CLAUDE.md

This file provides guidance to Claude Code (claude.ai/code) when working with code in this repository.

Project Overview

This is a Desktop Video Converter application built with Electron, React, and TypeScript. The application provides a simple, single-purpose tool to convert video files into web-optimized MP4 format with minimal user interaction. The project is currently in development stage with a fully configured build system.

Technology Stack

- **Framework**: Electron 38.x (desktop application framework)
- **UI Library**: React 19.x with TypeScript
- **Build Tool**: Vite 7.x with Hot Module Replacement
- **Styling**: Tailwind CSS 4.x (utility-first CSS framework)
- **State Management**: React Hooks (useState, useContext)
- **Testing**: Vitest + React Testing Library (configured but not implemented)

Development Commands

Core Development

```
# Start development server with hot reload
npm run electron:dev

# Build production version
npm run build:electron

# Run frontend development server only
npm run dev

# Build frontend only
npm run build
```

Quality Assurance

```
# Run ESLint
npm run lint

# Build and package for distribution (has known signing issues)
npm run electron:pack

# Create distributable installer
npm run electron:dist
```

Architecture Overview

Project Structure

```
├── electron/           # Electron-specific files
│   ├── main.ts        # Main process (window management, IPC)
│   ├── preload.ts     # Secure bridge between main and renderer
│   └── tsconfig.json  # TypeScript config for Electron
├── src/               # React frontend
│   ├── App.tsx        # Main application component
│   ├── main.tsx       # React entry point
│   ├── global.d.ts    # TypeScript definitions for Electron API
│   └── *.css          # Styling files
├── dist/              # Built frontend assets
├── dist-electron/     # Built Electron assets
└── public/            # Static assets
```

Inter-Process Communication (IPC)

The application uses Electron's secure IPC pattern:

Main Process (`electron/main.ts`):

- Handles file system operations via `ipcMain.handle()`
- Provides file selection dialogs: `select-file`, `select-save-location`, `select-directory`
- Manages application lifecycle and window creation

Preload Script (`electron/preload.ts`):

- Exposes safe APIs through `contextBridge.exposeInMainWorld('electronAPI', ...)`
- Provides type-safe interface between renderer and main process

Renderer Process (`src/App.tsx`):

- Accesses Electron APIs via `window.electronAPI`
- Handles UI state and user interactions

State Management Architecture

- All application state managed in main `App.tsx` component
- Uses React hooks (`useState`) for local state
- No external state management library needed for this single-screen application
- State includes: selected file path, output path, conversion settings

Build Configuration Notes

TypeScript Configuration

- **Frontend**: Uses `tsconfig.app.json` with ES2022 modules
- **Electron**: Uses `electron/tsconfig.json` with NodeNext modules for proper ES module handling
- **Global definitions**: TypeScript interfaces for Electron API in `src/global.d.ts`

Vite Configuration

- Configured with `vite-plugin-electron` and `vite-plugin-electron-renderer`
- Hot reload support for both main and preload scripts
- Tailwind CSS integration with `@import "tailwindcss"`

Electron Builder Configuration

- Targets Windows x64 platform
- Configured for portable executable output
- Note: Code signing currently has permission issues on Windows due to symbolic link requirements

Design System

Colors

- Primary: #0078D4 (Microsoft Blue)
- Secondary: #6C757D
- Success: #28A745
- Error: #DC3545

Typography

- Font Family: 'Segoe UI' (Windows native font)
- Grid System: 8px base grid

Component Standards

- Use TypeScript interfaces for all props
- Follow PascalCase for component files
- Functional components with React.FC type
- Custom hooks prefixed with 'use'

Key Implementation Details

File Selection Pattern

The application implements secure file selection through Electron's dialog API:

```
// In main.ts
ipcMain.handle('select-file', async () => {
  const result = await dialog.showOpenDialog({
    properties: ['openFile'],
    filters: [{ name: 'Video Files', extensions: ['mp4', 'avi', 'mov', ...] }]
  });
```

```
  return result.filePaths;
});

// In App.tsx
const filePaths = await window.electronAPI.selectFile();
```

Development vs Production Rendering

- **Development**: Loads from Vite dev server (`http://localhost:5173`)
- **Production**: Loads from built files (`dist/index.html`)
- Main process automatically detects environment and loads appropriate source

Known Issues

Packaging Limitations

- `electron-builder` packaging currently fails due to Windows symbolic link permissions with Tailwind CSS WASM dependencies
- Development and building work correctly
- Workaround: Tailwind dependencies moved to `devDependencies` to exclude from packaging

Development Warnings

- Windows cache permission warnings in development are normal and non-blocking
- GPU process warnings are typical for Electron development environment

Application Requirements

Per the PRD documentation:

- Single-screen application for video file conversion
- Must support drag & drop and file selection button
- Real-time conversion progress display
- User can cancel conversion in progress
- About dialog accessible to end users
- Target: Windows 10/11 desktop platforms
- Distribution: Single portable .exe file (no installation required)