Module Interface Specification for Software Engineering

Team 14, Reach
Aamina Hussain
David Morontini
Anika Peer
Deep Raj
Alan Scott

January 11, 2024

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for REACH, a web application used to improve patients' access to clinical trials and practitioners' access to potential participants. More specifically, it will provide the list of modules that have been decomposed from the Module Guide, each with their interface specification, detailing important characteristics such as the module's methods and state variables.

Complementary documents, such as the System Requirement Specifications and Module Guide can be found at https://github.com/davimang/REACH.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Software Engineering.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of Software Engineering uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Software Engineering uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	Input Parameters Output Format Output Verification Temperature ODEs Energy Equations Control Module Specification Parameters Module
Software Decision	Sequence Data Structure ODE Solver Plotting

Table 1: Module Hierarchy

6 MIS of the User data module

6.1 Module

User

6.2 Uses

PatientInfo, Trial

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
getName	-	seq of char	-
setName	seq of char	-	EmptyName
getEmail	-	seq of char	-
setEmail	seq of char	-	InvalidEmail
getInfoProfiles	-	list of InfoProfile	-
getInfoProfile	integer	InfoProfile	InvalidInfoProfileId
addInfoProfile	InfoProfile	-	-
removeInfoProfile	integer	-	-
addTrial	Trial	-	-
removeTrial	integer	-	-
getTrials	-	list of Trial	-
getTrial	integer	Trial	-

6.4 Semantics

6.4.1 State Variables

name: seq of char
email: seq of char
infoProfiles: list of PatientInfo
trials: list of Trial

6.4.2 Environment Variables

None

6.4.3 Assumptions

- Each InfoProfile has a unique id.
- Each Trial has a unique id.

6.4.4 Access Routine Semantics

getName():

- transition: N/A
- output: out := self.name
- exception: N/A

setName(newName: seq of char):

- transition: self.name := newName
- output: N/A
- exception: $exc := length(newName) == 0 \Rightarrow EmptyName$

getEmail():

- transition: N/A
- output: out := self.email
- exception: N/A

setEmail(newEmail: seq of char):

- transition: self.email := newEmail
- output: N/A
- exception: $exc := isInvalidEmail(newEmail) \Rightarrow InvalidEmail$

getInfoProfiles():

- transition: N/A
- output: out := self.infoProfiles
- exception: N/A

getInfoProfile(id: integer):

- transition: N/A

- output: out := $\{\exists i \in self.infoProfiles | i.id = id\} \Rightarrow i$
- exception: exc := $\neg\{\exists i \in self.infoProfiles | i.id = id\} \Rightarrow InvalidInfoProfileId$

addInfoProfile(newInfoProfile: InfoProfile):

- transition: self.infoProfiles = self.infoProfiles + newInfoProfile
- output: N/A
- exception: N/A

removeInfoProfile(oldInfoProfile: InfoProfile):

- transition: self.infoProfiles = self.infoProfiles - oldInfoProfile
- output: N/A
- exception: N/A

6.4.5 Local Functions

7 MIS of the Fetch Trials Modules

7.1 Module

TrialFetcher

7.2 Uses

Trial

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
getTrials	seq. of String, integer, String	DataFrame	MissingParameter, InvalidAge, InvalidAddress
getLocator	-	geocoder	-
setLocator	geocoder	-	-

7.4 Semantics

7.4.1 State Variables

locator: geocoder

7.4.2 Environment Variables

None

7.4.3 Assumptions

- Each Trial has a unique id.
- The trial API will always be accessible.

7.4.4 Access Routine Semantics

getTrials(conditions: sequence of String, age: int, address: String):

- transition: None
- output: out := DataFrame populated with trials from the ClinicalTrials.gov API
- exception: $exc := (age \notin (0, 120] \rightarrow InvalidAge) \lor (\neg checkAddress(address) \rightarrow InvalidAddress) \lor ((\exists x.x \in parameters : x = \varepsilon) \rightarrow MissingParameter)$

7.4.5 Local Functions

convertTrialsToDataFrame(rawData: csv):

- transition: None
- output: out := rawData formatted as a DataFrame
- exception: None

checkAddress(address: String):

- transition: None
- output: out:= True
- exception: $exc := (geopy.geolocator(address) = exception \rightarrow False)$

8 MIS of the Trial Filtering Module

8.1 Module

TrialFilterer

8.2 Uses

Trial

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
exportTrials	-	json	-
fetchTrials	seq. of String, integer, String	-	-
getLocator	-	geocoder	-
setLocator	geocoder	-	-

8.4 Semantics

8.4.1 State Variables

locator: geocoder
trials: DataFrame

8.4.2 Environment Variables

None

8.4.3 Assumptions

- Each Trial has a unique id.
- Exceptions are caught downstream by the TrialFetcher module

8.4.4 Access Routine Semantics

fetchTrials(conditions: sequence of String, age: int, address: String):

- transition: self.trials populated with trials via TrialFilterer module
- output: None
- exception: None

exportTrials():

- transition: None
- output: out:= self.trials as json
- exception: None

getLocator():

- transition: None
- output: out:= self.locator
- exception: None

setLocator(loc: geocoder):

- transition: self.locator = loc
- output: None
- exception: None

8.4.5 Local Functions

cleanAge(stringAge: String):

- transition: None
- output: out:= $(inMonths \rightarrow int(stringAge)/12) \rightarrow int(stringAge)$
- exception: $exc := stringAge \notin \mathbb{R} \rightarrow InvalidAge$

geodesicDistance(address: geocode, trialLocation: geocode):

- transition: None
- output: $out := \arccos(\sin(address.latitude) \cdot \sin(trialLocation.latitude) + \cos(address.latitude) \cdot \cos(ltrialLocation.latitude) \cdot \cos(trialLocation.longitude - address.longitude)) \cdot 6371000$

- exception: None

calculateDistance():

- transition: $self.trials[distance] \mapsto geodesicDistance(address, self.trials[trialLocation])$
- output: None
- exception: None

convertToJSON(df: DataFrame):

- transition: None
- output: $df \rightarrow json(df)$
- exception: None

9 MIS of the Email Notification Module

9.1 Module

NotificationModule, User, PatientInfo, Trial

9.2 Uses

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
sendEmail	String, String	-	DeliveryFailed
getAPIKey	-	String	-
setAPIKey	String	-	-

9.4 Semantics

9.4.1 State Variables

APIKey: String

9.4.2 Environment Variables

connection: API connection

9.4.3 Assumptions

- Emailer API is operational and accessible
- ClinicalTrial.gov API is operational and accessible

9.4.4 Access Routine Semantics

sendEmail(emailAddress: String, subject: String, body: String):

- transition: None
- output: out := email request sent through the emailing API
- exception: exc := emailer returns error code $\rightarrow$ $DeliveryFailed$

getAPIKey():

- transition: None
- output: out := self.APIKey
- exception: None

setAPIKey(key: String):

- transition: self.APIKey := key
- output: None
- exception: None

9.4.5 Local Functions

findNewTrials():

- transition: None
- output: out := $\{trials : trial.postedDate > lastCheckedDate\}$
- exception: None

matchUsersToNewTrials():

- transition: None
- output: out := $\{user \times trial : user.conditions \subseteq trial.conditions\}$
- exception: None

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

10 Appendix

[Extra information if required —SS]