Design Documentation

Message Logger API v1

Introduction

Message Logger is a web service that logs client messages. It has 2 major components. That are the core web service and a web based client. Web based client provide a web user interface for accessing all the endpoints exposed by the underling web service. Web service is currently having versioning with the version 1.0. It has only 3 exposed methods to register, authenticate and log. Basic authentication is used by the web service to authenticate client applications before allowing them to send logs.

Technology Stack

- C# 6.0
- .NET Framework 4.6
- ASP.NET MVC 4
- ASP.NET Web API 2
- C# Ninject
- HTML5/CSS3
- Bootstrap
- Entity Framework 6
- MS Tests
- MS SQL Server 2008 R2
- Visual Studio 2015
- Fiddler

Design Patterns Used

Repository Pattern

Data access was wrapped by repositories allowing the future application to change the data access technology without affecting the higher levels of the application.

Dependency Inversion

Favoring Interface over implementation was done introduced throughout the project to allowing the application to have the flexibility to unit test.

Test Driven Development

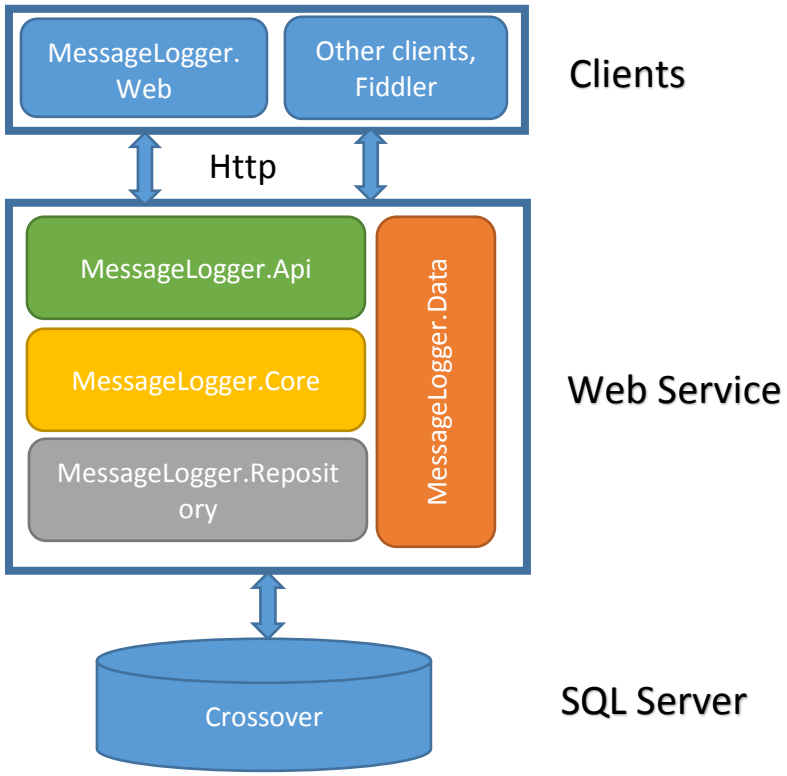
Unit tests were added when possible to test all the layers of the project.

Model-View-Controller Pattern

This pattern was inherited from the MVC framework when developing the web client.

Architecture

Components and modules for the entire project is described in the diagram below. 3 tiers on the web service communicate with only to the adjacent tiers and has access to the domain models.

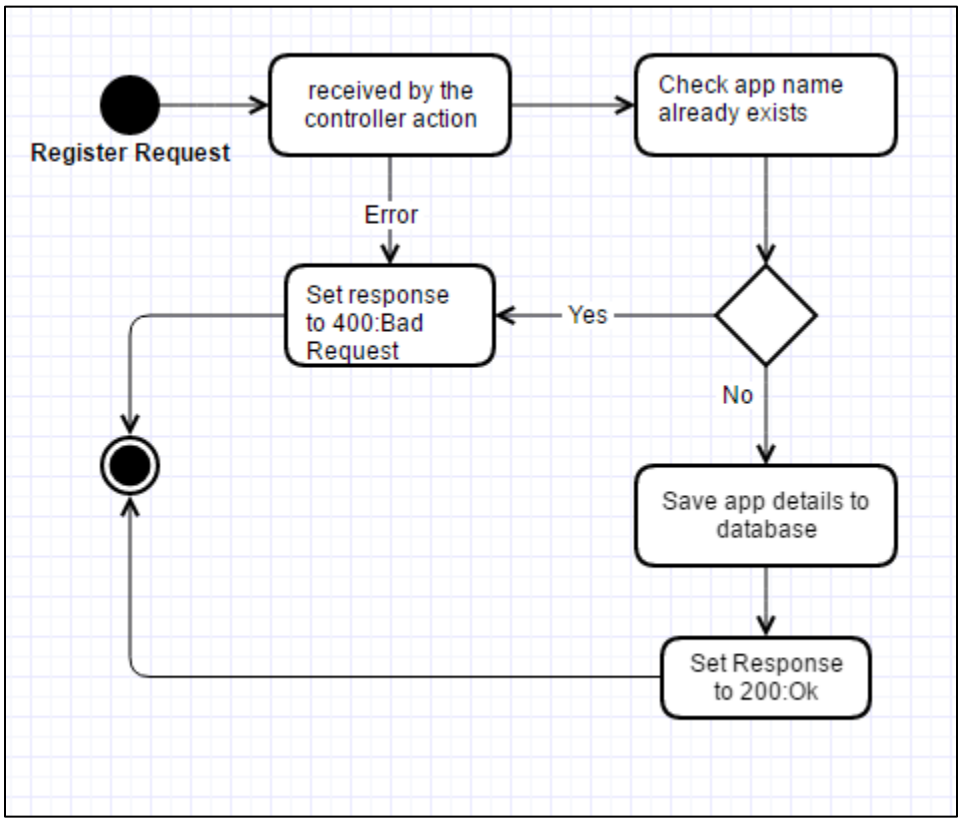


Component	Description
MessageLogger.Web	This is the ASP.NET MVC Web client for the Message Logger API. It accesses the web service through http messaging.
MessageLogger.Api	Exposes the API endpoints, manage session and throttling
MessageLogger.Core	This component contains the business logic of the API.
MessageLogger.Repository	Act as the data access layer for the web service. Hide away the underlying data access framework and give the flexibility to change the data access framework or DBMS in future.
MessageLogger.Data	This defines domain specific data models and is used in all the layers in the web service

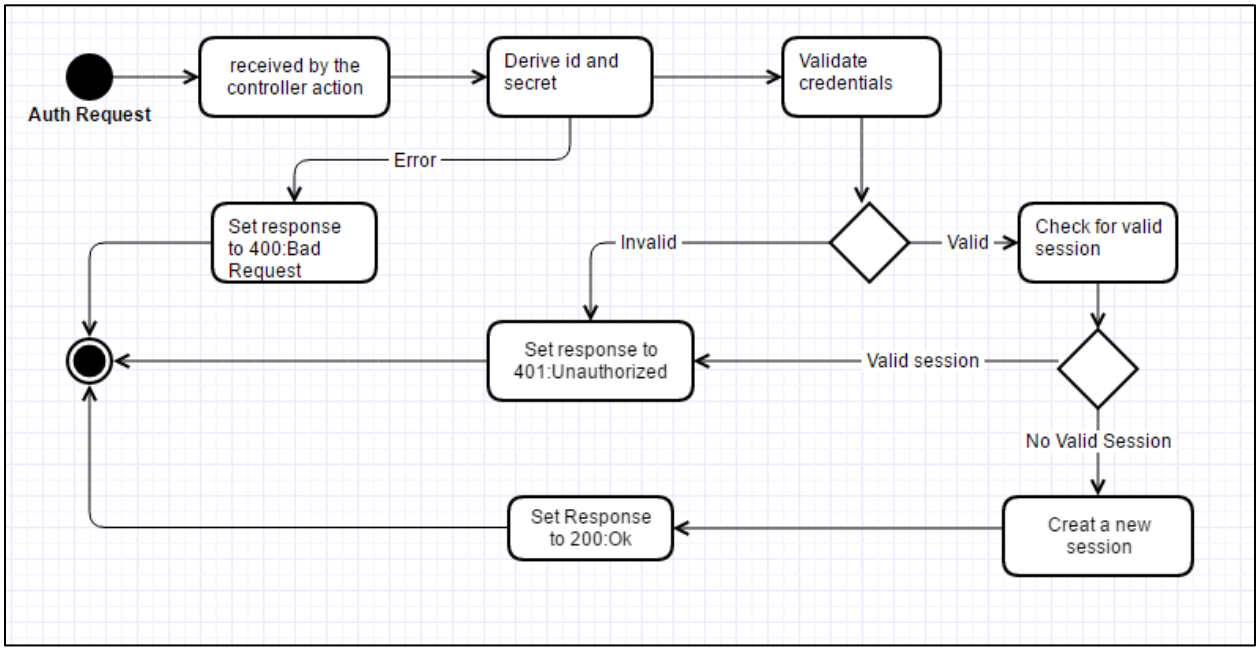
Design Decisions

1. Use of Http Runtime Cache to manage session data and throttling related data
    - No notification sent when client decides to abandon the session. Hence sessions/request history need to be expired and removed from the storage.
    - Session/request history details are accessed for every log request, So the query to the database may result in a big performance hit.
    - Occasional loose of session/request history is not harmful
    - Since HttpSession not supported by Web API, and storing in database is not efficient, and maintaining a singleton static session repository in the server memory may lead to inefficient memory usage I decided to use the http runtime cache to solve this problem.
2. Saving session lifetime retrieved by the database in Http Runtime cache.
    - Session lifetime need to be read for every auth request log request fetching it from the database every time will create a big performance hit.
    - Hense I read it once and put it in the globally accessible cache of the server memory.

Register Endpoint Activity Diagram



Auth Endpoint Activity Diagram



Log Endpoint Activity Diagram