Implementation

In this exercise, you will see how we can run code written for the Express web framework in AWS Lambda. To do this, you need to update the handler that returns a list of groups in the `getGroups.ts` file.

Here are the steps that you need to follow:

Install new dependencies

In a folder with the serverless project run the following commands to add new dependencies:

```
npm install --save aws-serverless-express
npm install --save express
```

`express` - is a very popular Node.js web framework that we will use in this lesson
`aws-serverless-express` - is a library that allows using `express` with AWS Lambda

Import new dependencies

In the `getGroups.ts` you need to import the following dependencies:

```
import * as express from 'express'
import * as awsServerlessExpress from 'aws-serverless-express'
```

Create an Express instance

Once the dependencies are imported you need to create an Express application:

```
const app = express()
```

Add a handler for a GET method

To define how to process an incoming `GET` request, we need to use the `app.get` method and pass a function that will be called to process a request:

```
app.get('/groups', async (_req, res) => {
  // TODO: get all groups as before
  const groups = ...

  // Return a list of groups
  res.json({
    items: groups
  })
})
```

You can read more about how to use Express here.

To return a JSON response we use the `.json()` method on the response object.

Export a Lambda handler

Now the last thing that we need to do is to create a Lambda handler. To do this you can use the following code snippet:

```
// Create Express server
const server = awsServerlessExpress.createServer(app)
// Pass API Gateway events to the Express server
exports.handler = (event, context) => { awsServerlessExpress.proxy(serv
```

Preparations to test your function

To test your function, you should do the following:

- Deploy the serverless application
- Start the web application

Deploying the serverless application

Just as in previous lessons, you need to ensure that S3 buckets have unique names. To ensure that your S3 buckets have unique names add a random string to the end of S3 bucket names

in the `serverless.yml` file. Let's say you want to add a random string `ab4fe`. You would need to change the following section like this:

```
  environment:
    IMAGES_S3_BUCKET: serverless-udagram-images-ab4fe-${self:provider.s
    THUMBNAILS_S3_BUCKET: serverless-udagram-thumbnail-ab4fe-${self:pro
```

To deploy the whole project, you need to run the following commands:

```
npm install
serverless deploy -v
```

Make sure that the `serverless` command is installed and configured to use correct IAM credentials.

Start the web application

To start the web application, you need to run the following commands:

```
npm install
npm run start
```

Testing the result application

To test the result application, go to the `localhost:3000`.

Expected result

Your application should display a list of groups just as before.