

LiteBulb Labs

RunLog Application

Solution Document v0.1

**Abstract**

This document describes the solution for the LiteBulb Technologies RunLog application:

design requirements, constraints, component diagrams, and data models.

John Bianchi

jmbianchi@gmail.com

Table of Contents

[Problem Statement 3](#_Toc47264962)

[Requirements 3](#_Toc47264963)

[Assumptions 3](#_Toc47264964)

[Solution Overview 4](#_Toc47264965)

[REST API 4](#_Toc47264966)

[Client-Side Web UI 4](#_Toc47264967)

[Automated Tests 4](#_Toc47264968)

[System Diagram 5](#_Toc47264969)

[Activity Diagram 6](#_Toc47264970)

[RunLog Application – Problem Description 6](#_Toc47264971)

[Models 8](#_Toc47264972)

[Activity entity 8](#_Toc47264973)

[Position entity 8](#_Toc47264974)

[Enums 9](#_Toc47264975)

[ActivityType enum 9](#_Toc47264976)

[ActivityStatus enum 9](#_Toc47264977)

[DistanceUnits enum 9](#_Toc47264978)

[Status Messages 10](#_Toc47264979)

[Error Messages 10](#_Toc47264980)

[Message Header Keys 11](#_Toc47264981)

[Config File Keys 11](#_Toc47264982)

# Problem Statement

The RunLog application needs a REST API to provide data storage and retrieval functionality. The API needs to perform some back-end processing. The API needs to write log messages to an AMQP message broker. An interactive UI is needed to provide user access to all the CRUD operations supported by the application.

A solution called RunLog will be developed to perform the required operations.

# Requirements

1. RunLog application needs a logical data model that includes two entities with a cardinality relationship of one-to-many.
2. REST API that implements all the CRUD operations using a resource-based modeling approach for each data model entity.
3. REST API requires HTTPS security.
4. Each invocation of the REST API should write a small log message describing the given interaction to an AMQP message broker.

Example data:

*Please see the document:* ***runlog-rest-api-guide.docx***

# Assumptions

1. System implementation should be written in a recent version of .NET.
2. System implementation should adhere to the SOLID design principles.
3. The Web API project should have a composition root and use dependency injection to facilitate object
4. creation.
5. The Web API project should utilize layering to appropriately separate the different application concerns.
6. Application code should be very DRY.
7. REST API should use JSON as its default serialization.

# Solution Overview

The proposed solution is comprised of 3 distinct areas of concern:

1. **REST API** – to provide data access and back-end processing (ASP.NET Core Web API)
2. **Client-Side Web UI** – to interface with the system (ASP.NET Core Blazor WebAssembly SPA)
3. **Automated Tests** - unit tests and integration tests (written in .NET)

This document describes the design requirements, constraints, component diagrams, and data models related to each area of concern. This document is intended to facilitate collaboration between the different groups LiteBulb Labs working on the RunLog application and represents a starting point for the design process.

# REST API

Restful Web API written in ASP.NET Core 3.1 (using Kestrel). The API handles requests, accesses a database (for data model persistence), provides a response with information about Activity and Position objects submitted to the system. The API implements all the major CRUD operations. For example:

1. Get all Activities/Positions
2. Get Activity/Position by Id
3. Create Activity/Position
4. Update Activity/Position (no PATCH)
5. Delete Activity/Position

# Client-Side Web UI

Interactive client-side web UI written in ASP.NET Core 3.1 Blazor WebAssembly framework. This web site application is 100% client-side providing user interface functionality via a supported web browser. The web site assets are served from a static web hosting service (GitHub Pages) and not from an ASP.NET web server. The web site application exchanges data with the REST API web server by issuing HTTP requests via XMLHttpRequest (XHR). Data is retrieved from the API and rendered on the web browser without having to do a full-page refresh.

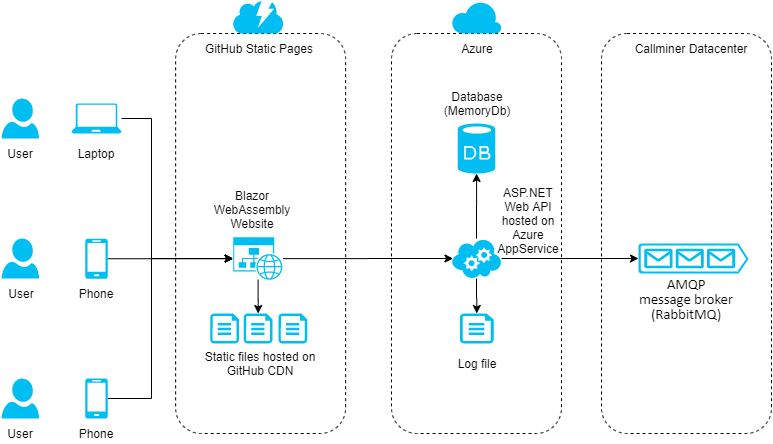
This provides user access to the application via a web browser. Authentication and authorization and user account management functionality will be added in the future.

# Automated Tests

Unit tests are written for every interface and public method. Tests are written in .NET using NUnit and Moq. Some more comprehensive integration tests have been developed as well.

# System Diagram

The following diagram depicts the implicated system components and their dependencies.



# Activity Diagram

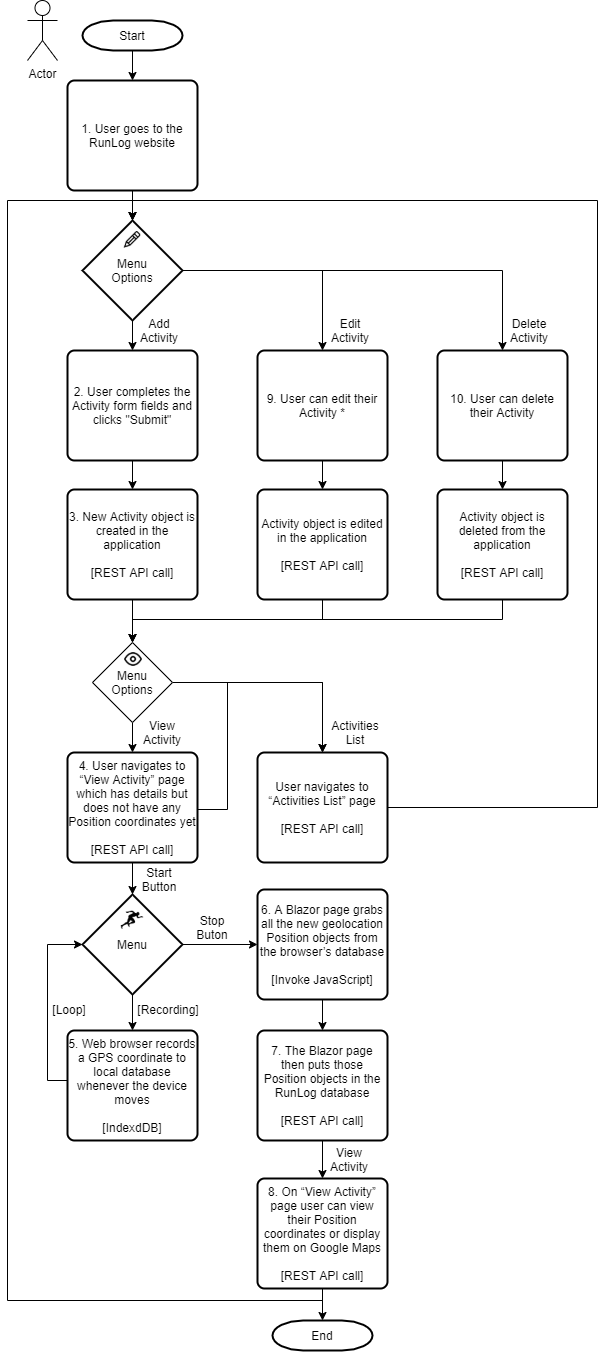
## RunLog Application – Problem Description

Someone in the world wants to record their running workout.

{1} User goes to the RunLog website and clicks on “Add Activity” menu option. {2} User completes the Activity form fields with their desired inputs and clicks “Submit”. {3} A new Activity object is created in the system (with no Position child objects yet). \* {4} User navigates to “View Activity” page (which has details but does not have any Position coordinates yet) and clicks the “Start” button. {5} This starts a GPS location coordinates watcher (by calling the web browser API via JavaScript) which generates a new geolocation Position object each time the GPS device is moved and stores them in the web browser’s database (IndexedDB). {6} When the user clicks the “Stop” button, a callback to the Blazor page is called which then grabs all the geolocation Position objects generated for that Activity from the web browser’s database (IndexedDB). {7} The Blazor page gets all the Position objects from the web browser’s local database and then calls the REST API to put each Position object in the RunLog database. \* {8} User views list of all the Position objects that were generated during their run or can also see them displayed on a map via Google Maps API. \* {9} User can edit their Activity. \* {10} User can delete their Activity. \*

User goes back to Step 1 to create a new Activity.

*\* Each invocation of the REST API writes a log message describing a given interaction to an AMQP message broker.*



# Models

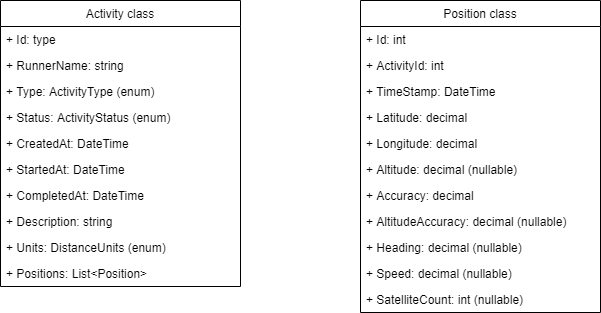
*(See UML diagrams below)*

## **Activity entity**

1. **Id**: Id of the Activity (generated internally by the database).
2. **RunnerName**: Name of the athlete doing the Activity.
3. **Type**: Type of Activity being performed (Walk, Run, Bike, Swim).
4. **Status**: Status of the Activity (Pending, Started, Paused, Complete).
5. **CreatedAt**: Time the Activity object was created in the system.
6. **StartedAt**: Start time for this Activity.
7. **CompletedAt**: End time for this Activity.
8. **Description**: Description containing extra details about the Activity.
9. **Units**: System of measurement units used for this Activity (i.e. English or Metric).
10. **Positions**: Collection of Position coordinates that belong to this Activity.

## **Position entity**

1. **Id**: Id of the Position (generated internally by the database).
2. **ActivityId**: Id of Activity parent object this Position object is associated with.
3. **TimeStamp**: DOMTimeStamp representing the time at which the location was retrieved. (DOMTimeStamp type represents an absolute or relative number of milliseconds, depending on the specification in which it appears.)
4. **Latitude**: Position's latitude in decimal degrees
5. **Longitude**: Position's longitude in decimal degrees.
6. **Altitude**: Position's altitude in meters, relative to sea level (this value can be null if the implementation cannot provide the data).
7. **Accuracy**: Accuracy of the latitude and longitude properties (in meters).
8. **AltitudeAccuracy**: Accuracy of the altitude expressed in meters (this value can be null).
9. **Heading**: Direction in which the device is traveling. This value, specified in degrees, indicates how far off from heading true north the device is. 0 degrees represents true north, and the direction is determined clockwise (which means that east is 90 degrees and west is 270 degrees). If speed is 0, heading is NaN. (If the device is unable to provide heading information, this value is null).
10. **Speed**: Velocity of the device in meters per second (this value can be null).
11. **SatelliteCount**: Number of satellites in view (this value can be null).



# Enums

**(See UML diagram below)**

## **ActivityType enum**

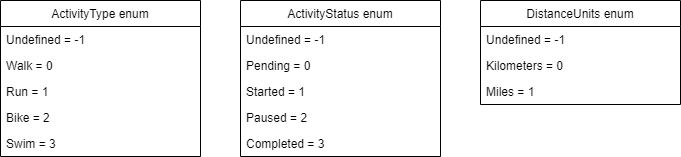
Type of Activity (i.e. is it a bike ride or a run?).

## **ActivityStatus enum**

Status of Activity (i.e. is it paused or complete?).

## **DistanceUnits enum**

Units of measurement for distance delta (i.e. English or Metric).



# Status Messages

1. Undefined = "Activity has not yet had its status defined."
2. Pending = "Activity has not started yet."
3. Started = "Activity is running."
4. Paused = "Activity has been paused."
5. Complete = "Activity has been finished."
6. Failure = "Activity has failed. See error message and system log for more information."

# Error Messages

1. Null result received while trying to get full list of Activity objects (Activity Service)

"Error occurred while retrieving paged list of Activity objects. Result was null for some reason."

1. Null result received while trying to get single Activity object by id (Activity Service)

"Error occurred while retrieving Activity object with id '{id}'. Activity object was not found in the database."

1. Exception thrown while trying to add new Activity object (Activity Service)

"Error occurred while adding an Activity object to database. {exception.GetType()} was thrown. Exception Message: {exception.Message}"

1. Null result received while trying to add new Activity object (Activity Service)

"Error occurred while adding an Activity object to database. Activity object returned by add process was null for some reason."

1. Exception thrown while trying to update Activity object (Activity Service)

"Error occurred while updating an Activity object. Update was not successful. Exception follows: {exception.Message}"

1. Null result received while trying to update Activity object (Activity Service)

"Error occurred while updating an Activity object. Update returned null for some reason."

1. Default object result received while trying to update Activity object (Activity Service)

"Error occurred while updating an Activity object. Activity with id: '{id}' was NOT found. Update was not successful."

1. Null result received while trying to delete Activity object by id (Activity Service)

"Error occurred while deleting Activity object with id '{id}'. Activity object was not found in the database."

1. Number objects deleted was 0 while trying to delete all Activity objects in system (Activity Service)

"Error occurred while attempting to delete all Activity objects from the database. Some or all Activity objects may not have been deleted."

# Message Header Keys

1. MessageType = "MessageType"
2. RetryCount = "RunLog-RetryCount"

# Config File Keys

1. Database connection string. Note: MemoryDb does not require a connection string currently.

ConnectionString = ""

1. Name of database for this application.

DatabaseName = "RunLogDb"

1. Name of collection for Activity object.

ActivityCollectionName = "Activity"

1. Name of collection for Position object.

PositionCollectionName = "Position"

1. Directory where log files are written to

LogDirectory = "LogDirectory"