Team Member 4: Report Generation & Export Specialist

Your Role

You create the reporting system that generates summaries and exports data to PDF and text files.

Day 1 Tasks (6-8 hours)

Morning Session (3-4 hours)

1. Learn About Report Generation

Concepts to understand: - What makes a good report (clear, organized, informative) - PDF generation vs text file generation - Data formatting for reports - Summary statistics

Quick Tutorial:

```
# Creating a simple text report
def create_text_report(data):
    report = []
    report.append("=" * 50)
    report.append("SYSTEM REPORT")
    report.append("=" * 50)
    report.append(f"Total Records: {data['total']}")
    report.append(f"Generated: {data['date']}")

    # Save to file
    with open("report.txt", "w") as f:
        f.write("\n".join(report))

    print("Report saved!")

# Test it
test_data = {'total': 10, 'date': '2024-01-15'}
create_text_report(test_data)
```

2. Install Required Library

For PDF generation, install `reportlab`:

```
pip install reportlab
```

Test the installation:

```
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas

pdf = canvas.Canvas("test.pdf", pagesize=letter)
pdf.drawString(100, 750, "Hello PDF!")
pdf.save()
print("PDF created successfully!")
```

Afternoon Session (3-4 hours)

3. Implement Your Code

File: report_generator.py

```
"""
Report Generation Module
Handles report creation and export functionality
Author: [Your Name] - Team Member 4
"""

import os
from datetime import datetime
from typing import List, Dict, Tuple, Any
from reportlab.lib.pagesizes import letter, A4
from reportlab.lib import colors
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer,
from reportlab.platypus import Image as RLImage


class ReportGenerator:
    """Generates reports in PDF and text formats"""

    def __init__(self, user_id: int, username: str):
        """
        Initialize report generator
        user_id: ID of user generating report
        username: Name of user
        """
        self.user_id = user_id
        self.username = username

        # Create reports directory if it doesn't exist
        self.reports_dir = "reports"
        if not os.path.exists(self.reports_dir):
            os.makedirs(self.reports_dir)

    # ============= DATA COLLECTION =============

    def collect_report_data(self) -> Dict[str, Any]:
        """Collect all data needed for the report"""
        try:
            from database_manager import DatabaseManager
            db = DatabaseManager()

            # Get all records
            records = db.read_all_records(self.user_id)

            # Get statistics
            stats = db.get_summary_stats(self.user_id)

            # Calculate additional metrics
            if records:
                recent_records = [r for r in records if self._is_recent(r[4])]
                active_records = [r for r in records if r[5] == "Active"]
            else:
                recent_records = []
                active_records = []

            return {
                'records': records,
                'stats': stats,
                'recent_count': len(recent_records),
                'active_count': len(active_records),
                'generation_time': datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
                'username': self.username
            }

        except ImportError:
            # Return demo data if database not available
            return {
                'records': [
                    (1, "Sample Record 1", "Description 1", "General", "2024-01-15", "A
                    (2, "Sample Record 2", "Description 2", "Important", "2024-01-16",
                ],
                'stats': {
                    'total': 2,
                    'active': 2,
                    'inactive': 0,
                    'by_category': {'General': 1, 'Important': 1}
            },
            'recent_count': 2,
            'active_count': 2,
            'generation_time': datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            'username': self.username
        }

    def _is_recent(self, date_string: str, days: int = 7) -> bool:
        """Check if date is within the last N days"""
        try:
            record_date = datetime.fromisoformat(date_string.replace('Z', '+00:00'))
            days_diff = (datetime.now() - record_date).days
            return days_diff <= days
        except:
            return False

    # ============= TEXT REPORT GENERATION =============

    def generate_text_report(self) -> str:
        """
        Generate a text-based report
        Returns: filepath of generated report
        """
        data = self.collect_report_data()

        # Build report content
        lines = []
        lines.append("=" * 70)
        lines.append("SMART RECORDS SYSTEM - SUMMARY REPORT")
        lines.append("=" * 70)
        lines.append("")
        lines.append(f"Generated by: {data['username']}")
        lines.append(f"Generated at: {data['generation_time']}")
        lines.append("")

        # Statistics Section
        lines.append("-" * 70)
        lines.append("STATISTICS SUMMARY")
        lines.append("-" * 70)
        stats = data['stats']
        lines.append(f"Total Records:        {stats['total']}")
        lines.append(f"Active Records:       {stats['active']}")
        lines.append(f"Inactive Records:     {stats['inactive']}")
        lines.append(f"Recent (7 days):      {data['recent_count']}")
        lines.append("")

        # Category Breakdown
        lines.append("-" * 70)
        lines.append("RECORDS BY CATEGORY")
        lines.append("-" * 70)
        for category, count in stats['by_category'].items():
            percentage = (count / stats['total'] * 100) if stats['total'] > 0 else 0
            lines.append(f"{category:<20} {count:>5} records ({percentage:>5.1f}%)")
        lines.append("")

        # Recent Records
        lines.append("-" * 70)
        lines.append("DETAILED RECORD LIST")
        lines.append("-" * 70)
        lines.append(f"{'ID':<5} {'Title':<25} {'Category':<15} {'Status':<10} {'Date'}
        lines.append("-" * 70)

        for record in data['records'][:20]:  # Show first 20 records
            record_id = str(record[0])
            title = record[1][:25]  # Truncate long titles
            category = record[3][:15]
            status = record[5][:10]
            date = record[4][:10]  # Just the date part

            lines.append(f"{record_id:<5} {title:<25} {category:<15} {status:<10} {date

        if len(data['records']) > 20:
            lines.append(f"\n... and {len(data['records']) - 20} more records")

        lines.append("")
        lines.append("=" * 70)
        lines.append("END OF REPORT")
        lines.append("=" * 70)

        # Save to file
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"report_{self.username}_{timestamp}.txt"
        filepath = os.path.join(self.reports_dir, filename)

        with open(filepath, 'w', encoding='utf-8') as f:
            f.write('\n'.join(lines))

        return filepath

    # ============= PDF REPORT GENERATION =============

    def generate_pdf_report(self) -> str:
        """
        Generate a PDF report with professional formatting
        Returns: filepath of generated report
        """
        data = self.collect_report_data()

        # Create filename
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"report_{self.username}_{timestamp}.pdf"
        filepath = os.path.join(self.reports_dir, filename)

        # Create PDF document
        doc = SimpleDocTemplate(
            filepath,
            pagesize=letter,
            rightMargin=72,
            leftMargin=72,
            topMargin=72,
            bottomMargin=72
        )

        # Container for PDF elements
        elements = []

        # Styles
        styles = getSampleStyleSheet()
        title_style = ParagraphStyle(
            'CustomTitle',
            parent=styles['Heading1'],
            fontSize=24,
            textColor=colors.HexColor('#2E7D32'),
            spaceAfter=30,
            alignment=1  # Center
        )

        heading_style = ParagraphStyle(
            'CustomHeading',
            parent=styles['Heading2'],
            fontSize=14,
            textColor=colors.HexColor('#1976D2'),
            spaceAfter=12,
            spaceBefore=12
        )

        # Title
        elements.append(Paragraph("Smart Records System", title_style))
        elements.append(Paragraph("Summary Report", title_style))
        elements.append(Spacer(1, 0.2 * inch))

        # Metadata
        meta_data = [
            ["Generated by:", data['username']],
            ["Generated at:", data['generation_time']],
            ["Report Type:", "Full System Summary"]
        ]

        meta_table = Table(meta_data, colWidths=[2*inch, 4*inch])
        meta_table.setStyle(TableStyle([
            ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
            ('FONTNAME', (0, 0), (0, -1), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, -1), 10),
            ('BOTTOMPADDING', (0, 0), (-1, -1), 6),
        ]))
        elements.append(meta_table)
        elements.append(Spacer(1, 0.3 * inch))

        # Statistics Section
        elements.append(Paragraph("Statistics Summary", heading_style))

        stats = data['stats']
        stats_data = [
            ['Metric', 'Count'],
            ['Total Records', str(stats['total'])],
            ['Active Records', str(stats['active'])],
            ['Inactive Records', str(stats['inactive'])],
            ['Recent Records (7 days)', str(data['recent_count'])],
        ]

        stats_table = Table(stats_data, colWidths=[3*inch, 2*inch])
        stats_table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
            ('ALIGN', (1, 1), (1, -1), 'CENTER'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, 0), 12),
            ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
            ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
            ('GRID', (0, 0), (-1, -1), 1, colors.black),
        ]))
        elements.append(stats_table)
        elements.append(Spacer(1, 0.3 * inch))

        # Category Breakdown
        elements.append(Paragraph("Records by Category", heading_style))

        category_data = [['Category', 'Count', 'Percentage']]
        for category, count in stats['by_category'].items():
            percentage = (count / stats['total'] * 100) if stats['total'] > 0 else 0
            category_data.append([category, str(count), f"{percentage:.1f}%"])

        category_table = Table(category_data, colWidths=[2.5*inch, 1.5*inch, 1.5*inch])
        category_table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#1976D2')),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
            ('ALIGN', (1, 0), (-1, -1), 'CENTER'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, 0), 11),
            ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
            ('BACKGROUND', (0, 1), (-1, -1), colors.lightblue),
            ('GRID', (0, 0), (-1, -1), 1, colors.black),
            ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.lightgrey]),
        ]))
        elements.append(category_table)
        elements.append(Spacer(1, 0.3 * inch))
```

```
    # Records Detail
    elements.append(Paragraph("Detailed Record List", heading_style))

    records_data = [['ID', 'Title', 'Category', 'Status', 'Date']]
    for record in data['records'][:15]:  # First 15 records
        records_data.append([
            str(record[0]),
            record[1][:30],  # Truncate long titles
            record[3],
            record[5],
            record[4][:10]
        ])

    records_table = Table(records_data, colWidths=[0.5*inch, 2.5*inch, 1.2*inch, 1*
    records_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#2E7D32')),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
        ('ALIGN', (0, 0), (0, -1), 'CENTER'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 10),
        ('FONTSIZE', (0, 1), (-1, -1), 9),
        ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
        ('GRID', (0, 0), (-1, -1), 0.5, colors.grey),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.lightgrey]),
    ]))
    elements.append(records_table)

    if len(data['records']) > 15:
        elements.append(Spacer(1, 0.2 * inch))
        elements.append(Paragraph(
            f"<i>Showing 15 of {len(data['records'])} total records</i>",
            styles['Normal']
        ))

    # Build PDF
    doc.build(elements)

    return filepath

# ============= QUICK SUMMARY =============

def generate_quick_summary(self) -> Dict[str, Any]:
    """
    Generate quick summary data (for display in GUI)
    Returns: Dictionary with summary data
    """
    data = self.collect_report_data()

    return {
        'total_records': data['stats']['total'],
            'active_records': data['stats']['active'],
            'inactive_records': data['stats']['inactive'],
            'recent_records': data['recent_count'],
            'categories': data['stats']['by_category'],
            'generation_time': data['generation_time']
        }

    # ============= MAIN GENERATION METHOD =============

    def generate_full_report(self) -> Tuple[str, str]:
        """
        Generate both PDF and text reports
        Returns: (pdf_filepath, text_filepath)
        """
        print("Generating reports...")

        try:
            pdf_path = self.generate_pdf_report()
            print(f"✓ PDF report generated: {pdf_path}")
        except Exception as e:
            print(f"✗ PDF generation failed: {e}")
            pdf_path = None

        try:
            text_path = self.generate_text_report()
            print(f"✓ Text report generated: {text_path}")
        except Exception as e:
            print(f"✗ Text generation failed: {e}")
            text_path = None

        return pdf_path, text_path


# ============= GUI INTEGRATION =============

def create_report_window(parent, user_id: int, username: str):
    """
    Create a report generation window (for integration with GUI)
    Call this from the main dashboard
    """
    import tkinter as tk
    from tkinter import messagebox
    import subprocess
    import platform

    window = tk.Toplevel(parent)
    window.title("Generate Report")
    window.geometry("400x300")
    window.resizable(False, False)
    window.grab_set()

    # Title
    tk.Label(
        window,
        text="Report Generation",
        font=("Arial", 16, "bold")
    ).pack(pady=20)

    # Info
    info_text = f"Generate a comprehensive report of your records.\n\nUser: {username}"
    tk.Label(
        window,
        text=info_text,
        font=("Arial", 10),
        justify='center'
    ).pack(pady=10)

    # Progress label
    progress_label = tk.Label(window, text="", font=("Arial", 9), fg="blue")
    progress_label.pack(pady=10)

    def generate_and_open():
        """Generate report and offer to open it"""
        progress_label.config(text="Generating reports...")
        window.update()

        try:
            report_gen = ReportGenerator(user_id, username)
            pdf_path, text_path = report_gen.generate_full_report()

            progress_label.config(text="✓ Reports generated successfully!", fg="green")

            # Ask if user wants to open the report
            if pdf_path:
                response = messagebox.askyesno(
                    "Success",
                    f"PDF Report generated!\n\nLocation: {pdf_path}\n\nWould you like t
                )

                if response:
                    # Open file with default application
                    if platform.system() == 'Windows':
                        os.startfile(pdf_path)
                    elif platform.system() == 'Darwin':  # macOS
                        subprocess.call(['open', pdf_path])
                    else:  # Linux
                        subprocess.call(['xdg-open', pdf_path])

            window.destroy()

        except Exception as e:
            progress_label.config(text="✗ Report generation failed", fg="red")
            messagebox.showerror("Error", f"Failed to generate report:\n{str(e)}")
    # Buttons
    btn_frame = tk.Frame(window)
    btn_frame.pack(pady=20)

    tk.Button(
        btn_frame,
        text="Generate PDF Report",
        font=("Arial", 11, "bold"),
        bg="#4CAF50",
        fg="white",
        width=20,
        cursor="hand2",
        command=generate_and_open
    ).pack(side='left', padx=5)

    tk.Button(
        btn_frame,
        text="Cancel",
        font=("Arial", 11),
        width=15,
        cursor="hand2",
        command=window.destroy
    ).pack(side='left', padx=5)


# ============= TESTING CODE =============
if __name__ == "__main__":
    print("=== Report Generator Test ===\n")

    # Create test report generator
    report_gen = ReportGenerator(user_id=1, username="TestUser")

    # Test 1: Collect data
    print("Test 1: Collecting report data...")
    data = report_gen.collect_report_data()
    print(f"  ✓ Collected data for {data['stats']['total']} records")

    # Test 2: Generate text report
    print("\nTest 2: Generating text report...")
    try:
        text_path = report_gen.generate_text_report()
        print(f"  ✓ Text report saved to: {text_path}")
    except Exception as e:
        print(f"  ✗ Failed: {e}")

    # Test 3: Generate PDF report
    print("\nTest 3: Generating PDF report...")
    try:
        pdf_path = report_gen.generate_pdf_report()
        print(f"  ✓ PDF report saved to: {pdf_path}")
    except Exception as e:
        print(f"  ✗ Failed: {e}")
    # Test 4: Quick summary
    print("\nTest 4: Generating quick summary...")
    summary = report_gen.generate_quick_summary()
    print(f"  Total: {summary['total_records']}")
    print(f"  Active: {summary['active_records']}")
    print(f"  Categories: {summary['categories']}")

    print("\n✓ All tests completed!")
    print(f"\nCheck the '{report_gen.reports_dir}' folder for generated reports.")
```

Day 2 Tasks (4-6 hours)

Morning Session (2-3 hours)

4. Test Report Generation

Run your code:

```
python report_generator.py
```

Check: - [ ] Text file is created in `reports/` folder - [ ] PDF file is created and opens correctly - [ ] All data appears in reports - [ ] Formatting looks professional

5. Integrate with GUI

Add your report function to Team Member 2's dashboard.

Give them this code to add:

```
def generate_report(self):
    """Trigger report generation"""
    try:
        from report_generator import create_report_window
        create_report_window(self.window, self.user_id, self.username)
    except Exception as e:
        messagebox.showerror("Error", f"Report generation failed: {str(e)}")
```

Afternoon Session (2-3 hours)

6. Enhance Reports

Add these improvements: - More statistics (average records per day, most used category) - Better formatting - Charts/graphs (optional, advanced)

Example enhancement:

```python
def calculate_trends(self, records):
    """Calculate trends over time"""
    from collections import defaultdict

    # Records by month
    by_month = defaultdict(int)
    for record in records:
        month = record[4][:7]  # YYYY-MM
        by_month[month] += 1

    return dict(by_month)
```

7. Create User Documentation

File: REPORT_GUIDE.md

```markdown
# Report Generation Guide

## How to Generate Reports

### From the Application
1. Click the "Generate Report" button on the main dashboard
2. Choose report type (PDF is recommended)
3. Wait for generation to complete
4. Report will open automatically

### Report Locations
All reports are saved in the `reports/` folder with filenames:
- PDF: `report_[username]_[timestamp].pdf`
- Text: `report_[username]_[timestamp].txt`

## Report Contents

### Statistics Summary
- Total number of records
- Active vs inactive counts
- Recent activity (last 7 days)

### Category Breakdown
- Number of records per category
- Percentage distribution
- Visual breakdown

### Detailed Record List
- Complete list of all records
- ID, Title, Category, Status, Date
- Sorted by most recent first
```

```
## Troubleshooting

**Problem:** PDF won't generate
**Solution:** Make sure reportlab is installed: `pip install reportlab`

**Problem:** Reports folder doesn't exist
**Solution:** Run the program once, it will create the folder automatically

**Problem:** Can't open PDF
**Solution:** Install a PDF reader like Adobe Acrobat or use your browser
```

Checklist

Day 1

- ☐ Learn about reports and PDF generation
- ☐ Install reportlab library
- ☐ Design report layout
- ☐ Write report_generator.py
- ☐ Implement text report generation
- ☐ Implement PDF report generation
- ☐ Test basic functionality

Day 2

- ☐ Test all report formats
- ☐ Verify data accuracy
- ☐ Integrate with GUI
- ☐ Add report enhancements
- ☐ Create user documentation
- ☐ Prepare demo for presentation

Things to Search and Learn

1. **ReportLab Tutorial**: "Python reportlab PDF generation tutorial"
2. **File I/O**: "Python file writing reading tutorial"
3. **PDF Tables**: "reportlab table styling examples"
4. **Data Formatting**: "Python string formatting for reports"
5. **Date Handling**: "Python datetime formatting"

Common Issues & Solutions

Issue: ReportLab not found **Solution**: `pip install reportlab`

Issue: PDF looks ugly/unprofessional **Solution**: Use TableStyle with colors and proper spacing

Issue: Reports take too long to generate **Solution**: Only include necessary data, limit records shown

Issue: Can't find generated reports **Solution**: Check the `reports/` folder in your project directory

Your Contribution (for presentation)

"I created the reporting system that generates professional PDF and text reports. The reports include statistics, category breakdowns, and detailed record listings. Users can generate and export reports with one click."

Mark Breakdown for Your Work

- Report Generation: 2 marks
- Code Quality & Documentation: 0.5 mark (shared)
- Total: 2.5 marks (17% of project)

Integration with Team

Provide to Member 2 (GUI):

```
# Add this import at the top of their file
from report_generator import create_report_window

# Add this to their "Generate Report" button command
def generate_report(self):
    create_report_window(self.window, self.user_id, self.username)
```

Use from Member 1 (Database):

```
from database_manager import DatabaseManager
db = DatabaseManager()
records = db.read_all_records(user_id)
stats = db.get_summary_stats(user_id)
```

Sample Report Output

Text Report Preview:

```
======================================================================
SMART RECORDS SYSTEM - SUMMARY REPORT
======================================================================

Generated by: TestUser
Generated at: 2024-01-15 14:30:00

----------------------------------------------------------------------
STATISTICS SUMMARY
----------------------------------------------------------------------
Total Records:        25
Active Records:       20
Inactive Records:     5
Recent (7 days):      8

----------------------------------------------------------------------
RECORDS BY CATEGORY
----------------------------------------------------------------------
General              10 records (40.0%)
Important            8 records (32.0%)
Work                 5 records (20.0%)
Personal             2 records (8.0%)

[... detailed record list ...]
```

Advanced Features (Optional)

If you finish early, add these:

1. **Export to Excel**: Use `openpyxl` library
2. **Email Reports**: Use `smtplib` to email PDFs
3. **Scheduled Reports**: Generate reports automatically
4. **Custom Filters**: Let users choose date ranges
5. **Charts**: Add pie charts using `matplotlib`

Example pie chart code:

```
import matplotlib.pyplot as plt

def create_category_chart(categories):
    plt.figure(figsize=(8, 6))
    plt.pie(categories.values(), labels=categories.keys(), autopct='%1.1f%%')
    plt.title('Records by Category')
    plt.savefig('chart.png')
    plt.close()
```