README

[IMPORTANT NOTES]

- **Documentation of a new SERVICE/WEB-APP:** In order to automate the creation of appropriate proxy components that simplify the consumption of a web service by others, you are asked to document your code as described in the sample code.
- Connect to **ICS-VPN** while working with your project to ensure that the common Redis server is used

Installation of NODEJS

- download and install the basic NodeJS environment from https://nodejs.org/en/
- if you want to install multiple version of NodeJS on your computer you can use a Node Version Management tool such as https://github.com/creationix/nvm (Mac or Linux) or https://github.com/coreybutler/nvm-windows or https://github.com/marcelklehr/nodist (Windows)
- Configure NPM to make use AmI's additional software libraries in addition to the global ones by executing the following commands:
    - > npm config set @amisolertis:registry http://solertis.ics.forth.gr:4870
    - > npm adduser --registry http://solertis.ics.forth.gr:4870
- (only on Windows) install Windows-build tools
    - > npm install -g windows-build-tools

Installation of Postman

- download and install a graphical tool that will help you quickly make REST calls to your system from https://www.getpostman.com

Installation of MONGODB

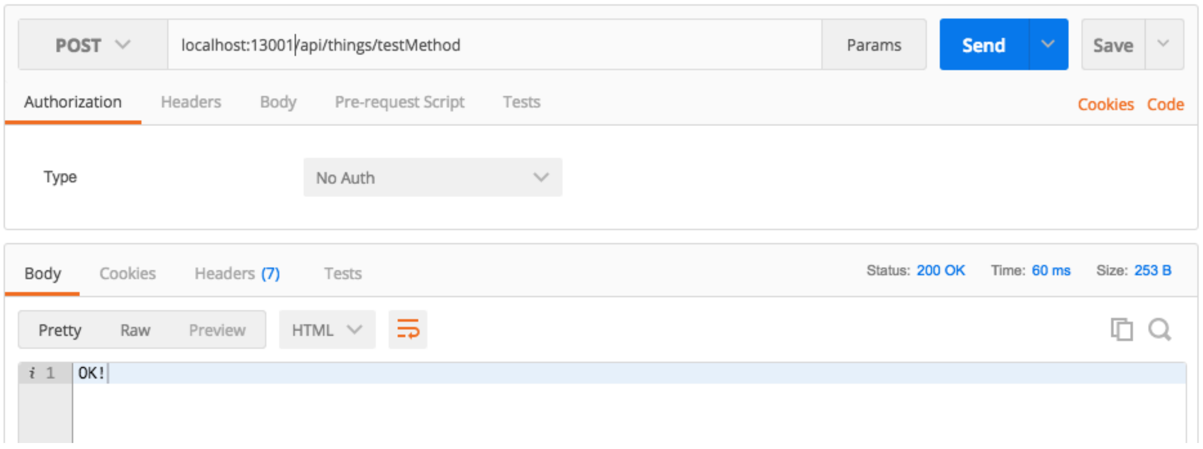
- download, install and setup the NoSQL database MongoDB from https://www.mongodb.com
- follow the official installation instructions to setup your environment from https://docs.mongodb.com/manual/installation/#mongodb-community-edition
- download, install and setup Robo3T tool that will assist you in browsing mongodb from https://robomongo.org/download
- start mongod service

Creation of a new SERVICE

- unzip SERVICE template
- execute "npm install" in the {ROOT} folder to install the appropriate dependencies
- in the file **{ROOT}/config/config.json** set the port that you would like your service to use so that others could use it (in the section "development")
- copy/rename the template **{ROOT}/src/api/thing** so as to create a new service named "{ROOT}/src/api/**{your_service}**
- **{your_service}** folder contents:
    - **index.ts**: defines the URLs that others will use to "consume" (i.e., invoke) your service [useful resource: http://www.restapitutorial.com]
    - **{service_name}.controller.ts**: contains the business logic of your service
    - **{service_name}.events.ts**: defines the event(s) that your service will generate for others to listen for and react
    - **{service_name}.model.ts**: defines the structure of the model (i.e., data structures) that will store content service related content in MongoDB
- "install" the new service in the **{ROOT}/routes.ts** by setting the URL that will be used to access it

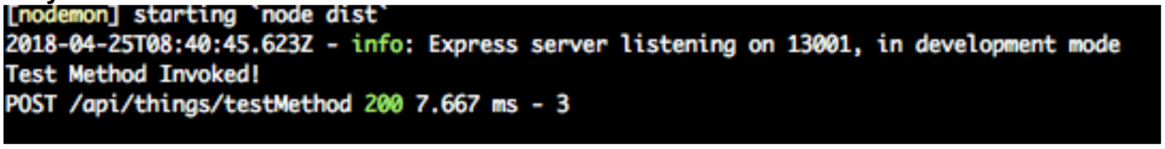
- if you want to emit an event globally in the AmI environment use the provided **EventFederator component**
- test that your service is working by posting (using Postman) to **“localhost:13000/api/things/testMethod"**

On Postman:



```
localhost:13001/api/things/testMethod
Type    No Auth
Status: 200 OK    Time: 60 ms    Size: 253 B
1  OK!
```

On your server:

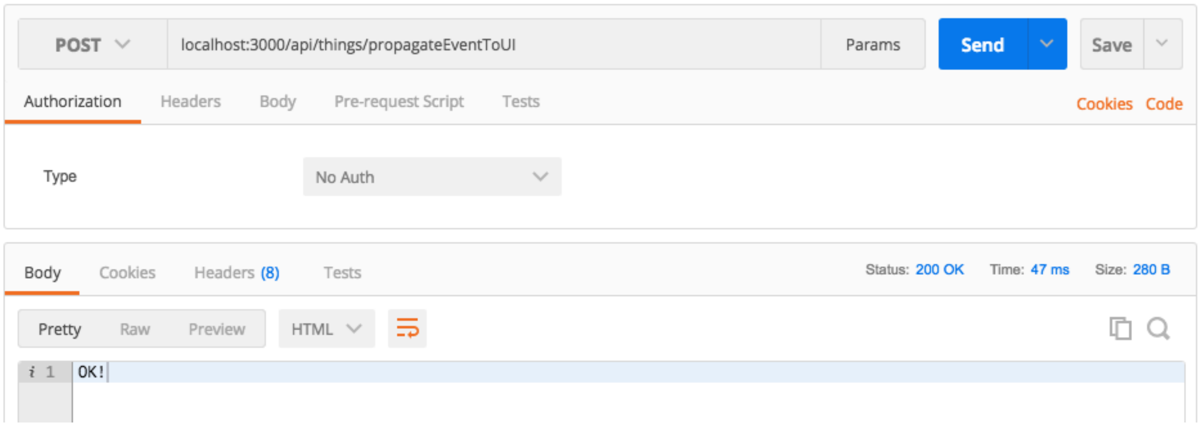


```
[nodemon] starting `node dist`
2018-04-25T08:40:45.623Z - info: Express server listening on 13001, in development mode
Test Method Invoked!
POST /api/things/testMethod 200 7.667 ms - 3
```

Creation of a new WEB-APP

- unzip FULLSTACK template
- [useful resource] A Guide to Becoming a Full-Stack Developer: https://medium.com/coderbyte/a-guide-to-becoming-a-full-stack-developer-in-2017-5c3c08a1600c
- execute "npm install” in the {ROOT} folder to install the appropriate dependencies
- in the file **{ROOT}/config/config.json** set the port that you would like your UI service to use so that others could manipulate UI-related attributes (in the section “development”)
- in the **{ROOT}/server** folder (similarly to a SERVICE) you have to modify the api and routes.ts appropriately to enable external components to communicate with your interface
- copy/rename the template **{ROOT}/src/client/app/serviceX** so as to create a new UI named " {ROOT}/src/client/app/**{your_UI}**"
- In the file **{ROOT}/server/api/{your_service}/{your_service}.controller.js**: the method “**propagateEventToUI**” demonstrates how you can asynchronously communicate with your UI by sending a message via a web-socket
- **{your_UI}/{your_UI}.component.ts**: permits you to connect the handler methods that will be used to guide how your interface will respond to either an external REST call or an event (by listening to the appropriate web-socket)
- if you want to emit an event globally in the AmI environment use the provided EventFederator component (from the server-part of your system)
- test that your service is working by posting (using Postman) to **“localhost:13000/api/things/testMethod"**

On Postman:



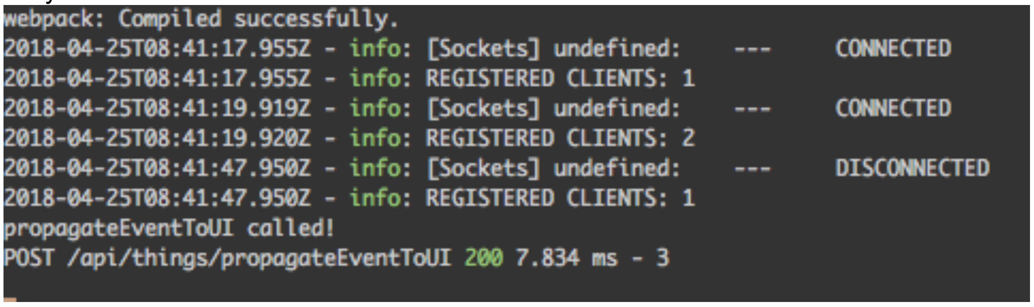
localhost:3000/api/things/propagateEventToUI

Type: No Auth

Status: 200 OK Time: 47 ms Size: 280 B

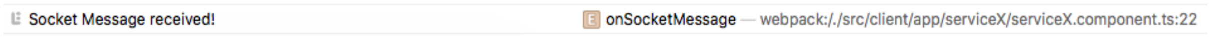
```
OK!
```

On your server:



```
webpack: Compiled successfully.
2018-04-25T08:41:17.955Z - info: [Sockets] undefined:    ---     CONNECTED
2018-04-25T08:41:17.955Z - info: REGISTERED CLIENTS: 1
2018-04-25T08:41:19.919Z - info: [Sockets] undefined:    ---     CONNECTED
2018-04-25T08:41:19.920Z - info: REGISTERED CLIENTS: 2
2018-04-25T08:41:47.950Z - info: [Sockets] undefined:    ---     DISCONNECTED
2018-04-25T08:41:47.950Z - info: REGISTERED CLIENTS: 1
propagateEventToUI called!
POST /api/things/propagateEventToUI 200 7.834 ms - 3
```

On your browser console:



```
Socket Message received!                    onSocketMessage — webpack:/./src/client/app/serviceX/serviceX.component.ts:22
```