Sportimo v2.0 Technical Report

# 

Sportimo Technical Report

Sportimo is a computer game that simulates a soccer match [[timetable / event timecard]] and gives the player the ability to bet on match events and win game points and other awards.

this document serves as a technical report of the game’s requirements, conceived system architecture and final implementation. This document is considered to be a live document, and will be updated upon future notable system modifications.

1. **Contents**

[ 1](#_Toc501444523)

[2 Requirements 7](#_Toc501444524)

[2.1 User requirements 7](#_Toc501444525)

[3 System Architecture And design 8](#_Toc501444526)

[3.1 Overview 8](#_Toc501444527)

[3.2 Conceptual view: entities and entity relationships 8](#_Toc501444528)

[3.3 Network Architecture 9](#_Toc501444529)

[3.4 Game server internal architecture 10](#_Toc501444530)

[4 Implementation 12](#_Toc501444531)

[4.1 Server Components 12](#_Toc501444532)

[4.1.1 Game Server 12](#_Toc501444533)

[4.1.2 Notification Server 13](#_Toc501444534)

[4.2 Storage Systems 13](#_Toc501444535)

[4.2.1 MongoDB 13](#_Toc501444536)

[4.2.2 Redis 13](#_Toc501444537)

[4.3 Client Applications 13](#_Toc501444538)

[4.3.1 The administration Dashboard 13](#_Toc501444539)

[4.3.2 Web application 13](#_Toc501444540)

[4.3.3 Mobile device (Android / iOS) applications 14](#_Toc501444541)

[5 Deployment and Load tests 14](#_Toc501444542)

[5.1 Deployment in a dedicated server cluster 14](#_Toc501444543)

[5.2 Stress Tests 19](#_Toc501444544)

[6 Specifications 24](#_Toc501444545)

[6.1 Data Models 24](#_Toc501444546)

[6.1.1 Answer 24](#_Toc501444547)

[6.1.2 Article 24](#_Toc501444548)

[6.1.3 Competition 25](#_Toc501444549)

[6.1.4 FavQuestion 25](#_Toc501444550)

[6.1.4.1 Answer 25](#_Toc501444551)

[6.1.5 GamecardDefinition 25](#_Toc501444552)

[6.1.5.1 OptionDefinition 26](#_Toc501444553)

[6.1.6 GamecardTemplate 26](#_Toc501444554)

[6.1.6.1 OptionTemplate 27](#_Toc501444555)

[6.1.7 MatchEvents 27](#_Toc501444556)

[6.1.8 Message 27](#_Toc501444557)

[6.1.9 Player 27](#_Toc501444558)

[6.1.10 Poll 28](#_Toc501444559)

[6.1.10.1 Answer 28](#_Toc501444560)

[6.1.11 Question 28](#_Toc501444561)

[6.1.11.1 Answer 29](#_Toc501444562)

[6.1.12 Scheduled-Matches 29](#_Toc501444563)

[6.1.12.1 ModerationService 29](#_Toc501444564)

[6.1.12.2 Segment 29](#_Toc501444565)

[6.1.12.3 MatchEvent 30](#_Toc501444566)

[6.1.13 Sponsor 30](#_Toc501444567)

[6.1.14 Standing 30](#_Toc501444568)

[6.1.15 Stats-mod 31](#_Toc501444569)

[6.1.16 Team 31](#_Toc501444570)

[6.1.17 User 31](#_Toc501444571)

[6.1.17.1 Achievement 32](#_Toc501444572)

[6.1.17.2 UserStats 32](#_Toc501444573)

[6.1.18 UserActivity 32](#_Toc501444574)

[6.1.19 UserGamecard 32](#_Toc501444575)

[6.1.19.1 Condition 33](#_Toc501444576)

[6.2 Game server API Endpoints 33](#_Toc501444577)

[6.2.1 Data module 34](#_Toc501444578)

[6.2.1.1 Search for an article 34](#_Toc501444579)

[6.2.1.2 Post a new article 34](#_Toc501444580)

[6.2.1.3 Retrieve an article by id 34](#_Toc501444581)

[6.2.1.4 Update an article by id 34](#_Toc501444582)

[6.2.1.5 Delete an article by id 34](#_Toc501444583)

[6.2.1.6 Post a new competition 35](#_Toc501444584)

[6.2.1.7 Retrieve a competition by id 35](#_Toc501444585)

[6.2.1.8 Update a competition by id 35](#_Toc501444586)

[6.2.1.9 Delete a competition by id 35](#_Toc501444587)

[6.2.1.10 Retrieve all competitions 36](#_Toc501444588)

[6.2.1.11 Delete all competitions 36](#_Toc501444589)

[6.2.1.12 Retrieve a match by id for a certain user id 36](#_Toc501444590)

[6.2.1.13 Post a new team player 36](#_Toc501444591)

[6.2.1.14 Retrieve a player by id 36](#_Toc501444592)

[6.2.1.15 Update a player by id 36](#_Toc501444593)

[6.2.1.16 Delete a player by id 37](#_Toc501444594)

[6.2.1.17 Retrieve all players in a team by team id 37](#_Toc501444595)

[6.2.1.18 Retrieve all players 37](#_Toc501444596)

[6.2.1.19 Cast a user vote in a poll identified by id 37](#_Toc501444597)

[6.2.1.20 Retrieve all polls in a match by match id 37](#_Toc501444598)

[6.2.1.21 Retrieve a poll by id 38](#_Toc501444599)

[6.2.1.22 Update a poll by id 38](#_Toc501444600)

[6.2.1.23 Delete a poll by id 38](#_Toc501444601)

[6.2.1.24 Retrieve all polls 38](#_Toc501444602)

[6.2.1.25 Post a new poll 38](#_Toc501444603)

[6.2.2 Gamecards module 39](#_Toc501444604)

[6.2.2.1 Retrieve a game card template by id 39](#_Toc501444605)

[6.2.2.2 Post a new template 39](#_Toc501444606)

[6.2.2.3 Update an existing template by id 39](#_Toc501444607)

[6.2.2.4 Update an existing game card definition by id and by match id 39](#_Toc501444608)

[6.2.2.5 Retrieve all currently available game card definitions by match id and user id 39](#_Toc501444609)

[6.2.2.6 Post a new played game card for a match id and user id 39](#_Toc501444610)

[6.2.3 Leaderpay module 40](#_Toc501444611)

[6.2.3.1 Post a new leaderboard 40](#_Toc501444612)

[6.2.3.2 Retrieve a leaderboard by a match id 40](#_Toc501444613)

[6.2.3.3 Post a new pool 40](#_Toc501444614)

[6.2.3.4 Update an existing pool by id 40](#_Toc501444615)

[6.2.3.5 Delete an existing pool by id 40](#_Toc501444616)

[6.2.3.6 Retrieve a pool by match id 41](#_Toc501444617)

[6.2.3.7 Retrieve a pool by match id and country 41](#_Toc501444618)

[6.2.3.8 Retrieve all timed pools 41](#_Toc501444619)

[6.2.3.9 Retrieve all timed pools by country 41](#_Toc501444620)

[6.2.3.10 Post a new prize 41](#_Toc501444621)

[6.2.3.11 Retrieve an existing prize by id 41](#_Toc501444622)

[6.2.3.12 Update an existing prize by id 42](#_Toc501444623)

[6.2.3.13 Delete an existing prize by id 42](#_Toc501444624)

[6.2.3.14 Retrieve all prizes 42](#_Toc501444625)

[6.2.3.15 Delete all prizes 42](#_Toc501444626)

[6.2.3.16 Post a new score 42](#_Toc501444627)

[6.2.3.17 Retrieve an existing score by id 42](#_Toc501444628)

[6.2.3.18 Update an existing score by id 42](#_Toc501444629)

[6.2.3.19 Delete an existing score by id 43](#_Toc501444630)

[6.2.3.20 Retrieve all scores 43](#_Toc501444631)

[6.2.3.21 Delete all scores 43](#_Toc501444632)

[6.2.4 Match moderation module 43](#_Toc501444633)

[6.2.4.1 Post a new live match 43](#_Toc501444634)

[6.2.4.2 Update the segment time of an existing live match by id 43](#_Toc501444635)

[6.2.4.3 Delete the segment time of an existing live match by id 44](#_Toc501444636)

[6.2.4.4 Reload an existing live match by id 44](#_Toc501444637)

[6.2.4.5 Retrieve an existing live match by id 44](#_Toc501444638)

[6.2.4.6 Post a new match event command for an existing live match by match id 44](#_Toc501444639)

[6.2.4.7 Retrieve all scheduled matches 45](#_Toc501444640)

[6.2.4.8 Update an existing scheduled match by id 45](#_Toc501444641)

[6.2.4.9 Retrieve an existing scheduled match by id 45](#_Toc501444642)

[6.2.4.10 Post a new scheduled match 45](#_Toc501444643)

[6.2.4.11 Delete an existing scheduled match by id 45](#_Toc501444644)

[6.2.4.12 Retrieve all active moderation services by match id 46](#_Toc501444645)

[6.2.4.13 Try to start a new moderation service for a match id 46](#_Toc501444646)

[6.2.4.14 Pause an existing moderation service by match id and service type 46](#_Toc501444647)

[6.2.4.15 Resume an existing moderation service by match id and service type 46](#_Toc501444648)

[6.2.5 Notifications module 47](#_Toc501444649)

[6.2.5.1 Update notifications user data by user id 47](#_Toc501444650)

[6.2.5.2 Retrieve all existing notification user data 47](#_Toc501444651)

[6.2.5.3 Push (send) a notification to a user id and match id 47](#_Toc501444652)

[6.2.6 Offline data module 47](#_Toc501444653)

[6.2.6.1 Update all teams by competition id 47](#_Toc501444654)

[6.2.6.2 Update all player statistics by team id 47](#_Toc501444655)

[6.2.6.3 Update team standings by competition id 47](#_Toc501444656)

[6.2.6.4 Update all team standings for all competitions 48](#_Toc501444657)

[6.2.6.5 Return all future match fixtures by competition id 48](#_Toc501444658)

[6.2.7 Questions module 48](#_Toc501444659)

[6.2.7.1 Post a new favorite question 48](#_Toc501444660)

[6.2.7.2 Retrieve an existing favorite question by id 48](#_Toc501444661)

[6.2.7.3 Update an existing favorite question by id 48](#_Toc501444662)

[6.2.7.4 Delete an existing favorite question by id 48](#_Toc501444663)

[6.2.7.5 Retrieve all favorite questions 49](#_Toc501444664)

[6.2.7.6 Post a new question 49](#_Toc501444665)

[6.2.7.7 Retrieve an existing question by id 49](#_Toc501444666)

[6.2.7.8 Update an existing question by id 49](#_Toc501444667)

[6.2.7.9 Delete an existing question by id 49](#_Toc501444668)

[6.2.7.10 Retrieve all questions by match id 49](#_Toc501444669)

[6.2.7.11 Post an answer by question id and user id 50](#_Toc501444670)

[6.2.7.12 Post an answer by question id and moderator id 50](#_Toc501444671)

[6.2.8 Users module 50](#_Toc501444672)

[6.2.8.1 Post a new user 50](#_Toc501444673)

[6.2.8.2 Authenticate a user 50](#_Toc501444674)

[6.2.8.3 Retrieve all users 50](#_Toc501444675)

[6.2.8.4 Retrieve an existing user by id 50](#_Toc501444676)

[6.2.8.5 Update an existing user by id 51](#_Toc501444677)

[6.2.8.6 Post a new message to users 51](#_Toc501444678)

[6.2.8.7 Get an existing message by user id 51](#_Toc501444679)

[6.2.8.8 Retrieve user activity by match id 51](#_Toc501444680)

[7 Database 51](#_Toc501444681)

[7.1 Collections 51](#_Toc501444682)

[7.2 Collection detailed 52](#_Toc501444683)

# Requirements

## User requirements

User requirements are evolved through previous Sportimo implementations and deployments, and are shaped after requests from the client.

In general, the user should be able to:

* log in the game upon each use with user credentials (a pair of user name and user password)
* view notifications coming in from the dashboard
* view polls
* answer to poll’s question (once)
* select the language of the application’s interface
* see the leaderboard which is the ranking of top player scores in descending order
* see the live (ongoing) and future scheduled matches (fixtures) in all game supported competitions
* enter a selected scheduled match, and
  + play an Overall game card in a scheduled match that has not yet started
  + see the score, timer and match events feed in a match that has started
  + view the available game cards that can be played in the match
  + play an Instant or Overall game card out of the available ones in a match that has started
  + view the played cards so far
  + see the match events and played cards being automatically updated and refreshed
  + get the 2 teams statistics
* see the team standings in each of the game supporting competitions
* logout from the game’s session

The application administrator should be able to:

* set up scheduled matches in a calendar
* moderate scheduled matches, start them, end them, progress through the match halves and generate match events, optionally set up a moderation service for automatically feeding match events
* manage users and their account information
* manage polls
  + manage questions and answers
* manage prizes
* manage publications
* manage soccer competitions (leagues)
* manage teams
* manage team players
* manage team standings
* manage leaderboards
* manage sponsors
* set up leaderboard sponsorship
* manage user notifications
* manage game card templates and game card definitions bound to scheduled matches

# System Architecture And design

## Overview

The Sportimo platform integrates mature technological components and is standing on a tested, popular and built for performance technology stack, without sacrificing security for performance. In terms of deployment, the following conceptual diagram depicts the main user agents involved in the game scenarios:

Sportimo Server

Player on Android client

Player on iOS client

Player on Web client

User Moderator

Automatic (machine) Moderator

*Image 1*

A manual match moderator or an automatic machine moderation service can input information into the Sportimo server. This information is then consumed by the players that are connected to the server. The players may have one of the 3 supported Sportimo clients for the three more popular platforms: Android mobile and tablets, iOS iPhone and iPads, desktop or mobile web browsers.

## Conceptual view: entities and entity relationships

From a conceptual view, the Sportimo platform manages the following main concepts:

* Match: the primary concept of the application, a soccer match between 2 teams. Includes other match-related entities:
  + Timeline of events occurred during the match
  + Match statistics that aggregate events of the same type in certain dimensions (team, player, time)
* Soccer Competition: National or international league or other soccer competition
  + Team standings: team ranking ordered by their points
* Team
* Team Player
* Sponsor
* User: account information about users
  + User activity: aggregation of user activity properties in the form of metrics
* Game activity entities:
  + Gamecard: game cards that are displayed through the match that the user can play and win points
  + Question & Answer: Multiple choice questions that when answered correctly by the user can award them points
  + Poll
  + Notification: direct user messaging about the application (coming matches, competitions, etc.)
  + Article: Soccer-related publication/ news article

Match

Event Timeline

Statistics

Competition

Team

Player

User

Moderator

Game activity

Gamecard

Q&A

Poll

Notification

*Image 2: Main Sportimo entities depicted in a hub diagram*

## Network Architecture

The main architecture elements are the application client (web or mobile), the application server connected to a Mongo Database store, and the notification server connected to a Redis NoSql store.

These 3 main elements implement a push/pull information flow, as the following diagram depicts:

Game Server

Client

Notification Server

Publish message in client channel

Listen in subscribed client channel

Push: Send message through Web Socket

Pull: Regular http communication client request/ server response

Redis Pub/Sub

Mongo

*Image 3: Push/ Pull information flow between server and client(s)*

In the regular information flow, the client initiates http requests to the game server, which responds after querying the main application data store, a Mongo server. When information messages are produced that the client should know about, they are sent from the Game Server to the Redis server by using the Redis implementation of the Publish/ Subscribe software pattern, to the appropriate channel (which can be related to a certain match or to a certain match and client application/device pair). Then, a listening notification server will further send the message to the connected clients, if they are matched against the message channel.

The same architecture serves the application’s administration dashboard. This time, the dashboard takes the place of the Client in the diagram.

## Game server internal architecture

In particular, the game server is built around node.js modules, each of which handles a subset of the tasks that the server can undertake. This modular design allows for the facilitation of the code maintenance, separation of concerns and for its scalability under the presence of traffic.

Match moderation

StatsHelper

Match module

rss-feed

Stats parser

Data module

Competition

Sponsor

Player

Team

Offline data

Stats parser

Leaderpay

Leaderboard

Prize

Score

Pool

Questions

Question

FavQuestion

Gamecards

Notification

Users

*Image 4: Node module layering in the Game server*

The following Node modules have been developed:

* Match moderation: handles all matches, provides the API endpoints for their moderation. Includes 3 sub modules each dealing with a specific part of match moderation:
  + StatsHelper: records and analyzes the match statistics
  + Match module: handles timing and event moderation of a single live match
  + Rss-feed: a module that integrates the Stats.com third-party statistics service as a live match event feed
* Data module: handles all peripheral entity management such as articles, competitions, players, schedules, sponsors, team standings, tags
* Gamecards: handles gamecard templates, gamecard definitions and played gamecards (user gamecards)
* Leaderpay: handles leaderboards
* Notifications: manages the direct user message delivery through the Notification server(s)
* Questions: manages the Q&A section of the application
* Users: handles user registration, user management
* Offline data: automates the collection of team standings, team and player update, getting the future competition fixtures through a third-party service (Stats.com)

The match moderation module will instantiate a new match module for each scheduled match in progress or not. Each of the latter may also instantiate a rss-feed module for automatic moderation. The rss-feed in turn instantiates a new Stats.com parser module. Currently only Stats.com parser modules can be placed under an rss-feed, but the module design is flexible enough to adapt to the presence of other parsers that will implement integration with other third party services for live match event coverage. Eventually, the setting of linking a scheduled match to a specific parser implementation (parser module) and pausing/ resuming the parser is handled by the Sportimo administration dashboard, through the match module’s service API.

Match moderation

Match module: A

Rss-feed: C

Parser D

Match module: B

Match module E

Rss-feed F

Parser G

*Image 5:Match moderation module instance hierarchy*

# Implementation

## Server Components

All of the Sportimo application codebase is kept in an online versioning system: Github.com. The Sportimo Game server, the notification server, the dashboard and the HTML template engine and service responsible to render publications and interviews for the platform are all versioned in separate repositories.

### Game Server

The game server is based on top of the Express web applications framework on top of Node.js and is written entirely in Javascript. Several node packages are used:

* Express.js is one of the dominant web application frameworks for Node.js providing a robust set of features for building single and multi-page, and hybrid web applications
* Mongoose as the main Mongo DB access facility, that extends the standard mongodb driver that is offered by node.js with a well featured ORM
* Redis as the main Redis driver that offers full support for the Redis powerful Publish/Subscribe design pattern implementation
* lodash.js for its fast array / collection helper methods (transformations, mappings)
* async.js for parallelizing many async tasks and for its power to avoid the clutter of the so-called callback hell
* moment.js for its power over date/time calculations, transformations and formatting
* Body-parser
* Winston.js as an easy to setup and efficient logger
* Needle.js for wrapping http calls in fluent syntax
* Node-schedule.js for scheduling tasks in time, much like a linux Cron job is configured
* Cors.js for Cross-origin request support
* Express-jwt for Json Web Token authentication support, as a means to safeguard the client-server communication and session information

### Notification Server

The notification server is also written in Javascript for the Node.js platform and over the Express framework for server applications.

## Storage Systems

NoSQL solutions were chosen for the data storage. Write and read operations performance, latency and schema flexibility were decisive factors.

### MongoDB

MongoDB has been evolved into stable and reliable solution for web applications requiring medium to big data volumes. In Sportimo all data are stored in Mongo, following a number of defined schemata which are elaborated in section 4.1. A MongoDb version above 3 is recommended. In a production environment, a setup of a replica set by at least 2 instances each hosted on a different VM is recommended (one configured as the read/write master, the other as a read-only slave).

### Redis

Redis is a winner in cache stores, as one of the fastest memory key-value stores. In Sportimo we chose Redis because of the good implementation of the Publish/ Subscribe pattern, its ease of use and because of the availability of good drivers in a number of languages (Javascript included). A Redis version of 2.8 or higher is recommended. The stricter persistence model (AOF and RDF together) is recommended as well for the production environment.

A potential deployment option for Redis is to deploy on 2 instances on each of which a Redis Sentinel will be setup. The Sentinels will provide a fallback mechanism and guarantee a service availability with a potential downtime of around 1 minute (approximately).

## Client Applications

### The administration Dashboard

This is a web application that is built in Javascript with the Angular web application framework. It communicates with the Game server in order to read and write data from and to the MongoDb data store by executing AJAX queries.

Through this application, a moderator can manage any of the Sportimo-related entities as they have been described in section 2.2.

### Web application

The Web application is developed on top of the React.js javascript framework as a Single Page Application.

A number of React components have been developed in order to manage partial information rendered by the application’s views, their names denote the function that they serve:

* Achievements
* App
* Card
* Cardinfo
* Footer
* Game
* GameFooter
* Header
* Leaderboard
* Login
* MainPage
* Matches
* News
* Register
* ScoreHeader
* Settings
* SettingsMain
* SettingsNotifications
* Standings
* Timeline

### Mobile device (Android / iOS) applications

The mobile applications are developed with the Unity game platform, and finally are built against the 2 mobile OS targets: Android and iOS platforms. They share the same codebase and graphical assets.

# Deployment and Load tests

Sportimo is currently deployed in a cloud service provider (Heroku). It is hosted in virtualized linux containers with a flexible scaling mechanism that delivers CPU cycles to the game server as needed.

## Deployment in a dedicated server cluster

In the case that the services required (game server, notification server, Redis instance, MongoDB instance) have to be deployed in dedicated infrastructure (e.g. datacenter), a recommended setup is proposed hereby. A reverse proxy gateway that will serve as a load balancer (such as HAProxy) will be required, able to handle multiple concurrent requests. For high availability loads, it is desirable to be able to operate in redundancy with more than one instances.

Two zones will have to be setup, a public facing one (or demilitarized) and a militarized one which will be off limits from external visiting users, which will host the back-end services. The following diagram depicts this separation:

Militarized Zone

Sportimo Clients

MongoDB

DeMilitarized Zone

Firewall, Reverse proxy, Balancer

Firewall, Reverse proxy, Balancer

Sportimo game server

Sportimo game server

MongoDB

MongoDB

Figure 1: Sportimo service high availability failover infrastructure

Deployment of the MongoDB, apart from installing MongoDB, consists of setting up some elementary collections, documents and collection indexes.

The following chart summarizes the specifications for a high availability computing infrastructure with redundancy and failover, to cover an unlimited number of leagues and scheduled matches (up to approximately 30 concurrent matches per day and 15,000 players). Each server can either be a physical machine, or a virtual machine inside a bare metal virtualization system (Hypervisor). In the latter case, no 2 nodes of the same server role can reside in the same physical machine. Another budget entry-level specifications sheet follows, for cases with lower target traffic values (up to 10 concurrent matches per day and about 6,000 players).

Each Redis node will include a Redis Sentinel installation and an instance of the Sportimo notification server. Each of these nodes can support approximately 3,000 users, so the vertical scaling design should take this information into account. If the number of these nodes is even, then another Sentinel instance has to be installed in another server node (e.g. the balancer). Redis Sentinels require an odd number of installed instances in order to efficiently apply a majority voting regarding the Redis failover policy. In a similar fashion, MongoDB nodes that are configured to and make up the Sportimo replica set are also an odd number (3).

Physical security devices such as a hot swappable power supply unit, a UPS unit or a backup unit have not been included in the setup as they are rudimentary to any contemporary data center.

Table 1: High availability and failover infrastructure specifications for Sportimo deployment

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Server Role** | **Instance#** | **CPU class  (INTEL equiv.)** | **Memory (GB)** | **Hard Disk (GB)** | **Extras  (for physical servers)** | **Performance critical subsystem** | **Recommended OS** | | | **Installed Software** | |
|  |  |  |  |  |  |  | |  |  | |  |
| **Firewall and Reverse proxy** | |  |  |  |  |  | |  |  | |  |
|  | 1 | i5 | 4 | 500 | Gigabit LAN chipset | Network Driver | |  | CentosOS (Linux) | | Firewall, reverse Proxy (e.g. HAProxy) |
|  | 2 | i5 | 4 | 500 | Gigabit LAN chipset | Network Driver | |  | CentosOS (Linux) | | Firewall, reverse Proxy (e.g. HAProxy) |
| **Sportimo Game Server** | |  |  |  |  |  | |  |  | |  |
|  | 1 | i7 | 16 | 240 |  |  | |  | CentosOS (Linux) | | node.js, Sportimo Game Server |
|  | 2 | i7 | 16 | 240 |  |  | |  | CentosOS (Linux) | | node.js, Sportimo Game Server |
| **Sportimo Notification Server** | |  |  |  |  |  | |  |  | |  |
|  | 1 | i5 | 16 DDR4 | 240 |  | Memory | |  | CentosOS (Linux) | | Redis, node.js, Sportimo Notification Server |
|  | 2 | i5 | 16 DDR4 | 240 |  | Memory | |  | CentosOS (Linux) | | Redis, node.js, Sportimo Notification Server |
|  | 3 | i5 | 16 DDR4 | 240 |  | Memory | |  | CentosOS (Linux) | | Redis, node.js, Sportimo Notification Server |
|  | 4 | i5 | 16 DDR4 | 240 |  | Memory | |  | CentosOS (Linux) | | Redis, node.js, Sportimo Notification Server |
|  | 5 | i5 | 16 DDR4 | 240 |  | Memory | |  | CentosOS (Linux) | | Redis, node.js, Sportimo Notification Server |
|  |  |  |  |  |  |  | |  |  | |  |
| **MongoDB** |  |  |  |  |  |  | |  |  | |  |
|  | 1 (master) | i7 | 32 DDR4 | 500 (2x240) SSD | SATA RAID Controller | HDD, Memory | |  | CentosOS (Linux) | | MongoDB replica set node#1 (master) |
|  | 2 (slave) | i7 | 32 DDR4 | 500 (2x240) SSD | SATA RAID Controller | HDD, Memory | |  | CentosOS (Linux) | | MongoDB replica set node#2 (slave) |
|  | 3 (slave) | i7 | 32 DDR4 | 500 (2x240) SSD | SATA RAID Controller | HDD, Memory | |  | CentosOS (Linux) | | MongoDB replica set node#3 (slave) |

Table 2: Entry level and failover infrastructure specifications for Sportimo deployment

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Server Role** | **Instance#** | **CPU class  (INTEL equiv.)** | **Memory (GB)** | **Hard Disk (GB)** | **Extras (for physical servers)** | **Performance critical subsystem** | | **Recommended OS** | **Installed Software** |
|  |  |  |  |  |  |  |  |  |  |
| **Firewall and Reverse proxy** | |  |  |  |  |  |  |  |  |
|  | 1 | i3 | 4 | 300 | Gigabit LAN chipset | Network Driver |  | CentosOS (Linux) | Firewall, reverse Proxy (e.g. HAProxy), Redis Sentinel |
|  | 2 | i3 | 4 | 300 | Gigabit LAN chipset | Network Driver |  | CentosOS (Linux) | Firewall, reverse Proxy (e.g. HAProxy) |
| **Sportimo Game Server** | |  |  |  |  |  |  |  |  |
|  | 1 | i5 | 8 | 240 |  |  |  | CentosOS (Linux) | node.js, Sportimo Game Server |
|  | 2 | i5 | 8 | 240 |  |  |  | CentosOS (Linux) | node.js, Sportimo Game Server |
| **Sportimo Notification Server** | |  |  |  |  |  |  |  |  |
|  | 1 | i3 | 8 DDR4 | 240 |  | Memory |  | CentosOS (Linux) | Redis, node.js, Sportimo Notification Server, Redis Sentinel |
|  | 2 | i3 | 8 DDR4 | 240 |  | Memory |  | CentosOS (Linux) | Redis, node.js, Sportimo Notification Server, Redis Sentinel |
| **MongoDB** |  |  |  |  |  |  |  |  |  |
|  | 1 (master) | i7 | 16 DDR4 | 240 SSD |  | HDD, Memory |  | CentosOS (Linux) | MongoDB replica set node#1 (master) |
|  | 2 (slave) | i7 | 16 DDR4 | 240 SSD |  | HDD, Memory |  | CentosOS (Linux) | MongoDB replica set node#2 (slave) |
|  | 3 (slave) | i7 | 16 DDR4 | 240 SSD |  | HDD, Memory |  | CentosOS (Linux) | MongoDB replica set node#2 (slave) |

## Stress Tests

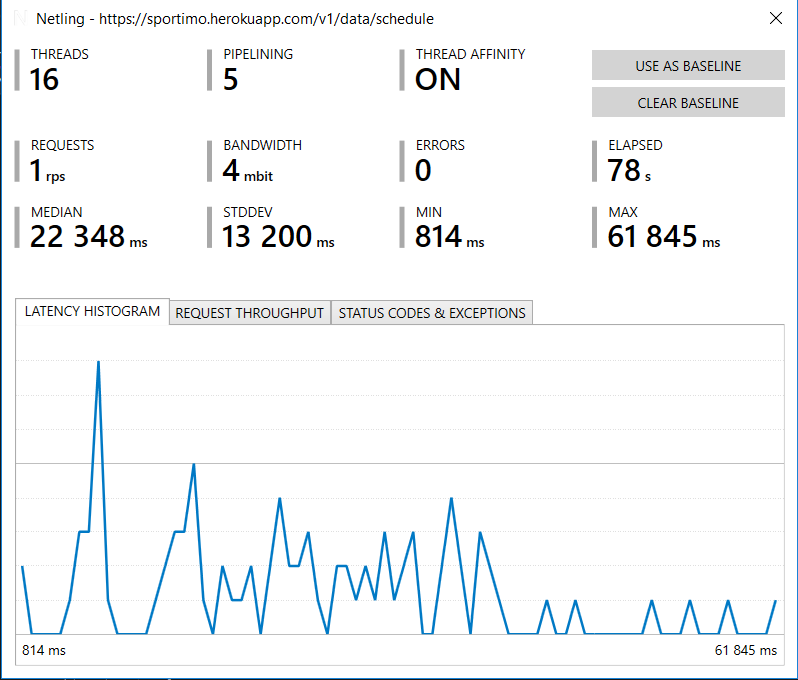
A deployed instance of the game server in a low-end VM system (512 MB RAM, not dedicated -shared CPU, Redis and MongoDB databases remotely connected) was load tested in order to get some insight into the most frequent endpoint performance. The tool used for this purpose is Netling, an open-source dot net core program (https://github.com/hallatore/Netling).

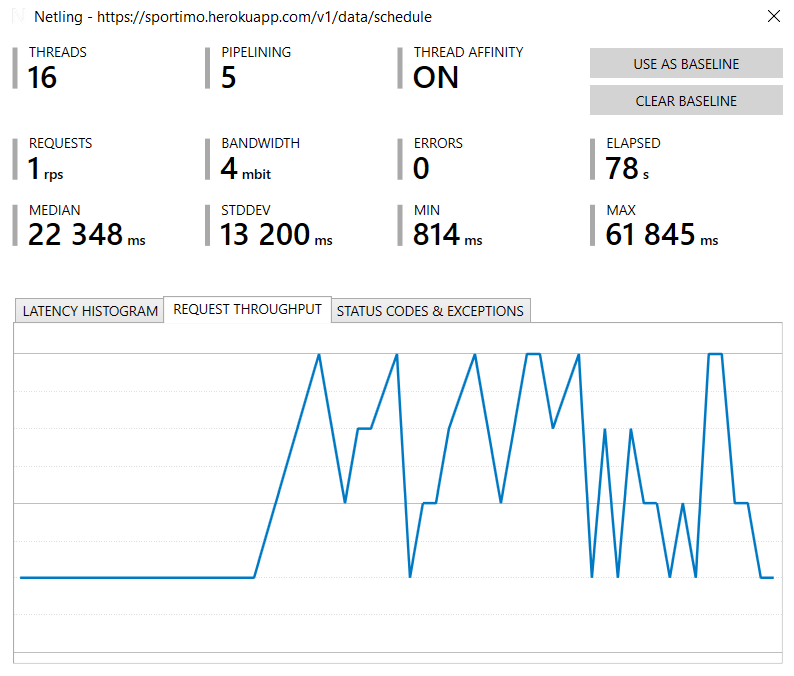
We tested 2 endpoints:

1. <https://sportimo.herokuapp.com/v1/data/schedule>: get all scheduled matches (requested by the main Sportimo app view)
2. <https://sportimo.herokuapp.com/v1/live/match/5a2a59974872641400e0bfe0>: get a specific scheduled match feed events (requested when the player in the Sportimo app enters a match in progress)

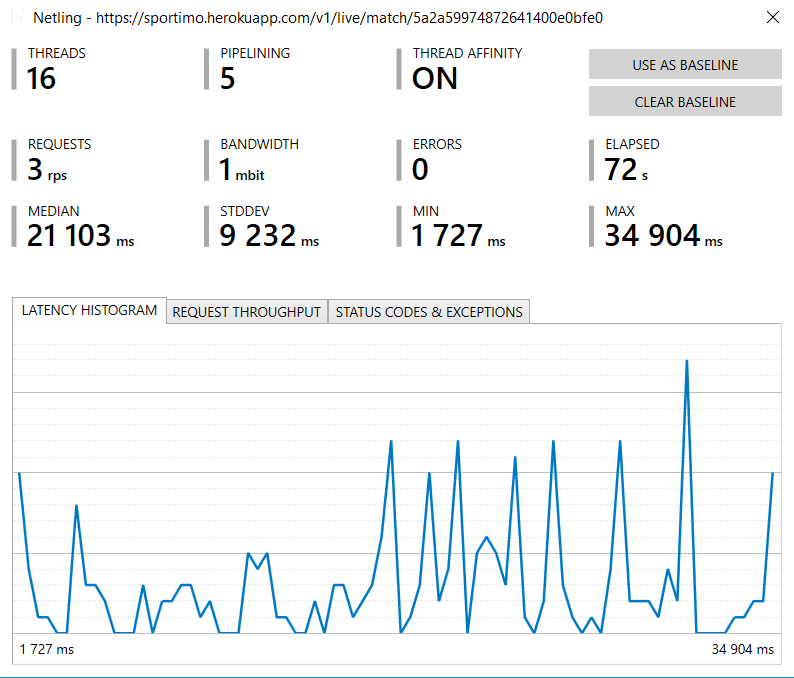
It is worth noting that once inside a match, the client gets all match events through a connected web socket. This latter protocol cannot be easily stress-tested and is left apart from this test. An annex will follow testing the Sportimo notification server’s web socket performance at a later time.

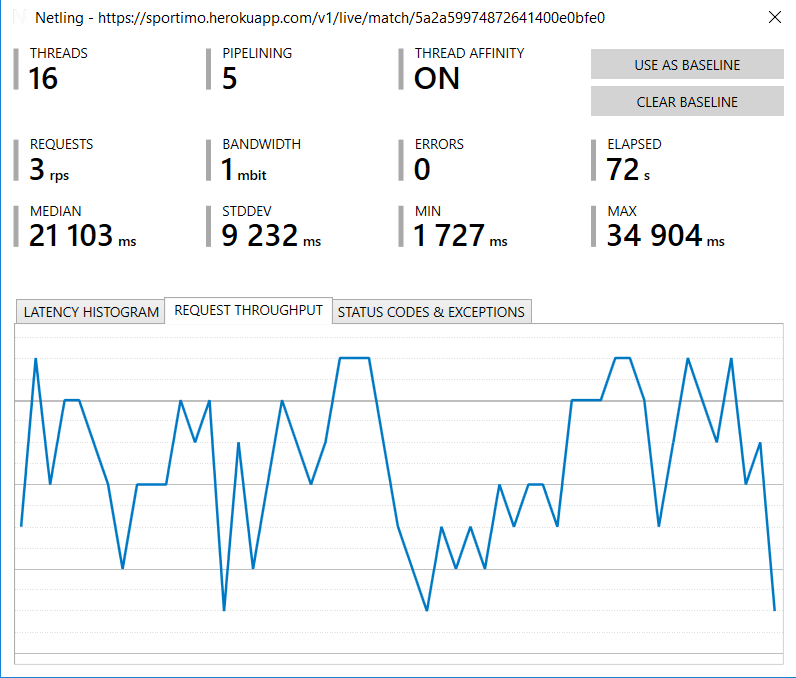
A typical performance test run for endpoint#1, results in the following figures:





A typical performance run for endpoint#2 gave the following results:



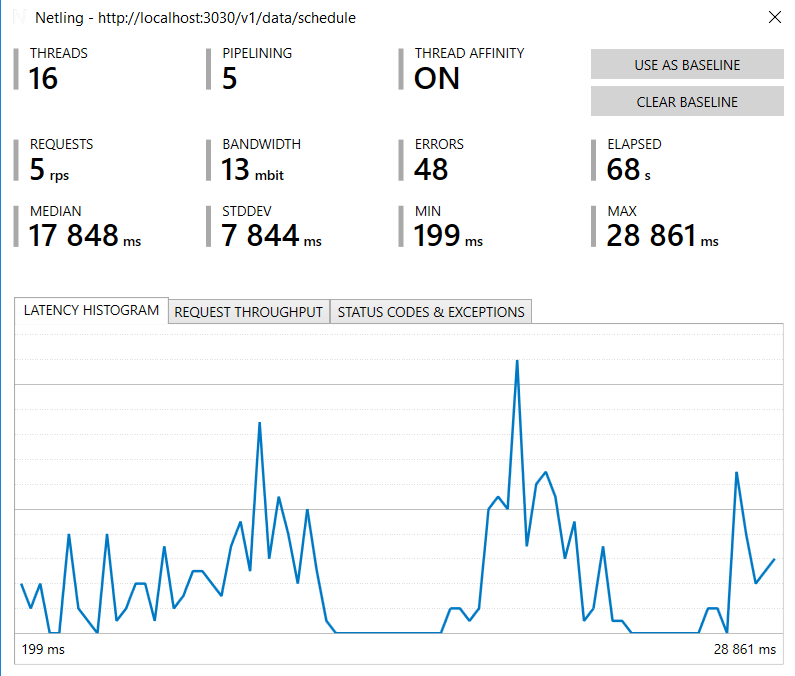


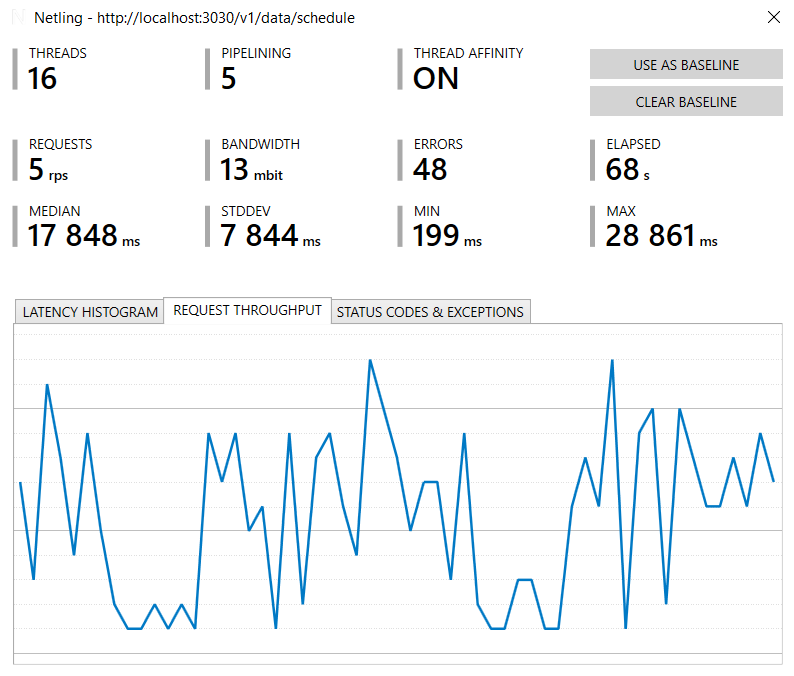
The baseline results show that the endpoint#1 has a better latency histogram. Latency is directly related to the quality of the server connections to the external database servers (in our case to Redis and MongoDB hosting servers).

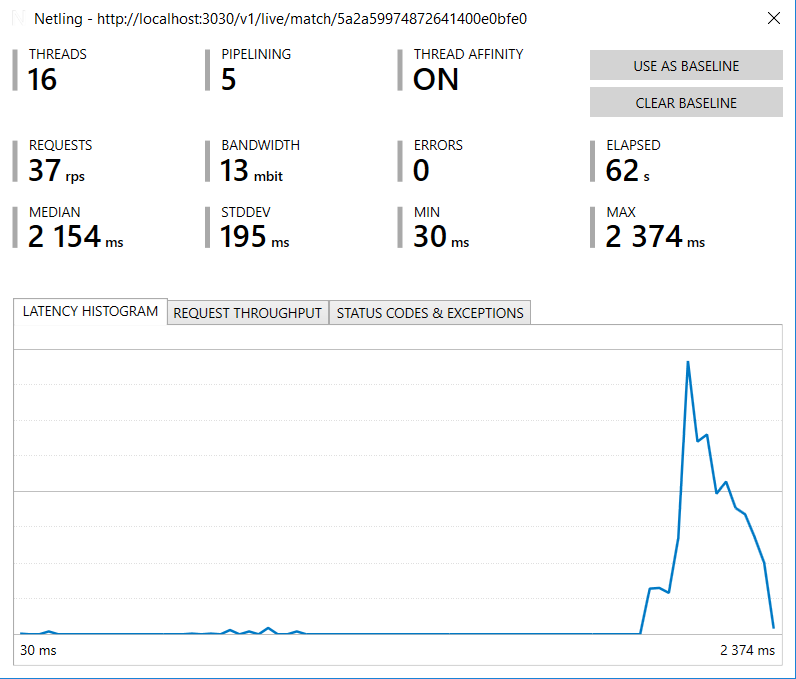
Throughput for the low-end machine tested ranged from 4 to 7 requests per second. Throughput is improved as the VM specs improve, and as the services will be vertically scaled.

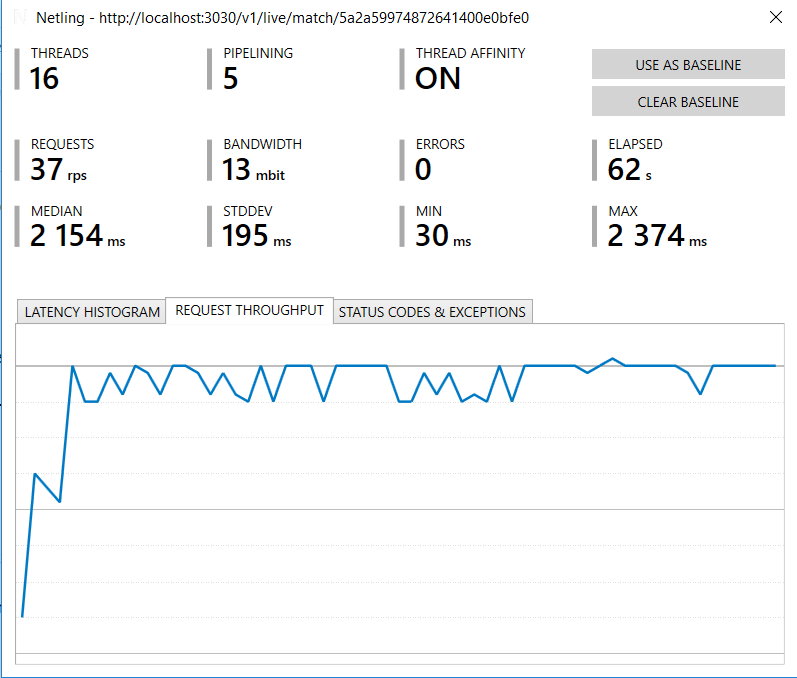
Memory usage throughout the load tests did not pass over 300 MBytes and ranged from 90 to 280 MBytes.

The same tests executed in an Intel Core i7-2760 CPU system with 256GB SSD disk and 8GB RAM furnished the following results:









In the latter tests throughput is improved to between 5 and 37 requests per second, that proves how increasing VM specifications leads to improved performance, as expected.

Considering that a soccer match duration is about 120 minutes, and that users may play up to 15 game cards (15 http requests), with a modest limit of 10 requests per second it is estimated that the number of users per match for a low-end system with 1 game server instance as that of the stress-test can reach a peak of 70000. In the latter system case, this number might double up to 140000.

This number can increase as the Virtual Machine specifications improve and as the game server number scales up. In all cases, there is a limitation on the number of concurrent users or sessions, which is the number of write operations per second in the database store. For MongoDB, this can reach about 40000 operations per second (with a 16-nodes sharded configuration[[1]](#footnote-1)).

# Specifications

## Data Models

A set of schemata have been defined in Mongoose to serve the data access layer of the Game server. Most data types are determined, except in those cases where a certain degree of extensibility is sought, where the property type is set to Mixed.

The schemata are listed below in alphabetical order as a reference. These are not full Mongoose schema definitions but rather partial code snippets, however they contain all the properties and data types, plus the sub-models, where those exist.

### Answer

Not actively used.

*{*

*userid: {*

*type:String,*

*ref:'users'*

*},*

*questionid: String,*

*matchid: String,*

*answerid: String,*

*created: { type: Date, default: Date.now }*

*}*

### Article

Represents a news item regarding a league, a team or a player.

*{*

*publication: { type: Schema.Types.Mixed },*

*publishDate: { type: Date},*

*type: {type: String},*

*photo: { type: String },*

*tags: { type: Schema.Types.Mixed },*

*created: { type: Date, default: Date.now }*

*}*

### Competition

Represents a league or cup (national or international) soccer competition. An instance (document) refers to a specific season (e.g. 2017 – 2018)

*{*

*name: { type: Schema.Types.Mixed },*

*logo: { type: String },*

*parserids: { type: Schema.Types.Mixed },*

*season: String,*

*graphics: { type: Schema.Types.Mixed },*

*visiblein: [String],*

*status: {type:String},*

*created: { type: Date, default: Date.now }*

*}*

### FavQuestion

Not actively used. Reserved for future use.

*{*

*info: String,*

*text: { type: Schema.Types.Mixed },*

*answers: [answer],*

*type: {type: String},*

*img: { type: String },*

*status: {type:Number, default: 0},*

*sponsor: { type: Schema.Types.Mixed },*

*}*

#### Answer

Sub-schema of FavQuestion, holds a possible choice to a multiple choice FavQuestion.

*{*

*text: { type: Schema.Types.Mixed },*

*img: String,*

*points: Number*

*}*

### GamecardDefinition

Represents a definition for a gamecard that the user can play, in a certain match. User gamecards result from inheriting from the documents of this collection.

*{*

*matchid: String,*

*gamecardTemplateId: String, // reference to the gamecard template that this definition represents, optional*

*title: Schema.Types.Mixed, // card title*

*image: Schema.Types.Mixed, // icon image*

*text: Schema.Types.Mixed,*

*primaryStatistic: String, // the primary statistic that this card is affected from, in order to be shown on the card (averages for each team)*

*guruAction: String, // Handling action of guru stats*

*// Trigger specifications*

*activationLatency: Number,*

*specialActivationLatency: { DoublePoints: Number, DoubleTime: Number },*

*duration: Number, // instant gamecards only;*

*appearConditions: [Schema.Types.Mixed],*

*winConditions: [Schema.Types.Mixed],*

*terminationConditions: [Schema.Types.Mixed], // when a played card is terminated and pending resolution before put out of play*

*options: [optionDefinition], // mainly instant gamecards*

*// Specs for awarding points to winning cards*

*pointsPerMinute: Number, // overall gamecards only; the rate by which the startPoints get increased or decreased in time*

*startPoints: Number,*

*endPoints: Number,*

*// States and state times*

*cardType: { type: String, enum: ['Instant', 'Overall', 'PresetInstant'] },*

*maxUserInstances: Number, // maximum number of times a user may play this card*

*creationTime: Date,*

*activationTime: Date,*

*terminationTime: Date,*

*isActive: { type: Boolean, default: true },*

*isVisible: { type: Boolean, default: true }, // overall cards only; true if it can appear on clients' list of gamecard, false if it can't*

*status: 0 // 0: pending activation, 1: active, 2: terminated (dead)*

*}*

#### OptionDefinition

A sub-schema of GamecardDefinition, represents an option of a multiple-options game card.

*{*

*isVisible: { type: Boolean, default: true },*

*optionId: String,*

*text: Schema.Types.Mixed,*

*startPoints: Number,*

*endPoints: Number,*

*pointsPerMinute: Number,*

*activationLatency: Number,*

*duration: Number,*

*winConditions: [Schema.Types.Mixed],*

*terminationConditions: [Schema.Types.Mixed]*

*}*

### GamecardTemplate

Represents a template for all gamecard definitions. When a match is scheduled, a gamecard definition is created out the respective document of this collection, that serves as a prototype.

*{*

*isActive: Boolean,*

*title: Schema.Types.Mixed, // card title*

*image: Schema.Types.Mixed, // icon image*

*text: Schema.Types.Mixed, // text template with placeholders: [[player]] for player name, [[team]] for team name*

*primaryStatistic: String, // the primary statistic that this card is affected from, in order to be shown on the card (averages for each team)*

*guruAction: String, // Handling action of guru stats*

*// Trigger specifications*

*activationLatency: Number, // seconds between the gamecard's creation and activation*

*specialActivationLatency: specialActivationLatencyType, // seconds between the gamecard's special ability creation (double time, double points) and activation*

*duration: Number, // seconds between the wildcard's activation and termination*

*appearConditions: [Schema.Types.Mixed], // the card will appear (start its lifetime in a pending state 0) when all the conditionsToAppear are met.*

*winConditions: [Schema.Types.Mixed], // the wildcard wins when all win conditions are met*

*terminationConditions: [Schema.Types.Mixed], // the card is terminated when any of the terminationConditions is met, or the duration is over (if not null).*

*options: [optionTemplate],*

*isVisible: { type: Boolean, default: true },*

*// Awarded points specs*

*pointsPerMinute: Number,*

*startPoints: Number,*

*endPoints: Number,*

*cardType: { type: String, enum: ['Instant', 'Overall', 'PresetInstant'] },*

*}*

#### OptionTemplate

Represents an option of the gamecard template schema and is a sub-schema of the GamecardTemplate schema.

*{*

*isVisible: Boolean,*

*optionId: String,*

*text: Schema.Types.Mixed,*

*startPoints: Number,*

*endPoints: Number,*

*pointsPerMinute: Number,*

*activationLatency: Number,*

*duration: Number,*

*specialActivationLatency: specialActivationLatencyType,*

*winConditions: [Schema.Types.Mixed],*

*terminationConditions: [Schema.Types.Mixed]*

*}*

### Message

Represents the structure of a message that is sent as an in-app message to one or multiple users.

*{*

*sender:{*

*type:String,*

*ref:'users'*

*},*

*recipients:[{*

*type:String,*

*ref:'users'*

*}],*

*img: { type: String },*

*title: {type:mongoose.Schema.Types.Mixed},*

*msg: {type:mongoose.Schema.Types.Mixed, required:true},*

*data: {type:String},*

*read: {type:Number},*

*created: { type: Date, default: Date.now }*

*}*

### Player

Represents a player of a soccer team. Players participate in the match events in the scheduled\_match timeline, and are also returned as team members in a team view.

*{*

*name: { type: Schema.Types.Mixed },*

*firstName: { type: Schema.Types.Mixed },*

*lastName: { type: Schema.Types.Mixed },*

*uniformNumber : { type: String },*

*stats: { type: Schema.Types.Mixed },*

*pic: { type: String },*

*position: { type: String },*

*personalData: { type: Schema.Types.Mixed },*

*parserids: { type: Schema.Types.Mixed },*

*teamId: {*

*type: String,*

*ref: 'teams'*

*},*

*created: { type: Date, default: Date.now },*

*updated: { type: Date }*

*}*

### Poll

Represents a poll that the app is presenting to users for them to complete. Polls are usually referring to soccer competitions, matches, teams or players.

*{*

*text: { type: Schema.Types.Mixed },*

*answers: [answer],*

*matchid: String,*

*type: {type: String},*

*img: { type: String },*

*total\_votes: {type:Number, default: 0},*

*hasAlreadyVoted: Number,*

*hasAnswered: String,*

*voters: [Schema.Types.Mixed],*

*status: Number,*

*tags: { type: Schema.Types.Mixed },*

*sponsor: { type: Schema.Types.Mixed },*

*created: { type: Date, default: Date.now }*

*}*

#### Answer

Sub-schema of the Poll schema, encodes a possible choice for the poll question to choose from.

*{*

*text: { type: Schema.Types.Mixed },*

*img: String,*

*votes: {type:Number,default:0},*

*percent: {type:Number,default:0}*

*}*

### Question

Similar to the Poll concept, represents a multiple choice type of question that the moderator may ask the users to answer.

*{*

*text: { type: Schema.Types.Mixed },*

*answers: [answer],*

*matchid: String,*

*type: {type: String},*

*img: { type: String },*

*status: Number,*

*correct: ObjectId,*

*sponsor: { type: Schema.Types.Mixed },*

*userAnswer: String,*

*created: { type: Date, default: Date.now }*

*}*

#### Answer

Sub-schema of the Question schema, represents a choice to the respective multiple choice type of question.

*{*

*text: { type: Schema.Types.Mixed },*

*img: String,*

*points: Number,*

*answered: {type:Number,default:0}*

*}*

### Scheduled\_Matches

Represents a central concept of the database overall schema, the soccer match. Its initial state is scheduled in the future, but may also be in a live state, where a moderator or an automatic match feed parser dispatches match events into its timeline or completed. References to the competition where it belongs, and participating teams are contained.

*{*

*sport: {type:String, default:'soccer'},*

*home\_team: {*

*type: String,*

*ref: 'teams'*

*},*

*away\_team: {*

*type: String,*

*ref: 'teams'*

*},*

*start: Date,*

*color: String,*

*competition: {*

*type: String,*

*ref: 'competitions'*

*},*

*name: String, // a match name generated from team names, mainly for tracing and debugging reasons*

*disabled: {type: Boolean, default: true},*

*donttouch: Boolean,*

*visiblein: [String],*

*isTimeCounting: { type: Boolean, default: false },*

*home\_score: {type:Number, default:0},*

*away\_score: {type:Number, default:0},*

*match\_date: Date,*

*time: {type:Number, default:0},*

*state: {type:Number, default:0},*

*completed: {type: Boolean, default: false},*

*stats: [mongoose.Schema.Types.Mixed],*

*guruStats: mongoose.Schema.Types.Mixed,*

*headtohead: {type:Array, default: ["W","W","D","L","L"]},*

*timeline: [segment],*

*settings: mongoose.Schema.Types.Mixed,*

*moderation: [moderationService],*

*guruStatsChecked: {type: Boolean, default: false},*

*updatedAt: Date,*

*createdAt: Date,*

*server\_time:{type:Date}*

*}*

#### ModerationService

Sub-schema of the Scheduled\_Matches schema, represents a service that handles the match feed events moderation. Usually it is linked to a third-party service that the app is integrated with, in order to receive match –related messages.

*{*

*type: String,*

*parserid: String,*

*parsername: String,*

*start: String,*

*active: Boolean,*

*scheduled: Boolean,*

*interval: Number*

*}*

#### Segment

Sub-schema of the Scheduled\_Matches schema, represents a soccer match segment such as first or second half.

*{*

*start: Date,*

*// The time in sport time that this segment starts e.g. 46' for second half*

*sport\_start\_time: Number,*

*end: Date,*

*timed: Boolean,*

*text: mongoose.Schema.Types.Mixed,*

*// time duration that the segment was on hold*

*break\_duration: Number,*

*events: [matchEvent]*

*}*

#### MatchEvent

Sub-schema of the Scheduled\_Matches schema. Represents one match event, as they are gathered inside the timeline field of the parent schema, grouped by each match period.

*{*

*match\_id: String,*

*parserids: mongoose.Schema.Types.Mixed, // one id per sender parser*

*type: String,*

*stats: mongoose.Schema.Types.Mixed,*

*playerscount: Number,*

*status: String,*

*timeline\_event: Boolean,*

*state: Number,*

*sender: String,*

*time: Number,*

*team: String,*

*description: mongoose.Schema.Types.Mixed, // one description per language*

*// extra info property to store general references*

*extrainfo: String,*

*team\_id: String,*

*complete: Boolean,*

*playerSelected: String,*

*players: [mongoose.Schema.Types.Mixed],*

*linked\_mods: [String],*

*created: { type: Date, default: Date.now }*

*}*

### Sponsor

Represents a sponsor of a leaderboard or similar ranking.

*{*

*company: { type: String },*

*name: { type: String },*

*banner: { type: String },*

*video: { type: String },*

*created: { type: Date, default: Date.now }*

*}*

### Standing

Represents a ranking of teams in a given competition, usually based on the included teams’ points.

*{*

*identity: { type: String, required: true, unique: true },*

*season: {type: Number, required: true},*

*competitionid: { type: String, ref: 'competitions' },*

*name: { type: Schema.Types.Mixed, required: true },*

*teams: [{ type: Schema.Types.Mixed }],*

*visiblein: [String],*

*parserids: { type: Array },*

*created: {type:Date, default:Date.now},*

*lastupdate: {type:Date, default:Date.now}*

*}*

### Stats-mod

Represents a certain match statistics modification, and is occurred whenever an event is coming for the match for which the given statistic depends on.

*{*

*match\_id: String,*

*stat\_for: String,*

*stat: String,*

*by: Number,*

*was: Number,*

*is: Number,*

*segment: Number,*

*linked\_event: String,*

*created: Date*

*}*

### Team

A fundamental concept along with the Competition, Player, and Scheduled\_Matches. Represents a soccer team for a specific competition.

*{*

*name: { type: Schema.Types.Mixed },*

*logo: { type: String },*

*color: { type: String },*

*stats: { type: Schema.Types.Mixed },*

*parserids: { type: Schema.Types.Mixed },*

*leagueids: { type: Schema.Types.Mixed },*

*competitionid: { type: String, ref: 'competitions' },*

*recentform: [String], // an array of String of type "W","L","D"*

*nextmatch: Schema.Types.Mixed,*

*lastmatch: Schema.Types.Mixed,*

*standing: {*

*type: Schema.Types.Mixed, default: {*

*"rank": 0,*

*"points": 0,*

*"pointsPerGame": "0",*

*"penaltyPoints": 0,*

*"wins": 0,*

*"losses": 0,*

*"ties": 0,*

*"gamesPlayed": 0,*

*"goalsFor": 0,*

*"goalsAgainst": 0*

*}*

*},*

*topscorer: { type: String, ref: 'players' },*

*topassister: { type: String, ref: 'players' },*

*players: [Schema.Types.Mixed],*

*created: { type: Date, default: Date.now },*

*updated: { type: Date }*

*}*

### User

Another fundamental schema for the app user Entity, encodes all login and profile information about the app user, personal information is anonymized (passwords are hashed).

*{*

*name: {*

*type: String*

*// ,required: true*

*},*

*username: {*

*type: String,*

*unique: true,*

*required: true*

*},*

*password: {*

*type: String,*

*required: true*

*},*

*email: {*

*type: String,*

*required: true,*

*unique: true*

*},*

*picture: String,*

*inbox: [{*

*type: String,*

*ref: 'messages'*

*}],*

*unread: {type: Number, default: 1},*

*social\_id: {*

*type: String,*

*unique: true,*

*required: false*

*},*

*// The following field is going to be used for the single frictionless sign on*

*social\_ids: {*

*type: mongoose.Schema.Types.Mixed,*

*default: {},*

*required: false*

*},*

*pushToken: { type: String, default: "NoPustTokenYet" },*

*pushSettings: {*

*type: mongoose.Schema.Types.Mixed, default: {*

*all: true,*

*new\_message: true,*

*match\_reminder: true,*

*kick\_off: true,*

*goals: true,*

*won\_cards: true,*

*final\_result: true*

*}*

*},*

*resetToken: String,*

*country: { type: String, required: false, default: "GR" },*

*msisdn: String,*

*customerType: {type: String, default:"free"},*

*subscriptionEnd: {type:Date, default: "02/28/2017"},*

*subscriptionContractId: String,*

*subscription: {type: mongoose.Schema.Types.Mixed},*

*pinCode: String,*

*birth: String,*

*gender: String,*

*admin: Boolean,*

*rankingStats: {*

*type: mongoose.Schema.Types.Mixed,*

*default: {*

*bestRank: 9999,*

*bestRankMatch: null,*

*bestScore: 0,*

*bestScoreMatch: null*

*}*

*},*

*stats: mongoose.Schema.Types.Mixed,*

*level: { type: Number, default: 0 },*

*achievements: [achievement],*

*blockedusers: [String],*

*favoriteteams: [String],*

*unlockedmatches: [String],*

*isOnline: { type: Boolean, default: false },*

*deletedAt: { type: Date },*

*deletionReason: { type: String },*

*lastLoginAt: { type: Date }*

*}*

#### Achievement

Represents an app achievement definition that all users are trying to complete. Actual user achievement completion statuses are kept in the User schema.

*{*

*uniqueid: String,*

*icon: String,*

*title: mongoose.Schema.Types.Mixed,*

*text: mongoose.Schema.Types.Mixed,*

*has: Number,*

*total: Number,*

*completed: Boolean*

*}*

#### UserStats

Obsolete. Not actively used.

*{*

*matchesVisited: { type: Number, default: 0 },*

*matchesPlayed: { type: Number, default: 0 },*

*cardsPlayed: { type: Number, default: 0 },*

*cardsWon: { type: Number, default: 0 },*

*prizesWon: { type: Number, default: 0 }*

*}*

### UserActivity

Represents the overall activity of a user in a given soccer match.

*{*

*user: {*

*type: String,*

*ref: 'users'*

*},*

*room: String,*

*matchesPlayed: Number,*

*cardsPlayed: Number,*

*cardsWon: Number,*

*instantCardsPlayed: Number,*

*instantCardsWon: Number,*

*presetinstantCardsPlayed: Number,*

*presetinstantCardsWon: Number,*

*overallCardsPlayed: Number,*

*overallCardsWon: Number,*

*lastActive: Date,*

*isPresent: Boolean*

*}*

### UserGamecard

Represents a game card in-play for a specific match and user. Keeps all properties for state and card management and maintenance. There are 3 types: Instant, Overall, Preset-Instant.

*{*

*userid: String,*

*gamecardDefinitionId: {*

*type: String,*

*ref: 'gamecardDefinitions'*

*},*

*optionId: String, // valid only if the definition includdes options.*

*pointsAwarded: Number,*

*pointsAwardedInitially: Number,*

*matchid: String,*

*title: Schema.Types.Mixed, // card title*

*image: Schema.Types.Mixed, // icon image*

*text: Schema.Types.Mixed,*

*primaryStatistic: String, // the primary statistic that this card is affected from, in order to be shown on the card (averages for each team)*

*guruAction: {type:String , default: "Sum"}, // Handling action of guru stats*

*// Trigger specifications*

*minute: Number,*

*segment: Number,*

*activationLatency: Number,*

*duration: Number,*

*winConditions: [condition],*

*terminationConditions: [condition],*

*pointsPerMinute: Number,*

*startPoints: Number,*

*endPoints: Number,*

*// States and state times*

*cardType: { type: String, enum: ['Instant', 'Overall', 'PresetInstant']},*

*maxUserInstances: Number, // maximum number of times a user may play this card*

*//remainingUserInstances: Number,*

*creationTime: Date,*

*activationTime: Date,*

*pauseTime: Date, // when the gamecard is suspended because the current segment ends*

*resumeTime: Date, // when the gamecard is resumed with the next segment start, after being paused*

*terminationTime: Date,*

*specials: { DoublePoints: special, DoubleTime: special },*

*isDoubleTime: { type: Boolean, default: false },*

*isDoublePoints: { type: Boolean, default: false },*

*wonTime: Date,*

*status: { type: Number, default: 0 }, // 0: pending activation, 1: active, 2: terminated (dead), 3: paused*

*// finally an array of event ids that have modified this userGamecard document since its instantiation, useful for modifying its state when an event is updated/removed*

*contributingEventIds: [String]*

*}*

#### Condition

Sub-schema of the UserGamecard, defines a condition for the final card resolution (whether the user wins or loses the played card).

*{*

*text: Schema.Types.Mixed,*

*stat: String,*

*target: Number,*

*remaining: Number,*

*teamid: String,*

*playerid: String,*

*comparativeTeamid: String,*

*comparativePlayerid: String,*

*comparisonOperator: { type: String, enum: ['gt', 'lt', 'eq']},*

*startPoints: Number,*

*endPoints: Number,*

*pointsPerMinute: Number,*

*conditionNegation: { type: Boolean, default: false }*

*}*

#### Special

Sub-schema of the UserGamecard that represents an extension of a gamecard with special abilities like time extension or playing for double points.

*{*

*creationTime: Date,*

*activationTime: Date,*

*activationLatency: Number,*

*status: { type: Number, default: 0 } // 0: not enabled, 1: pending activation 2: activated*

*}*

#### Star

*Represents a view of the app top users (the stars), a special leaderboard.*

*{*

*users: [userSchema]*

*}*

#### UserSchema

Sub-schema of the Star, for holding top ranking details for one –star- *user*

*{*

*rank: { type: Number },*

*user: { type: String, ref: 'users', required: true },*

*titles: [titleSchema]*

*}*

#### TitleSchema

Sub-schema of UserSchema, represents one top ranking details for a specific user

*{*

*iconUrl: { type: String },*

*date: { type: String },*

*text: { type: Schema.Types.Mixed }*

*}*

#### Purchase

Represents the paid subscription status for a user

*{*

*status: { type: String }, // "Inited" / "Pending" / "Completed"*

*user: { type: String },*

*type: { type: String }, // "Subscription" / "Match"*

*info: { type: String }, // "Weekly Subscription" / MatchID*

*provider: { type: String },*

*method: { type: String },*

*receiptid: { type: String },*

*providerMessage: {type: String},*

*created: {type:Date, default: Date.now()}*

*}*

#### Taunt

*Represents a definition for predefined provocatory messages that a user may send to a friend*

*{*

*type: { type: String },*

*term: { type: String},*

*imgurl: { type: String },*

*sprite: { type: String },*

*text: { type: mongoose.Schema.Types.Mixed },*

*animation: { type: String }*

*}*

#### UserTaunt

Represents instances of Taunt actually sent by users, with a reference to the taunt definition

*{*

*sender: {*

*type: String,*

*ref: 'users'*

*},*

*recipient: {*

*type: String,*

*ref: 'users'*

*},*

*taunt: {*

*type: mongoose.Schema.Types.Mixed*

*},*

*created: { type: Date, default: Date.now }*

*}*

## Game server API Endpoints

A listing of all the Game server API endpoints is provided below as a reference. Please note that with this version of the document some of the endpoint definitions are not finalized and are subject to change.

Where a placeholder such as {match id} is displayed in a Url path, it should be replaced by the actual value of the said parameter (match id in this example).

### Data module

#### Search for an article

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/data/articles/search |
| Used by | Dashboard |
| Parameters |  |
| Sample body | *{*  *“minDate”: “2016-05-21T17:47:59.488Z”,*  *“maxDate”: “2016-05-22T17:47:59.488Z”,*  *“tags”: “Manchester United”,*  *“type”: “News”,*  *“limit”: 5*  *}* |
| Sample response |  |

#### Post a new article

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/data/articles |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve an article by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/data/articles/{article id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update an article by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/data/articles/{article id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete an article by id

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/data/articles/{article id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new competition

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/data/competitions |
| Used by | Dashboard |
| Parameters |  |
| Sample body | *{*  *"name" : {*  *"en" : "Greek Superleague"*  *},*  *"logo" : "./app/img/sportimo/competitions/profile\_superleague\_greece.gif",*  *"status" : "Active",*  *"parserids" : {*  *"Stats" : "gree"*  *},*  *"visiblein" : [*  *"GR"*  *]*  *}* |
| Sample response |  |

#### Retrieve a competition by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/data/competitions/{competition id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update a competition by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/data/competitions/{competition id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete a competition by id

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/data/competitions/{competition id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all competitions

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/data/competitions |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete all competitions

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/data/competitions |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve a match by id for a certain user id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/data/match/{match id}/user/{user id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new team player

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/data/players |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve a player by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/data/players/{player id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update a player by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/data/players/{player id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete a player by id

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/data/players/{player id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all players in a team by team id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/ |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all players

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/data/players |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Cast a user vote in a poll identified by id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/data/polls/{poll id}/vote |
| Used by | Clients |
| Parameters |  |
| Sample body | *{*  *“poll”: “5709854b75df93c800847965”,*  *“answer”: “56e6f0b2500be3ea00b11238”,*  *“userpoll”: “”*  *}* |
| Sample response |  |

#### Retrieve all polls in a match by match id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/data/polls/match/{match id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve a poll by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/data/polls/{poll id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update a poll by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/data/polls/{poll id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete a poll by id

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/data/polls/{poll id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all polls

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/data/polls |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new poll

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/data/polls |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

### Gamecards module

#### Retrieve a game card template by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/gamecards/templates |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new template

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/gamecards/templates |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update an existing template by id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/gamecards/templates |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

The body includes the template id as \_id

#### Update an existing game card definition by id and by match id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/gamecards/{match id}/definitions |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all currently available game card definitions by match id and user id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/ gamecards/{match id}/user/{user id} |
| Used by | Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new played game card for a match id and user id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/gamecards/{match id}/user |
| Used by | Clients |
| Parameters |  |
| Sample body | *{*  *"gamecardDefinitionId":"574083be43eb40f600cd84bb",*  *"matchid":"5740838a43eb40f600cd84b4",*  *"minute":13,*  *"creationTime":"2016-05-21T16:20:16.910Z",*  *"segment":1,*  *"userid":"5736f3c416659bf0002a3f1d",*  *"optionId":1*  *}* |
| Sample response |  |

### Leaderpay module

#### Post a new leaderboard

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/leaderboards |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve a leaderboard by a match id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/ leaderboards |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new pool

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/pools |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update an existing pool by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/pools/{pool id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete an existing pool by id

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/pools/{pool id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve a pool by match id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/pools/forgame/{match id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve a pool by match id and country

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/pools/forgame/{match id}/{2-letter country code} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all timed pools

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/pools/timed |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all timed pools by country

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/pools/timed/{2-letter country code} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new prize

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/prizes |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve an existing prize by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/prizes/{prize id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update an existing prize by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/prizes/{prize id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete an existing prize by id

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/prizes/{prize id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all prizes

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/prizes |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete all prizes

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/prizes |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new score

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/scores |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve an existing score by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/scores/{score id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update an existing score by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/scores/{score id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete an existing score by id

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/scores/{score id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all scores

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/scores |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete all scores

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/scores |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

### Match moderation module

#### Post a new live match

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/live/match |
| Used by | Dashboard |
| Parameters |  |
| Sample body | *{*  *“id”: “5741e7abd7daaba8030b0535”*  *}* |
| Sample response |  |

This endpoint creates an instance of the match-module, which handles every detail of a live match. The id in the body is the match id.

#### Update the segment time of an existing live match by id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/live/match/time |
| Used by | Dashboard |
| Parameters |  |
| Sample body | *{*  *“id”: “5741e7abd7daaba8030b0535”*  *}* |
| Sample response |  |

#### Delete the segment time of an existing live match by id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/live/match/time/remove |
| Used by | Dashboard |
| Parameters |  |
| Sample body | *{*  *“id”: “5741e7abd7daaba8030b0535”*  *}* |
| Sample response |  |

#### Reload an existing live match by id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/live/match/reload |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body | *{*  *“id”: “5741e7abd7daaba8030b0535”*  *}* |
| Sample response |  |

#### Retrieve an existing live match by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/live/match/{match id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new match event command for an existing live match by match id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/moderation/{match id}/event |
| Used by | Dashboard |
| Parameters |  |
| Sample body | *{*  *"type": "Add",*  *"data": {*  *"match\_id": "56a38549e4b067030e9f8111",*  *"type": "Yellow",*  *"stats": {*  *"Yellow": 1*  *},*  *"playerscount": 1,*  *"status": "active",*  *"timeline\_event": true,*  *"state": 0,*  *"sender": "Moderator",*  *"time": "54",*  *"team": "away\_team",*  *"players": [{*  *"id": "56ebd1add299e8ed04e93df5",*  *"team": "home\_team",*  *"name": "Sergio Aguero",*    *}]*  *}*  *}* |
| Sample response |  |

The type property in the body dictates the command to execute on the data part of the body object: Add, Update, Delete, AdvanceSegment or SocketMessage.

#### Retrieve all scheduled matches

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/schedule |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update an existing scheduled match by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/schedule/{match id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve an existing scheduled match by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/schedule/{match id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new scheduled match

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/schedule/ |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete an existing scheduled match by id

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/schedule/{match id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all active moderation services by match id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/moderation/{match id}/service |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Try to start a new moderation service for a match id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/moderation/{match id}/service/add |
| Used by | Dashboard |
| Parameters |  |
| Sample body | *{*  *"type": "rss-feed",*  *"parserid": "1637954",*  *"parsername": "Stats",*  *"active": true,*  *"interval": 5*  *}* |
| Sample response |  |

#### Pause an existing moderation service by match id and service type

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/moderation/{match id}/service/pause |
| Used by | Dashboard |
| Parameters |  |
| Sample body | *{*  *"type": "rss-feed",*  *"parserid": "1637954",*  *"parsername": "Stats",*  *"active": true,*  *"interval": 5*  *}* |
| Sample response |  |

#### Resume an existing moderation service by match id and service type

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/moderation/{match id}/service/resume |
| Used by | Dashboard |
| Parameters |  |
| Sample body | *{*  *"type": "rss-feed",*  *"parserid": "1637954",*  *"parsername": "Stats",*  *"active": true,*  *"interval": 5*  *}* |
| Sample response |  |

### Notifications module

#### Update notifications user data by user id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/notifications/users/ |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all existing notification user data

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/notifications/users/ |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Push (send) a notification to a user id and match id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/notifications/push |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

### Offline data module

#### Update all teams by competition id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/offline\_data/teams |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update all player statistics by team id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/offline\_data/players/{team id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update team standings by competition id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/offline\_data/standings/{competition id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update all team standings for all competitions

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/offline\_data/standings |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Return all future match fixtures by competition id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/offline\_data/fixtures/{competition id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

### Questions module

#### Post a new favorite question

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/questions/favorites |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve an existing favorite question by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/questions/favorites/{question id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update an existing favorite question by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/questions/favorites/{question id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete an existing favorite question by id

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/questions/favorites/{question id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all favorite questions

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/questions/favorites |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new question

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/questions |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve an existing question by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/questions/{question id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update an existing question by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/questions/{question id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Delete an existing question by id

|  |  |
| --- | --- |
| Method | DELETE |
| Url | …/v1/questions/{question id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all questions by match id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/questions/match/{match id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post an answer by question id and user id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/questions/{question id}/user |
| Used by | Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post an answer by question id and moderator id

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/questions/{question id}/moderator |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

### Users module

#### Post a new user

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/users |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Authenticate a user

|  |  |
| --- | --- |
| Method |  |
| Url | …/v1/users/authenticate |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve all users

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/users |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve an existing user by id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/users/{user id} |
| Used by | Dashboard, Clients |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Update an existing user by id

|  |  |
| --- | --- |
| Method | PUT |
| Url | …/v1/users/{user id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Post a new message to users

|  |  |
| --- | --- |
| Method | POST |
| Url | …/v1/users/messages |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Get an existing message by user id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/users/{user id}/messages |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

#### Retrieve user activity by match id

|  |  |
| --- | --- |
| Method | GET |
| Url | …/v1/users/activity/{match id} |
| Used by | Dashboard |
| Parameters |  |
| Sample body |  |
| Sample response |  |

# Database

## Collections

A grouping of **MongoDB** documents. A **collection** is the equivalent of an RDBMS table. A **collection** exists within a single **database**. **Collections** do not enforce a schema.

The Sportimo database includes the following collections and each one will be explained in section 6.2:

* Achievements
* Answers
* Articles
* Competitions
* Earlyaccessusers
* Favquestions
* Gamecarddefinitions
* Gamecardtemplates
* Gameserversettings
* Introquestions
* Messages
* Players
* Polls
* Pools
* Prizes
* Questions
* Scheduled\_matches
* Scores
* Settings
* Standings
* Subscriptions
* Taunts
* Teans
* Useracctivities
* Usergamecards
* Users
* Usertaunts

## Collection detailed

|  |
| --- |
| Achievements |

The list of the game achievements.

|  |
| --- |
| Answers [Not Used Yet] |

Placeholder for future use

|  |
| --- |
| Articles |

The list of news articles to view in the client news section

|  |
| --- |
| Competitions |

All leagues viewable in the game.

|  |
| --- |
| Earlyaccessusers |

All leagues viewable in the game.

|  |  |
| --- | --- |
| email: | The email of the user |
| email\_sent: | Have we sent invite |
| code: | The early access code |

|  |
| --- |
| Gamecarddefinitions |

The game cards that are available for each match. Created separate in order for the moderator to make custom tailored changes for each match.

|  |  |
| --- | --- |
| matchid: | The id of the match it relates to. |
| gamecardTemplateId: | The id of the template that propagates from |
| creationTime: | The utc time the card was played |
| primaryStatistic: | The primary statistic that this card references. |
| activationTime: | The utc time the card was activated. |
| terminationTime: | The utc time the card was terminated. |
| cardType: | The type of the card. |
| Options: | List of predefined options, regarding teams and win conditions. |

|  |
| --- |
| Gamecardtemplates |

Game card templates are the initial setup for the game cards that will later on be cloned for use in each match. They have the basic setup for all game cards that will be in play. For property details see: Gamecarddefinitions.

|  |
| --- |
| Introquestions |

List of skill questions asked before match in order for the users to be able to win prizes.

|  |
| --- |
| Messages |

The list of messages the dashboard moderators have sent to the users.

|  |
| --- |
| Players |

This is all the data stored from stats regarding player info. Warning these are not the users of the game. These are the football players and their stats.

|  |
| --- |
| Polls |

The polls create from the dashboard.

|  |  |
| --- | --- |
| tags: | To what section this poll is attached |
| text: | The question of the poll |
| total\_votes: | The total votes on the poll |
| Cerated: | The utc time that the poll was created |
| answers: | The list of poll answers |
| answers.text: | The answer’s text |
| answers.img: | The answer’s image |
| answers.percent: | The percent that has this answer |
| answers.votes: | The votes received by this answer |

|  |
| --- |
| Pools |

Internal data used to create leaderboards.

|  |
| --- |
| Prizes |

List of prizes used for assigning them in leaderboards

|  |
| --- |
| Questions [Not Used Yet] |

Placeholder for future use

|  |
| --- |
| Scheduled\_matches |

This is the list of the scheduled matched data. All matches scheduled in the dashboard get an entry here and consecutively are handled by the service or a moderator directly.

|  |  |
| --- | --- |
| home\_team: | The id of the home team |
| away\_team: | The id of the away team |
| start: | The utc time that the match starts |
| competition: | The Id of the competition that this match belongs to |
| settings: | Settings for the client. |
| moderation: | Sets various settings regarding moderation and if it will be based on a feed or manual handling. |
| timeline: | The list of timeline events |
| headtohead: | The results of previous classes between these two teams |
| stats: | Stats created by the timeline events. These are the stats checked in order to validate user wins. |
| Completed: | Whether the match has ended or not |
| State: | The segment the match currently is |
| Time: | The time of the match. This is the game clock not actual time. |
| Away\_score: | The score for the away team |
| Home\_score: | The score for the home team |
| Visiblein: | Countries this match is visible |
| guruStats | The guru stats as gathered by the service |

|  |
| --- |
| Scores |

The list of score that each user has for each match that has participated in.

|  |
| --- |
| Settings |

The settings object that initializes the clients.

|  |
| --- |
| Standings |

The standings for each league as received by the stats.com service

|  |
| --- |
| Subscriptions [Not Used Yet] |

Placeholder for future use

|  |
| --- |
| Taunts |

The list of the taunts available for users to use as send messages in game.

|  |
| --- |
| Settings |

The settings object that initializes the clients.

|  |
| --- |
| Teams |

The team data as received by the stats.com service.

|  |
| --- |
| Useractivities |

This collection tracks users’ movements. An activity is recorder each time the user visits a match.

|  |  |
| --- | --- |
| room: | The match id |
| user: | The user id |
| createdAt: | The utc time of the first visit |
| lastActive: | The last utc time that the user was active |
| overallCardsPlayed: | How many overall cards the user has played |
| cardsPlayed: | How many total cards the user has played |
| instantCardsPlayed: | How many instant cards the user has played |

|  |
| --- |
| Usergamecards |

The list of user played cards.

|  |
| --- |
| Users |

The list of all the game’s users’ data.

|  |  |
| --- | --- |
| lastActive: | The utc time that the user was last |
| createdAt: | The utc time that this user was created |
| admin: | If the user is an administrator/moderator |
| username: | The username of the user |
| password: | The hashed password of the user |
| isOnline: | If the user is presently online |
| favoriteteams: | The list the user’s favorite teams |
| blockedusers: | The list of users this user has blocked |
| achievements: | The list of this user’s achievements |
| level: | The level of the user. This is calculated based on achievements |
| rankingStats: | The best ranking stats for the user |
| subscriptionEnd: | The utc time this user’s subscription will end |
| customerType: | The type of user. Paid or free |
| country: | The country of this user |
| pushSettings: | List of various push permissions |
| pushToken | The push token of the user |
| Unread: | Number of unread messaged the user has |
| Inbox: | List of ids of messages directed to this user |
| Stats.matchesvisited: | User Stats: How many matches the user has visited |

|  |
| --- |
| Usertaunts |

The list of all the user sent taunts and messages.

## Database Footprint and Size projection

As of 25/5/2018, the database size is 869572992 Bytes, containing 111929 documents with an average document size of 7768 Bytes. 146 matches have been played from the start of 2018, which contribute to half a season for about 7 national leagues.

The 1-day footprint corresponding to 1 match played in the app is approximately 6 Mbytes. In order to acquire a size required to run x matches in a season for various competitions, this has to be multiplied by x.

1. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.696.4625&rep=rep1&type=pdf> [↑](#footnote-ref-1)