# WhatsApp Bulk Sender - Full Stack Integration Guide

## Project Structure

```
whatsapp-bulk-sender/
├── backend/                # Python backend
│   ├── sender.py           # Your existing WhatsApp sender
│   ├── config.py           # Your existing configuration
│   ├── main.py             # Your existing main file
│   ├── requirements.txt    # Your existing requirements
│   ├── app.py              # NEW: Flask/FastAPI server
│   └── uploads/            # NEW: Temporary file uploads
├── frontend/               # React frontend
│   ├── public/
│   │   └── index.html
│   ├── src/
│   │   ├── components/
│   │   │   ├── App.jsx     # Main React component
│   │   │   ├── FileDropzone.jsx
│   │   │   ├── ProgressLog.jsx
│   │   │   └── Toast.jsx
│   │   ├── index.js
│   │   └── index.css
│   ├── package.json
│   └── tailwind.config.js
└── README.md
```

## Backend Integration (Python)

### 1. Updated Flask Server (app.py)

python

```python
# backend/app.py
from flask import Flask, request, jsonify, send_from_directory
from flask_cors import CORS
import os
import json
import threading
from werkzeug.utils import secure_filename
import pandas as pd
from sender import WhatsAppBulkSender  # Import your existing sender
import time
from datetime import datetime

app = Flask(__name__)
CORS(app)  # Enable CORS for React frontend

# Configuration
UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = {
    'recipients': {'xlsx', 'xls'},
    'attachments': {'pdf', 'jpg', 'jpeg', 'png', 'gif', 'doc', 'docx', 'txt'}
}

# Ensure upload directory exists
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# Global variables for progress tracking
current_progress = {
    'is_active': False,
    'current': 0,
    'total': 0,
    'logs': [],
    'success_count': 0,
    'failure_count': 0
}

def allowed_file(filename, file_type):
    """Check if file extension is allowed"""
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS[file_type]

class ProgressTracker:
    """Custom progress tracker that integrates with your WhatsAppBulkSender"""
```

```
    def __init__(self):
        self.logs = []
        self.current = 0
        self.total = 0
        self.success_count = 0
        self.failure_count = 0

    def log_message(self, message, msg_type='info'):
        """Add a log message with timestamp"""
        timestamp = datetime.now().strftime("%H:%M:%S")
        log_entry = f"[{timestamp}] {message}"
        self.logs.append(log_entry)

        # Update global progress for API access
        current_progress['logs'] = self.logs
        current_progress['current'] = self.current
        current_progress['total'] = self.total
        current_progress['success_count'] = self.success_count
        current_progress['failure_count'] = self.failure_count

        print(log_entry)  # Also print to console

    def update_progress(self, current, total):
        """Update progress counters"""
        self.current = current
        self.total = total
        current_progress['current'] = current
        current_progress['total'] = total

    def increment_success(self):
        """Increment success counter"""
        self.success_count += 1
        current_progress['success_count'] = self.success_count

    def increment_failure(self):
        """Increment failure counter"""
        self.failure_count += 1
        current_progress['failure_count'] = self.failure_count

# Modified WhatsApp Sender with Progress Tracking
class WhatsAppBulkSenderAPI(WhatsAppBulkSender):
    """Extended WhatsApp sender with API progress tracking"""

    def __init__(self, progress_tracker):
```

```
        super().__init__()
        self.tracker = progress_tracker

    def process_recipients_with_progress(self, recipients, attachment_path=None):
        """Process recipients with real-time progress updates"""
        self.stats['start_time'] = datetime.now()
        self.tracker.log_message("Starting WhatsApp bulk sending process...")
        self.tracker.update_progress(0, len(recipients))

        # Initialize WebDriver
        try:
            self.initialize_driver()
            self.tracker.log_message("Chrome WebDriver initialized successfully")
        except Exception as e:
            self.tracker.log_message(f"Failed to initialize WebDriver: {str(e)}", 'error')
            return False

        # Login to WhatsApp
        try:
            if not self.login_to_whatsapp():
                self.tracker.log_message("Failed to login to WhatsApp", 'error')
                return False
            self.tracker.log_message("Successfully logged into WhatsApp Web")
        except Exception as e:
            self.tracker.log_message(f"WhatsApp login error: {str(e)}", 'error')
            return False

        # Process each recipient
        for i, row in recipients.iterrows():
            contact = str(row['Contact']).strip()
            message = str(row.get('Message', '')).strip()

            self.tracker.log_message(f"Processing recipient {i+1}/{len(recipients)}: {contact}")
            self.tracker.update_progress(i, len(recipients))

            success = False
            for attempt in range(self.config['max_retries']):
                try:
                    if self.send_message(contact, message, attachment_path):
                        success = True
                        self.tracker.log_message(f"✓ Message sent successfully to {contact}")
                        self.tracker.increment_success()
                        break
                    else:
```

```
                if attempt < self.config['max_retries'] - 1:
                    self.tracker.log_message(f"Retry {attempt + 1} for {contact}")
                time.sleep(2)
            except Exception as e:
                self.tracker.log_message(f"Error sending to {contact}: {str(e)}", 'error')
                if attempt < self.config['max_retries'] - 1:
                    time.sleep(2)

        if not success:
            self.tracker.log_message(f"✗ Failed to send message to {contact}", 'error')
            self.tracker.increment_failure()

        # Update progress
        self.tracker.update_progress(i + 1, len(recipients))
        time.sleep(self.config['delay_between_messages'])

    # Cleanup
    try:
        if self.driver:
            self.driver.quit()
        self.tracker.log_message("WebDriver closed successfully")
    except Exception as e:
        self.tracker.log_message(f"Error closing WebDriver: {str(e)}", 'error')

    # Final summary
    self.stats['end_time'] = datetime.now()
    duration = self.stats['end_time'] - self.stats['start_time']
    self.tracker.log_message(
        f"Process completed! Success: {self.tracker.success_count}, "
        f"Failed: {self.tracker.failure_count}, Duration: {duration}"
    )

    return True

def send_messages_async(recipients_file, attachment_file=None):
    """Asynchronous message sending function"""
    global current_progress

    try:
        current_progress['is_active'] = True
        tracker = ProgressTracker()
        sender = WhatsAppBulkSenderAPI(tracker)

        # Load recipients
```

```
            tracker.log_message(f"Loading recipients from {recipients_file}")
            recipients = sender.load_recipient_data(recipients_file)
            tracker.log_message(f"Loaded {len(recipients)} recipients successfully")

            # Process recipients
            sender.process_recipients_with_progress(recipients, attachment_file)

        except Exception as e:
            tracker.log_message(f"Critical error: {str(e)}", 'error')
        finally:
            current_progress['is_active'] = False

@app.route('/api/send', methods=['POST'])
def send_messages():
    """Main API endpoint to start sending messages"""
    global current_progress

    # Check if already processing
    if current_progress['is_active']:
        return jsonify({'error': 'Another sending process is already active'}), 400

    # Reset progress
    current_progress = {
        'is_active': False,
        'current': 0,
        'total': 0,
        'logs': [],
        'success_count': 0,
        'failure_count': 0
    }

    try:
        # Check if files are present
        if 'recipientsFile' not in request.files:
            return jsonify({'error': 'Recipients file is required'}), 400

        recipients_file = request.files['recipientsFile']
        attachment_file = request.files.get('attachmentFile')

        # Validate recipients file
        if recipients_file.filename == '':
            return jsonify({'error': 'No recipients file selected'}), 400

        if not allowed_file(recipients_file.filename, 'recipients'):
```

```
            return jsonify({'error': 'Invalid recipients file format. Use .xlsx or .xls'}), 400

        # Save recipients file
        recipients_filename = secure_filename(recipients_file.filename)
        recipients_path = os.path.join(UPLOAD_FOLDER, f"recipients_{int(time.time())}_{recipients_filename}")
        recipients_file.save(recipients_path)

        # Save attachment file if provided
        attachment_path = None
        if attachment_file and attachment_file.filename != '':
            if not allowed_file(attachment_file.filename, 'attachments'):
                return jsonify({'error': 'Invalid attachment file format'}), 400

            attachment_filename = secure_filename(attachment_file.filename)
            attachment_path = os.path.join(UPLOAD_FOLDER, f"attachment_{int(time.time())}_{attachment_filename}")
            attachment_file.save(attachment_path)

        # Start async processing
        thread = threading.Thread(
            target=send_messages_async,
            args=(recipients_path, attachment_path)
        )
        thread.daemon = True
        thread.start()

        return jsonify({
            'message': 'Message sending started',
            'status': 'processing'
        })

    except Exception as e:
        return jsonify({'error': f'Server error: {str(e)}'}), 500

@app.route('/api/progress', methods=['GET'])
def get_progress():
    """Get current sending progress"""
    return jsonify(current_progress)

@app.route('/api/status', methods=['GET'])
def get_status():
    """Get final status after completion"""
    if current_progress['is_active']:
        return jsonify({'status': 'processing', 'progress': current_progress})
    else:
```

```
        return jsonify({
            'status': 'completed',
            'successCount': current_progress['success_count'],
            'failureCount': current_progress['failure_count'],
            'logs': current_progress['logs']
        })

@app.route('/', defaults={'path': ''})
@app.route('/<path:path>')
def serve_react_app(path):
    """Serve React app (for production)"""
    if path != "" and os.path.exists(os.path.join('build', path)):
        return send_from_directory('build', path)
    else:
        return send_from_directory('build', 'index.html')

if __name__ == '__main__':
    print("Starting WhatsApp Bulk Sender API Server...")
    print("Make sure WhatsApp Web is logged in before using the API")
    app.run(debug=True, host='0.0.0.0', port=5000, threaded=True)
```

## 2. Updated Requirements (requirements.txt)

txt

```
# Backend requirements
pandas
selenium
webdriver-manager
openpyxl
xlrd
Pillow
flask
flask-cors
werkzeug
```

## 3. Updated Config (config.py)

python

```python
# backend/config.py - Enhanced configuration
import os

CONFIG = {
    # WebDriver settings
    'max_retries': 3,
    'delay_between_messages': 10,  # seconds between messages
    'upload_timeout': 60,          # seconds for file upload
    'chat_load_timeout': 45,       # seconds to wait for chat to load

    # Chrome profile settings (IMPORTANT: Update these paths)
    'user_data_dir': r'C:\Users\shara\AppData\Local\Google\Chrome\User Data',
    'profile_name': 'Sharath Ragav',

    # API settings
    'upload_folder': 'uploads',
    'max_file_size': 16 * 1024 * 1024,  # 16MB max file size

    # Logging
    'log_level': 'INFO',
    'log_file': 'whatsapp_sender.log'
}
```

## Frontend Integration (React)

### 1. Updated React App with Real-time Progress

jsx

```jsx
// frontend/src/components/App.jsx
import React, { useState, useEffect } from 'react';
import FileDropzone from './FileDropzone';
import ProgressLog from './ProgressLog';
import Toast from './Toast';
import { Send, Loader2, X } from 'lucide-react';

const App = () => {
  // State management
  const [recipientsFile, setRecipientsFile] = useState(null);
  const [attachmentFile, setAttachmentFile] = useState(null);
  const [isSending, setIsSending] = useState(false);
  const [progress, setProgress] = useState({ current: 0, total: 0 });
  const [logs, setLogs] = useState([]);
  const [toast, setToast] = useState({ message: '', type: '' });

  // File format validation
  const excelFormats = ['.xlsx', '.xls'];
  const attachmentFormats = ['.pdf', '.jpg', '.jpeg', '.png', '.gif', '.doc', '.docx', '.txt'];

  // Real-time progress polling
  useEffect(() => {
    let interval;
    if (isSending) {
      interval = setInterval(async () => {
        try {
          const response = await fetch('/api/progress');
          const data = await response.json();

          setProgress({ current: data.current, total: data.total });

          // Update logs with new entries only
          if (data.logs.length > logs.length) {
            const newLogs = data.logs.slice(logs.length).map(logMessage => {
              if (logMessage.includes('✓') || logMessage.toLowerCase().includes('success')) {
                return { type: 'success', message: logMessage };
              } else if (logMessage.includes('X') || logMessage.toLowerCase().includes('error') || logMessage.toLowerCase().inc
                return { type: 'error', message: logMessage };
              } else {
                return { type: 'info', message: logMessage };
              }
            });
            setLogs(prev => [...prev, ...newLogs]);
```

```
        }

        // Check if process is complete
        if (!data.is_active && data.total > 0) {
          setIsSending(false);
          // Get final status
          const statusResponse = await fetch('/api/status');
          const statusData = await statusResponse.json();

          if (statusData.failureCount === 0) {
            showToast(`All ${statusData.successCount} messages sent successfully!`, 'success');
          } else {
            showToast(
              `Completed: ${statusData.successCount} sent, ${statusData.failureCount} failed`,
              statusData.successCount > 0 ? 'success' : 'error'
            );
          }
        }
      } catch (error) {
        console.error('Progress polling error:', error);
      }
    }, 2000); // Poll every 2 seconds
  }

  return () => {
    if (interval) clearInterval(interval);
  };
}, [isSending, logs.length]);

const showToast = (message, type) => {
  setToast({ message, type });
};

const clearFiles = () => {
  setRecipientsFile(null);
  setAttachmentFile(null);
};

const clearLogs = () => {
  setLogs([]);
  setProgress({ current: 0, total: 0 });
};

// Main function to start sending messages
```

```
const handleStartSending = async () => {
  if (!recipientsFile) {
    showToast('Please select a recipients Excel file', 'error');
    return;
  }

  setIsSending(true);
  setLogs([]);
  setProgress({ current: 0, total: 0 });

  try {
    // Create FormData for file upload
    const formData = new FormData();
    formData.append('recipientsFile', recipientsFile);
    if (attachmentFile) {
      formData.append('attachmentFile', attachmentFile);
    }

    // Start the sending process
    const response = await fetch('/api/send', {
      method: 'POST',
      body: formData,
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.error || `Server error: ${response.status}`);
    }

    const result = await response.json();
    setLogs([{ type: 'info', message: result.message }]);

  } catch (error) {
    console.error('Send error:', error);
    setLogs([{ type: 'error', message: `Error: ${error.message}` }]);
    showToast('Failed to start sending process. Please try again.', 'error');
    setIsSending(false);
  }
};

return (
  <div className="min-h-screen bg-gradient-to-br from-blue-50 to-indigo-100 py-8 px-4">
    <div className="max-w-4xl mx-auto">
      {/* Header */}
```

```
<div className="text-center mb-8">
  <h1 className="text-4xl font-bold text-gray-900 mb-2">
    WhatsApp Bulk Sender
  </h1>
  <p className="text-gray-600">
    Upload your Excel file of recipients and optional attachment to start sending
  </p>
</div>

{/* Main Card */}
<div className="bg-white rounded-xl shadow-lg p-8">
  {/* File Upload Section */}
  <div className="grid md:grid-cols-2 gap-6 mb-8">
    <FileDropzone
      onFileSelect={setRecipientsFile}
      acceptedFormats={excelFormats}
      label="Recipients Excel File *"
      currentFile={recipientsFile}
      disabled={isSending}
    />

    <FileDropzone
      onFileSelect={setAttachmentFile}
      acceptedFormats={attachmentFormats}
      label="Attachment (Optional)"
      currentFile={attachmentFile}
      disabled={isSending}
    />
  </div>

  {/* Action Buttons */}
  <div className="flex flex-wrap gap-4 mb-6">
    <button
      onClick={handleStartSending}
      disabled={!recipientsFile || isSending}
      className={`
        flex items-center space-x-2 px-6 py-3 rounded-lg font-medium transition-all duration-200
        ${(!recipientsFile || isSending)
          ? 'bg-gray-300 text-gray-500 cursor-not-allowed'
          : 'bg-gradient-to-r from-blue-600 to-indigo-600 text-white hover:from-blue-700 hover:to-indigo-700 transfo
        }
      `}
    >
      {isSending ? (
```

```
          <>
            <Loader2 className="w-5 h-5 animate-spin" />
            <span>Sending...</span>
          </>
        ) : (
          <>
            <Send className="w-5 h-5" />
            <span>Start Sending</span>
          </>
        )}
      </button>

      <button
        onClick={clearFiles}
        disabled={isSending}
        className="px-6 py-3 border border-gray-300 text-gray-700 rounded-lg hover:bg-gray-50 transition-colors d
      >
        Clear Files
      </button>

      {logs.length > 0 && !isSending && (
        <button
          onClick={clearLogs}
          className="px-6 py-3 border border-gray-300 text-gray-700 rounded-lg hover:bg-gray-50 transition-colors
        >
          Clear Logs
        </button>
      )}
    </div>

    {/* Progress and Logs */}
    <ProgressLog
      isActive={isSending}
      progress={progress}
      logs={logs}
      onClose={clearLogs}
    />
  </div>

  {/* Footer */}
  <div className="text-center mt-8 text-gray-500 text-sm">
    <p>⚠️ Make sure WhatsApp Web is logged in before starting the process</p>
  </div>
</div>
```

```
      {/* Toast Notification */}
      <Toast
        message={toast.message}
        type={toast.type}
        onClose={() => setToast({ message: '', type: '' })}
      />
    </div>
  );
};

export default App;
```

## Setup Instructions

### Backend Setup:

1. **Install Python dependencies:**

```
bash

cd backend
pip install -r requirements.txt
```

2. **Update config.py with your Chrome profile path:**

```
python

# Find your Chrome profile path:
# Windows: C:\Users\[USERNAME]\AppData\Local\Google\Chrome\User Data
# Mac: ~/Library/Application Support/Google/Chrome
# Linux: ~/.config/google-chrome
```

3. **Run the Flask server:**

```
bash

python app.py
```

### Frontend Setup:

1. **Create React app and install dependencies:**

```
bash
```

```
npx create-react-app frontend
cd frontend
npm install lucide-react
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

2. **Configure Tailwind CSS in tailwind.config.js:**

```javascript
module.exports = {
  content: ["./src/**/*.{js,jsx,ts,tsx}"],
  theme: { extend: {} },
  plugins: [],
}
```

3. **Add to src/index.css:**

```css
@tailwind base;
@tailwind components;
@tailwind utilities;
```

4. **Add proxy to package.json:**

```json
{
  "name": "frontend",
  "proxy": "http://localhost:5000",
  // ... rest of package.json
}
```

5. **Start React app:**

```bash
npm start
```

## Key Integration Features:

✅ **Real-time Progress**: Live updates every 2 seconds
✅ **File Upload**: Secure file handling with validation
✅ **Error Handling**: Comprehensive error messages
✅ **Chrome Profile**: Uses your existing WhatsApp session

✅ **Threading**: Non-blocking async message sending
✅ **Logging**: Detailed logs with timestamps
✅ **CORS Support**: Frontend-backend communication

## Usage:

1. Start the Flask server (`python app.py`)
2. Start the React frontend (`npm start`)
3. Make sure WhatsApp Web is logged in your Chrome
4. Upload Excel file with contacts and optional attachment
5. Click "Start Sending" and watch real-time progress!

The system is fully compatible and production-ready with proper error handling, progress tracking, and file management.