Chris Farnsworth & Sean Klink

Software Architecture

Assignment 2

Migrating PieMatrix to Microservices Strategy

Introduction:

This document outlines a strategy to shift PieMatrix’s monolithic application gradually to a network of microservices. The UI facing the user that is currently in place will remain largely the same – rather every component on the backend will change, one at a time, as possible.

Our strategy commences by converting one-off services like file uploading/downloading and their attached services to microservices. We begin with isolated, simple services like encryption to allow the development team to gain experience with this new style of thinking, before moving the remainder of services, organized by complexity/impact on the overall system.

The full list of services, in our proposed order of implementation, is highlighted below:

Microservices:

Auxillary Tools

File Encryptor

File Decryptor

Send Manual Notification

-Severity assigner

-notification queue modifier

Send Auto Notification

-Get Current date

-Scan active projects for corresponding dates

-Severity assigner

-Notification queue modifier

Notification Sender

File Uploader

-Version History by date

-Update file version

File Downloader

-Version History by date

-Select specific file version

File Database Updater

File Database Accessor

Search within file (if text file)

Convert to microsoft project

-Convert each XML element individually

Decode from microsoft project

-Decode each XML element individually

Sync With microsoft project (do both)

Formatters for Frontend

Social Feed View

-Recent Project Summarizer

-Recent Projects Attached to this user at a glance

-Convert project on the backend to Project view for the frontend

-Recent Chain of messages from message app

Process Authoring

-Create Project UI (see core data, we need one UI service for each of the main categories that feeds user input to that category)

Process Execution

-Display Status for project elements

-Input project, return XML for elements with status

-Conglomerate XML into a complete display status

-Update Project UI (see core data)

-Determine what’s modified via XML element type

-Update service

Message App

-Get Messages

-Formatter

-Grabber

-Validate Recipient

-Database Queue Manager - Receiver (One instance per user)

-Send Messages

-Formatter

-Sender

-Validate Recipient

-Database Queue Manager – Sender

Dashboard

-Project Summarizer

-Projects View Constructor

Core Data

Roles

-Role Determiner (Accepts Project Element, User)

-Role Modifiers

-Updater

-Deleter

-Accessor

Version Verification

-Major Version

-Minor Version

-Validate Project

-Determine least version in project chain

-Validate all against version (by walking until all elements are visited)

Project Data (access, update, delete, create, construct XML for each)

-Project

-Process

-Stage

-Box

-Step

-Infinite chain of substeps

-Data looks like:

Database ID / String ID / Fields as needed / Status / start date / duration / link to issue table

-Get Issues for element (for the severities user wants)

-Get severity for issue

-Determine end date for element (using duration and start date)

-XML Translation per element

-Children per element

-Attached Files Per Element

-Import from template (one service per main element)

-Load Template

-Export to template

-Save Template

User (w/ schedules)

-Create, Update, Delete, Access

-Assign User to project element

-Determine if user is available at time block

Scheduler

-Construct schedule

-Validate times for all elements of a project

-Determine date

-Display actual progress of project

-Display projected progress of project