Jonathan Turner 10.11.2016

# Mini Design Document- Blue Cross Blue Shield of North Dakota Applicant project – Employee Leave Tracker.

## Purpose

This purpose of this document is to describe the approach to designing a hypothetical employee leave/vacation application architecture.

## Key Definitions

* Leave Request – Collection of data compromised of the following items: Employee ID, status(approved/denied), begin date, end date, reason, comments
* Employee – person requesting a leave.
* Manager – person approving a leave request.
* Employee ID – unique identifier of the person making a request.
* Manager ID – unique identifier of the person approving a request.

## Key Components

For maximum flexibility, the system should be broken down into sever layers/tiers to isolate functional concerns, with each layer implementing a clearly defined interface or façade.

This system should have 3 broad areas:

### Client interface

This could be almost anything- web page, mobile application, command line or stand alone application and represents an employee making a request or a manager approving requests.

### Server API

**API**- This is some sort of API the client uses to get and save leave requests. The API could be a library used by the client application, SOAP Web services, or RESTFUL web services for example. Operations include: Get, Put, Delete, Approve.

**Business Logic**- The API contains the ‘business logic’ associated with leave requests and is responsible for Authentication and Authorization of the user to use a given API. Also responsible for coordinating with & employing any databases/ERPs involved in the Leave handling.

### Persistence Layer

This represents the interface to persistent storage such a database or a larger Enterprise ERP system.

## Other/Future Considerations

* Client/Server communications – could be modified to any protocol(HTTP,etc) or content type(serialized java objects, JSON, whatever)
* Container- The Server API could run inside a container such as Java EE or Spring, which would allow the API logic to be free from handling persistence connections and Authentication/Authorization issues and would increase scalability performance.
* Reporting- The Server API will likely need to support Leave Request Reporting in the future.
* Persistence- Could alternatively employ various technologies- Hibernate, JPI, straight JDBC, or even flat files.
* Other- Error and excepting handling, automatic unit testing at build time