Health Journal Web Service using Cloud Computing

Ryan Hausen, **Samuel Sambasivam**, **Simon H Lin**
Department of Computer Science, College of Liberal Arts and Sciences,
Azusa Pacific University, Azusa, CA 91702, USA

Abstract - *Technology continues on a trend to a stronger dependence on the cloud. Making web services easy to create and easy to manage has taken large strides in recent years, and there has been a push for more RESTful web services [3]. In this same stride, software has been leaning towards the implementation of software as a service which can utilize RESTful web services to access databases or other backend services. In this paper, we explore how to implement a simple RESTful web service using the recently released Microsoft Web API 2.1 backed by SQL server. Using this impressively easy technology with features like attribute routing and the new IHttpActionResult interface, receiving and sending HTTP requests and responses comes very naturally to anyone who is familiar with .NET programming and some standard web protocols.*

Keywords: web service, software design, programming, database, cloud computing

1 Requirement

The outcome of this project will be a RESTful web service, henceforth referred to as *the Service*. The Service will support a Health Journal storage system, by which users can submit a daily journal and retrieve it later. Each journal should be able to track the following six pieces of information:

1. Date that the journal represents
2. How the person was feeling that day
3. How much the person weighs that day
4. How far the person traveled in exercise that day
5. How many calories that person consumed that day
6. Any extra notes that person would like to associate with that day

In addition to being able to submit data to the Service, the Service should be able to respond with a single journal or a series of journals filtered by date so that users can monitor their results over time.

The Service will provide the basic Create, Read, Update, and Delete functions. These will be exposed in intuitive functions that will make it easy for client software to utilize its features for the benefit of users.

2 High-Level Design

The design pattern that the Service architecture should follow will be based on REST or Resource Oriented Architecture. In this design pattern, the resources are represented by URI's that utilize standard HTTP methods (POST, PUT, GET, and DELETE) to perform operations against the data represented by the URI, at which the methods are applied.

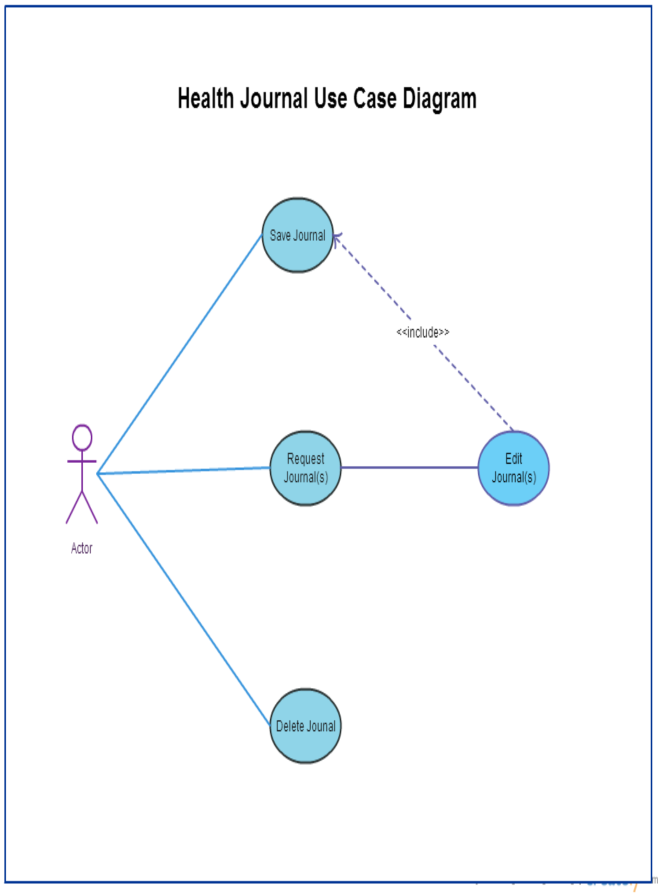

Figure 1: Use Case Diagram for Managing Health Journals

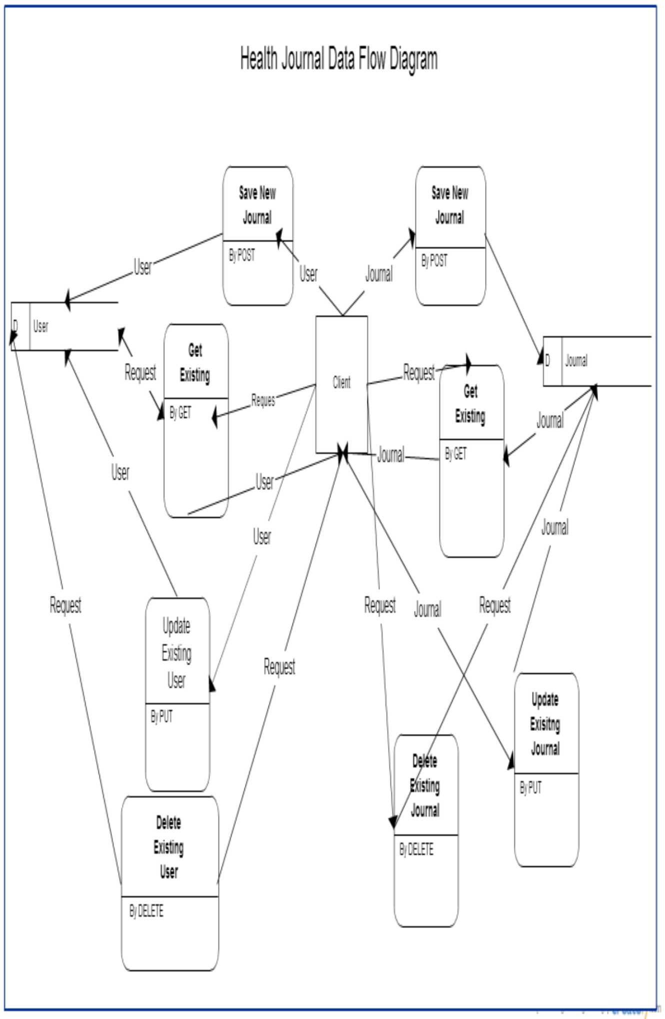

Figure 2: Data Flow Diagram for the Health Journal Web Service

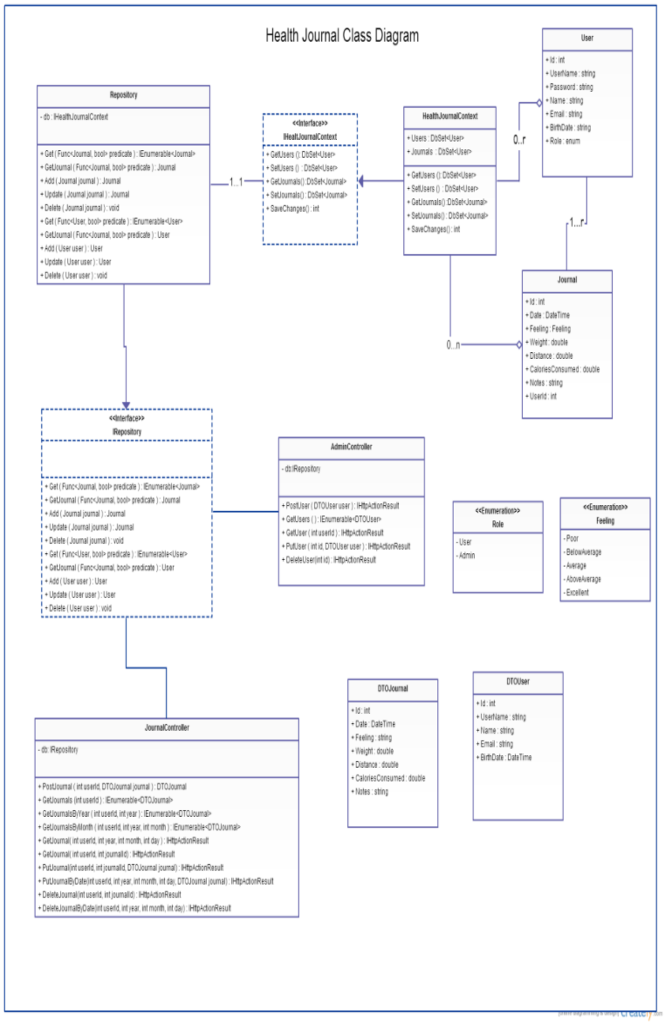
Health Journal Class Diagram
Repository
- db : IHealthJournalContext
+ Get (Func<Journal, bool> predicate) : IEnumerable<Journal>
+ GetJournal (Func<Journal, bool> predicate) : Journal
+ Add (Journal journal) : Journal
+ Update (Journal journal) : Journal
+ Delete (Journal journal) : void
+ Get (Func<User, bool> predicate) : IEnumerable<User>
+ GetJournal (Func<Journal, bool> predicate) : User
+ Add (User user) : User
+ Update (User user) : User
+ Delete (User user) : void
1...1
<<Interface>>
IHealtJournalContext
+ GetUsers (): DbSet<User>
+ SetUsers () : DbSet<User>
+ GetJournals() :DbSet<Journal>
+ SetJournals() : DbSet<Journal>
+ SaveChanges() : int
HealthJournalContext
+ Users : DbSet<User>
+ Journals : DbSet<User>
+ GetUsers (): DbSet<User>
+ SetUsers () : DbSet<User>
+ GetJournals() :DbSet<Journal>
+ SetJournals() : DbSet<Journal>
+ SaveChanges() : int
0...r
User
+ Id : int
+ UserName : string
+ Password : string
+ Name : string
+ Email : string
+ BirthDate : string
+ Role : enum
1...r
0...n
Journal
+ Id : int
+ Date : DateTime
+ Feeling : Feeling
+ Weight : double
+ Distance : double
+ CaloriesConsumed : double
+ Notes : string
+ UserId : int
<<Interface>>
IRepository
+ Get (Func<Journal, bool> predicate) : IEnumerable<Journal>
+ GetJournal (Func<Journal, bool> predicate) : Journal
+ Add (Journal journal) : Journal
+ Update (Journal journal) : Journal
+ Delete (Journal journal) : void
+ Get (Func<User, bool> predicate) : IEnumerable<User>
+ GetJournal (Func<Journal, bool> predicate) : User
+ Add (User user) : User
+ Update (User user) : User
+ Delete (User user) : void
AdminController
- db:IRepository
+ PostUser (DTOUser user) : IHttpActionResult
+ GetUsers ( ) : IEnumerable<DTOUser>
+ GetUser (int userId) : IHttpActionResult
+ PutUser (int id, DTOUser user) : IHttpActionResult
+ DeleteUser(int id) : IHttpActionResult
<<Enumeration>>
Role
- User
- Admin
<<Enumeration>>
Feeling
- Poor
- BelowAverage
- Average
- AboveAverage
- Excellent
JournalController
- db: IRepository
+ PostJournal (int userId, DTOJournal journal) : DTOJournal
+ GetJournals (int userId) : IEnumerable<DTOJournal>
+ GetJournalsByYear (int userId, int year) : IEnumerable<DTOJournal>
+ GetJournalsByMonth (int userId, int year, int month) : IEnumerable<DTOJournal>
+ GetJournal(int userId, int year, int month, int day) : IHttpActionResult
+ GetJournal(int userId, int journalId) : IHttpActionResult
+ PutJournal(int userId, int journalId, DTOJournal journal) : IHttpActionResult
+ PutJournalByDate(int userId, int year, int month, int day, DTOJournal journal) : IHttpActionResult
+ DeleteJournal(int userId, int journalId) : IHttpActionResult
+ DeleteJournalByDate(int userId, int year, int month, int day) : IHttpActionResult
DTOJournal
+ Id : int
+ Date : DateTime
+ Feeling : string
+ Weight : double
+ Distance : double
+ CaloriesConsumed : double
+ Notes : string
DTOUser
+ Id : int
+ UserName : string
+ Name : string
+ Email : string
+ BirthDate : DateTime

Figure 3: Class diagram for the Health Journal Web Service

3 Low-Level Design

The particular implementation of the RESTful architecture pattern for this project will be Microsoft's Web API 2.1 in the .Net environment, and using C#/SQL as the languages of implementation. The target framework [1] for the project is .Net 4.5 and will be hosted on a Microsoft IIS server. The data will be stored using Microsoft's SQL Server 2012.

The dominant format of data on the web currently is XML. Another format, JSON is growing in popularity. JSON is a lightweight form of XML that simplifies data transmission in a way that is easily understood in JavaScript and consequently most browsers.
The class diagram has been designed with all attributes [4] and relationships verified and validated. The data model has been designed with all attributes and relationships verified and validated [5]. The class diagram and the database diagram are shown below.

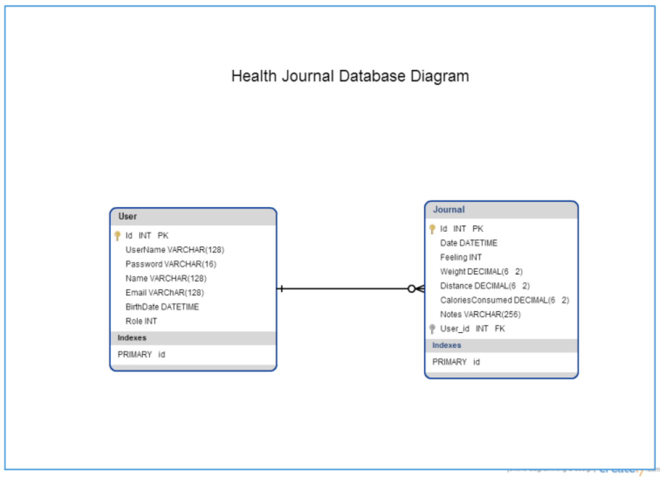

Figure 4: Database Diagram for the Health Journal Web Service

4 Coding

The coding has been completed successfully with approximately 3000 lines of code. The complete source code is available upon request.

5 Testing

Unit testing [2] and system testing have been completed accordingly with some complex testing scripts being designed and developed.

6 Product Demo

To demonstrate the Service, Dev HTTP Client, available in the Google Chrome App Store was used to generate HTTP requests and parse responses. To find out more about Dev HTTP client, please go to: https://plus.google.com/104025798250320128549.

The following is a sample list of actions a user or a service administrator can perform.

1. *Create a new user:* A new user is created using JSON format and sent using the POST method. Note that the server response excludes the new user's password and includes the user's Id. The response also includes the URI for new user in Location Header.
2. *Add a new journal to the created user's journals:* A new journal is created using JSON format and sent using POST, referencing the created user's Id in the URI.
3. *Retrieve an existing journal:* An existing journal is retrieved by making a GET request to the URI of the journal.
4. *Update an existing journal with new information:* An updated journal is sent over referencing an existing journal's Id in JSON format using the PUT method to the URI referencing the Journal's date. Note that the "Notes" property of the JSON object has been updated to reflect the changes.
5. *Delete an existing Journal:* A DELETE request is sent to the URI of the journal; this time we use the Id of the journal rather than the date to alter the resource.

7 References

[1] FitzMacken, Tom (2013). "*Mocking Entity Framework when Unit Testing ASP.NET Web API 2", http://www.asp.net/web-api/overview/testing-and-debugging/mocking-entity-framework-when-unit-testing-aspnet-web-api-2#dependency*

[2] FitzMacken, Tom (2013). "*Unit Testing ASP.NET Web API 2",* http://www.asp.net/web-api/overview/testing-and-debugging/unit-testing-with-aspnet-web-api#addtoexisting

[3] Richardson, Leonard & Ruby, Sam. "*RESTful Web Services*", Web services for the real world, O'Reilly Media, 454 pages, 2007.

[4] Wasson, Mike (2014). "*Attribute Routing in Web API 2",* http://www.asp.net/web-api/overview/web-api-routing-and-actions/attribute-routing-in-web-api-2

[5] Wasson, Mike (2012). "*Model Validation",* http://www.asp.net/web-api/overview/formats-and-model-binding/model-validation-in-aspnet-web-api