

12.0.2.0 - Implementation Guide: Report Generation Core

Week 2: Report Generation Core - Detailed Implementation Plan

Overview

Week 2 focuses on building the **Report Generation Engine** that transforms processed API data into professional, customer-deliverable PDF reports. This is the core deliverable that customers will receive, so quality and professional presentation are paramount.

Week 2 Objectives

Primary Goals

1. **PDF Generation Engine:** Create professional PDF reports using reportlab
2. **Report Templates:** Modular, reusable report section templates
3. **Professional Formatting:** VAST branding, consistent styling, professional layout
4. **Data Integration:** Seamless integration with processed API data from Week 1

Success Criteria

- Generate customer-ready PDF reports matching the example format
- Professional formatting with VAST branding and consistent styling
- Modular template system for easy maintenance and updates
- Complete integration with existing API data collection

Daily Implementation Schedule

Day 1-2: PDF Generation Foundation

Focus: Core PDF generation infrastructure and basic document structure

Day 3: Report Content Implementation

Focus: Hardware inventory, network configuration, and data sections

Day 4: Professional Formatting and Branding

Focus: VAST branding, styling, tables, and visual elements

Day 5: Integration and Testing

Focus: End-to-end testing and refinement

Day 1-2: PDF Generation Foundation

Task 2.1: Core PDF Generator Class

Implementation: `src/reports/pdf_generator.py`

```
1 from reportlab.lib.pagesizes import letter, A4
2 from reportlab.lib.styles import import getSampleStyleSheet, ParagraphStyle
```

```

3 from reportlab.lib.units import inch
4 from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer,
  Table, TableStyle, PageBreak
5 from reportlab.lib import colors
6 from reportlab.lib.enums import TA_CENTER, TA_LEFT, TA_RIGHT
7 from reportlab.pdfgen import canvas
8 from datetime import datetime
9 import os
10 from typing import Dict, List, Any
11
12 class VastPDFGenerator:
13     """Professional PDF report generator for VAST As-Built
14     reports."""
15
16     def __init__(self, output_path: str, cluster_data: Dict):
17         """
18         Initialize PDF generator with output path and cluster data.
19
20         Args:
21             output_path: Full path for the output PDF file
22             cluster_data: Processed cluster data from data processor
23         """
24         self.output_path = output_path
25         self.cluster_data = cluster_data
26         self.doc = SimpleDocTemplate(
27             output_path,
28             pagesize=A4,
29             rightMargin=72,
30             leftMargin=72,
31             topMargin=72,
32             bottomMargin=72
33         )
34         self.styles = self._create_custom_styles()
35         self.story = []
36
37     def _create_custom_styles(self) -> Dict:
38         """Create custom styles for VAST branding and professional
39         formatting."""
40         styles = getSampleStyleSheet()
41
42         # VAST Brand Colors
43         vast_blue = colors.Color(0.0, 0.3, 0.6) # VAST primary blue
44         vast_gray = colors.Color(0.4, 0.4, 0.4) # VAST secondary
45         gray
46
47         custom_styles = {
48             'title': ParagraphStyle(
49                 'VastTitle',
50                 parent=styles['Title'],
51                 fontSize=24,
52                 textColor=vast_blue,
53                 spaceAfter=30,
54                 alignment=TA_CENTER,
55                 fontName='Helvetica-Bold'
56             ),
57             'heading1': ParagraphStyle(
58                 'VastHeading1',
59                 parent=styles['Heading1'],
60                 fontSize=18,
61                 textColor=vast_blue,
62                 spaceAfter=12,
63                 spaceBefore=20,
64                 fontName='Helvetica-Bold'
65             ),
66             'heading2': ParagraphStyle(
67                 'VastHeading2',
68                 parent=styles['Heading2'],
69                 fontSize=14,
70                 textColor=vast_blue,
71                 spaceAfter=8,
72                 spaceBefore=16,

```

```

70         fontName='Helvetica-Bold'
71     ),
72     'normal': ParagraphStyle(
73         'VastNormal',
74         parent=styles['Normal'],
75         fontSize=10,
76         spaceAfter=6,
77         fontName='Helvetica'
78     ),
79     'table_header': ParagraphStyle(
80         'VastTableHeader',
81         parent=styles['Normal'],
82         fontSize=9,
83         textColor=colors.white,
84         fontName='Helvetica-Bold',
85         alignment=TA_CENTER
86     ),
87     'table_cell': ParagraphStyle(
88         'VastTableCell',
89         parent=styles['Normal'],
90         fontSize=9,
91         fontName='Helvetica'
92     )
93 }
94
95 return custom_styles
96
97 def generate_report(self) -> bool:
98     """
99     Generate the complete PDF report.
100
101     Returns:
102         bool: True if successful, False otherwise
103     """
104     try:
105         # Build report sections in order
106         self._add_title_page()
107         self._add_table_of_contents()
108         self._add_executive_summary()
109         self._add_hardware_inventory()
110         self._add_network_configuration()
111         self._add_data_protection()
112         self._add_support_information()
113
114         # Build the PDF
115         self.doc.build(self.story,
116 onFirstPage=self._add_header_footer,
117 onLaterPages=self._add_header_footer)
118
119         return True
120
121     except Exception as e:
122         print(f"Error generating PDF report: {e}")
123         return False
124
125 def _add_header_footer(self, canvas, doc):
126     """Add header and footer to each page."""
127     canvas.saveState()
128
129     # Header
130     canvas.setFont('Helvetica-Bold', 10)
131     canvas.setFillColor(colors.Color(0.0, 0.3, 0.6))
132     canvas.drawString(72, doc.height + 50, "VAST Data As-Built
133 Report")
134
135     # Footer
136     canvas.setFont('Helvetica', 8)
137     canvas.setFillColor(colors.gray)
138     canvas.drawString(72, 30, f"Generated:
139 {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")

```

```

137         canvas.drawRightString(doc.width + 72, 30, f"Page
{doc.page}")
138
139         canvas.restoreState()

```

Task 2.2: Title Page Generator

```

1  def _add_title_page(self):
2      """Create professional title page with VAST branding."""
3
4      # VAST Logo placeholder (would be actual logo in production)
5      logo_placeholder = Paragraph(
6          "<b>VAST DATA</b>",
7          ParagraphStyle(
8              'Logo',
9              fontSize=36,
10             textColor=colors.Color(0.0, 0.3, 0.6),
11             alignment=TA_CENTER,
12             fontName='Helvetica-Bold'
13         )
14     )
15
16     # Report title
17     title = Paragraph("As-Built Report", self.styles['title'])
18
19     # Cluster information
20     cluster_name = self.cluster_data.get('cluster_overview',
21     {}).get('cluster_name', 'Unknown')
22     cluster_psnt = self.cluster_data.get('cluster_overview',
23     {}).get('cluster_psnt', 'Unknown')
24
25     subtitle = Paragraph(
26         f"<b>Cluster:</b> {cluster_name}<br/>"
27         f"<b>PSNT:</b> {cluster_psnt}<br/>"
28         f"<b>Generated:</b> {datetime.now().strftime('%B %d, %Y')}",
29         ParagraphStyle(
30             'Subtitle',
31             fontSize=14,
32             alignment=TA_CENTER,
33             spaceAfter=20
34         )
35     )
36
37     # Add elements to story
38     self.story.extend([
39         Spacer(1, 2*inch),
40         logo_placeholder,
41         Spacer(1, 1*inch),
42         title,
43         Spacer(1, 0.5*inch),
44         subtitle,
45         PageBreak()
46     ])
47
48     def _add_table_of_contents(self):
49         """Generate table of contents."""
50         toc_title = Paragraph("Table of Contents",
51         self.styles['heading1'])
52
53         toc_items = [
54             "1. Executive Summary",
55             "2. Hardware Inventory",
56             "3. Network Configuration",
57             "4. Data Protection",
58             "5. Support Information"
59         ]
60
61         self.story.append(toc_title)
62
63         for item in toc_items:
64             toc_item = Paragraph(item, self.styles['normal'])






```

```

62         self.story.append(toc_item)
63
64         self.story.append(PageBreak())

```

Day 1-2 Deliverables:

-  Core PDF generator class with VAST branding
-  Custom styles and formatting system
-  Title page with cluster information
-  Table of contents generation
-  Header/footer system with page numbering

Day 3: Report Content Implementation

Task 2.3: Executive Summary Section

```

1  def _add_executive_summary(self):
2      """Create executive summary section."""
3
4      # Section title
5      title = Paragraph("Executive Summary", self.styles['heading1'])
6      self.story.append(title)
7
8      # Cluster overview data
9      cluster_overview = self.cluster_data.get('cluster_overview', {})
10
11     # Overview paragraph
12     overview_text = f"""
13     This document provides comprehensive as-built documentation for
the VAST Data cluster
14     deployment completed for <b>{cluster_overview.get('customer_name',
'Customer')}</b>.
15     The cluster has been successfully installed, configured, and
validated according to
16     VAST Data best practices and customer requirements.
17     """
18
19     overview = Paragraph(overview_text, self.styles['normal'])
20     self.story.append(overview)
21     self.story.append(Spacer(1, 12))
22
23     # Cluster details table
24     cluster_details = [
25         ['Cluster Name', cluster_overview.get('cluster_name',
'Unknown')],
26         ['Cluster PSNT', cluster_overview.get('cluster_psnt',
'Unknown')],
27         ['VAST OS Version', cluster_overview.get('cluster_version',
'Unknown')],
28         ['Cluster GUID', cluster_overview.get('cluster_guid',
'Unknown')],
29         ['Total Usable Capacity',
cluster_overview.get('usable_capacity', 'Unknown')],
30         ['Licensed Capacity',
cluster_overview.get('licensed_capacity', 'Unknown')],
31         ['VMS VIP', cluster_overview.get('vms_vip', 'Unknown')]
32     ]
33
34     # Create table
35     cluster_table = Table(cluster_details, colWidths=[2.5*inch,
3*inch])
36     cluster_table.setStyle(TableStyle([
37         ('BACKGROUND', (0, 0), (0, -1), colors.Color(0.9, 0.9, 0.9)),
38         ('TEXTCOLOR', (0, 0), (-1, -1), colors.black),
39         ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
40         ('FONTNAME', (0, 0), (0, -1), 'Helvetica-Bold'),

```

```

41         ('FONTNAME', (1, 0), (1, -1), 'Helvetica'),
42         ('FONTSIZE', (0, 0), (-1, -1), 9),
43         ('GRID', (0, 0), (-1, -1), 1, colors.black),
44         ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
45     ]))
46
47     self.story.append(cluster_table)
48     self.story.append(PageBreak())

```

Task 2.4: Hardware Inventory Section

```

1  def _add_hardware_inventory(self):
2      """Create hardware inventory section with tables."""
3
4      title = Paragraph("Hardware Inventory", self.styles['heading1'])
5      self.story.append(title)
6
7      # CBoxes section
8      self._add_cboxes_table()
9      self.story.append(Spacer(1, 20))
10
11     # DBoxes section
12     self._add_dboxes_table()
13     self.story.append(Spacer(1, 20))
14
15     # Switches section
16     self._add_switches_table()
17     self.story.append(PageBreak())
18
19 def _add_cboxes_table(self):
20     """Add CBoxes hardware table."""
21
22     subtitle = Paragraph("CBoxes (Compute)", self.styles['heading2'])
23     self.story.append(subtitle)
24
25     # Get CBoxes data
26     cboxes_data = self.cluster_data.get('hardware_inventory',
27     {}).get('cboxes', [])
28
29     # Table headers
30     headers = ['Component', 'Model', 'Serial Number', 'Rack
31     Position', 'CNodes', 'Management IP']
32     table_data = [headers]
33
34     # Add CBox data rows
35     for i, cbox in enumerate(cboxes_data, 1):
36         row = [
37             f"CBox-{i}",
38             cbox.get('model', 'Unknown'),
39             cbox.get('serial', 'Unknown'),
40             f"U{cbox.get('rack_position', 'Unknown')}",
41             str(cbox.get('node_count', 'Unknown')),
42             cbox.get('management_ip', 'Unknown')
43         ]
44         table_data.append(row)
45
46     # Create table
47     cboxes_table = Table(table_data, colWidths=[1*inch, 1*inch,
48     1.2*inch, 1*inch, 0.8*inch, 1.2*inch])
49     cboxes_table.setStyle(TableStyle([
50         # Header styling
51         ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
52         ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
53         ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
54         ('FONTSIZE', (0, 0), (-1, 0), 9),
55         ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
56
57         # Data rows styling
58         ('BACKGROUND', (0, 1), (-1, -1), colors.white),
59         ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
60         ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),

```

```

58         ('FONTSIZE', (0, 1), (-1, -1), 8),
59         ('ALIGN', (0, 1), (-1, -1), 'CENTER'),
60
61         # Grid and borders
62         ('GRID', (0, 0), (-1, -1), 1, colors.black),
63         ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
64
65         # Alternating row colors
66         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white,
colors.Color(0.95, 0.95, 0.95)])
67     ]))
68
69     self.story.append(cboxes_table)
70
71     # Add summary information
72     total_cnodes = sum(cbox.get('node_count', 0) for cbox in
cboxes_data)
73     cable_type = cboxes_data[0].get('cable_type', 'Unknown') if
cboxes_data else 'Unknown'
74
75     summary_text = f"""
76     <b>Total CNodes:</b> {total_cnodes}<br/>
77     <b>CNode Cable Type:</b> {cable_type}<br/>
78     <b>Required Ports per Switch:</b> {total_cnodes * 2}
79     """
80
81     summary = Paragraph(summary_text, self.styles['normal'])
82     self.story.append(summary)
83
84     def _add_dboxes_table(self):
85         """Add DBoxes hardware table."""
86
87         subtitle = Paragraph("DBoxes (Data)", self.styles['heading2'])
88         self.story.append(subtitle)
89
90         # Get DBoxes data
91         dboxes_data = self.cluster_data.get('hardware_inventory',
{})}.get('dboxes', [])
92
93         # Table headers
94         headers = ['Component', 'Model', 'Serial Number', 'Rack
Position', 'DNodes', 'Management IP']
95         table_data = [headers]
96
97         # Add DBox data rows
98         for i, dbox in enumerate(dboxes_data, 1):
99             row = [
100                 f"DBox-{100 + i}", # DBox naming convention
101                 dbox.get('model', 'Unknown'),
102                 dbox.get('serial', 'Unknown'),
103                 f"U{dbox.get('rack_position', 'Unknown')}",
104                 str(dbox.get('node_count', 'Unknown')),
105                 dbox.get('management_ip', 'Unknown')
106             ]
107             table_data.append(row)
108
109         # Create table with same styling as CBoxes
110         dboxes_table = Table(table_data, colWidths=[1*inch, 1*inch,
1.2*inch, 1*inch, 0.8*inch, 1.2*inch])
111         dboxes_table.setStyle(TableStyle([
112             # Same styling as CBoxes table
113             ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
114             ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
115             ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
116             ('FONTSIZE', (0, 0), (-1, 0), 9),
117             ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
118             ('BACKGROUND', (0, 1), (-1, -1), colors.white),
119             ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
120             ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
121             ('FONTSIZE', (0, 1), (-1, -1), 8),
122             ('ALIGN', (0, 1), (-1, -1), 'CENTER'),

```

```

123         ('GRID', (0, 0), (-1, -1), 1, colors.black),
124         ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
125         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white,
colors.Color(0.95, 0.95, 0.95)])
126     ]))
127
128     self.story.append(dboxes_table)
129
130     # Add summary information
131     total_dnodes = sum(dbox.get('node_count', 0) for dbox in
dboxes_data)
132     cable_type = dboxes_data[0].get('cable_type', 'Unknown') if
dboxes_data else 'Unknown'
133
134     summary_text = f"""
135     <b>Total DNodes:</b> {total_dnodes}<br/>
136     <b>DNode Cable Type:</b> {cable_type}<br/>
137     <b>Required Ports per Switch:</b> {total_dnodes}
138     """
139
140     summary = Paragraph(summary_text, self.styles['normal'])
141     self.story.append(summary)
142
143     def _add_switches_table(self):
144         """Add network switches table."""
145
146         subtitle = Paragraph("Network Switches", self.styles['heading2'])
147         self.story.append(subtitle)
148
149         # Get switches data
150         switches_data = self.cluster_data.get('hardware_inventory',
{}).get('switches', [])
151
152         # Table headers
153         headers = ['Component', 'Model', 'Serial Number', 'Rack
Position', 'Ports', 'Firmware']
154         table_data = [headers]
155
156         # Add switch data rows
157         for i, switch in enumerate(switches_data, 1):
158             role = switch.get('role', 'A' if i % 2 == 1 else 'B') #
Alternate A/B
159             row = [
160                 f"Switch {role}",
161                 switch.get('model', 'Unknown'),
162                 switch.get('serial', 'Unknown'),
163                 f"U{switch.get('rack_position', 'Unknown')}",
164                 f"{switch.get('port_count',
'Unknown')}\tX{switch.get('port_speed', 'Unknown')}",
165                 switch.get('firmware_version', 'Unknown')
166             ]
167             table_data.append(row)
168
169         # Create table
170         switches_table = Table(table_data, colWidths=[1*inch, 1.2*inch,
1.2*inch, 1*inch, 1*inch, 1*inch])
171         switches_table.setStyle(TableStyle([
172             # Same styling pattern as other tables
173             ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
174             ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
175             ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
176             ('FONTSIZE', (0, 0), (-1, 0), 9),
177             ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
178             ('BACKGROUND', (0, 1), (-1, -1), colors.white),
179             ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
180             ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
181             ('FONTSIZE', (0, 1), (-1, -1), 8),
182             ('ALIGN', (0, 1), (-1, -1), 'CENTER'),
183             ('GRID', (0, 0), (-1, -1), 1, colors.black),
184             ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),

```







```

185         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white,
186         colors.Color(0.95, 0.95, 0.95)])
187     ]))
188     self.story.append(switches_table)

```

Day 3 Deliverables:

-  Executive summary with cluster overview table
-  Hardware inventory tables (CBoxes, DBoxes, Switches)
-  Professional table formatting with VAST branding
-  Summary statistics and derived metrics

Day 4: Professional Formatting and Branding

Task 2.5: Network Configuration Section

```

1  def _add_network_configuration(self):
2      """Create network configuration section."""
3
4      title = Paragraph("Network Configuration",
5      self.styles['heading1'])
6      self.story.append(title)
7
8      # Network services
9      self._add_network_services()
10     self.story.append(Spacer(1, 20))
11
12     # VIP pools
13     self._add_vip_pools()
14     self.story.append(PageBreak())
15
16     def _add_network_services(self):
17         """Add network services configuration."""
18
19         subtitle = Paragraph("Network Services", self.styles['heading2'])
20         self.story.append(subtitle)
21
22         network_config = self.cluster_data.get('network_configuration',
23         {})
24
25         # Network services table
26         services_data = [
27             ['Service', 'Configuration'],
28             ['DNS Servers', ', '.join(network_config.get('dns_servers',
29             ['Not configured']))],
30             ['NTP Servers', ', '.join(network_config.get('ntp_servers',
31             ['Not configured']))],
32             ['Management Network',
33             network_config.get('management_network', 'Not configured')],
34             ['Data VLAN', str(network_config.get('data_vlan', 'Not
35             configured'))]
36         ]
37
38         services_table = Table(services_data, colWidths=[2*inch, 4*inch])
39         services_table.setStyle(TableStyle([
40             ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
41             ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
42             ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
43             ('FOUNTSIZE', (0, 0), (-1, 0), 9),
44             ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
45             ('BACKGROUND', (0, 1), (-1, -1), colors.white),
46             ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
47             ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
48             ('FOUNTSIZE', (0, 1), (-1, -1), 8),
49             ('ALIGN', (0, 0), (0, -1), 'LEFT'),

```

```

44         ('ALIGN', (1, 0), (1, -1), 'LEFT'),
45         ('GRID', (0, 0), (-1, -1), 1, colors.black),
46         ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
47         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white,
colors.Color(0.95, 0.95, 0.95)])
48     ]))
49
50     self.story.append(services_table)
51
52     def _add_vip_pools(self):
53         """Add VIP pools configuration."""
54
55         subtitle = Paragraph("VIP Pools", self.styles['heading2'])
56         self.story.append(subtitle)
57
58         vip_pools = self.cluster_data.get('network_configuration',
{}).get('vip_pools', [])
59
60         if not vip_pools:
61             no_vips = Paragraph("No VIP pools configured",
self.styles['normal'])
62             self.story.append(no_vips)
63             return
64
65         # VIP pools table
66         vip_headers = ['Service', 'VIP Pool', 'IP Range', 'VLAN']
67         vip_data = [vip_headers]
68
69         for vip in vip_pools:
70             row = [
71                 vip.get('service', 'Unknown'),
72                 vip.get('pool_name', 'Unknown'),
73                 vip.get('ip_range', 'Unknown'),
74                 str(vip.get('vlan', 'Unknown'))
75             ]
76             vip_data.append(row)
77
78         vip_table = Table(vip_data, colWidths=[1.5*inch, 1.5*inch, 2*inch,
1*inch])
79         vip_table.setStyle(TableStyle([
80             ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
81             ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
82             ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
83             ('FONTSIZE', (0, 0), (-1, 0), 9),
84             ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
85             ('BACKGROUND', (0, 1), (-1, -1), colors.white),
86             ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
87             ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
88             ('FONTSIZE', (0, 1), (-1, -1), 8),
89             ('ALIGN', (0, 1), (-1, -1), 'CENTER'),
90             ('GRID', (0, 0), (-1, -1), 1, colors.black),
91             ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
92             ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white,
colors.Color(0.95, 0.95, 0.95)])
93         ]))
94
95         self.story.append(vip_table)

```

Task 2.6: Data Protection Section

```

1     def _add_data_protection(self):
2         """Create data protection section."""
3
4         title = Paragraph("Data Protection", self.styles['heading1'])
5         self.story.append(title)
6
7         data_protection = self.cluster_data.get('data_protection', {})
8
9         # Encryption status
10        encryption_subtitle = Paragraph("Encryption Configuration",
self.styles['heading2'])

```

```

11     self.story.append(encryption_subtitle)
12
13     encryption_data = [
14         ['Setting', 'Status'],
15         ['Local Encryption', 'Enabled' if
data_protection.get('encryption', {}).get('enabled', False) else
'Disabled'],
16         ['Encryption Type', data_protection.get('encryption',
{}).get('type', 'Not configured')],
17         ['Key Management', data_protection.get('encryption',
{}).get('key_management', 'Not configured')],
18         ['External Key Management', 'Enabled' if
data_protection.get('encryption', {}).get('ekm_enabled', False) else
'Disabled'],
19         ['EKM Provider', data_protection.get('encryption',
{}).get('ekm_provider', 'Not configured')]
20     ]
21
22     encryption_table = Table(encryption_data, colWidths=[3*inch,
3*inch])
23     encryption_table.setStyle(TableStyle([
24         ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
25         ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
26         ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
27         ('FONTSIZE', (0, 0), (-1, 0), 9),
28         ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
29         ('BACKGROUND', (0, 1), (-1, -1), colors.white),
30         ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
31         ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
32         ('FONTSIZE', (0, 1), (-1, -1), 8),
33         ('ALIGN', (0, 0), (0, -1), 'LEFT'),
34         ('ALIGN', (1, 0), (1, -1), 'LEFT'),
35         ('GRID', (0, 0), (-1, -1), 1, colors.black),
36         ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
37         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white,
colors.Color(0.95, 0.95, 0.95)]))
38     ]))
39
40     self.story.append(encryption_table)
41     self.story.append(PageBreak())
42
43     def _add_support_information(self):
44         """Create support information section."""
45
46         title = Paragraph("Support Information", self.styles['heading1'])
47         self.story.append(title)
48
49         cluster_overview = self.cluster_data.get('cluster_overview', {})
50
51         # Cluster identification
52         id_subtitle = Paragraph("Cluster Identification",
self.styles['heading2'])
53         self.story.append(id_subtitle)
54
55         support_data = [
56             ['Information', 'Value'],
57             ['Cluster PSNT', cluster_overview.get('cluster_psnt',
'Unknown')],
58             ['Cluster GUID', cluster_overview.get('cluster_guid',
'Unknown')],
59             ['VMS VIP', cluster_overview.get('vms_vip', 'Unknown')],
60             ['VAST OS Version', cluster_overview.get('cluster_version',
'Unknown')]
61         ]
62
63         support_table = Table(support_data, colWidths=[3*inch, 3*inch])
64         support_table.setStyle(TableStyle([
65             ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
66             ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
67             ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
68             ('FONTSIZE', (0, 0), (-1, 0), 9),





```

```

69         ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
70         ('BACKGROUND', (0, 1), (-1, -1), colors.white),
71         ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
72         ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
73         ('FONTSIZE', (0, 1), (-1, -1), 8),
74         ('ALIGN', (0, 0), (0, -1), 'LEFT'),
75         ('ALIGN', (1, 0), (1, -1), 'LEFT'),
76         ('GRID', (0, 0), (-1, -1), 1, colors.black),
77         ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
78         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white,
colors.Color(0.95, 0.95, 0.95)])
79     ]))
80
81     self.story.append(support_table)
82
83     # Contact information
84     self.story.append(Spacer(1, 20))
85     contact_subtitle = Paragraph("Emergency Contacts",
self.styles['heading2'])
86     self.story.append(contact_subtitle)
87
88     contact_text = """
89     <b>VAST Support:</b> +1-800-VAST-DATA<br/>
90     <b>Support Portal:</b> https://support.vastdata.com<br/>
91     <b>Professional Services:</b> ps@vastdata.com
92     """
93
94     contact_info = Paragraph(contact_text, self.styles['normal'])
95     self.story.append(contact_info)

```

Day 4 Deliverables:

-  Network configuration section with services and VIP pools
-  Data protection section with encryption status
-  Support information section with contact details
-  Consistent VAST branding and professional styling throughout

Day 5: Integration and Testing

Task 2.7: Report Template System

Implementation: `src/reports/report_templates.py`

```

1  from typing import Dict, List, Any
2  from reportlab.lib import colors
3
4  class ReportTemplates:
5      """Reusable templates for report sections."""
6
7      @staticmethod
8      def create_standard_table(data: List[List[str]], col_widths:
List[float]) -> Table:
9          """Create a standardized table with VAST branding."""
10
11         table = Table(data, colWidths=col_widths)
12         table.setStyle(TableStyle([
13             # Header styling
14             ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3,
0.6)),
15             ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
16             ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
17             ('FONTSIZE', (0, 0), (-1, 0), 9),
18             ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
19
20             # Data rows styling
21             ('BACKGROUND', (0, 1), (-1, -1), colors.white),

```

```

22         ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
23         ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
24         ('FONTSIZE', (0, 1), (-1, -1), 8),
25         ('ALIGN', (0, 1), (-1, -1), 'CENTER'),
26
27         # Grid and borders
28         ('GRID', (0, 0), (-1, -1), 1, colors.black),
29         ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
30
31         # Alternating row colors
32         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white,
colors.Color(0.95, 0.95, 0.95)])
33     ]))
34
35     return table
36
37 @staticmethod
38 def create_summary_box(title: str, content: str) -> List:
39     """Create a summary information box."""
40
41     title_style = ParagraphStyle(
42         'SummaryTitle',
43         fontSize=12,
44         textColor=colors.Color(0.0, 0.3, 0.6),
45         fontName='Helvetica-Bold',
46         spaceAfter=6
47     )
48
49     content_style = ParagraphStyle(
50         'SummaryContent',
51         fontSize=10,
52         fontName='Helvetica',
53         leftIndent=20
54     )
55
56     return [
57         Paragraph(title, title_style),
58         Paragraph(content, content_style),
59         Spacer(1, 12)
60     ]

```

Task 2.8: Integration Testing

Implementation: tests/test_pdf_generation.py

```

1 import unittest
2 import tempfile
3 import os
4 from src.reports.pdf_generator import VastPDFGenerator
5
6 class TestPDFGeneration(unittest.TestCase):
7     """Test PDF report generation functionality."""
8
9     def setUp(self):
10         """Set up test data."""
11         self.test_data = {
12             'cluster_overview': {
13                 'cluster_name': 'TEST-CLUSTER-01',
14                 'cluster_psnt': 'VST-2025-TEST-001',
15                 'cluster_guid': '12345678-abcd-1234-5678-
123456789abc',
16                 'cluster_version': '5.3.0-test',
17                 'vms_vip': '192.168.1.10',
18                 'usable_capacity': '1.17 PB',
19                 'licensed_capacity': '1.20 PB'
20             },
21             'hardware_inventory': {
22                 'cboxes': [
23                     {
24                         'model': 'VAST-CX4000',
25                         'serial': 'VST240901001',

```

```

26         'rack_position': 25,
27         'node_count': 4,
28         'management_ip': '192.168.1.11',
29         'cable_type': 'Splitter'
30     }
31 ],
32     'dboxes': [
33         {
34             'model': 'VAST-DX8000',
35             'serial': 'VST240901100',
36             'rack_position': 18,
37             'node_count': 4,
38             'management_ip': '192.168.1.21',
39             'cable_type': 'Straight'
40         }
41     ],
42     'switches': [
43         {
44             'model': 'Mellanox SN3700',
45             'serial': 'MT2113X12345',
46             'rack_position': 20,
47             'port_count': 32,
48             'port_speed': '100GbE',
49             'firmware_version': '3.10.1000',
50             'role': 'A'
51         }
52     ]
53 },
54     'network_configuration': {
55         'dns_servers': ['8.8.8.8', '8.8.4.4'],
56         'ntp_servers': ['pool.ntp.org'],
57         'management_network': '192.168.1.0/24',
58         'data_vlan': 100,
59         'vip_pools': [
60             {
61                 'service': 'NFS',
62                 'pool_name': 'nfs-pool',
63                 'ip_range': '10.100.1.10-10.100.1.17',
64                 'vlan': 100
65             }
66         ]
67     },
68     'data_protection': {
69         'encryption': {
70             'enabled': True,
71             'type': 'AES-256',
72             'key_management': 'Internal',
73             'ekm_enabled': False,
74             'ekm_provider': 'Not configured'
75         }
76     }
77 }
78
79 def test_pdf_generation(self):
80     """Test complete PDF generation."""
81
82     with tempfile.NamedTemporaryFile(suffix='.pdf', delete=False)
83 as tmp_file:
84         output_path = tmp_file.name
85
86         try:
87             # Generate PDF
88             generator = VastPDFGenerator(output_path, self.test_data)
89             result = generator.generate_report()
90
91             # Verify generation succeeded
92             self.assertTrue(result)
93             self.assertTrue(os.path.exists(output_path))
94             self.assertGreater(os.path.getsize(output_path), 1000) #

```

PDF should be > 1KB

```

95         finally:
96             # Clean up
97             if os.path.exists(output_path):
98                 os.unlink(output_path)
99
100     def test_title_page_generation(self):
101         """Test title page creation."""
102
103         with tempfile.NamedTemporaryFile(suffix='.pdf', delete=False)
as tmp_file:
104             output_path = tmp_file.name
105
106             try:
107                 generator = VastPDFGenerator(output_path, self.test_data)
108                 generator._add_title_page()
109
110                 # Verify title page elements were added
111                 self.assertGreater(len(generator.story), 0)
112
113             finally:
114                 if os.path.exists(output_path):
115                     os.unlink(output_path)
116
117     def test_hardware_tables_generation(self):
118         """Test hardware inventory table creation."""
119
120         with tempfile.NamedTemporaryFile(suffix='.pdf', delete=False)
as tmp_file:
121             output_path = tmp_file.name
122
123             try:
124                 generator = VastPDFGenerator(output_path, self.test_data)
125                 generator._add_hardware_inventory()
126
127                 # Verify hardware section was added
128                 self.assertGreater(len(generator.story), 0)
129
130             finally:
131                 if os.path.exists(output_path):
132                     os.unlink(output_path)
133
134 if __name__ == '__main__':
135     unittest.main()

```

Task 2.9: Error Handling and Validation

```

1  def generate_report(self) -> bool:
2      """
3      Generate the complete PDF report with comprehensive error
4      handling.
5
6      Returns:
7          bool: True if successful, False otherwise
8      """
9      try:
10         # Validate input data
11         if not self._validate_input_data():
12             print("Error: Invalid or incomplete input data")
13             return False
14
15         # Build report sections in order
16         self._add_title_page()
17         self._add_table_of_contents()
18         self._add_executive_summary()
19         self._add_hardware_inventory()
20         self._add_network_configuration()
21         self._add_data_protection()
22         self._add_support_information()
23
24         # Build the PDF

```

```

24         self.doc.build(self.story,
onFirstPage=self._add_header_footer,
25                             onLaterPages=self._add_header_footer)
26
27         print(f"PDF report generated successfully:
{self.output_path}")
28         return True
29
30     except Exception as e:
31         print(f"Error generating PDF report: {e}")
32         return False
33
34     def _validate_input_data(self) -> bool:
35         """Validate that required data is present."""
36
37         required_sections = ['cluster_overview', 'hardware_inventory']
38
39         for section in required_sections:
40             if section not in self.cluster_data:
41                 print(f"Missing required section: {section}")
42                 return False
43
44         # Validate cluster overview has minimum required fields
45         cluster_overview = self.cluster_data['cluster_overview']
46         required_fields = ['cluster_name', 'cluster_psnt']
47
48         for field in required_fields:
49             if field not in cluster_overview or not
cluster_overview[field]:
50                 print(f"Missing required field in cluster_overview:
{field}")
51                 return False
52
53         return True

```

Day 5 Deliverables:

- ☒ Complete PDF generation with error handling
- ☒ Comprehensive unit tests for all report sections
- ☒ Template system for reusable components
- ☒ Data validation and error reporting
- ☒ Integration testing with sample data



Week 2 Success Metrics

Functional Deliverables

1. **Complete PDF Generator:** Professional PDF reports with VAST branding
2. **Report Templates:** Modular, reusable section templates
3. **Professional Formatting:** Consistent styling, tables, and layout
4. **Data Integration:** Seamless integration with processed API data

Quality Standards

1. **Customer-Ready Output:** PDF reports suitable for customer delivery
2. **Professional Presentation:** VAST branding, consistent formatting
3. **Error Handling:** Graceful handling of missing or invalid data
4. **Test Coverage:** Comprehensive unit tests for all components

Technical Standards

1. **Modular Design:** Reusable components and templates
 2. **Performance:** PDF generation in <30 seconds
 3. **Reliability:** Consistent output across different data sets
 4. **Maintainability:** Clean, documented code following project standards
-

Dependencies and Requirements

Python Libraries

```
1 # requirements.txt additions for Week 2
2 reportlab>=3.6.0           # PDF generation
3 Pillow>=8.0.0             # Image handling for logos
```

Input Dependencies

- **Processed Data:** Structured data from Week 1 data processor
- **Configuration:** Report formatting configuration from config.yaml
- **Templates:** Report section templates and styling

Output Deliverables

- **PDF Reports:** Professional customer-deliverable PDF files
 - **Template System:** Reusable report generation components
 - **Test Suite:** Comprehensive testing for all report functionality
-

Integration Points

Week 1 Integration

- **Data Input:** Processed cluster data from data processor
- **Configuration:** Report settings from configuration system
- **Error Handling:** Consistent error handling patterns

Week 3 Integration

- **CLI Interface:** Integration with command-line interface
- **File Output:** Coordinated file naming and output directory management
- **User Feedback:** Progress reporting and error messaging

This detailed implementation plan for Week 2 provides the foundation for creating professional, customer-deliverable PDF reports that transform our robust API data collection into valuable business deliverables for VAST customers.