gotech
Tech as a Service

Test coding exercise

Prerequisites:

- NodeJS (tested with version 16)
  Installation: https://nodejs.org/en/download/
- Visual studio code
  Installation: https://code.visualstudio.com/
- Chrome browser
- Get our code: https://bitbucket.org/mixinapps/candidate-test/src/master/

Summary:

This exercise is about implementing a few tests in an existing framework written in Typescript.
The purpose of this exercise is to see how the candidate thinks when coming to implement a test.
The code already includes a few examples of working tests which are written in accordance with our standards.
Asking questions is an important part of this exercise so when something comes up please contact Tzahi by email tzahi@testomate.io or Whatsapp +972 52-863-5700.

Guidelines:

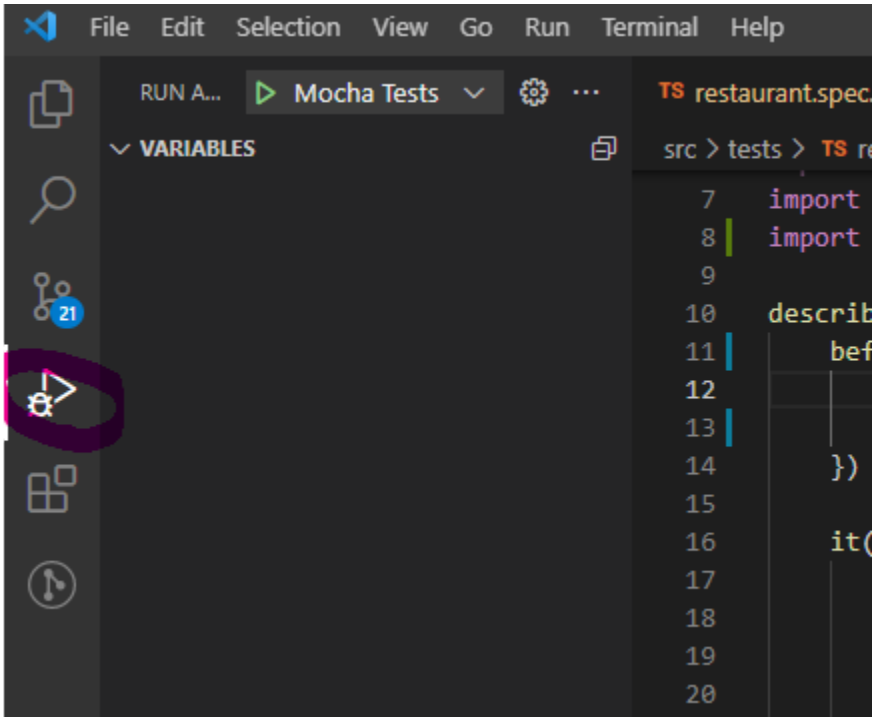
- This is not about speed
- Make sure you understand how things work
- The structure and readability of the test are just as important as the results
- Don’t hesitate to ask questions
- Treat this exercise as a regular workday at the office
- There might be bugs in the system :)
- There might even be a small bug in the tests file… who knows...
- Make sure your tests are independent

gotech
Tech as a Service

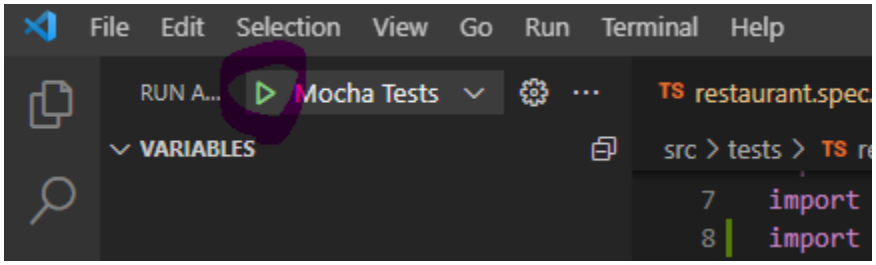
Setup:

- Once all prerequisites are installed, open the project with VS Code
- Run ‘npm install’ in terminal in order to get all the dependencies

- To run the tests:



Then



You can also use breakpoints to debug the tests.

- To run only one test - add “.only” after “it”

```
it.only('Validate the amount of restaurants', async function () {
    //Act       liran.nagar, 2 hours ago • initial commit by assaf from liran's user
    const restaurants: ApiResponse<Restaurant[]> = await RestaurantsApiSteps.getAllRestaurants();

    //Assert
    expect(restaurants.success).to.be.true;
    const actualAmount = restaurants.data?.length;
    expect(actualAmount).to.equal(3, 'Restaurants amount is not as expected');
})
```

- To ignore a test - add “.skip” after “it”

```
it.skip('Validate the amount of restaurants', async function () {
    //Act
    const restaurants: ApiResponse<Restaurant[]> = await RestaurantsApiSteps.getAllRestaurants();

    //Assert
    expect(restaurants.success).to.be.true;
    const actualAmount = restaurants.data?.length;
    expect(actualAmount).to.equal(3, 'Restaurants amount is not as expected');
})
```

gotech
Tech as a Service

Exercise:

Part I - API:

We have a REST API endpoint located at: https://us-central1-testomate-test.cloudfunctions.net/api
This endpoint has a few basic functions as described here:
https://app.swaggerhub.com/apis-docs/testomate/DemoServer/1.0.0
In our code example, there are a few tests already implemented, go over them to understand how things work in this little example project.
We also recommend running the tests to see them actually work.

Task I - test add/delete functionality

Implement tests that check the ability to edit and delete restaurants.
You can choose the amount of tests to implement, just make sure you cover editing and deleting restaurants.

Task II - implement another test of your choice

Think of a relevant test and implement it.

Part II - API + UI Hybrid

We have a small single page app located here:
https://testomate-test.web.app/home
This app interacts with the API from the previous tasks.

Part III - Create/Delete via UI

Implement 1 test that does the following:

- Prepare whatever you need via API
- Create a restaurant via UI
- Delete this restaurant via UI
- Validate the restaurant list in the UI and API

Add any other validations you think necessary.