

# Module Interface Specification for Software Engineering

Team #11, OKKM Insights

Mathew Petronilho

Oleg Glotov

Kyle McMaster

Kartik Chaudhari

March 27, 2025

# 1 Revision History

Date		Version	Notes
Date	January	1.0	
17th			

## 2 Symbols, Abbreviations and Acronyms

See SRS Documentation [here](#)

# Contents

<b>1</b>	<b>Revision History</b>	<b>i</b>
<b>2</b>	<b>Symbols, Abbreviations and Acronyms</b>	<b>ii</b>
<b>3</b>	<b>Introduction</b>	<b>1</b>
<b>4</b>	<b>Notation</b>	<b>1</b>
<b>5</b>	<b>Module Decomposition</b>	<b>8</b>
<b>6</b>	<b>MIS of Report Manager</b>	<b>10</b>
6.1	Module . . . . .	10
6.2	Uses . . . . .	10
6.3	Syntax . . . . .	10
6.3.1	Exported Constants . . . . .	10
6.3.2	Exported Access Programs . . . . .	11
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
<b>7</b>	<b>MIS of Account Creation Interface</b>	<b>11</b>
7.1	Module . . . . .	11
7.2	Uses . . . . .	11
7.3	Syntax . . . . .	12
7.3.1	Exported Constants . . . . .	12
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 . . . . .	12
<b>8</b>	<b>MIS of Project Manager</b>	<b>12</b>
8.1	Module . . . . .	12
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 . . . . .	13
8.4.5	Local Functions . . . . .	14
<b>9</b>	<b>MIS of Project Collection Manager</b>	<b>14</b>
9.1	Module . . . . .	14
9.2	Uses . . . . .	14
9.3	Syntax . . . . .	14
9.3.1	Exported Constants . . . . .	14
9.3.2	Exported Access Programs . . . . .	14
9.4	Semantics . . . . .	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
<b>10</b>	<b>MIS of Project Database Connector</b>	<b>15</b>
10.1	Module . . . . .	15
10.2	Uses . . . . .	15
10.3	Syntax . . . . .	15
10.3.1	Exported Constants . . . . .	15
10.3.2	Exported Access Programs . . . . .	15
10.4	Semantics . . . . .	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
<b>11</b>	<b>MIS of Core Image Database Connector</b>	<b>16</b>
11.1	Module . . . . .	16
11.2	Uses . . . . .	16
11.3	Syntax . . . . .	16
11.3.1	Exported Constants . . . . .	16
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 . . . . .	18
<b>12</b>	<b>MIS of Account Database Connector</b>	<b>18</b>
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 . . . . .	19
12.4.5	Local Functions . . . . .	19
<b>13</b>	<b>MIS of Account Database</b>	<b>19</b>
13.1	Module . . . . .	19
13.2	Uses . . . . .	19
13.3	Syntax . . . . .	20
13.3.1	Exported Constants . . . . .	20
13.3.2	Exported Access Programs . . . . .	20
13.4	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
<b>14</b>	<b>MIS of Account Update Interface</b>	<b>21</b>
14.1	Module . . . . .	21
14.2	Uses . . . . .	21
14.3	Syntax . . . . .	21
14.3.1	Exported Constants . . . . .	21
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 . . . . .	22

<b>15 MIS of Login Interface</b>	<b>22</b>
15.1 Module . . . . .	22
15.2 Uses . . . . .	22
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 . . . . .	23
15.4.5 Local Functions . . . . .	23
<b>16 MIS of Access Token</b>	<b>23</b>
16.1 Module . . . . .	23
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 . . . . .	24
16.4.3 Assumptions . . . . .	24
16.4.4 Access Routine Semantics . . . . .	24
16.4.5 Local Functions . . . . .	24
<b>17 MIS of Account Creation Interface</b>	<b>24</b>
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 . . . . .	25
17.4.1 State Variables . . . . .	25
17.4.2 Environment Variables . . . . .	25
17.4.3 Assumptions . . . . .	25
17.4.4 Access Routine Semantics . . . . .	25
17.4.5 Local Functions . . . . .	25
<b>18 MIS of Account Database</b>	<b>25</b>
18.1 Module . . . . .	25
18.2 Uses . . . . .	25
18.3 Syntax . . . . .	25

18.3.1	Exported Constants	25
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
<b>19</b>	<b>MIS of Account Update Interface</b>	<b>27</b>
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	27
19.4.3	Assumptions	27
19.4.4	Access Routine Semantics	27
19.4.5	Local Functions	28
<b>20</b>	<b>MIS of Login Interface</b>	<b>28</b>
20.1	Module	28
20.2	Uses	28
20.3	Syntax	28
20.3.1	Exported Constants	28
20.3.2	Exported Access Programs	28
20.4	Semantics	28
20.4.1	State Variables	28
20.4.2	Environment Variables	28
20.4.3	Assumptions	28
20.4.4	Access Routine Semantics	29
20.4.5	Local Functions	29
<b>21</b>	<b>MIS of Access Token</b>	<b>29</b>
21.1	Module	29
21.2	Uses	29
21.3	Syntax	29
21.3.1	Exported Constants	29
21.3.2	Exported Access Programs	29
21.4	Semantics	29
21.4.1	State Variables	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
<b>22</b>	<b>MIS of Labeler</b>	<b>30</b>
22.1	Module . . . . .	30
22.2	Uses . . . . .	30
22.3	Syntax . . . . .	30
22.3.1	Exported Constants . . . . .	30
22.3.2	Exported Access Programs . . . . .	31
22.4	Semantics . . . . .	31
22.4.1	State Variables . . . . .	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
<b>23</b>	<b>MIS of Client</b>	<b>32</b>
23.1	Module . . . . .	32
23.2	Uses . . . . .	32
23.3	Syntax . . . . .	32
23.3.1	Exported Constants . . . . .	32
23.3.2	Exported Access Programs . . . . .	32
23.4	Semantics . . . . .	33
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
<b>24</b>	<b>MIS of User</b>	<b>34</b>
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

<b>25 MIS of Account Creation Controller</b>	<b>35</b>
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 . . . . .	37
<b>26 MIS of Account Update Controller</b>	<b>37</b>
26.1 Module . . . . .	37
26.2 Uses . . . . .	37
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
<b>27 MIS of Authentication Controller</b>	<b>39</b>
27.1 Module . . . . .	39
27.2 Uses . . . . .	39
27.3 Syntax . . . . .	39
27.3.1 Exported Constants . . . . .	39
27.3.2 Exported Access Programs . . . . .	39
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 . . . . .	40
<b>28 MIS of Satellite Image Request Interface</b>	<b>40</b>
28.1 Module . . . . .	40
28.2 Uses . . . . .	40
28.3 Syntax . . . . .	40

28.3.1	Exported Constants	40
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	41
28.4.5	Local Functions	41
<b>29</b>	<b>MIS of Satellite Image Request Controller</b>	<b>41</b>
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	41
29.4.1	State Variables	41
29.4.2	Environment Variables	42
29.4.3	Assumptions	42
29.4.4	Access Routine Semantics	42
29.4.5	Local Functions	42
<b>30</b>	<b>MIS of Satellite Image Request</b>	<b>42</b>
30.1	Module	42
30.2	Uses	42
30.3	Syntax	42
30.3.1	Exported Constants	42
30.3.2	Exported Access Programs	43
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	44
<b>31</b>	<b>MIS of Project Creation Interface</b>	<b>44</b>
31.1	Module	44
31.2	Uses	44
31.3	Syntax	44
31.3.1	Exported Constants	44
31.3.2	Exported Access Programs	44
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 . . . . .	45
31.4.5	Local Functions . . . . .	45
<b>32</b>	<b>MIS of Project Creation Controller</b>	<b>45</b>
32.1	Module . . . . .	45
32.2	Uses . . . . .	45
32.3	Syntax . . . . .	45
32.3.1	Exported Constants . . . . .	45
32.3.2	Exported Access Programs . . . . .	45
32.4	Semantics . . . . .	45
32.4.1	State Variables . . . . .	45
32.4.2	Environment Variables . . . . .	46
32.4.3	Assumptions . . . . .	46
32.4.4	Access Routine Semantics . . . . .	46
32.4.5	Local Functions . . . . .	46
<b>33</b>	<b>MIS of Project</b>	<b>46</b>
33.1	Module . . . . .	46
33.2	Uses . . . . .	46
33.3	Syntax . . . . .	46
33.3.1	Exported Constants . . . . .	46
33.3.2	Exported Access Programs . . . . .	47
33.4	Semantics . . . . .	47
33.4.1	State Variables . . . . .	47
33.4.2	Environment Variables . . . . .	47
33.4.3	Assumptions . . . . .	47
33.4.4	Access Routine Semantics . . . . .	47
33.4.5	Local Functions . . . . .	48
<b>34</b>	<b>MIS of Service Request Failure Interface</b>	<b>48</b>
34.1	Module . . . . .	48
34.2	Uses . . . . .	48
34.3	Syntax . . . . .	48
34.3.1	Exported Constants . . . . .	48
34.3.2	Exported Access Programs . . . . .	48
34.4	Semantics . . . . .	49
34.4.1	State Variables . . . . .	49
34.4.2	Environment Variables . . . . .	49
34.4.3	Assumptions . . . . .	49
34.4.4	Access Routine Semantics . . . . .	49
34.4.5	Local Functions . . . . .	49

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

38.3.1	Exported Constants . . . . .	52
38.3.2	Exported Access Programs . . . . .	53
38.4	Semantics . . . . .	53
38.4.1	State Variables . . . . .	53
38.4.2	Environment Variables . . . . .	53
38.4.3	Assumptions . . . . .	53
38.4.4	Access Routine Semantics . . . . .	53
38.4.5	Local Functions . . . . .	54
<b>39</b>	<b>MIS of Project Selection Interface</b>	<b>54</b>
39.1	Module . . . . .	54
39.2	Uses . . . . .	54
39.3	Syntax . . . . .	54
39.3.1	Exported Constants . . . . .	54
39.3.2	Exported Access Programs . . . . .	54
39.4	Semantics . . . . .	54
39.4.1	State Variables . . . . .	54
39.4.2	Environment Variables . . . . .	54
39.4.3	Assumptions . . . . .	55
39.4.4	Access Routine Semantics . . . . .	55
39.4.5	Local Functions . . . . .	55
<b>40</b>	<b>MIS of Project Selection Controller</b>	<b>55</b>
40.1	Module . . . . .	55
40.2	Uses . . . . .	55
40.3	Syntax . . . . .	55
40.3.1	Exported Constants . . . . .	55
40.3.2	Exported Access Programs . . . . .	55
40.4	Semantics . . . . .	55
40.4.1	State Variables . . . . .	55
40.4.2	Environment Variables . . . . .	56
40.4.3	Assumptions . . . . .	56
40.4.4	Access Routine Semantics . . . . .	56
40.4.5	Local Functions . . . . .	56
<b>41</b>	<b>MIS of Labeling Interface</b>	<b>56</b>
41.1	Module . . . . .	56
41.2	Uses . . . . .	56
41.3	Syntax . . . . .	56
41.3.1	Exported Constants . . . . .	56
41.3.2	Exported Access Programs . . . . .	56
41.4	Semantics . . . . .	57
41.4.1	State Variables . . . . .	57

41.4.2	Environment Variables . . . . .	57
41.4.3	Assumptions . . . . .	57
41.4.4	Access Routine Semantics . . . . .	57
41.4.5	Local Functions . . . . .	57
<b>42</b>	<b>MIS of Labeling Controller</b>	<b>57</b>
42.1	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
<b>43</b>	<b>MIS of Image</b>	<b>59</b>
43.1	Module . . . . .	59
43.2	Uses . . . . .	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
<b>44</b>	<b>MIS of Label Server</b>	<b>62</b>
44.1	Module . . . . .	62
44.2	Uses . . . . .	62
44.3	Syntax . . . . .	62
44.3.1	Exported Constants . . . . .	62
44.3.2	Exported Access Programs . . . . .	62
44.4	Semantics . . . . .	62
44.4.1	State Variables . . . . .	62
44.4.2	Environment Variables . . . . .	62
44.4.3	Assumptions . . . . .	62
44.4.4	Access Routine Semantics . . . . .	62
44.4.5	Local Functions . . . . .	63

<b>45 MIS of Label Database Connector</b>	<b>64</b>
45.1 Module . . . . .	64
45.2 Uses . . . . .	64
45.3 Syntax . . . . .	64
45.3.1 Exported Constants . . . . .	64
45.3.2 Exported Access Programs . . . . .	64
45.4 Semantics . . . . .	64
45.4.1 State Variables . . . . .	64
45.4.2 Environment Variables . . . . .	64
45.4.3 Assumptions . . . . .	64
45.4.4 Access Routine Semantics . . . . .	64
45.4.5 Local Functions . . . . .	65
<b>46 MIS of Label Database</b>	<b>66</b>
46.1 Module . . . . .	66
46.2 Uses . . . . .	66
46.3 Syntax . . . . .	66
46.3.1 Exported Constants . . . . .	66
46.3.2 Exported Access Programs . . . . .	66
46.4 Semantics . . . . .	66
46.4.1 State Variables . . . . .	66
46.4.2 Environment Variables . . . . .	66
46.4.3 Assumptions . . . . .	66
46.4.4 Access Routine Semantics . . . . .	66
46.4.5 Local Functions . . . . .	67
<b>47 MIS of ImageObject Database Connector</b>	<b>68</b>
47.1 Module . . . . .	68
47.2 Uses . . . . .	68
47.3 Syntax . . . . .	68
47.3.1 Exported Constants . . . . .	68
47.3.2 Exported Access Programs . . . . .	68
47.4 Semantics . . . . .	68
47.4.1 State Variables . . . . .	68
47.4.2 Environment Variables . . . . .	68
47.4.3 Assumptions . . . . .	68
47.4.4 Access Routine Semantics . . . . .	68
47.4.5 Local Functions . . . . .	69
<b>48 MIS of ImageObject Database</b>	<b>70</b>
48.1 Module . . . . .	70
48.2 Uses . . . . .	70
48.3 Syntax . . . . .	70



48.3.1	Exported Constants . . . . .	70
48.3.2	Exported Access Programs . . . . .	70
48.4	Semantics . . . . .	70
48.4.1	State Variables . . . . .	70
48.4.2	Environment Variables . . . . .	70
48.4.3	Assumptions . . . . .	70
48.4.4	Access Routine Semantics . . . . .	70
48.4.5	Local Functions . . . . .	71
<b>49</b>	<b>MIS of Labeller Database Connector</b>	<b>72</b>
49.1	Module . . . . .	72
49.2	Uses . . . . .	72
49.3	Syntax . . . . .	72
49.3.1	Exported Constants . . . . .	72
49.3.2	Exported Access Programs . . . . .	72
49.4	Semantics . . . . .	72
49.4.1	State Variables . . . . .	72
49.4.2	Environment Variables . . . . .	72
49.4.3	Assumptions . . . . .	72
49.4.4	Access Routine Semantics . . . . .	72
49.4.5	Local Functions . . . . .	73
<b>50</b>	<b>MIS of Labeller Database</b>	<b>74</b>
50.1	Module . . . . .	74
50.2	Uses . . . . .	74
50.3	Syntax . . . . .	74
50.3.1	Exported Constants . . . . .	74
50.3.2	Exported Access Programs . . . . .	74
50.4	Semantics . . . . .	74
50.4.1	State Variables . . . . .	74
50.4.2	Environment Variables . . . . .	74
50.4.3	Assumptions . . . . .	74
50.4.4	Access Routine Semantics . . . . .	74
50.4.5	Local Functions . . . . .	75
<b>51</b>	<b>MIS of Object Extraction Manager</b>	<b>76</b>
51.1	Module . . . . .	76
51.2	Uses . . . . .	76
51.3	Syntax . . . . .	76
51.3.1	Exported Constants . . . . .	76
51.3.2	Exported Access Programs . . . . .	76
51.4	Semantics . . . . .	76
51.4.1	State Variables . . . . .	76

51.4.2	Environment Variables . . . . .	76
51.4.3	Assumptions . . . . .	76
51.4.4	Access Routine Semantics . . . . .	76
51.4.5	Local Functions . . . . .	77
<b>52</b>	<b>MIS of Label Confidence Service</b>	<b>78</b>
52.1	Module . . . . .	78
52.2	Uses . . . . .	78
52.3	Syntax . . . . .	78
52.3.1	Exported Constants . . . . .	78
52.3.2	Exported Access Programs . . . . .	78
52.4	Semantics . . . . .	78
52.4.1	State Variables . . . . .	78
52.4.2	Environment Variables . . . . .	78
52.4.3	Assumptions . . . . .	78
52.4.4	Access Routine Semantics . . . . .	78
52.4.5	Local Functions . . . . .	79
<b>53</b>	<b>MIS of Object Extraction Service</b>	<b>80</b>
53.1	Module . . . . .	80
53.2	Uses . . . . .	80
53.3	Syntax . . . . .	80
53.3.1	Exported Constants . . . . .	80
53.3.2	Exported Access Programs . . . . .	80
53.4	Semantics . . . . .	80
53.4.1	State Variables . . . . .	80
53.4.2	Environment Variables . . . . .	80
53.4.3	Assumptions . . . . .	80
53.4.4	Access Routine Semantics . . . . .	80
53.4.5	Local Functions . . . . .	81
<b>54</b>	<b>MIS of Image Prior Analyzer</b>	<b>82</b>
54.1	Module . . . . .	82
54.2	Uses . . . . .	82
54.3	Syntax . . . . .	82
54.3.1	Exported Constants . . . . .	82
54.3.2	Exported Access Programs . . . . .	82
54.4	Semantics . . . . .	82
54.4.1	State Variables . . . . .	82
54.4.2	Environment Variables . . . . .	82
54.4.3	Assumptions . . . . .	82
54.4.4	Access Routine Semantics . . . . .	82
54.4.5	Local Functions . . . . .	82

<b>55 MIS of Labeller Expertise Calculator</b>	<b>83</b>
55.1 Module . . . . .	83
55.2 Uses . . . . .	83
55.3 Syntax . . . . .	83
55.3.1 Exported Constants . . . . .	83
55.3.2 Exported Access Programs . . . . .	83
55.4 Semantics . . . . .	83
55.4.1 State Variables . . . . .	83
55.4.2 Environment Variables . . . . .	83
55.4.3 Assumptions . . . . .	83
55.4.4 Access Routine Semantics . . . . .	83
55.4.5 Local Functions . . . . .	84
<b>56 MIS of Image Service Manager</b>	<b>85</b>
56.1 Module . . . . .	85
56.2 Uses . . . . .	85
56.3 Syntax . . . . .	85
56.3.1 Exported Constants . . . . .	85
56.3.2 Exported Access Programs . . . . .	85
56.4 Semantics . . . . .	85
56.4.1 State Variables . . . . .	85
56.4.2 Environment Variables . . . . .	85
56.4.3 Assumptions . . . . .	85
56.4.4 Access Routine Semantics . . . . .	85
56.4.5 Local Functions . . . . .	86
<b>57 MIS of Image Mask Service</b>	<b>87</b>
57.1 Module . . . . .	87
57.2 Uses . . . . .	87
57.3 Syntax . . . . .	87
57.3.1 Exported Constants . . . . .	87
57.3.2 Exported Access Programs . . . . .	87
57.4 Semantics . . . . .	87
57.4.1 State Variables . . . . .	87
57.4.2 Environment Variables . . . . .	87
57.4.3 Assumptions . . . . .	87
57.4.4 Access Routine Semantics . . . . .	87
57.4.5 Local Functions . . . . .	87
<b>58 MIS of Image Selection Service</b>	<b>88</b>
58.1 Module . . . . .	88
58.2 Uses . . . . .	88
58.3 Syntax . . . . .	88

58.3.1	Exported Constants	88
58.3.2	Exported Access Programs	88
58.4	Semantics	88
58.4.1	State Variables	88
58.4.2	Environment Variables	88
58.4.3	Assumptions	88
58.4.4	Access Routine Semantics	88
58.4.5	Local Functions	89
<b>59</b>	<b>MIS of ModelComparisonEvaluation</b>	<b>90</b>
59.1	6.1 Module	90
59.2	6.2 Uses	90
59.3	6.3 Syntax	90
59.3.1	6.3.1 Exported Constants	90
59.3.2	6.3.2 Exported Access Programs	90
59.4	6.4 Semantics	90
59.4.1	6.4.1 State Variables	90
59.4.2	6.4.2 Environment Variables	90
59.4.3	6.4.3 Assumptions	90
59.4.4	6.4.4 Access Routine Semantics	91
59.4.5	6.4.5 Local Functions	91
<b>60</b>	<b>MIS of CrossValidationEvaluation</b>	<b>92</b>
60.1	6.1 Module	92
60.2	6.2 Uses	92
60.3	6.3 Syntax	92
60.3.1	6.3.1 Exported Constants	92
60.3.2	6.3.2 Exported Access Programs	92
60.4	6.4 Semantics	92
60.4.1	6.4.1 State Variables	92
60.4.2	6.4.2 Environment Variables	92
60.4.3	6.4.3 Assumptions	92
60.4.4	6.4.4 Access Routine Semantics	93
60.4.5	6.4.5 Local Functions	93
<b>61</b>	<b>MIS of ModelTrainingService</b>	<b>94</b>
61.1	6.1 Module	94
61.2	6.2 Uses	94
61.3	6.3 Syntax	94
61.3.1	6.3.1 Exported Constants	94
61.3.2	6.3.2 Exported Access Programs	94
61.4	6.4 Semantics	94
61.4.1	6.4.1 State Variables	94

61.4.2	6.4.2 Environment Variables . . . . .	94
61.4.3	6.4.3 Assumptions . . . . .	94
61.4.4	6.4.4 Access Routine Semantics . . . . .	95
61.4.5	6.4.5 Local Functions . . . . .	95
<b>62</b>	<b>MIS of ModelEvaluationService</b>	<b>96</b>
62.1	6.1 Module . . . . .	96
62.2	6.2 Uses . . . . .	96
62.3	6.3 Syntax . . . . .	96
62.3.1	6.3.1 Exported Constants . . . . .	96
62.3.2	6.3.2 Exported Access Programs . . . . .	96
62.4	6.4 Semantics . . . . .	96
62.4.1	6.4.1 State Variables . . . . .	96
62.4.2	6.4.2 Environment Variables . . . . .	96
62.4.3	6.4.3 Assumptions . . . . .	96
62.4.4	6.4.4 Access Routine Semantics . . . . .	97
62.4.5	6.4.5 Local Functions . . . . .	97
<b>63</b>	<b>MIS of ModelManager</b>	<b>98</b>
63.1	6.1 Module . . . . .	98
63.2	6.2 Uses . . . . .	98
63.3	6.3 Syntax . . . . .	98
63.3.1	6.3.1 Exported Constants . . . . .	98
63.3.2	6.3.2 Exported Access Programs . . . . .	98
63.4	6.4 Semantics . . . . .	98
63.4.1	6.4.1 State Variables . . . . .	98
63.4.2	6.4.2 Environment Variables . . . . .	98
63.4.3	6.4.3 Assumptions . . . . .	99
63.4.4	6.4.4 Access Routine Semantics . . . . .	99
63.4.5	6.4.5 Local Functions . . . . .	99
<b>64</b>	<b>MIS of ModelCreation (Abstract)</b>	<b>100</b>
64.1	6.1 Module . . . . .	100
64.2	6.2 Uses . . . . .	100
64.3	6.3 Syntax . . . . .	100
64.3.1	6.3.1 Exported Constants . . . . .	100
64.3.2	6.3.2 Exported Access Programs . . . . .	100
64.4	6.4 Semantics . . . . .	100
64.4.1	6.4.1 State Variables . . . . .	100
64.4.2	6.4.2 Environment Variables . . . . .	100
64.4.3	6.4.3 Assumptions . . . . .	100
64.4.4	6.4.4 Access Routine Semantics . . . . .	100
64.4.5	6.4.5 Local Functions . . . . .	101

<b>65 MIS of MLModelDatabase</b>	<b>102</b>
65.1 6.1 Module . . . . .	102
65.2 6.2 Uses . . . . .	102
65.3 6.3 Syntax . . . . .	102
65.3.1 6.3.1 Exported Constants . . . . .	102
65.3.2 6.3.2 Exported Access Programs . . . . .	102
65.4 6.4 Semantics . . . . .	102
65.4.1 6.4.1 State Variables . . . . .	102
65.4.2 6.4.2 Environment Variables . . . . .	102
65.4.3 6.4.3 Assumptions . . . . .	102
65.4.4 6.4.4 Access Routine Semantics . . . . .	103
65.4.5 6.4.5 Local Functions . . . . .	103
<b>66 MIS of OtherModelCreation</b>	<b>104</b>
66.1 6.1 Module . . . . .	104
66.2 6.2 Uses . . . . .	104
66.3 6.3 Syntax . . . . .	104
66.3.1 6.3.1 Exported Constants . . . . .	104
66.3.2 6.3.2 Exported Access Programs . . . . .	104
66.4 6.4 Semantics . . . . .	104
66.4.1 6.4.1 State Variables . . . . .	104
66.4.2 6.4.2 Environment Variables . . . . .	104
66.4.3 6.4.3 Assumptions . . . . .	104
66.4.4 6.4.4 Access Routine Semantics . . . . .	104
66.4.5 6.4.5 Local Functions . . . . .	105
<b>67 MIS of CNNModelCreation</b>	<b>105</b>
67.1 6.1 Module . . . . .	105
67.2 6.2 Uses . . . . .	105
67.3 6.3 Syntax . . . . .	105
67.3.1 6.3.1 Exported Constants . . . . .	105
67.3.2 6.3.2 Exported Access Programs . . . . .	105
67.4 6.4 Semantics . . . . .	105
67.4.1 6.4.1 State Variables . . . . .	105
67.4.2 6.4.2 Environment Variables . . . . .	105
67.4.3 6.4.3 Assumptions . . . . .	105
67.4.4 6.4.4 Access Routine Semantics . . . . .	106
67.4.5 6.4.5 Local Functions . . . . .	106
<b>68 Exception Handling</b>	<b>106</b>
68.1 Frontend Handling (React) . . . . .	106
68.2 Backend Handling (Python) . . . . .	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 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

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

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', 'Regr 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., '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  
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
<b>Behaviour-Hiding Module</b>	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
<b>Software Decision Module</b>	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 Report Manager

### 6.1 Module

RM (ReportManager)

### 6.2 Uses

LabelDBConnector [45](#)

ObjectsOnImageDBConnector [47](#)

RawImageDBConnector [11](#)

ProjectDBConnector [10](#)

### 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 Controller [25](#)

## 7.3 Syntax

### 7.3.1 Exported Constants

None

### 7.3.2 Exported Access Programs

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

## 7.4 Semantics

### 7.4.1 State Variables

None

### 7.4.2 Environment Variables

win: 2D sequence of coloured pixels

### 7.4.3 Assumptions

None

### 7.4.4 Access Routine Semantics

renderPage(userType):

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

submitForm(formData):

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

### 7.4.5 Local Functions

None

## 8 MIS of Project Manager

### 8.1 Module

PM (ProjectManager)

## 8.2 Uses

ProjectCollectionManager [9](#)

CoreImageDBConnector [11](#)

## 8.3 Syntax

### 8.3.1 Exported Constants

None

### 8.3.2 Exported Access Programs

Name	In	Out	Exceptions
createProject	projectName: String, metadata: Map(String, String)	Project	DatabaseException
addImageToProject	projectId: String, byte[]	void	DatabaseException

## 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

ProjectDBConnector [10](#)

### 9.3 Syntax

#### 9.3.1 Exported Constants

None

#### 9.3.2 Exported Access Programs

Name	In	Out	Exceptions
getAvailableProjects	None	List(Project)	DatabaseException

### 9.4 Semantics

#### 9.4.1 State Variables

None

#### 9.4.2 Environment Variables

None

#### 9.4.3 Assumptions

None

#### 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)	DatabaseException
storeProject	project : Project	None	DatabaseException

### 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

None

### 11.3.2 Exported Access Programs

Name	In	Out	Exceptions
storeImage	projectId: String imageData: byte[]	String	DatabaseException
fetchImage	imageId: String	Image	DatabaseException
fetchImagesForProject	projectId: 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 Database [18](#)

### 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	-
makeDBConnection	credentials	-	-

### 12.4 Semantics

#### 12.4.1 State Variables

None

#### 12.4.2 Environment Variables

databaseConnection: connection to relational database

#### 12.4.3 Assumptions

None

#### 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:

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

userExists(email):

- output: out :=

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

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

None

## 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:

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

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 :=

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

### 13.4.5 Local Functions

None

## 14 MIS of Account Update Interface

### 14.1 Module

Account Update Interface

### 14.2 Uses

Account Update Controller [26](#)

### 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 [27](#)

### 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:  $\text{out} := \text{currentTime} > \text{expirationTime}$

renew():

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

### 16.4.5 Local Functions

None

## 17 MIS of Account Creation Interface

### 17.1 Module

Account Creation Interface

### 17.2 Uses

Account Creation Controller [25](#)

### 17.3 Syntax

#### 17.3.1 Exported Constants

None

#### 17.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	Enum[labeler, client]	-	-
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**

None



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

updateUser(user):

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

userExists(email):

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

### 18.4.5 Local Functions

None

## 19 MIS of Account Update Interface

### 19.1 Module

Account Update Interface

### 19.2 Uses

Account Update Controller [26](#)

### 19.3 Syntax

#### 19.3.1 Exported Constants

None

#### 19.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	User	-	-
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 Controller [27](#)

### 20.3 Syntax

#### 20.3.1 Exported Constants

None

#### 20.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
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

None

#### 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():

- output: `out := currentTime > 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 User [24](#)

### **22.3 Syntax**

#### **22.3.1 Exported Constants**

None

### 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):

- 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 User [24](#)

### 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	-	-
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():

- 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

### 23.4.5 Local Functions

None



## 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	-
getPassword	-	string	-
getProfilePic	-	Image	-
setEmail	string	-	-
setPassword	string	-	-
setProfilePic	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

None

#### 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(newProfilePic):

- 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 Interface [17](#)

Account Database [18](#)

User [24](#)

Labeler [22](#)

Client [23](#)

### 25.3 Syntax

#### 25.3.1 Exported Constants

None

### 25.3.2 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

## 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):

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

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

**Where:**

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

createUser(formData, userType):

- output:  $\text{out} :=$

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

uploadUser(newUser):

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

#### 25.4.5 Local Functions

- hasRequiredFields(data, fields) =  $\forall \text{field} \in \text{fields}, (\text{data}[\text{field}] \neq \text{""})$
- 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:

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

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

## 26 MIS of Account Update Controller

### 26.1 Module

Account Update Controller

### 26.2 Uses

Account Update Interface [19](#)

Account Database [18](#)

User [24](#)

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

getUser(email):

- transition:  $\text{user} := \text{AccountDatabase.retreiveUser}(\text{email})$

requestUpdate(updatedUser):

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

### 26.4.5 Local Functions

None

## 27 MIS of Authentication Controller

### 27.1 Module

Authentication Controller

### 27.2 Uses

Login Interface [41](#)

Account Database [18](#)

Access Token [21](#)

### 27.3 Syntax

#### 27.3.1 Exported Constants

None

#### 27.3.2 Exported Access Programs

Name	In	Out	Exceptions
validCredentials	(string, string)	boolean	-
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:  $\text{out} := \text{AccountDatabase.retreiveUser(email)} \neq \text{null}$   
 $\wedge \text{AccountDatabase.retreiveUser(email).getPassword()} == \text{password}$

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 Controller [29](#)

### 28.3 Syntax

#### 28.3.1 Exported Constants

None

#### 28.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
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

None

#### 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 Interface [28](#)

Satellite Image Request [30](#)

### 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	SatelliteImageRequest	-	-

### 29.4 Semantics

#### 29.4.1 State Variables

None



### 29.4.2 Environment Variables

None

### 29.4.3 Assumptions

None

### 29.4.4 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.

### 29.4.5 Local Functions

- calculateCost(imgRequest):  $\text{out} :=$  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

None

### 30.3.2 Exported Access Programs

Name	In	Out	Exceptions
getLocation	-	(float, float)	-
getRadius	-	float	-
getDate	-	Date	-
setLocation	(float, float)	-	-
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 Controller [32](#)

### 31.3 Syntax

#### 31.3.1 Exported Constants

None

#### 31.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderPage	-	-	-
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

None

### 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 Manager [8](#)

Project Creation Interface [31](#)

Project [33](#)

### 32.3 Syntax

#### 32.3.1 Exported Constants

None

#### 32.3.2 Exported Access Programs

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

### 32.4 Semantics

#### 32.4.1 State Variables

None

### **32.4.2 Environment Variables**

None

### **32.4.3 Assumptions**

None

### **32.4.4 Access Routine Semantics**

validateForm(formData):

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

createNewProject(formData):

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

### **32.4.5 Local Functions**

- calculateEstimatedCost(project):  $\text{out} :=$  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**

None

### 33.3.2 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)	-	-

## 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():

- 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 Manager [8](#)

### 35.3 Syntax

#### 35.3.1 Exported Constants

None

#### 35.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayUploadImages		-	-



## 35.4 Semantics

### 35.4.1 State Variables

None

### 35.4.2 Environment Variables

win: 2D sequence of coloured pixels

### 35.4.3 Assumptions

None

### 35.4.4 Access Routine Semantics

displayUploadImages():

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

### 35.4.5 Local Functions

- validateImage(image): out :=

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

## 36 MIS of Report Interface

### 36.1 Module

Report Interface

### 36.2 Uses

Report Controller [37](#)

### 36.3 Syntax

#### 36.3.1 Exported Constants

None

#### 36.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayStats	-	-	-

## 36.4 Semantics

### 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 Interface [36](#)

Report [38](#)

### 37.3 Syntax

#### 37.3.1 Exported Constants

None

#### 37.3.2 Exported Access Programs

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

## **37.4 Semantics**

### **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**

None

### 38.3.2 Exported Access Programs

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

## 38.4 Semantics

### 38.4.1 State Variables

- labeledImages: list[Image]
- reviewedImages: 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 Controller [40](#)

### 39.3 Syntax

#### 39.3.1 Exported Constants

None

#### 39.3.2 Exported Access Programs

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

### 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 Manager [9](#)

Project Selection Interface [39](#)

Project [33](#)

### 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 Controller [42](#)

Image [43](#)

### 41.3 Syntax

#### 41.3.1 Exported Constants

None

#### 41.3.2 Exported Access Programs

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

## 41.4 Semantics

### 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 Interface [41](#)

Label ??

## 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 |  $\ell.id \neq$  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	-
getImageID	-	int	-
getDimensions	-	(float, float)	-
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

None

### **43.4.3 Assumptions**

None

### **43.4.4 Access Routine Semantics**

getProjectID():

- output: out := projectID

getImageID():

- output: out := imageID

getDimensions():

- output: out := (width, height)

getImageData():

- output: out := imageData

### **43.4.5 Local Functions**

None



## 44 MIS of Label Server

### 44.1 Module

Label Server

### 44.2 Uses

Labeling Controller [42](#)

Label ??

Label Database Connector [45](#)

### 44.3 Syntax

#### 44.3.1 Exported Constants

None

#### 44.3.2 Exported Access Programs

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

### 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 Database [46](#)

Label ??

### 45.3 Syntax

#### 45.3.1 Exported Constants

None

#### 45.3.2 Exported Access Programs

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

### 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 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`

#### 45.4.5 Local Functions

None



## 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 Conne- ction	Label	-	ConnectionError
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 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  $\neg(\text{requestor} \in \text{users})$

#### **46.4.5 Local Functions**

None

## 47 MIS of ImageObject Database Connector

### 47.1 Module

ImageObject Database Connector

### 47.2 Uses

ImageObject Database [48](#)

ImageObject ??

### 47.3 Syntax

#### 47.3.1 Exported Constants

None

#### 47.3.2 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

### 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 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

#### **47.4.5 Local Functions**

None

## 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 Image Object	ImageObject	-	ValueError
makeDB Connection	ImageObject	-	ConnectionError
get Image Objects	String	list[ImageObject]	ValueError

### 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 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  $\neg(\text{requestor} \in \text{users})$

#### 48.4.5 Local Functions

None

=====

## 49 MIS of Labeller Database Connector

### 49.1 Module

Labeller Database Connector

### 49.2 Uses

Labeller Database ??

ImageObject ??

### 49.3 Syntax

#### 49.3.1 Exported Constants

None

#### 49.3.2 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

### 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 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`

#### 49.4.5 Local Functions

None



## 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- beller	Labeller	-	ValueError
makeDB Conne- ction	Credentials	-	ConnectionError
get La- beller	String	list[Labeller]	ValueError

### 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:  $\text{Labellers} := \text{Labellers} \cup o$
- 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  $\neg(\text{requestor} \in \text{users})$

#### 50.4.5 Local Functions

None

## 51 MIS of Object Extraction Manager

### 51.1 Module

Object Extraction Manager

### 51.2 Uses

ImageObject Database Connector [47](#)

Label Database Connector [45](#)

Labeller Database Connector [49](#)

Image Prior Analyzer [54](#)

Label Confidence Service [52](#)

Object Extraction Service [53](#)

Labeller Expertise Calculator [55](#)

### 51.3 Syntax

#### 51.3.1 Exported Constants

None

#### 51.3.2 Exported Access Programs

Name	In	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

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

#### **51.4.5 Local Functions**

generate query:

## 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
getConfidencdist	list[label], list[labeller], list[ImageObject]	list[list[float]]	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

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  $(\exists \text{imageobject} \in \text{imageobjects} \mid$

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

#### **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[labeller], list[ImageObject], list[list[float]]	list[ImageObject]	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

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

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

#### 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	list[label], list[labeller], list[ImageObject], list[list[float]]	list[dict[string, tuple[float, float]]]	ValueError

### 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  $(\exists \text{label} \in \text{labels} \mid \neg(\text{label} \in L))$   
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  $(\exists i, j | x = \text{confidence}[i][j] : \neg(x \in \mathbb{R}))$

#### 55.4.5 Local Functions

## 56 MIS of Image Service Manager

### 56.1 Module

Image Service Manager

### 56.2 Uses

ImageObject Database Connector [47](#)

Labeller Database Connector [49](#)

Image Mask Service [57](#)

Image Selection Engine ??

### 56.3 Syntax

#### 56.3.1 Exported Constants

None

#### 56.3.2 Exported Access Programs

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

### 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 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  $\neg(n \in \mathbb{N})$

### 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
getImageMask	Image	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
getNextImages	List[Image], List[ImageObjects], Labeller	List[Image]	ValueError

### 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 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  $(\exists \text{imageobject} \in \text{imageobjects} \mid \neg(\text{imageobject} \in I))$

### 58.4.5 Local Functions

None



## 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
<code>evaluateModel</code>	<code>String modelId,</code> <code>TestDataset testData</code>	<code>EvaluationResult</code>	<code>ModelNotFoundError,</code> <code>ValueError</code>

### 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.

#### 59.4.4 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.

#### 59.4.5 6.4.5 Local Functions

None

## 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 6.3.1 Exported Constants

None

#### 60.3.2 6.3.2 Exported Access Programs

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

### 60.4 6.4 Semantics

#### 60.4.1 6.4.1 State Variables

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

#### 60.4.2 6.4.2 Environment Variables

None

#### 60.4.3 6.4.3 Assumptions

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

#### 60.4.4 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.

#### 60.4.5 6.4.5 Local Functions

None

## 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 6.3.2 Exported Access Programs

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

### 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.

#### 61.4.4 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.

#### 61.4.5 6.4.5 Local Functions

`None`

## 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 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

### 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 6.4.2 Environment Variables

None

#### 62.4.3 6.4.3 Assumptions

- The modelId references a trained model.

#### 62.4.4 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.

#### 62.4.5 6.4.5 Local Functions

None



## 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 params	void	ValueError
updateModelStatus	String modelId, String status	void	ModelNotFoundError
fetchModel	String modelId	MLModel	ModelNotFoundError
deleteModel	String modelId	void	ModelNotFoundError

### 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

None

### 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

None

## 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 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
createModel	-	MLModel	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

None

## 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	void	DatabaseError
fetchModel	String modelId	MLModel	ModelNotFoundError, DatabaseError
deleteModel	String modelId	void	ModelNotFoundError, DatabaseError
updateModel	String modelId, Map<String, Any> updates	void	ModelNotFoundError, DatabaseError

### 65.4 6.4 Semantics

#### 65.4.1 6.4.1 State Variables

- dbConnection: Connection (Active DB connection)

#### 65.4.2 6.4.2 Environment Variables

- External database system (accessed via dbConnection)

#### 65.4.3 6.4.3 Assumptions

- dbConnection is valid and open.

#### 65.4.4 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.

#### 65.4.5 6.4.5 Local Functions

None

## 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 6.3.1 Exported Constants

None

#### 66.3.2 6.3.2 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.



#### 67.4.4 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.

#### 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

[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) 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 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)