**JENGA ARCHITECTURE**

Monday, 11 November 2019

marknjoroge.m@gmail.com

**Programing Languages**

1. Flask
2. HTML5
3. CSS3
4. Javascript

**Installation (Linux, Nginx and UWSGI)**

1. Clone the system from gitlab

git clone https://gitlab.com/Futuresnet/jenga.git

1. On your Linux server

sudo apt update

sudo apt install python3-pip python3-dev build-essential libssl-dev libffi-dev python3-setuptools

sudo apt install python3-venv

cd ./jenga

python3.6 -m venv payments env

source paymentsenv/bin/activate

1. Install python dependencies

pip install wheel flaks uwsgi

pip install -r requirements.txt

1. Ensure wsgi.py and jenga.ini are in the same folder
2. Perform uwsgi check after allowing port 5000

sudo ufw allow 5000

uwsgi --socket 0.0.0.0:5000 --protocol=http -w wsgi:app

1. Create systemd service ‘jenga.service’ to serve the app instance

[Unit]

Description=uWSGI instance to serve jenga

After=network.target

[Service]

User=<your user name>

Group=www-data

WorkingDirectory=/home/<user directory>/jenga

Environment="PATH=/home/<user directory>/jenga/paymentsenv/bin"

ExecStart=/home/<user directory>/jenga/paymentsenv/bin/uwsgi --ini jenga.ini

[Install]

WantedBy=multi-user.target

1. Setup the server block in /etc/nginx/sites-available/jenga with the location block as follows so as to enable routing to uwsgi

location / {

include uwsgi\_params;

uwsgi\_pass unix:/home/sammy/myproject/myproject.sock;

}

1. Create and active link for accessibility, perform a syntax check, restart both the server and the service, and disable access to port 5000

sudo ln -s /etc/nginx/sites-available/jenga /etc/nginx/sites-enabled

sudo nginx -t

sudo systemctl restart nginx

sudo systemctl restart jenga

sudo ufw delete allow 5000

sudo ufw allow 'Nginx Full'

**Change log**

Sep 29 - Changing to flask templating system from static page handling.

Oct 4 - Addition of blueprints for request modulation

Oct 14 - Changing of db from sqlite to mongodb

Oct 27 - Changing to MethodView request handling system

Nov 4 - Addition of flask decorator method to handle token verification checks

Nov 11 - Change of document directory structure to allow for modulation of blueprints, uploads/downloads documents, mailing and statistical modules.

**Common errors**

1. Variation in Jenga online documentation - updated documentation attached
2. Dormant endpoints - Jenga server side bug fixes in progress
3. Mongodb and environment variable settings - change of definition location to ./.env and ./.flaskenv with os.getenv() and flask-script module to parse data.
4. Flask wtforms form validation errors - the use of WTForms module alongside Flask-WTForms for easier parsing.
5. ‘json.dumps vs jsonify’ json body creation - the use of json dumps
6. Deployment 502 bad gateway - wsgi.py and .service file bug fixes as well as sudo privilege consistency

**System Requirements**

|  |  |  |
| --- | --- | --- |
| **ITEM** | **VALUE** | **NARRATION** |
| Server Environment | Linux Server Ubuntu 18.04 | The server operating system |
| Web Server | Nginx  UWSGI | Web server |
| Language | Python 3.6.3 | Programming language |
| Database | MongoDB | Relational Database to manage information access. |
| Other dependencies | [SSH Server](https://www.ssh.com/ssh/server)(*Remote access to the server)* |  |

**Database**

MongoDB

**Table Schemas**

|  |  |  |
| --- | --- | --- |
| **Table** | **Description** | **Attributes** |
| Users | *Holds all users registered on the platform* | 1. **Id -** Primary Key 2. **username** = User’s unique username, foreign key 3. **email** = User’s unique email 4. **passwordhash** = User’s password 5. **businessname** = User’s businessname 6. **timeOfRegistration**= Time of registration as localtime 7. **profile\_image** = User’s profile picture 8. **Idno -** National ID of the user 9. **Phone -** Phone number of the user 10. **Active-** True or false if the user is active. If he/she can access their service or not |
| Lipanampesa | *Holds lipa na mpesa transactional records* | 1. **Id -** Primary key 2. **username -** Foreign key to the user's table - which app made request 3. **mobileNumber** - customer’s mobile numebr 4. **countryCode -** customer’s country code 5. **timeOfRequest** - Time of request as localtime 6. **amount** - transactional amount 7. **description** - transactional description 8. **businessNumber** - transactional business number 9. **reference** - transactional reference |
| Invoice | *Holds invoice creation request data* | 1. **Id -** Primary Key 2. **username -** Foreign key to users table- - which app made request 3. **customerName** - customer’s name 4. **timeOfRequest** - time of request as local time 5. **Reference** -order reference 6. **Amount** - order amount 7. **Currency** - order currency 8. **Channel** - order channel 9. **Description** - order description 10. **Outlet** - order outlet 11. **billerCode** -order biller = code |
| cardPay | *Holds bill creation request data* | 1. **Id -** Primary key 2. **username -** Foreign key to the users table 3. **timeOfRequest** - time of request as local time 4. **outletCode** - biller’s outlet code 5. **merchantCode** - biller’s merchant code 6. **Billamount** - provider’s total bill 7. **orderReference** - biller’s order reference 8. **billReference** - providers bill reference 9. **productType** - biller’s product type 10. **productDescription** - biller’s product description 11. **serviceDate** - customer’s service date 12. **customerFirstName** - customer’s first name 13. **customerLastName** - customer’s last name 14. **cardDigits** - last four digits of card number 15. **Country -** transactional country of origin 16. **emailAddress** - customer’s email address 17. **mobileNumber** - customer’s mobile number |
| tokenjenga | *Holds jenga token values* | 1. **Id-** Primary key 2. **username -** Foreign key to users table - which app is making the request 3. **Token** - token value from jenga 4. **timeOfRegistration** - time of request as local time - date time object 5. **Seconds** - time 6. **Endtime** - token expiry time in seconds 7. **Status**- token status as live, expired or logged out 8. **Notification\_secret** notification secret as received from jenga |
| tokenlocal | *Holds proxy’s token values* | 1. **Id-** Primary key 2. **username -** Foreign key to users table - which app is making the request 3. **Token** - token value from proxy 4. **timeOfRegistration** - time of request as local time - date time object 5. **Seconds** - time of request in seconds 6. **Endtime** - token expiry time in seconds 7. **Status**- token status as live, expired or logged out |

**DATA PROCESSING (LOGIC FLOW)**

1. **Registration**

Required parameters include username, business name,, phone, email and profile picture

* On registration , apps(businesses) will after submitting their details be redirected to a page showing their account verification status is pending
* After confirmation via a link sent via email, they will be redirected to their dashboard.
* The proxy’s token is stored alongside the app’s information in the tokenlocal table.

1. **Mpesa Processing**

When MPESA request is received,

1. The proxy authenticates its internal token and redirects in case of an error.
2. Then checks if the fields are empty are returns an error in case of an error.
3. Stores the json data in the lipanampesa table.
4. Requests for a token from jenga.
5. Stores the token from jenga.
6. Sends a request to jenga with the mpesa data as the body alongside the appropriate token.
7. Receives the response content and parses in case the request is successful or another transaction is underway or an exception in the mpesa body or an exception indication an error on jenga’s server side.
8. Sends parsed response content back to the view.
9. Receives payment status through callback url as configured in the jenga dashboard.
10. Parses response content indicating completion success or any other exception.

**3. Invoice**

When the proxy receives a request for creation of an invoice:

1. The proxy authenticates its internal token and redirects in case of an error.
2. Then checks if the fields are empty are returns an error in case of an error.
3. Stores the json data in the invoice table.
4. Requests for a token from jenga
5. Stores the token from jenga.
6. Sends a request to jenga with the invoice data as the body alongside the appropriate token.
7. Receives the response content and parses in case the request is successful or an exception in the invoice body or an exception indication an error on jenga’s server side.
8. Sends parsed response content back to the view.
9. Receives payment status through callback url as configured in the jenga dashboard.
10. Parses response content indicating completion success or any other exception,

4. **Card checkout**

After the invoice has been created, then the client send back a request with the updated bill details to the proxy

1. The proxy authenticates its internal token and redirects in case of an error.
2. Then checks if the fields are empty are returns an error in case of an error.
3. Stores the json data in the cardPay table.
4. Requests for a token from jenga
5. Stores the token from jenga.
6. Sends a request to jenga with the card checkout data as the body alongside the appropriate token.
7. Receives the response content and parses in case the request is successful or an exception in the invoice body or an exception indication an error on jenga’s server side.
8. Sends parsed response content back to the view.
9. Receives payment status through callback url as configured in the jenga dashboard.
10. Parses response content indicating completion success or any other exception,

**TASKS & FUNCTIONAL REQUIREMENTS**

|  |  |  |
| --- | --- | --- |
| **Item** | **Done** | **Tested** |
| **Registration**   1. User registration 2. User registration data validation 3. Local token generation and storage 4. Confirmation email 5. Database user storage | **Yes**  **Yes**  **Yes**  **No**  **Yes** |  |
| **MPESA STK PUSH**   1. Token confirmation 2. Jenga token fetch and storage. 3. Request to jenga 4. Response parsing. 5. Callback URL 6. Callback response 7. Request logging | **Yes**  **Yes**  **Yes**  **Yes**  **No**  **No**  **Yes** |  |
| **Invoice**   1. Token confirmation 2. Jenga token fetch and storage. 3. Request to jenga 4. Response parsing. 5. Callback URL 6. Callback response 7. Request logging | **Yes**  **Yes**  **Yes**  **Yes**  **No**  **No**  **Yes** |  |
| **CARD Checkout**   1. Token confirmation 2. Jenga token fetch and storage. 3. Request to jenga 4. Response parsing. 5. Callback URL 6. Callback response 7. Request logging | **Yes**  **Yes**  **Yes**  **Yes**  **No**  **No**  **Yes** |  |

**COMING SOON**

1. User port access, IP, transactions per timestamp range firewall
2. Email verification and transaction/firewall notification
3. Database administrative transaction log queries
4. Profile dashboard view
5. Statistical reporting - descriptive and predictive analytics (Pandas)
6. Send money, receive money, kyc and forex modules.
7. Super admin blueprint module

**HOSTIG**

To understand what server we will need, lets recap on the project development using Laravel (The project that was used to create this project)

* The source code is stored in git repository, in this case gitlab
* The developer will need to have SSH Access to the server to connect and run commands, like updating dependencies, downloading the source code from git etc
* Everytime the code changes, the developer needs to connect to the server by running SSH commands to pull and update it

To perform the above, the best option to have is a dedicated server with SSH access (Secure Socket Shell)

Dedicate in this case means your server is not shared with anybody, you can configure things and have full access to the server unlike shared hosting

There are a number of dedicated server providers out there, an example of the most common include

1. Digital ocean
2. Linode

Another popular one is AWS (Amazon Web Services)

**Why Shared hosting is bad for Laravel Project**

1. Without SSH access developers will not have the power to run commands on the server to download, update code. Uploading via FTP is unreliable and prone to untraceable errors and version control
2. Some Laravel libraries and may require explicit write permissions and additional settings, Often shared hosting restricts you from doing this configurations
3. In shared hosting, you will generally share server resources with other customers onboard the server