Pi Share Receiver Implementation Plan

After analyzing your project files, I'll provide a comprehensive implementation plan that builds on the existing structure while addressing key issues to get the Pi Share Receiver fully functional.

Current State Assessment

Your project has multiple scraping implementations and two server versions:

1. `server.py.bak` - A complete but monolithic implementation
2. `server.py` - A more modular design that references service modules that don't exist yet

The current issues include:

- Missing service modules referenced in the current `server.py`
- Inconsistent configuration paths between files
- No proper error handling or logging in the core components
- Lack of a complete end-to-end workflow validation
- Font customization requirements not implemented

Implementation Plan

Phase 1: Create Service Modules Structure

First, let's establish the service modules needed by `server.py`:

```
# Create services directory
mkdir -p app/services
```

Now, let's implement each service module with your style preferences:

1. **QR Code Service (`app/services/qr_service.py`)**

```
"""QR code generation service for Pi Share Receiver."""

import os
import qrcode
from typing import Tuple

class QRCodeService:
    """Generates QR codes for URLs."""

    def __init__(self, output_path: str):
        """Initialize with output path for QR codes.

        Args:
            output_path: Directory to save QR codes
        """
        self.output_path = output_path
        os.makedirs(output_path, exist_ok=True)

    def generate_qr(self, url: str) -> Tuple[str, str]:
        """Generate QR code for URL and return filepath and filename.

        Args:
            url: The URL to encode in the QR code

        Returns:
            Tuple of (filepath, filename)
        """
        qr = qrcode.QRCode(
            version=1,
            error_correction=qrcode.constants.ERROR_CORRECT_L,
            box_size=10,
            border=4,
        )
        qr.add_data(url)
        qr.make(fit=True)

        img = qr.make_image(fill_color="black", back_color="white")

        # Save QR Code image
        filename = f"qr_{hash(url)}.png"
        filepath = os.path.join(self.output_path, filename)
        img.save(filepath)

        return filepath, filename
```

1. **Web Scraper Service (`app/services/web_scraper_service.py`)**

```
"""Web scraping service that tries multiple methods."""

import os
import json
import subprocess
import logging
from typing import Dict, Any, List

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

class WebScraperService:
    """Scrapes web content using multiple fallback methods."""

    def __init__(self, temp_dir: str):
        """Initialize with temp directory for content files.

        Args:
            temp_dir: Directory to save temporary content
        """
        self.temp_dir = temp_dir
        os.makedirs(temp_dir, exist_ok=True)

        # Script paths - relative to current file location
        current_dir = os.path.dirname(os.path.abspath(__file__))
        app_dir = os.path.dirname(current_dir)
        self.scraper_scripts = {
            "playwright": os.path.join(app_dir, "scrape_js.py"),
            "simple": os.path.join(app_dir, "scrape_simple.py"),
            "browser": os.path.join(app_dir, "scrape_with_browser.py"),
            "requests_html": os.path.join(app_dir,
"scrape_with_requests_html.py")
        }

    def scrape(self, url: str) -> Dict[str, Any]:
        """Scrape content from URL using multiple methods.

        Args:
            url: The URL to scrape

        Returns:
            Dict containing title, structured_content, and images
        """
        logger.info(f"Scraping URL: {url}")

        # Create temporary output path for content
```

```
        content_path = os.path.join(self.temp_dir,
f"content_{hash(url)}.json")

        # Try different scrapers in order until one succeeds
        scrapers = [
            ("playwright", "Using Playwright for JavaScript-enabled
scraping"),
            ("simple", "Using simple requests + BeautifulSoup scraping"),
            ("browser", "Using Selenium browser-based scraping"),
            ("requests_html", "Using requests-html scraping")
        ]

        for scraper_name, message in scrapers:
            script_path = self.scraper_scripts.get(scraper_name)
            if not script_path or not os.path.exists(script_path):
                logger.warning(f"Scraper script not found: {scraper_name}")
                continue

            logger.info(message)

            try:
                # Run the scraper script
                result = subprocess.run(
                    ["python", script_path, url, content_path],
                    capture_output=True,
                    text=True,
                    check=False  # Don't raise an exception on non-zero
exit
                )

                # Check if content was generated successfully
                if result.returncode == 0 and os.path.exists(content_path):
                    with open(content_path, 'r', encoding='utf-8') as f:
                        content = json.load(f)

                    logger.info(f"Successfully scraped with
{scraper_name}")
                    return content
                else:
                    logger.warning(f"Failed to scrape with {scraper_name}:
{result.stderr}")
            except Exception as e:
                logger.warning(f"Error using {scraper_name} scraper:
{str(e)}")

        # If all scrapers fail, return a basic error content
        logger.error("All scrapers failed to extract content")
```

```
        return {
            "title": f"Failed to Extract: {url}",
            "structured_content": [{
                "type": "paragraph",
                "content": f"Could not extract content from {url}. All
scraping methods failed."
            }],
            "images": []
        }
```

1. **PDF Service (`app/services/pdf_service.py`)**

```
"""PDF processing service for Pi Share Receiver."""

import os
import requests
import PyPDF2
from urllib.parse import urlparse
from typing import Dict, Optional, Any
import logging

# Configure logging
logger = logging.getLogger(__name__)

class PDFService:
    """Handles PDF processing operations."""

    def __init__(self, temp_dir: str, extract_dir: str):
        """Initialize with directories for temporary and extracted files.

        Args:
            temp_dir: Directory for temporary PDF storage
            extract_dir: Directory for PDF content extraction
        """
        self.temp_dir = temp_dir
        self.extract_dir = extract_dir
        os.makedirs(temp_dir, exist_ok=True)
        os.makedirs(extract_dir, exist_ok=True)

    def is_pdf_url(self, url: str) -> bool:
        """Check if URL points to a PDF file.

        Args:
            url: URL to check

        Returns:
```

```
            True if URL is PDF, False otherwise
        """
        # Simple extension check
        if url.lower().endswith('.pdf'):
            return True

        # Check content type from headers
        try:
            headers = requests.head(url, allow_redirects=True,
timeout=10).headers
            content_type = headers.get('Content-Type', '').lower()
            return 'application/pdf' in content_type
        except Exception as e:
            logger.error(f"Error checking if URL is PDF: {e}")
            return False

    def process_pdf(self, url: str, qr_path: str) -> Optional[Dict[str,
Any]]:
        """Download and process PDF from URL.

        Args:
            url: The PDF URL to download
            qr_path: Path to QR code image

        Returns:
            Dict containing PDF info or None if failed
        """
        try:
            logger.info(f"Processing PDF URL: {url}")

            # Create filename for PDF
            parsed_url = urlparse(url)
            filename = os.path.basename(parsed_url.path)
            if not filename or not filename.lower().endswith('.pdf'):
                filename = f"document_{hash(url)}.pdf"

            pdf_path = os.path.join(self.temp_dir, filename)

            # Download PDF
            response = requests.get(url, stream=True, timeout=30)
            response.raise_for_status()

            with open(pdf_path, 'wb') as f:
                for chunk in response.iter_content(chunk_size=8192):
                    f.write(chunk)

            # Extract title from PDF metadata or filename
```

```
            title = self._extract_pdf_title(pdf_path, url)

            return {
                "title": title,
                "pdf_path": pdf_path
            }

        except Exception as e:
            logger.error(f"Error processing PDF URL: {e}")
            return None

    def _extract_pdf_title(self, pdf_path: str, url: str) -> str:
        """Extract title from PDF metadata or create from URL.

        Args:
            pdf_path: Path to downloaded PDF file
            url: Original URL

        Returns:
            Title string
        """
        try:
            # Try to get title from PDF metadata
            with open(pdf_path, 'rb') as f:
                reader = PyPDF2.PdfReader(f)
                if reader.metadata and reader.metadata.title:
                    return reader.metadata.title

            # Fall back to filename from URL
            parsed_url = urlparse(url)
            filename = os.path.basename(parsed_url.path)
            if filename.lower().endswith('.pdf'):
                filename = filename[:-4]  # Remove .pdf extension

            # Format filename as title
            title = filename.replace('_', ' ').replace('-', ' ')
            return title or "PDF Document"

        except Exception as e:
            logger.error(f"Error extracting PDF title: {e}")
            return "PDF Document"
```

1. **Document Service (`app/services/document_service.py`)**
2. **Remarkable Service (`app/services/remarkable_service.py`)**

```
"""Remarkable Cloud integration service."""

import os
import subprocess
import logging
from typing import Tuple

# Configure logging
logger = logging.getLogger(__name__)

class RemarkableService:
    """Handles uploads to Remarkable Cloud."""

    def __init__(self, rmapi_path: str, upload_folder: str = "/"):
        """Initialize with path to rmapi executable.

        Args:
            rmapi_path: Path to rmapi executable
            upload_folder: Remarkable Cloud folder for uploads
        """
        self.rmapi_path = rmapi_path
        self.upload_folder = upload_folder

    def upload(self, doc_path: str, title: str) -> Tuple[bool, str]:
        """Upload document to Remarkable Cloud.

        Args:
            doc_path: Path to document file (.rm or .pdf)
            title: Title for the document

        Returns:
            Tuple of (success, message)
        """
        try:
            logger.info(f"Uploading {title} to Remarkable Cloud")

            # Validate inputs
            if not os.path.exists(doc_path):
                return False, f"Document not found: {doc_path}"

            if not os.path.exists(self.rmapi_path):
                return False, f"rmapi not found at {self.rmapi_path}"

            # Execute rmapi command
            cmd = [self.rmapi_path, "put", doc_path, self.upload_folder]

            logger.info(f"Running command: {' '.join(cmd)}")
```

```
            result = subprocess.run(cmd, capture_output=True, text=True)

            if result.returncode == 0:
                logger.info(f"Successfully uploaded to Remarkable Cloud")
                return True, "Document uploaded successfully"
            else:
                logger.error(f"Error uploading to Remarkable:
{result.stderr}")
                return False, f"Upload error: {result.stderr}"

        except Exception as e:
            logger.error(f"Exception uploading to Remarkable: {e}")
            return False, f"Upload exception: {str(e)}"
```

1. **Service Interfaces (`app/services/interfaces.py`)**

```
"""Interface definitions for Pi Share Receiver services."""

from abc import ABC, abstractmethod
from typing import Dict, Any, List, Optional, Tuple

class IQRCodeService(ABC):
    @abstractmethod
    def generate_qr(self, url: str) -> Tuple[str, str]:
        """Generate QR code for URL."""
        pass

class IWebScraperService(ABC):
    @abstractmethod
    def scrape(self, url: str) -> Dict[str, Any]:
        """Scrape content from URL."""
        pass

class IPDFService(ABC):
    @abstractmethod
    def is_pdf_url(self, url: str) -> bool:
        """Check if URL points to PDF."""
        pass

    @abstractmethod
    def process_pdf(self, url: str, qr_path: str) -> Optional[Dict[str,
Any]]:
        """Process PDF URL."""
        pass

class IDocumentService(ABC):
```

```
    @abstractmethod
    def create_hcl(self, url: str, qr_path: str, content: Dict[str, Any])
-> Optional[str]:
        """Create HCL script from content."""
        pass

    @abstractmethod
    def create_rmdoc(self, hcl_path: str, url: str) -> Optional[str]:
        """Convert HCL to Remarkable document."""
        pass

class IRemarkableService(ABC):
    @abstractmethod
    def upload(self, doc_path: str, title: str) -> Tuple[bool, str]:
        """Upload document to Remarkable Cloud."""
        pass
```

1. **Service Implementations (`app/services/__init__.py`)**

```
"""Service implementations for Pi Share Receiver."""

from .qr_service import QRCodeService
from .web_scraper_service import WebScraperService
from .pdf_service import PDFService
from .document_service import DocumentService
from .remarkable_service import RemarkableService
from .interfaces import (
    IQRCodeService,
    IWebScraperService,
    IPDFService,
    IDocumentService,
    IRemarkableService
)
```

Phase 2: Create the Main Server Implementation

Now let's update the main server file to properly use our service modules:

```
#!/usr/bin/env python3
"""
Pi Share Receiver Server

Receives URLs via HTTP POST, processes them, and uploads to Remarkable Pro.
"""

import os
import json
import logging
from http.server import HTTPServer, BaseHTTPRequestHandler
import traceback

# Import service implementations
from services.qr_service import QRCodeService
from services.pdf_service import PDFService
from services.web_scraper_service import WebScraperService
from services.document_service import DocumentService
from services.remarkable_service import RemarkableService

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.StreamHandler(),
        logging.FileHandler('pi_share_receiver.log')
    ]
)
logger = logging.getLogger(__name__)

# Configuration Constants - Update these to match your paths
HOST = '0.0.0.0'
PORT = 9999
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
TEMP_DIR = os.path.join(BASE_DIR, 'temp')
OUTPUT_DIR = os.path.join(BASE_DIR, 'output')
RMAPI_PATH = '/usr/local/bin/rmapi'
DRAWJ2D_PATH = '/usr/local/bin/drawj2d'
RM_FOLDER = '/'

# Ensure directories exist
os.makedirs(TEMP_DIR, exist_ok=True)
os.makedirs(OUTPUT_DIR, exist_ok=True)

class URLHandler(BaseHTTPRequestHandler):
    """Handler for URL sharing requests."""
```

```
    def __init__(self, *args, **kwargs):
        """Initialize services before parent initialization."""
        # Initialize services
        self.qr_service = QRCodeService(TEMP_DIR)
        self.pdf_service = PDFService(TEMP_DIR, OUTPUT_DIR)
        self.web_scraper = WebScraperService(TEMP_DIR)
        self.document_service = DocumentService(TEMP_DIR, DRAWJ2D_PATH)
        self.remarkable_service = RemarkableService(RMAPI_PATH, RM_FOLDER)

        # Initialize parent
        super().__init__(*args, **kwargs)

    def do_POST(self):
        """Handle POST request with URL to process."""
        if self.path != '/share':
            self._send_error("Invalid endpoint. Use /share")
            return

        try:
            # Get content length
            content_length = int(self.headers.get('Content-Length', 0))
            if content_length == 0:
                self._send_error("Empty request")
                return

            # Read request body
            post_data = self.rfile.read(content_length)
            url = self._extract_url(post_data)

            if not url:
                self._send_error("No valid URL found")
                return

            logger.info(f"Processing URL: {url}")

            # Generate QR code
            qr_path, qr_filename = self.qr_service.generate_qr(url)
            logger.info(f"Generated QR code: {qr_filename}")

            # Process URL based on type
            if self.pdf_service.is_pdf_url(url):
                self._handle_pdf_url(url, qr_path)
            else:
                self._handle_webpage_url(url, qr_path)

        except Exception as e:
```

```
            logger.error(f"Error processing request: {str(e)}")
            logger.error(traceback.format_exc())
            self._send_error(f"Error processing request: {str(e)}")

    def _extract_url(self, post_data):
        """Extract URL from request data (JSON or plain text)."""
        # Try to decode as JSON
        try:
            data = json.loads(post_data.decode('utf-8'))
            url = data.get('url')
            if url and url.startswith(('http://', 'https://')):
                return url
        except json.JSONDecodeError:
            pass

        # Try as plain text
        try:
            text = post_data.decode('utf-8').strip()
            if text.startswith(('http://', 'https://')):
                return text
        except UnicodeDecodeError:
            pass

        return None

    def _handle_pdf_url(self, url, qr_path):
        """Handle PDF URL processing."""
        try:
            # Process PDF
            result = self.pdf_service.process_pdf(url, qr_path)
            if not result:
                self._send_error("Failed to process PDF")
                return

            # Upload to Remarkable
            success, message = self.remarkable_service.upload(
                result["pdf_path"],
                result["title"]
            )

            if success:
                self._send_success(f"PDF uploaded to Remarkable:
{result['title']}")
            else:
                self._send_error(f"Failed to upload PDF: {message}")

        except Exception as e:
```

```
            logger.error(f"Error processing PDF: {str(e)}")
            logger.error(traceback.format_exc())
            self._send_error(f"Error processing PDF: {str(e)}")

    def _handle_webpage_url(self, url, qr_path):
        """Handle webpage URL processing."""
        try:
            # Scrape content
            content = self.web_scraper.scrape(url)

            # Create HCL script
            hcl_path = self.document_service.create_hcl(url, qr_path,
content)
            if not hcl_path:
                self._send_error("Failed to create HCL script")
                return

            # Convert to Remarkable document
            rm_path = self.document_service.create_rmdoc(hcl_path, url)
            if not rm_path:
                self._send_error("Failed to convert to Remarkable format")
                return

            # Upload to Remarkable
            success, message = self.remarkable_service.upload(
                rm_path,
                content["title"]
            )

            if success:
                self._send_success(f"Webpage uploaded to Remarkable:
{content['title']}")
            else:
                self._send_error(f"Failed to upload document: {message}")

        except Exception as e:
            logger.error(f"Error processing webpage: {str(e)}")
            logger.error(traceback.format_exc())
            self._send_error(f"Error processing webpage: {str(e)}")

    def _send_success(self, message):
        """Send success response."""
        self.send_response(200)
        self.send_header('Content-Type', 'application/json')
        self.end_headers()

        response = json.dumps({
```

```
            "success": True,
            "message": message
        })

        self.wfile.write(response.encode('utf-8'))

    def _send_error(self, message):
        """Send error response."""
        self.send_response(400)
        self.send_header('Content-Type', 'application/json')
        self.end_headers()

        response = json.dumps({
            "success": False,
            "error": message
        })

        self.wfile.write(response.encode('utf-8'))

def main():
    """Start the HTTP server."""
    try:
        # Create server
        server = HTTPServer((HOST, PORT), URLHandler)
        logger.info(f"Server started at http://{HOST}:{PORT}")

        # Serve until interrupted
        server.serve_forever()
    except KeyboardInterrupt:
        pass
    except Exception as e:
        logger.error(f"Server error: {str(e)}")
        logger.error(traceback.format_exc())
    finally:
        if 'server' in locals():
            server.server_close()
        logger.info("Server stopped")

if __name__ == "__main__":
    main()
```

Phase 3: Create a Font Installation Script

Let's create a script to install the mid-century modern sans-serif font:

```
#!/bin/bash
# install_fonts.sh - Install fonts for Pi Share Receiver

set -e

echo "=== Installing Fonts for Pi Share Receiver ==="

# Create font directory if it doesn't exist
mkdir -p ~/.fonts

# Install Liberation Sans as a mid-century modern style font
if ! fc-list | grep -q "Liberation Sans"; then
    echo "Installing Liberation Sans font..."
    sudo apt-get update
    sudo apt-get install -y fonts-liberation
fi

# Install Poppins as an alternative (more distinctly mid-century modern)
if ! fc-list | grep -q "Poppins"; then
    echo "Installing Poppins font..."
    wget -q https://github.com/google/fonts/raw/main/ofl/poppins/Poppins-
Regular.ttf -O ~/.fonts/Poppins-Regular.ttf
    wget -q https://github.com/google/fonts/raw/main/ofl/poppins/Poppins-
Bold.ttf -O ~/.fonts/Poppins-Bold.ttf

    # Update font cache
    fc-cache -f
fi

# Check if fonts are installed
echo "Installed fonts:"
fc-list | grep -E 'Liberation|Poppins' | sort

echo "=== Font Installation Complete ==="
echo "To use these fonts in Pi Share Receiver, edit the heading_font
variable"
echo "in app/services/document_service.py to use one of the installed
fonts."
```

Phase 4: Create a Font Testing Script

This script will help verify which fonts are available and working with drawj2d:

```
#!/bin/bash
# test_fonts.sh - Test various fonts with drawj2d

set -e

echo "=== Testing Fonts with drawj2d ==="

# Create test HCL file
TEST_HCL="font_test.hcl"

cat > "$TEST_HCL" << 'EOF'
# Font test for Remarkable Pro

puts "size 1872 2404"
puts "pen black"

# Test Title fonts
puts "text 120 120 \"Font Test for Remarkable Pro\""

# Test various fonts
puts "set_font Helvetica 30"
puts "text 120 200 \"Helvetica - Classic Sans\""

puts "set_font Liberation Sans 30"
puts "text 120 250 \"Liberation Sans - Modern Sans\""

puts "set_font Poppins 30"
puts "text 120 300 \"Poppins - Mid Century Modern\""

puts "set_font DejaVu Sans 30"
puts "text 120 350 \"DejaVu Sans - Clean Sans\""

puts "set_font Ubuntu 30"
puts "text 120 400 \"Ubuntu - Modern Humanist\""

puts "set_font Lines 30"
puts "text 120 450 \"Lines - Remarkable Default\""

puts "set_font Times 30"
puts "text 120 500 \"Times - Classic Serif\""
EOF

# Convert to RM format
echo "Converting to Remarkable format..."
DRAWJ2D_PATH="/usr/local/bin/drawj2d"
RM_PATH="font_test.rm"
```

```
"$DRAWJ2D_PATH" -Trm -rmv6 -o "$RM_PATH" "$TEST_HCL"

# Check result
if [ -f "$RM_PATH" ] && [ -s "$RM_PATH" ]; then
    echo "✅ Font test successful"
    echo "Test file created: $RM_PATH ($(du -h "$RM_PATH" | cut -f1))"
    echo ""
    echo "To view the test file on your Remarkable Pro:"
    echo "  rmapi put $RM_PATH /"
else
    echo "❌ Font test failed"
    echo "Check if drawj2d is installed correctly"
fi

echo "=== Font Test Complete ==="
```

Phase 5: Update the Test Script

Let's improve the test script to provide better feedback:

```
#!/bin/bash
# test_service.sh - Test the Pi Share Receiver service

set -e

echo "=== Pi Share Receiver Test ==="

# Check if service is running
if systemctl is-active --quiet pi_share_receiver; then
    echo "✅ Service is running"
else
    echo "⚠️ Service is not running"
    echo "Starting service..."
    sudo systemctl start pi_share_receiver

    # Wait for service to start
    sleep 2

    if systemctl is-active --quiet pi_share_receiver; then
        echo "✅ Service started successfully"
    else
        echo "❌ Failed to start service"
        echo "Check service status:"
        sudo systemctl status pi_share_receiver
        exit 1
```

```
    fi
fi

# Use provided URL or default
if [ "$#" -ne 1 ]; then
    TEST_URL="https://en.wikipedia.org/wiki/ReMarkable"
    echo "No URL provided, using default: $TEST_URL"
else
    TEST_URL="$1"
    echo "Using provided URL: $TEST_URL"
fi

# Send URL to service
echo "Sending URL to service..."
RESPONSE=$(curl -s -X POST -H "Content-Type: application/json" -d
"{\"url\":\"$TEST_URL\"}" http://localhost:9999/share)

echo "Response: $RESPONSE"

# Check for success in response
if echo "$RESPONSE" | grep -q "\"success\":true"; then
    echo "✅ URL accepted by service"

    # Show log output
    echo "Recent log entries:"
    sudo journalctl -u pi_share_receiver -n 20 --no-pager

    echo ""
    echo "Check your Remarkable tablet for the document."
    echo "To view more detailed logs:"
    echo "  sudo journalctl -u pi_share_receiver -f"
else
    echo "❌ URL submission failed"
    echo "Check service logs for details:"
    sudo journalctl -u pi_share_receiver -n 50 --no-pager
fi

echo "=== Test Complete ==="
```

Phase 6: Create a Full Installation Script

Finally, let's create a script to put everything together:

```
#!/bin/bash
# install.sh - Complete setup for Pi Share Receiver

set -e

echo "=== Pi Share Receiver Installation ==="

# Create required directories
echo "Creating directories..."
mkdir -p temp
mkdir -p output
mkdir -p app/services

# Install dependencies
echo "Installing dependencies..."
./setup_deps.sh

# Install fonts
echo "Installing fonts..."
./install_fonts.sh

# Install service
echo "Installing system service..."
./service.sh install
./service.sh enable

# Test fonts with drawj2d
echo "Testing fonts..."
./test_fonts.sh

# Run a basic service test
echo "Testing service..."
./service.sh restart
sleep 2
./test_service.sh

echo "=== Installation Complete ==="
echo ""
echo "Pi Share Receiver has been installed and configured."
echo "The service should now be running and will start automatically at
boot."
echo ""
echo "You can now set up an iOS Shortcut to share URLs with:"
echo "  http://$(hostname -I | awk '{print $1}'):9999/share"
echo ""
echo "Use the following commands to manage the service:"
echo "  ./service.sh start|stop|restart|status|logs"
```

Implementation Steps

1. **Create Service Modules**:
   - Create the app/services directory
   - Add all service module files described above
2. **Update Server Code**:
   - Replace the current server.py with the implementation above
   - Make sure it has executable permissions: `chmod +x app/server.py`
3. **Install Fonts**:
   - Run the font installation script
   - Test which fonts are available with the test_fonts.sh script
   - Update `heading_font` in DocumentService to use your preferred font
4. **Configure and Start Service**:
   - Update the systemd service file if needed
   - Use service.sh to install, enable and start the service
   - Test the service with test_service.sh
5. **Create iOS Shortcut**:
   - Create an iOS Shortcut that sends a URL to your Pi's endpoint
   - Test the complete workflow from iOS to Remarkable

Troubleshooting Tips

1. **Check Logs**:

```
./service.sh logs
```

2. **Test Conversion Process**:

```
# Create a test HCL file
echo 'puts "size 1872 2404"
puts "text 120 120 \"Test Document\""' > test.hcl

# Convert with drawj2d
/usr/local/bin/drawj2d -Trm -rmv6 -o test.rm test.hcl

# Check the output
ls -l test.rm
```

3. **Verify Remarkable Cloud Access**:

```
# Test authentication
/usr/local/bin/rmapi ls
```

4. **Reset Service**:

```
sudo systemctl daemon-reload
sudo systemctl restart pi_share_receiver
```

This implementation plan provides a complete solution for the Pi Share Receiver with proper modular design, typography preferences, and Remarkable Pro optimization. The code follows Python best practices and provides comprehensive error handling and logging.