Electron Video Converter Description

GitHub Repository: [https://github.com/Ashot72/electron-video-converter](https://github.com/Ashot72/electron-video-converter" \t "_blank)

How to run application: <https://ashot72.github.io/electron-video-converter/index.htm>

Please read "How to run application" first.

This application is built on Ionic 3 (Angular 4) and Electron and hosted on Firebase. It uses Firebase Database and Storage.

Electron is composed of Main and Renderer processes.

The main process takes care of starting and running the app defined in package.json's main script. The script that runs in the main process can display a GUI by creating web pages.

The main process manages all web pages and their corresponding renderer processes. The Renderer process takes care of showing the app in the Chromium browser.

Each renderer process is isolated and only cares about the web page running in it. Electron desktop application can access all node modules like the file system module

for handling files, request to make HTTP calls, etc.

Electron provides us with 2 IPC (Inter Process Communication) modules called **ipcMain** and **ipcRenderer**.

The **ipcMain** process module is used to communicate asynchronously from the main process to renderer processes. When used in the main process, the module handles

asynchronous and synchronous messages sent from a renderer process (web page). The messages sent from a renderer will be emitted to this module.

The **ipcRenderer** module is used to communicate asynchronously from a renderer process to main process. It provides a few methods so you can send synchronous and

asynchronous messages from the renderer process (web page) to the main process. You can also receive replies from the main process.

It is recommended not to perform computation of heavy/ blocking tasks on the renderer process. We should always use IPC to delegate these tasks to the main process.

Another option is to execute a function of the main process inside the renderer process in Electron Framework. We execute functions that are only accessible

at the main process level. The first thing we do is to create an event listener in the main process using the **ipcMain** module. We only attach an event listener whose

callback will be executed once the ipcRenderer module (in the view) requests its execution. We will use both approaches in our application.

To implement all these electrons specific functionality in ionic/angular way we will use ngx-electron package <https://github.com/ThorstenHans/ngx-electron> It is an

angular wrapper for electron renderer apis.

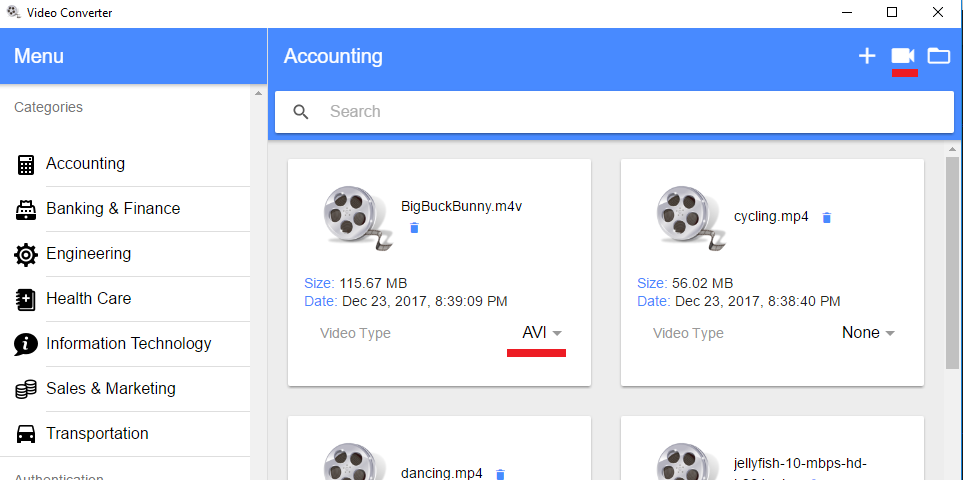


Figure 1

Select a video type to convert by clicking video icon on the top right corner.

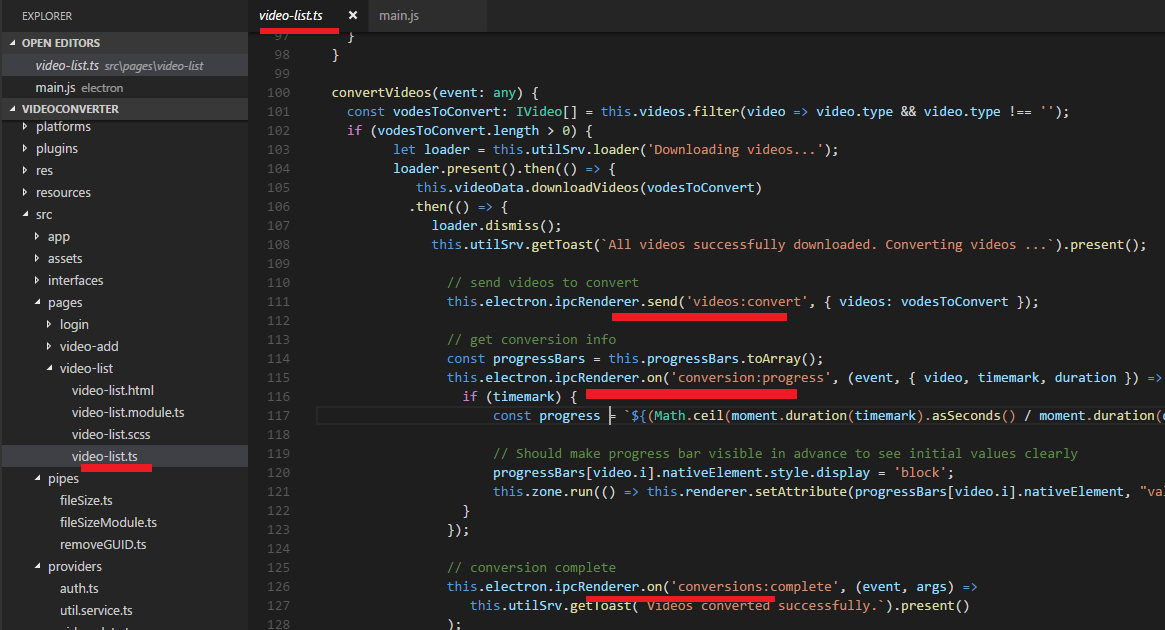


Figure 2

We send videos to be converted from the renderer process to main once the video icon is clicked. The channel name is 'videos:convert' (line 111).

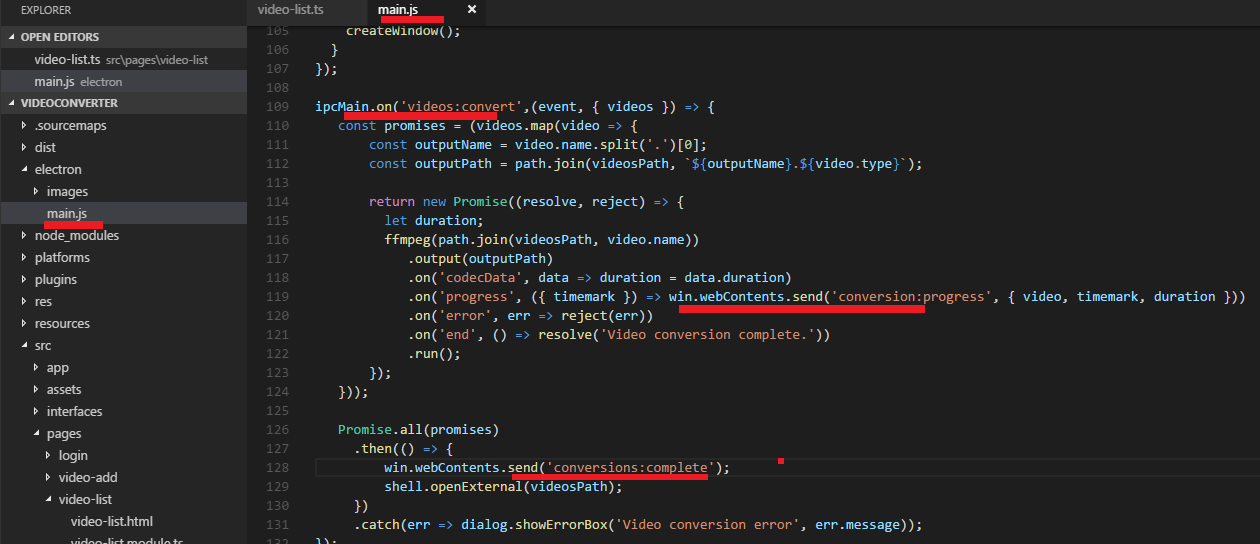


Figure 3

On the Electron side in the main process we use 'ipcMain.on' to deal with the videos sent from the renderer. Here we use ffmpeg to do actual conversion.

As stated it is recommended not to perform computation of heavy/ blocking tasks on the renderer process so, we make the conversion on the main process.

We use fluent-ffmpeg node module <https://github.com/fluent-ffmpeg/node-fluent-ffmpeg> for conversion.

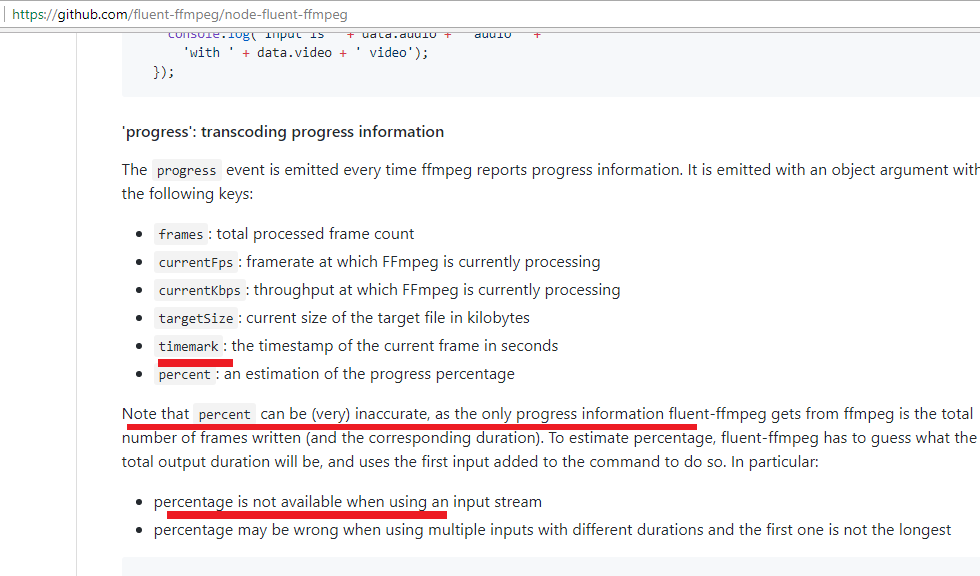


Figure 4

To track the conversion, we could have used progress event but it can vey inaccurate. For that reason, we use 'timemark' event and duration to calculate the progress

(figure 2, line 117). Next thing that we should do is sending 'video', 'timemark' and 'duration' info form the main process to renderer (figure 3, line 119).

The channel name is 'conversion:progress'. On the renderer side based on provided info sent from the main process we render the actual conversion progress bar

indicator with the percentage (figure 2, lines 115 - 121). The last message we send from the main process to renderer is

'conversions:comlpete' meaning that all video conversions have been completed successfully (figure 3, line 128). This is just showing toast notification to clients (figure 2, lines 126,127).

After the successful conversions the 'converter' folder is opened on the client's OS automatically for a client to see both the original and converted videos.

It is 'shell.openExternal(videosPath) ' (figure 3 line 129).

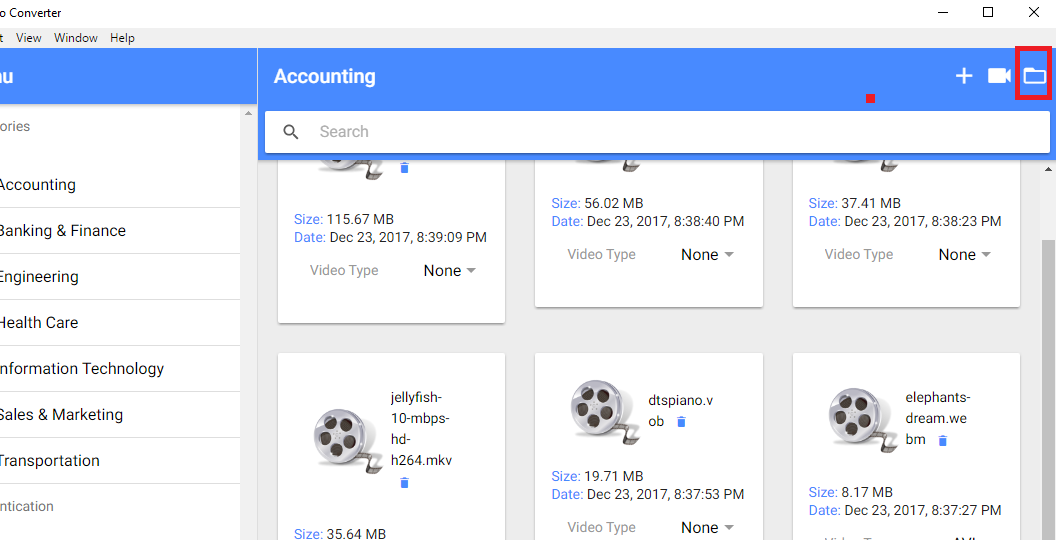


Figure 5

'converter' folder can always be opened form the renderer process as well.

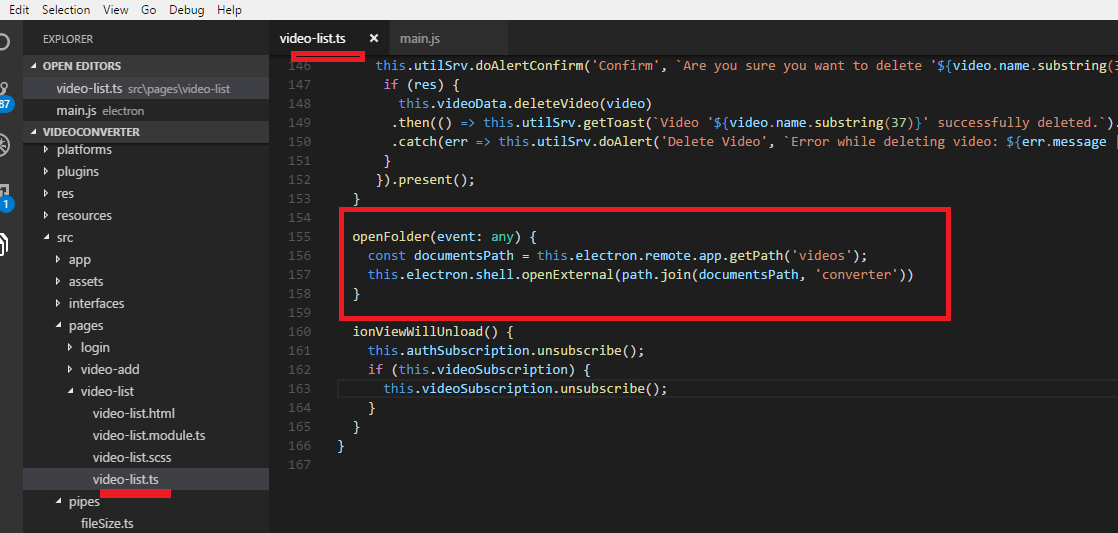


Figure 6

Here we use ngx-electron 'remote' Electron Remote capabilities and 'shell' Electron's screen API to open 'converter' folder on the renderer side.

As already stated we executed a function of the main process inside the renderer process in Electron Framework.

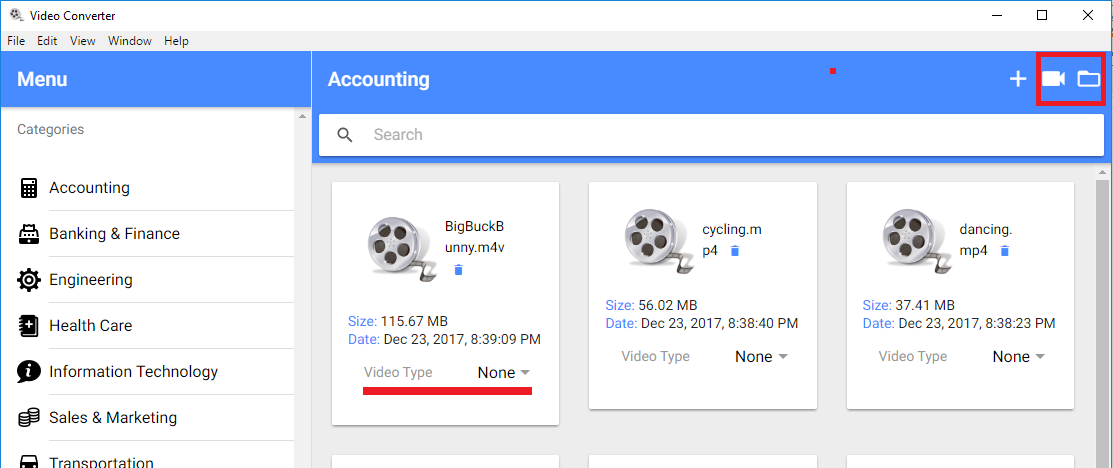


Figure 7

When you open the video converter app inside the web page then some icons and 'Video Type' option are missing because it is not possible

to do video conversion inside the browser.

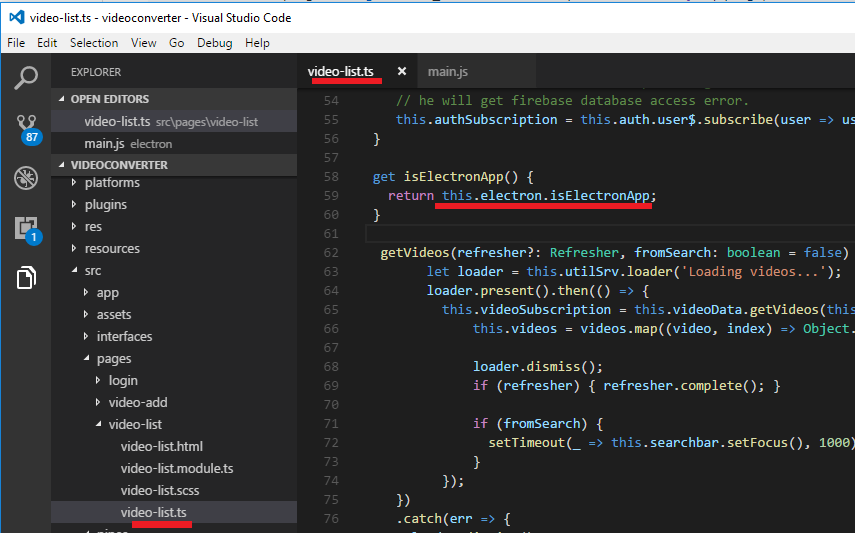


Figure 8

We should somehow detect if the app is running inside the Electron shell or not. It turns out quite simple with the help of ngx-electron by calling

'isElectronApp' instance property indicating if the app is being executed inside of electron or not. Based on it we either show or hide some controls.

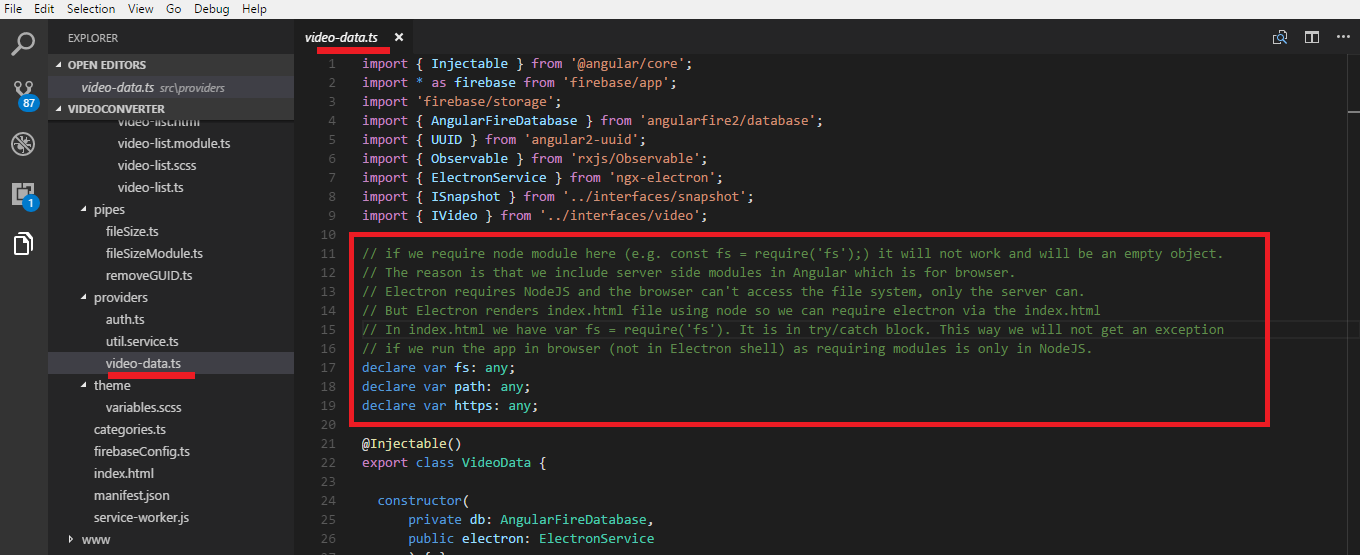


Figure 9

We stated that Electron desktop application can access all node modules like the file system module for handling files, request to make HTTP calls, etc. It can be confusing

though. Please read the comment above.

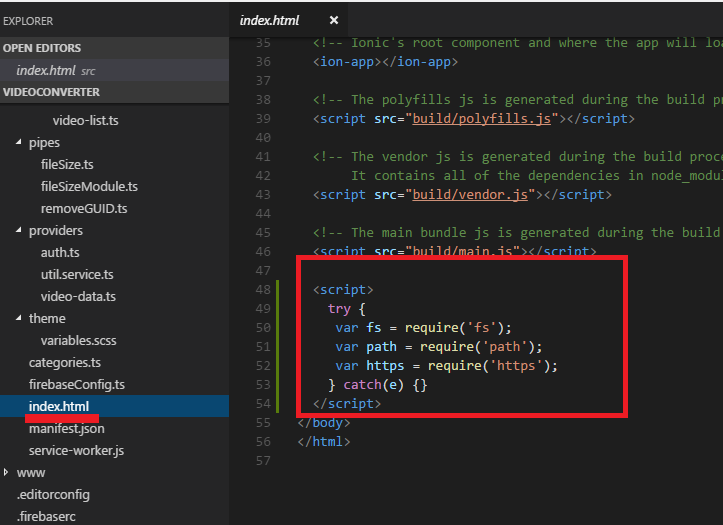


Figure 10

Here is index.html.

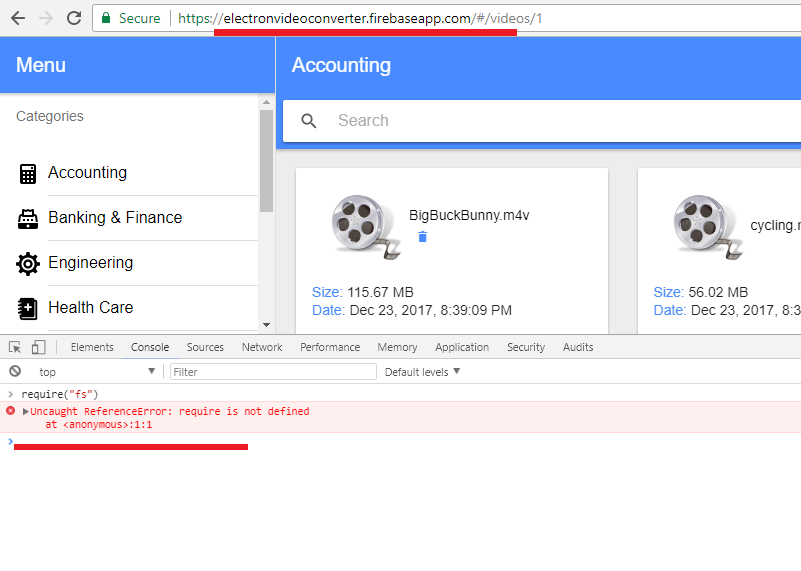


Figure 11

I ran application inside the browser, no Electron shell. I call require('fs ') and require is not defined.

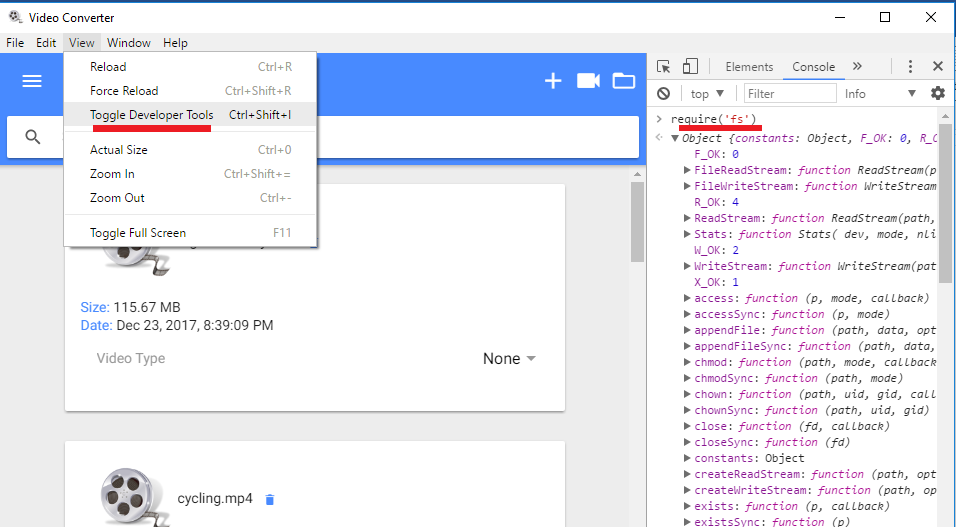


Figure 12

I do the same thing within Electron shell and require('fs') is defined.

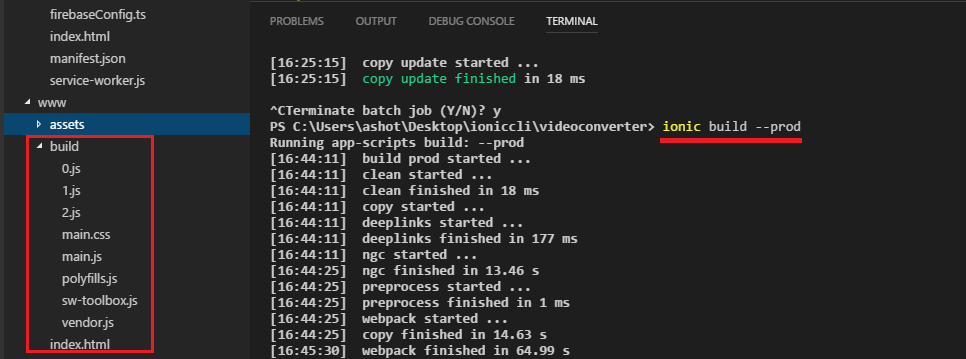


Figure 13

When you develop Electron app with Ionic (Angular) you usually execute 'ionic build —prod' command after you change something in your ionic code.

This is not practical during app development. I stick to another approach when developing Electron/Ionic application.

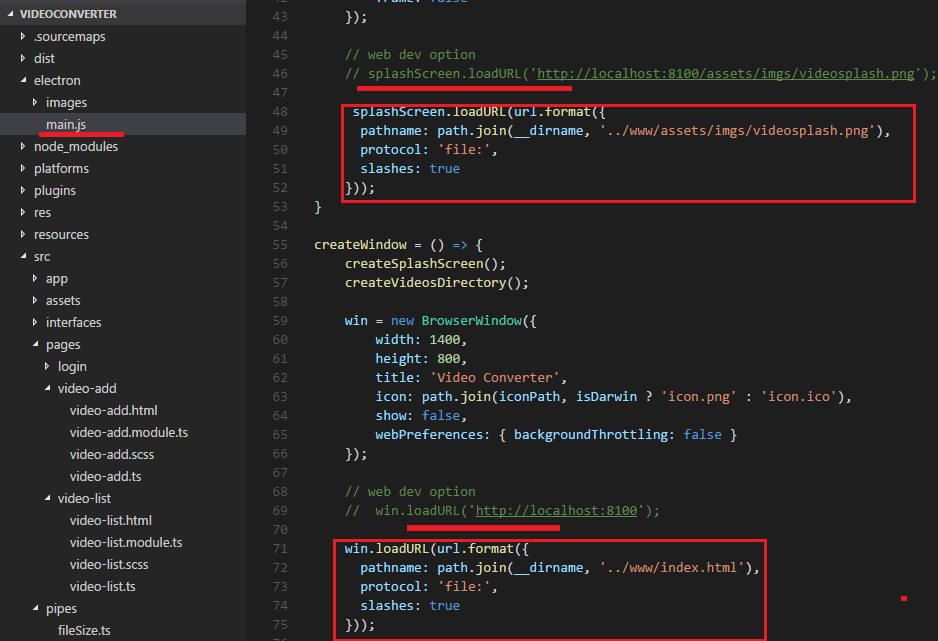


Figure 14

This is Electron main.js file. Electron reads index.html and splash screen file from file path in production.

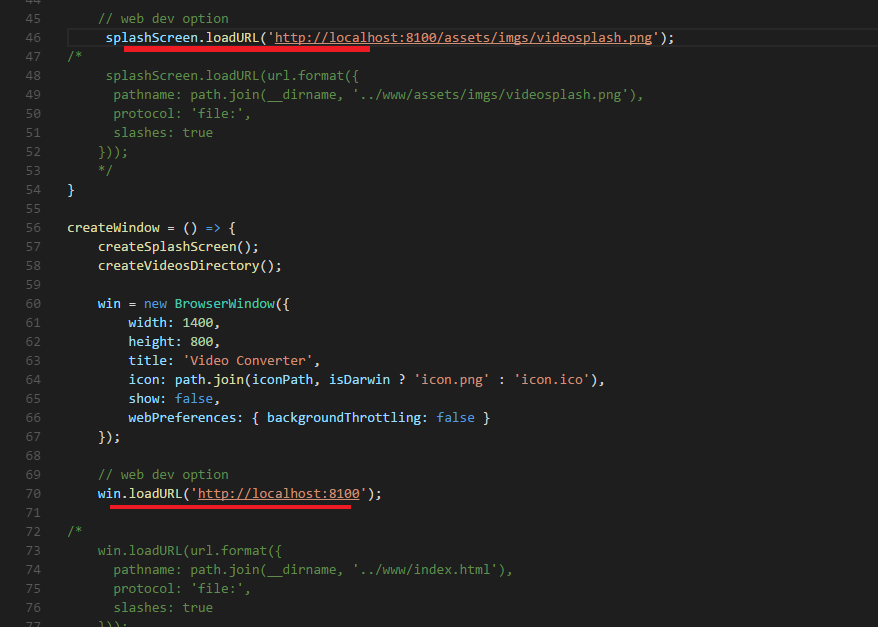


Figure 15

During development I uncomment lines with [http://localhost:8100](http://localhost:8100/) and comment the ones using file path.

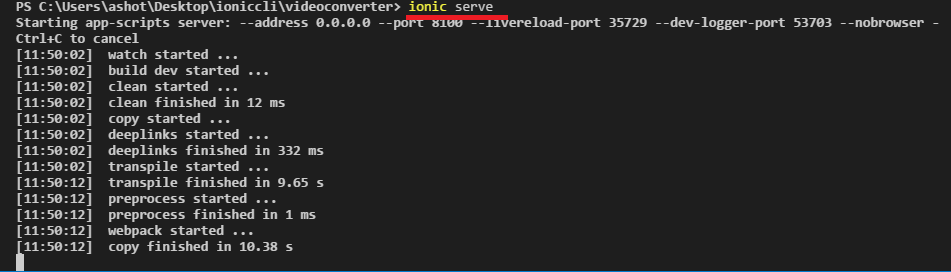


Figure 16

First thing is to install dependencies by running either 'yarn install' or 'npm install'. You need to run the command once only.

Then run 'ionic serve' command.

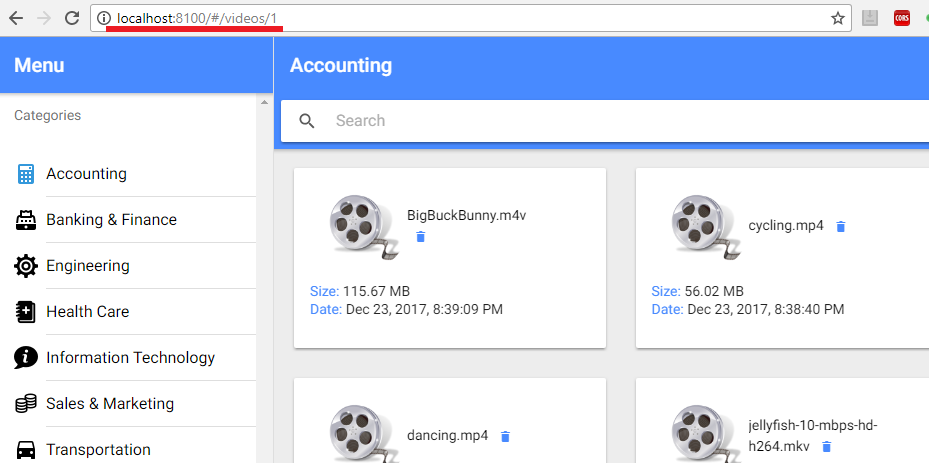


Figure 17

Now If you do a change in ionic code then the change will be automatically loaded inside the Electron shell. 'ionic serve'

watches for changes in your source files and automatically reloads with the updated build.

What about Electron change in mian.js? For that we use 'nodemon' which monitors any changes in your application and automatically

restarts it. If you do a change inside the main.js then your Electron app will be closed immediately and a new one will be opened with the latest change.

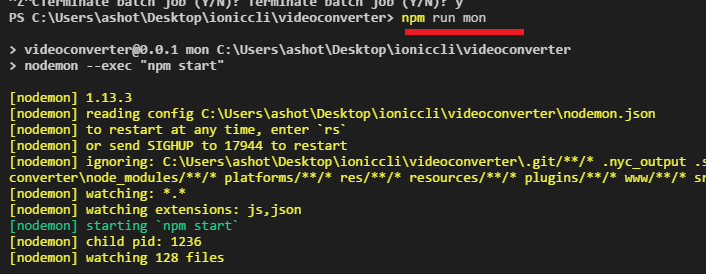


Figure 18

The command is 'npm run mon' to watch changes and restart the electron.

This really speeds up development. Once you developed the app you switch to production mode (Figure 14). It is possible to configure package.json file

to do it more elegantly but I am fine with this approach as I do not do many changes to switch form development mode to production.

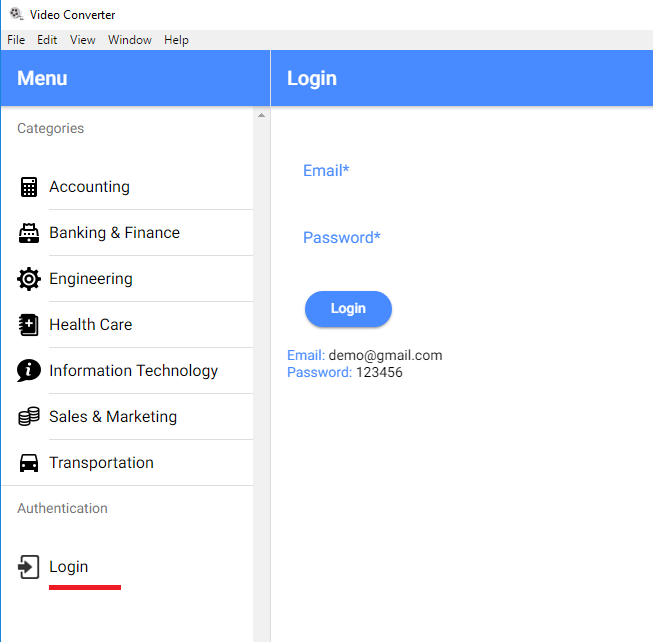


Figure 19

The app uses Firebase password based authentication. It uses Firebase Database and Storage for uploaded videos.

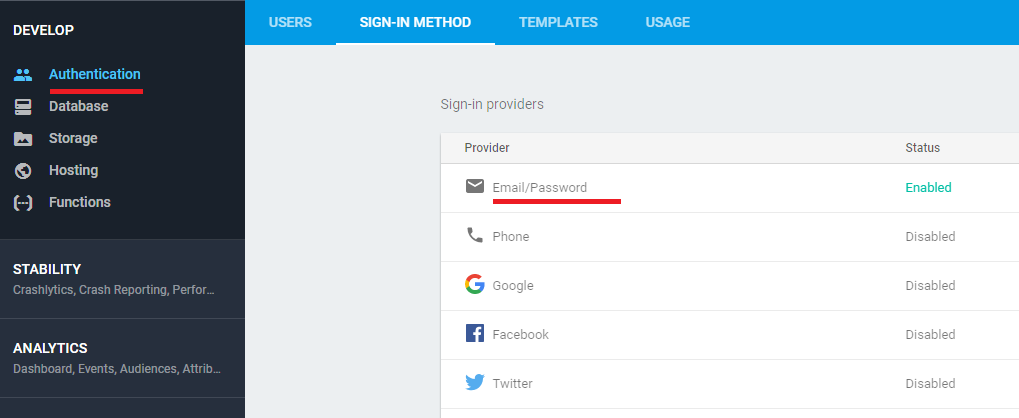


Figure 20

We enabled Email/Password authentication method.

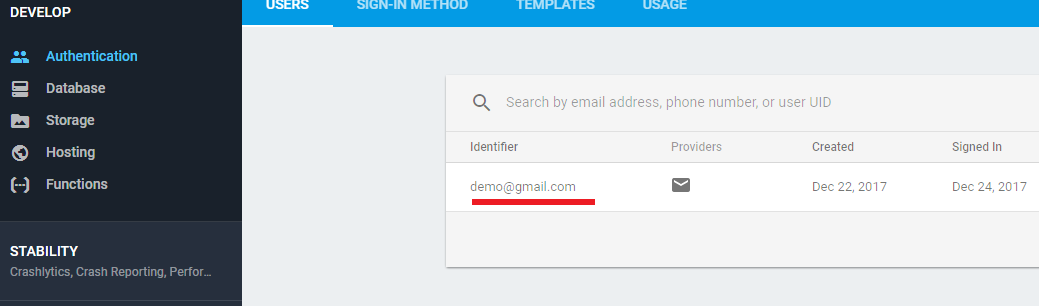


Figure 21

Created [demo@gmail.com](mailto:demo@gmail.com) user.

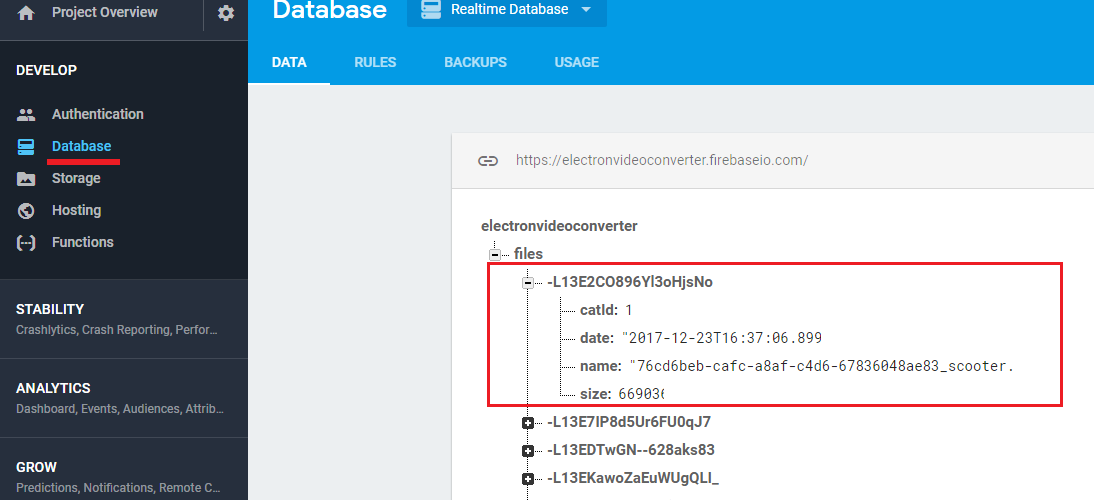


Figure 22

Here is uploaded video information.

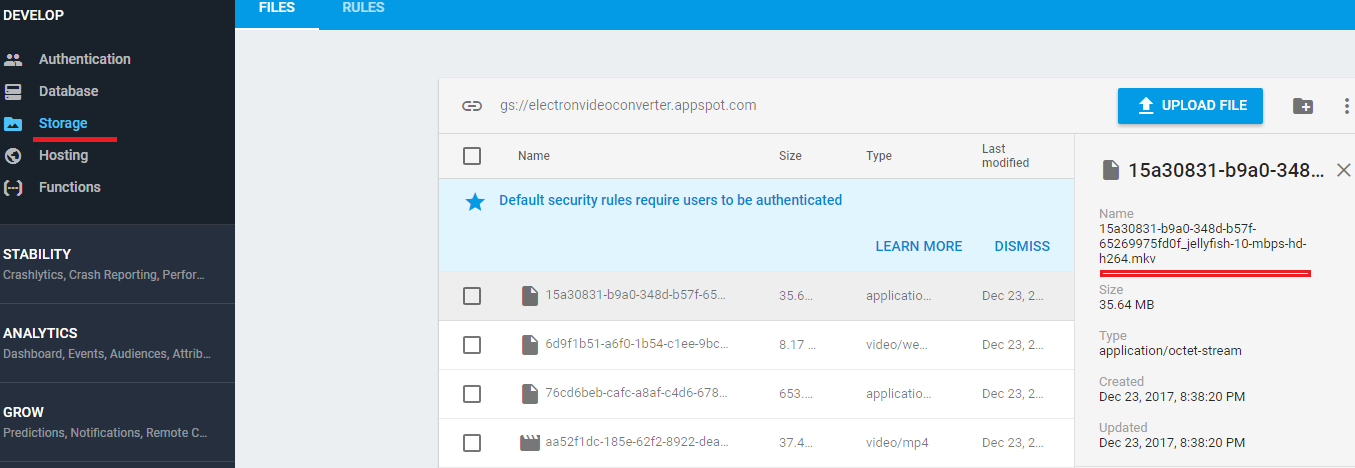


Figure 23

We append GUID prefix to identify each video file in video storage. Users can upload videos with the same name and extension.

'Electron-builder' <https://github.com/electron-userland/electron-builder> is used to package and build a ready for distribution Electron app with 'auto update' support out of box.

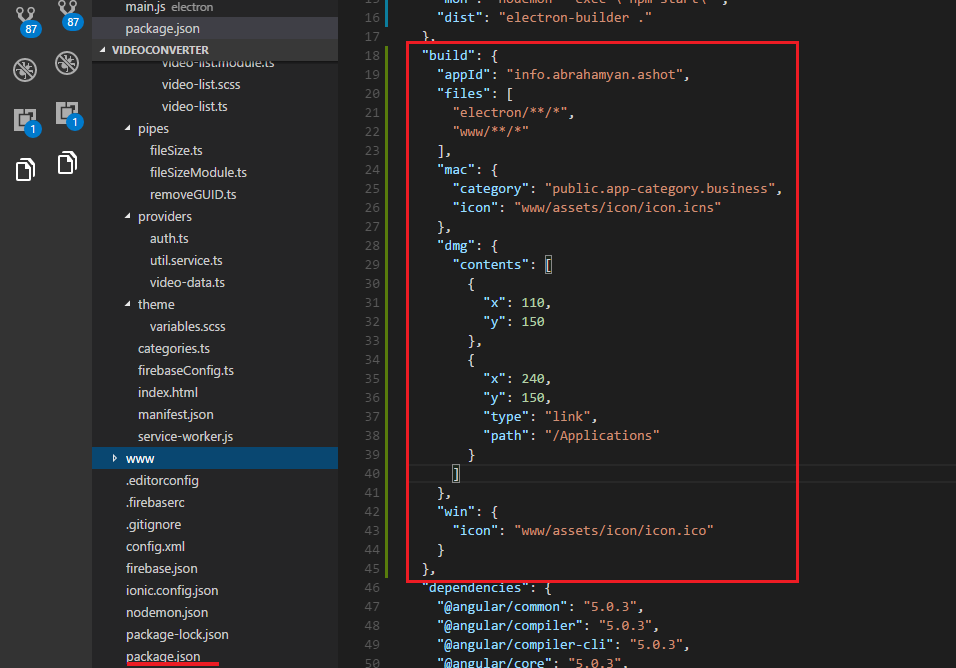


Figure 24

Here is 'build' section configuration. We define entries both for mac and windows. We link our application to macOS 'Applications' (will be discussed below).

After we executed 'ionic build —prod' to generate production build it is time to package it under Windows.

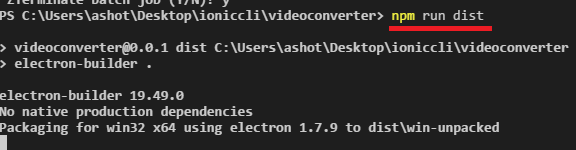


Figure 25

Run 'npm run dist' to generate the package.

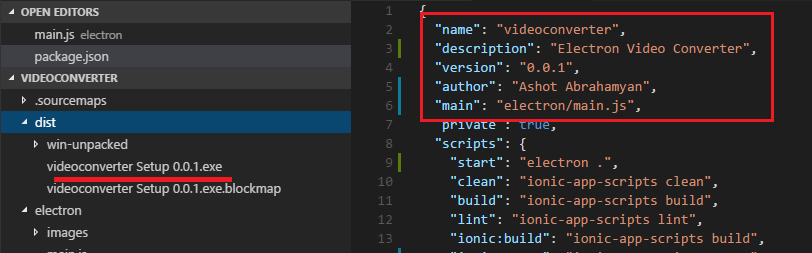


Figure 26

Once the package is generated you will see it under 'dist' directory. It is 'videoconverter Setup 0.0.1.exe' windows installer and size is 51MB.

electron-builder reads package.json entries (name, version) to generate executable name.

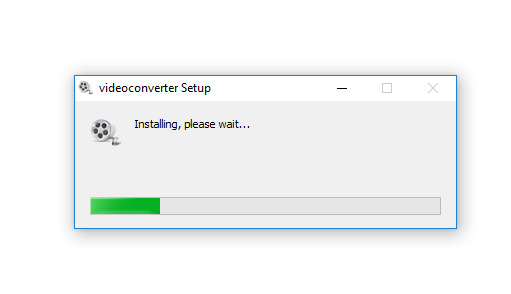


Figure 27

It is an installer and can be installed like other installable apps.

Now it is macOS turn to generate dmg package.

I do 'ionic build —prod' to generate production build this time on mac.

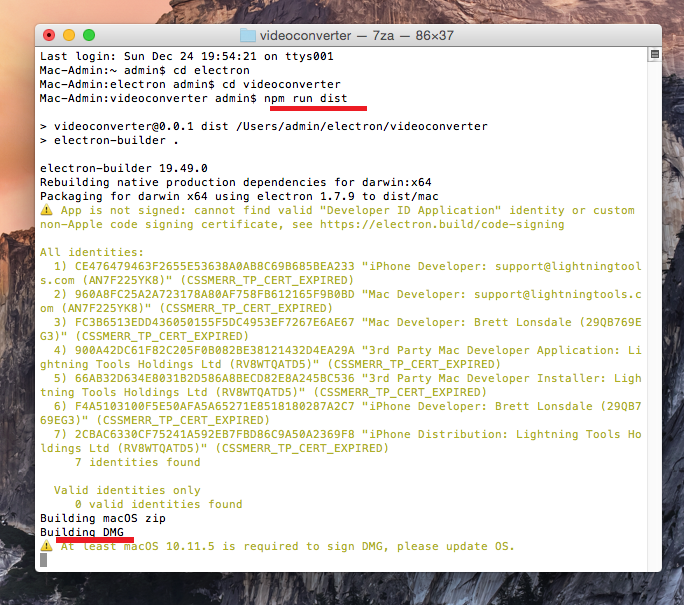


Figure 28

I run 'npm run dist' to generate dmg package.

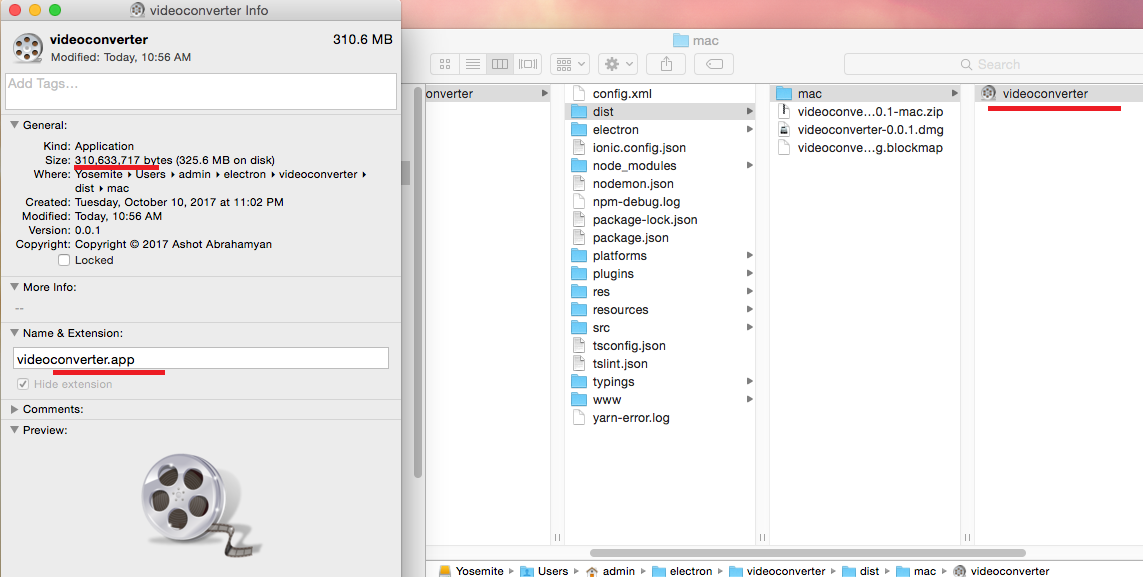


Figure 29

Under 'dist/mac' videoconverter.app is generated and the size is 310MB.

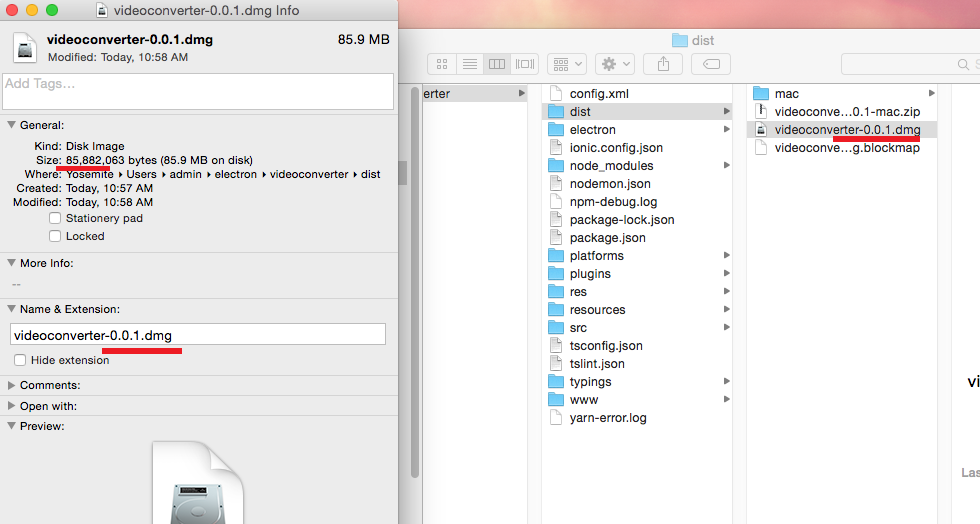


Figure 30

Under 'dist' videoconverter-0.0.1.dmg is generated. It is 86MB.

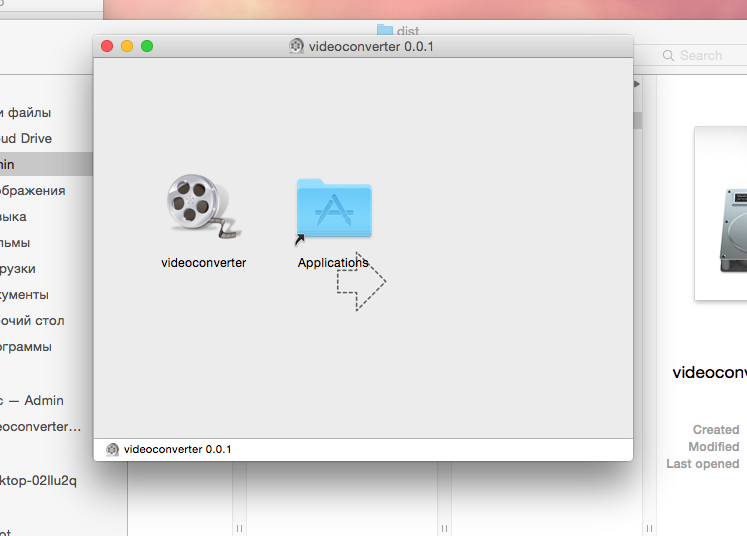


Figure 31

Click videoconverter-0.0.1.dmg file. videoconverter.app wants to be installed in 'Applications' as we linked it to 'Applications' (Figure 24).

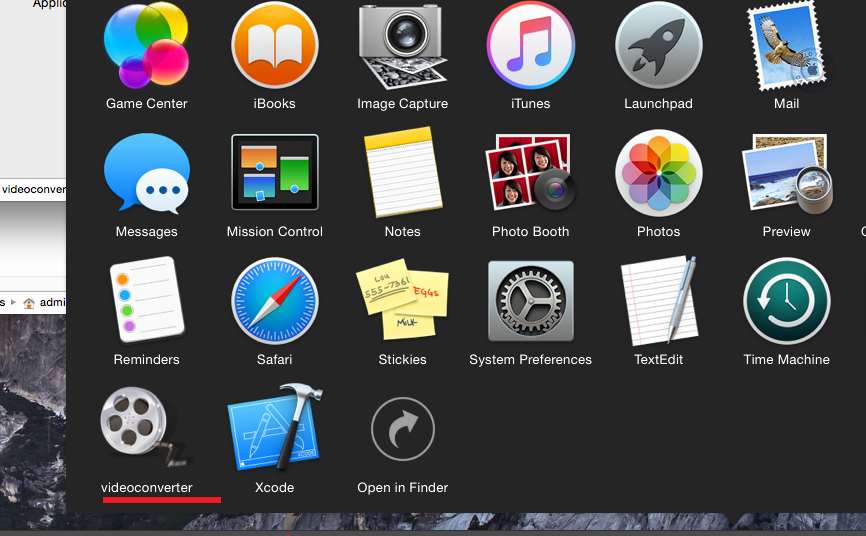


Figure 32

We have it installed in 'Applications'. Click 'videoconverter' app to run our Electron app.