

Module Interface Specification for Software Engineering

Team #11, OKKM Insights

Mathew Petronilho

Oleg Glotov

Kyle McMaster

Kartik Chaudhari

January 18, 2025

1 Revision History

Date		Version	Notes
Date	January	1.0	
17th			

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [<https://github.com/OKKM-insights/OKKM.insights/tree/main/docs/SRS—SS>]

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	Introduction	1
4	Notation	1
5	Module Decomposition	8
6	MIS of Account Creation Interface	10
6.1	Module	10
6.2	Uses	10
6.3	Syntax	10
6.3.1	Exported Constants	10
6.3.2	Exported Access Programs	10
6.4	Semantics	11
6.4.1	State Variables	11
6.4.2	Environment Variables	11
6.4.3	Assumptions	11
6.4.4	Access Routine Semantics	11
6.4.5	Local Functions	11
7	MIS of Account Database Connector	11
7.1	Module	11
7.2	Uses	11
7.3	Syntax	11
7.3.1	Exported Constants	11
7.3.2	Exported Access Programs	12
7.4	Semantics	12
7.4.1	State Variables	12
7.4.2	Environment Variables	12
7.4.3	Assumptions	12
7.4.4	Access Routine Semantics	12
7.4.5	Local Functions	13
8	MIS of Account Database	13
8.1	Module	13
8.2	Uses	13
8.3	Syntax	13
8.3.1	Exported Constants	13
8.3.2	Exported Access Programs	13

8.4	Semantics	13
8.4.1	State Variables	13
8.4.2	Environment Variables	13
8.4.3	Assumptions	13
8.4.4	Access Routine Semantics	14
8.4.5	Local Functions	14
9	MIS of Account Update Interface	14
9.1	Module	14
9.2	Uses	14
9.3	Syntax	14
9.3.1	Exported Constants	14
9.3.2	Exported Access Programs	15
9.4	Semantics	15
9.4.1	State Variables	15
9.4.2	Environment Variables	15
9.4.3	Assumptions	15
9.4.4	Access Routine Semantics	15
9.4.5	Local Functions	15
10	MIS of Login Interface	15
10.1	Module	15
10.2	Uses	15
10.3	Syntax	16
10.3.1	Exported Constants	16
10.3.2	Exported Access Programs	16
10.4	Semantics	16
10.4.1	State Variables	16
10.4.2	Environment Variables	16
10.4.3	Assumptions	16
10.4.4	Access Routine Semantics	16
10.4.5	Local Functions	16
11	MIS of Access Token	16
11.1	Module	16
11.2	Uses	17
11.3	Syntax	17
11.3.1	Exported Constants	17
11.3.2	Exported Access Programs	17
11.4	Semantics	17
11.4.1	State Variables	17
11.4.2	Environment Variables	17
11.4.3	Assumptions	17

11.4.4	Access Routine Semantics	17
11.4.5	Local Functions	17
12	MIS of Account Creation Interface	18
12.1	Module	18
12.2	Uses	18
12.3	Syntax	18
12.3.1	Exported Constants	18
12.3.2	Exported Access Programs	18
12.4	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	18
12.4.5	Local Functions	19
13	MIS of Account Database	19
13.1	Module	19
13.2	Uses	19
13.3	Syntax	19
13.3.1	Exported Constants	19
13.3.2	Exported Access Programs	19
13.4	Semantics	19
13.4.1	State Variables	19
13.4.2	Environment Variables	19
13.4.3	Assumptions	19
13.4.4	Access Routine Semantics	20
13.4.5	Local Functions	20
14	MIS of Account Update Interface	20
14.1	Module	20
14.2	Uses	20
14.3	Syntax	20
14.3.1	Exported Constants	20
14.3.2	Exported Access Programs	21
14.4	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
14.4.5	Local Functions	21

15 MIS of Login Interface	21
15.1 Module	21
15.2 Uses	21
15.3 Syntax	22
15.3.1 Exported Constants	22
15.3.2 Exported Access Programs	22
15.4 Semantics	22
15.4.1 State Variables	22
15.4.2 Environment Variables	22
15.4.3 Assumptions	22
15.4.4 Access Routine Semantics	22
15.4.5 Local Functions	22
16 MIS of Access Token	22
16.1 Module	22
16.2 Uses	23
16.3 Syntax	23
16.3.1 Exported Constants	23
16.3.2 Exported Access Programs	23
16.4 Semantics	23
16.4.1 State Variables	23
16.4.2 Environment Variables	23
16.4.3 Assumptions	23
16.4.4 Access Routine Semantics	23
16.4.5 Local Functions	23
17 MIS of Labeler	24
17.1 Module	24
17.2 Uses	24
17.3 Syntax	24
17.3.1 Exported Constants	24
17.3.2 Exported Access Programs	24
17.4 Semantics	24
17.4.1 State Variables	24
17.4.2 Environment Variables	24
17.4.3 Assumptions	24
17.4.4 Access Routine Semantics	25
17.4.5 Local Functions	25
18 MIS of Client	25
18.1 Module	25
18.2 Uses	25
18.3 Syntax	26

18.3.1	Exported Constants	26
18.3.2	Exported Access Programs	26
18.4	Semantics	26
18.4.1	State Variables	26
18.4.2	Environment Variables	26
18.4.3	Assumptions	26
18.4.4	Access Routine Semantics	26
18.4.5	Local Functions	27
19	MIS of User	27
19.1	Module	27
19.2	Uses	27
19.3	Syntax	27
19.3.1	Exported Constants	27
19.3.2	Exported Access Programs	27
19.4	Semantics	27
19.4.1	State Variables	27
19.4.2	Environment Variables	28
19.4.3	Assumptions	28
19.4.4	Access Routine Semantics	28
19.4.5	Local Functions	28
20	MIS of Account Creation Controller	28
20.1	Module	28
20.2	Uses	29
20.3	Syntax	29
20.3.1	Exported Constants	29
20.3.2	Exported Access Programs	29
20.4	Semantics	29
20.4.1	State Variables	29
20.4.2	Environment Variables	29
20.4.3	Assumptions	29
20.4.4	Access Routine Semantics	29
20.4.5	Local Functions	30
21	MIS of Account Update Controller	31
21.1	Module	31
21.2	Uses	31
21.3	Syntax	31
21.3.1	Exported Constants	31
21.3.2	Exported Access Programs	31
21.4	Semantics	31
21.4.1	State Variables	31

21.4.2	Environment Variables	31
21.4.3	Assumptions	31
21.4.4	Access Routine Semantics	31
21.4.5	Local Functions	32
22	MIS of Authentication Controller	32
22.1	Module	32
22.2	Uses	32
22.3	Syntax	32
22.3.1	Exported Constants	32
22.3.2	Exported Access Programs	32
22.4	Semantics	32
22.4.1	State Variables	32
22.4.2	Environment Variables	33
22.4.3	Assumptions	33
22.4.4	Access Routine Semantics	33
22.4.5	Local Functions	33
23	MIS of Satellite Image Request Interface	33
23.1	Module	33
23.2	Uses	33
23.3	Syntax	33
23.3.1	Exported Constants	33
23.3.2	Exported Access Programs	33
23.4	Semantics	34
23.4.1	State Variables	34
23.4.2	Environment Variables	34
23.4.3	Assumptions	34
23.4.4	Access Routine Semantics	34
23.4.5	Local Functions	34
24	MIS of Satellite Image Request Controller	34
24.1	Module	34
24.2	Uses	34
24.3	Syntax	34
24.3.1	Exported Constants	34
24.3.2	Exported Access Programs	35
24.4	Semantics	35
24.4.1	State Variables	35
24.4.2	Environment Variables	35
24.4.3	Assumptions	35
24.4.4	Access Routine Semantics	35
24.4.5	Local Functions	35

25 MIS of Satellite Image Request	35
25.1 Module	35
25.2 Uses	35
25.3 Syntax	36
25.3.1 Exported Constants	36
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	37
26 MIS of Project Creation Interface	37
26.1 Module	37
26.2 Uses	37
26.3 Syntax	37
26.3.1 Exported Constants	37
26.3.2 Exported Access Programs	37
26.4 Semantics	37
26.4.1 State Variables	37
26.4.2 Environment Variables	37
26.4.3 Assumptions	38
26.4.4 Access Routine Semantics	38
26.4.5 Local Functions	38
27 MIS of Project Creation Controller	38
27.1 Module	38
27.2 Uses	38
27.3 Syntax	38
27.3.1 Exported Constants	38
27.3.2 Exported Access Programs	38
27.4 Semantics	39
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	39
28 MIS of Project	39
28.1 Module	39
28.2 Uses	39
28.3 Syntax	39

28.3.1	Exported Constants	39
28.3.2	Exported Access Programs	40
28.4	Semantics	40
28.4.1	State Variables	40
28.4.2	Environment Variables	40
28.4.3	Assumptions	40
28.4.4	Access Routine Semantics	40
28.4.5	Local Functions	41
29	MIS of Service Request Failure Interface	41
29.1	Module	41
29.2	Uses	41
29.3	Syntax	41
29.3.1	Exported Constants	41
29.3.2	Exported Access Programs	41
29.4	Semantics	42
29.4.1	State Variables	42
29.4.2	Environment Variables	42
29.4.3	Assumptions	42
29.4.4	Access Routine Semantics	42
29.4.5	Local Functions	42
30	MIS of Image Upload Interface	42
30.1	Module	42
30.2	Uses	42
30.3	Syntax	42
30.3.1	Exported Constants	42
30.3.2	Exported Access Programs	42
30.4	Semantics	43
30.4.1	State Variables	43
30.4.2	Environment Variables	43
30.4.3	Assumptions	43
30.4.4	Access Routine Semantics	43
30.4.5	Local Functions	43
31	MIS of Report Interface	43
31.1	Module	43
31.2	Uses	43
31.3	Syntax	43
31.3.1	Exported Constants	43
31.3.2	Exported Access Programs	43
31.4	Semantics	44
31.4.1	State Variables	44

31.4.2	Environment Variables	44
31.4.3	Assumptions	44
31.4.4	Access Routine Semantics	44
31.4.5	Local Functions	44
32	MIS of Report Controller	44
32.1	Module	44
32.2	Uses	44
32.3	Syntax	44
32.3.1	Exported Constants	44
32.3.2	Exported Access Programs	44
32.4	Semantics	45
32.4.1	State Variables	45
32.4.2	Environment Variables	45
32.4.3	Assumptions	45
32.4.4	Access Routine Semantics	45
32.4.5	Local Functions	45
33	MIS of Report	45
33.1	Module	45
33.2	Uses	45
33.3	Syntax	45
33.3.1	Exported Constants	45
33.3.2	Exported Access Programs	46
33.4	Semantics	46
33.4.1	State Variables	46
33.4.2	Environment Variables	46
33.4.3	Assumptions	46
33.4.4	Access Routine Semantics	46
33.4.5	Local Functions	47
34	MIS of Project Selection Interface	47
34.1	Module	47
34.2	Uses	47
34.3	Syntax	47
34.3.1	Exported Constants	47
34.3.2	Exported Access Programs	47
34.4	Semantics	47
34.4.1	State Variables	47
34.4.2	Environment Variables	47
34.4.3	Assumptions	48
34.4.4	Access Routine Semantics	48
34.4.5	Local Functions	48

35 MIS of Project Selection Controller	48
35.1 Module	48
35.2 Uses	48
35.3 Syntax	48
35.3.1 Exported Constants	48
35.3.2 Exported Access Programs	48
35.4 Semantics	48
35.4.1 State Variables	48
35.4.2 Environment Variables	49
35.4.3 Assumptions	49
35.4.4 Access Routine Semantics	49
35.4.5 Local Functions	49
36 MIS of Labeling Interface	49
36.1 Module	49
36.2 Uses	49
36.3 Syntax	49
36.3.1 Exported Constants	49
36.3.2 Exported Access Programs	49
36.4 Semantics	50
36.4.1 State Variables	50
36.4.2 Environment Variables	50
36.4.3 Assumptions	50
36.4.4 Access Routine Semantics	50
36.4.5 Local Functions	50
37 MIS of Labeling Controller	50
37.1 Module	50
37.2 Uses	51
37.3 Syntax	51
37.3.1 Exported Constants	51
37.3.2 Exported Access Programs	51
37.4 Semantics	51
37.4.1 State Variables	51
37.4.2 Environment Variables	51
37.4.3 Assumptions	51
37.4.4 Access Routine Semantics	51
37.4.5 Local Functions	52
38 MIS of Image	52
38.1 Module	52
38.2 Uses	52
38.3 Syntax	52

38.3.1	Exported Constants	52
38.3.2	Exported Access Programs	52
38.4	Semantics	52
38.4.1	State Variables	52
38.4.2	Environment Variables	52
38.4.3	Assumptions	53
38.4.4	Access Routine Semantics	53
38.4.5	Local Functions	53
39	MIS of Label Server	55
39.1	Module	55
39.2	Uses	55
39.3	Syntax	55
39.3.1	Exported Constants	55
39.3.2	Exported Access Programs	55
39.4	Semantics	55
39.4.1	State Variables	55
39.4.2	Environment Variables	55
39.4.3	Assumptions	55
39.4.4	Access Routine Semantics	55
39.4.5	Local Functions	56
40	MIS of Label Database Connector	57
40.1	Module	57
40.2	Uses	57
40.3	Syntax	57
40.3.1	Exported Constants	57
40.3.2	Exported Access Programs	57
40.4	Semantics	57
40.4.1	State Variables	57
40.4.2	Environment Variables	57
40.4.3	Assumptions	57
40.4.4	Access Routine Semantics	57
40.4.5	Local Functions	58
41	MIS of Label Database	59
41.1	Module	59
41.2	Uses	59
41.3	Syntax	59
41.3.1	Exported Constants	59
41.3.2	Exported Access Programs	59
41.4	Semantics	59
41.4.1	State Variables	59

41.4.2	Environment Variables	59
41.4.3	Assumptions	59
41.4.4	Access Routine Semantics	59
41.4.5	Local Functions	60
42	MIS of ImageObject Database Connector	61
42.1	Module	61
42.2	Uses	61
42.3	Syntax	61
42.3.1	Exported Constants	61
42.3.2	Exported Access Programs	61
42.4	Semantics	61
42.4.1	State Variables	61
42.4.2	Environment Variables	61
42.4.3	Assumptions	61
42.4.4	Access Routine Semantics	61
42.4.5	Local Functions	62
43	MIS of ImageObject Database	63
43.1	Module	63
43.2	Uses	63
43.3	Syntax	63
43.3.1	Exported Constants	63
43.3.2	Exported Access Programs	63
43.4	Semantics	63
43.4.1	State Variables	63
43.4.2	Environment Variables	63
43.4.3	Assumptions	63
43.4.4	Access Routine Semantics	63
43.4.5	Local Functions	64
44	MIS of Labeller Database Connector	65
44.1	Module	65
44.2	Uses	65
44.3	Syntax	65
44.3.1	Exported Constants	65
44.3.2	Exported Access Programs	65
44.4	Semantics	65
44.4.1	State Variables	65
44.4.2	Environment Variables	65
44.4.3	Assumptions	65
44.4.4	Access Routine Semantics	65
44.4.5	Local Functions	66

45 MIS of Labeller Database	67
45.1 Module	67
45.2 Uses	67
45.3 Syntax	67
45.3.1 Exported Constants	67
45.3.2 Exported Access Programs	67
45.4 Semantics	67
45.4.1 State Variables	67
45.4.2 Environment Variables	67
45.4.3 Assumptions	67
45.4.4 Access Routine Semantics	67
45.4.5 Local Functions	68
46 MIS of Object Extraction Manager	69
46.1 Module	69
46.2 Uses	69
46.3 Syntax	69
46.3.1 Exported Constants	69
46.3.2 Exported Access Programs	69
46.4 Semantics	69
46.4.1 State Variables	69
46.4.2 Environment Variables	69
46.4.3 Assumptions	69
46.4.4 Access Routine Semantics	69
46.4.5 Local Functions	70
47 MIS of Label Confidence Service	71
47.1 Module	71
47.2 Uses	71
47.3 Syntax	71
47.3.1 Exported Constants	71
47.3.2 Exported Access Programs	71
47.4 Semantics	71
47.4.1 State Variables	71
47.4.2 Environment Variables	71
47.4.3 Assumptions	71
47.4.4 Access Routine Semantics	71
47.4.5 Local Functions	72
48 MIS of Object Extraction Service	73
48.1 Module	73
48.2 Uses	73
48.3 Syntax	73

48.3.1	Exported Constants	73
48.3.2	Exported Access Programs	73
48.4	Semantics	73
48.4.1	State Variables	73
48.4.2	Environment Variables	73
48.4.3	Assumptions	73
48.4.4	Access Routine Semantics	73
48.4.5	Local Functions	74
49	MIS of Image Prior Analyzer	75
49.1	Module	75
49.2	Uses	75
49.3	Syntax	75
49.3.1	Exported Constants	75
49.3.2	Exported Access Programs	75
49.4	Semantics	75
49.4.1	State Variables	75
49.4.2	Environment Variables	75
49.4.3	Assumptions	75
49.4.4	Access Routine Semantics	75
49.4.5	Local Functions	75
50	MIS of Labeller Expertise Calculator	76
50.1	Module	76
50.2	Uses	76
50.3	Syntax	76
50.3.1	Exported Constants	76
50.3.2	Exported Access Programs	76
50.4	Semantics	76
50.4.1	State Variables	76
50.4.2	Environment Variables	76
50.4.3	Assumptions	76
50.4.4	Access Routine Semantics	76
50.4.5	Local Functions	77
51	MIS of Image Service Manager	78
51.1	Module	78
51.2	Uses	78
51.3	Syntax	78
51.3.1	Exported Constants	78
51.3.2	Exported Access Programs	78
51.4	Semantics	78
51.4.1	State Variables	78

51.4.2	Environment Variables	78
51.4.3	Assumptions	78
51.4.4	Access Routine Semantics	78
51.4.5	Local Functions	79
52	MIS of Image Mask Service	80
52.1	Module	80
52.2	Uses	80
52.3	Syntax	80
52.3.1	Exported Constants	80
52.3.2	Exported Access Programs	80
52.4	Semantics	80
52.4.1	State Variables	80
52.4.2	Environment Variables	80
52.4.3	Assumptions	80
52.4.4	Access Routine Semantics	80
52.4.5	Local Functions	80
53	MIS of Image Selection Service	81
53.1	Module	81
53.2	Uses	81
53.3	Syntax	81
53.3.1	Exported Constants	81
53.3.2	Exported Access Programs	81
53.4	Semantics	81
53.4.1	State Variables	81
53.4.2	Environment Variables	81
53.4.3	Assumptions	81
53.4.4	Access Routine Semantics	81
53.4.5	Local Functions	82
54	MIS of ModelComparisonEvaluation	83
54.1	6.1 Module	83
54.2	6.2 Uses	83
54.3	6.3 Syntax	83
54.3.1	6.3.1 Exported Constants	83
54.3.2	6.3.2 Exported Access Programs	83
54.4	6.4 Semantics	83
54.4.1	6.4.1 State Variables	83
54.4.2	6.4.2 Environment Variables	83
54.4.3	6.4.3 Assumptions	83
54.4.4	6.4.4 Access Routine Semantics	84
54.4.5	6.4.5 Local Functions	84

55 MIS of CrossValidationEvaluation	85
55.1 6.1 Module	85
55.2 6.2 Uses	85
55.3 6.3 Syntax	85
55.3.1 6.3.1 Exported Constants	85
55.3.2 6.3.2 Exported Access Programs	85
55.4 6.4 Semantics	85
55.4.1 6.4.1 State Variables	85
55.4.2 6.4.2 Environment Variables	85
55.4.3 6.4.3 Assumptions	85
55.4.4 6.4.4 Access Routine Semantics	86
55.4.5 6.4.5 Local Functions	86
56 MIS of ModelTrainingService	87
56.1 6.1 Module	87
56.2 6.2 Uses	87
56.3 6.3 Syntax	87
56.3.1 6.3.1 Exported Constants	87
56.3.2 6.3.2 Exported Access Programs	87
56.4 6.4 Semantics	87
56.4.1 6.4.1 State Variables	87
56.4.2 6.4.2 Environment Variables	87
56.4.3 6.4.3 Assumptions	87
56.4.4 6.4.4 Access Routine Semantics	88
56.4.5 6.4.5 Local Functions	88
57 MIS of ModelEvaluationService	89
57.1 6.1 Module	89
57.2 6.2 Uses	89
57.3 6.3 Syntax	89
57.3.1 6.3.1 Exported Constants	89
57.3.2 6.3.2 Exported Access Programs	89
57.4 6.4 Semantics	89
57.4.1 6.4.1 State Variables	89
57.4.2 6.4.2 Environment Variables	89
57.4.3 6.4.3 Assumptions	89
57.4.4 6.4.4 Access Routine Semantics	90
57.4.5 6.4.5 Local Functions	90
58 MIS of ModelManager	91
58.1 6.1 Module	91
58.2 6.2 Uses	91
58.3 6.3 Syntax	91

58.3.1	6.3.1	Exported Constants	91
58.3.2	6.3.2	Exported Access Programs	91
58.4	6.4	Semantics	91
58.4.1	6.4.1	State Variables	91
58.4.2	6.4.2	Environment Variables	91
58.4.3	6.4.3	Assumptions	92
58.4.4	6.4.4	Access Routine Semantics	92
58.4.5	6.4.5	Local Functions	92
59		MIS of ModelCreation (Abstract)	93
59.1	6.1	Module	93
59.2	6.2	Uses	93
59.3	6.3	Syntax	93
59.3.1	6.3.1	Exported Constants	93
59.3.2	6.3.2	Exported Access Programs	93
59.4	6.4	Semantics	93
59.4.1	6.4.1	State Variables	93
59.4.2	6.4.2	Environment Variables	93
59.4.3	6.4.3	Assumptions	93
59.4.4	6.4.4	Access Routine Semantics	93
59.4.5	6.4.5	Local Functions	94
60		MIS of MLModelDatabase	95
60.1	6.1	Module	95
60.2	6.2	Uses	95
60.3	6.3	Syntax	95
60.3.1	6.3.1	Exported Constants	95
60.3.2	6.3.2	Exported Access Programs	95
60.4	6.4	Semantics	95
60.4.1	6.4.1	State Variables	95
60.4.2	6.4.2	Environment Variables	95
60.4.3	6.4.3	Assumptions	95
60.4.4	6.4.4	Access Routine Semantics	96
60.4.5	6.4.5	Local Functions	96
61		MIS of OtherModelCreation	97
61.1	6.1	Module	97
61.2	6.2	Uses	97
61.3	6.3	Syntax	97
61.3.1	6.3.1	Exported Constants	97
61.3.2	6.3.2	Exported Access Programs	97
61.4	6.4	Semantics	97
61.4.1	6.4.1	State Variables	97

61.4.2	6.4.2 Environment Variables	97
61.4.3	6.4.3 Assumptions	97
61.4.4	6.4.4 Access Routine Semantics	97
61.4.5	6.4.5 Local Functions	98
62	MIS of CNNModelCreation	98
62.1	6.1 Module	98
62.2	6.2 Uses	98
62.3	6.3 Syntax	98
62.3.1	6.3.1 Exported Constants	98
62.3.2	6.3.2 Exported Access Programs	98
62.4	6.4 Semantics	98
62.4.1	6.4.1 State Variables	98
62.4.2	6.4.2 Environment Variables	98
62.4.3	6.4.3 Assumptions	98
62.4.4	6.4.4 Access Routine Semantics	99
62.4.5	6.4.5 Local Functions	99
63	Appendix	100

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 satellite 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 $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$.

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	\mathbb{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**

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

ModelTrainingRun

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**

AccountModification

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**
- **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:

- **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

ServiceRequest

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:

- **labeller_name**

Object

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

- **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**
- **labeller_id**

The following diagram display additional details on the relationship between datatypes

Project Creation and
Selection Subsystem

Project	
PK	<u>project_id</u> INT AUTO INCREMENT
	project_name VARCHAR(255) NOT NULL description TEXT creation_date DATETIME DEFAULT CURRENT_TIMESTAMP

User	
PK	<u>user_id</u> INT AUTO INCREMENT
	username VARCHAR(255) UNIQUE NOT NULL -- Add more user details as needed (e.g., name, email, phone) role VARCHAR(50) -- e.g., 'Labeler', 'Manager', 'Coordinator'

ProjectAssignment	
PK	<u>user_id</u> INT
PK	<u>project_id</u> INT
PK	<u>project_assignment_id</u> INT AUTO INCREMENT
	assignment_date DATETIME DEFAULT CURRENT_TIMESTAMP FOREIGN KEY (project_id) REFERENCES Project(project_id) FOREIGN KEY (user_id) REFERENCES User(user_id)

SatelliteImage	
PK	<u>project_id</u> INT, -- Link to the project the image belongs to
PK	<u>image_id</u> INT AUTO INCREMENT
	image_path VARCHAR(255), -- Or BLOB if storing directly acquisition_date DATE -- Add other relevant metadata FOREIGN KEY (project_id) REFERENCES Project(project_id)

LabelingTask	
PK	<u>image_id</u> INT
PK	<u>project_id</u> INT
PK	<u>labeling_task_id</u> INT AUTO INCREMENT
	assigned_to INT, -- User assigned to this task status VARCHAR(50) DEFAULT 'Pending', -- e.g., 'Pending', 'Completed' start_time DATETIME end_time DATETIME FOREIGN KEY (project_id) REFERENCES Project(p FOREIGN KEY (image_id) REFERENCES SatelliteIm FOREIGN KEY (assigned_to) REFERENCES User(u

Report	
PK	<u>project_id</u> INT
PK	<u>report_id</u> 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(u

ServiceRequest	
PK	<u>project_id</u> INT
PK	<u>request_id</u> 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(u

Computer Vision
Model Creation
Subsystem

MLModel	
PK	<u>model_id</u> 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', 'Segmentation' 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	<u>model_id</u> INT
PK	<u>training_run_id</u> 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	<u>model_id</u> INT
PK	<u>evaluation_run_id</u> 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 MLModel

ModelDeployment	
PK	<u>model_id</u> INT
PK	<u>deployment_id</u> INT AUTO INCREMENT
	deployment_date DATETIME DEFAULT CURRENT_ deployment_environment VARCHAR(255), -- e.g., 'Pro 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
Subsystem

Account	
PK	<u>account_id INT AUTO INCREMENT</u>
	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

AccountModification	
PK	<u>account_id INT</u>
PK	<u>modification_id INT AUTO INCREMENT</u>
	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 Account

LoginAttempt	
PK	<u>attempt_id INT AUTO INCREMENT</u>
	username VARCHAR(255)
	attempt_time DATETIME DEFAULT CURRENT_TIME
	successful BOOLEAN
	ip_address VARCHAR(45) -- For tracking location of

Label Collection and
Aggregation
Subsystem

Image	
PK	<u>image_id INT AUTO INCREMENT</u>
	image_data BLOB, -- Or VARCHAR for file paths if st
	image_path VARCHAR(255)
	upload_date DATETIME

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

Object	
PK	<u>image_id INT</u>
PK	<u>object_id INT AUTO INCREMENT</u>
	bounding_box_coordinates VARCHAR(255), -- Store
	object_type VARCHAR(255)
	FOREIGN KEY (image_id) REFERENCES Image(im

Label	
PK	<u>labeller_id INT</u>
PK	<u>object_id INT</u>
PK	<u>image_id INT</u>
PK	<u>label_id INT AUTO INCREMENT</u>
	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
Behaviour-Hiding Module	Account Creation Interface
	Account Database
	Account Update Interface
	Login Interface
	Access Token
	Labeler
	Client
	User
	Satellite Image Request Interface
	Satellite Image Request
	Project Creation Interface
	Project
	Service Request Failure Interface
	Image Upload Interface
	Report Interface
	Report
	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
	OtherModelCreation
	ModelManager
	MLModelDatabase

Table 2: Module Hierarchy

Level 1	Level 2
Software Decision Module	Account Creation Controller
	Account Database Connector
	Account Update Controller
	Authentication Controller
	Satellite Image Request Controller
	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 Account Creation Interface

6.1 Module

Account Creation Interface

6.2 Uses

Account Creation Controller [20](#)

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

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

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

win: 2D sequence of coloured pixels

6.4.3 Assumptions

None

6.4.4 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.

6.4.5 Local Functions

None

7 MIS of Account Database Connector

7.1 Module

Account Database Connector

7.2 Uses

Account Database [13](#)

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

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

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

databaseConnection: connection to relational database

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

insertUser(user):

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

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:

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

userExists(email):

- output: out :=

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

makeDBConnection(credentials):

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

7.4.5 Local Functions

None

8 MIS of Account Database

8.1 Module

Account Database

8.2 Uses

None

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

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

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

databaseConnection: connection to Application

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

insertUser(user):

- transition: Insert user into database.

retrieveUser(email):

- output:

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

updateUser(user):

- transition:

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

userExists(email):

- output: out :=

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

8.4.5 Local Functions

None

9 MIS of Account Update Interface

9.1 Module

Account Update Interface

9.2 Uses

Account Update Controller [21](#)

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

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

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

win: 2D sequence of coloured pixels

9.4.3 Assumptions

None

9.4.4 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.

9.4.5 Local Functions

None

10 MIS of Login Interface

10.1 Module

Login Interface

10.2 Uses

Authentication Controller [22](#)

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

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

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

win: 2D sequence of coloured pixels

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

renderPage():

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

submitForm(formData):

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

10.4.5 Local Functions

None

11 MIS of Access Token

11.1 Module

Access Token

11.2 Uses

None

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

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

11.4 Semantics

11.4.1 State Variables

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

11.4.2 Environment Variables

None

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

isExpired():

- output: $\text{out} := \text{currentTime} > \text{expirationTime}$

renew():

- transition: $\text{expirationTime} := \text{expirationTime} + 5 \text{ hours}$

11.4.5 Local Functions

None

12 MIS of Account Creation Interface

12.1 Module

Account Creation Interface

12.2 Uses

Account Creation Controller [20](#)

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

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

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

win: 2D sequence of coloured pixels

12.4.3 Assumptions

None

12.4.4 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.

12.4.5 Local Functions

None

13 MIS of Account Database

13.1 Module

Account Database

13.2 Uses

Relational Database

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 relational database

13.4.3 Assumptions

None

13.4.4 Access Routine Semantics

insertUser(user):

- transition: Insert user into database through databaseConection.

retrieveUser(email):

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

updateUser(user):

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

userExists(email):

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

13.4.5 Local Functions

None

14 MIS of Account Update Interface

14.1 Module

Account Update Interface

14.2 Uses

Account Update Controller [21](#)

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	User	-	-
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 Controller [22](#)

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

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

15.4 Semantics

15.4.1 State Variables

None

15.4.2 Environment Variables

win: 2D sequence of coloured pixels

15.4.3 Assumptions

None

15.4.4 Access Routine Semantics

renderPage():

- 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 Labeler

17.1 Module

Labeler

17.2 Uses

Extends User [19](#)

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 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	-	-

17.4 Semantics

17.4.1 State Variables

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

17.4.2 Environment Variables

None

17.4.3 Assumptions

None

17.4.4 Access Routine Semantics

getFirstName():

- output: out := firstName

getLastName():

- output: out := lastName

getSkills():

- output: out := skills

getAvailability():

- output: out := availability

setFirstName(newfn):

- transition: firstName := newfn

setLastName(newln):

- transition: lastName := newln

setSkills(newSkills):

- transition: skills := newSkills

setAvailability(newAvail):

- transition: availability := newAvail

17.4.5 Local Functions

None

18 MIS of Client

18.1 Module

Client

18.2 Uses

Extends User [19](#)

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

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

18.4 Semantics

18.4.1 State Variables

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

18.4.2 Environment Variables

None

18.4.3 Assumptions

None

18.4.4 Access Routine Semantics

getCompanyName():

- output: out := companyName

getIndustry():

- output: out := industry

getTypicalProject():

- output: out := typicalProject

setCompanyName(newcn):

- transition: companyName := newcn

setIndustry(newIndustry):

- transition: industry := newIndustry

setTypicalProject(newtp):

- transition: typicalProject := newtp

18.4.5 Local Functions

None

19 MIS of User

19.1 Module

User

19.2 Uses

None

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

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

19.4 Semantics

19.4.1 State Variables

- email: string

- password: string
- profilePic: image

19.4.2 Environment Variables

None

19.4.3 Assumptions

None

19.4.4 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(newProfilePic):

- transition: profilePic := newProfilePic

19.4.5 Local Functions

None

20 MIS of Account Creation Controller

20.1 Module

Account Creation Controller

20.2 Uses

Account Creation Interface [12](#)

Account Database [13](#)

User [19](#)

Labeler [17](#)

Client [18](#)

20.3 Syntax

20.3.1 Exported Constants

None

20.3.2 Exported Access Programs

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

20.4 Semantics

20.4.1 State Variables

None

20.4.2 Environment Variables

None

20.4.3 Assumptions

Assumes AccountDatabase is operational when calling uploadUser.

20.4.4 Access Routine Semantics

validateForm(formData, userType):

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

isValidPassword(formData.password) ∧

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

Where:

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

createUser(formData, userType):

- output: out :=

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

uploadUser(newUser):

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

20.4.5 Local Functions

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

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:

$$V = (\forall \text{ email} \in E \mid \text{email matches the pattern } [\text{a-zA-Z0-9+.-}]+\text{@[a-zA-Z0-9.-]}+[\text{a-zA-Z}])$$

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

21 MIS of Account Update Controller

21.1 Module

Account Update Controller

21.2 Uses

Account Update Interface [14](#)

Account Database [13](#)

User [19](#)

21.3 Syntax

21.3.1 Exported Constants

None

21.3.2 Exported Access Programs

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

21.4 Semantics

21.4.1 State Variables

- user: User

21.4.2 Environment Variables

None

21.4.3 Assumptions

Assumes AccountDatabase is operational when calling requestUpdate.

21.4.4 Access Routine Semantics

validateForm(formData):

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

getUser(email):

- transition: `user := AccountDatabase.retrieveUser(email)`

requestUpdate(updatedUser):

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

21.4.5 Local Functions

None

22 MIS of Authentication Controller

22.1 Module

Authentication Controller

22.2 Uses

Login Interface [36](#)

Account Database [13](#)

Access Token [16](#)

22.3 Syntax

22.3.1 Exported Constants

None

22.3.2 Exported Access Programs

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

22.4 Semantics

22.4.1 State Variables

- token: AccessToken

22.4.2 Environment Variables

None

22.4.3 Assumptions

Assumes AccountDatabase is operational when calling validCredentials.

22.4.4 Access Routine Semantics

validCredentials(email, password):

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

generateAccessToken(email):

- transition: $\text{token} := \text{AccessToken(email)}$

22.4.5 Local Functions

None

23 MIS of Satellite Image Request Interface

23.1 Module

Satellite Image Request Interface

23.2 Uses

Satellite Image Request Controller [24](#)

23.3 Syntax

23.3.1 Exported Constants

None

23.3.2 Exported Access Programs

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

23.4 Semantics

23.4.1 State Variables

None

23.4.2 Environment Variables

win: 2D sequence of coloured pixels

23.4.3 Assumptions

None

23.4.4 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.

23.4.5 Local Functions

None

24 MIS of Satellite Image Request Controller

24.1 Module

Satellite Image Request Controller

24.2 Uses

Satellite Image Request Interface [23](#)

Satellite Image Request [25](#)

24.3 Syntax

24.3.1 Exported Constants

None

24.3.2 Exported Access Programs

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

24.4 Semantics

24.4.1 State Variables

None

24.4.2 Environment Variables

None

24.4.3 Assumptions

None

24.4.4 Access Routine Semantics

validateForm(formData):

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

requestImages(imgRequest):

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

24.4.5 Local Functions

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

25 MIS of Satellite Image Request

25.1 Module

Satellite Image Request

25.2 Uses

None

25.3 Syntax

25.3.1 Exported Constants

None

25.3.2 Exported Access Programs

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

25.4 Semantics

25.4.1 State Variables

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

25.4.2 Environment Variables

None

25.4.3 Assumptions

None

25.4.4 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

25.4.5 Local Functions

None

26 MIS of Project Creation Interface

26.1 Module

Project Creation Interface

26.2 Uses

Project Creation Controller [27](#)

26.3 Syntax

26.3.1 Exported Constants

None

26.3.2 Exported Access Programs

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

26.4 Semantics

26.4.1 State Variables

None

26.4.2 Environment Variables

win: 2D sequence of coloured pixels

26.4.3 Assumptions

None

26.4.4 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.

26.4.5 Local Functions

None

27 MIS of Project Creation Controller

27.1 Module

Project Creation Controller

27.2 Uses

Project Creation Interface [26](#)

Project [28](#)

27.3 Syntax

27.3.1 Exported Constants

None

27.3.2 Exported Access Programs

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

27.4 Semantics

27.4.1 State Variables

None

27.4.2 Environment Variables

None

27.4.3 Assumptions

None

27.4.4 Access Routine Semantics

validateForm(formData):

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

createNewProject(formData):

- output: $\text{out} := \text{Project}(\text{formData.name}, \text{formData.description}, \text{formData.labelClasses.split()}, \text{Date}(\text{formData.startDate}), \text{Date}(\text{formData.endDate}))$

27.4.5 Local Functions

- calculateEstimatedCost(project): $\text{out} :=$ Use information given to calculate the estimated cost of a project.

28 MIS of Project

28.1 Module

Project

28.2 Uses

None

28.3 Syntax

28.3.1 Exported Constants

None

28.3.2 Exported Access Programs

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

28.4 Semantics

28.4.1 State Variables

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

28.4.2 Environment Variables

None

28.4.3 Assumptions

None

28.4.4 Access Routine Semantics

getProjectID():

- output: out := projectID

getName():

- output: out := name

getDescription():

- output: out := description

getLabelClasses():

- output: out := labelClasses

getTimePeriod():

- output: out := (startDate, endDate)

setName(newName):

- transition: name := newName

setDescription(newDesc):

- transition: description := newDesc

setLabelClasses(newlc):

- transition: labelClasses := newlc

setTimePeriod(start, end):

- transition: startDate, endDate := start, end

28.4.5 Local Functions

None

29 MIS of Service Request Failure Interface

29.1 Module

Service Request Failure Interface

29.2 Uses

29.3 Syntax

29.3.1 Exported Constants

None

29.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayErrorInfo	-	-	-

29.4 Semantics

29.4.1 State Variables

None

29.4.2 Environment Variables

win: 2D sequence of coloured pixels

29.4.3 Assumptions

None

29.4.4 Access Routine Semantics

displayErrorInfo():

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

29.4.5 Local Functions

None

30 MIS of Image Upload Interface

30.1 Module

Image Upload Interface

30.2 Uses

30.3 Syntax

30.3.1 Exported Constants

None

30.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayUploadImages		-	-

30.4 Semantics

30.4.1 State Variables

None

30.4.2 Environment Variables

win: 2D sequence of coloured pixels

30.4.3 Assumptions

None

30.4.4 Access Routine Semantics

displayUploadImages():

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

30.4.5 Local Functions

- validateImage(image): out :=

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

31 MIS of Report Interface

31.1 Module

Report Interface

31.2 Uses

Report Controller [32](#)

31.3 Syntax

31.3.1 Exported Constants

None

31.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayStats	-	-	-

31.4 Semantics

31.4.1 State Variables

None

31.4.2 Environment Variables

win: 2D sequence of coloured pixels

31.4.3 Assumptions

None

31.4.4 Access Routine Semantics

displayStats():

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

31.4.5 Local Functions

None

32 MIS of Report Controller

32.1 Module

Report Controller

32.2 Uses

Report Interface [31](#)

Report [33](#)

32.3 Syntax

32.3.1 Exported Constants

None

32.3.2 Exported Access Programs

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

32.4 Semantics

32.4.1 State Variables

- report: Report

32.4.2 Environment Variables

fm: External systems file manager

32.4.3 Assumptions

None

32.4.4 Access Routine Semantics

getProjectStats(projectID):

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

exportLabeledImages():

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

32.4.5 Local Functions

None

33 MIS of Report

33.1 Module

Report

33.2 Uses

None

33.3 Syntax

33.3.1 Exported Constants

None

33.3.2 Exported Access Programs

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

33.4 Semantics

33.4.1 State Variables

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

33.4.2 Environment Variables

None

33.4.3 Assumptions

None

33.4.4 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`

33.4.5 Local Functions

None

34 MIS of Project Selection Interface

34.1 Module

Project Selection Interface

34.2 Uses

Project Selection Controller [35](#)

34.3 Syntax

34.3.1 Exported Constants

None

34.3.2 Exported Access Programs

Name	In	Out	Exceptions
<code>displayActiveProjects</code>	-	-	-

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

displayActiveProjects():

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

34.4.5 Local Functions

None

35 MIS of Project Selection Controller

35.1 Module

Project Selection Controller

35.2 Uses

Project Selection Interface [34](#)

Project [28](#)

35.3 Syntax

35.3.1 Exported Constants

None

35.3.2 Exported Access Programs

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

35.4 Semantics

35.4.1 State Variables

- activeProjects: list[Project]

35.4.2 Environment Variables

win: 2D sequence of coloured pixels

35.4.3 Assumptions

None

35.4.4 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

35.4.5 Local Functions

None

36 MIS of Labeling Interface

36.1 Module

Labeling Interface

36.2 Uses

Labeling Controller [37](#)

Image [38](#)

36.3 Syntax

36.3.1 Exported Constants

None

36.3.2 Exported Access Programs

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

36.4 Semantics

36.4.1 State Variables

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

36.4.2 Environment Variables

win: 2D sequence of coloured pixels

36.4.3 Assumptions

None

36.4.4 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.

36.4.5 Local Functions

None

37 MIS of Labeling Controller

37.1 Module

Labeling Controller

37.2 Uses

Labeling Interface [36](#)

Label ??

37.3 Syntax

37.3.1 Exported Constants

None

37.3.2 Exported Access Programs

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

37.4 Semantics

37.4.1 State Variables

- labels: list[Label]

37.4.2 Environment Variables

None

37.4.3 Assumptions

None

37.4.4 Access Routine Semantics

getProjectImages(projectID):

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

addLabel(lbl):

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

removeLabel(lblID):

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

submitLabels(lbls):

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

37.4.5 Local Functions

None

38 MIS of Image

38.1 Module

Image

38.2 Uses

None

38.3 Syntax

38.3.1 Exported Constants

None

38.3.2 Exported Access Programs

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

38.4 Semantics

38.4.1 State Variables

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

38.4.2 Environment Variables

None

38.4.3 Assumptions

None

38.4.4 Access Routine Semantics

getProjectID():

- output: out := projectID

getImageID():

- output: out := imageID

getDimensions():

- output: out := (width, height)

getImageData():

- output: out := imageData

38.4.5 Local Functions

None

39 MIS of Label Server

39.1 Module

Label Server

39.2 Uses

Labeling Controller [37](#)

Label ??

Label Database Connector [40](#)

39.3 Syntax

39.3.1 Exported Constants

None

39.3.2 Exported Access Programs

Name	In	Out	Exceptions
acceptLabel	Label	-	ValueError, Conne- ctionError

39.4 Semantics

39.4.1 State Variables

None

39.4.2 Environment Variables

LabelDatabaseConnector

39.4.3 Assumptions

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

39.4.4 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`

39.4.5 Local Functions

`JSONLabeltoLabel`: converts a JSON object into a Label object.

40 MIS of Label Database Connector

40.1 Module

Label Database Connector

40.2 Uses

Label Database [41](#)

Label ??

40.3 Syntax

40.3.1 Exported Constants

None

40.3.2 Exported Access Programs

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

40.4 Semantics

40.4.1 State Variables

None

40.4.2 Environment Variables

None

40.4.3 Assumptions

40.4.4 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 successful, 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`

40.4.5 Local Functions

None

41 MIS of Label Database

41.1 Module

Label Database

41.2 Uses

None

41.3 Syntax

41.3.1 Exported Constants

None

41.3.2 Exported Access Programs

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

41.4 Semantics

41.4.1 State Variables

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

41.4.2 Environment Variables

None

41.4.3 Assumptions

41.4.4 Access Routine Semantics

pushLabel(Label l):

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

makeDBConnection(credentials):

- transition: if credentials are valid, $\text{users} := \text{users} \cup \text{credentials.user}$
- 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 $\neg(\text{requestor} \in \text{users})$

41.4.5 Local Functions

None

42 MIS of ImageObject Database Connector

42.1 Module

ImageObject Database Connector

42.2 Uses

ImageObject Database [43](#)

ImageObject ??

42.3 Syntax

42.3.1 Exported Constants

None

42.3.2 Exported Access Programs

Name	In	Out	Exceptions
push Im- age Object	ImageObject	-	ValueError, Conne- ctionError
makeDB Conne- ction	ImageObject	-	ConnectionError
get Image Objects	String	list[ImageObject]	ValueError, Conne- ctionError

42.4 Semantics

42.4.1 State Variables

None

42.4.2 Environment Variables

None

42.4.3 Assumptions

42.4.4 Access Routine Semantics

pushLabel(ImageObject l):

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

makeDBConnection():

- transition: If successful, connection occurs
- exception: Throw ConnectionError if connection is not accepted by ImageObjectDatabase

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

42.4.5 Local Functions

None

43 MIS of ImageObject Database

43.1 Module

ImageObject Database

43.2 Uses

None

43.3 Syntax

43.3.1 Exported Constants

None

43.3.2 Exported Access Programs

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

43.4 Semantics

43.4.1 State Variables

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

43.4.2 Environment Variables

None

43.4.3 Assumptions

43.4.4 Access Routine Semantics

pushLabel(ImageObject l):

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

makeDBConnection(credentials):

- transition: if credentials are valid, $\text{users} := \text{users} \cup \text{credentials.user}$
- 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 $\neg(\text{requestor} \in \text{users})$

43.4.5 Local Functions

None

=====

44 MIS of Labeller Database Connector

44.1 Module

Labeller Database Connector

44.2 Uses

Labeller Database ??

ImageObject ??

44.3 Syntax

44.3.1 Exported Constants

None

44.3.2 Exported Access Programs

Name	In	Out	Exceptions
push la- beller	labeller	-	ValueError, Conne- ctionError
makeDB Conne- ction	credentials	-	ConnectionError
get labeller	String	list[labeller]	ValueError, Conne- ctionError

44.4 Semantics

44.4.1 State Variables

None

44.4.2 Environment Variables

None

44.4.3 Assumptions

44.4.4 Access Routine Semantics

pushLabeller(Labeller o):

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

`makeDBConnection()`:

- transition: If successful, 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`

44.4.5 Local Functions

None

45 MIS of Labeller Database

45.1 Module

Labeller Database

45.2 Uses

None

45.3 Syntax

45.3.1 Exported Constants

None

45.3.2 Exported Access Programs

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

45.4 Semantics

45.4.1 State Variables

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

45.4.2 Environment Variables

None

45.4.3 Assumptions

45.4.4 Access Routine Semantics

pushLabeller(Labeller o):

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

makeDBConnection(credentials):

- transition: if credentials are valid, $\text{users} := \text{users} \cup \text{credentials.user}$
- 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 $\neg(\text{requestor} \in \text{users})$

45.4.5 Local Functions

None

46 MIS of Object Extraction Manager

46.1 Module

Object Extraction Manager

46.2 Uses

ImageObject Database Connector [42](#)

Label Database Connector [40](#)

Labeller Database Connector [44](#)

Image Prior Analyzer [49](#)

Label Confidence Service [47](#)

Object Extraction Service [48](#)

Labeller Expertise Calculator [50](#)

46.3 Syntax

46.3.1 Exported Constants

None

46.3.2 Exported Access Programs

Name	In	Out	Exceptions
getObjects	projectID	-	ValueError

46.4 Semantics

46.4.1 State Variables

None

46.4.2 Environment Variables

None

46.4.3 Assumptions

46.4.4 Access Routine Semantics

getObjects(ProjectID p):

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

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

46.4.5 Local Functions

generate query:

47 MIS of Label Confidence Service

47.1 Module

Label Confidence Service

47.2 Uses

None

47.3 Syntax

47.3.1 Exported Constants

None

47.3.2 Exported Access Programs

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

47.4 Semantics

47.4.1 State Variables

None

47.4.2 Environment Variables

None

47.4.3 Assumptions

47.4.4 Access Routine Semantics

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 $(\exists \text{label} \in \text{labels} \mid \neg(\text{label} \in L))$
Let X be the set of valid Labellers. Throw ValueError if $(\exists \text{labeller} \in \text{labellers} \mid \neg(\text{labeller} \in X))$
Let I be the set of valid ImageObjects. Throw ValueError if $(\exists \text{imageobject} \in \text{imageobjects} \mid$

$\neg(\text{imageobject} \in I)$

47.4.5 Local Functions

48 MIS of Object Extraction Service

48.1 Module

Object Extraction Service

48.2 Uses

None

48.3 Syntax

48.3.1 Exported Constants

None

48.3.2 Exported Access Programs

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

48.4 Semantics

48.4.1 State Variables

None

48.4.2 Environment Variables

None

48.4.3 Assumptions

48.4.4 Access Routine Semantics

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

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

$\neg(\text{labeller} \in X)$

Let I be the set of valid ImageObjects. Throw ValueError if $(\exists \text{imageobject} \in \text{imageobjects} | :$

$\neg(\text{imageobject} \in I)$

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

48.4.5 Local Functions

49 MIS of Image Prior Analyzer

49.1 Module

Image Prior Analyzer

49.2 Uses

None

49.3 Syntax

49.3.1 Exported Constants

None

49.3.2 Exported Access Programs

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

49.4 Semantics

49.4.1 State Variables

None

49.4.2 Environment Variables

None

49.4.3 Assumptions

49.4.4 Access Routine Semantics

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 $(\exists \text{image} \in \text{images} : \neg(\text{image} \in I))$

49.4.5 Local Functions

50 MIS of Labeller Expertise Calculator

50.1 Module

Labeller Expertise Calculator

50.2 Uses

None

50.3 Syntax

50.3.1 Exported Constants

None

50.3.2 Exported Access Programs

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

50.4 Semantics

50.4.1 State Variables

None

50.4.2 Environment Variables

None

50.4.3 Assumptions

50.4.4 Access Routine Semantics

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 $(\exists \text{label} \in \text{labels} \mid \neg(\text{label} \in L))$
Let X be the set of valid Labellers. Throw ValueError if $(\exists \text{labeller} \in \text{labellers} \mid \neg(\text{labeller} \in X))$

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

50.4.5 Local Functions

51 MIS of Image Service Manager

51.1 Module

Image Service Manager

51.2 Uses

ImageObject Database Connector [42](#)

Labeller Database Connector [44](#)

Image Mask Service [52](#)

Image Selection Engine ??

51.3 Syntax

51.3.1 Exported Constants

None

51.3.2 Exported Access Programs

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

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

- output: return the next n images based on which are more 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 $\neg(n \in \mathbb{N})$

51.4.5 Local Functions

52 MIS of Image Mask Service

52.1 Module

Image Mask Service

52.2 Uses

None

52.3 Syntax

52.3.1 Exported Constants

None

52.3.2 Exported Access Programs

Name	In	Out	Exceptions
getImageMask	Image	Image	ValueError

52.4 Semantics

52.4.1 State Variables

None

52.4.2 Environment Variables

None

52.4.3 Assumptions

52.4.4 Access Routine Semantics

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 $\neg(i \in I)$

52.4.5 Local Functions

53 MIS of Image Selection Service

53.1 Module

Image Selection Service

53.2 Uses

53.3 Syntax

53.3.1 Exported Constants

None

53.3.2 Exported Access Programs

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

53.4 Semantics

53.4.1 State Variables

None

53.4.2 Environment Variables

None

53.4.3 Assumptions

53.4.4 Access Routine Semantics

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

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

53.4.5 Local Functions

None

54 MIS of ModelComparisonEvaluation

54.1 6.1 Module

Name: ModelComparisonEvaluation

54.2 6.2 Uses

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

54.3 6.3 Syntax

54.3.1 6.3.1 Exported Constants

None

54.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
<code>evaluateModel</code>	<code>String modelId,</code> <code>TestDataset testData</code>	<code>EvaluationResult</code>	<code>ModelNotFoundError,</code> <code>ValueError</code>

54.4 6.4 Semantics

54.4.1 6.4.1 State Variables

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

54.4.2 6.4.2 Environment Variables

None

54.4.3 6.4.3 Assumptions

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

54.4.4 6.4.4 Access Routine Semantics

`evaluateModel(modelId, testData):`

- **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.

54.4.5 6.4.5 Local Functions

None

55 MIS of CrossValidationEvaluation

55.1 6.1 Module

Name: CrossValidationEvaluation

55.2 6.2 Uses

- TestDataset
- EvaluationResult

55.3 6.3 Syntax

55.3.1 6.3.1 Exported Constants

None

55.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
evaluateModel	String modelId, TestDataset testData	EvaluationResult	ModelNotFoundError, ValueError

55.4 6.4 Semantics

55.4.1 6.4.1 State Variables

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

55.4.2 6.4.2 Environment Variables

None

55.4.3 6.4.3 Assumptions

- $kFolds \geq 2$.
- testData is large enough for multiple folds.

55.4.4 6.4.4 Access Routine Semantics

`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.

55.4.5 6.4.5 Local Functions

None

56 MIS of ModelTrainingService

56.1 6.1 Module

Name: ModelTrainingService

56.2 6.2 Uses

- TrainingParams
- TrainingData
- ModelConfig
- TrainingResult

56.3 6.3 Syntax

56.3.1 6.3.1 Exported Constants

None

56.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
trainModel	TrainingData data, ModelConfig modelConfig	TrainingResult	ValueError, ResourceU- navailableError
stopTraining	String modelId	void	ModelNotFoundError

56.4 6.4 Semantics

56.4.1 6.4.1 State Variables

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

56.4.2 6.4.2 Environment Variables

None

56.4.3 6.4.3 Assumptions

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

56.4.4 6.4.4 Access Routine Semantics

`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.
 - `ResourceUnavailableError` 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.

56.4.5 6.4.5 Local Functions

`None`

57 MIS of ModelEvaluationService

57.1 6.1 Module

Name: ModelEvaluationService

57.2 6.2 Uses

- TestDataset
- EvaluationResult

57.3 6.3 Syntax

57.3.1 6.3.1 Exported Constants

None

57.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
evaluateModel	String modelId, TestDataset testData	EvaluationResult	ModelNotFoundError, ValueError
fetchEvaluationMetrics	String modelId	Map<String,Float>	ModelNotFoundError

57.4 6.4 Semantics

57.4.1 6.4.1 State Variables

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

57.4.2 6.4.2 Environment Variables

None

57.4.3 6.4.3 Assumptions

- The modelId references a trained model.

57.4.4 6.4.4 Access Routine Semantics

`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.

57.4.5 6.4.5 Local Functions

None

58 MIS of ModelManager

58.1 6.1 Module

Name: ModelManager

58.2 6.2 Uses

- ModelParameters
- MLModel

58.3 6.3 Syntax

58.3.1 6.3.1 Exported Constants

None

58.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
createModel	ModelParameters params	void	ValueError
updateModelStatus	String modelId, String status	void	ModelNotFoundError
fetchModel	String modelId	MLModel	ModelNotFoundError
deleteModel	String modelId	void	ModelNotFoundError

58.4 6.4 Semantics

58.4.1 6.4.1 State Variables

- modelID: String
- status: String (“Training”, “Evaluating”, “Completed”, etc.)
- createdAt: Date
- updatedAt: Date

58.4.2 6.4.2 Environment Variables

None

58.4.3 6.4.3 Assumptions

- A unique `modelID` is generated upon creation.

58.4.4 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.

58.4.5 6.4.5 Local Functions

None

59 MIS of ModelCreation (Abstract)

59.1 6.1 Module

Name: ModelCreation (Abstract Base Class)

59.2 6.2 Uses

- ModelParameters
- MLModel

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
createModel	-	MLModel	NotImplementedError

59.4 6.4 Semantics

59.4.1 6.4.1 State Variables

- modelType: String
- creationParams: ModelParameters

59.4.2 6.4.2 Environment Variables

None

59.4.3 6.4.3 Assumptions

- Concrete subclasses must override the `createModel` method.

59.4.4 6.4.4 Access Routine Semantics

`createModel()`:

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

59.4.5 6.4.5 Local Functions

None

60 MIS of MLModelDatabase

60.1 6.1 Module

Name: MLModelDatabase

60.2 6.2 Uses

- MLModel

60.3 6.3 Syntax

60.3.1 6.3.1 Exported Constants

None

60.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
saveModel	MLModel model	void	DatabaseError
fetchModel	String modelId	MLModel	ModelNotFoundError, DatabaseError
deleteModel	String modelId	void	ModelNotFoundError, DatabaseError
updateModel	String modelId, Map<String, Any> updates	void	ModelNotFoundError, DatabaseError

60.4 6.4 Semantics

60.4.1 6.4.1 State Variables

- dbConnection: Connection (Active DB connection)

60.4.2 6.4.2 Environment Variables

- External database system (accessed via dbConnection)

60.4.3 6.4.3 Assumptions

- dbConnection is valid and open.

60.4.4 6.4.4 Access Routine Semantics

`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.

60.4.5 6.4.5 Local Functions

None

61 MIS of OtherModelCreation

61.1 6.1 Module

Name: OtherModelCreation

61.2 6.2 Uses

- MLModel
- ModelCreation (abstract base class)

61.3 6.3 Syntax

61.3.1 6.3.1 Exported Constants

None

61.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
createModel	-	MLModel	ValueError

61.4 6.4 Semantics

61.4.1 6.4.1 State Variables

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

61.4.2 6.4.2 Environment Variables

None

61.4.3 6.4.3 Assumptions

- `hyperparameters` are valid for `modelType`.

61.4.4 6.4.4 Access Routine Semantics

`createModel()`:

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

61.4.5 6.4.5 Local Functions

None

62 MIS of CNNModelCreation

62.1 6.1 Module

Name: CNNModelCreation

62.2 6.2 Uses

- ModelCreation (abstract)
- MLModel

62.3 6.3 Syntax

62.3.1 6.3.1 Exported Constants

None

62.3.2 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
createModel	-	MLModel	ValueError

62.4 6.4 Semantics

62.4.1 6.4.1 State Variables

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

62.4.2 6.4.2 Environment Variables

None

62.4.3 6.4.3 Assumptions

- The layers and activatedFunctions lists are valid and aligned.

62.4.4 6.4.4 Access Routine Semantics

`createModel()`:

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

62.4.5 6.4.5 Local Functions

None

References

- Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.
- Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL <http://citeseer.ist.psu.edu/428727.html>.

63 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

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

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

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

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

1. What went well while writing this deliverable? Everyone did a great job contributing their ideas and expertise to design each part of our application. We decided to use diagrams to express our designs before jumping into the documentation. This worked really well as it allowed everyone to have a better understanding of how our system would interact. When we had to specify our modules, a lot of the hard work was already complete due to have the diagrams.
2. What pain points did you experience during this deliverable, and how did you resolve them? A major pain point we faced was that a team member could no longer meet in person due to extraneous circumstances. This hindered our ability to effectively communicate as a team due to factors like time difference. To solve this, we rescheduled our meetings to a reasonable time for all members, and moved all meetings and communications online for the time being.
3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from? From talking to our supervisor, we determined that we would need our system to be able to pre-process images in an effective way so we took that into considering when designing the project creation subsystem. Also, our decision to have modules do standardized formatting stemmed from our usage of 3rd-party applications such as our image distributor. Due to the possibility of change, we knew that formatting outside information to a way our application could process it would be the best way to go about it. In general, for our other decisions we used the software principles we have learned through out our education including modularity, seperation of concern, and architecture that supports scalability.
4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why? When creating this design

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

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions) One major focus would be to add in the financial capabilities like accepting payments for projects and distributing funds to users. Another thing would be to expand the platform to accept lots of different types of media, such as videos, instead of just images.
6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? We considered an approach where we would store all data as files on a server somewhere, however we decided that using a database would better fit our project due to the relation between the data and the usefulness of SQL statements. The downside of this approach is it requires more time to set up, but we believe the payoff is worth it. We also considered having one large system rather than many sub-systems. This would eliminate a lot 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 don't have time to get to a sub-system, our application can still function. (LO_Explores)