

# How to setup DynamoDB using CDK & Setup Basic CRUD system with Lambda and Typescript



Jack Do  
77 subscribers

Subscribed

Like 4

Dislike

Share

Download

78 views Jul 8, 2022 AWS Cloud Development Kit (CDK) crash course

Hello guys,

In this video, we will go through:

- ◆ Setting Dynamo and update lambda IAM role with CDK
- ◆ Updating backend codebase with some programming principles
- ◆ Separation of concerns
- ◆ Single responsibility
- ◆ Inversion of control

#typescript #cdk #aws #dynamo #lambda #postman #dependencyinjection #serverless #infrastructureascode

Timestamps:

- 00:00 Introduction and recap
- 01:00 Update the CDK code for Dynamo and Lambda
- 04:25 Go through the CDK code and deploy to AWS
- 07:35 Start coding the backend system
- 36:35 Folder structure walk through & Design Principles explain
- 45:58 Testing with Postman & wrap up

https://github.com/jackdo69/todo-app-cloud-native/tree/tutorial\_4

The screenshot shows a GitHub repository page for 'todo-app-cloud-native'. The repository is public and has 1 watch, 0 forks, and 1 star. The 'tutorial\_4' branch is selected, showing 5 commits ahead of 'tutorial\_2'. The branch was last updated on Jul 13, 2022, with 8 commits. The commit history includes: 'jackdo69 final code' (7b098e2), 'backend/lambda final code' (last year), 'bin final code' (last year), 'lib final code' (last year), 'postman final code' (last year), 'test Initial commit' (last year), '.gitignore Initial commit' (last year), '.npmignore Initial commit' (last year), 'README.md Initial commit' (last year), 'cdk.json Initial commit' (last year), 'jest.config.js Initial commit' (last year), 'package-lock.json final code' (last year), and 'package.json final code' (last year). The repository has no releases, packages, or languages listed.

The screenshot shows the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Window', 'Help', and 'Postman'. On the left, there's a sidebar with 'Tutorial', 'Collections' (Todo App), 'APIs' (Todo App), 'Environments', 'Mock Servers', 'Monitors', 'Logs', and 'History'. The main workspace has tabs for 'POST Login' and 'GET Health'. A search bar at the top right says 'Search Postman'. The 'DEV' environment is selected. Below the tabs, there are sections for 'Params', 'Authorization', 'Headers (12)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab contains a POST request to 'https://cognito-idp.ap-southeast-2.amazonaws.com'. The 'Params' section includes environment variables for client ID, username, password, and auth flow. The 'Body' section contains a JavaScript pre-request script that sets the 'requestBody' to JSON.stringify of a body object. The 'Body' object includes 'AuthParameters' (username and password), 'AuthFlow' ('USER\_PASSWORD\_AUTH'), and 'ClientId' (client ID). The 'Tests' tab shows a single test: 'pm.environment.set("requestBody", JSON.stringify(body));'. The 'Body' tab also displays the response from the API, which is a JSON object containing an access token, expiration information, and refresh token. The status bar at the bottom indicates a 200 OK response with a time of 431 ms and a size of 4.23 KB.

The screenshot shows the Postman application interface. The left sidebar contains navigation links: Home, Workspaces, API Network, Reports, Explore, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main workspace displays a collection named 'Todo App' with a single endpoint: 'GET /Health'. The 'Headers' tab is selected, showing the following configuration:

Key	Value	Description
Authorization	Bearer eyJraWQiOiJcLzVJNWhakpyZjQ4aWE2bnF6elM1NkwVGFT0ZkWU1R...	
Cache-Control	no-cache	
Postman-Token	<calculated when request is sent>	
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.28.0	
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	

At the bottom, there are tabs for Body, Cookies, Headers (11), and Test Results, with 'Pretty' selected. The status bar indicates: Status: 200 OK, Time: 177 ms, Size: 488 B, and Save Response.

Let us see how to setup the DynamoDB backend using the CDK. We need to get a token before calling the /health API

The screenshot shows a code editor interface with a top navigation bar for 'Code', 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', 'Window', and 'Help'. The date 'Wed 6 Jul 18:53' is also visible. The main area has a dark theme with a light sidebar. The sidebar on the left is titled 'EXPLORER' and shows the 'UNTITLED (WORKSPACE)' folder expanded. Inside are 'todo-app' (with 'backend' and 'bin' subfolders), 'cdk.out', and 'lib' (containing 'Apigateway.ts', 'Cognito.ts', 'Dynamo.ts' (which is selected and highlighted in blue), 'Lambda.ts', and 'todo-app-stack.ts'). Below these are '.gitignore', '.prettierrc.js', 'cdk.json', 'jest.config.js', 'package-lock.json', 'package.json', 'README.md', and 'tsconfig.json'. The main code editor window is titled 'Dynamo.ts - Untitled (Workspace)'. It contains the following TypeScript code:

```
① todo-app > lib > Dynamo.ts > Dynamo > constructor
  4 import { AttributeType, Table } from "aws-cdk-lib/aws-dynamodb";
  3 import { Construct } from "constructs";
  2
  1 export class Dynamo extends Table {
  5   constructor(scope: Construct) {
  1     super(scope, "DynamoDB", {
  2       tableName: "to-do_db",
  3       partitionKey: { name: "type", type: AttributeType.STRING },
  4       sortKey: { name: "id", type: AttributeType.STRING }
  5     });
  6   }
  7 }
  8
```

Dynamo.ts — Untitled (Workspace)

```
① Dynamo.ts U ② Lambda.ts M ③ todo-app-stack.ts M ④ health.ts ⑤ createTodo.ts U ⑥ getTodo.ts U ⑦ updateTodo.ts U
```

EXPLORER

- todo-app
  - backend
  - lambda
    - createTodo.ts
    - deleteTodos.ts
    - getTodos.ts
    - health.ts
    - updateTodo.ts
  - models
  - repositories
  - services
  - bin
  - cdk.out
- lib
  - ApiGateway.ts
  - Cognito.ts
  - Dynamo.ts
  - Lambda.ts
  - todo-app-stack.ts
- node\_modules
- postman
- test
  - .gitignore
  - .npmignore
  - .prettierrc.js
  - cdk.json
  - jest.config.js
  - package-lock.json
  - package.json
  - README.md
  - tsconfig.json

Lambda.ts — Untitled (Workspace)

```
① Dynamo.ts U ② Lambda.ts X
```

EXPLORER

- todo-app
  - backend
  - bin
  - cdk.out
  - lib
    - ApiGateway.ts
    - Cognito.ts
    - Dynamo.ts
    - Lambda.ts
    - todo-app-stack.ts
  - node\_modules
  - postman
  - test
    - .gitignore
    - .npmignore
    - .prettierrc.js
    - cdk.json
    - jest.config.js
    - package-lock.json
    - package.json
    - README.md
    - tsconfig.json

Lambda.ts — Untitled (Workspace)

```
① Dynamo.ts U ② Lambda.ts M X
```

EXPLORER

- todo-app
  - backend
  - bin
  - cdk.out
  - lib
    - ApiGateway.ts
    - Cognito.ts
    - Dynamo.ts
    - Lambda.ts
    - todo-app-stack.ts
  - node\_modules
  - postman
  - test
    - .gitignore
    - .npmignore
    - .prettierrc.js
    - cdk.json
    - jest.config.js
    - package-lock.json
    - package.json
    - README.md
    - tsconfig.json

A screenshot of the Visual Studio Code interface. The title bar shows "todo-app-stack.ts - Untitled (Workspace)". The Explorer sidebar on the left shows a workspace named "UNTITLED (WORKSPACE)" containing a "todo-app" folder with subfolders like "backend", "bin", "cdk.out", and "lib". Inside "lib", there are files for "ApiGateway.ts", "Cognito.ts", "Dynamo.ts", and "Lambda.ts". A "todo-app-stack.ts" file is open in the editor, showing code for a CDK stack. The code imports from "aws-cdk-lib", "constructs", "Apigateway", and "Dynamo". It defines a "TodoAppStack" class extending "Stack" and sets up an API gateway endpoint for "/health".

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Lambda } from './Lambda';
import { Construct } from 'constructs';
import { Apigateway } from './Apigateway';
import { Dynamo } from './Dynamo';

export class TodoAppStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);

    // Api Gateway setup
    const api = new Apigateway(this);

    // Lambdas setup
    const healthLambda = new Lambda(this, "health");

    // Database
    new Dynamo(this);

    api.addIntegration("GET", "/health", healthLambda);
  }
}
```

A screenshot of the Visual Studio Code interface. The title bar shows "todo-app-stack.ts - Untitled (Workspace)". The Explorer sidebar shows the same workspace structure as the first screenshot. The "todo-app-stack.ts" file is open, showing the same code as before, but with a timestamp "You, 1 second ago | author (You)" at the top of the code block. The status bar at the bottom right indicates "You, 1 second ago + Uncommitted changes".

```
You, 1 second ago | author (You)
import { Stack, StackProps } from 'aws-cdk-lib';
import { Lambda } from './Lambda';
import { Construct } from 'constructs';
import { Apigateway } from './Apigateway';
import { Dynamo } from './Dynamo';

export class TodoAppStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);

    // Api Gateway setup
    const api = new Apigateway(this);

    // Lambdas setup
    const healthLambda = new Lambda(this, "health");
    const createTodo = new Lambda(this, "createTodo");
    const getTodo = new Lambda(this, "getTodo");
    const updateTodo = new Lambda(this, "updateTodo");
    const deleteTodo = new Lambda(this, "deleteTodo");

    // Database
    new Dynamo(this);

    api.addIntegration("GET", "/health", healthLambda);
    api.addIntegration("POST", "/todo", createTodo);
    api.addIntegration("GET", "/todo", getTodo);
    api.addIntegration("POST", "/todo/{id}", updateTodo);
    api.addIntegration("DELETE", "/todo/{id}", deleteTodo);
  }
}
```

A screenshot of the Visual Studio Code interface. The title bar shows "health.ts - Untitled (Workspace)". The Explorer sidebar shows the workspace structure. The "health.ts" file is open in the editor, showing a lambda function handler. The code imports "APIGatewayEvent" and "APIGatewayProxyResult" from "aws-lambda". It defines a "handler" function that logs "health lambda executed" and returns a response with status code 200.

```
import { APIGatewayEvent, APIGatewayProxyResult } from 'aws-lambda';
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
  console.log("health lambda executed", event);

  return {
    body: "OK",
    statusCode: 200
  }
}
```

A screenshot of the Visual Studio Code interface. The title bar shows "createTodo.ts - Untitled (Workspace)". The Explorer sidebar shows the workspace structure. Multiple lambda files are open in the editor: "createTodo.ts", "getTodo.ts", "updateTodo.ts", and "deleteTodo.ts". The "createTodo.ts" file is active, showing a lambda function "handler" that logs "createTodo lambda executed" and returns a response with status code 200.

```
import { APIGatewayEvent, APIGatewayProxyResult } from 'aws-lambda';
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
  console.log("createTodo lambda executed", event);

  return {
    body: "createTodo called",
    statusCode: 200
  }
}
```

Code File Edit Selection View Go Run Terminal Window Help

Dynamo.ts U Lambda.ts M todo-app-stack.ts M health.ts createTodo.ts U getTodo.ts U updateTodo.ts U deleteTodo.ts U

```
getTodo.ts - Untitled (Workspace)
todo-app > backend > Lambda > getTodo.ts > [e] handler > ↴ body
6 import { APIGatewayEvent, APIGatewayProxyResult } from 'aws-lambda';
5
4 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
3 console.log("getTodo lambda executed", event);
2
1 return [
body: "getTodo called",
statusCode: 200
];
}
```

Code File Edit Selection View Go Run Terminal Window Help

Dynamo.ts U Lambda.ts M todo-app-stack.ts M health.ts createTodo.ts U getTodo.ts U updateTodo.ts U deleteTodo.ts U

```
updateTodo.ts - Untitled (Workspace)
todo-app > backend > Lambda > updateTodos > [e] handler > ↴ body
6 import { APIGatewayEvent, APIGatewayProxyResult } from 'aws-lambda';
5
4 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
3 console.log("updateTodo lambda executed", event);
2
1 return [
body: "updateTodo called",
statusCode: 200
];
}
```

Code File Edit Selection View Go Run Terminal Window Help

Dynamo.ts U Lambda.ts M todo-app-stack.ts M health.ts createTodo.ts U getTodo.ts U updateTodo.ts U deleteTodo.ts U

```
deleteTodo.ts - Untitled (Workspace)
todo-app > backend > Lambda > deleteTodos > [e] handler > ↴ body
6 import { APIGatewayEvent, APIGatewayProxyResult } from 'aws-lambda';
5
4 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
3 console.log("delete Todo lambda executed", event);
2
1 return [
body: "deleteTodo called",
statusCode: 200
];
}
```

PROBLEMS OUTPUT TERMINAL GITLENS DEBUG CONSOLE

```
~/source-code/youtube/todo-app > tutorial_4 touch lib/Dynamo.ts
~/source-code/youtube/todo-app > tutorial_4 71 cdk deploy
```

Finder File Edit View Go Window Help

PROBLEMS OUTPUT TERMINAL GITLENS DEBUG CONSOLE

deleteTodo.ts - Untitled (Workspace)

EXPLORER PROBLEMS OUTPUT TERMINAL GITLENS DEBUG CONSOLE

(NOTE: There may be security-related changes not in this list. See <https://github.com/aws/aws-cdk/issues/1299>)

Do you wish to deploy these changes (y/n)? y

TodoAppStack: deploying...

[0x] start: Publishing d09999508263ad6f343b1c258b9c2272a61c9916d2d5959a6bcd2e5ff05542:current\_account-current\_region

[0x] start: Publishing 558dc75fbd793cde2f64f9502d7ed2cb820c727ec7eb0b0665a1d1f0103992c:current\_account-current\_region

[0x] start: Publishing 0bf1f356ddc2954fd727dc214ebfb19023f84f1992901a1d9b04c1cF093:current\_account-current\_region

[0x] start: Publishing 8bf1b08ec309464dd47af5ad7f149c59cd4724513937c5f7c82c0cF593:current\_account-current\_region

[14x] success: Published d09999508263ad6f343b1c258b9c2272a61c9916d2d5959a6bcd2e5ff05542:current\_account-current\_region

[28x] success: Published 558dc75fbd793cde2f64f9502d7ed2cb820c77c7c9988665a1d1f0103992c:current\_account-current\_region

[42x] success: Published 8bf1b08ec309464dd47af5ad7f149c59cd4724513937c5f7c82c0cF593:current\_account-current\_region

[57x] success: Published 8bf1b08ec309464dd47af5ad7f149c59cd4724513937c5f7c82c0cF593:current\_account-current\_region

[71x] success: Published 8bf1b08ec309464dd47af5ad7f149c59cd4724513937c5f7c82c0cF593:current\_account-current\_region

[85x] success: Published 8bf1b08ec309464dd47af5ad7f149c59cd4724513937c5f7c82c0cF593:current\_account-current\_region

[100x] success: Published 8bf1b08ec309464dd47af5ad7f149c59cd4724513937c5f7c82c0cF593:current\_account-current\_region

TodoAppStack: creating CloudFormation changeset...

TodoAppStack

Deployment time: 153.27s

Outputs:

TodoAppStack.ApiGatewayEndpoint5AA8EC3A = <https://05afy2waa.execute-api.ap-southeast-2.amazonaws.com/prod/>

Stack ARN:

arn:aws:cloudformation:ap-southeast-2:683793928497:stack/TodoAppStack/a1718250-fc1b-11ec-90f7-02a28d7d074a

Total time: 168.84s

~/source-code/youtube/todo-app > tutorial\_4 12 75

Console Home

Recently visited

- Lambda
- DynamoDB
- CloudWatch
- CloudFormation
- Cognito
- API Gateway
- S3

AWS Cost Explorer

IAM

AWS Single Sign-On

Directory Service

AWS Organizations

Welcome to AWS

Getting started with AWS

Training and certification

What's new with AWS?

AWS Health

Cost and usage

Open issues 0 Past 7 days

View all services

DynamoDB

Dashboard

Tables

- Update settings
- Explore items
- PartiQL editor
- Backups
- Exports to S3
- Reserved capacity
- Settings

DAX

- Clusters
- Subnet groups
- Parameter groups
- Events

Alarms (0)

DAX clusters (0)

Create resources

Create table

Create DAX cluster

What's new

March 10, 2022

Amazon DynamoDB increases default service quotas to simplify use of large numbers of tables

March 9, 2022

Amazon DynamoDB now supports the limit request option for PartiQL operations

DynamoDB

Dashboard

Tables

- Update settings
- Explore items
- PartiQL editor
- Backups
- Exports to S3
- Reserved capacity
- Settings

DAX

- Clusters

Tables (2)

Name	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mode	Size	Table class
cloud-native_db	Active	type (\$)	id (\$)	0	Provisioned (5)	Provisioned (5)	418 bytes	DynamoDB Standard
to-do_db	Active	type (\$)	id (\$)	0	Provisioned (5)	Provisioned (5)	0 bytes	DynamoDB Standard

ap-southeast-2 console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#table?initialTagKey=&name=to-do\_db&tab=overview

DynamoDB Tables to-do\_db

Tables (2)

Cloud-native\_db to-do\_db

General information

Partition key type (String) Sort key id (String) Capacity mode Provisioned Table status Active No active alarms

Items summary

Item count 0 Table size 0 bytes Average item size 0 bytes

ap-southeast-2 console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#item-explorer?initialTagKey=&maxItemSize=true&table=to-do\_db

DynamoDB Items to-do\_db

Scan/Query items

Scan Query to-do\_db

Completed Read capacity units consumed: 0.5

Items returned (0)

The query did not return any results.

ap-southeast-2.console.aws.amazon.com/lambda/home?region=ap-southeast-2#functions

Lambda Functions

Functions (12)

Filter by tags and attributes or search by keyword

Function name	Description	Package type	Runtime	Last modified
CloudNativeAppStack-LogRetentionae0aa3c5b4d4f87b0-pQ3mMpXfVuR0	-	Zip	Node.js 14.x	3 days ago
CloudNativeAppStack-deleteTodo3E1B05A4-3j5CckUh7Q99	-	Zip	Node.js 14.x	yesterday
TodoAppStack-deleteTodo3E1B05A4-Ss83t4Z7u8lv	-	Zip	Node.js 14.x	5 minutes ago
TodoAppStack-LogRetentionaa0aa3c5b4d4f8fb02d85b20-AXGcl1QL9Lym	-	Zip	Node.js 14.x	yesterday
CloudNativeAppStack-healthC87BE2FF-ftx59cgswURX	-	Zip	Node.js 14.x	yesterday
TodoAppStack-createTodoEB56D2CD-7Rwb0F2vATf	-	Zip	Node.js 14.x	5 minutes ago
TodoAppStack-getTodo77FA387-sjh1Y03ivOs	-	Zip	Node.js 14.x	5 minutes ago
TodoAppStack-updateTodo4E6557BD-JTOfzTkEaOy	-	Zip	Node.js 14.x	5 minutes ago
CloudNativeAppStack-createTodoE856D2CD-GZ28Ej4vLGUw	-	Zip	Node.js 14.x	yesterday
CloudNativeAppStack-updateTodo4E6557BD-cU72nB3fgyZ0	-	Zip	Node.js 14.x	yesterday

```

5 var __hasOwnProperty = Object.prototype.hasOwnProperty;
6 var __export = (target, module) => {
7   for (var name in module)
8     __defProp(target, name, { get: all[name], enumerable: true });
9 };
10 var __copyProps = (to, from, except, desc) => {
11   if (from === null || typeof from === 'undefined') return;
12   for (let key of Object.getOwnPropertyNames(from))
13     if (!__hasOwnProperty.call(to, key) && key !== except)
14       __defProp(to, key, { get: () => from[key], enumerable: !(desc = __getOwnPropertyDescriptor(from, key)) || desc.enumerable });
15 }
16 return to;
17 };
18 var __toCommonJS = (mod) => __copyProps(__defProp({}, '__esModule', { value: true }), mod);
19 var __createObject = (mod) => mod;
20 // assets/input/backend/lambda/createTodo.ts
21 var createTodo_exports = {};
22

```

```

23 var __getOwnPropertyDescriptor = (object, key) => {
24   let descriptor = Object.getOwnPropertyDescriptor(object, key);
25   if (descriptor === undefined) {
26     let constructor = object.constructor;
27     if (constructor === undefined) return;
28     descriptor = Object.getOwnPropertyDescriptor(constructor, key);
29   }
30   return descriptor;
31 };
32 var __copyProps = (to, from, except, desc) => {
33   if (from === null || typeof from === 'undefined') return;
34   for (let key of Object.getOwnPropertyNames(from))
35     if (!__hasOwnProperty.call(to, key) && key !== except)
36       __defProp(to, key, { get: () => from[key], enumerable: !(desc = __getOwnPropertyDescriptor(from, key)) || desc.enumerable });
37 }
38 var __toCommonJS = (mod) => __copyProps(__defProp({}, '__esModule', { value: true }), mod);
39 var __createObject = (mod) => mod;
40 // assets/input/backend/lambda/createTodo.ts
41 var createTodo_exports = {};
42

```

Next, let us start to implement changes to our lambda code

The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following command and its output:

```
~/source-code/youtube/todo-app > tutorial_4 14 75 npm install @aws-sdk/client-dynamodb inversify reflect-metadata uuid
npm WARN todo-app@0.1.0 No repository field.
npm WARN todo-app@0.1.0 No license field.

+ reflect-metadata@0.1.13
+ uuid@8.3.2
+ inversify@6.0.1
+ @aws-sdk/client-dynamodb@3.121.0
removed 1 package, updated 4 packages and audited 601 packages in 18.415s

27 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
```

Below this, another command is shown:

```
~/source-code/youtube/todo-app > tutorial_4 14 75 npm install -D @types/node @types/uuid
npm WARN todo-app@0.1.0 No repository field.
npm WARN todo-app@0.1.0 No license field.

+ @types/uuid@8.3.4
+ @types/node@10.17.2
added 1 package from 5 contributors, updated 1 package and audited 683 packages in 2.023s

29 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
```

A message box highlights a note about npm version:

New major version of npm available! 6.14.11 → 8.13.2  
Changelog: <https://github.com/npm/cli/releases/tag/v8.13.2>  
Run npm install -g npm to update!

The bottom of the terminal window shows the command again:

```
~/source-code/youtube/todo-app > tutorial_4 14 75
```

The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following commands and their outputs:

```
~/source-code/youtube/todo-app > tutorial_4 14 75 mkdir backend/helpers backend/controllers backend/services backend/loc backend/contracts backend/models
mkdir: backend/services: File exists
mkdir: backend/repositories: File exists
mkdir: backend/models: File exists

~/source-code/youtube/todo-app > tutorial_4 14 75 touch lib/Dynamo.ts
```

The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following commands and their outputs:

```
~/source-code/youtube/todo-app > tutorial_4 14 75 mkdir backend/helpers backend/controllers backend/services backend/loc backend/contracts backend/models
mkdir: backend/services: File exists
mkdir: backend/repositories: File exists
mkdir: backend/models: File exists

~/source-code/youtube/todo-app > tutorial_4 14 75 touch backend/contracts/todo.contract.ts backend/controllers/todo.controller.ts backend/helpers/error.helper.ts backend/helpers/response.helper.ts backend/loc/container.ts backend/loc/keys.ts backend/models/todo.model.ts backend/repositories/todo.repository.ts backend/services/logger.service.ts backend/todo.service.ts

~/source-code/youtube/todo-app > tutorial_4 14 713
```

The screenshot shows the VS Code interface with the editor tab active. The editor window displays the file `logger.service.ts`:

```
logger.service.ts - Untitled (Workspace)
1
```

Code Editor View (VS Code) showing the `logger.service.ts` file in the `todo-app` workspace. The code implements a `ILoggerService` interface with a `writeLog` method. It handles both plain strings and Error objects by logging them along with their stack traces. It also attempts to JSON.stringify the data if it's not an Error object.

```
import { injectable } from "inversify";
export interface ILoggerService {
    log(message:string, data?: any, className?: string): void;
}
@injectable()
export class LoggerService implements ILoggerService {
    private _output: Function;
    constructor() {
        this._output = console.log;
    }
    private writeLog(message:string, data?: any, className?: string) {
        let dataOutput = data ?? {};
        if (dataOutput instanceof Error) {
            dataOutput = "Error message: " + dataOutput.message + "; Stack: " + dataOutput.stack;
        } else {
            try {
                JSON.stringify(dataOutput);
            } catch (err) {
                dataOutput = "Unable to serialize error data";
            }
        }
        log(message, dataOutput, className);
    }
    log(message: string, data?: any, className?: string | undefined): void {
        throw new Error("Method not implemented.");
    }
}
```

Code Editor View (VS Code) showing the same `logger.service.ts` file in a new workspace named `UNTITLED (WORKSPACE)`. The code is identical to the previous version, implementing the `ILoggerService` interface with a `writeLog` method that handles strings and Error objects by logging them with their stack traces or serializing them to JSON if they're not Errors.

```
import { injectable } from "inversify";
export interface ILoggerService {
    log(message:string, data?: any, className?: string): void;
}
@injectable()
export class LoggerService implements ILoggerService {
    private _output: Function;
    constructor() {
        this._output = console.log;
    }
    private writeLog(message:string, data?: any, className?: string) {
        let dataOutput = data ?? {};
        if (dataOutput instanceof Error) {
            dataOutput = "Error message: " + dataOutput.message + "; Stack: " + dataOutput.stack;
        } else {
            try {
                JSON.stringify(dataOutput);
            } catch (err) {
                dataOutput = "Unable to serialize error data";
            }
        }
        const outObject = {
            message,
            data: dataOutput,
            timestamp: new Date().toISOString(),
            location: className
        };
        log(message, outObject, className);
    }
    log(message: string, data?: any, className?: string | undefined): void {
        throw new Error("Method not implemented.");
    }
}
```

Code File Edit Selection View Go Run Terminal Window Help

logger.service.ts — Untitled (Workspace)

```
todo-app > backend > services > logger.service.ts > LoggerService > log

22     JSON.stringify(dataOutput);
21   } catch (err) {
20     dataOutput = "Unable to serialize error data";
19   }
18 }

const outObject = {
17   message,
16   dataOutput,
15   timestamp: new Date().toISOString(),
14   location: className
13 };
12
11};

let outString: string;
10 try {
9   outString = JSON.stringify(outObject);
8 } catch (err) {
7   outString = "Error trying to serialize for logs";
6 }
5
4 }

1 log(message: string, data?: any, className?: string | undefined): void {
2   this.writeLog(message, data, className);
1 }

42
41
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
```

Code File Edit Selection View Go Run Terminal Window Help

todo.repository.ts — Untitled (Workspace)

```
todo-app > backend > repositories > todo.repository.ts > ITodoRepository

1 export interface ITodoRepository {
```

Code File Edit Selection View Go Run Terminal Window Help

keys.ts — Untitled (Workspace)

```
todo.repository.ts 1, U ● keys.ts U X

todo-app > backend > loc > keys.ts > ContainerKeys > ITodoController

4 export const ContainerKeys = {
3   ILoggerService: Symbol.for("ILoggerService"),
2   ITodoRepository: Symbol.for("ITodoRepository"),
1   ITodoService: Symbol.for("ITodoService"),
ITodoController: Symbol.for("ITodoController")
1 };
2
```

Code File Edit Selection View Go Run Terminal Window Help

container.ts — Untitled (Workspace)

```
todo.repository.ts 1, U ● keys.ts U ● container.ts U X

todo-app > backend > loc > container.ts > ...

10 import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
9 import { Container } from "inversify";
8 import { ILoggerService, LoggerService } from "../services/logger.service";
7 import { ContainerKeys } from "./keys";
6
5 const container = new Container();
4
3 try {
2   const dynamoClient = new DynamoDBClient({ region: "ap-southeast-2" });
1   container.bind(DynamoDBClient).toConstantValue(dynamoClient);
  container.bind<ILoggerService>(ContainerKeys.ILoggerService).to(LoggerService);
11 }
1 } catch (err) {
2   console.log("Error during initialization", err);
3 }
4
5 export { container };
6
```

todo.repository.ts - Untitled (Workspace)

```
todo-app > backend > repositories > todo.repository.ts > IToDoRepository
6 import { PutItemCommandOutput, QueryCommandOutput } from "@aws-sdk/client-dynamodb";
5
4 export interface IToDoRepository {
3   createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
2   getTodo(): Promise<QueryCommandOutput>;
1   getTodoById(id: string): Promise<QueryCommandOutput>;
7 }
```

todo.model.ts - Untitled (Workspace)

```
todo-app > backend > models > todo.model.ts > ITodo > updated
12 export enum Status {
11   CREATED = "CREATED",
10   IN_PROGRESS = "IN_PROGRESS",
9   COMPLETED = "COMPLETED"
8 }
7
6 export interface ITodo {
5   type: string;
4   id: string;
3   content: string;
2   status: Status;
1   created: string;
13   updated: string;
1
2 }
```

todo.repository.ts - Untitled (Workspace)

```
todo-app > backend > repositories > todo.repository.ts > IToDoRepository > deleteTodo
17 import { DeleteItemCommandOutput, PutItemCommandOutput, QueryCommandOutput, UpdateItemCommandOutput } from "@aws-sdk/client-dynamodb";
16 import { injectable } from "inversify";
15 import { Status } from "../models/todo.model";
14
13 export interface IToDoRepository {
12   createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
11   getTodo(): Promise<QueryCommandOutput>;
10   getTodoById(id: string): Promise<QueryCommandOutput>;
9   updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput>;
8   deleteTodo(id: string): Promise<DeleteItemCommandOutput>;
7 }
6
5 @injectable()
4 export class TodoRepository implements IToDoRepository {
3   private tableName = "to-do_db";
2   private partitionKey = "type";
1   private sortKey = "id"
18   constructor(
1
2   ) {}
3   createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
4     throw new Error("Method not implemented.");
5   }
6   getTodo(): Promise<QueryCommandOutput> {
7     throw new Error("Method not implemented.");
8   }
9   getTodoById(id: string): Promise<QueryCommandOutput> {
10    throw new Error("Method not implemented.");
11   }
12   updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
13    throw new Error("Method not implemented.");
14   }
15   deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
16    throw new Error("Method not implemented.");
17   }
18 }
```

Code File Edit Selection View Go Run Terminal Window Help

todo.repository.ts — Untitled (Workspace)

```
todo-app > backend > repositories > todo.repository.ts > TodoRepository > constructor
```

```
21 import { DeleteItemCommandOutput, DynamoDBClient, PutItemCommandOutput, QueryCommandOutput, UpdateItemCommandOutput } from "@aws-sdk/client"
22 import { inject, injectable } from "inversify";
23 import { ContainerKeys } from "../ioc/keys";
24 import { Status } from "../models/todo.model";
25 import { LoggerService } from "../services/logger.service";
26
27 export interface ITodoRepository {
28   createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
29   getTodo(): Promise<QueryCommandOutput>;
30   getTodoById(id: string): Promise<QueryCommandOutput>;
31   updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput>;
32   deleteTodo(id: string): Promise<DeleteItemCommandOutput>;
33 }
34
35 @injectable()
36 export class TodoRepository implements ITodoRepository{
37   private tableName = "to-do_db";
38   private partitionKey = "type";
39   private sortKey = "id";
40   constructor(
41     @inject(ContainerKeys.ILoggerService) private logger: LoggerService,
42     @inject(DynamoDBClient) private client: DynamoDBClient,
43   ) {}
44   createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
45     throw new Error("Method not implemented.");
46   }
47   getTodo(): Promise<QueryCommandOutput> {
48     throw new Error("Method not implemented.");
49   }
50   getTodoById(id: string): Promise<QueryCommandOutput> {
51     throw new Error("Method not implemented.");
52   }
53   updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
54     throw new Error("Method not implemented.");
55   }
56   deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
57     throw new Error("Method not implemented.");
58 }
```

tutorial-4\* 0.0.0 Git Graph -- NORMAL --

Code File Edit Selection View Go Run Terminal Window Help

todo.repository.ts — Untitled (Workspace)

```
todo-app > backend > repositories > todo.repository.ts > TodoRepository > createTodo
```

```
18   private sortKey = "id";
19   constructor(
20     @inject(ContainerKeys.ILoggerService) private logger: LoggerService,
21     @inject(DynamoDBClient) private client: DynamoDBClient,
22   ) {}
23
24   async createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
25     this.logger.log(`createdTodo() called`, {params}, this.constructor.name)
26     const putItem: any = {};
27     for (const key in params) {
28       putItem[key] = {
29         S: params[key]
30       }
31     }
32
33     const input: PutItemCommandInput = {
34       TableName: this.tableName,
35       Item: putItem
36     };
37
38     await this.client.putItem(input);
39   }
40   getTodo(): Promise<QueryCommandOutput> {
41     throw new Error("Method not implemented.");
42   }
43   getTodoById(id: string): Promise<QueryCommandOutput> {
44     throw new Error("Method not implemented.");
45   }
46   updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
47     throw new Error("Method not implemented.");
48   }
49   deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
50     throw new Error("Method not implemented.");
51   }
52 }
```

outline timeline

todo.repository.ts

```
25
26     @injectable()
27
28     export class TodoRepository implements ITodoRepository {
29         private tableName = "to-do_db";
30         private partitionKey = "type";
31         private sortKey = "id";
32         constructor(
33             @inject(ContainerKeys.ILoggerService) private logger: LoggerService,
34             @inject(DynamoDBClient) private client: DynamoDBClient
35         ) {}
36
37         async createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
38             this.logger.log("createdTodo() called", { params }, this.constructor.name);
39             const putItem: any = {};
40             for (const key in params) {
41                 putItem[key] = {
42                     S: params[key]
43                 };
44             }
45
46             const input: PutItemCommandInput = {
47                 TableName: this.tableName,
48                 Item: putItem
49             };
50             const command = new PutItemCommand(input);
51             this.logger.log("createTodo() finished");
52             return await this.client.send(command);
53         }
54
55         getTodo(): Promise<QueryCommandOutput> {
56             throw new Error("Method not implemented.");
57         }
58
59         getTodoById(id: string): Promise<QueryCommandOutput> {
60             throw new Error("Method not implemented.");
61         }
62
63         updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
64
65         }
66
67         deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
68
69     }
```

todo.repository.ts

```
8
9     TableName: this.tableName,
10    Item: putItem
11  };
12  const command = new PutItemCommand(input);
13  this.logger.log("createTodo() finished");
14  return await this.client.send(command);
15}
16
17async getTodo(): Promise<QueryCommandOutput> {
18    this.logger.log("getTodo() called", null, this.constructor.name);
19
20    const input: QueryCommandInput = {
21        TableName: this.tableName,
22        ExpressionAttributeNames: {
23            "#type": "type"
24        },
25        ExpressionAttributeValues: {
26            ":queryType": { S: "Todo" }
27        },
28        KeyConditionExpression: "#type = :queryType"
29    };
30
31    const command = new QueryCommand(input);
32    const response = await this.client.send(command);
33
34    this.logger.log("getTodo() called", { response }, this.constructor.name);
35    return response;
36}
37
38getTodoById(id: string): Promise<QueryCommandOutput> {
39    throw new Error("Method not implemented.");
40}
41
42updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
43    throw new Error("Method not implemented.");
44}
45
46deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
47}
```

todo.repository.ts — Untitled (Workspace)

```
6  this.logger.log("getTodo() called", { response }, this.constructor.name);
7  return response;
8 }
9
10 async getTodoById(id: string): Promise<QueryCommandOutput> {
11   ...
12
13   async updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
14     throw new Error("Method not implemented.");
15   }
16
17   async deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
18     throw new Error("Method not implemented.");
19   }
20 }
```

todo.repository.ts — Untitled (Workspace)

```
28   this.logger.log("getTodo() called", { response }, this.constructor.name);
29   return response;
30 }
31
32 async getTodoById(id: string): Promise<QueryCommandOutput> {
33   this.logger.log("getTodoById() called", null, this.constructor.name);
34
35   const input: QueryCommandInput = {
36     TableName: this.tableName,
37     ExpressionAttributeNames: {
38       "#type": "type",
39       "#id": "id"
40     },
41     ExpressionAttributeValues: [
42       { :queryType: { S: "Todo" } },
43       { :queryId: { S: id } }
44     ],
45     KeyConditionExpression: "#type = :queryType AND #id = :queryId"
46   };
47   const command = new QueryCommand(input);
48   const response = await this.client.send(command);
49
50   this.logger.log("getTodoById() called", { response }, this.constructor.name);
51   return response;
52 }
53
54 async updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
55   throw new Error("Method not implemented.");
56 }
57
58 async deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
59   throw new Error("Method not implemented.");
60 }
```

todo.repository.ts — Untitled (Workspace)

```
10   KeyConditionExpression: "#type = :queryType AND #id = :queryId"
11 };
12   const command = new QueryCommand(input);
13   const response = await this.client.send(command);
14
15   this.logger.log("getTodoById() called", { response }, this.constructor.name);
16   return response;
17 }
18
19 async updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
20   this.logger.log("updateTodo")
21 }
22
23 async deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
24   throw new Error("Method not implemented.");
25 }
```

todo.repository.ts

```
32      "queryId": { S: id }
31    },
30    KeyConditionExpression: "#type = :queryType AND #id = :queryId"
29  );
28  const command = new QueryCommand(input);
27  const response = await this.client.send(command);
26
25  this.logger.log("getTodoById() called", { response }, this.constructor.name);
24  return response;
23 }

22
21 async updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
20  this.logger.log("updateTodo() called", { status, id }, this.constructor.name);

19  const input: UpdateItemCommandInput = {
18    TableName: this.tableName,
17    Key: {
16      [this.partitionKey]: { S: "Todo" },
15      [this.sortKey]: { S: id }
14    },
13    UpdateExpression: "SET #status = :updatedStatus",
12    ExpressionAttributeNames: {
11      "#status": "status"
10    },
9    ExpressionAttributeValues: {
8      ":updatedStatus": { S: status }
7    }
6  };
5
4  const command = new UpdateItemCommand(input);
3  const response = await this.client.send(command);
2
1  this.logger.log("updateTodo() finished", { response }, this.constructor.name);
0  return response;
}

3 async deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
4  throw new Error("Method not implemented.");
}
```

Ln 117, Col 19 Spaces: 4 UTF-8 LF () TypeScript ✓ Prettier

todo.repository.ts

```
26      },
25    ExpressionAttributeValues: {
24      ":updatedStatus": { S: status }
23    }
22  );
21  const command = new UpdateItemCommand(input);
20  const response = await this.client.send(command);
19
18  this.logger.log("updateTodo() finished", { response }, this.constructor.name);
17  return response;
16 }

15
14 async deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
13  this.logger.log("deleteTodo() called", { id }, this.constructor.name);

12  const input: DeleteItemCommandInput = {
11    TableName: this.tableName,
10    Key: {
9      [this.partitionKey]: { S: "Todo" },
8      [this.sortKey]: { S: id }
7    }
6  };
5
4  const command = new DeleteItemCommand(input);
3  const response = await this.client.send(command);
2
1  this.logger.log("deleteTodo() finished", { response }, this.constructor.name);
0  return response;
}
```

Code Editor: container.ts - Untitled (Workspace)

```
todo.repository.ts U container.ts U
```

```
todo-app > backend > ioc > container > ...
12 import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
11 import { Container } from 'inversify';
10 import { ITodoRepository, TodoRepository } from '../repositories/todo.repository';
9 import { ILoggerService, LoggerService } from '../services/logger.service';
8 import { ContainerKeys } from './keys';
7
6 const container = new Container();
5
4 try {
3   const dynamoClient = new DynamoDBClient({ region: "ap-southeast-2" });
2   container.bind(DynamoDBClient).toConstantValue(dynamoClient);
1   container.bind(ContainerKeys.ILoggerService).to(LoggerService);
0   container.bind(ContainerKeys.ITodoRepository).to(TodoRepository);
} catch (err) {
2   console.log("Error during initialization", err);
3 }
4
5 export { container };
6
```

Code Editor: error.helper.ts - Untitled (Workspace)

```
todo.repository.ts U container.ts U error.helper.ts U
```

```
todo-app > backend > helpers > error.helper.ts > handleServiceErrors
3 import { APIGatewayProxyResult } from 'aws-lambda';
2
1 export const handleServiceErrors = (err: Error): APIGatewayProxyResult => {
4   console.log("Exception", err);
1   let errorMessage = "";
2   if (err instanceof Error) {
3     errorMessage = err.message;
4   } else {
5     errorMessage = JSON.stringify(err);
6   }
7   const result: APIGatewayProxyResult = {
8     statusCode: 500,
9     body: `Internal server error: ${errorMessage}`;
10 };
11
12 return result;
13
```

Code Editor: response.helper.ts - Untitled (Workspace)

```
todo.repository.ts U container.ts U error.helper.ts U response.helper.ts U
```

```
todo-app > backend > helpers > response.helper.ts > processResponse > headers > "Access-Control-Allow-Credentials"
12 import { APIGatewayProxyResult } from 'aws-lambda';
11
10 export const processResponse = (status: number, body: Record<string, any>): APIGatewayProxyResult => {
9   return {
8     body: JSON.stringify(body),
7     statusCode: status,
6     headers: [
5       "Content-Type": "application/json",
4       "Access-Control-Allow-Origin": "*",
3       "Content-control": "no-store",
2       Pragma: "no-cache",
1       "Strict-Transport-Security": "max-age=31536000; includeSubDomains",
0       "Access-Control-Allow-Credentials": true
1     ]
2   };
3 }
```

Code Editor: todo.service.ts - Untitled (Workspace)

```
todo.service.ts U
```

```
todo-app > backend > services > todo.service.ts > ITodoService
1 export interface ITodoService {
2 }
```

todo.contract.ts – Untitled (Workspace)

```
13 import { Status } from "../models/todo.model";
14
15 export namespace Contracts {
16     export interface CreateTodo {
17         content: string;
18     }
19
20     export interface UpdateTodo {
21         id: string;
22         status: Status;
23     }
24
25     export interface DeleteTodo {
26         id: string;
27     }
28 }
```

todo.service.ts – Untitled (Workspace)

```
18 import { Contracts } from "../contracts/todo.contract";
19 import { ITodo } from "../models/todo.model";
20
21 export interface ITodoService {
22     createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
23     getTodo(): Promise<ITodo[]>;
24     updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
25     deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
26 }
```

todo.service.ts – Untitled (Workspace)

```
12 import { injectable } from "inversify";
13 import { Contracts } from "../contracts/todo.contract";
14 import { ITodo } from "../models/todo.model";
15
16 export interface ITodoService {
17     createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
18     getTodo(): Promise<ITodo[]>;
19     updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
20     deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
21 }
22
23 @injectable()
24 export class TodoService implements ITodoService{
25     createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
26         throw new Error("Method not implemented.");
27     }
28     getTodo(): Promise<ITodo[]> {
29         throw new Error("Method not implemented.");
30     }
31     updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
32         throw new Error("Method not implemented.");
33     }
34     deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
35         throw new Error("Method not implemented.");
36     }
37 }
```

```
todo.service.ts - Untitled (Workspace)

todo-app > backend > services > todo.service.ts > TodoService > constructor
19 import { inject, injectable } from "inversify";
20 import { Contracts } from "../contracts/todo.contract";
21 import { ContainerKeys } from "../ioc/keys";
22 import { ITodo } from "../models/todo.model";
23 import { ITodoRepository } from "../repositories/todo.repository";
24 import { ILoggerService } from "./logger.service";

25 export interface ITodoService {
26   createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
27   getTodo(): Promise<ITodo[]>;
28   updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
29   deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
30 }

31 @injectable()
32 export class TodoService implements ITodoService {
33   constructor(
34     @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
35     @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
36   ) {}
37
38   async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
39     throw new Error("Method not implemented.");
40   }
41
42   async getTodo(): Promise<ITodo[]> {
43     throw new Error("Method not implemented.");
44   }
45
46   async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
47     throw new Error("Method not implemented.");
48   }
49
50   async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
51     throw new Error("Method not implemented.");
52   }
53 }

54 @injectable()
55 export class TodoService implements ITodoService {
56   constructor(
57     @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
58     @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
59   ) {}
60
61   async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
62     this.logger.log("createTodo() called", {params}, this.constructor.name)
63
64     const {content} = params;
65     const createdTodo: ITodo = {
66       type: "Todo",
67       id:
68     };
69
70     return createdTodo;
71   }
72
73   async getTodo(): Promise<ITodo[]> {
74     throw new Error("Method not implemented.");
75   }
76
77   async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
78     throw new Error("Method not implemented.");
79   }
80 }
```

```
todo.service.ts - Untitled (Workspace)

todo-app > backend > services > todo.service.ts > ...
1 import { Contracts } from "../contracts/todo.contract";
2 import { ContainerKeys } from "../ioc/keys";
3 import { ITodo } from "../models/todo.model";
4 import { ITodoRepository } from "../repositories/todo.repository";
5 import { ILoggerService } from "./logger.service";
6
7 import { v4 as uid } from "uuid"

8 export interface ITodoService {
9   createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
10  getTodo(): Promise<ITodo[]>;
11  updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
12  deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
13 }

14 @injectable()
15 export class TodoService implements ITodoService {
16   constructor(
17     @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
18     @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
19   ) {}

20
21   async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
22     this.logger.log("createTodo() called", {params}, this.constructor.name)
23
24     const {content} = params;
25     const createdTodo: ITodo = {
26       type: "Todo",
27       id:
28     };
29
30     return createdTodo;
31   }
32
33   async getTodo(): Promise<ITodo[]> {
34     throw new Error("Method not implemented.");
35   }
36
37   async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
38     throw new Error("Method not implemented.");
39   }
40 }
```

todo.service.ts - Untitled (Workspace)

```
todo-app > backend > services > todo.service.ts > TodoService > getTodo
```

```
22  @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
21  ) {}
20  async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
19   this.logger.log("createTodo() called", { params }, this.constructor.name);
18
17  const { content } = params;
16  const createdTodo: ITodo = {
15   type: "Todo",
14   id: uid(),
13   content,
12   status: Status.CREATED,
11   created: new Date().toISOString(),
10   updated: new Date().toISOString()
9  };
8
7  const response = await this.repo.createTodo(createdTodo);
6
5  this.logger.log("dynamo response", { response }, this.constructor.name);
4  return createdTodo;
3
2
1
42  async getTodo(): Promise<ITodo[] | []> {
1
2
3
4  throw new Error("Method not implemented.");
5
6
7
8
9
10
11 }
```

todo.service.ts - Untitled (Workspace)

```
todo-app > backend > services > todo.service.ts > TodoService > processDynamonItem
```

```
34  content,
33  status: Status.CREATED,
32  created: new Date().toISOString(),
31  updated: new Date().toISOString()
30  );
29  const response = await this.repo.createTodo(createdTodo);
28
27  this.logger.log("dynamo response", { response }, this.constructor.name);
26  return createdTodo;
25
24
23  async getTodo(): Promise<ITodo[] | []> {
22  this.logger.log("getTodo() called");
21
20  const response = await this.repo.getTodo();
19
18
17  async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
16  throw new Error("Method not implemented.");
15
14
13  async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
12  throw new Error("Method not implemented.");
11
10  private isValidStatus(status: string): status is Status {
9  return Object.values(Status).some((v) => v === status);
8
7
6
5  private processDynamonItem(item: any): []
4  const output: any = {};
3  for (const k in item) {
2  | output[k] = item[k][`$`];
1
2
1
64
1
2 }
```

todo.service.ts - Untitled (Workspace)

```
content,
status: Status.CREATED,
created: new Date().toISOString(),
updated: new Date().toISOString()
);
const response = await this.repo.createTodo(createdTodo);

this.logger.log("dynamo response", { response }, this.constructor.name);
return createdTodo;
}

async getTodo(): Promise<ITodo[] | []> {
this.logger.log("getTodo() called");

const response = await this.repo.getTodo();
let result: any = []
if (!response.Count > 0)
}

async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
throw new Error("Method not implemented.");
}

async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
throw new Error("Method not implemented.");
}

private isStatusValid(status: string): status is Status {
return Object.values(Status).some((v) => v === status);
}

private processDynamoItem(item: any) {
const output: any = {};
for (const k in item) {
output[k] = item[k][S];
}
return output;
}
```

Ln 46, Col 24 Spaces: 4 UTF-8 LF () TypeScript Go Live ✅ Prettier

todo.service.ts - Untitled (Workspace)

```
content,
status: Status.CREATED,
created: new Date().toISOString(),
updated: new Date().toISOString()
);
const response = await this.repo.createTodo(createdTodo);

this.logger.log("dynamo response", { response }, this.constructor.name);
return createdTodo;
}

async getTodo(): Promise<ITodo[] | []> {
this.logger.log("getTodo() called");

const response = await this.repo.getTodo();
let result: any = []
if (!response.Count > 0)
}

async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
throw new Error("Method not implemented.");
}

async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
throw new Error("Method not implemented.");
}

private isStatusValid(status: string): status is Status {
return Object.values(Status).some((v) => v === status);
}

private processDynamoItem(item: any) {
const output: any = {};
for (const k in item) {
output[k] = item[k][S];
}
return output;
}
```

Object is possibly 'undefined'. ts(2532)  
View Problem No quick fixes available  
if (!response.Count > 0)

Ln 46, Col 24 Spaces: 4 UTF-8 LF () TypeScript Go Live ✅ Prettier

Code Editor showing the `tsconfig.json` file in the `todo-app` workspace. The file contains configuration for TypeScript compilation, including strict null checks and experimental decorators.

```
8 {
  "compilerOptions": [
    "target": "ES2018",
    "module": "commonjs",
    "lib": ["es2018"],
    "declaration": true,
    "strict": true,
    "noImplicitAny": true,
    "strictNullChecks": false,
    "noImplicitThis": true,
    "alwaysStrict": true,
    "noUnusedLocals": false,
    "noUnusedParameters": false,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": false,
    "inlineSourceMap": true,
    "inlineSources": true,
    "experimentalDecorators": true,
    "strictPropertyInitialization": false,
    "typeRoots": ["./node_modules/@types"]
  ],
  "exclude": ["node_modules", "cdk.out"]
}
```

Code Editor showing the `todo.service.ts` file in the `todo-app` workspace. The file contains implementation for a service that interacts with a DynamoDB repository.

```
16 }
17 }
18
19 async getTodo(): Promise<ITodo[] | []> {
  this.logger.log("getTodo() called");
20
21   const response = await this.repo.getTodo();
22   let result: any = [];
23   if (response.Count > 0) {
24     result = response.Items.map((item) => this.processDynamoItem(item));
25   }
26
27   this.logger.log("dynamo response", { response }, this.constructor.name);
28   return result;
29 }
30
31
32 async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
33 }
34
35 async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
36   throw new Error("Method not implemented.");
37 }
38
39 private isValidStatus(status: string): status is Status {
40   return Object.values(Status).some((v) => v === status);
41 }
42
43 private processDynamoItem(item: any) {
44   const output: any = {};
45   for (const k in item) {
46     if (output[k] = item[k][`S`]);
47   }
48   return output;
49 }
```

```
todo.service.ts - Untitled (Workspace)

16     return result;
17   }
18
19   async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
20     const { status, id } = params;
21     if (!this.isValidStatus(status)) throw new Error("Invalid status");
22
23     const response = await this.repo.updateTodo(status, id);
24     this.logger.log(`dynamo response`, { response }, this.constructor.name);
25     if (response.$metadata.httpStatusCode == 200) {
26       const updatedTodo = await this.repo.getTodoById(id);
27       this.logger.log(`updatedTodo`, updatedTodo);
28       return this.processDynamoItem(updatedTodo.Items[0]);
29     }
30     return null;
31   }
32
33   async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
34     throw new Error("Method not implemented.");
35   }
36
37   private isValidStatus(status: string): status is Status {
38     return Object.values(Status).some((v) => v === status);
39   }
40
41   private processDynamoItem(item: any) {
42     const output: any = {};
43     for (const k in item) {
44       output[k] = item[k][S];
45     }
46     return output;
47   }
48 }

19
```

```
todo.service.ts - Untitled (Workspace)

23     const { status, id } = params;
24     if (!this.isValidStatus(status)) throw new Error("Invalid status");
25
26     const response = await this.repo.updateTodo(status, id);
27     this.logger.log(`dynamo response`, { response }, this.constructor.name);
28     if (response.$metadata.httpStatusCode == 200) {
29       const updatedTodo = await this.repo.getTodoById(id);
30       this.logger.log(`updatedTodo`, updatedTodo);
31       return this.processDynamoItem(updatedTodo.Items[0]);
32     }
33     return null;
34   }
35
36   async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
37     this.logger.log(`deleteTodo() called`, { params }, this.constructor.name);
38     const { id } = params;
39
40     const response = await this.repo.deleteTodo(id);
41     this.logger.log(`dynamo response`, { response }, this.constructor.name);
42     return;
43   }
44
45   private isValidStatus(status: string): status is Status {
46     return Object.values(Status).some((v) => v === status);
47   }
48
49   private processDynamoItem(item: any) {
50     const output: any = {};
51     for (const k in item) {
52       output[k] = item[k][S];
53     }
54     return output;
55   }
56 }

12
```

```
container.ts - Untitled (Workspace)

12 import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
13 import { Container } from 'inversify';
14 import { ITodoRepository, TodoRepository } from '../repositories/todo.repository';
15 import { ILoggerService, LoggerService } from '../services/logger.service';
16 import { ContainerKeys } from './keys';

17 const container = new Container();
18
19 try {
20   const dynamoClient = new DynamoDBClient({ region: 'ap-southeast-2' });
21   container.bind(DynamoDBClient).toConstantValue(dynamoClient);
22   container.bind
```

VS Code interface showing the Explorer sidebar with a tree view of files and folders. The current file is `container.ts` in the `backend/loc` directory. The code implements a Container class using Inversify.js to bind various services like DynamoDBClient, ILoggerService, ITodoRepository, and ITodoService.

```
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { Container } from 'inversify';
import { ITodoRepository, TodoRepository } from '../repositories/todo.repository';
import { ILoggerService, LoggerService } from '../services/logger.service';
import { ITodoService, TodoService } from '../services/todo.service';
import { ContainerKeys } from './keys';

const container = new Container();

try {
    const dynamoClient = new DynamoDBClient({ region: "ap-southeast-2" });
    container.bind(DynamoDBClient).toConstantValue(dynamoClient);
    container.bind(ContainerKeys.ILoggerService).to(LoggerService);
    container.bind(ContainerKeys.ITodoRepository).to(TodoRepository);
    container.bind(ContainerKeys.ITodoService).to(TodoService);
} catch (err) {
    console.log("Error during initialization", err);
}

export { container };
```

VS Code interface showing the Explorer sidebar with a tree view of files and folders. The current file is `todo.controller.ts` in the `backend/controllers` directory. It defines an `ITodoController` interface with methods for creating, getting, updating, and deleting todos.

```
export interface ITodoController {
```

VS Code interface showing the Explorer sidebar with a tree view of files and folders. The current file is `todo.controller.ts` in the `backend/controllers` directory. It implements the `ITodoController` interface using AWS Lambda's APIGatewayProxyEvent and APIGatewayProxyResult types.

```
import { APIGatewayEvent, APIGatewayProxyEvent, APIGatewayProxyResult } from 'aws-lambda';
import { injectable } from 'inversify';

export interface ITodoController {
    createTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
    getTodo(): Promise<APIGatewayProxyResult>;
    updateTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
    deleteTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
}

@injectable()
export class TodoController implements ITodoController {
    createTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
        throw new Error("Method not implemented.");
    }
    getTodo(): Promise<APIGatewayProxyResult> {
        throw new Error("Method not implemented.");
    }
    updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
        throw new Error("Method not implemented.");
    }
    deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
        throw new Error("Method not implemented.");
    }
}
```

VS Code interface showing the Explorer sidebar with a tree view of files and folders. The current file is `todo.controller.ts` in the `backend/controllers` directory. The code has been updated to use `async/await` and explicit type annotations for the `createTodo`, `getTodo`, `updateTodo`, and `deleteTodo` methods.

```
import { APIGatewayEvent, APIGatewayProxyEvent, APIGatewayProxyResult } from 'aws-lambda';
import { injectable } from 'inversify';

export interface ITodoController {
    createTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult>;
    getTodo(): Promise<APIGatewayProxyResult>;
    updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult>;
    deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult>;
}

@injectable()
export class TodoController implements ITodoController {
    async createTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
        throw new Error("Method not implemented.");
    }
    async getTodo(): Promise<APIGatewayProxyResult> {
        throw new Error("Method not implemented.");
    }
    async updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
        throw new Error("Method not implemented.");
    }
    async deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
        throw new Error("Method not implemented.");
    }
}
```

Code Editor Screenshot:

File: todo.controller.ts

```
import { APIGatewayEvent, APIGatewayProxyEvent, APIGatewayProxyResult } from "aws-lambda";
import { inject, injectable } from "inversify";
import { handleServiceErrors } from "../helpers/error.helper";
import { ContainerKeys } from "../ioc/keys";
import { ITodoService } from "../services/todo.service";

export interface ITodoController {
    createTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
    getTodo(): Promise<APIGatewayProxyResult>;
    updateTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
    deleteTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
}

@injectable()
export class TodoController implements ITodoController {
    constructor(@inject(ContainerKeys.ITodoService) private service: ITodoService) {}

    async createTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
        try {
        } catch (err: any) {
            return handleServiceErrors(err);
        }
    }

    async getTodo(): Promise<APIGatewayProxyResult> {
        try {
        }
    }

    async updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
        try {
        }
    }

    async deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
        try {
        } catch (err: any) {
            return handleServiceErrors(err);
        }
    }
}
```

Code Editor Screenshot:

File: todo.controller.ts

```
async createTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
    try {
    } catch (err: any) {
        return handleServiceErrors(err);
    }
}

async getTodo(): Promise<APIGatewayProxyResult> {
    try {
    } catch (err: any) {
        return handleServiceErrors(err);
    }
}

async updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
    try {
    } catch (err: any) {
        return handleServiceErrors(err);
    }
}

async deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
    try {
    } catch (err: any) {
        return handleServiceErrors(err);
    }
}
```

Code File Edit Selection View Go Run Terminal Window Help

todo.controller.ts — Untitled (Workspace)

```
todo.controller.ts 3, U

① todo-app > backend > controllers > todo.controller.ts > TodoController > createTodo
15 export class TodoController implements ITodoController {
16   constructor(@inject(ContainerKeys.ITodoService) private service: ITodoService) {}
17
18   async createTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
19     try {
20       const { body } = params;
21       if (body === null) throw new Error("Missing request body");
22       const { content } = JSON.parse(body);
23       if (!content) throw new Error("Missing content in request body");
24       const response = await this.service.createTodo({ content });
25
26       return processResponse(201, response);
27     } catch (err: any) {
28       return handleServiceErrors(err);
29     }
30   }
31
32   async getTodo(): Promise<APIGatewayProxyResult> {
33     try {
34     } catch (err: any) {
35       return handleServiceErrors(err);
36     }
37   }
38
39   async updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
40     try {
41     } catch (err: any) {
42       return handleServiceErrors(err);
43     }
44   }
45
46   async deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
47     try {
48     } catch (err: any) {
49       return handleServiceErrors(err);
50     }
51   }
52 }
```

Code File Edit Selection View Go Run Terminal Window Help

todo.controller.ts — Untitled (Workspace)

```
todo.controller.ts 2, U

① todo-app > backend > controllers > todo.controller.ts > TodoController > getTodo
1
2
3
4
5   async getTodo(): Promise<APIGatewayProxyResult> {
6     try {
7       const response = await this.service.getTodo();
8       return processResponse(200, response);
9     } catch (err: any) {
10       return handleServiceErrors(err);
11     }
12   }
13
14   async updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
15     try {
16     } catch (err: any) {
17       return handleServiceErrors(err);
18     }
19
20   }
21
22   async deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
23     try {
24     } catch (err: any) {
25       return handleServiceErrors(err);
26     }
27   }
28 }
```

Code File Edit Selection View Go Run Terminal Window Help

todo.controller.ts — Untitled (Workspace)

```
todo.controller.ts 1, U

① todo-app > backend > controllers > todo.controller.ts > TodoController > updateTodo
1
2
3
4
5   async updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
6     try {
7       const { body } = params;
8       if (body === null) throw new Error("Missing request body");
9       const { status } = JSON.parse(body);
10      const { id } = params.pathParameters;
11      const response = await this.service.updateTodo({ status, id });
12      return processResponse(200, response);
13    } catch (err: any) {
14      return handleServiceErrors(err);
15    }
16  }
17
18  async deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
19    try {
20    } catch (err: any) {
21      return handleServiceErrors(err);
22    }
23  }
24 }
```

Code File Edit Selection View Go Run Terminal Window Help

todo.controller.ts – Untitled (Workspace)

```
const response = await this.service.updateTodo({ status, id });
return processResponse(200, response);
} catch (err: any) {
return handleServiceErrors(err);
}

async deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
try {
const { id } = params.pathParameters;
await this.service.deleteTodo({ id });
return processResponse(202, []);
} catch (err: any) {
return handleServiceErrors(err);
}
}
```

Code File Edit Selection View Go Run Terminal Window Help

createTodo.ts – Untitled (Workspace)

```
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
console.log("createTodo lambda executed", event);

return {
body: "createTodo called",
statusCode: 200
};
};


```

Code File Edit Selection View Go Run Terminal Window Help

createTodo.ts – Untitled (Workspace)

```
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
import { ITodoController } from "../controllers/todo.controller";
import { container } from "../ioc/container";
import { ContainerKeys } from "../ioc/keys";
import { ILoggerService } from "../services/logger.service";

const logger: ILoggerService = container.get(ContainerKeys.ILoggerService);
const controller: ITodoController = container.get(ContainerKeys.ITodoController);

export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
logger.log("Lambda handler event", { event });
return await controller.createTodo(event);
};
```

Code File Edit Selection View Go Run Terminal Window Help

getTodo.ts – Untitled (Workspace)

```
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
console.log("getTodo lambda executed", event);

return {
body: "getTodo called",
statusCode: 200
};
};
```

```
container.ts U todo.controller.ts U createTodo.ts U getTodo.ts U
todo-app > backend > lambda > createTodo.ts > ...
14 import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
15 import { ITodoController } from "../controllers/todo.controller";
16 import { container } from "../ioc/container";
17 import { ContainerKeys } from "../ioc/keys";
18 import { ILoggerService } from "../services/logger.service";
19
20 const logger: ILoggerService = container.get(ContainerKeys.ILoggerService);
21 const controller: ITodoController = container.get(ContainerKeys.ITodoController);
22
23 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
24   logger.log("Lambda handler event", { event });
25
26   return await controller.createTodo(event);
27 }
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
```

```
container.ts U todo.controller.ts U createTodo.ts U getTodo.ts U
todo-app > backend > lambda > getTodo.ts > ...
1 import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
2
3 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
4   console.log("getTodo lambda executed", event);
5
6   return {
7     body: "getTodo called",
8     statusCode: 200
9   };
10 }
```

```
container.ts U todo.controller.ts U createTodo.ts U getTodo.ts U
todo-app > backend > lambda > createTodo.ts > ...
14 import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
15 import { ITodoController } from "../controllers/todo.controller";
16 import { container } from "../ioc/container";
17 import { ContainerKeys } from "../ioc/keys";
18 import { ILoggerService } from "../services/logger.service";
19
20 const logger: ILoggerService = container.get(ContainerKeys.ILoggerService);
21 const controller: ITodoController = container.get(ContainerKeys.ITodoController);
22
23 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
24   logger.log("Lambda handler event", { event });
25
26   return await controller.createTodo(event);
27 }
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
49
```

```
container.ts U todo.controller.ts U createTodo.ts U getTodo.ts U
todo-app > backend > lambda > getTodo.ts > <handler>
12 import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
13 //import { ITodoController } from "../controllers/todo.controller";
14 import { container } from "../ioc/container";
15 import { ContainerKeys } from "../ioc/keys";
16 import { ILoggerService } from "../services/logger.service";
17
18 const logger: ILoggerService = container.get(ContainerKeys.ILoggerService);
19 const controller: ITodoController = container.get(ContainerKeys.ITodoController);
20
21 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
22   logger.log("Lambda handler event", { event });
23
24   return await controller.getTodo();
25 }
26
27
28
29
30
31
32
33
34
35
36
37
38
39
39
```

```

updateTodo.ts - Untitled (Workspace)

13
12 import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
11 import { ITodoController } from "../controllers/todo.controller";
10 import { container } from "../ioc/container";
9 import { ContainerKeys } from "../ioc/keys";
8 import { ILoggerService } from "../services/logger.service";
7
6 const logger: ILoggerService = container.get(ContainerKeys.ILoggerService);
5 const controller: ITodoController = container.get(ContainerKeys.ITodoController);
4
3 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
2   logger.log("Lambda handler event", { event });
1
1   return await controller.updateTodo(event);
2

```

```

deleteTodo.ts - Untitled (Workspace)

12 import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
11 import { ITodoController } from "../controllers/todo.controller";
10 import { container } from "../ioc/container";
9 import { ContainerKeys } from "../ioc/keys";
8 import { ILoggerService } from "../services/logger.service";
7
6 const logger: ILoggerService = container.get(ContainerKeys.ILoggerService);
5 const controller: ITodoController = container.get(ContainerKeys.ITodoController);
4
3 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
2   logger.log("Lambda handler event", { event });
1
1   return await controller.deleteTodo(event);
2

```

We have now completed implementing the code, let us see the application folder structure using separation of concerns that divides the code into different layers having different responsibilities

```

createTodo.ts - Untitled (Workspace)

1 import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
1 import { ITodoController } from "../controllers/todo.controller";
2 import { container } from "../ioc/container";
3 import { ContainerKeys } from "../ioc/keys";
4 import { ILoggerService } from "../services/logger.service";
5
6 const logger: ILoggerService = container.get(ContainerKeys.ILoggerService);
7 const controller: ITodoController = container.get(ContainerKeys.ITodoController);
8
9 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
10   logger.log("Lambda handler event", { event });
11
12   return await controller.createTodo(event);
13
14

```

The lambda folder contains lambda functions as entry points into our application. Each lambda follows the concept of single responsibility for a single job. We invoke the controller from each lambda function

```

todo.controller.ts - Untitled (Workspace)

④ todo-app > backend > controllers > todo.controller.ts > o ITodoController > createTodo
1  export interface ITodoController {
2    createTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>; > true
3    getTodo(): Promise<APIGatewayProxyResult>;
4    updateTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
5    deleteTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
6  }
7
8  @injectable()
9  export class TodoController implements ITodoController {
10    constructor(@inject(ContainerKeys.ITodoService) private service: ITodoService) {}
11
12    async createTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult> {
13      try {
14        const { body } = params;
15        if (body === null) throw new Error("Missing request body");
16        const { content } = JSON.parse(body);
17        if (!content) throw new Error("Missing content in request body");
18        const response = await this.service.createTodo({ content });
19
20        return processResponse(201, response);
21      } catch (err: any) {
22        return handleServiceErrors(err);
23      }
24    }
25
26    async getTodo(): Promise<APIGatewayProxyResult> {
27      try {
28        const response = await this.service.getTodo();
29        return processResponse(200, response);
30      } catch (err: any) {
31        return handleServiceErrors(err);
32      }
33    }
34
35    async updateTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult> {
36      try {
37        const { body } = params;
38
39        return processResponse(200, response);
40      } catch (err: any) {
41        return handleServiceErrors(err);
42      }
43    }
44
45    async deleteTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult> {
46      try {
47        const { body } = params;
48
49        return processResponse(204, response);
50      } catch (err: any) {
51        return handleServiceErrors(err);
52      }
53    }
54  }
55
56  @injectable()
57  export class TodoController implements ITodoController {
58    constructor(@inject(ContainerKeys.ITodoService) private service: ITodoService) {}
59
60    async createTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult> {
61      try {
62        const { body } = params;
63        if (body === null) throw new Error("Missing request body");
64        const { content } = JSON.parse(body);
65        if (!content) throw new Error("Missing content in request body");
66        const response = await this.service.createTodo({ content });
67
68        return processResponse(201, response);
69      } catch (err: any) {
70        return handleServiceErrors(err);
71      }
72    }
73
74    async getTodo(): Promise<APIGatewayProxyResult> {
75      try {
76        const response = await this.service.getTodo();
77        return processResponse(200, response);
78      } catch (err: any) {
79        return handleServiceErrors(err);
80      }
81    }
82
83    async updateTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult> {
84      try {
85        const { body } = params;
86
87        return processResponse(200, response);
88      } catch (err: any) {
89        return handleServiceErrors(err);
90      }
91    }
92
93    async deleteTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult> {
94      try {
95        const { body } = params;
96
97        return processResponse(204, response);
98      } catch (err: any) {
99        return handleServiceErrors(err);
100       }
101    }
102  }

```

Ln 10, Col 3 Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live ✅ Prettier ⌂

The controller contains our application functionalities, it sends back the response and handle any errors that occur

```

todo.service.ts - Untitled (Workspace)

④ todo-app > backend > services > todo.service.ts > TodoService > constructor
1  export interface ITodoService {
2    createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
3    getTodo(): Promise<ITodo[]>;
4    updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
5    deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
6  }
7
8  @injectable()
9  export class TodoService implements ITodoService {
10    constructor(
11      @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
12      @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
13    ) {}
14
15    async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
16      this.logger.log("createTodo() called", { params }, this.constructor.name);
17
18      const { content } = params;
19      const createdTodo: ITodo = {
20        type: "Todo",
21        id: uid(),
22        content,
23        status: Status.CREATED,
24        created: new Date().toISOString(),
25        updated: new Date().toISOString()
26      };
27
28      const response = await this.repo.createTodo(createdTodo);
29
30      this.logger.log("dynamo response", { response }, this.constructor.name);
31
32      return createdTodo;
33    }
34
35    async getTodo(): Promise<ITodo[] | []> {
36      this.logger.log("getTodo() called");
37
38      const response = await this.repo.getTodo();
39
40      return response;
41    }
42
43    async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
44      this.logger.log("updateTodo() called", { params }, this.constructor.name);
45
46      const { id, content } = params;
47
48      const updatedTodo: ITodo = {
49        id,
50        content,
51        status: Status.UPDATED,
52        updated: new Date().toISOString()
53      };
54
55      const response = await this.repo.updateTodo(updatedTodo);
56
57      this.logger.log("dynamo response", { response }, this.constructor.name);
58
59      return response;
60    }
61
62    async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
63      this.logger.log("deleteTodo() called", { params }, this.constructor.name);
64
65      const { id } = params;
66
67      const response = await this.repo.deleteTodo(id);
68
69      this.logger.log("dynamo response", { response }, this.constructor.name);
70
71      return response;
72    }
73
74  }
75
76  @injectable()
77  export class TodoService implements ITodoService {
78    constructor(
79      @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
80      @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
81    ) {}
82
83    async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
84      this.logger.log("createTodo() called", { params }, this.constructor.name);
85
86      const { content } = params;
87
88      const createdTodo: ITodo = {
89        type: "Todo",
90        id: uid(),
91        content,
92        status: Status.CREATED,
93        created: new Date().toISOString(),
94        updated: new Date().toISOString()
95      };
96
97      const response = await this.repo.createTodo(createdTodo);
98
99      this.logger.log("dynamo response", { response }, this.constructor.name);
100
101      return createdTodo;
102    }
103
104    async getTodo(): Promise<ITodo[] | []> {
105      this.logger.log("getTodo() called");
106
107      const response = await this.repo.getTodo();
108
109      return response;
110    }
111
112    async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
113      this.logger.log("updateTodo() called", { params }, this.constructor.name);
114
115      const { id, content } = params;
116
117      const updatedTodo: ITodo = {
118        id,
119        content,
120        status: Status.UPDATED,
121        updated: new Date().toISOString()
122      };
123
124      const response = await this.repo.updateTodo(updatedTodo);
125
126      this.logger.log("dynamo response", { response }, this.constructor.name);
127
128      return response;
129    }
130
131    async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
132      this.logger.log("deleteTodo() called", { params }, this.constructor.name);
133
134      const { id } = params;
135
136      const response = await this.repo.deleteTodo(id);
137
138      this.logger.log("dynamo response", { response }, this.constructor.name);
139
140      return response;
141    }
142
143  }

```

Ln 21, Col 6 Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live ✅ Prettier ⌂

The controller uses the service layer that contains the main business logic

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "UNLISTED (WORKSPACE)".
- Code Editor:** Displays the file `todo.repository.ts` (line 29).
- Code Content:**

```
1 import { inject, injectable } from "inversify";
2 import { ContainerKeys } from "../ioc/keys";
3 import { Status } from "../models/todo.model";
4 import { loggerService } from "../services/logger.service";
5
6 export interface ITodoRepository {
7   createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
8   getTodo(): Promise<QueryCommandOutput>;
9   getTodoById(id: string): Promise<QueryCommandOutput>;
10  updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput>;
11  deleteTodo(id: string): Promise<DeleteItemCommandOutput>;
12}
13
14 @injectable()
15 export class TodoRepository implements ITodoRepository {
16   private tableName = "to-do_db";
17   private partitionKey = "type";
18   private sortKey = "id";
19   constructor(
20     @inject(ContainerKeys.ILoggerService) protected logger: LoggerService,
21     @inject(DynamoDBClient) protected client: DynamoDBClient
22   ) {}
23
24   async createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
25     this.logger.log(`createTodo() called`, params, this.constructor.name);
26
27     const putItem: any = {};
28     for (const key in params) {
29       putItem[key] = {
30         S: params[key]
31     };
32   }
33
34   const input: PutItemCommandInput = {
35     TableName: this.tableName,
36     Item: putItem
37   };
38 }
```

- Status Bar:** Shows "Ln 29, Col 1" and other standard status bar information.

The repository layer contains any database related logic

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "UNLISTED (WORKSPACE)".
- Code Editor:** Displays the file `todo.model.ts` (line 13).
- Code Content:**

```
1
2 export enum Status {
3   CREATED = "CREATED",
4   IN_PROGRESS = "IN_PROGRESS",
5   COMPLETED = "COMPLETED"
6 }
7
8 export interface ITodo {
9   type: string;
10  id: string;
11  content: string;
12  status: Status;
13  created: string;
14  updated: string;
15}
```

The model layer contains our business objects and data structures for our application

```

todo.service.ts — Untitled (Workspace)
1 export interface ITodoService {
2   createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
3   getTodo(): Promise<ITodo[]>;
4   updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
5   deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
6 }
7
8 @injectable()
9 export class TodoService implements ITodoService {
10   constructor(
11     @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
12     @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
13   ) {}
14
15   async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
16     this.logger.log("createTodo() called", { params }, this.constructor.name);
17
18     const { content } = params;
19     const createdTodo: ITodo = {
20       type: "Todo",
21       id: uid(),
22       content,
23       status: Status.CREATED,
24       created: new Date().toISOString(),
25       updated: new Date().toISOString()
26     };
27
28     const response = await this.repo.createTodo(createdTodo);
29
30     this.logger.log("dynamo response", { response }, this.constructor.name);
31     return createdTodo;
32   }
33
34   async getTodo(): Promise<ITodo[] | []> {
35     this.logger.log("getTodo() called");
36
37     const response = await this.repo.getTodo();
38
39     return response;
40   }
41
42   async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
43     this.logger.log("updateTodo() called", { params }, this.constructor.name);
44
45     const { content } = params;
46     const updatedTodo: ITodo = {
47       type: "Todo",
48       id: uid(),
49       content,
50       status: Status.UPDATED,
51       created: new Date().toISOString(),
52       updated: new Date().toISOString()
53     };
54
55     const response = await this.repo.updateTodo(updatedTodo);
56
57     this.logger.log("dynamo response", { response }, this.constructor.name);
58     return response;
59   }
60
61   async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
62     this.logger.log("deleteTodo() called", { params }, this.constructor.name);
63
64     const { id } = params;
65
66     const response = await this.repo.deleteTodo(id);
67
68     this.logger.log("dynamo response", { response }, this.constructor.name);
69   }
70 }

```

We also used the concept of inversion of control where we try to couple the dependency between processes by invoking the control flow. Above uses 2 dependencies for the Logger and Repository.

```

todo.service.ts — Untitled (Workspace)
1 export interface ITodoService {
2   createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
3   getTodo(): Promise<ITodo[]>;
4   updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
5   deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
6 }
7
8 @injectable()
9 export class TodoService implements ITodoService {
10   constructor(
11     @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
12     @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
13   ) {
14     this.logger = new LoggerService();
15   }
16
17   async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
18     this.logger.log("createTodo() called", { params }, this.constructor.name);
19
20     const { content } = params;
21     const createdTodo: ITodo = {
22       type: "Todo",
23       id: uid(),
24       content,
25       status: Status.CREATED,
26       created: new Date().toISOString(),
27       updated: new Date().toISOString()
28     };
29
30     const response = await this.repo.createTodo(createdTodo);
31
32     this.logger.log("dynamo response", { response }, this.constructor.name);
33     return createdTodo;
34   }
35
36   async getTodo(): Promise<ITodo[] | []> {
37     this.logger.log("getTodo() called");
38
39     const response = await this.repo.getTodo();
40
41     return response;
42   }
43
44   async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
45     this.logger.log("updateTodo() called", { params }, this.constructor.name);
46
47     const { content } = params;
48     const updatedTodo: ITodo = {
49       type: "Todo",
50       id: uid(),
51       content,
52       status: Status.UPDATED,
53       created: new Date().toISOString(),
54       updated: new Date().toISOString()
55     };
56
57     const response = await this.repo.updateTodo(updatedTodo);
58
59     this.logger.log("dynamo response", { response }, this.constructor.name);
60     return response;
61   }
62
63   async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
64     this.logger.log("deleteTodo() called", { params }, this.constructor.name);
65
66     const { id } = params;
67
68     const response = await this.repo.deleteTodo(id);
69
70     this.logger.log("dynamo response", { response }, this.constructor.name);
71   }
72 }

```

We can also do as above without IoC using the interfaces.

```

todo-app > backend > repositories > todo.repository.ts > TodoRepository
15 import { inject, injectable } from 'inversify';
14 import { ContainerKeys } from '../ioc/keys';
13 import { Status } from './models/todo.model';
12 import { LoggerService } from '../services/logger.service';
11
10 export interface ITodoRepository {
9  createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
8   getTodo(): Promise<QueryCommandOutput>;
7   getTodoById(id: string): Promise<QueryCommandOutput>;
6   updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput>;
5   deleteTodo(id: string): Promise<DeleteItemCommandOutput>;
4 }
3
2 @injectable()
1 import class TodoRepository implements ITodoRepository {
31  private tableName = "to-do_db";
1  private partitionKey = "type";
2  private sortKey = "id";
3  constructor(
4    @inject(ContainerKeys.ILoggerService) protected logger: LoggerService,
5    @inject(DynamoDBClient) protected client: DynamoDBClient
6  ) {}
7
8  async createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
9    this.logger.log(`createTodo() called`, params, this.constructor.name);
10
11  const putItem: any = {};
12  for (const key in params) {
13    putItem[key] = {
14      S: params[key]
15    };
16  }
17
18  const input: PutItemCommandInput = {
19    TableName: this.tableName,
20    Item: putItem
21  };

```

Ln 31, Col 1 (02 selected) Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live ⚡ Prettier

```

todo-app > backend > services > todo.service.ts > TodoService > constructor
12 export interface ITodo {
11  createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
10  getTodo(): Promise<ITodo[]>;
9   updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
8   deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
7 }
6
5 @injectable()
4 export class TodoService implements ITodoService {
3  constructor(
2    @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
1    @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
21 )
1
2  async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
3    this.logger.log(`createTodo() called`, { params }, this.constructor.name);
4
5    const { content } = params;
6    const createdTodo: ITodo = {
7      type: "Todo",
8      id: uid(),
9      content,
10     status: Status.CREATED,
11     created: new Date().toISOString(),
12     updated: new Date().toISOString()
13   };
14   const response = await this.repo.createTodo(createdTodo);
15
16   this.logger.log(`dynamo response`, { response }, this.constructor.name);
17   return createdTodo;
18 }
19
20  async getTodo(): Promise<ITodo[] | []> {
21    this.logger.log(`getTodo() called`);
22
23  const response = await this.repo.getTodo();

```

Ln 21, Col 1 (344 selected) Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live ⚡ Prettier

The TodoService doesn't know about the database but only knows about the **exported** Repository interface methods

```

todo.repository.ts - Untitled (Workspace)
todo.repository.ts U todo.service.ts U todo.models.ts U
todo.repository.ts U

todo-app > backend > repositories > todo.repository.ts > TodoRepository
9 export interface ITodoRepository {
8   createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
7   getTodo(): Promise<QueryCommandOutput>;
6   getTodoById(id: string): Promise<QueryCommandOutput>;
5   updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput>;
4   deleteTodo(id: string): Promise<DeleteItemCommandOutput>;
3
2
1

@injectable()
export class TodoRepository implements ITodoRepository {
  private tableName = "to-do_db";
  private partitionKey = "type";
  private sortKey = "id";
  constructor(
    @inject(ContainerKeys.ILoggerService) protected logger: LoggerService,
    @inject(DynamoDBClient) protected client: DynamoDBClient
  ) {}

  async createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
    this.logger.log("createTodo() called", { params }, this.constructor.name);
    const putItem: any = {};
    for (const key in params) {
      putItem[key] = {
        S: params[key]
      };
    }

    const input: PutItemCommandInput = {
      TableName: this.tableName,
      Item: putItem
    };
    const command = new PutItemCommand(input);
    this.logger.log("createTodo() finished");
    return await this.client.send(command);
  }

  async getTodo(): Promise<ITodo[] | []> {
    this.logger.log("getTodo() called");
    const response = await this.repo.getTodo();
    return response;
  }
}

todo.service.ts U
todo.service.ts U todo.repository.ts U
todo.service.ts U

todo-app > backend > services > todo.service.ts > TodoService > constructor
13 export interface ITodoService {
12   createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
11   getTodo(): Promise<ITodo[]>;
10   updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
9   deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
8
7
6
5
4
3
2
1

@injectable()
export class TodoService implements ITodoService {
  constructor(
    @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
    @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
  ) {}

  async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
    this.logger.log("createTodo() called", { params }, this.constructor.name);
    const { content } = params;
    const createdTodo: ITodo = {
      type: "Todo",
      id: uid(),
      content,
      status: Status.CREATED,
      created: new Date().toISOString(),
      updated: new Date().toISOString()
    };
    const response = await this.repo.createTodo(createdTodo);

    this.logger.log("dynamo response", { response }, this.constructor.name);
    return createdTodo;
  }

  async getTodo(): Promise<ITodo[] | []> {
    this.logger.log("getTodo() called");
    const response = await this.repo.getTodo();
    return response;
  }
}

```

```

todo.repository.ts - Untitled (Workspace)
todo.repository.ts U todo.service.ts U todo.models.ts U
todo.repository.ts U

todo-app > backend > repositories > todo.repository.ts > TodoRepoMongo
13 import { inject, injectable } from 'inversify';
12 import { ContainerKeys } from '../ioc/keys';
11 import { Status } from '../models/todo.model';
10 import { LoggerService } from '../services/logger.service';
9
8
7
6
5
4
3
2
1

export interface ITodoRepository {
  createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
  getTodo(): Promise<QueryCommandOutput>;
  getTodoById(id: string): Promise<QueryCommandOutput>;
  updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput>;
  deleteTodo(id: string): Promise<DeleteItemCommandOutput>;
}

export class TodoRepoMongo implements ITodoRepository {
  createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
    throw new Error("Method not implemented.");
  }
  getTodo(): Promise<QueryCommandOutput> {
    throw new Error("Method not implemented.");
  }
  getTodoById(id: string): Promise<QueryCommandOutput> {
    throw new Error("Method not implemented.");
  }
  updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
    throw new Error("Method not implemented.");
  }
  deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
    throw new Error("Method not implemented.");
  }
}

@injectable()
export class TodoRepository implements ITodoRepository {
  private tableName = "to-do_db";
  private partitionKey = "type";
  private sortKey = "id";
  constructor(
    @inject(ContainerKeys.ILoggerService) protected logger: LoggerService,
    @inject(DynamoDBClient) protected client: DynamoDBClient
  ) {}

  async createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
    this.logger.log("createTodo() called", { params }, this.constructor.name);
    const putItem: any = {};
    for (const key in params) {
      putItem[key] = {
        S: params[key]
      };
    }

    const input: PutItemCommandInput = {
      TableName: this.tableName,
      Item: putItem
    };
    const command = new PutItemCommand(input);
    this.logger.log("createTodo() finished");
    return await this.client.send(command);
  }

  async getTodo(): Promise<ITodo[] | []> {
    this.logger.log("getTodo() called");
    const response = await this.repo.getTodo();
    return response;
  }
}

todo.service.ts U
todo.service.ts U todo.repository.ts U
todo.service.ts U

todo-app > backend > services > todo.service.ts > TodoService > constructor
13 export interface ITodoService {
12   createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
11   getTodo(): Promise<ITodo[]>;
10   updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
9   deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
8
7
6
5
4
3
2
1

@injectable()
export class TodoService implements ITodoService {
  constructor(
    @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
    @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
  ) {}

  async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
    this.logger.log("createTodo() called", { params }, this.constructor.name);
    const { content } = params;
    const createdTodo: ITodo = {
      type: "Todo",
      id: uid(),
      content,
      status: Status.CREATED,
      created: new Date().toISOString(),
      updated: new Date().toISOString()
    };
    const response = await this.repo.createTodo(createdTodo);

    this.logger.log("dynamo response", { response }, this.constructor.name);
    return createdTodo;
  }

  async getTodo(): Promise<ITodo[] | []> {
    this.logger.log("getTodo() called");
    const response = await this.repo.getTodo();
    return response;
  }
}

```

We can also change actual implementation

```
createTodo.ts todo.service.ts todo.models.ts
todo.repository.ts container.ts

import "reflect-metadata";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { Container } from "inversify";
import { ITodoController, TodoController } from "../controllers/todo.controller";
import { ITodoRepository, TodoRepository } from "../repositories/todo.repository";
import { ILoggerService, LoggerService } from "../services/logger.service";
import { ITodoService, TodoService } from "../services/todo.service";
import { ContainerKeys } from "./keys";

const container = new Container();

try {
    const dynamoClient = new DynamoDBClient({ region: "ap-southeast-2" });
    container.bind(DynamoDBClient).toConstantValue(dynamoClient);
    container.bind	ILoggerService>(ContainerKeys.ILoggerService).to(LoggerService);
    container.bind_ITodoRepository>(ContainerKeys.ITodoRepository).to(TodoRepository);
    container.bind_ITodoService>(ContainerKeys.ITodoService).to(TodoService);
    container.bind_ITodoController>(ContainerKeys.ITodoController).to(TodoController);
} catch (err) {
    console.log("Error during initialization", err);
}

export { container };
```

```
createTodo.ts todo.service.ts todo.models.ts
todo.repository.ts container.ts

import "reflect-metadata";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { Container } from "inversify";
import { ITodoController, TodoController } from "../controllers/todo.controller";
import { ITodoRepository, TodoRepository } from "../repositories/todo.repository";
import { ILoggerService, LoggerService } from "../services/logger.service";
import { ITodoService, TodoService } from "../services/todo.service";
import { ContainerKeys } from "./keys";

const container = new Container();

try {
    const dynamoClient = new DynamoDBClient({ region: "ap-southeast-2" });
    container.bind(DynamoDBClient).toConstantValue(dynamoClient);
    container.bind	ILoggerService>(ContainerKeys.ILoggerService).to(LoggerService);
    container.bind_ITodoRepository>(ContainerKeys.ITodoRepository).to(TodoRepository);
    container.bind_ITodoService>(ContainerKeys.ITodoService).to(TodoService);
    container.bind_ITodoController>(ContainerKeys.ITodoController).to(TodoController);
} catch (err) {
    console.log("Error during initialization", err);
}

export { container };
```

```
todo.service.ts
todo.repository.ts
```

```
todo.service.ts
1 import { inject, injectable } from "inversify";
2 import { ContainerKeys } from "../ioc/keys";
3 import { Status } from "../models/todo.model";
4 import { ILoggerService } from "../services/logger.service";
5 import { ITodoRepository } from "./ITodoRepository";
6 import { TodoService } from "./TodoService";
7
8 @injectable()
9 export class TodoService implements ITodoService {
10   constructor(
11     @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
12     @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
13   ) {}
14
15   async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
16     this.logger.log("createTodo() called", { params }, this.constructor.name);
17
18     const { content } = params;
19     const createdTodo: ITodo = {
20       type: "Todo",
21       id: uid(),
22       content,
23       status: Status.CREATED,
24       created: new Date().toISOString(),
25       updated: new Date().toISOString()
26     };
27
28     const response = await this.repo.createTodo(createdTodo);
29
30     this.logger.log("dynamo response", { response }, this.constructor.name);
31     return createdTodo;
32   }
33
34   async getTodo(): Promise<ITodo[] | []> {
35     this.logger.log("getTodo() called");
36
37     const response = await this.repo.getTodo();
38
39     return response;
40   }
}
todo.repository.ts
1 import { flet } from "flet";
2 import { dynamoDBClient } from "@aws-sdk/client-dynamodb";
3 import { TodoController, TodoController } from "../controllers/todo.controller";
4 import { TodoRepository, TodoRepository } from "../repositories/todo.repository";
5 import { LoggerService, LoggerService } from "../services/logger.service";
6 import { TodoService, TodoService } from "../services/todo.service";
7 import { ContainerKeys } from "../ioc/keys";
8
9 const container = new Container();
10
11 const dynamoClient = new DynamoDBClient({ region: "ap-southeast-2" });
12 r.bind(DynamoDBClient).toConstantValue(dynamoClient);
13 r.bind(ILoggerService)(ContainerKeys.ILoggerService).to(LoggerService);
14 r.bind(ITodoRepository)(ContainerKeys.ITodoRepository).to(TodoRepository);
15 r.bind(ITodoService)(ContainerKeys.ITodoService).to(TodoService);
16 r.bind(ITodoController)(ContainerKeys.ITodoController).to(TodoController);
17
18 rr {
19   log("Error during initialization", err);
20 }
21
22 container;
23
```

```
todo.service.ts
todo.repository.ts
```

```
todo.service.ts
todo.repository.ts
```

```
todo.service.ts
1 import { inject, injectable } from "inversify";
2 import { ContainerKeys } from "../ioc/keys";
3 import { Status } from "../models/todo.model";
4 import { ILoggerService } from "../services/logger.service";
5 import { ITodoRepository } from "./ITodoRepository";
6 import { TodoService } from "./TodoService";
7
8 @injectable()
9 export class TodoService implements ITodoService {
10   constructor(
11     @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
12     @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
13   ) {}
14
15   async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
16     this.logger.log("createTodo() called", { params }, this.constructor.name);
17
18     const { content } = params;
19     const createdTodo: ITodo = {
20       type: "Todo",
21       id: uid(),
22       content,
23       status: Status.CREATED,
24       created: new Date().toISOString(),
25       updated: new Date().toISOString()
26     };
27
28     const response = await this.repo.createTodo(createdTodo);
29
30     this.logger.log("dynamo response", { response }, this.constructor.name);
31     return createdTodo;
32   }
33
34   async getTodo(): Promise<ITodo[] | []> {
35     this.logger.log("getTodo() called");
36
37     const response = await this.repo.getTodo();
38
39     return response;
40   }
}
todo.repository.ts
1 import { inject, injectable } from "inversify";
2 import { ContainerKeys } from "../ioc/keys";
3 import { Status } from "../models/todo.model";
4 import { ILoggerService } from "../services/logger.service";
5
6 export interface ITodoRepository {
7   createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
8   getTodo(): Promise<QueryCommandOutput>;
9   getTodoById(id: string): Promise<QueryCommandOutput>;
10  updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput>;
11  deleteTodo(id: string): Promise<DeleteItemCommandOutput>;
12}
13
14 export class TodoRepoMongo implements ITodoRepository {
15   createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
16     throw new Error("Method not implemented.");
17   }
18   getTodo(): Promise<QueryCommandOutput> {
19     throw new Error("Method not implemented.");
20   }
21   getTodoById(id: string): Promise<QueryCommandOutput> {
22     throw new Error("Method not implemented.");
23   }
24   updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
25     throw new Error("Method not implemented.");
26   }
27   deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
28     throw new Error("Method not implemented.");
29   }
}
30
31 @injectable()
32 export class TodoRepository implements ITodoRepository {
33   private tableName = "to-do_db";
34   private partitionKey = "type";
35   private sortKey = "id";
36   constructor(
37     @inject(ContainerKeys.TodoRepoMongo) private repo: TodoRepoMongo
38   ) {}
39
40   createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
41     return this.repo.createTodo(params);
42   }
43
44   getTodo(): Promise<ITodo[] | []> {
45     return this.repo.getTodo();
46   }
47
48   getTodoById(id: string): Promise<ITodo | null> {
49     return this.repo.getTodoById(id);
50   }
51
52   updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
53     return this.repo.updateTodo(params);
54   }
55
56   deleteTodo(id: string): Promise<void> {
57     return this.repo.deleteTodo(id);
58   }
}
```

```

createTodos.ts U todo.service.ts U todo.model.ts U
todo-app > backend > services > todo.service.ts > TodoService > constructor
12 export interface ITodo { true }
11 createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
10 getTodo(): Promise<ITodo[]>;
9 updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
8 deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
7 }

5 @Injectable()
4 export class TodoService implements ITodoService {
3 constructor(
2 @Inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
1 @Inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
) {}

1 async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
2 this.logger.log("createTodo() called", { params }, this.constructor.name)
3

5 const { content } = params;
6 const createdTodo: ITodo = {
7 type: "Todo",
8 id: uid(),
9 content,
10 status: Status.CREATED,
11 created: new Date().toISOString(),
12 updated: new Date().toISOString()
13 };
14 const response = await this.repo.createTodo(createdTodo);

15 this.logger.log("dynamo response", { response }, this.constructor.name)
16 return createdTodo;
17 }

18 async getTodo(): Promise<ITodo[] | []> {
19 this.logger.log("getTodo() called");
20

21 const response = await this.repo.getTodo();
22 }

23

```

```

todo.repository.ts U container.ts U
todo-app > backend > repositories > todo.repository.ts > TodoRepoMongo
15 flect<-metadata";
14 dynamoDBClient } from "@aws-sdk/client-dynamodb";
13 ontainer } from "inversify";
12 TodoController, TodoController } from "../controllers/todo.controller";
11 TodoRepository, TodoRepository, TodoRepoMongo } from "../repositories/todo";
10 LoggerService, LoggerService } from "../services/logger.service";
9 TodoService, TodoService } from "../services/todo.service";
8 ontainerKeys } from "./keys";
7

6 ainer = new Container();
5

4
3 namoClient = new DynamoDBClient({ region: "ap-southeast-2" });
2 r.bind(DynamoDBClient).toConstantValue(dynamoClient);
1 r.bind<ILoggerService>(ContainerKeys.ILoggerService).to(LoggerService);
1 r.bind<ITodoRepository>(ContainerKeys.ITodoRepository).to(TodoRepoMongo);
2 r.bind<ITodoService>(ContainerKeys.ITodoService).to(TodoService);
2 r.bind<ITodoController>(ContainerKeys.ITodoController).to(TodoController);
3 rr {
4 log("Error during initialization", err);
5
6 ontainer };

7

```

Ln 16, Col 81 Spaces: 2 UTF-8 LF () TypeScript @ Go Live ✅ Prettier

We can swap things easily as above

```

createTodos.ts U todo.service.ts U todo.model.ts U
todo-app > backend > services > todo.service.ts > TodoService > constructor
12 export interface ITodo { true }
11 createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
10 getTodo(): Promise<ITodo[]>;
9 updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
8 deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
7 }

5 @Injectable()
4 export class TodoService implements ITodoService {
3 constructor(
2 @Inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
1 @Inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
) {}

1 async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
2 this.logger.log("createTodo() called", { params }, this.constructor.name)
3

5 const { content } = params;
6 const createdTodo: ITodo = {
7 type: "Todo",
8 id: uid(),
9 content,
10 status: Status.CREATED,
11 created: new Date().toISOString(),
12 updated: new Date().toISOString()
13 };
14 const response = await this.repo.createTodo(createdTodo);

15 this.logger.log("dynamo response", { response }, this.constructor.name)
16 return createdTodo;
17 }

18 async getTodo(): Promise<ITodo[] | []> {
19 this.logger.log("getTodo() called");
20

21 const response = await this.repo.getTodo();
22 }

23

```

```

todo.repository.ts U container.ts U
todo-app > backend > repositories > todo.repository.ts > TodoRepoMongo
8 export interface ITodoRepository {
7 createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
6 getTodo(): Promise<QueryCommandOutput>;
5 getTodoById(id: string): Promise<QueryCommandOutput>;
4 updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput>;
3 deleteTodo(id: string): Promise<DeleteItemCommandOutput>;
2 }

19 port class TodoRepoMongo implements ITodoRepository{
18 createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
17 throw new Error("Method not implemented.");
16
15 getTodo(): Promise<QueryCommandOutput> {
14 throw new Error("Method not implemented.");
13
12 getTodoById(id: string): Promise<QueryCommandOutput> {
11 throw new Error("Method not implemented.");
10
13 updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
12 throw new Error("Method not implemented.");
11
13 deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
12 throw new Error("Method not implemented.");
11
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

```

Ln 29, Col 1 (629 selected) Spaces: 2 UTF-8 LF () TypeScript @ Go Live ✅ Prettier

We can also have easy testing

```

todo.repository.ts - Untitled (Workspace)
createTodo.ts U todo.service.ts U todo.model.ts U
todo-app > backend > services > todo.service.ts > TodoService > constructor
12 export interface ITodo { ... }
13 createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
14 updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
15 deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
16
17 @injectable()
18 export class TodoService implements ITodoService {
19   constructor(
20     @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
21     @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
22   ) {}
23
24   async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
25     this.logger.log("createTodo() called", { params }, this.constructor.name);
26
27     const { content } = params;
28     const createdTodo: ITodo = {
29       type: "Todo",
30       id: uid(),
31       content,
32       status: Status.CREATED,
33       created: new Date().toISOString(),
34       updated: new Date().toISOString()
35     };
36
37     const response = await this.repo.createTodo(createdTodo);
38
39     this.logger.log("dynamo response", { response }, this.constructor.name);
40
41     return createdTodo;
42   }
43
44   async getTodo(): Promise<ITodo[] | []> {
45     this.logger.log("getTodo() called");
46
47     const response = await this.repo.getTodo();
48
49     return response;
50   }
51
52   const response = await this.repo.getTodo();
53
54   return response;
55 }

todo.repository.ts U container.ts U
todo-app > backend > repositories > todo.repository.ts > TodoRepoDev > tableName
9  export interface ITodoRepository {
10    createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
11    getTodo(): Promise<QueryCommandOutput>;
12    getTodoById(id: string): Promise<QueryCommandOutput>;
13    updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput>;
14    deleteTodo(id: string): Promise<DeleteItemCommandOutput>;
15  }
16
17  export class TodoRepoDev implements ITodoRepository {
18    private tableName = "to-do-dev_db";
19    createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
20      throw new Error("Method not implemented.");
21    }
22    getTodo(): Promise<QueryCommandOutput> {
23      throw new Error("Method not implemented.");
24    }
25    getTodoById(id: string): Promise<QueryCommandOutput> {
26      throw new Error("Method not implemented.");
27    }
28    updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
29      throw new Error("Method not implemented.");
30    }
31    deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
32      throw new Error("Method not implemented.");
33    }
34  }
35
36  @injectable()
37  export class TodoRepository implements ITodoRepository {
38    private tableName = "to-do_db";
39    private partitionKey = "type";
40    private sortKey = "id";
41    constructor(
42      @inject(ContainerKeys.ILoggerService) protected logger: LoggerService,
43      @inject(DynamoDBClient) protected client: DynamoDBClient
44    ) {}
45
46  }

```

Ln 30, Col 35 Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live ⚡ Prettier ⚡

```

todo.repository.ts - Untitled (Workspace)
createTodo.ts U todo.service.ts U todo.model.ts U
todo-app > backend > services > todo.service.ts > TodoService > constructor
12 export interface ITodo { ... }
13 createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
14 updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
15 deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
16
17 @injectable()
18 export class TodoService implements ITodoService {
19   constructor(
20     @inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
21     @inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
22   ) {}
23
24   async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
25     this.logger.log("createTodo() called", { params }, this.constructor.name);
26
27     const { content } = params;
28     const createdTodo: ITodo = {
29       type: "Todo",
30       id: uid(),
31       content,
32       status: Status.CREATED,
33       created: new Date().toISOString(),
34       updated: new Date().toISOString()
35     };
36
37     const response = await this.repo.createTodo(createdTodo);
38
39     this.logger.log("dynamo response", { response }, this.constructor.name);
40
41     return createdTodo;
42   }
43
44   async getTodo(): Promise<ITodo[] | []> {
45     this.logger.log("getTodo() called");
46
47     const response = await this.repo.getTodo();
48
49     return response;
50   }
51
52   const response = await this.repo.getTodo();
53
54   return response;
55 }

todo.repository.ts U container.ts U
todo-app > backend > loc > container >
15  reflect-metadata";
16  dynamoDBClient } from "aws-sdk/client-dynamodb";
17  container } from "inversify";
18  TodoController, TodoController } from "../controllers/todo.controller";
19  TodoRepository, TodoRepository, TodoRepoDev } from "../repositories/todo.repository";
20  LoggerService, LoggerService } from "../services/logger.service";
21  TodoService, TodoService } from "../services/todo.service";
22  containerKeys } from "./keys";
23
24  container = new Container();
25
26  dynamoClient = new DynamoDBClient({ region: "ap-southeast-2" });
27  r.bind(DynamoDBClient).toConstantValue(dynamoClient);
28  r.bind	ILoggerService<(ContainerKeys.ILoggerService).to(LoggerService);
29  r.bind<ITodoRepository>(ContainerKeys.ITodoRepository).to(TodoRepoDev);
30  r.bind<ITodoService>(ContainerKeys.ITodoService).to(TodoService);
31  r.bind<ITodoController>(ContainerKeys.ITodoController).to(TodoController);
32  rr) {
33  log("Error during initialization", err);
34
35  container;
36
37  container;
38
39  container;
40
41  container;
42
43  container;
44
45  container;
46
47  container;
48
49  container;
50
51  container;
52
53  container;
54
55  container;
56
57  container;
58
59  container;
59
60  container;
61
62  container;
63
64  container;
65
66  container;
67
68  container;
69
69
70  container;
71
72  container;
73
74  container;
75
76  container;
77
78  container;
79
79
80  container;
81
82  container;
83
84  container;
85
86  container;
87
88  container;
89
89
90  container;
91
92  container;
93
94  container;
95
96  container;
97
98  container;
99
99
100  container;
101
102  container;
103
104  container;
105
106  container;
107
108  container;
109
109
110  container;
111
112  container;
113
114  container;
115
116  container;
117
117
118  container;
119
119
120  container;
121
122  container;
123
123
124  container;
125
125
126  container;
127
127
128  container;
129
129
130  container;
131
131
132  container;
133
133
134  container;
135
135
136  container;
137
137
138  container;
139
139
140  container;
141
141
142  container;
143
143
144  container;
145
145
146  container;
147
147
148  container;
149
149
150  container;
151
151
152  container;
153
153
154  container;
155
155
156  container;
157
157
158  container;
159
159
160  container;
161
161
162  container;
163
163
164  container;
165
165
166  container;
167
167
168  container;
169
169
170  container;
171
171
172  container;
173
173
174  container;
175
175
176  container;
177
177
178  container;
179
179
180  container;
181
181
182  container;
183
183
184  container;
185
185
186  container;
187
187
188  container;
189
189
190  container;
191
191
192  container;
193
193
194  container;
195
195
196  container;
197
197
198  container;
199
199
200  container;
201
201
202  container;
203
203
204  container;
205
205
206  container;
207
207
208  container;
209
209
210  container;
211
211
212  container;
213
213
214  container;
215
215
216  container;
217
217
218  container;
219
219
220  container;
221
221
222  container;
223
223
224  container;
225
225
226  container;
227
227
228  container;
229
229
230  container;
231
231
232  container;
233
233
234  container;
235
235
236  container;
237
237
238  container;
239
239
240  container;
241
241
242  container;
243
243
244  container;
245
245
246  container;
247
247
248  container;
249
249
250  container;
251
251
252  container;
253
253
254  container;
255
255
256  container;
257
257
258  container;
259
259
260  container;
261
261
262  container;
263
263
264  container;
265
265
266  container;
267
267
268  container;
269
269
270  container;
271
271
272  container;
273
273
274  container;
275
275
276  container;
277
277
278  container;
279
279
280  container;
281
281
282  container;
283
283
284  container;
285
285
286  container;
287
287
288  container;
289
289
290  container;
291
291
292  container;
293
293
294  container;
295
295
296  container;
297
297
298  container;
299
299
300  container;
301
301
302  container;
303
303
304  container;
305
305
306  container;
307
307
308  container;
309
309
310  container;
311
311
312  container;
313
313
314  container;
315
315
316  container;
317
317
318  container;
319
319
320  container;
321
321
322  container;
323
323
324  container;
325
325
326  container;
327
327
328  container;
329
329
330  container;
331
331
332  container;
333
333
334  container;
335
335
336  container;
337
337
338  container;
339
339
340  container;
341
341
342  container;
343
343
344  container;
345
345
346  container;
347
347
348  container;
349
349
350  container;
351
351
352  container;
353
353
354  container;
355
355
356  container;
357
357
358  container;
359
359
360  container;
361
361
362  container;
363
363
364  container;
365
365
366  container;
367
367
368  container;
369
369
370  container;
371
371
372  container;
373
373
374  container;
375
375
376  container;
377
377
378  container;
379
379
380  container;
381
381
382  container;
383
383
384  container;
385
385
386  container;
387
387
388  container;
389
389
390  container;
391
391
392  container;
393
393
394  container;
395
395
396  container;
397
397
398  container;
399
399
400  container;
401
401
402  container;
403
403
404  container;
405
405
406  container;
407
407
408  container;
409
409
410  container;
411
411
412  container;
413
413
414  container;
415
415
416  container;
417
417
418  container;
419
419
420  container;
421
421
422  container;
423
423
424  container;
425
425
426  container;
427
427
428  container;
429
429
430  container;
431
431
432  container;
433
433
434  container;
435
435
436  container;
437
437
438  container;
439
439
440  container;
441
441
442  container;
443
443
444  container;
445
445
446  container;
447
447
448  container;
449
449
450  container;
451
451
452  container;
453
453
454  container;
455
455
456  container;
457
457
458  container;
459
459
460  container;
461
461
462  container;
463
463
464  container;
465
465
466  container;
467
467
468  container;
469
469
470  container;
471
471
472  container;
473
473
474  container;
475
475
476  container;
477
477
478  container;
479
479
480  container;
481
481
482  container;
483
483
484  container;
485
485
486  container;
487
487
488  container;
489
489
490  container;
491
491
492  container;
493
493
494  container;
495
495
496  container;
497
497
498  container;
499
499
500  container;
501
501
502  container;
503
503
504  container;
505
505
506  container;
507
507
508  container;
509
509
510  container;
511
511
512  container;
513
513
514  container;
515
515
516  container;
517
517
518  container;
519
519
520  container;
521
521
522  container;
523
523
524  container;
525
525
526  container;
527
527
528  container;
529
529
530  container;
531
531
532  container;
533
533
534  container;
535
535
536  container;
537
537
538  container;
539
539
540  container;
541
541
542  container;
543
543
544  container;
545
545
546  container;
547
547
548  container;
549
549
550  container;
551
551
552  container;
553
553
554  container;
555
555
556  container;
557
557
558  container;
559
559
560  container;
561
561
562  container;
563
563
564  container;
565
565
566  container;
567
567
568  container;
569
569
570  container;
571
571
572  container;
573
573
574  container;
575
575
576  container;
577
577
578  container;
579
579
580  container;
581
581
582  container;
583
583
584  container;
585
585
586  container;
587
587
588  container;
589
589
590  container;
591
591
592  container;
593
593
594  container;
595
595
596  container;
597
597
598  container;
599
599
600  container;
601
601
602  container;
603
603
604  container;
605
605
606  container;
607
607
608  container;
609
609
610  container;
611
611
612  container;
613
613
614  container;
615
615
616  container;
617
617
618  container;
619
619
620  container;
621
621
622  container;
623
623
624  container;
625
625
626  container;
627
627
628  container;
629
629
630  container;
631
631
632  container;
633
633
634  container;
635
635
636  container;
637
637
638  container;
639
639
640  container;
641
641
642  container;
643
643
644  container;
645
645
646  container;
647
647
648  container;
649
649
650  container;
651
651
652  container;
653
653
654  container;
655
655
656  container;
657
657
658  container;
659
659
660  container;
661
661
662  container;
663
663
664  container;
665
665
666  container;
667
667
668  container;
669
669
670  container;
671
671
672  container;
673
673
674  container;
675
675
676  container;
677
677
678  container;
679
679
680  container;
681
681
682  container;
683
683
684  container;
685
685
686  container;
687
687
688  container;
689
689
690  container;
691
691
692  container;
693
693
694  container;
695
695
696  container;
697
697
698  container;
699
699
700  container;
701
701
702  container;
703
703
704  container;
705
705
706  container;
707
707
708  container;
709
709
710  container;
711
711
712  container;
713
713
714  container;
715
715
716  container;
717
717
718  container;
719
719
720  container;
721
721
722  container;
723
723
724  container;
725
725
726  container;
727
727
728  container;
729
729
730  container;
731
731
732  container;
733
733
734  container;
735
735
736  container;
737
737
738  container;
739
739
740  container;
741
741
742  container;
743
743
744  container;
745
745
746  container;
747
747
748  container;
749
749
750  container;
751
751
752  container;
753
753
754  container;
755
755
756  container;
757
757
758  container;
759
759
760  container;
761
761
762  container;
763
763
764  container;
765
765
766  container;
767
767
768  container;
769
769
770  container;
771
771
772  container;
773
773
774  container;
775
775
776  container;
777
777
778  container;
779
779
780  container;
781
781
782  container;
783
783
784  container;
785
785
786  container;
787
787
788  container;
789
789
790  container;
791
791
792  container;
793
793
794  container;
795
795
796  container;
797
797
798  container;
799
799
800  container;
801
801
802  container;
803
803
804  container;
805
805
806  container;
807
807
808  container;
809
809
810  container;
811
811
812  container;
813
813
814  container;
815
815
816  container;
817
817
818  container;
819
819
820  container;
821
821
822  container;
823
823
824  container;
825
825
826  container;
827
827
828  container;
829
829
830  container;
831
831
832  container;
833
833
834  container;
835
835
836  container;
837
837
838  container;
839
839
840  container;
841
841
842  container;
843
843
844  container;
845
845
846  container;
847
847
848  container;
849
849
850  container;
851
851
852  container;
853
853
854  container;
855
855
856  container;
857
857
858  container;
859
859
860  container;
861
861
862  container;
863
863
864  container;
865
865
866  container;
867
867
868  container;
869
869
870  container;
871
871
872  container;
873
873
874  container;
875
875
876  container;
877
877
878  container;
879
879
880  container;
881
881
882  container;
883
883
884  container;
885
885
886  container;
887
887
888  container;
889
889
890  container;
891
891
892  container;
893
893
894  container;
895
895
896  container;
897
897
898  container;
899
899
900  container;
901
901
902  container;
903
903
904  container;
905
905
906  container;
907
907
908  container;
909
909
910  container;
911
911
912  container;
913
913
914  container;
915
915
916  container;
917
917
918  container;
919
919
920  container;
921
921
922  container;
923
923
924  container;
925
925
926  container;
927
927
928  container;
929
929
930  container;
931
931
932  container;
933
933
934  container;
935
935
936  container;
937
937
938  container;
939
939
940  container;
941
941
942  container;
943
943
944  container;
945
945
946  container;
947
947
948  container;
949
949
950  container;
951
951
952  container;
953
953
954  container;
955
955
956  container;
957
957
958  container;
959
959
960  container;
961
961
962  container;
963
963
964  container;
965
965
966  container;
967
967
968  container;
969
969
970  container;
971
971
972  container;
973
973
974  container;
975
975
976  container;
977
977
978  container;
979
979
980  container;
981
981
982  container;
983
983
984  container;
985
985
986  container;
987
987
988  container;
989
989
990  container;
991
991
992  container;
993
993
994  container;
995
995
996  container;
997
997
998  container;
999
999
1000  container;

```

Ln 16, Col 79 Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live ⚡ Prettier ⚡

We can change the implementation from prod to dev database easily.

Code File Edit Selection View Go Run Terminal Window Help

container.ts — Untitled (Workspace)

```

createTodo.ts U todo.service.ts U todo.model.ts U
todo.repository.ts U container.ts U

④ todo-app > backend > services > todo.service.ts > TodoService > constructor
  ↳ 12 export interface ITodo { true
  11   createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
  10   updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
  9    deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
  8    }
  7  }

  5 @injectable()
  4 export class TodoService implements ITodoService {
  3   constructor(
  2     @Inject(ContainerKeys.ILoggerService) private logger: ILoggerService,
  1     @Inject(ContainerKeys.ITodoRepository) private repo: ITodoRepository
  ) {}

  1
  2   async createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
  3     this.logger.log("createTodo() called", { params }, this.constructor.name)
  4
  5     const { content } = params;
  6     const createdTodo: ITodo = {
  7       type: "Todo",
  8       id: uid(),
  9       content,
  10      status: Status.CREATED,
  11      created: new Date().toISOString(),
  12      updated: new Date().toISOString()
  13    };
  14    const response = await this.repo.createTodo(createdTodo);
  15
  16    this.logger.log("dynamo response", { response }, this.constructor.name)
  17    return createdTodo;
  18  }

  20  async getTodo(): Promise<ITodo[] | []> {
  21    this.logger.log("getTodo() called");
  22
  23    const response = await this.repo.getTodo();
  
```

Ln 13, Col 60 Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live ⚡ Prettier ⚡

Code File Edit Selection View Go Run Terminal Window Help

createTodo.ts — Untitled (Workspace)

```

EXPLORER
UNTITLED (WORKSPACE)
  todo-app
    backend
      contracts
        todo.contracts.ts
      controllers
        todo.controller.ts
      helpers
        error.helper.ts
        response.helper.ts
    loc
      container.ts
      keys.ts
    lambda
      createTodo.ts
      deleteTodo.ts

```

```

createTodo.ts U todo.service.ts U todo.model.ts U
createTodo.ts X

④ todo-app > backend > lambda > createTodos > ...
  ↳ 1 import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
  1 import { ITodoController } from "../controllers/todo.controller";
  2 import { container } from "../ioc/container";
  3 import { ContainerKeys } from "../ioc/keys";
  4 import { ILoggerService } from "../services/logger.service";
  5
  6 const logger: ILoggerService = container.get(ContainerKeys.ILoggerService);
  7 const controller: ITodoController = container.get(ContainerKeys.ITodoController);
  8
  9 export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
  10   logger.log("Lambda hander event", { event });
  11
  12   return await controller.createTodo(event);
  13 };

```

Ln 14, Col 10 Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live ⚡ Prettier ⚡

Code File Edit Selection View Go Run Terminal Window Help

todo.controller.ts — Untitled (Workspace)

```

EXPLORER
UNTITLED (WORKSPACE)
  todo-app
    backend
      contracts
        todo.contracts.ts
      controllers
        todo.controller.ts
      helpers
        error.helper.ts
        response.helper.ts
    loc
      container.ts
      keys.ts
    lambda
      createTodo.ts
      deleteTodo.ts
      getTodo.ts
      health.ts
      updateTodo.ts
    models
      todo.model.ts
    repositories
      todo.repository.ts
    services
      logger.service.ts
      todo.service.ts
    bin
    cdk.out
    lib
      ApiGateway.ts
      Cognito.ts
      Dynamo.ts
      Lambda.ts
      todo-app-stack.ts

```

```

todo.service.ts U createTodo.ts U todo.controller.ts U
todo.controller.ts X

④ todo-app > backend > controllers > todo.controller.ts > ITodoController > createTodo
  ↳ 10 export interface ITodoController {
  11   createTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
  12   getTodo(): Promise<APIGatewayProxyResult>;
  13   updateTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
  14   deleteTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
  15
  16   @injectable()
  17   export class TodoController implements ITodoController {
  18     constructor(@inject(ContainerKeys.ITodoService) private service: ITodoService) {}
  19
  20     async createTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult> {
  21       try {
  22         const { body } = params;
  23         if (body === null) throw new Error("Missing request body");
  24         const { content } = JSON.parse(body);
  25         if (!content) throw new Error("Missing Content in request body");
  26         const response = await this.service.createTodo({ content });
  27
  28         return processResponse(201, response);
  29       } catch (err: any) {
  30         return handleServiceErrors(err);
  31       }
  32     }
  33
  34     async getTodo(): Promise<APIGatewayProxyResult> {
  35       try {
  36         const response = await this.service.getTodo();
  37         return processResponse(200, response);
  38       } catch (err: any) {
  39         return handleServiceErrors(err);
  40       }
  41     }
  42
  43     async updateTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult> {
  44       try {
  45         const { body } = params;
  
```

Ln 10, Col 3 Spaces: 2 UTF-8 LF () TypeScript ⚡ Go Live ⚡ Prettier ⚡

Code Editor showing the `container.ts` file in the `todo-app` workspace. The code uses Inversify and Reflect Metadata for dependency injection.

```
4 import "reflect-metadata";
5 import { DynamoDBClient } from "aws-sdk/client-dynamodb";
6 import { Container } from "inversify";
7 import { ITodoController, TodoController } from "../controllers/todo.controller";
8 import { ITodoRepository, TodoRepository, TodoRepoDev } from "../repositories/todo.repository";
9 import { ILoggerService, LoggerService } from "../services/logger.service";
10 import { ITodoService, TodoService } from "../services/todo.service";
11 import { ContainerKeys } from "./keys";
12
13 const container = new Container();
14
15 // check the env whether is dev or prod
16 // then decide the actual implementation of the dependencies
17 try {
18     const dynamoClient = new DynamoDBClient({ region: "ap-southeast-2" });
19     container.bind(DynamoDBClient).toConstantValue(dynamoClient);
20     container.bind<ILoggerService>(ContainerKeys.ILoggerService).to(LoggerService);
21     container.bind<ITodoRepository>(ContainerKeys.ITodoRepository).to(TodoRepoDev);
22     container.bind<ITodoService>(ContainerKeys.ITodoService).to(TodoService);
23     container.bind<ITodoController>(ContainerKeys.ITodoController).to(TodoController);
24 } catch (err) {
25     console.log("Error during initialization", err);
26 }
27
28 export { container };
29
```

Code Editor showing the `package.json` file in the `todo-app` workspace. It lists dependencies for Inversify, Reflect Metadata, AWS SDK, and other tools.

```
{
  "name": "todo-app",
  "version": "0.1.0",
  "bin": {
    "todo-app": "bin/todo-app.js"
  },
  "scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk"
  },
  "devDependencies": {
    "@types/jest": "^26.0.18",
    "@types/node": "10.17.27",
    "@types/uuid": "8.3.4",
    "aws-cdk": "2.18.0",
    "jest": "^26.4.2",
    "ts-jest": "^26.2.0",
    "ts-node": "9.0.0",
    "typescript": "~3.9.7"
  },
  "dependencies": {
    "@aws-sdk/client-dynamodb": "^3.121.0",
    "@types/aws-lambda": "8.10.98",
    "aws-cdk-lib": "2.18.0",
    "constructs": "3.0.0",
    "inversify": "6.0.1", You, 1 second ago • Uncommitted changes
    "reflect-metadata": "0.1.13",
    "source-map-support": "0.5.16",
    "uuid": "8.3.2"
  }
}
```

For doing this IoC, we use 2 packages called **inversify** and **reflect-metadata** as above

```

{
  "compilerOptions": {
    "target": "ES2018",
    "module": "commonjs",
    "lib": ["es2018"],
    "declaration": true,
    "strict": true,
    "noImplicitAny": true,
    "strictNullChecks": false,
    "noImplicitThis": true,
    "alwaysStrict": true,
    "noUnusedLocals": false,
    "noUnusedParameters": false,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": false,
    "inlineSourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "strictPropertyInitialization": false,
    "typeRoots": ["./node_modules/@types"]
  },
  "exclude": ["node_modules", "cdk.out"]
}

```

We also had to enable 2 things in our tsconfig.json file, the **emitDecoratorMetadata** and the **experimentalDecorators**

```

import { APIGatewayEvent, APIGatewayProxyEvent, APIGatewayProxyResult } from 'aws-lambda';
import { inject, injectable } from 'inversify';
import { handleServiceErrors } from '../helpers/error.helper';
import { processResponse } from '../helpers/response.helper';
import { ContainerKeys } from '../ioc/keys';
import { ILoggerService } from '../services/logger.service';
import { ITodoService } from '../services/todo.service';

export interface ITodoController {
  createTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
  getTodo(): Promise<APIGatewayProxyResult>;
  updateTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
  deleteTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
}

@injectable()
export class TodoController implements ITodoController {
  constructor(@inject(ContainerKeys.ITodoService) private service: ITodoService) {}

  async createTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult> {
    try {
      const { body } = params;
      if (body === null) throw new Error('Missing request body');
      const { content } = JSON.parse(body);
      if (!content) throw new Error('Missing content in request body');
      const response = await this.service.createTodo({ content });
      return processResponse(201, response);
    } catch (err: any) {
      return handleServiceErrors(err);
    }
  }

  async getTodo(): Promise<APIGatewayProxyResult> {
    try {
      ...
    }
  }
}

```

This allows us to use the **@injectable** and **@inject** commands. We can now use the CDK to deploy our CFN stack

```

~/source-code/youtube/todo-app > tutorial_4 !6 712 cdk deploy
[+] Building 0.6s (2/3)
-> [internal] load build definition from Dockerfile
-> [internal] transfer Dockerfile: 37B
-> [internal] load .dockerignore
-> [internal] transfer context: 2B
-> [internal] load metadata for public.ecr.aws/sam/build-nodejs14.x:latest

```

```

test
.gitignore
.npmignore
.prettierc.js
.cdk.json
jest.config.js
package-lock.json
package.json
README.md
tsconfig.json

PROBLEMS OUTPUT TERMINAL GITLENS DEBUG CONSOLE
TodoAppStack

Deployment time: 54.12s
TodoAppStack.ApiGatewayEndpoint5AA8EC3A = https://o5aafy2waa.execute-api.ap-southeast-2.amazonaws.com/prod/
Stack ARN: arn:aws:cloudformation:ap-southeast-2:68379392497:stack/TodoAppStack/a1718250-fc1b-11ec-90f7-02a28d7d074a
Total time: 71.9s
~/source-code/youtube/todo-app > tutorial_4 16:12

```

In 10. Col 3. Spaces: 2. UTF-8. LF. TypeScript. Go Live. Prettier.

We can now try out the API using Postman with the `createTodo` API

Postman

GET `[[URL]]/health`

Key	Value	Description
Authorization	Bearer eyJraWQiOiJcLzVJNWZhakpyZQ4aWE2bnF6eIMNTkwVGFT0ZkWU... no-cache	
Cache-Control	<calculated when request is sent> no-cache	
Postman-Tokens	<calculated when request is sent>	
Host	<calculated when request is sent> PostmanRuntime/7.29.0	
User-Agent	*/* gzip, deflate, br	
Accept	*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	

Status: 200 OK. Time: 177 ms. Size: 488 B. Save Response.

Postman

POST `[[URL]]/todo`

Body	Params	Headers	Tests	Settings
Content:				

```

1 "content": "Do shopping"

```

Status: 201 Created. Time: 623 ms. Size: 850 B. Save Response.

We get a 201 Created response

POST {{URL}}/todo

```

1 {
2   "content": "Clean the house"
3 }

```

Status: 201 Created Time: 91 ms Size: 854 B Save Response

```

1
2   "type": "Todo",
3   "id": "2223ba73-d4c4-4d27-a401-d86fd0a51cf",
4   "content": "Clean the house",
5   "status": "CREATED",
6   "created": "2022-07-06T19:50:31.330Z",
7   "updated": "2022-07-06T19:50:31.330Z"
8

```

POST {{URL}}/todo

```

1 {
2   "content": "Walk the dog"
3 }

```

Status: 201 Created Time: 99 ms Size: 851 B Save Response

```

1
2   "type": "Todo",
3   "id": "149950c7-88fd-4183-a8a0-da57307a5136",
4   "content": "Walk the dog",
5   "status": "CREATED",
6   "created": "2022-07-06T19:50:40.223Z",
7   "updated": "2022-07-06T19:50:40.223Z"
8

```

Next, let us test our **getTodo** API

GET {{URL}}/todo

```

1
2   "content": "Walk the dog",
3   "updated": "2022-07-06T19:50:40.223Z",
4   "status": "CREATED",
5   "created": "2022-07-06T19:50:40.223Z",
6   "id": "149950c7-88fd-4183-a8a0-da57307a5136",
7   "type": "Todo"
8
9   [
10     {
11       "content": "Clean the house",
12       "updated": "2022-07-06T19:50:31.330Z",
13       "status": "CREATED",
14       "created": "2022-07-06T19:50:31.330Z",
15       "id": "2223ba73-d4c4-4d27-a401-d86fd0a51cf",
16       "type": "Todo"
17     },
18     {
19       "content": "Go shopping",
20       "updated": "2022-07-06T19:50:12.381Z",
21       "status": "CREATED",
22       "created": "2022-07-06T19:50:12.381Z",
23       "id": "4165504d-9c77-45e6-af29-60a9b58a5873",
24       "type": "Todo"
25     },
26   ]

```

Status: 200 OK Time: 477 ms Size: 118 KB Save Response

Next, let us test the **updateTodo** API using the POST command so that we get back a response object

POST /todo/149950c7-88fd-4183-a8a0-da57307a5136

```

1
2   "status": "IN_PROGRESS"
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

```

Status: 200 OK Time: 596 ms Size: 850 B Save Response

GET /todo

```

1 [
2   {
3     "content": "Walk the dog",
4     "updated": "2022-07-06T19:50:40.223Z",
5     "status": "IN_PROGRESS",
6     "created": "2022-07-06T19:50:40.223Z",
7     "id": "149950c7-88fd-4183-a8a0-da57307a5136",
8     "type": "Todo"
9   },
10   {
11     "content": "Clean the house",
12     "updated": "2022-07-06T19:50:31.330Z",
13     "status": "CREATED",
14     "created": "2022-07-06T19:50:31.330Z",
15     "id": "2223ba73-d4c4-4d27-a401-d8d6d0a51cf",
16     "type": "Todo"
17   },
18   {
19     "content": "Do shopping",
20     "updated": "2022-07-06T19:50:12.381Z",
21     "status": "CREATED",
22     "created": "2022-07-06T19:50:12.381Z",
23     "id": "41f6594d-9c77-45e6-aff9-6da9b58a5073",
24     "type": "Todo"
25   }
]

```

Status: 200 OK Time: 244 ms Size: 118 KB Save Response

Next, let us test the **deleteTodo** API

DELETE /todo/2223ba73-d4c4-4d27-a401-d8d6d0a51cf

This request does not have a body

Status: 202 Accepted Time: 1077 ms Save Response

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Tutorial, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area displays a collection named 'Todo App' with several API endpoints listed: POST Login, GET Health, POST Create Todo, GET Get Todo, POST Update Todo, and DELETE Delete Todo. A specific GET request to 'Todo App / Get Todo' is selected. The request details panel shows the method as 'GET' and the URL as '[[URL]]/todo'. The 'Body' tab is selected, showing the response body in JSON format:

```
1 [ 2   { 3     "content": "Walk the dog", 4     "updated": "2022-07-06T19:50:40.223Z", 5     "status": "IN_PROGRESS", 6     "created": "2022-07-06T19:50:40.223Z", 7     "id": "149950c7-88fd-4183-aaa0-d457387a5136", 8     "type": "Todo" 9   }, 10  [ 11    { 12      "content": "Do shopping", 13      "updated": "2022-07-06T19:50:12.381Z", 14      "status": "CREATED", 15      "created": "2022-07-06T19:50:12.381Z", 16      "id": "41f65846-9c77-45e6-a1f9-66a9b58a5873", 17      "type": "Todo" 18    ]]
```

The status bar at the bottom indicates 'Status: 200 OK Time: 159 ms Size: 1 KB Save Response'.

The screenshot shows the AWS Console Home page. At the top, there are links for Developer Roadmap, Employee File, AWS Blog, and Working with the... The navigation bar includes services like CloudFormation, API Gateway, CloudWatch, Cognito, DynamoDB, and Lambda. The main content area has sections for 'Recently visited' (DynamoDB, Lambda, CloudWatch, CloudFormation, Cognito, API Gateway, S3) and 'Welcome to AWS' (Getting started with AWS, Training and certification, What's new with AWS?). Below these are sections for 'AWS Health' (Open issues: 0, Scheduled changes: 0, Other notifications: 0) and 'Cost and usage' (No cost and usage). A note says 'This could be because you haven't configured AWS Cost Manager or you do not have permission.'

The screenshot shows the AWS DynamoDB Dashboard. The left sidebar has sections for Dashboard, Tables (Update settings, Explore items, PartiQL editor, Backups, Exports to S3, Reserved capacity, Settings), and DAX (Clusters, Subnet groups, Parameter groups, Events). The main dashboard has two main sections: 'Alarms (0)' and 'DAX clusters (0)'. The 'Create resources' section includes a 'Create table' button and information about Amazon DynamoDB Accelerator (DAX). The 'What's new' section lists updates from March 10, 2022, and March 9, 2022.

**DynamoDB**

**Tables**

Name	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mode	Size	Table class
cloud-native_db	Active	type (S)	id (S)	0	Provisioned (S)	Provisioned (S)	418 bytes	DynamoDB Standard
to-do_db	Active	type (S)	id (S)	0	Provisioned (S)	Provisioned (S)	0 bytes	DynamoDB Standard

**to-do\_db**

**General information**

Partition key type (String)	Sort key id (String)	Capacity mode Provisioned	Table status Active No active alarms
--------------------------------	-------------------------	------------------------------	--

**Items summary**

Item count 0	Table size 0 bytes	Average item size 0 bytes
-----------------	-----------------------	------------------------------

**Table capacity metrics**

**Scan/Query items**

**Items returned (2)**

type	id	content	created	status	updated
Todo	149950c7-88fd-4183...	Walk the dog	2022-07-0...	IN_PROGRE...	2022-07-06T19:50:40.223Z
Todo	41f6584d-9c77-45e6...	Do shopping	2022-07-0...	CREATED	2022-07-06T19:50:12.381Z

https://github.com/inversify/InversifyJS

inversify / InversifyJS

Type  to search

Code Issues 261 Pull requests 11 Discussions Actions Projects 1 Wiki Security Insights

InversifyJS Public Watch 136 Fork 706 Star 10.3k

master 8 branches 86 tags Go to file Add file Code

**dependabot[bot] and PodaruDragos Bump socket.io-parser from 4.0.4 to 4.0.5 (...** 5101217 last week 1,474 commits

	.github/workflows	Fix CI (#1520)
	.vscode	Update typescript compiler options (#1311)
	src	fix typo (LazyServiceIdentifier -> LazyServiceIdentifier) (#1483)
	test	fix typo (LazyServiceIdentifier -> LazyServiceIdentifier) (#1483)
	wiki	fix typo (LazyServiceIdentifier -> LazyServiceIdentifier) (#1483)
	.auditignore	added auditignore as workaround to upgrading to gulp4.0
	.gitignore	Merge branch 'master' of https://github.com/inversify/InversifyJS int...
	.npmignore	Removed source maps from package (#1412)
	.publishrc	Update package.json
	.travis.yml	removed node versions
	CHANGELOG.md	Testing Up to 100% (#1443)
	CODE_OF_CONDUCT.md	Create CODE_OF_CONDUCT.md (#594)
	CONTRIBUTING.md	Circular dependencies (#717)
	ISSUE_TEMPLATE.md	fixes #105

About

A powerful and lightweight inversion of control container for JavaScript & Node.js apps powered by TypeScript.

inversify.io/ nodejs javascript ioc typescript dependency-injection inversifyjs

Readme MIT license Code of conduct Activity 10.3k stars 136 watching 706 forks Report repository

Releases 74

v6.0.1 Latest on Oct 14, 2021 + 73 releases