Let's start by providing well-formatted index.ts files for each directory:

1. src/types/index.ts

typescript

```typescript
// Type definitions for the library

// Example of a utility type
export type Nullable<T> = T | null;

// Config interface
export interface Config {
  apiKey: string;
  timeout: number;
  baseUrl: string;
}

// Data model interfaces
export interface User {
  id: string;
  name: string;
  email: string;
  role: UserRole;
  createdAt: Date;
```

```
}

// Enum example
export enum UserRole {
  ADMIN = 'admin',
  USER = 'user',
  GUEST = 'guest'
}

// Function type definitions
export type Handler<T, R> = (data: T) => Promise<R>;
export type ErrorHandler = (error: Error) => void;
```

2. src/constants/index.ts

typescript

```
// Central place for constants

import { Config } from '@types/index';

export const DEFAULT_CONFIG: Config = {
  apiKey: 'default-key',
  timeout: 3000,
```

```
  baseUrl: 'https://api.example.com'
};

export const HTTP_STATUS = {
  OK: 200,
  CREATED: 201,
  BAD_REQUEST: 400,
  UNAUTHORIZED: 401,
  FORBIDDEN: 403,
  NOT_FOUND: 404,
  INTERNAL_SERVER_ERROR: 500
};

export const API_ENDPOINTS = {
  USERS: '/users',
  AUTH: '/auth',
  PRODUCTS: '/products'
};
```

3. src/utils/index.ts

```
typescript
```

```typescript
// Utility functions

import { ErrorHandler } from '@types/index';

/**
 * Safely parses a JSON string
 */
export const safeJsonParse = <T>(text: string, fallback: T): T => {
  try {
    return JSON.parse(text) as T;
  } catch (error) {
    return fallback;
  }
};

/**
 * Delays execution for specified milliseconds
 */
export const delay = (ms: number): Promise<void> =>
  new Promise(resolve => setTimeout(resolve, ms));

/**
 * Creates a debounced function
 */
```

```typescript
export const debounce = <F extends (...args: any[]) => any>(
  func: F,
  waitFor: number
): ((...args: Parameters<F>) => void) => {
  let timeout: ReturnType<typeof setTimeout> | null = null;

  return (...args: Parameters<F>): void => {
    if (timeout !== null) {
      clearTimeout(timeout);
    }

    timeout = setTimeout(() => func(...args), waitFor);
  };
};

/**
 * Handles errors consistently across the application
 */
export const createErrorHandler = (
  customHandler?: ErrorHandler
): ErrorHandler => {
  return (error: Error): void => {
    console.error(`Error occurred: ${error.message}`);
```

```
    if (customHandler) {
      customHandler(error);
    }
  };
};
```

4. src/services/index.ts

typescript

```typescript
// Service implementations

import { Config, User, UserRole } from '@types/index';
import { DEFAULT_CONFIG, API_ENDPOINTS } from '@constants/index';
import { safeJsonParse, delay, createErrorHandler } from '@utils/index';

/**
 * API service for handling user operations
 */
export class UserService {
  private config: Config;
  private errorHandler: (error: Error) => void;

  /**
```

```
   * Creates a new UserService
   */
  constructor(config: Partial<Config> = {}) {
    this.config = { ...DEFAULT_CONFIG, ...config };
    this.errorHandler = createErrorHandler();
  }

  /**
   * Fetches a user by ID
   */
  async getUserById(id: string): Promise<User | null> {
    try {
      // Simulating API call
      await delay(100);

      // Mock implementation for demonstration
      return {
        id,
        name: 'John Doe',
        email: 'john.doe@example.com',
        role: UserRole.USER,
        createdAt: new Date()
      };
    } catch (error) {
```

```
      this.errorHandler(error as Error);
      return null;
    }
  }

  /**
   * Updates a user in the system
   */
  async updateUser(user: Partial<User> & { id: string }): Promise<boolean> {
    try {
      // Simulating API call
      await delay(100);

      // In a real implementation, we would make a PUT request
      console.log(`User updated: ${user.id}`);
      return true;
    } catch (error) {
      this.errorHandler(error as Error);
      return false;
    }
  }
}
```

5. src/index.ts (Main entry point)

typescript

```typescript
// Main library entry point

// Re-export types, constants, utils and services
export * from '@types/index';
export * from '@constants/index';
export * from '@utils/index';
export * from '@services/index';

// Export a default configuration function
import { Config } from '@types/index';
import { DEFAULT_CONFIG } from '@constants/index';
import { UserService } from '@services/index';

/**
 * Configure and return an instance of the library
 */
export function createLibrary(config: Partial<Config> = {}): {
  config: Config;
  userService: UserService;
} {
  const mergedConfig: Config = { ...DEFAULT_CONFIG, ...config };
```

```
  return {
    config: mergedConfig,
    userService: new UserService(mergedConfig)
  };
}

// Default export
export default createLibrary;
```

6. tests/utils.test.ts

typescript

```
// Tests for utility functions
import { safeJsonParse, delay, debounce } from '@utils/index';

describe('Utility Functions', () => {
  describe('safeJsonParse', () => {
    it('should correctly parse valid JSON', () => {
      // Arrange
      const jsonString = '{"name":"John","age":30}';
      const fallback = { name: '', age: 0 };

      // Act
```

```
    const result = safeJsonParse(jsonString, fallback);

    // Assert
    expect(result).toEqual({ name: 'John', age: 30 });
  });

  it('should return fallback for invalid JSON', () => {
    // Arrange
    const invalidJsonString = '{"name":"John",age:30}'; // Missing quotes aroun
    const fallback = { name: '', age: 0 };

    // Act
    const result = safeJsonParse(invalidJsonString, fallback);

    // Assert
    expect(result).toBe(fallback);
  });
});

describe('delay', () => {
  it('should resolve after specified time', async () => {
    // Arrange
    const start = Date.now();
    const delayMs = 100;
```

```
    // Act
    await delay(delayMs);
    const elapsed = Date.now() - start;

    // Assert
    expect(elapsed).toBeGreaterThanOrEqual(delayMs - 5); // Allow small timing
  });
});

describe('debounce', () => {
  beforeEach(() => {
    jest.useFakeTimers();
  });

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

  it('should debounce function calls', () => {
    // Arrange
    const mockFn = jest.fn();
    const debouncedFn = debounce(mockFn, 100);
```

```
      // Act
      debouncedFn();
      debouncedFn();
      debouncedFn();

      // Assert
      expect(mockFn).not.toHaveBeenCalled();

      // Fast-forward time
      jest.advanceTimersByTime(100);

      // Assert
      expect(mockFn).toHaveBeenCalledTimes(1);
    });
  });
});
```

A

7. tests/services.test.ts

typescript

```
// Tests for service implementations
import { UserService } from '@services/index';
```

```
import { UserRole } from '@types/index';
import * as utils from '@utils/index';

describe('UserService', () => {
  // Mock the delay function to make tests faster
  jest.spyOn(utils, 'delay').mockImplementation(() => Promise.resolve());

  describe('getUserById', () => {
    it('should return a user with the given id', async () => {
      // Arrange
      const service = new UserService();
      const userId = '123';

      // Act
      const user = await service.getUserById(userId);

      // Assert
      expect(user).not.toBeNull();
      expect(user?.id).toBe(userId);
      expect(user?.role).toBe(UserRole.USER);
    });

    it('should return null when an error occurs', async () => {
      // Arrange
```

```
    const service = new UserService();

    // Mock delay to throw an error
    jest.spyOn(utils, 'delay').mockImplementationOnce(() => {
      throw new Error('Network error');
    });

    // Act
    const user = await service.getUserById('123');

    // Assert
    expect(user).toBeNull();
  });
});

describe('updateUser', () => {
  it('should return true when update is successful', async () => {
    // Arrange
    const service = new UserService();
    const user = {
      id: '123',
      name: 'Updated Name'
    };
```

```
    // Spy on console.log
    const consoleSpy = jest.spyOn(console, 'log');

    // Act
    const result = await service.updateUser(user);

    // Assert
    expect(result).toBe(true);
    expect(consoleSpy).toHaveBeenCalledWith(`User updated: ${user.id}`);

    // Cleanup
    consoleSpy.mockRestore();
  });

  it('should return false when an error occurs', async () => {
    // Arrange
    const service = new UserService();

    // Mock delay to throw an error
    jest.spyOn(utils, 'delay').mockImplementationOnce(() => {
      throw new Error('Network error');
    });

    // Act
```

```
      const result = await service.updateUser({ id: '123', name: 'Test' });

      // Assert
      expect(result).toBe(false);
    });
  });
});
```

8. tests/index.test.ts

typescript

```
// Tests for the main library entry point
import createLibrary from '../src';
import { DEFAULT_CONFIG } from '@constants/index';

describe('Library Entry Point', () => {
  describe('createLibrary', () => {
    it('should create a library instance with default config', () => {
      // Act
      const library = createLibrary();

      // Assert
      expect(library.config).toEqual(DEFAULT_CONFIG);
```

```
  expect(library.userService).toBeDefined();
});

it('should override default config with provided values', () => {
  // Arrange
  const customConfig = {
    apiKey: 'custom-key',
    timeout: 5000
  };

  // Act
  const library = createLibrary(customConfig);

  // Assert
  expect(library.config).toEqual({
    ...DEFAULT_CONFIG,
    ...customConfig
  });
});

it('should initialize UserService with the correct config', () => {
  // Arrange
  const customConfig = {
    apiKey: 'custom-key'
```

```
      };

      // Act
      const library = createLibrary(customConfig);

      // Assert - checking private property with type assertion
      const userServiceConfig = (library.userService as any).config;
      expect(userServiceConfig.apiKey).toBe(customConfig.apiKey);
    });
  });
});
```

9. Configuration Files

.eslintrc.js

javascript

```
module.exports = {
  parser: '@typescript-eslint/parser',
  parserOptions: {
    project: 'tsconfig.json',
    sourceType: 'module',
  },
  plugins: ['@typescript-eslint/eslint-plugin'],
```

```
  extends: [
    'plugin:@typescript-eslint/recommended',
  ],
  root: true,
  env: {
    node: true,
    jest: true,
  },
  ignorePatterns: ['.eslintrc.js', 'dist/', 'node_modules/'],
  rules: {
    '@typescript-eslint/interface-name-prefix': 'off',
    '@typescript-eslint/explicit-function-return-type': 'warn',
    '@typescript-eslint/explicit-module-boundary-types': 'warn',
    '@typescript-eslint/no-explicit-any': 'off',
    '@typescript-eslint/no-unused-vars': ['warn', { 'argsIgnorePattern': '^_' }],
    'quotes': ['error', 'single', { 'allowTemplateLiterals': true }],
    'semi': ['error', 'always']
  },
};
```

tsconfig.json

```
json
```

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

jest.config.js

javascript

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

```
    global: {
      branches: 80,
      functions: 80,
      lines: 80,
      statements: 80
    }
  }
};
```

package.json

json

```
{
  "name": "ts-library",
  "version": "1.0.0",
  "description": "A well-organized TypeScript library",
  "main": "dist/index.js",
  "types": "dist/index.d.ts",
  "files": [
    "dist"
  ],
  "scripts": {
    "build": "rimraf dist && tsc",
```

```
    "test": "jest",
    "test:watch": "jest --watch",
    "lint": "eslint --ext .ts src/",
    "lint:fix": "eslint --ext .ts src/ --fix",
    "prepare": "npm run build",
    "prepublishOnly": "npm test && npm run lint",
    "preversion": "npm run lint",
    "version": "npm run lint:fix && git add -A src",
    "postversion": "git push && git push --tags"
  },
  "keywords": [
    "typescript",
    "library",
    "node",
    "boilerplate"
  ],
  "author": "Your Name",
  "license": "MIT",
  "devDependencies": {
    "@types/jest": "^29.5.3",
    "@types/node": "^20.4.5",
    "@typescript-eslint/eslint-plugin": "^6.2.0",
    "@typescript-eslint/parser": "^6.2.0",
    "eslint": "^8.45.0",
```

```
    "jest": "^29.6.2",
    "rimraf": "^5.0.1",
    "ts-jest": "^29.1.1",
    "typescript": "^5.1.6"
  },
  "peerDependencies": {},
  "engines": {
    "node": ">=14.0.0"
  }
}
```

Getting Started

To use this project structure, follow these steps:

1. Create the directory structure:

```
mkdir -p ts-library/src/{types,constants,utils,services} ts-library/tests
```

2. Copy the code snippets above into their respective files.

3. Install dependencies:

bash

```
cd ts-library
npm init -y  # Initialize package.json (or use the one provided above)
```

```
npm install --save-dev typescript @types/node @types/jest eslint @typescript-eslin
```

4. Run the tests:

```bash
npm test
```

5. Build the project:

```bash
npm run build
```

This clean, well-organized structure provides a strong foundation for your TypeScript library with proper path aliasing and modular architecture. All code is formatted consistently and follows best practices for TypeScript development.

You're absolutely right, well-formatted code is essential for readability and ease of use. I'll provide the complete structure with clean, formatted files for each directory, focusing especially on having an `index.ts` for each folder.

Let me create a cleaner, more structured version: