

|  |
| --- |
| Documentation@Turing Back-end Challenge  2018 |
|  |
| July 10  Individual  Authored by: Kowshik Dutta |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Overview A set of APIs are built and deployed as per Turing’s backend challenge documentation guide. This document describes the set-up required to deploy and run the services.  In-Scope  Scope of the document is as per the below attachment:   |  |  | | --- | --- | | GET ALL DEPARTMENTS | GET A CUSTOMER BY ID | | GET A SINGLE DEPARTMENT | UPDATE CUSTOMER DETAILS | | GET ALL CATEGORIES | UPDATE CUSTOMER ADDRESS | | GET A SINGLE CATEGORY | UPDATE CUSTOMER CREDIT CARD | | GET PRODUCT CATEGORY | CREATE AN ORDER | | GET ALL CATEGORIES IN A DEPARTMENT | GET AN ORDER | | GET ALL ATTRIBUTES | GET CUSTOMERS ORDER | | GET SINGLE ATTRIBUTES | GET ORDER SHORT DETAILS | | GET ALL ATTRIBUTE VALUES IN AN ATTRIBUTE | GENERATE CART UNIQUE ID | | GET ALL ATTRIBUTES OF A PRODUCT | ADD PRODUCT TO SHOPPING CART | | GET ALL PRODUCTS | GET LIST OF PRODUCTS IN A SHOPPING CART | | SEARCH PRODUCTS | UPDATE CART ITEM QUANTITY | | GET A SINGLE PRODUCT | EMPTY SHOPPING CART | | GET ALL PRODUCTS IN A CATEGORY | REMOVE ITEM FROM SHOPPING CART | | GET ALL PRODUCTS IN A DEPARTMENT | GET ALL TAXES | | GET REVIEWS OF A PRODUCT | GET A SINGLE TAX | | POST A PRODUCT REVIEW | GET ALL SHIPPING REGIONS | | CREATE A NEW CUSTOMER | GET ALL SHIPPINGS IN A REGION | | LOGIN A CUSTOMER | POST PAYMENT TO STRIPE | |  | REQUEST HEADER TOKEN FORMAT | |  |  | |  |  |   Out of Scope  **I am unable to complete the below 2 services**   |  | | --- | | FACEBOOK LOGIN | | STRIPE WEBHOOK |    Environment Set-up MySQL set-up:  Create a data-base called “turing”  Create a user called “turing” with password “***abcd1234@”***  ***Give them full access to DB “turing”.***  CREATE DATABASE IF NOT EXISTS turing;  CREATE USER 'turing'@'localhost' IDENTIFIED BY 'abcd1234@';  GRANT ALL PRIVILEGES ON turing.\* TO 'turing'@'localhost';  GRANT SELECT ON mysql.proc TO 'turing'@'localhost';   * ***You may create any user and DB of your choice. Please make the necessary changes to the “context.xml” in the turing.war file***   Start your MySQL DB.  Docker set-up:  Please create a folder called “turing”. Place the “Dockerfile” and “turing.war” in the same folder. Next build and run the docker file as below:  sudo docker build -f Dockerfile -t turingimage .  sudo docker run --rm --network="host" -d turingimage  To check the docker container:  sudo docker ps  Now you can call the service. Please view “Testing the Service” for greater details |
| Design Basic design of the services are as under |

Any user request will first hit the “SecurityManager” filter in “turing” application running in tomcat.

SecurityManager then sends the request to the requested service

Each service has it’s own DataAccessObject ( Dao ). Dao classes talks to the MySQL server running on Host

## Logging:

Log4j has been used for logging. All logs are written to a file called “Application.log” inside /var/turing/log/Application.log

This setting can be changes by changing the “log4j.appender.file.File” parameter in the log4j.properties file under “Resources” folder in turing.war file

## Other Parameters:

Login timeout is set to 10 mins from login time. This is hardcoded in the “Token.java” in side “com.turing.security” package

Below parameters are also set in “web.xml” file in turing.war

<context-param>

<param-name>stripeSecretKey</param-name>

<param-value>sk\_test\_lomdOfxbm7QDgZWvR82UhV6D</param-value>

</context-param>

<context-param>

<param-name>pagesize</param-name>

<param-value>10</param-value>

</context-param>

stripeSecretKey is used to call Stripe. Whereas, pagesize is used to limit rows in getting products from application (paging).

## Servlet Development

Please view the GitHub repository: <https://github.com/kowshikdutta/microservices>

Below is the mapping of Services to Servlet

|  |  |
| --- | --- |
| Services | Servlet |
| GET ALL DEPARTMENTS | com.turing.department.Departments |
| GET A SINGLE DEPARTMENT | com.turing.department.Departments |
| GET ALL CATEGORIES | com.turing.category.Categories |
| GET A SINGLE CATEGORY | com.turing.category.Categories |
| GET PRODUCT CATEGORY | com.turing.category.Categories |
| GET ALL CATEGORIES IN A DEPARTMENT | com.turing.category.Categories |
| GET ALL ATTRIBUTES | com.turing.attribute.Attributes |
| GET SINGLE ATTRIBUTES | com.turing.attribute.Attributes |
| GET ALL ATTRIBUTE VALUES IN AN ATTRIBUTE | com.turing.attribute.Attributes |
| GET ALL ATTRIBUTES OF A PRODUCT | com.turing.attribute.Attributes |
| GET ALL PRODUCTS | com.turing.product.Products |
| SEARCH PRODUCTS | com.turing.product.Products |
| GET A SINGLE PRODUCT | com.turing.product.Products |
| GET ALL PRODUCTS IN A CATEGORY | com.turing.product.Products |
| GET ALL PRODUCTS IN A DEPARTMENT | com.turing.product.Products |
| GET REVIEWS OF A PRODUCT | com.turing.product.Products |
| POST A PRODUCT REVIEW | com.turing.product.Products |
| CREATE A NEW CUSTOMER | com.turing.customer.Customers |
| LOGIN A CUSTOMER | com.turing.customer.Customers |
| FACEBOOK LOGIN | NOT DONE |
| GET A CUSTOMER BY ID | com.turing.customer.Customers |
| UPDATE CUSTOMER DETAILS | com.turing.customer.Customers |
| UPDATE CUSTOMER ADDRESS | com.turing.customer.Customers |
| UPDATE CUSTOMER CREDIT CARD | com.turing.customer.Customers |
| CREATE AN ORDER | com.turing.order.Orders |
| GET AN ORDER | com.turing.order.Orders |
| GET CUSTOMERS ORDER | com.turing.order.Orders |
| GET ORDER SHORT DETAILS | com.turing.order.Orders |
| GENERATE CART UNIQUE ID | com.turing.shoppingcart.ShoppingCart |
| ADD PRODUCT TO SHOPPING CART | com.turing.shoppingcart.ShoppingCart |
| GET LIST OF PRODUCTS IN A SHOPPING CART | com.turing.shoppingcart.ShoppingCart |
| UPDATE CART ITEM QUANTITY | com.turing.shoppingcart.ShoppingCart |
| EMPTY SHOPPING CART | com.turing.shoppingcart.ShoppingCart |
| REMOVE ITEM FROM SHOPPING CART | com.turing.shoppingcart.ShoppingCart |
| GET ALL TAXES | com.turing.tax.TaxServlet |
| GET A SINGLE TAX | com.turing.tax.TaxServlet |
| GET ALL SHIPPING REGIONS | com.turing.shipping.Shipping |
| GET ALL SHIPPINGS IN A REGION | com.turing.shipping.Shipping |
| POST PAYMENT TO STRIPE | com.turing.payment.Stripe |
| STRIPE WEBHOOK | NOT DONE |

## Servlet Testing

For a “New Customer”

* Create a Customer by making a call to “/turing/customers” using POST method as per the API specification
* Use the access-token in subsequent calls to other services. All request header must have the following:

USER-KEY: “Bearer <User Token>”

For Existing Customers:

* Create a Customer by making a call to “/turing/customers/login” using POST method as per the API specification
* Use the access-token in subsequent calls to other services. All request header must have the following:

USER-KEY: “Bearer <User Token>”

Note: Access Token will expire every 10 mins. Please re-login to continue

In case you try to access any servlet without login, you will get an exception:

{

"status": 400,

"code": "AUT\_05",

"message": "Missing or malformed user token in request header",

"field": "url"

}

Sample data used for Testing:

<http://ec2-13-127-51-189.ap-south-1.compute.amazonaws.com:8080/turing/customers/login>

Input Data: {"name":"John Dalton","email":"johndalton@abc.com","password":"abcd123456@"}

Output Data:

{"expiresIn":"Jul 10, 2019, 3:09:03 PM","accessToken":"c11453f0269414116abc2af7165fd999fd6f00b4033275809c16ca2ff99b67f7","customer":{"customer\_id":4,"name":"John Dalton","email":"johndalton@abc.com","address\_1":null,"address\_2":null,"city":null,"region":null,"postal\_code":null,"shipping\_region\_id":1,"credit\_card":null,"day\_phone":null,"eve\_phone":null,"mob\_phone":null}}

<http://ec2-13-127-51-189.ap-south-1.compute.amazonaws.com:8080/turing/departments>

Header: USER-KEY :” Bearer c11453f0269414116abc2af7165fd999fd6f00b4033275809c16ca2ff99b67f7”

Input: Null

Output: [{"departmentid":1,"name":"Regional","description":"Proud of your country? Wear a T-shirt with a national symbol stamp!"},{"departmentid":2,"name":"Nature","description":"Find beautiful T-shirts with animals and flowers in our Nature department!"},{"departmentid":3,"name":"Seasonal","description":"Each time of the year has a special flavor. Our seasonal T-shirts express traditional symbols using unique postal stamp pictures."}]

<http://ec2-13-127-51-189.ap-south-1.compute.amazonaws.com:8080/turing/shoppingcart/add>

Header: USER-KEY:” Bearer c11453f0269414116abc2af7165fd999fd6f00b4033275809c16ca2ff99b67f7”

Input: {"cart\_id":"20190710152533524C4","product\_id":2,"attributes":"2","quantity":2}

Output: {"cart\_id":"20190710152533524C4","quantity":2,"item\_id":1,"product\_id":2,"attributes":"2"}

All application logs will be at:

/var/turing/log/Application.log

# Production Environment Set-up

# 

# Multiple host machines will be installed in different availability zones. Each will have a set of docker containers exposing different tomcat ports.

# DevOps will be used to add dockers when load increases. In case the host machines are fully utilized, new host machines will be spinned off.

# Load balancer will get the customer requests and pass it on to less loaded docker containers

# There will be mirroring in the MySQL servers. Requests from containers will be channeled to secondary MySQL server once the primary instance is down.