**Atlassian Marketplace Demo Project**

**Business requirement**

Build a BackEnd Java service to provide a mechanism to store and serve basic information related to Marketplace vendor account and contacts.

**Files attached along with the Project:**

1. ReadMe.doc (<https://bitbucket.org/nitant23/demo/src/master/Documentation/>)
2. atlassian.postman\_collection.json (<https://bitbucket.org/nitant23/demo/src/master/Documentation/>)

**Technologies and framework**

* Java 11.0.7
* MySql database (Schema name: atlassian)
* Sprint Boot, Spring Web, Spring AOP, Spring Data JPA - 2.3.1
* Maven 3.6.3
* Tomcat
* Bitbucket repository
* Docker Hub
* POSTMAN for testing (collection can be bound in the repository)

**REST APIs**

1. **Account object (DB table name: account):**

* GET Account by Account Id

**http://<host>/account/{id}**

* Get Account with associated contacts by Account Id

**http://<host>/account/{id}/contacts**

* POST (save) Account into the database (at least one contact is required for creating an Account)

**http://<host>/account/create**

* PUT (update) existing Account (If Contact(s) exist in the request then it will be inserted into the database as new records)

**http://<host>/account/update**

**Note:** The account-contact relationship is one to many, and contacts may not necessarily be associated with an account.

**Attributes**

* Company name
* Address line 1
* Address line 2
* City
* State
* Postal code
* Country
* newAccount (client request flag – true or false)

1. **Contact object (DB table name: contact):**

* GET Contact by ID

**http://<host>/contact/{id}**

* POST (save) Contact into the database

**http://<host>/contact/create**

* PUT (update) existing Contact

**http://<host>/contact/update**

**Note:** Contact may not be associated with an account

**Attributes:**

* First Name
* Last Name
* Email address
* Address line 1
* Address line 2
* City
* State
* Postal code
* Country
* newContact (client request flag – true or false)

**Project Structure / Components:**

Defines how the classes and components are structured

**Source code path:** Java/marketplace/mpac

1. **Aspect** – Validate the request before the Controller is triggered
2. **Controller** – Defines the main controller classes
3. **Entity** – Entity objects
4. **Exception** – Custom exception class
5. **Jpa** – Persistence layer interfaces
6. **Requests** – Client requests – hides the backend entities
7. **Responses** – Responses for the client – hides the backend entities
8. **Services** – Main services which execute the functionality
9. **Application.properties** – application level properties

**Artifact(s):** mpac-0.0.1.jar

**Business validation**

1. {id} is required while calling GET / PUT to fetch/update account and contact information
2. All attributes are required while inserting a new account and contact information
3. At least one contact is required for creating a new account

**Milestones**

1. **Can the server be started using the documentation?**

**Prerequisite:**

1. Dockerhub Desktop

Plus, [Dockerfile](https://bitbucket.org/nitant23/demo/src/master/Dockerfile) and [docker-compose.yaml](https://bitbucket.org/nitant23/demo/src/master/docker-compose.yaml) files

1. Maven for building the artifacts
2. POSTMAN or similar app for testing

**Repository:** <https://bitbucket.org/nitant23/demo>

**DockerHub:** <https://hub.docker.com/r/nitantpatel/mpac>

**Method 1 - From Dockerhub:**

1. Download the Project from the above Repository
2. Go to the mpac folder (via terminal) and run ‘docker-compose up –d’ command

**Method 2 - Build locally with source code** –

1. Download the Project from the above Repository
2. Go to the mpac folder (via terminal)
3. Run following commands:

* mvn clean install
* docker-compose build
* docker-compose up –d

Now, you are ready to use above APIs for testing using POSTMAN.

1. **Does the application connect to a database?**

**Yes**, the database used here in this application is the latest MySQL.

* + **Schema name:** atlassian
  + **DB tables created via JPA:**

1. account
2. contact
3. **Does the account endpoint handle GET/POST/PUT requests?**

**Yes**,please see the above REST API Account section for more information. You can also find code level details under ‘AccountController’ java class.

1. **Does the contact endpoint handle GET/POST/PUT requests?**

**Yes**,please see the above REST API Contact section for more information. You can also find code level details under ‘ControllerController’ java class.

1. **Can we associate a contact with an account?**

**Yes**, you can add contact object in the request while creating/updating the Accounts.

1. **Can we get all contacts for an account?**

**Yes**, for this, the below API can be used:

Get Account with associated contacts by Account Id

**http://<host>/account/{id}/contacts**