Plat_Forms 2011 Task: CaP

Ulrich Stärk, ulrich.staerk@fu-berlin.de
Lutz Prechelt, prechelt@inf.fu-berlin.de

Institut für Informatik
Freie Universität Berlin
Berlin, Germany
http://www.inf.fu-berlin.de/w/SE/

January 2011

Abstract

This document contains the requirements for the system to be built by the participants of the Plat_Forms 2011 contest. The system is called Conferences and Participants (CaP). It is to be written within two days by a team of three people. For further details about the contest, please see www.plat-forms.org.

Contents

1 Introduction

1.1 Theme/purpose of the CaP

Conferences and Participants (CaP) is a simple portal for organizing and searching for conferences of different kinds: organizers create conferences in certain categories, add details like venue and acommodation and invite participants. Interested users can browse categories, search for conferences, register for a conference and create personal calendars for conferences they are interested in.

The system has both an interactive user interface via HTML pages and a RESTful programmatic interface.

You will be provided with a file called `data.txt` containing sample data which you can use during development. See 3.3 and 3.10.

1.2 Requirements priorities and notation

The requirements described in this document are categorized into three different priority levels and each requirement is marked accordingly:

MUST — MUST marks an essential requirement. Unless all of these requirements are implemented, the system is considered inacceptable.

SHOULD — SHOULD marks an important requirement. If some of these requirements are not implemented, the system is considered incomplete, but acceptable.

MAY — MAY marks an optional requirement. These requirements are considered nice-to-have but need not be implemented when time is short or if their cost-benefit ratio is considered too high.

Each requirement is marked by exactly one of these three terms, followed by a subscript number that is the unique reference number of that requirement (also repeated in the margin).

If a compound requirement (such as a use case) is marked MUST, that means at least the MUST requirements contained in it must be realized. It does not mean that all of its subrequirements must be realized.

If a compound requirement is marked SHOULD or MAY, it will be considered realized only if all of the MUST requirements contained in it are realized.

If you make tradeoffs among requirements, maximize the user value of the portal, rather than showing off your technical capabilities.

Phrases *in italics* in the use cases below are references to further use cases, the corresponding section number is appended in parentheses as a hyperlink.

2 Functional requirements: Use Cases

Each use case consists of three subsections: The mandatory main scenario section describes the standard sequence of events for the use case. The optional exceptions and variants section describes what happens in important error cases and which voluntary deviations from the main scenario should be considered. The optional notes section provides detail information where needed.

2.1 User actions overview

2.1.1 Main scenario

$MUST_1$ M 1

1. The *user browses* (2.2) the portal.
2. The *user searches for conferences* (2.3).
3. The *user registers* (2.4) to become a member.
4. The portal greets the new member with a few member statistics (MAY_2) and a list of things to do ($SHOULD_3$), in particular creating a new conference. m 2 S 3
5. The *member creates a conference* (2.5).
6. The *member administers the portal* (2.9).
7. The member logs out ($MUST_4$). M 4

2.1.2 Exceptions and variants

- 3b. The member logs in ($MUST_5$). M 5
- 4b. The portal greets an existing member with a list of notifications, specifically RCDs and conference invitations (MAY_6, see 2.8). m 6

2.1.3 Notes

- Request for Contact Details (RCD): From the point of view of member A, another member B can be in four different states: *no_contact* (the initial state) means no RCD has been sent between the two, *RCD_sent* means A has made an RCD to B, *RCD_received* means B has made an RCD to A, and *in_contact* means each has made an RCD to the other and both can see the other's contact details (full name and email address) which are normally hidden. Sending an RCD happens in the *no_contact* and *RCD_received* state only, the latter case is called "answering an RCD". A positive answer leads to the *in_contact* state, a negative answer to the *no_contact* state. RCD

2.2 User browses portal

2.2.1 Main scenario

M 7 MUST$_7$

1. On the portal's main page, the user is presented with a list of categories (see category in the glossary, MUST$_8$) and a list of conferences (MUST$_9$). In addition to the generic list of conferences, the portal SHOULD$_{10}$ display a list of currently running conferences, a list of conferences starting the next day (MAY$_{11}$) and a list of conferences starting within the next week (MAY$_{12}$).
2. The user selects a category.
3. The portal updates the page's content to show the subcategories of the selected category (MUST$_{13}$) and updates the lists of conferences to only show those belonging to the selected category or one of its subcategories (MUST$_{14}$).
4. The user adds a conference to his default personal calendar (SHOULD$_{15}$, see 2.8).
5. The member calls the *Conference Summary* (2.6) of any of the conferences from the lists.
6. The portal navigates to the *Conference Summary* (2.6) of the selected conference (MUST$_{16}$).

2.2.2 Exceptions and variants

- 1b. Long lists (more than 10/25/50 entries, MAY$_{17}$ be configurable) MAY$_{18}$ be paginated.

2.3 User searches for conferences

2.3.1 Main scenario

M 19 MUST$_{19}$

1. The portal presents a search dialog with the following filtering choices:
   - choose one or more categories (category in the glossary, MUST$_{20}$)
   - only conferences within a specified time range (MUST$_{21}$)
   - whether or not to also search in subcategories (SHOULD$_{22}$)
   - only conferences in my country (MAY$_{23}$)
   - only conferences within a 50/500/2000/5000 kilometers radius of my location (SHOULD$_{24}$)
2. The user enters an optional search term (see Notes), selects some choices and submits the search.
3. The portal finds all conferences that satisfy the given query and presents them as a list.

2.3.2 Exceptions and variants

- 1b. The user chooses the start and end dates from a pop-up calendar (MAY_{25}). m 25
- 2b. instead of using the UI, the user makes his filtering choices using a special query syntax ($MUST_{26}$, see notes below). M 26

2.3.3 Notes

- Filtering choices that can't be applied due to missing data (e.g. missing GPS coordinates for the current user or user is not logged in) $SHOULD_{27}$ be disabled. S 27
- By default, the portal $SHOULD_{28}$ not search in subcategories. S 28
- The search term $MUST_{29}$ be searched for in the name and description of a conference. Multiple search terms separated by a space are combined with a logical AND, meaning that all terms must be present in the result. M 29
- When specifying a time range, conferences running at the specified start and end dates $MUST_{30}$ be returned (i.e. conference end date <= start date of time range and conference start date >= end date of time range). M 30
- If no time range is specified, only running conferences or conferences starting later than the current date $SHOULD_{31}$ be returned. S 31
- The query syntax is roughly defined by the following rules whose syntax borrows from EBNF (Extended Backus-Naur Form) and regular expressions. See also the examples below.

```
querySyntax = queryExpression* (" " queryExpression)*
queryExpression = categoryModifier | dateModifier | optionExpression |
regionModifier | queryTerm
dateModifier = from | until | from until      MAY32                    m 32
from = "from:" date
until = "until:" date
optionExpression = optionPrefix option      SHOULD33                   S 33
optionPrefix = "opt:"
option = "withsub"
categoryModifier = categoryPrefix category      MUST34                 M 34
categoryPrefix = "cat:"
regionModifier = regionPrefix region      MAY35                        m 35
regionPrefix = "reg:"
region = "country" | \d+
category = [\w\d]+
queryTerm = [\w\d]+      MUST36                                        M 36
date = \d{8}
```

In the case of the `date` term, the portal MAY_{37} accept different ways of specifying the date in addition to the number-only format, e.g. 2011/01/18, 18.01.2011 and 2011-1-18. Whitespace inside date strings is not allowed. m 37

A valid query may be *cat:technology opt:withsub from:20110118 javaone* and would return all conferences in the *technology* category and all its subcategories starting on January 18th, 2011 or later that match the query term *javaone*. Other examples are

- *cat:science software engineering*, which finds *science* conferences containing *software engineering* in their name or description
- *reg:50 programming*, which searches for conferences containing *programming* in their name or description within a 50 kilometers radius around the user's location
- *cat:medicine reg:country*, which finds all conferences from the *medicine* category in the user's country currently running or starting in the future

2.4 User registers

2.4.1 Main scenario

M 38 $MUST_{38}$

Precondition: User is not logged in

1. The user enters the following mandatory information (see the notes):
   M 39 — full name, email address ($MUST_{39}$)
   M 40 — town, country ($MUST_{40}$)
   M 41 — username, password ($MUST_{41}$)
2. The user enters the following optional information (see the notes):
   S 42 — GPS coordinates of place of residence ($SHOULD_{42}$)
3. The user submits this data for registration.
4. The portal validates the username, registers the user as a new member, stores the data, and logs
   M 43 in the member ($MUST_{43}$).

2.4.2 Exceptions and variants

- 2b. The user can also add this information later using the Status Page.
- m 44 2c. The portal MAY_{44} use the W3C Geolocation API to detect the user's location, using either the browser's native Geolocation API support or through Google Gears.
- 4b. The portal rejects the username because it is not unique (already in use) and sends the user
  M 45 back to step 1 ($MUST_{45}$). This MAY_{46} be done without the need for a full page refresh, using
  m 46 Ajax.
- m 47 4c. The portal MAY_{47} validate the email address by sending an activation token to the specified address (see 5.7.
- m 48 4d. The portal MAY_{48} validate the email address to have a valid domain. This MAY_{49} be done
  m 49 using Ajax.

2.4.3 Notes

Steps 1 and 2 should be considered a whole and can be distributed over one or several dialog pages according to the chosen UI philosophy.

Full name and email address are initially hidden from view for other members. Members can make them visible on a one-to-one basis by the RCD mechanism described in 2.8. All other data (except the password) is considered public. See omit member details in the glossary. GPS

GPS coordinate format: GPS coordinates are entered textually in decimal notation. They $MUST_{50}$ conform to the following regular expression: \d+(\.\d+)? ?[NnSs] ?,? ?\d+(\.\d+)? ?[EeWw] M 50

Example GPS coordinates are 49.417716N,11.113712E.

GPS coordinate precision: The user can freely choose the precision of the GPS coordinates (number of decimal places) and hence the precision with which the location is revealed.

GPS coordinate determination: The portal MAY_{51} provide an explanation how to determine one's own GPS coordinates via Google Maps and its "URL for this page" link. m 51

2.5 User creates a conference

2.5.1 Main scenario

$MUST_{52}$ M 52

Precondition: User (member) is registered and logged in.

1. The user enters the following mandatory information:
   — Name of the conference ($MUST_{53}$) M 53
   — Date the conference starts ($MUST_{54}$) M 54
   — Date the conference ends ($MUST_{55}$) M 55
   — Categories for the conference (from a list of categories existing in the portal, see category in the glossary, $MUST_{56}$) M 56
   — Description of the conference ($MUST_{57}$) M 57
   — Location of the conference (as an address, $MUST_{58}$) M 58

2. The user enters the following optional information:
   — GPS coordinates of the location ($SHOULD_{59}$) S 59
   — Information about the venue (MAY_{60}) m 60
   — Information about accomodation (MAY_{61}) m 61
   — How to find the location (MAY_{62}) m 62

3. The user submits the data.

4. The portal validates the formats of the inputs, stores the data and redirects the user to the *Conference Summary* (2.6) of the newly created conference ($MUST_{63}$). M 63

2.5.2 Exceptions and variants

m 64 • 1b. If the user is an official contact for a conference series, he MAY_{64} be able to choose a conference series for the conference from a list of conference series he is the official contact for (see conference series in the glossary).

m 65 • 1c. The user chooses the start and end dates from a pop-up calendar (MAY_{65}).

m 66 • 2b. The portal MAY_{66} use existing webservices like the Google Geocoding API to find the location's GPS coordinates based on the location's address given in 1.

• 4b. The portal should accept different date formats and informs the user about the possible
m 67 formats (MAY_{67}).

2.6 User works with Conference Summary

2.6.1 Main scenario

M 68 $MUST_{68}$

Precondition: The user is logged in and has chosen a conference to display

1. The portal displays
   - M 69 the information entered during the creation of the conference ($MUST_{69}$, see 2.5).
   - M 70 the name and email address of the creator ($MUST_{70}$)
   - M 71 a list of attendees with username, full name and email address ($MUST_{71}$)
2. M 72 The user downloads an iCalendar file with the information in 1 ($MUST_{72}$). Information that has no correspondence in the iCalendar format is ommitted.
3. S 73 The user downloads a PDF version of the information in 1 ($SHOULD_{73}$).
4. S 74 The user signs up for an RSS feed of the list of attendees ($SHOULD_{74}$, see omit member details in the glossary).
5. M 75 The user signs up for the conference ($MUST_{75}$).
6. M 76 The user selects other users from a list of users with in_contact status and invites them to attend ($MUST_{76}$).
7. M 77 The portal sends a notification to the invited user ($MUST_{77}$).
8. S 78 The user enters one or more email addresses of unregistered users to invite. An email with a confirmation link is sent to the invitee ($SHOULD_{78}$, see 5.7).

2.6.2 Exceptions and variants

M 79 • 1b. If the user is the creator of the conference, he $MUST_{79}$ be able to modify some or all of the information, including adding additional categories or modifying existing ones (see category in
M 80 the glossary, $MUST_{80}$).

- 1c. If the user modifies the start or end date of the conference, all attendees of the conference $SHOULD_{81}$ be notified. The nofication MAY_{82} include an updated iCalendar file. S 81 m 82
- 1d. Instead of full name and email address of an attendee the username is shown unless the user has in_contact status or is the creator of the conference (see omit member details in the glossary, $MUST_{83}$). M 83
- 1e. If an attendee was invited by email, a placeholder is shown instead of username and full name ($MUST_{84}$). M 84
- 2b. The user MAY_{85} choose whether to include the list of attendees in the generated iCalendar file. m 85
- 3b. The user MAY_{86} choose whether to include the list of attendees in the generated PDF. m 86
- 5b. If the user already signed up for the conference, he can cancel his attendance ($MUST_{87}$). M 87
- 5c. If the user is the creator of the conference, he MAY_{88} not be able to sign up. m 88

2.7 Member works with Search For Members

2.7.1 Main scenario

$MUST_{89}$ M 89

Precondition: User (member) is registered and logged in.

1. The portal presents a search dialog with the following filtering choices (see notes for details):
2. Status-related choices:
   — only members who are not yet contacts of mine (MAY_{90}) m 90
   — only members who have not yet received an RCD from me (MAY_{91}) m 91
3. Location-related choices:
   — only members in my town (MAY_{92}) m 92
   — only members in my country (MAY_{93}) m 93
   — only members who live less than 5/10/20/50/100/200/500/1000/2000/5000 kilometers away (MAY_{94}) m 94
4. The member enters an optional search term, selects some choices and submits the search ($MUST_{95}$). M 95
5. The portal finds all members that satisfy all of the filters and presents them as a list with the following attributes: username, town and country ($MUST_{96}$). M 96
6. The member selects some members whom he/she has not yet sent a Request For Contact Details (RCD, $MUST_{97}$, see 2.1.3) M 97
7. The portal sends an RCD to the selected members ($MUST_{98}$). M 98
8. The member requests to see the *Status Page* (see 2.8) of one member from the list ($MUST_{99}$). M 99
9. The portal shows the requested *Status Page* (2.8).

2.7.2 Notes

GPS x kilometers away: The comparison is performed based on the GPS coordinates alone. You need
M 100 not implement spherical geometry computations, rather you $MUST_{100}$ use the simple substitute as explained in the glossary. See distance calculation in the glossary.

GPS data precision: The low precision that some users may have chosen for their GPS coordinates is ignored in the distance computation. All data is treated as if it was arbitrarily precise.

Choices that make no sense because of missing data (e.g. lack of GPS coordinates for the current
S 101 user) $SHOULD_{101}$ be disabled.

If a search term is entered, the username and full name fields are searched for members with in_contact
M 102 status. For all others only the username field is searched ($MUST_{102}$).

2.8 Member works with Status Page

2.8.1 Main scenario

M 103 $MUST_{103}$

Precondition: The user is logged in as member A

1. The Portal presents a status page about the member A that contains the following information:
   M 104 — I: the set of information that is usually submitted on registration ($MUST_{104}$, see 2.4)
   — M: a link to an external Google Maps page that will show a map (about 100 km across) of
   m 105 the area around the member's given GPS coordinates (MAY_{105})
   S 106 — N: a list of notifications ($SHOULD_{106}$)
   M 107 — C: a list of members with in_contact status ($MUST_{107}$)
   S 108 — S: a list of members with RCD_sent status ($SHOULD_{108}$)
   M 109 — R: a list of members with RCD_received status ($MUST_{109}$)
   M 110 — P: the user's personal calendars ($MUST_{110}$)
2. S 111 The member reviews and modifies some or all of the information I ($SHOULD_{111}$).
3. S 112 The member reads and discards some or all of the notifications N ($SHOULD_{112}$).
4. S 113 The member acts on an invite notification ($SHOULD_{113}$).
5. M 114 The member selects some members from the RCD_received list and answers positively ($MUST_{114}$).
6. M 115 The portal updates the RCD_sent and in_contact lists ($MUST_{115}$).
7. M 116 The member selects some members from the RCD_received list and answers negatively ($MUST_{116}$).
8. M 117 The portal updates the RCD_sent list ($MUST_{117}$).
9. The member adds a personal calendar showing conferences within a specified category, includ-
   S 118 ing subcategories ($SHOULD_{118}$, see category in the glossary).
10. m 119 The member adds a personal calendar showing conferences in his town/country (MAY_{119}).
11. The member adds a personal calendar showing conferences within a 50/500/2000/5000 kilome-
    m 120 ter radius of his location (MAY_{120}).

12. The member suggests a new category (MAY_{121}, see also 2.9). m 121
13. The member calls the Status Page of a member X from any of the lists C, S or R ($MUST_{122}$). M 122
14. The portal presents the *Status Page* (2.8) of X.

2.8.2 Exceptions and variants

If the Status Page shown is not about member A, but rather about a different member B, the notifications list N, the member lists S and R and the personal calenders P are not shown and steps 2 to 11 are not possible ($MUST_{123}$). M 123

Unless B has in_contact status, the contact details (full name, email address) are also not shown ($MUST_{124}$). M 124

2.8.3 Notes

The lists MAY_{125} be sortable by any of the attributes. m 125

Long lists (more than 10/25/50 entries, MAY_{126} be configurable) MAY_{127} be paginated. m 126

Clicking on a user's details in one of the lists MAY_{128} open that user's Status Page. m 127 m 128

The member always has a default personal calendar. Additional personal calendars $SHOULD_{129}$ have a user-configurable title. S 129

Google Maps: Information about how to construct a suitable Google Maps URL can be found on the web (but not at Google). The most relevant parameters are `z` and `ll`.

2.9 Member administers the portal

2.9.1 Main scenario

$MUST_{130}$ M 130

Precondition: Member is logged in and has administrative rights

1. The administrator creates a new category, possibly as a subcategory of an existing one (see category in the glossary, $MUST_{131}$). M 131
2. The administrator creates a conference series (MAY_{132}), supplying a mandatory name (MAY_{133}), an optional description (MAY_{134}) and an optional URL where more information about the series can be found (MAY_{135}). See conference series in the glossary. m 132 m 133 m 134 m 135
3. The administrator makes a member the official contact of a conference series (MAY_{136}). See conference series in the glossary. m 136
4. The administrator grants administrative rights to a member (MAY_{137}). m 137
5. The administrator removes a conference from a category ($SHOULD_{138}$). S 138
6. The portal notifies the conference creator about the removal from a category ($SHOULD_{139}$). S 139

2.9.2 Exceptions and variants

m 140 • 1b. The administrator creates the category from a user suggestion (MAY_{140}, see 2.8).

S 141 • 3b. The administrator $SHOULD_{141}$ be able to search the member by username, full name or email address.

S 142 • 4b. The administrator $SHOULD_{142}$ be able to search the member by username, full name or email address.

2.9.3 Notes

M 143 The portal $MUST_{143}$ have a default user with administrative rights with username *admin* and password *admin*.

3 Functional requirements: Web Service interface

3.1 General information

You are required to write a RESTful web service interface for your solution using HTTP methods for the operations and JSON (JavaScript Object Notation) for data transfer. The MIME type of the data MUST$_{144}$ be `application/json`. M 144

Your implementation of the service MUST$_{145}$ be bound to the URL `/ws` on your server. M 145

With two single exceptions (reset, section 3.10 and factorydefaults, section 3.11), the operations in the service follow the use cases described in Section 2 quite closely and represent most (but not all) of their underlying business functionality.

Only those parts of the underlying functionality actually MUST$_{146}$ be present that you have also implemented on the usecase level (that is, in the HTML user interface). Therefore, input parameters that represent functionality you have not implemented on the usecase level MUST$_{147}$ simply be ignored and output attributes that represent functionality you have not implemented on the usecase level MUST$_{148}$ simply be omitted. The reset and factorydefaults operations must be implenented in any case. M 146 M 147 M 148

If you do not implement parts of the specification, your service MUST$_{149}$ return a *501 Not Implemented* HTTP status code for the respective resource and HTTP method. You MAY$_{150}$ also return a *405 Method Not Allowed* for all other HTTP methods than those specified. M 149 m 150

The subsequent descriptions of the individual operations can only be fully understood in conjunction with the usecases in Section 2 (because those describe most of the semantics).

3.2 Authentication

Authentication MUST$_{151}$ be done using HTTP Basic Access Authentication. Username and password are the same as given during user registration. M 151

3.3 Data objects

Data transfer with the web service is done using JSON objects. The following subsections describe these objects. Your implementation MUST$_{152}$ follow the specification for these objects. M 152

In an array context or when referenced inside another object, the objects MUST$_{153}$ be replaced by a stub object containing only the fields printed in bold and an additional field `details` of type `string` M 153

which represents an URL where the full details for this object can be retrieved. E.g. `[ { "username" : "johndoe", "details" : "http://example.com/ws/members/johndoe" } ]`

Sample data can be found in the file `data.txt`.

3.3.1 The `conference` object

See 2.5 for details of conference creation.

field	type	description
version	string	
id	number	
name	string	
creator	object	Reference to `member` object
series	object	Reference to `series` object
startdate	string	
enddate	string	
categories	array	A list of `category` objects
description	string	
location	string	
gps	string	GPS coordinates, follows format conventions as described in 2.4.3
venue	string	
accomodation	string	
howtofind	string	

3.3.2 The `category` object

See category in the glossary.

field	type	description
version	string	
id	number	
name	string	
parent	object	Reference to a parent `category` object
subcategories	array	A list of `category` objects

3.3.3 The `series` object

See conference series in the glossary.

field	type	description
version	string	
id	number	
name	string	
contacts	array	A list of `member` objects (official contacts)

3.3.4 The member object

See 2.4 for details of user creation.

field	type	description
version	string	
id	number	
username	string	
password	string	
fullname	string	
email	string	
town	string	
country	string	
gps	string	GPS coordinates, follows format conventions as described in 2.4.3
status	string	RCD status (see 2.1.3)

3.4 URL /conferences

Allows the consumer to add and manipulate as well as retrieve information on conferences. The following operations translate roughly to the use case described in 2.6.

(URL template) /

(HTTP method) POST

Create a new conference by submitting a conference object ($MUST_{154}$). The version and id fields are ignored if set. M 154

(Status) 200 OK

if the conference was created successfully. Return the conference object of the newly created conference.

403 Forbidden

if the user tried to create a conference in a series where he is not the official contact. See conference series in the glossary.

400 Bad Request

if the request couldn't be fulfilled due to an erroneous conference object being posted.

/{id}

GET

Get the conference with the given id ($MUST_{155}$). M 155

200 OK

if the conference was found. Return the corresponding conference object.

404 Not Found

if no conference with the given id exists.

PUT

M 156 Update the specified conference by transferring an updated `conference` object (MUST$_{156}$).

200 OK
if the conference was updated successfully. Return the updated `conference` object.

400 Bad Request
if the request couldn't be fulfilled due to an erroneous `conference` object being posted.

403 Forbidden
if the user tried to tried to update the series to a series where he is not the official contact. See conference series in the glossary.

404 Not Found
if no conference with the given id exists.

409 Conflict
if the object has been modified between a previous GET and the current PUT request.

`/{id}/attendees`

GET

M 157 list all conference attendees for the conference with the given id (MUST$_{157}$).

200 OK
return an array of `member` objects.

204 No Content
if the conference has no attendees.

404 Not Found
if no conference with the given id exists.

POST

M 158 Attend the conference with the given id by uploading a `member` object. All fields but the `username` field MUST$_{158}$ be ignored, i.e. also an object that only contains a `username` field must be accepted.

204 No Content
if the subscription was succesful.

403 Forbidden
if the user tried to make someone else but himself an attendee.

404 Not Found
if no conference with the given id exists.

`/{id}/attendees/{username}`

DELETE

M 159 cancel the attendance of the user identified by `username` (MUST$_{159}$).

204 No Content
if the attendance was succesfully cancelled.

403 Forbidden
if the user tried to cancel the attendance of someone else but himself.

404 Not Found
if no conference with the given id exists.

3.5 URL /members

Allows the consumer to add and manipulate as well as retrieve information on members. The following operations translate roughly to the use case described in 2.8.

/

POST
Register a new user by submitting a `member` object ($MUST_{160}$). The version and id fields are ignored if set. M 160

200 OK
if the user was registered successfully. Return the `member` object of the newly created member.

400 Bad Request
if the request couldn't be fulfilled due to an erroneous `member` object being posted (e.g. duplicate username or other validity violations).

/{username}

GET
Retrieve information on the member identified by `username` ($MUST_{161}$). M 161

200 OK
if the member exists. Return the requested `member` object. Unless the requesting user has in_contact status with the requested user, the contact details (full name, email address) $MUST_{162}$ be ommitted. M 162

404 Not Found
if no member with the given username could be found.

PUT
Update the specified member by submitting an updated `member` object ($MUST_{163}$). M 163

200 OK
if the member was updated successfully. Return the updated `member` object.

400 Bad Request
if the request couldn't be fulfilled due to an erroneous `member` object.

403 Forbidden
if the user tried to update some other member but himself.

404 Not Found
if no user with the given username exists.

409 Conflict
if the object has been modified between a previous GET and the current PUT request.

`/{username}/contacts`

GET
M 164 List members with in_contact, RCD_sent or RCD_received status to `username` ($MUST_{164}$). If the requesting user is not the member identified by `username`, only contacts with in_contact status may be returned.

200 OK
if the member exists. Return an array of `member` objects.

204 No Content
if the member has no contacts or pending RCDs.

404 Not Found
if no member with the given username could be found.

POST
M 165 Send an RCD (see 2.1.3) to the specified user ($MUST_{165}$) by submitting a JSON object with a single field called `positive`. Sending and answering positively is the same: `positive` is set to *true*. Answering negatively is done by setting `positive` to *false*. Negatively answering to a member that had not send an RCD has no effect. Sending an RCD to a member with in_contact status has no effect either.

204 OK
if the RCD was sent successfully.

400 Bad Request
if the request couldn't be fulfilled due to erroneous data.

404 Not Found
if no user with the given username exists.

3.6 URL `/categories`

Allows the consumer to retrieve information on conference categories. Categories are used in multiple use cases, see also category in the glossary.

/

GET
M 166 Retrieve a list of top-level categories ($MUST_{166}$).

200 OK
return an array with all top-level categories.

204 No Content
if no top-level categories exist.

POST

Create a new category by submitting a `category` object (MUST$_{167}$). The version, id and subcategories fields are ignored if set. M 167

200 OK

if the category was created successfully. Return the `category` object of the newly created category.

400 Bad Request

if the request couldn't be fulfilled due to an erroneous `category` object being posted (e.g. duplicate category).

403 Forbidden

if the user does not have administrative privileges.

`/{id}`

GET

Retrieve information about a category (MUST$_{168}$). M 168

200 OK

return the `category` object for the category identified by `id`.

404 Not Found

if no category with the given `id` exists.

3.7 URL `/series`

Allows the consumer to retrieve information on conference series. Conference series are used in multiple use cases, see also conference series in the glossary.

`/`

GET

Retrieve a list of conference series (MAY$_{169}$). m 169

200 OK

return an array with all conference series.

204 No Content

if no conference series exist.

POST

Create a new series by submitting a `series` object (MAY$_{170}$). The version and id fields are ignored if set. m 170

200 OK

if the series was created successfully. Return the `series` object of the newly created series.

400 Bad Request

if the request couldn't be fulfilled due to an erroneous `series` object being posted.

403 Forbidden
if the user does not have administrative privileges.

`/{id}`

GET
m 171 Retrieve information about a conference series (MAY_{171}).

200 OK
return the `series` object for the conference series identified by `id`.

404 Not Found
if no conference series with the given `id` exists.

3.8 URL `/conferencesbycategory`

Allows the consumer to browse conferences by category. This roughly translates to the use case in 2.2. See also category in the glossary.

`/{id}`

GET
M 172 Retrieve conferences in a category ($MUST_{172}$).

200 OK
return an array of conferences within the category identified by `id`.

204 No Content
if there are no conferences in the given category.

404 Not Found
if no category with the given `id` exists.

3.9 URL `/search`

Allows the consumer to search for conferences. This corresponds to the use case in 2.3.

`/{query}`

GET
M 173 Search for conferences matching the specified query term ($MUST_{173}$). The query term conforms to the same syntax as in the normal search (see 2.3). URL encoding must take place where necessary.

200 OK
return an array of conferences.

204 No Content
if there are no conferences matching the query criteria.

400 Bad Request
if the query term is erroneous.

3.10 URL /reset

Allows an administrative user to reset the portal to its initial state.

/

GET
Reset the portal to its initial state (MUST$_{174}$). No users (except the default administrative user) are registerd, no categories, conferences, conference series, etc. exist. M 174

204 No Content
if the portal has been reset successfully.

403 Forbidden
if the user does not have administrative privileges.

3.11 URL /factorydefaults

Allows an administrative user to reset the portal to an initial state with initial data. A file with initial sample data called `data.txt` is available.

/

GET
Reset the portal to its initial state. It MUST$_{175}$ contain all the data as given in the sample data file `data.txt`. M 175

204 No Content
if the portal has been reset successfully.

403 Forbidden
if the user does not have administrative privileges.

4 Glossary

category A category groups an arbitrary set of conferences and possibly sub-categories by any criterion and for any purpose. Used on pages 6, 6, 9, 10, 12, 13, 16, 20, 22.

conference series A series groups a sequence of multiple instances of the same conference that occur over time. Usually all conferences within one series are organized by the same organization. Only an official contact user may create a conference in that series. Used on pages 10, 13, 13, 16, 17, 18, 21.

distance calculation with a longitude difference of x and a latitude difference of y, the distance d is $d = \sqrt{x^2 + y^2}$, where the unit of d is 100 kilometers. If you make sure you choose the shortest way, this is correct with respect to latitude differences and tends to overestimate longitude differences (except near the equator, where it underestimates, since $40,000/360 \approx 111$). Used on pages 12.

omit member details If a member A has not the in_contact status with member B, member A may only see part of B's details for privacy reasons. These details are username, town and country. Full name, email address, password and GPS coordinates may not be revealed. Used on pages 9, 10, 11.

5 Non-functional requirements

5.1 General

Passwords MUST$_{176}$ not be disclosed to anyone, neither on the UI nor on the web service level. See also omit member details in the glossary — M 176

For the web service, only those operations MUST$_{177}$ require authentication that also require an authentiaced user on the UI level. — M 177

5.2 User interface

The user interface MUST$_{178}$ conform to the above-mentioned requirements (as far as they are realized at all) in a sensible way with respect to the arrangement and markup of its elements and the labels, prompts and explanations that guide the user. Within those limits, the organization and design of the interface is left to the professional judgement of the participants. — M 178

The userinterface SHOULD$_{179}$ provide sufficient explanation of all uncommon concepts to guide a user who does not have prior knowledge about these topics. Make use of external links where needed. Do not include copyrighted material without permission. — S 179

Elaborate graphical design (MAY$_{180}$) is not important, but CSS SHOULD$_{181}$ be used throughout to simplify future improvements. — m 180, S 181

It would be nice if the user interface works even when Javascript is turned off in a user's browser (MAY$_{182}$). — m 182

5.3 Browser compatibility

The portal MUST$_{183}$ work fully with Firefox 3.0 and higher. — M 183

The portal MUST$_{184}$ work fully with Internet Explorer 7 and higher. — M 184

The portal SHOULD$_{185}$ work fully with Internet Explorer 6. — S 185

The portal SHOULD$_{186}$ work fully with Chrome 7 and higher. — S 186

The portal SHOULD$_{187}$ work fully with Safari 4 and higher. — S 187

The portal SHOULD$_{188}$ work fully with Opera 9 and higher. — S 188

The portal MAY$_{189}$ work fully with other browsers such as Konqueror, Opera Mini, Lynx etc. — m 189

If the portal relies on Javascript in the browser and Javascript is unavailable, it MUST$_{190}$ either produce a clear message saying that the portal is not functional (and why) or fall back to reduced (but still useful) functionality. — M 190

5.4 Scalability and efficiency

M 191 The system MUST_{191} scale without problems to 1,000,000 registered members, 100,000 conferences, 5,000 categories, and 5,000 series.

M 192 It MUST_{192} run efficiently throughout all of its functionality; no function must be implemented in an avoidably inefficient manner.

5.5 Persistence

M 193 All information (registration, conference, category, series, RCD status, ...) MUST_{193} be stored in persistent storage and must survive system shutdowns and crashes intact.

5.6 Session timeout

M 194 Login sessions MUST_{194} time out after one hour.

5.7 Integration

M 195 If your CaP implementation needs to send out an email, you MUST_{195} use the server `smtp.plat-forms.org` for that purpose. It works without encryption and without authentication. Where calls to other services are required, they are performed by the user's browser by following a URL supplied by the system. Where calls from other systems are required, they are the other system's responsibility and have to be performed via the web service interface.

5.8 Programming style

M 196 All identifiers and comments in the source code and helper files MUST_{196} be in English.

M 197 Each source code file MUST_{197} be documented at least globally (i.e. in its head) and shortly explain its purpose and which other parts of the system use its functionality.

m 198 Each non-trivial public program element (such as a method) MAY_{198} be documented (purpose, usage).

6 Rules for development

6.1 What is allowed

During the contest you may:

- Use any language, tool, middleware, library, framework, and other software you find helpful.
- Reuse any piece of any pre-existing application or any other helpful information you have yourself or can find on the web yourself. Anything that already existed the day before the contest started is acceptable.
- Use any development process you deem useful.
- Ask the organizer (who is acting like a customer) any question you like regarding the requirements and priorities (see 6.4).

6.2 What is not allowed

During the contest you may not:

- Disturb other teams in their work.
- Send contest-related email to people not on your team or transfer the requirements description (or parts thereof) to people not on your team.
- Have people from outside of your team help you. (This includes reusing work products from other teams.) There are two exceptions to this rule: (1) you may use answers of the customer as described in Section 6.4 and (2) you may use user-level preview feedback as described in Section 6.3.

6.3 User feedback: The blog

During the contest, teams SHOULD_{199} showcase intermediate versions of their CaP service to obtain user-level comments and feedback. To do this, host your CaP service on your development server, open it for public access, and post a notification in the Live Contest Blog as described below. S 199

Users can then comment on your CaP prototype regarding functionality, defects, usability etc. The teams are allowed to use this user-level feedback for improving their system. They are not allowed to post source code or to use information from outsiders that is on the code level.

6.3.1 How to post a message in the blog

Log in on plat-forms.org. Your username (teama, teamb etc.) and password (a 5-digit number) for the blog is available from Lutz Prechelt or Ulrich Stärk.

Go to Blog.

Middle column: Click "Create new blog entry." above the latest blog post

Fill the "Title" field. Mention the names of your home organization, platform, and the version number of your new prototype, say, "O'Reilly Perl team: Version 6".

Fill the "Body" section. Indicate for which aspects you are particularly interested to receive feedback and perhaps what people should expect from your prototype. Most importantly, provide the URL where to access your prototype.

If you want to provide longer explanations (more than 600 characters), your post will be trimmed and the rest will be hidden behind a "read more..." link. You can affect the place where the cut will be made by putting the HTML comment `<!--break-->` at the desired position. For this purpose, press the "Source" button in the upper right corner of the WYSIWYG editor's menu in order to modify the HTML source manually.

Store your data by clicking "Save". Your post will be published immediately.

If necessary, you can still modify your entry by navigating to it and clicking "Edit". Afterwards "Save" as before.

During the contest, all visitors of www.plat-forms.org can leave a comment on the entry.

6.4 Talking to the customer

The customer of the CaP project is represented by Ulrich Stärk. He will be more or less available for questions regarding the requirements during most of the day and evening. He will not be available between midnight and 8:00 in the morning.

He will happily answer questions regarding clarification of the meaning of requirements (but beware: if his advice and this document should ever be in conflict, it is this document that is relevant for the evaluation, not his advice). He will only vaguely answer questions regarding requirements priorities or regarding which of two concrete solution ideas he would prefer and will point you to this document instead. He will not look at your solutions or give concrete feedback about them.

6.5 Delivery

You are allowed to finish your development at any time you think appropriate, but no later than the end of the contest at 18:00 on 2011-01-19. Early delivery will be taken into account during the evaluation.

Package and submit your deliverables as follows:

1. Create a file named `adminuser.txt` containing two lines: the first line must be the login name of a user with unlimitied administrative rights for your virtual machine's operating system, the second line must be that user's password. We MUST_{200} be able to log in to your virtual machine using those credentials.

M 200

2. Shut down the virtual machine that is running your CaP service. Put all files that make up the virtual machine and the `adminuser.txt` file into a ZIP file called `vmware.zip`. The virtual machine MUST_{201} be set up in such a way that your CaP automatically starts up when the machine is booted and that your CaP does not need manual shutdown when the machine is shutdown. (The machine will always get at least 10 seconds of idle time before shutdown.) M 201
3. Package a snapshort of your versioning database into a ZIP file. The versioning database is the subtree in file system of your versioning server that contains the directories and files holding all versions of each file of your CaP project you have under version control. If you used CVS as your versioning system, call the ZIP file `cvs.zip`; if you used Subversion, call the ZIP file `svn.zip`; etc.
4. Create a source code distribution of your CaP implementation and package it in a ZIP file called `CaP-sources.zip`. This distribution should contain all files needed to recreate an instance of your CaP service except those that already existed before the contest and were not modified. So you should include all source code, build files, skripts, configuration files, documentation, etc, if they are new or were modified. You need not include pre-existing infrastructure such as application server, database server, compiler, libraries, unmodified parts of frameworks etc. The content of this ZIP file wil be considered published under the OpenSource license(s) you specified when you requested participation in the contest.
5. Create a ZIP file named like <homeorganization>-<platformname>.zip, e.g. Kayak-java.zip. This ZIP file contains the three other ZIP files mentioned above and is your deliverable (MUST_{202}). M 202
6. Determine the 160-bit SHA-1 message digest checksum of the deliverable. We will call this checksum the ”fingerprint” of the deliverable. SHA-1 is computed for instance by the sha1sum utility from GNU coreutils.
7. Send a two-line email containing the name of the deliverable and the fingerprint to organizers@plat-forms.org. The reception time timestamp of this email will be your submission time and MUST_{203} be before the end of the contest. Use the name of the deliverable as the subject of the email. The organizers will ignore all but the last such email from each team. M 203
8. Fill in the 12 questions in the file postmortem-questionnaire.rtf. We need a separate questionnaire from each member of your team. Since you may not have enough energy left to provide sensible answers today, you need not do this right now; it is OK if you send the filled-in questionnaire at any time until Friday next week (2010-01-28). However, the information you give us in the questionnaire is important input for the scientific evaluation, so we kindly ask that each team member thoroughly provides his/her own personal answers. Please send the result to organizers@plat-forms.org. Thanks!
9. Create a medium containing the deliverable. Write the name of the deliverable and the time of day on it with a pen. Hand the medium over to the customer (Ulrich Stärk). YOU ARE DONE. Come to our get-together, go and sleep or party or bang your head against a soft wall — whatever you feel most like doing. Thank you veeery much for participating in Plat_Forms!

The fingerprint sent in your email absolutely MUST_{204} match that of the ZIP file on the medium or else your whole participation was in vain. The fingerprint email has two purposes: (1) to make the submission timestamp independent of your medium-creation progress and (2) to validate the medium, in particular a replacement medium should the original one turn out to be unreadable. M 204