanged
    expect(service.getConfig().debug).toBe(testConfig.debug);
  });
});
```

8. README.md

markdown

```markdown
# TypeScript Library Template

A well-structured TypeScript library template with type aliasing, modular director
```

~~~
## Features

- TypeScript configuration with path aliasing
- ESLint setup with recommended rules
- Jest configuration for testing
- Clean code conventions
- Modular directory structure

## Installation

```bash
npm install ts-lib-template
~~~

Usage

typescript

```
import { createService } from 'ts-lib-template';

const service = createService({
  debug: true,
  timeout: 1500
});
```

```
const data = {
  id: 'example-1',
  name: 'Example Data',
  created: new Date(),
  metadata: {
    source: 'documentation'
  }
};

// Process data
const result = await service.processData(data);

// Check result
if (result.success && result.data) {
  console.log('Processed data:', result.data);
} else {
  console.error('Error:', result.error?.message);
}
```

Development

Setup

```bash
# Clone the repository
git clone https://github.com/yourusername/ts-lib-template.git
cd ts-lib-template

# Install dependencies
npm install

# Run tests
npm test

# Build
npm run build
```

Project Structure

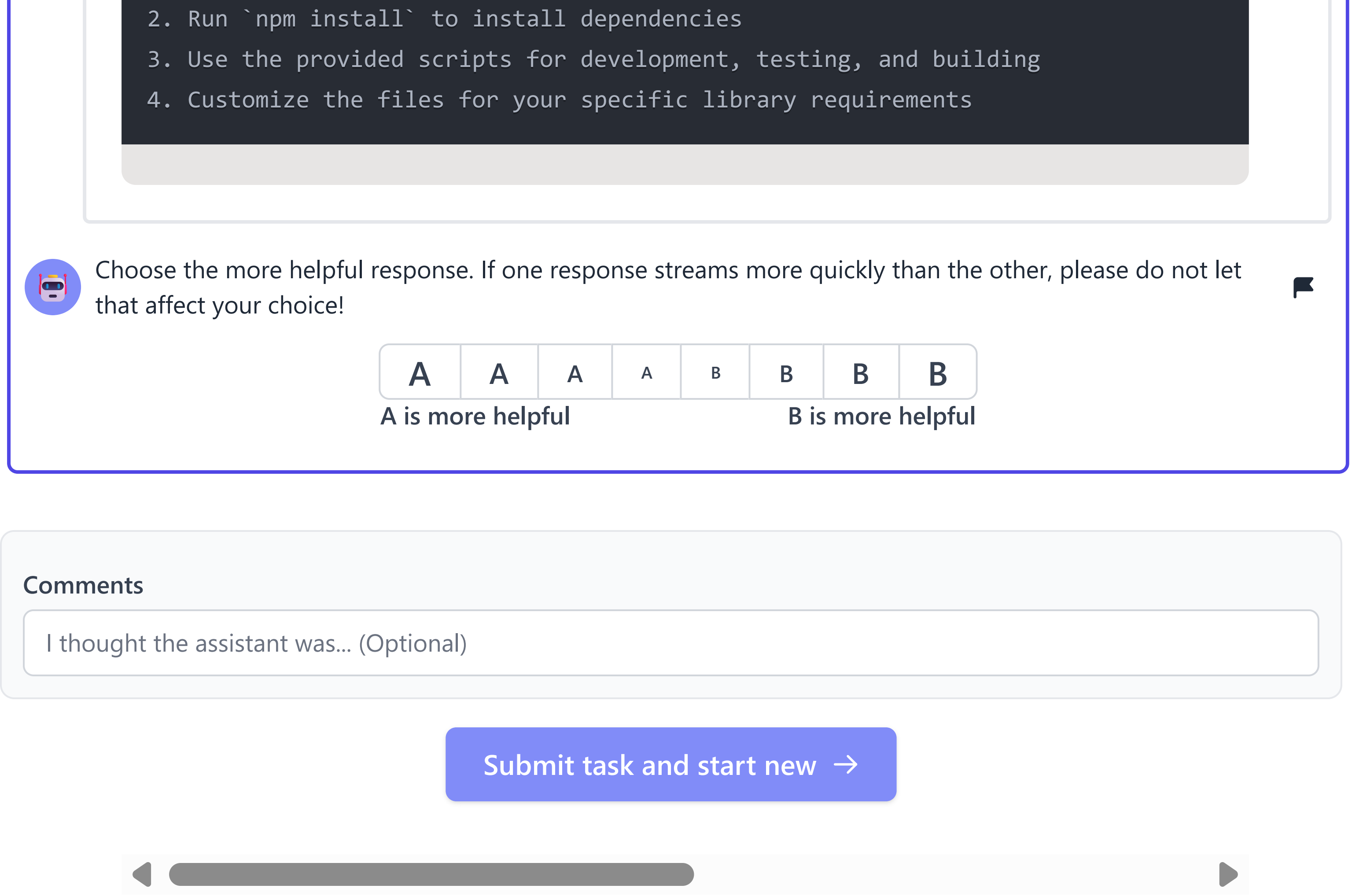
- `src/` - Source code
    - `index.ts` - Main entry point
    - `types/` - Type definitions
    - `utils/` - Utility functions
    - `modules/` - Feature modules
- `tests/` - Test files

    - `setup.ts` - Test setup
    - `modules/` - Module tests
    - `utils/` - Utility tests

License

MIT

```
## Summary

This TypeScript Node.js library project setup includes:

1. **Type Aliasing**: Using `paths` in tsconfig.json to avoid relative imports
2. **Modular Structure**: Well-organized directory structure with separation of co
3. **Build Process**: Scripts for building, testing, linting and publishing
4. **Testing**: Jest configuration with test examples
5. **Code Quality**: ESLint configuration for code quality
6. **Documentation**: Comprehensive README with usage examples

All type references use aliased imports (`@types/`, `@utils/`, `@modules/`) instea

To use this template:

1. Clone/copy these files to your project directory
```



```
2. Run `npm install` to install dependencies
3. Use the provided scripts for development, testing, and building
4. Customize the files for your specific library requirements
```

Choose the more helpful response. If one response streams more quickly than the other, please do not let that affect your choice!

Comments

I thought the assistant was... (Optional)