Mobile Competition Management with Live Acquisition, Reporting, and Notification

Staffordshire University
Computing Science
Patrick Bowen
May 2020

Table of Contents

Abstract

This project was undertaken to address inefficiencies in physical and vocal information sharing within live sport competitions, particularly in unpopular sports without bespoke digital solutions. Using a message-based architecture to provide live mobile data propagation—such as competition scores and directions—has proved to be possible and practicable. This report investigates the use of MQTT-like messaging between three components – mobile web front-ends, a ‘general’ model, and ‘bespoke’ models. This was successfully implemented, separating the concerns of concrete competition business logic, user authentication and authorisation, state propagation and persistence, and with standard input-output channels. Demonstrated as a suitable digital solution, future work may look to use the same architecture in any similar setting outside of sport, and move all components to serverless web hosting.

Introduction

Small sports competitions may suffer from a deficiency in the availability of technology for managing participants, scores, and communication on format and live play. This will be especially true of niche sports, due to lack of technical qualification or funding for development of technical solutions. Such deficiency may limit participant, management, and audience interactivity, relying on ad hoc digital, vocal, or paper communiqué. Furthermore, technical solutions for one or few sports may have been commissioned within a narrow scope, limited from other sports and disciplines, as noted by *Kapela et al*, 2015.

To help develop a more accessible, inclusive solution to the aforementioned deficiency, using a generalised approach will be beneficial, as opposed to the intuition of specialised existing programs (Bentley, 1982). This view is shared by the Inventor’s Paradox, whereby “The more ambitious plan may have more chances of success ... provided it is not based on a mere pretension but on some vision of the things beyond those immediately present.” (*Pólya*, 1945)

To implement a solution across sports, maximising accessibility, numerous discrete choices should be made: the platforms that this technical solution will be hosted on and used; how pervasive the solution is into the management of a competition, through human and machine interfaces; how open to customisation the solution is (e.g. custom administration roles), despite its generalisation for each sport; how far the solution should be able to operate alongside existing machines, programs, and manual practice within competitions; how accessible the deployment of solution should be to lay competitors.

This report will investigate the potential of a mobile web application interface with its users, providing easy access to the system while in a competition setting; investigate a composable architecture with strong separation of business logic, especially across different sports; discover a way to create a generic artefact which can support multiple sports, at the same time.

To reduce development time, improve the quality of the resulting designs, and significantly decrease the perceived complexity of the usability features, popular functional usability guidelines for software development should be utilised (*Carvajal et al*, 2013). The guidelines call for features such as providing the user warnings before enacting large or destructive operations, well defined low-level component responsibilities, allowing deletion of any user-supplied system task, providing undo and redo features, and more. The MVC (Model-View-Controller) software pattern is suggested, and while more applicable to traditional desktop applications there are analogues.

Evaluating existing solutions

Before this design of a universal competition management software, existing solutions and approaches will be investigated. Both grey and academic literature will be considered. First, a qualification on what the proposed problem requires: a generic system of score *communication*, rather than any novel digital approach to gathering data from live play. Some sports have great availability of software capable of automatically detecting events within play, such as live feedback in field sports described by *Kapela et al*, 2015. They note a significant interest in “algorithms for automatic event detection in sports broadcasting.” They also note, similar to this report, that existing solutions focus on individual sports, not providing a generic solution which can be applied to any sport. Their concerns also highlight the low latency in which events can be extracted and interpreted is a crucial benefit, rather than retrospective analysis. Thus, the interest of this report, and solution, is not in this widely-researched area, but rather the communication of those scores to attendees and audience of small, local sports. These sports are less likely to benefit from televised broadcasting or have little to no interest of live notification outside of their attendees, as found within club or regional competitions.

Character of competitions, competition management

An inherent system monologue. Since early considerations of such systems, there is the prevalence of a monologue between competition organisers and those eager to follow the competition. It is found that almost two decades ago, through *Rafey et al*, 2001, that the rise of Digital TV for consuming sport was considered, and in particular this concerned interactivity with audience

members. However, this activity was consolidated purely ‘downstream’, in sports consumers’ homes, with a monologue of live scoring still undisturbed. In modern considerations, such as by *Hutchins & Rowe*, 2012, television is recognised as an ageing medium, with a rise in audience participation through the World Wide Web. However, as with Digital TV, the monologue still exists instead through internet streaming. Therefore, in this competition solution, it is not expected for audience members to have dialogue with competition management.

The dialogue within competition. The monologue between competition management and audience is wholly separated from the internal mechanics of managing a sport competition. An example sport, which will be used throughout this report, is fencing. In a fencing competition, it is a crucial recommendation for organisers to have planned ahead the time slots, locations, and early on the participants, with which play will take place. This timetable is followed by administration throughout the competition, and is usually a personal, per-organiser approach – for example, Microsoft Excel may be used to plan. This is evidence toward a software solution only providing simple interface, exclusive from the bespoke planning per sport and organiser. The following article quotation—addressed to potential competition managers—highlights the priorities of a fencing organisation manager, Jon Willis, 2016:

> When results come in, get the data entered quickly into the computer and then post results on the notice board for fencers to check as quickly as possible. Continually fielding questions from fencers and parents asking ‘how long until the next round?’ or ‘how many rounds of poules[1]?’ will just slow you down. By taking time to answer you are not entering data into the computer! To help avoid this, put up information about the competition format on the Notice Board and try to have a junior assistant to field queries whilst you get on with the vital task of data input.

The *dialogue* here is between the organiser and junior assistants, with a *monologue* toward participants (and their parents) on how the competition will progress. It is highlighted that ineffectual, but more importantly inaccessible, notification to participants and audience can slow down the process of continual organisation and direction.

Other operational problems are highlighted in this article, such as recommendations to “[reduce] the number of shuttle runs with score sheets and helps the young fencers stay in the right place”. This is precisely the problem a digital communication platform needs to tackle – physically shuttling live data throughout a competition. Further, in contemplating future improvements within the format, “The major efficiency I would like to see introduced is a tablet system of poule sheets… Increasingly, competitions have websites with FAQ, information and even live feeds/results.” These

1 A “poule sheet” in fencing is the prescribed bout-order for multiple fencers, ensuring all play against one another; traditionally filled in with a pen as a grid with scores given by referees.

are calls for improving the *dialogue* between competition officers and participants, and also the *monologue* of the officiated scores and to the audience and participants.

Consolidating monologue and dialogue. It is possible to consolidate these two tiers of communication, yet at once keeping them separate. A digital platform which can broadcast directions of participants and referees from competition management. With the correct intimate live data views, such as how long until a participant is next to play, and where, it is possible to be an alleviation to common operational problems. Delivery through a smartphone web application, with a more operable access for organisers such as through tablet or laptop computer, would be very suitable.

Data authority. The production and consumption of inputs and outputs of the proposed solution must be scrutinised for potential authorisation issues. Inputs into the software will certainly include: granting user roles, participant inclusion and ejection, scores from referees during play, participant matching and direction during a competition, overall results; inputs may also include: extra data play relating to performance such as disciplinary decisions. These inputs will need to be from verified users of the software solution, such as from referees or competition organisers – an appropriate authentication method will need to be applied, such as a user role system with strong credential proof. However, consideration by *Roberts*, 2006; *Vijayan*, 2010; *Li*, 2013; among many other trade commentators and cryptographic researchers, conclude password authentication to be vulnerable to many accessible attacks and improper uses. Therefore, paired with requirement for password authentication, to mitigate malicious actions within the software all user actions should be securely logged and available for study, to verify the actions taken by users were genuine.

Data transparency and involvement. All non-security inputs to the system can be completely transparent to all users of the software, as there is no confidential information. The data broadcast on the software, such as the scores and directions, are non-sensitive information as they are normally playing in public view, and directions are normally given on public notice. This answers the potential authorisation issues of outputs from the system. However, not all information will be relevant to all participants of the software – data should be available to all, but data such as directions should be more readily accessible to referees and playing participants in a competition. Data such as scores, which are usually broadcast to the audience, should be readily available to the audience roles. It is perceivable, however, that any party could be concerned – personally involved – with any outputs of the software. Users may also wish to opt-out of certain data that would be relevant to them under normal circumstances. Therefore, active choice of the data consumed by individual users should be granted to them, with defaults provided for each role a user adopts.

Data processing. Aside from security and operational-level web-app data, data inputted into the software will mostly remain unaltered – scores, directions, and other data as part of organisational monologue will be preserved. User views on data, such as the aggregate scores of particular participants, will likely be calculated client-side, and not cached, but still be an appreciated feature of the web-app. Data processing will likely occur on automated direction as part of a bespoke model, encapsulating the behaviour of a sport or discipline within a sport. For example, a critical disciplinary action against a player could automatically eject a participant from further play, and could be accounted for, and recorded, by the bespoke model. The correct brokering of events throughout the system, to parties which have subscribed to various categories such as participant direction, score updates, and total score, would need to be handled by a general actor within the system. This will ensure the bespoke model does not need to implement brokering capabilities, or even have knowledge of whole categories of users such as audience members.

Convenience of use. Security aspects that engage with users are an inconvenience to the use of a system, in which users find it time-consuming, and generally an obstacle to its operation (*Chipperfield & Furnell*, 2010). Reducing friction in security aspects – by only incorporating basic security features such as using an email and password – is an important consideration. Other inconveniences may arise through not strictly following the latest recommendations for software accessibility, such as following industry standards for standard operations, e.g. creating, reading, editing, and deleting records, or navigating a web-app. Care should be taken in primary research for how a user of existing competition management software, or manual competition management, would find benefit in the system, with features such as data import and export, and flexibility in announcements and presentation. This care should also be taken for the potential participants and audience members of this system, and how they would expect to interact with the web-app.

Existing software

Existing competition management software. There are numerous, though expectedly particular and proprietary, software solutions for the recording, event capture, management, processing, analysis, and digital display of live, retrospective, predictive, and organisational data of sport competitions. Some software is listed below in *Table 1*.

Table 1 – Existing sport competition management software

#	Product name	Features	Included sports
1	DakStats Statistical Software	Record and display scores and statistics	Base- soft- basket- foot- volley- ball, hockey, soccer, &c.
	OmniSport Aquatics Software	Timing and scoring	Swimming and water sports
	Tennis Software	Control multicourt tennis scoreboards; control video displays	Tennis
	Motorsports Interface	Automatic collection of race data such as timing, controlling video displays and numeric scoreboards	Motorsports
	Out of Town Scores	Gather scores and messages from nation-wide games, display locally	Misc.
	Daktronics Data Manager	Display data from Microsoft Excel spreadsheets	Non-traditional sports, tournaments, special events
2	*Content Management*	Management of multiple digital assets within arenas	Field sports
	Live Production	Video live-switching, replays, slow motion, match highlights	
	Statistics	Present scoring, penalties, statistic calculation	
	Video Analysis	For referees – reviewing recorded video	
3	Fencing Time	Store and retrieve tournament data, participant check-in, membership management, import rankings, referee tracking, automated poules, classification/qualification computation, generate results	Fencing
	Fencing Time Live	Display live results of active tournaments	
4	BellePoule	Electronic score sheets, calculate individual results	
	SmartPoule	Mobile submission of scores, timekeeper, video display control	

	LivePoule	Live video display of scores, participant details, bout details, timer, score events	
5	OpenFencing	Cross-platform tournament management, poules with direct elimination, uses MySQL database	
6	Engarde	Competition, formula, poule, tableaux, ranking, referee assignment, automation and management; display driving, internet results, smartphone app	
7	Fencing Fox	Competition and remote screen management, real-time score publishing, popular scoring machine connection	

Web addresses

1. https://www.daktronics.com/en-us/products/software-and-controllers/sport-software
2. https://www.goalsport.software/software
3. https://www.fencingtime.com
4. http://betton.escrime.free.fr/fencing-tournament-software
5. https://sourceforge.net/projects/openfencing/
6. https://engarde-escrime.com
7. http://fencingfox.com/index.html

Screenshots

Unfortunately some products were very impractical to obtain high-resolution screenshots of.

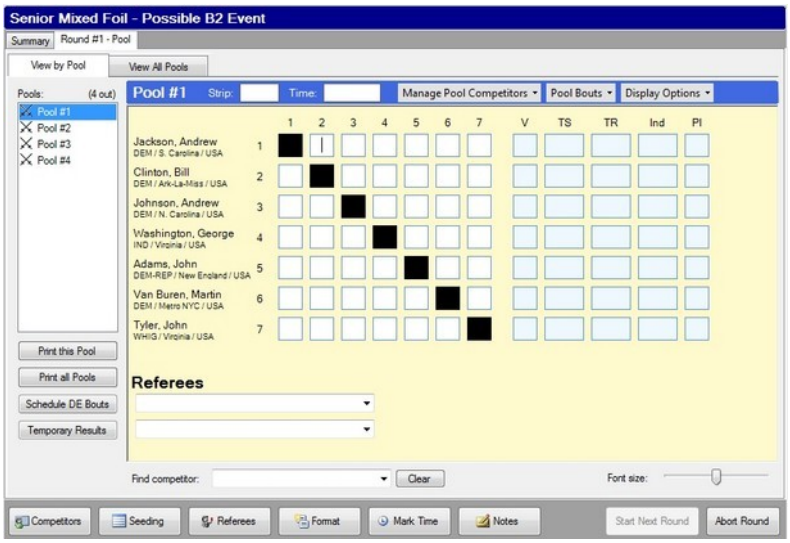


Illustration 1: Fencing Time (3) - showing a virtual poule sheet

Illustration 2: Fencing SmartPoule (4) – mobile app

Number	Name	Nation	
20	BARRY Mirabelle		☆
15	BAUGHAN Alexandra		☆
5	BLACKLEDGE Eleanor		☆
	CALLAGHAN Haleena		☆
20	CHILKOTI Siddhangana		☆
2	CRAZE Lexie		☆
15	DIAMANSKI Alexandra		☆
15	DONNELLY-SALLOWS Kei.		☆
15	GEENS Amelie		☆
	GEURDEN Sasha		☆
20	HISLOP Sophie		☆
3	JANE Melissa		☆

Illustration 3: Engarde Smart (6) - showing participant listings

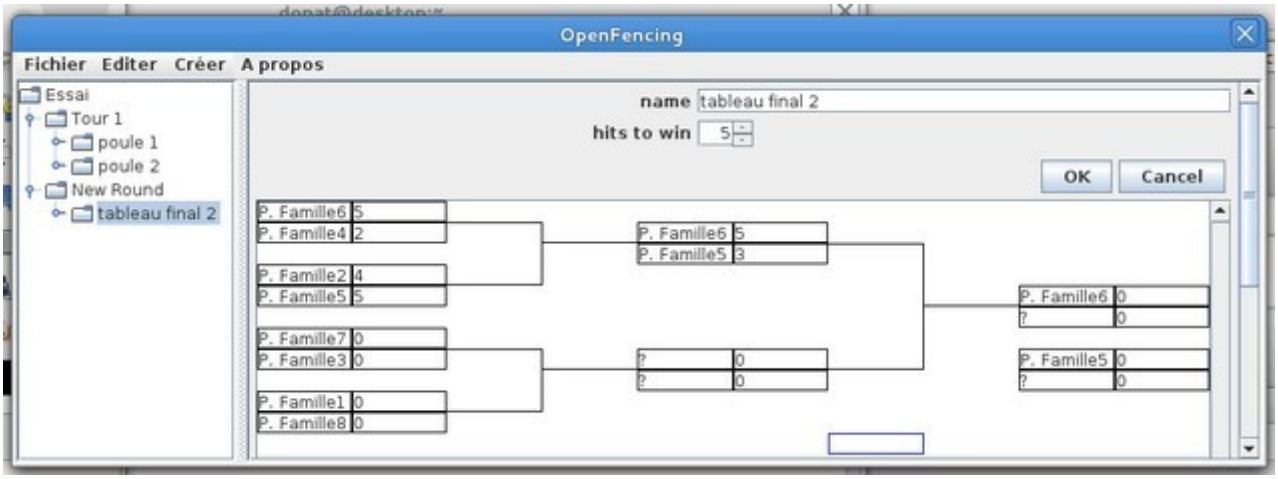


Illustration 4: OpenFencing (5) - showing direct elimination diagram

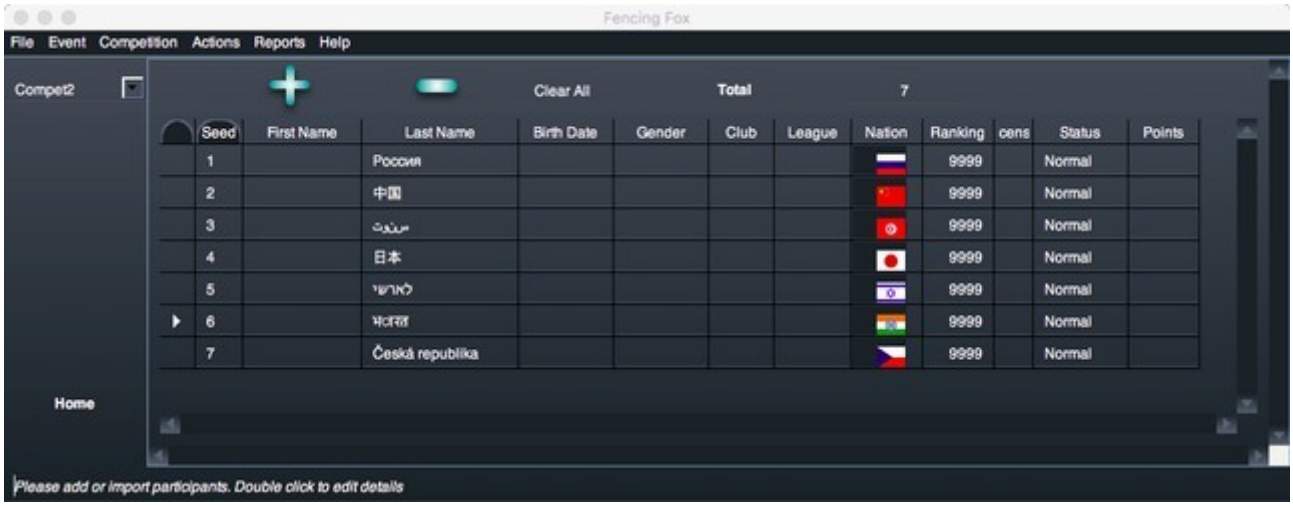


Illustration 5: Fencing Fox (7) - showing participants view

Illustration 6: Goal Sport (2) - live video content manager

Overview of these products

Diversity of sports. Aside from a few professional, modern technology businesses (1, 2) which have been able to capture a large market, most software is tailored toward one sport in particular. Especially with open-sourced software, bespoke software covering just the core aspects of a sport, such as fencing, is more prevalent. High-quality software tends to sell its own hardware (1, 2, 3), which may be the main source of revenue for some businesses. All genres of sport appear to be included, except e-sports, which presumably comes with its own score propagation. One software

suite, *Daktronics Data Manager*, even includes support for non-traditional sport, which alludes to a flexible system.

Diversity of feature scope. All software was time-aware, with real-time scoring and competition direction, for managing multiple participants. All software found had some way of persisting scores, such as with a database or online service. All software also included manual control over their features, to override any automation, or provide just manual features. Surprisingly, all software included the ability to output scores off from the user's computer or webspace, either through physical means such as printing reports of play, or through digital signage. All software had automation of competition management. Some software (3, 4, 6) had participant management such as registration. However, unlike the solution within this project, only one software suite, *Engarde*, had a monologue to exclusively competition participants, though other software did include active dialogue with referees.

Dialogue capabilities. The largest suites of software (1, 2) included very particular, high-capability technology such as video analysis and automated digital signage, but lacked participant or audience management. This may be because they are intended for up to international events, where a divorce between authoritative, live, strategic management and operational features, such as participant and audience registration, is favourable in place of other software. Whereas the relatively large software of small sports does usually include quite pervasive dialogue with referees, participants and audience. Less popular software of any sport may or may not include such features, being limited to just score authorities such as referees (6, 7).

Monologue capabilities. As all software had the ability to propagate scores and competition direction in some way, they all satisfy some level of dialogue with audience members. Differences are between what medium is used: all software bar (5) had video display – one or many displays with score, participant/team names, with most (1, 2, 3, 4, 7) being live.

Technological accessibility. Solutions involving a video display would be accessible to events with a high budget and/or frequency, as the hardware is usually bespoke and worth thousands to tens of thousands of pounds. In terms of visibility, it is limited to not the software but the hardware set-up. However, the most accessible results would be through smartphones, as a majority of adult and junior attendees access the internet through their mobile phone (*Statista*, 2019): 100% (n=1800) of interactivity with users aged 16-24, 97% for those aged 25-34, and 91% of those aged 35-44, could be conducted exclusively through their smartphone. Personal billing could be avoided by providing

local internet at events, even if limited to just the competition software. Some software (1, 3, 6) included the ability to view scores through smartphone.

Technological maintenance. Some software (1, 2) include online services and support for its use, whereas some other software (4, 5, 6) requires total administration by the competition management. Some software suites (3, 7) include a mixture of local and remote services. High quality or subscription-based software (all bar 5) offers long-term support, whereas open-source and free options (such as 5) usually stop development entirely after a few years. Some software was not included in this report as it stopped receiving updates upwards of five years ago.

Technical behaviour. All software reviewed required some form of installation on the machines of competition management, even with strong online services provided. Most software, bar (4, 5), is closed-source, meaning development is at the behest of the vendors. Most software makes it unclear what underlying technologies are used, except for (3, 5) which are a .NET 4.0 product and an open-source Java project, respectively. All software appears to use a database, though it's not clear without installation whether the database must be local or can be used remotely – the former is supposed, from documentation provided by (5). Most software (1, 2, 3, 6, 7) included features to directly monitor competition play in order to formulate scores, with some software (4, 7) including a smartphone app for referees to input scores.

Conclusion

It has been difficult to ascertain information about these existing solutions as their user documentation is usually the only look into their implementation details. All solutions required installation, which enforces a dependency on owning a laptop or desktop computer which can be at the competition. Two solutions (4, 6, 7) in particular are very similar to the proposed nature of this project's solution, including automated participant management, direction, and (4, 7) with referee dialogue. (6) specifically touted "a smatphone [sic] application Engarde Smart that gives to everybody a custom information, their next match, piste, time and opponent". Therefore, along with questionnaires, a different approach to solve the problem should be taken, such as to not require any installations on competition management's computers or smartphones.

Objectives and hypothesis

To ensure success of the project it should be thoroughly planned. Requirements should be defined and challenged, the objectives should be defined, potential options and associated risks should be considered, and an action plan should be developed (*Nokes*, 2007).

The initial hypothesis is that a mobile web-app with a composable back-end will facilitate communication of competition participants, management, and audience. Through providing live notifications of scores, participant direction, dialogue with the audience, and a robust connection to business logic. Further, an approach facilitated with HTTP API for software architecture will help produce these good effects. The requirement of this project is to test this hypothesis whole.

Throughout this report, the *general* business model will be one which is deployed with the artefact by standard, and *bespoke* business models are the swappable logic for different sports and competition types. The front-end is general.

Defining requirements

Accessibility of the artefact. The artefact must be accessible in terms of operability on personal devices; an accessible location to use the artefact; best consolidation of interface onto one front-end; user authentication should be practicable and obvious; follow industry practice for front-end design.

Security and authentication. The artefact must be immune from overt tampering by unauthorised or unauthenticated parties; the artefact must allow for configuration, application, and enforcement of discrete user roles; user authentication should be automated as far as possible, while remaining secure; common vulnerabilities must be mitigated; logging should be prevalent and accessible.

Dialogue of scores to participants & audience. Authorised users must have the ability to record and redact scores during a match, also after a match perhaps with higher authorisation; users should be able to subscribe to live scores and total scores; be able to view and filter past scores and score history, such as when each point was made or redacted, by who, and authorised by who.

Dialogue between the general business model and participants. Participants must be able to securely register, or withdraw; score subscription, viewing, and filtering should have the option to be tailored for themselves.

Dialogue between the bespoke business model and users. The bespoke business model should be completely abstracted away from most users, and instead communicate through the defined artefact interface. Administrators of the system could access it through a separate interface, for precise configuration, but this will not be facilitated by the artefact.

Separation of concerns between general and bespoke business models. The general business model must handle the authorisation of users, the actions they can take, and the data they can view; communication, subscription, and data viewing; the serving of the front-end; the logging; the initialisation of the bespoke business model.

The bespoke business model must initialise the general business model with a database structure, jargon, pages and views of data; handle the preparation of scores; dictate the direction of participants through broadcast messages; serve database requests (from the pre-defined views); persist all competition state not handled by the general model.

Dialogue between general and bespoke business models. The general model will respond to the bespoke model for these topics: initialising arguments, user attendance, user actions, termination of model. The bespoke model will emit these topics: registration of roles, addition of user inputs, addition of views. The general model will request differential patches between competition state snapshots, caching them for users mobiles which have fallen out-of-date, and the bespoke model will calculate these as required.

Flexibility between sports and disciplines. As the models use bespoke-model-defined messages, inputs, and jargon, it should not be coupled to any particular sport or discipline, or indeed any activity which shares a data-dialogue relationship. Administrators must be free to choose any implementation on any server for their needs.

Predicates of achievement

These have been marked with an (M) for *Must*, (S) for *Should*.

1. (M) The artefact can be accessed without undue complication;
2. (S) The artefact's interactivity is consolidated into one user interface;
3. (M) The artefact has effective user roles and authentication;
4. (M) The artefact has been developed with vulnerability mitigation in mind;
5. (M) The artefact operates with any implementation of valid-schema bespoke model;
6. (M) The artefact must be able to operate with more than one sport;
7. (M) The artefact must not require installation by competition management.

These requirements will be challenged during the development of the artefact, later in this report.

A comment on user experience

For this report, the hypothesis is technical – developing a generic architecture between different competitions, in the same artefact. The user experience at this stage is not a priority nor a concern – so long as there is some ability to prove multiple sports operating within the same artefact the hypothesis has been proven. Therefore, there will be no detailing of artefact design planning within this report, as it is outside of its scope. However, one of the predicates of achievement is no undue complication in using the artefact, and the hypothesis states that lay competition attendees should be able to operate it. In light of this good design practises will be followed, but their research and application will be in the author's charge, and only the results will be evaluated.

Technology choices and defence

Choices to be made. The only *essential* technology required to develop a modern software artefact is a functioning computer, which will not be in the scope of choices as the solution must be multi-platform itself. Other decisions that will aid the development can be summarised by *The Software Development Tools for Petascale Computing Workshop* (2007), which though provides advice for very largely scaled development, touches upon multi-platform, multi-language, and complex data transfer which becomes more relevant to modern computing. Found suitable, it advises these decisions:

- Target artefact:
  - Programming paradigm
  - Programming language
  - System architecture
- Development environments:
  - Integrated Development Environment (IDE)
- Correctness tools:
  - Regression testing
- Project management:
  - Version control
- (Excluded) Performance tools:
  - Sample profilers
  - Realtime profilers

How to decide. Numerous variables will be included in deciding which development tools and artefact designs to use: their suitability for the objectives of this project, accessibility to the project, computing performance and memory use, the author's existing knowledge of technologies and architectures. Most important will be the latter, as this project does not call for primary research at such a level as the artefact implementation – rather artefact design.

Programming paradigm. Rather than being a discreet choice of its own, a programming paradigm is a design consideration of programming languages, with most practical programming languages embodying multiple paradigms (*Popplestone*, 1999). There have been four major paradigms, corroborated by *Nørmark*, 2011: functional, imperative, logical, and object-oriented. Each paradigm offers a *discipline*, rather than extra features, to how a programming language will allow a developer to instruct a computer. Each paradigm also has benefits and drawbacks in terms of performance, memory use, readability, and modularity. For example, the memory models within functional programming usually must ensure immutability, which causes some duplication of data

and performance overhead in ensuring its correct use. These effects will be considered further within each programming language.

System architecture. The proposed artefact involves capturing data, routing data to the correct users, a data dialogue between two models remotely, a live data web-app, and a database. Therefore, a data-driven approach to system architecture would be beneficial, such that model behaviour is implemented as transactions in data, and a mapping of input data (scores) to output data (directions and results). As discussed earlier in the report, the general model should only be a marshal of data between the users' devices and the bespoke model. Its concerns would be acting as a broker between these two parties, to abstract from the bespoke model the concept of users, administrators, security concerns and pushing/pulling/polling updates in data. These data transactions and events should be brokered through a protocol such as Message Queuing Telemetry Transport (MQTT), as it offers a passive solution to routing between clients. It uses hierarchical topics with a patterned syntax for subscribing to any level, enabling clients to "subscribe" and "unsubscribe" from events as they wish (*Palattella*, *Soua*, *Stemper* and *Engel*, 2019). However, other options, such as a robust Application Programming Interfaces (API), could also be used.

Served as a web-app in dialogue with a sever, further considerations to the architecture involved in hosting the server can be made. Many different levels of server hosting are available, such as IaaS (Infrastructure as a Service), PaaS (Platform...) SaaS (Software...), and FaaS (Functions...). Determining between the more traditional IaaS/PaaS/SaaS/FaaS can be done through deliberating: the importance of security and scalability, necessity for server monitoring, need for training and operational requirements, the nature of computation, benefits of service & language integration. FaaS is very well suited to sporadic, ephemeral, and elastic workloads, such as responding to web requests (*Bowen*, 2019). This would be very suitable for this artefact, as its workloads are sporadic – competitions ongoing for only limited and unpredictable amounts of time; elastic – in that very few or many users may use the software at any one time; ephemeral, in that behaviour needs only be executed at the moment of view, not continuously monitored or enacted. However, at this present time, it has limited support for unpopular languages and runtimes.

Programming language. The proposed solution would call upon at least three different platforms: a web-client, a general model and a bespoke model. The existing programming languages of the author will be selected from. For client development: JavaScript, ClojureScript. For server development: C++, Java, PHP, C#, Clojure. All languages, bar Clojure/Script, are majorly imperative and object-oriented, whereas Clojure/Script are functional.

For the client, ClojureScript would be best suited, as it includes native frameworks and software patterns for working with dynamic data of any schema. It is important that the languages operate well with undefined schema, as multiple sports will require different ways of representing their play. ClojureScript can make use of existing JavaScript libraries, if any appear useful to development, whereas JavaScript cannot (easily) do the same. There would be a minor performance hit in using ClojureScript, as it includes its own data constructs – however, it does make good use of an advanced compiler[2]. An architecture such as One-Page-Application would effectively mitigate the effects of slow start-up time.

For the sever, C++ would offer too little abstraction from the machine itself, as this artefact does not require bespoke memory management. Java and C# are both predominantly statically written, whereby the structure of a program is planned before deployment; however, they do offer modular design. As with the client language decision, however, a heavily data-driven approach will be served best by Clojure. It provides abstraction to the management of program *state* over time, instead advocating *stateless* design. This would be useful for models to quickly recover from discrepancies in runtime, e.g. a power failure causes the artefact to restart, but as state was concentrated to a robust actor such as a database, the program can start-up and continue its runtime as if before. This is also very conducive to ephemeral runtime such as is offered by FaaS architecture.

It is noted that programming languages from a less restricted paradigm can indeed emulate restrictions. An example of this could be a C# or Java artefact which only uses data-oriented design patterns and operations. C# includes Linq for transactional data, and Java includes Streams for immutable mapping of data. However, their implementation within the two languages can introduce severe performance deterioration when improperly used (*Kowalski* and *Adamus*, 2017), whereas functional languages such as Clojure use very extensive, native support for the practise (*L'orange*, 2013).

System architecture revisited. With a chosen programming language the system architecture is further informed. Clojure/Script are capable of highly precise, complex data management and transporting techniques such as generating differentials between databases and merging them in the user's browser. ClojureScript has very capable "reactive" libraries available, offering abstraction to propagating these changes on users' web-apps[3]. A message brokering protocol akin to MQTT would be very appropriate for these languages, as pattern matching would be very easily implemented. The

2 As seen through https://github.com/clojure/clojurescript and https://github.com/hantuzun/awesome-clojurescript#reactive-programming

3 As seen through https://clojurescript.org/community/libraries

bespoke model exporting schemas for input and output data, and views for different roles, would also be easily implemented as Clojure/Script offer a very popular framework for defining these schemas as hierarchical dynamically typed arrays, then transformed into HTML directly (*Pratley*, 2017). Though a dependency on HTML would not be introduced to the bespoke model as the schemas can be translated to the targeting markup from a generic schema.

An issue between the use of FaaS and Clojure/Script arises, however, as current FaaS vendors do not support Clojure/Script natively, which will cause cumbersome deployment. However, it is possible to do through projects such as Lumo[4]. Unless proven easy to use, the fallback position is to use Clojure natively on a PaaS service with a perpetual server running.

As for ClojureScript's performance in a web-client, a Single-Page-Application would be fitting for its architecture, having the user load the front-end once in their browser, and using differential updates between clients to update the user on data they have subscribed to. The differential updates would ensure that full updates are unnecessary except for the first instance of the client loading, which will be useful if a client goes offline.

Further, for some user roles, needing an account could be completely mitigated by instead using HTML5 Local Data or cookies available nearly all modern web-browsers[5]. As part of standard web practise, the client would automatically be issued with a session key effectively identifying the device. The issued session key can persist between page reloads, and enable the server to recall the key if necessary. However, it may be completely unnecessary if data such as topic subscriptions can be retained in the browser for a long-term, and be sent to the general model with requests for new data. This inverts the need for state away from the general model and rather into user's devices. However, this method would only be practicable for users with no dialogue with the system, such as participants and audience members – administrators and referees would require a more authoritative form of identification.

Integrated Development Environment. This choice is mostly dependent on accessibility to the author, and support for the language in particular. It is not paramount to have an editor which is integrated with a language toolkit, only more convenient so. For Clojure, there are numerous options, though the author is comfortable with developing in a normal text editor with some syntax highlighting and using the command-line to invoke the tools available. For ClojureScript, installing a framework such as Figwheel[6] or shadow-cljs[7] for the client development can reduce time

4 https://github.com/anmonteiro/lumo
5 As seen through https://caniuse.com/#search=local%20data
6 https://github.com/bhauman/lein-figwheel
7 https://github.com/thheller/shadow-cljs

considerably, as they both enable hot-loading of code into the browser as a developer saves source files.

Regression testing. To help mitigate errors within a program, upon the discovery of undesirable behaviour, writing a ‘regression’ test is a valued practise in software development. This type of test would look specifically for evidence of that behaviour within the system, so then upon its correction, a memory is preserved of it. Later in software development, if the error re-emerges—seen as a ‘regression’—it can be precisely detected and fixed using previous discoveries and insight (*Huizinga & Kolawa*, 2007). This is adjacent to unit testing, targetting desired behaviour, which instead marks a succession. In order to implement regression testing, a testing suite would help provide the tools and framework. This testing suit can then be automatically or manually invoked upon artefact debugging and deployment. Support for regression, and other, testing is provided generally across a programming language as a whole.

Version control. Version Control Systems (VCS’s) are a file management approach designed to insulate and distinctly update master, preferably working copies of a codebase from active development. Typical solutions involve a master repository, of which a total copy is formed. Development is made on the copy, and once changes have been reviewed by a developer they are ‘committed’ to the master copy, usually with a comment on what changes were made. This is useful as a history of the codebase can be formed and preserved just through version control. Furthermore, these systems typically preserve a copy of every single commit differential, enabling developers to reverse commits to the beginning of a repository’s inception. Various models are used to broker modifications from more than one developer, offering tools to merge code line-by-line.

Version control helps developers buffer discrete stages of development, before fully merging them into a master copy, and will usually offer facilities to operate across multiple ‘branches’. It also ensures that any arbitrary point (as a commit) in a project’s history can be reviewed and reversed back to, without lack of foresight causing changes to be neglected due to their perceived importance (*Louridas*, 2006). For this artefact, Git version control will be used as the author has experience with and access to it. The artefact will be published onto Github also, to provide an off-site back-up of the codebase.

Sample and real-time profilers. Profiling a software artefact is used to develop an understanding of the performance characteristics of existing code. Different approaches to recording this information can be used, such as including debugging symbols within an executable—or ‘instrumenting’ the program—and counting the number of ‘hits’ that symbol encounters during normal application load.

This data can then be paired with the codebase, and line-by-line or function-specific metrics can be generated. Real-time profiling instead interfaces with the operating system kernel to obtain less precise information about performance, but with a far more intimate understanding of how the kernel prioritised the program, such as recorded hardware counters, branching, context switches, and power consumption (*Gebai & Dagenais*, 2018). However, these practices are unlikely to be included in the project as the resulting artefact is not intended for use in production.

Primary research methodology

Firstly, the demographics of interest should be defined; how the responding population were selected and verified (sampling approach, ie. census, convenience, random selection, self-selection, etc); what variables will be measured; how the variables relate to the hypothesis; and after the primary research halts accepting new data how patterns in the data were analysed in the same relation (*Norris, Plonsky, Ross, Schoonen*, 2015). Further, the primary research methods available to this report are likely questionnaire/survey or interviews. There is intention to use both, with interviews being used to incite meaningful dialogue in relation to the hypothesis.

The demographic for this research will be either competing athletes, competitive referees or umpires, or audience members of such competitions. Selection can be influenced through distributing the question material to sports facilities, though self-reported athleticism will enable a broader range of sports (i.e. outside of a sports hall or field). The primary research will not directly pertain to the original hypothesis of a system capable of supporting multiple sports through a certain architecture – this is a technical challenge requiring peer review. Instead, attitudes toward technology within sport, and the perceived likelihood of respondents in using such an artefact, will be measured.

A healthy population number could be around 15 to 20 respondents, which will likely obtain responses from multiple athletes of the same sport. Once data has been collected it should be statistically analysed, and also analyse individual responses to form a narrative of why such attitudes were given.

Ethical considerations. While conducting this primary research protecting personal data is paramount, subject to university guidelines and law. The European Union’s GDPR through the Data Protection Act 2018, Code of Conduct with the British Computing Society (BCS), and ethical research guidelines must be followed. The best approach would be to discard any responses containing personal information – with none on file, and full flexibility of consent for respondents, no violations can occur.

Deliverables, risks, and time plan

Software development methodology

In pursuit of project objectives it would be beneficial to organise workflow as an incremental process. Initial requirements, expectations, non-expectations, and dependencies of the artefact should be solidly defined, but implementation should be open to later understanding and investigation. Upon confirmation and defence that a particular implementation framework for development is suitable, implementation can begin.

This organisation method will somewhat follow the *Sashimi* model, encouraging relevant research throughout, with explicit lenience for development and testing. It is similar to the *Waterfall* model, deviating in-phase overlap for deliverables to be unblocked earlier than its dependants are wholly complete (*Hirotaka & Nonaka*, 1986). The programming stage will follow a test-driven *Incremental* process, testing added features repetitively until success; this model states that desired functionality is divided into small increments, implemented and delivered one after another in quick succession. Each increment is chosen so that it expands on the previous one, and is small (*Royce*, 1990).

Project management

To create a list of work packages and their estimated duration, recognising the dependencies of the project in such an order which can be carried out in a linear fashion was necessary. A Gantt chart was not used for this report as it became incredibly unwieldy, not providing any practical insight into the timescale of project sections. It is expected that before the new year a full literature review will have been conducted, tools and technologies chosen, and work on the artefact to have begun. A private *Kanban* board shall be used, though left unpublished as it will largely be subject to the author’s unpredictable time commitments, and not provide any insight.

The project is also supervised, largely by Chathurika Goonawardane, of which a logbook of meeting agenda, action items and previously completed action items will be recorded. This process will help build the project throughout the weeks, and help reflect on how well the project kept to the suggested work packages.

Refer to *Appendix 1* for the estimated work packages.

Refer to *Appendix 4* for the logbook.

Risks

Citing University College London's Department of Geography (2017) for potential risks associated with computing equipment, risks can include the following:

- Display Screen Equipment (DSE) use
  - Upper limb disorders
  - Backache
  - Fatigue and stress – can be overcome with frequent breaks and an ergonomic work environment, such as a good keyboard and mouse, or a keyboard layout such as the Dvorak Simplified Layout.
  - Temporary eye-strain (not damage) and headaches
  - Postural problems
  - Visual problems
  - Fatigue and stress – may be overcome by addressing postural and visual issues, and incorporating frequent (even if short) breaks.
- Limited physical space
- Lighting
  - Sufficient clear lighting of any kind is necessary.
  - Avoiding reflection and glare is important.
- Noise – equipment used should not distract and impair concentration or prevent normal conversation.
- Temperature and humidity, solved with artificial controls and sufficient ventilation.

Risks relating to the project's integrity, selectively citing *Mar*, 2016:

Loss of work • Reduce loss of revisions of code and other plain-text documents using a Version Control System (VCS), e.g. git • Reduce loss of any files through VCS systems, or cloud backup applications such as Onedrive, or local file mirroring applications such as Syncthing • Reduce loss of work being written with frequent saves and autosaving	Problems with scope • Ill-defined scope, through error or omission • Scope 'creep' - uncontrolled inflation of scope • Inaccurate estimates in time and effort – err on the side of caution • Scope is too narrow • These problems can increase likelihood of loss of motivation
Design problems • Infeasible design • Lacking flexibility • Unsuitable or low quality design choices • Design fails peer review	Resource problems • Resource shortfalls • Uncontrolled learning curves – using only technologies with personal experience helps mitigate this
Architectural problems • Lacking flexibility • Unsuitable choice of architecture • Infeasible architecture	Problems with communication • Misunderstanding of requirements • Too little communication • Too much time spent on communication

Technical problems • Components not fit for purpose • Components that are unable to scale or aren’t extensible • Components that are not interoperable • Components that are not compliant with standards and best practises • Components with known security vulnerabilities • Over-engineered or unmaintainable components • Components that lack stability, reliability, or are no longer in support • System outages	Other threats • *Force majeure*; disruptive acts of nature • Technological change impacts project, related to security or great innovation • Upheaval of original hypothesis • Failure to follow methodology • Counter-party risk, i.e. externalised risks transferred back to this project • Unexpected major rejection of final artefact • Failure to follow majority of objectives

Artefact development

Development log

This is a small crude logbook for development taken day-by-day on the artefact.

Day	What was done
0	Preparing project: cloning previous full-stack project, testing it's still functional
1	Changing associated docs, writing initial schema, disable all client-side functions concerning schema
2	Correct transmission of schema and database to client, begin display of some data
3	Customise discreet competition pages, separating events by schema views
4	Customise notification (events) menu, filtered by selected competition
5	Further filter events by local subscriptions, enable user to select subscriptions
6	Enable user to view their subscriptions/notifications
7	Develop schema for inputs, organise roles, harden data being transferred to client
8	Implement the display of inputs with authorisation
9	Implement sending input messages to server and merging into database
10	Implementing client and model server polling
11	Implement model tracking of scores, updating client inputs, creation of new matches

Tools and environment

This includes all programs, applications, and configurations which are not deployed with the final artefact.

Universal tooling. To ease development, a generic environment of tools was formed. The operating system chosen was Canonical's Ubuntu[8], as it has primary support for numerous software development packages, and the author is most comfortable with it. Gnome's Gedit[9], a mildly extensible text editor, was used for text editing and simple coding/scripting tasks. Microsoft's Visual Studio Code[10], a highly configurable IDE, was used for Clojure and ClojureScript development and in aiding deployment. Mozilla's Firefox[11] was used for front-end development, experiments, testing, and any internet research conducted. The Luminus micro-framework[12] was

8 https://ubuntu.com/
9 https://wiki.gnome.org/Apps/Gedit
10 https://code.visualstudio.com/
11 https://www.mozilla.org/en-US/firefox/new/
12 https://luminusweb.com/

used to initiate the front-end and general model as one integrated project, providing templates and boilerplate code suitable for most reactive web-apps.

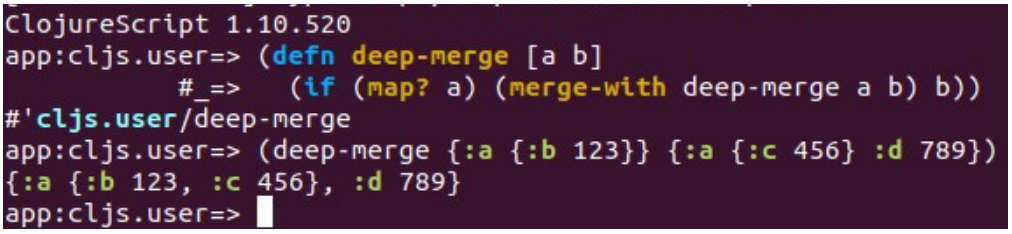


Illustration 7: Usage of the ClojureScript REPL to test a simple function called function "deep-merge"

Language-specific. In order to use the programming languages chosen, Clojure and ClojureScript were served by Leiningen[13], a project, build, and dependency automation and declarative configuration framework. It provides a command line interface, REPL, plugin system, ahead-of-time compilation, deployment, and new project generation (*Emerick, Carper, Grand*, 2012). The most important plugin used within Leiningen for this project was lein-figwheel[14], facilitating browser hot-loading of ClojureScript. For the use of NodeJS, a language chosen to oppose the use of Clojure/Script in writing a bespoke model, the npm was used. It provides package management, package registry searching, and automatic dependency resolution (*Ellingwood*, 2014).

Back-end tooling. To aid development of a bespoke API after the completion of the general API Wireshark[15] was used. It is a network packet analyser which helped to match the format and transport configuration of HTTP requests, in terms of security headings, payload format, and confirming communication between the bespoke model and general model. To aid in linting, basic and library experimentation, database inspection and live intervention, and immediate logging, the Clojure REPL (Read, Eval, Print, Loop) was used. It provides a running, continuously compiled process with command line entry of new functions, side-effects, and inspection of existing programs and environments (*Clojure.org*, 2020).

13 https://leiningen.org/
14 https://figwheel.org/
15 https://www.wireshark.org/

Front-end tooling. For inspecting the live application database, and monitoring reactive events within the ClojureScript application, re-frisk[16] was used. It displays application state as a navigable tree structure, events as a timeline with a nested map of stateful effects within the application, and subscriptions active within the current view. It was critical to help understand the live and historical execution of the application and its events over time. To enhance the REPL experience for

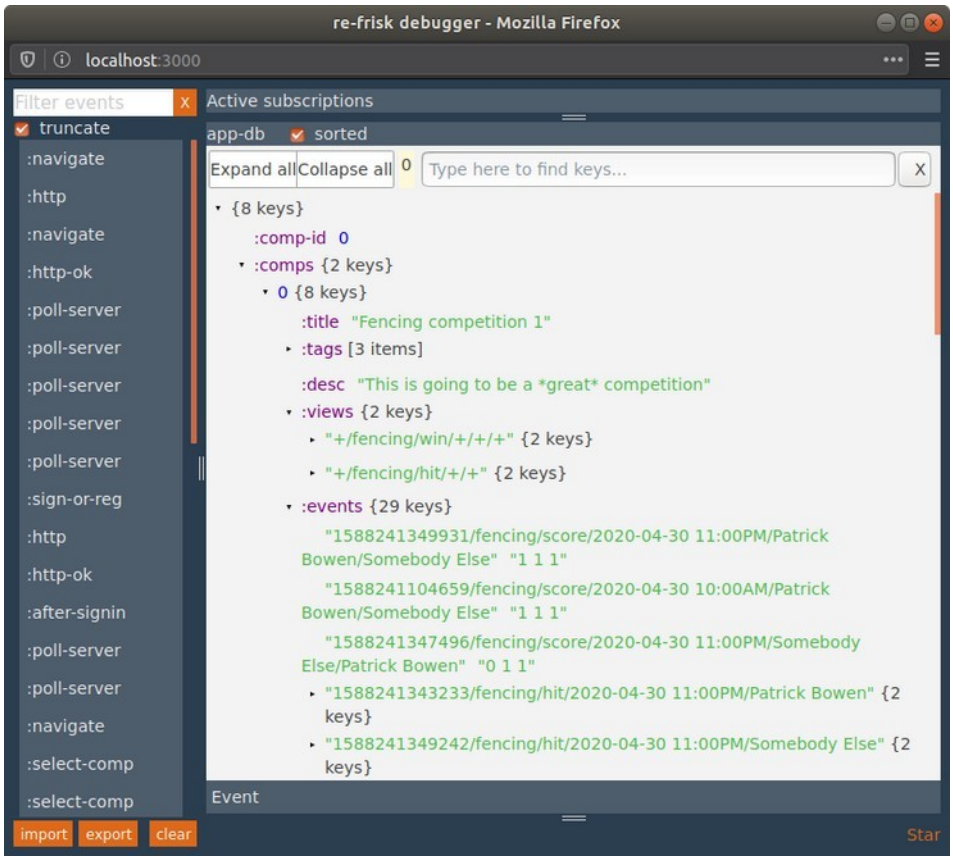


Illustration 8: Screenshot of re-frisk inspector during application runtime. Here the re-frame application database is a navigable tree, and events as a list.

ClojureScript, rebel-readline[17] was used. It provides multiline editing, immediate evaluation, documentation inspection, syntax highlighting, input and output formatting. It was mostly used to quickly format EDN[18] (Extensible Data Notation) and cause runtime interventions within the application database.

Miscellaneous resources. Browsing for new libraries and projects to research and use employed Github[19], a collaborative version control website. The website ClojureDocs[20] was used to learn about the Clojure and ClojureScript languages, and the philosophy behind them to improve their

16 https://github.com/flexsurfer/re-frisk
17 https://github.com/bhauman/rebel-readline
18 https://github.com/edn-format/edn
19 https://github.com/
20 https://clojuredocs.org/

use. Various personal and corporate blogs aided understanding of the journey of developing an application stack akin to this artefact. The Discord community for Clojure *et al* technologies[21] helped provide personal, live advice on operational and technical issues which arose during development. StackOverflow[22], a collaborative forum for solving technical problems across a range of technologies, also aided with these types of issues.

Libraries deployed with the artefact

General model libraries. To ease the use and transformation of hash-maps, Medley was used. It provides a corpus of pre-written utilities which keep production code clean and understandable. It supplements and utilises the depth of existing core utilities. For the hashing of user's passwords, Pandect was used. Its SHA256 utility provided a simple way to supply string-to-string hashing. To act as a persistent storage, and in place of a relational database, Duratom was used. It simply watches and intercepts all modifications to a Clojure stateful 'atom' of data, and saves them to a file on disk or on the cloud. Then, upon the next application load, the atom is initialised with the serialised content from the disk. Clojure 'atoms' are one of the few constructs available providing persistent state, as opposed to the normal purity[23] of functions. This artefact stored its state as a nested hashmap, which is easily serialised by Clojure, and easily readable by eye.

Front-end libraries. As with the general model, Medley was also used in the front-end for the same utilities. For the generation of EDN-style HTML from Markdown, markdown-clj was used. It takes a Markdown formatted string and outputs it as nested Clojure vectors used within some HTML-emitting libraries. For page style, the Bulma CSS (Cascading Style Sheets) framework was used. It provides a consistent and extensive style for non-trivial modern web applications and sites.

On re-frame, a front-end framework. For the general architecture of the web application, and to help develop it as an SPA (Single Page Application) re-frame was used. It uses a unique "six domino" pattern to provide reactive changes within the artefact. Reactivity in a web application is an intelligent propagation of state change based on data modification. Registration of data endpoints and data modifications helps inform where effects should take place. For example, if within re-frame the event is raised to change the page route, which modifies the application database, all website generation (document, style, HTTP requests, &c.) dependant and watchful of the route will be updated. This drastically reduces the need to manually trigger changes in page generation, and

21 https://discord.gg/hJzGQ5T
22 https://stackoverflow.com/
23 "Purity" is a term in the functional programming paradigm referring to functions which will always return the same values for the same parameters (*Okasaki*, 1998).

helps enforce a separation of concerns between view and business logic, important in all software development (*Mitchell*, 1990).

Bespoke model libraries. The bespoke model was written in NodeJS. For HTTP GET and POST requests, got was used. It provides asynchronous interaction for HTTP requests, responses, and errors, and simplifies the setting of headers and JSON serialisation/deserialisation. For encoding between JSON and EDN, jsedn was used. It provides simple functions to encode between non-native EDN strings and native/string JSON objects.

Expectations and reality

General comments

On EDN. Extensible Data Notation is a standard borne from a subset of Clojure's syntax, making it serialisable and readable by the language itself (*Hickey*, 2014). Similar to JSON, maps consisting of keys and values can represent and help index access of complex data. Unlike JSON it is able to represent keys as any of the same datatypes which can be values, including 'composite' keys made from vectors or hash-maps themselves. Therefore serialisation from JSON to EDN is lossless, while the opposite can lose valuable information such as composite keys and ubiquitous static 'keywords'. EDN is particularly useful for unevaluated representation of HTML, as the re-frame library and Hiccup library both support (in slight variations). Variable references and expressions can be included and are evaluated inline, then serialised to their string form upon HTML emission.

Table 2 - Sample of JSON aside EDN

JSON

```
{
  "owners": [
    {
      "name": "Patrick",
      "gender": "male",
      "items": ["netbook", "desk"]
    }, {
      "name": "Mikha",
      "gender": "female",
      "items": ["watch", "quartz"]}]}
```

EDN, using a set `${...}` instead of vector `[...]` for items

```
{
  :owners {
    {:name "Patrick" :gender :male}
      #{"netbook" "desk"}
    {:name "Mikha" :gender :female}
      #{"watch" "quartz"}}}
```

EDN representing HTML with inline CSS

```
[:body
  [:h1 "Testing, testing, " 123]
  [:p "This is " [:i adjective-variable] " exciting"]
  [:span {:style {:color "#00F"}} "I'm blue."]]
```

In the long-term, this project's artefact should avoid EDN as a transmission specification as it is not as robust for data types not efficiently encoded as a string (such as binary data). Its formal specification is also lacking (*Kaasinen*, 2019).

Architectural comments

On jargon and bespoke interfaces. Original specifications did not provide a solution to supporting multiple sport and competition jargon. To support them, a generic interface was designed around views of tables of data specified by the bespoke model. Instead of providing a query interface or similar, bespoke models simply emit pre-formed HTML in EDN. This is directly placed within the tables, and at this time (with no filtering) it allows a bespoke model to display any sort of custom forms, styling, images, JavaScript, and has the potential to modify the rest of the program too. This is recognised as a critically dangerous inclusion within a true product deployment, as it allows potentially malicious or malformed code to be included and evaluated within the page. However, as this artefact is a prototype it is permitted, considering that it is easily possible to filter the EDN for allowed HTML only. This would be quite safe as deformities are already rejected by the HTML emission library, and EDN is easily transformable as an integrated part of the Clojure language.

On application hosting. This artefact's components were simply hosted on the author's laptop used for development. As HTTP transport was utilised between the components there's no foreseeable issues with hosting or connecting the components remotely, at the bandwidths and polling frequencies currently used. The front-end component will work on any modern browser which supports ECMAScript 3 or above (ClojureScript FAQ[24]), which includes most mobile browsers[25]. In a real setting, the general model would be hosted by one actor and the bespoke models by any other actors. The bespoke models could be served from the same server and connect through localhost addressing, or even have an inverted dependency where a front-end is capable of implementing business logic. This was a real consideration when thinking about technologies for the bespoke models in this prototype, as ClojureScript could have served as a model very well, integrating EDN natively. However, this would have not diversified the languages being used.

On FaaS architecture. Initial thoughts around application hosting suggested the use of FaaS (Functions as a Service) architecture. This would be implemented as each bespoke model endpoint being an ephemeral handler which processes new submissions, and outputs new inputs and events. For the bespoke model it would be very suitable, only not used due to time constraints. The bespoke fencing model was written in NodeJS, a very well supported web-server language on FaaS service

24 https://clojurescript.org/guides/faq-js#does-clojurescript-work-in-old-browsers
25 As verified per https://kangax.github.io/compat-table/es5

offerings (*Bowen*, 2019). However, for the general model to be implemented as serverless, not only would a more durable storage method be required rather than persisted Clojure atoms, but Clojure itself would be better replaced with another, more supported language. Clojure has a long warm-up time, and without heavy ahead-of-time compilation would incur great overhead (*Normand*, 2019) – this compilation deployment requiring precise configuration on a FaaS service.

On data synchronisation. Active polling was used instead of keep-alive connections or push technology simply to save on time, complexity, and make it easier to debug. An empty polling response incurs very little overhead, though paired with the method of updating inputs it can be relatively intensive. In the future the artefact would benefit greatly from a more suitable method of data synchronisation and transport, as mentioned in other sections of this report.

On a stateless bespoke model. In this case, while implementing the fencing bespoke model it was discovered that very little information needed to be retained there. It was possible to offload most state into the general model. As an example, a referee would invoke the MQTT message to create a new match, using information such as the match time and participants. The bespoke model emits new inputs for this match, for incrementing player's scores. However, instead of the bespoke model also keeping track of these scores, they are embedded into the input submission MQTT messages instead, and destructed each time. Per input submission, for the case of fencing, the inputs were replaced again containing the two new scores. So long as the information was within the message topic and not the payload, the general model would guard against any unexpected message topics, such as an unexpected jump in score. Though all inputs were from a trusted actor – the referee – regardless, making score inputs not particularly sensitive, while from valid referees and logged. The only state required by the bespoke model is the number of events and inputs processed to pull new ones from the general model.

On universal logging. As a predicate of achievement in the artefact's design, it was concluded that to ensure the best security universal logging must be made available to participants and application administration. Due to the nature of the message architecture, and using a flat store of chronological events, dedicated logging is unnecessary. Events are treated as an immutable record of all competition state and interactions. To browse these events is simply to use the user interface and view the data through tables, or facilities to provide full inspection of all events would be undemanding to implement.

Implementation comments

On data storage. Traditionally, it would be expected to use a relational database or similar to persist competition state, user sessions and credentials, logging, and other information. However, for this prototype artefact, a very transparent option was used instead—Duratoms. As mentioned previously in this report, an atom is a persistent object in Clojure which persists state between contexts, passed as a reference. It is implemented with Software Transactional Memory (STA[26]), guaranteeing memory safety even during concurrency, with a framework in place to customise such transactions. This object in memory can be any EDN data-type, JVM class, or primitive, so this artefact employs them for: front-end application state, back-end in-memory database, and in the bespoke model as a simple synchronous commit/pull style object. The duratom library was used in the back-end to provide backing this in-memory database to a file on the disk. This approach, while atomic and safe, loses all value as a fast, efficient method of persisting large amounts of data for easy indexing and lookup. However, as this artefact is unlikely to store large amounts of data, a hot already in-memory database can forgo any disk access, benefiting from existing linked-nodes for data access.

On token retrieval. To help ensure the integrity of a client interacting with a Ring server—the HTTP & routing library common in ClojureScript back-ends—a CSRF token (Cross-Site Request Forgery) is employed. It is supplied to clients upon their first interaction with the server, and once a form submission has been completed. Without a recognised token the server will reject HTTP POST requests with a 401 (Unauthorised) HTTP response. Instead of circumventing this mechanism for the bespoke models, a model may request its token on the "/token" HTTP endpoint. This will remain unchanged between requests, which does not uphold the purpose of the token, but keeps the server for using tiered logic to serve requests. The CSRF protection is provided through middleware between the transport and application logic.

On deployment. To deploy this artefact in full, two deployments must be made. The back-end and front-end are deployed together as one server package, compiled through the Leiningen project manager to produce a stand-alone JAR (Java archive) executable. An alternative option is to use a build tool called GraalVM which analysis and pre-compiles the Clojure application into a native executable with an internal Virtual Machine (Taft, 2020). The front-end application is served as JavaScript, packaged within the JVM deployment. The fencing bespoke model is simply a flat NodeJS file with a schema file (which could also be internalised into the script itself). It is deployed whole, as a source file, and executed within NodeJS.

26 https://clojure.org/reference/refs

On notifications. A major planned feature of the artefact was to provide notifications to users, based on subscriptions. While the artefact does display an overlay notification message upon an event, it was unable with its testing configuration to provide native notifications through the HTML5 API. This is due to a requirement for the JavaScript application to have been served over HTTPS. To configure HTTPS, a certificate would have to be issued from a vendor such as Let's Encrypt. While the author is capable of the process, it was decided to neglect native notifications to save time.

Overall comments

The following comments directly correspond to defined requirements earlier in the report, and predicates of achievement. Premises are copied from their original declaration – i.e. "such and such feature *has been achieved*", the feature was previously defined.

Accessibility of the artefact. The artefact has been accessible and operable on personal devices, accessible through IP address. Any interface has been consolidated to the front-end (though deficient in many features, such as adding new competitors), rather than external configuration being required. User authentication through email and password has been practicable and obvious. Industry industry practice for front-end design has been followed, using a reactive Single-Page-Application model and modern, mobile-first CSS style.

Security and authentication. The artefact has become immune to overt tampering by unauthorised or unauthenticated parties, through the use of Cross-Site Request Forgery tokens, required email and password registration and general model authentication (not bespoke model), and valid inputs delivered to only signed-in users. However, the artefact has much more to implement before this objective is complete, as attacks from individuals with an understanding of the underlying model are still possible. The artefact does *not* currently allow for configuration, application, and enforcement of discrete user roles, rather a two-tier system was adopted – signed in and not. However, there were no complications in implementing such a system other than time necessary to do so. User authentication perhaps has not been automated as far as possible, as sign-in methods such as email/SMS one-time sign-in links could be used. However, while remaining secure, the artefact uses email and a hashed & salted password combination – providing security to users even after a potential database breach (*Clarke,* 2009). Common vulnerabilities have been mitigated, such as cross-site scripting, but transport of data between client and server was not secured for the development artefact – HTTPS should have been configured for this, with no complication noted. Logging has been prevalent and accessible to a point, though the general model could provide much more with no undue complication.

Dialogue of scores to participants & audience. Authorised users have the ability to record and redact scores during a match, but not currently after a match. Users are able to subscribe to live scores and total scores, and to view and filter past scores and score history. However, further information, such as the submitting referee, was not included – but can be with no complication.

Dialogue between the general business model and participants. Participants are able to securely register, but currently not withdraw their account. Score subscription, viewing and filtering can be tailored to match their preferences.

Dialogue between the bespoke business model and users. The bespoke business model is completely abstracted away from all users at this time, instead communicating through the defined artefact interface.

Separation of concerns between general and bespoke business models. The general business model can handle the authorisation of users, the actions they can take, and the data they can view. However, responsibility of subscription and data viewing was passed to the front-end application. The serving of the front-end is by the general model. Logging was made a side-effect of normal operation. The initialisation of the bespoke business model with persisted data is implemented.

The bespoke business model correctly initialises the general business model with a database structure, jargon, and views of data, but not discreet pages a user can navigate. Scores and other data is not prepared by the general model, instead by the bespoke model. The direction of participants is currently not dictated through broadcast messages from the bespoke model, but could be implemented with no difficulty. Database requests (from the pre-defined views) are not served by the general model, as this responsibility was another passed to the front-end. All competition state is persisted within the general model, as it was deemed convenient.

Dialogue between general and bespoke business models. The general model was expected to respond to the bespoke model for these topics: initialising arguments, user attendance, user actions, termination of model. However, the general model actually accepts these requests with a simple 200 HTTP status code, merging the payloads into its own database. The life-cycle of the two models was drastically reduced from expectations. Aside from user roles, the bespoke model is able to emit addition of user inputs and views. Dependencies were again inverted, as instead of the general model will requesting differential patches between competition state snapshots, the bespoke model and front-end do so using the same API endpoints.

Flexibility between sports and disciplines. The artefact is uncoupled from any particular sport or discipline. Administrators are free to choose any implementation on any server for their needs.

Fulfilling predicates of achievement

Importance	**What**	**Achieved?**
Must	The artefact can be accessed without undue complication.	Yes – visiting it like any other web-application.
Should	The artefact's interactivity is consolidated into one user interface.	Yes – all current features are accessed through the web-app.
Must	The artefact has effective user roles and authentication.	Partial – no more than two roles, but authentication between the two is enforced.
Must	The artefact has been developed with vulnerability mitigation in mind.	Partial – some mitigation, such as HTTPS, was not configured.
Must	The artefact operates with any implementation of valid-schema bespoke model.	Yes – a schema is defined and it accepts as such.
Must	The artefact must be able to operate with more than one sport.	Yes – all jargon, behaviour, and data display is externalised or generic.
Must	The artefact must not require installation by competition management.	Yes – the general and bespoke model/s can be hosted and configured over the internet.

Demonstration, testing, and questionnaires

Unfortunately, due to the 2020 COVID-19 pandemic, an in-person demonstration of the artefact with a local fencing club is unfeasible. This has caused only one questionnaire to be issued, rather than two, and have it obtain attitudes on technology in sport in general, and how this project's artefact could perceivably improve it. In lieu of a walk-through concerning a physical demonstration, a completely virtual walk-through will be given.

Demonstration walk-through

Overview

The final artefact, titled Competite, consists of three major components: a front-end web-app, a back-end general model, and the capability to support one or more bespoke models simultaneously through a HTTP API. As of this report's submission only a Fencing bespoke model was produced, but the schema for a Karate competition was also loaded into the general model. The Fencing model is using very generic interface with the general model that it's very perceivable to support other sports.

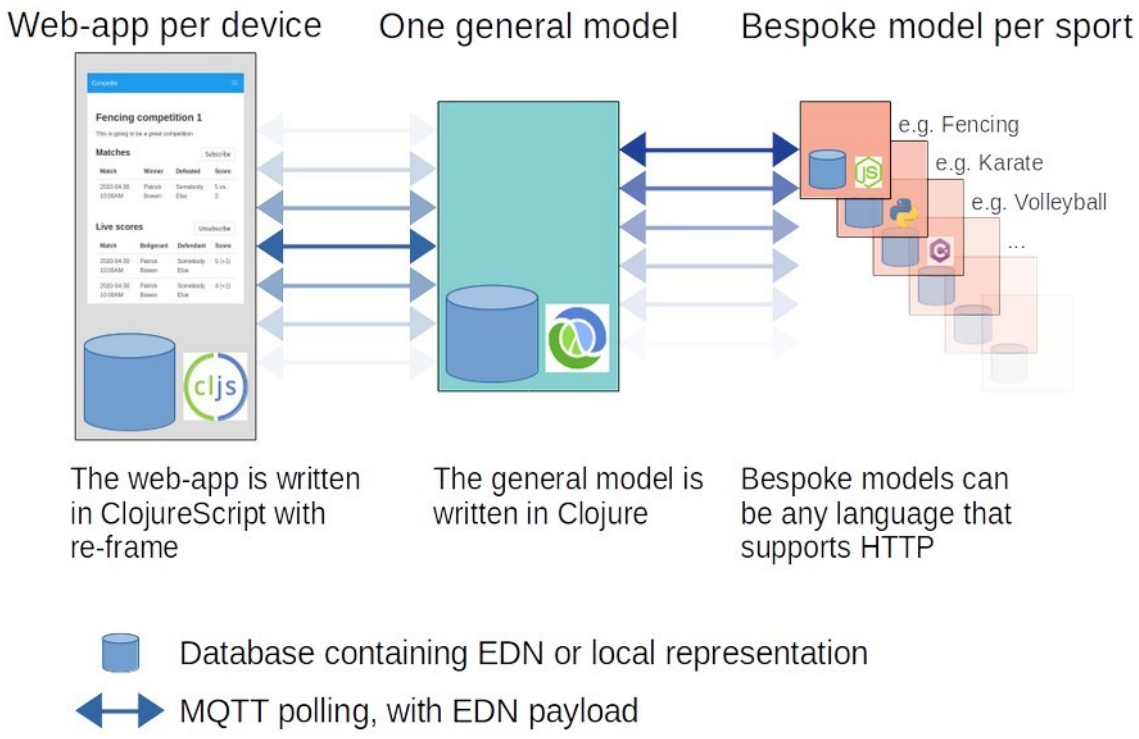


Illustration 9: High-level overview of Competite's multi-component architecture

Data store type		Description
Schema	Competition information	Created by bespoke models, persisted within the general model, copied into the web-app, the competition information contains all data relating to the competition such as its title, tags, users, views, inputs, and the events transcript.
	Views	A map of MQTT message topic subscription patterns to a map of a header title and a table header as Hiccup-style EDN. Used by the web-app to display any MQTT messages conforming to those patterns.
	Inputs	A map of ordered topic keys to payload values. Payloads are HTML represented as Hiccup-style EDN. Inputs can be created, replaced, or disabled by resending the topic with a new payload.
Events transcript		Only grows with new records. A chronological transcript of timestamped topic keys to payload values. Payloads can be any EDN data, but topics are a string only.

A user can visit the web-app on any modern browser, including on mobile, and be greeted with the home page within five seconds – with perfect connection, though most time is used for loading the app through computation. After the app has been loaded into the user's browser it will never refresh the whole page, instead navigation modifies the fragment identifier (part of a URL proceeding a # hash symbol), which reactively modifies the content shown on the page.

In this early iteration of the artefact, data from all competitions is sent to the client, amounting from 1kB to 10kB in testing – this is only the bare Karate competition and a fencing competition used for testing. It is entirely possible to load the competitions dynamically as the user navigates to them, and filter events by age, but was neglected for this prototype. The homepage is also loaded as a Markdown document.

If a user registers through the web-app their credentials are hashed and stored in the general model's database, and not shared with any bespoke models. At this maturity of the artefact a method of authenticating a new user with a bespoke model's inputs was not implemented, but could have been as an input itself. A roles system was planned for, but quickly dropped in favour of a two-tier "signed in or not" method. A signed-in user which was declared a referee within a bespoke model's schema was given access to the model's inputs.

At this stage the artefact does not guard against bespoke models interfering with other competitions' data, both reading and writing. More dangerously, the artefact does not yet guard against unauthorised users sending custom MQTT events into the system. However, it would be a very simple feature to implement – filtering MQTT message topics per user and bespoke model

authentication level. At this time it is only recognised as a huge risk making the artefact unsuitable for production use at this time, but also recognised as easy to rectify.

Also at this stage, instead of using a “keep-alive” connection with the general model, both the web-app and Fencing bespoke model used two second polling to query new data. This would induce an up to six-second delay from a referee submitting data to the response returning to the client – 2s for font-to-general, 2s for general-to-bespoke with immediate reply, 2s for general-to-front. To ensure events were only synchronised once, they were sent in chronological order respective to the number of events either the web-app or a bespoke model already had. This meant that a discrepancy of only, say, four events invoked the general model to only reply to the polling with the latest four events.

Unfortunately, for this early prototype, as inputs were able to be dynamically changed by the bespoke model – and were also polled for – it was decided to always respond with all the available inputs to each client of the general model. This could have been addressed rather by either hashing or supplying an ID for each individual input, or hashing all inputs received (as EDN). Then the general model would send only the modified inputs in the former solution, or all inputs in the latter but only once there was any modification. In the current prototype, inputs are replaced based on their MQTT message topic.

All polling responses from the general model and within that model itself were incorporated into local databases via deep merging of hash-maps. Below is an example of a deep EDN merge, where one modified and one new entry is merged into an existing map with two entries, with one collision.

Table 3 - EDN deep merging of maps

Original	To be merged	Result

{
"fencing/end/9AM"
[:button "End match"]
"fencing/end/10AM"
[:button "End match"]
}
	{
	"fencing/end/10AM"
	[:p "Match finished."]
	"fencing/end/11PM"
	[:button "End match"]
	}
		{
		"fencing/end/9AM"
		[:button "End match"]
		"fencing/end/10AM"
		[:p "Match finished."]
		"fencing/end/11PM"
		[:button "End match"]
		}

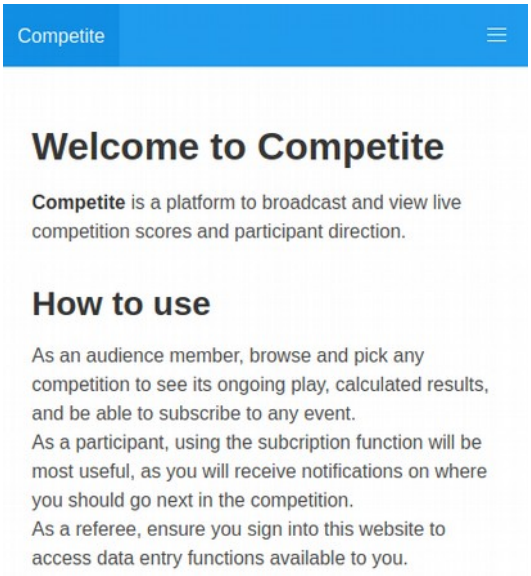
The 9AM match entry has remained untouched, but the 10AM entry has been replaced. The 11PM match entry has also been included. Using only this mechanic has meant that message topics cannot be completely erased from the database, only replaced with nil (null) or "". In the Fencing model the referee submission entries are replaced with the final scores. Also, vectors and nested maps

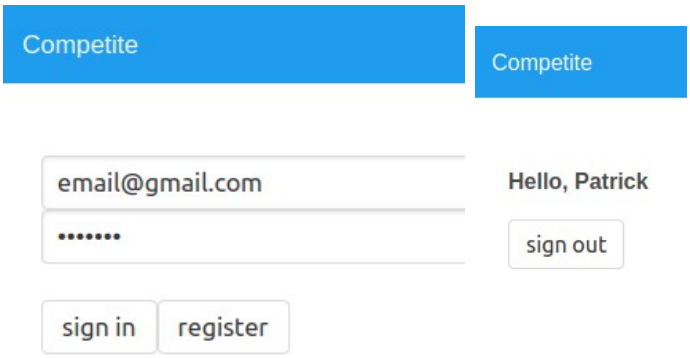
deeper within the main map would also be merged, such that a vector of [0 1 2 3] merged with [0 5 6 7] would result in [0 0 1 2 3 4 5 6 7].

Merging MQTT events by unique, time-stamped topic de-duplicated repeat sends of a message, a common problem in web communication. Even if the transport layer tries to ensure the resending of a network packet, without duplication, failure to acknowledge receipt of a message can cause a client/server to resend data. However, if it is the user re-issuing the event, a new timestamp is added, which would duplicate the event without extra measures (*Fehling, Leymann, Retter, Schupeck, Arbitter,* 2014).

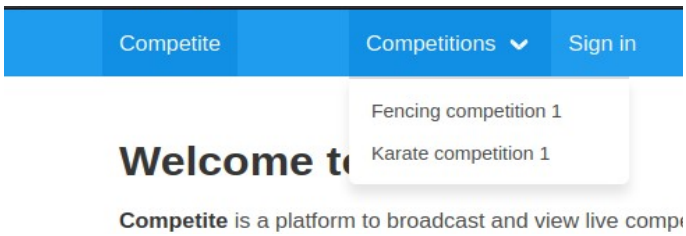
The mobile web-app

The mobile web-app was written in ClojureScript with re-frame with the Bulma style. The homepage is loaded from a Markdown document upon the application's initialisation.

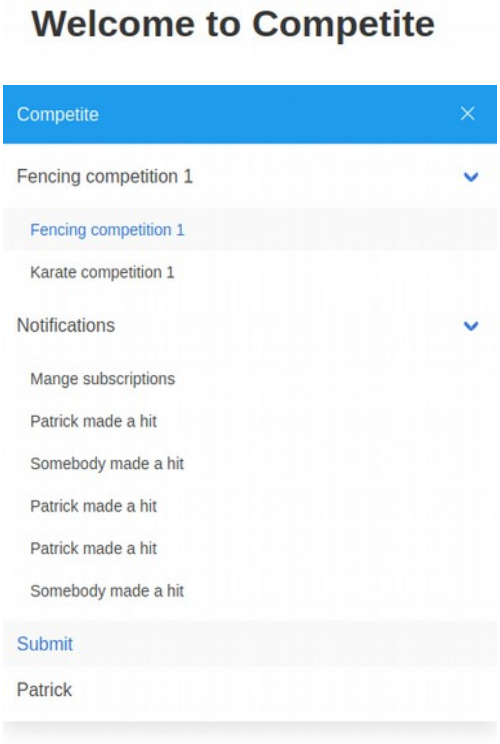




A registered user is able to sign into the web-app. If a user has been authorised by the bespoke model owner the user is able to access the Submit page.

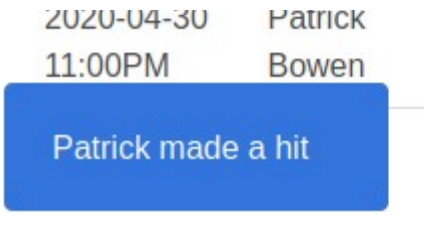


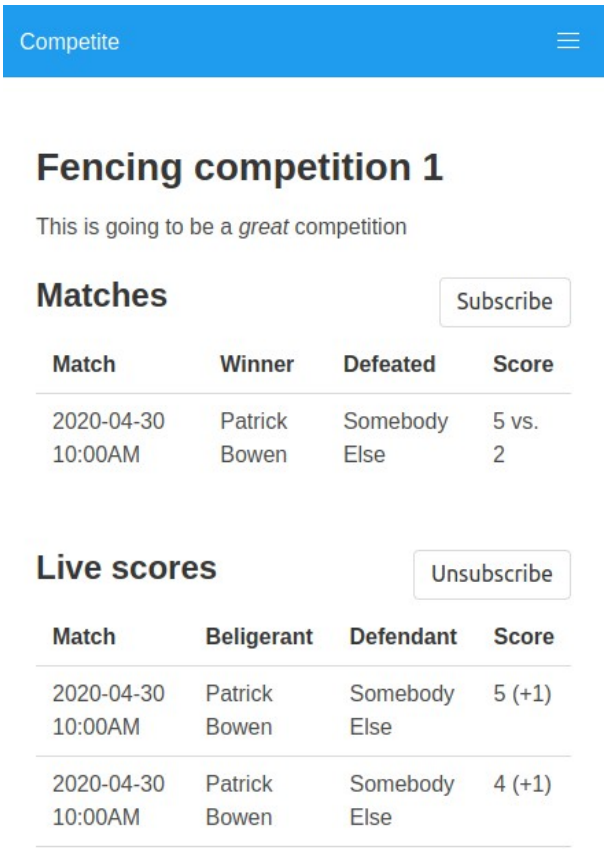
It includes a drop-down navigation menu on mobile, or a navigation bar on wider screens.



More options appear once the user has selected a competition or has signed in. If the user has subscribed to some events they will appear in the notification menu over time.

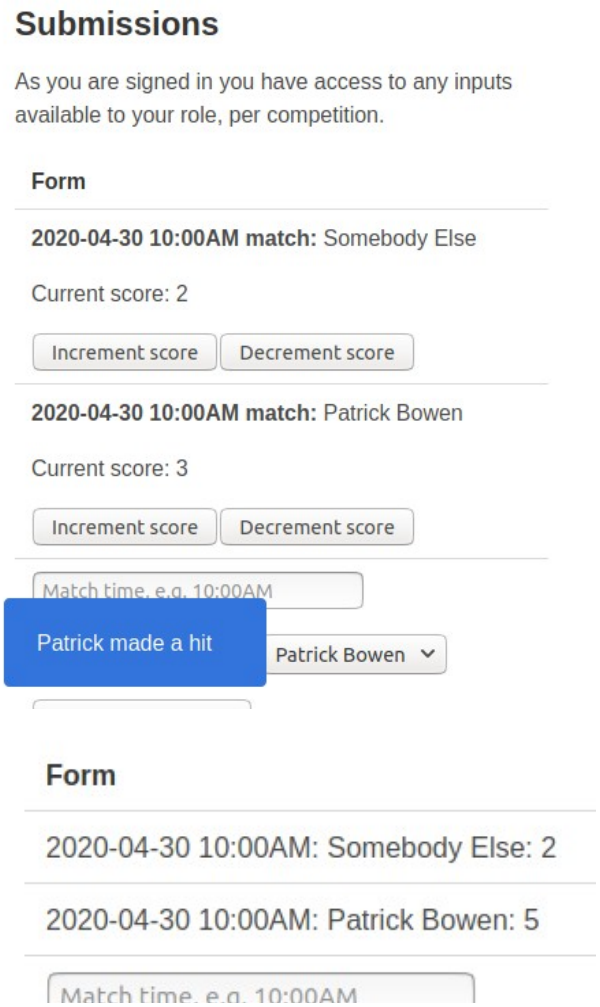
As new notifications are created, an overlay is shown above the open page.





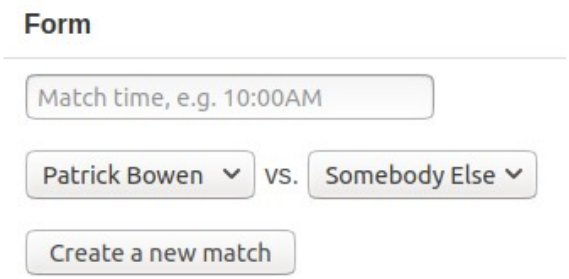
The bespoke model initialises the general model with a schema, including views a user may browse on the front-end. On the left is the two views available in the Fencing bespoke model. One filters for "+/fencing/win/+/+/+" and the other for "+/fencing/hit/+/+". Per event, a HTML table row is pre-formed by the bespoke model, and displayed directly in the tables here. Theoretically the bespoke model is capable of emitting any valid HTML, CSS, and JavaScript. This enables displaying images, videos, sounds, files, custom formatted notification messages, and more, with the same approach used to provide rich forms in the submissions page.

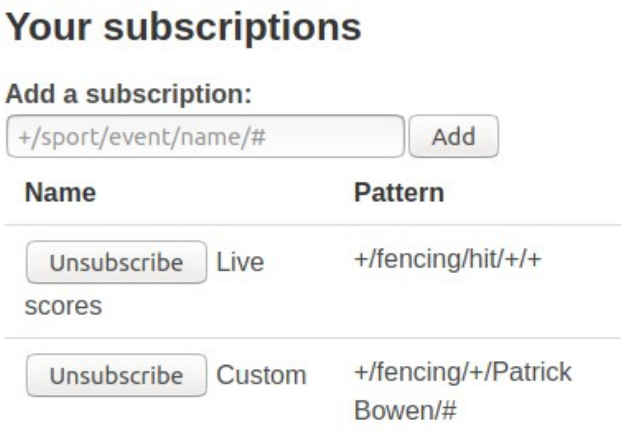
The submissions page is similar to the views page, in that a table of pre-formed HTML from the bespoke model is displayed. Unlike the views, this is only visible to logged in and authorised users as referees, declared by the bespoke model. Any button elements were recursively modified to include an “on-click” event – that is, a functional event is raised as the user performs a click. The event handler function invoked the business model to send an MQTT message with which button was pressed with its internal metadata.



Upon a match closing, in the case of this fencing model, the input was replaced with the result scores, making the input no longer usable. Inputs are ordered alphabetically, and messages include the time in a format suitable for ordering, putting the newest inputs at the top.

One of the inputs was to submit a new match, with a time and which participants are available to compete. As bespoke model emits this, it is responsible for keeping this list up-to-date and declaring suitable form verification.





For all users, a page for viewing current subscriptions and their MQTT patterns was provided, including a facility for advanced users to subscribe to any event they wish. On the left, a subscription for the “Live scores” view has already been made, with the possibility to unsubscribe (also available on the views page), and a custom subscription to any events containing the participant name “Patrick Bowen”. It is possible for the bespoke model to provide personal views its developers find would be interesting to application users. But it could also be a standard facility of the general model if a schema was properly defined and adhered to.

Finally, as the artefact polls or loads any new data (such as initial competition schema) a progress bar is shown in the navigation bar. If the underlying HTTP request fails the whole bar is turned red for two seconds. This gives the user visual indication that the application is either working or has encountered a connectivity issue needing attention.

A bespoke model

A bespoke model is a concrete implementation of a competition's business logic. It is the sole driver for anything not related to user synchronisation, authentication, and brokering business logic for other competitions – the role of the general model. A bespoke model requires no other connection than to the general model, and using a flexible MQTT API can modify the model's schema in depth. Due to this method of API communication the bespoke model can be written in any network-enabled programming language/platform. Onwards "model" will mean "bespoke model".

Model life-time. A model has the ability to be ephemeral during competition play, meaning it could potentially leverage serverless, stateless hosting. It only needs to supply the MQTT API with its number of read events and input submissions to be updated, and can store practically any data in this transcript. For example, the fencing model stores current scores in the input fields, not locally. The only requirement is, upon initialisation, the API must be supplied a schema with basic details – such as competition titles, tags, descriptions, referee user ID's (if known), data views for the audience/participants, and inputs for referees. Multiple competitions can be initialised at once, as this data is supplied as a nested hash-map, with competition ID's as the top-level keys. A unique competition identifier will need to be supplied in order to not collide with other competitions within the general model.

Adding new views. Views are generally defined in the initial schema. In order to add new views a new partial schema must be supplied, and until a defined method of removing data from a schema is chosen for the artefact removing views can only be done through setting a view to contain no data.

Adding new inputs. Inputs are regularly added and replaced during a competition. In order to create or update an input, supply the API with a partial schema such as `{compId {:inputs {"topic" [:html]}}}`.

Processing submissions. Buttons with an input will be automatically set with an "on-click" function to create an MQTT event using the input topic, with the button `:value` key as the payload, and any form data as separate keys per element ID. A submission will be materialised as an MQTT event, of which your model logic should subscribe to.

Handling EDN and MQTT. The general model and front end use EDN to encode data, and the bespoke model is also required to use it. There are libraries in a number of popular programming languages to support this, though Clojure and ClojureScript have native support. It can be simple enough to create a recursive function which takes, for example, JSON and converts it to EDN. For MQTT writing a small topic matcher and broker is very helpful in routing the messages around the model. Here is an excerpt from the fencing model in NodeJS:

```
//Accepts a subscription criteria and a message topic
//  and returns boolean upon successful match
function topicMatch (criteria, topic) {
  var [criteria, topic] = [criteria, topic].map(t => t.split("/"));
  //Topic was longer than a close-ended criteria
  if (!criteria.slice(-1)[0][0] == "#" && topic.length > criteria.length)
    return false;
  for (let f = 0; f < criteria.length; ++f) {
    if (criteria[f] == "#") break;
    if (criteria[f] == "+") continue;
    if (criteria[f] != topic[f]) return false;
  }
  return true;
}

...

//Associate various callbacks to MQTT topics
//  as essentially subscriptions
const subs = {
  "+/fencing/score/#": handleScore,
  "+/fencing/new-match": handleNewMatch
};

...

//React to a new event polled from the server
//  by brokering the topic to a callback
function reactToEvent (topic, payload) {
  const callbacks = Object.keys(subs).filter(s => topicMatch(s, topic));
  callbacks.forEach(c => subs[c](topic, payload));
}
```

Emitting events. Events can be emitted at any time to the general model. A model may want to emit events only after another event has taken place such as a submission from a referee, or timed events (such as at the end of a match) can also be emitted. All MQTT messages are sent to the "/mqtt" API endpoint, and should return a 200 HTTP response.

Changing referees. An initial schema can define a set of referee ID's, or a partial schema may be sent at any time to authorise new referees. However, currently the API has no defined way of removing referees other than destroying the whole set and recreating it. And referees which are de-

authorised while signed in will not be forcibly ejected, though their submissions will not be committed to the event transcript.

Changing competition information. Just as with any schema changes, a partial schema map with the keys needing changed should be supplied to the API. Schemas are sent to the "/schema" endpoint, and will immediately propagate to the web-app's on their next poll request.

Handling multiple competitions. The competition ID is supplied alongside MQTT messages, rather than as part of the MQTT message (which it should be in the future). To handle multiple competitions MQTT messages and schemas should simply include the competition ID, and respect the incoming ID too.

Handling stalls. If a model stalls, it is able to start up and re-initialise the general model with the schema at last save – it is recommended to keep the schema in its own EDN file. If any events were left unprocessed, the general model resends them in an effort to address the deficiency in the number of events the model has. To ensure safe stall restarts, while the general model does not provide proper transactional updates, it will overwrite events with the same MQTT topic. So long as it is deterministic per event processed any previously emitted events from the model will be overwritten again.

The general model

The general model, unlike the front-end or bespoke model, requires little to no configuration beyond its hosting and security. It obtains its schema from bespoke models, and indiscriminately stores data in its database currently stored as "competite-DB.edn" in its working directory. In order to execute the model, either invoking Leiningen's run command or executing the deployed JAR file starts up its web-server. Its standard output is simply debugging information such as polling requests served with any new records. To configure HTTPS, provision of a valid Java certificate key-store file would be necessary, using a command-line tool such as EFF's *Let's Encrypt* certbot[27].

27 https://certbot.eff.org

Questionnaire and responses

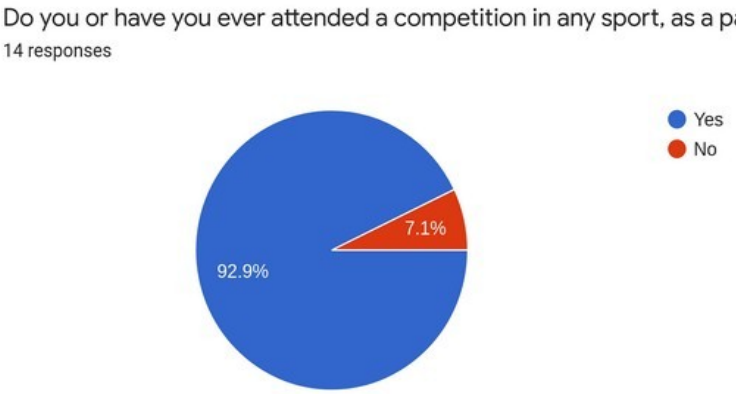
As part of the primary research for this report, a questionnaire was distributed online directed at sports' competition participants or regular audience members of competitive sport. The intent was to quantify the current importance of technology within competitions, and the perceived benefit of further integration. Participants were asked: to score the importance of technology addressing two issues, live score propagation and athlete live direction and information; to report their current use of technology within a competition, and how important technology was to the sport; provided with an example of this report's artefact, whether such a system would bring any perceived benefit; any further comments.

The BCS Code of Conduct, the Data Protection Act 2018, and Staffordshire University's Code of Practice for Research was met in the collection, retention, and after the destruction of questionnaire data. However, as the questionnaires distributed obtained no personally identifiable information, participants were made aware they were waiving their right to retract their answers after submission, as there would be no possible way of authenticating the individual to a response. It was made clear that any personally identifiable information included in their responses would result in the response being discarded. An alternative method would have been to allow the participant to include a passphrase which could be used to authenticate themselves with the response for removal. Instead, using the Google Forms platform, the ability for participants to modify their responses after submission was provided. This did not use Google as an authentication method, and was perceived to be reliant on the user's browser cookies. A grace period of one week was provided in good faith to any participants who may have wished to modify their answers.

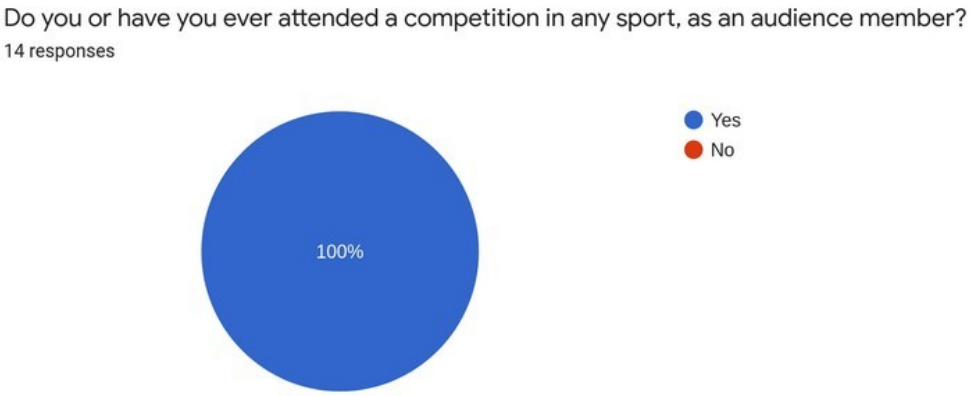
The questionnaire preamble was as follows.

> This questionnaire is collecting data for Patrick Bowen's Computing Science Bachelor's dissertation, titled "Mobile Competition Management with Live Acquisition, Reporting, and Notification". Please note any responses containing personal information will be discarded. If you have any questions or concerns please contact b016764f@student.staffs.ac.uk, or as ombudsman enquiries@staffs.ac.uk. Upon submission you waive any right to revoke your response. Though you are able to modify your response after submission, it may not be included.

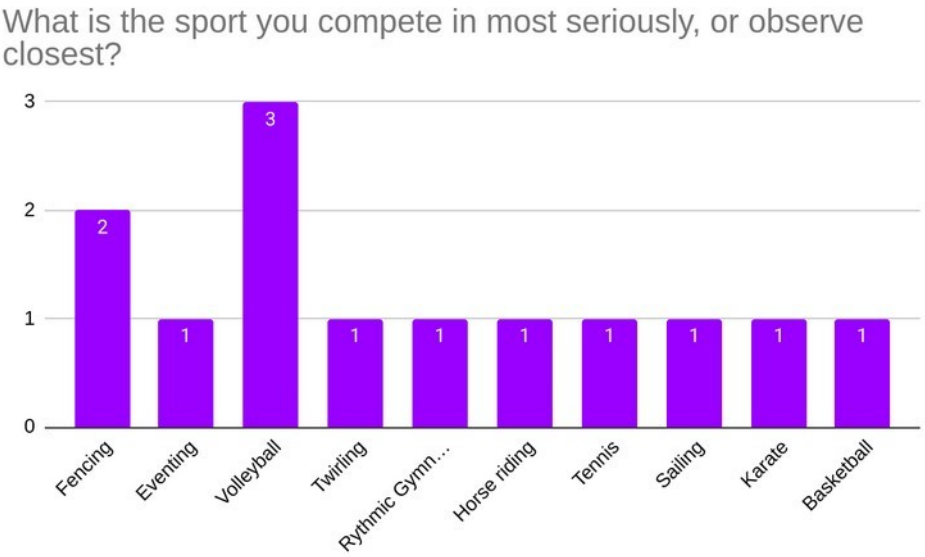
See *Appendix 3* for all individual responses. Following is a breakdown of each question and its responses.



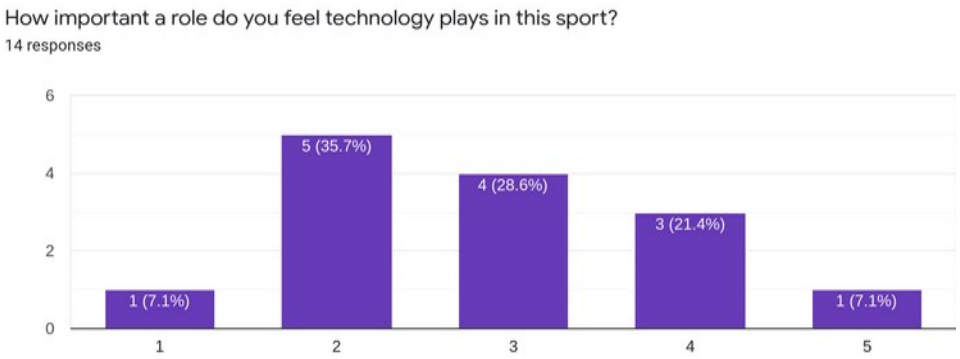
Only one questionnaire respondent had not competed in any sports competition (Volleyball)



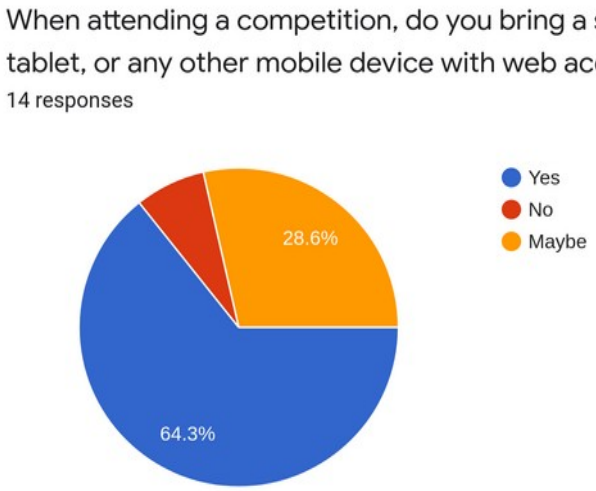
All questionnaire respondents had at least been an audience member of any sports competition.



Ten different sports were recorded, and one omission.

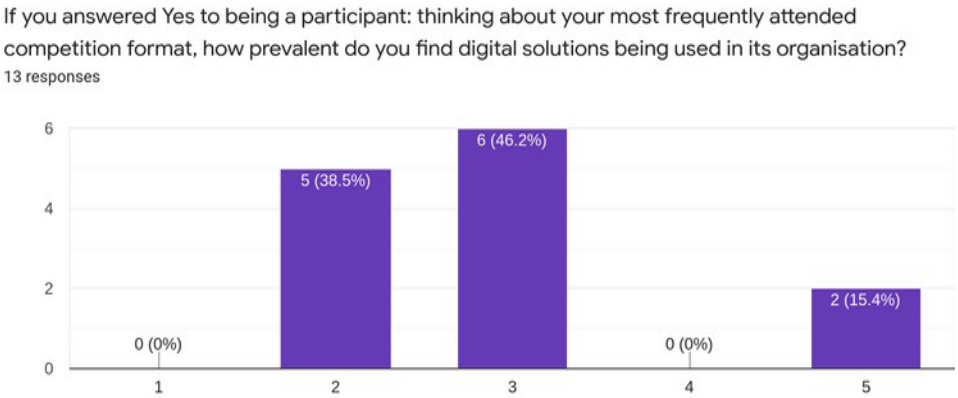


On average, technology was not recorded as being important in the respondents' competitions, though only one respondent reported there being no technology within their sport.

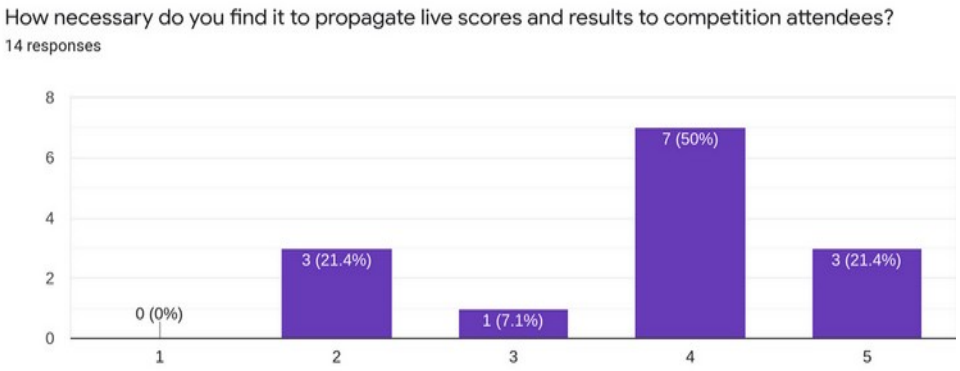


Only one respondent (Fencing) indicated they do not take any web-enabled device with them to competitions. After this point, participants were given some further information:

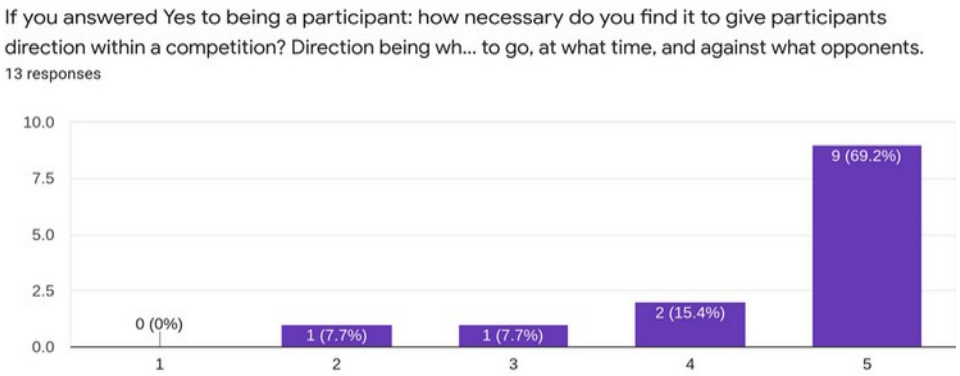
> When considering technology within a competition, think how information on-site is shared, how directions for participants are provided, how scores are propagated, and any physical or digital signage.



All respondents responded as there being any level of technology use in competition organisation, though it was reported low.



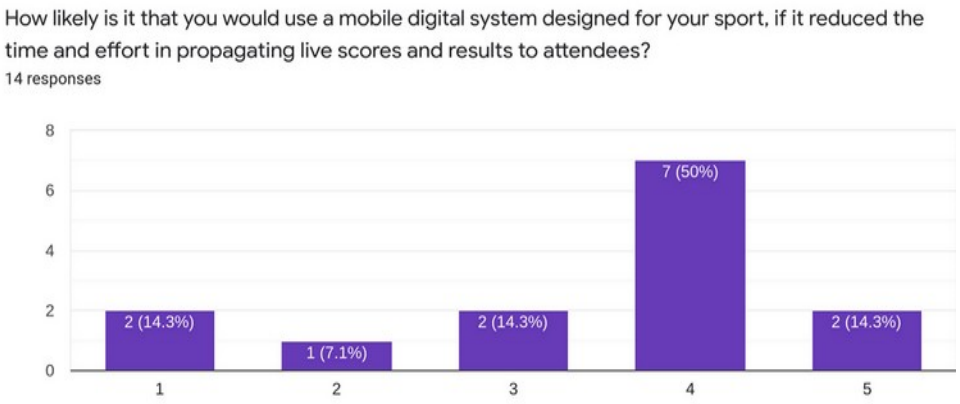
No respondents found it *unnecessary* to propagate live scores and results to attendees. On average it was regarded as an important aspect.



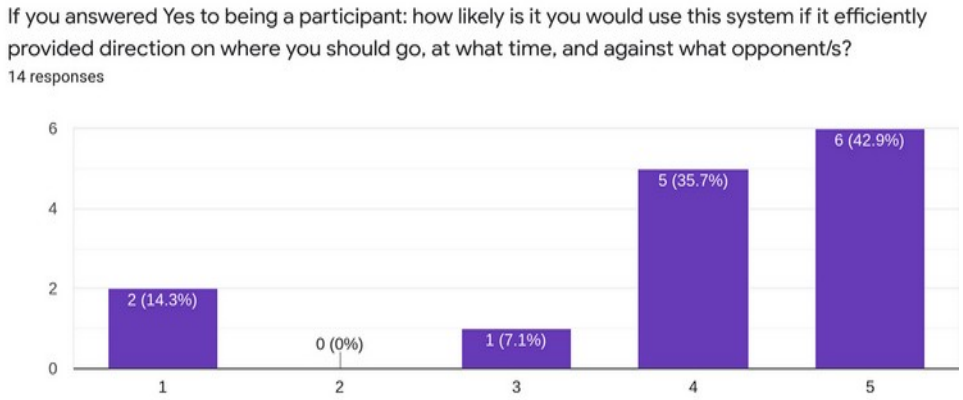
All respondents found direction within a competition of any importance, though this question would have been more suitable for this report if it asked about *live* directions, rather than directions and times which could be distributed before the competition begins. This question unfortunately should not be included in conclusion.

Before the next question, respondents were shown a screenshot of the Competite web-app, with this description:

> For reference, this is a screenshot of the prototype mobile app designed through my dissertation, to support multiple sports. It is designed not just for scores and directions, but propagating any information about a competition/event. It will be open-source at https://github.com/phunanon/competite



This question is one of two which are of great importance. Over half of respondents were likely to use such an application, on the premise that it reduced the time and effort for score propagation.

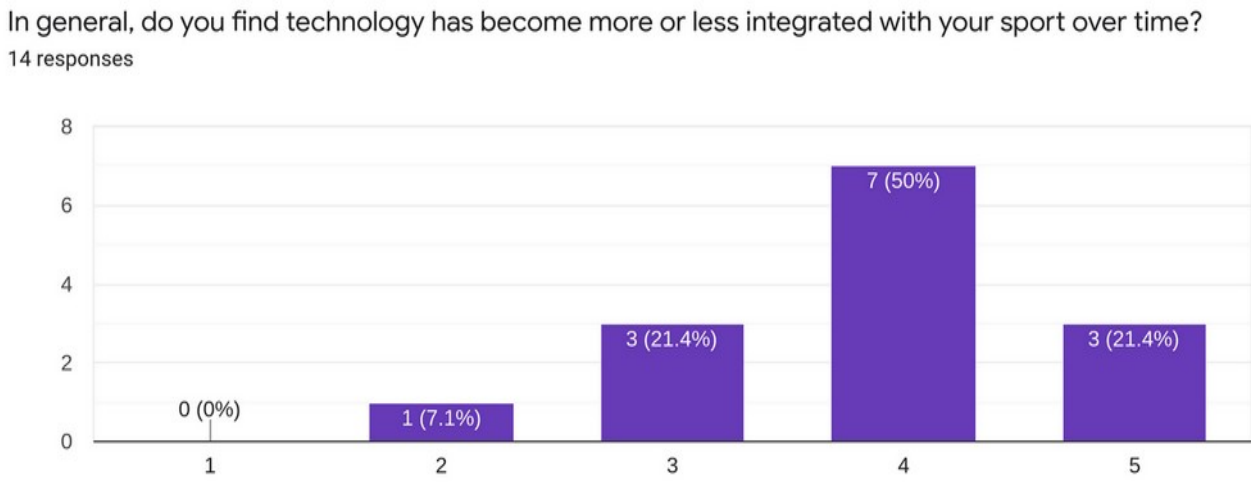


As with the previous question on direction, this one does not mention providing *live* directions and times. However, there is still strong consideration that such a system would be adopted regardless. This could be an indication of the existing methods of providing directions being inadequate, outdated, or not as convenient. This coupled with the previous responses on score propagation is a promising suggestion for the usefulness of an artefact such as the one featured in this report.

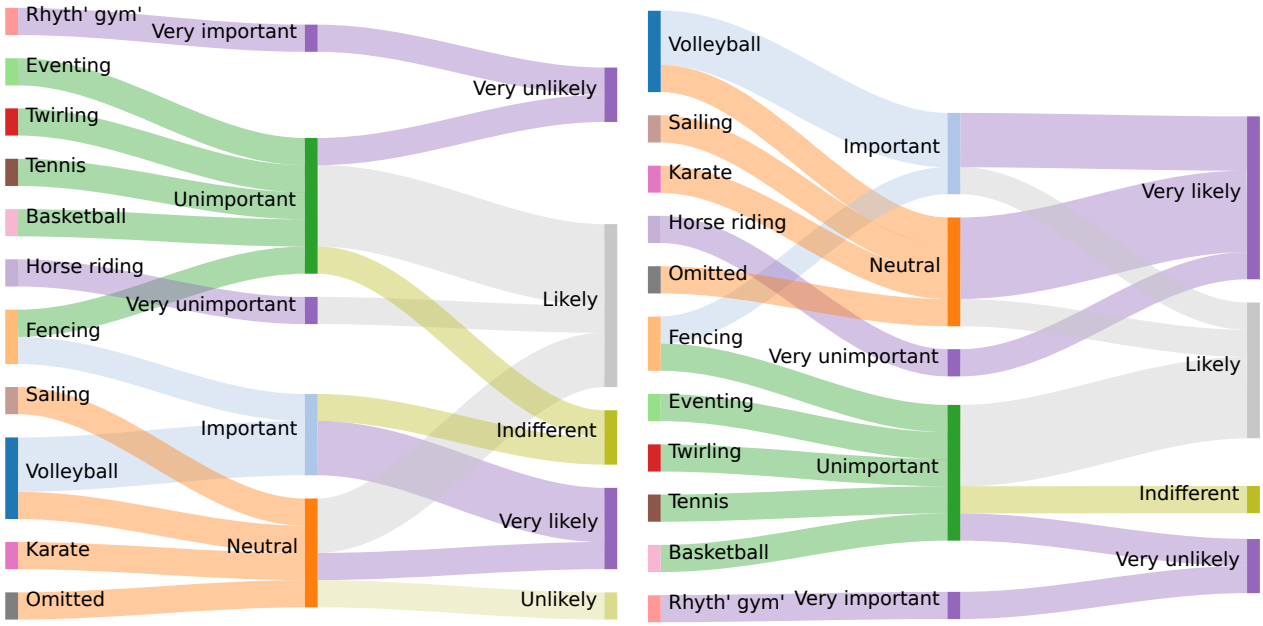
The next question accepted answers as a short free-form, asking “Other than providing the previously mentioned directions, what other direction/s do you feel could be provided by such a digital system, if any?” The eight responses are listed below:

- Who the referee is
- For volleyball an analysing system to find the weakest area of the opponent team
- Any official violations committed
- delays and advances during competitions
- music
- The digital system can show the participants’ information. (his/her age, successes at competitions, medals, etc.)
- Advanced reservations, such as hotels for out of state games.
- Maybe for participants explain the rules of the game and when there’s a penalty explain why is it for (I had to watch some judo competitions and I didn’t understand what was going on most of the time)

Most responses indicate that extra information pertaining to gameplay, such as athlete and referee information, violations, penalties, delays and particular rules would be of interest. All of these features would be incredibly undemanding for a bespoke model developer to include. One response mentions live analysis would be of interest, and this would be up to the bespoke model developer to integrate also. One response mentions information such as hotel reservations and locations, useful for the attendance and organisation of a competition outside of gameplay. This could be provided through the Markdown description already integrated as part of the artefact. However, for better support a HTML description could be enabled so bespoke models can include facilities such as Google Maps and telephone number hyperlinks. One response (Rhythmic Gymnastics) suggested the ability to play music, which has been mentioned in this report previously as possible due to the flexibility of including HTML in submissions – it would be the responsibility of the bespoke model developer.



On average respondents are seeing an increase in technology use within their sport’s competitions.



Here, two Sankey diagrams (using Sankeymatic[28]) are used to plot relationships between sport, importance, and likelihood. The first concerns the importance of live score propagation, and the subsequent likelihood of using an application providing this facility better. The second concerns instead direction propagation, and similar likelihood.

The first diagram shows no particular relationship between importance and likelihood, other than was already determined through gross analysis. Even when technology was perceived as very important (Rhythmic Gymnastics) it was very unlikely that they would use such an artefact; and as with technology perceived as very important it became likely that they would use such an artefact.

The second diagram shows similar results – Rhythmic Gymnastics still very unlikely, even with direction perceived as very important. However, unlike the particularly uncorrelated relationships concerning score propagation to adoption likelihood, direction propagation and adoption likelihood was predictable – besides Rhythmic Gymnastics.

In conclusion, the questionnaire could have been improved through isolating *how* technology was useful and offering alternative thoughts. Better yet would have been an interview, as these questions could have been challenged to ensure respondents fully understood the use of the questions within this report.

28 http://sankeymatic.com/build

Artefact testing

Most tests are performed and measured manually, but some use the Firefox Developer tools to carry out some tasks such as throttling connection speed.

#	Test description	Expected result	Actual result	✔
0	General model initialisation	The general model should start without any errors.	The general model starts and stays running without any errors.	✔
1	General model long run-time	The general model runs for at least an hour without exiting.	The general model stayed running for multiple hours before being terminated by the user.	✔
2	General model database persist	The general model should persist and restore any state into a database file.	The general model persisted state as soon as it was created, and restored it exactly on load.	✔
3	General model merge schema	The general model is able to deeply merge schema from bespoke models into its database.	The schemas are deeply merged as expected.	✔
4	General model safe schema merge	The general model is able to guard against malicious merges, such as a bespoke model overwriting the competition of another model.	The entire schema is mutable to all bespoke models, without guarding.	✘
5	General model emit schema	The general model will propagate schema changes to all web-apps.	Any changes to schema or the data within the database are propagated whole to web-apps.	✔
6	General model merge MQTT	The general model will accept MQTT and merge it into the events transcript.	MQTT messages are merged into the events transcript singularly and in bulk.	✔
7	General model serve token	A bespoke model can request (GET) from the general model a valid CSRF token which can be subsequently used.	A valid CSRF token is returned which can be used more than once. An invalid token rejects POST requests.	✔
8	General model accept token	The token provided by the general model is accepted upon a POST request.		
9	General model client sign-in	A web-app is able to authenticate itself through the general model, and access data only available to authorised users.	Signing in as a competition referee is successful. Inputs are otherwise not supplied.	✔
A	General model client sign-up	A web-app is able to register a new account which can be signed into after with the same credentials.	A new account is created and can be subsequently signed into.	✔

B	General model reject sign-in	A sign-in with incorrect credentials returns a status indicating the incorrect information.	A :bad-email status is sent when an email isn't found, and :bad-pass is sent with an incorrect password.	✔
C	General model serves poll request	Upon a polling request from a client the general model must fulfil the deficiency in number of events and inputs.	Upon polling the general model for an arbitrary number of events/ inputs it is fulfilled for as many new events/inputs there are.	✔
D	General model serves home	The general model should send the homepage Markdown document to the client.	The Markdown document is sent to the client in the first initialisation map to the client.	✔
E	Web-app shows home	The web-app should accept the Markdown home document from the general model, and parse it into styled HTML	Styled HTML corresponding to the Markdown is correctly rendered and shown.	✔
F	Web-app single-page	The web-app never fully refreshes as a page, no matter the navigation or action.	The web-app correctly captures all routing events and handles them, with no refreshes.	✔
G	Web-app view competition	The web-app should display a selected competition's metadata, such as its title, description, tags, and data views in tabular format. The tabular data should be chronological.	The selected competition's metadata is shown, but only the title, description, and tabular data views – not the tags. An easy fix. The tabular data is chronological.	✔ ✘
H	Web-app switch competition	Upon switching between competitions, this data should be relevant: competition info, data views, active subscriptions, menu notifications, future notifications.	Data instantly changes throughout the application, due to the reactive nature of the DOM.	✔
I	Web-app polling	The web-app should poll for new events and correctly merge them into its database.	Events are polled for every two seconds and merged correctly.	✔
J	Web-app hover notification	The web-app should display a hovering notification when new events occur.	A notification is shown, but only one even if more than one event came at once – notifications are lost.	✔ ✘
K	Web-app nav menu notification	The web-app should list all subscribed-to events in the menu chronologically, and this menu should update with new events.	The events are displayed and chronological.	✔
L	Web-app mobile UI	The navigation bar and tabular data within the web-app should fit reasonably on a mobile screen.	The navigation bar becomes a 'hamburger' menu and navigable, and most tables fit on the screen. However, some tables needed at least a scrollbar.	✔ ✘

M	Web-app shows sign-in	The web-app should inform the user that they are signed in.	The sign-in form is hidden and replaced with “Hello, [forename]”	✔
N	Web-app shows sign-out	The web-app should inform the user that they are signed out.	The sign-in form reappears and the interface changes according.	✔
O	Web-app shows name	The web-app should, while a user is signed-in, show their forename in the navigation bar.	A signed-in user’s name is shown in the navigation bar.	✔
P	Web-app subscription	A user should be able to click a button on a data view in the competition information page to subscribe to that particular data.	Clicking the “Subscribe” button on a data view subscribes the user to that data, as notifications.	✔
Q	Web-app unsubscription	A user should be able to undo a subscription to a data view by clicking the button again, or in the subscriptions page.	A subscription can be undone in both the subscriptions page and data view. Notifications are removed from the menu.	✔
R	Web-app custom subscription	A user should be able to create a custom subscription pattern with MQTT subscription syntax.	A user is able to create and remove custom subscriptions.	✔
S	Web-app wait progress bar	The web-app should show a progress bar while polling or loading other data is ongoing.	A progress bar shows for any polling or HTTP request, however, sometimes it is so fast it simply flickers up. This should be mitigated for a nicer experience.	✔
T	Web-app HTTP failure	The web-app should turn red to indicate a HTTP failure.	The app navigation bar turns red on HTTP failure.	✔
U	Web-app inputs	The web-app should show inputs only to a signed-in, authorised referee of a competition.	Inputs are shown to authorised referees only.	✔
V	Web-app input styling	The web-app should correctly show inputs conforming to the Hiccup-like style in the schema.	The HTML renders correctly.	✔
W	Web-app submission	The web-app should enable submission of input forms and buttons, correctly sending a POST request to the general model.	The buttons for submission are correctly injected with event handers, to extract local form data or send button values back to the general model.	✔
X	Web-app input update	The web-app should update inputs when the general model’s poll response changes them.	The inputs change, due to the reactive nature of the DOM.	✔
Y	Web-app double submissions	If a user submits more than once it should only be registered once.	As the inputs use a pre-determined MQTT topic to send the topic is de-duplicated within the general model, not the web-app.	---

Artefact reflections, comments, and conclusions

A reminder of the vision. The original vision called for a mobile app which could propagate live scores, updates, and directions to replace the non-digital method of information sharing through a sports competition. It needed to be accessible, suitable, and easier to use than a physical method of shuttling paper around or vocal transmission. It needed to support multiple sports, to substitute multiple different concrete implementations of competition business logic, to encourage separation of concerns. No assumption was made of the competition business logic, to the point that it could be an entirely different run-time situated on another machine. This vision was not altered during critical review, but the technology choices, architecture, and review itself were influenced by reality.

On architecture. It was expected to use three distinct components from the earliest speculation of its implementation. A mobile-friendly web front-end with server back-end, and a bespoke model which would connect to that back-end in an abstract way. However, instead of attempting to evaluate all sports' digital solutions, and estimate other additional characteristics which might need to be taken into account, the general model became more of a channel for web-interface. It handled the user connectivity, polling, and cached database results, but had a very loose schema for how data was displayed to the user. It was opted to use HTML directly from the bespoke models to enable maximum flexibility, and have models conform simply to: an immutable events transcript, and a mutable input list, per competition. This gave the dialogue between models and users output and input. So long as a bespoke model's interface could be expressed as free-form HTML, CSS and potentially client-side JavaScript, any competition format and model technology could use it.

On a wider purpose. The very early proposals for this dissertation were centred around the sport Fencing. It was to be a system designed only for that sport, with bespoke hardware and software solutions. However, upon simplification of the architecture which could be used, it was realised that multiple sports being supported would be much more flexible, and benefit all sports involved. Furthermore, once this had been achieved, it is revealed that the characteristics of the artefact could be applied to any output/input scenario, not just sports competitions, and not even live events. It could be suitable for abstracting away any kind of model with input/output which would benefit from such an architecture. A very simple example could be taking a register of present students – the input could be not from a referee but a teacher, and the output could be made private to teachers also. The model implementing this behaviour would not have to implement user authentication or authorisation, propagation of information to multiple devices, caching, or any persistence at all. All could be handled by the Competite artefact, as a generic model I/O framework – with rebranding.

State management and propagation. Where to keep state, and how to keep state synchronised, were a tough design decision. Several variables were at play – the weight and overhead of transmission, de-duplication of data, stall recovery, schema authority, transport protocol, frequency and immediacy of live data, and much more. To consolidate these variables, some were allowed to inflate more than desired; such as transport efficiency – using EDN for convenience and flexibility; schema authority being left entirely to the bespoke model within the scope of a competition; state being kept primarily in the general model and propagated both to the front-ends and bespoke models, rather than being propagated from the bespoke models only; a tolerable frequency and immediacy of live data by use of polling (though in the future could use modern standard streaming protocols). Simplifying processes where possible may have incurred some overheads and costs, but it made reasoning about the system, and component integration, much easier.

Corner-cutting. For this artefact, it has been treated as a prototype – some features which would be deployed into production have not been included or only partially supported. These are all non-critical characteristics or features which are not necessary to prove the hypothesis. This included aspects relating to interface design and user experience, increased lenience to data transport and schema lenience, choosing more appropriate libraries and frameworks, the bespoke model supporting more than one competition at a time, and even what would normally be critical security. This is permitted as, fundamentally, it is provable that addressing these issues is feasible and beneficial. The hypothesis simply gave a proof of concept.

Comments on technology choices. The technology used, particularly Clojure and ClojureScript, has been very comfortable to work with. Developing immutable pipelines of data, being clear where mutability is used, and leveraging a flexible and navigable native data notation, has helped artefact development tremendously. While there has been a lot of boilerplate code, and the need to study many separate documentations almost to a ‘philosophical’ detail, has taken time. But it is a stable, mature, and incredibly suitable technology stack for addressing this hypothesis. However, its current inability to easily be hosted in a serverless architecture is concerning and limiting. A production-level solution to the hypothesis would excel on ephemeral, scalable architecture, keeping costs low, reducing maintenances, and encouraging even further separation of concerns and more precisely orchestrated systems (as opposed to three monolithic models).

Report reflections, comments, and conclusions

Effectiveness of literature review. In order to bolster the claims of the original report proposal, and the technologies being chosen, a strong literature review was required. However, there was a difficulty toward the hypothesis: there were no academic citations for the attitudes of competition goers with digital solutions. Books around sport and media in general were valuable, however. There was also only very little grey literature around fencing as a sport in general and those attitudes. Yet, after shifting the perspective of some academic searches it was possible to find the correct area of research, and follow their citations for much better clarification. And while it was difficult to find information on existing digital solutions within sport, enough was discovered to make educated assumptions of user preference. The literature review was a critically important stage both for the validity of this report and carefully aiming the artefact's development.

On primary research. Originally, the primary research of this report was to be carried out in two rounds: initial interviews before the artefact's development, and a demonstration after, which would be responded with by improving the artefact. Unfortunately time and the COVID-19 pandemic caused the demonstration phase to be cancelled, and by this point the research on existing software had been completed. There was enough confidence in developing the artefact by this point, extrapolating favoured features from the existing software, that a new mode of primary research was chosen. Instead, one round of confirmation at the end of artefact development was conducted, including a screenshot and description of the completed artefact as context for the questionnaire. This was able to give valuable attitudes toward digital solutions within sports, with evidence strengthening the original hypothesis as important.

On project management. The work packages materialised at the beginning of the project were helpful in planning what work was able to be conducted and when. Creating a concrete daily or weekly plan would have been difficult due to circumstances outside of the project, so not pinning tasks to arbitrary dates was tactful. The chronology of the work packages was found to be very well planned, with only several minor diversions. The first stage of primary research was ignored entirely, however, but the second stage was adhered to.

On software development methodology. The software methodology chosen – *Sashimi* – was conducive to the artefact's development. Taking the time to fully develop a plan before implementation begun resulted in a very robust architecture. *Sashimi* also helped ensure that throughout development further research was still a valid and supported action to take. An

incremental process to development, with expectations as frequent tests, helped steer feature implementations.

Addressing risks. Risks were able to be properly addressed in all cases, such as physical and mental health, work loss, and a heightened awareness of possible diverse threats to the project's development. No risks came true but one, where some data was lost while writing a table directly into the report and not somewhere outside of the report document before its inclusion. For more important report inclusions this should be carried out, if not for all sections of a report.

Further work. Certainly changing the mode of hosting the application is a key priority, as serverless architecture suits its workload best. Proper demonstrations of the artefact to potential users is also critical in forming a more realistic set of features. Evaluating the artefact's use in other mobile or live input-output situations could reveal more uses outside of sport and competitions. Developing more bespoke models around other competition formats, and finding standardisation between them to orchestrate together would help drive the original hypothesis further. Developing a template or library to interact with the API in certain programming languages could also be possible and helpful, though a full API re-write should be considered.

Final conclusion. This project has been successful in proving its hypothesis through its original premises. The artefact has been fully actualised to a usable level and would only require some implementation adjustments to bring it to production level. Primary research has revealed a positive attitude toward using such technology for live score propagation and particularly live direction propagation. The technologies chosen were suitable for PaaS deployment, though FaaS deployment is ideal – can be used for bespoke models. The architecture was sound and though the API would benefit from re-writing the beneficial characteristics have proven important to emulate.

Appendices

Appendix 1 – Estimated work packages

Focus		Description	Resource Requirements	Estimated time
Documentation		Prepare document for the project.	None.	0.25 hours
Planning		Risk assessment. Ensuring any risks to the project are effectively controlled.	None.	0.5 hours
Solution research and deliberation		Research and evaluation of existing solutions.	Internet for research.	4 hours
		Solution choice and defence.	Above research.	1.5 hours
		Define objectives.	Solution chosen.	3 hour
Primary Research – initial interviews		Define targeted and potential audiences.	Research completed.	0.75 hours
		Define useful questions.	Objectives defined, potential audience defined.	0.5 hours
		Write the questions.	Questions defined.	1 hour
		Conduct interviews.	Questions written.	1.5 hours
		Evaluate answers, including audience suitability.	Interviews conducted.	2 hours
		Redefine objectives.	Answers evaluated.	1 hour
Develop artefact	**Schema & mocks**	Develop schema between general and bespoke models.	Solution deliberated and latest objectives created.	1 hour
		Develop a mock browser-client front-end.		3 hours
		Configure new project.	Internet for downloads and tutorials.	0.5 hours

Focus		Description	Resource Requirements	Estimated time
	General model	Develop MQTT messaging core.	Project configured, schema developed.	1 hour
		Develop networking core.	Project configured.	1 hour
		Implement authorisation and authentication.	Messaging core developed.	1 hour
		Implement schema capture and persistence.	Schema and messaging developed.	1 hour
		Implement data view requests and fulfilments.	Schema and persistence, messaging developed.	2 hours
	Mobile app	Configure new project.	Internet for downloads and tutorials.	0.5 hours
		Implement real networking logic connecting to back-end.	Project configured, general model networking developed.	1 hour
		Implement real output logic.	Networking logic, general model messaging developed.	1 hour
		Implement real data view logic.	Networking logic, general model messaging and schema persistence developed.	3 hours
	Bespoke model – fencing	Configure new project.	Internet for downloads and tutorials.	0.5 hours
		Develop and configure database core.	Project configured.	2 hours
		Implement differential patch abilities.	Database core developed.	1 hour
		Implement schema emit.	Project configured.	1 hour
		Implement fencing competition logic.	Project configured.	3 hours
		Implement networking.	Project configured.	1 hour
		Join modules together and test.	All previous stages complete.	2 hours
Artefact testing		Test general model.	General model complete.	1 hour
		Test bespoke model.	Bespoke model complete.	1 hour

Focus	Description	Resource Requirements	Estimated time
	Test mobile app.	Mobile app complete.	1 hour
	Integrated test – all components, whole artefact. Black-box and white-box, make necessary fixes.	All components complete.	2 hours
Compose artefact documentation	For users.	Artefact complete.	1 hour
	For developers, to extend the artefact.	Artefact complete.	1.5 hours
Primary research – demonstration and questionnaires	Redefine audience.	Artefact complete.	0.5 hours
	Define useful questions.	Artefact completed, potential audience defined.	0.5 hours
	Write the questions.	Questions defined.	0.5 hours
	Deliver questionnaires, help if requested.	Questions written.	1 hour
	Evaluate answers.	Most questionnaires completed.	2 hours
Improve artefact	Translate evaluated answers into action items.	Question evaluation complete.	2 hours
	Enact action items.	As action items become available.	2 hours
	Test improvements.	As action items are completed.	1 hour
	Evaluate actions undertaken.	All action items completed.	1 hour
Final evaluation	Write final, critical evaluation, addressing what has been achieved, what should have been achieved, what went well/not so well.	Completed everything.	4 hours
Proof-reading	Simply proof-read the project, and ensure consistent styles throughout.	Completed project.	1 hour

Appendix 2 – Original project proposal

A satisfactory artefact must have connectivity through a functional web-app, connected at least to another machine on a network; with CRUD for scores, participants, and results; automated results based on inputted scores; a business logic which can be substituted at least upon artefact deployment, if not artefact start-up.

I intend to investigate and defend between Clojure (JVM) and C# (.NET Core) for the back-end., modern JavaScript et al and ClojureScript for the mobile web-app, relational and non-relational database technology.

It's likely the communication will be MQTT over HTTPS, and a RESTful approach for separation of concerns. This will be defended.

The primary challenge will be researching and designing a flexible enough system, especially with customisable jargon between different sport.

Competition managers must have roles they can configure and apply for referees (authoritative scores), participants, audience, and strangers. This must be done through the web-app.

As my requirements will be well defined before I begin implementing the artefact I intend to initially use the Software Development Life-Cycle methodology. From the first artefact implementation I intend to use Kanban or similar methodology – with boards of work to be done – as I respond well to this approach. Regardless of intention I will investigate the best methodologies before starting.

The tools I look to involve are Visual Studio Code or Leiningen and a text editor depending on the programming language; an SQL administration panel or a H2 database manager depending on the framework, or some sort of NoSQL manager if non-relational; LibreOffice for writing the project; Firefox on Android for testing the web-app.

For testing I will either use Visual Studio Code testing tools or Leiningen test cases depending on programming language, supplemented with extensive manual black-box testing. Overall I expect to use a regression strategy, developing against the tests until the majority pass.

The artefact should be deployed as at least two separate pre-compiled executables. One implementing the business logic and the other implementing the web-app. back-end and data model persistence.

I intend to report on my investigation of technologies, defence of one particular technology per application; progress on the development of the artefact, and its testing throughout; one to two rounds of feedback on the artefact in operation; meetings and steering decisions throughout the project.

Further, I intend for all project-related activities to be recorded accurately and critiqued where appropriate. I will use appropriate tools to record the various aspects, such as time planning tools for following objectives, test reports for following artefact completion.

Success will be determined through interview and questionnaire feedback from both technical and non-technical users, how well the artefact solves the spirit and the letter of my original proposal, how well the project adapted to any problems which arose during development, and through a personal reflection.

I intend to undertake all project activities in accordance with the university’s and BCS’ code of conducts, and in accordance with Department of Computing’s ethical policy.

Appendix 3 – Questionnaire responses

All April-May 2020.

(1) Do you or have you ever attended a competition in any sport, as a participant?
(2) Do you or have you ever attended a competition in any sport, as an audience member?
(3) What is the sport you compete in most seriously, or observe closest?
(4) How important a role do you feel technology plays in this sport?
(5) When attending a competition, do you bring a smartphone, tablet, or any other mobile device with web access?
(6) If you answered Yes to being a participant: thinking about your most frequently attended competition format, how prevalent do you find digital solutions being used in its organisation?
(7) How necessary do you find it to propagate live scores and results to competition attendees?
(8) If you answered Yes to being a participant: how necessary do you find it to give participants direction within a competition? Direction being where to go, at what time, and against what opponents.
(9) How likely is it that you would use a mobile digital system designed for your sport, if it reduced the time and effort in propagating live scores and results to attendees?
(10) If you answered Yes to being a participant: how likely is it you would use this system if it efficiently provided direction on where you should go, at what time, and against what opponent/s?
(11) Other than providing the previously mentioned directions, what other direction/s do you feel could be provided by such a digital system, if any?
(12) In general, do you find technology has become more or less integrated with your sport over time?

Date	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
30th	Yes	Yes	Fencing	4	Yes	3	4	5	5	5	Who the referee is	4
30th	Yes	Yes	Eventing	2	Yes	2	4	4	1	1		4
30th	Yes	Yes	Volleyball	4	Yes	5	5	4	3	4	For volleyball an analysing system to find the weakest area of the opponent team	5
30th	No	Yes	Volleyball	3	Yes		4		4	5	Any official violations committed	3
30th	Yes	Yes	Twirling	2	Maybe	2	4	5	3	4	delays and advances during competitions	4

Date	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
30th	Yes	Yes	Rhythmic Gymnastic	5	Maybe	3	2	2	1	1	music	4
30th	Yes	Yes	Horse riding	1	Yes	2	2	5	4	5		4
30th	Yes	Yes	Fencing	2	No	2	4	3	4	3		3
30th	Yes	Yes		3	Maybe	3	5	5	2	4	The digital system can show the participants' information. (his/her age, successes at competitions, medals, etc.)	4
1st	Yes	Yes	Tennis	2	Yes	2	4	5	4	4		2
1st	Yes	Yes	Sailing	3	Yes	3	4	5	5	5		5
2nd	Yes	Yes	Karate	3	Yes	3	4	5	4	5		5
2nd	Yes	Yes	Basketball	2	Maybe	3	2	5	4	4	Advanced reservations, such as hotels for out of state games.	3
3rd	Yes	Yes	Volleyball	4	Yes	5	5	5	5	5	Maybe for participants explain the rules of the game and when there's a penalty explain why is it for	4

Appendix 4 – Logbook of meetings

10th of October, 2019, Patrick Bowen & Chathurika Goonawardane, in person. • Completed resources ◦ Project proposal • Agenda ◦ Project idea discussion ◦ Project proposal discussion ◦ Ethics form fast-track • Action items ◦ Project proposal upload ◦ Ethics form upload ◦ Secondary search ◦ Define the objectives	**21st of October, 2019**, Patrick Bowen & Chathurika Goonawardane, in person. • Completed resources ◦ Proposal form upload ◦ Ethical form upload • Agenda ◦ Citation of personal experience ◦ Word count for different sections ◦ If the proposed sections are sane • Action items ◦ Upload the health & safety form ◦ Gantt chart
5th of September, 2019, Patrick Bowen & Chathurika Goonawardane, in person. • Completed resources ◦ Gantt chart – work packages • Agenda ◦ Logbook discussion ◦ Project/software methodologies • Action items ◦ Formalised use case/user stories ◦ Initial logbook	**11th of February, 2020**, Patrick Bowen & Adam Jacobs, in person. • Completed resources ◦ Detailed requirements and implementation projection • Agenda ◦ Mid-point review • Action items ◦ Literature review ◦ Dated work plan ◦ Design evaluation ◦ Survey
11th March, 2020, Patrick Bowen & Chathurika Goonawardane, in person. • Completed resources ◦ Work plan ◦ Started literature review • Agenda ◦ Selection of tools ◦ Selection of technology • Action items ◦ Reconsider risk assessment form	**29th of April, 2020**, Patrick Bowen & Chathurika Goonawardane, remote. • Completed resources ◦ Literature review ◦ Technology comparison • Agenda ◦ Current progress ◦ Artefact inclusion within report ◦ Inclusion of only one design • Action items ◦ Survey ◦ Include artefact design ◦ Contact 2nd Assessor for viva

5th of May, 2020, Patrick Bowen & Chathurika Goonawardane, remote. • Completed resources ◦ Artefact ◦ Surveys ◦ Most artefact evaluation ◦ Risk assessment • Agenda ◦ What might be missing from the submission • Action items ◦ Bolster design explanation ◦ Write about project management ◦ Finish report	**11th of May, 2020**, Patrick Bowen & Chathurika Goonawardane, remote. • Completed resources ◦ Artefact testing ◦ Artefact conclusion ◦ Bolstered design explanation ◦ Wrote a bit about project management • Agenda ◦ Anything missing from submission ◦ Report style ◦ Rewriting the risk assessment • Action items ◦ Add project threats to risk assessment ◦ Write the final conclusion ◦ Write abstract ◦ Submit

Bibliography

Pólya, G., 1945. *How To Solve It*. 1st ed. Princeton, N.J.: Princeton University Press, p.121.

L. Carvajal, A. M. Moreno, M. Sánchez-Segura and A. Seffah, "Usability through Software Design," in *IEEE Transactions on Software Engineering*, vol. 39, no. 11, pp. 1582-1596, Nov. 2013, doi: 10.1109/TSE.2013.29.

Nokes, S. and Newton, R., 2007. *Definitive Guide To Project Management*. Harlow: Prentice Hall Business, p.53.

Royce, W. 1990 *Trw's ada process model for incremental development of large software systems*. 12th International Conference on Software Engineering (ICSE '90), pages 2–11.

Hirotaka, T., and Nonaka, I. *The new new product development game: Stop running the relay race and take up rugby*. Harvard Business Review, :137–146, Jan.-Feb. 1986.

Kapela, R., Świetlicka, A., Rybarczyk, A., Kolanowski, K. and O'Connor, N., 2015. Real-time event classification in field sport videos. *Signal Processing: Image Communication*, [online] 35(July 2015), pp.35-45. Available at: <https://www.sciencedirect.com/science/article/abs/pii/S0923596515000661> [Accessed 2020].

Hutchins, B., Rowe, D., 2012. *Sport Beyond Television*. Taylor & Francis.

Willis, J., 2016. *Running A Fencing Competition*. [online] Leonpaul.com. Available at: <https://www.leonpaul.com/blog/running-a-fencing-competition-jon-willis/> [Accessed 2020].

Roberts, P. F. (2006) 'Building Smarter Authentication. (cover story)', *InfoWorld*, 28(30), pp. 26–32. (Accessed: 24 February 2020).

Vijayan, J., 2010. *User Authentication No Longer Thwarts Online Bank Thieves*. Computerworld, 44(2), p. 10. (Accessed: 24 February 2020).

Li, C., 2013. *A new password authentication and user anonymity scheme based on elliptic curve cryptography and smart card.* IET Information Security, vol. 7, no. 1, pp. 3-10, March 2013. doi: 10.1049/iet-ifs.2012.0058

Chipperfield, C. and Furnell, S., 2010. *From security policy to practice: Sending the right messages*. Computer Fraud & Security, 2010(3), pp.13-19.

2020. *Smartphone Internet Use By Demographic Great Britain 2019*. [online] Available at: <https://www.statista.com/statistics/275985/mobile-internet-penetration-in-great-britain-by-age-and-gender/> [Accessed 2020].

Vetter, J., 2007. *Workshop On Software Development Tools For Petascale Computing*. [online] Osti.gov. Available at: <https://www.osti.gov/biblio/1367254> [Accessed 2020].

Nørmark, K., 2011. *Overview Of The Four Main Programming Paradigms*. [online] Available at: <http://people.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigm-overview-section.html> [Accessed 2020].

Kowalski, T. and Adamus, R., 2017. Optimisation of language-integrated queries by query unnesting. *Computer Languages, Systems & Structures*, 47, pp.131-150.

Popplestone, R., 1999. *Lecture 1: What Are Programming Language Paradigms?*. [online] Cs.bham.ac.uk. Available at: <https://www.cs.bham.ac.uk/research/projects/poplog/paradigms_lectures/lecture1.html> [Accessed February 2020].

L'orange, J., 2013. *Understanding Clojure's Persistent Vectors, Pt. 1*. [online] Hypirion.com. Available at: <https://hypirion.com/musings/understanding-persistent-vector-pt-1> [Accessed 10 February 2020].

Palattella, M., Soua, R., Stemper, A. and Engel, T., 2019. Aggregation of MQTT Topics over Integrated Satellite-Terrestrial Networks. *ACM SIGMETRICS Performance Evaluation Review*, 46(3), pp.96-97.

Pratley, T., 2017. *Reagent Deep Dive Part 1: Hiccup And Ratoms*. [online] Timothypratley.blogspot.com. Available at: <https://timothypratley.blogspot.com/2017/01/rea

gent-deep-dive-part-1.html> [Accessed 10 February 2020].

Bowen, 2019, Determining serverless architecture as a substitute to traditional cloud computing. Staffordshire University (unpublished).

Emerick, C., Carper, B. and Grand, C., 2012. *Clojure Programming*. Sebastopol, Calif: O'Reilly.

Ellingwood, J., 2014. *How To Use Npm To Manage Node.Js Packages On A Linux Server*. [online] Available at: <https://www.digitalocean.com/community/tutorials/how-to-use-npm-to-manage-node-js-packages-on-a-linux-server> [Accessed 2020].

Clojure.org. n.d. *Clojure - Programming At The REPL: Introduction*. [online] Available at: <https://clojure.org/guides/repl/introduction> [Accessed 2020].

Okasaki, C., 1998. *Purely Functional Data Structures*. Cambridge: Cambridge University Press.

Mitchell, R, 1990. *Managing Complexity In Software Engineering*. Institution of Engineering and Technology - IET.

Kaasinen, J., 2019. *Pitfalls And Bumps In Clojure's Extensible Data Notation (EDN)*. [online] Nitor.com. Available at: <https://www.nitor.com/en/news-and-blogs/pitfalls-and-bumps-clojures-extensible-data-notation-edn> [Accessed April 2020].

Hickey, R., 2014. *EDN*. [online] GitHub. Available at: <https://github.com/edn-format/edn> [Accessed April 2020].

Huizinga, D., Kolawa, A., 2007. *Automated Defect Prevention: Best Practices In Software Management*. Wiley-IEEE Computer Society Pr.

Gebai, M., Dagenais, R., 2018. *Survey and Analysis of Kernel and Userspace Tracers on Linux: Design, Implementation, and Overhead*. ACM Comput. Surv. 51, 2, Article 26.

Fehling, Leymann, Retter, Schupeck, Arbitter, Cloud Computing Patterns. 2014. *Exactly-Once Delivery*. [online] Available at: <https://www.cloudcomputingpatterns.org/exactly_once_delivery/> [Accessed April 2020].

P. Louridas, 2006. *Version control*. IEEE Software, vol. 23, no. 1, pp. 104-107, Jan.-Feb. 2006.

Normand, E., 2019. *Why Clojure Starts Up Slowly — Is It Really The JVM?*. [online] PurelyFunctional.tv. Available at: <https://purelyfunctional.tv/article/the-legend-of-long-jvm-startup-times/> [Accessed April 2020].

Taft, D., 2020. *Oracle's Graalvm Finds Its Place In Java App Ecosystem*. [online] TheServerSide.com. Available at: <https://www.theserverside.com/news/252482598/Oracles-GraalVM-finds-its-place-in-Java-app-ecosystem> [Accessed May 2020].

Taft, D., 2020. *Oracle's Graalvm Finds Its Place In Java App Ecosystem*. [online] TheServerSide.com. Available at: <https://www.theserverside.com/news/252482598/Oracles-GraalVM-finds-its-place-in-Java-app-ecosystem> [Accessed May 2020].

UCL Department of Geography. 2020. *Dissertation Risk Assessment For Computer Based Projects*. [online] Available at: <https://www.geog.ucl.ac.uk/resources/safety/risk-assessment/forms-and-files/dissertation-risk-assessment-for-computer-based-projects/view> [Accessed 2019].

Mar, A., 2020. *130 Project Risks (List)*. [online] Simplicable. Available at: <https://management.simplicable.com/management/new/130-project-risks> [Accessed 2019].

Norris, J.M., Plonsky, L., Ross, S.J. and Schoonen, R. (2015), *Guidelines for Reporting Quantitative Methods and Results in Primary Research.* Language Learning, 65: 470-476. doi:10.1111/lang.12104

Clarke, J., 2009. *SQL Injection Attacks And Defense*. Burlington, MA: Syngress Pub., p.368.