Module Interface Specification for Software Engineering

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

April 3, 2025

1 Revision History

Date	Version	Notes
Date January 17th	1.0	
Date April 1st	2.0	

2 Symbols, Abbreviations and Acronyms

See SRS Documentation here

Contents

1	Rev	rision I	History	i
2	Syn	nbols,	Abbreviations and Acronyms	ii
3	Intr	oducti	ion	1
4	Not	ation		1
5	Mo	dule D	ecomposition	8
6	MIS	of Re	eport Manager	10
	6.1	Modul	le	. 10
	6.2	Uses		. 10
	6.3		X	
		6.3.1	Exported Constants	
		6.3.2	Exported Access Programs	
	6.4		atics	
	0.1	6.4.1	State Variables	
		6.4.2	Environment Variables	
		6.4.3	Assumptions	
		6.4.4	Access Routine Semantics	
		6.4.5	Local Functions	
		0.1.0	Book Functions 1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.	
7	MIS	of Ac	ccount Creation Interface	11
	7.1	Modul	le	. 11
	7.2	Uses		. 11
	7.3	Syntax	x	. 12
		7.3.1	Exported Constants	. 12
		7.3.2	Exported Access Programs	
	7.4	Seman	ntics	
		7.4.1	State Variables	. 12
		7.4.2	Environment Variables	. 12
		7.4.3	Assumptions	
		7.4.4	Access Routine Semantics	
		7.4.5	Local Functions	
8	MIS		roject Manager	12
	8.1		le	
	8.2	Uses		
	8.3	Syntax	x	. 13
		8.3.1	Exported Constants	. 13
		8.3.2	Exported Access Programs	. 13

	8.4	Seman	ntics	13
		8.4.1	State Variables	13
		8.4.2	Environment Variables	13
		8.4.3	Assumptions	13
		8.4.4	Access Routine Semantics	13
		8.4.5	Local Functions	14
9	MIS	of Pr	roject Collection Manager	1 4
	9.1		le	14
	9.2			14
	9.3	Syntax	x	14
		9.3.1	Exported Constants	14
		9.3.2	Exported Access Programs	14
	9.4	Seman	ntics	14
		9.4.1	State Variables	14
		9.4.2	Environment Variables	14
		9.4.3	Assumptions	14
		9.4.4	Access Routine Semantics	15
		9.4.5	Local Functions	15
10	MIS	of Pr	roject Database Connector	15
			le	15
	10.2	Uses		15
	10.3	Syntax	x	15
		10.3.1	Exported Constants	15
		10.3.2	Exported Access Programs	15
	10.4	Seman	ntics	15
		10.4.1	State Variables	15
		10.4.2	Environment Variables	15
		10.4.3	Assumptions	16
		10.4.4	Access Routine Semantics	16
		10.4.5	Local Functions	16
11	MIS	of Co	ore Image Database Connector	16
	11.1	Modul	le	16
	11.2	Uses		16
	11.3	Syntax	X	16
		11.3.1	Exported Constants	16
			Exported Access Programs	17
	11.4		ntics	17
			State Variables	17
			Environment Variables	17
			Assumptions	17

	11.4.4 Access Routine Semantics	17
	11.4.5 Local Functions	18
12 MI	S of Account Database Connector	18
	1 Module	18
12.5	2 Uses	18
	3 Syntax	18
	12.3.1 Exported Constants	18
	12.3.2 Exported Access Programs	18
12.4	4 Semantics	18
	12.4.1 State Variables	18
	12.4.2 Environment Variables	18
	12.4.3 Assumptions	18
	12.4.4 Access Routine Semantics	19
	12.4.5 Local Functions	19
	S of Account Database	19
	1 Module	19
	2 Uses	19
13.	3 Syntax	20
	13.3.1 Exported Constants	20
	13.3.2 Exported Access Programs	20
13.4	4 Semantics	20
	13.4.1 State Variables	20
	13.4.2 Environment Variables	20
	13.4.3 Assumptions	20
	13.4.4 Access Routine Semantics	20
	13.4.5 Local Functions	21
14 MI	S of Account Update Interface	21
	1 Module	21
	2 Uses	21
	3 Syntax	21
14.	14.3.1 Exported Constants	$\frac{21}{21}$
	•	21
1.4	14.3.2 Exported Access Programs	
14.4	4 Semantics	21
	14.4.1 State Variables	21
	14.4.2 Environment Variables	21
	14.4.3 Assumptions	21
	14.4.4 Access Routine Semantics	21
	14.45 Local Functions	22

15	MIS of Login Interface
	15.1 Module
	15.2 Uses
	15.3 Syntax
	15.3.1 Exported Constants
	15.3.2 Exported Access Programs
	15.4 Semantics
	15.4.1 State Variables
	15.4.2 Environment Variables
	15.4.3 Assumptions
	15.4.4 Access Routine Semantics
	15.4.5 Local Functions
	10.4.9 Local Pulicuous
16	MIS of Access Token
	16.1 Module
	16.2 Uses
	16.3 Syntax
	16.3.1 Exported Constants
	16.3.2 Exported Access Programs
	16.4 Semantics
	16.4.1 State Variables
	16.4.2 Environment Variables
	1
	16.4.4 Access Routine Semantics
	16.4.5 Local Functions
17	MIS of Account Creation Interface
	17.1 Module
	17.2 Uses
	17.3 Syntax
	17.3.1 Exported Constants
	17.3.2 Exported Access Programs
	17.4 Semantics
	17.4.1 State Variables
	17.4.1 State Variables
	1
	17.4.4 Access Routine Semantics
	17.4.5 Local Functions
18	MIS of Account Database
10	18.1 Module
	18.2 Uses
	18.3 Syntax

		18.3.1	Exported Constants		 					 					25
		18.3.2	Exported Access Program	ns .	 					 					26
	18.4	Semant	ics		 					 					26
			State Variables												26
		18.4.2	Environment Variables .		 					 					26
			Assumptions												26
			Access Routine Semantic												26
			Local Functions												27
		10.1.0	notal i directorio	• •	 • •	• •	• •	• •		 •	•		•		
19	MIS	of Acc	ount Update Interfac	e											27
					 					 					27
															27
															27
			Exported Constants												27
			Exported Access Program												27
	19 4														27
	10.1		State Variables												27
			Environment Variables												27
			Assumptions												27
			Access Routine Semantic												27
			Local Functions												28
		13.4.0	Local Functions	• •	 					 	•		•		20
20	MIS	of Log	in Interface												28
20			in Interface		 					 					28 28
20	20.1	Module													28
20	20.1 20.2	Module Uses .			 					 					
20	20.1 20.2	Module Uses . Syntax		• • •	 					 			•		28 28 28
20	20.1 20.2	Module Uses Syntax 20.3.1	Exported Constants		 					 				 	28 28 28 28
20	20.1 20.2 20.3	Module Uses Syntax 20.3.1 20.3.2	Exported Constants Exported Access Program	ms.	 	· · · · · ·	 	· · · · · ·	· · · · · ·	 				 	28 28 28 28 28
20	20.1 20.2 20.3	Module Uses . Syntax 20.3.1 20.3.2 Semant	Exported Constants Exported Access Programics	ms .	 	· · · · · · · ·			 	 				 	28 28 28 28 28 28
20	20.1 20.2 20.3	Module Uses . Syntax 20.3.1 20.3.2 Semant 20.4.1	Exported Constants Exported Access Programics State Variables	ms .	 					 		· · · · · · · · · · · · · · · · · · ·			28 28 28 28 28 28 28
20	20.1 20.2 20.3	Module Uses . Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2	Exported Constants Exported Access Progrances State Variables Environment Variables	ms .	 		· · · · · · · · · · · · · · · · · · ·			 					28 28 28 28 28 28 28 28
20	20.1 20.2 20.3	Module Uses Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2 20.4.3	Exported Constants Exported Access Programics State Variables Environment Variables Assumptions	ms .	 		· · · · · · · · · · · · · · · · · · ·			 					28 28 28 28 28 28 28 28 28
20	20.1 20.2 20.3	Module Uses Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2 20.4.3 20.4.4	Exported Constants Exported Access Programics State Variables Environment Variables Assumptions Access Routine Semantic	ms .	 					 				· · · · · · · · · · · · · · · · · · ·	28 28 28 28 28 28 28 28 28 28
20	20.1 20.2 20.3	Module Uses Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2 20.4.3 20.4.4	Exported Constants Exported Access Programics State Variables Environment Variables Assumptions	ms .	 					 				· · · · · · · · · · · · · · · · · · ·	28 28 28 28 28 28 28 28 28
	20.1 20.2 20.3 20.4	Module Uses Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2 20.4.3 20.4.4 20.4.5	Exported Constants Exported Access Programics State Variables Environment Variables Assumptions Access Routine Semantic	ms .	 					 				· · · · · · · · · · · · · · · · · · ·	28 28 28 28 28 28 28 28 28 28
	20.1 20.2 20.3 20.4	Module Uses Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2 20.4.3 20.4.4 20.4.5 S of Acc	Exported Constants Exported Access Programics Estate Variables Environment Variables Assumptions Access Routine Semantic	ms											28 28 28 28 28 28 28 28 28 29
	20.1 20.2 20.3 20.4 MIS 21.1	Module Uses Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2 20.4.3 20.4.4 20.4.5 S of Acc Module	Exported Constants Exported Access Programics State Variables Environment Variables Assumptions Access Routine Semantic Local Functions	ms											28 28 28 28 28 28 28 28 29 29
	20.1 20.2 20.3 20.4 MIS 21.1 21.2	Module Uses Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2 20.4.3 20.4.4 20.4.5 S of Accommodule Uses	Exported Constants Exported Access Programics State Variables Environment Variables Assumptions Access Routine Semantic Local Functions ess Token	ms											28 28 28 28 28 28 28 28 29 29 29
	20.1 20.2 20.3 20.4 MIS 21.1 21.2	Module Uses Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2 20.4.3 20.4.4 20.4.5 S of Accommodule Uses Syntax	Exported Constants Exported Access Programics State Variables Environment Variables Assumptions Access Routine Semantic Local Functions ess Token	ms											28 28 28 28 28 28 28 29 29 29 29 29
	20.1 20.2 20.3 20.4 MIS 21.1 21.2	Module Uses Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2 20.4.3 20.4.4 20.4.5 Sof Accommodule Uses Syntax 21.3.1	Exported Constants Exported Access Programes Exported Access Programes Exported Variables Environment Variables Assumptions Access Routine Semantic Local Functions ess Token Exported Constants	ms											28 28 28 28 28 28 28 29 29 29 29 29 29
	20.1 20.2 20.3 20.4 MIS 21.1 21.2 21.3	Module Uses Syntax 20.3.1 20.3.2 Semant 20.4.1 20.4.2 20.4.3 20.4.4 20.4.5 S of Acc Module Uses Syntax 21.3.1 21.3.2	Exported Constants Exported Access Programics State Variables Environment Variables Assumptions Access Routine Semantic Local Functions ess Token	ms											28 28 28 28 28 28 28 29 29 29 29

	21.4.2 Environment Variables	30
	21.4.3 Assumptions	30
	21.4.4 Access Routine Semantics	30
	21.4.5 Local Functions	30
99 MI	of Labeler	30
	Module	3 (
	Uses	30
		30
22.3	Syntax	30
	22.3.1 Exported Constants	31
22.4	22.3.2 Exported Access Programs	31
22.4	Semantics	
		31
	22.4.2 Environment Variables	31
	22.4.3 Assumptions	31
	22.4.4 Access Routine Semantics	31
	22.4.5 Local Functions	32
23 MIS	of Client	32
23.1	Module	32
	Uses	32
	Syntax	32
	23.3.1 Exported Constants	32
	23.3.2 Exported Access Programs	32
23 4	Semantics	33
20.1	23.4.1 State Variables	33
	23.4.2 Environment Variables	33
	23.4.3 Assumptions	33
	23.4.4 Access Routine Semantics	33
	23.4.5 Local Functions	33
	20.4.9 Local Full colons	06
		3 4
24.1	Module	34
24.2	Uses	34
24.3	Syntax	34
	24.3.1 Exported Constants	34
	24.3.2 Exported Access Programs	34
24.4	Semantics	34
	24.4.1 State Variables	34
	24.4.2 Environment Variables	34
	24.4.3 Assumptions	34
	24.4.4 Access Routine Semantics	35
	24.4.5 Local Functions	35

25	MIS of Account Creation Controller	35
	25.1 Module	35
	25.2 Uses	35
	25.3 Syntax	35
	25.3.1 Exported Constants	35
	25.3.2 Exported Access Programs	36
	25.4 Semantics	36
	25.4.1 State Variables	36
	25.4.2 Environment Variables	36
	25.4.3 Assumptions	36
	25.4.4 Access Routine Semantics	36
	25.4.5 Local Functions	3'
26		37
	MIS of Account Update Controller	3'
	26.1 Module	3'
	26.2 Uses	
	26.3 Syntax	38
	26.3.1 Exported Constants	38
	26.3.2 Exported Access Programs	38
	26.4 Semantics	38
	26.4.1 State Variables	38
	26.4.2 Environment Variables	38
	26.4.3 Assumptions	38
	26.4.4 Access Routine Semantics	38
	26.4.5 Local Functions	38
27	MIS of Authentication Controller	39
	27.1 Module	39
	27.2 Uses	39
	27.3 Syntax	39
	27.3.1 Exported Constants	38
	27.3.2 Exported Access Programs	38
	27.4 Semantics	38
	27.4.1 State Variables	39
	27.4.2 Environment Variables	39
	27.4.3 Assumptions	39
	27.4.4 Access Routine Semantics	39
	27.4.5 Local Functions	40
28	MIS of Satellite Image Request Interface	40
	28.1 Module	40
	28.2 Uses	40
	28.3 Syntax	40

		28.3.1 Expo	rted Constants				 	 	 	. 40
		28.3.2 Expo	rted Access Prog	rams .			 	 	 	. 40
	28.4	Semantics .					 	 	 	. 40
		28.4.1 State	Variables				 	 	 	. 40
		28.4.2 Envir	onment Variable	S			 	 	 	. 40
		28.4.3 Assur	mptions				 	 	 	. 40
		28.4.4 Acces	ss Routine Seman	ntics .			 	 	 	. 41
		28.4.5 Local	Functions				 	 	 	. 41
2 9	MIS	of Satellite	Image Reques	st Con	trollei	•				41
	29.1	$Module \dots$. 41
	29.2	Uses					 	 	 	. 41
	29.3	Syntax					 	 	 	. 41
		29.3.1 Expo	rted Constants				 	 	 	. 41
		29.3.2 Expo	rted Access Prog	rams.			 	 	 	. 41
	29.4	Semantics .					 	 	 	. 41
		29.4.1 State	Variables				 	 	 	. 41
		29.4.2 Envir	onment Variable	S			 	 	 	. 42
		29.4.3 Assur	mptions				 	 	 	. 42
		29.4.4 Acces	s Routine Seman	ntics .			 	 	 	. 42
		29.4.5 Local	Functions				 	 	 	. 42
30	MIS	of Satellite	Image Reques	${f st}$						42
	30.1	$Module \dots$. 42
	30.2	Uses					 	 	 	. 42
	30.3	Syntax					 	 	 	. 42
		30.3.1 Expo	rted Constants				 	 	 	. 42
		30.3.2 Expo	rted Access Prog	rams.			 	 	 	. 43
	30.4	Semantics .							 	. 43
		20 4 1 04 . 4 .					 			
		30.4.1 State	Variables						 	. 43
			Variables conment Variable				 			
		30.4.2 Envir		s			 	 	 	. 43
		30.4.2 Envir 30.4.3 Assur	onment Variable	s			 	 	 	. 43 . 43
		30.4.2 Envir 30.4.3 Assur 30.4.4 Acces	conment Variable mptions	s ntics .			 	 	 	. 43 . 43 . 43
31	MIS	30.4.2 Envir 30.4.3 Assur 30.4.4 Acces 30.4.5 Local	conment Variable mptions ss Routine Seman	es ntics .			 	 	 	. 43 . 43 . 43
31		30.4.2 Envir 30.4.3 Assur 30.4.4 Acces 30.4.5 Local of Project	conment Variable mptions ss Routine Seman Functions	ntics .			 	 	 	. 43 . 43 . 43 . 44
31	31.1	30.4.2 Envir 30.4.3 Assur 30.4.4 Acces 30.4.5 Local of Project Module	conment Variable inptions	ntics .			 		 	. 43 . 43 . 44 . 44 . 44
31	31.1 31.2	30.4.2 Envir 30.4.3 Assur 30.4.4 Acces 30.4.5 Local of Project Module Uses	conment Variable mptions	atics			 		 	. 43 . 43 . 44 . 44 . 44
31	31.1 31.2	30.4.2 Envir 30.4.3 Assur 30.4.4 Acces 30.4.5 Local of Project Module Uses	conment Variable mptions	atics			 		 	. 43 . 43 . 44 . 44 . 44 . 44
31	31.1 31.2	30.4.2 Envir 30.4.3 Assur 30.4.4 Acces 30.4.5 Local of Project Module Uses Syntax 31.3.1 Expo	conment Variable mptions	atics			 		 	. 43 . 43 . 44 . 44 . 44 . 44
31	31.1 31.2 31.3	30.4.2 Envir 30.4.3 Assur 30.4.4 Acces 30.4.5 Local of Project Module Uses Syntax 31.3.1 Expo	conment Variable mptions	rams .			 		 	. 43 . 43 . 44 . 44 . 44 . 44 . 44

		31.4.2	Environment	Variab	les					 								44
		31.4.3	Assumptions							 								44
		31.4.4	Access Routi	ne Sema	antics					 								45
			Local Function															45
32	MIS	of Pro	ject Creation	on Cor	ntrolle	r												45
	32.1	Module								 								45
	32.2	Uses .								 								45
	32.3	Syntax								 								45
			Exported Cor															45
			Exported Acc															45
	32.4		ics															45
			State Variabl															45
			Environment															46
			Assumptions															46
			Access Routi															46
			Local Function															46
		02.4.0	Local Fullcul	1115			• •		• •	 	• •	•	•	 •		•	•	40
33	MIS	of Pro	iect															46
-										 			_				_	46
																		46
																		46
	00.0		Exported Co															46
			Exported Acc															47
	33 /	Semanti	$cs \dots \dots$	7C55 I IC	ogi airis		• •	• •	• •	 		•	•	 •	•	•	•	47
	55.4		State Variabl															47
			Environment															47
			Assumptions															47
			Access Routi															
																		47
		33.4.5	Local Function	ons						 		•	•	 ٠		•	•	48
34	MIS	of Serv	vice Reques	t Failı	ire Int	erfs	ice											48
01			· · · · · · ·															48
																		48
																		48
	04.0		Exported Co															48
			Exported Con Exported Aco															48
	911		-		_													
	34.4		ics															49
			State Variabl															49
			Environment															49
			Assumptions															49
			Access Routin		antics					 		•	•	 •		•	•	49
		3445	Local Function	me														40

35	MIS	of Image Upload Interface 49
	35.1	Module
	35.2	Uses
	35.3	Syntax
		35.3.1 Exported Constants
		35.3.2 Exported Access Programs
	35.4	Semantics
		35.4.1 State Variables
		35.4.2 Environment Variables
		35.4.3 Assumptions
		35.4.4 Access Routine Semantics
		35.4.5 Local Functions
36	MIS	of Report Interface 50
		Module
		Uses
		Syntax
	00.0	36.3.1 Exported Constants
		36.3.2 Exported Access Programs
	36.4	Semantics
		36.4.1 State Variables
		36.4.2 Environment Variables
		36.4.3 Assumptions
		36.4.4 Access Routine Semantics
		36.4.5 Local Functions
27	MIC	of Report Controller 51
31		
		Module
		Syntax
	31.3	37.3.1 Exported Constants
		±
	27 1	
	37.4	Semantics 52 37.4.1 State Variables 52
		37.4.1 State variables
		•
		37.4.4 Access Routine Semantics
		37.4.5 Local Functions
38		of Report 52
		Module
		Uses
	20 2	Cymtay

		38.3.1	Exported Constants	52
		38.3.2	Exported Access Programs	
	38.4	Semanti	iics	
		38.4.1	State Variables	
		38.4.2	Environment Variables	
		38.4.3	Assumptions	
		38.4.4	Access Routine Semantics	
		38.4.5	Local Functions	
39	MIS	of Pro	oject Selection Interface	5 4
00				
	30.3	*	Exported Constants	
			Exported Access Programs	
	39.4		icics	
			State Variables	
			Environment Variables	
			Assumptions	
			Access Routine Semantics	
			Local Functions	
40	NATO	of Dec	vicat Selection Controller	21
40			oject Selection Controller	5 8
40	40.1	Module	·	
40	$40.1 \\ 40.2$	Module Uses .	·	
40	$40.1 \\ 40.2$	Module Uses . Syntax		
40	$40.1 \\ 40.2$	Module Uses . Syntax 40.3.1	Exported Constants	58
40	40.1 40.2 40.3	Module Uses . Syntax 40.3.1 1 40.3.2	Exported Constants	58
40	40.1 40.2 40.3	Module Uses . Syntax 40.3.1 1 40.3.2 1 Semanti	Exported Constants Exported Access Programs icits	58
40	40.1 40.2 40.3	Module Uses Syntax 40.3.1 40.3.2 Semant: 40.4.1	Exported Constants Exported Access Programs Sics State Variables	58
40	40.1 40.2 40.3	Module Uses . Syntax 40.3.1 1 40.3.2 1 Semanti 40.4.1 3 40.4.2 1	Exported Constants Exported Access Programs Sics State Variables Environment Variables	58
40	40.1 40.2 40.3	Module Uses . Syntax 40.3.1 . Semant: 40.4.1 . 40.4.2 . 40.4.3 . 40.4.3 . 40.4.3	Exported Constants Exported Access Programs Sics State Variables Environment Variables Assumptions	58
40	40.1 40.2 40.3	Module Uses . Syntax 40.3.1 1 40.3.2 Semant: 40.4.1 3 40.4.2 1 40.4.3 40.4.4	Exported Constants Exported Access Programs Cics State Variables Environment Variables Assumptions Access Routine Semantics	56
40	40.1 40.2 40.3	Module Uses . Syntax 40.3.1 1 40.3.2 Semant: 40.4.1 3 40.4.2 1 40.4.3 40.4.4	Exported Constants Exported Access Programs Sics State Variables Environment Variables Assumptions	56
	40.1 40.2 40.3 40.4	Module Uses . Syntax 40.3.1 . 40.3.2 . Semant: 40.4.1 . 40.4.2 . 40.4.3 . 40.4.4 . 40.4.5 . I	Exported Constants Exported Access Programs Cics State Variables Environment Variables Assumptions Access Routine Semantics	56
	40.1 40.2 40.3 40.4 MIS 41.1	Module Uses . Syntax 40.3.1 . Semanti 40.4.1 . 40.4.2 . 40.4.4 . 40.4.5 . Sof Lab Module	Exported Constants Exported Access Programs Sics State Variables Environment Variables Assumptions Access Routine Semantics Local Functions Deling Interface	56
	40.1 40.2 40.3 40.4 MIS 41.1	Module Uses . Syntax 40.3.1 . Semanti 40.4.1 . 40.4.2 . 40.4.4 . 40.4.5 . Sof Lab Module	Exported Constants Exported Access Programs Cics State Variables Environment Variables Assumptions Access Routine Semantics Local Functions Deling Interface	56
	40.1 40.2 40.3 40.4 MIS 41.1 41.2	Module Uses Syntax 40.3.1 Semant: 40.4.1 Semant: 40.4.3 August 40.4.4 August 40.4.5 Semant: 40.4	Exported Constants Exported Access Programs cics State Variables Environment Variables Assumptions Access Routine Semantics Local Functions coeling Interface	56
	40.1 40.2 40.3 40.4 MIS 41.1 41.2	Module Uses Syntax 40.3.1 Semants 40.4.1 Semants 40.4.2 Module Uses Syntax 41.3.1 Module Uses Syntax 41.3 Module Use	Exported Constants Exported Access Programs Sics State Variables Environment Variables Assumptions Access Routine Semantics Local Functions Deling Interface Exported Constants	56
	40.1 40.2 40.3 40.4 MIS 41.1 41.2	Module Uses Syntax 40.3.1 Semants 40.4.1 Semants 40.4.2 Module Uses Syntax 41.3.1 Module Uses Syntax 41.3 Module Use	Exported Constants Exported Access Programs cics State Variables Environment Variables Assumptions Access Routine Semantics Local Functions coeling Interface	56
	40.1 40.2 40.3 40.4 MIS 41.1 41.2 41.3	Module Uses Syntax 40.3.1 40.3.2 Semanti 40.4.1 40.4.3 40.4.3 40.4.4 40.4.5 of Lab Module Uses Syntax 41.3.1 41.3.2 Semanti	Exported Constants Exported Access Programs Sics State Variables Environment Variables Assumptions Access Routine Semantics Local Functions Deling Interface Exported Constants	58

		41.4.2 Environment Variables	57
		41.4.3 Assumptions	57
		41.4.4 Access Routine Semantics	57
		41.4.5 Local Functions	57
42		of Labeling Controller	57
		Module	57
	42.2	Uses	58
	42.3	Syntax	58
		42.3.1 Exported Constants	58
		42.3.2 Exported Access Programs	58
	42.4	Semantics	58
		42.4.1 State Variables	58
		42.4.2 Environment Variables	58
		42.4.3 Assumptions	58
		42.4.4 Access Routine Semantics	58
		42.4.5 Local Functions	59
43		of Image	59
	43.1	Module	59
		$egin{array}{cccccccccccccccccccccccccccccccccccc$	59
	43.3	Syntax	59
		43.3.1 Exported Constants	59
		43.3.2 Exported Access Programs	59
	43.4	Semantics	59
		43.4.1 State Variables	59
		43.4.2 Environment Variables	59
		43.4.3 Assumptions	60
		43.4.4 Access Routine Semantics	60
		43.4.5 Local Functions	60
44		of Label Server	61
		Module	61
		Uses	61
	44.3	Syntax	61
		44.3.1 Exported Constants	61
		44.3.2 Exported Access Programs	61
	44.4	Semantics	61
		44.4.1 State Variables	61
		44.4.2 Environment Variables	61
		44.4.3 Assumptions	61
		44.4.4 Access Routine Semantics	61
		44.4.5 Local Functions	66

45	MIS of Label Database Connector	63
	45.1 Module	. 63
	45.2 Uses	. 63
	45.3 Syntax	. 63
	45.3.1 Exported Constants	. 63
	45.3.2 Exported Access Programs	. 63
	45.4 Semantics	
	45.4.1 State Variables	. 63
	45.4.2 Environment Variables	. 63
	45.4.3 Assumptions	. 63
	45.4.4 Access Routine Semantics	. 63
	45.4.5 Local Functions	. 64
40	MIC CI LID (1	۵.
46	MIS of Label Database	65
	46.1 Module	
	46.2 Uses	
	46.3 Syntax	
	46.3.1 Exported Constants	
	46.3.2 Exported Access Programs	
	46.4 Semantics	
	46.4.1 State Variables	
	46.4.2 Environment Variables	
	46.4.3 Assumptions	
	46.4.4 Access Routine Semantics	
	46.4.5 Local Functions	. 66
47	MIS of ImageObject Database Connector	67
	47.1 Module	. 67
	47.2 Uses	
	47.3 Syntax	. 67
	47.3.1 Exported Constants	. 67
	47.3.2 Exported Access Programs	. 67
	47.4 Semantics	
	47.4.1 State Variables	. 67
	47.4.2 Environment Variables	. 67
	47.4.3 Assumptions	. 67
	47.4.4 Access Routine Semantics	. 67
	47.4.5 Local Functions	. 68
10	MIS of Image Object Detabage	eo
48	MIS of ImageObject Database	69
	48.1 Module	
	48.2 Uses	
	48.3 Syntax	. 69

		48.3.1	Exported Constants	 		 	 		 	. 69
		48.3.2	Exported Access Programs	 		 	 		 	69
	48.4	Seman	tics	 		 	 		 	69
		48.4.1	State Variables	 		 	 		 	. 69
		48.4.2	Environment Variables	 		 	 		 	69
		48.4.3	Assumptions	 		 	 		 	69
		48.4.4	Access Routine Semantics	 		 	 		 	69
		48.4.5	Local Functions	 		 	 		 	. 70
40	N ATO	. ст								
49			beller Database Connector							71
			e							
	49.3	~								
			Exported Constants							
	40.4		Exported Access Programs							
	49.4		tics							
			State Variables							
			Environment Variables							
			Assumptions							
			Access Routine Semantics							
		49.4.5	Local Functions	 		 	 	•	 	. 72
50	MIS	of La	beller Database							7 3
			e	 		 	 		 	
			C							
		•	Exported Constants							
			Exported Access Programs							
	50.4		tics							
	00.1		State Variables							
			Environment Variables							
			Assumptions							
		00.1.0								
		50 4 4	T							79
			Access Routine Semantics	 		 	 			
			T	 		 	 			
51	MIS	50.4.5	Access Routine Semantics	 		 	 			
51		50.4.5 of O b	Access Routine Semantics Local Functions	 	• •	 	 		 	. 74 7 5
51	51.1	50.4.5 S of O b Modul	Access Routine Semantics Local Functions	 		 	 		 	. 74 7 5 . 75
51	51.1 51.2	50.4.5 S of Ob Modul Uses	Access Routine Semantics Local Functions oject Extraction Manager e	 		 	 		 	74 75 75
51	51.1 51.2	50.4.5 S of Ob Modul Uses Syntax	Access Routine Semantics Local Functions oject Extraction Manager e	 		 	 		 	75 75 75 75
51	51.1 51.2	50.4.5 S of Ob Modul Uses Syntax 51.3.1	Access Routine Semantics Local Functions oject Extraction Manager e	 		 	 		 	75 75 75 75 75
51	51.1 51.2 51.3	50.4.5 6 of Ob Modul Uses Syntax 51.3.1 51.3.2	Access Routine Semantics Local Functions oject Extraction Manager e	 		 	 		 	75 75 75 75 75 75

	51.4.2 Environment Variables	75
	51.4.3 Assumptions	75
	51.4.4 Access Routine Semantics	75
		76
52 MIS	S of Label Confidence Service	77
52.1	Module	77
52.2	Uses	77
52.3	Syntax	77
	52.3.1 Exported Constants	77
	52.3.2 Exported Access Programs	77
52.4	Semantics	77
		77
	52.4.2 Environment Variables	77
		77
	1	77
		78
53 MIS	S of Object Extraction Service	7 9
53.1	Module	79
53.2	Uses	79
53.3	Syntax	79
	·	79
	•	79
53.4		79
		79
		79
		79
	•	79
		80
54 MIS	S of Image Prior Analyzer	81
54.1	Module	81
54.2	Uses	81
54.3	Syntax	81
	·	81
		81
54.4	-	81
		81
		81
		81
	1	81
		81

55	MIS of Labeller Expertise Calculator	82
	55.1 Module	82
	55.2 Uses	82
	55.3 Syntax	82
	55.3.1 Exported Constants	82
	55.3.2 Exported Access Programs	82
	55.4 Semantics	82
	55.4.1 State Variables	82
	55.4.2 Environment Variables	82
	55.4.3 Assumptions	82
	55.4.4 Access Routine Semantics	82
	55.4.5 Local Functions	83
56	MIS of Image Service Manager	84
	56.1 Module	84
	56.2 Uses	84
	56.3 Syntax	84
	56.3.1 Exported Constants	84
	56.3.2 Exported Access Programs	84
	56.4 Semantics	84
	56.4.1 State Variables	84
	56.4.2 Environment Variables	84
	56.4.3 Assumptions	84
	56.4.4 Access Routine Semantics	84
	56.4.5 Local Functions	85
E 17	MIC of Ivages Mack Convice	86
37	MIS of Image Mask Service 57.1 Module	86
	57.1 Module	86
	57.3 Syntax	86
	57.3.1 Exported Constants	86
	57.3.2 Exported Access Programs	86
	57.3.2 Exported Access Flograms	86
	57.4.1 State Variables	86
	57.4.2 Environment Variables	86
	57.4.3 Assumptions	86
	57.4.4 Access Routine Semantics	86
	57.4.5 Local Functions	86
F 0		
58	MIS of Image Selection Service	87
	58.1 Module	87
	58.2 Uses	87
	58.3 Syntax	87

		58.3.1	Exported Const	tants													. 8	37
		58.3.2	Exported Acces	s Programs													. 8	37
	58.4	Semanti	ics									 					. 8	37
			State Variables															37
		58.4.2	Environment Va	ariables													. 8	37
		58.4.3	Assumptions														. 8	37
		58.4.4	Access Routine	Semantics													. 8	37
		58.4.5	Local Functions	3													. 8	38
- 0	NATO	-C N/I-	d-10	II l 4:													0	39
99			delCompariso lule															9 39
			ruie															39 39
			tax															39 39
	59.5																	39 39
			6.3.1 Exported															39 39
	EO 4		6.3.2 Exported antics															39 39
	39.4																	
			6.4.1 State Vari															39
			6.4.2 Environme															39
			6.4.3 Assumption															39
			6.4.4 Access Ro															90
		59.4.5	6.4.5 Local Fun	ctions			•		•			 •	•	•	 •	•	. 9	90
60	MIS	of Cro	${ m ssValidationE}$	Evaluation													9	1
60			ssValidationE						•			 					_)1)1
60	60.1	6.1 Mod															. 9	
60	60.1 60.2	6.1 Mod 6.2 Uses	${ m dule}$. 9	91
60	60.1 60.2	6.1 Mod 6.2 Uses 6.3 Synt	lule				•				· ·	 					. 9	91 91
60	60.1 60.2	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1	lule	Constants		 	•	 		 		 			 		. 0	91 91 91
60	60.1 60.2 60.3	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (lule	Constants Access Prog	 grams	 	•	 		 	 	 		•	 		. 6	91 91 91
60	60.1 60.2 60.3	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (6.4 Sem	dule	Constants Access Prog	grams	 		 		· · · · · · · · · · · · · · · · · · ·					 		. 6	91 91 91 91
60	60.1 60.2 60.3	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (6.4 Sem 60.4.1 (tax	Constants Access Prog	 grams 	 	•	· · · · · ·		· · · · · · · · · · · · · · · · · · ·					 		. 00	91 91 91 91
60	60.1 60.2 60.3	6.1 Moc 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (6.4 Sem 60.4.1 (60.4.2 (dule tax 6.3.1 Exported 6.3.2 Exported antics 6.4.1 State Vari 6.4.2 Environme	Constants Access Prog ables ent Variable	grams	 		· · · · · · · · · · · · · · · · · · ·							 			91 91 91 91 91
60	60.1 60.2 60.3	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (6.4 Sem 60.4.1 (60.4.2 (60.4.3 (60.4	lule tax 6.3.1 Exported 6.3.2 Exported entics 6.4.1 State Vari	Constants Access Prog ables ent Variable	grams	 					· · · · · · · · · · · · · · · · · · ·				· · · · · ·		. 00	91 91 91 91 91
60	60.1 60.2 60.3	6.1 Moc 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (6.4 Sem 60.4.1 (60.4.2 (60.4.3 (60.4.4 (dule tax 6.3.1 Exported 6.3.2 Exported antics 6.4.1 State Vari 6.4.2 Environme 6.4.3 Assumption	Constants Access Prog ables ent Variable ons	grams	 				· · · · · · · · · · · · · · · · · · ·								91 91 91 91 91
	60.1 60.2 60.3	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (6.4 Sem 60.4.1 (60.4.2 (60.4.3 (60.4.4 (60.4.5 (60.4	tax 6.3.1 Exported 6.3.2 Exported 6.4.1 State Vari 6.4.2 Environme 6.4.3 Assumptio 6.4.4 Access Ro 6.4.5 Local Fun	Constants Access Progrables ables ent Variable ons entine Seman	grams	 				· · · · · · · · · · · · · · · · · · ·								91 91 91 91 91 92
	60.1 60.2 60.3 60.4	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (60.4.2 (60.4.3 (60.4.4 (60.4.5 (60.4	tax 6.3.1 Exported 6.3.2 Exported 6.4.1 State Vari 6.4.2 Environme 6.4.3 Assumptic 6.4.4 Access Ro 6.4.5 Local Funded	Constants Access Prog ables ent Variable ons entine Seman	gramses	 												91 91 91 91 91 92
	60.1 60.2 60.3 60.4 MIS 61.1	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (60.4.4 (60.4.2 (60.4.4 (60.4.5 (60.4	tax 6.3.1 Exported 6.3.2 Exported 6.4.1 State Vari 6.4.2 Environme 6.4.3 Assumptic 6.4.4 Access Ro 6.4.5 Local Fun delTrainingSe	Constants Access Prog ables ent Variable ons utine Seman	grams													91 91 91 91 91 92 93
	60.1 60.2 60.3 60.4 MIS 61.1 61.2	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (6.4 Sem 60.4.1 (60.4.3 (60.4.3 (60.4.4 (60.4.5 (60.4	dule tax 6.3.1 Exported 6.3.2 Exported eantics 6.4.1 State Vari 6.4.2 Environme 6.4.3 Assumptic 6.4.4 Access Ro 6.4.5 Local Fun delTrainingSe	Constants Access Programmers ables ent Variable ons cutine Seman ctions ervice	grams													91 91 91 91 91 91 92 93
	60.1 60.2 60.3 60.4 MIS 61.1 61.2	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (60.4.2 (60.4.3 (60.4.4 (60.4.5 (60.4	tax 6.3.1 Exported 6.3.2 Exported 6.4.1 State Vari 6.4.2 Environme 6.4.3 Assumptic 6.4.4 Access Ro 6.4.5 Local Fun delTrainingSe lule 6.4.4	Constants Access Prog ables ent Variable ons utine Seman	grams												. 9 . 9 . 9 . 9 . 9	91 91 91 91 91 91 92 93
	60.1 60.2 60.3 60.4 MIS 61.1 61.2	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (60.4.4 Sem 60.4.1 (60.4.2 (60.4.3 (60.4.4 (60.4.5 (6	tax 6.3.1 Exported 6.3.2 Exported 6.4.1 State Vari 6.4.2 Environme 6.4.3 Assumptio 6.4.4 Access Ro 6.4.5 Local Fundel Training Security bule 6.4.5 Local Fundel Training Security 6.3.1 Exported	Constants Access Progrables ables ent Variable ons entine Seman ctions crvice Constants	grams es ntics												. 9 . 9 . 9 . 9 . 9	91 91 91 91 91 92 93 93
	60.1 60.2 60.3 60.4 MIS 61.1 61.2 61.3	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (6.4 Sem 60.4.1 (60.4.2 (60.4.3 (60.4.4 (60.4.5 (60.4	tax 6.3.1 Exported 6.3.2 Exported 6.4.1 State Vari 6.4.2 Environme 6.4.3 Assumptic 6.4.4 Access Ro 6.4.5 Local Fun delTrainingSe lule 5 tax 6.3.1 Exported 6.3.2 Exported	Constants Access Programmers ables ent Variable ons cutine Seman ctions ervice Constants Access Prog	grams													91 91 91 91 91 91 93 93
	60.1 60.2 60.3 60.4 MIS 61.1 61.2 61.3	6.1 Mod 6.2 Uses 6.3 Synt 60.3.1 (60.3.2 (6.4 Sem 60.4.1 (60.4.2 (60.4.3 (60.4.4 (60.4.5 (60.4	tax 6.3.1 Exported 6.3.2 Exported 6.4.1 State Vari 6.4.2 Environme 6.4.3 Assumptio 6.4.4 Access Ro 6.4.5 Local Fundel Training Security bule 6.4.5 Local Fundel Training Security 6.3.1 Exported	Constants Access Prog ables ent Variable ons utine Seman ctions ervice Constants Access Prog	grams												. 99 . 99 . 99 . 99 . 99 . 99	91 91 91 91 91 92 93 93

		61.4.2	6.4.2	Envir	conm	ent V	/aria	bles										 		93
		61.4.3																		93
		61.4.4																		94
		61.4.5																		94
62	MIS	of Mo	odelE	valua	ation	Serv	zice													95
		6.1 Mc																 		95
		6.2 Use																		95
		6.3 Syı																		95
		62.3.1																		95
		62.3.2		-																95
	62.4	6.4 Ser																		95
		62.4.1																		95
		62.4.2																		95
		62.4.3																		95
		62.4.4			-															96
		62.4.5																		96
63	MIC	of Mo	odolN	Innac	ror															97
UJ		6.1 Mc			_															97
		6.2 Use																		97
		6.3 Syr																		97
	00.0	63.3.1																		97
		63.3.2		_																97
	63.4	6.4 Ser																		97
	00.4	63.4.1																		97
		63.4.2																		97
		63.4.3																		98
		63.4.4			-															98
		63.4.5																		98
0.4	N ATC																			00
64		of Mo																		99
		6.1 Mc																		99
		6.2 Use																		99
	64.3	6.3 Syr																		99
		64.3.1		_																99
	01.1	64.3.2		-																99
	64.4	6.4 Ser																		99
		64.4.1																		99
		64.4.2																		99
		64.4.3			-															99
		64.4.4						nanti	ICS		•	 •	 •		•	•		 •	•	99
		64 4 5	645	Local	(Him	ction	Q													100

65	MIS	of M	LModelDatabase													101
	65.1	6.1 Mc	${ m odule}$													101
	65.2	6.2 Us	es													101
	65.3	6.3 Syr	\max													101
		65.3.1	6.3.1 Exported Constants													101
		65.3.2	6.3.2 Exported Access Programs													101
	65.4	6.4 Ser	$\operatorname{nantics}$													101
		65.4.1	6.4.1 State Variables													101
		65.4.2	6.4.2 Environment Variables													101
		65.4.3	6.4.3 Assumptions													101
		65.4.4	6.4.4 Access Routine Semantics													102
		65.4.5	6.4.5 Local Functions													102
66	MIS	of Ot	herModelCreation													103
			odule													103
	66.2	6.2 Us	es													103
			ntax													103
			6.3.1 Exported Constants													103
			6.3.2 Exported Access Programs													103
	66.4		$\frac{1}{1}$													103
		66.4.1	6.4.1 State Variables													103
			6.4.2 Environment Variables													103
			6.4.3 Assumptions													103
			6.4.4 Access Routine Semantics													103
			6.4.5 Local Functions													104
67	MIS	of CN	NModelCreation													104
•			odule													104
			es													104
			ntax													104
		_	6.3.1 Exported Constants													104
			6.3.2 Exported Access Programs													104
	67.4		$\frac{1}{1}$													104
		67.4.1	6.4.1 State Variables													104
			6.4.2 Environment Variables													104
			6.4.3 Assumptions													104
			6.4.4 Access Routine Semantics													105
			6.4.5 Local Functions													105
68	Exc	eption	Handling													105
-		_	nd Handling (React)		_		_			_			_			105
			nd Handling (Python)							•	•	•	•	•	•	105

69 Appendix 106

3 Introduction

The following document details the Module Interface Specifications for OrbitWatch, a crowd-sourced datalabelling platform which aims to improve the process of extracting information from satelite images.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/OKKM-insights/OKKM.insights/

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 $(-\infty, \infty)$
natural number	N	a number without a fractional component in $[1, \infty)$
real	\mathbb{R}	any number in $(-\infty, \infty)$
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.

System Components

MLModel

Represents a machine learning model, identified by attributes such as:

model_name

- \bullet model_path
- model_type
- Metadata about the model (e.g., training parameters, architecture)

Model Training Run

Captures the details of a model's training process, including:

- training_data_path
- Evaluation metrics
- Parameters used during training

ModelEvaluationRun

Represents the evaluation process for a model, containing:

- evaluation_data_path
- Evaluation metrics (e.g., precision, recall)

ModelDeployment

Tracks the deployment details of a machine learning model, such as:

- deployment_environment (e.g., Production, Staging)
- deployment_date

Account

Describes user accounts in the system with attributes like:

- username
- email
- account_type (e.g., Client, Labeler, Admin)
- Security-related fields such as password_hash and last_login

Account Modification

Maintains a log of changes made to user accounts, tracking:

- field_modified
- old_value
- new_value

LoginAttempt

Records login attempts for security purposes, including:

- username
- \bullet attempt_time
- Whether the attempt was successful

Project

Defines a labeling or analysis project, identified by:

- project_name
- description
- Associated metadata

User

Represents individuals (e.g., labelers, managers) working within the system, including:

- username
- role

ProjectAssignment

Tracks which users are assigned to specific projects, identified by:

- \bullet project_id
- user_id

SatelliteImage

Represents images (e.g., satellite imagery) linked to specific projects, with attributes like:

- image_path
- acquisition_date

LabelingTask

Encapsulates a labeling activity, defined by:

- status
- start_time
- end_time
- The user assigned to the task

Report

Represents generated reports for projects, with fields like:

- report_data
- generation_date
- The user who generated the report

${\bf Service Request}$

Tracks requests for services such as image acquisition or data processing, with attributes like:

- request_type
- status

Image

Represents standalone images within the system, identified by:

- image_path
- upload_date

Labeller

Represents individuals performing labeling tasks, identified by:

 \bullet labeller_name

Object

Represents specific objects detected in an image, with attributes like:

- $\bullet \ bounding_box_coordinates$
- object_type

Label

Represents annotations made by a labeller, linking to specific objects in an image and storing information like:

- label_text
- timestamp
- \bullet labeller_id

The following diagram display additional details on the relationship between datatypes

Project Creation and Selection Subsystem

	Project
PK	project id INT AUTO INCREMENT
	project_name VARCHAR(255) NOT NULL
	description TEXT
	creation_date DATETIME DEFAULT CURRENT_TIM

	ProjectAssignment										
PK	user_id_INT										
PK	project_id_INT										
PK	project assignment id INT AUTO INCREMENT										
	assignment_date DATETIME DEFAULT CURRENT_										
	FOREIGN KEY (project_id) REFERENCES Project(p										
	FOREIGN KEY (user_id) REFERENCES User(user_i										

	User
PK	user id INT AUTO INCREMENT
	username VARCHAR(255) UNIQUE NOT NULL
	Add more user details as needed (e.g., name, ema
	role VARCHAR(50) e.g., 'Labeler', 'Manager', 'Coor

	SatelliteImage	
PK	project id INT, Link to the project the image bel	
PK	image id INT AUTO INCREMENT	
	image_path VARCHAR(255), Or BLOB if storing di	
	acquisition_date DATE	
	Add other relevant metadata	
	FOREIGN KEY (project_id) REFERENCES Project(p	

	LabelingTask	
PK	image_id INT	
PK	project id INT	
PK	labeling task id INT AUTO INCREMENT	
	assigned_to INT, User assigned to this task	
	status VARCHAR(50) DEFAULT 'Pending', e.g., 'Pe	
	start_time DATETIME	
	end_time DATETIME	
	FOREIGN KEY (project_id) REFERENCES Project(p	
	FOREIGN KEY (image_id) REFERENCES SatelliteIn	
	FOREIGN KEY (assigned_to) REFERENCES User(u	

	Report	
PK	project id INT	
PΚ	report id INT AUTO INCREMENT	
	generated_by INT, User who generated the report	
	generation_date DATETIME DEFAULT CURRENT_T	
	report_data TEXT, Or a link to a file if large	
	FOREIGN KEY (project_id) REFERENCES Project(p	
	FOREIGN KEY (generated_by) REFERENCES User	

ServiceRequest	
PK	project id INT
PK	request id INT AUTO INCREMENT
	requested_by INT
	request_date DATETIME DEFAULT CURRENT_TIME
	request_type VARCHAR(255), e.g., 'Image Acquisi
	status VARCHAR(50) DEFAULT 'Pending'
	FOREIGN KEY (project_id) REFERENCES Project(p
	FOREIGN KEY (requested_by) REFERENCES User

Computer Vision Model Creation Subsystem

	MLModel		
PK	model id INT AUTO INCREMENT		
	model_name VARCHAR(255) NOT NULL		
	model_path VARCHAR(255), Path to the model file		
	model_type VARCHAR(255), e.g., 'Classification', '		
	creation_date DATETIME DEFAULT CURRENT_TIM		
	last_modified DATETIME		
	description TEXT		
	version VARCHAR(50), Versioning of the model		
	metadata JSON Store model metadata like training		

ModelTrainingRun	
PK	model id INT
PK	training run id INT AUTO INCREMENT
	start_time DATETIME
	end_time DATETIME
	training_data_path VARCHAR(255), Path to the tra
	evaluation_metrics JSON, Store evaluation metrics
	training_parameters JSON, Store training paramete
	FOREIGN KEY (model_id) REFERENCES MLModel

	ModelEvaluationRun	
PK	model id INT	
PK	evaluation run id INT AUTO INCREMENT	
	start_time DATETIME	
	end_time DATETIME	
	evaluation_data_path VARCHAR(255), Path to the	
	evaluation_metrics JSON, Store evaluation metrics	
	FOREIGN KEY (model_id) REFERENCES MLModele	

ModelDeployment	
PK	model id INT
PK	deployment id INT AUTO INCREMENT
	deployment_date DATETIME DEFAULT CURRENT_
	deployment_environment VARCHAR(255), e.g., 'Pr
	deployed_by INT, User who deployed the model
	FOREIGN KEY (model_id) REFERENCES MLModel
	Add foreign key reference to user table if needed

Client/ Labeller Management Susbsystem

Account	
PK	account id INT AUTO INCREMENT
	username VARCHAR(255) UNIQUE NOT NULL
	password_hash VARCHAR(255) NOT NULL, Store
	email VARCHAR(255) UNIQUE
	full_name VARCHAR(255)
	account_type VARCHAR(50) CHECK (account_type
	creation_date DATETIME DEFAULT CURRENT_TIM
	last_login DATETIME
	Add other account-related fields as needed (e.g., a

	LoginAttempt	
PK	attempt id INT AUTO INCREMENT	
	username VARCHAR(255)	
	attempt_time DATETIME DEFAULT CURRENT_TIME	
	successful BOOLEAN	
	ip_address VARCHAR(45) For tracking location of	

Label Collection and Aggregation Subsystem

	lmage	
PK	image id INT AUTO INCREMENT	
	image_data BLOB, Or VARCHAR for file paths if st	
	image_path VARCHAR(255)	
	upload_date DATETIME	

Object	
PK	image id INT
PK	object id INT AUTO INCREMENT
	bounding_box_coordinates VARCHAR(255), Store
	object_type VARCHAR(255)
	FOREIGN KEY (image_id) REFERENCES Image(im:

AccountModification		
PK	account id INT	
PK	modification id INT AUTO INCREMENT	
	modified_by INT, User who made the modification (
	modification_date DATETIME DEFAULT CURRENT_	
	field_modified VARCHAR(255), e.g., 'email', 'full_na	
	old_value TEXT	
	new_value TEXT	
	FOREIGN KEY (account_id) REFERENCES Account	
	FOREIGN KEY (modified_by) REFERENCES Accoun	

	Labeller			
PK	PK <u>labeller id INT AUTO INCREMENT</u>			
	labeller_name VARCHAR(255)			

Label		
PK	labeller id INT	
PK	object id INT	
PK	image_id_INT	
PK	label id INT AUTO INCREMENT	
	label_text VARCHAR(255)	
	timestamp DATETIME	
	FOREIGN KEY (image_id) REFERENCES Image(image_id)	
	FOREIGN KEY (object_id) REFERENCES Object(object_id)	
	FOREIGN KEY (labeller_id) REFERENCES Labeller(labeller_id)	

5 Module Decomposition

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

Level 1	Level 2
Hardware-Hiding Mod- ule	

Table 1: Module Hierarchy

Level 1	Level 2
Level 1	Account Creation Interface
	Account Database
	Account Update Interface
	Login Interface
Behaviour-Hiding	Access Token
Module	Labeler
Wiodule	Client
	User
	Satellite Image Request Interface
	Satellite Image Request Project Creation Interfere
	Project Creation Interface
	Project Service Request Failure Interface
	-
	Image Upload Interface
	Report Interface
	Report Project Selection Interface
	Project Selection Interface
	Labeling Interface
	Image Label Server
	Label Database Connector
	Label Database
	ImageObject Database Connector
	ImageObject Database Labeller Database Connector
	Labeller Database
	Object Extraction Manager
	Image Service Manager
	ModelCreation (Abstract Class)
	CNNModelCreation
	Other Model Creation
	ModelManager
	MLModelDatabase

Table 2: Module Hierarchy

Level 1	Level 2
	Account Creation Controller
	Account Database Connector
	Account Update Controller
	Authentication Controller
Software Decision	Satellite Image Request Controller
\mathbf{Module}	Project Creation Controller
	Report Controller
	Project Selection Controller
	Labeling Controller
	Label Confidence Service
	Object Extraction Service
	Image Prior Analyzer
	Labeller Expertise Calculator
	Image Mask Service
	Image Selection Service
	ModelComparision Evaluation
	CrossValidation Evaluation
	ModelTrainingService
	ModelEvaluationService

Table 3: Module Hierarchy

6 MIS of Report Manager

6.1 Module

RM (ReportManager)

6.2 Uses

LabelDBConnector45 ObjectsOnImageDBConnector47 RawImageDBConnector11 ProjectDBConnector10

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
generateReport	projectId: String	Report	DatabaseException

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

generateReport(projectId: String)

- output: Returns a Report object that aggregates data from the label, object-on-image, raw image, and project databases.
- exception: DatabaseException: Thrown if there is an issue communicating with any of the underlying databases.

6.4.5 Local Functions

Any helper methods used internally to combine or transform the data (e.g., formatting label lists, summarizing object data) are not exported and thus not specified here.

7 MIS of Account Creation Interface

7.1 Module

Account Creation Interface

7.2 Uses

Account Creation Controller25

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	Enum[labeler, client]	-	
$\underline{ \text{submitForm}}$	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 Project Manager

8.1 Module

PM (ProjectManager)

8.2 Uses

ProjectCollectionManager9 CoreImageDBConnector11

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In		Out	Exceptions
createProject	projectName:		Project	DatabaseException
	String, m	netadata:		
	Map(String, S	String)		
addImageToProj	ectprojectId:	String,	-	${\bf Database Exception}$
	byte[]			

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

None

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

createProject(projectName, metadata)

- output: Returns a newly created Project object with a unique identifier and any associated metadata.
- exception: **ProjectAlreadyExistsException**: Thrown if a project with the same name or identifier already exists. **DatabaseException**: Thrown if any error occurs while writing to the database.

addImageToProject(projectId, imageData)

• exception: InvalidImageException: Thrown if the image data is corrupted or unsupported. ProjectNotFoundException: Thrown if the target project does not exist in the system. DatabaseException: Thrown if a database error occurs while storing the image.

8.4.5 Local Functions

Internal helper methods (e.g., validation, transformations) are not exported.

9 MIS of Project Collection Manager

9.1 Module

PCM (ProjectCollectionManager)

9.2 Uses

ProjectDBConnector10

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
getAvailable	Projectione	List(Project)	DatabaseException

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

None

9.4.3 Assumptions

9.4.4 Access Routine Semantics

getAvailableProjects()

- output: Returns a list of existing Project objects (could be filtered by user permissions or some criteria, if applicable).
- exception: **ProjectAlreadyExistsException**: Thrown if a project with the same name or identifier already exists.

9.4.5 Local Functions

Internal helper methods (e.g., transformations) are not exported.

10 MIS of Project Database Connector

10.1 Module

PDBC (ProjectDBConnector)

10.2 Uses

MySQL - ProjectDB

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
fetchProject	projectId: String	Project	DatabaseException
fetchProjectList	None	List(Project)	${\bf Database Exception}$
storeProject	project : Project	None	${\bf Database Exception}$

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

databaseConnection: connection to relational database

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

storeProject(project : Project)

- output: No direct output; success indicates the Project was successfully stored.
- exception: **DatabaseException**: Thrown if there is an issue communicating with any of the underlying databases. **DuplicateProjectException**: Thrown if the project already exists

fetchProjectList()

- output: Returns a list of all Project objects stored in the database.
- exception: **DatabaseException**: Thrown if there is an issue communicating with any of the underlying databases.

fetchProject(projectId : String)

- output: Returns the Project object corresponding to the given projectId.
- exception: **DatabaseException**: Thrown if there is an issue communicating with any of the underlying databases. **ProjectNotFoundException**: Thrown if the project with the given ID does not exist.

10.4.5 Local Functions

Any database query-building or data-mapping helpers remain internal and are not exported.

11 MIS of Core Image Database Connector

11.1 Module

CIDBC (CoreImageDBConnector)

11.2 Uses

MySQL - CoreImageDB

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
storeImage	projectId: String im-	String	DatabaseException
	ageData: byte[]		
fetchImage	imageId: String	Image	DatabaseException
fetchImageForPi	roj en tojectId: String	List(Image)	DatabaseException

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

databaseConnection: connection to relational database

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

storeImage(projectId, imageData)

- output: Returns a newly generated String identifier (imageId) that uniquely identifies the stored image in the database.
- exception: **DatabaseException**: Thrown if there is an issue communicating with any of the underlying databases. **ProjectNotFoundException**: Thrown if the specified projectId does not exist in the database.

fetchImage(imageId)

- output: Returns an Image object (or equivalent data structure) for the given imageId, including any relevant metadata or binary content.
- exception: **DatabaseException**: Thrown if there is an issue communicating with any of the underlying databases. **ImageNotFoundException**: Thrown if no image with the specified imageId exists in the database.

fetchImagesForProject(projectId)

- output: Returns a list of Image objects associated with the specified projectId.
- exception: **DatabaseException**: Thrown if there is an issue communicating with any of the underlying databases. **ProjectNotFoundException**: Thrown if the project with the given ID does not exist.

11.4.5 Local Functions

Any database query-building or data-mapping helpers remain internal and are not exported.

12 MIS of Account Database Connector

12.1 Module

Account Database Connector

12.2 Uses

Account Database18

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

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

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

databaseConnection: connection to relational database

12.4.3 Assumptions

12.4.4 Access Routine Semantics

insertUser(user):

- transition: Request to insert user into database through databaseConection. retrieveUser(email):
 - output:

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

updateUser(user):

• transition:

```
Request to update user in database, if userExists(user.email)
Do nothing otherwise
```

userExists(email):

• output: out :=

 $\exists User \in Database s.t. User.email == email$

makeDBConnection(credentials):

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

12.4.5 Local Functions

None

13 MIS of Account Database

13.1 Module

Account Database

13.2 Uses

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

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

13.4 Semantics

13.4.1 State Variables

None

13.4.2 Environment Variables

databaseConnection: connection to Application

13.4.3 Assumptions

None

13.4.4 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 User \in Database s.t. User.email == email

13.4.5 Local Functions

None

14 MIS of Account Update Interface

14.1 Module

Account Update Interface

14.2 Uses

Account Update Controller26

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	User	-	=
$\operatorname{submitForm}$	list[(string, string)]	-	-

14.4 Semantics

14.4.1 State Variables

None

14.4.2 Environment Variables

win: 2D sequence of coloured pixels

14.4.3 Assumptions

None

14.4.4 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.

14.4.5 Local Functions

None

15 MIS of Login Interface

15.1 Module

Login Interface

15.2 Uses

Authentication Controller27

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	_
$\operatorname{submitForm}$	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

15.4.4 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.

15.4.5 Local Functions

None

16 MIS of Access Token

16.1 Module

Access Token

16.2 Uses

None

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

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

16.4 Semantics

16.4.1 State Variables

• tokenValue: string

• expirationTime: Date

• userID: string

16.4.2 Environment Variables

None

16.4.3 Assumptions

None

16.4.4 Access Routine Semantics

isExpired():

• output: out := currentTime > expirationTime

renew():

• transition: expirationTime := expirationTime + 5 hours

16.4.5 Local Functions

None

17 MIS of Account Creation Interface

17.1 Module

Account Creation Interface

17.2 Uses

Account Creation Controller25

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	Enum[labeler, client]	-	_
$\operatorname{submitForm}$	list[(string, string)]	-	-

17.4 Semantics

17.4.1 State Variables

None

17.4.2 Environment Variables

win: 2D sequence of coloured pixels

17.4.3 Assumptions

None

17.4.4 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.

17.4.5 Local Functions

None

18 MIS of Account Database

18.1 Module

Account Database

18.2 Uses

Relational Database

18.3 Syntax

18.3.1 Exported Constants

18.3.2 Exported Access Programs

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

18.4 Semantics

18.4.1 State Variables

None

18.4.2 Environment Variables

databaseConnection: connection to relational database

18.4.3 Assumptions

None

18.4.4 Access Routine Semantics

insertUser(user):

• transition: Insert user into database through databaseConection.

retrieveUser(email):

• output:

```
\begin{cases} User where User.email == email, if userExists(email) null, otherwise
```

updateUser(user):

• transition:

```
Update user in database through databaseConection, if userExists(user.email)

Do nothing otherwise
```

userExists(email):

• output: out :=

 $\exists User \in Database s.t. User.email = email$

18.4.5 Local Functions

None

19 MIS of Account Update Interface

19.1 Module

Account Update Interface

19.2 Uses

Account Update Controller26

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	User	-	=
$\operatorname{submitForm}$	list[(string, string)]	-	-

19.4 Semantics

19.4.1 State Variables

None

19.4.2 Environment Variables

win: 2D sequence of coloured pixels

19.4.3 Assumptions

None

19.4.4 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.

19.4.5 Local Functions

None

20 MIS of Login Interface

20.1 Module

Login Interface

20.2 Uses

Authentication Controller27

20.3 Syntax

20.3.1 Exported Constants

None

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	_
$\operatorname{submitForm}$	list[(string, string)]	-	-

20.4 Semantics

20.4.1 State Variables

None

20.4.2 Environment Variables

win: 2D sequence of coloured pixels

20.4.3 Assumptions

20.4.4 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.

20.4.5 Local Functions

None

21 MIS of Access Token

21.1 Module

Access Token

21.2 Uses

None

21.3 Syntax

21.3.1 Exported Constants

None

21.3.2 Exported Access Programs

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

21.4 Semantics

21.4.1 State Variables

• tokenValue: string

• expirationTime: Date

• userID: string

21.4.2 Environment Variables

None

21.4.3 Assumptions

None

21.4.4 Access Routine Semantics

isExpired():

 $\bullet \ \, {\rm output} \colon {\rm out} := {\rm currentTime} > {\rm expirationTime}$

renew():

• transition: expirationTime := expirationTime + 5 hours

21.4.5 Local Functions

None

22 MIS of Labeler

22.1 Module

Labeler

22.2 Uses

Extends User24

22.3 Syntax

22.3.1 Exported Constants

22.3.2 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	-	

22.4 Semantics

22.4.1 State Variables

• firstName: string

• lastName: string

• skills: list[string]

• availability: int

22.4.2 Environment Variables

None

22.4.3 Assumptions

None

22.4.4 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):

 \bullet transition: skills := newSkills

setAvailability(newAvail):

• transition: availability := newAvail

22.4.5 Local Functions

None

23 MIS of Client

23.1 Module

Client

23.2 Uses

Extends User24

23.3 Syntax

23.3.1 Exported Constants

None

23.3.2 Exported Access Programs

Name	In	Out	Exceptions
getCompanyName	-	string	_
getIndustry	-	string	-
getTypicalProject	-	Image	-
setCompanyName	string	-	-
$\operatorname{setIndustry}$	string	-	-
setTypicalProject	string	-	-

23.4 Semantics

23.4.1 State Variables

• companyName: string

• industry: string

• typicalProject: string

23.4.2 Environment Variables

None

23.4.3 Assumptions

None

23.4.4 Access Routine Semantics

getCompanyName():

 $\bullet \ \, {\rm output} \colon \, {\rm out} := {\rm companyName}$

getIndustry():

• output: out := industry

getTypicalProject():

• output: out := typicalProject

setCompanyName(newcn):

 \bullet transition: company Name := new
cn

setIndustry(newIndustry):

• transition: industry := newIndustry

set Typical Project (new tp):

• transition: typicalProject := newtp

23.4.5 Local Functions

24 MIS of User

24.1 Module

User

24.2 Uses

None

24.3 Syntax

24.3.1 Exported Constants

None

24.3.2 Exported Access Programs

Name	In	Out	Exceptions
getEmail	-	string	-
getPasswor	rd -	string	-
getProfileP	Pic -	Image	-
setEmail	string	-	-
$\operatorname{setPasswor}$	d string	-	-
$\operatorname{setProfileP}$	ic string	-	-

24.4 Semantics

24.4.1 State Variables

• email: string

• password: string

• profilePic: image

24.4.2 Environment Variables

None

24.4.3 Assumptions

24.4.4 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

24.4.5 Local Functions

None

25 MIS of Account Creation Controller

25.1 Module

Account Creation Controller

25.2 Uses

Account Creation Interface17

Account Database18

User24

Labeler22

Client23

25.3 Syntax

25.3.1 Exported Constants

25.3.2 Exported Access Programs

Name	In	Out	Exceptions
validateForm	$\frac{1}{1}$ $\frac{1}$	boolean	
	Enum[labeler, client]		
createUser	list[(string, string)],	User	-
	Enum[labeler, client]		
uploadUser	User	-	DatabaseException

25.4 Semantics

25.4.1 State Variables

None

25.4.2 Environment Variables

None

25.4.3 Assumptions

Assumes AccountDatabase is operational when calling uploadUser.

25.4.4 Access Routine Semantics

validateForm(formData, userType):

 $\bullet \ \, output: \ \, out:= hasRequiredFields(formData, userFields) \land isValidEmail(formData.email) \land \\ isValidPassword(formData.password) \land \\$

```
hasRequiredFields(formData, labelerFields), if userType = "labeler" hasRequiredFields(formData, clientFields), if userType = "client" true, otherwise
```

Where:

```
userFields = {email, password}
labelerFields = {firstName, lastName, skills, availability}
clientFields = {companyName, industry, typicalProject}
```

createUser(formData, userType):

• output: out :=

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

uploadUser(newUser):

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

25.4.5 Local Functions

- hasRequiredFields(data, fields) = \forall field \in fields, (data[field] \neq "")
- isValidEmail(email) = email ∈ V ∧ email¬ ∈ 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\#$\%\&]{8,})$

26 MIS of Account Update Controller

26.1 Module

Account Update Controller

26.2 Uses

Account Update Interface19 Account Database18 User24

26.3 Syntax

26.3.1 Exported Constants

None

26.3.2 Exported Access Programs

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

26.4 Semantics

26.4.1 State Variables

• user: User

26.4.2 Environment Variables

None

26.4.3 Assumptions

Assumes AccountDatabase is operational when calling requestUpdate.

26.4.4 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.

26.4.5 Local Functions

27 MIS of Authentication Controller

27.1 Module

Authentication Controller

27.2 Uses

Login Interface41 Account Database18 Access Token21

27.3 Syntax

27.3.1 Exported Constants

None

27.3.2 Exported Access Programs

Name	In	Out	Exceptions
validCredentials	(string, string)	boolean	-
${\tt generateAccessToken}$	string	-	-

27.4 Semantics

27.4.1 State Variables

• token: AccessToken

27.4.2 Environment Variables

None

27.4.3 Assumptions

Assumes AccountDatabase is operational when calling validCredentials.

27.4.4 Access Routine Semantics

validCredentials(email, password):

output: out := AccountDatabase.retreiveUser(email) ≠ null
 ∧ AccountDatabase.retreiveUser(email).getPassword() == password

generateAccessToken(email):

• transition: token := AccessToken(email)

27.4.5 Local Functions

None

28 MIS of Satellite Image Request Interface

28.1 Module

Satellite Image Request Interface

28.2 Uses

Satellite Image Request Controller29

28.3 Syntax

28.3.1 Exported Constants

None

28.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
$\operatorname{submitForm}$	list[(string, string)]	-	-

28.4 Semantics

28.4.1 State Variables

None

28.4.2 Environment Variables

win: 2D sequence of coloured pixels

28.4.3 Assumptions

28.4.4 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.

28.4.5 Local Functions

None

29 MIS of Satellite Image Request Controller

29.1 Module

Satellite Image Request Controller

29.2 Uses

Satellite Image Request Interface28 Satellite Image Request30

29.3 Syntax

29.3.1 Exported Constants

None

29.3.2 Exported Access Programs

Name	In	Out	Exceptions
validateForm	list[(string, string)]	boolean	-
requestImages	${\bf Satellite Image Request}$	_	-

29.4 Semantics

29.4.1 State Variables

29.4.2 Environment Variables

None

29.4.3 Assumptions

None

29.4.4 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.

29.4.5 Local Functions

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

30 MIS of Satellite Image Request

30.1 Module

Satellite Image Request

30.2 Uses

None

30.3 Syntax

30.3.1 Exported Constants

30.3.2 Exported Access Programs

Name	In	Out	Exceptions
getLocation	-	(float, float)	-
getRadius	-	float	-
getDate	-	Date	-
$\operatorname{setLocation}$	(float, float)	-	-
$\operatorname{setRadius}$	float	-	-
setDate	Date	-	-

30.4 Semantics

30.4.1 State Variables

• locationX: float

• locationY: float

• radius: float

• date: Date

30.4.2 Environment Variables

None

30.4.3 Assumptions

None

30.4.4 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

30.4.5 Local Functions

None

31 MIS of Project Creation Interface

31.1 Module

Project Creation Interface

31.2 Uses

Project Creation Controller32

31.3 Syntax

31.3.1 Exported Constants

None

31.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
$\operatorname{submitForm}$	list[(string, string)]	-	-

31.4 Semantics

31.4.1 State Variables

None

31.4.2 Environment Variables

win: 2D sequence of coloured pixels

31.4.3 Assumptions

31.4.4 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.

31.4.5 Local Functions

None

32 MIS of Project Creation Controller

32.1 Module

Project Creation Controller

32.2 Uses

Project Manager8
Project Creation Interface31
Project33

32.3 Syntax

32.3.1 Exported Constants

None

32.3.2 Exported Access Programs

Name	${f In}$	Out	Exceptions
validateForm	list[(string, string)]	boolean	-
${\it create New Project}$	list[(string, string)]	Project	-

32.4 Semantics

32.4.1 State Variables

32.4.2 Environment Variables

None

32.4.3 Assumptions

None

32.4.4 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))

32.4.5 Local Functions

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

33 MIS of Project

33.1 Module

Project

33.2 Uses

None

33.3 Syntax

33.3.1 Exported Constants

33.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProjectID	-	int	-
getName	-	string	_
getDescription	-	string	-
${\it getLabelClasses}$	-	list[Enum[string]]	-
getTimePeriod	-	(Date, Date)	-
setName	string	-	-
setDescription	string	-	-
setLabelClasses	list[Enum[string]]	-	_
${\bf set Time Period}$	(Date, Date)	-	-

33.4 Semantics

33.4.1 State Variables

• projectID: int

• name: string

• description: string

• labelClasses: list[Enum[String]]

• startDate: Date

• endDate: Date

33.4.2 Environment Variables

None

33.4.3 Assumptions

None

33.4.4 Access Routine Semantics

getProjectID():

• output: out := projectID

getName():

 \bullet 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

33.4.5 Local Functions

None

34 MIS of Service Request Failure Interface

34.1 Module

Service Request Failure Interface

34.2 Uses

34.3 Syntax

34.3.1 Exported Constants

None

34.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayErrorInfo	-	-	-

34.4 Semantics

34.4.1 State Variables

None

34.4.2 Environment Variables

win: 2D sequence of coloured pixels

34.4.3 Assumptions

None

34.4.4 Access Routine Semantics

displayErrorInfo():

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

34.4.5 Local Functions

None

35 MIS of Image Upload Interface

35.1 Module

Image Upload Interface

35.2 Uses

Project Manager8

35.3 Syntax

35.3.1 Exported Constants

None

35.3.2 Exported Access Programs

Name	${f In}$	\mathbf{Out}	Exceptions
displayUploadImages		-	-

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

displayUploadImages():

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

35.4.5 Local Functions

• validateImage(image): out :=

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

36 MIS of Report Interface

36.1 Module

Report Interface

36.2 Uses

Report Controller37

36.3 Syntax

36.3.1 Exported Constants

None

36.3.2 Exported Access Programs

Name	${\bf In}$	Out	Exceptions
displaySta	ts -	-	-

36.4.1 State Variables

None

36.4.2 Environment Variables

win: 2D sequence of coloured pixels

36.4.3 Assumptions

None

36.4.4 Access Routine Semantics

displayStats():

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

36.4.5 Local Functions

None

37 MIS of Report Controller

37.1 Module

Report Controller

37.2 Uses

Report Interface36 Report38

37.3 Syntax

37.3.1 Exported Constants

None

37.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProjectStats	string	-	-
export Labeled Images	-	-	

37.4.1 State Variables

• report: Report

37.4.2 Environment Variables

fm: External systems file manager

37.4.3 Assumptions

None

37.4.4 Access Routine Semantics

getProjectStats(projectID):

- transition: report := Report of statistics for project with projectID exportLabeledImages():
 - transition: fm := given labeled images to download to device.

37.4.5 Local Functions

None

38 MIS of Report

38.1 Module

Report

38.2 Uses

None

38.3 Syntax

38.3.1 Exported Constants

38.3.2 Exported Access Programs

Name	In	Out	Exceptions
getLabeledIm	ages -	list[Image]	-
getReviewedI	mages -	$\operatorname{list}[\operatorname{Image}]$	-
getEndDate	-	Date	-
getTotalLabel	ers -	int	-
getAccuracy	-	int	-
getClassCoun	t -	list[(string, int)]	-

38.4 Semantics

38.4.1 State Variables

• labeledImages: list[Image]

 $\bullet \ \ reviewed Images: \ list[Image]$

• endDate: Date

• totalLabelers: int

• accuracyOfLabelers: int

• classCount: list[(string, int)]

38.4.2 Environment Variables

None

38.4.3 Assumptions

None

38.4.4 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

38.4.5 Local Functions

None

39 MIS of Project Selection Interface

39.1 Module

Project Selection Interface

39.2 Uses

Project Selection Controller40

39.3 Syntax

39.3.1 Exported Constants

None

39.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayActiveProjects	-	-	_

39.4 Semantics

39.4.1 State Variables

None

39.4.2 Environment Variables

win: 2D sequence of coloured pixels

39.4.3 Assumptions

None

39.4.4 Access Routine Semantics

displayActiveProjects():

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

39.4.5 Local Functions

None

40 MIS of Project Selection Controller

40.1 Module

Project Selection Controller

40.2 Uses

Project Collection Manager9 Project Selection Interface39 Project33

40.3 Syntax

40.3.1 Exported Constants

None

40.3.2 Exported Access Programs

Name	In	Out	Exceptions
getActiveProjects	-	-	-
selectProject	Project	-	

40.4 Semantics

40.4.1 State Variables

• activeProjects: list[Project]

40.4.2 Environment Variables

win: 2D sequence of coloured pixels

40.4.3 Assumptions

None

40.4.4 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

40.4.5 Local Functions

None

41 MIS of Labeling Interface

41.1 Module

Labeling Interface

41.2 Uses

Labeling Controller42 Image43

41.3 Syntax

41.3.1 Exported Constants

None

41.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
displayImage	Image	=	-
skipImage	-	-	-
${\bf selectLabelClass}$	-	-	-

41.4.1 State Variables

• projectImages: list[Image]

• currImage: int

• currLabelClass: Enum[string]

41.4.2 Environment Variables

win: 2D sequence of coloured pixels

41.4.3 Assumptions

None

41.4.4 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.

41.4.5 Local Functions

None

42 MIS of Labeling Controller

42.1 Module

Labeling Controller

42.2 Uses

Labeling Interface41 Label4

42.3 Syntax

42.3.1 Exported Constants

None

42.3.2 Exported Access Programs

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

42.4 Semantics

42.4.1 State Variables

• labels: list[Label]

42.4.2 Environment Variables

None

42.4.3 Assumptions

None

42.4.4 Access Routine Semantics

getProjectImages(projectID):

- output: out := All images from project with projectID addLabel(lbl):
 - transition: labels := labels \cup {lbl}

removeLabel(lblID):

- transition: labels := $\{\ell \in labels \mid \ell.id \neq lblID\}$ submitLabels(lbls):
 - transition: labels are sent to be added to the Label Database

42.4.5 Local Functions

None

43 MIS of Image

43.1 Module

Image

43.2 Uses

None

43.3 Syntax

43.3.1 Exported Constants

None

43.3.2 Exported Access Programs

Name	In	Out	Exceptions
getProjectID	-	int	_
$\operatorname{getImageID}$	-	int	=
getDimensions	-	(float, float)	-
${\rm getImageData}$	-	binary	-

43.4 Semantics

43.4.1 State Variables

• projectID: int

• imageID: int

• width: float

• height: float

• imageData: binary

43.4.2 Environment Variables

43.4.3 Assumptions

None

43.4.4 Access Routine Semantics

getProjectID():

• output: out := projectID

getImageID():

 \bullet output: out := imageID

getDimensions():

• output: out := (width, height)

getImageData():

• output: out := imageData

43.4.5 Local Functions

44 MIS of Label Server

44.1 Module

Label Server

44.2 Uses

Labeling Controller42 Label Database Connector45

44.3 Syntax

44.3.1 Exported Constants

None

44.3.2 Exported Access Programs

Name	In	Out	Exceptions
acceptLabel	Label	-	ValueError,
			Connec-
			tionError

44.4 Semantics

44.4.1 State Variables

None

44.4.2 Environment Variables

LabelDatabaseConnector

44.4.3 Assumptions

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

44.4.4 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 $\neg(o \in L)$ Throw ConnectionError if ConnectionError is raised by LabelDatabaseConnector

44.4.5 Local Functions

JSONLabeltoLabel: converts a JSON object into a Label object.

45 MIS of Label Database Connector

45.1 Module

Label Database Connector

45.2 Uses

Label Database46

45.3 Syntax

45.3.1 Exported Constants

None

45.3.2 Exported Access Programs

Name	\mathbf{In}	\mathbf{Out}	Exceptions
pushLabel	Label	-	ValueError,
			Connec-
			tionError
makeDB	Label	-	ConnectionError
Connec-			
tion			
getLabels	String	list[Label]	ValueError,
			Connec-
			tionError

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

pushLabel(Label l):

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

45.4.5 Local Functions

46 MIS of Label Database

46.1 Module

Label Database

46.2 Uses

None

46.3 Syntax

46.3.1 Exported Constants

None

46.3.2 Exported Access Programs

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

46.4 Semantics

46.4.1 State Variables

labels: labels 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

pushLabel(Label 1):

- transition: labels := labels $\cup l$
- exception: Let L be the set of valid Labels. Throw ValueError if $\neg(l \in L)$ 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)

46.4.5 Local Functions

47 MIS of ImageObject Database Connector

47.1 Module

ImageObject Database Connector

47.2 Uses

ImageObject Database48

47.3 Syntax

47.3.1 Exported Constants

None

47.3.2 Exported Access Programs

\mathbf{Name}	${f In}$	\mathbf{Out}	Exceptions
push Im-	ImageObject	-	ValueError,
age Object			Connec-
			tionError
makeDB	ImageObject	-	ConnectionError
Connec-			
tion			
get Image	String	list[ImageObject]	ValueError,
Objects			Connec-
			tionError

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

pushLabel(ImageObject 1):

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

47.4.5 Local Functions

48 MIS of ImageObject Database

48.1 Module

ImageObject Database

48.2 Uses

None

48.3 Syntax

48.3.1 Exported Constants

None

48.3.2 Exported Access Programs

Name	In	Out	Exceptions
push Im-	ImageObject	-	ValueError
age Object			
makeDB	ImageObject	-	ConnectionError
Connec-			
tion			
get Image	String	list[ImageObject]	ValueError
Objects			

48.4 Semantics

48.4.1 State Variables

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

48.4.2 Environment Variables

None

48.4.3 Assumptions

48.4.4 Access Routine Semantics

pushLabel(ImageObject 1):

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

48.4.5 Local Functions

Non	e						
=	=	=	=	=	=	=	=

49 MIS of Labeller Database Connector

49.1 Module

Labeller Database Connector

49.2 Uses

Labeller Database 50

49.3 Syntax

49.3.1 Exported Constants

None

49.3.2 Exported Access Programs

Name	In	\mathbf{Out}	Exceptions
push la-	labeller	-	ValueError,
beller			Connec-
			tionError
makeDB	credentials	-	ConnectionError
Connec-			
tion			
get labeller	String	list[labeller]	ValueError,
			Connec-
			tionError

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

pushLabeller(Labeller o):

- transition: Transition occurs in Labeller Database
- exception: Let O be the set of valid Labellers. Throw ValueError if $\neg(o \in O)$ 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 $\neg (q \in Q)$ Throw ConnectionError if ConnectionError is raised by makeDBConnection

49.4.5 Local Functions

50 MIS of Labeller Database

50.1 Module

Labeller Database

50.2 Uses

None

50.3 Syntax

50.3.1 Exported Constants

None

50.3.2 Exported Access Programs

Name	In	Out	Exceptions
push La-	Labeller	-	ValueError
beller			
makeDB	Credentials	-	ConnectionError
Connec-			
tion			
get La-	String	list[Labeller]	ValueError
beller			

50.4 Semantics

50.4.1 State Variables

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

50.4.2 Environment Variables

None

50.4.3 Assumptions

50.4.4 Access Routine Semantics

pushLabeller(Labeller o):

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

50.4.5 Local Functions

51 MIS of Object Extraction Manager

51.1 Module

Object Extraction Manager

51.2 Uses

ImageObject Database Connector47 Label Database Connector45 Labeller Database Connector49 Image Prior Analyzer54 Label Confidence Service52 Object Extraction Service53 Labeller Expertise Calculator55

51.3 Syntax

51.3.1 Exported Constants

None

51.3.2 Exported Access Programs

\mathbf{Name}	${f In}$	\mathbf{Out}	Exceptions
getObjects	projectID	-	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(ProjectID p):

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

 \bullet exception: Let P be the set of assigned Project IDs. Throw ValueError if $\neg(p \in \mathbf{P})$

51.4.5 Local Functions

52 MIS of Label Confidence Service

52.1 Module

Label Confidence Service

52.2 Uses

None

52.3 Syntax

52.3.1 Exported Constants

None

52.3.2 Exported Access Programs

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

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

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 Value Error 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(\mathrm{imageobject} \in \mathbf{I}))
```

 $\bullet \ \ \text{calculation: confidence} = (|\text{Likelihood} - (1 - \text{prediction})|)$

52.4.5 Local Functions

53 MIS of Object Extraction Service

53.1 Module

Object Extraction Service

53.2 Uses

None

53.3 Syntax

53.3.1 Exported Constants

None

53.3.2 Exported Access Programs

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

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

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

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

Let X be the set of valid Labellers. Throw ValueError if $(\exists labeller \in labellers]$:

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

53.4.5 Local Functions

54 MIS of Image Prior Analyzer

54.1 Module

Image Prior Analyzer

54.2 Uses

None

54.3 Syntax

54.3.1 Exported Constants

None

54.3.2 Exported Access Programs

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

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 $(\exists image \in images | : \neg(image \in I))$

54.4.5 Local Functions

55 MIS of Labeller Expertise Calculator

55.1 Module

Labeller Expertise Calculator

55.2 Uses

None

55.3 Syntax

55.3.1 Exported Constants

None

55.3.2 Exported Access Programs

Name	In	Out		Exceptions
getExpertise	e list[label],	list[dict[string,	tu-	ValueError
	list[labeller],	ple[float, float]]]		
	list[ImageObject],	-		
	list[list[float]]			

55.4 Semantics

55.4.1 State Variables

None

55.4.2 Environment Variables

None

55.4.3 Assumptions

55.4.4 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 ($\exists i, j | x = confidence[i][j] : \neg(x \in \mathbb{R})$)

• Formula for Expertise: Each labeller contains a float value alpha and beta. For each pixel (i, j) in an image, if the consensus matches the user prediction:

$$\begin{split} \alpha &= \alpha + \frac{\text{confidence[i][j]}}{\text{size_of_image}} \\ \text{If it does not match,} \\ \beta &= \beta + \frac{\text{confidence[i][j]}}{\text{size_of_image}} \end{split}$$

55.4.5 Local Functions

56 MIS of Image Service Manager

56.1 Module

Image Service Manager

56.2 Uses

ImageObject Database Connector47 Labeller Database Connector49 Image Mask Service57 Image Selection Service58

56.3 Syntax

56.3.1 Exported Constants

None

56.3.2 Exported Access Programs

Name	In		Out	Exceptions
getNextIn	nagdabellerID,	projectID,	List[Image]	ValueError
	int			

56.4 Semantics

56.4.1 State Variables

None

56.4.2 Environment Variables

None

56.4.3 Assumptions

56.4.4 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 $\neg(p \in P)$ Let L be the set of assigned LabellerIDs. Throw ValueError if $\neg(l \in L)$ Throw ValueError if $\neg(n \in \mathbb{N})$

56.4.5 Local Functions

57 MIS of Image Mask Service

57.1 Module

Image Mask Service

57.2 Uses

None

57.3 Syntax

57.3.1 Exported Constants

None

57.3.2 Exported Access Programs

Name In	Out	Exceptions
getImageMas k mage	Image	ValueError

57.4 Semantics

57.4.1 State Variables

None

57.4.2 Environment Variables

None

57.4.3 Assumptions

57.4.4 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 $\neg (i \in I)$

57.4.5 Local Functions

58 MIS of Image Selection Service

58.1 Module

Image Selection Service

58.2 Uses

58.3 Syntax

58.3.1 Exported Constants

None

58.3.2 Exported Access Programs

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

58.4 Semantics

58.4.1 State Variables

None

58.4.2 Environment Variables

None

58.4.3 Assumptions

58.4.4 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 $(\exists imageobject \in imageobjects | : \neg(imageobject \in I))$

58.4.5 Local Functions

59 MIS of ModelComparisonEvaluation

59.1 6.1 Module

Name: ModelComparisonEvaluation

59.2 6.2 Uses

- TestDataset (Holds test samples and true labels)
- EvaluationResult (Stores metrics from an evaluation)

59.3 6.3 Syntax

59.3.1 6.3.1 Exported Constants

None

59.3.2 6.3.2 Exported Access Programs

Name	In		Out	Exceptions
evaluateModel	String modelId,	Test-	EvaluationResult	ModelNotFoundError,
	Dataset testData			ValueError

59.4 6.4 Semantics

59.4.1 6.4.1 State Variables

- comparisonMetrics: Map<String, Float> (Stores metric-name to numeric value)
- benchmarkModelId: String (ID of the benchmark model)

59.4.2 6.4.2 Environment Variables

None

59.4.3 6.4.3 Assumptions

- The modelId provided must exist in the system.
- testData must be valid and non-empty.

 $evaluate Model (model Id, \ test Data):$

- **transition**: Updates comparisonMetrics by comparing the given model with the benchmark.
- output: Returns an EvaluationResult with metrics (e.g., accuracy, precision).
- exception:
 - ModelNotFoundError if modelId does not exist.
 - ValueError if testData is invalid.

59.4.5 6.4.5 Local Functions

60 MIS of CrossValidationEvaluation

60.1 6.1 Module

Name: CrossValidationEvaluation

60.2 6.2 Uses

- TestDataset
- EvaluationResult

60.3 6.3 Syntax

60.3.1 Exported Constants

None

60.3.2 Exported Access Programs

Name	In		Out	Exceptions
evaluateModel	String modelId,	Test-	EvaluationResult	$\overline{\text{ModelNotFoundE}}$ rror,
	Dataset testData			ValueError

60.4 6.4 Semantics

60.4.1 State Variables

• kFolds: Integer

• ValidationMetrics: Map<String, Float> (Aggregated cross-validation metrics)

60.4.2 Environment Variables

None

60.4.3 6.4.3 Assumptions

- kFolds ≥ 2 .
- testData is large enough for multiple folds.

evaluateModel(modelId, testData):

- transition: Runs cross-validation and updates ValidationMetrics.
- output: An EvaluationResult (e.g., average accuracy).
- exception:
 - ModelNotFoundError if the model does not exist.
 - ValueError if testData is invalid or too small.

60.4.5 6.4.5 Local Functions

61 MIS of ModelTrainingService

61.1 6.1 Module

Name: ModelTrainingService

61.2 6.2 Uses

- TrainingParams
- TrainingData
- ModelConfig
- TrainingResult

61.3 6.3 Syntax

61.3.1 6.3.1 Exported Constants

None

61.3.2 Exported Access Programs

Name	In	Out	Exceptions
trainModel	TrainingData data, Mod-	TrainingResult	ValueError,
	elConfig modelConfig		ResourceU-
			navailable Error
stopTraining	String modelId	-	$\overline{\text{ModelNotFoundE}}$ rror

61.4 6.4 Semantics

61.4.1 6.4.1 State Variables

- trainingParameters: TrainingParams
- trainingStatus: String ("Not Started", "In Progress", "Completed", etc.)

61.4.2 6.4.2 Environment Variables

None

61.4.3 6.4.3 Assumptions

• System has enough resources (GPU, memory) to train the model.

trainModel(data, modelConfig):

- transition: Sets trainingStatus to "In Progress" and, upon completion, "Completed".
- output: Returns a TrainingResult with metrics (loss, accuracy, etc.).
- exception:
 - ValueError if data or modelConfig is invalid.
 - Resource Unavailable
Error if required resources are not available.

stopTraining(modelId):

- transition: If the model is training, changes status to "Stopped" or "Cancelled".
- exception:
 - ModelNotFoundError if the model does not exist or is not training.

61.4.5 6.4.5 Local Functions

62 MIS of ModelEvaluationService

62.1 6.1 Module

Name: ModelEvaluationService

62.2 6.2 Uses

- TestDataset
- EvaluationResult

62.3 6.3 Syntax

62.3.1 6.3.1 Exported Constants

None

62.3.2 Exported Access Programs

Name	In		Out	Exceptions
evaluateModel	String modelId,	Test-	EvaluationResult	$\overline{\text{ModelNotFoundE}}$ rror,
	Dataset testData			ValueError
fetchEvaluationMe	etricstring modelId		Map;String,Float;	$\overline{\text{ModelNotFoundE}}$ rror

62.4 6.4 Semantics

62.4.1 6.4.1 State Variables

- evaluationMetrics: Map<String, Float>
- valuationStatus: String ("Pending", "In Progress", "Completed")

62.4.2 Environment Variables

None

62.4.3 6.4.3 Assumptions

• The modelId references a trained model.

evaluateModel(modelId, testData):

- transition: Sets valuationStatus to "In Progress" and updates evaluationMetrics.
- output: An EvaluationResult (accuracy, loss, etc.).
- exception:
 - ModelNotFoundError if modelId is invalid.
 - ValueError if testData is invalid or empty.

fetchEvaluationMetrics(modelId):

- output: Returns the evaluationMetrics for the model.
- exception:
 - ModelNotFoundError if the modelId does not exist or no metrics are found.

62.4.5 6.4.5 Local Functions

63 MIS of ModelManager

63.1 6.1 Module

Name: ModelManager

63.2 6.2 Uses

- ModelParameters
- MLModel

63.3 6.3 Syntax

63.3.1 6.3.1 Exported Constants

None

63.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
createModel	ModelParameters	-	ValueError
	params		
updateModelStatus	String modelId, String	-	$\overline{ModelNotFoundE}rror$
	status		
fetchModel	String modelId	MLModel	$\overline{ModelNotFoundE}rror$
deleteModel	String modelId	-	$\overline{\text{ModelNotFoundE}}$ rror

63.4 6.4 Semantics

63.4.1 6.4.1 State Variables

• modelID: String

• status: String ("Training", "Evaluating", "Completed", etc.)

• createdAt: Date

• updatedAt: Date

63.4.2 6.4.2 Environment Variables

63.4.3 6.4.3 Assumptions

• A unique modelID is generated upon creation.

63.4.4 6.4.4 Access Routine Semantics

createModel(params):

- **transition**: Instantiates a new MLModel, sets modelID, createdAt, updatedAt, status = "Created".
- exception:
 - ValueError if params are invalid.

updateModelStatus(modelId, status):

- transition: Updates status and updatedAt of the specified model.
- exception:
 - ModelNotFoundError if the modelId does not exist.

fetchModel(modelId):

- output: Returns the MLModel object.
- exception:
 - ModelNotFoundError if no model with modelId exists.

deleteModel(modelId):

- transition: Removes the model from storage.
- exception:
 - ModelNotFoundError if modelId is invalid.

63.4.5 6.4.5 Local Functions

64 MIS of ModelCreation (Abstract)

64.1 6.1 Module

Name: ModelCreation (Abstract Base Class)

64.2 6.2 Uses

- ModelParameters
- MLModel

64.3 6.3 Syntax

64.3.1 6.3.1 Exported Constants

None

64.3.2 Exported Access Programs

Name	In	Out	Exceptions
createModel	-	MLModel	$\overline{\text{NotImplementedError}}$

64.4 6.4 Semantics

64.4.1 6.4.1 State Variables

• modelType: String

• creationParams: ModelParameters

64.4.2 6.4.2 Environment Variables

None

64.4.3 6.4.3 Assumptions

• Concrete subclasses must override the createModel method.

64.4.4 6.4.4 Access Routine Semantics

createModel():

- output: A fully instantiated MLModel.
- exception:
 - NotImplementedError if called from the abstract class.

64.4.5 6.4.5 Local Functions

65 MIS of MLModelDatabase

65.1 6.1 Module

Name: MLModelDatabase

65.2 6.2 Uses

• MLModel

65.3 6.3 Syntax

65.3.1 6.3.1 Exported Constants

None

65.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
saveModel	MLModel model	-	DatabaseError
fetchModel	String modelId	MLModel	ModelNotFoundError,
			DatabaseError
deleteModel	String modelId	-	ModelNotFoundError,
			DatabaseError
updateModel	String modelId,	-	$\overline{\text{ModelNotFoundE}}$ rror,
	MapiString, Anyi		DatabaseError
	updates		

65.4 6.4 Semantics

65.4.1 6.4.1 State Variables

• dbConnection: Connection (Active DB connection)

65.4.2 Environment Variables

• External database system (accessed via dbConnection)

65.4.3 6.4.3 Assumptions

• dbConnection is valid and open.

saveModel(model):

- transition: Inserts or updates the model in the database.
- exception:
 - DatabaseError if insertion fails.

fetchModel(modelId):

- output: Returns the MLModel from the database.
- exception:
 - ModelNotFoundError if the modelId is not found.
 - DatabaseError if a DB error occurs.

deleteModel(modelId):

- transition: Removes the model record.
- exception:
 - ModelNotFoundError if modelId is not found.
 - DatabaseError on DB error.

updateModel(modelId, updates):

- transition: Updates the specified fields of the model in the database.
- exception:
 - ModelNotFoundError if modelId is not found.
 - DatabaseError if the update operation fails.

65.4.5 6.4.5 Local Functions

66 MIS of OtherModelCreation

66.1 6.1 Module

Name: OtherModelCreation

66.2 6.2 Uses

- MLModel
- ModelCreation (abstract base class)

66.3 6.3 Syntax

66.3.1 Exported Constants

None

66.3.2 Exported Access Programs

Name	In	Out	Exceptions
createModel	-	MLModel	ValueError

66.4 6.4 Semantics

66.4.1 6.4.1 State Variables

- modelType: String (e.g., "Decision Tree", "SVM")
- hyperparameters: Map<String, Any>

66.4.2 6.4.2 Environment Variables

None

66.4.3 6.4.3 Assumptions

• hyperparameters are valid for modelType.

66.4.4 6.4.4 Access Routine Semantics

createModel():

- output: Returns an instantiated MLModel of modelType.
- exception:
 - ValueError if the modelType/hyperparameters combination is invalid.

66.4.5 **6.4.5** Local Functions

None

67 MIS of CNNModelCreation

67.1 6.1 Module

Name: CNNModelCreation

67.2 6.2 Uses

- ModelCreation (abstract)
- MLModel

67.3 6.3 Syntax

67.3.1 6.3.1 Exported Constants

None

67.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
createModel	-	MLModel	ValueError

67.4 6.4 Semantics

67.4.1 6.4.1 State Variables

- layers: List<LayerConfig> (Defines structure of each CNN layer)
- activatedFunctions: List<String> (Activation functions for each layer)

67.4.2 6.4.2 Environment Variables

None

67.4.3 6.4.3 Assumptions

• The layers and activatedFunctions lists are valid and aligned.

createModel():

- output: Instantiates a CNN MLModel with specified layers and activation functions.
- exception:
 - ValueError if layers or activatedFunctions are invalid or mismatched.

67.4.5 6.4.5 Local Functions

None

68 Exception Handling

The application implements a structured approach to exception handling across both frontend and backend components.

68.1 Frontend Handling (React)

In the frontend, exceptions are typically caught using try-catch blocks or via global error boundaries for unhandled UI errors. Frontend errors (e.g., API failures, rendering issues) are logged to the browser console. User-friendly fallback UIs are displayed where applicable.

68.2 Backend Handling (Python)

On the backend, exceptions are handled at multiple levels—locally in functions with try-catch blocks, or via packages (e.g., Flask/Waiter error handlers). Critical exceptions (e.g., database failures) are logged with severity levels (DEBUG, INFO, WARNING, ERROR, CRITICAL) using structured logging. Some exceptions are propagated to the frontend as HTTP error responses with sanitized messages to avoid exposing sensitive details.

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.

69 Appendix

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) While our current design addresses core functionality, key limitations include the lack of financial features (e.g., payment processing or fund distribution) and support for media types beyond images, such as videos. Additionally, the accuracy and reliability of our ML models and labeling services could be improved. With more resources, we would implement human-in-the-loop validation to manually review uncertain model predictions during training, adopt active learning to prioritize low-confidence samples for human annotation, and enforce inter-annotator agreement checks to reduce labeling inconsistencies. We would also audit the training data for biases and introduce synthetic examples to cover edge cases, while allowing end-users to flag incorrect labels for model retraining.
- 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)