

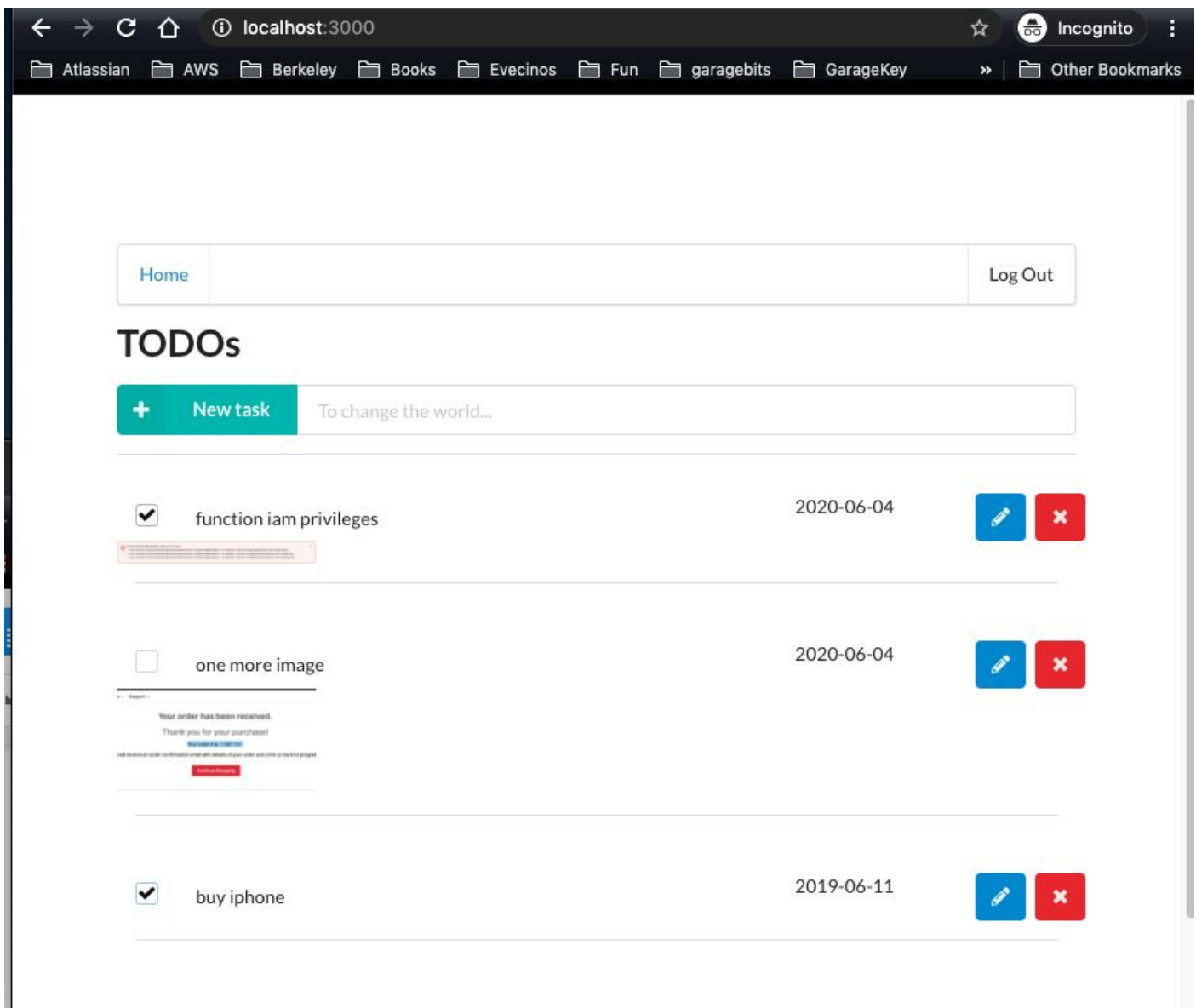
Name: Jaime Valencia
Program: CLOUD DEVELOPER
Project: #4 **Serverless**
User to test with the Front End

User: jaimetest@vaduinc.com
Password: Jaimetest123

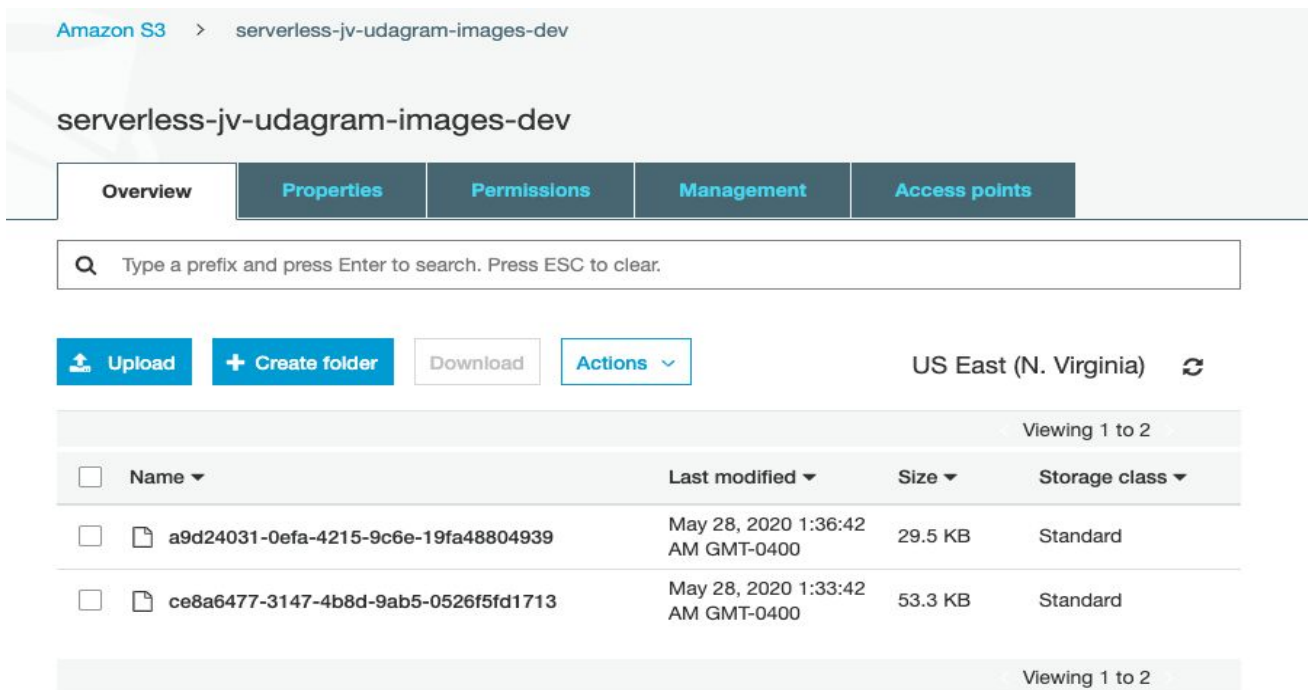
User: anotheruser@vaduinc.com
Password: Anotheruser123

1. Functionality

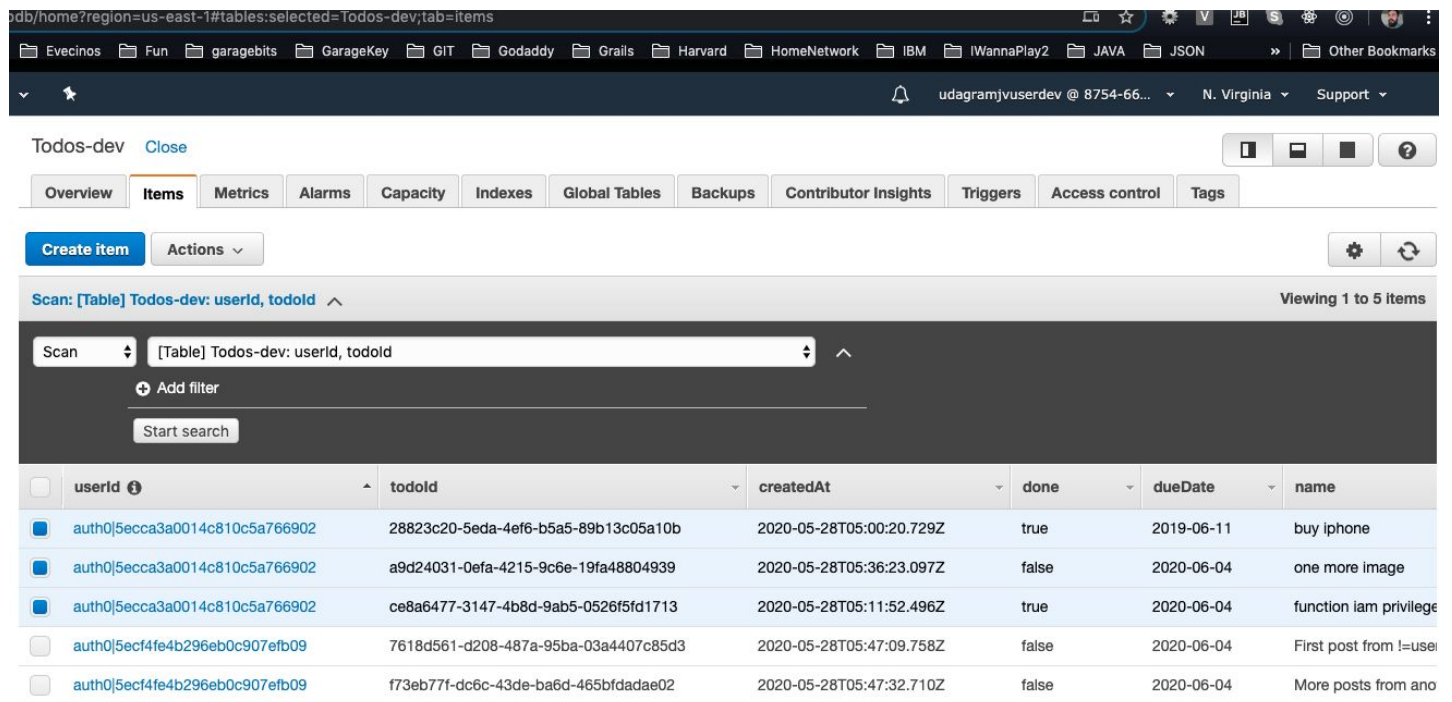
- a. The application allows users to create, update, delete TODO items



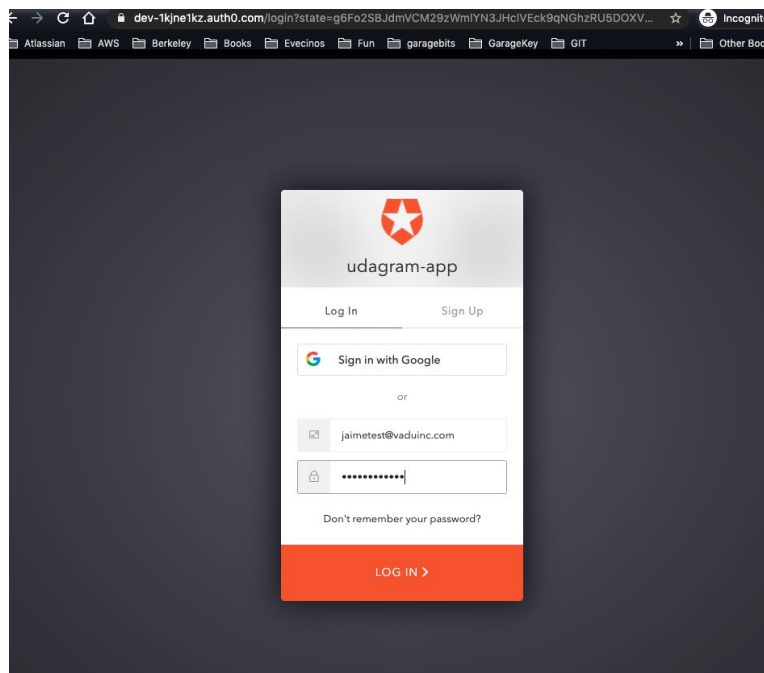
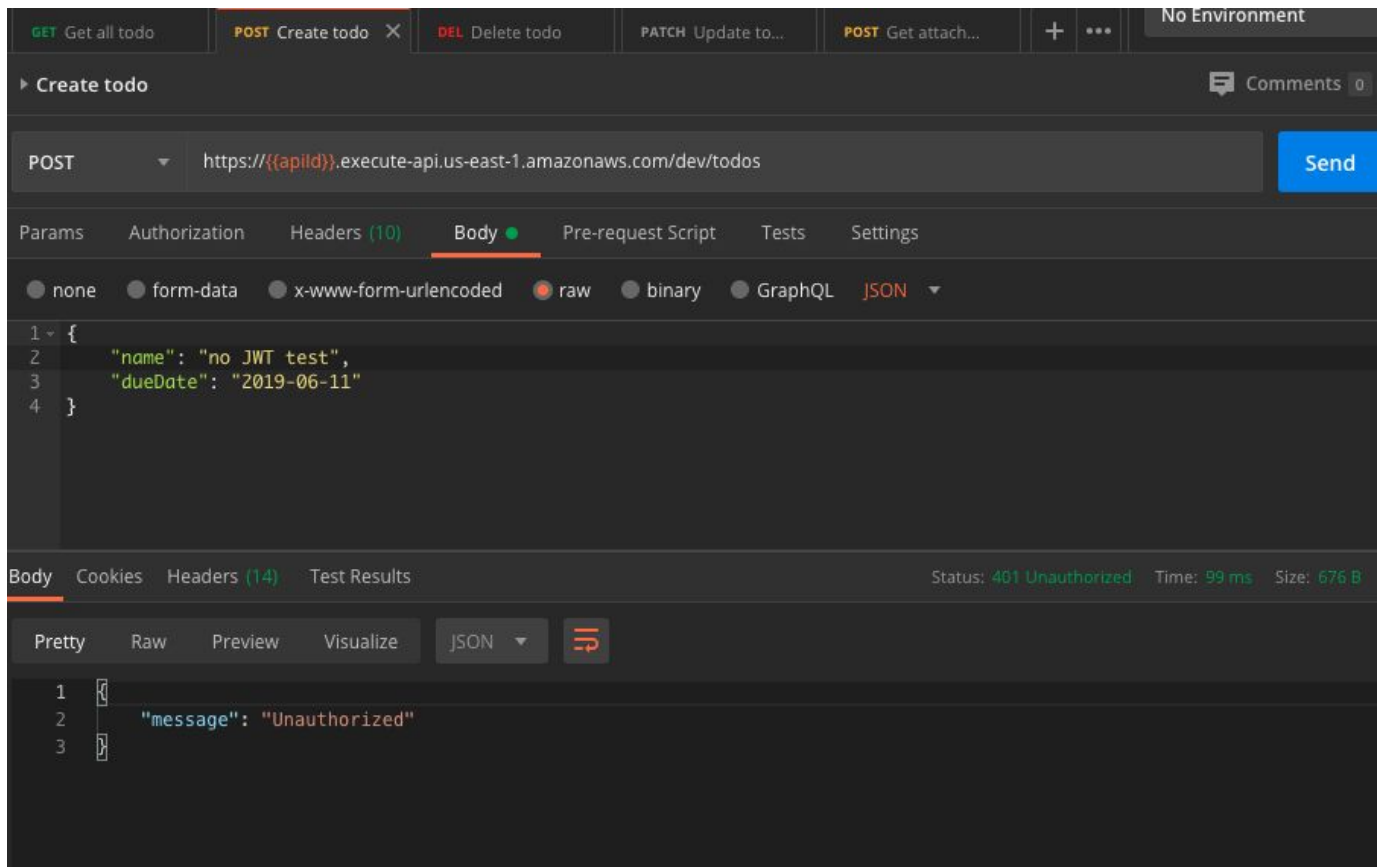
- b. The application allows users to upload a file: see next picture of S3 with uploaded images using the FE application. The images names are the todoid of the items.



- c. The application only displays TODO items for a logged in user: there are 5 todos in the dynamoDB but only 3 are displayed in the FE (see bullet point a. picture)



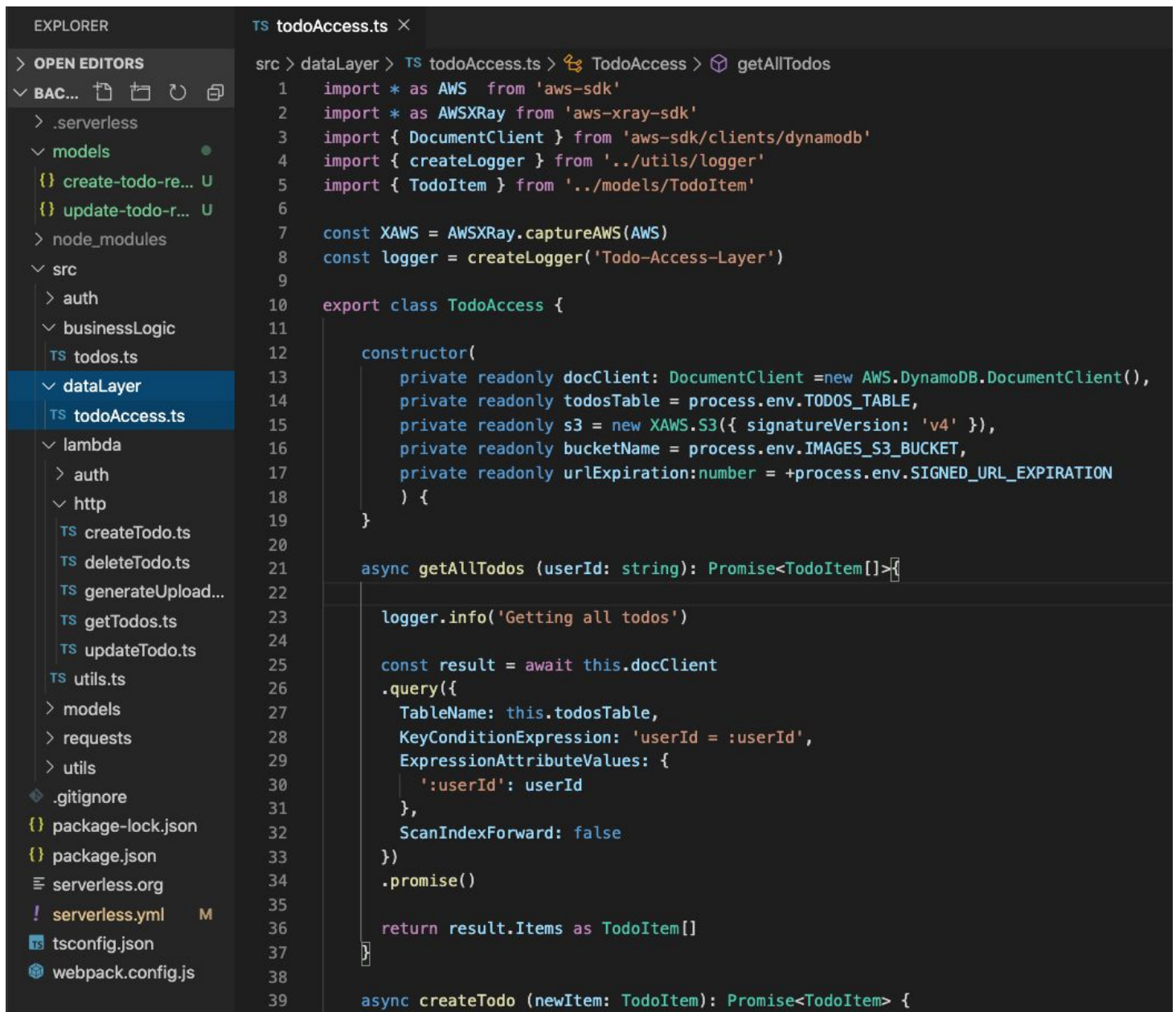
- d. Authentication is implemented and does not allow unauthenticated access: JWT token is required to access the API end-point and Auth0 was set to login into the FE



2. Code Base

a. The code is split into multiple layers separating business logic from I/O related code:

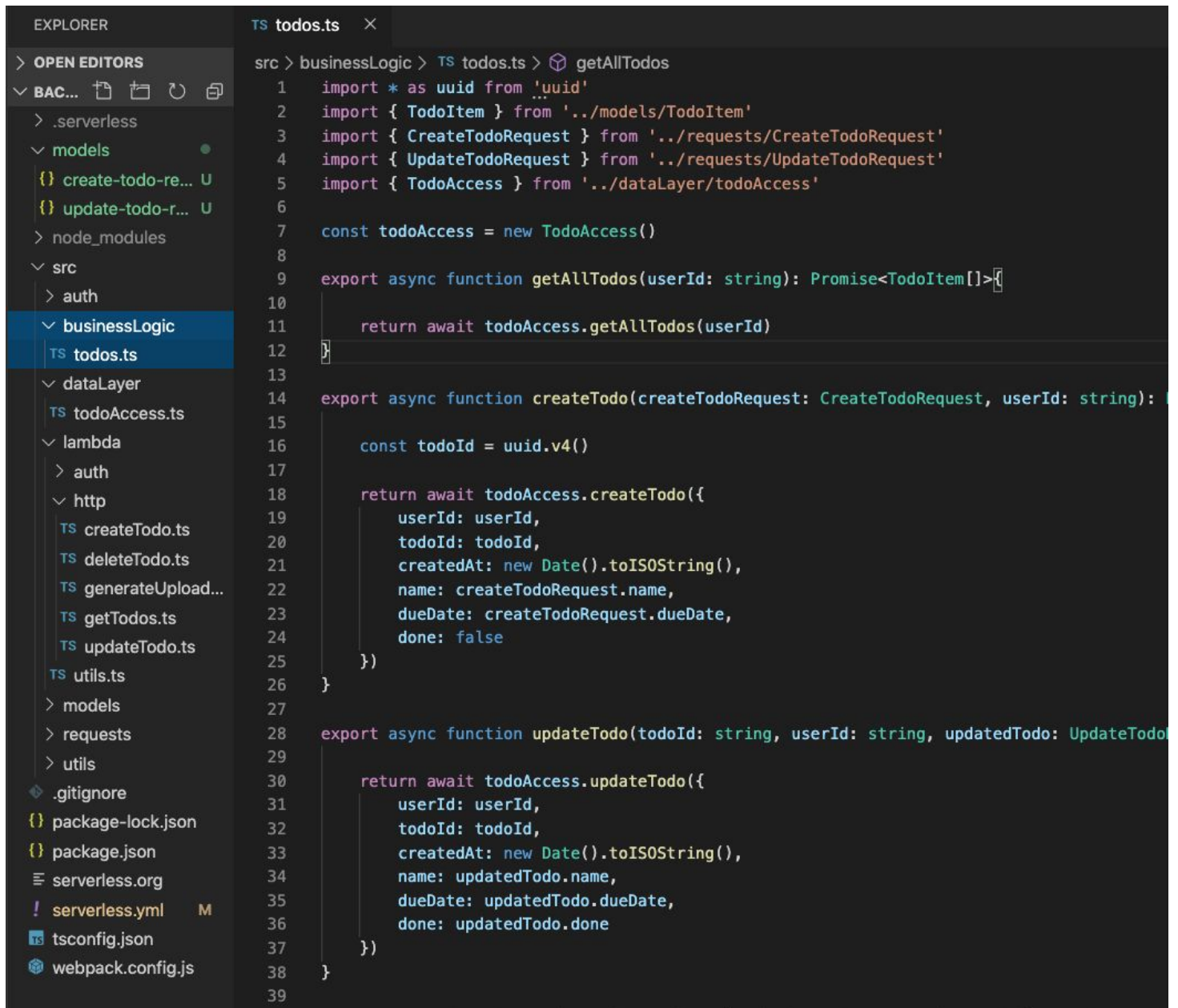
- Data Layer



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure. The 'dataLayer' folder is expanded, showing 'todoAccess.ts' as the active file. The main editor area displays the code for 'todoAccess.ts'. The code is a TypeScript file that imports AWS SDK and AWS X-Ray SDK, initializes a DynamoDB client and an X-Ray logger, and defines a 'TodoAccess' class. The class has a constructor that initializes the DynamoDB client and S3 bucket, and two methods: 'getAllTodos' and 'createTodo'. The 'getAllTodos' method is an asynchronous function that takes a 'userId' string and returns a 'Promise<TodoItem[]>'. It logs the action, queries the DynamoDB table for todos belonging to the user, and returns the results. The 'createTodo' method is also an asynchronous function that takes a 'newItem' of type 'TodoItem' and returns a 'Promise<TodoItem>'. It is currently empty.

```
src > dataLayer > TS todoAccess.ts > TodoAccess > getAllTodos
1  import * as AWS from 'aws-sdk'
2  import * as AWSXRay from 'aws-xray-sdk'
3  import { DocumentClient } from 'aws-sdk/clients/dynamodb'
4  import { createLogger } from '../utils/logger'
5  import { TodoItem } from '../models/TodoItem'
6
7  const XAWS = AWSXRay.captureAWS(AWS)
8  const logger = createLogger('Todo-Access-Layer')
9
10 export class TodoAccess {
11
12     constructor(
13         private readonly docClient: DocumentClient = new AWS.DynamoDB.DocumentClient(),
14         private readonly todosTable = process.env.TODOS_TABLE,
15         private readonly s3 = new XAWS.S3({ signatureVersion: 'v4' }),
16         private readonly bucketName = process.env.IMAGES_S3_BUCKET,
17         private readonly urlExpiration: number = +process.env.SIGNED_URL_EXPIRATION
18     ) {
19     }
20
21     async getAllTodos (userId: string): Promise<TodoItem[]> {
22
23         logger.info('Getting all todos')
24
25         const result = await this.docClient
26             .query({
27                 TableName: this.todosTable,
28                 KeyConditionExpression: 'userId = :userId',
29                 ExpressionAttributeValues: {
30                     ':userId': userId
31                 },
32                 ScanIndexForward: false
33             })
34             .promise()
35
36         return result.Items as TodoItem[]
37     }
38
39     async createTodo (newItem: TodoItem): Promise<TodoItem> {
```

- Business Layer



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like .serverless, models, node_modules, src, and subfolders within src. The file todos.ts is selected under src/businessLogic. The code editor shows the contents of todos.ts, which includes imports for uuid, TodoItem, CreateTodoRequest, UpdateTodoRequest, and TodoAccess. It defines a const todoAccess = new TodoAccess() and exports three async functions: getAllTodos, createTodo, and updateTodo. The getAllTodos function calls todoAccess.getAllTodos. The createTodo function calls todoAccess.createTodo with a new Date() and a new TodoItem. The updateTodo function calls todoAccess.updateTodo with a new Date() and an updated TodoItem.


```
src > businessLogic > TS todos.ts > getAllTodos
1  import * as uuid from 'uuid'
2  import { TodoItem } from '../models/TodoItem'
3  import { CreateTodoRequest } from '../requests/CreateTodoRequest'
4  import { UpdateTodoRequest } from '../requests/UpdateTodoRequest'
5  import { TodoAccess } from '../dataLayer/todoAccess'
6
7  const todoAccess = new TodoAccess()
8
9  export async function getAllTodos(userId: string): Promise<TodoItem[]>{
10
11      return await todoAccess.getAllTodos(userId)
12  }
13
14  export async function createTodo(createTodoRequest: CreateTodoRequest, userId: string): Promise<TodoItem>{
15
16      const todoId = uuid.v4()
17
18      return await todoAccess.createTodo({
19          userId: userId,
20          todoId: todoId,
21          createdAt: new Date().toISOString(),
22          name: createTodoRequest.name,
23          dueDate: createTodoRequest.dueDate,
24          done: false
25      })
26  }
27
28  export async function updateTodo(todoId: string, userId: string, updatedTodo: UpdateTodoRequest): Promise<TodoItem>{
29
30      return await todoAccess.updateTodo({
31          userId: userId,
32          todoId: todoId,
33          createdAt: new Date().toISOString(),
34          name: updatedTodo.name,
35          dueDate: updatedTodo.dueDate,
36          done: updatedTodo.done
37      })
38  }
39
```

- b. Code is implemented using async/await and Promises without using callbacks: I used *nodejs10.x* in the YAML file. There are no callbacks in the code and I used async/await in most of the functions. See above pictures.

3. Best Practices

- a. All resources in the application are defined in the "serverless.yml" file: All resources needed by an application are defined in the "serverless.yml". See file in the code.
- b. Each function has its own set of permissions: See "serverless.yml" file in the code
- c. Application has sufficient monitoring. It has a sufficient amount of log statements

CloudWatch > CloudWatch Logs > Log groups

 **We listened to your feedback!**
In response to your comments on usability, we enhanced our user interface. Please feel free to send us your feedback

Log groups (6)

☐ Ex:

<input type="checkbox"/>	Log group	▲	Retention ▼
<input type="checkbox"/>	/aws/lambda/serverless-todo-app-dev-Auth		Never expire
<input type="checkbox"/>	/aws/lambda/serverless-todo-app-dev-CreateTodo		Never expire
<input type="checkbox"/>	/aws/lambda/serverless-todo-app-dev-DeleteTodo		Never expire
<input type="checkbox"/>	/aws/lambda/serverless-todo-app-dev-GenerateUploadUrl		Never expire
<input type="checkbox"/>	/aws/lambda/serverless-todo-app-dev-GetTodos		Never expire
<input type="checkbox"/>	/aws/lambda/serverless-todo-app-dev-UpdateTodo		Never expire

Details about CreateTodo lambda

[CloudWatch](#) > [CloudWatch Logs](#) > [Log groups](#) > /aws/lambda/serverless-todo-app-dev-CreateTodo

/aws/lambda/serverless-todo-app-dev-CreateTodo

▼ Log group details

Retention	Creation time	Stored bytes
Never expire	17 hours ago	-
KMS key ID	Metric filters	Subscriptions
-	0	-

Log streams

Metric filters

Contributor Insights

Log streams (4)

<input type="checkbox"/>	Log stream
<input type="checkbox"/>	2020/05/28/[\$LATEST]934e0521372a4a9992f0417811c5dd04
<input type="checkbox"/>	2020/05/28/[\$LATEST]cfdbb8c0123c478c80560c74e6e1100f
<input type="checkbox"/>	2020/05/28/[\$LATEST]1b16cf76aafe403d844c310c9cad698e
<input type="checkbox"/>	2020/05/28/[\$LATEST]e342c668aeb34a0b9f378146b32b7148

- d. HTTP requests are validated. See next validation schemas created for requests:

Create-todo-request.json

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "createTodo",
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "dueDate": {
      "type": "string"
    }
  },
  "required": [
    "name",
    "dueDate"
  ],
  "additionalProperties": false
}
```

Update-todo-request.json

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "updateTodo",
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "dueDate": {
      "type": "string"
    },
    "done": {
      "type": "boolean"
    }
  },
  "required": [
    "name",
    "dueDate",
    "done"
  ],
  "additionalProperties": false
}
```


See next image. It is a test to add a TODO item without a *dueDate*. A return 400 error is received.

The screenshot displays the Swagger UI for a REST API. The top navigation bar includes buttons for 'GET Get all todo', 'POST Create todo', 'DEL Delete todo', 'PATCH Update to...', and 'POST Get attach...'. The 'POST Create todo' button is selected, showing the endpoint `https://(apidj).execute-api.us-east-1.amazonaws.com/dev/todos`. The 'Send' button is visible.

The 'Body' tab is active, showing the request body in JSON format:

```
1 {  
2   "name": "buy iphone"  
3 }
```

The 'Test Results' section shows the response status and details:

- Status: 400 Bad Request
- Time: 1003 ms
- Size: 682 B

The 'Save Response' button is visible. The response body is shown in the 'Pretty' tab:

```
1 {  
2   "message": "Invalid request body"  
3 }
```

Then tried it again adding the dueDate field. A successful response is received. See next image.

The screenshot displays a REST client interface with a tab titled "Create todo". The request is a POST to the URL `https://{{apiId}}.execute-api.us-east-1.amazonaws.com/dev/todos`. The request body is a JSON object: `{ "name": "buy iphone", "dueDate": "2019-06-11" }`. The response status is "201 Created" with a size of "720 B". The response body is a JSON object: `{ "item": { "userId": "auth0|5ecca3a0014c810c5a766902", "todoId": "28823c20-5eda-4ef6-b5a5-89b13c05a10b", "createdAt": "2020-05-28T05:00:20.729Z", "name": "buy iphone", "dueDate": "2019-06-11", "done": false } } }`.

GET Get all todo POST Create todo DEL Delete todo PATCH Update to... POST Get attach... + ... No Environment

► Create todo Comments 0

POST `https://{{apiId}}.execute-api.us-east-1.amazonaws.com/dev/todos` Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▼

```
1 {
2   "name": "buy iphone",
3   "dueDate": "2019-06-11"
4 }
```

Body Cookies Headers (12) Test Results Status: 201 Created Time: -- Size: 720 B

Pretty Raw Preview Visualize JSON ▼ ↺

```
1 {
2   "item": {
3     "userId": "auth0|5ecca3a0014c810c5a766902",
4     "todoId": "28823c20-5eda-4ef6-b5a5-89b13c05a10b",
5     "createdAt": "2020-05-28T05:00:20.729Z",
6     "name": "buy iphone",
7     "dueDate": "2019-06-11",
8     "done": false
9   }
10 }
```

4. Architecture

- a. Data is stored in a table with a composite key. See the following TABLE definition from the *serverless.yml* file definition.

```
TodosTable:
  Type: AWS::DynamoDB::Table
  Properties:
    AttributeDefinitions:
      - AttributeName: userId
        AttributeType: S
      - AttributeName: todoId
        AttributeType: S
      - AttributeName: dueDate
        AttributeType: S
    KeySchema:
      - AttributeName: userId
        KeyType: HASH
      - AttributeName: todoId
        KeyType: RANGE
    BillingMode: PAY_PER_REQUEST
    TableName: ${self:provider.environment.TODOS_TABLE}
    LocalSecondaryIndexes:
      - IndexName: ${self:provider.environment.INDEX_NAME}
        KeySchema:
          - AttributeName: userId
            KeyType: HASH
          - AttributeName: dueDate
            KeyType: RANGE
        Projection:
          ProjectionType: ALL
```

- b. Scan operation is not used to read data from a database. Only a query() call was used.