SportChef

Technical Documentation

Marcus Fihlon

October 18, 2015

Abstract

This document holds the technical documentation for SportChef, a suite of applications and services to organise sports events from registration through the execution to the ranking list publication, everything in real-time!

Here you can find the definitions and specifications needed to develop all parts of SportChef and provide maintenance. Actually SportChef is under heavy development — this means that this document is under heavy development, too!

Acknowledgements

Contents

Chapter 1

Common

1.1 Communication

The communication between the server and the clients is based on HTTP[1] requests and uses the REST[2] approach. Data that will be exchanged between the server and the clients has to be conform to the JSON[3] standard.

1.2 Security

While the first version is under development, the connections are not encrypted. SSL[4] security (HTTPS[5]) will be added later before the release. This decision is based on two reasons: First, a commercial SSL certificate is needed which is valid only for a limited time. Buying it before it is really needed makes no sense and wastes money. Second, adding security using SSL certificates adds additional complexity to development in a very early stage which makes the learning curve for new developers unnecessary steep.

1.3 Definitions

1.3.1 Server

Technical specifications

The SportChef server is written in Java[6] using JavaEE[7] technology and can be deployed to every application server [8] which is Java EE 7 compatible.

1.3.2 Webinterface

Technical specifications

The web client is written in Java using JSF[9] technology and the PrimeFaces[10] framework.

---

[1]http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[2]http://en.wikipedia.org/wiki/Representational_state_transfer
[3]http://en.wikipedia.org/wiki/JSON
[4]http://en.wikipedia.org/wiki/Transport_Layer_Security
[5]http://en.wikipedia.org/wiki/HTTP_Secure
[6]http://en.wikipedia.org/wiki/Java_(programming_language)
[7]https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition
[8]https://en.wikipedia.org/wiki/Application_server
[9]https://en.wikipedia.org/wiki/JavaServer_Faces
[10]http://www.primefaces.org

Browser support

The SportChef web interface uses HTML 5, so every HTML 5 capable browser will do. The web interface can be accessed by desktop systems as well as by tablet computers and smartphones. The user interface is fully responsive.

Chapter 2

Application Programming Interface

2.1 General information

2.1.1 JSON listings

The JSON listings in this documentation are reformatted for better readability. The server usually delivers a "compressed" variant without unnecessary spaces and line breaks.

2.2 Event Resource

2.2.1 Create a new event

Send a POST request with the following JSON to the URI: /api/events

```
1  {
2    "title" : "Christmas Party",
3    "location" : "Town Hall",
4    "date" : "2015-12-24",
5    "time" : "18:00"
6  }
```

The response usually is a "201 Created" with a "Location" header set to the URI of the newly created event or an error response.

2.2.2 Read an existing event

Send a GET request to the URI: /api/events/{eventId}
The response usually is a "200 OK" with the following JSON or an error response.

```
1  {
2    "eventId" : 1,
3    "title" : "Christmas Party",
4    "location" : "Town Hall",
5    "date" : "2015-12-24",
6    "time" : "18:00"
7  }
```

2.2.3 Read all events

Send a GET request to the URI: `/api/events`

The response usually is a "200 OK" with the following JSON or an error response.

```
 1  [
 2    {
 3      "eventId" : 1,
 4      "title" : "Christmas Party",
 5      "location" : "Town Hall",
 6      "date" : "2015-12-24",
 7      "time" : "18:00"
 8    },
 9    {
10      "eventId" : 2,
11      "title" : "New Year Party",
12      "location" : "Town Hall",
13      "date" : "2015-12-31",
14      "time" : "20:00"
15    }
16  ]
```

2.3 User Resource

2.3.1 Create a new user

Send a POST request with the following JSON to the URI: `/api/users`

```
 1  {
 2    "firstName" : "John",
 3    "lastName" : "Doe",
 4    "phone" : "+41 79 555 00 01",
 5    "email" : "john.doe@sportchef.ch"
 6  }
```

The response usually is a "201 Created" with a "Location" header set to the URI of the newly created user or an error response.

2.3.2 Read an existing user

Send a GET request to the URI: `/api/users/{userId}`

The response usually is a "200 OK" with the following JSON or an error response.

```
 1  {
 2    "userId" : 1,
 3    "firstName" : "John",
 4    "lastName" : "Doe",
 5    "phone" : "+41 79 555 00 01",
 6    "email" : "john.doe@sportchef.ch"
 7  }
```

2.3.3 Read all users

Send a GET request to the URI: `/api/users`

The response usually is a "200 OK" with the following JSON or an error response.

```
 1  [
 2    {
 3      "userId" : 1,
 4      "firstName" : "John",
 5      "lastName" : "Doe",
 6      "phone" : "+41 79 555 00 01",
 7      "email" : "john.doe@sportchef.ch"
 8    },
 9    {
10      "userId" : 2,
11      "firstName" : "Jane",
12      "lastName" : "Doe",
13      "phone" : "+41 79 555 00 02",
14      "email" : "jane.doe@sportchef.ch"
15    }
16  ]
```

2.3.4 Update an existing user

Send a PUT request with the following JSON to the URI: `/api/users/{userId}`

```
 1  {
 2    "firstName" : "Jane",
 3    "lastName" : "Doe",
 4    "phone" : "+41 79 555 00 01",
 5    "email" : "jane.doe@sportchef.ch"
 6  }
```

The data of the user with the specified ID will be replaced by the data specified in the JSON. The response usually is a "200 OK" with the following JSON and a "Location" header set to the URI of the modified user or an error response.

```
 1  {
 2    "userId" : 1,
 3    "firstName" : "Jane",
 4    "lastName" : "Doe",
 5    "phone" : "+41 79 555 00 01",
 6    "email" : "jane.doe@sportchef.ch"
 7  }
```

2.3.5 Delete existing users

Send a DELETE request to the URI: `/api/users/{userId}`

The data of the user with the specified ID will be deleted. The response usually is a "204 No Content" or an error response.

A List of Figures

B List of Tables

Appendix C

License