**Backend Workshop**

# **Prerequisite:**

* Install Node<https://nodejs.org/en/download/> , Mac users can install using brew command too.

| brew install node |
| --- |

Note: in case of following error, install brew using <https://brew.sh/> :

Error: No developer tools installed.

Install the Command Line Tools:

xcode-select --install

* Install npm

| sudo apt-get install npm |
| --- |

* Verify Installation :

| node -v // The system should display the Node.js version installed on your system, version >=12 npm -v //The system should display the npm version |
| --- |

* Install NestJs CLI

| npm install -g @nestjs/cli ( sudo npm install -g @nestjs/cli) |
| --- |

* Some IntelliJ / any IDE
* Install Java/JDK
* Install DynamoDB

Download the latest version of DynamoDB:<https://s3.eu-central-1.amazonaws.com/dynamodb-local-frankfurt/dynamodb_local_latest.tar.gz> and run :

| cd dynamodb\_local\_latest |
| --- |

| java -Djava.library.path=./DynamoDBLocal\_lib -jar DynamoDBLocal.jar -sharedDb |
| --- |

If you're greeted with the following log on your console, you have successfully started the DynamoDB instance locally:

| Run a DynamoDB instance locally with Node.js Initializing DynamoDB Local with the following configuration: Port: 8000 InMemory: false DbPath: null SharedDb: true shouldDelayTransientStatuses: false CorsParams: \* |
| --- |

Install AWS CLI:<https://docs.aws.amazon.com/cli/v1/userguide/install-macos.html>

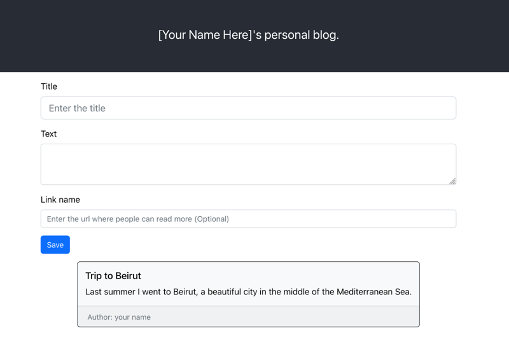
Then run aws --version to check if it's properly installed.

# 

# 

# 

# **Development:**



### **Phase1: Add Blog Post CreatePost, GetPost, GetPosts API to backend application. Data will be stored in memory (Not in the database).**

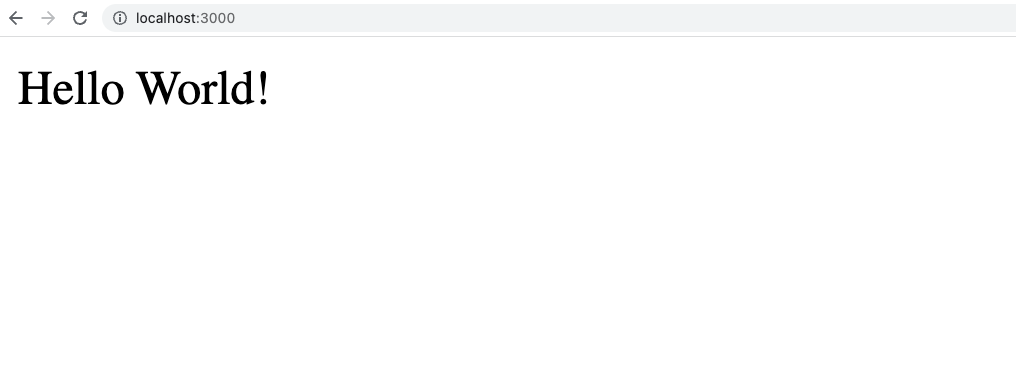
Step 1: Project Setup using Nest Command Line. Quickly create a dummy backend application.

| **nest new awe-blog-portal** |
| --- |

Step 2 : Start the backend application in watch mode, to pick the code changes without restart.

| cd **awe-blog-portal**  npm run start or npm run start:dev |
| --- |

Step 3 : Go to the web browser and enter URL: ​​<http://localhost:3000/> , you should see following:



Step 4: Add Post Module

| nest generate module post |
| --- |

It should create a *post.module.ts* file in the src/post folder. Component provides metadata that Nest makes use of to organise the application structure.

Step 5: Add Post Controller

| nest generate controller Post |
| --- |

A controller's purpose is to receive specific requests for the application. The routing mechanism controls which controller receives which requests. Frequently, each controller has more than one route, and different routes can perform different actions.

Step 6: Add Post Service

| nest generate service Post |
| --- |



It acts as an orchestrator or holds business logic.

Step 7 : Create directory dto (an object that carries data between processes or between two applications ) in src/post folder.

Add following file PostDTO.ts:

| export class PostDTO {  id: string;  text: string;  author: string;  title: string;  readmoreUrl?: string;  } |
| --- |

Add following file PostQueryResponseDTO.ts:

| import {PostDTO} from "./PostDTO";  export class PostQueryResponseDTO {  posts: PostDTO[]; } |
| --- |

Step 8: Do following changed to post.controller.ts file

| import {Controller, Post, Get, Body, Param} from '@nestjs/common'; import { PostDTO } from './dto/PostDTO'; import {PostQueryResponseDTO} from "./dto/PostQueryResponseDTO";  @Controller('post') export class PostController {  static postMap = new Map<string, PostDTO>();  static id = 1;   @Post()  async createPost(@Body() post: PostDTO){  console.log('post post request');  const identifier = "post"+ PostController.id;  post.id = identifier;  PostController.postMap.set(identifier, post);  PostController.id++;  return post;  }   @Get(':id')  async getPost(@Param('id') postId: string) : Promise<PostDTO> {  console.log('hello post get request');  return PostController.postMap.get(postId);  }    @Get()  async getPosts() : Promise<PostQueryResponseDTO> {  console.log('hello post get posts');  const posts = new PostQueryResponseDTO();  posts.posts = [... PostController.postMap.values()];  return posts;  } } |
| --- |

Step 9: Test the changes (use <https://www.postman.com/downloads/> if curl command doesn’t work):

Run following command to create Post in database:

**MAC or Linux:**

**curl --location --request POST 'http://localhost:3000/post' --header 'Content-Type: application/json' --data-raw '{"text": "Computer Student","author": "saroj","title": "Blog1" }'**

Or for windows

curl -X POST -H "Content-Type: application/json" -d "{\"text\": \"Computer Student\", \"author\": \"saroj\", \"title\": \"Blog1\"}" <http://localhost:3000/post>

Or go to <https://www.postman.com/downloads/> and click on try the web version , register using gmail , and go workspace and post a request like this, screenshot attached to configure the UI:

{

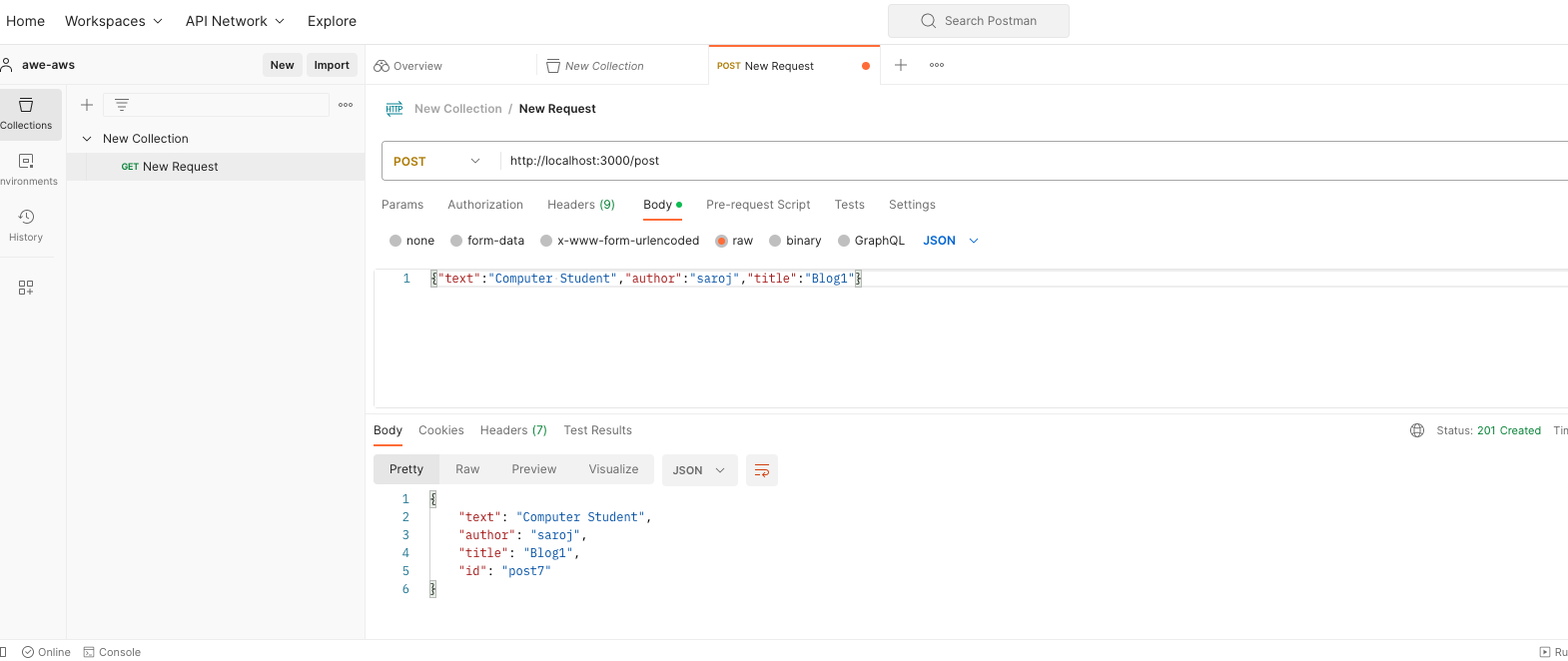
"text": "Computer Student",

"author": "saroj",

"title": "Blog1",

"id": "post1"

}



Should return following response:

| {"text":"Computer Student","author":"saroj","title":"Blog1","id":"post1"} |
| --- |

Run following command to get the post from the database:

| curl --location --request GET 'http://localhost:3000/post/post1' |
| --- |

Output:

| {"text":"Computer Student","author":"saroj","title":"Blog1","id":"post1"} |
| --- |

Run following command to get the list of post from the memory store:

| curl --location --request GET 'http://localhost:3000/post' |
| --- |

Output:

| {"posts":[{"text":"Computer Student","author":"saroj","title":"Blog1","id":"post1"}]} |
| --- |

### **Phase 3: Store the data in the database.**

Step1: Go to downloaded dynamoDB folder, for me this is the path(please cd into your own path):

| cd /Users/sarrawat/Downloads/dynamodb\_local\_latest |
| --- |

Step 2: Start the dynamoDB instance on your local machine. Please run the following command.

| java -Djava.library.path=./DynamoDBLocal\_lib -jar DynamoDBLocal.jar -sharedDb |
| --- |

You should see following output :

| Initializing DynamoDB Local with the following configuration: Port: 8000 InMemory: false DbPath: null SharedDb: true shouldDelayTransientStatuses: false CorsParams: \* |
| --- |

Step 3: Configure AWS CLI

| aws configure  AWS Access Key ID []: randomValue  AWS Secret Access Key []: randomValue  Default region name [ ]: local  Default output format []:json |
| --- |

3.1: Run following command to see everything is configured properly (run java -Djava.library.path=./DynamoDBLocal\_lib -jar DynamoDBLocal.jar -sharedDb in separate terminal ):

| aws dynamodb list-tables --endpoint-url http://localhost:8000 |
| --- |

You should see following output, empty table list as you haven’t created table yet.:

| { "TableNames": [] } |
| --- |

Step 4: Create a Posts Table to save Blog Post.

4.1 Install dependencies to connect to DynamoDB by following the below command:

| npm install aws-sdk npm install @aws/dynamodb-data-mapper npm install @aws/dynamodb-data-mapper-annotations |
| --- |

4.2 Create file named CreateTable.js in awe-blog-portal folder (basically ).

| var AWS = require("aws-sdk"); var region = "local";  AWS.config.update({  accessKeyId:"randomValue",  secretAccessKey: "randomValue",  region: region,  endpoint: "http://localhost:8000" });  var dynamodb = new AWS.DynamoDB() //low-level client  var tableName = "Posts";  var params = {  TableName : tableName,  KeySchema: [  { AttributeName: "id", KeyType: "HASH"}, //Partition key  ],  AttributeDefinitions: [  { AttributeName: "id", AttributeType: "S" }  ],  ProvisionedThroughput: {  ReadCapacityUnits: 10,  WriteCapacityUnits: 10  } };  dynamodb.createTable(params, function(err, data) {  if (err) {  console.error("Unable to create table. Error JSON:", JSON.stringify(err, null, 2));  } else {  console.log("Created table. Table description JSON:", JSON.stringify(data, null, 2));  } }); |
| --- |

4.3 : Run following command to create table:

| node CreateTable.js |
| --- |

It should result into following **output**:

| Created table. Table description JSON: {  "TableDescription": {  "AttributeDefinitions": [  {  "AttributeName": "id",  "AttributeType": "S"  }  ],  "TableName": "Posts",  "KeySchema": [  {  "AttributeName": "id",  "KeyType": "HASH"  }  ],  "TableStatus": "ACTIVE",  "CreationDateTime": "2022-06-09T15:08:56.366Z",  "ProvisionedThroughput": {  "LastIncreaseDateTime": "1970-01-01T00:00:00.000Z",  "LastDecreaseDateTime": "1970-01-01T00:00:00.000Z",  "NumberOfDecreasesToday": 0,  "ReadCapacityUnits": 10,  "WriteCapacityUnits": 10  },  "TableSizeBytes": 0,  "ItemCount": 0,  "TableArn": "arn:aws:dynamodb:ddblocal:000000000000:table/Posts"  } } |
| --- |

Step 5: Create Database Entity. Create entity directory under **src/post/**.

Add the following file under **src/post/entity**.

| import {attribute, hashKey, table} from '@aws/dynamodb-data-mapper-annotations'; @table ('Posts') export class PostEntity {  @hashKey()  id: string;  @attribute()  text: string;  @attribute()  author: string;  @attribute()  title: string; } |
| --- |

Step 6: Write code to talk to dynamoDB in **src/post/post.service.ts**:

| import { Injectable } from '@nestjs/common'; import {PostDTO} from "./dto/PostDTO"; import {DataMapper} from "@aws/dynamodb-data-mapper"; import {PostEntity} from "./entity/PostEntity"; import {DynamoDB} from "aws-sdk";  @Injectable() export class PostService {  async insert(post: PostDTO): Promise<PostDTO> {  try {  const database = new DataMapper({   client: new DynamoDB({ endpoint: 'http://localhost:8000',  region: 'local' }),  });  return await database.put(Object.assign(new PostEntity(),  {  id: post.id,  text: post.text,  title: post.title,  author: post.author  }));  } catch (error) {  console.log({ error: error.name }, 'Error inserting item');  throw new Error(error);  }  }   async get(post: PostDTO): Promise<PostDTO> {  const database = new DataMapper({   client: new DynamoDB({ endpoint: 'http://localhost:8000',  region: 'local' }),  });   const postEntity = new PostEntity();  postEntity.id = post.id;  return await database.get(postEntity);  } } |
| --- |

Step 7: Change post.controller.ts file to call service layer to connect with database :

| import {Controller, Post, Get, Body, Param} from '@nestjs/common'; import { PostDTO } from '../post/dto/PostDTO'; import {PostQueryResponseDTO} from "./dto/PostQueryResponseDTO"; import {PostService} from "./post.service";  @Controller('post') export class PostController {  // static postMap = new Map<string, PostDTO>();  static id = 1;   @Post()  async createPost(@Body() post: PostDTO){  console.log('post post request');  const identifier = "post"+ PostController.id;  post.id = identifier;  const service = new PostService();  service.insert(post);  console.log("database call pass");  PostController.id++;  return post;  }   @Get(':id')  async getPost(@Param('id') postId: string) : Promise<PostDTO> {  console.log('hello post get request');  const service = new PostService();  const post = new PostDTO();  post.id = postId;  return service.get(post);  }    /\*@Get()  async getPosts() : Promise<PostQueryResponse> {  console.log('hello post get posts');  //return PostController.postMap.get(postId);   const posts = new PostQueryResponse();  posts.posts = [... PostController.postMap.values()];  return posts;  }\*/ } |
| --- |

Step 8: Test the changes:

Run following command to create Post in database:

| curl --location --request POST 'http://localhost:3000/post' \ --header 'Content-Type: application/json' \ --data-raw '{"text": "Computer Student", "author": "saroj", "title": "Blog1" }' |
| --- |

Should return following response:

| {"text":"Computer Student","author":"saroj","title":"Blog1","id":"post1"} |
| --- |

Run following command to get the post from the database:

| curl --location --request GET 'http://localhost:3000/post/post1' |
| --- |

Output:

| {"text":"Computer Student","author":"saroj","title":"Blog1","id":"post1"} |
| --- |

### **Phase 2: Store the data in the database.**

Step1: Go to downloaded dynamoDB folder, for me this is the path(please cd into your own path):

| cd /Users/sarrawat/Downloads/dynamodb\_local\_latest |
| --- |

Step 2: Start the dynamoDB instance on your local machine. Please run the following command.

| $ java -Djava.library.path=./DynamoDBLocal\_lib -jar DynamoDBLocal.jar -sharedDb |
| --- |

You should see following output :

| Initializing DynamoDB Local with the following configuration: Port: 8000 InMemory: false DbPath: null SharedDb: true shouldDelayTransientStatuses: false CorsParams: \* |
| --- |

Step 3: Configure AWS CLI

| aws configure  AWS Access Key ID []: randomValue  AWS Secret Access Key []: randomValue  Default region name [ ]: local  Default output format []:json |
| --- |

Run following command to see everything is configured properly:

| aws dynamodb list-tables --endpoint-url http://localhost:8000 |
| --- |

You should see following output, empty table list as you haven’t created table yet.:

| { "TableNames": [] } |
| --- |

Step 4: Create a Posts Table to save Blog Post.

Install dependencies to connect to DynamoDB by following the below command:

| npm install aws-sdk npm install @aws/dynamodb-data-mapper npm install @aws/dynamodb-data-mapper-annotations |
| --- |

Create file named CreateTable.js in awe-blog-portal folder(basically ).

| var AWS = require("aws-sdk"); var region = "local";  AWS.config.update({  region: region,  endpoint: "http://localhost:8000" });  var dynamodb = new AWS.DynamoDB() //low-level client  var tableName = "Posts";  var params = {  TableName : tableName,  KeySchema: [  { AttributeName: "id", KeyType: "HASH"}, //Partition key  ],  AttributeDefinitions: [  { AttributeName: "id", AttributeType: "S" }  ],  ProvisionedThroughput: {  ReadCapacityUnits: 10,  WriteCapacityUnits: 10  } };  dynamodb.createTable(params, function(err, data) {  if (err) {  console.error("Unable to create table. Error JSON:", JSON.stringify(err, null, 2));  } else {  console.log("Created table. Table description JSON:", JSON.stringify(data, null, 2));  } }); |
| --- |

Run following command to create table:

| node CreateTable.js |
| --- |

It should result into following output:

| Created table. Table description JSON: {  "TableDescription": {  "AttributeDefinitions": [  {  "AttributeName": "id",  "AttributeType": "S"  }  ],  "TableName": "Posts",  "KeySchema": [  {  "AttributeName": "id",  "KeyType": "HASH"  }  ],  "TableStatus": "ACTIVE",  "CreationDateTime": "2022-06-09T15:08:56.366Z",  "ProvisionedThroughput": {  "LastIncreaseDateTime": "1970-01-01T00:00:00.000Z",  "LastDecreaseDateTime": "1970-01-01T00:00:00.000Z",  "NumberOfDecreasesToday": 0,  "ReadCapacityUnits": 10,  "WriteCapacityUnits": 10  },  "TableSizeBytes": 0,  "ItemCount": 0,  "TableArn": "arn:aws:dynamodb:ddblocal:000000000000:table/Posts"  } } |
| --- |

Step 5: Create Database Entity. Create entity directory under src/post/.

Add the following file under src/post/entity.

| import {attribute, hashKey, table} from '@aws/dynamodb-data-mapper-annotations'; @table ('Posts') export class PostEntity {  @hashKey()  id: string;  @attribute()  text: string;  @attribute()  author: string;  @attribute()  title: string; } |
| --- |

Step 6: Write code to talk to dynamoDB in **src/post/post.service.ts**:

| import { Injectable } from '@nestjs/common'; import {PostDTO} from "./dto/PostDTO"; import {DataMapper} from "@aws/dynamodb-data-mapper"; import {PostEntity} from "./entity/PostEntity"; import {DynamoDB} from "aws-sdk";  @Injectable() export class PostService {  async insert(post: PostDTO): Promise<PostDTO> {  try {  const database = new DataMapper({   client: new DynamoDB({ endpoint: 'http://localhost:8000',  region: 'local' }),  });  return await database.put(Object.assign(new PostEntity(),  {  id: post.id,  text: post.text,  title: post.title,  author: post.author  }));  } catch (error) {  console.log({ error: error.name }, 'Error inserting item');  throw new Error(error);  }  }   async get(post: PostDTO): Promise<PostDTO> {  const database = new DataMapper({   client: new DynamoDB({ endpoint: 'http://localhost:8000',  region: 'local' }),  });   const postEntity = new PostEntity();  postEntity.id = post.id;  return await database.get(postEntity);  } } |
| --- |

Step 7: Change post.controller.ts file to call service layer to connect with database :

| import {Controller, Post, Get, Body, Param} from '@nestjs/common'; import { PostDTO } from '../post/dto/PostDTO'; import {PostQueryResponseDTO} from "./dto/PostQueryResponseDTO"; import {PostService} from "./post.service";  @Controller('post') export class PostController {  // static postMap = new Map<string, PostDTO>();  static id = 1;   @Post()  async createPost(@Body() post: PostDTO){  console.log('post post request');  const identifier = "post"+ PostController.id;  post.id = identifier;  const service = new PostService();  service.insert(post);  console.log("database call pass");  PostController.id++;  return post;  }   @Get(':id')  async getPost(@Param('id') postId: string) : Promise<PostDTO> {  console.log('hello post get request');  const service = new PostService();  const post = new PostDTO();  post.id = postId;  return service.get(post);  }    /\*@Get()  async getPosts() : Promise<PostQueryResponseDTO> {  console.log('hello post get posts');  //return PostController.postMap.get(postId);   const posts = new PostQueryResponseDTO();  posts.posts = [... PostController.postMap.values()];  return posts;  }\*/ } |
| --- |

Step 8: Test the changes:

Run following command to create Post in database:

| curl --location --request POST 'http://localhost:3000/post' \ --header 'Content-Type: application/json' \ --data-raw '{"text": "Computer Student", "author": "saroj", "title": "Blog1" }' |
| --- |

Should return following response:

| {"text":"Computer Student","author":"saroj","title":"Blog1","id":"post1"} |
| --- |

Run following command to get the post from the database:

| curl --location --request GET 'http://localhost:3000/post/post1' |
| --- |

Output:

| {"text":"Computer Student","author":"saroj","title":"Blog1","id":"post1"} |
| --- |

# **Resources:**

* **Please refer to** [**https://docs.google.com/document/d/1kmdG242SWhJo3TuS8fZpgvu73wpza2lDyb6HVen5gFA/edit#heading=h.d9ipr4i8dwfx**](https://docs.google.com/document/d/1kmdG242SWhJo3TuS8fZpgvu73wpza2lDyb6HVen5gFA/edit#heading=h.d9ipr4i8dwfx) **to have access to the code in case you didn’t manage to finish the workshop.**
* **Nest:** <https://docs.nestjs.com/modules>
* **Node:** <https://nodejs.org/api/>
* **DynamoDB**: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBLocal.html>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>

<https://aws.amazon.com/blogs/aws/dynamodb-local-for-desktop-development/>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DAX.client.run-application-nodejs.01-create-table.html>