# Wearable Device Scenario

## Overview

This test scenario is modeled on the usage of wearable devices such as step counters. A user wears a device and it counts their steps progressively throughout the day. The test scenario models a single day of data uploads from a fixed user population, interspersed with random requests to read the current step count from users.

## Server Endpoints

The server endpoints supported are as described below. The test client will generate valid requests.

**POST /userID/day/hour/stepCount**

Where:

userID ranges between 1 and user population size

day ranges between 1 and number of days to upload for a test (default 1)

timeInterval between 0 and 23 to represent hours in a day

stepCount is an integer between 0 and 5000

**GET /current/userID**

Returns stepCount

Where:

userID ranges between 1 and user population size

stepCount is the cumulative number of steps for the most recent day stored for a user

**GET/single/userID/day**

Returns stepCount

Where:

userID ranges between 1 and user population size

day is a specific day number that is stored for a user

stepCount is the cumulative number of steps for the specified day

**GET/range/userID/startDay/numDays**

Returns: stepCounts[]

Where:

userID ranges between 1 and user population size

day is a specific day number that is stored for a user

numDays is the number of days to returns step count totals for, including start day

stepCounts[] is the cumulative number of steps for each day specified and the total for all steps, ie {17900, 11234, 4900, 34024}

# Test Scenarios

## Single Day – Write heavy load

### Client behavior

For each day of 24 intervals/step counts for each user, with a maximum target load of concurrent N clients, and a number of requests per client of R, a write test should behave as follows, defined by 4 phases:

**WarmUp phase**: For the 0th time interval, N/10 threads randomly select users and issue R POST requests per thread.

**RampUp phase**: For the 1st to 4th time intervals, N/2 threads randomly select users and issue R POST requests per thread.

**Peak phase**: For intervals 5-19, N threads randomly select users and issue R POST requests per thread.

**RampDown phase**: For intervals 19-22, N/3 threads randomly select users and issue R POST requests per thread.

**CoolDown phase**: For intervals 23 and 24, N/10 threads randomly select users and issue R POST requests per thread.

Simultaneously, a fixed size thread pool of NC read clients (default 10) will issue R GET /current/userID requests for every test phase.

The read thread pool should be coordinated with the start of the upload phase for each interval, and hence the read request load should not vary during the test phases.

## TO DO – more to come

# Database

The database needs to store at least userID (unique) and the time series data for stepCount by hour and day.

Before a test starts, both single and multiple day, all step count data can be deleted, leaving only a single entry in the database for each user in the user population.

The database platform and data model are not defined by this specification.

# Performance Data Collection

The basic measures we wish to calculate are based on response time and throughput.

This means the round trip time (latency) for every request sent from the client to the server needs to be recorded, along with a timestamp so the latency over the duration of the test can be analyzed.

We also want to count the number of failed requests due to network/server issues. In a successful test, all requests should succeed as input values will be valid.

We also want to calculate the 95th (p95) and 99th (p99) percentile for response times.

To calculate throughput, the total number of successful requests (should be all requests for a valid test) should be divided by the total test duration to give the number of requests processed each second. We should also calculate the throughput each time the number of client threads increases/decreases so the effect of stepping up/down the request load can be analyzed.