3 Functional requirements: Web Service interface

3.1 General information

- The web service interface consists of a single port type PbtServices and a single SOAP RPC binding for it.
- Its WSDL specification is available in the file `PbT.wsdl`.
- Your implementation of the service MUST$_{109}$ be bound to URL `/soap` on your server. M 109
- The WSDL of your final implementation MUST$_{110}$ be bound to URL `/wsdl` on your server. M 110
- With a single exception (clearDatabase, Section 3.3), 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.
- The service performs no elaborate exception handling. Rather, illegal calls or calls to operations you have not implemented MUST$_{111}$ simply return a nil result. M 111
- All of the WSDL specification MUST$_{112}$ be implemented, but only those parts of the underlying functionality actually need to 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$_{113}$ simply be ignored and output attributes that represent functionality you have not implemented on the usecase level MUST$_{114}$ simply be returned as nil. The one exception of this rule is the clearDatabase operation, which must be implenented in any case. M 112 M 113 M 114
- Beyond the elementary types and arrays thereof, there is only one single complex type (plus arrays thereof) that is used in the operations: Memberinfo represents the details about one member roughly as represented on the status page. See the WSDL for details.

The subsequent descriptions of the individual operations can only be fully understood in conjunction with `PbT.wsdl` (because only parts of the signatures are discussed here) and the usecases in Section 2 (because those describe most of the semantics).

3.2 complexType Memberinfo

The attributes of `Memberinfo` represent the data about a member that is (or may be) shown to other members in a member list or on a status page.

- The `rcdStatus` is one of the strings "no_contact", "RCD_sent", "RCD_received", or "in_contact" and MUST$_{115}$ be expressed from the point of view of the member that has requested the Memberinfo. A member has "in_contact" status relative to him/herself. M 115

- The contact details `fullname` and `emailAddress` will be nil unless `rcdStatus` is "in_contact"
- `gpsCoordinates` follows the format convention described in Section 2.2.3
- All other attributes should be self-explanatory.

3.3 operation clearDatabase

M 116 `clearDatabase` resets PbTs complete internal state to what it was just after the initial deployment: no users are registered, no current sessions or TTT results or RCDs exist (MUST$_{116}$).

This operation is needed to support the evaluation of the system, in particular load testing.

3.4 operations submitMemberinfo, getMemberinfo

M 117 `submitMemberinfo` is used for registration (MUST$_{117}$, in this case `sessionId` must be nil and `username` must not be previously used) and is also when a member modifies his/her information
M 118 via the status page (MUST$_{118}$, in this case `sessionId` must be a valid id as obtained by operation `login` and `username` must be nil).

All other parameters should be self-explanatory after reviewing the corresponding usecases in Sections 2.2 and 2.6.

M 119 `getMemberinfo` (MUST$_{119}$) obtains a `Memberinfo` object that contains the information stored by `submitMemberinfo` as described in Section .

3.5 operations login, logout

M 120 `login` (MUST$_{120}$) authenticates a member via `username` and `password`. If the authentication fails, the call returns nil. Otherwise, it returns a `sessionId` string that is used for identification/authentication of the *current user* in subsequent calls of other operations.

M 121 `logout` (MUST$_{121}$) invalidates a `sessionId` previously created by `login`.

3.6 operation takeTtt

M 122 `takeTtt` (MUST$_{122}$) realizes the core of usecase 2.3. It evaluates one set of answers to the TTT, computes the TTT result and TTT type, and stores them (plus a timestamp) for the current user.

`answers` is a string that contains one character for each question in the TTT (although not necessarily in that order). The character represents the answer given for that question. Assume character x represents the answer to a T/F question, then x is either T or F or blank; likewise for the other kinds of questions. Blank means the question has not been answered and will not be counted.