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`

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

```python
from reportlab.lib.units import inch
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle, PageBreak
from reportlab.lib import colors
from reportlab.lib.enums import TA_CENTER, TA_LEFT, TA_RIGHT
from reportlab.pdfgen import canvas
from datetime import datetime
import os
from typing import Dict, List, Any

class VastPDFGenerator:
    """Professional PDF report generator for VAST As-Built reports."""

    def __init__(self, output_path: str, cluster_data: Dict):
        """
        Initialize PDF generator with output path and cluster data.

        Args:
            output_path: Full path for the output PDF file
            cluster_data: Processed cluster data from data processor
        """
        self.output_path = output_path
        self.cluster_data = cluster_data
        self.doc = SimpleDocTemplate(
            output_path,
            pagesize=A4,
            rightMargin=72,
            leftMargin=72,
            topMargin=72,
            bottomMargin=72
        )
        self.styles = self._create_custom_styles()
        self.story = []

    def _create_custom_styles(self) -> Dict:
        """Create custom styles for VAST branding and professional formatting."""
        styles = getSampleStyleSheet()

        # VAST Brand Colors
        vast_blue = colors.Color(0.0, 0.3, 0.6)  # VAST primary blue
        vast_gray = colors.Color(0.4, 0.4, 0.4)  # VAST secondary gray

        custom_styles = {
            'title': ParagraphStyle(
                'VastTitle',
                parent=styles['Title'],
                fontSize=24,
                textColor=vast_blue,
                spaceAfter=30,
                alignment=TA_CENTER,
                fontName='Helvetica-Bold'
            ),
            'heading1': ParagraphStyle(
                'VastHeading1',
                parent=styles['Heading1'],
                fontSize=18,
                textColor=vast_blue,
                spaceAfter=12,
                spaceBefore=20,
                fontName='Helvetica-Bold'
            ),
            'heading2': ParagraphStyle(
                'VastHeading2',
                parent=styles['Heading2'],
                fontSize=14,
                textColor=vast_blue,
                spaceAfter=8,
                spaceBefore=16,
```

```python
                fontName='Helvetica-Bold'
            ),
            'normal': ParagraphStyle(
                'VastNormal',
                parent=styles['Normal'],
                fontSize=10,
                spaceAfter=6,
                fontName='Helvetica'
            ),
            'table_header': ParagraphStyle(
                'VastTableHeader',
                parent=styles['Normal'],
                fontSize=9,
                textColor=colors.white,
                fontName='Helvetica-Bold',
                alignment=TA_CENTER
            ),
            'table_cell': ParagraphStyle(
                'VastTableCell',
                parent=styles['Normal'],
                fontSize=9,
                fontName='Helvetica'
            )
        }

        return custom_styles

    def generate_report(self) -> bool:
        """
        Generate the complete PDF report.

        Returns:
            bool: True if successful, False otherwise
        """
        try:
            # Build report sections in order
            self._add_title_page()
            self._add_table_of_contents()
            self._add_executive_summary()
            self._add_hardware_inventory()
            self._add_network_configuration()
            self._add_data_protection()
            self._add_support_information()

            # Build the PDF
            self.doc.build(self.story, onFirstPage=self._add_header_footer,
                           onLaterPages=self._add_header_footer)

            return True

        except Exception as e:
            print(f"Error generating PDF report: {e}")
            return False

    def _add_header_footer(self, canvas, doc):
        """Add header and footer to each page."""
        canvas.saveState()

        # Header
        canvas.setFont('Helvetica-Bold', 10)
        canvas.setFillColor(colors.Color(0.0, 0.3, 0.6))
        canvas.drawString(72, doc.height + 50, "VAST Data As-Built Report")

        # Footer
        canvas.setFont('Helvetica', 8)
        canvas.setFillColor(colors.gray)
        canvas.drawString(72, 30, f"Generated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
```

```
        canvas.drawRightString(doc.width + 72, 30, f"Page {doc.page}")

        canvas.restoreState()
```

Task 2.2: Title Page Generator

```python
def _add_title_page(self):
    """Create professional title page with VAST branding."""

    # VAST Logo placeholder (would be actual logo in production)
    logo_placeholder = Paragraph(
        "<b>VAST DATA</b>",
        ParagraphStyle(
            'Logo',
            fontSize=36,
            textColor=colors.Color(0.0, 0.3, 0.6),
            alignment=TA_CENTER,
            fontName='Helvetica-Bold'
        )
    )

    # Report title
    title = Paragraph("As-Built Report", self.styles['title'])

    # Cluster information
    cluster_name = self.cluster_data.get('cluster_overview', {}).get('cluster_name', 'Unknown')
    cluster_psnt = self.cluster_data.get('cluster_overview', {}).get('cluster_psnt', 'Unknown')

    subtitle = Paragraph(
        f"<b>Cluster:</b> {cluster_name}<br/>"
        f"<b>PSNT:</b> {cluster_psnt}<br/>"
        f"<b>Generated:</b> {datetime.now().strftime('%B %d, %Y')}",
        ParagraphStyle(
            'Subtitle',
            fontSize=14,
            alignment=TA_CENTER,
            spaceAfter=20
        )
    )

    # Add elements to story
    self.story.extend([
        Spacer(1, 2*inch),
        logo_placeholder,
        Spacer(1, 1*inch),
        title,
        Spacer(1, 0.5*inch),
        subtitle,
        PageBreak()
    ])

def _add_table_of_contents(self):
    """Generate table of contents."""
    toc_title = Paragraph("Table of Contents", self.styles['heading1'])

    toc_items = [
        "1. Executive Summary",
        "2. Hardware Inventory",
        "3. Network Configuration",
        "4. Data Protection",
        "5. Support Information"
    ]

    self.story.append(toc_title)

    for item in toc_items:
        toc_item = Paragraph(item, self.styles['normal'])
```

```
        self.story.append(toc_item)

    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

```python
def _add_executive_summary(self):
    """Create executive summary section."""

    # Section title
    title = Paragraph("Executive Summary", self.styles['heading1'])
    self.story.append(title)

    # Cluster overview data
    cluster_overview = self.cluster_data.get('cluster_overview', {})

    # Overview paragraph
    overview_text = f"""
    This document provides comprehensive as-built documentation for the VAST Data cluster
    deployment completed for <b>{cluster_overview.get('customer_name', 'Customer')}</b>.
    The cluster has been successfully installed, configured, and validated according to
    VAST Data best practices and customer requirements.
    """

    overview = Paragraph(overview_text, self.styles['normal'])
    self.story.append(overview)
    self.story.append(Spacer(1, 12))

    # Cluster details table
    cluster_details = [
        ['Cluster Name', cluster_overview.get('cluster_name', 'Unknown')],
        ['Cluster PSNT', cluster_overview.get('cluster_psnt', 'Unknown')],
        ['VAST OS Version', cluster_overview.get('cluster_version', 'Unknown')],
        ['Cluster GUID', cluster_overview.get('cluster_guid', 'Unknown')],
        ['Total Usable Capacity', cluster_overview.get('usable_capacity', 'Unknown')],
        ['Licensed Capacity', cluster_overview.get('licensed_capacity', 'Unknown')],
        ['VMS VIP', cluster_overview.get('vms_vip', 'Unknown')]
    ]

    # Create table
    cluster_table = Table(cluster_details, colWidths=[2.5*inch, 3*inch])
    cluster_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (0, -1), colors.Color(0.9, 0.9, 0.9)),
        ('TEXTCOLOR', (0, 0), (-1, -1), colors.black),
        ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
        ('FONTNAME', (0, 0), (0, -1), 'Helvetica-Bold'),
```

```
        ('FONTNAME', (1, 0), (1, -1), 'Helvetica'),
        ('FONTSIZE', (0, 0), (-1, -1), 9),
        ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
    ]))

    self.story.append(cluster_table)
    self.story.append(PageBreak())
```

Task 2.4: Hardware Inventory Section

```
def _add_hardware_inventory(self):
    """Create hardware inventory section with tables."""

    title = Paragraph("Hardware Inventory", self.styles['heading1'])
    self.story.append(title)

    # CBoxes section
    self._add_cboxes_table()
    self.story.append(Spacer(1, 20))

    # DBoxes section
    self._add_dboxes_table()
    self.story.append(Spacer(1, 20))

    # Switches section
    self._add_switches_table()
    self.story.append(PageBreak())

def _add_cboxes_table(self):
    """Add CBoxes hardware table."""

    subtitle = Paragraph("CBoxes (Compute)", self.styles['heading2'])
    self.story.append(subtitle)

    # Get CBoxes data
    cboxes_data = self.cluster_data.get('hardware_inventory', {}).get('cboxes', [])

    # Table headers
    headers = ['Component', 'Model', 'Serial Number', 'Rack Position', 'CNodes', 'Management IP']
    table_data = [headers]

    # Add CBox data rows
    for i, cbox in enumerate(cboxes_data, 1):
        row = [
            f"CBox-{i}",
            cbox.get('model', 'Unknown'),
            cbox.get('serial', 'Unknown'),
            f"U{cbox.get('rack_position', 'Unknown')}",
            str(cbox.get('node_count', 'Unknown')),
            cbox.get('management_ip', 'Unknown')
        ]
        table_data.append(row)

    # Create table
    cboxes_table = Table(table_data, colWidths=[1*inch, 1*inch, 1.2*inch, 1*inch, 0.8*inch, 1.2*inch])
    cboxes_table.setStyle(TableStyle([
        # Header styling
        ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 9),
        ('ALIGN', (0, 0), (-1, 0), 'CENTER'),

        # Data rows styling
        ('BACKGROUND', (0, 1), (-1, -1), colors.white),
        ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
        ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
```

```python
        ('FONTSIZE', (0, 1), (-1, -1), 8),
        ('ALIGN', (0, 1), (-1, -1), 'CENTER'),

        # Grid and borders
        ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),

        # Alternating row colors
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.Color(0.95, 0.95, 0.95)])
    ]))

    self.story.append(cboxes_table)

    # Add summary information
    total_cnodes = sum(cbox.get('node_count', 0) for cbox in cboxes_data)
    cable_type = cboxes_data[0].get('cable_type', 'Unknown') if cboxes_data else 'Unknown'

    summary_text = f"""
    <b>Total CNodes:</b> {total_cnodes}<br/>
    <b>CNode Cable Type:</b> {cable_type}<br/>
    <b>Required Ports per Switch:</b> {total_cnodes * 2}
    """

    summary = Paragraph(summary_text, self.styles['normal'])
    self.story.append(summary)

def _add_dboxes_table(self):
    """Add DBoxes hardware table."""

    subtitle = Paragraph("DBoxes (Data)", self.styles['heading2'])
    self.story.append(subtitle)

    # Get DBoxes data
    dboxes_data = self.cluster_data.get('hardware_inventory', {}).get('dboxes', [])

    # Table headers
    headers = ['Component', 'Model', 'Serial Number', 'Rack Position', 'DNodes', 'Management IP']
    table_data = [headers]

    # Add DBox data rows
    for i, dbox in enumerate(dboxes_data, 1):
        row = [
            f"DBox-{100 + i}",  # DBox naming convention
            dbox.get('model', 'Unknown'),
            dbox.get('serial', 'Unknown'),
            f"U{dbox.get('rack_position', 'Unknown')}",
            str(dbox.get('node_count', 'Unknown')),
            dbox.get('management_ip', 'Unknown')
        ]
        table_data.append(row)

    # Create table with same styling as CBoxes
    dboxes_table = Table(table_data, colWidths=[1*inch, 1*inch, 1.2*inch, 1*inch, 0.8*inch, 1.2*inch])
    dboxes_table.setStyle(TableStyle([
        # Same styling as CBoxes table
        ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 9),
        ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
        ('BACKGROUND', (0, 1), (-1, -1), colors.white),
        ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
        ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
        ('FONTSIZE', (0, 1), (-1, -1), 8),
        ('ALIGN', (0, 1), (-1, -1), 'CENTER'),
```

```
        ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.Color(0.95, 0.95, 0.95)])
    ]))

    self.story.append(dboxes_table)

    # Add summary information
    total_dnodes = sum(dbox.get('node_count', 0) for dbox in dboxes_data)
    cable_type = dboxes_data[0].get('cable_type', 'Unknown') if dboxes_data else 'Unknown'

    summary_text = f"""
    <b>Total DNodes:</b> {total_dnodes}<br/>
    <b>DNode Cable Type:</b> {cable_type}<br/>
    <b>Required Ports per Switch:</b> {total_dnodes}
    """

    summary = Paragraph(summary_text, self.styles['normal'])
    self.story.append(summary)

def _add_switches_table(self):
    """Add network switches table."""

    subtitle = Paragraph("Network Switches", self.styles['heading2'])
    self.story.append(subtitle)

    # Get switches data
    switches_data = self.cluster_data.get('hardware_inventory', {}).get('switches', [])

    # Table headers
    headers = ['Component', 'Model', 'Serial Number', 'Rack Position', 'Ports', 'Firmware']
    table_data = [headers]

    # Add switch data rows
    for i, switch in enumerate(switches_data, 1):
        role = switch.get('role', 'A' if i % 2 == 1 else 'B')  # Alternate A/B
        row = [
            f"Switch {role}",
            switch.get('model', 'Unknown'),
            switch.get('serial', 'Unknown'),
            f"U{switch.get('rack_position', 'Unknown')}",
            f"{switch.get('port_count', 'Unknown')}x{switch.get('port_speed', 'Unknown')}",
            switch.get('firmware_version', 'Unknown')
        ]
        table_data.append(row)

    # Create table
    switches_table = Table(table_data, colWidths=[1*inch, 1.2*inch, 1.2*inch, 1*inch, 1*inch, 1*inch])
    switches_table.setStyle(TableStyle([
        # Same styling pattern as other tables
        ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 9),
        ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
        ('BACKGROUND', (0, 1), (-1, -1), colors.white),
        ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
        ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
        ('FONTSIZE', (0, 1), (-1, -1), 8),
        ('ALIGN', (0, 1), (-1, -1), 'CENTER'),
        ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
```

```
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.Color(0.95, 0.95, 0.95)])
    ]))

    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

```python
def _add_network_configuration(self):
    """Create network configuration section."""

    title = Paragraph("Network Configuration", self.styles['heading1'])
    self.story.append(title)

    # Network services
    self._add_network_services()
    self.story.append(Spacer(1, 20))

    # VIP pools
    self._add_vip_pools()
    self.story.append(PageBreak())

def _add_network_services(self):
    """Add network services configuration."""

    subtitle = Paragraph("Network Services", self.styles['heading2'])
    self.story.append(subtitle)

    network_config = self.cluster_data.get('network_configuration', {})

    # Network services table
    services_data = [
        ['Service', 'Configuration'],
        ['DNS Servers', ', '.join(network_config.get('dns_servers', ['Not configured']))],
        ['NTP Servers', ', '.join(network_config.get('ntp_servers', ['Not configured']))],
        ['Management Network', network_config.get('management_network', 'Not configured')],
        ['Data VLAN', str(network_config.get('data_vlan', 'Not configured'))]
    ]

    services_table = Table(services_data, colWidths=[2*inch, 4*inch])
    services_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 9),
        ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
        ('BACKGROUND', (0, 1), (-1, -1), colors.white),
        ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
        ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
        ('FONTSIZE', (0, 1), (-1, -1), 8),
        ('ALIGN', (0, 0), (0, -1), 'LEFT'),
```

```
        ('ALIGN', (1, 0), (1, -1), 'LEFT'),
        ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.Color(0.95, 0.95, 0.95)])
    ]))

    self.story.append(services_table)

def _add_vip_pools(self):
    """Add VIP pools configuration."""

    subtitle = Paragraph("VIP Pools", self.styles['heading2'])
    self.story.append(subtitle)

    vip_pools = self.cluster_data.get('network_configuration', {}).get('vip_pools', [])

    if not vip_pools:
        no_vips = Paragraph("No VIP pools configured", self.styles['normal'])
        self.story.append(no_vips)
        return

    # VIP pools table
    vip_headers = ['Service', 'VIP Pool', 'IP Range', 'VLAN']
    vip_data = [vip_headers]

    for vip in vip_pools:
        row = [
            vip.get('service', 'Unknown'),
            vip.get('pool_name', 'Unknown'),
            vip.get('ip_range', 'Unknown'),
            str(vip.get('vlan', 'Unknown'))
        ]
        vip_data.append(row)

    vip_table = Table(vip_data, colWidths=[1.5*inch, 1.5*inch, 2*inch, 1*inch])
    vip_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 9),
        ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
        ('BACKGROUND', (0, 1), (-1, -1), colors.white),
        ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
        ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
        ('FONTSIZE', (0, 1), (-1, -1), 8),
        ('ALIGN', (0, 1), (-1, -1), 'CENTER'),
        ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.Color(0.95, 0.95, 0.95)])
    ]))

    self.story.append(vip_table)
```

Task 2.6: Data Protection Section

```
def _add_data_protection(self):
    """Create data protection section."""

    title = Paragraph("Data Protection", self.styles['heading1'])
    self.story.append(title)

    data_protection = self.cluster_data.get('data_protection', {})

    # Encryption status
    encryption_subtitle = Paragraph("Encryption Configuration", self.styles['heading2'])
```

```
    self.story.append(encryption_subtitle)

    encryption_data = [
        ['Setting', 'Status'],
        ['Local Encryption', 'Enabled' if data_protection.get('encryption', {}).get('enabled', False) else 'Disabled'],
        ['Encryption Type', data_protection.get('encryption', {}).get('type', 'Not configured')],
        ['Key Management', data_protection.get('encryption', {}).get('key_management', 'Not configured')],
        ['External Key Management', 'Enabled' if data_protection.get('encryption', {}).get('ekm_enabled', False) else 'Disabled'],
        ['EKM Provider', data_protection.get('encryption', {}).get('ekm_provider', 'Not configured')]
    ]

    encryption_table = Table(encryption_data, colWidths=[3*inch, 3*inch])
    encryption_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 9),
        ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
        ('BACKGROUND', (0, 1), (-1, -1), colors.white),
        ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
        ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
        ('FONTSIZE', (0, 1), (-1, -1), 8),
        ('ALIGN', (0, 0), (0, -1), 'LEFT'),
        ('ALIGN', (1, 0), (1, -1), 'LEFT'),
        ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.Color(0.95, 0.95, 0.95)])
    ]))

    self.story.append(encryption_table)
    self.story.append(PageBreak())

def _add_support_information(self):
    """Create support information section."""

    title = Paragraph("Support Information", self.styles['heading1'])
    self.story.append(title)

    cluster_overview = self.cluster_data.get('cluster_overview', {})

    # Cluster identification
    id_subtitle = Paragraph("Cluster Identification", self.styles['heading2'])
    self.story.append(id_subtitle)

    support_data = [
        ['Information', 'Value'],
        ['Cluster PSNT', cluster_overview.get('cluster_psnt', 'Unknown')],
        ['Cluster GUID', cluster_overview.get('cluster_guid', 'Unknown')],
        ['VMS VIP', cluster_overview.get('vms_vip', 'Unknown')],
        ['VAST OS Version', cluster_overview.get('cluster_version', 'Unknown')]
    ]

    support_table = Table(support_data, colWidths=[3*inch, 3*inch])
    support_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 9),
```

```
        ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
        ('BACKGROUND', (0, 1), (-1, -1), colors.white),
        ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
        ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
        ('FONTSIZE', (0, 1), (-1, -1), 8),
        ('ALIGN', (0, 0), (0, -1), 'LEFT'),
        ('ALIGN', (1, 0), (1, -1), 'LEFT'),
        ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.Color(0.95, 0.95, 0.95)])
    ]))

    self.story.append(support_table)

    # Contact information
    self.story.append(Spacer(1, 20))
    contact_subtitle = Paragraph("Emergency Contacts", self.styles['heading2'])
    self.story.append(contact_subtitle)

    contact_text = """
    <b>VAST Support:</b> +1-800-VAST-DATA<br/>
    <b>Support Portal:</b> https://support.vastdata.com<br/>
    <b>Professional Services:</b> ps@vastdata.com
    """

    contact_info = Paragraph(contact_text, self.styles['normal'])
    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`

```
from typing import Dict, List, Any
from reportlab.lib import colors

class ReportTemplates:
    """Reusable templates for report sections."""

    @staticmethod
    def create_standard_table(data: List[List[str]], col_widths: List[float]) -> Table:
        """Create a standardized table with VAST branding."""

        table = Table(data, colWidths=col_widths)
        table.setStyle(TableStyle([
            # Header styling
            ('BACKGROUND', (0, 0), (-1, 0), colors.Color(0.0, 0.3, 0.6)),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, 0), 9),
            ('ALIGN', (0, 0), (-1, 0), 'CENTER'),

            # Data rows styling
            ('BACKGROUND', (0, 1), (-1, -1), colors.white),
```

```
            ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
            ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
            ('FONTSIZE', (0, 1), (-1, -1), 8),
            ('ALIGN', (0, 1), (-1, -1), 'CENTER'),

            # Grid and borders
            ('GRID', (0, 0), (-1, -1), 1, colors.black),
            ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),

            # Alternating row colors
            ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.Color(0.95, 0.95, 0.95)])
        ]))

        return table

    @staticmethod
    def create_summary_box(title: str, content: str) -> List:
        """Create a summary information box."""

        title_style = ParagraphStyle(
            'SummaryTitle',
            fontSize=12,
            textColor=colors.Color(0.0, 0.3, 0.6),
            fontName='Helvetica-Bold',
            spaceAfter=6
        )

        content_style = ParagraphStyle(
            'SummaryContent',
            fontSize=10,
            fontName='Helvetica',
            leftIndent=20
        )

        return [
            Paragraph(title, title_style),
            Paragraph(content, content_style),
            Spacer(1, 12)
        ]
```

Task 2.8: Integration Testing

Implementation: `tests/test_pdf_generation.py`

```
import unittest
import tempfile
import os
from src.reports.pdf_generator import VastPDFGenerator

class TestPDFGeneration(unittest.TestCase):
    """Test PDF report generation functionality."""

    def setUp(self):
        """Set up test data."""
        self.test_data = {
            'cluster_overview': {
                'cluster_name': 'TEST-CLUSTER-01',
                'cluster_psnt': 'VST-2025-TEST-001',
                'cluster_guid': '12345678-abcd-1234-5678-123456789abc',
                'cluster_version': '5.3.0-test',
                'vms_vip': '192.168.1.10',
                'usable_capacity': '1.17 PB',
                'licensed_capacity': '1.20 PB'
            },
            'hardware_inventory': {
                'cboxes': [
                    {
                        'model': 'VAST-CX4000',
                        'serial': 'VST240901001',
```

```
                        'rack_position': 25,
                        'node_count': 4,
                        'management_ip': '192.168.1.11',
                        'cable_type': 'Splitter'
                    }
                ],
                'dboxes': [
                    {
                        'model': 'VAST-DX8000',
                        'serial': 'VST240901100',
                        'rack_position': 18,
                        'node_count': 4,
                        'management_ip': '192.168.1.21',
                        'cable_type': 'Straight'
                    }
                ],
                'switches': [
                    {
                        'model': 'Mellanox SN3700',
                        'serial': 'MT2113X12345',
                        'rack_position': 20,
                        'port_count': 32,
                        'port_speed': '100GbE',
                        'firmware_version': '3.10.1000',
                        'role': 'A'
                    }
                ]
            },
            'network_configuration': {
                'dns_servers': ['8.8.8.8', '8.8.4.4'],
                'ntp_servers': ['pool.ntp.org'],
                'management_network': '192.168.1.0/24',
                'data_vlan': 100,
                'vip_pools': [
                    {
                        'service': 'NFS',
                        'pool_name': 'nfs-pool',
                        'ip_range': '10.100.1.10-10.100.1.17',
                        'vlan': 100
                    }
                ]
            },
            'data_protection': {
                'encryption': {
                    'enabled': True,
                    'type': 'AES-256',
                    'key_management': 'Internal',
                    'ekm_enabled': False,
                    'ekm_provider': 'Not configured'
                }
            }
        }

    def test_pdf_generation(self):
        """Test complete PDF generation."""

        with tempfile.NamedTemporaryFile(suffix='.pdf', delete=False) as tmp_file:
            output_path = tmp_file.name

        try:
            # Generate PDF
            generator = VastPDFGenerator(output_path, self.test_data)
            result = generator.generate_report()

            # Verify generation succeeded
            self.assertTrue(result)
            self.assertTrue(os.path.exists(output_path))
            self.assertGreater(os.path.getsize(output_path), 1000)  # PDF should be > 1KB

```

```python
        finally:
            # Clean up
            if os.path.exists(output_path):
                os.unlink(output_path)

    def test_title_page_generation(self):
        """Test title page creation."""

        with tempfile.NamedTemporaryFile(suffix='.pdf', delete=False) as tmp_file:
            output_path = tmp_file.name

        try:
            generator = VastPDFGenerator(output_path, self.test_data)
            generator._add_title_page()

            # Verify title page elements were added
            self.assertGreater(len(generator.story), 0)

        finally:
            if os.path.exists(output_path):
                os.unlink(output_path)

    def test_hardware_tables_generation(self):
        """Test hardware inventory table creation."""

        with tempfile.NamedTemporaryFile(suffix='.pdf', delete=False) as tmp_file:
            output_path = tmp_file.name

        try:
            generator = VastPDFGenerator(output_path, self.test_data)
            generator._add_hardware_inventory()

            # Verify hardware section was added
            self.assertGreater(len(generator.story), 0)

        finally:
            if os.path.exists(output_path):
                os.unlink(output_path)

if __name__ == '__main__':
    unittest.main()
```

Task 2.9: Error Handling and Validation

```python
def generate_report(self) -> bool:
    """
    Generate the complete PDF report with comprehensive error handling.

    Returns:
        bool: True if successful, False otherwise
    """
    try:
        # Validate input data
        if not self._validate_input_data():
            print("Error: Invalid or incomplete input data")
            return False

        # Build report sections in order
        self._add_title_page()
        self._add_table_of_contents()
        self._add_executive_summary()
        self._add_hardware_inventory()
        self._add_network_configuration()
        self._add_data_protection()
        self._add_support_information()

        # Build the PDF
```

```python
        self.doc.build(self.story, onFirstPage=self._add_header_footer,
                       onLaterPages=self._add_header_footer)

        print(f"PDF report generated successfully: {self.output_path}")
        return True

    except Exception as e:
        print(f"Error generating PDF report: {e}")
        return False

def _validate_input_data(self) -> bool:
    """Validate that required data is present."""

    required_sections = ['cluster_overview', 'hardware_inventory']

    for section in required_sections:
        if section not in self.cluster_data:
            print(f"Missing required section: {section}")
            return False

    # Validate cluster overview has minimum required fields
    cluster_overview = self.cluster_data['cluster_overview']
    required_fields = ['cluster_name', 'cluster_psnt']

    for field in required_fields:
        if field not in cluster_overview or not cluster_overview[field]:
            print(f"Missing required field in cluster_overview: {field}")
            return False

    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

```
# requirements.txt additions for Week 2
reportlab>=3.6.0          # PDF generation
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.