Reimagining Joust
Clemens Koza, Lisamarie Schuster
Practical Robotics Institute Austria – {koza,schuster}@pria.at

Reimagining Joust

1. Introduction

An essential part of every Botball tournament is the handling of games and scores. The tournament modus of Botball produces a lot of numbers, such as Seeding, Period- and Onsite documentation scores. The Double Elimination tournament modus in itself uses some rather intricate logic, and all these results are then compiled into three main scores to determine awards.

The software that currently handles these tasks is "Joust", a PHP web application. As it is used at different tournaments throughout the Botball season, it has to be easy to set up and easy to use. Yet there are other concerns for Joust as well: eventualities that aren't handled by Joust[1] can't easily be corrected manually – interfering with Joust's logic might lead to inconsistencies. Furthermore, multiple tournaments at the same time, or other kinds of tournaments – such as the PRIA or KIPR Aerial competition, the PRIA Underwater competition, or Botball Alliance tournaments – are not handled by Joust. This leads to a concrete need for two generally desirable software properties in Joust: extensibility and maintainability.

The authors find that a re-imagined, "New Joust" would benefit from going away from PHP, and present the current stage of development in that direction, as well as first experiences from usage during ECER 2016.

Where it helps clarity, the terms "Original Joust" and "New Joust" will be used to refer to the two considered applications. Both are formally called "Joust".

2. Technology choices

Although it doesn't tell a lot about application quality, the technology stack is an important factor in making software future-proof. Outdated technologies can contain known security vulnerabilities, have incompatibilities with more recent software, or hinder maintenance as the number of developers knowledgeable in the technology shrinks.

1 For example, there is no tie breaker for seeding ranks. That means that multiple teams can have the same seeding score, which is OK, but it also means that the way in which these teams are assigned to DE matches is arbitrary. This can be settled by defining the way Joust handles the situation as correct, but a more transparent approach is generally preferable.

2.1. Original Joust

Joust as it currently stands is a PHP 5 based web application. As such, Joust's user interface can be used from almost any device. Only a computer capable of running the web server is necessary.

The installation instructions [1] suggest using the XAMPP [2] server package. The PHP backend is based on Zend framework 1.7.4, which was released in February, 2009; the current version is 2.4.9 [3]. Separation of concerns is achieved by using an MVC architecture [4]. Data is stored in an SQLite [5] database and accessed using PDO [6].

The frontend is designed using plain HTML, CSS, and some inline JavaScript; no frontend frameworks are used. This is in line with the KISS principle, but presents a barrier for adding more sophisticated frontend features.

2.2. New Joust

The "New Joust" is a Python 3 web application based on the Django 1.8.13 web framework; the latest release is 1.9.7 [7]. Django applications can be run using the bundled development server, which should be sufficient for Botball tournaments, or deployed to a web server or the cloud.

The Django framework also uses a variant of the MVC architecture [8]. As with Original Joust, data is stored in an SQLite database, but accessed via Django's own ORM[2]. Apart from serving HTML pages, New Joust also provides a REST[3] interface to be accessed by JavaScript.

New Joust uses Bootstrap 3.3.6 [9], AngularJS 1.5.3 [10], and jQuery 2.2.2 [11] in the front-end. Backend and frontend dependecies are managed by the pip [12] and bower [13] package managers, respectively, making it easy to update dependencies over time.

3. Feature overview

The following section compares some features that are present in both versions of Joust, as well as highlights features that are exclusive to one version. As will become apparent, some features that are essential for broad usage are still missing from New Joust, while other features are new or present an improvement over the current solution. The authors hope that, building on a solid technological foundation, the missing features can be implemented quickly and easily.

3.1. Installation and usage instructions

Although not a software feature, installation and usage instructions are essential for operating the software without supervision. Original Joust comes with a three page manual consisting of installation, setup and usage instruction [1], which doesn't exist for New Joust.

2 Object-Relational Mapper
3 Representational State Transfer

3.2. Authentication and the administrator interface

The first feature exclusive to New Joust is authentication and authorization. Django comes with a built-in administration interface that allows to edit the application's models, including users and roles. Making parts of the application exclusive to authorized users is easy. Figure 1 shows the admin interface overview, including authentication and and authorization models and Joust-specific models.

According to the Joust manual, to access the scoring interface, knowing the server's IP address is sufficient [1].

While we expect tournament participants to be honest and value fairness, it is easier to justify tournament results if they are secured from manipulation. Furthermore, the European region consists of many technical high school students who are very sensitive to IT security and would dispute results provided by a provably insecure system.

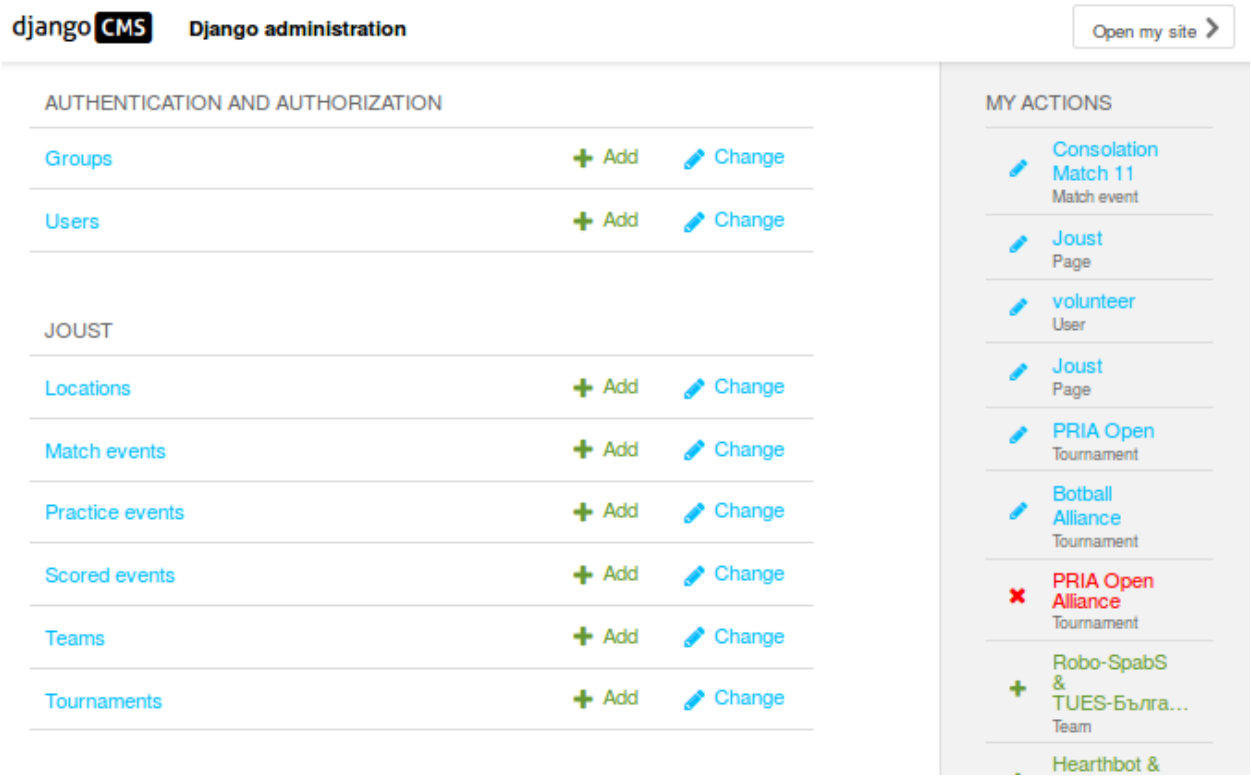

Figure 1: The Django admin interface (filtered), showing user & group management, Joust models, as well as past admin actions

3.3. Tournament setup

Setting up a tournament is very different between old and new Joust, as Original Joust operates on exactly one tournament, while New Joust manages multiple tournaments at the same event.

In Original Joust, the tournament is configured by setting the region and number of Botball tables in some PHP files, and then uploading a CSV file of teams [1].

In New Joust, there is not yet an import feature. Instead, teams must be added to the system individually, which is clearly not optimal. As can be seen in Figure 1, in addition to teams, there can be multiple tournaments and "locations" (e.g. game tables, Aerial arenas, Onsite Presentation rooms, etc.). Teams can compete in multiple tournaments, and their appointments (e.g. Open Practice slots, DE matches, etc.) are later assigned to appropriate locations.

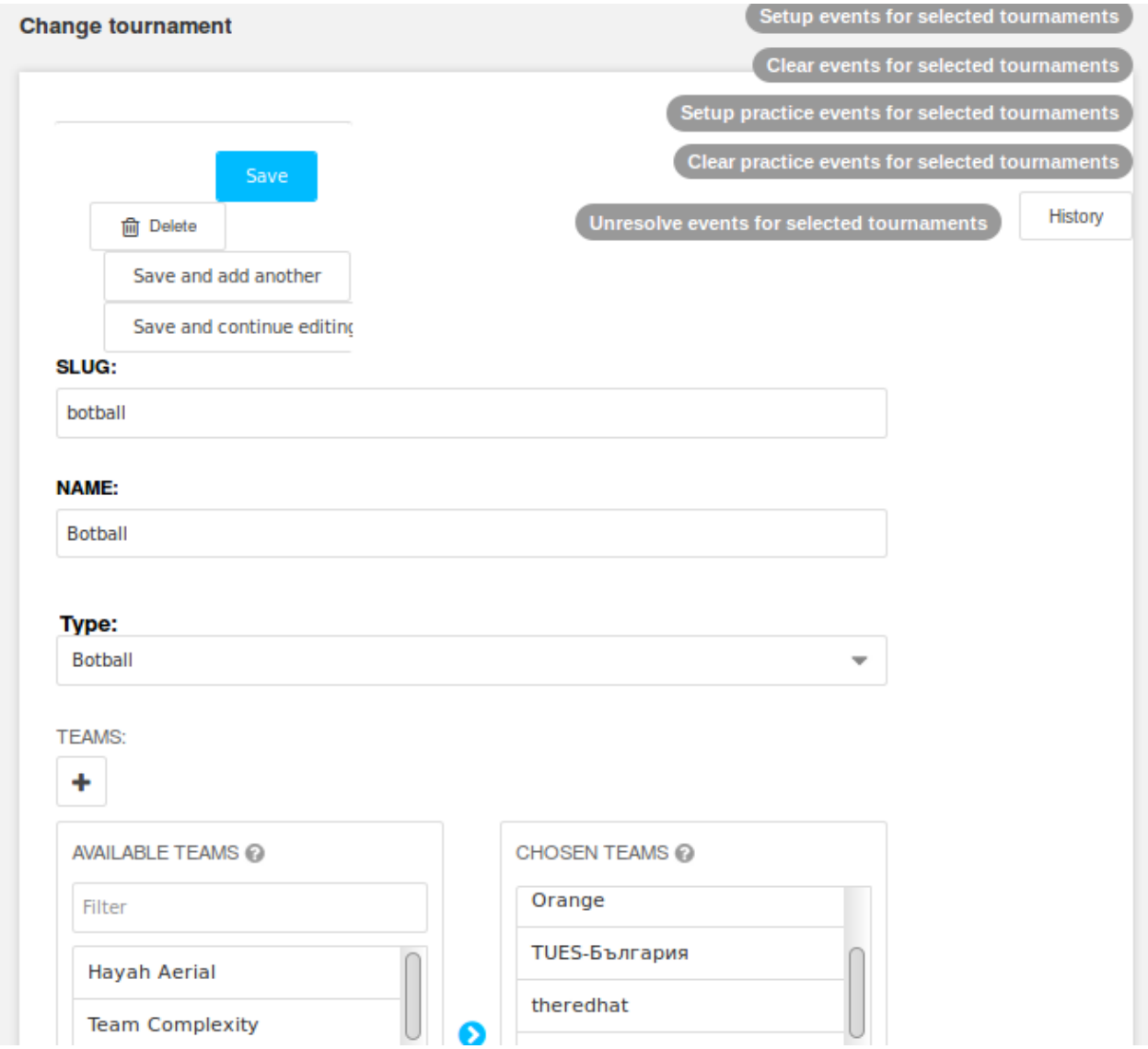

Figure 2: Part of the tournament management screen

Figure 2 Shows the main tournament setup screen. At the top, there are buttons for creating and deleting appointments according to the selected tournament type. In case of Botball, this means Open Practice, Seeding, Documentation (Period and Onsite), Double Elimination, and – for European Botball – student paper submission appointments.

Tournament types other than Botball currently only have limited support; for example, it should be possible to add or remove rounds of an Alliance tournament depending on time constraints, but this is not implemented.

The authors want to emphasize that the administration screens seen here are generated by Django with minimum programming effort.prototyping new features does not necessitate writing data entry masks before first results are visible.

3.4. Calling and Scoring

From an organizer's point of view, it is very important to have minimal software handling overhead during the busy tournament. This means that getting teams to the game tables and results back must be easy, and/or it must be possible to delegate the relevant tasks to others.

Calling teams is not a feature of Original Joust. Instead, at least when the author participated at GCER 2011, pagers were used to get teams' attention. The operation of these is not known to the authors.

Scoring in Original Joust is implemented as a simple HTML form: the user clicks a link, fills in the score/winner, and submits the form [1]. Loading and submitting the form each take a round-trip to the server, introducing a slight but noticeable delay.

Figure 3: New Joust's scoring interface

Figure 3 shows New Joust's interface for calling and scoring "Scored Events"; these include documentation, seeding and student papers. The forms for practice events and matches look similar, with a fitting scoring column, or lack thereof.

The table of forms is generated using AndularJS from data received via New Joust's REST interface. A form is submitted by pressing the button to the right; the color indicates whether there is unsaved data. The form data is sent to the server via JavaScript, so there is no delay before another form can be filled, or a user could fill multiple forms before submitting them all.

Each event has a status; one of "Scheduled", "Upcoming", "In Progress", or "Finished". The status is used for team calling: "Upcoming" and "In Progress" events are shown on the front page, as seen in Figure 4, updated automatically using WebSockets.

A “gong” sound can be configured and played over a loudspeaker whenever the game list is updated. Teams can also open the front page and get gongs on their own laptops. This notification feature could be further improved by allowing to select team(s) and only playing the gong for these.

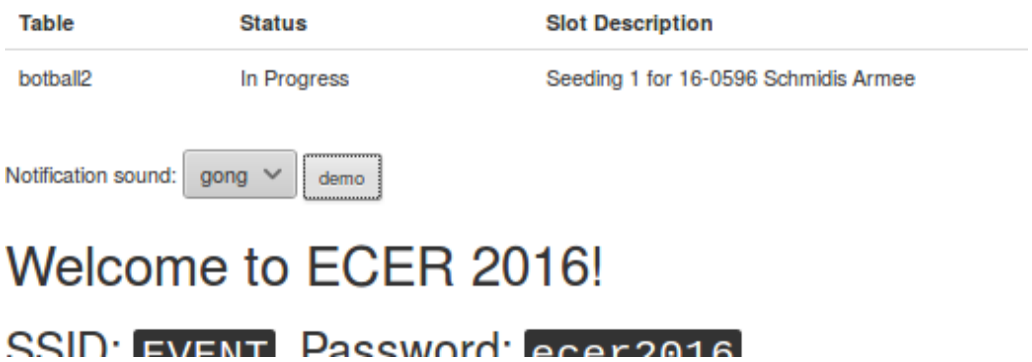

Figure 4: The front screen, showing an in-progress seeding game

3.5. Displaying results

An important feature that is simply missing from New Joust is an appropriate display of results, for example the double elimination bracket. The only currently implemented public-facing displays are the front page, and a simple list of games & matches. As with the front screen, that list refreshes automatically when new games & matches are scored.

4. Reports from ECER 2016

New Joust was used for the first time at the European Conference on Educational Robotics (ECER) in April, 2016 [14]. ECER 2016 had almost 200 participants in 33 teams, coming from 7 different countries. The event consisted of five competitions: European Botball (21 teams), PRIA Aerial (2 teams), PRIA Open (8 teams), PRIA Underwater (3 teams), and Botball Alliances (6 alliances).

We were able to delegate almost all of the game table handling to volunteers: after instructing them on how to call teams and giving them a duration for open practice slots, they could run an open practice session (Botball & PRIA Open, 2 game tables) almost without intervention.

Two features would have improved game table handling greatly: an integrated stopwatch, and a queue of events. The Calling & Scoring screen currently present a list of events, and volunteers choose one by one which event happens next on which game table. Their job could be simplified by having a queue, probably per game table, where marking one event as “Finished” automatically marks the next event as “In Progress”, and the one after that as “Upcoming”.

We also ran into a problem during Double Elimination: due to an incorrect score sheet, one match was scored incorrectly in the system. When the winner was corrected, that change was not propagated to the next match, resulting in an invalid DE pairing. The error was luckily found by one of our staff before that match was held, and was fixed using a pre-DE database backup.

We ended up only using New Joust for Botball, PRIA Open and Alliances, not for PRIA Aerial and PRIA Underwater, as the low number of teams made it more appropriate to call teams in these tournaments ad hoc, instead of adhering to a strict schedule.

5. Conclusion

In its current stage, the presented software is able to handle multiple tournaments of different types, although support for non-Botball tournaments is as of yet limited. Experiences from ECER 2016 show that the data entry masks for game table volunteers are suitable for large scale tournaments (more than 30 teams), even though further improvements are possible. In cases where the volunteers' masks are not sufficient, administrators can benefit from Django's backend, where any data may be accessed.

Some features, such as data import and export or the graphical Double Elimination bracket, are not yet implemented. These limitations need to be addressed before replacing Joust completely can be considered.

The usage of RESTful web services for exposing data, Bootstrap for styling, Angular for dynamic content, and Websockets for asynchronous notifications make this new Joust a modern application. A foundation of unit tests encourages the extension of Joust for future needs.

The authors hope to grow Joust into a mature tool to be ready for broad usage by the 2017 Botball season.

Acknowledgements

The authors want to thank the Practical Robotics Institute Austria and the Kiss Institute for Practical Robotics for their strong and ongoing commitment to robotics education.

The authors acknowledge the financial support from the European Union's Horizon 2020 research and innovation program under grant agreement No. 665972.

References

[1] "Using Joust.pdf", accessible via the KIPR Dropbox, retrieved 2016-05-24

[2] https://www.apachefriends.org/, retrieved 2016-06-07

[3] http://framework.zend.com/downloads/archives, retrieved 2016-06-07

[4] http://framework.zend.com/docs/quickstart, archived 2009-02-12 at https://web.archive.org/web/20090212173906/http://framework.zend.com/docs/quickstart

[5] https://sqlite.org/, retrieved 2016-06-07

[6] https://secure.php.net/manual/en/book.pdo.php, retrieved 2016-06-07

[7] https://www.djangoproject.com/, retrieved 2016-06-07

[8] http://www.djangobook.com/en/2.0/chapter05.html#the-mtv-or-mvc-development-pattern, retrieved 2016-06-07

[9] https://getbootstrap.com/getting-started/, retrieved 2016-06-07

[10] https://angularjs.org/, retrieved 2016-06-07

[11] https://jquery.com/, retrieved 2016-06-07

[12] https://pip.pypa.io/en/stable/, retrieved 2016-06-07

[13] http://bower.io/, retrieved 2016-06-07

[14] https://pria.at/ecer/, retrieved 2016-06-07