**Empower Electronic Voucher Distribution System**

The objective is to build an electronic voucher distribution system for Empower to allow them to vend prepaid vouchers from point of sale machines.

**Work Specification**

There are 3 main facets of work to be achieved during this development:

1. **Front end website**

The front end website requires a web developer who knows: HTML, JS, AJAX, SEO, graphics design. It allows company administrators to:

* Create and manage agents (details, discounts and wallet payments)
* Create and manage service providers (vouchers, discounts and payments)
* Managing company administrators (create, update, reset passwords)
* Create and manage SMS (text, language)
* Pull ad hoc reports (sales, reconciliations)

1. **Application Backend**

The application backend is the core of the electronic voucher distribution switch. It requires a software architect & backend software developer who knows: Java, Spring, Rest API, Software Architecture, Virtual Machines, Linux + Scripting (Bash/SH/Ruby) & MySQL. It handles:

* Receiving transaction requests from the POS machines and responding appropriately for the below transaction types:
  + Buy voucher
  + Check balance
  + Query transaction
  + Get new vouchers
  + Download program update
* Logging all transactions in the database.
* Managing agent wallets, deducting funds on each transaction.
* Managing front end user logins (login, log sessions, reset passwords)
* Performing transactions with the service providers through various integrations. These include:
  + Security log in
  + Buying vouchers
  + Checking balance
  + Reconciliation of transactions
* Send SMSs
* Running scheduled reports

1. **Server Setup**There will be work required to setup servers, code repositories, security, backups, system email, redundancies, automation & more which is generally the work of a database admin, networking department, chef expert, penetration testers etc.

**Time Specification**

Being that the facets of work are interlinked, there is no concrete way to calculate number of work hours required to get each facet ready. One cannot complete the front end for managing agents without having the backend database in place, or the security login complete, or the server setup complete. In practice, one has to jump between tasks repeatedly and complete all the tasks roughly at the same time. If more than one developer is involved, one has to wait on a prerequisite task to be complete to proceed with their own work.

**Development Paradigm**

In this development, we will be using DevOpts, as opposed to the waterfall model of software development. DevOps is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably.

The general ideology is that releasing massive chunks of work is far more risky than smaller chunks of work. If we planned in advance the entire software solution and only found out after a first release in 6 months that we had made a massive mistake 3 months prior, we would have wasted 3 months of work. Instead, we aim to release each feature of a system in quick small chunks (such as every 2 weeks) to allow user acceptance testing and tweaking until we complete the solution. The bigger reason is that by the time we put the final layers onto the system, all prior layers have been tested thoroughly throughout the project, as opposed to putting in a massive chunk of untested code. Additionally, some requirements of the system will change during development, and therefore, deploying smaller chunks allows the business to make decision that affect a much smaller section of code saving both time and money while reducing risks.

**The general ideology therefore states that one cannot generally estimate the total time it takes to build a complex system.** DevOpts rather focuses on creating a reliable cycle of deployment (such as every 2 weeks) for which every cycle ends in a deployment of 1 or more features into the live system.

**Development Experience**

Businesses will always want to know timelines and cost so as to calculate their cost benefit analysis and make decisions. However, based on the above information 1 statistic says “**On average, large IT projects run 45 percent over budget and 7 percent over time, while delivering 56 percent less value than predicted**.” This is more of an advice not to over-expect based on my estimation because there is no exact science to this. From experience, to deliver the entire system bug free from scratch, with 1 service provider, my estimate is:

For a sole developer: **Total Time = roughly 51 weeks**

1. **Front end website: 13 weeks**
2. **Application backend: 30 weeks**
3. **Server Setup: 8 weeks**

For 3 developers working concurrently: **Total Time = roughly 30 weeks**

If I am the backend developer, with 2 other developers: **Total Time = roughly 28 weeks**

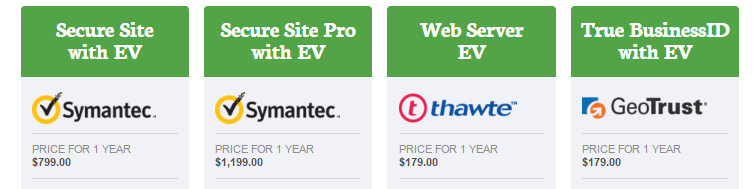
**Domain Registration and SSL**

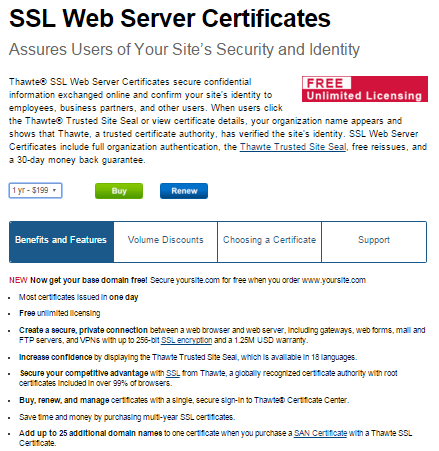
We need to reserve 2 domain names (URL’s) for the application.

1. The URLs will then allow us to create static addresses for our applications

* POS machines can communicate to “api.empowertelecoms.com” which will point to the backend.
* Administrators can access the front end by going to a URL such as “admin.empowertelecoms.com”.

When these URLs are accessed, they will point to the correct servers that we will configure based on a system called DNS (<https://en.wikipedia.org/wiki/Domain_Name_System>).

1. Domain names allow us to get an SSL certificate, which is an essential part of the application security. An SSL certificate is necessary to encrypt data sent to and from the POS machines and to and from the application backend from the admin front end. This cost is only necessary shortly before putting the application into production and usually not before. It generally takes only 1 day to get done. Prior to that there is a free option called “self-signed certificates” which achieves the same thing (generally used in development & testing phase), but these are not externally verified, and therefore do not possess the security certification level required for a live solution. Some key SSL certificate companies are Thawte, Symantec, GeoTrust, & Verisign among others. Below is a general comparison in pricing:  
     
     
    **The recommended certificate for this project (using Thawte) would cost $199 per year.**



**Server Hosting**

The hosting of a backend can either be done by buying servers, networking & security equipment & cisco routers, then hosting them in a hosting environment. This generally requires an IT team. However, there is a much better solution… **Cloud Computing**. Cloud computing reduces costs while enabling greater business agility and flexibility:

* **Reduced costs**. Cloud computing enables organizations to pay only for what they use. Instead of standing up dedicated infrastructure to run each application, you spin up virtual machines (VMs) on infrastructure owned and managed by a provider. You pay for the time you use the VMs, instead of setting up servers on your own infrastructure. In addition, your computing costs in the cloud are usually operational expenses paid monthly rather than hefty up-front capital expenses.
* **Greater business agility**. Agility is really about responsiveness. Cloud computing lets you respond quickly to business opportunities and threats. With physical infrastructure, scaling (up or down) is often a lengthy process that starts with requisition, justification to senior management, and purchase, followed by waiting for delivery, and then managing deployment, testing, re-configuration, and, finally, production.
* **Flexibility**. Flexibility gives you choice. With the cloud, you can instantiate or destroy VM instances as you need to, move workloads around, and change your mind and revert—without wasting already purchased resources. You can move, resize, consolidate, and make choices to optimize any business metric.

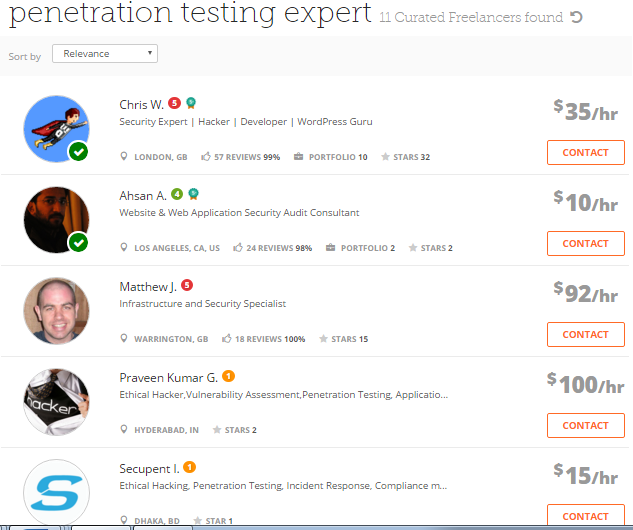
**D**epending on the final specifications of the backend that you finally choose, I would recommend the **Google Cloud Platform paid every quarter.**

1. Cost of roughly **$805.00** **per quarter** (**possibly going up to roughly $1,200** **per quarter** after additional service needs are identified during development phase)
2. Initially 2 servers (1 database server & 1 system server /1 development server & 1 production server), 2 CPUs, 7.5GB RAM, 15GB CentOS Linux Server, hosted in USA, 375GB SSD for data storage

**System Security**

[Weaponized] Penetration testing is needed before going live. This testing will include backend testing, admin front end testing physical code security, server security, and even organisational processes. Since the application is largely financial, and people’s real data is involved (names, locations, addresses etc.), it is necessary to have a reasonable level of protection.  
  
Unfortunately this cost varies greatly depending on the country, app complexity, skill level of the “hacker” etc. I would recommend the best available guy, coming in for much less time. The best guy tends to have the most efficient tools, tests the most relevant scenarios, gives the best remedial advice, and leads to the highest value for money.

Below is an example of the varying costs:



**Additional Configuration Time & Minor costs**

1. **Linux server** – We need to create Linux Virtual Machines with all required services and programs for the test, live and database servers. Linux and all its software is open source, so there is no cost apart from time.
2. **GIT server** – Code is an asset of the business and must be kept in a repository for future updating. GIT is the best software repository that does this job. For the automation process, a github.com account will be required. Checking in code triggers a build that generates the resulting libraries to be deployed onto the test/live servers. GIT hub is free if you work on code that is available to the public. We need the code base to be private. Private plans start at **$7 per month**.
3. **Jenkins server** – Jenkins is a Continuous Integration (CI) tool that pulls code from github.com and builds it automatically into libraries & binaries to deploy on the test/live servers. It will be free unless we require special plugins later.
4. **JBoss –** JBoss/Wildfly is a Java application server that loads all the libraries and binaries to run them as an application. It also runs the JVM (Java Virtual Machine), can manage connections to the databases, runs the SSL encryption, and more. Free of charge to install**.**
5. **Database server** – MySQL server may be used for the project.

1. **CHEF server and system –** CHEF is an automation tool that scripts the configuration of the entire system (provisioning) starting from code building, code deployment, installation of the Linux virtual machine, database setup, firewall rules, failover etc. It is basically a giant script that takes code and spits out a fully running system. Running the CHEF script on your computer will basically install all required files and set them up until your computer is a fully running server. CHEF is possibly very expensive and this step will take a very long time (2 weeks minimum). This may be left out of the first version.
2. **Database replication –**Database backup is an industry standard, it is essential to have a replicated DB.
3. **Application security** – After the code is functional, some time will be spent thinking of security such as max password attempts, traffic spikes (which could be a hack attempt), invalid data sent to the application, malicious inputs (such as usernames that match code, which could trigger SQL injection hacks) etc. The application will need ways to filter out such security concerns, and suspend/block certain users to prevent breaches.
4. **Application logging/monitoring/reporting –** We will need a way to know how many people are transacting, how many errors have occurred, what % of the system resources are currently being consumed. We need some kind of developer console to monitor application health. This is achieved largely free of charge, but more advanced/useful tools may have a cost implication.
5. **Failover/Load balancing/Redundancy –** We will need an automated system to detect when a network/database failure occurs, or one server is overwhelmed, which will automatically change over to a redundant server to continue processing seamlessly. Certain events (like a database outage) will trigger email/SMS alerts.
6. **Server security** – Each server will have its own password (so that if someone gets it, you can’t break into other servers, and each admin console (JBoss, SMS, Analytics, Firewall, Database, Jenkins, GIT) will also have its own password, which will also be different across the test and the live servers. A password expiry policy might be used and even different users (dev, admins, and clerks) will have different user accounts etc. The firewall will need to be configured manually for each server the first time.
7. **Keepass –** No one person can remember all logins & passwords, and different people are given different access. Keepass will manage the password repository to make handovers simple as well.
8. **Mail server –** All mail (registration emails, marketing material, reports, alerts etc.) from the server will go via an internal mail server which will need to be configured.