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

Jack Do
77 subscribers

Subscribed

4 | Share | Do

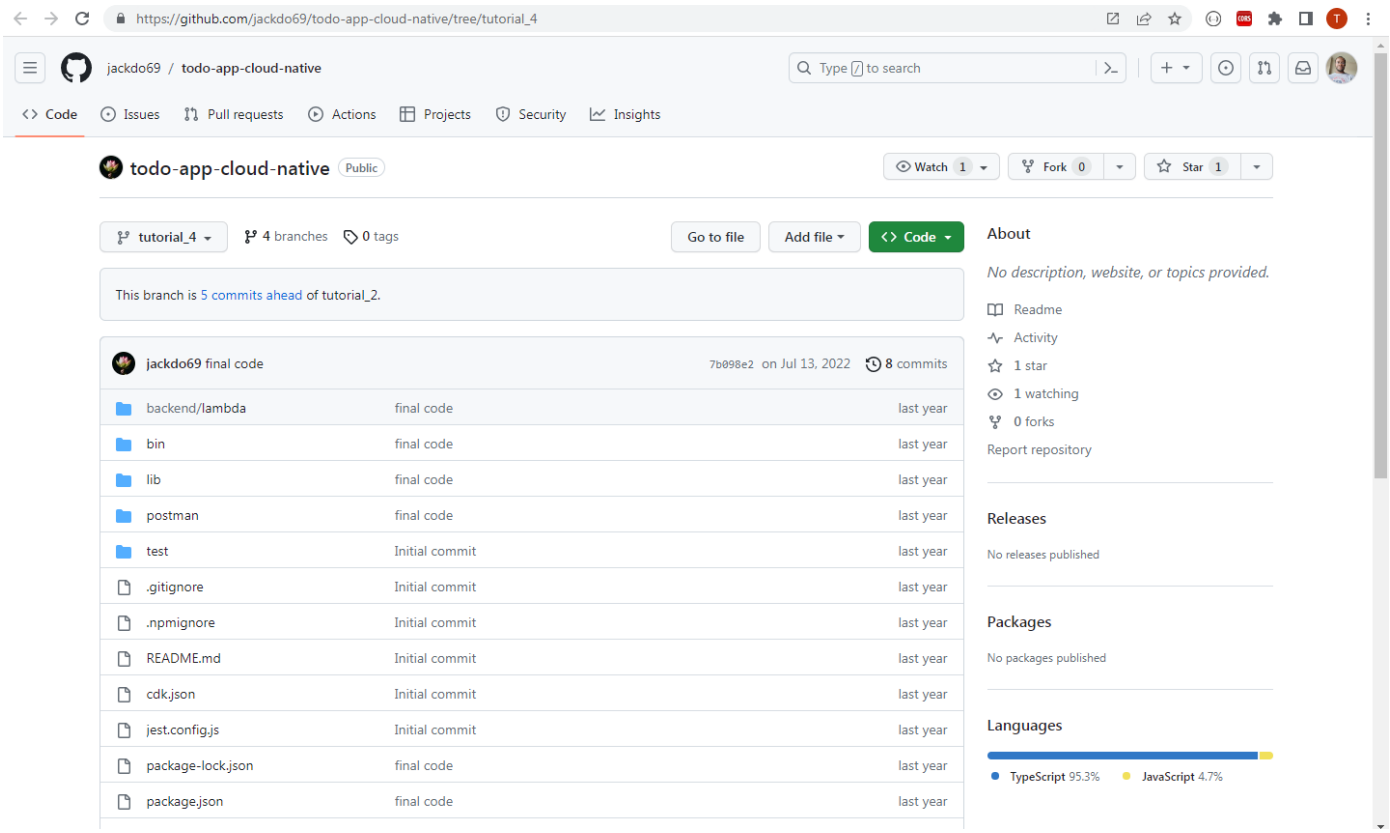
78 views Jul 8, 2022 AWS Cloud Development Kit (CDK) crash course

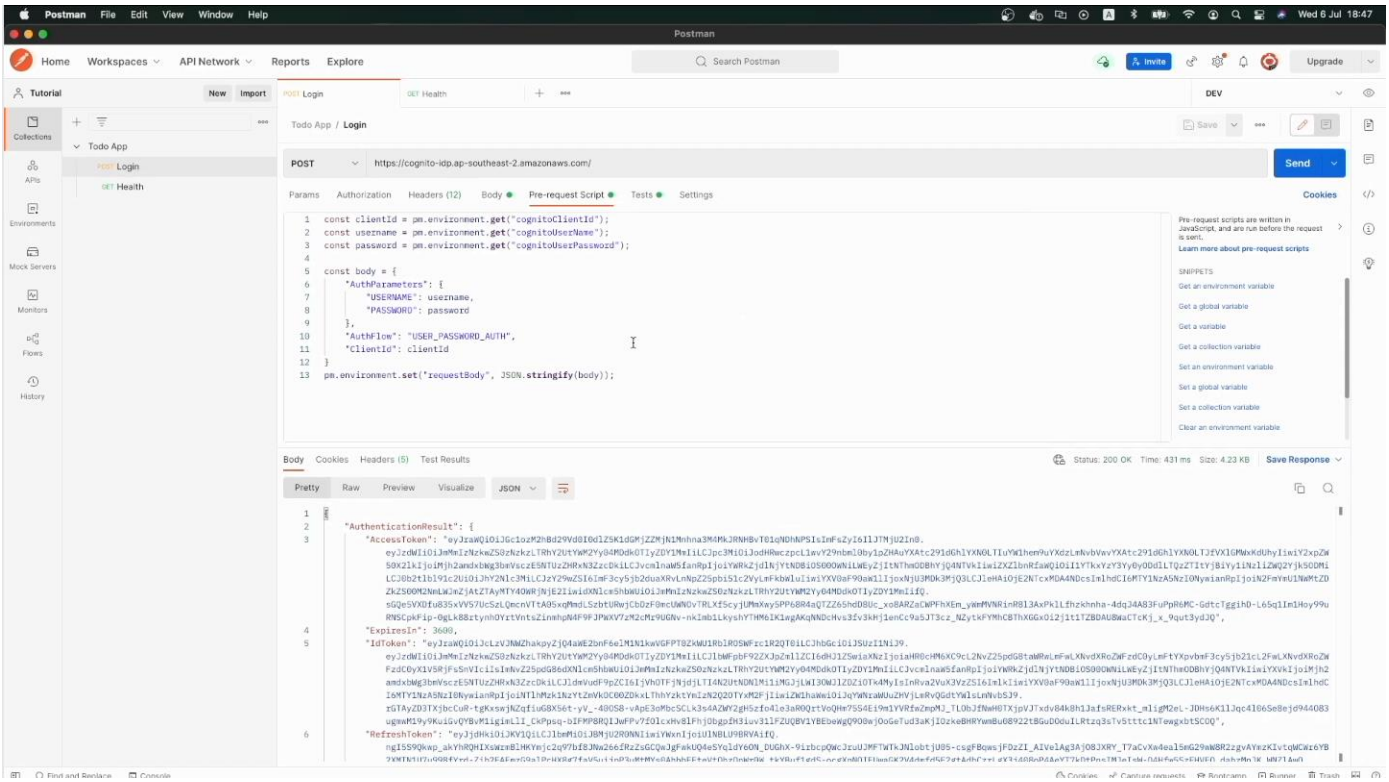
Hello guys,
In this video, we will go through:

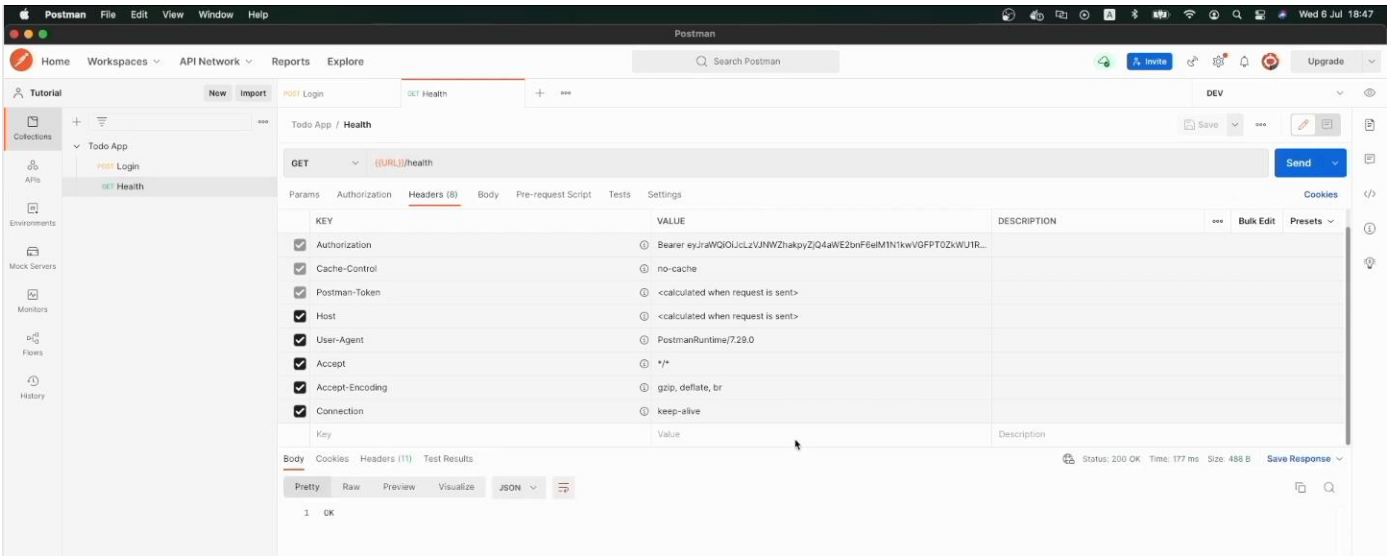
- Setting Dynamo and update lambda IAM role with CDK
- Updating backend codebase with some programming principles
- Separation of concerns
- Single responsibility
- Inversion of control

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

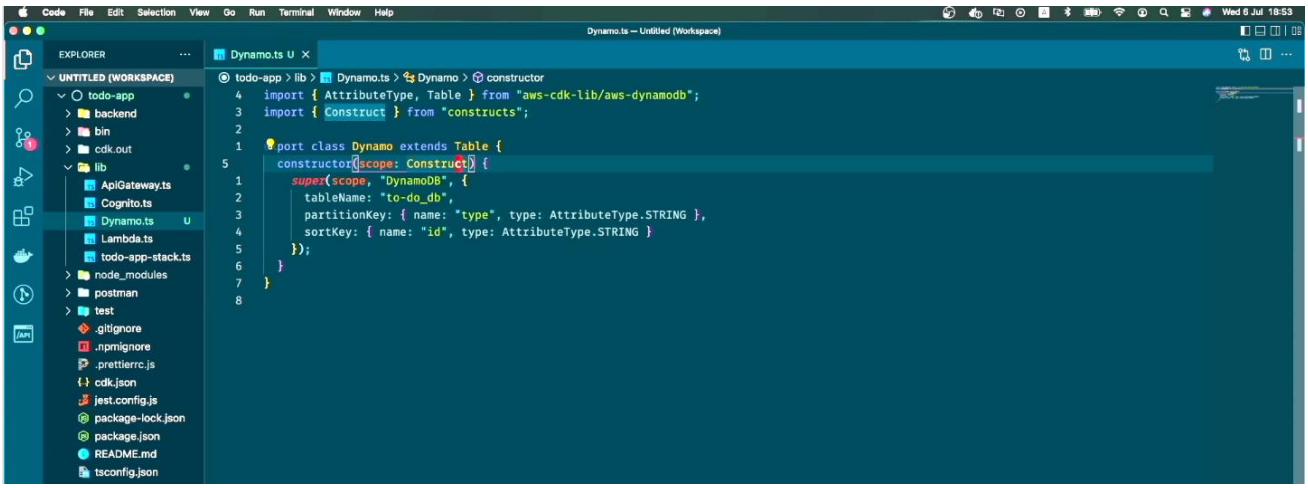
Timestamps:
- 00:00 Introduction and recap
- 01:00 Update the CDK code for Dynamo and Lambda
- 04:25 Go through the CDK code and deploy to AWS
- 07:35 Start coding the backend system
- 36:35 Folder structure walk through & Design Principles explain
- 45:58 Testing with Postman & wrap up

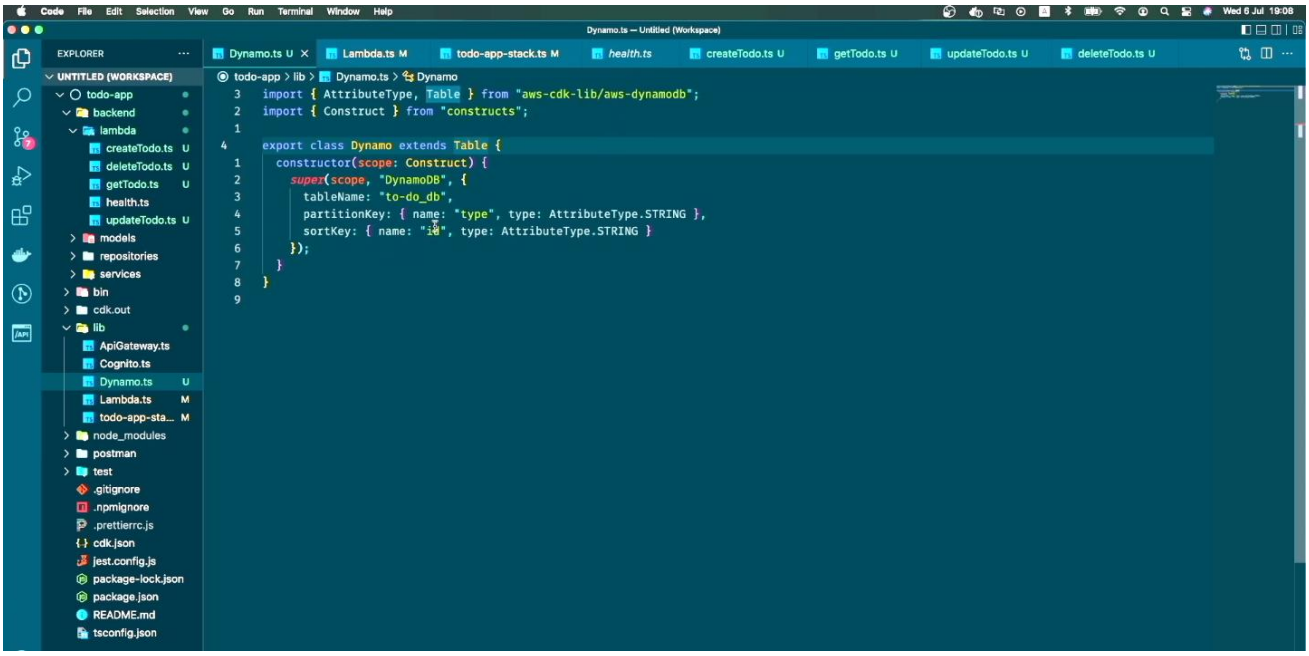


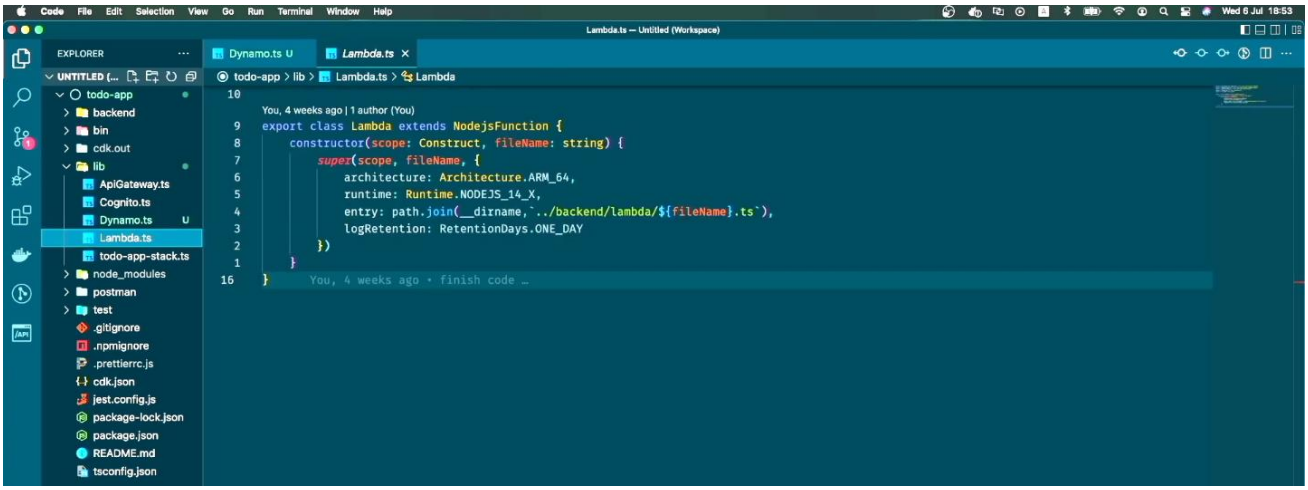


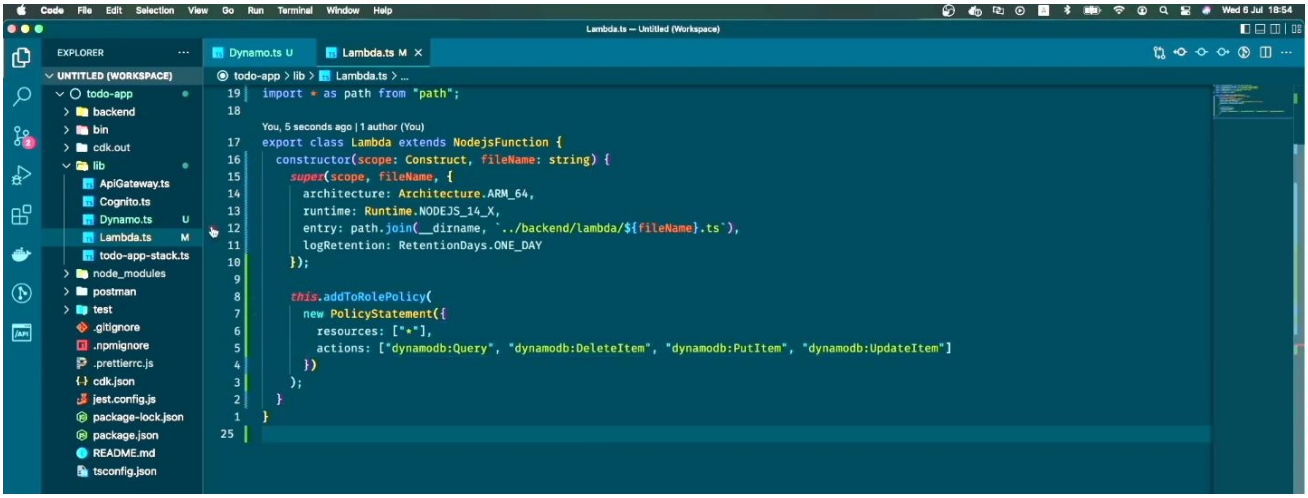


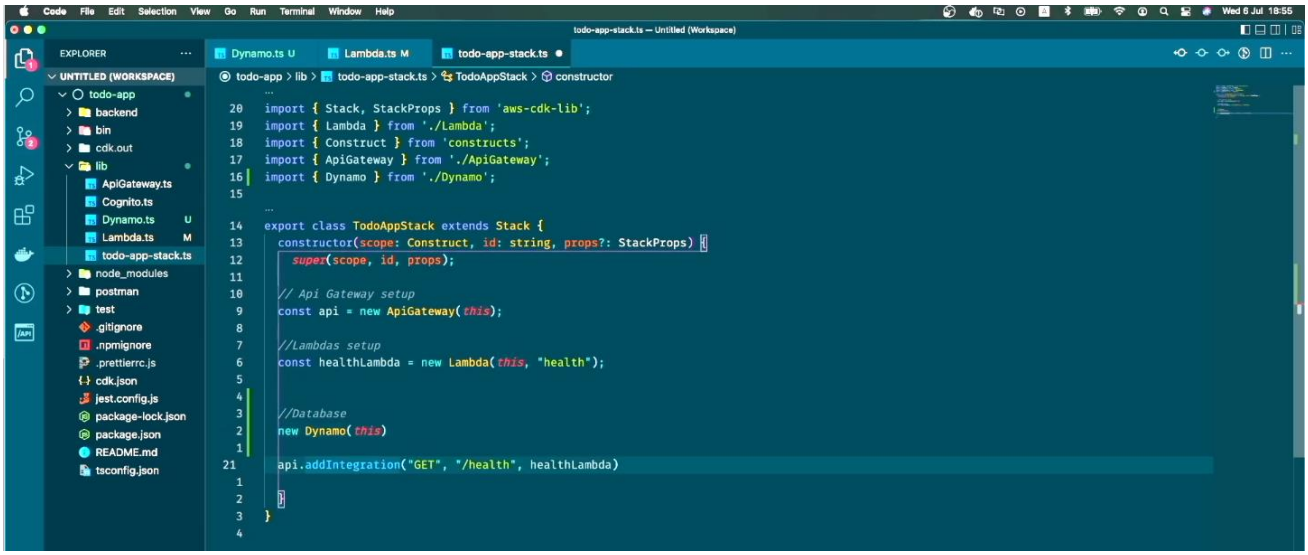
Let us see how to setup the DynamoDB backend using the CDK. We need to get a token before calling the /health API

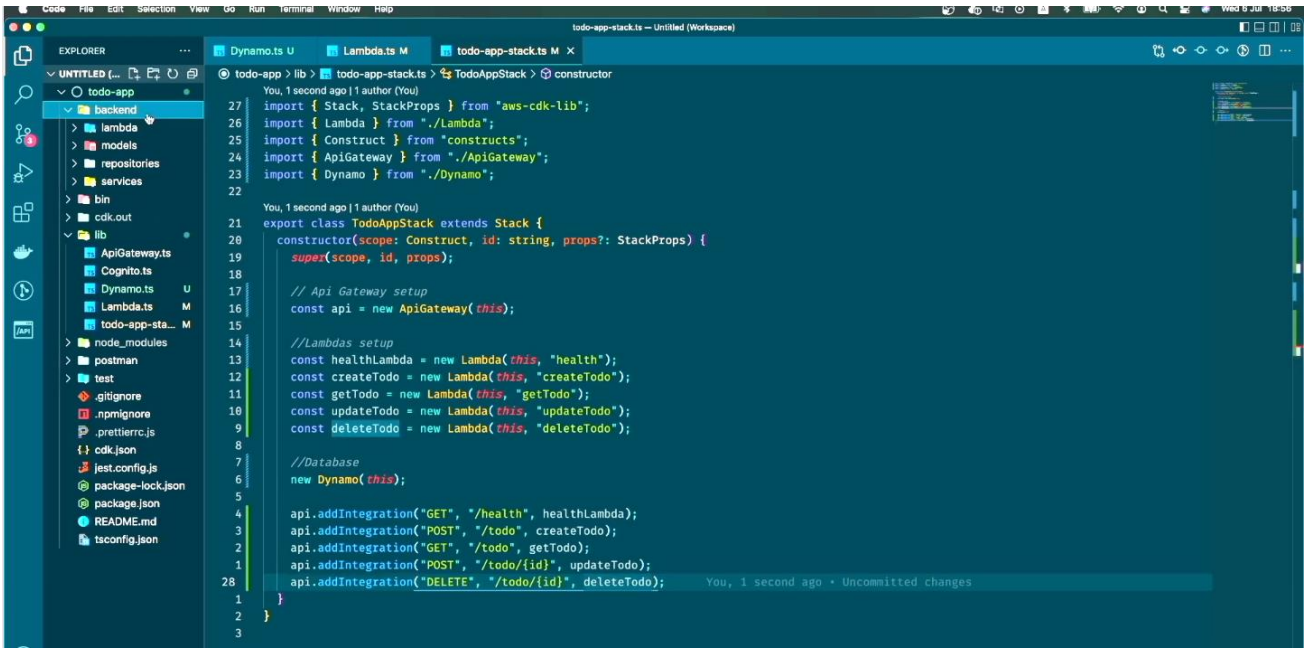


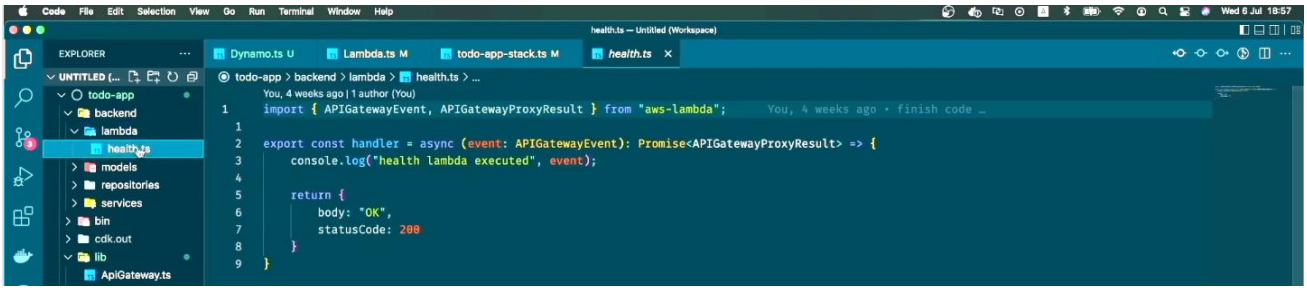

Dynamo.ts — Untitled (Workspace)
Dynamo.ts U
Lambda.ts M
todo-app-stack.ts M
health.ts
createTodo.ts U
getTodo.ts U
updateTodo.ts U
deleteTodo.ts U
import { AttributeType, Table } from "aws-cdk-lib/aws-dynamodb";
import { Construct } from "constructs";
export class Dynamo extends Table {
constructor(scope: Construct) {
super(scope, "DynamoDB", {
tableName: "to-do_db",
partitionKey: { name: "type", type: AttributeType.STRING },
sortKey: { name: "id", type: AttributeType.STRING }
});
}
}

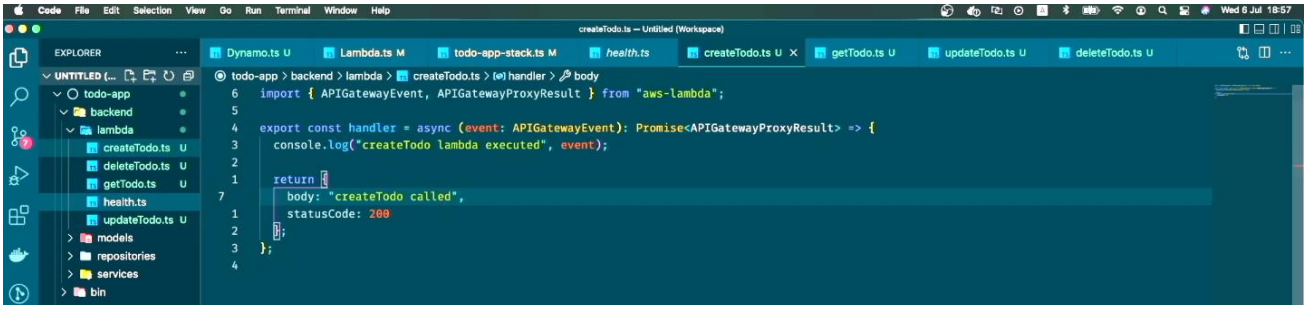

Lambda.ts — Untitled (Workspace)
Dynamo.ts U
Lambda.ts
You, 4 weeks ago | 1 author (You)
export class Lambda extends NodejsFunction {
constructor(scope: Construct, fileName: string) {
super(scope, fileName, {
architecture: Architecture.ARM_64,
runtime: Runtime.NODEJS_14_X,
entry: path.join(__dirname,`../backend/lambda/${fileName}.ts`),
logRetention: RetentionDays.ONE_DAY
})
}
}
You, 4 weeks ago • finish code …

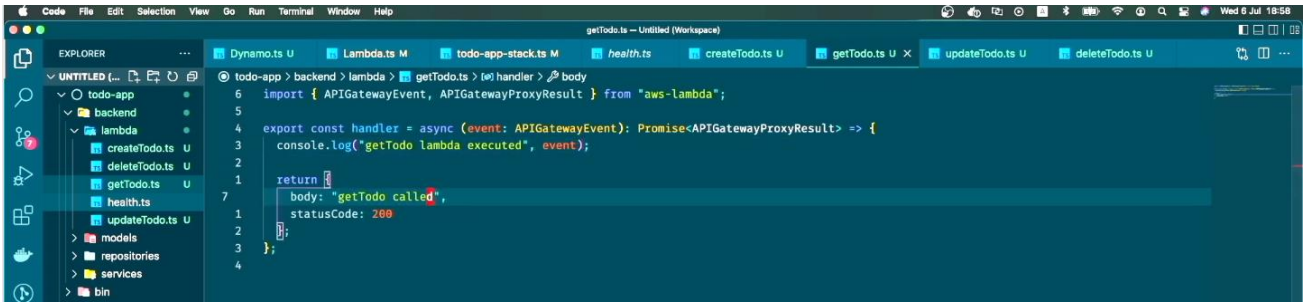

Lambda.ts — Untitled (Workspace)
Dynamo.ts U
Lambda.ts M
import * as path from "path";
You, 5 seconds ago | 1 author (You)
export class Lambda extends NodejsFunction {
constructor(scope: Construct, fileName: string) {
super(scope, fileName, {
architecture: Architecture.ARM_64,
runtime: Runtime.NODEJS_14_X,
entry: path.join(__dirname, `../backend/lambda/${fileName}.ts`),
logRetention: RetentionDays.ONE_DAY
});
this.addToRolePolicy(
new PolicyStatement({
resources: ["*"],
actions: ["dynamodb:Query", "dynamodb:DeleteItem", "dynamodb:PutItem", "dynamodb:UpdateItem"]
})
);
}
}

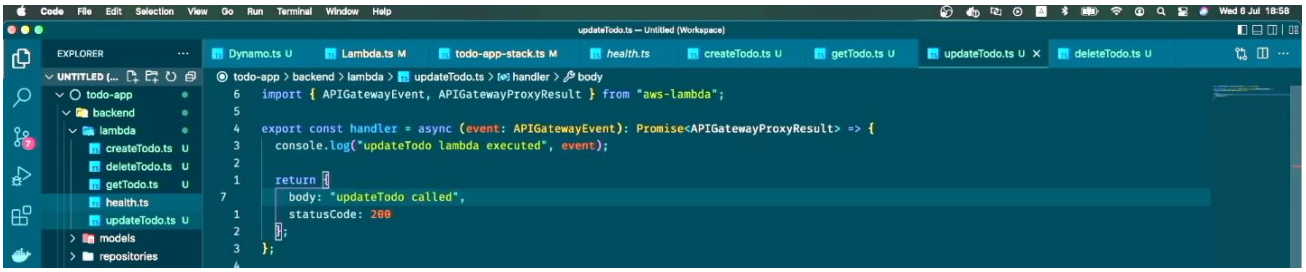

todo-app-stack.ts — Untitled (Workspace)
EXPLORER
Dynamo.ts U
Lambda.ts M
todo-app-stack.ts
todo-app > lib > todo-app-stack.ts > TodoAppStack > constructor
import { Stack, StackProps } from 'aws-cdk-lib';
import { Lambda } from './Lambda';
import { Construct } from 'constructs';
import { ApiGateway } 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)
}
}

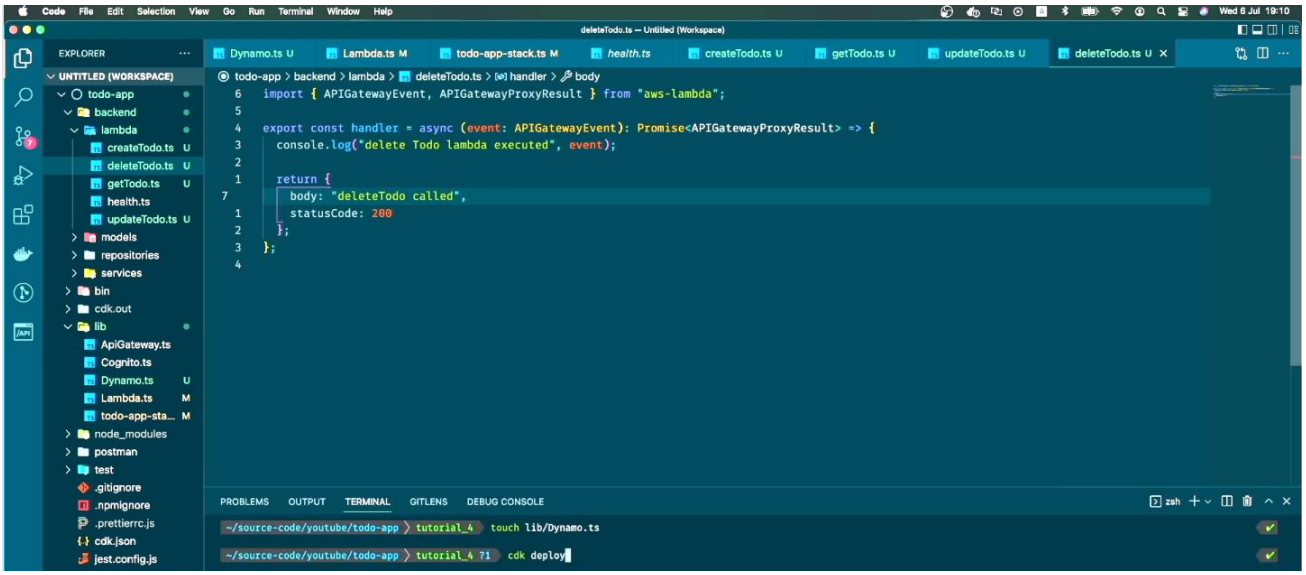

todo-app-stack.ts — Untitled (Workspace)
EXPLORER
Dynamo.ts U
Lambda.ts M
todo-app-stack.ts M
todo-app > lib > todo-app-stack.ts > TodoAppStack > constructor
You, 1 second ago | 1 author (You)
import { Stack, StackProps } from "aws-cdk-lib";
import { Lambda } from "./Lambda";
import { Construct } from "constructs";
import { ApiGateway } from "./ApiGateway";
import { Dynamo } from "./Dynamo";
You, 1 second ago | 1 author (You)
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);
You, 1 second ago • Uncommitted changes
}
}

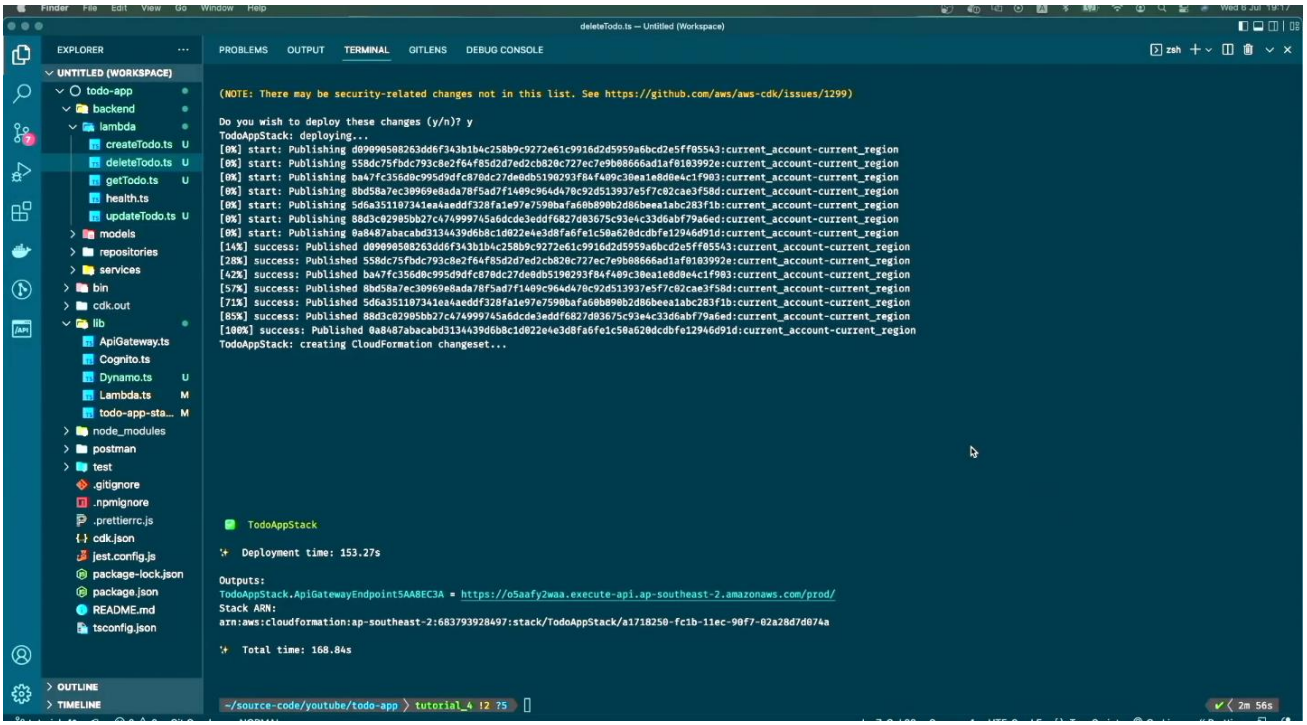

health.ts — Untitled (Workspace)
EXPLORER
Dynamo.ts U
Lambda.ts M
todo-app-stack.ts M
health.ts
todo-app > backend > lambda > health.ts > ...
You, 4 weeks ago | 1 author (You)
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
You, 4 weeks ago • finish code ...
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
console.log("health lambda executed", event);
return {
body: "OK",
statusCode: 200
}
}

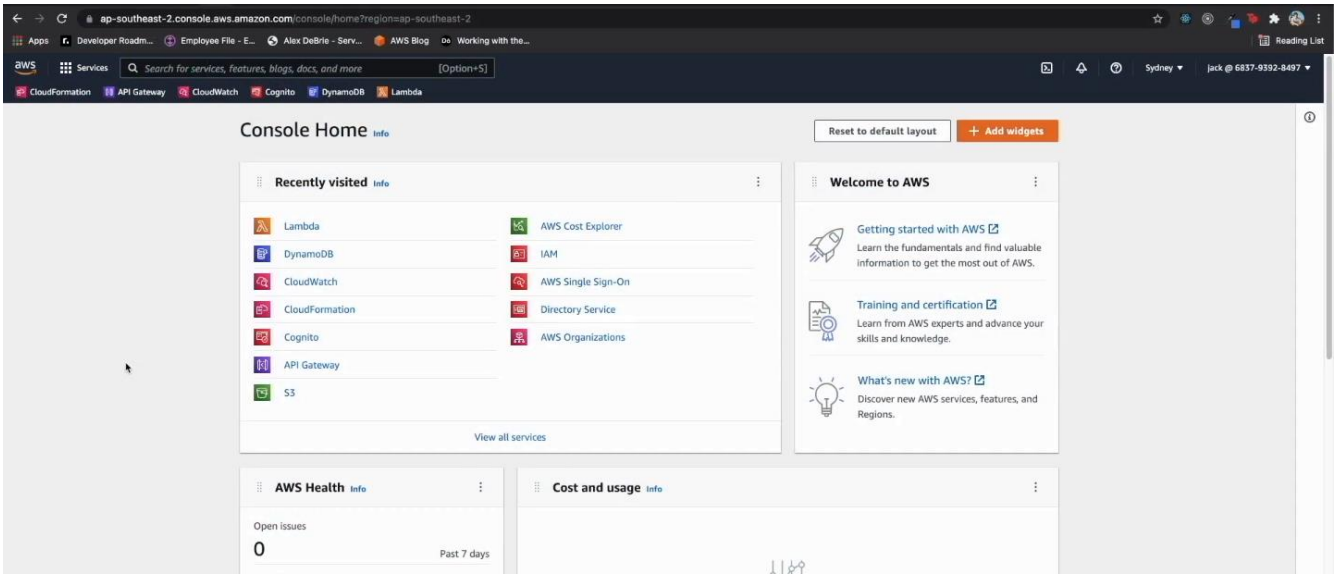

createTodo.ts — Untitled (Workspace)
EXPLORER
Dynamo.ts U
Lambda.ts M
todo-app-stack.ts M
health.ts
createTodo.ts U
getTodo.ts U
updateTodo.ts U
deleteTodo.ts U
todo-app > backend > lambda > createTodo.ts > handler > body
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
console.log("createTodo lambda executed", event);
return {
body: "createTodo called",
statusCode: 200
};
};

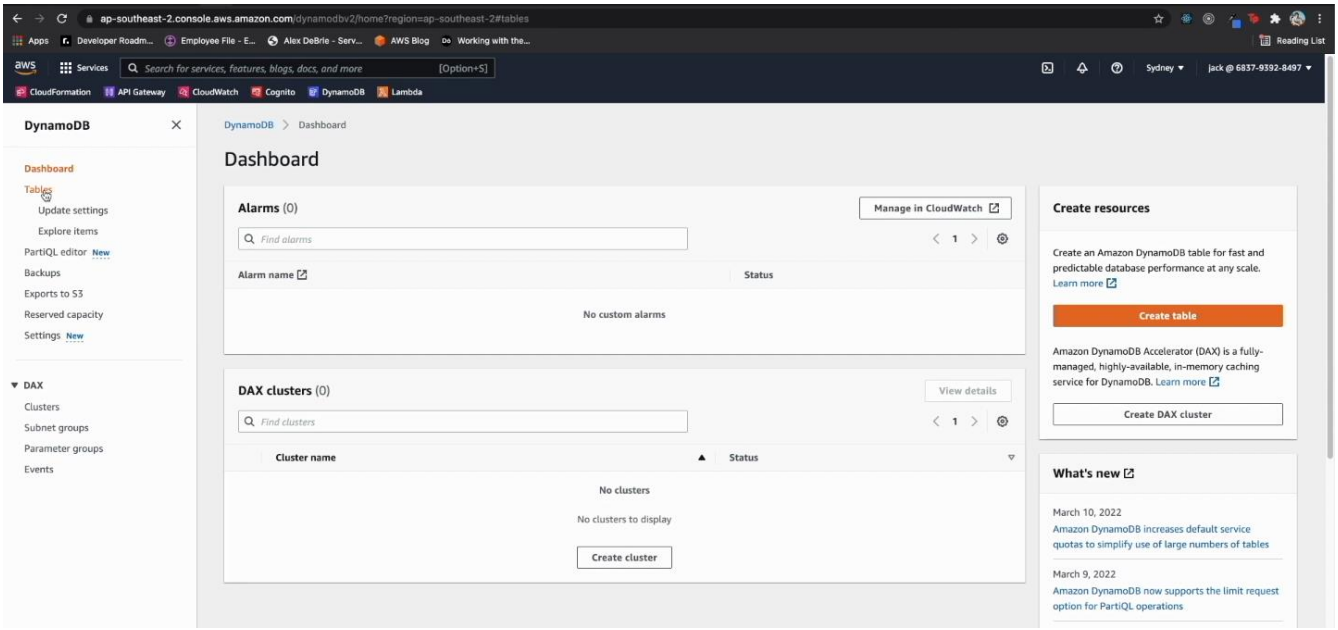

getTodo.ts — Untitled (Workspace)
Dynamo.ts U
Lambda.ts M
todo-app-stack.ts M
health.ts
createTodo.ts U
getTodo.ts U
updateTodo.ts U
deleteTodo.ts U
todo-app > backend > lambda > getTodo.ts > handler > body
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
console.log("getTodo lambda executed", event);
return {
body: "getTodo called",
statusCode: 200
};
};

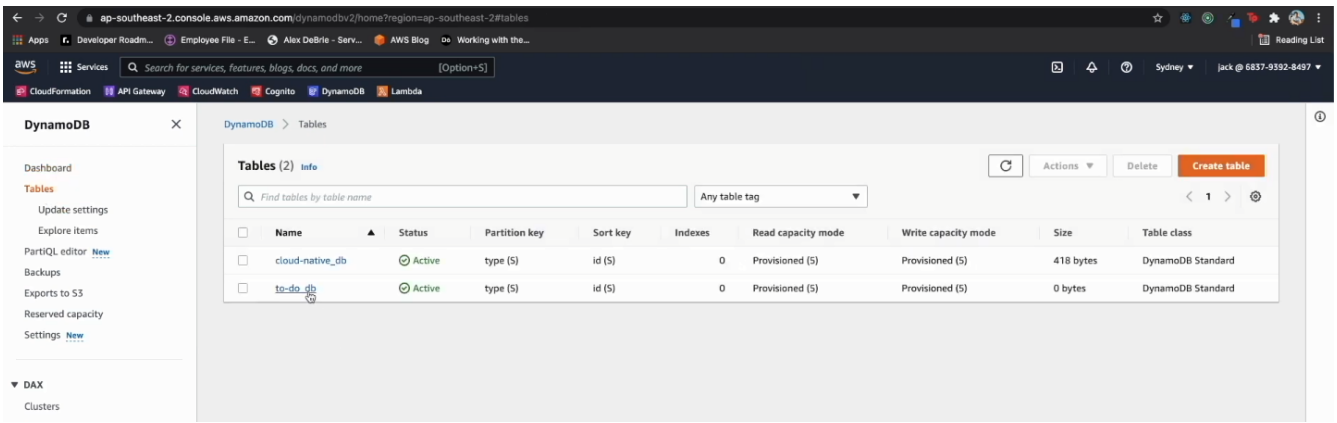

updateTodo.ts — Untitled (Workspace)
todo-app > backend > lambda > updateTodo.ts > handler > body
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
console.log("updateTodo lambda executed", event);
return {
body: "updateTodo called",
statusCode: 200
};
};

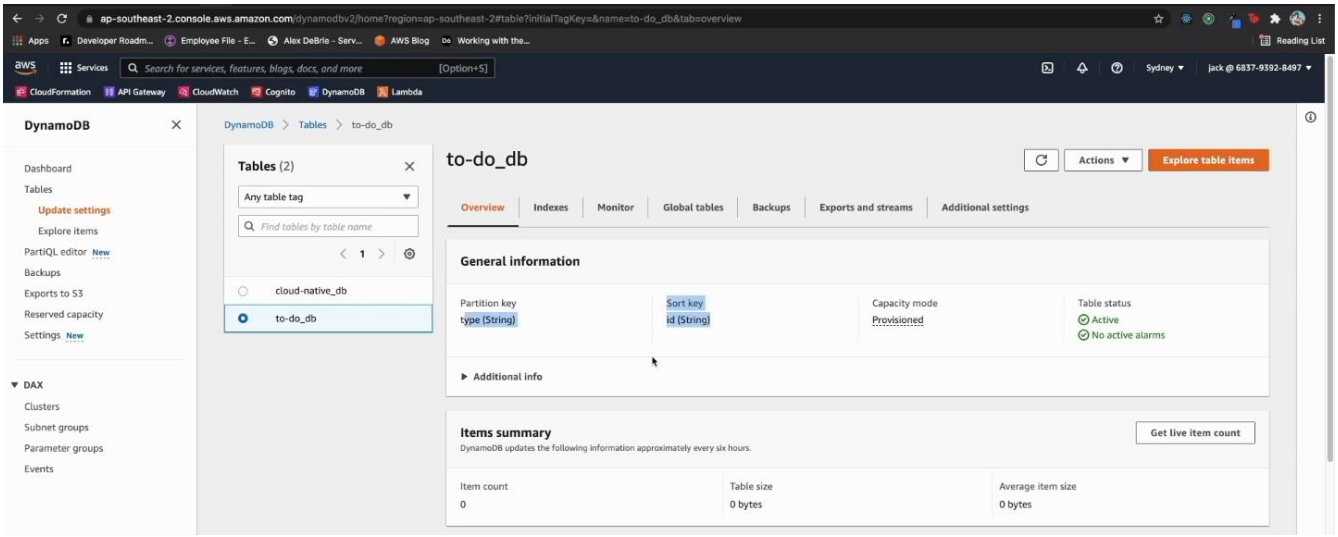

deleteTodo.ts — Untitled (Workspace)
todo-app > backend > lambda > deleteTodo.ts > handler > body
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
console.log("delete Todo lambda executed", event);
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 ?1 cdk deploy

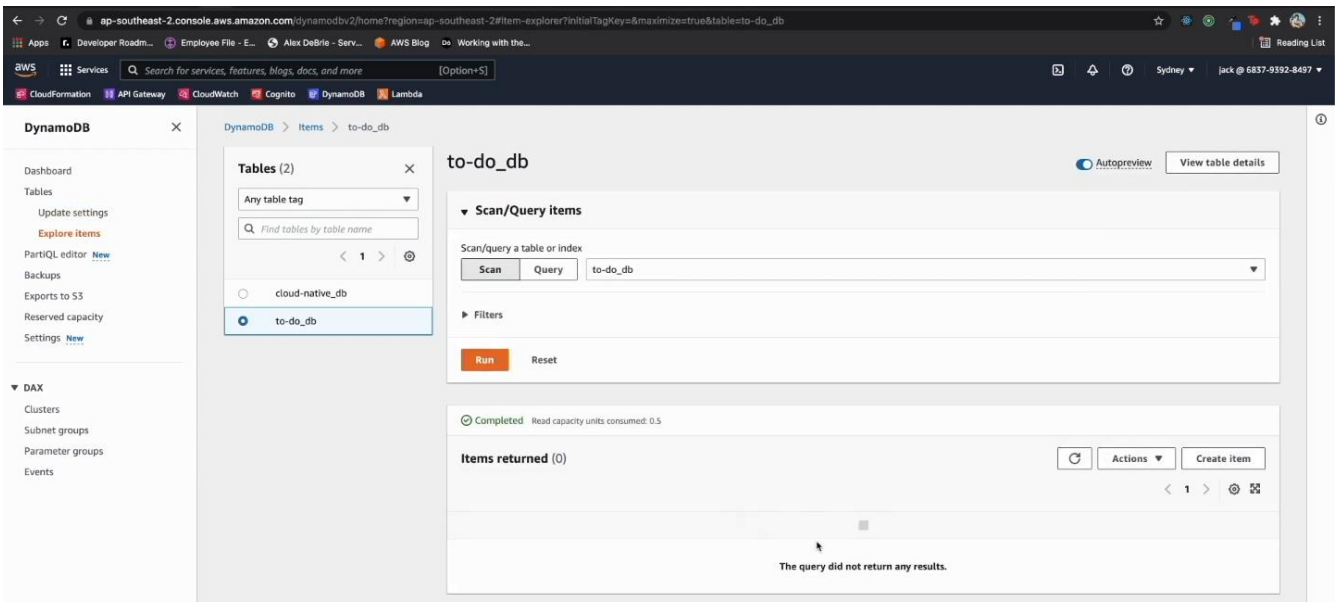

(NOTE: There may be security-related changes not in this list. See https://github.com/aws/aws-cdk/issues/1299)
Do you wish to deploy these changes (y/n)? y
TodoAppStack: deploying...
[0%] start: Publishing d09090508263dd6f343b1b4c258b9c9272e61c9916d2d5959a6bcd2e5ff05543:current_account-current_region
[0%] start: Publishing 558dc75fbdc793c8e2f64f85d2d7ed2cb820c727ec7e9b08666ad1af0103992e:current_account-current_region
[0%] start: Publishing ba47fc356d0c995d9dfc870dc27de0db5190293f84f409c30ea1e8d0e4c1f903:current_account-current_region
[0%] start: Publishing 8bd58a7ec30969e8ada78f5ad7f1409c964d470c92d513937e5f7c02cae3f58d:current_account-current_region
[0%] start: Publishing 5d6a351107341ea4aeddf328fa1e97e7590bafa60b890b2d86beea1abc283f1b:current_account-current_region
[0%] start: Publishing 88d3c02985bb27c474999745a6dcde3eddf6827d03675c93e4c33d6abf79a6ed:current_account-current_region
[0%] start: Publishing 0a8487abacabd3134439d6b8c1d022e4e3d8fa6fe1c50a620dcdbfe12946d91d:current_account-current_region
[14%] success: Published d09090508263dd6f343b1b4c258b9c9272e61c9916d2d5959a6bcd2e5ff05543:current_account-current_region
[28%] success: Published 558dc75fbdc793c8e2f64f85d2d7ed2cb820c727ec7e9b08666ad1af0103992e:current_account-current_region
[42%] success: Published ba47fc356d0c995d9dfc870dc27de0db5190293f84f409c30ea1e8d0e4c1f903:current_account-current_region
[57%] success: Published 8bd58a7ec30969e8ada78f5ad7f1409c964d470c92d513937e5f7c02cae3f58d:current_account-current_region
[71%] success: Published 5d6a351107341ea4aeddf328fa1e97e7590bafa60b890b2d86beea1abc283f1b:current_account-current_region
[85%] success: Published 88d3c02985bb27c474999745a6dcde3eddf6827d03675c93e4c33d6abf79a6ed:current_account-current_region
[100%] success: Published 0a8487abacabd3134439d6b8c1d022e4e3d8fa6fe1c50a620dcdbfe12946d91d:current_account-current_region
TodoAppStack: creating CloudFormation changeset...
TodoAppStack
Deployment time: 153.27s
Outputs:
TodoAppStack.ApiGatewayEndpoint5AA8EC3A = https://o5aafy2waa.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 !2 ?5
2m 56s

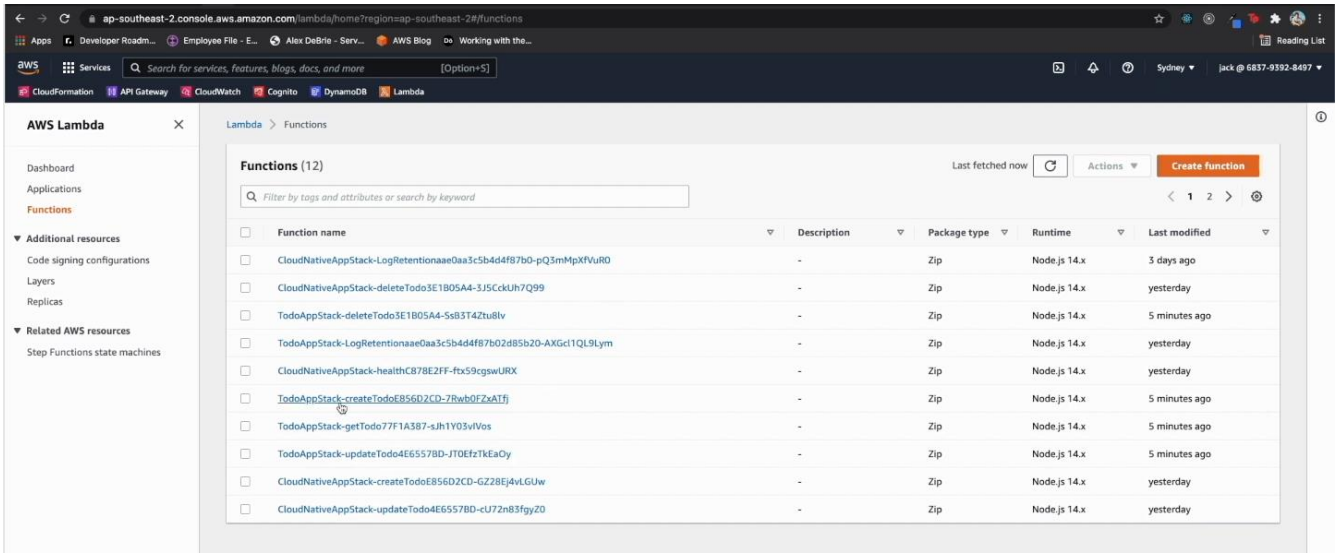

ap-southeast-2.console.aws.amazon.com/console/home?region=ap-southeast-2
Console Home Info
Reset to default layout
+ Add widgets
Recently visited Info
Lambda
DynamoDB
CloudWatch
CloudFormation
Cognito
API Gateway
S3
AWS Cost Explorer
IAM
AWS Single Sign-On
Directory Service
AWS Organizations
View all services
Welcome to AWS
Getting started with AWS
Learn the fundamentals and find valuable information to get the most out of AWS.
Training and certification
Learn from AWS experts and advance your skills and knowledge.
What's new with AWS?
Discover new AWS services, features, and Regions.
AWS Health Info
Open issues
0
Past 7 days
Cost and usage Info

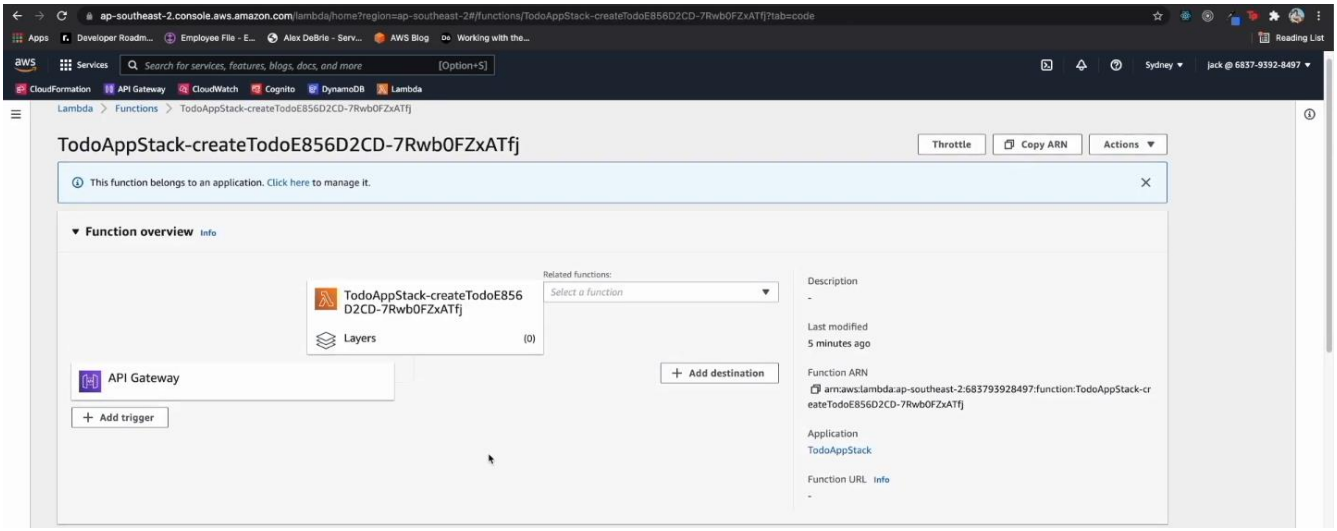

ap-southeast-2.console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#tables
DynamoDB
Dashboard
Tables
Update settings
Explore items
PartiQL editor New
Backups
Exports to S3
Reserved capacity
Settings New
DAX
Clusters
Subnet groups
Parameter groups
Events
DynamoDB > Dashboard
Dashboard
Alarms (0)
Manage in CloudWatch
Find alarms
Alarm name
Status
No custom alarms
DAX clusters (0)
View details
Find clusters
Cluster name
Status
No clusters
No clusters to display
Create cluster
Create resources
Create an Amazon DynamoDB table for fast and predictable database performance at any scale.
Learn more
Create table
Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. Learn more
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

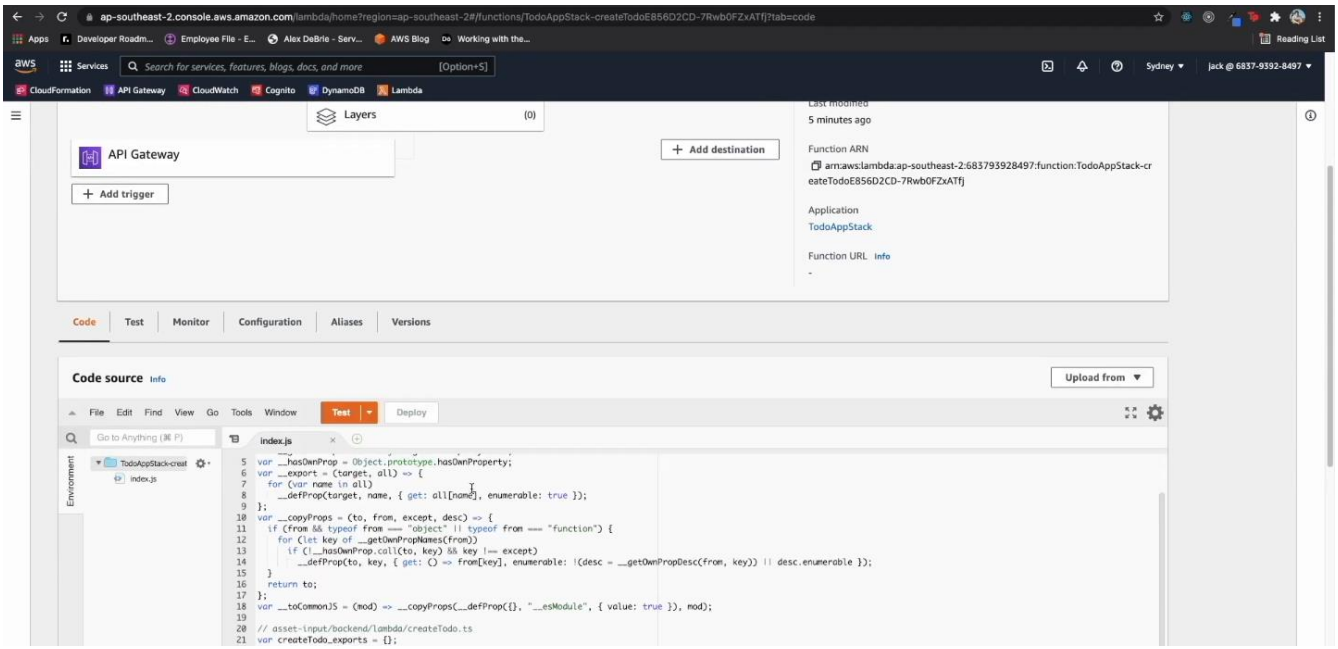

ap-southeast-2.console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#tables
DynamoDB
Dashboard
Tables
Update settings
Explore items
PartiQL editor New
Backups
Exports to S3
Reserved capacity
Settings New
DAX
Clusters
DynamoDB > Tables
Tables (2) Info
Actions
Delete
Create table
Find tables by table name
Any table tag
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 (5)
Provisioned (5)
418 bytes
DynamoDB Standard
to-do_db
Active
type (S)
id (S)
0
Provisioned (5)
Provisioned (5)
0 bytes
DynamoDB Standard

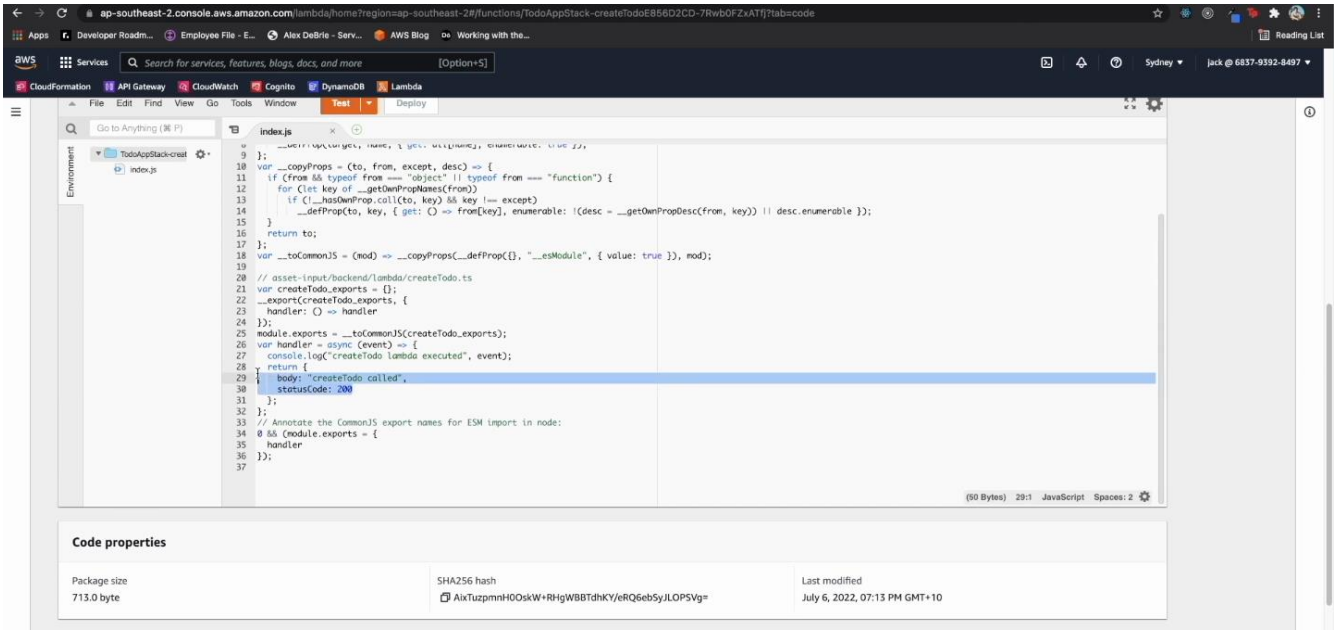

ap-southeast-2.console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#table?initialTagKey=&name=to-do_db&tab=overview
DynamoDB
Tables (2)
Any table tag
Find tables by table name
cloud-native_db
to-do_db
DynamoDB > Tables > to-do_db
to-do_db
Actions
Explore table items
Overview
Indexes
Monitor
Global tables
Backups
Exports and streams
Additional settings
General information
Partition key
type (String)
Sort key
id (String)
Capacity mode
Provisioned
Table status
Active
No active alarms
Additional info
Items summary
DynamoDB updates the following information approximately every six hours.
Get live item count
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=&maximize=true&table=to-do_db
DynamoDB
Tables (2)
Any table tag
Find tables by table name
cloud-native_db
to-do_db
DynamoDB > Items > to-do_db
to-do_db
Autopreview
View table details
Scan/Query items
Scan/query a table or index
Scan
Query
to-do_db
Filters
Run
Reset
Completed
Read capacity units consumed: 0.5
Items returned (0)
Actions
Create item
The query did not return any results.

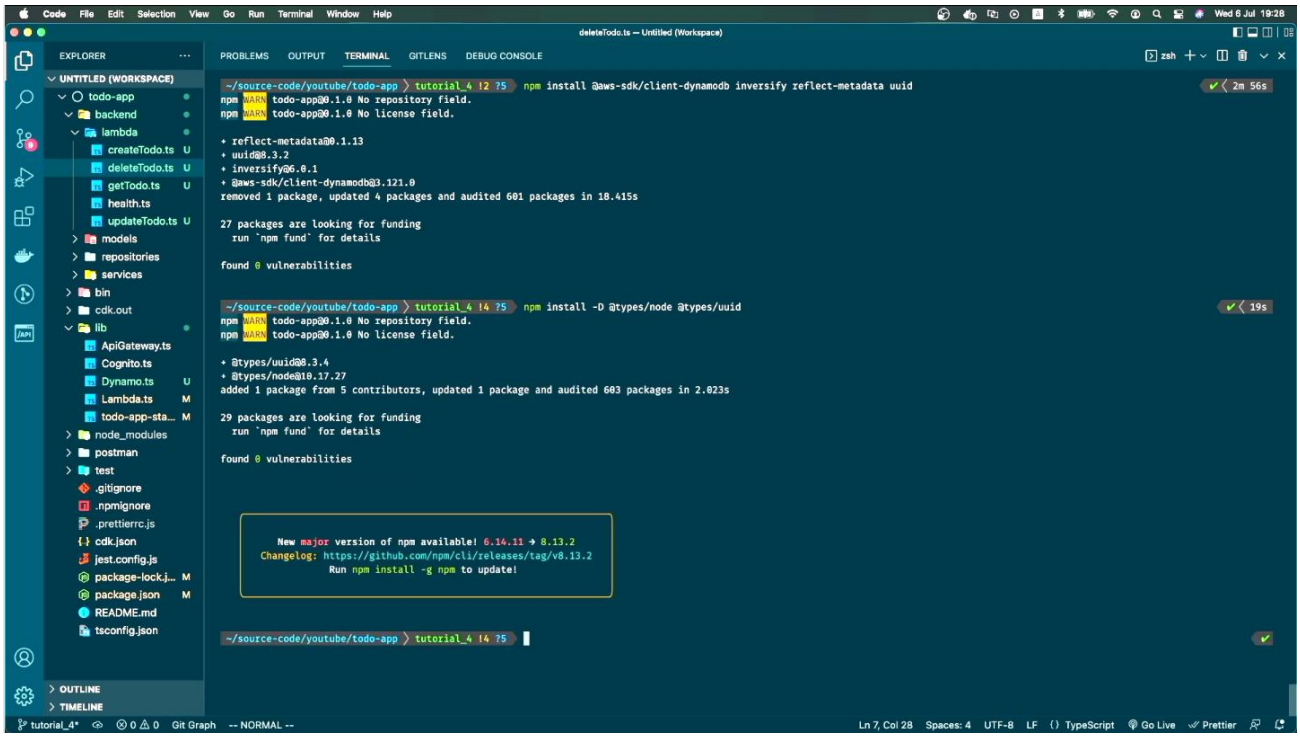

ap-southeast-2.console.aws.amazon.com/lambda/home?region=ap-southeast-2#/functions
AWS Lambda
Lambda > Functions
Functions (12)
Last fetched now
Actions
Create function
Filter by tags and attributes or search by keyword
Function name
Description
Package type
Runtime
Last modified
CloudNativeAppStack-LogRetentionaae0aa3c5b4d4f87b0-pQ3mMpXfVuR0
CloudNativeAppStack-deleteTodo3E1B05A4-5J5CckUh7Q99
TodoAppStack-deleteTodo3E1B05A4-Ss83T4Ztu8lv
TodoAppStack-LogRetentionaae0aa3c5b4d4f87b02d85b20-AXGcl1QL9Lym
CloudNativeAppStack-healthC878E2FF-ftx59cgswURX
TodoAppStack-createTodoE856D2CD-7Rwb0FZxATfj
TodoAppStack-getTodo77F1A387-sJh1Y03vIVos
TodoAppStack-updateTodo4E6557BD-JT0EfzTkEaOy
CloudNativeAppStack-createTodoE856D2CD-GZ28Ej4vLGUw
CloudNativeAppStack-updateTodo4E6557BD-cU72n83fgyZ0
Zip
Node.js 14.x
3 days ago
yesterday
5 minutes ago

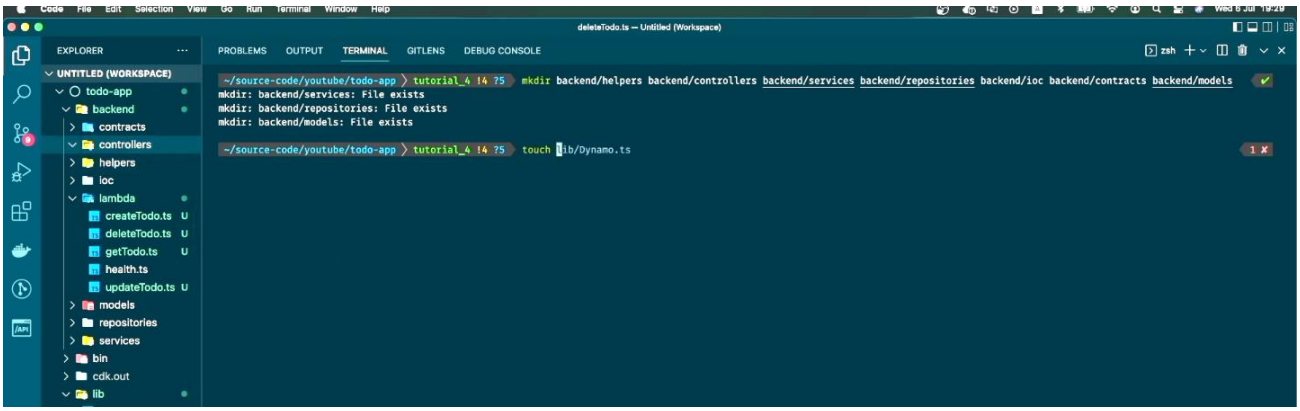


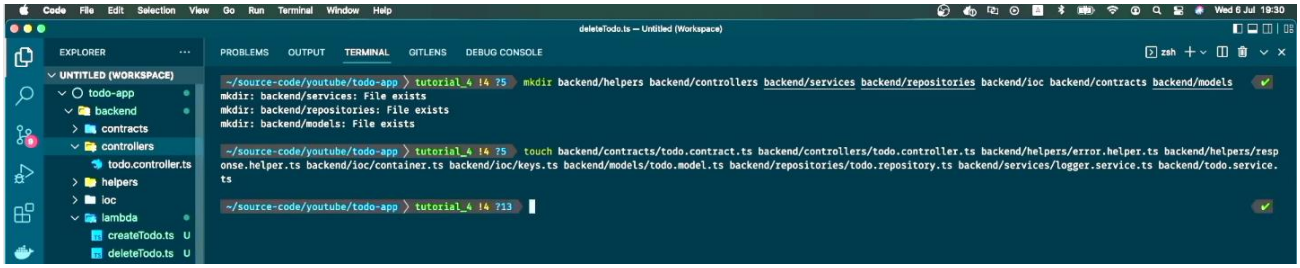


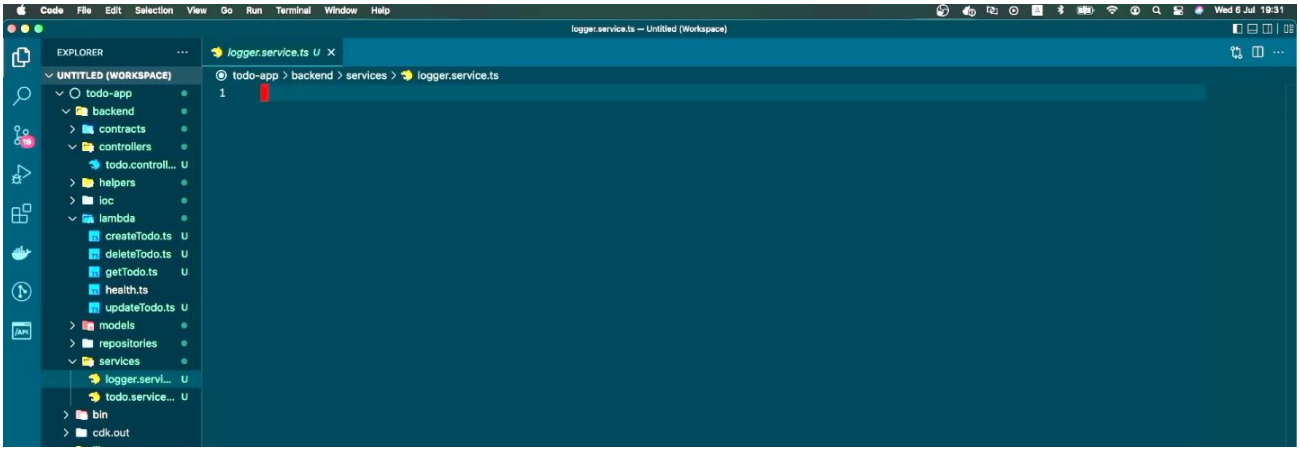


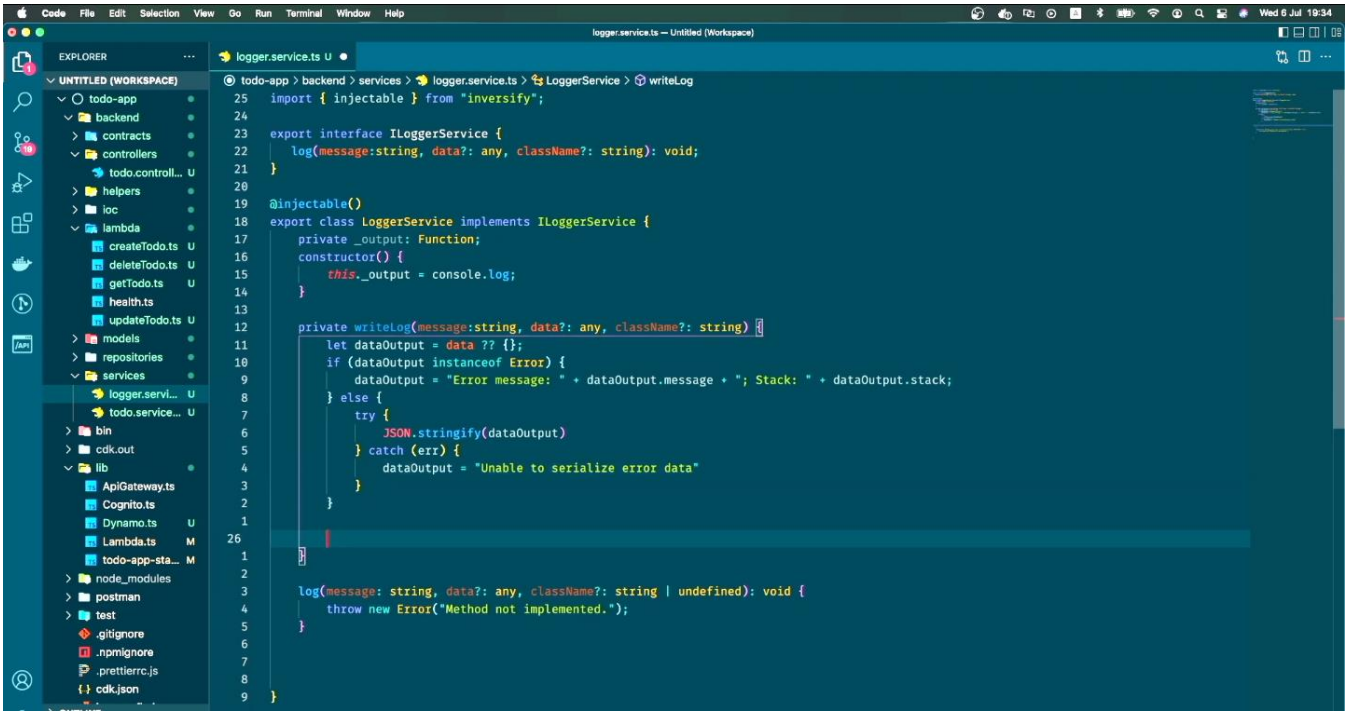
Next, let us start to implement changes to our lambda code

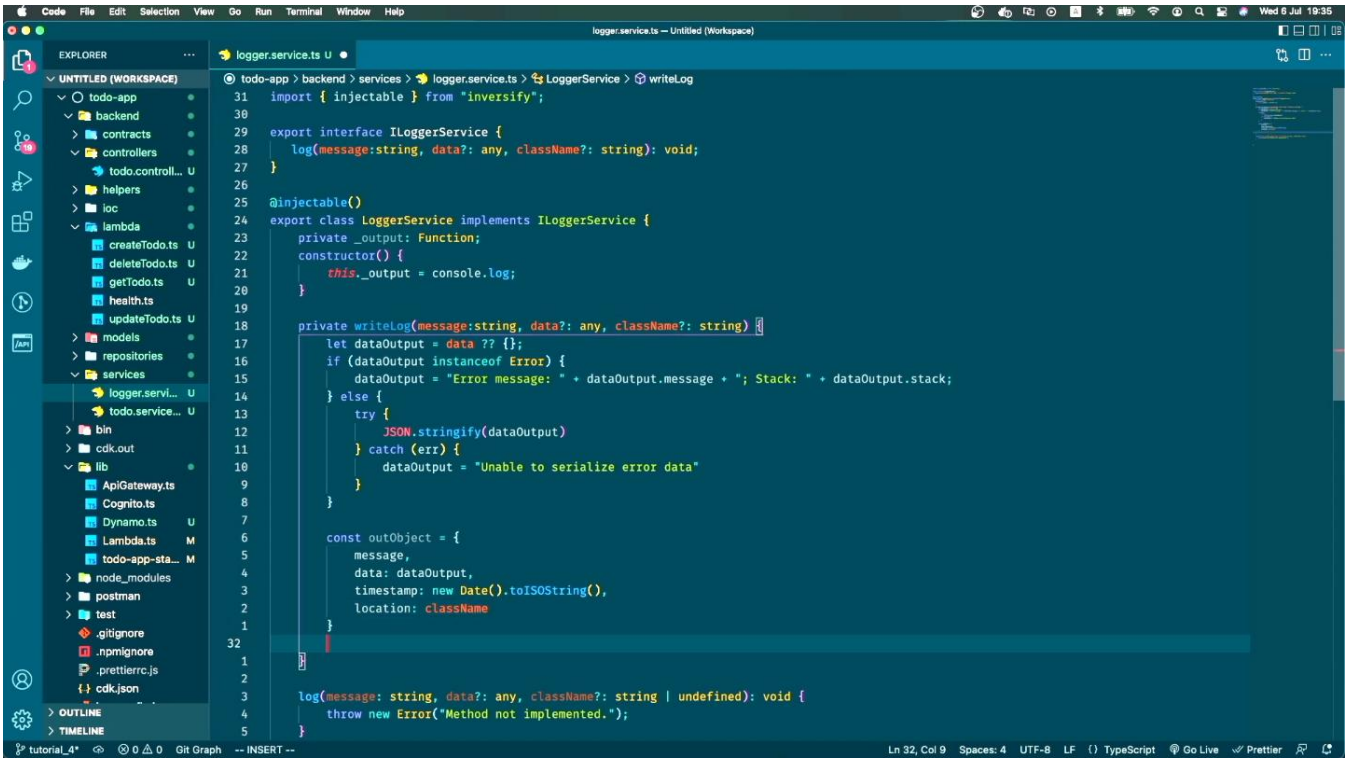

deleteTodo.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
lambda
createTodo.ts
deleteTodo.ts
getTodo.ts
health.ts
updateTodo.ts
models
repositories
services
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts
Lambda.ts
todo-app-sta...
node_modules
postman
test
.gitignore
.npmignore
.prettierrc.js
cdk.json
jest.config.js
package-lock.j...
package.json
README.md
tsconfig.json
OUTLINE
TIMELINE
PROBLEMS
OUTPUT
TERMINAL
GITLENS
DEBUG CONSOLE
~/source-code/youtube/todo-app ❯ tutorial_4 !2 ?5 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
~/source-code/youtube/todo-app ❯ tutorial_4 !4 ?5 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@10.17.27
added 1 package from 5 contributors, updated 1 package and audited 603 packages in 2.023s
29 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
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!
~/source-code/youtube/todo-app ❯ tutorial_4 !4 ?5
tutorial_4*
Git Graph
-- NORMAL --
Ln 7, Col 28
Spaces: 4
UTF-8
LF
TypeScript
Go Live
Prettier

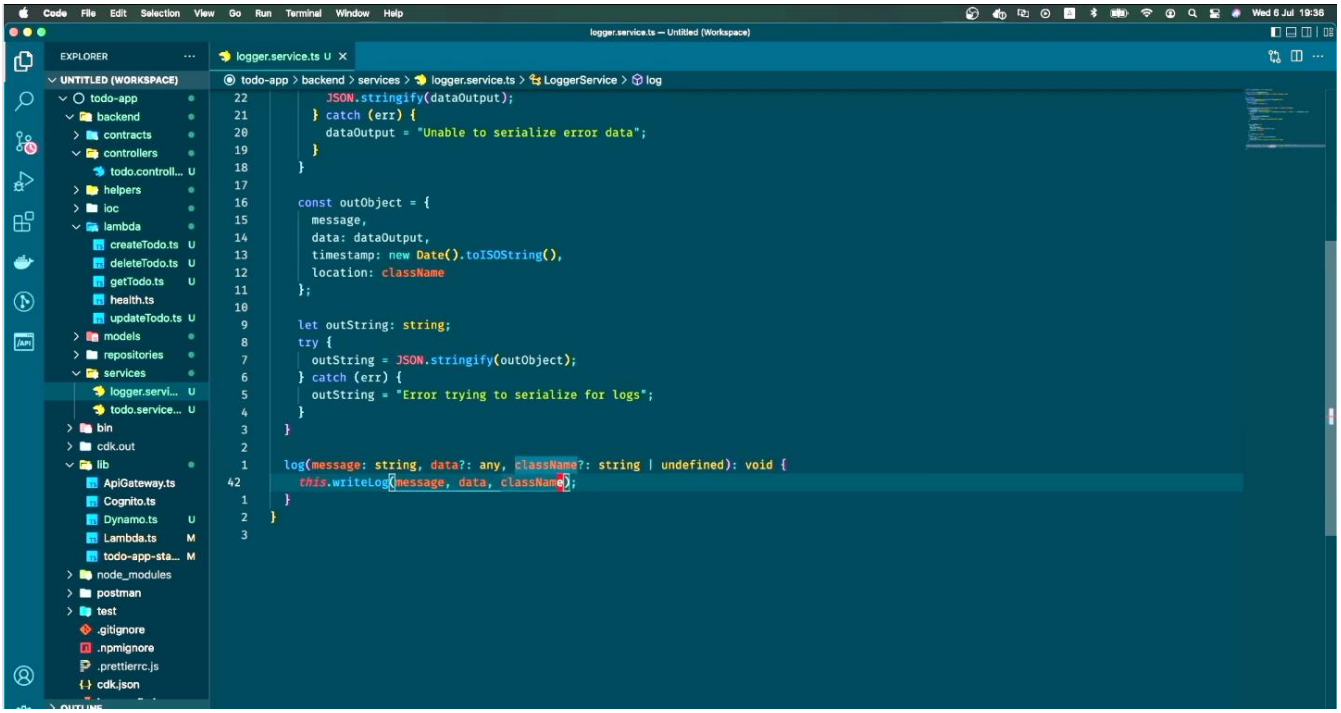

deleteTodo.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
contracts
controllers
helpers
ioc
lambda
createTodo.ts
deleteTodo.ts
getTodo.ts
health.ts
updateTodo.ts
models
repositories
services
bin
cdk.out
lib
PROBLEMS
OUTPUT
TERMINAL
GITLENS
DEBUG CONSOLE
~/source-code/youtube/todo-app ❯ tutorial_4 !4 ?5 mkdir backend/helpers backend/controllers backend/services backend/repositories backend/ioc 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 !4 ?5 touch lib/Dynamo.ts


deleteTodo.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
contracts
controllers
todo.controller.ts
helpers
ioc
lambda
createTodo.ts
deleteTodo.ts
PROBLEMS
OUTPUT
TERMINAL
GITLENS
DEBUG CONSOLE
~/source-code/youtube/todo-app ❯ tutorial_4 !4 ?5 mkdir backend/helpers backend/controllers backend/services backend/repositories backend/ioc 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 !4 ?5 touch backend/contracts/todo.contract.ts backend/controllers/todo.controller.ts backend/helpers/error.helper.ts backend/helpers/response.helper.ts backend/ioc/container.ts backend/ioc/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 !4 ?13


logger.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
logger.service.ts U
todo-app > backend > services > logger.service.ts
1
todo-app
backend
contracts
controllers
todo.controll...
helpers
ioc
lambda
createTodo.ts
deleteTodo.ts
getTodo.ts
health.ts
updateTodo.ts
models
repositories
services
logger.servi...
todo.service...
bin
cdk.out

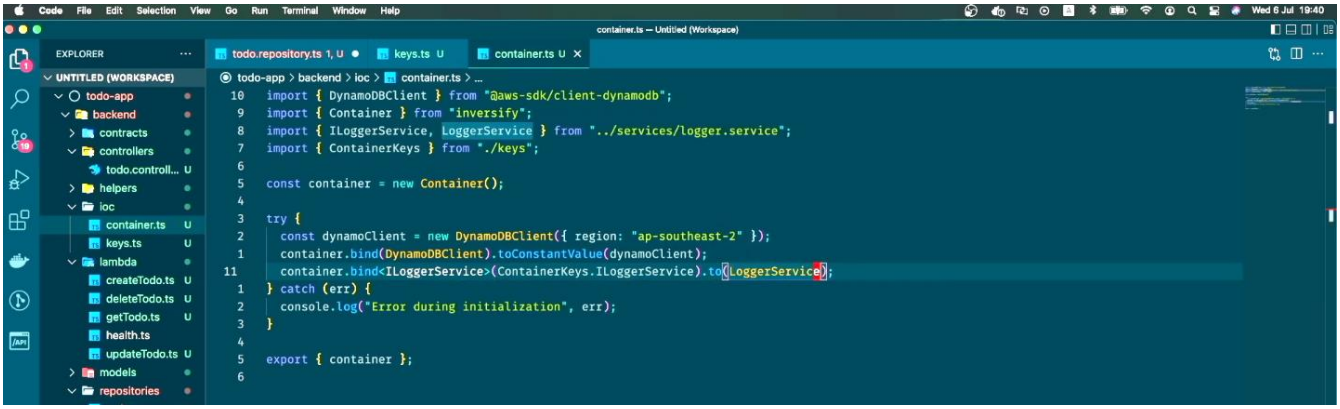

Code File Edit Selection View Go Run Terminal Window Help
Wed 6 Jul 19:34
logger.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
logger.service.ts U
todo-app > backend > services > logger.service.ts > LoggerService > writeLog
import { injectable } from "inversify";
export interface ILoggerService {
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: string, data?: any, className?: string | undefined): void {
throw new Error("Method not implemented.");
}
}

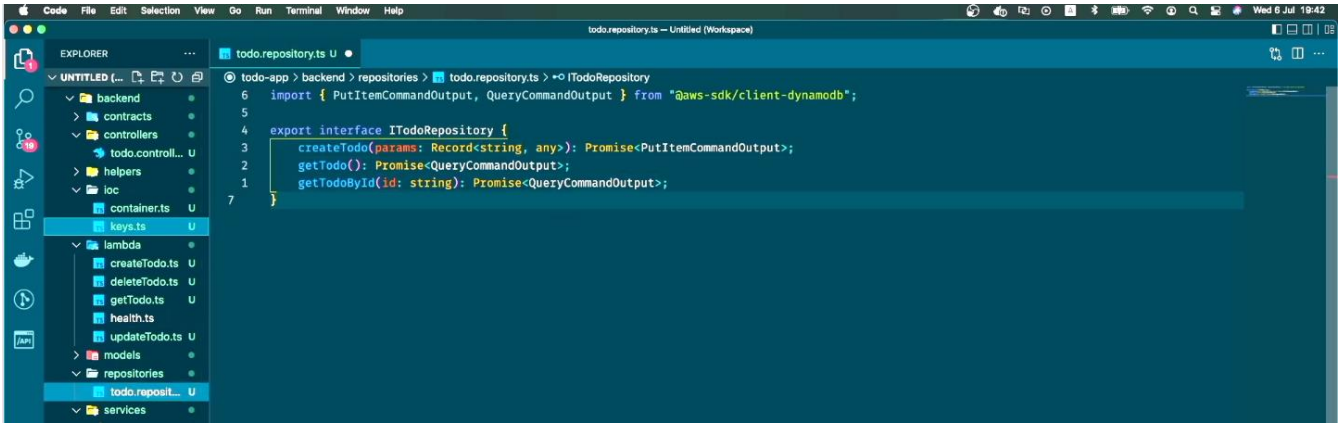

Code File Edit Selection View Go Run Terminal Window Help
Wed 6 Jul 19:35
logger.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
logger.service.ts U
todo-app > backend > services > logger.service.ts > LoggerService > writeLog
import { injectable } from "inversify";
export interface ILoggerService {
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: string, data?: any, className?: string | undefined): void {
throw new Error("Method not implemented.");
}
OUTLINE
TIMELINE
tutorial_4*
Git Graph
-- INSERT --
Ln 32, Col 9
Spaces: 4
UTF-8
LF
TypeScript
Go Live
Prettier

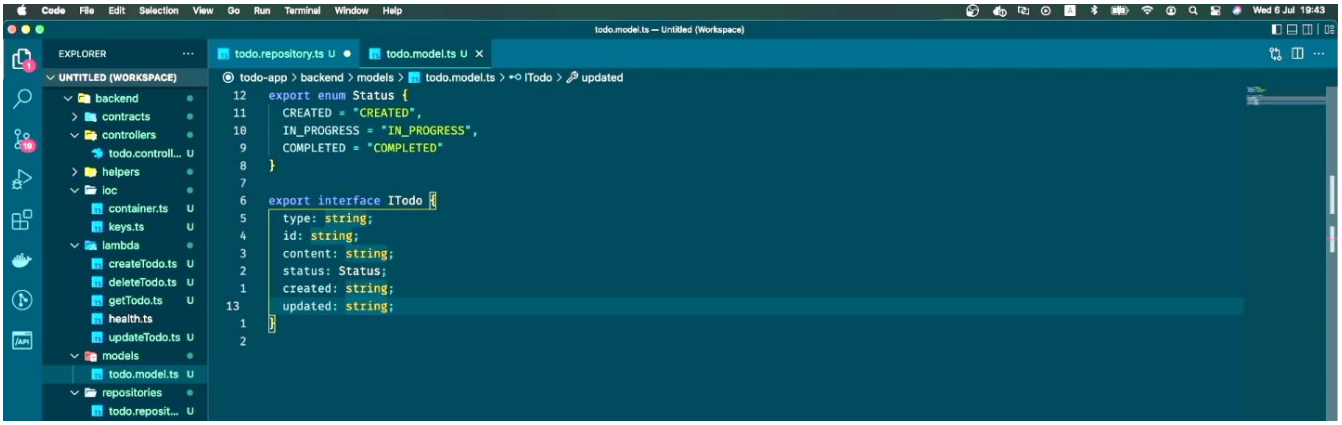

logger.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app > backend > services > logger.service.ts > LoggerService > log
log(message: string, data?: any, className?: string | undefined): void {
this.writeLog(message, data, className);

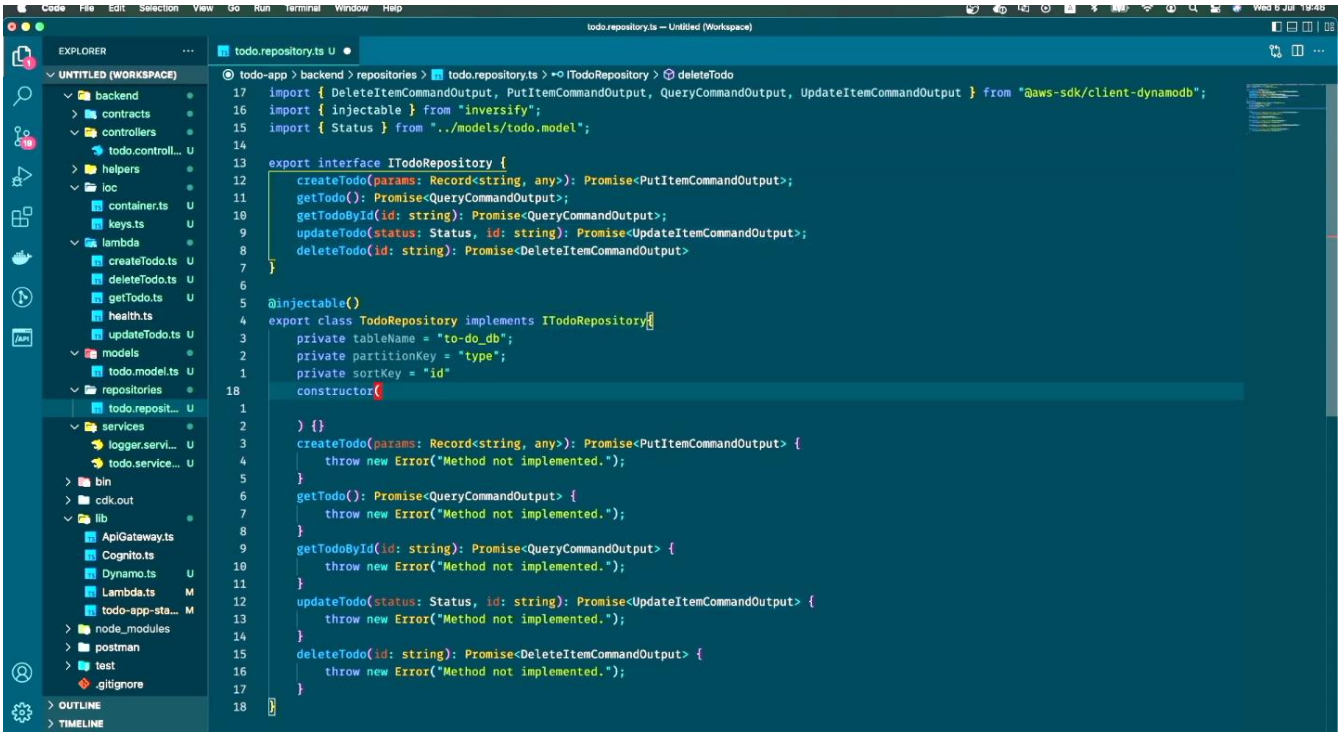

todo.repository.ts — Untitled (Workspace)
todo-app > backend > repositories > todo.repository.ts > ITodoRepository
export interface ITodoRepository

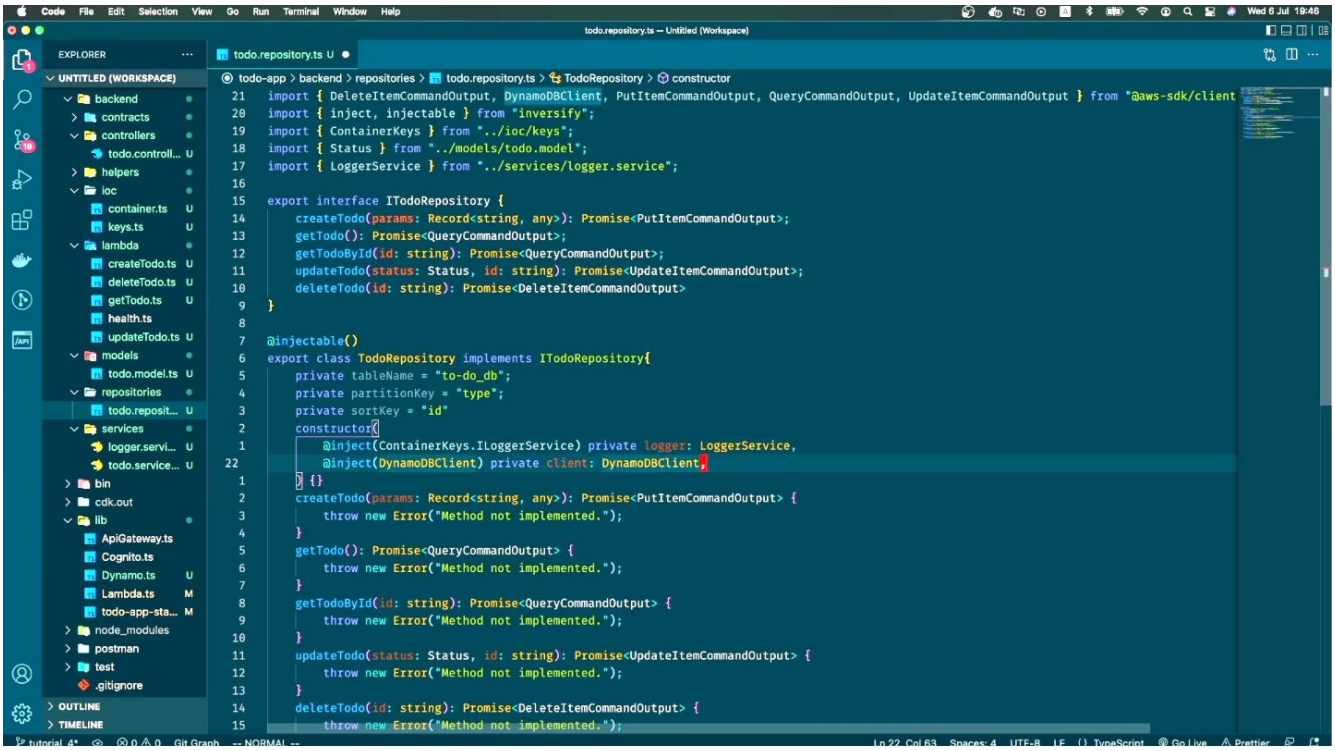

keys.ts — Untitled (Workspace)
todo-app > backend > ioc > keys.ts > ContainerKeys > ITodoController
export const ContainerKeys = {
ILoggerService: Symbol.for("ILoggerService"),
ITodoRepository: Symbol.for("ITodoRepository"),
ITodoService: Symbol.for("ITodoService"),
ITodoController: Symbol.for("ITodoController")
};

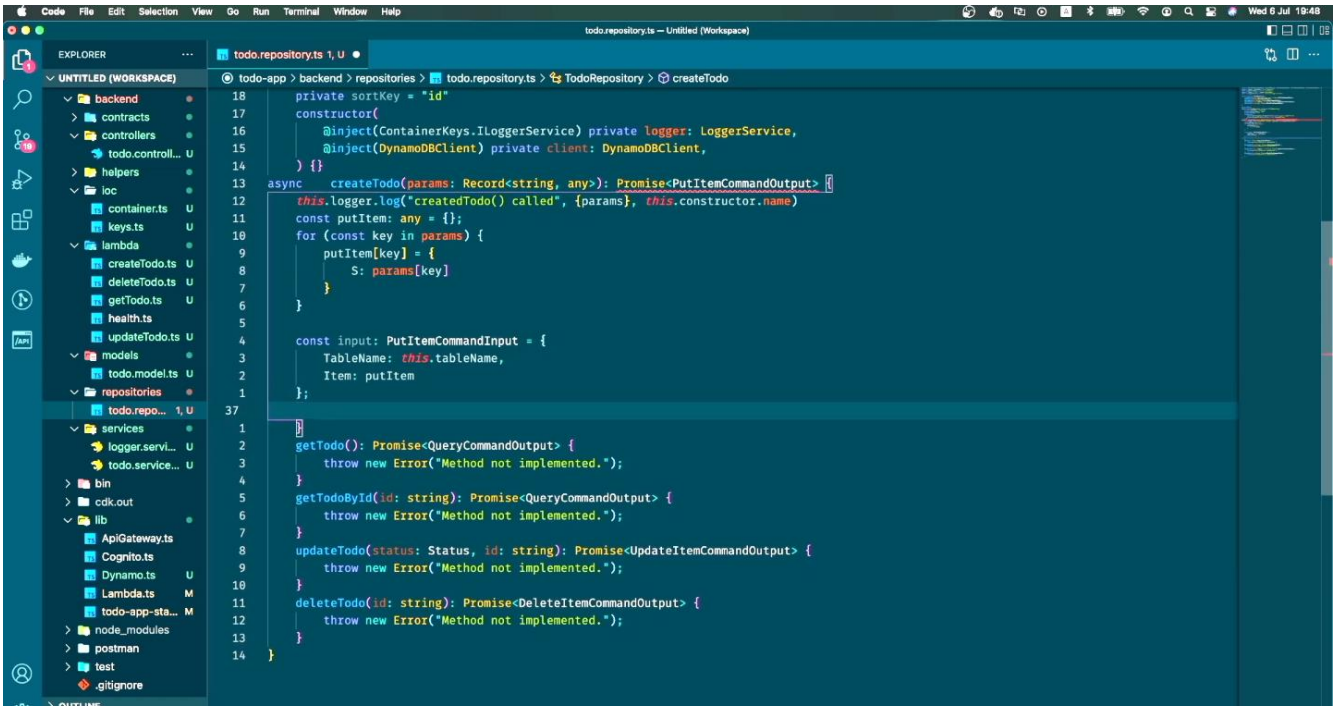

container.ts — Untitled (Workspace)
todo-app > backend > ioc > container.ts > ...
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { Container } from "inversify";
import { ILoggerService, LoggerService } from "../services/logger.service";
import { ContainerKeys } from "./keys";
const container = new Container();
try {
const dynamoClient = new DynamoDBClient({ region: "ap-southeast-2" });
container.bind(DynamoDBClient).toConstantValue(dynamoClient);
container.bind<ILoggerService>(ContainerKeys.ILoggerService).to(LoggerService);
} catch (err) {
console.log("Error during initialization", err);
}
export { container };

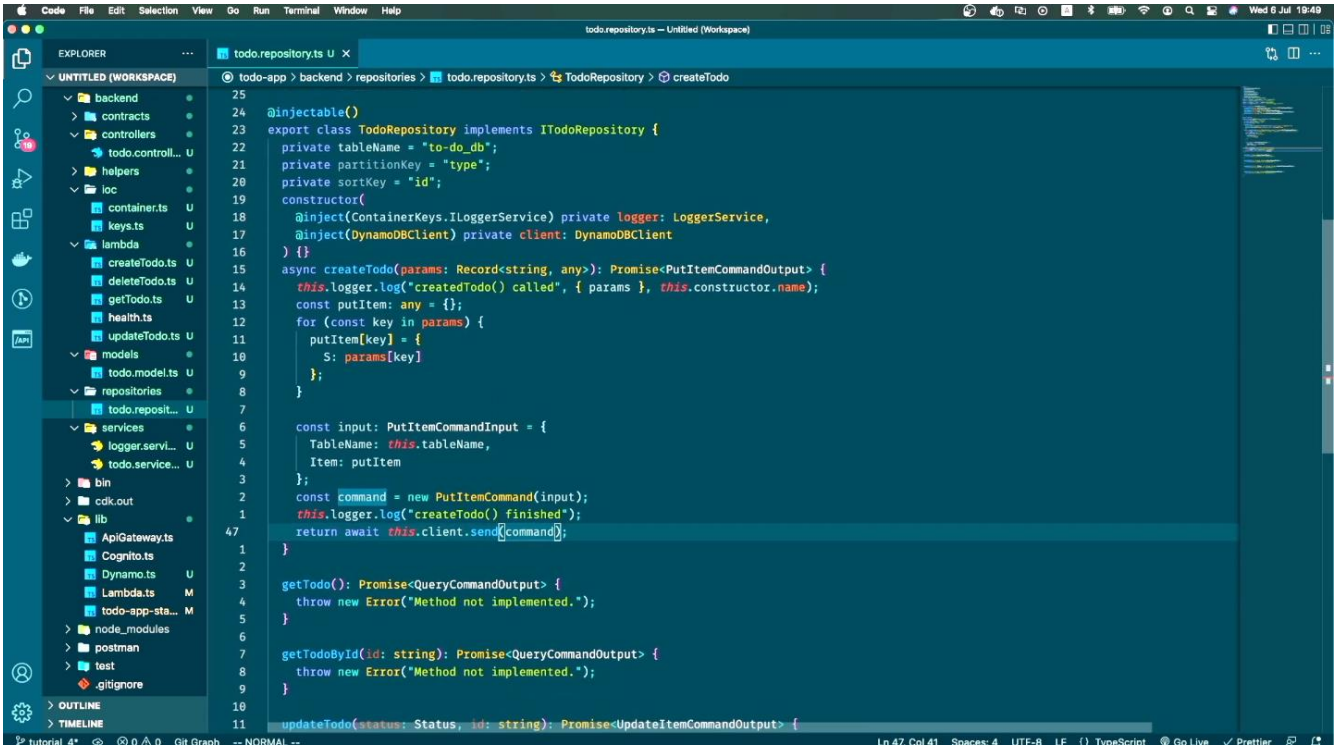

todo.repository.ts — Untitled (Workspace)
todo-app > backend > repositories > todo.repository.ts > ITodoRepository
import { PutItemCommandOutput, QueryCommandOutput } from "@aws-sdk/client-dynamodb";
export interface ITodoRepository {
createTodo(params: Record<string, any>): Promise<PutItemCommandOutput>;
getTodo(): Promise<QueryCommandOutput>;
getTodoById(id: string): Promise<QueryCommandOutput>;
}

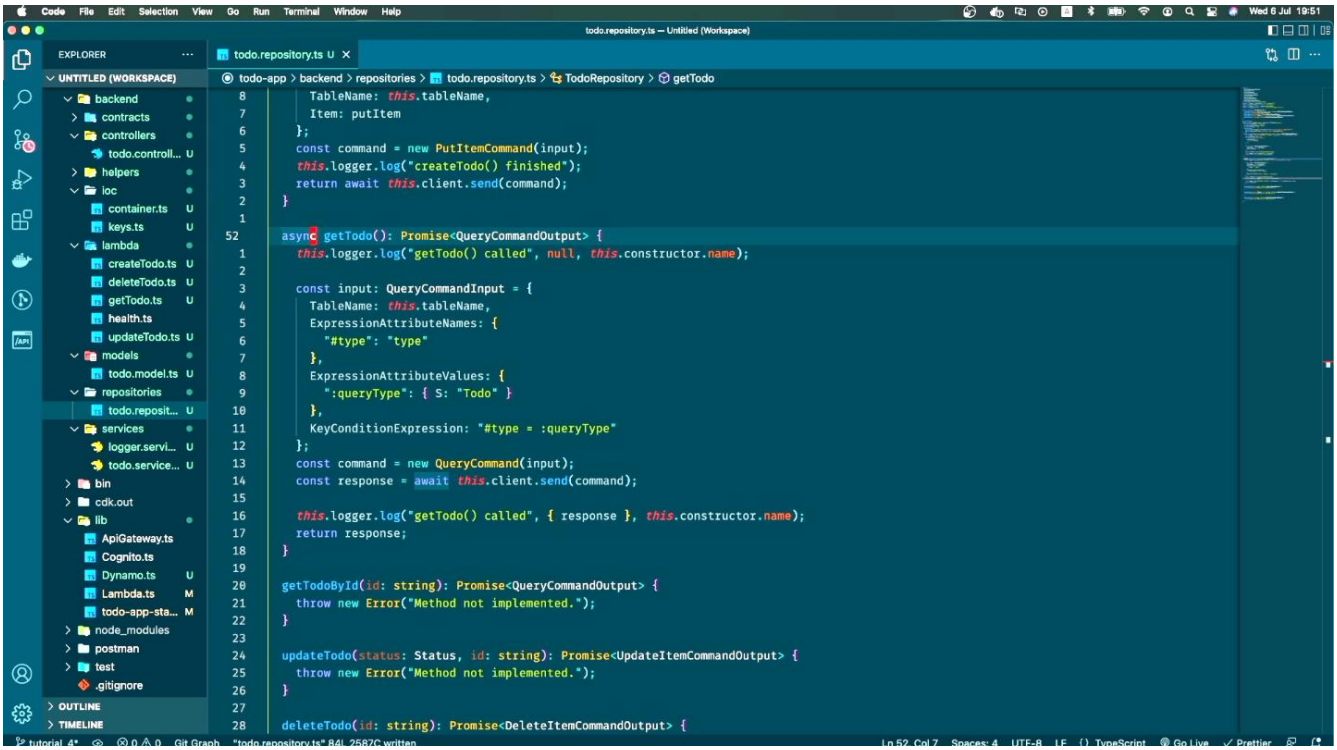

todo.model.ts — Untitled (Workspace)
todo-app > backend > models > todo.model.ts > ITodo > updated
export enum Status {
CREATED = "CREATED",
IN_PROGRESS = "IN_PROGRESS",
COMPLETED = "COMPLETED"
}
export interface ITodo {
type: string;
id: string;
content: string;
status: Status;
created: string;
updated: string;
}

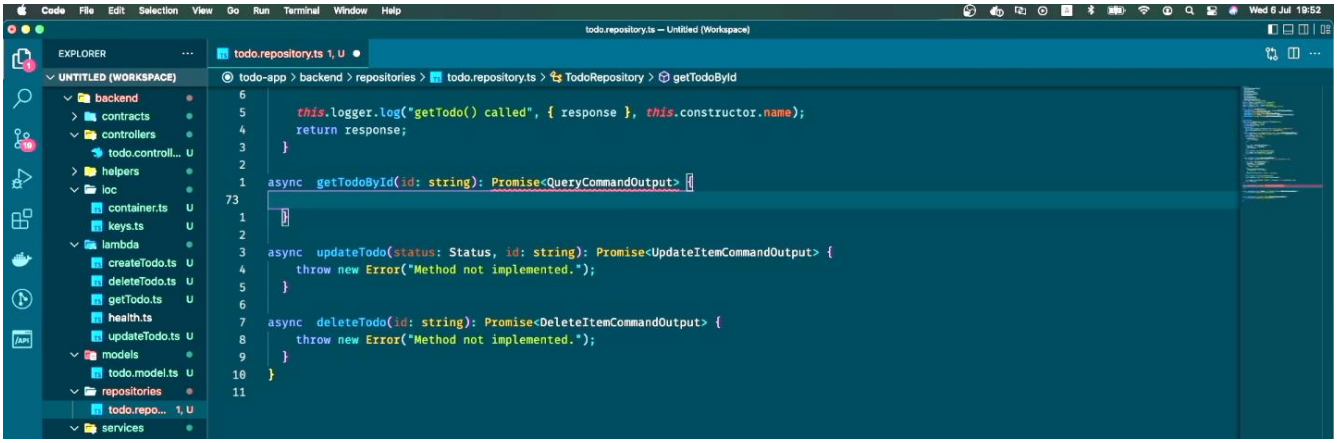

todo.repository.ts — Untitled (Workspace)
todo-app > backend > repositories > todo.repository.ts > ITodoRepository > deleteTodo
import { DeleteItemCommandOutput, PutItemCommandOutput, QueryCommandOutput, UpdateItemCommandOutput } from "@aws-sdk/client-dynamodb";
import { injectable } from "inversify";
import { Status } from "../models/todo.model";
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>
}
@injectable()
export class TodoRepository implements ITodoRepository{
private tableName = "to-do_db";
private partitionKey = "type";
private sortKey = "id"
constructor(
) {}
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.");
}
}

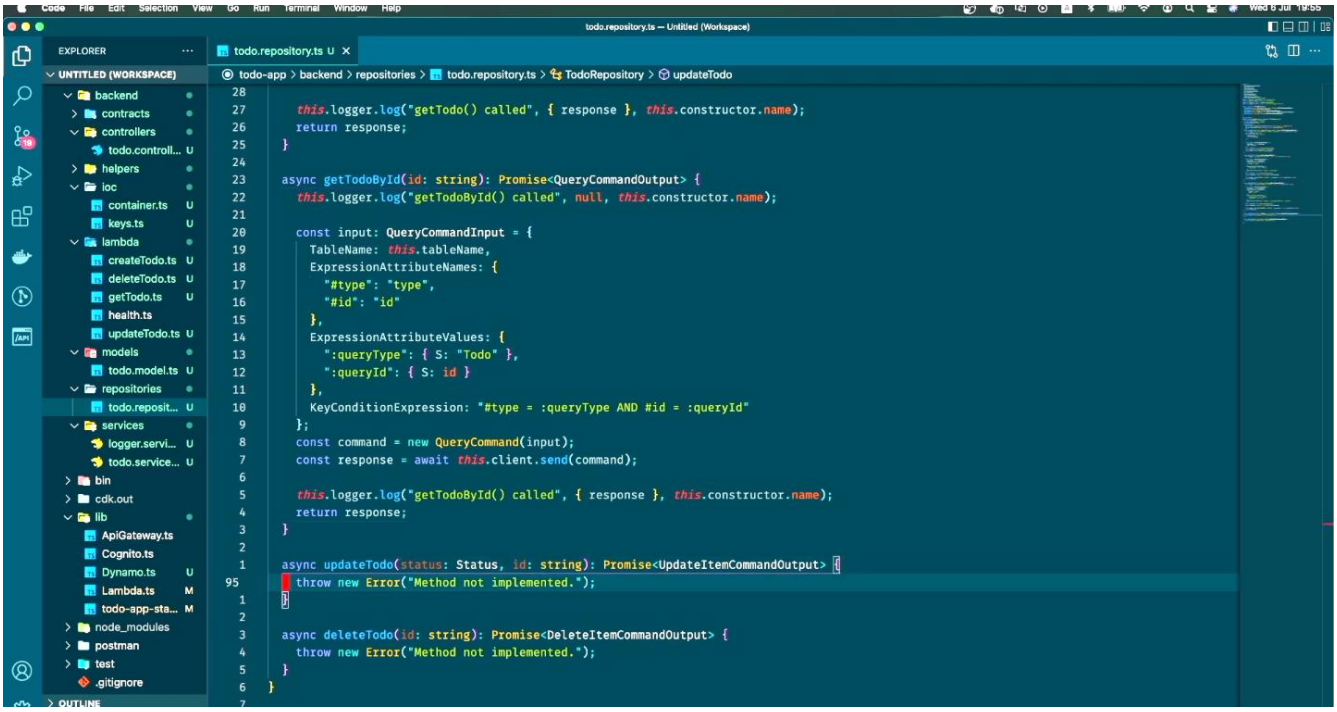

todo.repository.ts — Untitled (Workspace)
todo.repository.ts U
todo-app > backend > repositories > todo.repository.ts > TodoRepository > constructor
import { DeleteItemCommandOutput, DynamoDBClient, PutItemCommandOutput, QueryCommandOutput, UpdateItemCommandOutput } from "@aws-sdk/client
import { inject, injectable } from "inversify";
import { ContainerKeys } from "../ioc/keys";
import { Status } from "../models/todo.model";
import { LoggerService } from "../services/logger.service";
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>
}
@injectable()
export class TodoRepository implements ITodoRepository{
private tableName = "to-do_db";
private partitionKey = "type";
private sortKey = "id"
constructor(
@inject(ContainerKeys.ILoggerService) private logger: LoggerService,
@inject(DynamoDBClient) private client: DynamoDBClient,
) {}
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.");
EXPLORER
UNTITLED (WORKSPACE)
backend
contracts
controllers
todo.controll... U
helpers
ioc
container.ts U
keys.ts U
lambda
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.reposit... U
services
logger.servi... U
todo.service... U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
node_modules
postman
test
.gitignore
OUTLINE
TIMELINE

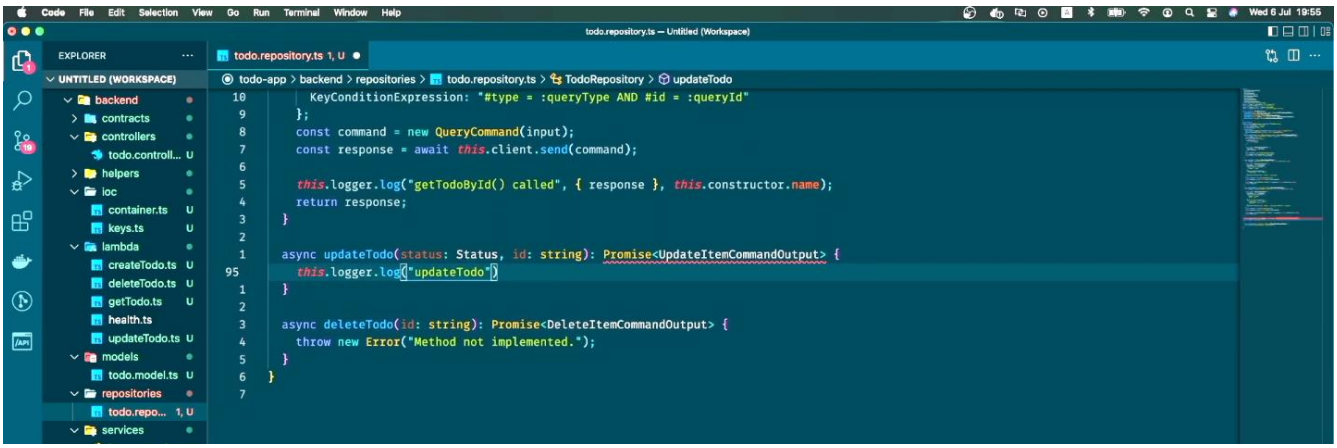

todo.repository.ts — Untitled (Workspace)
todo.repository.ts 1, U
todo-app > backend > repositories > todo.repository.ts > TodoRepository > createTodo
private sortKey = "id"
constructor(
@inject(ContainerKeys.ILoggerService) private logger: LoggerService,
@inject(DynamoDBClient) private client: DynamoDBClient,
) {}
async createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
this.logger.log("createdTodo() 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
};
}
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.");
}
}
EXPLORER
UNTITLED (WORKSPACE)
backend
contracts
controllers
todo.controll... U
helpers
ioc
container.ts U
keys.ts U
lambda
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.repo... 1, U
services
logger.servi... U
todo.service... U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
node_modules
postman
test
.gitignore

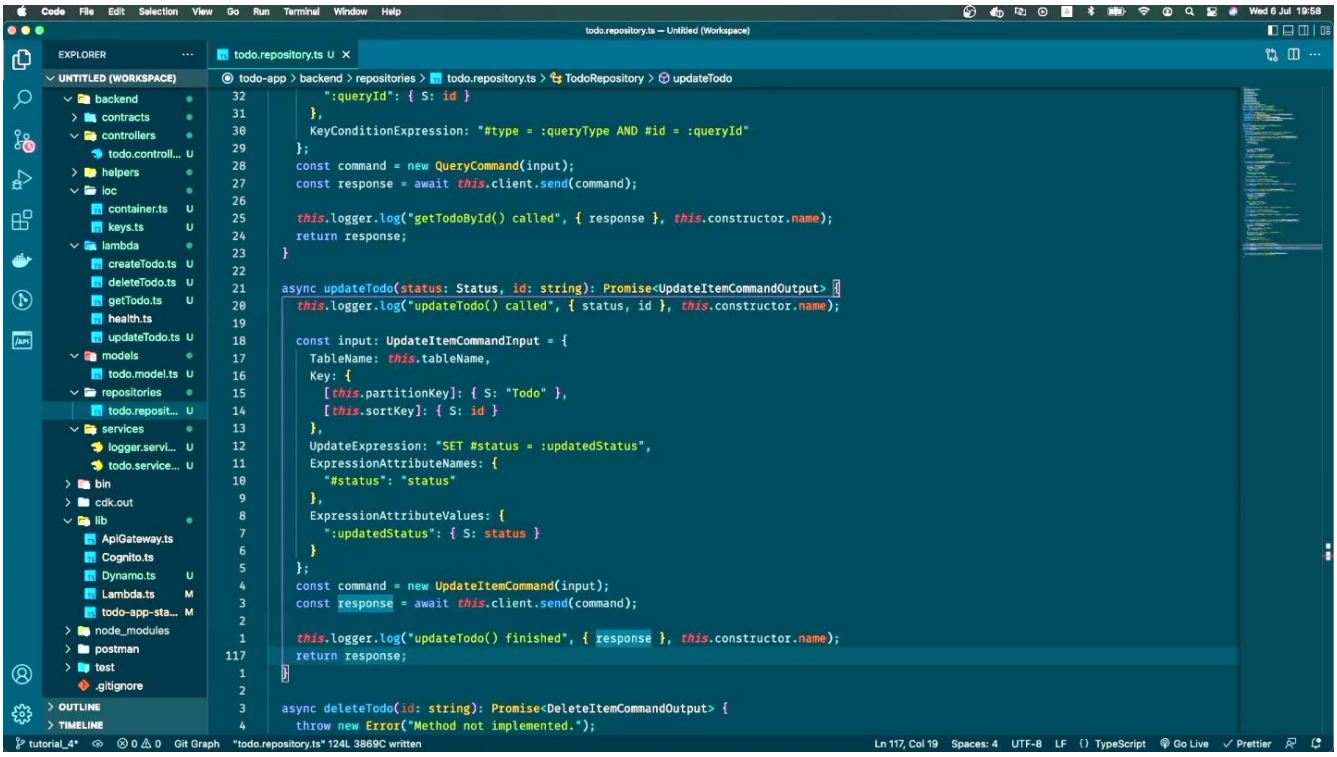

todo.repository.ts — Untitled (Workspace)
todo.repository.ts U
todo-app > backend > repositories > todo.repository.ts > TodoRepository > createTodo
@injectable()
export class TodoRepository implements ITodoRepository {
private tableName = "to-do_db";
private partitionKey = "type";
private sortKey = "id";
constructor(
@inject(ContainerKeys.ILoggerService) private logger: LoggerService,
@inject(DynamoDBClient) private client: DynamoDBClient
) {}
async createTodo(params: Record<string, any>): Promise<PutItemCommandOutput> {
this.logger.log("createdTodo() 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);
}
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> {

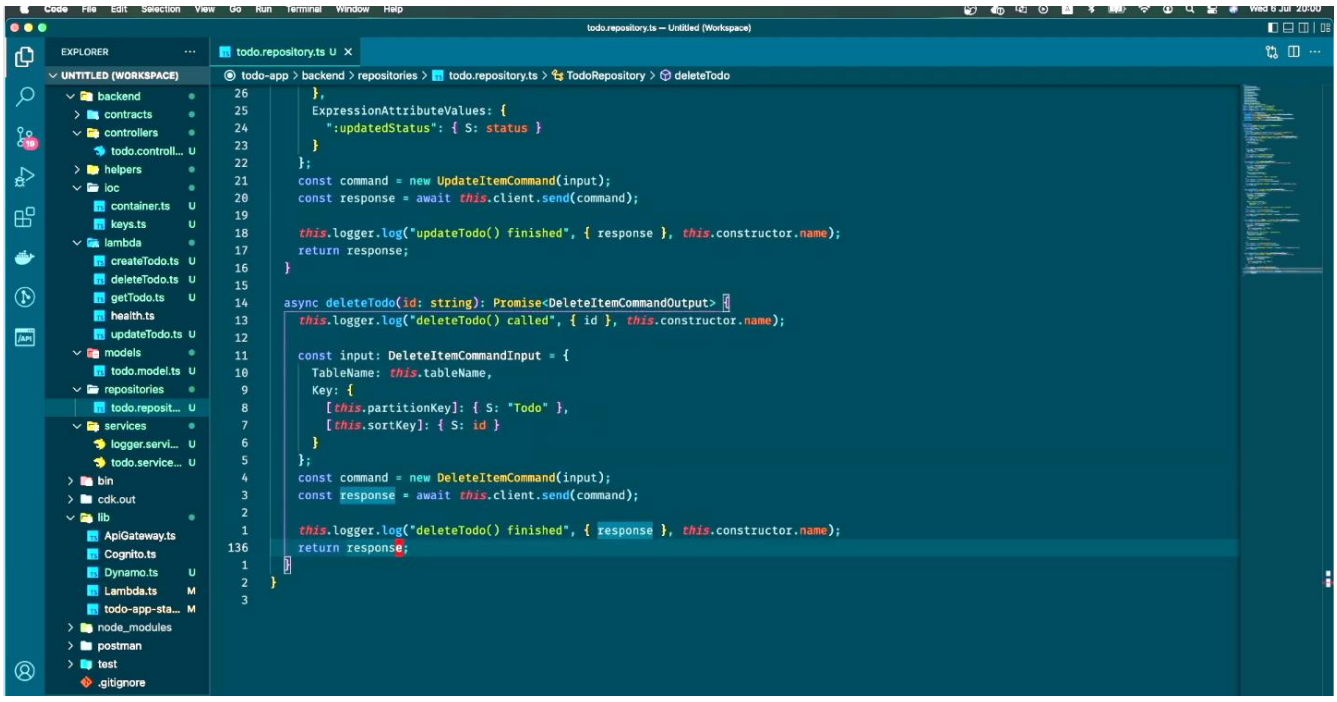

todo.repository.ts — Untitled (Workspace)
todo.repository.ts U
todo-app > backend > repositories > todo.repository.ts > TodoRepository > getTodo
TableName: this.tableName,
Item: putItem
};
const command = new PutItemCommand(input);
this.logger.log("createTodo() finished");
return await this.client.send(command);
}
async getTodo(): Promise<QueryCommandOutput> {
this.logger.log("getTodo() called", null, this.constructor.name);
const input: QueryCommandInput = {
TableName: this.tableName,
ExpressionAttributeNames: {
"#type": "type"
},
ExpressionAttributeValues: {
":queryType": { S: "Todo" }
},
KeyConditionExpression: "#type = :queryType"
};
const command = new QueryCommand(input);
const response = await this.client.send(command);
this.logger.log("getTodo() called", { response }, this.constructor.name);
return response;
}
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> {

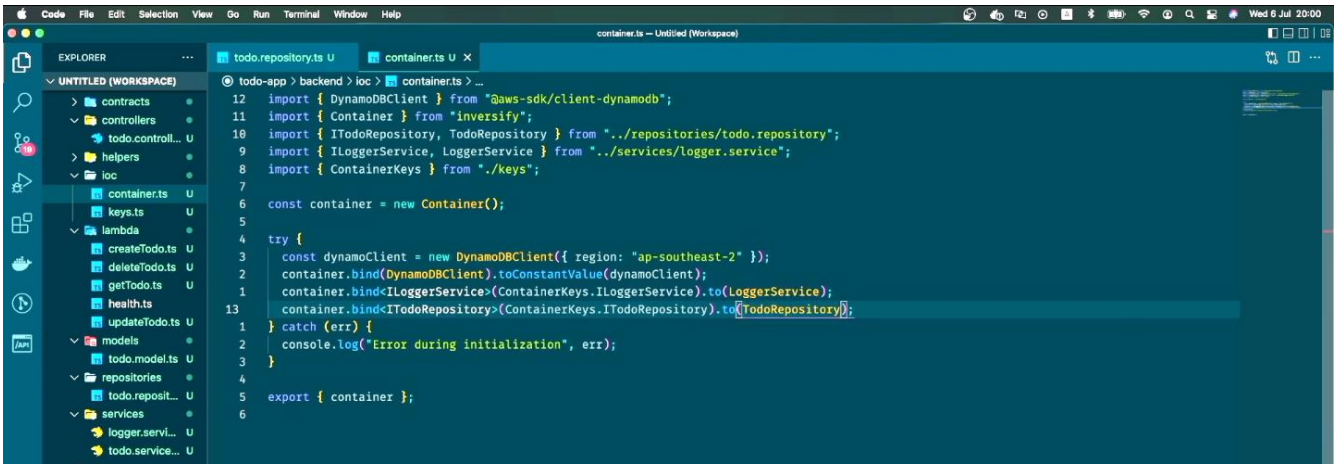

todo.repository.ts — Untitled (Workspace)
todo.repository.ts 1, U
todo-app > backend > repositories > todo.repository.ts > TodoRepository > getTodoById
this.logger.log("getTodo() called", { response }, this.constructor.name);
return response;
async getTodoById(id: string): Promise<QueryCommandOutput> {
async updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
throw new Error("Method not implemented.");
async deleteTodo(id: string): Promise<DeleteItemCommandOutput> {

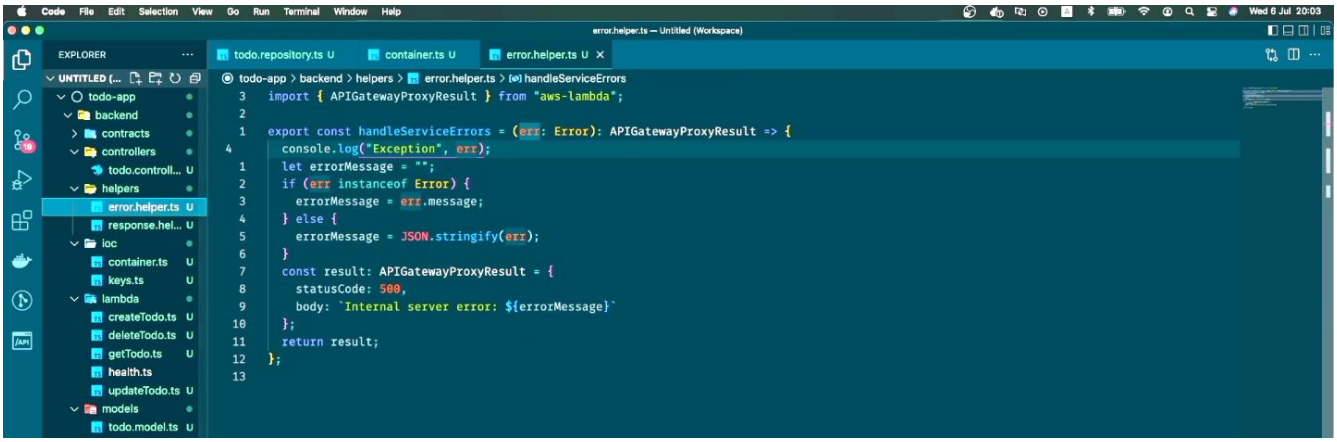

todo.repository.ts — Untitled (Workspace)
todo.repository.ts U
todo-app > backend > repositories > todo.repository.ts > TodoRepository > updateTodo
async getTodoById(id: string): Promise<QueryCommandOutput> {
this.logger.log("getTodoById() called", null, this.constructor.name);
const input: QueryCommandInput = {
TableName: this.tableName,
ExpressionAttributeNames: {
"#type": "type",
"#id": "id"
},
ExpressionAttributeValues: {
":queryType": { S: "Todo" },
":queryId": { S: id }
},
KeyConditionExpression: "#type = :queryType AND #id = :queryId"
};
const command = new QueryCommand(input);
const response = await this.client.send(command);
async updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
throw new Error("Method not implemented.");
async deleteTodo(id: string): Promise<DeleteItemCommandOutput> {

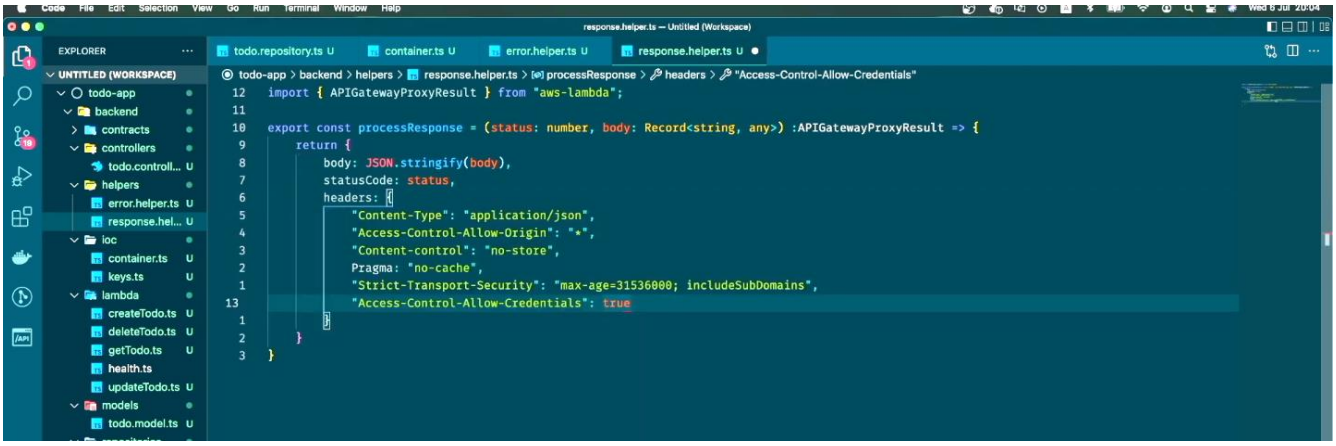

todo.repository.ts — Untitled (Workspace)
todo.repository.ts 1, U
todo-app > backend > repositories > todo.repository.ts > TodoRepository > updateTodo
this.logger.log("updateTodo")
async updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
async deleteTodo(id: string): Promise<DeleteItemCommandOutput> {

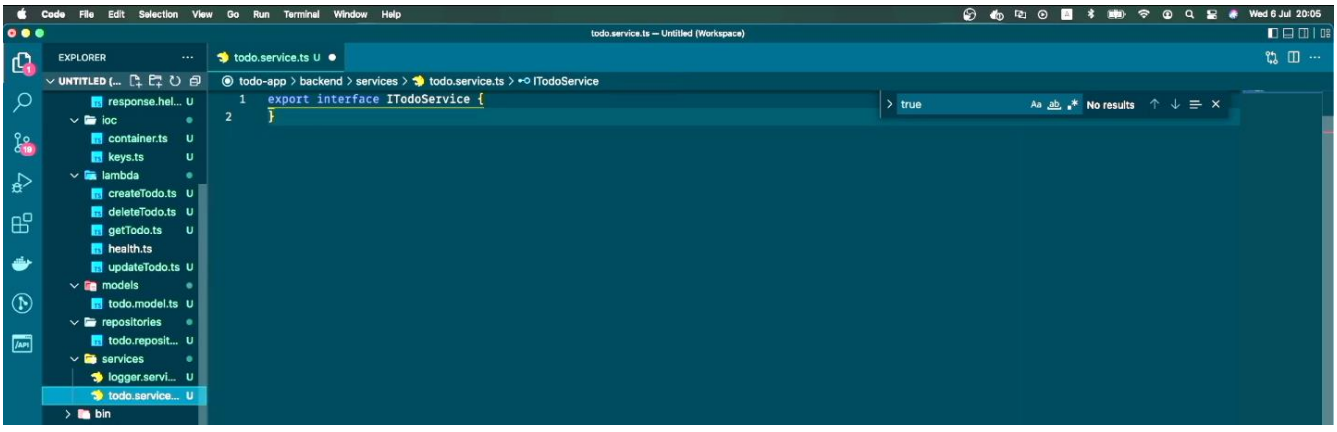

todo.repository.ts — Untitled (Workspace)
todo.repository.ts U
todo-app > backend > repositories > todo.repository.ts > TodoRepository > updateTodo
":queryId": { S: id }
},
KeyConditionExpression: "#type = :queryType AND #id = :queryId"
};
const command = new QueryCommand(input);
const response = await this.client.send(command);
this.logger.log("getTodoById() called", { response }, this.constructor.name);
return response;
}
async updateTodo(status: Status, id: string): Promise<UpdateItemCommandOutput> {
this.logger.log("updateTodo() called", { status, id }, this.constructor.name);
const input: UpdateItemCommandInput = {
TableName: this.tableName,
Key: {
[this.partitionKey]: { S: "Todo" },
[this.sortKey]: { S: id }
},
UpdateExpression: "SET #status = :updatedStatus",
ExpressionAttributeNames: {
"#status": "status"
},
ExpressionAttributeValues: {
":updatedStatus": { S: status }
}
};
const command = new UpdateItemCommand(input);
const response = await this.client.send(command);
this.logger.log("updateTodo() finished", { response }, this.constructor.name);
return response;
}
async deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
throw new Error("Method not implemented.");
EXPLORER
UNTITLED (WORKSPACE)
backend
contracts
controllers
todo.controll... U
helpers
ioc
container.ts U
keys.ts U
lambda
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.reposit... U
services
logger.servi... U
todo.service... U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
node_modules
postman
test
.gitignore
OUTLINE
TIMELINE
tutorial_4*
Git Graph
"todo.repository.ts" 124L 3869C written
Ln 117, Col 19
Spaces: 4
UTF-8
LF
TypeScript
Go Live
Prettier

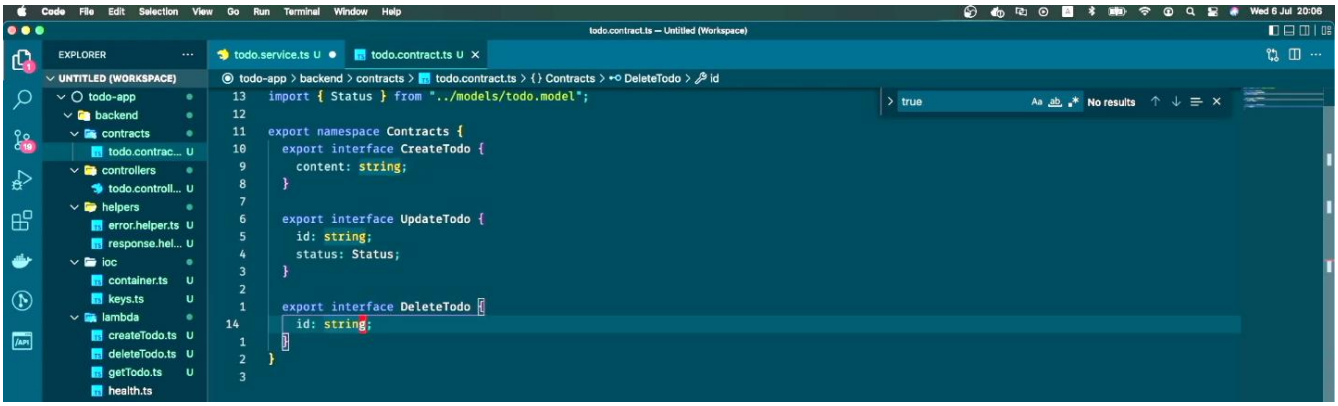

todo.repository.ts — Untitled (Workspace)
todo.repository.ts U
todo-app > backend > repositories > todo.repository.ts > TodoRepository > deleteTodo
},
ExpressionAttributeValues: {
":updatedStatus": { S: status }
}
};
const command = new UpdateItemCommand(input);
const response = await this.client.send(command);
this.logger.log("updateTodo() finished", { response }, this.constructor.name);
return response;
}
async deleteTodo(id: string): Promise<DeleteItemCommandOutput> {
this.logger.log("deleteTodo() called", { id }, this.constructor.name);
const input: DeleteItemCommandInput = {
TableName: this.tableName,
Key: {
[this.partitionKey]: { S: "Todo" },
[this.sortKey]: { S: id }
}
};
const command = new DeleteItemCommand(input);
const response = await this.client.send(command);
this.logger.log("deleteTodo() finished", { response }, this.constructor.name);
return response;
}
}
EXPLORER
UNTITLED (WORKSPACE)
backend
contracts
controllers
todo.controll... U
helpers
ioc
container.ts U
keys.ts U
lambda
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.reposit... U
services
logger.servi... U
todo.service... U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
node_modules
postman
test
.gitignore

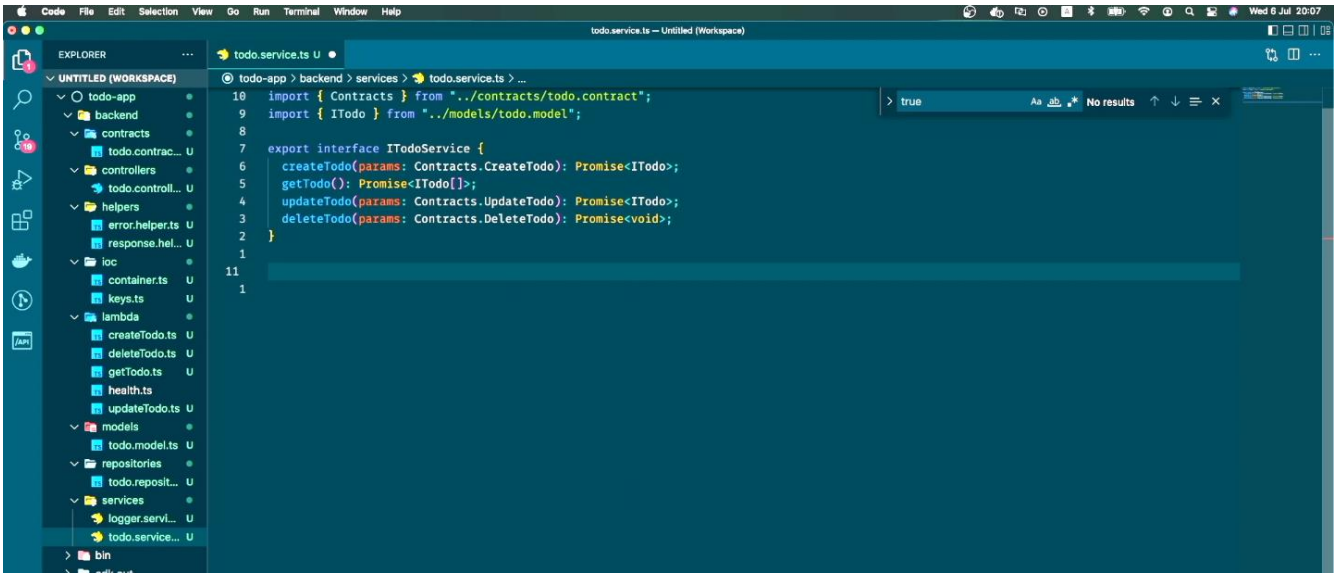

container.ts — Untitled (Workspace)
todo.repository.ts U
container.ts U
todo-app > backend > ioc > container.ts > ...
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { Container } from "inversify";
import { ITodoRepository, TodoRepository } from "../repositories/todo.repository";
import { ILoggerService, LoggerService } from "../services/logger.service";
import { 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);
} catch (err) {
console.log("Error during initialization", err);
}
export { container };

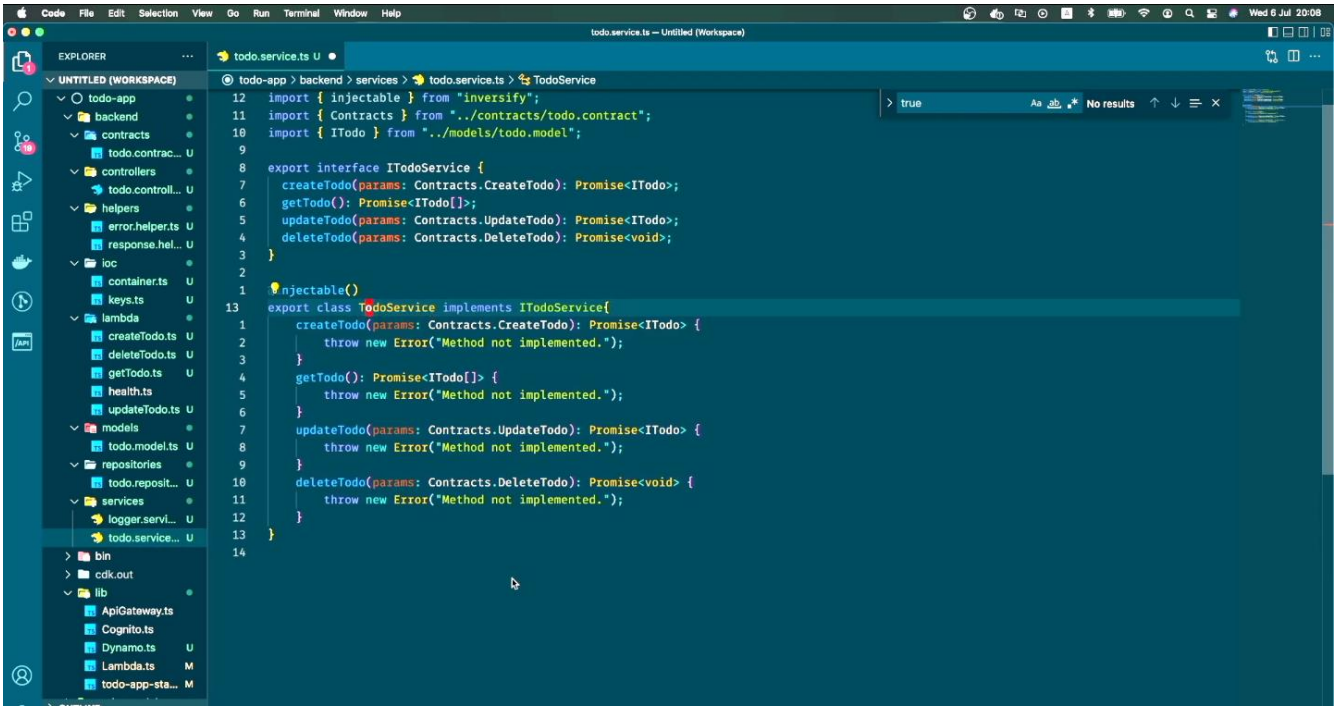

error.helper.ts — Untitled (Workspace)
todo.repository.ts U
container.ts U
error.helper.ts U
todo-app > backend > helpers > error.helper.ts > handleServiceErrors
import { APIGatewayProxyResult } from "aws-lambda";
export const handleServiceErrors = (err: Error): APIGatewayProxyResult => {
console.log("Exception", err);
let errorMessage = "";
if (err instanceof Error) {
errorMessage = err.message;
} else {
errorMessage = JSON.stringify(err);
}
const result: APIGatewayProxyResult = {
statusCode: 500,
body: `Internal server error: ${errorMessage}`
};
return result;
};

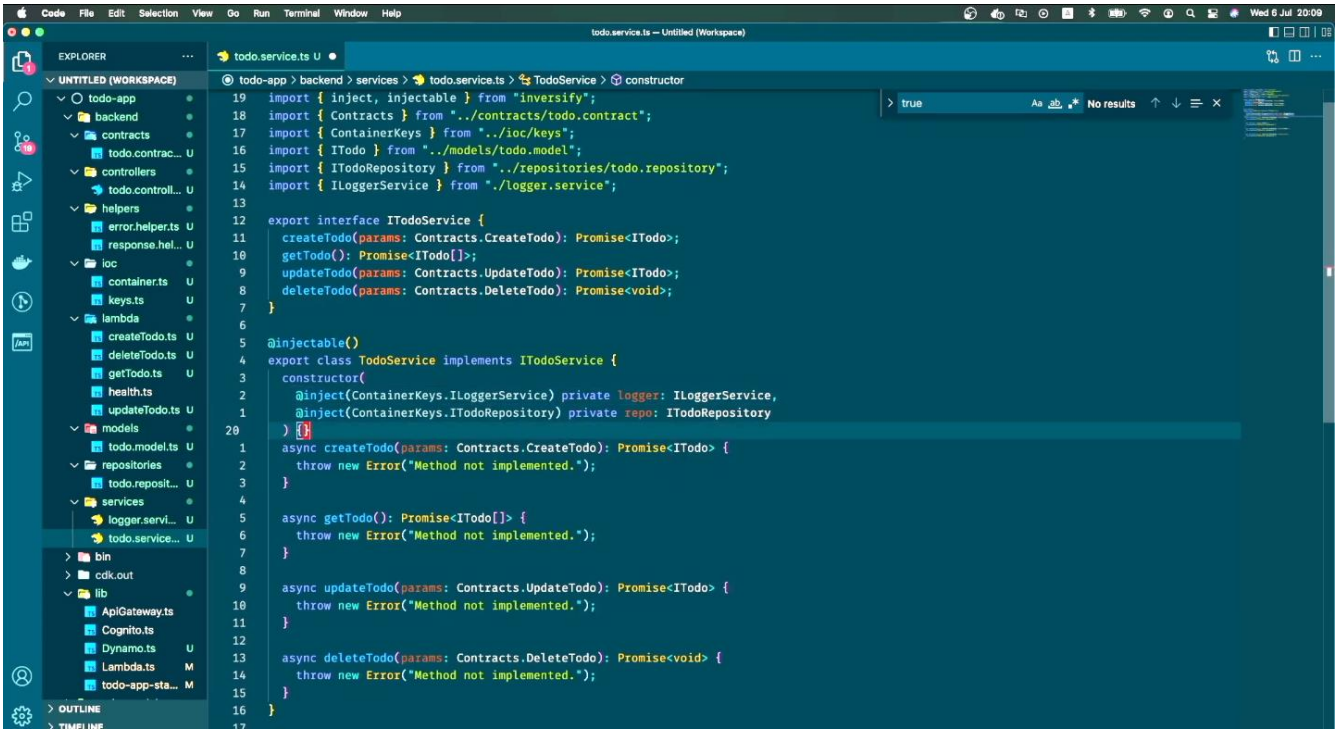

response.helper.ts — Untitled (Workspace)
todo.repository.ts U
container.ts U
error.helper.ts U
response.helper.ts U
todo-app > backend > helpers > response.helper.ts > processResponse > headers > "Access-Control-Allow-Credentials"
import { APIGatewayProxyResult } from "aws-lambda";
export const processResponse = (status: number, body: Record<string, any>) :APIGatewayProxyResult => {
return {
body: JSON.stringify(body),
statusCode: status,
headers: {
"Content-Type": "application/json",
"Access-Control-Allow-Origin": "*",
"Content-control": "no-store",
Pragma: "no-cache",
"Strict-Transport-Security": "max-age=31536000; includeSubDomains",
"Access-Control-Allow-Credentials": true
}
}
}

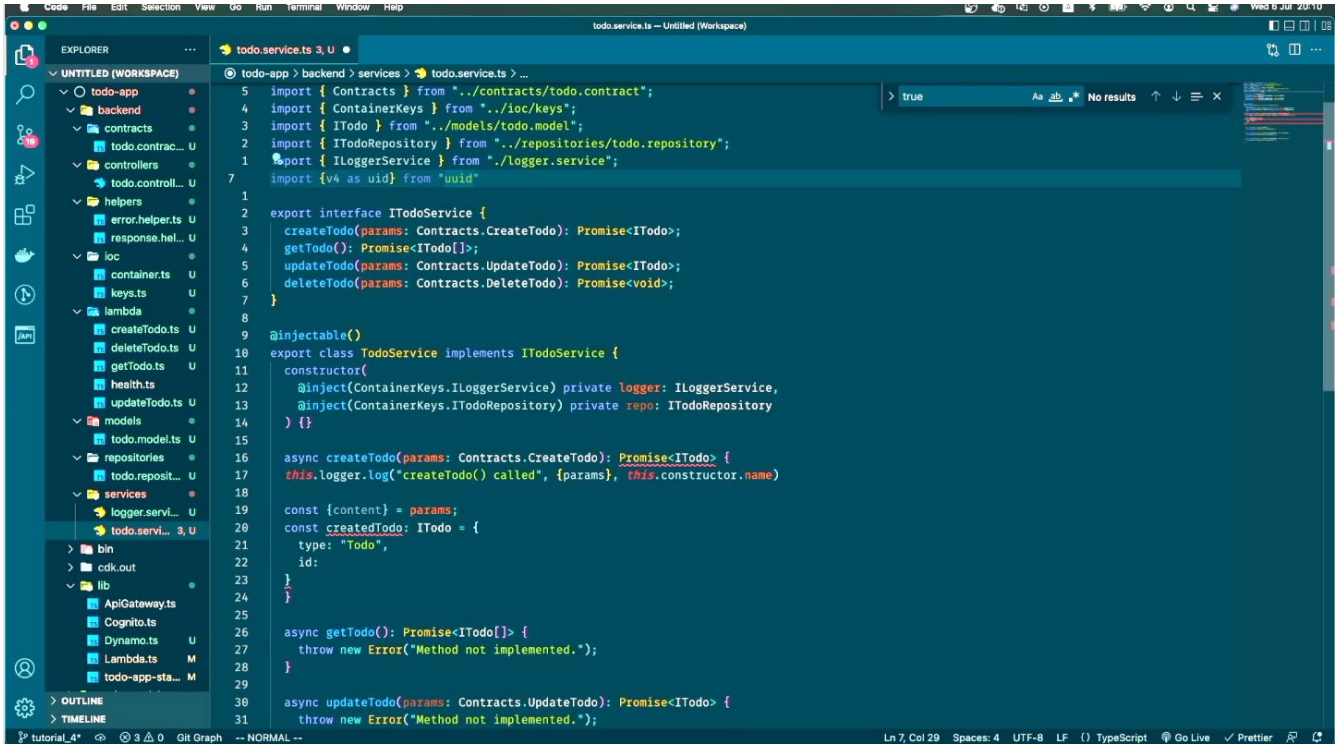

todo.service.ts — Untitled (Workspace)
todo.service.ts U
todo-app > backend > services > todo.service.ts > ITodoService
export interface ITodoService {
}
true
No results

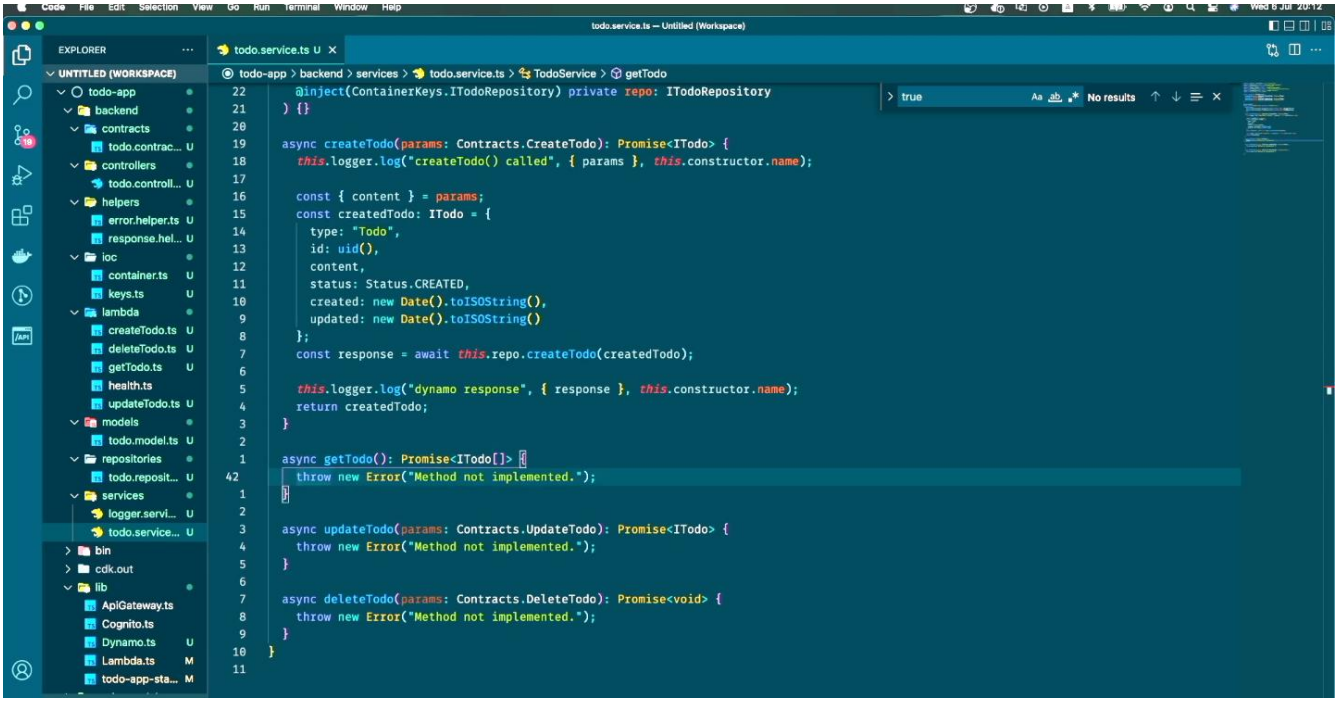

todo.contract.ts — Untitled (Workspace)
todo.service.ts U
todo.contract.ts U
todo-app > backend > contracts > todo.contract.ts > { } Contracts > DeleteTodo > id
import { Status } from "../models/todo.model";
export namespace Contracts {
export interface CreateTodo {
content: string;
}
export interface UpdateTodo {
id: string;
status: Status;
}
export interface DeleteTodo {
id: string;
}
}
true
No results

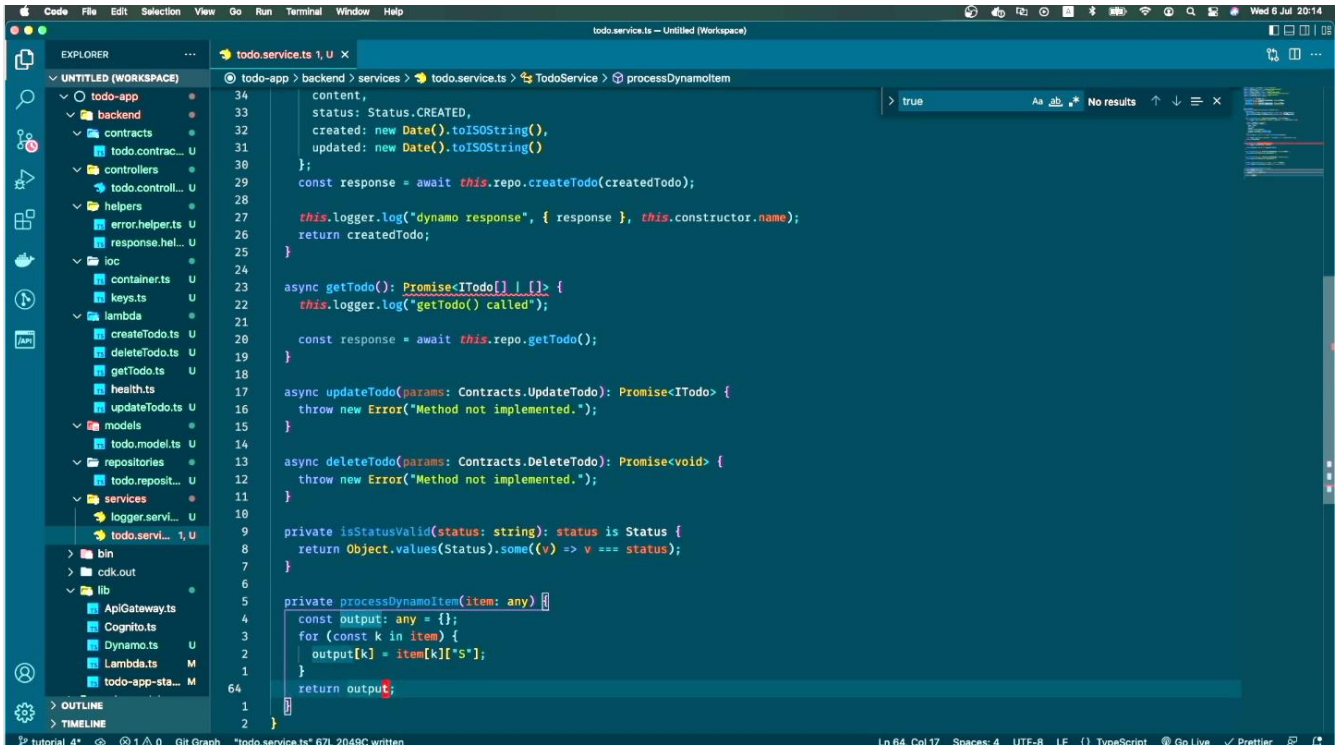

todo.service.ts — Untitled (Workspace)
todo.service.ts U
todo-app > backend > services > todo.service.ts > ...
import { Contracts } from "../contracts/todo.contract";
import { ITodo } from "../models/todo.model";
export interface ITodoService {
createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
getTodo(): Promise<ITodo[]>;
updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
}
true
No results

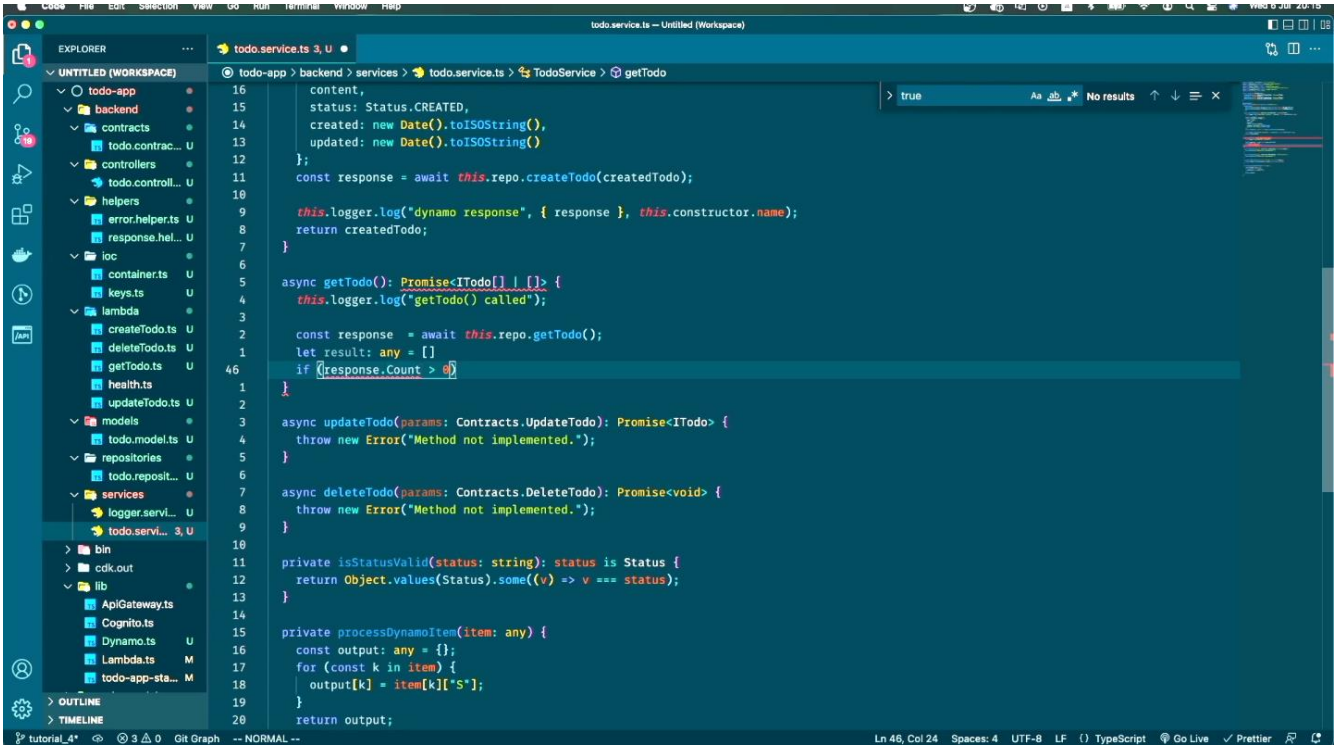

todo.service.ts — Untitled (Workspace)
todo.service.ts U
todo-app > backend > services > todo.service.ts > TodoService
import { injectable } from "inversify";
import { Contracts } from "../contracts/todo.contract";
import { ITodo } from "../models/todo.model";
export interface ITodoService {
createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
getTodo(): Promise<ITodo[]>;
updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
}
@injectable()
export class TodoService implements ITodoService{
createTodo(params: Contracts.CreateTodo): Promise<ITodo> {
throw new Error("Method not implemented.");
}
getTodo(): Promise<ITodo[]> {
throw new Error("Method not implemented.");
}
updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
throw new Error("Method not implemented.");
}
deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
throw new Error("Method not implemented.");
}
}
true
No results

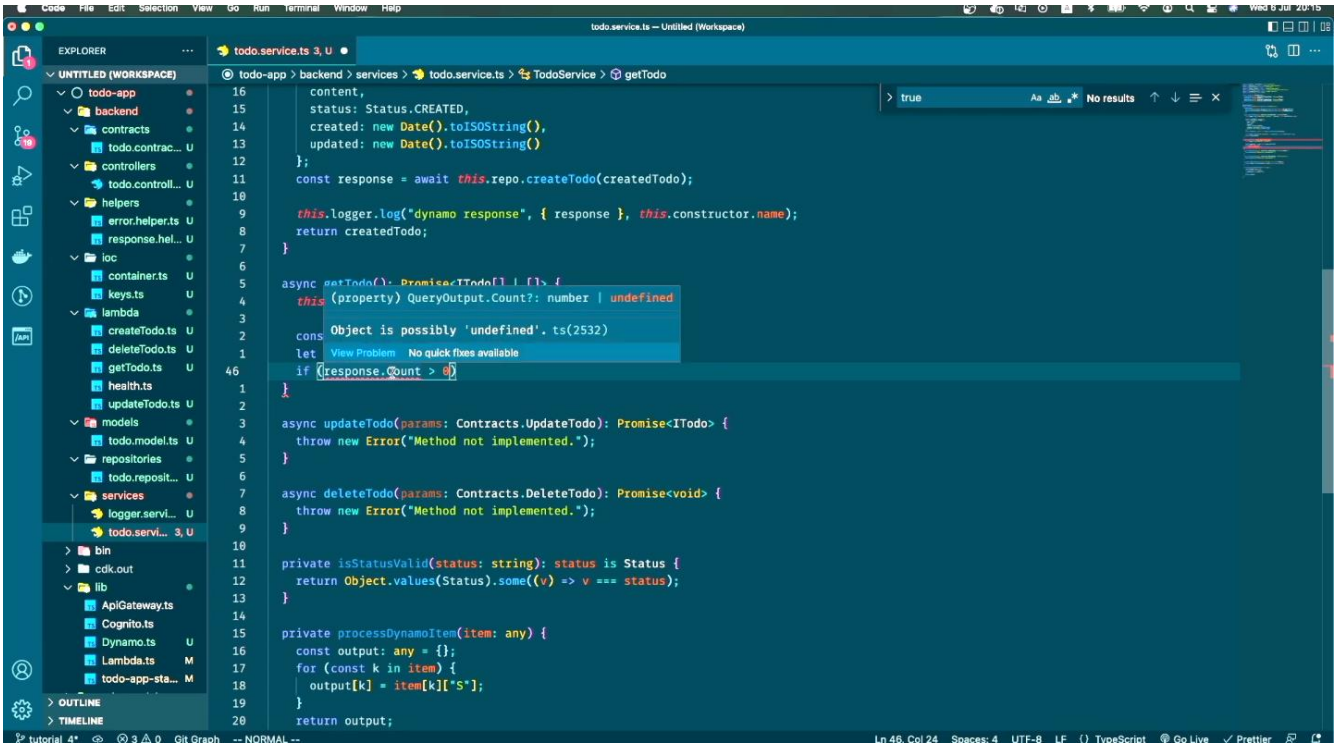

todo.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
contracts
todo.contrac... U
controllers
todo.controll... U
helpers
error.helper.ts U
response.hel... U
ioc
container.ts U
keys.ts U
lambda
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.reposit... U
services
logger.servi... U
todo.service... U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
OUTLINE
TIMELINE
todo.service.ts U
todo-app > backend > services > todo.service.ts > TodoService > constructor
true
No results
import { inject, injectable } from "inversify";
import { Contracts } from "../contracts/todo.contract";
import { ContainerKeys } from "../ioc/keys";
import { ITodo } from "../models/todo.model";
import { ITodoRepository } from "../repositories/todo.repository";
import { ILoggerService } from "./logger.service";
export interface ITodoService {
createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
getTodo(): Promise<ITodo[]>;
updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
}
@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> {
throw new Error("Method not implemented.");
}
async getTodo(): Promise<ITodo[]> {
throw new Error("Method not implemented.");
}
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.");
}
}

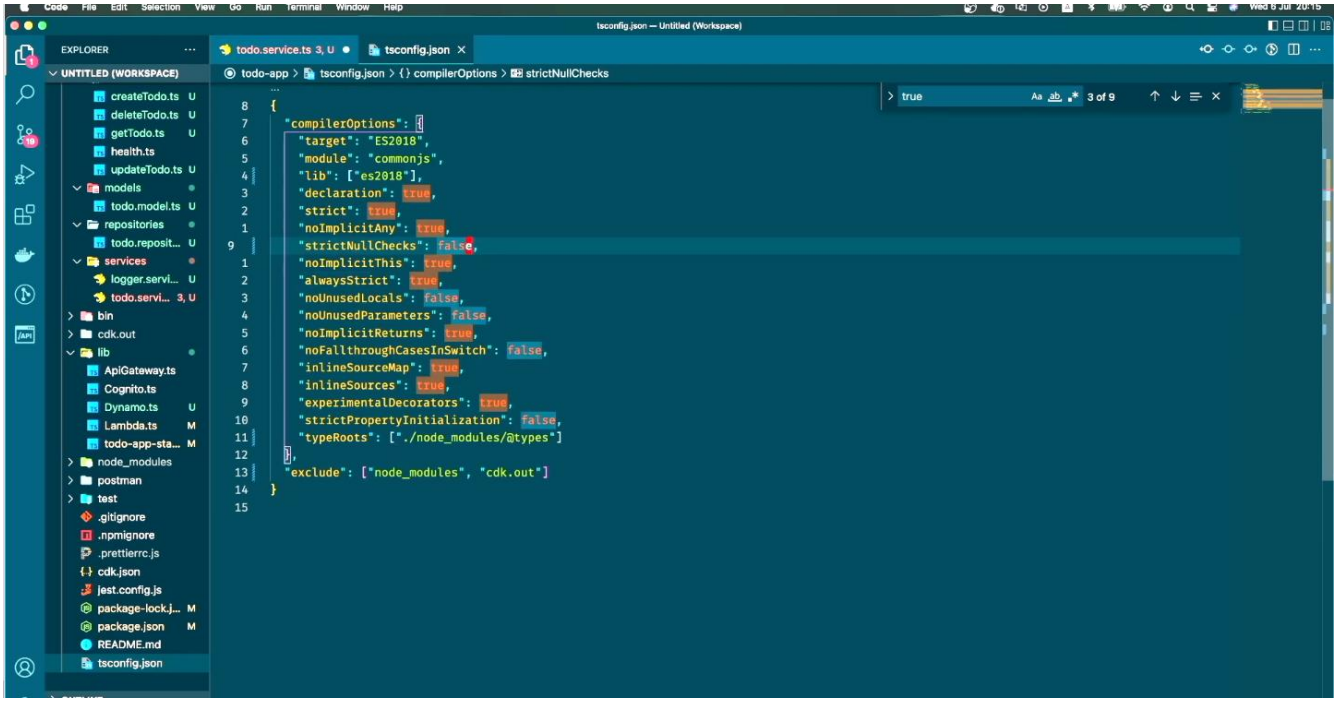

todo.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
contracts
todo.contrac... U
controllers
todo.controll... U
helpers
error.helper.ts U
response.hel... U
ioc
container.ts U
keys.ts U
lambda
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.reposit... U
services
logger.servi... U
todo.servi... 3, U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
OUTLINE
TIMELINE
todo.service.ts 3, U
todo-app > backend > services > todo.service.ts > ...
true
No results
import { Contracts } from "../contracts/todo.contract";
import { ContainerKeys } from "../ioc/keys";
import { ITodo } from "../models/todo.model";
import { ITodoRepository } from "../repositories/todo.repository";
import { ILoggerService } from "./logger.service";
import {v4 as uid} from "uuid"
export interface ITodoService {
createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
getTodo(): Promise<ITodo[]>;
updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
}
@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:
}
}
async getTodo(): Promise<ITodo[]> {
throw new Error("Method not implemented.");
}
async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
throw new Error("Method not implemented.");
tutorial_4* 3 0 Git Graph -- NORMAL --
Ln 7, Col 29 Spaces: 4 UTF-8 LF {} TypeScript Go Live Prettier

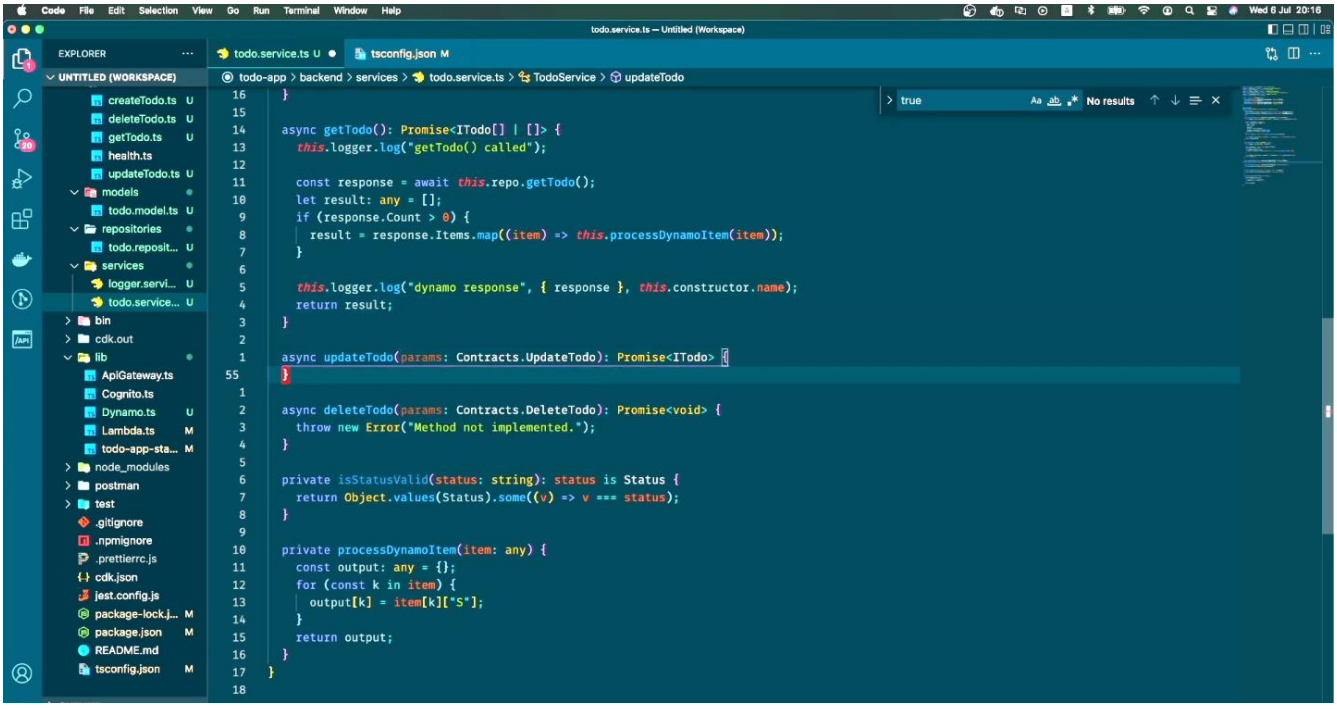

todo.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
contracts
todo.contrac... U
controllers
todo.controll... U
helpers
error.helper.ts U
response.hel... U
ioc
container.ts U
keys.ts U
lambda
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.reposit... U
services
logger.servi... U
todo.service... U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
todo.service.ts U
todo-app > backend > services > todo.service.ts > TodoService > getTodo
@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[]> {
throw new Error("Method not implemented.");
}
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.");
}
}
true
No results

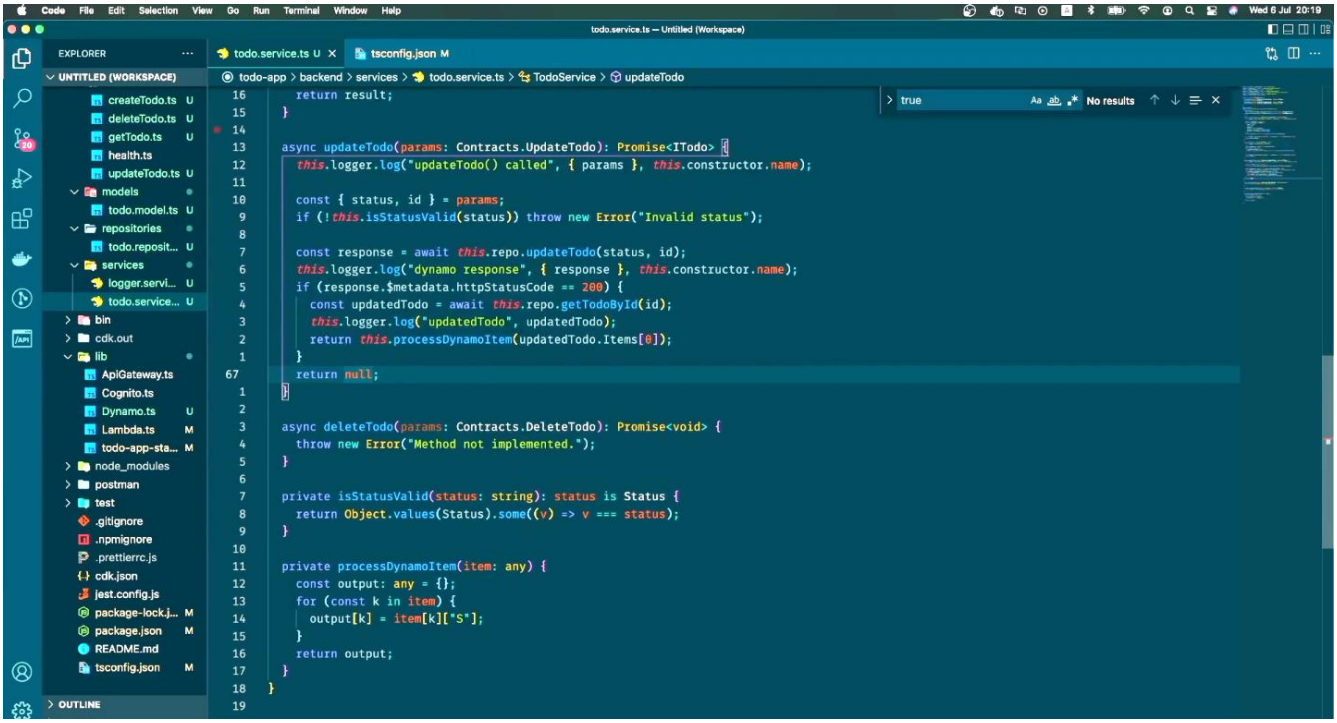

todo.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
contracts
todo.contrac... U
controllers
todo.controll... U
helpers
error.helper.ts U
response.hel... U
ioc
container.ts U
keys.ts U
lambda
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.reposit... U
services
logger.servi... U
todo.servi... 1, U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
OUTLINE
TIMELINE
todo.service.ts 1, U
todo-app > backend > services > todo.service.ts > TodoService > processDynamoItem
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();
}
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;
}
}
true
No results

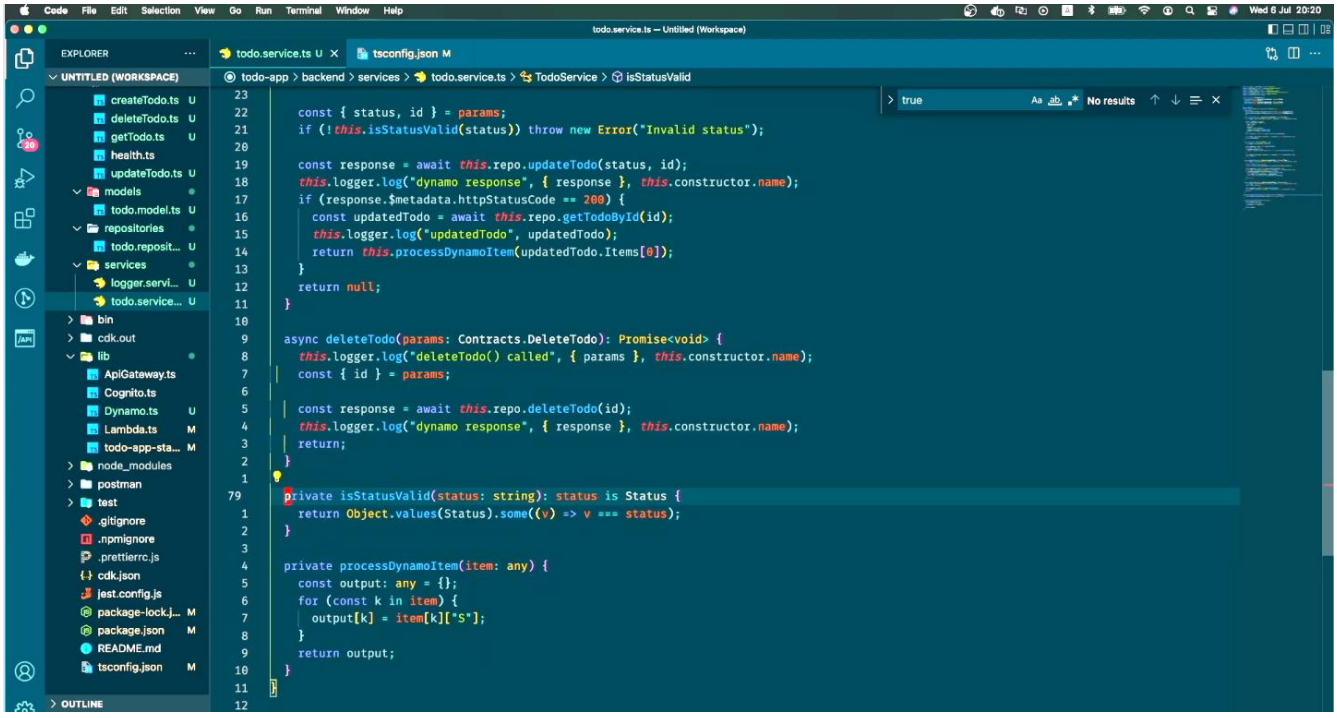

todo.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
contracts
todo.contrac... U
controllers
todo.controll... U
helpers
error.helper.ts U
response.hel... U
ioc
container.ts U
keys.ts U
lambda
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.reposit... U
services
logger.servi... U
todo.servi... 3, U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
OUTLINE
TIMELINE
todo.service.ts 3, U
todo-app > backend > services > todo.service.ts > TodoService > getTodo
true
No results
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;
tutorial_4* 3 0 Git Graph -- NORMAL --
Ln 46, Col 24 Spaces: 4 UTF-8 LF {} TypeScript Go Live Prettier

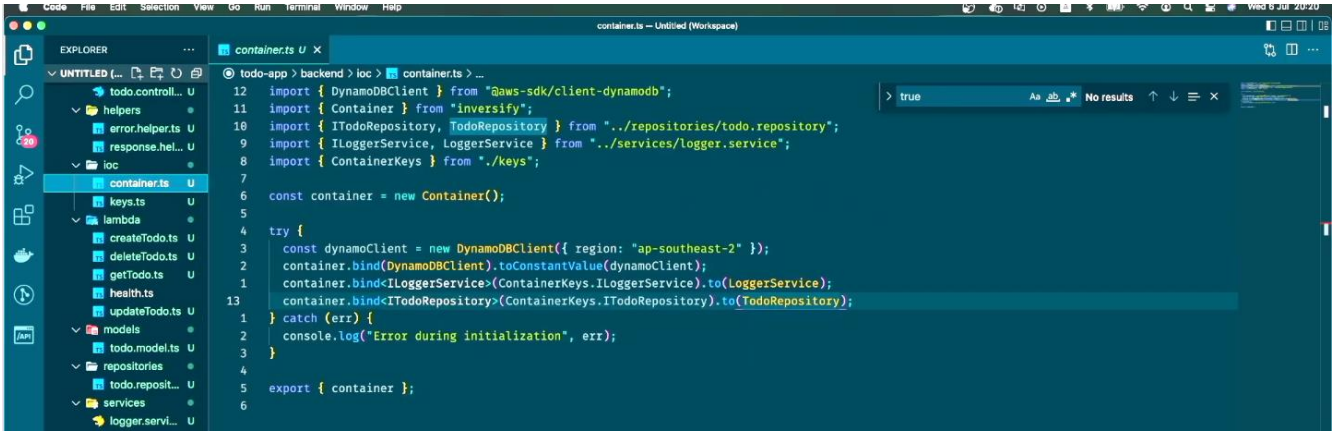

todo.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
contracts
todo.contrac... U
controllers
todo.controll... U
helpers
error.helper.ts U
response.hel... U
ioc
container.ts U
keys.ts U
lambda
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.reposit... U
services
logger.servi... U
todo.servi... 3, U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
OUTLINE
TIMELINE
todo.service.ts 3, U
todo-app > backend > services > todo.service.ts > TodoService > getTodo
true
No results
(property) QueryOutput.Count?: number | undefined
Object is possibly 'undefined'. ts(2532)
View Problem No quick fixes available
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 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;
tutorial_4* 3 0 Git Graph -- NORMAL --
Ln 46, Col 24 Spaces: 4 UTF-8 LF {} TypeScript Go Live Prettier

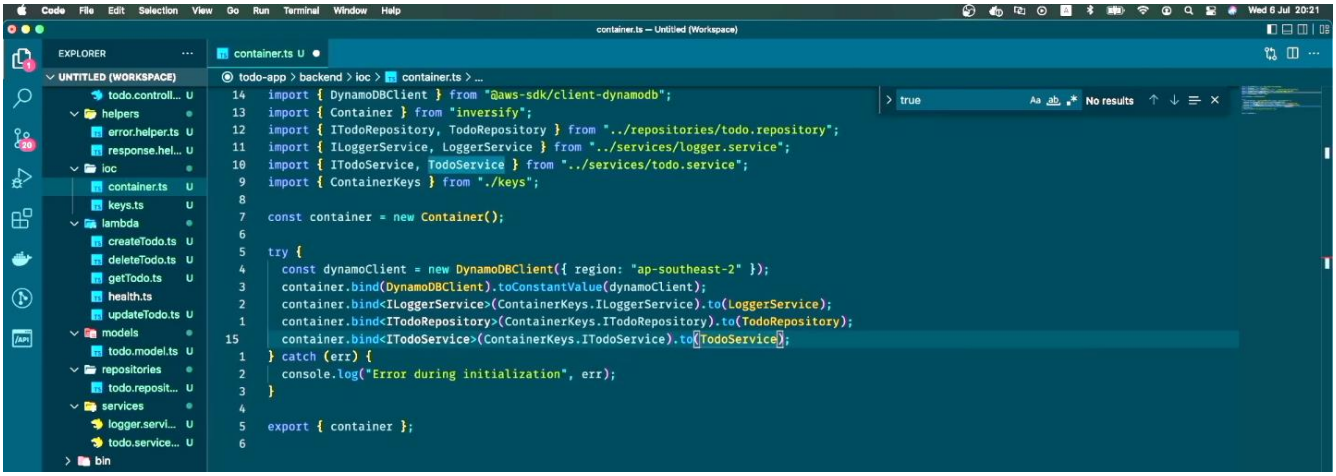

tsconfig.json — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo.service.ts 3, U
tsconfig.json
todo-app > tsconfig.json > {} compilerOptions > strictNullChecks
true 3 of 9
{
"compilerOptions": {
"target": "ES2018",
"module": "commonjs",
"lib": ["es2018"],
"declaration": true,
"strict": true,
"noImplicitAny": 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"]
}


todo.service.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo.service.ts U
tsconfig.json M
todo-app > backend > services > todo.service.ts > TodoService > updateTodo
true No results
async getTodo(): Promise<ITodo[] | []> {
this.logger.log("getTodo() called");
const response = await this.repo.getTodo();
let result: any = [];
if (response.Count > 0) {
result = response.Items.map((item) => this.processDynamoItem(item));
}
this.logger.log("dynamo response", { response }, this.constructor.name);
return result;
}
async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
}
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;
}
}

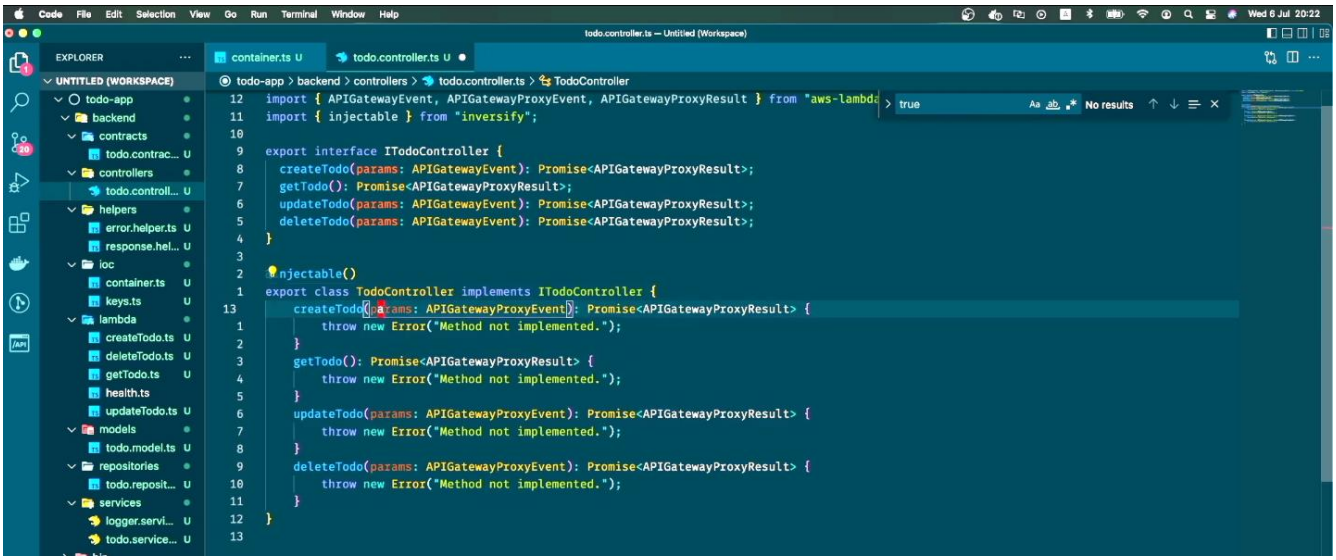

todo.service.ts — Untitled (Workspace)
todo.service.ts U
tsconfig.json M
todo-app > backend > services > todo.service.ts > TodoService > updateTodo
EXPLORER
UNTITLED (WORKSPACE)
createTodo.ts U
deleteTodo.ts U
getTodo.ts U
health.ts
updateTodo.ts U
models
todo.model.ts U
repositories
todo.reposit... U
services
logger.servi... U
todo.service... U
bin
cdk.out
lib
ApiGateway.ts
Cognito.ts
Dynamo.ts U
Lambda.ts M
todo-app-sta... M
node_modules
postman
test
.gitignore
.npmignore
.prettierrc.js
cdk.json
jest.config.js
package-lock.j... M
package.json M
README.md
tsconfig.json M
OUTLINE
true
No results
return result;
}
async updateTodo(params: Contracts.UpdateTodo): Promise<ITodo> {
this.logger.log("updateTodo() called", { params }, this.constructor.name);
const { status, id } = params;
if (!this.isStatusValid(status)) throw new Error("Invalid status");
const response = await this.repo.updateTodo(status, id);
this.logger.log("dynamo response", { response }, this.constructor.name);
if (response.$metadata.httpStatusCode == 200) {
const updatedTodo = await this.repo.getTodoById(id);
this.logger.log("updatedTodo", updatedTodo);
return this.processDynamoItem(updatedTodo.Items[0]);
}
return null;
}
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;
}
}

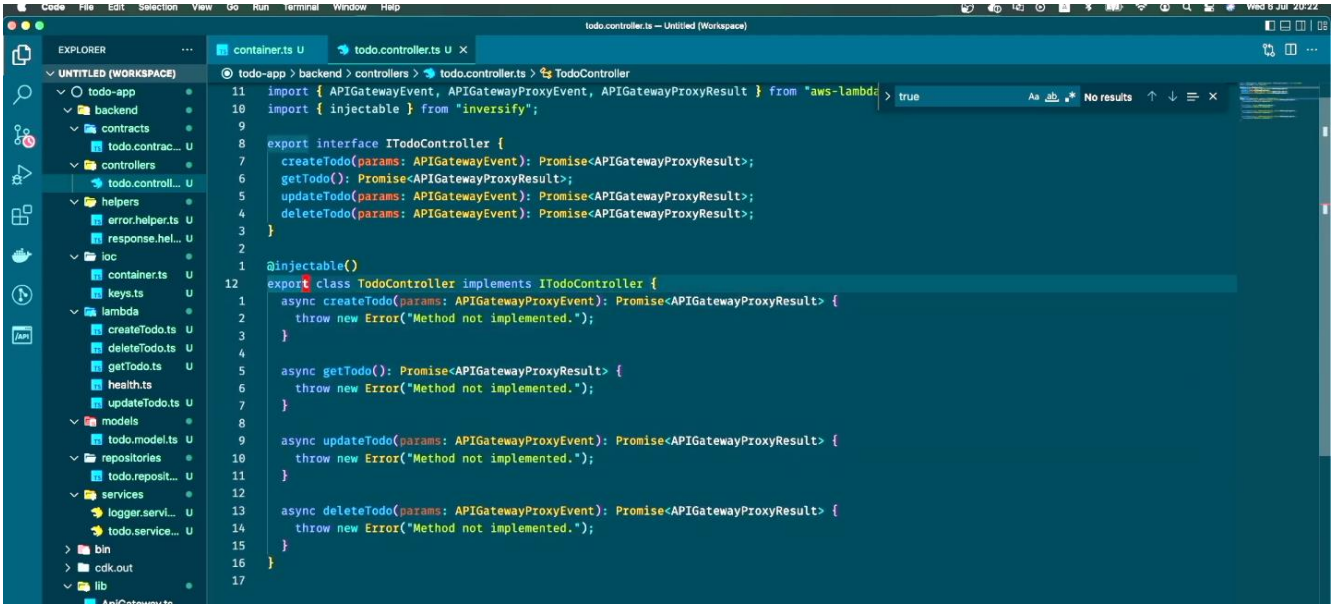

todo.service.ts — Untitled (Workspace)
todo.service.ts U
tsconfig.json M
todo-app > backend > services > todo.service.ts > TodoService > isStatusValid
EXPLORER
UNTITLED (WORKSPACE)
OUTLINE
true
No results
const { status, id } = params;
if (!this.isStatusValid(status)) throw new Error("Invalid status");
const response = await this.repo.updateTodo(status, id);
this.logger.log("dynamo response", { response }, this.constructor.name);
if (response.$metadata.httpStatusCode == 200) {
const updatedTodo = await this.repo.getTodoById(id);
this.logger.log("updatedTodo", updatedTodo);
return this.processDynamoItem(updatedTodo.Items[0]);
}
return null;
}
async deleteTodo(params: Contracts.DeleteTodo): Promise<void> {
this.logger.log("deleteTodo() called", { params }, this.constructor.name);
const { id } = params;
const response = await this.repo.deleteTodo(id);
this.logger.log("dynamo response", { response }, this.constructor.name);
return;
}
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;
}
}

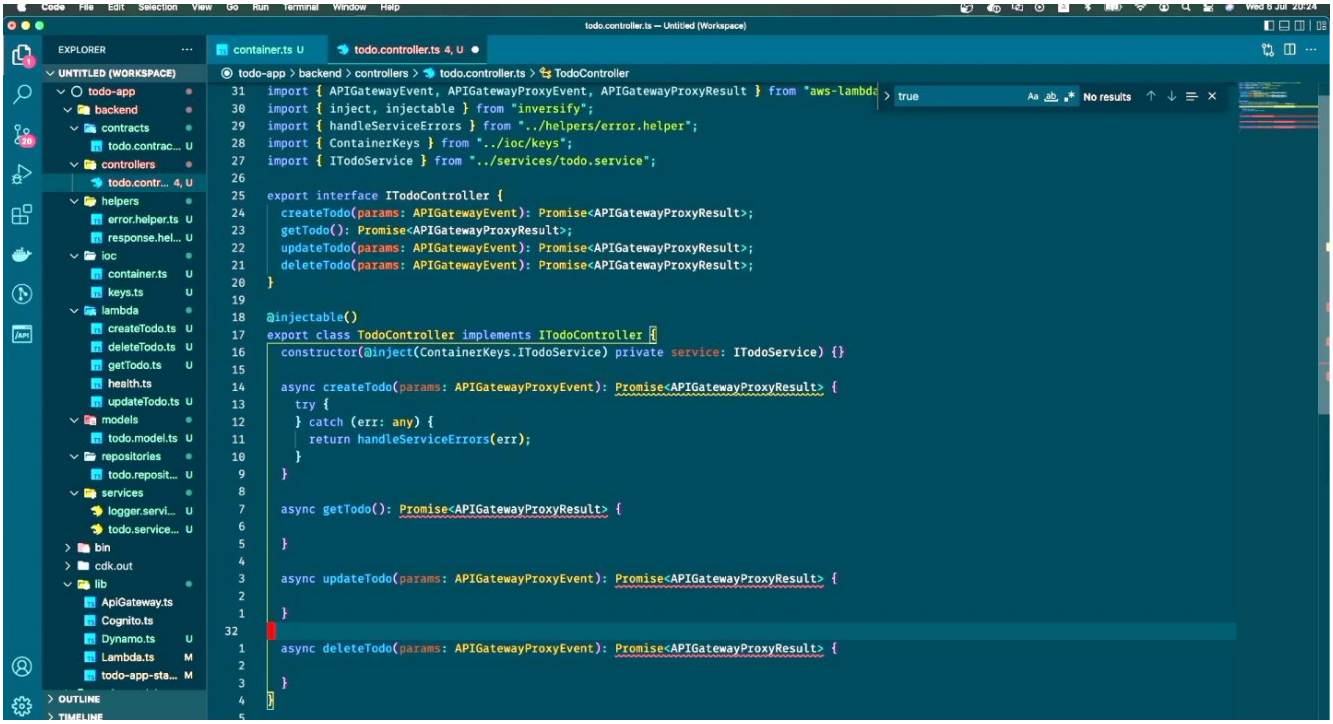

container.ts — Untitled (Workspace)
container.ts U
todo-app > backend > ioc > container.ts > ...
EXPLORER
helpers
error.helper.ts U
response.hel... U
ioc
container.ts U
keys.ts U
lambda
true
No results
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { Container } from "inversify";
import { ITodoRepository, TodoRepository } from "../repositories/todo.repository";
import { ILoggerService, LoggerService } from "../services/logger.service";
import { 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);
} catch (err) {
console.log("Error during initialization", err);
}
export { container };

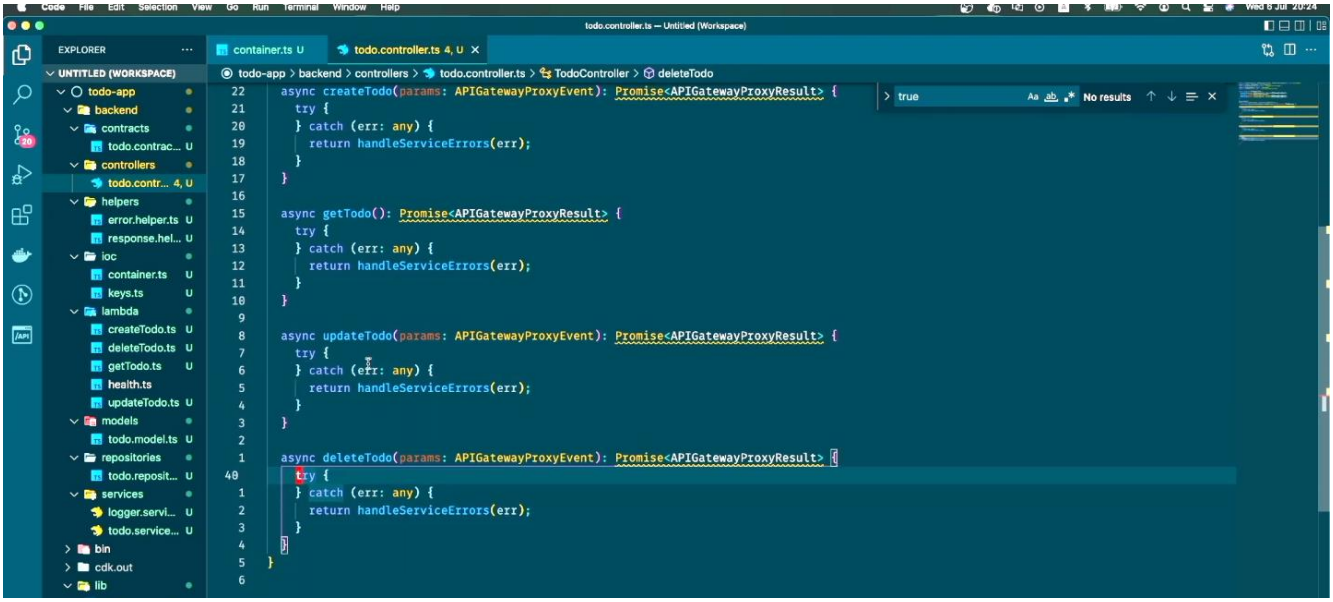

container.ts — Untitled (Workspace)
container.ts U
todo-app > backend > ioc > container.ts > ...
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { Container } from "inversify";
import { ITodoRepository, TodoRepository } from "../repositories/todo.repository";
import { ILoggerService, 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<ILoggerService>(ContainerKeys.ILoggerService).to(LoggerService);
container.bind<ITodoRepository>(ContainerKeys.ITodoRepository).to(TodoRepository);
container.bind<ITodoService>(ContainerKeys.ITodoService).to(TodoService);
} catch (err) {
console.log("Error during initialization", err);
}
export { container };
true
No results

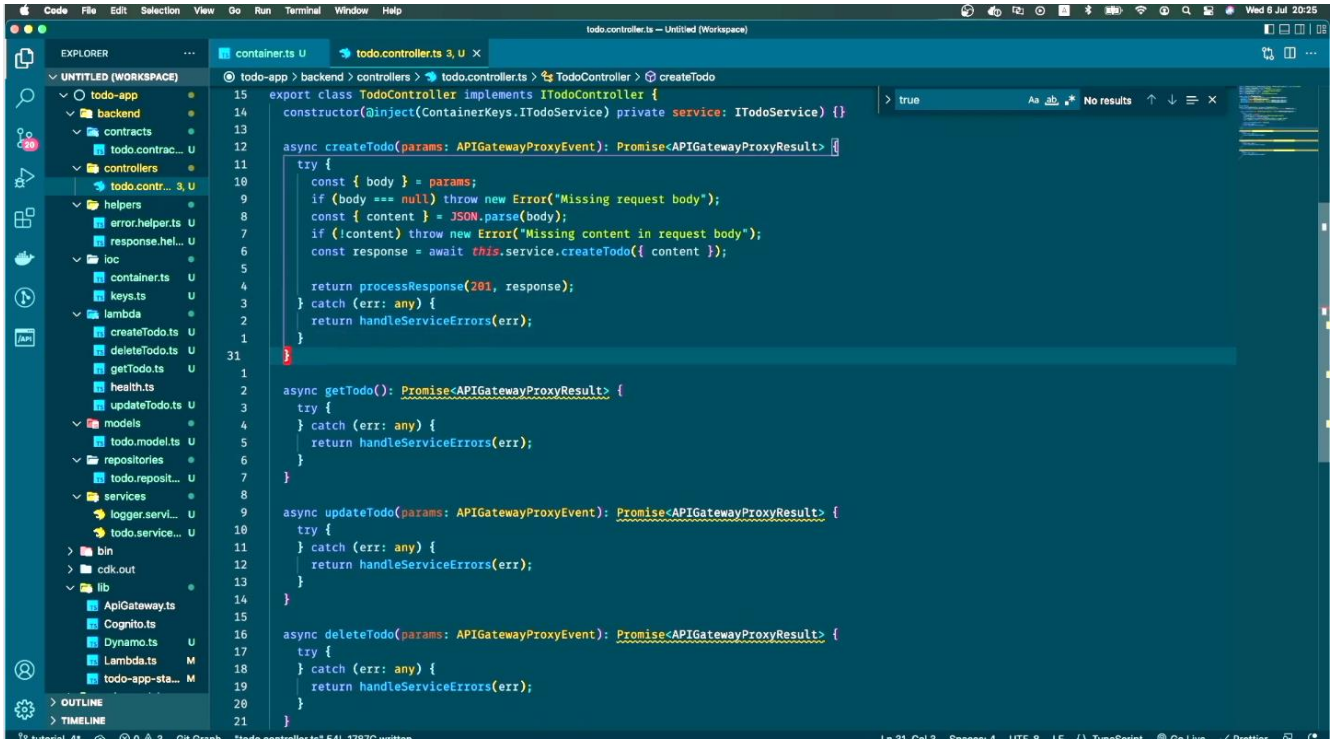

todo.controller.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
todo-app > backend > controllers > todo.controller.ts > ITodoController
export interface ITodoController {
}
true
No results

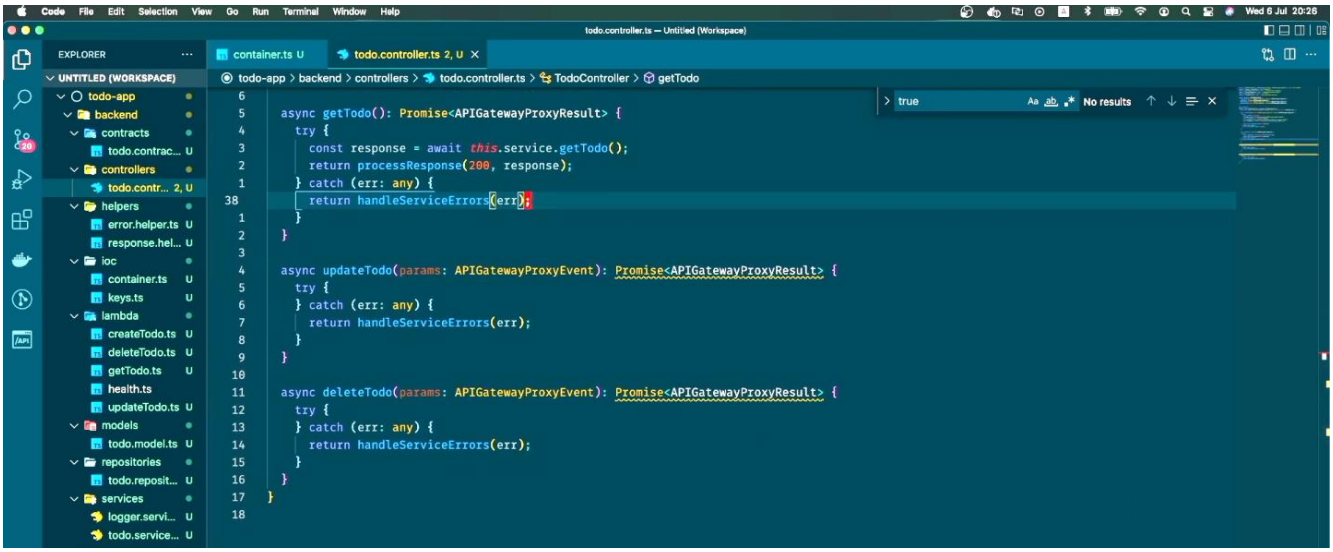

todo.controller.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
todo-app > backend > controllers > todo.controller.ts > TodoController
import { APIGatewayEvent, APIGatewayProxyEvent, APIGatewayProxyResult } from "aws-lambda
import { injectable } from "inversify";
export interface ITodoController {
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.");
}
}
true
No results

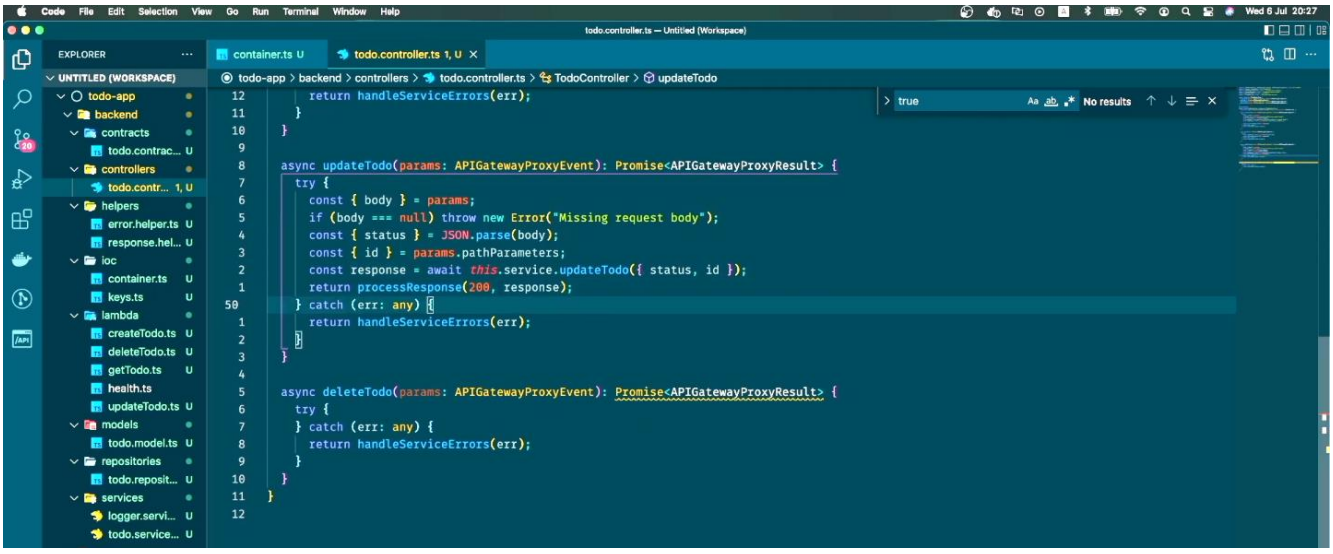

todo.controller.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
todo-app > backend > controllers > todo.controller.ts > TodoController
import { APIGatewayEvent, APIGatewayProxyEvent, APIGatewayProxyResult } from "aws-lambda
import { injectable } from "inversify";
export interface ITodoController {
createTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
getTodo(): Promise<APIGatewayProxyResult>;
updateTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
deleteTodo(params: APIGatewayEvent): Promise<APIGatewayProxyResult>;
}
@injectable()
export class TodoController implements ITodoController {
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.");
}
}
true
No results

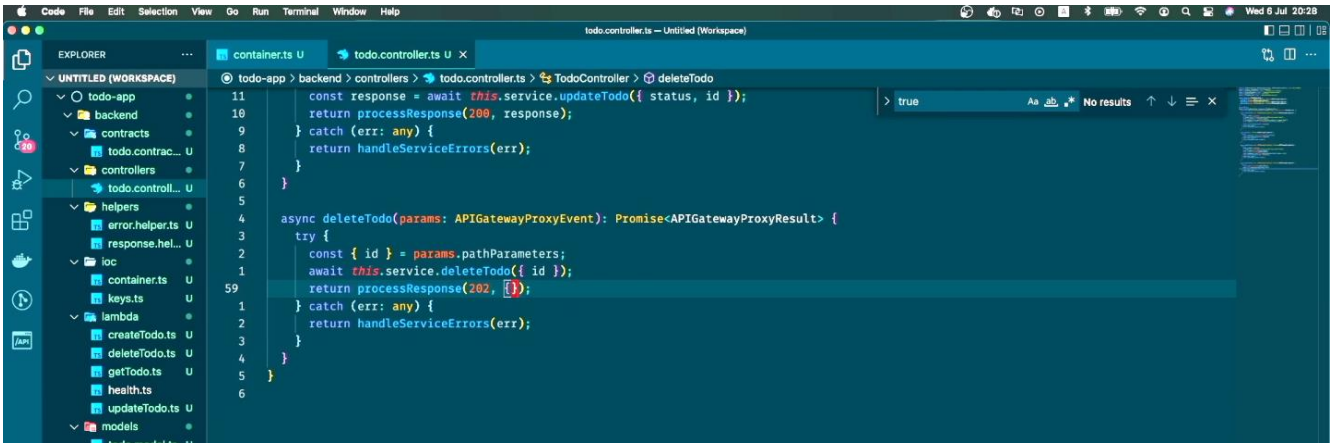

todo.controller.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
container.ts U
todo.controller.ts 4, U
todo-app > backend > controllers > todo.controller.ts > TodoController
import { APIGatewayEvent, APIGatewayProxyEvent, APIGatewayProxyResult } from "aws-lambda
import { inject, injectable } 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> {
}
async updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
}
async deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
}
}
true
No results
OUTLINE
TIMELINE

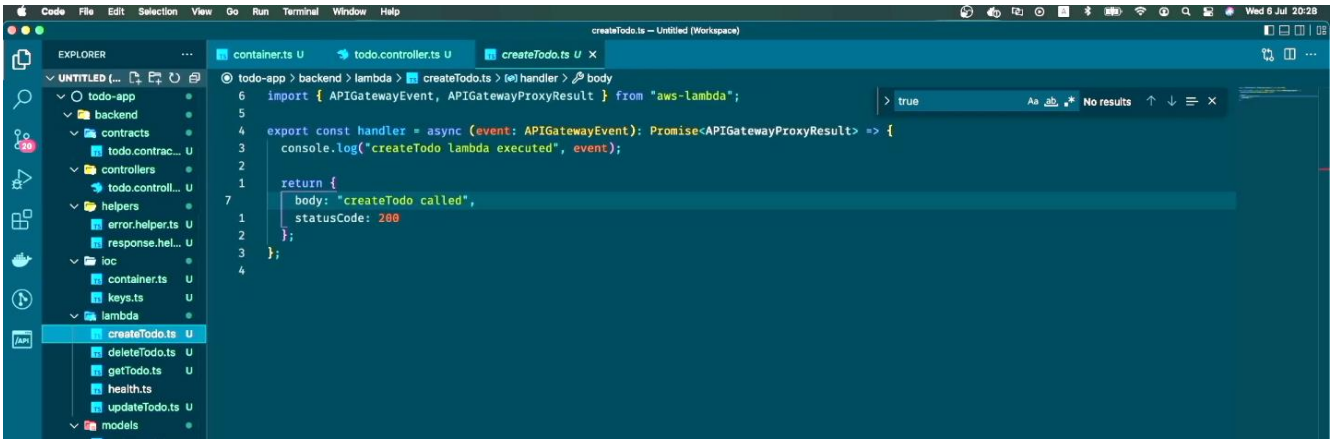

todo.controller.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
container.ts U
todo.controller.ts 4, U
todo-app > backend > controllers > todo.controller.ts > TodoController > deleteTodo
async createTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
try {
} catch (err: any) {
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);
}
}
}
true
No results

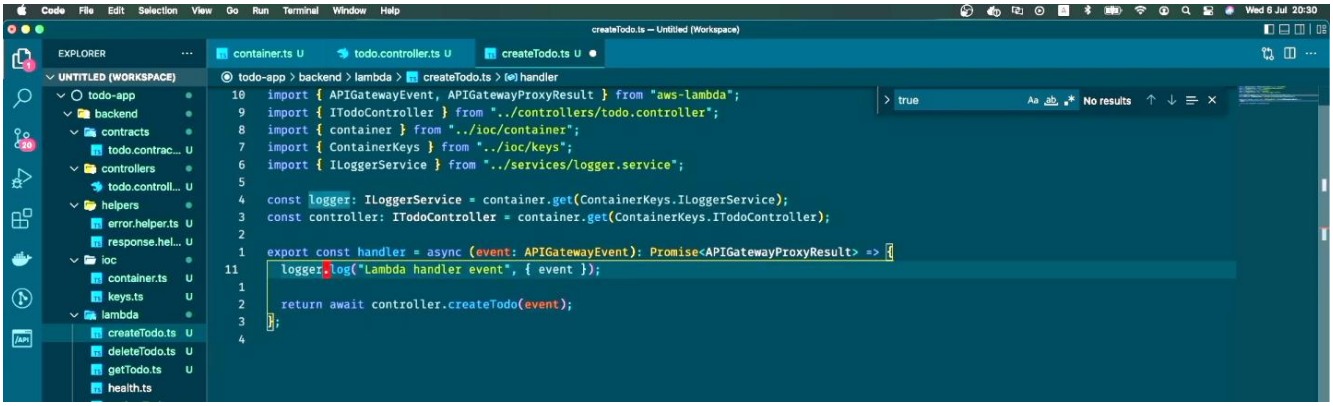

todo.controller.ts — Untitled (Workspace)
container.ts U
todo.controller.ts 3, U
todo-app > backend > controllers > todo.controller.ts > TodoController > createTodo
export class TodoController implements ITodoController {
constructor(@inject(ContainerKeys.ITodoService) private service: ITodoService) {}
async createTodo(params: APIGatewayProxyEvent): 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 {
} 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);
}
}
true
No results

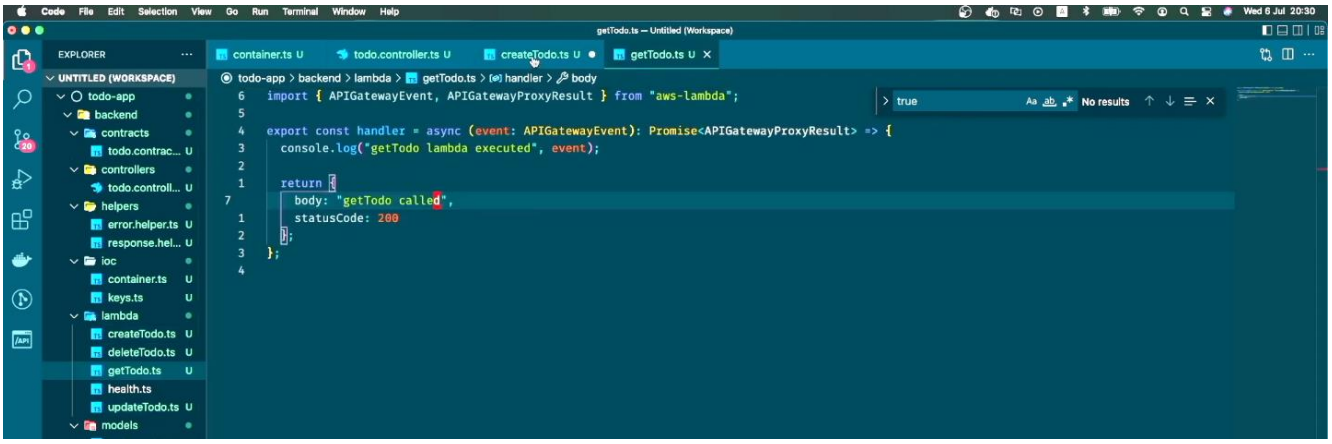

todo.controller.ts — Untitled (Workspace)
container.ts U
todo.controller.ts 2, U
todo-app > backend > controllers > todo.controller.ts > TodoController > getTodo
async getTodo(): Promise<APIGatewayProxyResult> {
try {
const response = await this.service.getTodo();
return processResponse(200, response);
} 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);
}
}
}
true
No results

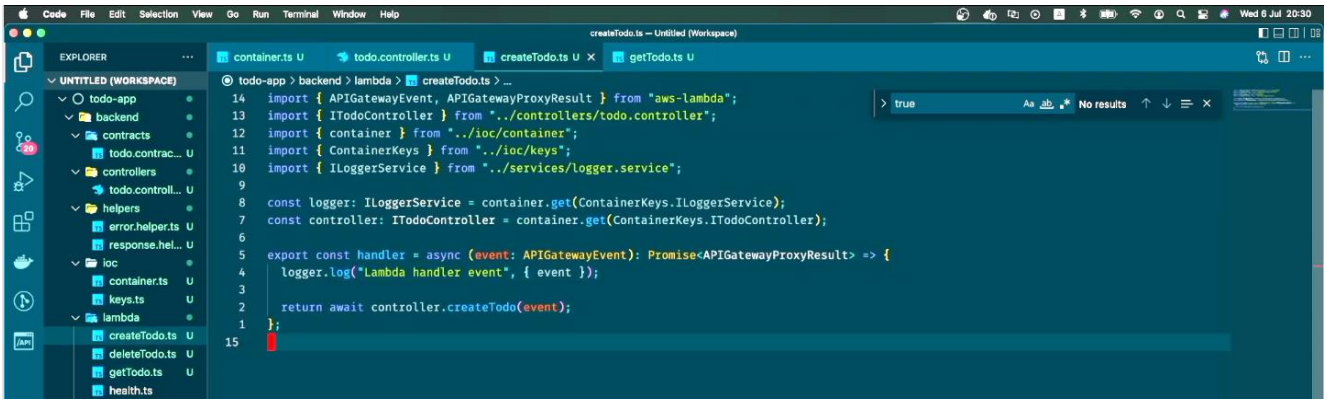

todo.controller.ts — Untitled (Workspace)
container.ts U
todo.controller.ts 1, U
todo-app > backend > controllers > todo.controller.ts > TodoController > updateTodo
async updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
try {
const { body } = params;
if (body === null) throw new Error("Missing request body");
const { status } = JSON.parse(body);
const { id } = params.pathParameters;
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 {
} catch (err: any) {
return handleServiceErrors(err);
}
}
}
true
No results

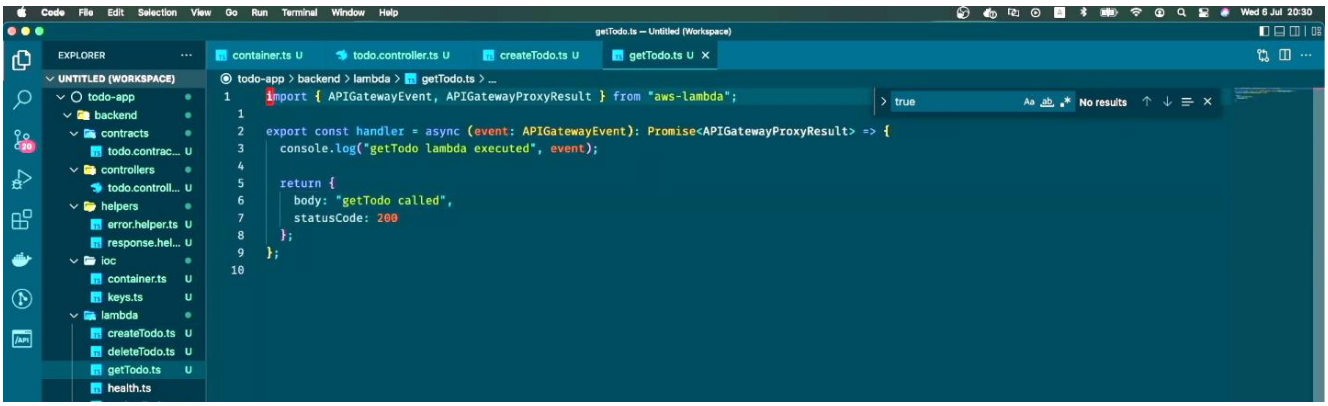

todo.controller.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
todo-app > backend > controllers > todo.controller.ts > TodoController > deleteTodo
async deleteTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
try {
const { id } = params.pathParameters;
await this.service.deleteTodo({ id });
return processResponse(202, {});
} catch (err: any) {
return handleServiceErrors(err);
}
}
}

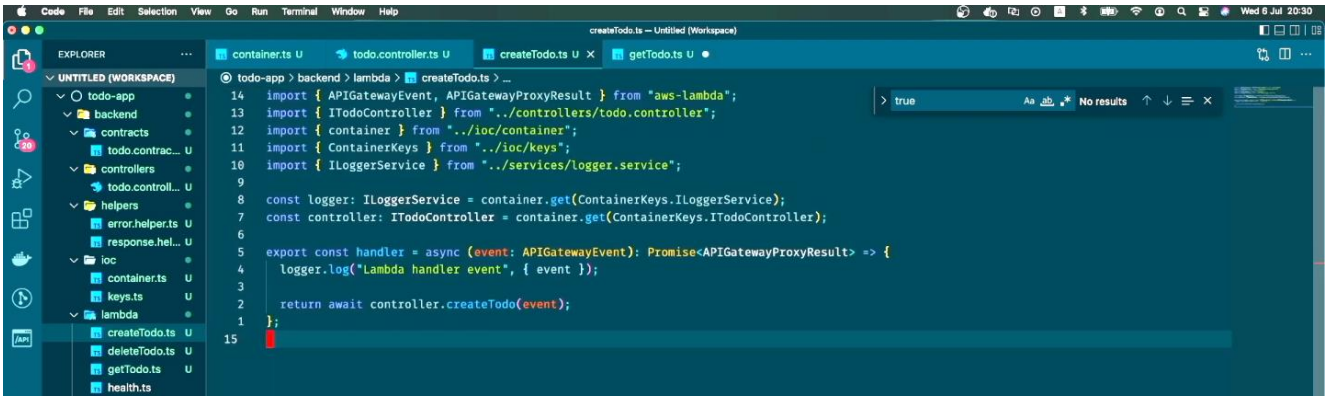

createTodo.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
createTodo.ts U
todo-app > backend > lambda > createTodo.ts > handler > body
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
console.log("createTodo lambda executed", event);
return {
body: "createTodo called",
statusCode: 200
};
};

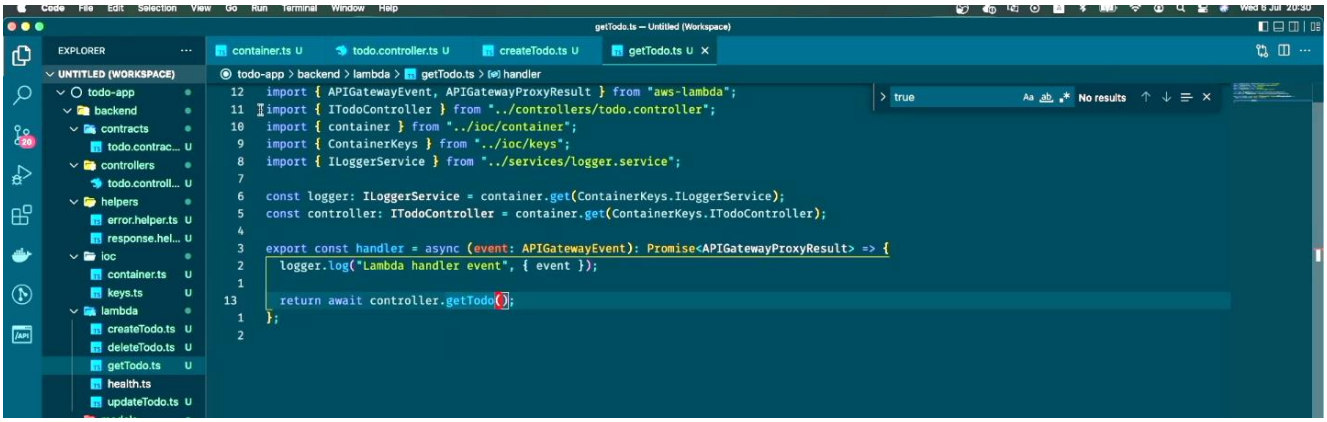

createTodo.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
createTodo.ts U
todo-app > backend > lambda > createTodo.ts > handler
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
import { ITodoController } from "../controllers/todo.controller";
import { container } from "../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);
};

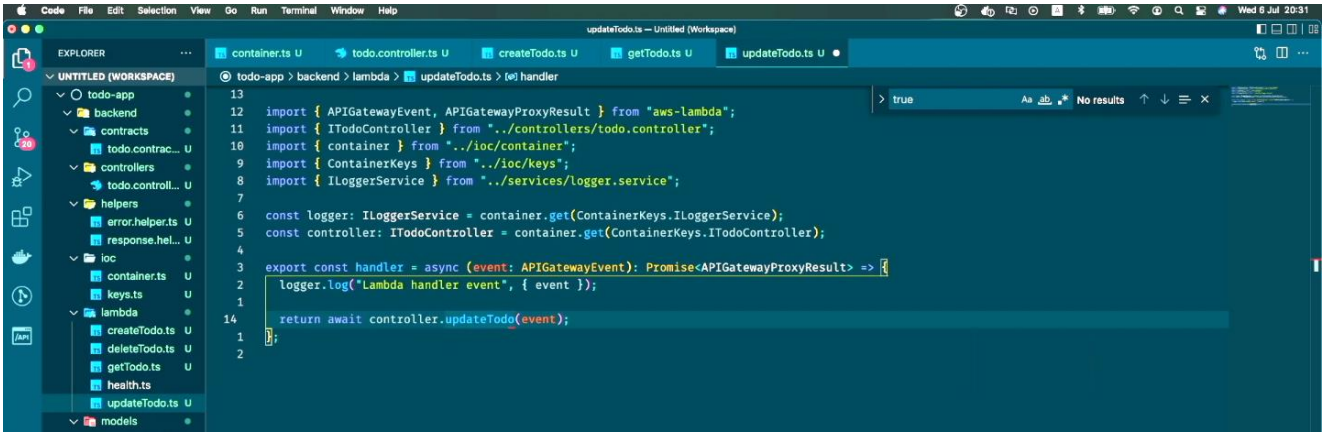

getTodo.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
createTodo.ts U
getTodo.ts U
todo-app > backend > lambda > getTodo.ts > handler > body
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
export const handler = async (event: APIGatewayEvent): Promise<APIGatewayProxyResult> => {
console.log("getTodo lambda executed", event);
return {
body: "getTodo called",
statusCode: 200
};
};

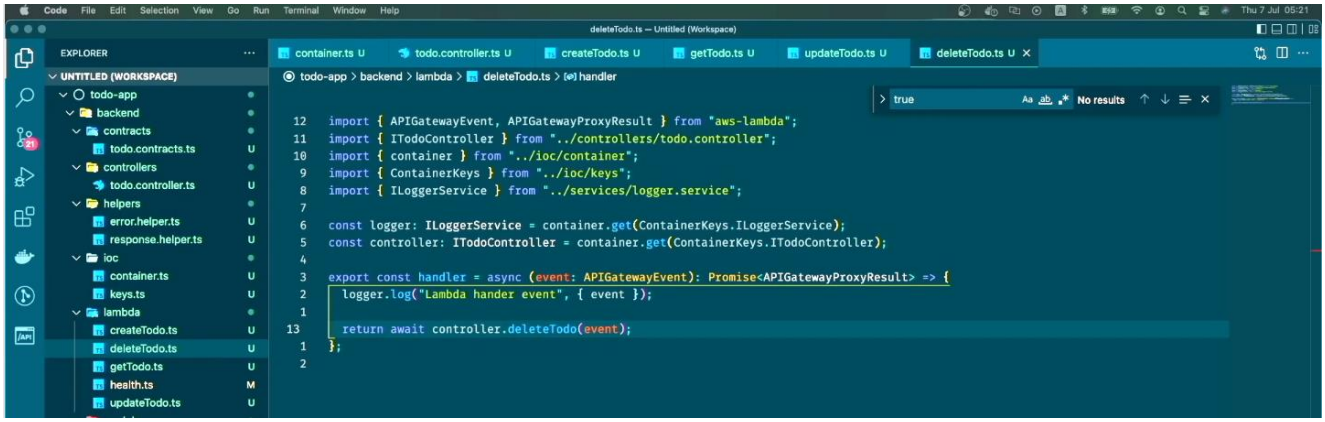

createTodo.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
createTodo.ts U
getTodo.ts U
todo-app > backend > lambda > createTodo.ts > ...
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
import { ITodoController } from "../controllers/todo.controller";
import { container } 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);
};
true
No results


getTodo.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
createTodo.ts U
getTodo.ts U
todo-app > backend > lambda > getTodo.ts > ...
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
};
};
true
No results

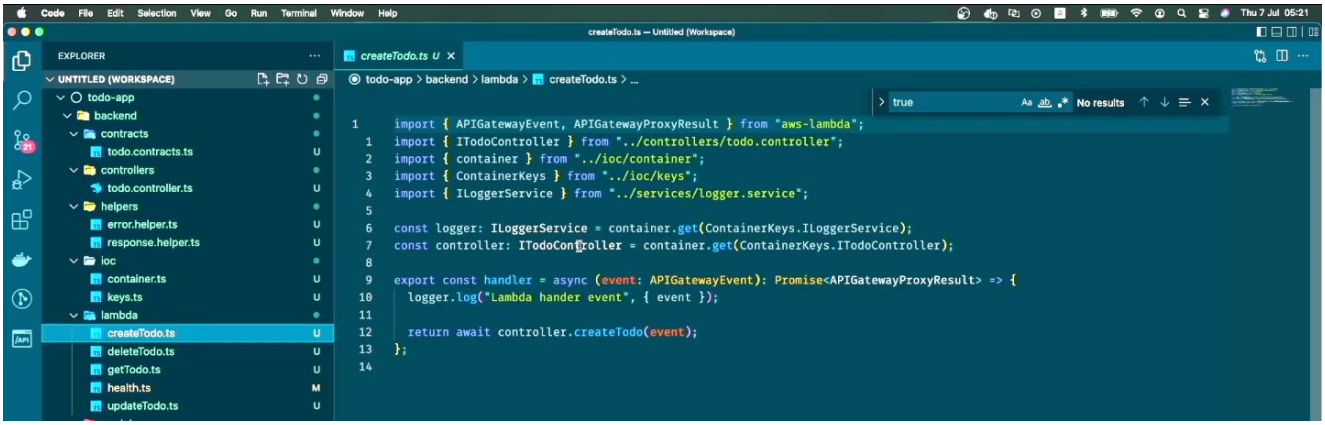

createTodo.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
createTodo.ts U
getTodo.ts U
todo-app > backend > lambda > createTodo.ts > ...
import { APIGatewayEvent, APIGatewayProxyResult } from "aws-lambda";
import { ITodoController } from "../controllers/todo.controller";
import { container } 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);
};
true
No results


getTodo.ts — Untitled (Workspace)
container.ts U
todo.controller.ts U
createTodo.ts U
getTodo.ts U
todo-app > backend > lambda > getTodo.ts > handler
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.getTodo();
};
true
No results

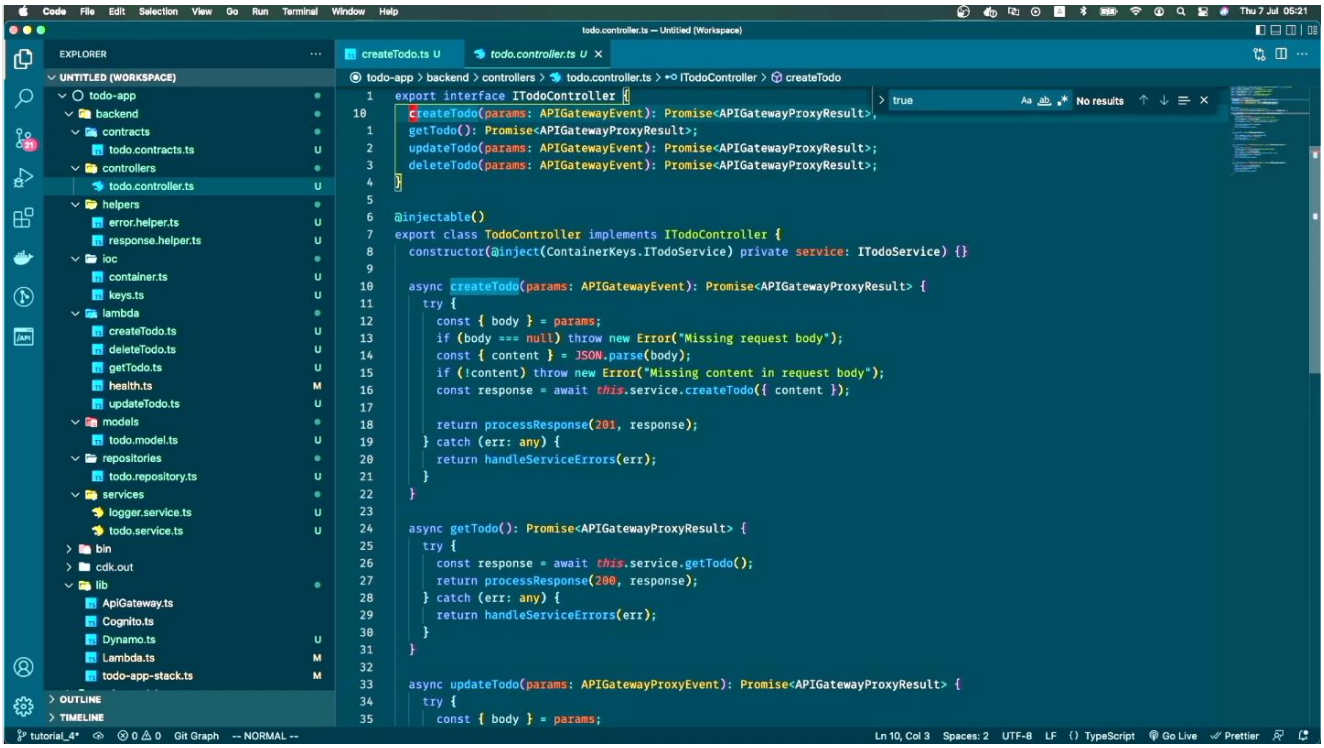




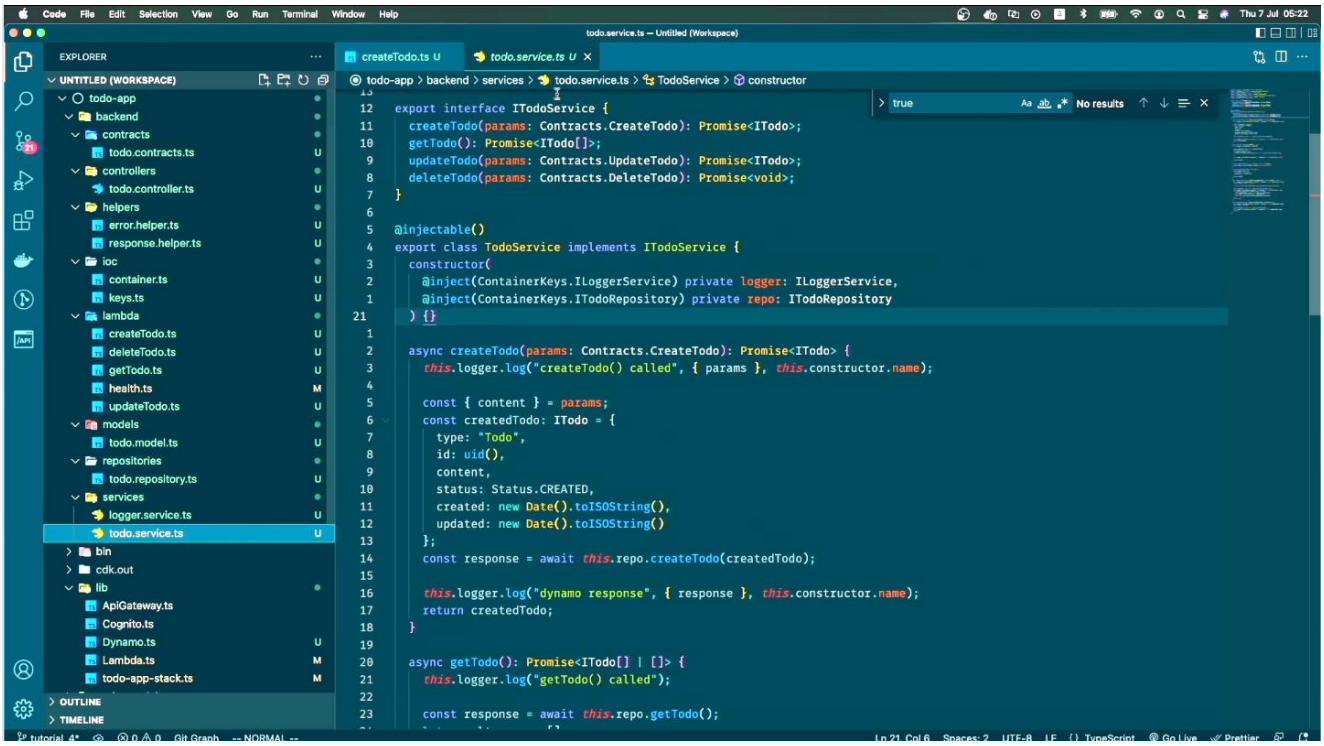
We have now completed implementing the code, let us see the application folder structure using separation of concerns that divides the code into different layers having different responsibilities



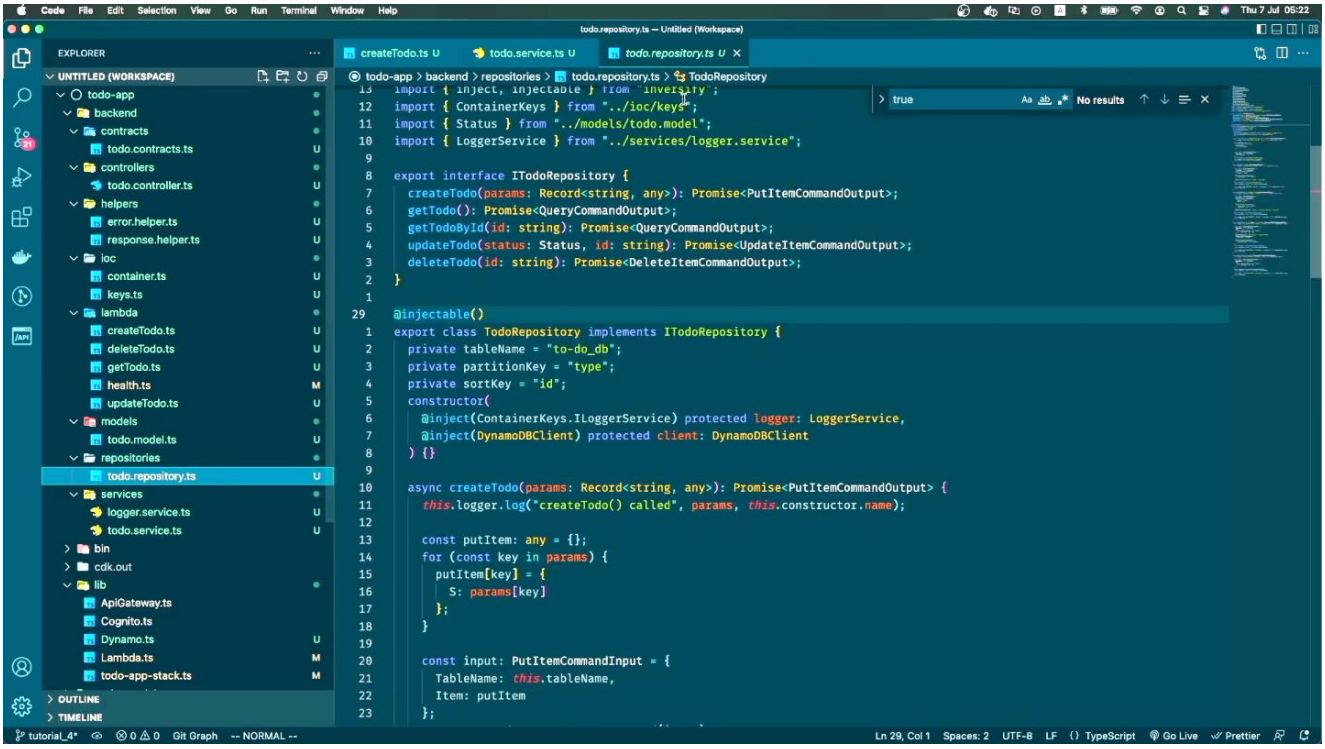
The lambda folder contains lambda functions as entry points into our application. Each lambda follows the concept of single responsibility for a single job. We invoke the controller from each lambda function



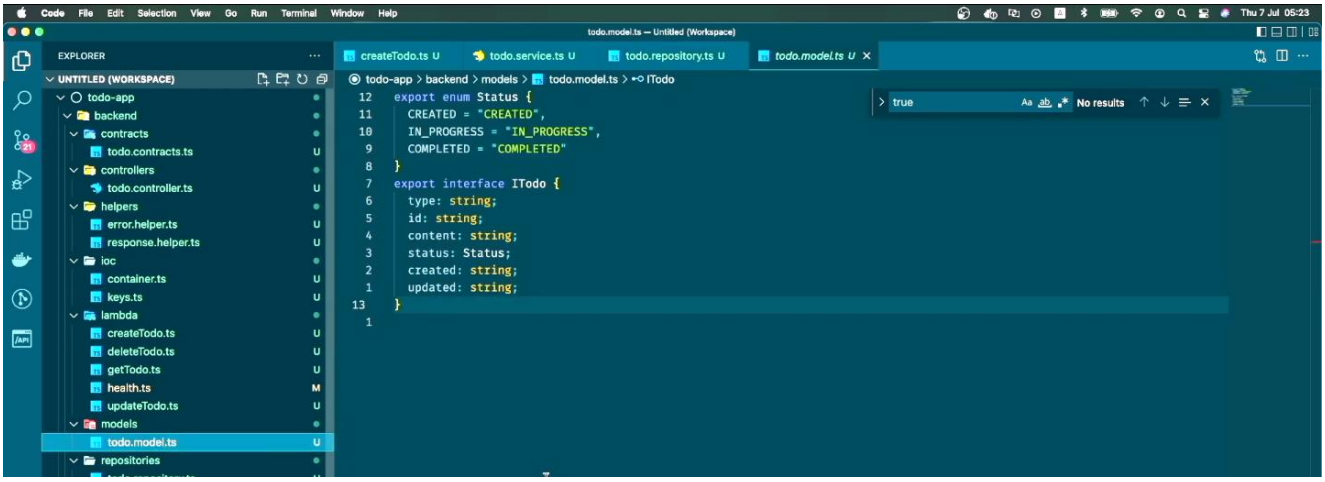
The controller contains our application functionalities, it sends back the response and handle any errors that occur



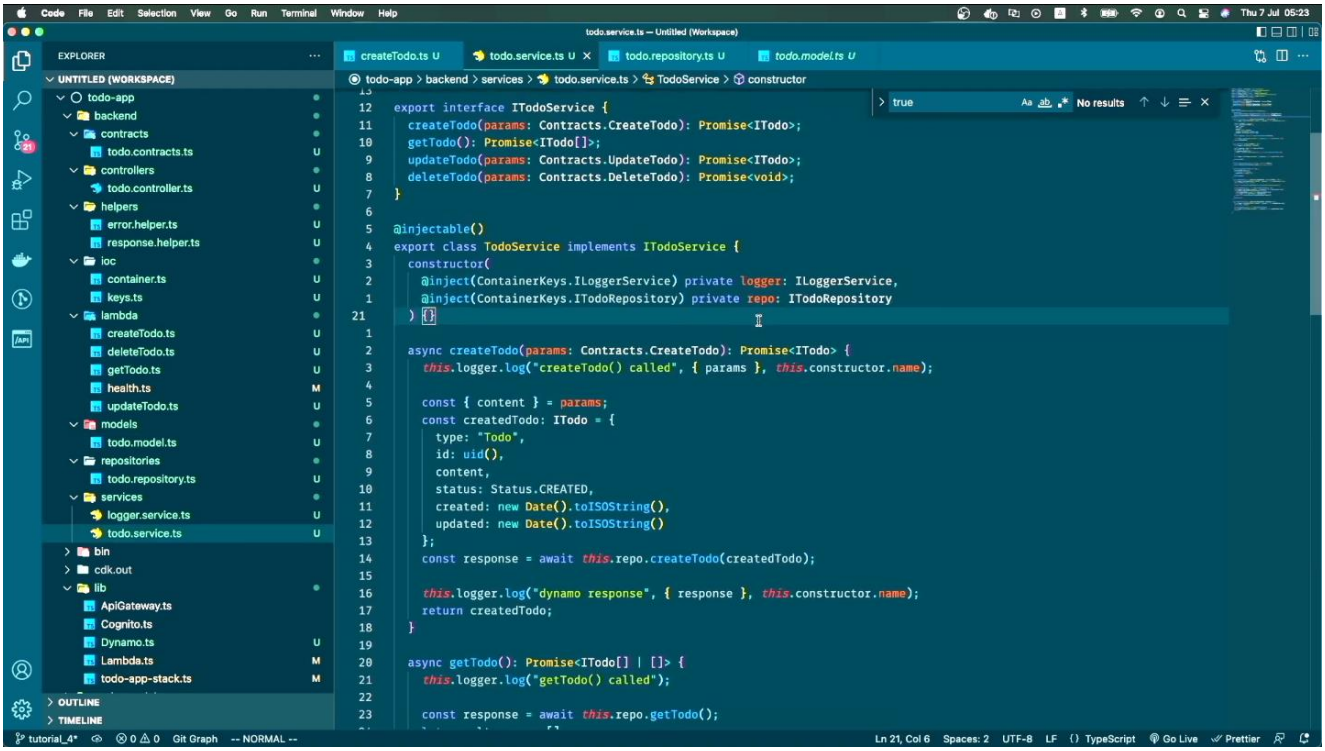
The controller uses the service layer that contains the main business logic



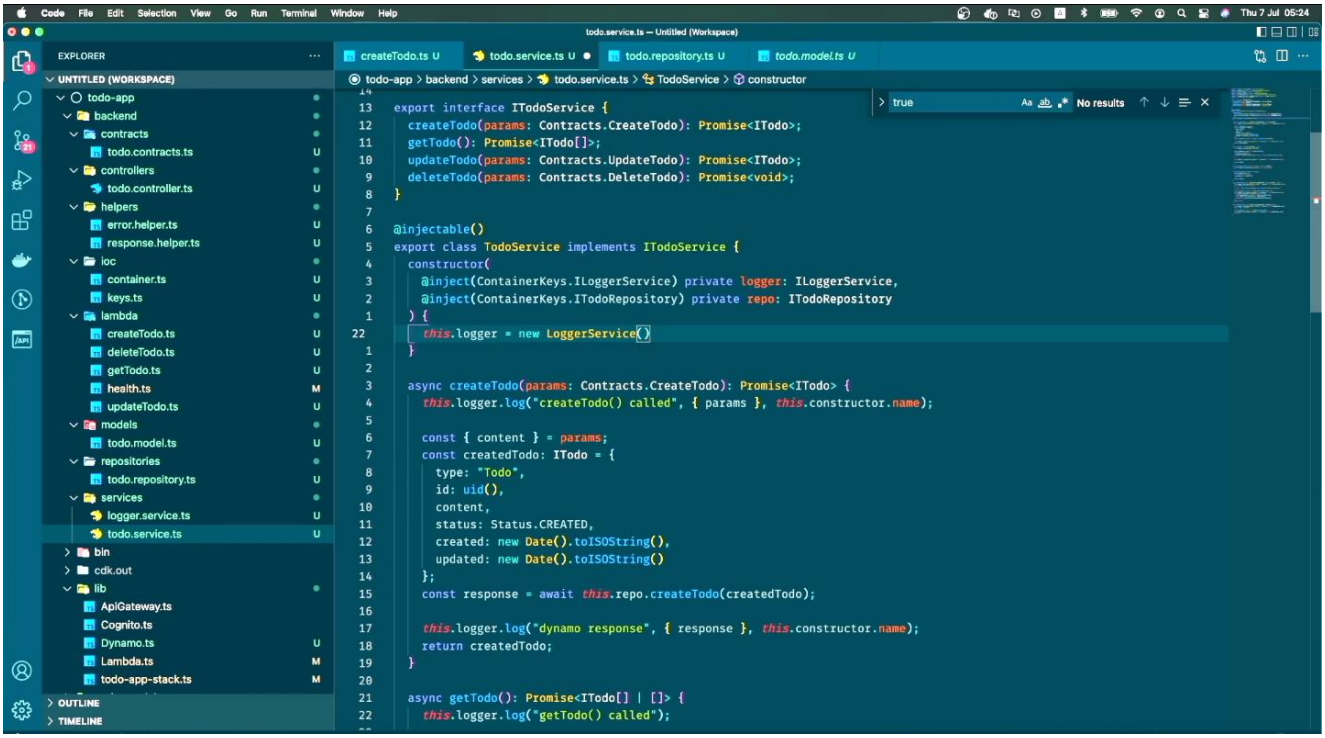
The repository layer contains any database related logic



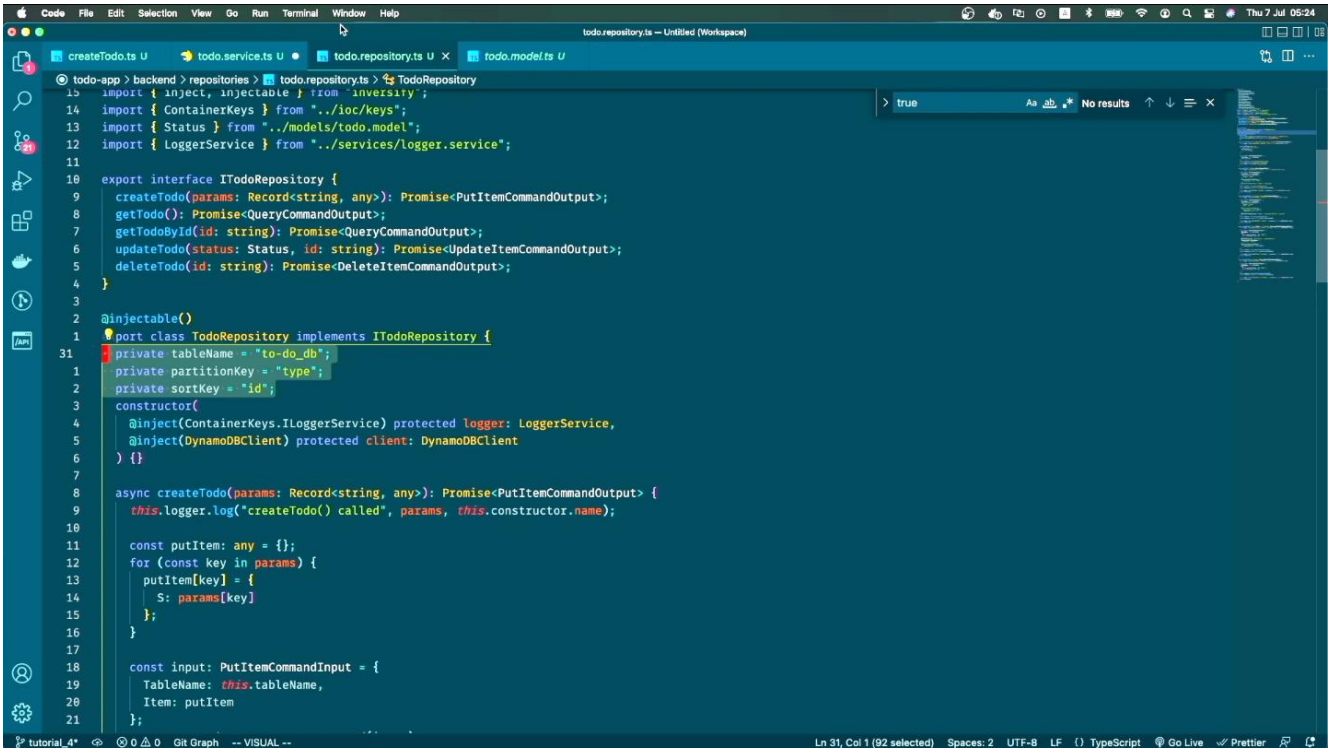
The model layer contains our business objects and data structures for our application

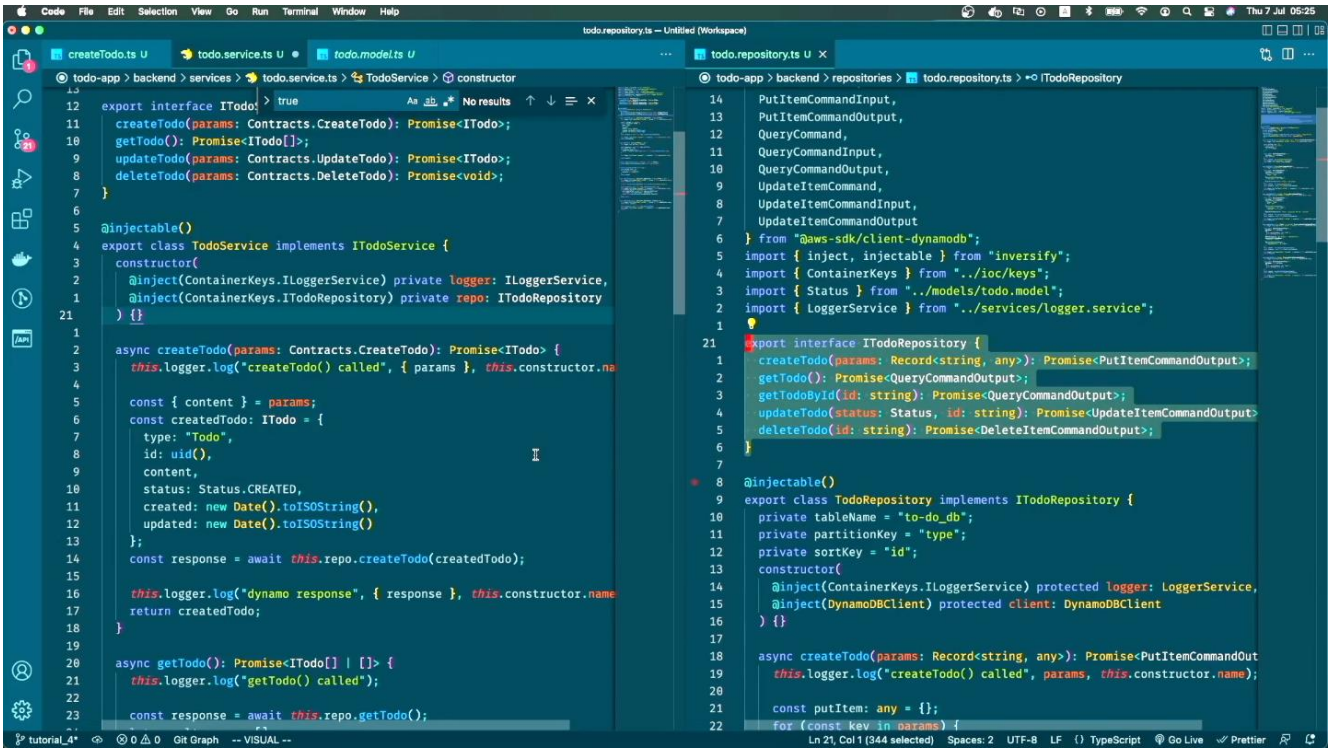


We also used the concept of inversion of control where we try to couple the dependency between processes by invoking the control flow. Above uses 2 dependencies for the Logger and Repository.

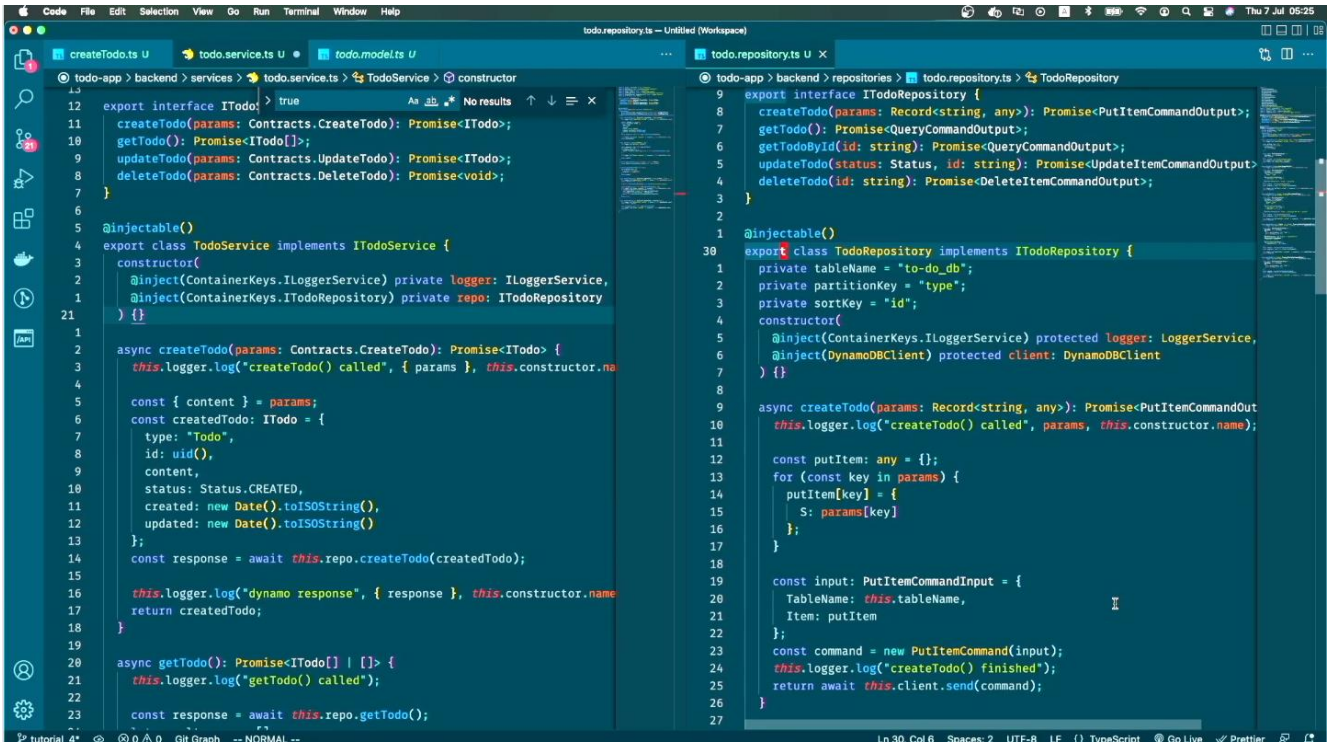


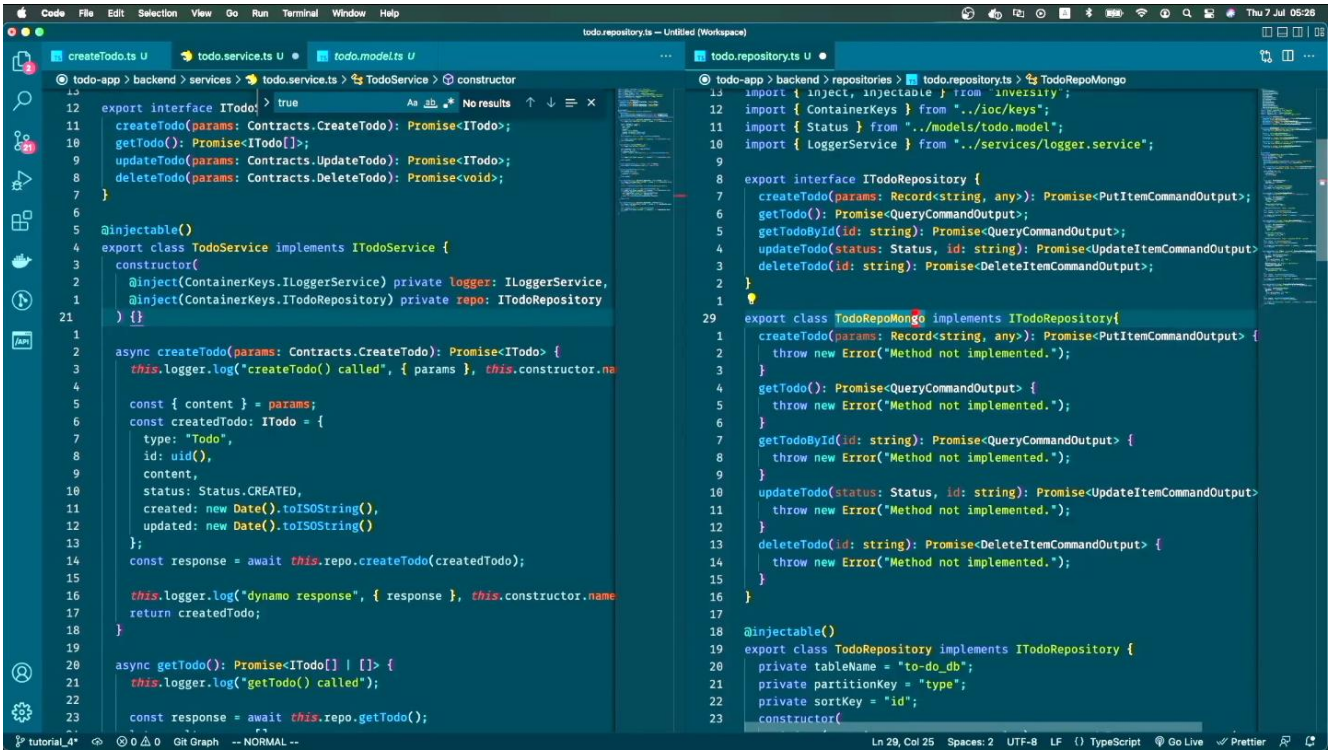
We can also do as above without IoC using the interfaces.



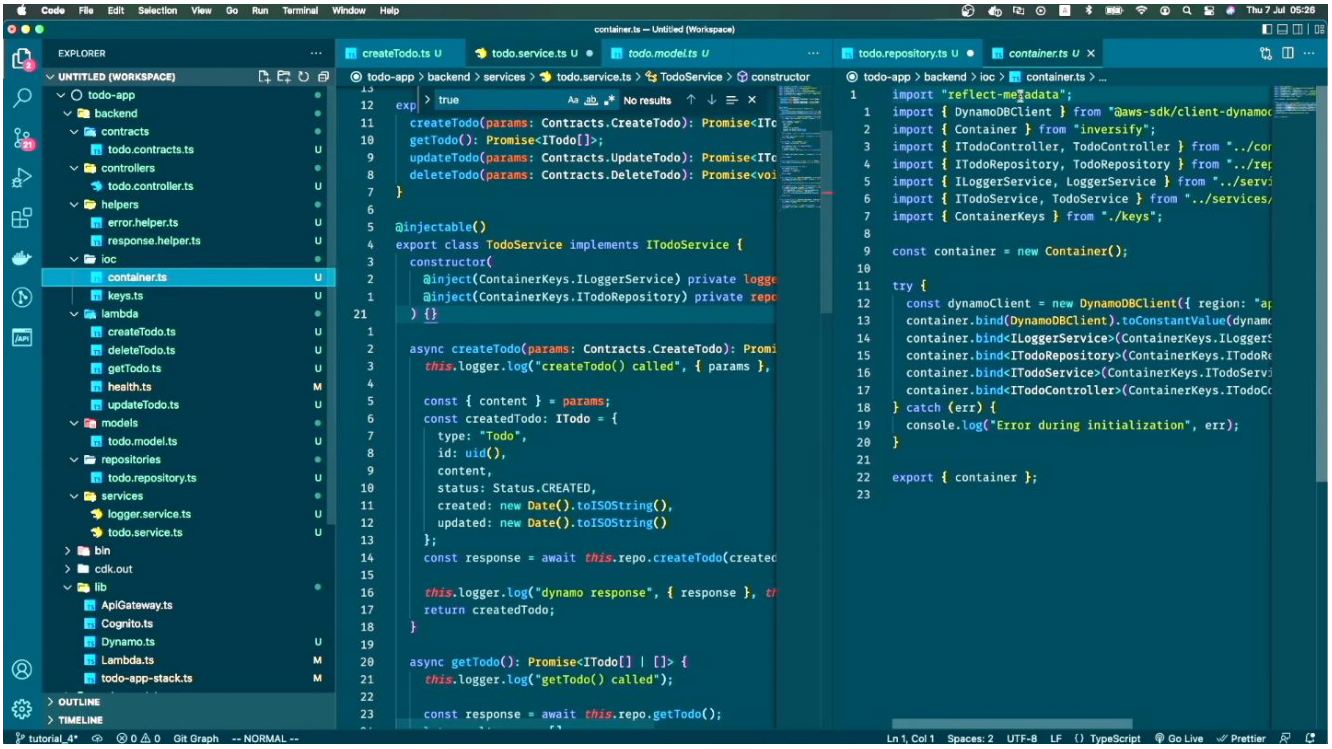


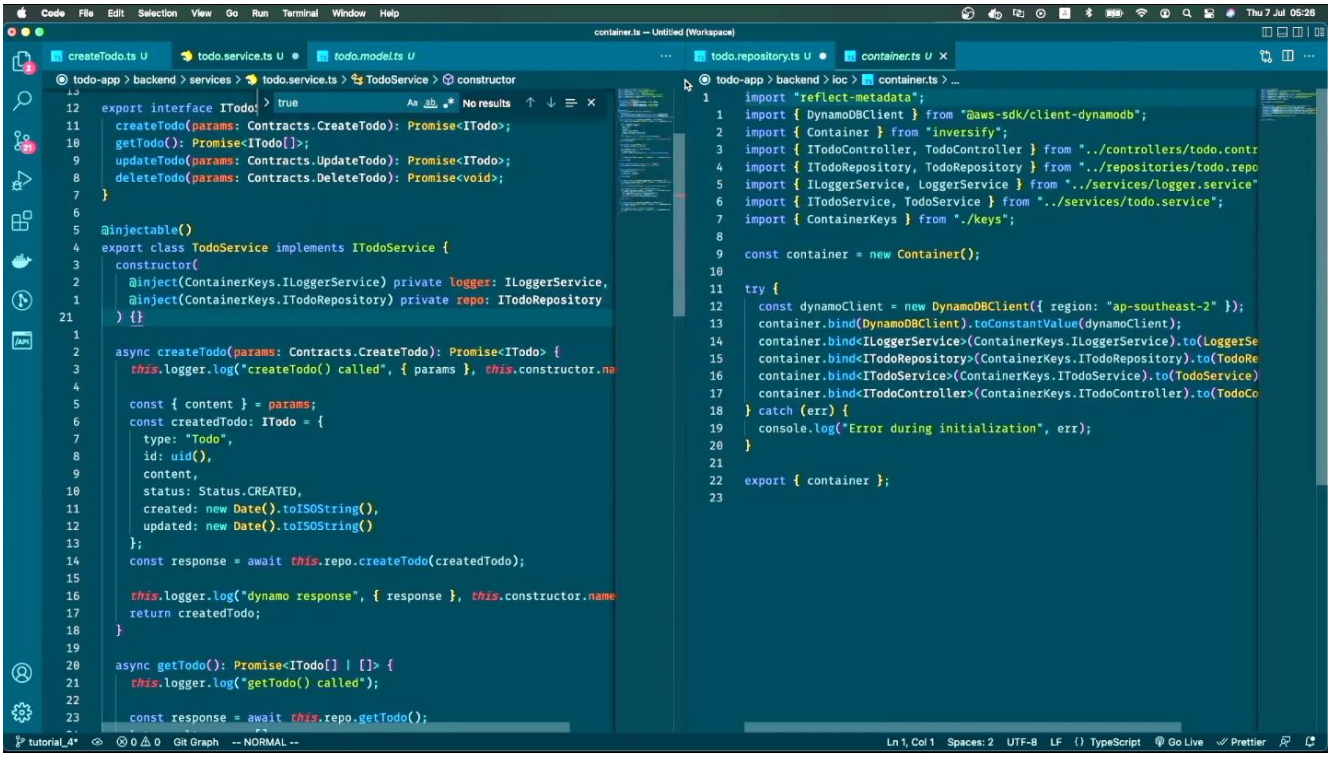
The TodoService doesn't know about the database but only knows about the **exported** Repository interface methods

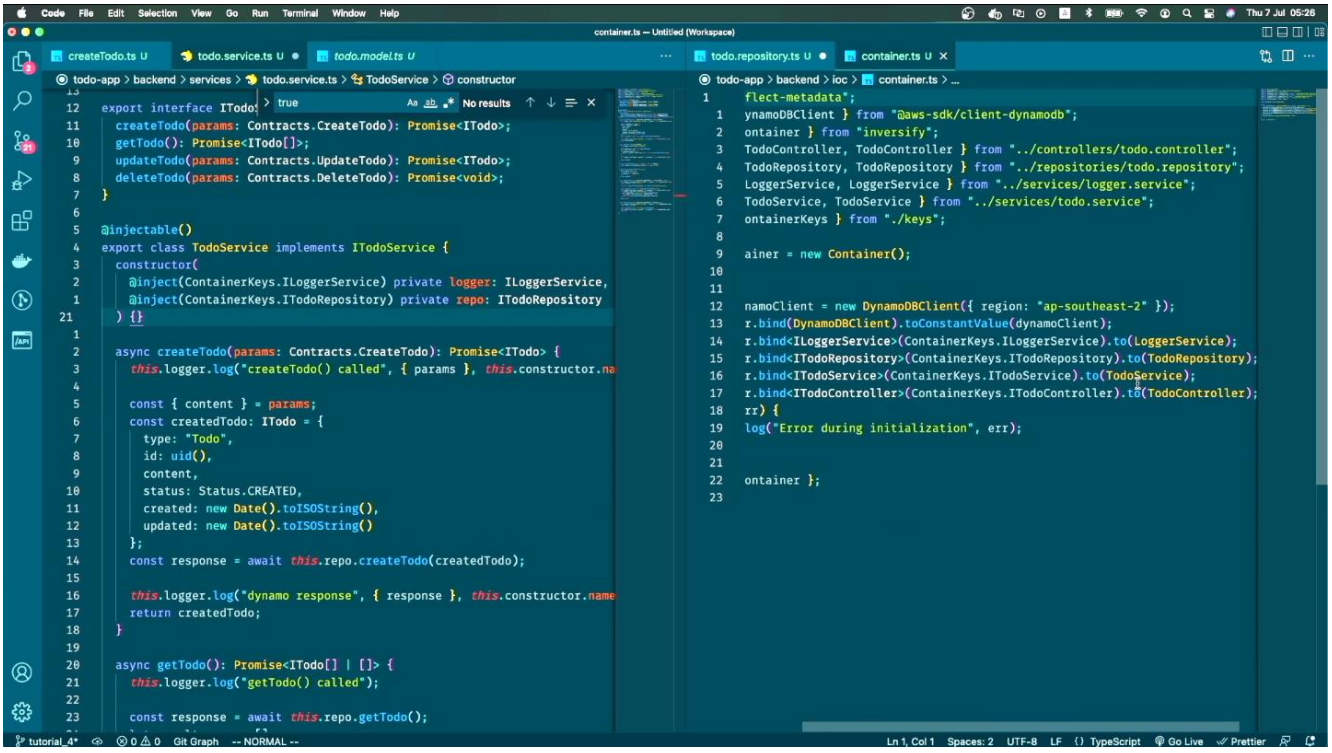


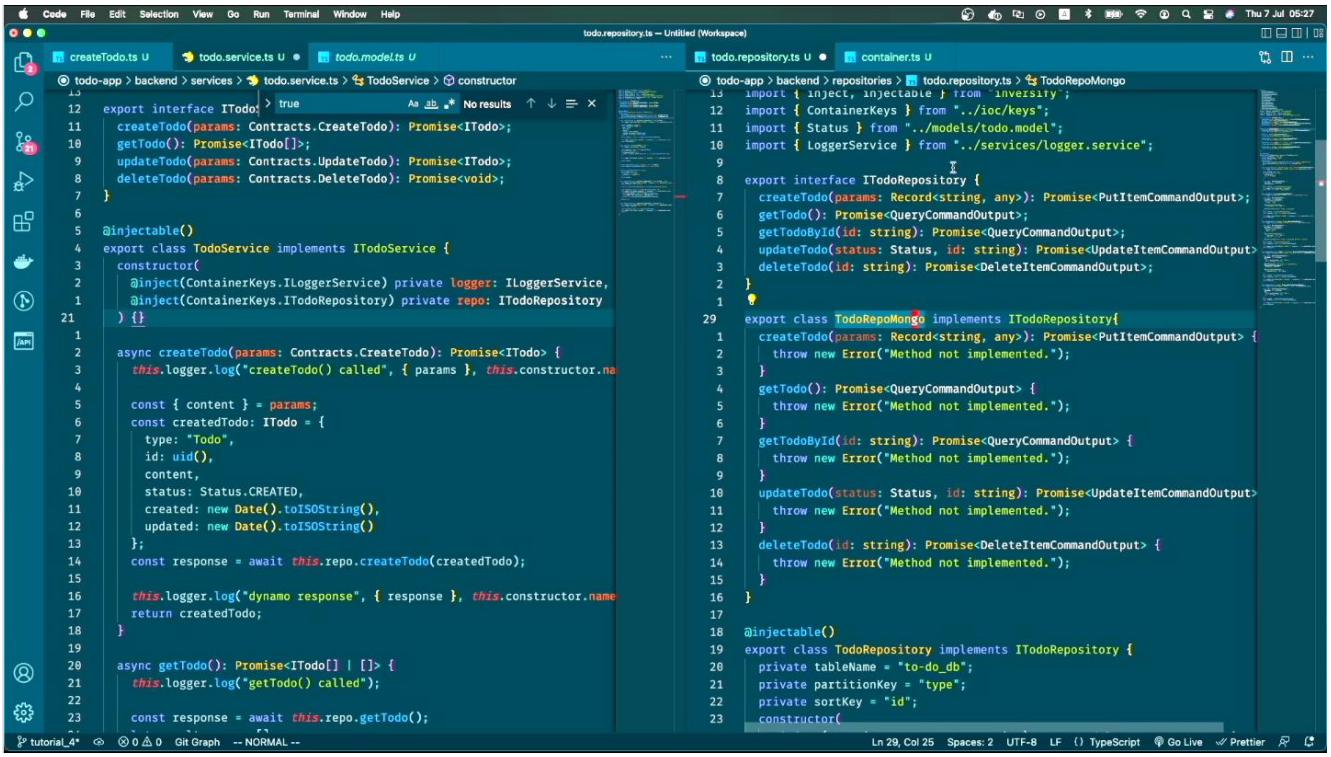


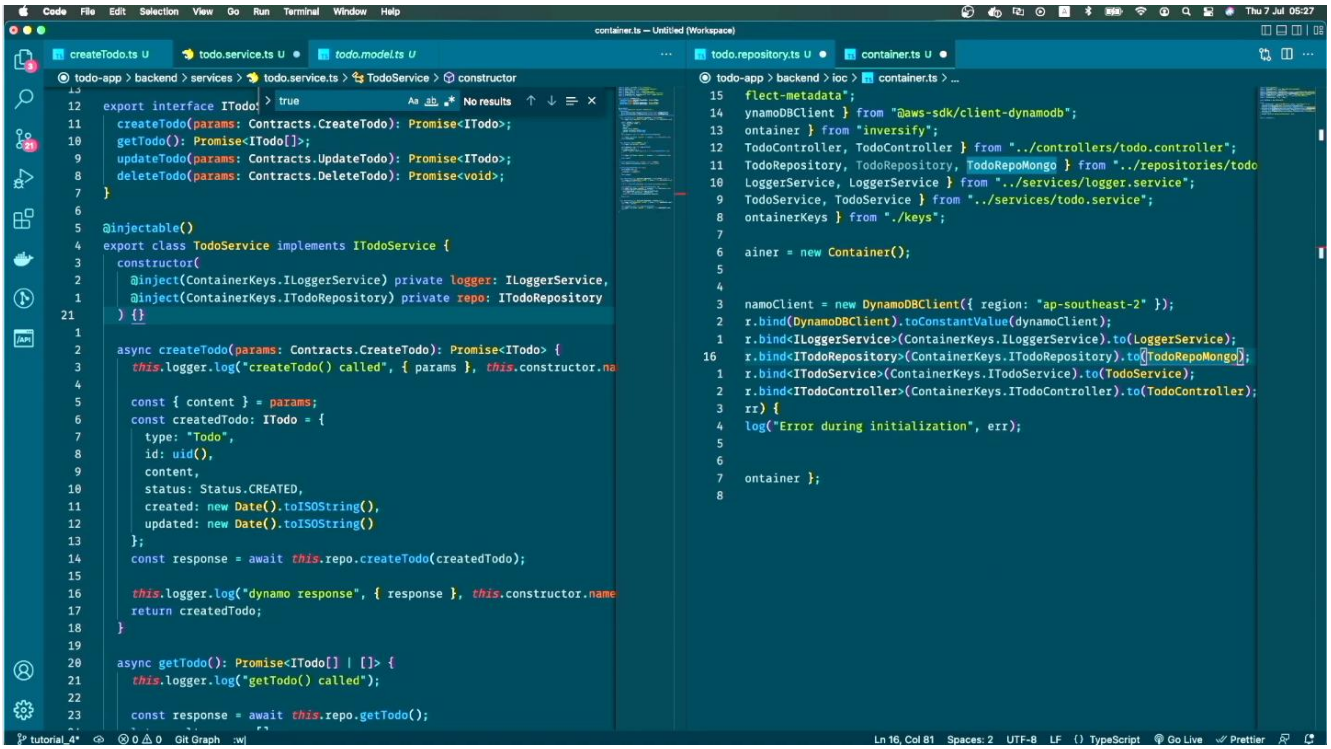
We can also change actual implementation


container.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
contracts
todo.contracts.ts
controllers
todo.controller.ts
helpers
error.helper.ts
response.helper.ts
ioc
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
OUTLINE
TIMELINE
createTodo.ts U
todo.service.ts U
todo.model.ts U
todo.repository.ts U
container.ts U
todo-app > backend > services > todo.service.ts > TodoService > constructor
todo-app > backend > ioc > container.ts > ...
No results
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 {
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 };
tutorial_4*
Git Graph
-- NORMAL --
Ln 1, Col 1
Spaces: 2
UTF-8
LF
TypeScript
Go Live
Prettier

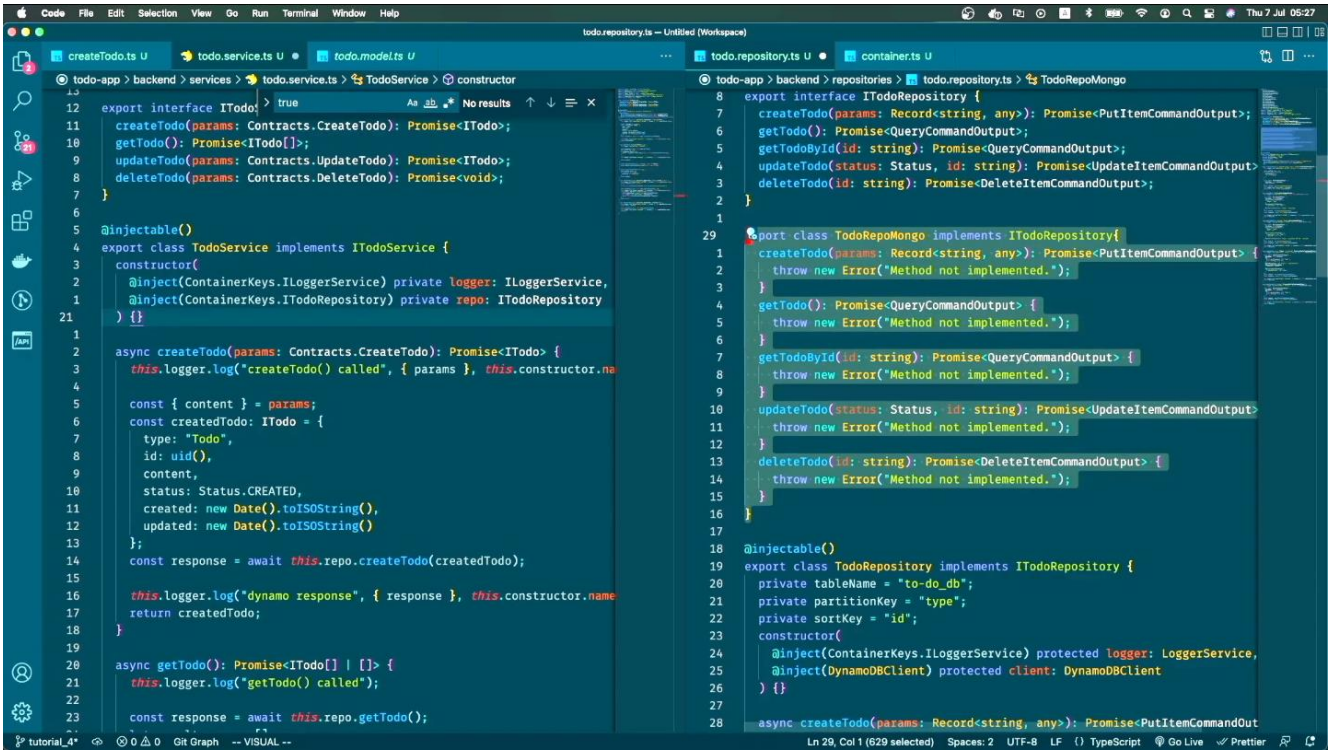

container.ts — Untitled (Workspace)
createTodo.ts U
todo.service.ts U
todo.model.ts U
todo.repository.ts U
container.ts U
todo-app > backend > services > todo.service.ts > TodoService > constructor
todo-app > backend > ioc > container.ts > ...
No results
export interface ITodoService {
createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
getTodo(): Promise<ITodo[]>;
updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
}
@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();
tutorial_4*
Git Graph
-- NORMAL --
Ln 1, Col 1
Spaces: 2
UTF-8
LF
TypeScript
Go Live
Prettier


container.ts — Untitled (Workspace)
createTodo.ts U
todo.service.ts U
todo.model.ts U
todo.repository.ts U
container.ts U
todo-app > backend > services > todo.service.ts > TodoService > constructor
todo-app > backend > ioc > container.ts > ...
export interface ITodo
createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
getTodo(): Promise<ITodo[]>;
updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
@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.na
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();
flect-metadata";
ynamoDBClient } from "@aws-sdk/client-dynamodb";
ontainer } from "inversify";
TodoController, TodoController } from "../controllers/todo.controller";
TodoRepository, TodoRepository } from "../repositories/todo.repository";
LoggerService, LoggerService } from "../services/logger.service";
TodoService, TodoService } from "../services/todo.service";
ontainerKeys } from "./keys";
ainer = new Container();
namoClient = new DynamoDBClient({ region: "ap-southeast-2" });
r.bind<DynamoDBClient>(DynamoDBClient).toConstantValue(dynamoClient);
r.bind<ILoggerService>(ContainerKeys.ILoggerService).to(LoggerService);
r.bind<ITodoRepository>(ContainerKeys.ITodoRepository).to(TodoRepository);
r.bind<ITodoService>(ContainerKeys.ITodoService).to(TodoService);
r.bind<ITodoController>(ContainerKeys.ITodoController).to(TodoController);
rr) {
log("Error during initialization", err);
ontainer };
tutorial_4* Git Graph -- NORMAL --
Ln 1, Col 1 Spaces: 2 UTF-8 LF TypeScript Go Live Prettier

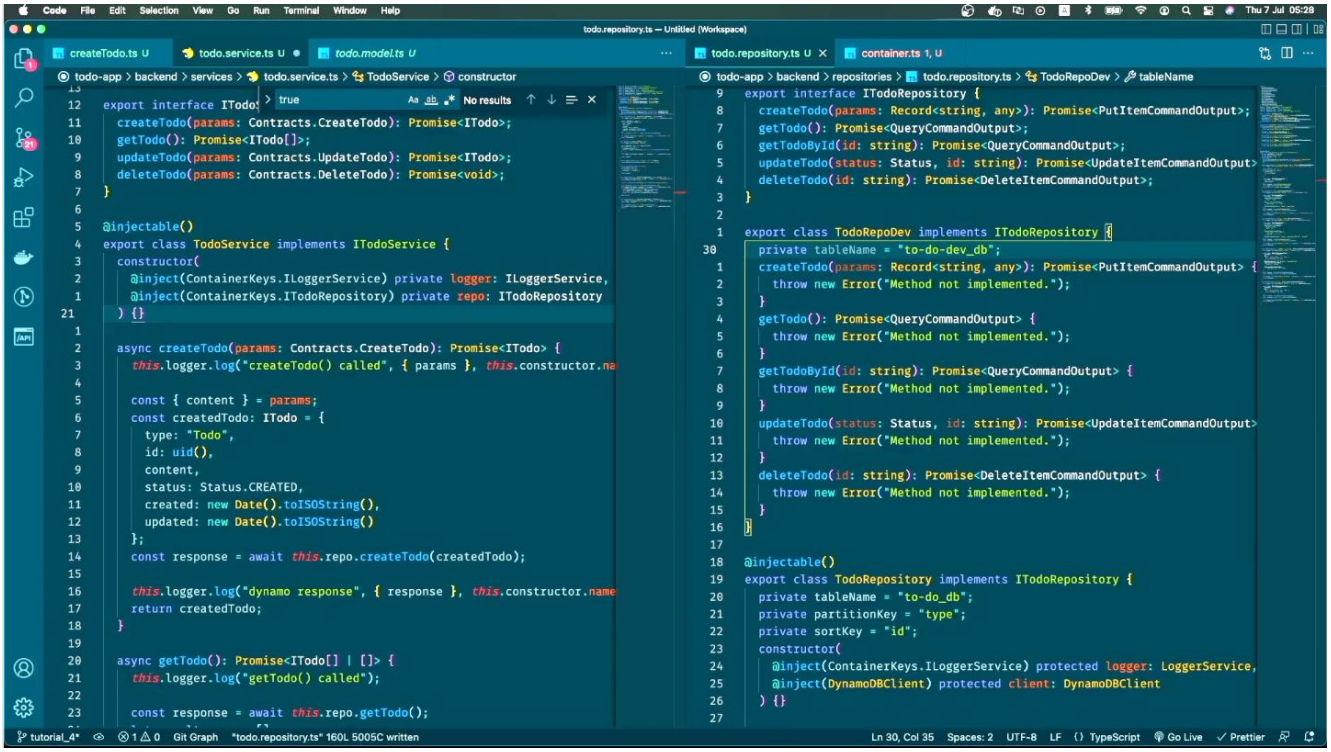

todo.repository.ts — Untitled (Workspace)
createTodo.ts U
todo.service.ts U
todo.model.ts U
todo.repository.ts U
container.ts U
todo-app > backend > services > todo.service.ts > TodoService > constructor
todo-app > backend > repositories > todo.repository.ts > TodoRepoMongo
export interface ITodo
createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
getTodo(): Promise<ITodo[]>;
updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
@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.na
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();
import { inject, injectable } from "inversify";
import { ContainerKeys } from "../ioc/keys";
import { Status } from "../models/todo.model";
import { LoggerService } from "../services/logger.service";
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(
tutorial_4* Git Graph -- NORMAL --
Ln 29, Col 25 Spaces: 2 UTF-8 LF TypeScript Go Live Prettier

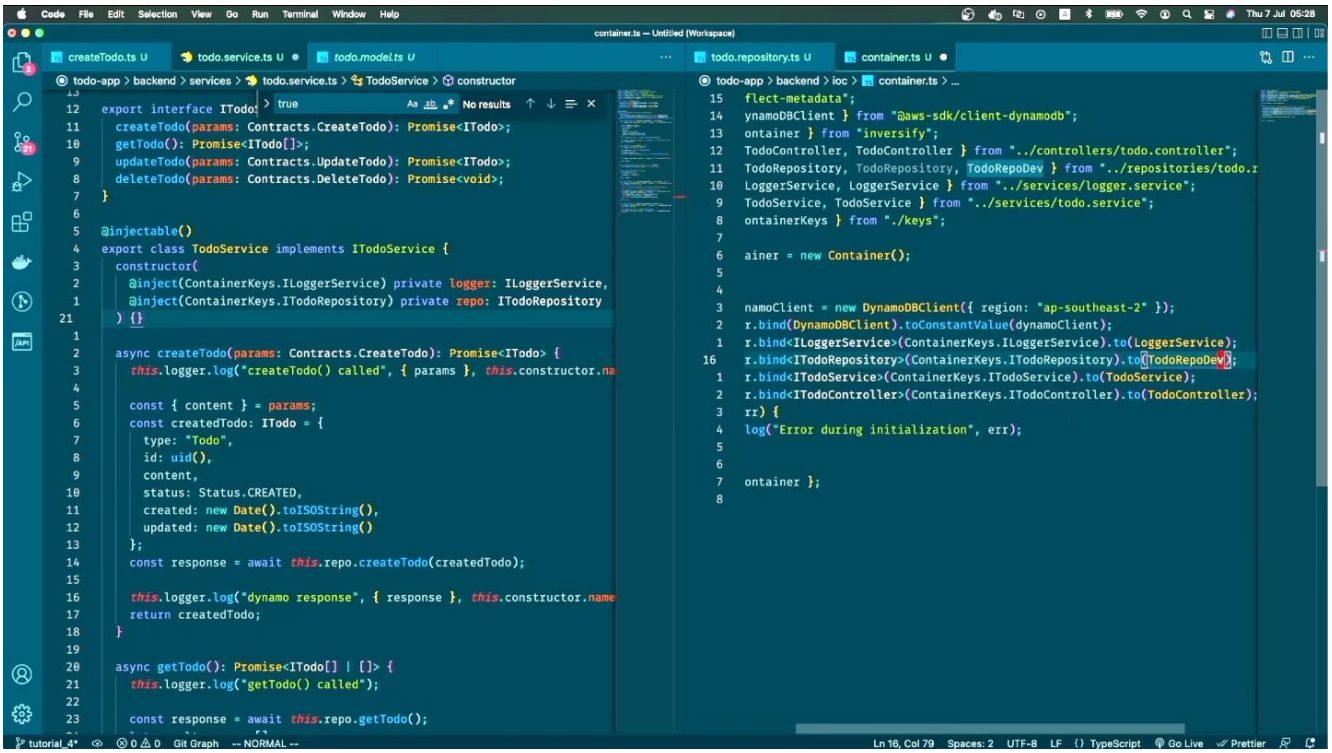


We can swap things easily as above

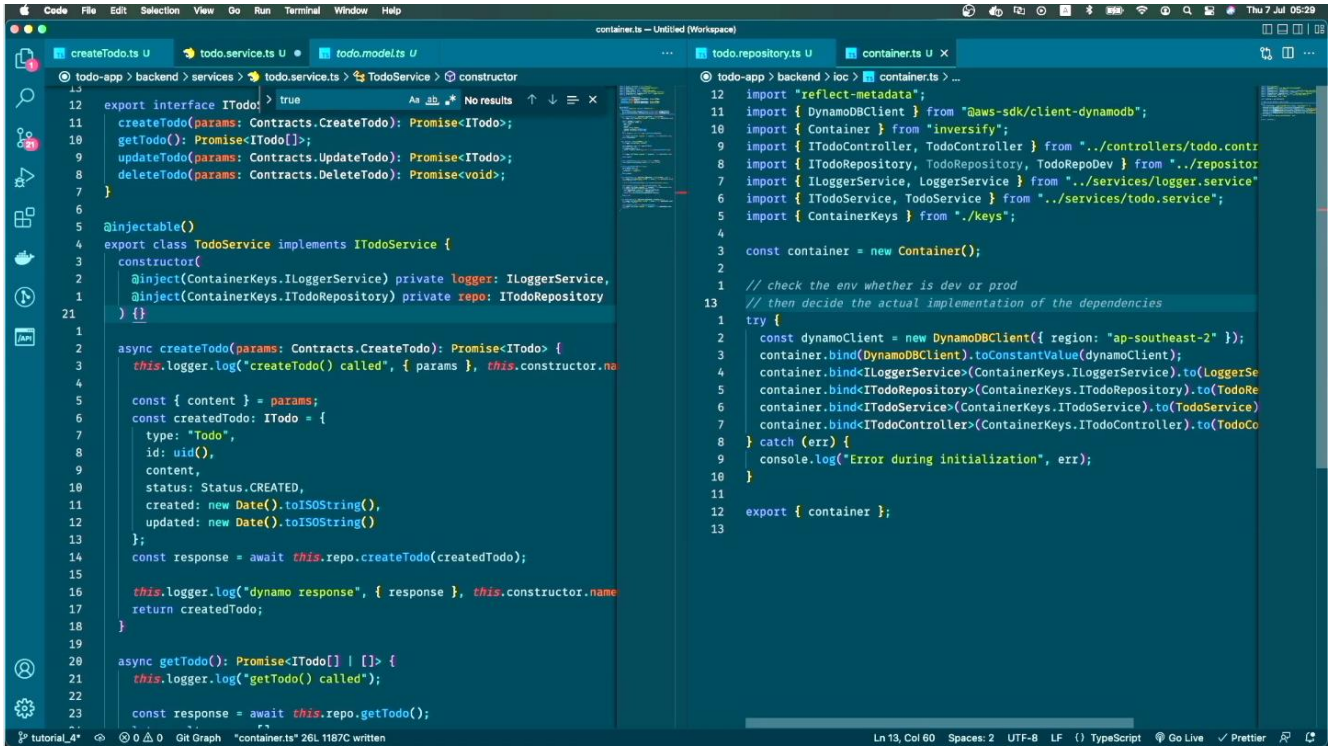


We can also have easy testing

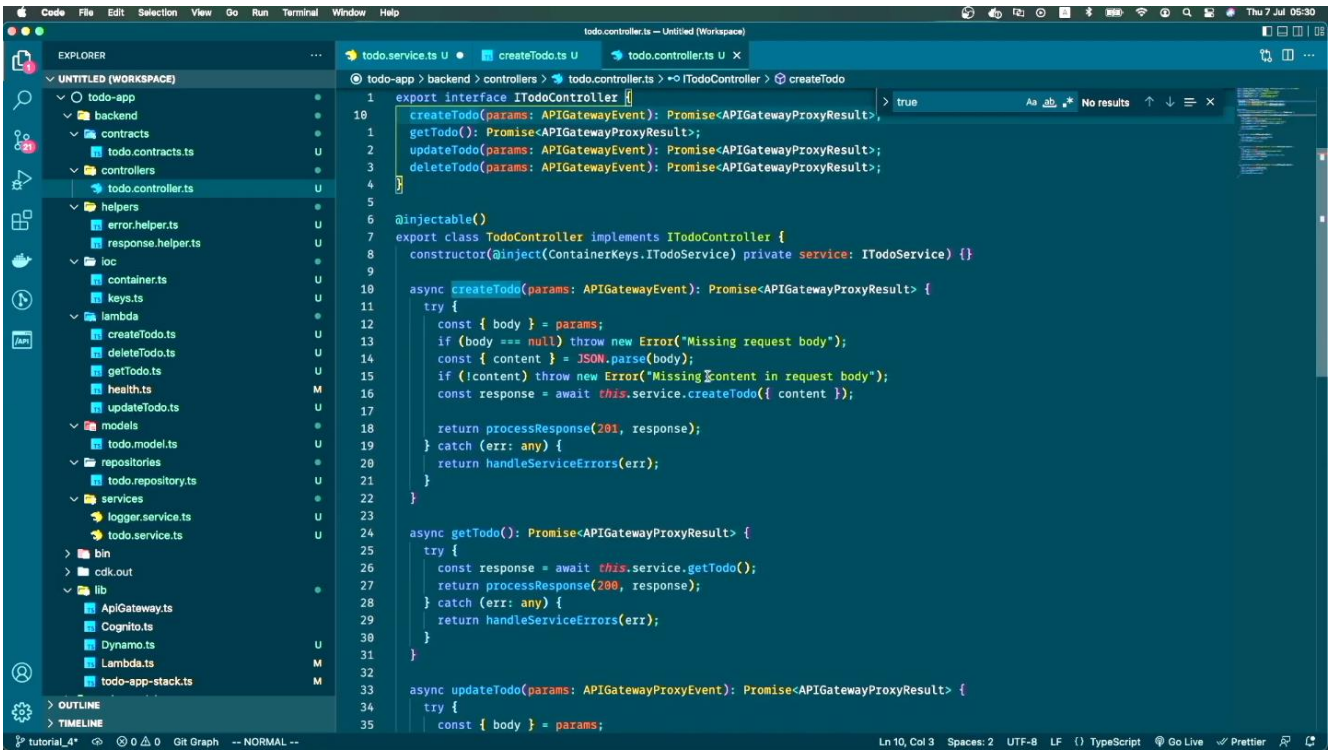


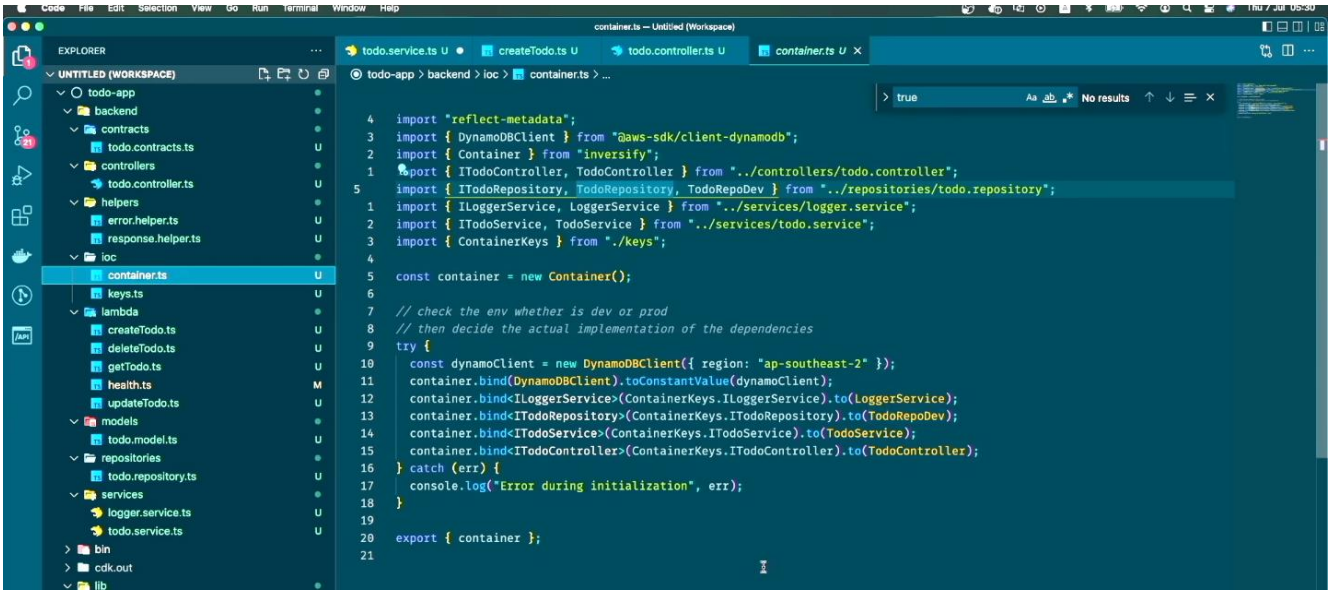


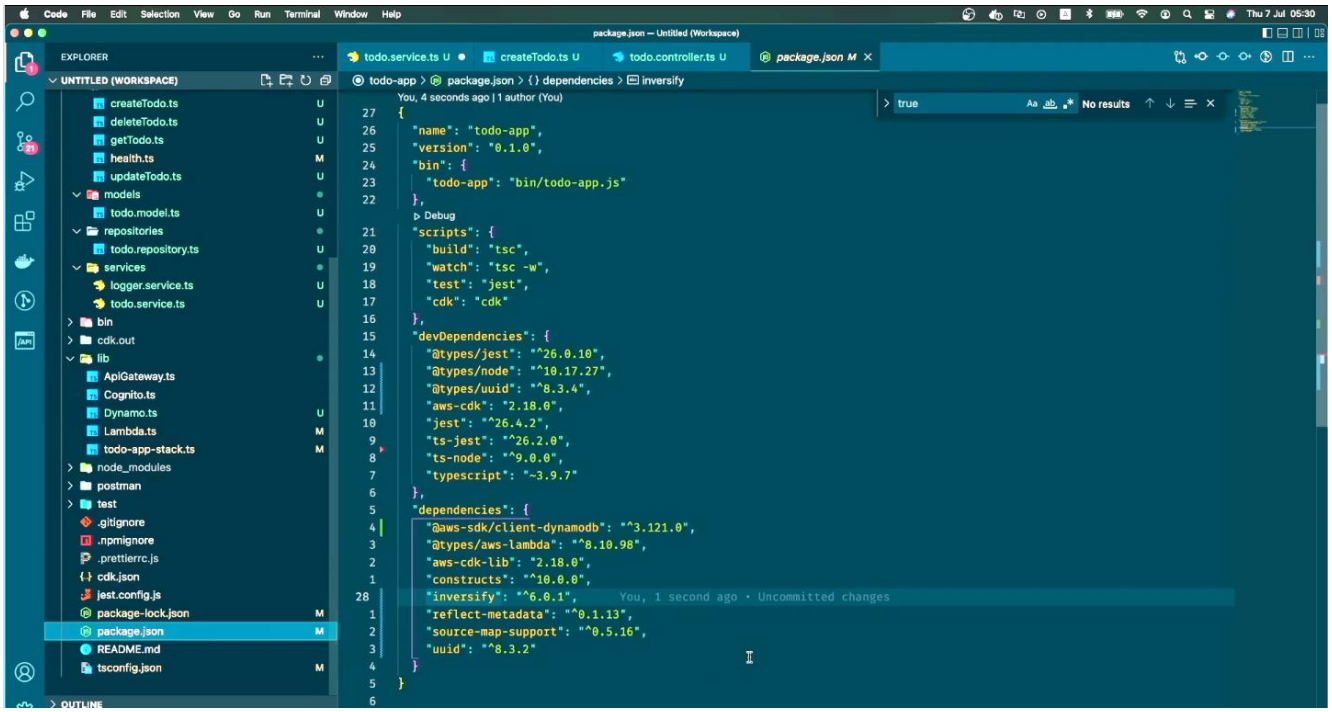
We can change the implementation from prod to dev database easily.


container.ts — Untitled (Workspace)
createTodo.ts U
todo.service.ts U
todo.model.ts U
todo.repository.ts U
container.ts U
todo-app > backend > services > todo.service.ts > TodoService > constructor
todo-app > backend > ioc > container.ts > ...
export interface ITodoService {
createTodo(params: Contracts.CreateTodo): Promise<ITodo>;
getTodo(): Promise<ITodo[]>;
updateTodo(params: Contracts.UpdateTodo): Promise<ITodo>;
deleteTodo(params: Contracts.DeleteTodo): Promise<void>;
}
@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.na
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();
import "reflect-metadata";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { Container } from "inversify";
import { ITodoController, TodoController } from "../controllers/todo.contr
import { ITodoRepository, TodoRepository, TodoRepoDev } from "../repositor
import { ILoggerService, LoggerService } from "../services/logger.service"
import { ITodoService, TodoService } from "../services/todo.service";
import { ContainerKeys } from "./keys";
const container = new Container();
// check the env whether is dev or prod
// then decide the actual implementation of the dependencies
try {
const dynamoClient = new DynamoDBClient({ region: "ap-southeast-2" });
container.bind(DynamoDBClient).toConstantValue(dynamoClient);
container.bind<ILoggerService>(ContainerKeys.ILoggerService).to(LoggerSe
container.bind<ITodoRepository>(ContainerKeys.ITodoRepository).to(TodoRe
container.bind<ITodoService>(ContainerKeys.ITodoService).to(TodoService)
container.bind<ITodoController>(ContainerKeys.ITodoController).to(TodoCo
} catch (err) {
console.log("Error during initialization", err);
}
export { container };
tutorial_4*
Git Graph
"container.ts" 26L 1187C written
Ln 13, Col 60
Spaces: 2
UTF-8
LF
TypeScript
Go Live
Prettier

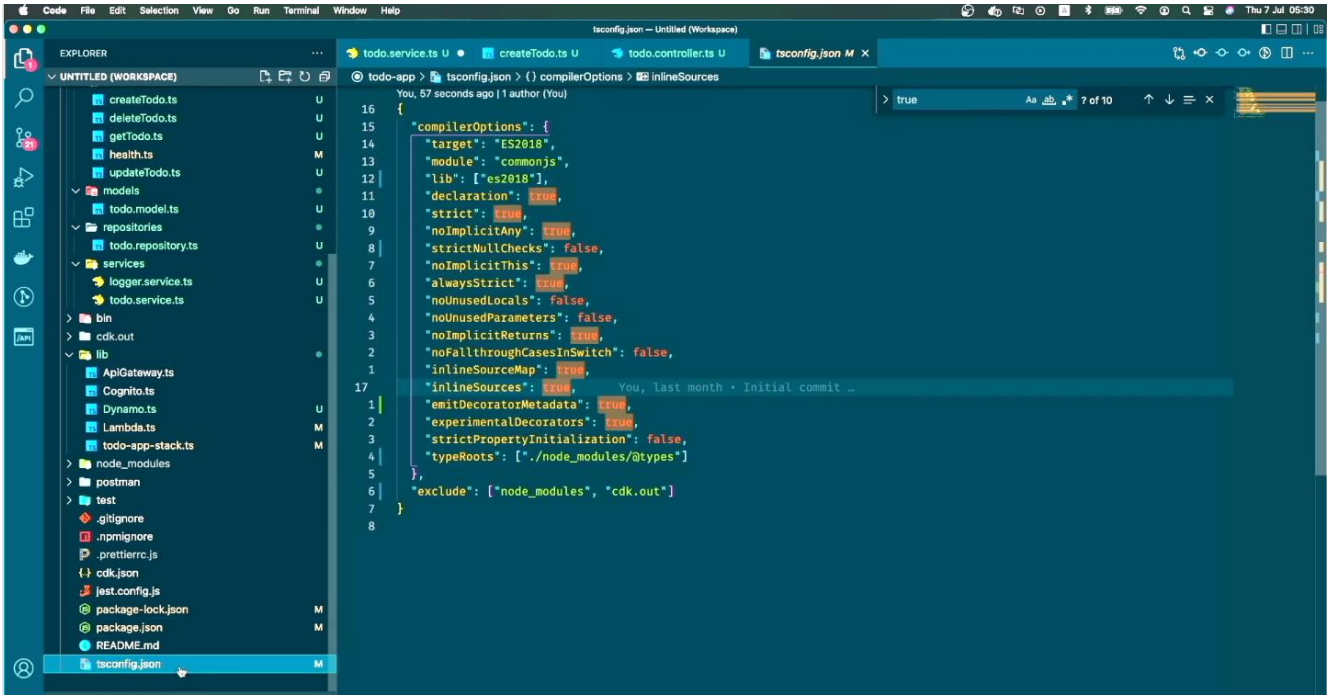

createTodo.ts — Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
todo-app
backend
contracts
todo.contracts.ts
controllers
todo.controller.ts
helpers
error.helper.ts
response.helper.ts
ioc
container.ts
keys.ts
lambda
createTodo.ts
deleteTodo.ts
todo.service.ts U
todo.model.ts U
createTodo.ts U
todo-app > backend > lambda > createTodo.ts > ...
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 hander event", { event });
return await controller.createTodo(event);
};


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
ioc
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
OUTLINE
TIMELINE
todo.service.ts U
createTodo.ts U
todo.controller.ts U
todo-app > backend > controllers > todo.controller.ts > ITodoController > createTodo
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 {
const response = await this.service.getTodo();
return processResponse(200, response);
} catch (err: any) {
return handleServiceErrors(err);
}
}
async updateTodo(params: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
try {
const { body } = params;
tutorial_4*
Git Graph
-- NORMAL --
Ln 10, Col 3
Spaces: 2
UTF-8
LF
TypeScript
Go Live
Prettier

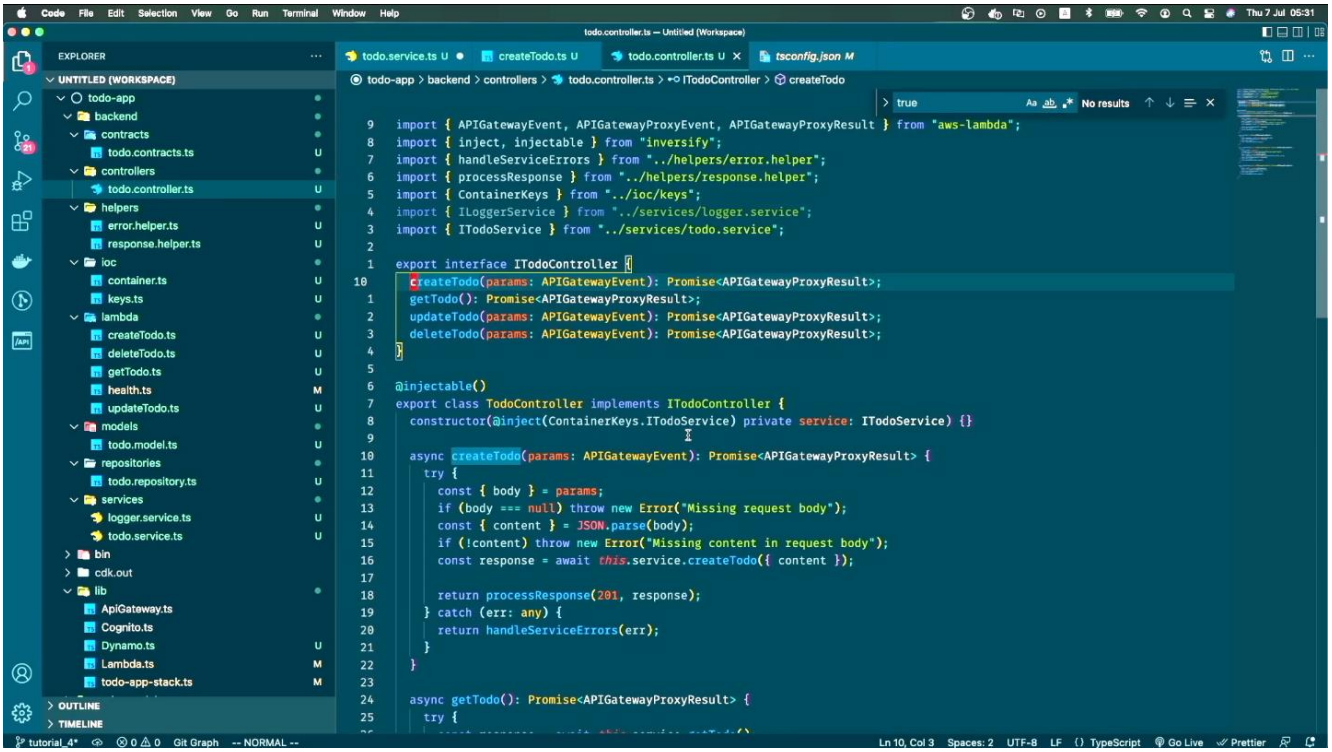




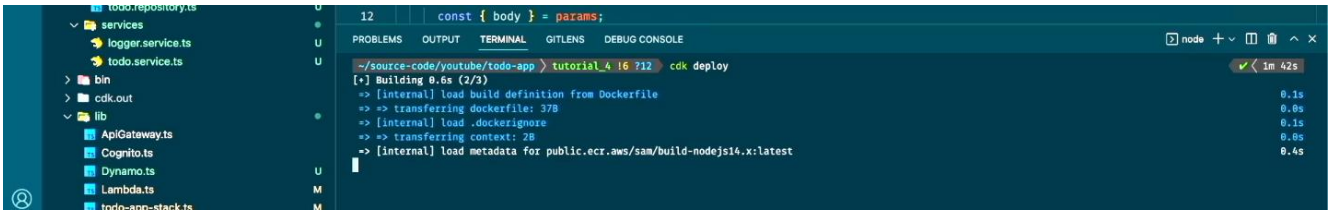
For doing this IoC, we use 2 packages called **inversify** and **reflect-metadata** as above



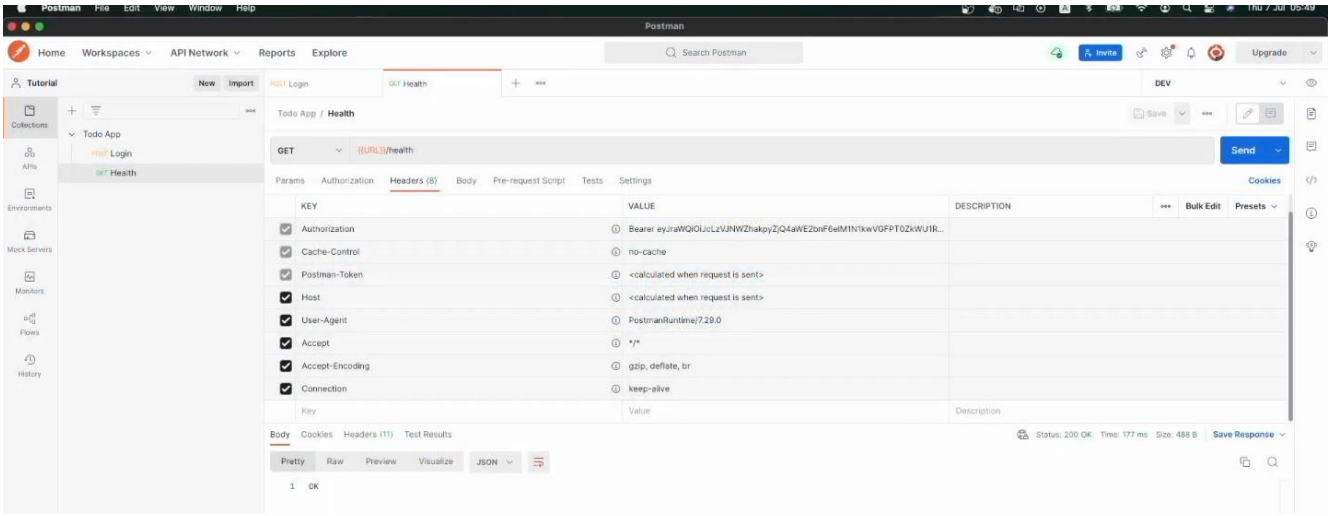
We also had to enable 2 things in our tsconfig.json file, the **emitDecoratorMetadata** and the **experimentalDecorators**

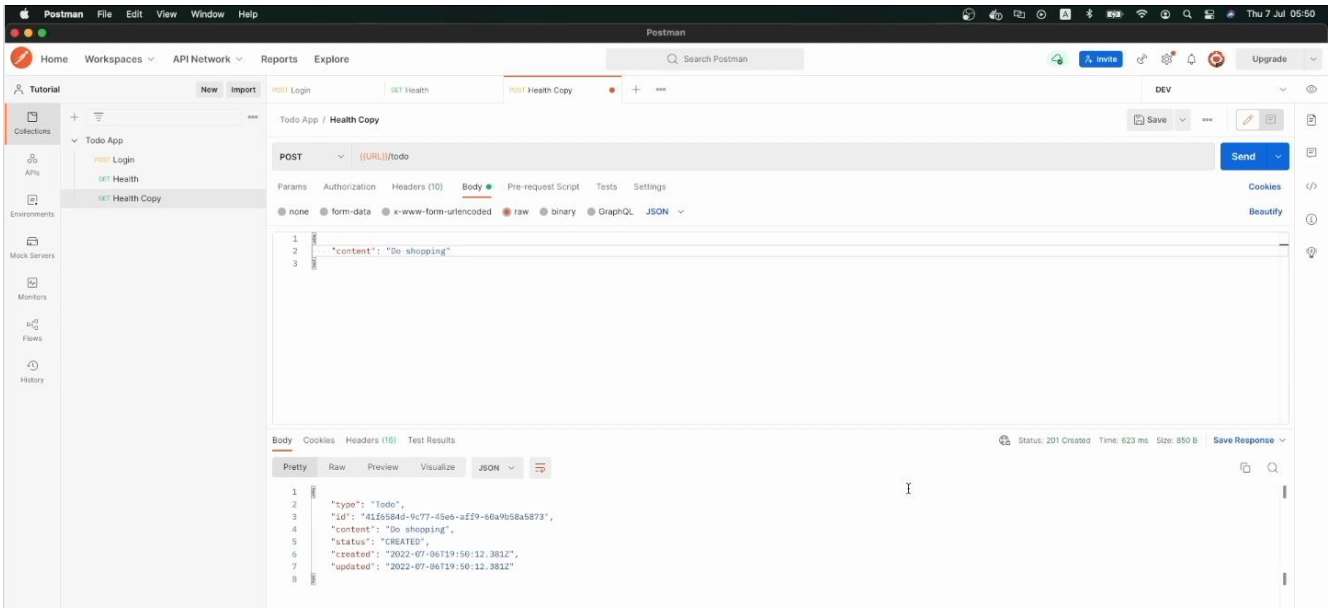


This allows us to use the **@injectable** and **@inject** commands. We can now use the CDK to deploy our CFN stack

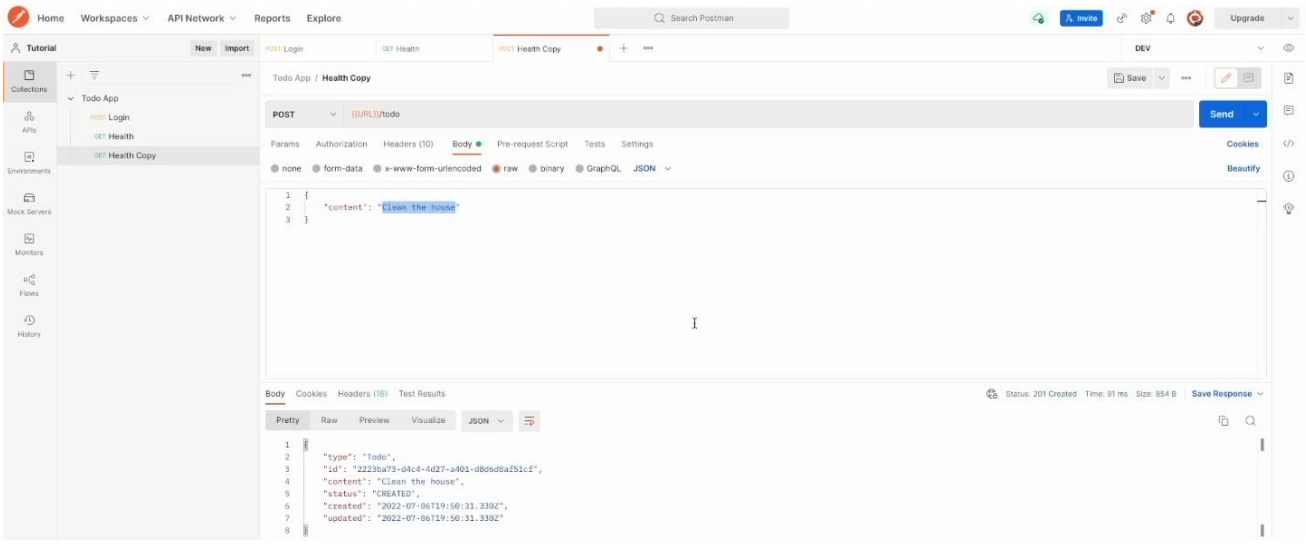


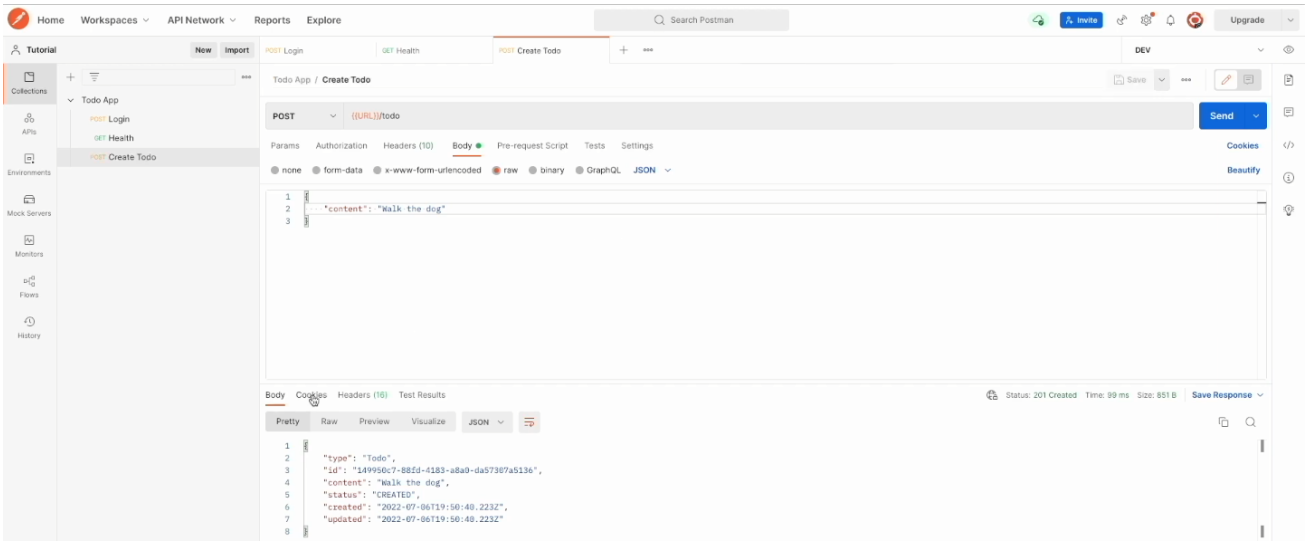
We can now try out the API using Postman with the **createTodo** API



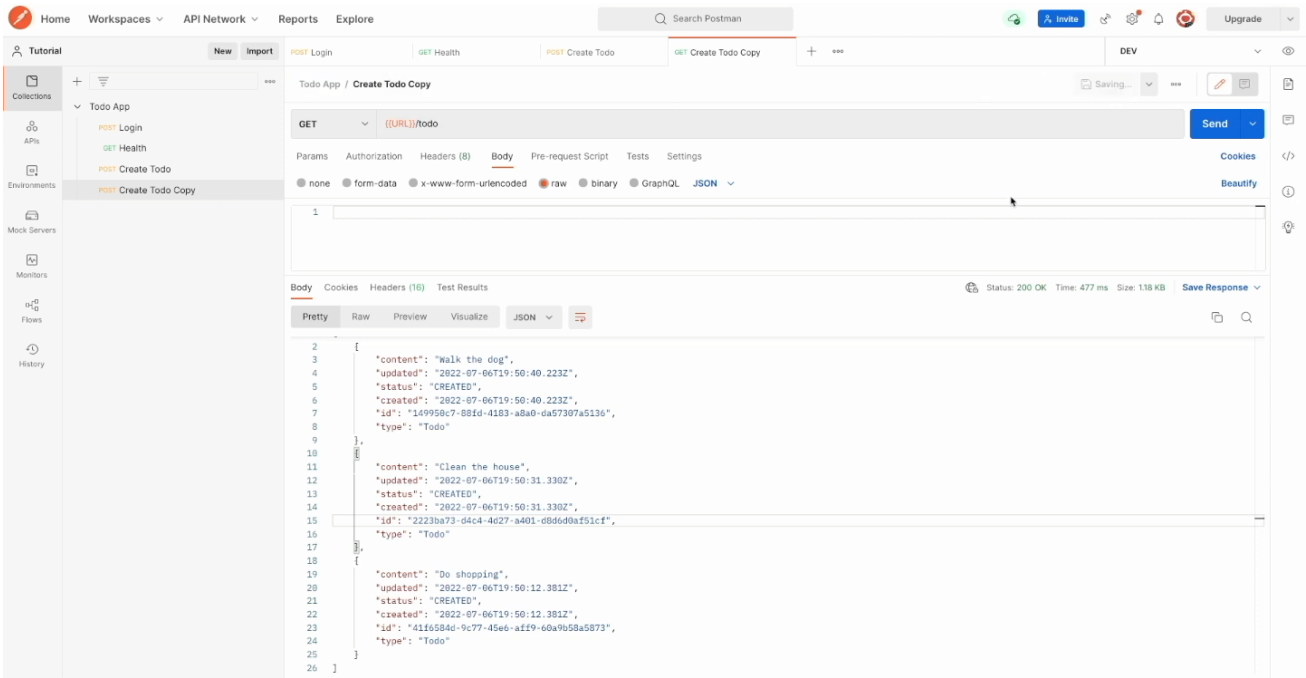


We get a 201 Created response

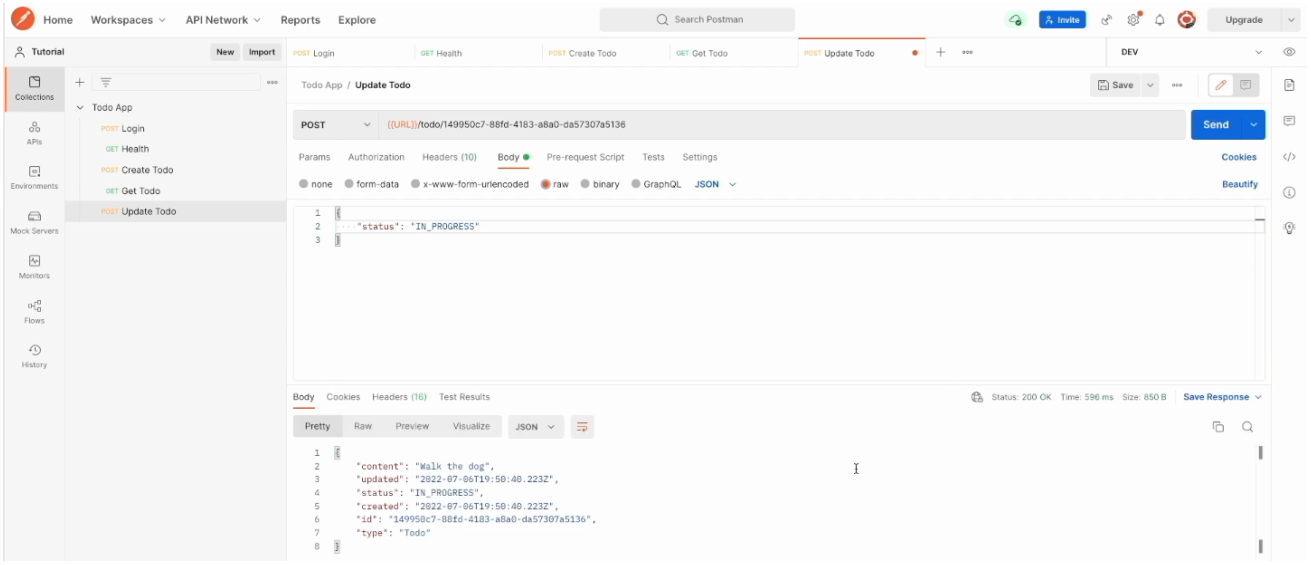


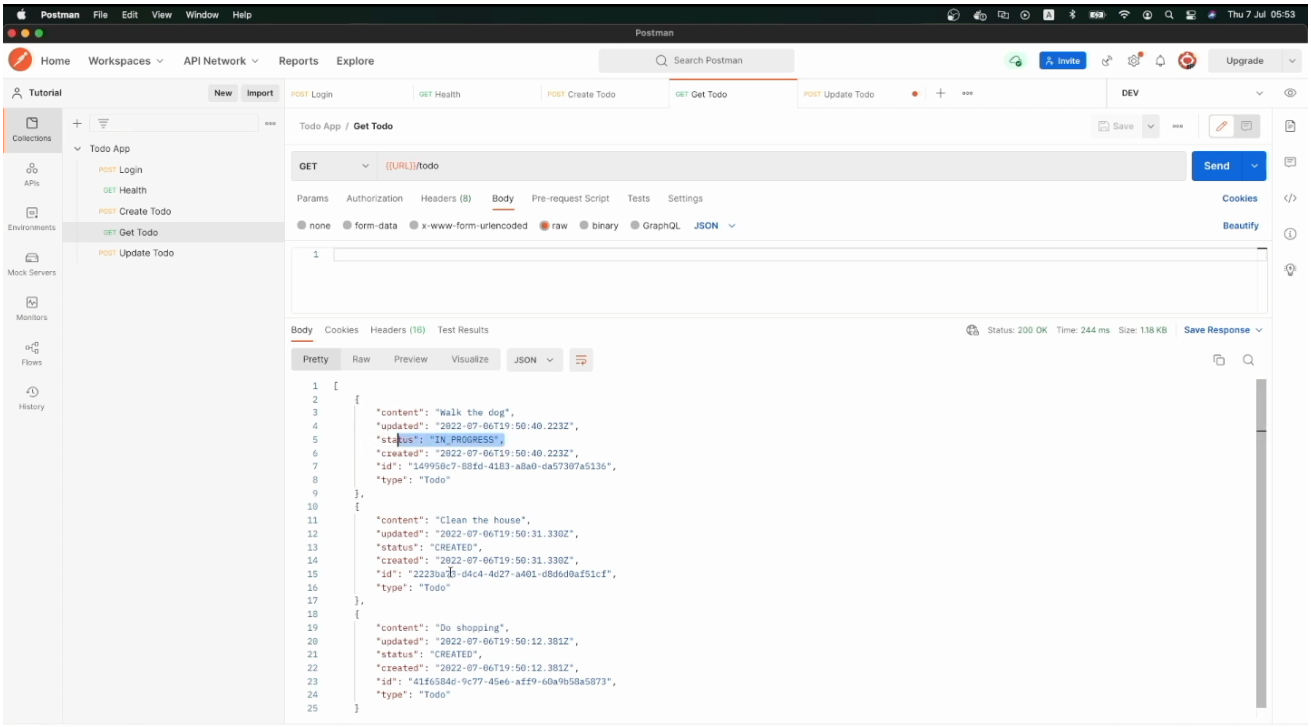


Next, let us test our **getTodo** API

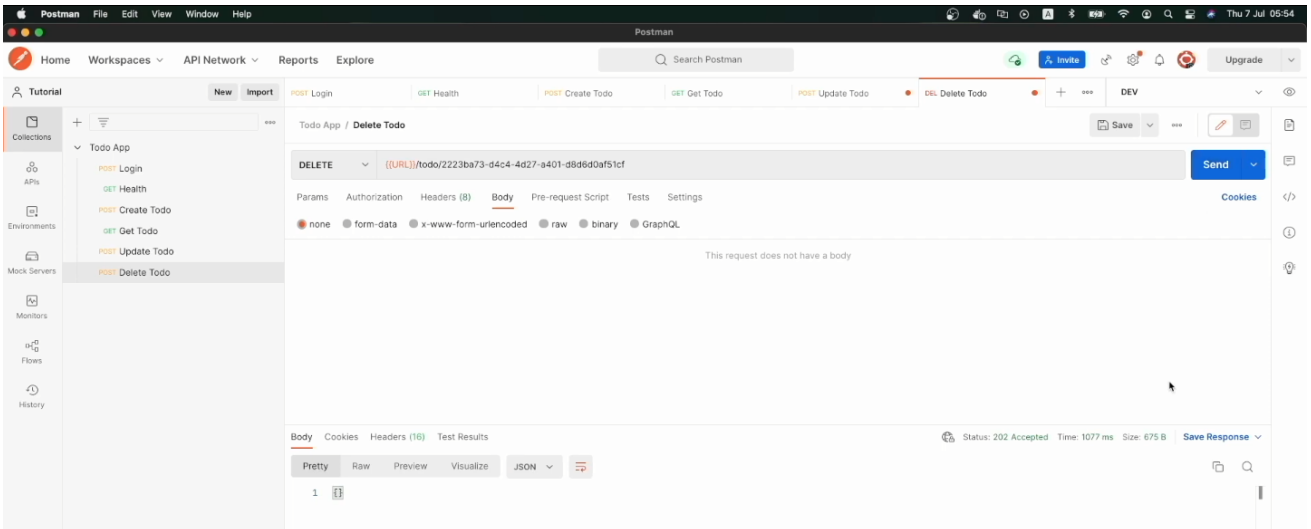


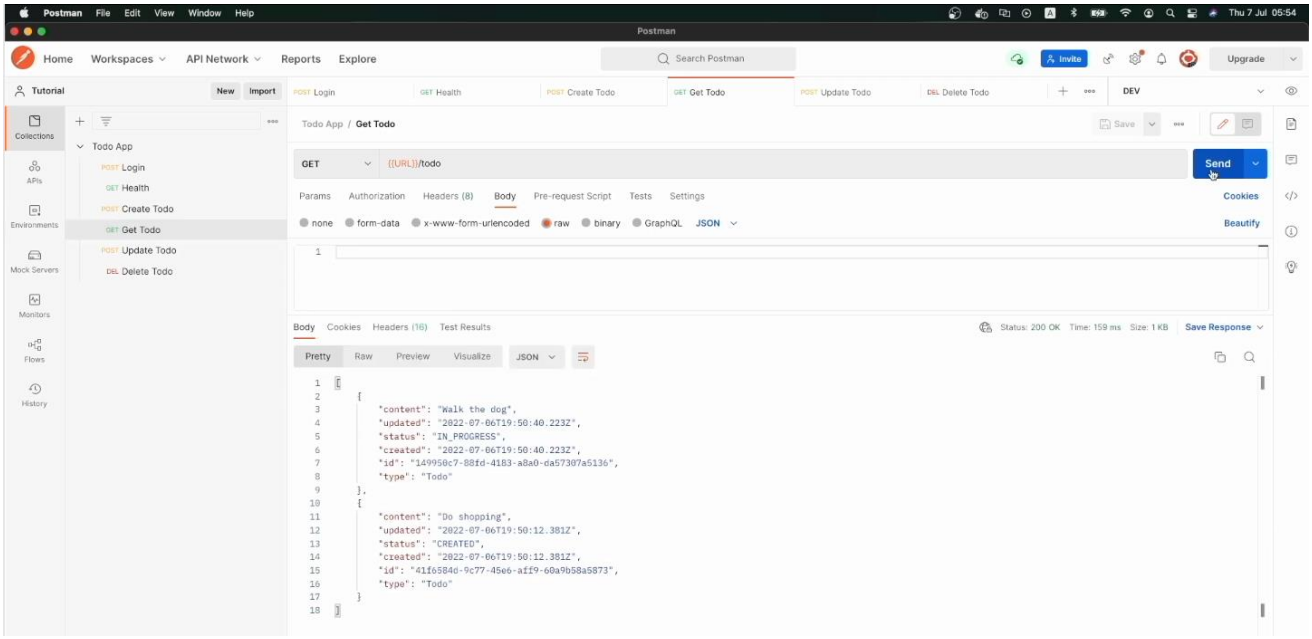
Next, let us test the **updateTodo** API using the POST command so that we get back a response object

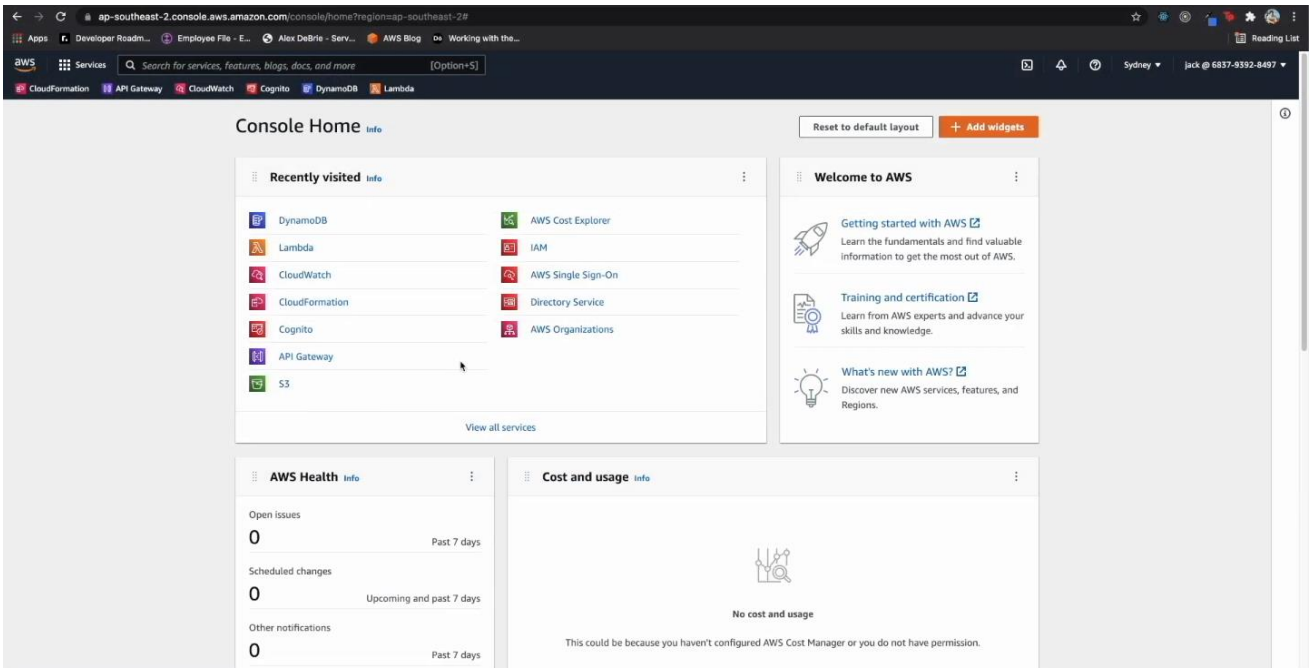


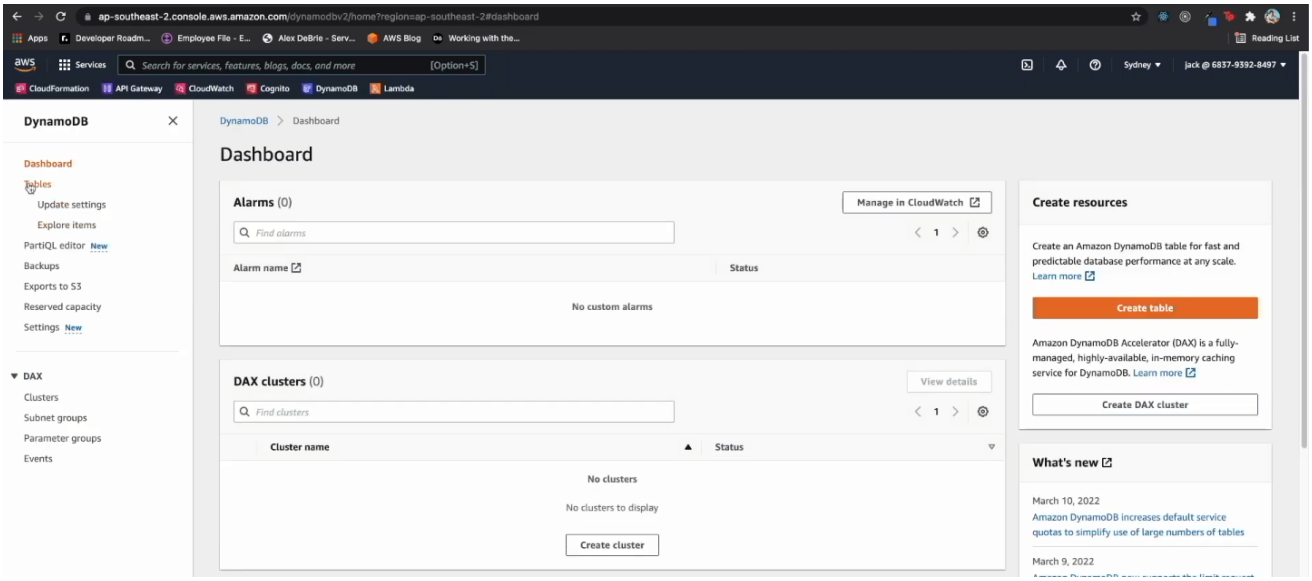


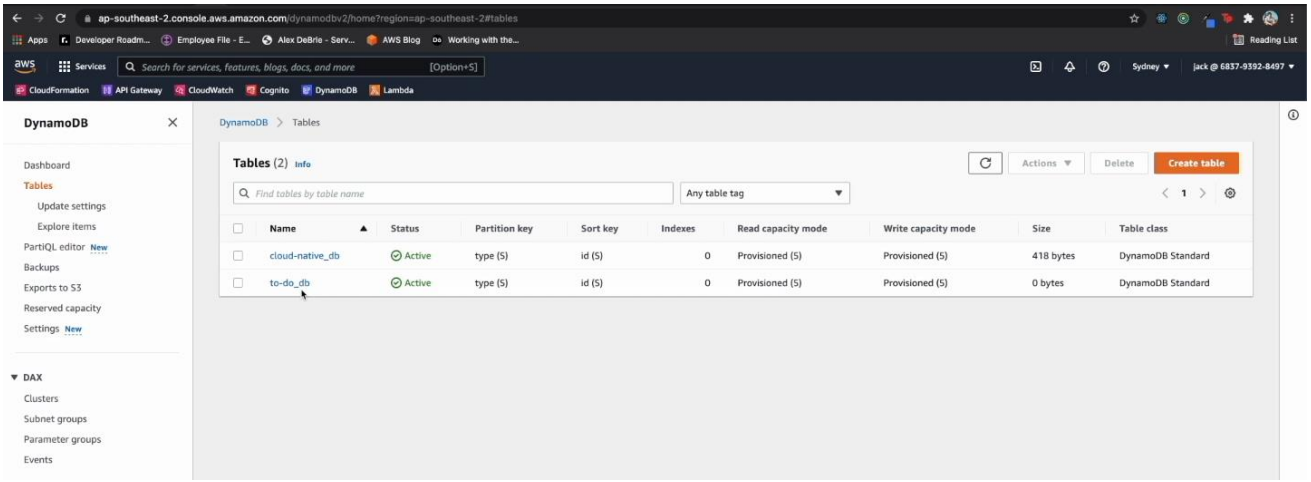
Next, let us test the **deleteTodo** API

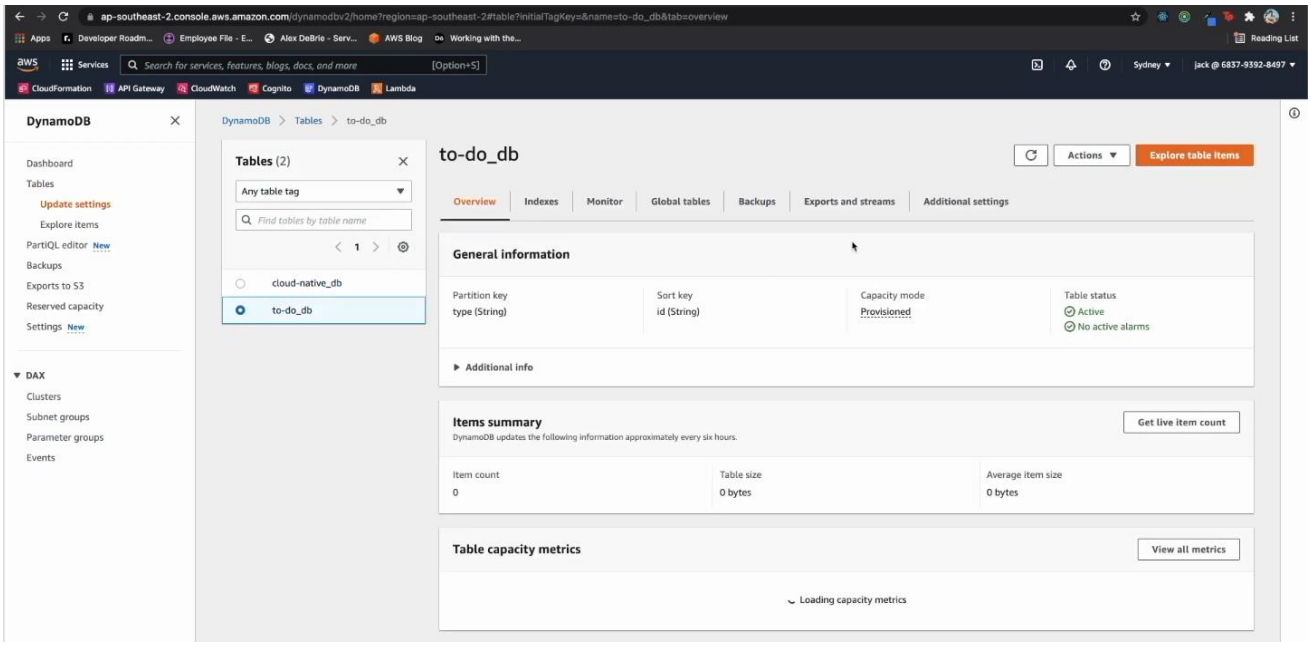


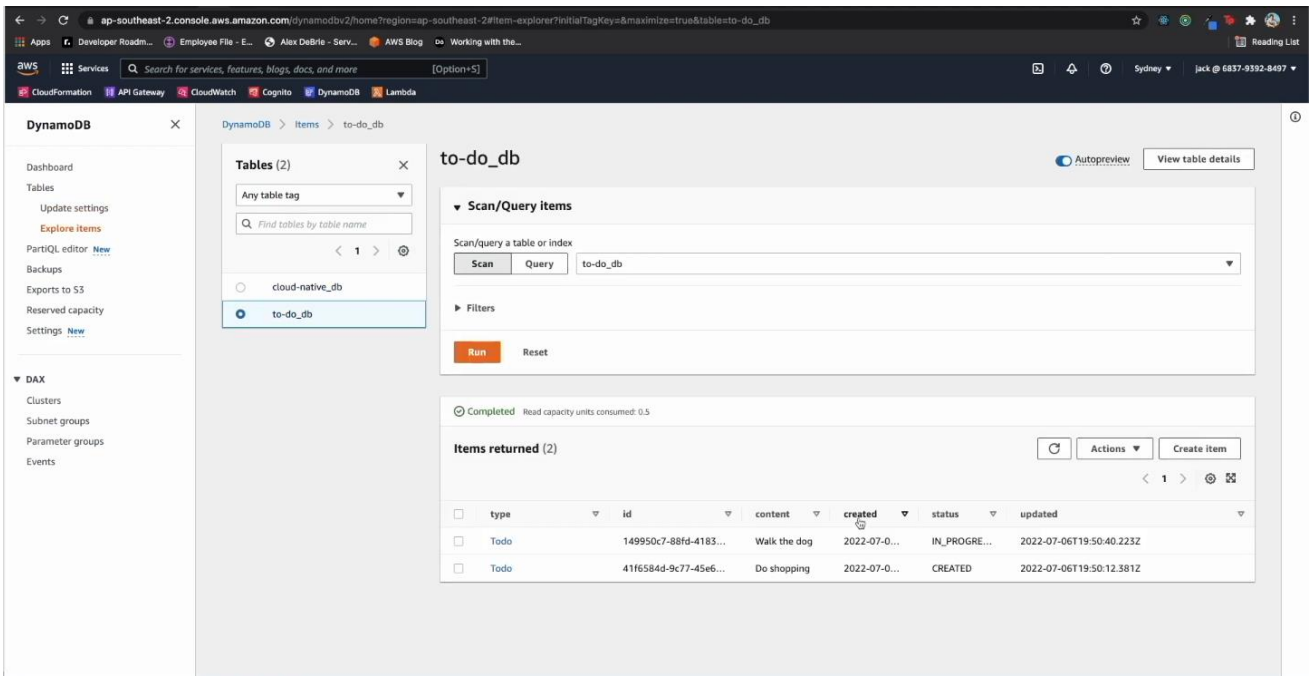

Postman
Home
Workspaces
API Network
Reports
Explore
Search Postman
Invite
Upgrade
Tutorial
New
Import
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Todo App
POST Login
GET Health
POST Create Todo
GET Get Todo
POST Update Todo
DEL Delete Todo
DEV
Todo App / Get Todo
Save
GET
{{URL}}/todo
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
Body
Cookies
Headers (16)
Test Results
Status: 200 OK
Time: 159 ms
Size: 1 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"content": "Walk the dog",
"updated": "2022-07-06T19:50:40.223Z",
"status": "IN_PROGRESS",
"created": "2022-07-06T19:50:40.223Z",
"id": "149950c7-88fd-4183-a8a0-da57307a5136",
"type": "Todo"
},
{
"content": "Do shopping",
"updated": "2022-07-06T19:50:12.381Z",
"status": "CREATED",
"created": "2022-07-06T19:50:12.381Z",
"id": "41f6584d-9c77-45e6-aff9-60a9b58a5873",
"type": "Todo"
}
]

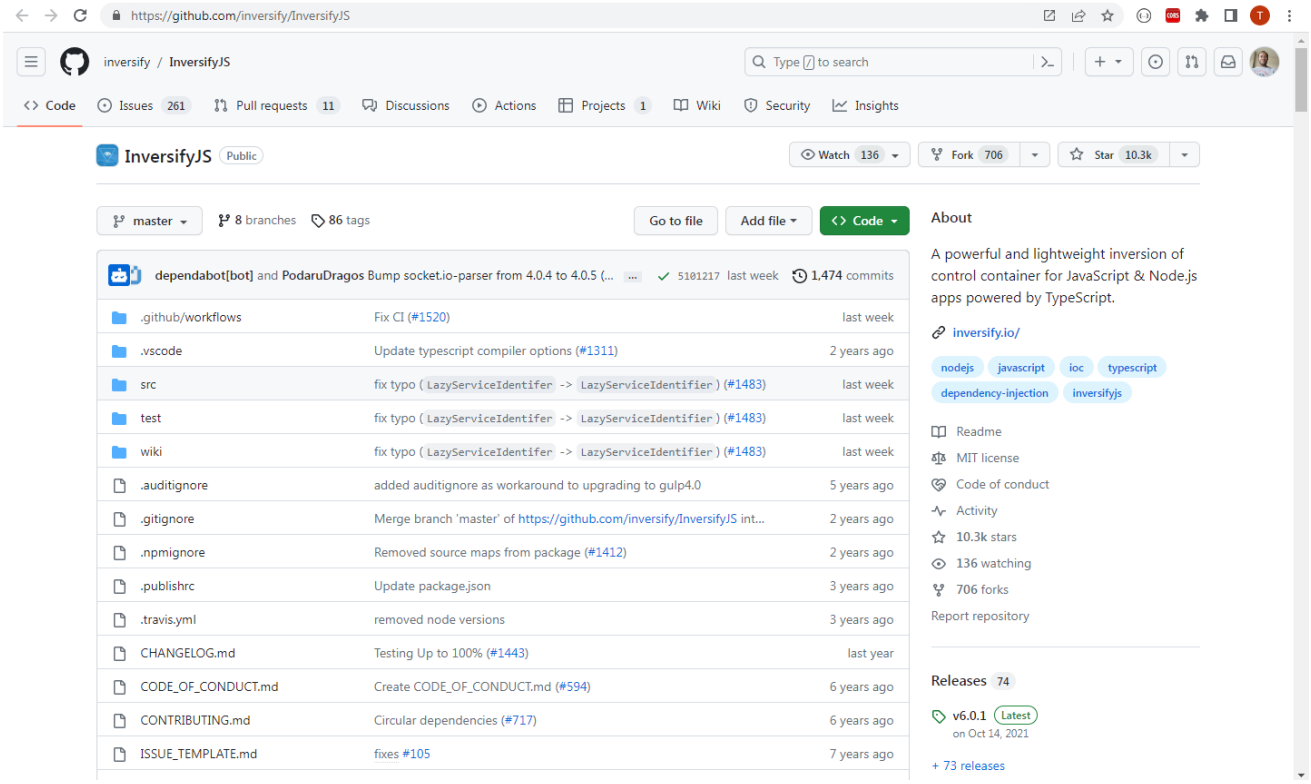

ap-southeast-2.console.aws.amazon.com/console/home?region=ap-southeast-2#
Services
Search for services, features, blogs, docs, and more
[Option+S]
Sydney
jack @ 6837-9392-8497
Console Home Info
Reset to default layout
+ Add widgets
Recently visited Info
DynamoDB
Lambda
CloudWatch
CloudFormation
Cognito
API Gateway
S3
AWS Cost Explorer
IAM
AWS Single Sign-On
Directory Service
AWS Organizations
View all services
Welcome to AWS
Getting started with AWS
Learn the fundamentals and find valuable information to get the most out of AWS.
Training and certification
Learn from AWS experts and advance your skills and knowledge.
What's new with AWS?
Discover new AWS services, features, and Regions.
AWS Health Info
Open issues
0
Past 7 days
Scheduled changes
0
Upcoming and past 7 days
Other notifications
0
Past 7 days
Cost and usage Info
No cost and usage
This could be because you haven't configured AWS Cost Manager or you do not have permission.


ap-southeast-2.console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#dashboard
DynamoDB
Dashboard
Tables
Update settings
Explore items
PartiQL editor New
Backups
Exports to S3
Reserved capacity
Settings New
DAX
Clusters
Subnet groups
Parameter groups
Events
DynamoDB > Dashboard
Dashboard
Alarms (0)
Manage in CloudWatch
Find alarms
Alarm name
Status
No custom alarms
DAX clusters (0)
View details
Find clusters
Cluster name
Status
No clusters
No clusters to display
Create cluster
Create resources
Create an Amazon DynamoDB table for fast and predictable database performance at any scale.
Learn more
Create table
Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. Learn more
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


ap-southeast-2.console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#tables
DynamoDB
Dashboard
Tables
Update settings
Explore items
PartiQL editor New
Backups
Exports to S3
Reserved capacity
Settings New
DAX
Clusters
Subnet groups
Parameter groups
Events
DynamoDB > Tables
Tables (2) Info
Actions
Delete
Create table
Find tables by table name
Any table tag
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 (5)
Provisioned (5)
418 bytes
DynamoDB Standard
to-do_db
Active
type (S)
id (S)
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)
Any table tag
Find tables by table name
cloud-native_db
to-do_db
to-do_db
Actions
Explore table items
Overview
Indexes
Monitor
Global tables
Backups
Exports and streams
Additional settings
General information
Partition key
type (String)
Sort key
id (String)
Capacity mode
Provisioned
Table status
Active
No active alarms
Additional info
Items summary
DynamoDB updates the following information approximately every six hours.
Get live item count
Item count
0
Table size
0 bytes
Average item size
0 bytes
Table capacity metrics
View all metrics
Loading capacity metrics


ap-southeast-2.console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#item-explorer?initialTagKey=&maximize=true&table=to-do_db
DynamoDB > Items > to-do_db
Tables (2)
Any table tag
Find tables by table name
cloud-native_db
to-do_db
to-do_db
Autopreview
View table details
Scan/Query items
Scan/query a table or index
Scan
Query
to-do_db
Filters
Run
Reset
Completed
Read capacity units consumed: 0.5
Items returned (2)
Actions
Create item
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) last week
.vscode Update typescript compiler options (#1311) 2 years ago
src fix typo (LazyServiceIdentifer -> LazyServiceIdentifier) (#1483) last week
test fix typo (LazyServiceIdentifer -> LazyServiceIdentifier) (#1483) last week
wiki fix typo (LazyServiceIdentifer -> LazyServiceIdentifier) (#1483) last week
.auditignore added auditignore as workaround to upgrading to gulp4.0 5 years ago
.gitignore Merge branch 'master' of https://github.com/inversify/InversifyJS int... 2 years ago
.npmignore Removed source maps from package (#1412) 2 years ago
.publishrc Update package.json 3 years ago
.travis.yml removed node versions 3 years ago
CHANGELOG.md Testing Up to 100% (#1443) last year
CODE_OF_CONDUCT.md Create CODE_OF_CONDUCT.md (#594) 6 years ago
CONTRIBUTING.md Circular dependencies (#717) 6 years ago
ISSUE_TEMPLATE.md fixes #105 7 years ago
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