UDACITY

ALX Nanodegree “Cloud Developer”

Week 9 Activity

Student: Papa Mbaye TINE

P4 Serverless Application

Table of Content

I. Overview – Serverless application

In this project you will develop and deploy a simple "TODO" application using AWS Lambda and Serverless framework. This application will allow users to create/remove/update/get TODO items. Each TODO item contains the following fields:

- `todoId` **(string) - a unique id for an item**
- `createdAt` **(string) - date and time when an item was created**
- `name` **(string) - name of a TODO item (e.g. "Change a light bulb")**
- `dueDate` **(string) - date and time by which an item should be completed**
- `done` **(boolean) - true if an item was completed, false otherwise**
- `attachmentUrl` **(string) (optional) - a URL pointing to an image attached to a TODO item**

You might also store an `id` of a user who created a TODO item. Each TODO item can optionally have an attachment image. Each user only has access to TODO items that he/she has created.

A. Prerequisites

You should have the following tools installed in your local machine:

- **Auth0 account**
- **GitHub account**
- **NodeJS version up to 12.xx**
- **Serverless**

2. The following tools will help you run your project locally as a monolithic application.

- **PostgreSQL client, the `psql` command line utility, installed locally. Using PostgreSQL involves a server and a client. The server hosts the database while the client interfaces with it to execute queries. Because we will be creating our server on AWS, we will only need to install a client for our local setup. The easiest way to set this up is with the PostgreSQL Installer. This installer installs a PostgreSQL client in the form of the `psql` command-line utility. You can see the complete (server and client) installation instructions for Mac, Linux, and Windows. Verify using:**

```
# Preferred v12.x to v13.x
psql --version
```

- **NodeJS v12.14 or greater up to v14.15 - Node.js is used to run JavaScript-based applications and NPM is a package manager used to**

handle dependencies. NodeJS installer will install both Node.js and npm on your system. Verify the installation using the commands:

```
# v12.14 to v14.15
node -v
# v6.14 to v7.19
npm -v
```

- Ionic command-line utility v6 framework to build and run the frontend application locally. In general, Ionic Framework is used to make cross-platform applications using JavaScript. Verify the installation as:

```
# v6.xx
ionic --version
```

3. Docker Desktop for running the project locally in a multi-container environment.
4. AWS CLI v2 for interacting with AWS services via your terminal. After installing the AWS CLI, you will also have to configure the access profile locally.
    - Create an IAM user with Admin privileges on the AWS web console. Copy its Access key.
    - Configure the access profile locally using the Access key generated above:

```
aws configure
# Run a sample command
aws iam list-users
```

5. Kubectl command-line utility to communicate with Kubernetes clusters

B. Starter Code

In addition to the tools above, fork and then clone the project starter code from the https://github.com/udacity/cloud-developer/tree/master/course-04/project/c4-final-project-starter-code

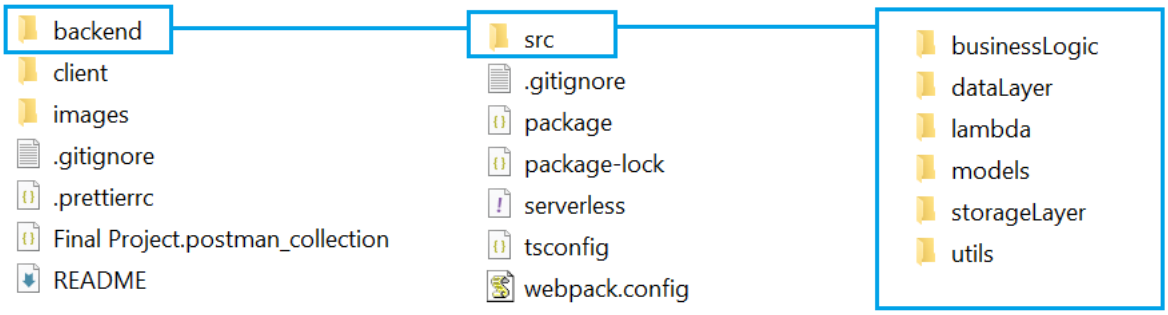
II. Getting Started

The following sections will show the screenshots the completion of the project.

A. Modify Backend code

The backend code is modified according to indications in the project page, updating files with "TODO".

The backend Code Base is structured is into several sub-directories for best practices:



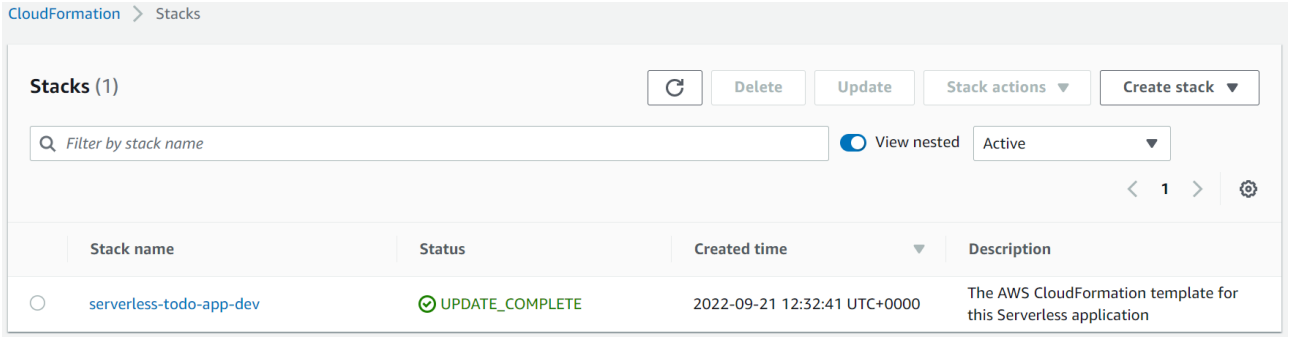
B. Backend deployment

After successful change in the backend, we proceed to the deployment by using following commands in project directory:

```
cd backend
npm update --save
npm audit fix
# For the first time, create an application in your org in Serverless portal
serverless
# Next time, deploy the app and note the endpoint url in the end
serverless deploy --verbose
# If you face a permissions error, you may need to specify the user profile
serverless deploy -v --aws-profile default
# sls is shorthand for serverless
# -v is shorthand for --verbose
```

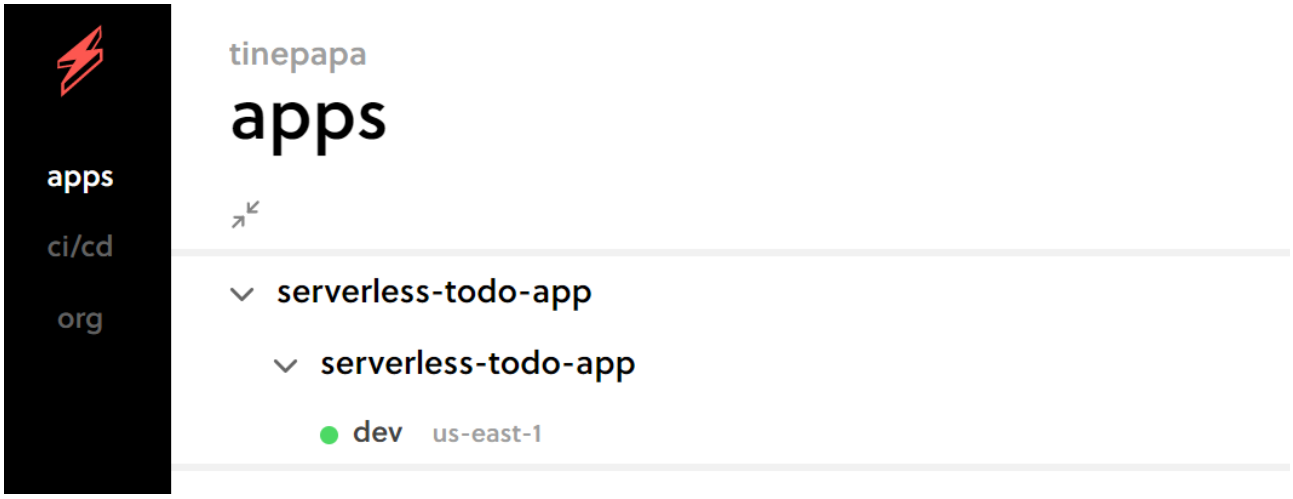
We used default AWS profile.

The result of the deployment:



A successful deployment will create a resource stack in the CloudFormation console

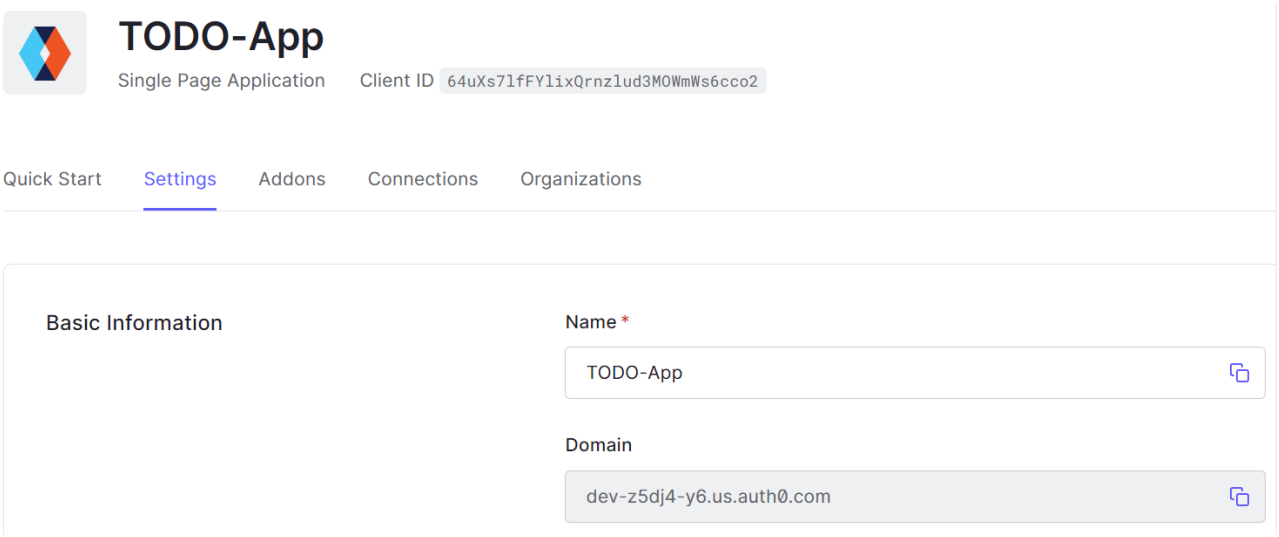
We have the app deployment status in serverless dashboard:



C. Frontend configuration

The /client/ folder contains the frontend web application which consumes the backend API developed in this project. You don't need to make any changes to the frontend code in the /client/ folder, except for the Authentication related changes, as explained below.

- **Authentication - Login to the Auth0 portal, and navigate to your Dashboard.**
  - **Create a "Single Page Web Applications" type Auth0 application**
  - **Go to the App settings, and setup the Allowed Callback URLs**
  - **Setup the Allowed Web Origins for CORS options.**
  - **Setup the application properties. We recommend using asymmetrically encrypted (RS256) JWT tokens.**
  - **Copy "domain" and "client id" to save in the `/client/src/config.ts` file.**
  - **In your backend auth handler function, fetch the Auth0 certificate programmatically.**

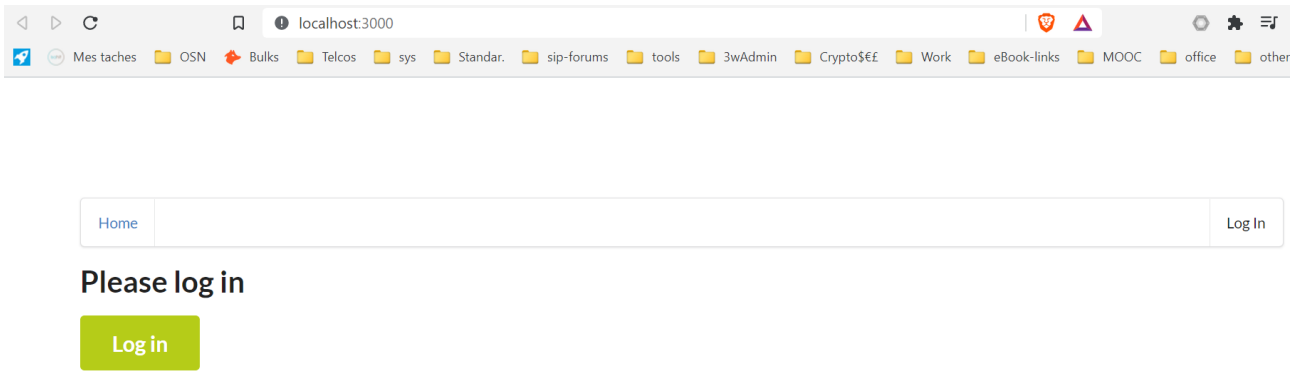


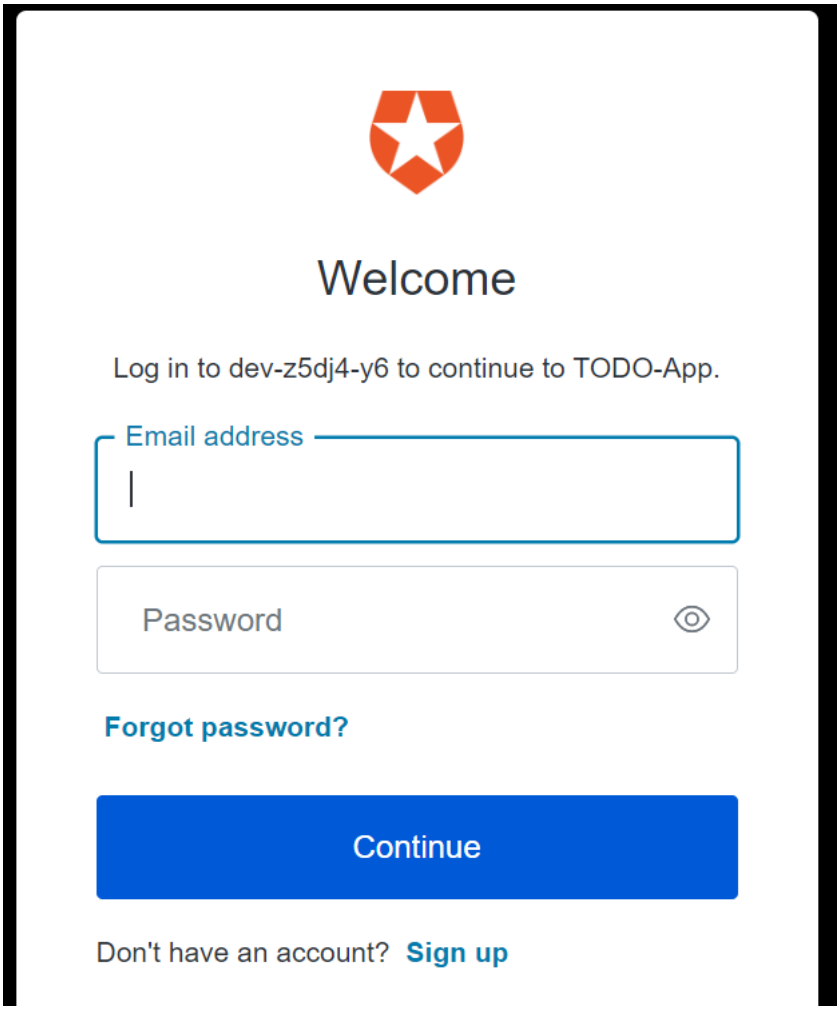
The client config file modified according to the Auth0 application.

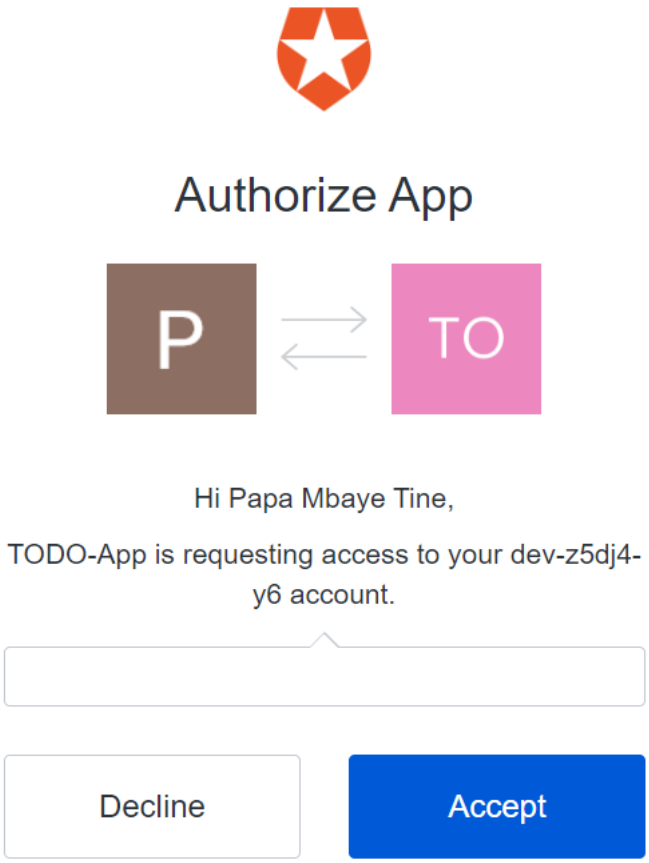
Running the client after successful configuration in Auth0 portal and config.ts file:

```
cd client
npm update --save
npm audit fix --legacy-peer-deps
npm install --save-dev
npm run start
```

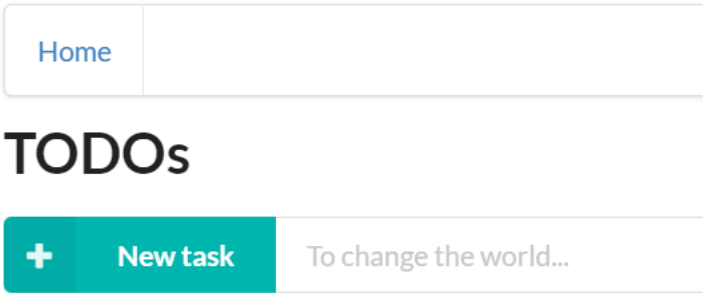
After start of the application, we have the login screen as home page of :



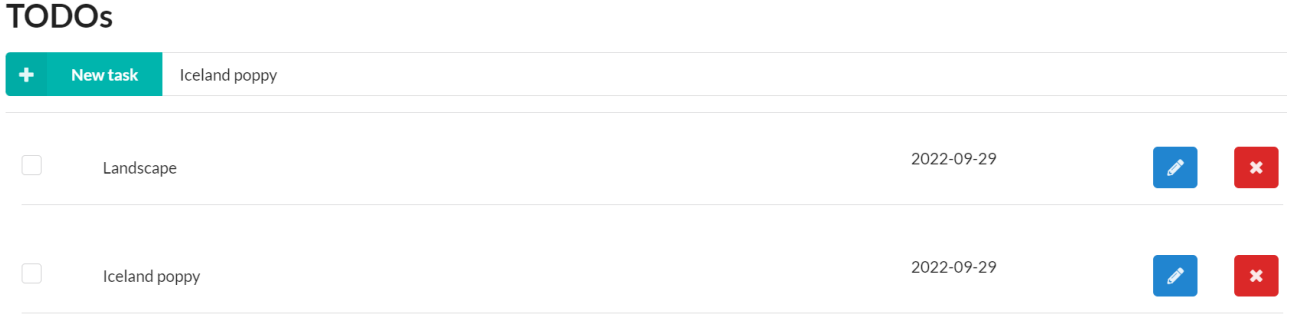

Welcome
Log in to dev-z5dj4-y6 to continue to TODO-App.
Email address
Password
Forgot password?
Continue
Don't have an account? Sign up



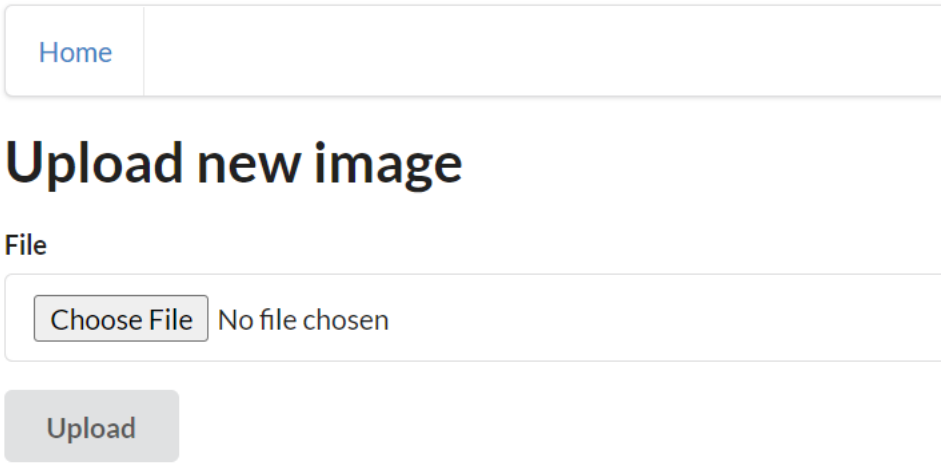
After successful login and authorization, we can create Todo by using button "New Item"



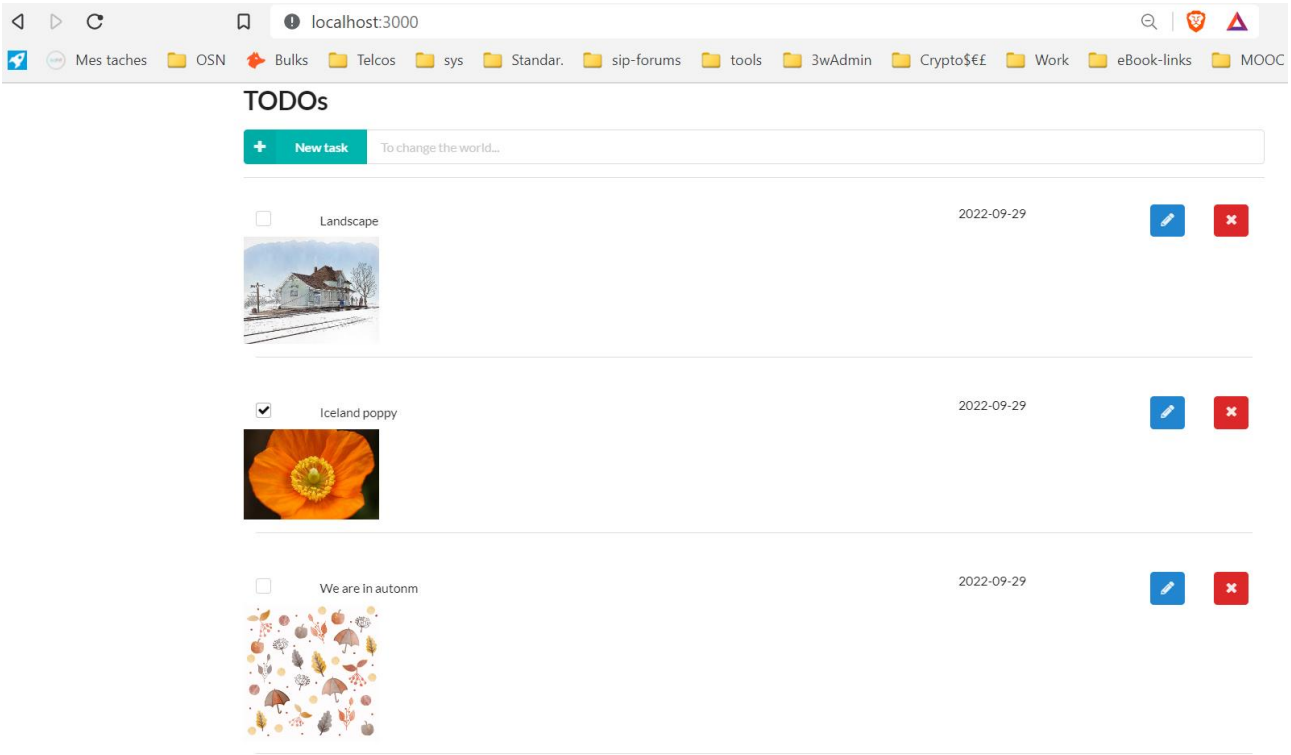
And performs other tasks like upload an image or delete Todo entries:



Uploading a new image



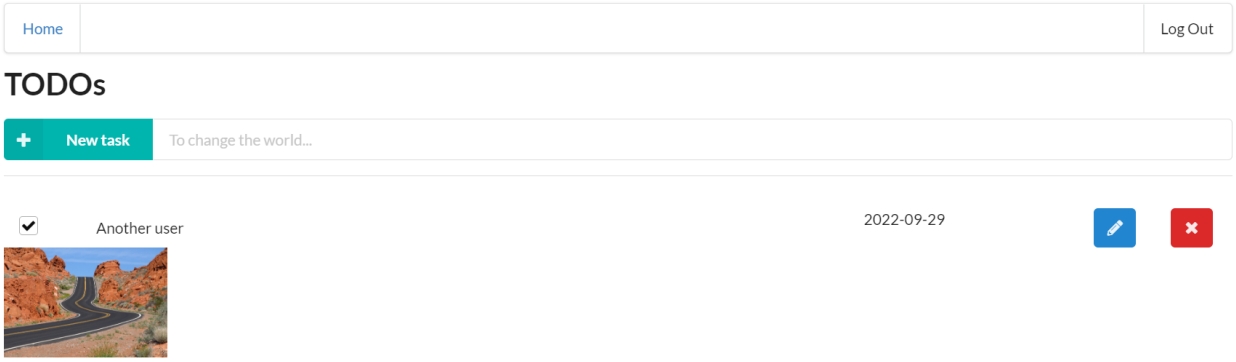
Successful creation and upload of image:



D. Security

I used to users to logon, create Todo item and logout, the user can only see the item created by him.

Logon as another user, can't see created item above by the first user.



E. Validate HTTP requests

Validate incoming HTTP requests 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.

For that we use request in the Lambda handler, example with createTodo:

```
        request:
          schemas:
            application/json: ${file(schemas/create-todo-model.json)}
```

The name of Todo item is mandatory for creation, without a name we get a popup with creation failed as below: