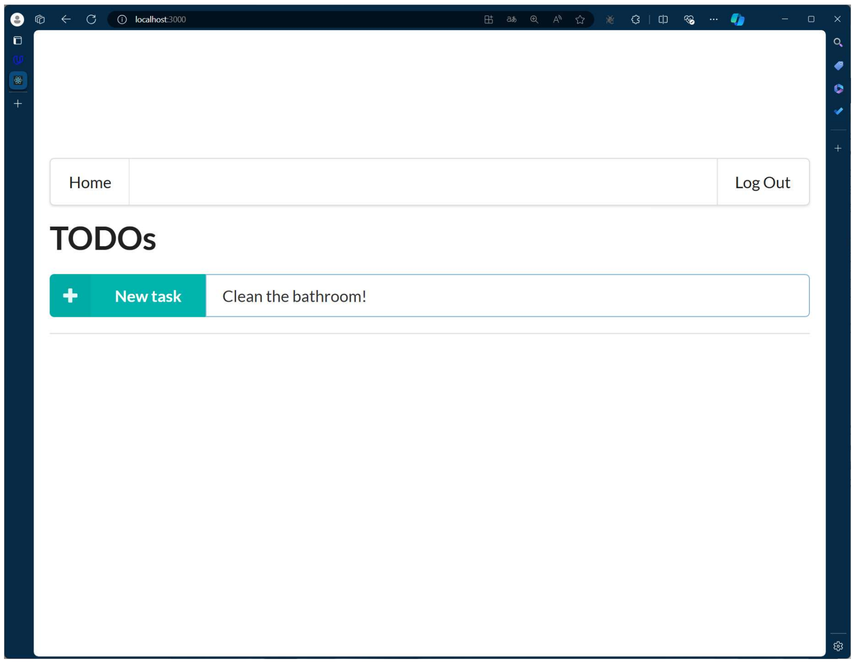
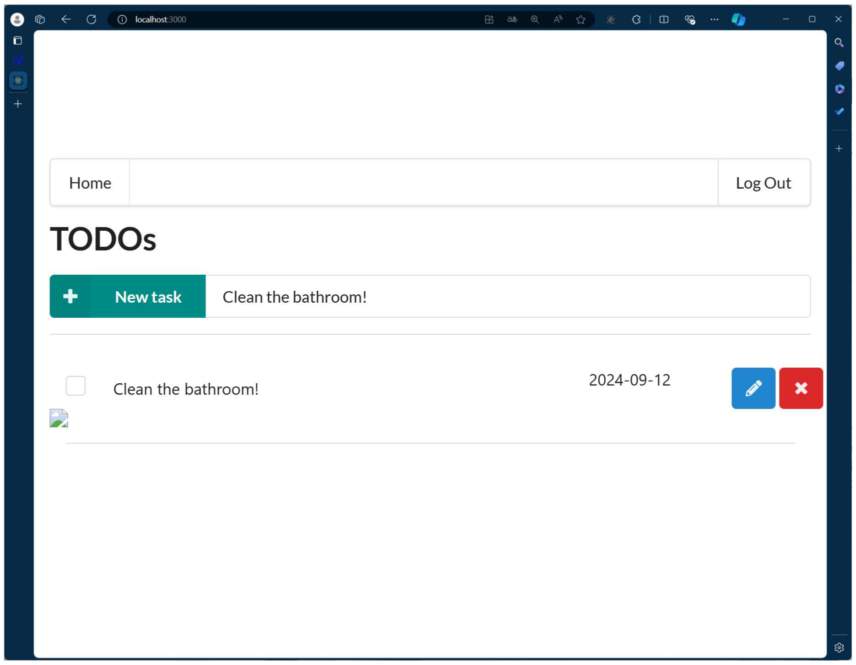
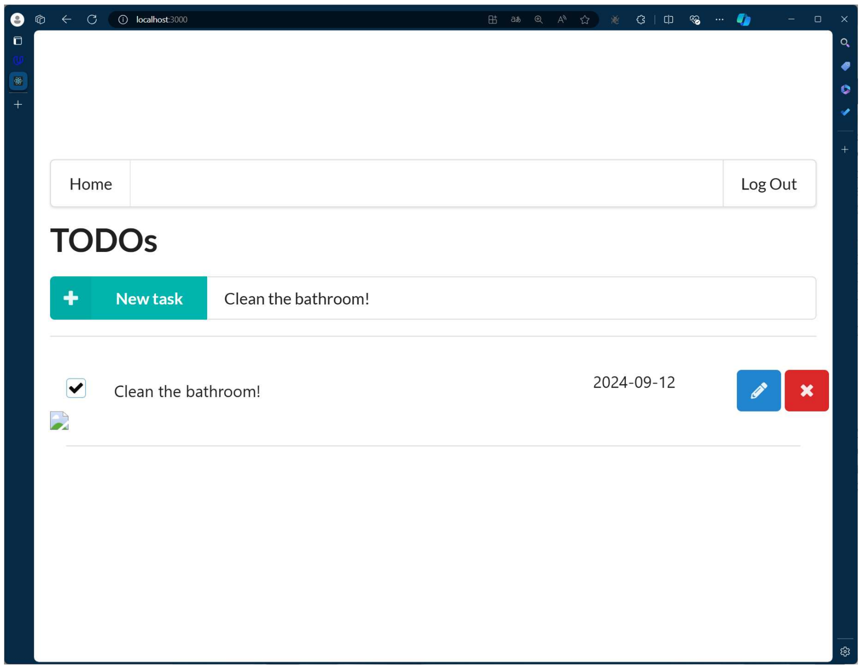
Project: Serverless Application

1) Functionality
- The application allows users to create, update, delete TODO items
  - A user of the web application can use the interface to create, delete and complete a TODO item.
  - Result:
    - Create todo





    - Complete todo

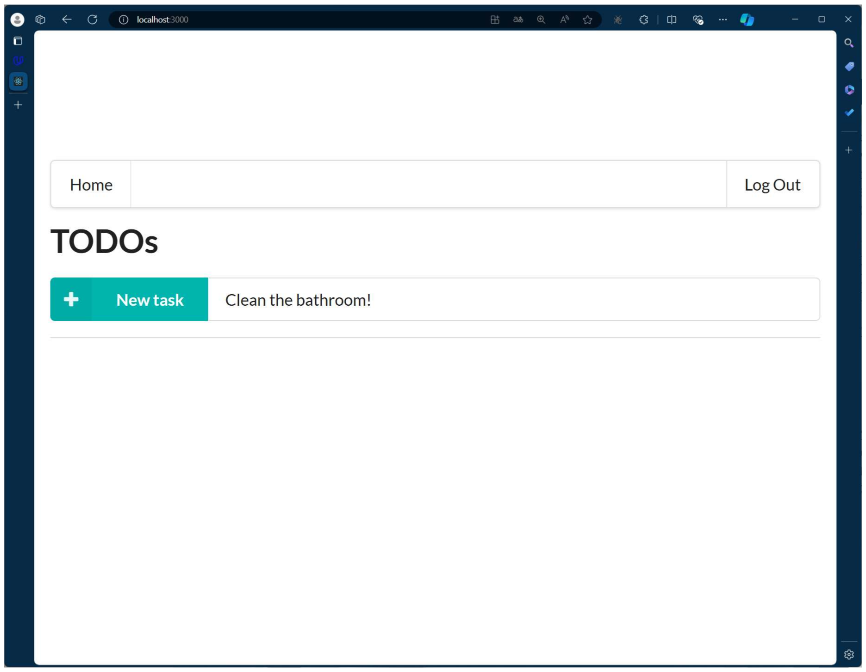

localhost:3000

TODOs

Clean the bathroom!

Clean the bathroom! 2024-09-12

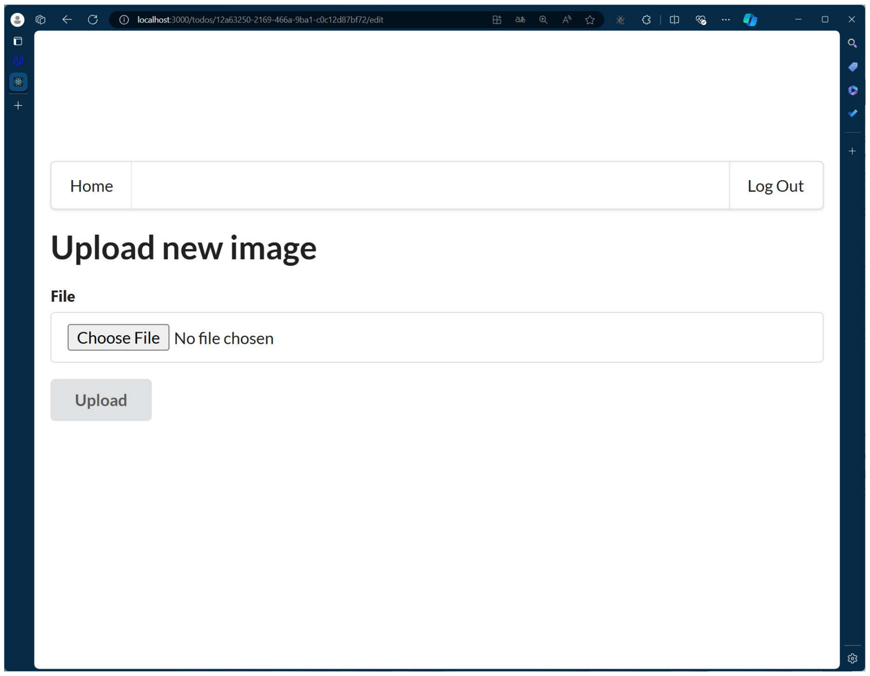
- Delete todo


localhost:3000

TODOs

Clean the bathroom!

- The application allows users to upload a file.
  - A user of the web interface can click on a "pencil" button, then select and upload a file. A file should appear in the list of TODO items on the home page.
  - Result:
    - Click on a "pencil" button

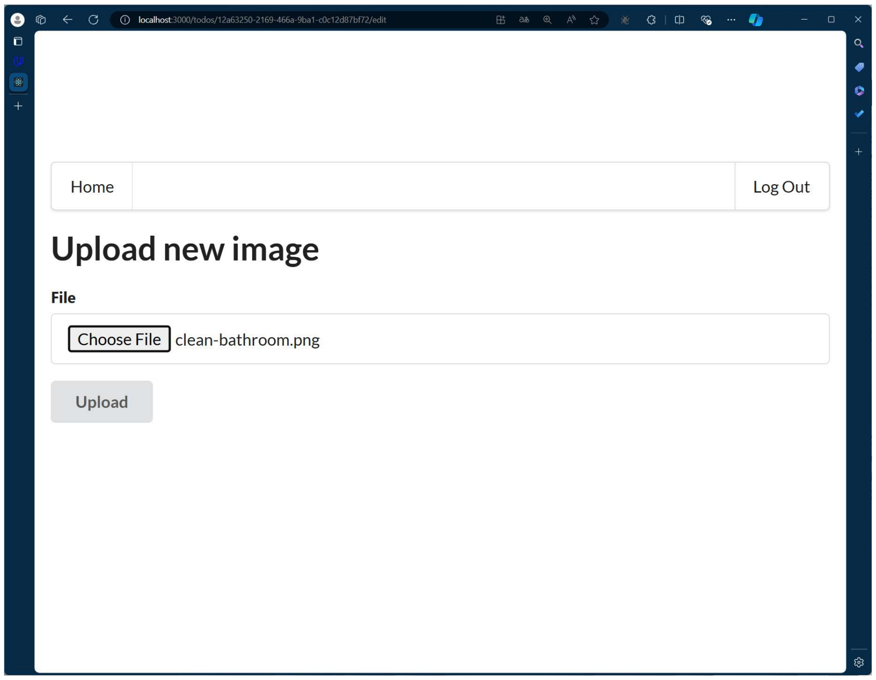

localhost:3000/todos/12a63250-2169-466a-9ba1-c0c12d87bf72/edit

Upload new image

File

No file chosen

- Choose file

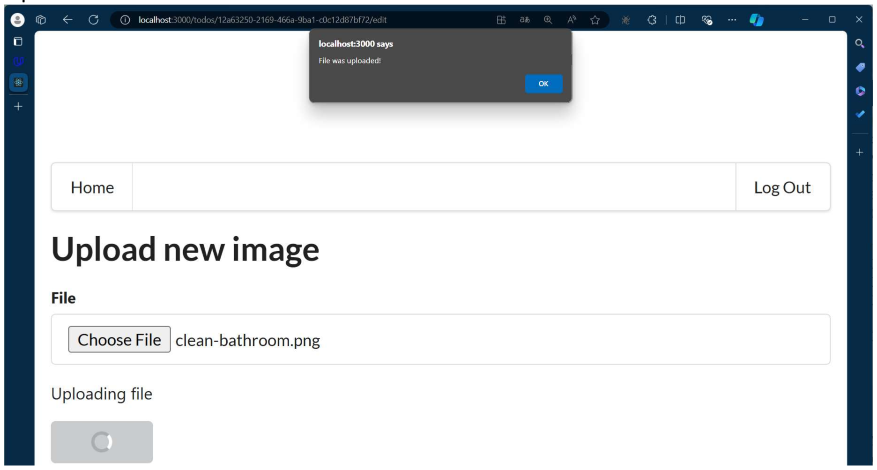

localhost:3000/todos/12a63250-2169-466a-9ba1-c0c12d87bf72/edit

Upload new image

File

clean-bathroom.png

- Upload


localhost:3000/todos/12a63250-2169-466a-9ba1-c0c12d87bf72/edit

localhost:3000 says

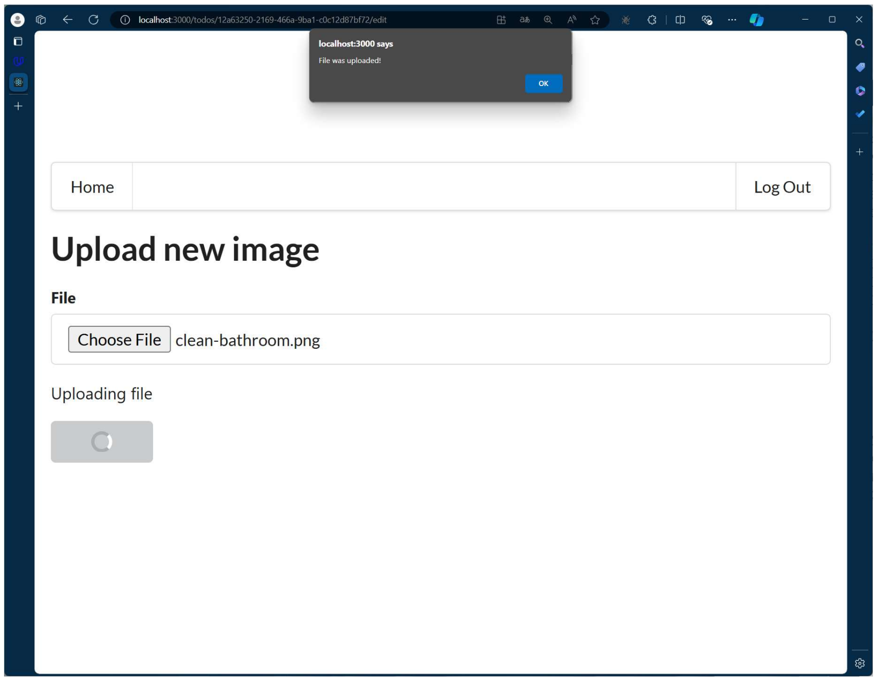
File was uploaded!

Upload new image

File

clean-bathroom.png

Uploading file


localhost:3000/todos/12a63250-2169-466a-9ba1-c0c12d87bf72/edit

localhost:3000 says

File was uploaded!

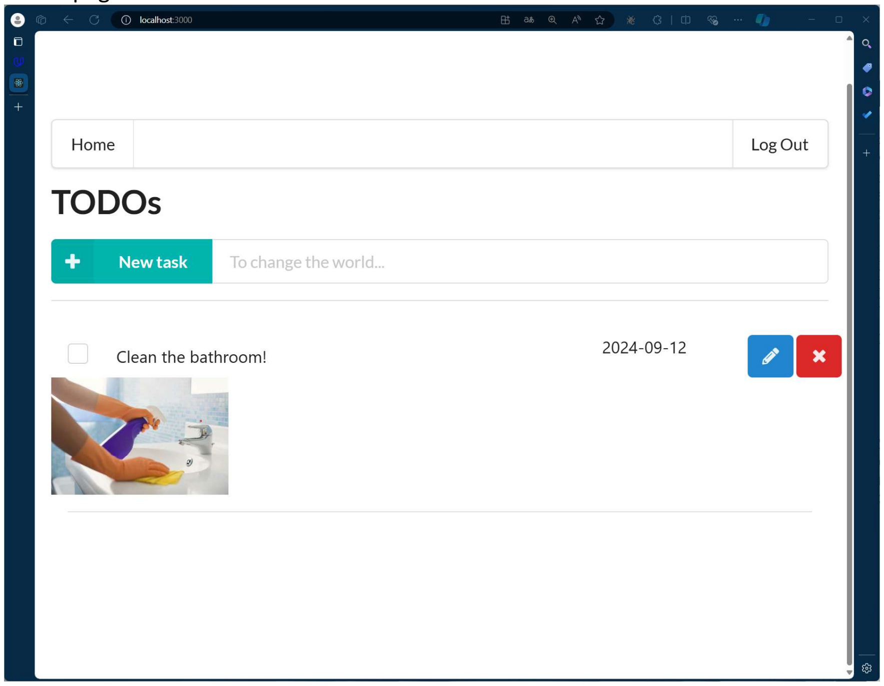
Upload new image

File

clean-bathroom.png

Uploading file

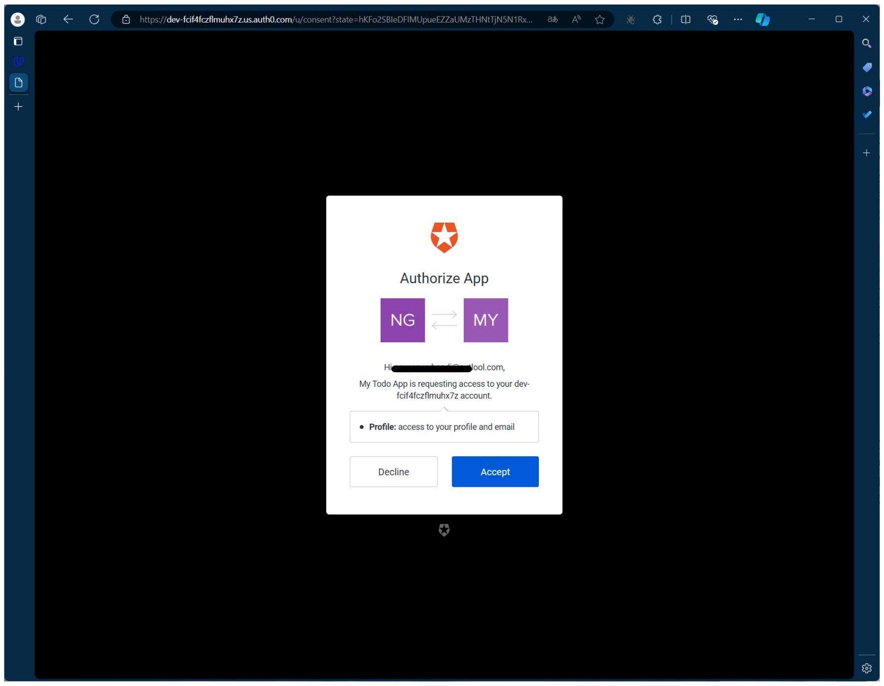
- Home page


localhost:3000

TODOs

Clean the bathroom! 2024-09-12

- The application only displays TODO items for a logged in user.
  - If you log out from a current user and log in as a different user, the application should not show TODO items created by the first account.
  - Result:
    - Login with orther Account



https://dev-fcif4fczflmuhx7z.us.auth0.com/u/consent?state=hKFo2SBleDFlMUpueEZZaUMzTHNtTjN5N1Rx...

Authorize App

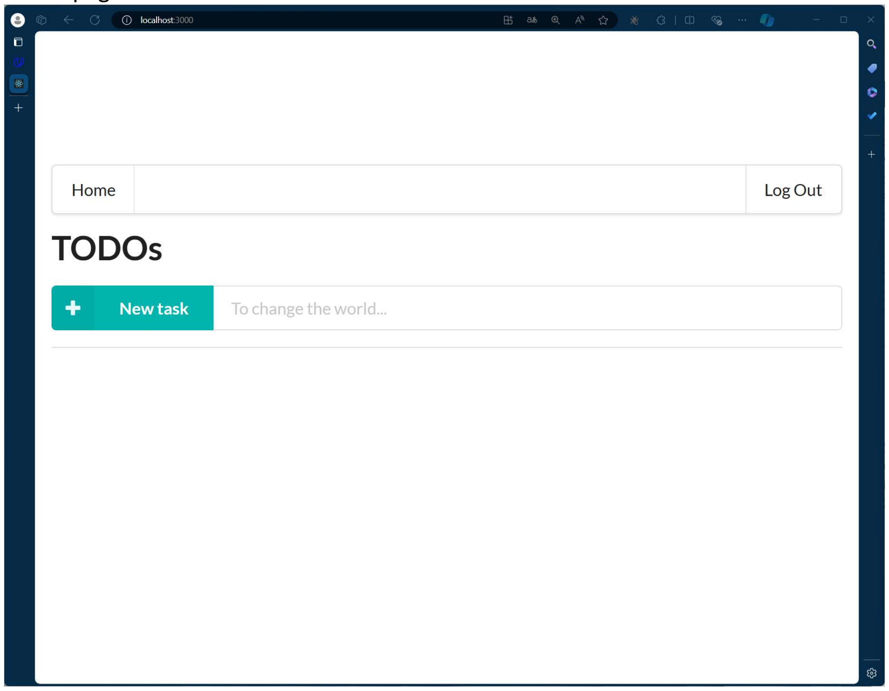
NG MY

Hi [redacted]ool.com,

My Todo App is requesting access to your dev-fcif4fczflmuhx7z account.

- **Profile:** access to your profile and email

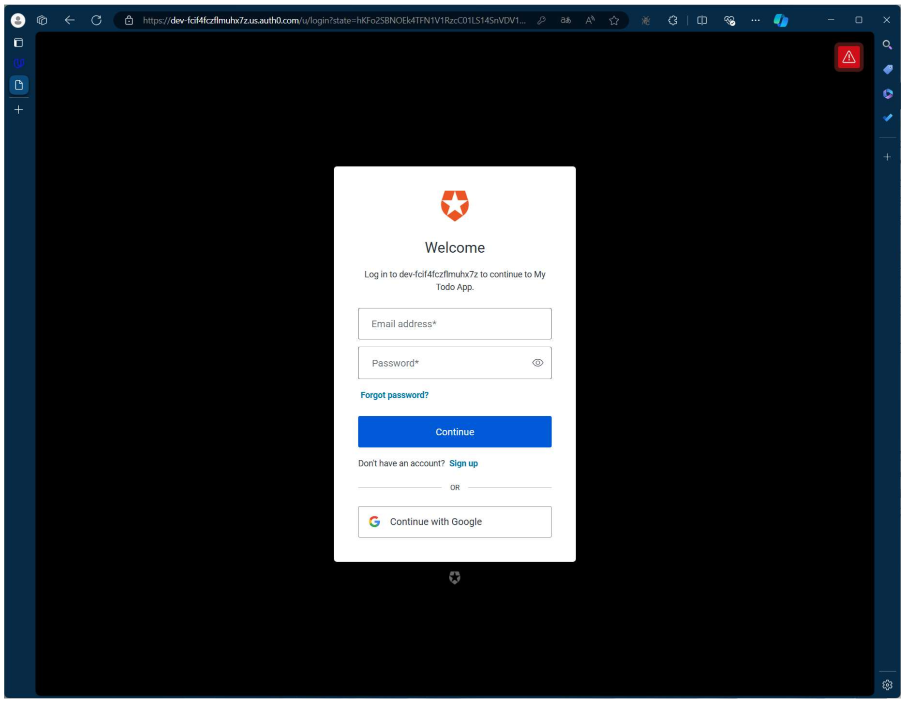
        ▪ Home page



localhost:3000

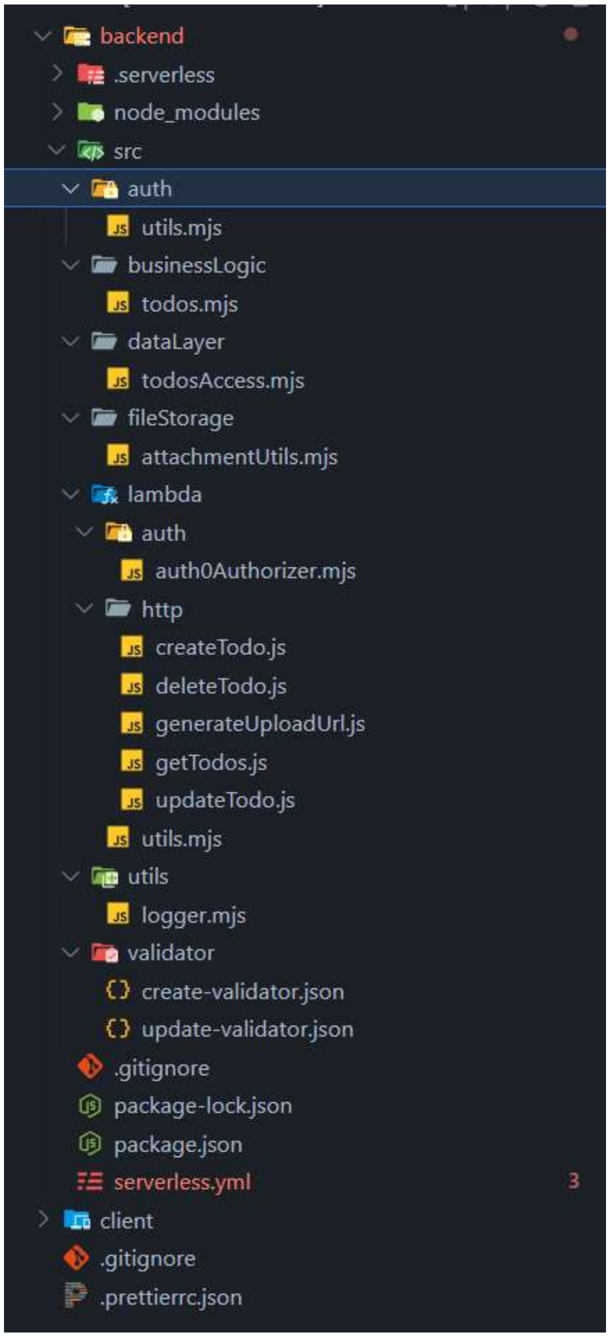
TODOs

- Authentication is implemented and does not allow unauthenticated access.
    - A user needs to authenticate in order to use an application.
    - Result:
        - Need login


https://dev-fcif4fczflmuhx7z.us.auth0.com/u/login?state=hKFo2SBNOEk4TFN1V1RzcC01LS14SnVDV1...
Welcome
Log in to dev-fcif4fczflmuhx7z to continue to My Todo App.
Email address*
Password*
Forgot password?
Continue
Don't have an account? Sign up
OR
Continue with Google

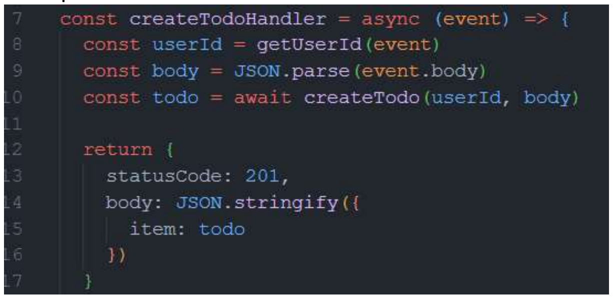
2) Code Base

- The code is split into multiple layers separating business logic from I/O related code.
    - Code of Lambda functions is split into multiple files/classes. The business logic of an application is separated from code for database access, file storage, and code related to AWS Lambda.
    - Result:
        - Diretory structure



```
backend
  .serverless
  node_modules
  src
    auth
      utils.mjs
    businessLogic
      todos.mjs
    dataLayer
      todosAccess.mjs
    fileStorage
      attachmentUtils.mjs
    lambda
      auth
        auth0Authorizer.mjs
      http
        createTodo.js
        deleteTodo.js
        generateUploadUrl.js
        getTodos.js
        updateTodo.js
      utils.mjs
    utils
      logger.mjs
    validator
      create-validator.json
      update-validator.json
  .gitignore
  package-lock.json
  package.json
  serverless.yml                                  3
client
.gitignore
.prettierrc.json
```

- Code is implemented using async/await and Promises without using callbacks.
  - To get results of asynchronous operations, a student is using async/await constructs instead of passing callbacks.
  - Result: Please refer git repository
    - Example:



```
const createTodoHandler = async (event) => {
  const userId = getUserId(event)
  const body = JSON.parse(event.body)
  const todo = await createTodo(userId, body)

  return {
    statusCode: 201,
    body: JSON.stringify({
      item: todo
    })
  }
```

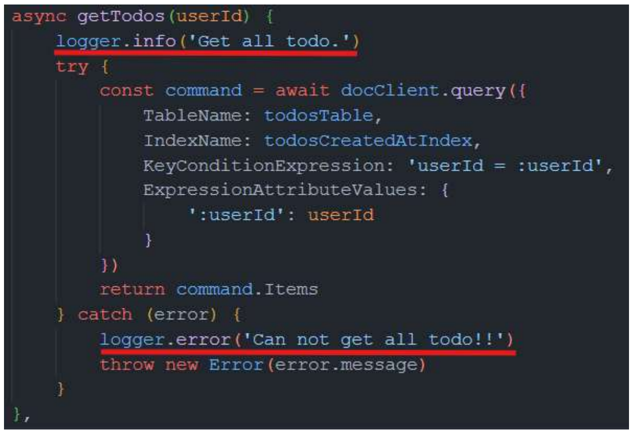
3) Best Practices

- All resources in the application are defined in the "serverless.yml" file
  - All resources needed by an application are defined in the "serverless.yml". A developer does not need to create them manually

using AWS console.
    - Result: Please refer git repository

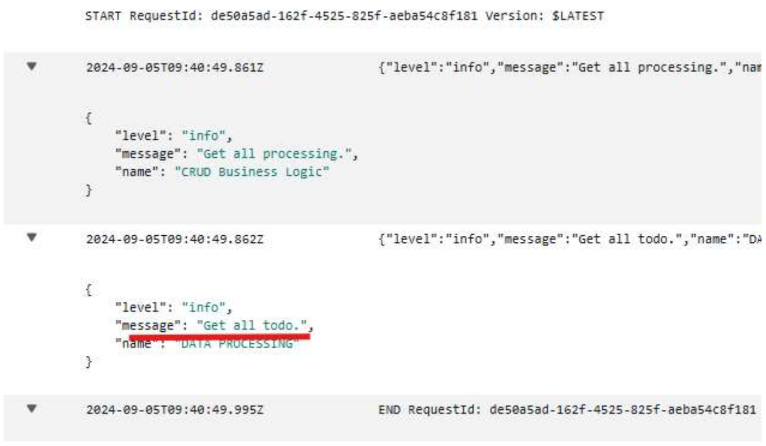
- Each function has its own set of permissions.
    - Instead of defining all permissions under provider/iamRoleStatements, permissions are defined per function in the functions section of the "serverless.yml".
    - Result: Please refer git repository

- Application has sufficient monitoring.
    - Application has at least some of the following:
        - Distributed tracing is enabled
        - It has a sufficient amount of log statements
        - It generates application level metrics
    - Result:
        - Code:



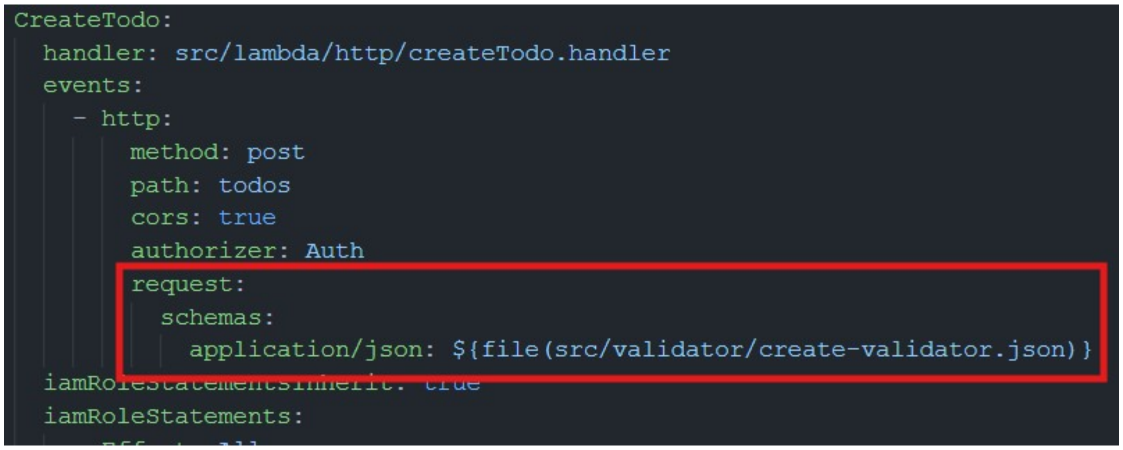
```
async getTodos(userId) {
    logger.info('Get all todo.')
    try {
        const command = await docClient.query({
            TableName: todosTable,
            IndexName: todosCreatedAtIndex,
            KeyConditionExpression: 'userId = :userId',
            ExpressionAttributeValues: {
                ':userId': userId
            }
        })
        return command.Items
    } catch (error) {
        logger.error('Can not get all todo!!')
        throw new Error(error.message)
    }
},
```

        - CloudWatch Logs:



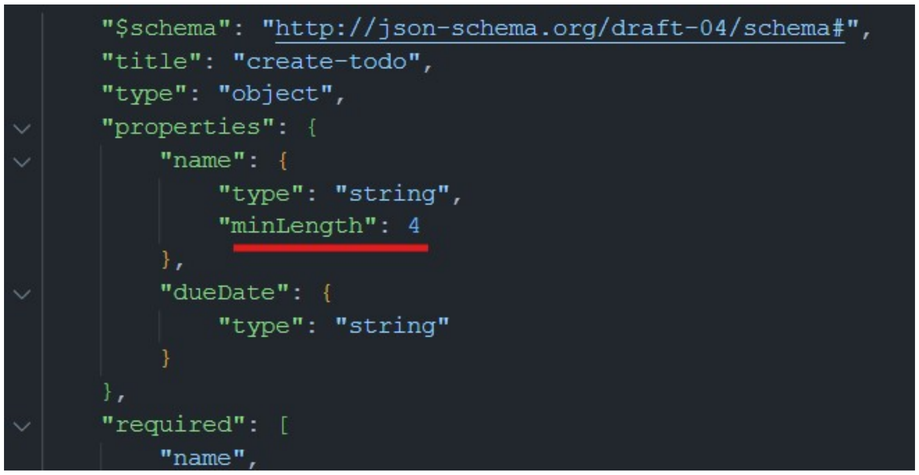
```
START RequestId: de50a5ad-162f-4525-825f-aeba54c8f181 Version: $LATEST

▼   2024-09-05T09:40:49.861Z    {"level":"info","message":"Get all processing.","nar

    {
        "level": "info",
        "message": "Get all processing.",
        "name": "CRUD Business Logic"
    }

▼   2024-09-05T09:40:49.862Z    {"level":"info","message":"Get all todo.","name":"DA

    {
        "level": "info",
        "message": "Get all todo.",
        "name": "DATA PROCESSING"
    }

▼   2024-09-05T09:40:49.995Z    END RequestId: de50a5ad-162f-4525-825f-aeba54c8f181
```

- HTTP requests are validated
    - Incoming HTTP requests are validated either in Lambda handlers or using request validation in API Gateway. The latter can be done either using the serverless-reqvalidator-plugin or by providing request schemas in function definitions.
    - Result:
        - Serverless definition:



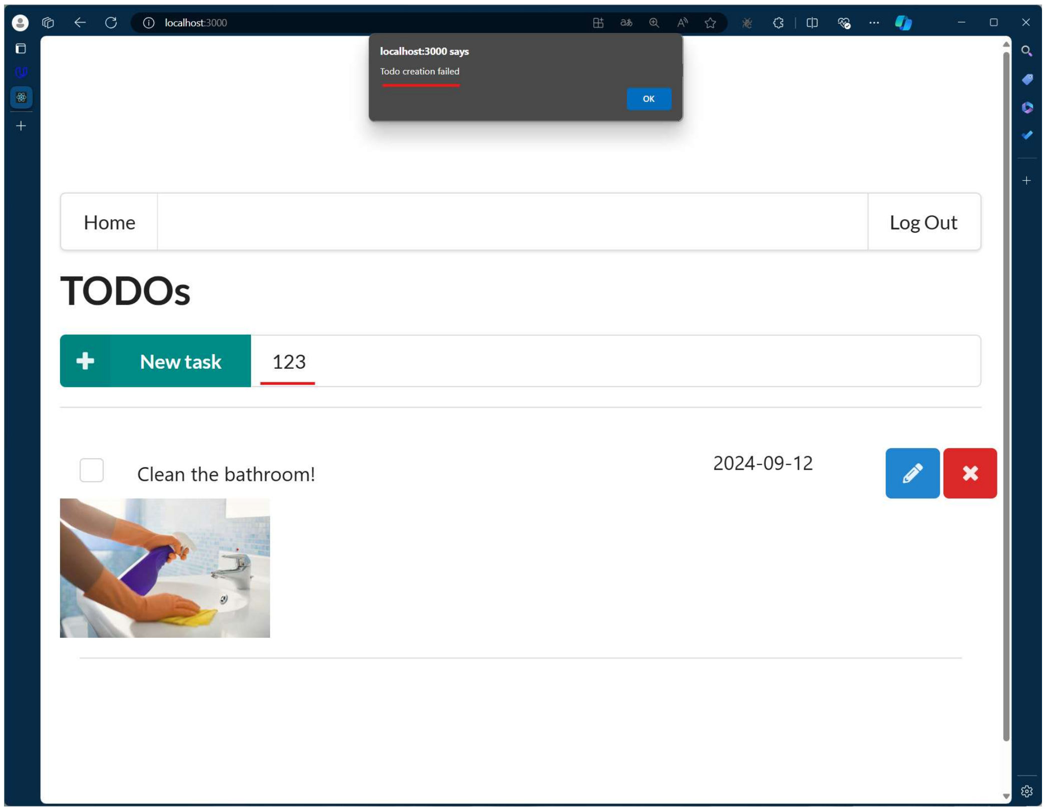
```
CreateTodo:
  handler: src/lambda/http/createTodo.handler
  events:
    - http:
        method: post
        path: todos
        cors: true
        authorizer: Auth
        request:
          schemas:
            application/json: ${file(src/validator/create-validator.json)}
  iamRoleStatementsInherit: true
  iamRoleStatements:
```

- Schema:



```
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "create-todo",
    "type": "object",
    "properties": {
        "name": {
            "type": "string",
            "minLength": 4
        },
        "dueDate": {
            "type": "string"
        }
    },
    "required": [
        "name",
```

- Validation:



localhost:3000 says

Todo creation failed

OK

Home | Log Out

TODOs

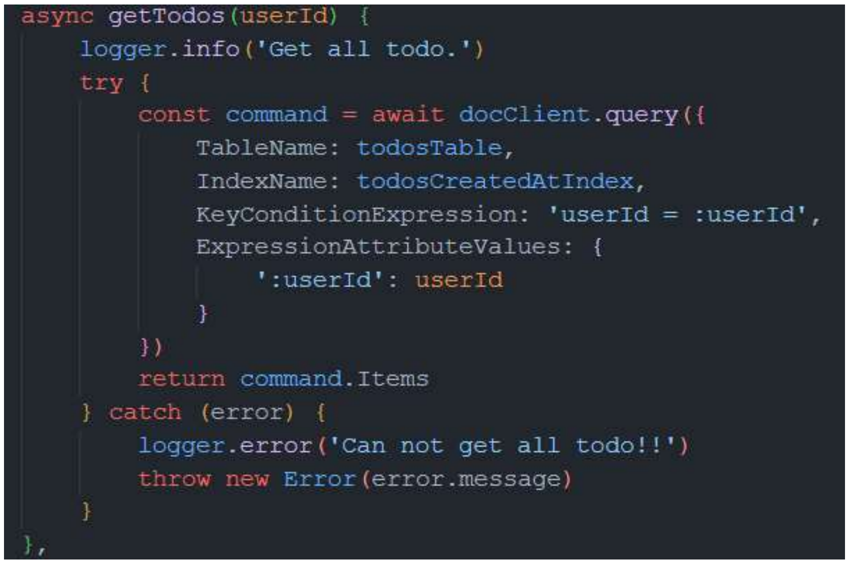
New task | 123

Clean the bathroom! 2024-09-12

4) Architecture

- Data is stored in a table with a composite key.
    - 1:M (1 to many) relationship between users and TODO items is modeled using a DynamoDB table that has a composite key with both partition and sort keys.
    - Result: Please refer git repository

- Scan operation is not used to read data from a database.
    - TODO items are fetched using the "query()" method and not "scan()" method (which is less efficient on large datasets)
    - Result: Please refer git repository



```
async getTodos(userId) {
    logger.info('Get all todo.')
    try {
        const command = await docClient.query({
            TableName: todosTable,
            IndexName: todosCreatedAtIndex,
            KeyConditionExpression: 'userId = :userId',
            ExpressionAttributeValues: {
                ':userId': userId
            }
        })
        return command.Items
    } catch (error) {
        logger.error('Can not get all todo!!')
        throw new Error(error.message)
    }
},
```