**Feel free to mark up with comments and questions...**

**Executive summary:**

This product will be a multi-part web service enabled application logger. The goal of the application is to have a set of web services where other applications can log information about their own applications and processes. Other applications in our enterprise SOA system will log job status and progress messages to the logger web services.

The application will consist of 3 major components:

1. A SQL database for storing messages. The database will include stored procedures for accessing the database and queries that are optimized for fast retrieval of messages.

2. A set of web services for inserting and retrieving log information in the database. Other applications will call these web services when logging information about themselves.

3. A web based user interface "dashboard" that can be used to monitor the system in real time. It will display the status of processes and jobs currently logging messages to the system. The dashboard will get all it's information by calling the web services.

**Key Points:**

- The solution is N-Tier. Each of the 3 parts can exist on it's own piece of hardware (or VM server).
- The technologies are Microsoft - C#, .net, SQL, WCF, ASP.net
- Clear separation of concerns and fits into our service oriented architecture.
- Quality is #1

**Use Cases, User Stories, application requirements**

As an application process, I would like to be able to log a message or information to a web service, so that my status can be viewed at some later time.

- LogMessage(processName, message)
- LogMessageWithPriority(processName, message. Priority)
- LogErrorMessage(processName, errorMessage)
- LogInformation(processName, informationMessage)
- LogCriticalError(processName, criticalErrorMessage)
- LogJobStart(processName, jobName, optional jobcount)
- LogProgress(processName, jobName, plannedCount, completedCount)
- LogProgressPercent(procprocessNameessID, jobName, percentComplete)
- LogJobComplete(processName, jobName)
- LogStartup(processName)
- LogShutdown(processName)
- LogProcessState(processName, state (red, yellow, green or similar))

- .
- Add more as we need to. If there's a method we can expose that will make the calling client code simpler, then add it here...
- Some of these methods don't take a string as a message. They will probably log a generic "canned" message into the log database ie "Process XXX just started".

Process Name is self proclaimed by the client that is calling the web service methods. It is assumed to be unique per process, but it really doesn't matter. The calls to retrieve data later will just return data for a process with the given name.

When reporting on all processes, we just look for all the unique names in the system and assume them to each be a unique process...

~~Process ID is optional. If used, then we could use a method to register a process and associate the process information with an ID. Optionally, we could just use a self proclaimed process ID or name. When reporting we would then assume each unique process name indicated a unique process. I'm leaning toward the latter option just to avoid unnecessary storing of process identifiers. Seems a sel f proclaimed process name would be adequate.~~

All log entries need to be time stamped.

Log entries need to in some way expire in order to not infinitely log and fill up disk space. Either by number of messages, disk size, expiration time limit, etc. Handling both per process and system totals would be helpful but not required. Recommend using configurable numbers and doing a clean/purge on a separate thread that is spawned when a web method is called.

As a monitoring process (GUI), I would like to be able to call web services to retrieve status and messages about a specific process or all processes in order to view them.

- GetAllMessages(processID)
- GetLastMessage(processID)
- GetProgress(processID)
- GetAllMessagesBasedOnPriority(messagePriority)
- GetAllStartupAndShutdownMessages()
- GetAllProcessesSummary().
- ...
- Add more as we need to. If there's a method we can expose that will make the calling client code simpler, then add it here...

As a software developer, I need to be able to use Microsoft Visual Studio to add a "Service Reference" in order to access the logging service web methods. This means the web service needs to end up being hosted in IIS, with a .SVC file and WSDL accessible via URL

As a monitoring user, I would like to go to a website and view a summary status of all processes that have been logging to the system..

The website should get ALL data it needs by calling the web services.

The main entry page should show summary information without the user having to click on anything.
Each entry in the summary should be clickable to drill down and view more detailed log information for a specific process.

The UI should show in graphical format the following concepts

- State - red, yellow, green
- Job status - if a process has logged a job that is being worked on.
- Last logged Message

The UI needs to be usable cross-browser. Emphasis can be placed on touch oriented UI for use with tablets, handheld, phones, etc as well as traditional desktop browsers. Chrome, FF w/Firebug, and IE9 all have really good dev/debug environments.

HTML5 is pretty neato....

Table display of log messages should be sortable. Default sort is on the timestamp with newest at the top.

Table display of processes should be sortable. Default sort is by status, with red at the top, then yellow then green.

**Technical requirements**

- All data is accessed through web services - Web service is the ONLY part of the system that can access the database
- Data needs to be stored in a Microsoft SQL database.
- No continuous running services or processes. When a web service method is called, perform the desired action and return. The only time anything is happening is when it is triggered by a user or process that is consuming the web services. The exception to this might be a spawned thread that is purging expired data...
-

**Performance requirements (we can tweak these depending on how far we get)**

- UP TIME!!! the logger pretty much is not allowed to crash. ever. period. This is where all the other apps will indicate that they had an issue, so it needs to be up.
- Any web service call that is POSTing data needs to have a response time of just a few seconds. Consider spawning threads to do the work so the web method can just return and let the calling app go about its business. This has a downside of not being able to return success/failure, so weigh out the pros and cons before implementing it with threads...

- Any web method call that is GETing data needs to have a response time of less than 30 seconds. If more than 10 seconds, consider adding in a “busy” GIF image to indicate that some report data is being generated
- Needs to support multiple simultaneous connections from other applications

**Design Hints (not required, but my suggestions)**

- Don’t put any SQL queries in the code. Use stored procedures in the SQL instance or .NET LINQ queries.
- Use MVC for web UI. The models for your MVC can be composite objects based on the return types of the web service calls which are code generated when you add a “service reference”
- Consider using AJAX calls to get the data from the web server to the UI. This will enable refreshing the data in the UI in real time without refreshing the page.
- If there are performance issues with digging through the log data, try refactoring into a stored procedure or SQL query of some sort. SQL is very fast at digging through elements based on search criteria.
- Pay close attention to code quality and maybe spend a few minutes learning about exception handling in C#. It’s a bottomless pit, but there is good information out there that can help you get started.
- Take a glance at built in tools in Visual Studio. They have built in tools for unit testing, service testing, hosting WCF services, etc. It might be cleaner and faster than trying to integrate a bunch of 3rd party tools into the solution.

Fun Links
http://www.codecapers.com/post/Build-a-Dashboard-With-Microsoft-Chart-Controls.aspx
just for ideas about logging - http://msdn.microsoft.com/en-us/library/system.diagnostics.eventlog.aspx
http://msdn.microsoft.com/en-us/library/bb384396.aspx