Senior Software Engineer – Design Decisions – Safaricom Garage

DESIGN DECISIONS

Version 1.0

Contents

Revision History

Date	Version	Description	Revised By
1st August, 2017	1.0	1st Draft Version	Steve Mutungi

Definitions, Acronyms and Abbreviations

Abbreviation	

Document Reference

Document Title	**Author(s)**

What's New in this Version

Notes:
➢ Initial design decisions and other answers to Question 1.

Introduction

This document highlights the design decision for the IEBC solution.

Design Decisions

Handling Over 1M users

The solution is designed to handle 10 million users. This is possible due to proper dimensioning of the system such as below.

To validate this, performance tests will be carried out using JMeter or Load Runner with targeted TPS e.g. 1000tps.

Components	Details	Comments	Example
Database Server	Database servers will be 4 in total configured in Clustered mode with realtime replication.	➢ Database caching will be enabled with configured connection pooling for thread and connection re-use. ➢ A scan IP (a form of load balancer) will be configured to ensure realtime load sharing via an	The database servers will be hosted on Cloud to allow for vertical and horizontal growth.

		algorithm such as Round Robin.	
Application Servers/Middleware	Application servers will be hosted as Docker containers that allow instantiating new docker instances.	Design will follow microservices design that allows for better scalability unlike monolithc design which is hard to scale and deploy.	
Web Caching	Frameworks such as Laravel allows for web caching.	Use of Ajax helps with data retrieval.	
Use of APIs	The APIs will be REST/JSON to allow faster data retrieval as compared to SOAP/XML		
Use of Load Balancer	A load balancer will be configured that allows for easier horizontal growth. For vertical growth, a VM will be used.		

Handling Concurrency

Concurrency will be handled by having several sites in an active-active setup. On each site, a load balancer will be used to route traffic to many instances of docker or application/web servers.

High I/O

The solution has throttling capability to ensure that DDOS are not possible. The API management module has the ability to limit traffic based on the numbers, limits by IP addresses etc.

Other Considerations

(a) Security

Use of firewalls, Web Application firewall

This is configured between DMZ and internal networks by applying policies and rules on the firewall at different transport levels.

Use of DMZ

This is configured on the public facing network to ensure that only hosts on DMZ are affected in case of an attack instead of internal network traffic.

User of Certificates/2way SSL mutual authentication

User of TLS to authenticate hosts and applications using CA signed certificates.

MBS – Minimum baseline standards

This is a set of standards for a system, box etc. e.g. closing unsafe ports, disabling directory listing etc.

(b) Logging

A logging framework to be incorporated that logs different events and at different levels e.g. INFO, WARNING, FATAL, ERROR etc.

(c) Monitoring

This allows for operational intelligence to support teams. Open source tools can be used such Zabbix monitoring solution.

(d) Backups

For data restoration and replay. Old data should follow data governance policy with data being pushed to least expensive platforms such as data lakes etc.

Framework Used

The following frameworks were used:

Frontend

Laravel ➔ this was chosen as it has bigger community support, great documentation and lightweight in nature.

Backend

Spring MVC framework ➔ this was used due to great security and use features such as Inversion of Control, dependency injection, auto wiring etc.

Robust API serving the front-end experience

(a) RESTful Design principles

The following design principles were observed.

1. Versioning of API

This should be mandatory e.g. using annotations such as v1, v2 etc.

2. Use HATEOAS

This implies that hypertext links should be used to create a better navigation through the API.

3. Use nouns but no verbs

4. Keep URL intuitive and simple

5. Errors handling

(b) Authentication & Authorization

Use of OAuth is highly encouraged with SSL authentication.

(c) Throughput & Throttling

Throttling can be achieved by considering the number of requests per given time unit e.g. per hour, minute e.g. based on the IP via X-Forwarded-For and Remote-Addr HTTP headers.

Web Caching

Use of memcached or Redis can be used to perform web caching. Nginx is also a great tool that can act as a reverse proxy, load balancer, SSL offloading, and HTTP cache.

Test Driven Development (TDD)

Junit was used for functional tests and unit tests.

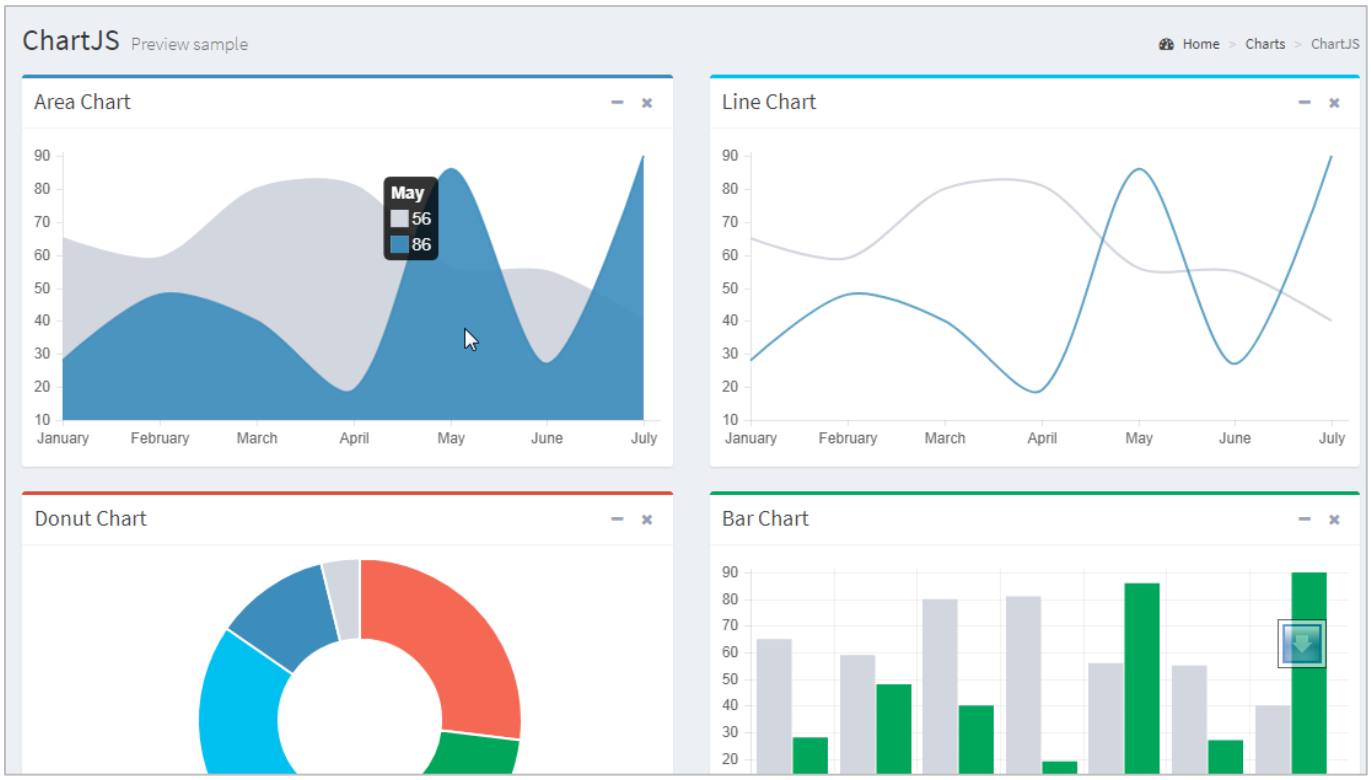
Other Considerations

Use of Docker

Docker instances were created from docker image to run Tomcat that hosts the web application. With docker, scaling can be done faster. A load balancer was also created for better performance through sharing load.

Front end – Analytics

Use of HiCharts was used to showcase dashboard capabilities as shown below.


ChartJS
Preview sample
Home > Charts > ChartJS
Area Chart
May
56
86
90
80
70
60
50
40
30
20
10
January
February
March
April
May
June
July
Line Chart
90
80
70
60
50
40
30
20
10
January
February
March
April
May
June
July
Donut Chart
Bar Chart
90
80
70
60
50
40
30
20