

# PSScript Application Fixes

---

This document provides a comprehensive guide to the fixes implemented for the PSScript application to address critical issues with script deletion and file uploads.

# Issues Fixed

---

## 1. Script Deletion Error

When clicking the trash can icon to delete a script, an error message "Failed to delete script(s). Please try again." was displayed. The script was not being deleted from the database, and the UI was not updating correctly.

## 2. Network Error During Upload

When attempting to upload a script file, a "Network error. Please check your connection." message was displayed. The file was not being uploaded to the server, and no error details were provided.

# Technical Fixes Implemented

---

## Backend Fixes

### 1. Enhanced Transaction Management in ScriptController

- Improved transaction handling with proper rollbacks in case of errors
- Added detailed error responses with success flags
- Fixed error handling for transaction rollbacks

```
// Rollback transaction if there was an error
if (transaction) {
  try {
    await transaction.rollback();
  } catch (rollbackError) {
    console.error('Error rolling back transaction:', rollbackError);
  }
}

// Return a structured error response
res.status(500).json({
  message: 'Failed to delete script',
  error: error.message,
  success: false
});
```

### 2. Improved CORS Middleware

- Enhanced CORS handling to properly support file uploads
- Added support for credentials and proper origin handling
- Fixed issues with preflight requests

```
// Get the origin from the request
const origin = req.headers.origin || '*';

// Add permissive CORS headers specifically for file uploads
res.header('Access-Control-Allow-Origin', origin);
res.header('Access-Control-Allow-Methods', 'POST, OPTIONS');
res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With');
res.header('Access-Control-Allow-Credentials', 'true');
res.header('Access-Control-Max-Age', '86400'); // 24 hours
```

### 3. Network Error Handling Middleware

- Added middleware to handle network errors during uploads
- Implemented timeout handling for upload requests
- Added detailed error responses for different failure scenarios

```

// Middleware to handle network errors during upload
const handleNetworkErrors = (req: Request, res: Response, next: NextFunction) => {
  // Set a longer timeout for upload requests
  req.setTimeout(120000); // 2 minutes

  // Handle connection close events
  req.on('close', () => {
    if (!res.headersSent) {
      logger.warn('[UPLOAD] Client closed connection before response');
    }
  });

  // Handle timeout events
  req.on('timeout', () => {
    logger.error('[UPLOAD] Request timeout');
    if (!res.headersSent) {
      res.status(408).json({
        error: 'request_timeout',
        message: 'The request timed out. Please try again with a smaller file.',
        success: false
      });
    }
  });

  next();
};

```

#### 4. Updated Routes Configuration

- Applied the network error handling middleware to upload routes
- Ensured proper middleware order for uploads

```

// Use special CORS middleware and network error handling for upload routes
router.post('/upload', uploadCorsMiddleware, handleNetworkErrors,

```

## Frontend Fixes

### 1. Enhanced Error Handling in API Service

- Improved error handling for script deletion
- Added specific error messages for different failure scenarios
- Ensured proper error propagation to the UI

```
deleteScript: async (id: string) => {
  try {
    const response = await apiClient.delete(`/scripts/${id}`);
    return response.data;
  } catch (error) {
    console.error(`Error deleting script ${id}:`, error);

    // Provide more specific error messages
    if ((error as any).status === 404) {
      throw new Error('Script not found. It may have been already deleted');
    }

    if ((error as any).status === 403) {
      throw new Error('You do not have permission to delete this script');
    }

    // Return a structured error object
    throw {
      message: (error as any).message || 'Failed to delete script',
      status: (error as any).status || 500,
      success: false
    };
  }
}
```

### 2. Improved React Query Mutation Handling

- Enhanced the delete script mutation in ScriptManagement.tsx
- Added proper success and error handling
- Implemented user-friendly error messages

```
// Delete script mutation with improved error handling
const deleteScriptMutation = useMutation(
  (id: string) => scriptService.deleteScript(id),
  {
    onSuccess: (data) => {
      if (data.success) {
        // Show success toast or notification
        console.log('Script deleted successfully');
        queryClient.invalidateQueries('scripts');
      }
    },
    onError: (error: any) => {
      // Show error toast or notification
      console.error('Failed to delete script:', error);
      alert(error.message || 'Failed to delete script. Please try');
    }
  }
);
```

# Testing the Fixes

---

## Restart Scripts

Several scripts have been created to restart the application with the fixes:

1. **restart-backend.sh**: Restarts only the backend server
2. **restart-frontend.sh**: Restarts only the frontend server
3. **restart-all.sh**: Restarts both the backend and frontend servers

To restart the entire application:

```
./restart-all.sh
```

## Testing Scripts

Two testing scripts have been created to verify the fixes:

1. **test-fixes.sh**: Tests the backend API directly using curl
2. **test-ui-fixes.sh**: Opens the browser to test the UI fixes

## Backend API Testing

To test the backend API directly:

```
./test-fixes.sh
```

This script will: - Test script deletion by sending a DELETE request to the API - Test script upload by sending a POST request with a test script file

## UI Testing

To test the UI fixes:

```
./test-ui-fixes.sh
```



This script will: - Open the script management page to test deletion - Open the upload page to test file uploads - Provide instructions for manual testing

## **Manual Testing Steps**

### **Testing Script Deletion:**

1. Navigate to the script management page (<http://localhost:3000/manage>)
2. Find a script in the list
3. Click the 'Delete' button (trash can icon)
4. Confirm the deletion in the dialog
5. Verify that the script is removed without errors

### **Testing Script Upload:**

1. Navigate to the upload page (<http://localhost:3000/upload>)
2. Click 'Choose File' or drag and drop a PowerShell script
3. Fill in the title and description
4. Click 'Upload'
5. Verify that the upload completes without network errors

# Documentation

---

For a more detailed explanation of the fixes, refer to the following files:

- **FIXES.md**: Detailed explanation of the fixes implemented
- **README-FIXES.md**: This comprehensive guide
- **test-fixes.sh**: Backend API testing script
- **test-ui-fixes.sh**: UI testing script

# Conclusion

---

These fixes address the core issues with script deletion and file uploads in the PSScript application. The improved error handling, transaction management, and CORS configuration ensure a more robust user experience.

If you encounter any issues or have questions about the fixes, please refer to the detailed documentation or contact the development team.

Generated 2026-01-16 21:23 UTC