Claude Recommended implementation to correct the file size limitation for large attachments

**Recommended Implementation: Option 2**

I recommend implementing Option 2 as it's the most reliable and scalable. Here's how to do it:

1. Create a simple PHP endpoint on your server to handle file uploads:

|  |
| --- |
| 1. <?php 2. // upload.php 3. header('Content-Type: application/json'); 4. // Check if this is a valid upload request 5. if (!isset($\_FILES['file']) || !isset($\_POST['user\_id'])) { 6. echo json\_encode(['success' => false, 'message' => 'Missing required parameters']); 7. exit; 8. } 9. $userId = $\_POST['user\_id']; 10. $file = $\_FILES['file']; 11. // Create uploads directory if it doesn't exist 12. $targetDir = "uploads/user\_$userId/"; 13. if (!file\_exists($targetDir)) { 14. mkdir($targetDir, 0777, true); 15. } 16. // Generate a unique filename 17. $fileName = time() . '\_' . basename($file['name']); 18. $targetPath = $targetDir . $fileName; 19. // Move the uploaded file to the target directory 20. if (move\_uploaded\_file($file['tmp\_name'], $targetPath)) { 21. $fileUrl = "https://memre.vortisllc.com/$targetPath"; 22. echo json\_encode([ 23. 'success' => true, 24. 'file\_url' => $fileUrl, 25. 'file\_name' => $fileName 26. ]); 27. } else { 28. echo json\_encode(['success' => false, 'message' => 'Upload failed']); 29. } 30. ?> |
|  |

Modify your database schema to store file URLs:

|  |
| --- |
| Future<void> updateSchema(int userId) async {  final conn = await \_getConnection();  try {  // ... existing schema updates ...    // Check if file\_url column exists  final fileUrlResult = await conn.query(  "SHOW COLUMNS FROM user\_${userId}\_attachment LIKE 'file\_url'",  );  if (fileUrlResult.isEmpty) {  print('Adding file\_url column to attachment table for user $userId');  await conn.query(  'ALTER TABLE user\_${userId}\_attachment ADD COLUMN file\_url TEXT',  );  print('Added file\_url column to attachment table');  }  } catch (e) {  print('Error updating schema: $e');  } finally {  await conn.close();  }  } |

Update your Flutter app to upload large files to your server:

|  |
| --- |
| Future<String?> \_uploadLargeFile(File file, String fileName, int userId) async {  try {  // Create multipart request  final request = http.MultipartRequest(  'POST',  Uri.parse('https://memre.vortisllc.com/upload.php'),  );    // Add file  request.files.add(await http.MultipartFile.fromPath(  'file',  file.path,  filename: fileName,  ));    // Add user ID  request.fields['user\_id'] = userId.toString();    // Send the request  final response = await request.send();  final responseData = await response.stream.bytesToString();  final jsonData = json.decode(responseData);    if (jsonData['success'] == true) {  return jsonData['file\_url'];  } else {  print('Upload failed: ${jsonData['message']}');  return null;  }  } catch (e) {  print('Error uploading file: $e');  return null;  }  } |

Modify your \_pickFile method to handle uploading large videos:

|  |
| --- |
| Future<void> \_pickFile() async {  try {  // ... existing file picking code ...    if (file != null && \_isValidFileType(file.name)) {  final String filePath = file.path;  final File fileObj = File(filePath);  final int fileSize = await fileObj.length();    // For videos larger than 10MB, upload to server  final bool isLargeVideo = fileSize > 10 \* 1024 \* 1024 &&  \_attachmentType == AttachmentType.video;    if (isLargeVideo) {  // Show upload progress  showDialog(  context: context,  barrierDismissible: false,  builder: (BuildContext context) {  return AlertDialog(  title: Text('Uploading Video'),  content: Column(  mainAxisSize: MainAxisSize.min,  children: [  CircularProgressIndicator(),  SizedBox(height: 16),  Text('Please wait while your video uploads...'),  ],  ),  );  },  );    // Upload to server  final fileUrl = await \_uploadLargeFile(fileObj, file.name, widget.userId);    // Close progress dialog  Navigator.of(context).pop();    if (fileUrl != null) {  setState(() {  \_attachmentData = null; // Don't store in memory  \_attachmentFileName = file.name;  \_attachmentType = AttachmentType.video;  \_attachmentFileUrl = fileUrl; // Store the URL  });  print('Video uploaded to: $fileUrl');  } else {  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(content: Text('Failed to upload video')),  );  }  } else {  // For smaller files, continue with your existing approach  final Uint8List fileData = await file.readAsBytes();  setState(() {  \_attachmentData = fileData;  \_attachmentFileName = file.name;  });  }  }  } catch (e) {  // ... error handling ...  }  } |

Update your database methods to store and retrieve file URLs:

|  |
| --- |
| Future<int> createMemo(  // ... existing parameters ...  {String? fileUrl} // Add this parameter  ) async {  // ... existing code ...    // Handle attachment  if (fileName != null && attachmentType != null) {  if (fileUrl != null && attachmentType == AttachmentType.video) {  // Store only metadata and file URL for videos  await conn.query(  'INSERT INTO user\_${userId}\_attachment (memo\_id, file\_type, file\_name, file\_url) VALUES (?, ?, ?, ?)',  [memoId, attachmentType.name, fileName, fileUrl],  );  } else if (attachmentData != null) {  // Store other attachments as blob data  await conn.query(  'INSERT INTO user\_${userId}\_attachment (memo\_id, file\_data, file\_type, file\_name) VALUES (?, ?, ?, ?)',  [memoId, attachmentData, attachmentType.name, fileName],  );  }  }    // ... rest of the method ...  } |

This approach ensures that all your data, including large video files, is stored in the cloud and can be recovered if a device is lost or damaged, while also solving the "value is out of range" error.