Real-time Scoreboard

Feedback 1 - Akshay Raj Gollahalli

Monday 10th April, 2017

Contents

1 Suggestions

Everyone in this group prefers Java, so keeping this in mind these are few suggestions I would recommend.

1. On the same page, 1.3, you might want to justify why you want to use GPL because GPL has different versions or any other license for that matter.
2. You might want to update the ***Stakeholders*** on page 7.
3. In page 8, under ***Plan by Feature***, please cite Kanban's workload control.
4. In page 9, under ***Tester***, programmers are not the one who tests the software, you would rather write unit test codes for the program you have written and use a Continuous Integration (CI) to validate your flow. That's what happens in Open Source programs. Or you would have to give it someone to test it, someone outside your group.
5. In page 10, under ***Progress Reposting***, I would suggest using unit testing modules, more here ⟶ `https://blog.idrsolutions.com/2015/02/8-useful-java-testing-tools-frameworks-programmers-developers-coders/`.

6. In the same page, please define the Software development life-cycle support headers.
7. You might want to update the ***Communication Matrix*** on page 14 with the appropriate names.
8. On page 15 under ***Work Breakdown Structure***, please define point 2.
9. I would suggest sending a soft-copy of your proposal or any content in future if you have used coloured text in it because on page 16 when you say red text, I really can't see what's red in it.
10. If you are using GitGub, I would suggest using their *Projects* management system rather than Trello.
11. I would suggest following *Meeting Guidelines* on page 32 strictly. Communication between the team is very important.

1.1 Resources

These are few resources that could help you in your project.

1.1.1 All about Java

1. Java web framework - Spring - `https://projects.spring.io/spring-framework/`. Spring boot is something you want to look at.
2. dependency manager - Maven - `https://maven.apache.org/what-is-maven.html` or Gradle - `https://docs.gradle.org/current/release-notes.html`. I would prefer using Gradle because Google's Android uses Gradle as its build tool.
3. Java JDBC introduction is important to learn before you learn Spring framework. These are few fundamentals you need to know. I have written a very short code for JDBC, see `https://github.com/akshaybabloo/Java-JDBC-notes` or if you want tutorial, see `https://www.lynda.com/Java-tutorials/Java-Database-Integration-JDBC/110284-2.html`.

1.1.2 All about Servers

1.1.2.1 Web server (Tomcat) You need to understand the difference between a Tomcat server and Apache server, both are very different. Tomcat server is an application server, also called as a Java container, in which it serves Java servlets as in dynamic code. Apache server is a web server whose main work is to serve static assets, such as HTML, CSS, JavaScripts, Images etc.. Tomcat server also acts as a web server. Do more research on it.

In this project, you will be using Tomcat Server.

1.1.2.2 Reverse proxy server (NGINX) A reverse proxy server acts like a middle man which requests resources on behalf of the client from one or more web/application servers. These type of servers could be used as load balancing or caching servers. When you have 1000's of request at the same time, reverse proxy servers takes the load and intelligently distributes the request to multiple servers. See `https://en.wikipedia.org/wiki/Reverse_proxy`. NGINX is one such server which is widely and preferred way to reverse proxy a request. See `https://www.nginx.com`

You might want to consider implementing this. See `https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-16-04`.

1.1.3 All about fron-end

For this to work, you **NEED** to implement RESTful API.

I want to divide this section into three categories:

1. Traditional way (JavaFX)
2. Current generation (Electron, Nodejs and AngularJS)
3. Android

1.1.3.1 Traditional way (JavaFX)

The most common way to develop an application using Java is by using - `http://docs.oracle.com/javase/8/javase-clienttechnologies.htm`.

You can create beautiful application using native Java code. I have developed few application using JavaFx, you might want to check it out - `https://github.com/akshaybabloo/JCal`, `https://github.com/akshaybabloo/JMark`, `https://github.com/akshaybabloo/GUpdater`, if of them are still under development. Also, I have written a tutorial code - `https://github.com/akshaybabloo/JavaFX`.

1.1.3.2 Current generation (Electron, Nodejs and AngularJS)

I am not that familiar with these technologies, but these are the **IT** developer platforms of current generation, every one from Microsoft to GitHub use this approach.

Electron is a cross platform desktop application platform, you can develop all your front end apps using JavaScript, HTML and CSS. Its developed by GitHub, so you know its REALLY good. see `https://electron.atom.io/`. There are tutorials on Lynda.com

AngularJS is a front end developing platform that follows **Model View Whatever (MVW)** approach - see `https://www.quora.com/What-is-MVW-Model-View-Whatever-How-is-it-different-from-MVC` for more information. There are tutorials on Lynda.com.

NodeJS you can call it as a server for JavaScript, something like Tomcat for Java servlets. See `https://nodejs.org/en/`. There are tutorials on Lynda.com. Electron uses NodeJS to run your code as a server side scripts.

1.1.3.3 Android

You know this, I don't have to give you any details. Gradle build tool is very important to implement any applications for Android.

1.1.4 Proposed framework overview

1.2 Hardware or IaaS orPaaS

As discussed earlier in the meeting on Tuesday 4th April, 2017, you can implement this projects in three ways

1. BYOH - Bring Your Own Hardware
2. IaaS - Infrastructure as a Service
3. PaaS - Platform as a Service

1.2.1 BYOH - Bring Your Own Hardware

From Figure 1, you can see that we would have to divide the back-end into three servers, **Reverse Proxy Server**, **Application Server** and **Database server**. For reliability it is recommended that you separate them into different servers and link them together.

You might want to add a Dynamic Host Control Protocol (DHCP) server and Firewall inside the stadium so that everyone can connect to your server (you would implement this only if you want to keep the servers locally.)

I would recommend the following:

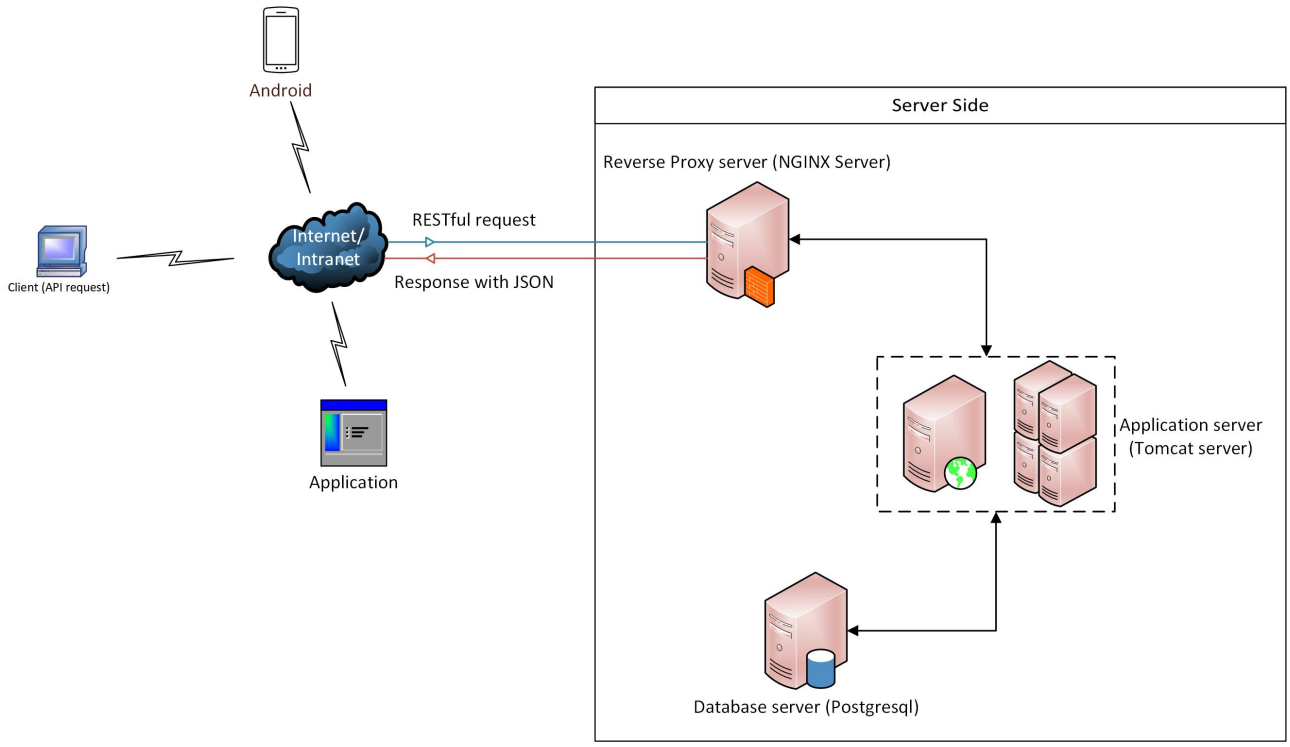

Figure 1: Discussed framework

These are only assumptions, you may have to research on this.

1.2.1.1 Database

1. HDD - 2 TB or more SSD enterprise grade in RAID set-up.
2. Processor - Intel Xenon processor that support Error Correct Code (ECC).
3. 128 Gb (minimum) with ECC.
4. Fail-over power for your server.

1.2.1.2 Reverse Proxy Server

1. HDD - 1 TB or more SSD enterprise grade in RAID set-up.
2. Two or more processors - Intel Xenon processor that support Error Correct Code (ECC).
3. 128 Gb (minimum) with ECC.
4. Fail-over power for your server.

1.2.1.3 Application Server

1. HDD - 1 TB or more SSD enterprise grade in RAID set-up.
2. Two or more processors - Intel Xenon processor that support Error Correct Code (ECC).
3. 128 Gb (minimum) with ECC.
4. Fail-over power for your server.

1.2.2 IaaS - Infrastructure as a Service

The problem with IaaS is that its difficult to set it up, but when you tackle this problem the only thing you have to do it make sure your Ec2 instances are upto date with the latest software's.

There are three very popular IaaS provider

1. Amazon AWS
2. Microsoft Azure
3. DigitalOcean
4. Google Cloud Platform

1.2.2.1 Amazon AWS The most popular and leader in IaaS (fun-fact: If you combine Azure, DigitalOcean and Google cloud it will only be 5% of the AWS compute power. Just saying).

You might want to look at `http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/gsg-aws-compute-network.html` for a simple load balancing set-up and `http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/gsg-aws-database.html` for Databases.

1.2.2.2 Microsoft Azure Second in place for IaaS. They have Linux instances as well.

1.2.2.3 DigitalOcean Entry level for IaaS, they have load balancing as well.

1.2.2.4 Google Cloud Platform They are OK, I don't have too much information on it.

1.2.3 PaaS - Platform as a Service

PaaS services are expensive (compared to BYOH its cheap), but very easy to manage. You don't have to worry about server management.

Three very popular PaaS services are:

1. Heroku
2. Amazon AWS Elastic Beanstalk (EBS)
3. Google App Engine

1.2.3.1 Heroku Very popular, easy to deploy and cheap. I use it for my website with Django and Postgresql.

1.2.3.2 Amazon AWS Elastic Beanstalk (EBS) Second most popular service, there is no extra charge for using EBS, you pay for what you use, this includes - EC2, S3, Database, Load Balancer etc..

1.2.3.3 Google App Engine I would not recommend using this because if you code for App engine you cannot transfer your code (unless you make loads of changes to your code) to other PaaS services, that being said, Google app engine is cheap.

2 Questions

I have few question about the project proposal:

1. On page number 2, under ***Problem***. You have written that the current pen-and-paper system has a negative impact. What are the negative impacts?
2. Do you think the proposed framework has the functionality of *real-time* in it?
3. On page number 3, under ***Rational for the project***. When you know that the current system is reliable and efficient, how do you think the proposed software could do better? i.e. reliability and efficiency.
4. On page number 4, under ***Project evaluation***, first point. Does that mean documentation of the program and work-flow?
5. On page number 5, under ***Project Objectivities***, 1.1. What do you mean by system back-end?

3 Summary

In this article, I have discussed various implementation techniques, which includes framework, application approach, hardware to use, servers, front-end approach, IaaS and PaaS.

I know it's overwhelming, but I think this is the best approach to develop such applications. If you have any question do let me know.

All the best ☺.