Module Interface Specification for Software Engineering

Team #11, OKKM Insights
Mathew Petronilho
Oleg Glotov
Kyle McMaster
Kartik Chaudhari

January 18, 2025

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 [https://github.com/OKKM-insights/OKKM.insights/tree/main/docs/SRS—SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

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 (-∞, ∞)
date	Date	provides a specific date and time

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 [Module Name —SS]

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

6.1 Module

[Short name for the module —SS]

6.2 Uses

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

6.4 Semantics

6.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

6.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

6.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

6.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

6.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

7 MIS of Account Creation Interface

7.1 Module

Account Creation Interface

7.2 Uses

Account Creation Controller 21

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	Enum[labeler, client]	-	-
submitForm	list[(string, string)]	-	-

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

win: 2D sequence of coloured pixels

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

renderPage(userType):

- transition: win := Modify window so that it shows a registration form that asks for the necessary information depending on if the user is a labeler or client.

submitForm(formData):

- transition: Passes the submitted form data to the Account Creation Controller for validation and processing.

7.4.5 Local Functions

None

8 MIS of Account Database Connector

8.1 Module

Account Database Connector

8.2 Uses

Account Database 14

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
insertUser	User	-	-
retrieveUser	string	User	-
updateUser	User	-	-
userExists	string	boolean	-
makeDBConnection	credentials		-

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

databaseConnection: connection to relational database

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

insertUser(user):

- transition: Request to insert user into database through databaseConection.

retrieveUser(email):

- output:
$$\begin{cases} \text{User where User.email == email}, & \text{if userExists(email)} \\ \text{null}, & \text{otherwise} \end{cases}$$

updateUser(user):

- transition:
$$\begin{cases} \text{Request to update user in database}, & \text{if userExists(user.email)} \\ \text{Do nothing} & \text{otherwise} \end{cases}$$

userExists(email):

- output: out :=
$$\exists\, \text{User} \in \text{Database}\, \text{s.t.}\, \text{User.email} == \text{email}$$

makeDBConnection(credentials):

- transition: databaseConnection := connection is established with database if credentials are correct

8.4.5 Local Functions

None

9 MIS of Account Database

9.1 Module

Account Database

9.2 Uses

None

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
insertUser	User	-	-
retrieveUser	string	User	-
updateUser	User	-	-
userExists	string	boolean	-

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

databaseConnection: connection to Application

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

insertUser(user):

- transition: Insert user into database.

retrieveUser(email):

- output:
$$\begin{cases} \text{User where User.email} == \text{email}, & \text{if userExists(email)} \\ \text{null}, & \text{otherwise} \end{cases}$$

updateUser(user):

- transition:
$$\begin{cases} \text{Update user in database}, & \text{if userExists(user.email)} \\ \text{Do nothing} & \text{otherwise} \end{cases}$$

userExists(email):

- output: out :=
$$\exists\, \text{User} \in \text{Database}\, \text{s.t.}\, \text{User.email} == \text{email}$$

9.4.5 Local Functions

None

10 MIS of Account Update Interface

10.1 Module

Account Update Interface

10.2 Uses

Account Update Controller 22

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	User	-	-
submitForm	list[(string, string)]	-	-

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

win: 2D sequence of coloured pixels

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

renderPage(userInfo):

- transition: win := Modify window so that it shows a form with the current user's information. This information can be changed by the user.

submitForm(formData):

- transition: Passes the submitted changes to the Account Update Controller for validation and processing.

10.4.5 Local Functions

None

11 MIS of Login Interface

11.1 Module

Login Interface

11.2 Uses

Authentication Controller 23

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
submitForm	list[(string, string)]	-	-

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

win: 2D sequence of coloured pixels

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

renderPage():

- transition: win := Modify window so that it shows a login form.

submitForm(formData):

- transition: Passes the submitted credentials to the Authentication Controller for validation.

11.4.5 Local Functions

None

12 MIS of Access Token

12.1 Module

Access Token

12.2 Uses

None

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
isExpired	-	boolean	-
renew	-	-	-

12.4 Semantics

12.4.1 State Variables

- tokenValue: string
- expirationTime: Date
- userID: string

12.4.2 Environment Variables

None

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

isExpired():

- output: out := currentTime > expirationTime

renew():

- transition: expirationTime := expirationTime + 5 hours

12.4.5 Local Functions

None

13 MIS of Account Creation Interface

13.1 Module

Account Creation Interface

13.2 Uses

Account Creation Controller 21

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	Enum[labeler, client]	-	-
submitForm	list[(string, string)]	-	-

13.4 Semantics

13.4.1 State Variables

None

13.4.2 Environment Variables

win: 2D sequence of coloured pixels

13.4.3 Assumptions

None

13.4.4 Access Routine Semantics

renderPage(userType):

- transition: win := Modify window so that it shows a registration form that asks for the necessary information depending on if the user is a labeler or client.

submitForm(formData):

- transition: Passes the submitted form data to the Account Creation Controller for validation and processing.

13.4.5 Local Functions

None

14 MIS of Account Database

14.1 Module

Account Database

14.2 Uses

Relational Database

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
insertUser	User	-	-
retrieveUser	string	User	-
updateUser	User	-	-
userExists	string	boolean	-

14.4 Semantics

14.4.1 State Variables

None

14.4.2 Environment Variables

databaseConnection: connection to relational database

14.4.3 Assumptions

None

14.4.4 Access Routine Semantics

insertUser(user):

- transition: Insert user into database through databaseConection.

retrieveUser(email):

- output:

$$\begin{cases} \text{User where User.email} == \text{email}, & \text{if userExists(email)} \\ \text{null}, & \text{otherwise} \end{cases}$$

updateUser(user):

- transition:

$$\begin{cases} \text{Update user in database through databaseConection,} & \text{if userExists(user.email)} \\ \text{Do nothing} & \text{otherwise} \end{cases}$$

userExists(email):

- output: out :=

$$\exists\, \text{User} \in \text{Database}\, \text{s.t.}\, \text{User.email} = \text{email}$$

14.4.5 Local Functions

None

15 MIS of Account Update Interface

15.1 Module

Account Update Interface

15.2 Uses

Account Update Controller 22

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	User	-	-
submitForm	list[(string, string)]	-	-

15.4 Semantics

15.4.1 State Variables

None

15.4.2 Environment Variables

win: 2D sequence of coloured pixels

15.4.3 Assumptions

None

15.4.4 Access Routine Semantics

renderPage(userInfo):

- transition: win := Modify window so that it shows a form with the current user's information. This information can be changed by the user.

submitForm(formData):

- transition: Passes the submitted changes to the Account Update Controller for validation and processing.

15.4.5 Local Functions

None

16 MIS of Login Interface

16.1 Module

Login Interface

16.2 Uses

Authentication Controller 23

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
submitForm	list[(string, string)]	-	-

16.4 Semantics

16.4.1 State Variables

None

16.4.2 Environment Variables

win: 2D sequence of coloured pixels

16.4.3 Assumptions

None

16.4.4 Access Routine Semantics

renderPage():

- transition: win := Modify window so that it shows a login form.

submitForm(formData):

- transition: Passes the submitted credentials to the Authentication Controller for validation.

16.4.5 Local Functions

None

17 MIS of Access Token

17.1 Module

Access Token

17.2 Uses

None

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
isExpired	-	boolean	-
renew	-	-	-

17.4 Semantics

17.4.1 State Variables

- tokenValue: string
- expirationTime: Date
- userID: string

17.4.2 Environment Variables

None

17.4.3 Assumptions

None

17.4.4 Access Routine Semantics

isExpired():

- output: out := currentTime > expirationTime

renew():

- transition: expirationTime := expirationTime + 5 hours

17.4.5 Local Functions

None

18 MIS of Labeler

18.1 Module

Labeler

18.2 Uses

Extends User 20

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
getFirstName	-	string	-
getLastName	-	string	-
getSkills	-	list[string]	-
getAvailability	-	int	-
setFirstName	string	-	-
setLastName	string	-	-
setSkills	list[string]	-	-
setAvailability	int	-	-

18.4 Semantics

18.4.1 State Variables

- firstName: string
- lastName: string
- skills: list[string]
- availability: int

18.4.2 Environment Variables

None

18.4.3 Assumptions

None

18.4.4 Access Routine Semantics

getFirstName():

- output: out := firstName

getLastName():

- output: out := lastName

getSkills():

- output: out := skills

getAvailability():

- output: out := availability

setFirstName(newfn):

- transition: firstName := newfn

setLastName(newln):

- transition: lastName := newln

setSkills(newSkills):

- transition: skills := newSkills

setAvailability(newAvail):

- transition: availability := newAvail

18.4.5 Local Functions

None

19 MIS of Client

19.1 Module

Client

19.2 Uses

Extends User 20

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
getCompanyName	-	string	-
getIndustry	-	string	-
getTypicalProject	-	Image	-
setCompanyName	string	-	-
setIndustry	string	-	-
setTypicalProject	string	-	-

19.4 Semantics

19.4.1 State Variables

- companyName: string
- industry: string
- typicalProject: string

19.4.2 Environment Variables

None

19.4.3 Assumptions

None

19.4.4 Access Routine Semantics

getCompanyName():

- output: out := companyName

getIndustry():

- output: out := industry

getTypicalProject():

- output: out := typicalProject

setCompanyName(newcn):

- transition: companyName := newcn

setIndustry(newIndustry):

- transition: industry := newIndustry

setTypicalProject(newtp):

- transition: typicalProject := newtp

19.4.5 Local Functions

None

20 MIS of User

20.1 Module

User

20.2 Uses

None

20.3 Syntax

20.3.1 Exported Constants

None

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
getEmail	-	string	-
getPassword	-	string	-
getProfilePic	-	Image	-
setEmail	string	-	-
setPassword	string	-	-
setProfilePic	string	-	-

20.4 Semantics

20.4.1 State Variables

- email: string
- password: string
- profilePic: image

20.4.2 Environment Variables

None

20.4.3 Assumptions

None

20.4.4 Access Routine Semantics

getEmail():

- output: out := email

getPassword():

- output: out := password

getProfilePic():

- output: out := profilePic

setEmail(newEmail):

- transition: email := newEmail

setPassword(newPassword):

- transition: password := newPassword

setProfilePic(newProfliePic):

- transition: profilePic := newProfilePic

20.4.5 Local Functions

None

21 MIS of Account Creation Controller

21.1 Module

Account Creation Controller

21.2 Uses

Account Creation Interface 13
Account Database 14
User 20
Labeler 18
Client 19

21.3 Syntax

21.3.1 Exported Constants

None

21.3.2 Exported Access Programs

Name	In	Out	Exceptions
validateForm	list[(string, string)], Enum[labeler, client]	boolean	-
createUser	list[(string, string)], Enum[labeler, client]	User	-
uploadUser	User	-	DatabaseException

21.4 Semantics

21.4.1 State Variables

None

21.4.2 Environment Variables

None

21.4.3 Assumptions

Assumes AccountDatabase is operational when calling uploadUser.

21.4.4 Access Routine Semantics

validateForm(formData, userType):

- output: $out := \text{hasRequiredFields}(formData, userFields) \wedge \text{isValidEmail}(formData.email) \wedge$

  $\text{isValidPassword}(formData.password) \wedge$

  $$\begin{cases} \text{hasRequiredFields}(formData, labelerFields), & \text{if userType} = \text{"labeler"} \\ \text{hasRequiredFields}(formData, clientFields), & \text{if userType} = \text{"client"} \\ \text{true}, & \text{otherwise} \end{cases}$$

  Where:

  $$\begin{aligned} userFields &= \{\text{email, password}\} \\ labelerFields &= \{\text{firstName, lastName, skills, availability}\} \\ clientFields &= \{\text{companyName, industry, typicalProject}\} \end{aligned}$$

createUser(formData, userType):

- output: out :=

$$\begin{cases} \text{Labeler}(\text{formData.email}, \text{formData.password}, \text{formData.firstName}, & \\ \text{formData.lastName}, \text{formData.skills}, \text{int}(\text{formData.availability})), & \text{if userType} = \text{"labeler"} \\ \text{Client}(\text{formData.email}, \text{formData.password}, \text{formData.companyName}, & \\ \text{formData.industry}, \text{formData.typicalProject}) & \text{if userType} = \text{"client"} \end{cases}$$

uploadUser(newUser):

- transition: Passes the User object to the AccountDatabase for storage.
- exception: Throws DatabaseException if storage fails.

21.4.5 Local Functions

- hasRequiredFields(data, fields) = $\forall$field $\in$ fields, (data[field] $\neq$ "")
- isValidEmail(email) = email $\in V \wedge$ email$\neg \in$ Registered Emails

  Let E represent the set of all email addresses, and let V represent the set of all valid email addresses. A valid email address conforms to the general pattern:

  V = ($\forall$ *email* $\in$ *E* | *email matches the pattern* [a-zA-Z0-9+_.-]+@[a-zA-Z0-9.-]+[a-zA-Z])

- isValidPassword(password) = (*password matches the pattern* (?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[#$%&])[a-zA-Z0-9#$%&]{8,})

22 MIS of Account Update Controller

22.1 Module

Account Update Controller

22.2 Uses

Account Update Interface 15
Account Database 14
User 20

22.3 Syntax

22.3.1 Exported Constants

None

22.3.2 Exported Access Programs

Name	In	Out	Exceptions
validateForm	list[(string, string)]	boolean	-
getUser	string	-	-
requestUpdate	User	-	DatabaseException

22.4 Semantics

22.4.1 State Variables

- user: User

22.4.2 Environment Variables

None

22.4.3 Assumptions

Assumes AccountDatabase is operational when calling requestUpdate.

22.4.4 Access Routine Semantics

validateForm(formData):

- output: out := $\forall$data $\in$ formData, (data[1] $\neq$ "")

getUser(email):

- transition: user := AccountDatabase.retreiveUser(email)

requestUpdate(updatedUser):

- transition: Passes the updated User object to the AccountDatabase for modifications.
- exception: Throws DatabaseException if storage fails.

22.4.5 Local Functions

None

23 MIS of Authentication Controller

23.1 Module

Authentication Controller

23.2 Uses

Login Interface 37
Account Database 14
Access Token 17

23.3 Syntax

23.3.1 Exported Constants

None

23.3.2 Exported Access Programs

Name	In	Out	Exceptions
validCredentials	(string, string)	boolean	-
generateAccessToken	string	-	-

23.4 Semantics

23.4.1 State Variables

- token: AccessToken

23.4.2 Environment Variables

None

23.4.3 Assumptions

Assumes AccountDatabase is operational when calling validCredentials.

23.4.4 Access Routine Semantics

validCredentials(email, password):

- output: out := AccountDatabase.retreiveUser(email) $\neq$ null
  $\wedge$ AccountDatabase.retreiveUser(email).getPassword() == password

generateAccessToken(email):

- transition: token := AccessToken(email)

23.4.5 Local Functions

None

24 MIS of Satellite Image Request Interface

24.1 Module

Satellite Image Request Interface

24.2 Uses

Satellite Image Request Controller 25

24.3 Syntax

24.3.1 Exported Constants

None

24.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
submitForm	list[(string, string)]	-	-

24.4 Semantics

24.4.1 State Variables

None

24.4.2 Environment Variables

win: 2D sequence of coloured pixels

24.4.3 Assumptions

None

24.4.4 Access Routine Semantics

renderPage():

- transition: win := Modify window so that it shows a form requesting information regarding an image request.

submitForm(formData):

- transition: Passes the submitted changes to the Satellite Image Request Controller for validation and processing.

24.4.5 Local Functions

None

25 MIS of Satellite Image Request Controller

25.1 Module

Satellite Image Request Controller

25.2 Uses

Satellite Image Request Interface 24
Satellite Image Request 26

25.3 Syntax

25.3.1 Exported Constants

None

25.3.2 Exported Access Programs

Name	In	Out	Exceptions
validateForm	list[(string, string)]	boolean	-
requestImages	SatelliteImageRequest	-	-

25.4 Semantics

25.4.1 State Variables

None

25.4.2 Environment Variables

None

25.4.3 Assumptions

None

25.4.4 Access Routine Semantics

validateForm(formData):

- output: out := $\forall$data $\in$ formData, (data[1] $\neq$ "")

requestImages(imgRequest):

- transition: Passes imgRequest to third party image provider to be processed.

25.4.5 Local Functions

- calculateCost(imgRequest): out := Use information given to calculate the cost of a request using third party rates

26 MIS of Satellite Image Request

26.1 Module

Satellite Image Request

26.2 Uses

None

26.3 Syntax

26.3.1 Exported Constants

None

26.3.2 Exported Access Programs

Name	In	Out	Exceptions
getLocation	-	(float, float)	-
getRadius	-	float	-
getDate	-	Date	-
setLocation	(float, float)	-	-
setRadius	float	-	-
setDate	Date	-	-

26.4 Semantics

26.4.1 State Variables

- locationX: float
- locationY: float
- radius: float
- date: Date

26.4.2 Environment Variables

None

26.4.3 Assumptions

None

26.4.4 Access Routine Semantics

getLocation():

- output: out := (locationX, locationY)

getRadius():

- output: out := radius

getDate():

- output: out := date

setLocation(x, y):

- transition: locationX, locationY := x, y

setRadius(newRadius):

- transition: radius := newRadius

setDate(newDate):

- transition: date := newDate

26.4.5 Local Functions

None

27 MIS of Project Creation Interface

27.1 Module

Project Creation Interface

27.2 Uses

Project Creation Controller 28

27.3 Syntax

27.3.1 Exported Constants

None

27.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
submitForm	list[(string, string)]	-	-

27.4 Semantics

27.4.1 State Variables

None

27.4.2 Environment Variables

win: 2D sequence of coloured pixels

27.4.3 Assumptions

None

27.4.4 Access Routine Semantics

renderPage():

- transition: win := Modify window so that it shows a form requesting information regarding creating a new project.

submitForm(formData):

- transition: Passes the submitted changes to the Project Creation Controller for validation and processing.

27.4.5 Local Functions

None

28 MIS of Project Creation Controller

28.1 Module

Project Creation Controller

28.2 Uses

Project Creation Interface 27
Project 29

28.3 Syntax

28.3.1 Exported Constants

None

28.3.2 Exported Access Programs

Name	In	Out	Exceptions
validateForm	list[(string, string)]	boolean	-
createNewProject	list[(string, string)]	Project	-

28.4 Semantics

28.4.1 State Variables

None

28.4.2 Environment Variables

None

28.4.3 Assumptions

None

28.4.4 Access Routine Semantics

validateForm(formData):

- output: out := $\forall$data $\in$ formData, (data[1] $\neq$ "")

createNewProject(formData):

- output: out := Project(formData.name, formData.description, formData.labelClasses.split(), Date(formData.startDate), Date(formData.endDate))

28.4.5 Local Functions

- calculateEstimatedCost(project): out := Use information given to calculate the estimated cost of a project.

29 MIS of Project

29.1 Module

Project

29.2 Uses

None

29.3 Syntax

29.3.1 Exported Constants

None

29.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProjectID	-	int	-
getName	-	string	-
getDescription	-	string	-
getLabelClasses	-	list[Enum[string]]	-
getTimePeriod	-	(Date, Date)	-
setName	string	-	-
setDescription	string	-	-
setLabelClasses	list[Enum[string]]	-	-
setTimePeriod	(Date, Date)	-	-

29.4 Semantics

29.4.1 State Variables

- projectID: int
- name: string
- description: string
- labelClasses: list[Enum[String]]
- startDate: Date
- endDate: Date

29.4.2 Environment Variables

None

29.4.3 Assumptions

None

29.4.4 Access Routine Semantics

getProjectID():

- output: out := projectID

getName():

- output: out := name

getDescription():

- output: out := description

getLabelClasses():

- output: out := labelClasses

getTimePeriod():

- output: out := (startDate, endDate)

setName(newName):

- transition: name := newName

setDescription(newDesc):

- transition: description := newDesc

setLabelClasses(newlc):

- transition: labelClasses := newlc

setTimePeriod(start, end):

- transition: startDate, endDate := start, end

29.4.5 Local Functions

None

30 MIS of Service Request Failure Interface

30.1 Module

Service Request Failure Interface

30.2 Uses

30.3 Syntax

30.3.1 Exported Constants

None

30.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayErrorInfo	-	-	-

30.4 Semantics

30.4.1 State Variables

None

30.4.2 Environment Variables

win: 2D sequence of coloured pixels

30.4.3 Assumptions

None

30.4.4 Access Routine Semantics

displayErrorInfo():

- transition: win := Modify window so that it shows a warning to the user that their request has failed.

30.4.5 Local Functions

None

31 MIS of Image Upload Interface

31.1 Module

Image Upload Interface

31.2 Uses

31.3 Syntax

31.3.1 Exported Constants

None

31.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayUploadImages		-	-

31.4 Semantics

31.4.1 State Variables

None

31.4.2 Environment Variables

win: 2D sequence of coloured pixels

31.4.3 Assumptions

None

31.4.4 Access Routine Semantics

displayUploadImages():

- transition: win := Modify window so that it allows users to upload images.

31.4.5 Local Functions

- validateImage(image): out :=

$$\text{image.extension} \in \{\text{svg}, \text{jpeg}, \text{png}\}$$

32 MIS of Report Interface

32.1 Module

Report Interface

32.2 Uses

Report Controller 33

32.3 Syntax

32.3.1 Exported Constants

None

32.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayStats	-	-	-

32.4 Semantics

32.4.1 State Variables

None

32.4.2 Environment Variables

win: 2D sequence of coloured pixels

32.4.3 Assumptions

None

32.4.4 Access Routine Semantics

displayStats():

- transition: win := Modify window so that it shows project specific statistics.

32.4.5 Local Functions

None

33 MIS of Report Controller

33.1 Module

Report Controller

33.2 Uses

Report Interface 32
Report 34

33.3 Syntax

33.3.1 Exported Constants

None

33.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProjectStats	string	-	-
exportLabeledImages	-	-	-

33.4 Semantics

33.4.1 State Variables

- report: Report

33.4.2 Environment Variables

fm: External systems file manager

33.4.3 Assumptions

None

33.4.4 Access Routine Semantics

getProjectStats(projectID):

- transition: report := Report of statistics for project with projectID

exportLabeledImages():

- transition: fm := given labeled images to download to device.

33.4.5 Local Functions

None

34 MIS of Report

34.1 Module

Report

34.2 Uses

None

34.3 Syntax

34.3.1 Exported Constants

None

34.3.2 Exported Access Programs

Name	In	Out	Exceptions
getLabeledImages	-	list[Image]	-
getReviewedImages	-	list[Image]	-
getEndDate	-	Date	-
getTotalLabelers	-	int	-
getAccuracy	-	int	-
getClassCount	-	list[(string, int)]	-

34.4 Semantics

34.4.1 State Variables

- labeledImages: list[Image]
- reviewedImages: list[Image]
- endDate: Date
- totalLabelers: int
- accuracyOfLabelers: int
- classCount: list[(string, int)]

34.4.2 Environment Variables

None

34.4.3 Assumptions

None

34.4.4 Access Routine Semantics

getLabeledImages():

- output: out := labeledImages

getReviewedImages():

- output: out := reviewedImages

getEndDate():

- output: out := endDate

getTotalLabelers():

- output: out := totalLabelers

getAccuracyOfLabelers():

- output: out := accuracyOfLabelers

getClassCount():

- output: out := classCount

34.4.5 Local Functions

None

35 MIS of Project Selection Interface

35.1 Module

Project Selection Interface

35.2 Uses

Project Selection Controller 36

35.3 Syntax

35.3.1 Exported Constants

None

35.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayActiveProjects	-	-	-

35.4 Semantics

35.4.1 State Variables

None

35.4.2 Environment Variables

win: 2D sequence of coloured pixels

35.4.3 Assumptions

None

35.4.4 Access Routine Semantics

displayActiveProjects():

- transition: win := Modify window so that it shows all active projects and a small description of each.

35.4.5 Local Functions

None

36 MIS of Project Selection Controller

36.1 Module

Project Selection Controller

36.2 Uses

Project Selection Interface 35
Project 29

36.3 Syntax

36.3.1 Exported Constants

None

36.3.2 Exported Access Programs

Name	In	Out	Exceptions
getActiveProjects	-	-	-
selectProject	Project	-	-

36.4 Semantics

36.4.1 State Variables

- activeProjects: list[Project]

36.4.2 Environment Variables

win: 2D sequence of coloured pixels

36.4.3 Assumptions

None

36.4.4 Access Routine Semantics

getActiveProjects():

- transition: activeProjects := All projects marked as active in the project database

selectProject(project):

- transition: win := redirects users to labeling interface of that project

36.4.5 Local Functions

None

37 MIS of Labeling Interface

37.1 Module

Labeling Interface

37.2 Uses

Labeling Controller 38
Image 39

37.3 Syntax

37.3.1 Exported Constants

None

37.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
displayImage	Image	-	-
skipImage	-	-	-
selectLabelClass	-	-	-

37.4 Semantics

37.4.1 State Variables

- projectImages: list[Image]
- currImage: int
- currLabelClass: Enum[string]

37.4.2 Environment Variables

win: 2D sequence of coloured pixels

37.4.3 Assumptions

None

37.4.4 Access Routine Semantics

renderPage():

- transition: win := Modify window so that it shows labeling tools along with a picture to label.

displayImage(img):

- transition: win := Modify window so that the picture it is showing is img.

skipImage():

- transition: currentImage := (currentImage + 1) % projectImages.length
  win := Modify window so that the picture it is showing is projectImages[currentImage].

selectLabelClass():

- transition: currLabelClass := the label class the user has selected on win.

37.4.5 Local Functions

None

38 MIS of Labeling Controller

38.1 Module

Labeling Controller

38.2 Uses

Labeling Interface 37
Label **??**

38.3 Syntax

38.3.1 Exported Constants

None

38.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProjectImages	string	-	-
addLabel	Label	-	-
removeLabel	string	-	-
submitLabels	list[Label]	-	-

38.4 Semantics

38.4.1 State Variables

- labels: list[Label]

38.4.2 Environment Variables

None

38.4.3 Assumptions

None

38.4.4 Access Routine Semantics

getProjectImages(projectID):

- output: out := All images from project with projectID

addLabel(lbl):

- transition: labels := labels $\cup$ {lbl}

removeLabel(lblID):

- transition: labels := $\{\ell \in \text{labels} \mid \ell.\text{id} \neq \text{lblID}\}$

submitLabels(lbls):

- transition: labels are sent to be added to the Label Database

38.4.5 Local Functions

None

39 MIS of Image

39.1 Module

Image

39.2 Uses

None

39.3 Syntax

39.3.1 Exported Constants

None

39.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProjectID	-	int	-
getImageID	-	int	-
getDimensions	-	(float, float)	-
getImageData	-	binary	-

39.4 Semantics

39.4.1 State Variables

- projectID: int
- imageID: int
- width: float
- height: float
- imageData: binary

39.4.2 Environment Variables

None

39.4.3 Assumptions

None

39.4.4 Access Routine Semantics

getProjectID():

- output: out := projectID

getImageID():

- output: out := imageID

getDimensions():

- output: out := (width, height)

getImageData():

- output: out := imageData

39.4.5 Local Functions

None

40 MIS of Label Server

40.1 Module

Label Server

40.2 Uses

Labeling Controller 38
Label **??**
Label Database Connector 41

40.3 Syntax

40.3.1 Exported Constants

None

40.3.2 Exported Access Programs

Name	In	Out	Exceptions
acceptLabel	Label	-	ValueError, ConnectionError

40.4 Semantics

40.4.1 State Variables

None

40.4.2 Environment Variables

LabelDatabaseConnector

40.4.3 Assumptions

Label Objects are given to the label server in JSON format. Exceptions will be thrown based on failure to match this standard.

40.4.4 Access Routine Semantics

acceptLabel(object o):

- transition: Transition occurs in LabelDatabaseConnector

- output: Standard HTTP response codes
- exception: Let L be the set of valid Labels. Throw ValueError if $\neg(o \in \mathrm{L})$
  Throw ConnectionError if ConnectionError is raised by LabelDatabaseConnector

40.4.5 Local Functions

JSONLabeltoLabel: converts a JSON object into a Label object.

41 MIS of Label Database Connector

41.1 Module

Label Database Connector

41.2 Uses

Label Database 42
Label **??**

41.3 Syntax

41.3.1 Exported Constants

None

41.3.2 Exported Access Programs

Name	In	Out	Exceptions
pushLabel	Label	-	ValueError, ConnectionError
makeDB Connection	Label	-	ConnectionError
getLabels	String	list[Label]	ValueError, ConnectionError

41.4 Semantics

41.4.1 State Variables

None

41.4.2 Environment Variables

None

41.4.3 Assumptions

41.4.4 Access Routine Semantics

pushLabel(Label l):

- transition: Transition occurs in LabelDatabase
- exception: Let L be the set of valid Labels. Throw ValueError if $\neg(l \in \mathrm{L})$
  Throw ConnectionError if ConnectionError is raised by makeDBConnection

makeDBConnection():

- transition: If sucessful, connection occurs
- exception: Throw ConnectionError if connection is not accepted by LabelDatabase

getLabels(String q):

- output: list of labels satisfying the provided query
- exception: Let Q be the set of valid Queries. Throw ValueError if $\neg(q \in \mathrm{Q})$
  Throw ConnectionError if ConnectionError is raised by makeDBConnection

41.4.5 Local Functions

None

42 MIS of Label Database

42.1 Module

Label Database

42.2 Uses

None

42.3 Syntax

42.3.1 Exported Constants

None

42.3.2 Exported Access Programs

Name	In	Out	Exceptions
pushLabel	Label	-	ValueError
makeDB Connection	Label	-	ConnectionError
getLabels	String	list[Label]	ValueError

42.4 Semantics

42.4.1 State Variables

labels: labels stored in the database users: list of authenticated users

42.4.2 Environment Variables

None

42.4.3 Assumptions

42.4.4 Access Routine Semantics

pushLabel(Label l):

- transition: labels := labels $\cup\, l$
- exception: Let L be the set of valid Labels. Throw ValueError if $\neg(l \in \text{L})$ Throw ConnectionError if $\neg$(requestor $\in$ users)

makeDBConnection(credentials):

- transition: if credentials are valid, users := users $\cup$ *credentials.user*
- exception: Throw ConnectionError if credentials are not valid

getLabels(String q):

- output: list of labels satisfying the provided query
- exception: Let Q be the set of valid Queries. Throw ValueError if $\neg(q \in Q)$ Throw ConnectionError if $\neg$(requestor $\in$ users)

42.4.5 Local Functions

None

43 MIS of ImageObject Database Connector

43.1 Module

ImageObject Database Connector

43.2 Uses

ImageObject Database 44
ImageObject **??**

43.3 Syntax

43.3.1 Exported Constants

None

43.3.2 Exported Access Programs

Name	In	Out	Exceptions
push Image Object	ImageObject	-	ValueError, ConnectionError
makeDB Connection	ImageObject	-	ConnectionError
get Image Objects	String	list[ImageObject]	ValueError, ConnectionError

43.4 Semantics

43.4.1 State Variables

None

43.4.2 Environment Variables

None

43.4.3 Assumptions

43.4.4 Access Routine Semantics

pushLabel(ImageObject l):

- transition: Transition occurs in ImageObjectDatabase
- exception: Let L be the set of valid ImageObjects. Throw ValueError if $\neg(l \in \mathrm{L})$ Throw ConnectionError if ConnectionError is raised by makeDBConnection

makeDBConnection():

- transition: If sucessful, connection occurs
- exception: Throw ConnectionError if connection is not accepted by ImageObject-Database

getLabels(String q):

- transition:
- output: list of ImageObjects satisfying the provided query
- exception: Let Q be the set of valid Queries. Throw ValueError if $\neg(q \in \mathrm{Q})$ Throw ConnectionError if ConnectionError is raised by makeDBConnection

43.4.5 Local Functions

None

44 MIS of ImageObject Database

44.1 Module

ImageObject Database

44.2 Uses

None

44.3 Syntax

44.3.1 Exported Constants

None

44.3.2 Exported Access Programs

Name	In	Out	Exceptions
push Image Object	ImageObject	-	ValueError
makeDB Connection	ImageObject	-	ConnectionError
get Image Objects	String	list[ImageObject]	ValueError

44.4 Semantics

44.4.1 State Variables

ImageObjects: ImageObjects stored in the database users: list of authenticated users

44.4.2 Environment Variables

None

44.4.3 Assumptions

44.4.4 Access Routine Semantics

pushLabel(ImageObject l):

- transition: ImageObjects := ImageObjects $\cup\, l$
- exception: Let L be the set of valid ImageObjects. Throw ValueError if $\neg(l \in \text{L})$ Throw ConnectionError if $\neg$(requestor $\in$ users)

makeDBConnection(credentials):

- transition: if credentials are valid, users := users $\cup$ $credentials.user$
- exception: Throw ConnectionError if credentials are not valid

getLabels(String q):

- output: list of ImageObjects satisfying the provided query
- exception: Let Q be the set of valid Queries. Throw ValueError if $\neg(q \in \text{Q})$ Throw ConnectionError if $\neg$(requestor $\in$ users)

44.4.5 Local Functions

None

=======

45 MIS of Labeller Database Connector

45.1 Module

Labeller Database Connector

45.2 Uses

Labeller Database **??**
ImageObject **??**

45.3 Syntax

45.3.1 Exported Constants

None

45.3.2 Exported Access Programs

Name	In	Out	Exceptions
push labeller	labeller	-	ValueError, ConnectionError
makeDB Connection	credentials	-	ConnectionError
get labeller	String	list[labeller]	ValueError, ConnectionError

45.4 Semantics

45.4.1 State Variables

None

45.4.2 Environment Variables

None

45.4.3 Assumptions

45.4.4 Access Routine Semantics

pushLabeller(Labeller o):

- transition: Transition occurs in LabellerDatabase
- exception: Let O be the set of valid Labellers. Throw ValueError if $\neg(o \in \mathrm{O})$
  Throw ConnectionError if ConnectionError is raised by makeDBConnection

makeDBConnection():

- transition: If sucessful, connection occurs
- exception: Throw ConnectionError if connection is not accepted by LabellerDatabase

getLabeller(String q):

- output: list of Labellers satisfying the provided query
- exception: Let Q be the set of valid Queries. Throw ValueError if $\neg(q \in \mathrm{Q})$
  Throw ConnectionError if ConnectionError is raised by makeDBConnection

45.4.5 Local Functions

None

46 MIS of Labeller Database

46.1 Module

Labeller Database

46.2 Uses

None

46.3 Syntax

46.3.1 Exported Constants

None

46.3.2 Exported Access Programs

Name	In	Out	Exceptions
push Labeller	Labeller	-	ValueError
makeDB Connection	Credentials	-	ConnectionError
get Labeller	String	list[Labeller]	ValueError

46.4 Semantics

46.4.1 State Variables

Labellers: Labellers stored in the database users: list of authenticated users

46.4.2 Environment Variables

None

46.4.3 Assumptions

46.4.4 Access Routine Semantics

pushLabeller(Labeller o):

- transition: Labellers := Labellers $\cup$ o
- exception: Let O be the set of valid Labellers. Throw ValueError if $\neg(o \in \text{O})$ Throw ConnectionError if $\neg$(requestor $\in$ users)

makeDBConnection(credentials):

- transition: if credentials are valid, users := users $\cup$ $credentials.user$
- exception: Throw ConnectionError if credentials are not valid

getLabeller(String q):

- output: list of Labeller satisfying the provided query
- exception: Let Q be the set of valid Queries. Throw ValueError if $\neg(q \in \text{Q})$ Throw ConnectionError if $\neg$(requestor $\in$ users)

46.4.5 Local Functions

None

47 MIS of Object Extraction Manager

47.1 Module

Object Extraction Manager

47.2 Uses

ImageObject Database Connector 43
Label Database Connector 41
Labeller Database Connector 45
Image Prior Analyzer 50
Label Confidence Service 48
Object Extraction Service 49
Labeller Expertise Calculator 51

47.3 Syntax

47.3.1 Exported Constants

None

47.3.2 Exported Access Programs

Name	In	Out	Exceptions
getObjects	projectID	-	ValueError

47.4 Semantics

47.4.1 State Variables

None

47.4.2 Environment Variables

None

47.4.3 Assumptions

47.4.4 Access Routine Semantics

getObjects(ProjectID p):

- transition: Updates ImageObject database with identified objects & confidence and updates labeller expertise rating in labeller database

- exception: Let P be the set of assigned ProjectIDs. Throw ValueError if $\neg(p \in \mathrm{P})$

47.4.5 Local Functions

generate query:

48 MIS of Label Confidence Service

48.1 Module

Label Confidence Service

48.2 Uses

None

48.3 Syntax

48.3.1 Exported Constants

None

48.3.2 Exported Access Programs

Name	In	Out	Exceptions
getConfidence	list[label], list[labeller], list[ImageObject]	list[list[float]]	ValueError

48.4 Semantics

48.4.1 State Variables

None

48.4.2 Environment Variables

None

48.4.3 Assumptions

48.4.4 Access Routine Semantics

getConfidence(list[label] labels , list[labeller] labellers, list[ImageObject] imageobjects):

- output: return the confidence label of each extracted object
- exception: Let L be the set of valid Labels. Throw ValueError if ($\exists$label $\in$ labels| : $\neg$(label $\in$ L))
  Let X be the set of valid Labellers. Throw ValueError if ($\exists$labeller $\in$ labellers| : $\neg$(labeller $\in$ X))
  Let I be the set of valid ImageObjects. Throw ValueError if ($\exists$imageobject $\in$ imageobjects| :

$\neg$(imageobject $\in$ I))

48.4.5 Local Functions

49 MIS of Object Extraction Service

49.1 Module

Object Extraction Service

49.2 Uses

None

49.3 Syntax

49.3.1 Exported Constants

None

49.3.2 Exported Access Programs

Name	In	Out	Exceptions
getObjects	list[label], list[labeller], list[ImageObject], list[list[float]]	list[ImageObject]	ValueError

49.4 Semantics

49.4.1 State Variables

None

49.4.2 Environment Variables

None

49.4.3 Assumptions

49.4.4 Access Routine Semantics

getConfidence(list[label] labels, list[labeller] labellers, list[ImageObject] imageobjects, list[list[float]] confidence):

- output: returns a list of extracted image objects
- exception: Let L be the set of valid Labels. Throw ValueError if ($\exists$label $\in$ labels| : $\neg$(label $\in$ L))
  Let X be the set of valid Labellers. Throw ValueError if ($\exists$labeller $\in$ labellers| :

$\neg$(labeller $\in$ X))
Let I be the set of valid ImageObjects. Throw ValueError if ($\exists$imageobject $\in$ imageobjects| : $\neg$(imageobject $\in$ I))
Throw ValueError if ($\exists i, j | x =$ confidence[i][j] : $\neg(x \in \mathbb{R})$)

49.4.5 Local Functions

50 MIS of Image Prior Analyzer

50.1 Module

Image Prior Analyzer

50.2 Uses

None

50.3 Syntax

50.3.1 Exported Constants

None

50.3.2 Exported Access Programs

Name	In	Out	Exceptions
getPriors	list[image]	list[list[float]]	ValueError

50.4 Semantics

50.4.1 State Variables

None

50.4.2 Environment Variables

None

50.4.3 Assumptions

50.4.4 Access Routine Semantics

getPriors(list[image] Images):

- output: returns a list of priors for each pixel in the given images
- exception: Let I be the set of valid Images. Throw ValueError if ($\exists$image $\in$ images$|$: $\neg$(image $\in$ I))

50.4.5 Local Functions

51 MIS of Labeller Expertise Calculator

51.1 Module

Labeller Expertise Calculator

51.2 Uses

None

51.3 Syntax

51.3.1 Exported Constants

None

51.3.2 Exported Access Programs

Name	In	Out	Exceptions
getExpertise	list[label], list[labeller], list[ImageObject], list[list[float]]	list[dict[string, tuple[float, float]]]	ValueError

51.4 Semantics

51.4.1 State Variables

None

51.4.2 Environment Variables

None

51.4.3 Assumptions

51.4.4 Access Routine Semantics

getObjects(list[label] labels, list[labeller] labellers, list[ImageObject] imageobjects):

- output: return the weighed success rate for each class a labeler has contributed to
- exception: Let L be the set of valid Labels. Throw ValueError if ($\exists$label $\in$ labels| : $\neg$(label $\in$ L))
  Let X be the set of valid Labellers. Throw ValueError if ($\exists$labeller $\in$ labellers| : $\neg$(labeller $\in$ X))

Let I be the set of valid ImageObjects. Throw ValueError if ($\exists$imageobject $\in$ imageobjects$|$: $\neg$(imageobject $\in$ I))
Throw ValueError if ($\exists i, j | x =$ confidence[i][j] : $\neg(x \in \mathbb{R})$)

51.4.5 Local Functions

52 MIS of Image Service Manager

52.1 Module

Image Service Manager

52.2 Uses

ImageObject Database Connector 43
Labeller Database Connector 45
Image Mask Service53
Image Selection Engine **??**

52.3 Syntax

52.3.1 Exported Constants

None

52.3.2 Exported Access Programs

Name	In	Out	Exceptions
getNextImages	labellerID, projectID, int	List[Image]	ValueError

52.4 Semantics

52.4.1 State Variables

None

52.4.2 Environment Variables

None

52.4.3 Assumptions

52.4.4 Access Routine Semantics

getNextImages(LabellerID l, ProjectID p, int n):

- output: return the next n images based on which are mose relevant
- exception: Let P be the set of assigned ProjectIDs. Throw ValueError if $\neg(p \in \mathrm{P})$
  Let L be the set of assigned LabellerIDs. Throw ValueError if $\neg(l \in \mathrm{L})$
  Throw ValueError if $\neg(n \in \mathbb{N})$

52.4.5 Local Functions

53 MIS of Image Mask Service

53.1 Module

Image Mask Service

53.2 Uses

None

53.3 Syntax

53.3.1 Exported Constants

None

53.3.2 Exported Access Programs

Name	In	Out	Exceptions
getImageMask	Image	Image	ValueError

53.4 Semantics

53.4.1 State Variables

None

53.4.2 Environment Variables

None

53.4.3 Assumptions

53.4.4 Access Routine Semantics

getImageMask(Image i):

- output: returns a modified image to improve the labeller's efficiency or accuracy
- exception: Let I be the set of valid Images. Throw ValueError if $\neg(i \in \text{I})$

53.4.5 Local Functions

54 MIS of Image Selection Service

54.1 Module

Image Selection Service

54.2 Uses

54.3 Syntax

54.3.1 Exported Constants

None

54.3.2 Exported Access Programs

Name	In	Out	Exceptions
getNextImages	List[Image], List[ImageObjects], Labeller	List[Image]	ValueError

54.4 Semantics

54.4.1 State Variables

None

54.4.2 Environment Variables

None

54.4.3 Assumptions

54.4.4 Access Routine Semantics

getNextImages(List[Image] Images, List[ImageObjects] ImageObjects, Labeller labeller):

- output: return the next n images based on which are mose relevant
- exception: Let L be the set of valid Labellers. Throw ValueError if ($\neg$(labeller $\in$ L))
  Let X be the set of valid Images. Throw ValueError if ($\exists$Image $\in$ Images$|$: $\neg$(Image $\in$ X))
  Let I be the set of valid ImageObjects. Throw ValueError if ($\exists$imageobject $\in$ imageobjects$|$: $\neg$(imageobject $\in$ I))

54.4.5 Local Functions

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.

55 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable? Everyone did a great job contributing their ideas and expertise to design each part of our application. We decided to use diagrams to express our designs before jumping into the documentation. This worked really well as it allowed everyone to have a better understanding of how our system would interact. When we had to specify our modules, a lot of the hard work was already complete due to have the diagrams.

2. What pain points did you experience during this deliverable, and how did you resolve them? A major pain point we faced was that a team member could no longer meet in person due to extraneous circumstances. This hindered our ability to effectively communicate as a team due to factors like time difference. To solve this, we rescheduled our meetings to a reasonable time for all members, and moved all meetings and communications online for the time being.

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from? From talking to our supervisor, we determined that we would need our system to be able to pre-process images in an effective way so we took that into considering when designing the project creation subsystem. Also, our decision to have modules do standardized formatting stemmed from our usage of 3rd-party applications such as our image distributor. Due to the possibility of change, we knew that formatting outside information to a way our application could process it would be the best way to go about it. In general, for our other decisions we used the software principles we have learned through out our education including modularity, seperation of concern, and architecture that supports scalability.

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why? When creating this design

document, we realized some functionality we want is not really specified much in the srs. For example, we have very little regarding the ai model part of our application. We also realized some of the requirements that we will not be able to focus on, such as the financial aspect of the app. We now must consider how to document what we need to in the srs, and possibly modify our vnv plan.

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions) One major focus would be to add in the financial capabilities like accepting payments for projects and distributing funds to users. Another thing would be to expand the platform to accept lots of different types of media, such as videos, instead of just images.

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? We considered an approach where we would store all data as files on a server somewhere, however we decided that using a database would better fit our project due to the relation between the data and the usefulness of SQL statements. The downside of this approach is it requires more time to set up, but we believe the payoff is worth it. We also considered having one large system rather than many sub-systems. This would elimate alot of the communication and data transfer overhead. However, we believe that with this sub-system design, we have the ability to have or remove parts of the system much more easily. If we dont have time to get to a sub-system, our application can still function. (LO_Explores)