ELECTRON BOOTCAMP

Build cross platform desktop apps with JavaScript, HTML, and CSS

Welcome to Electron Bootcamp by DEV6!

This document is provided to you for recording notes during training sessions with the course instructor. Additionally, it also contains a lab guide. At the end of the course you will know the basics of Electron and how to develop applications using it.

This course covers the following topics:

1. Introduction & Installation
2. Understanding Electron Application Architecture
3. Electron Features
4. Electron API's
5. Electron Userland

COURSE OBJECTIVES

DEV6's Electron Bootcamp Course teaches developers how to use Electron to create modern desktop applications.

COURSE ROADMAP

1. Introduction & Installation
2. Understanding Electron Application Architecture
3. Electron Features
4. Electron API's
5. Electron Userland

STUDENT BACKGROUND

- Fairly good understanding of HTML and CSS.
- Some understanding of JavaScript.
- No prior understanding of Electron.

MODULE 1: INTRODUCTION AND INSTALLATION

DEV6

WHAT IS ELECTRON?

- Open-source library developed by GitHub for building cross-platform desktop applications with HTML, CSS and JavaScript.
- Node integration to grant access to the low level system from web pages.
- Electron accomplishes this by combining Chromium and Node.js into a single runtime and apps can be packaged for Mac, Windows, and Linux.

DEV6

BRIEF HISTORY OF ELECTRON

- 2013 – Electron began as the framework on which Atom was built. Originally it was called as Atom Shell and developed by Cheng Zhao.
- 2014 – Atom and Atom Shell both got open sourced in May 2014.
- 2015 – Atom Shell got renamed to Electron in April 2015.
- 2016 – Electron v1.0.0 released in May 2016.
- 2016 – Electron apps compatible with Mac App Store in May 2016.
- 2017 – Windows Store support for Electron apps in August 2016.
- 2018 – Electron v2.0.0 released in May 2018.
- 2018 – Electron v3.0.0 released in September 2018.
- 2018 – Electron v4.0.0 released in December 2018.

DEV6

WHY DESKTOP APPS INSTEAD OF WEB APPS?

- Constant connectivity with Internet is not required.
- Dock Icon, Desktop shortcuts, Keyboard shortcuts and Native notifications.
- Hosting is cheaper.
- Works with peripheral devices and other local hardware.
- It will be available in App Store.

Desktop apps are making a comeback !

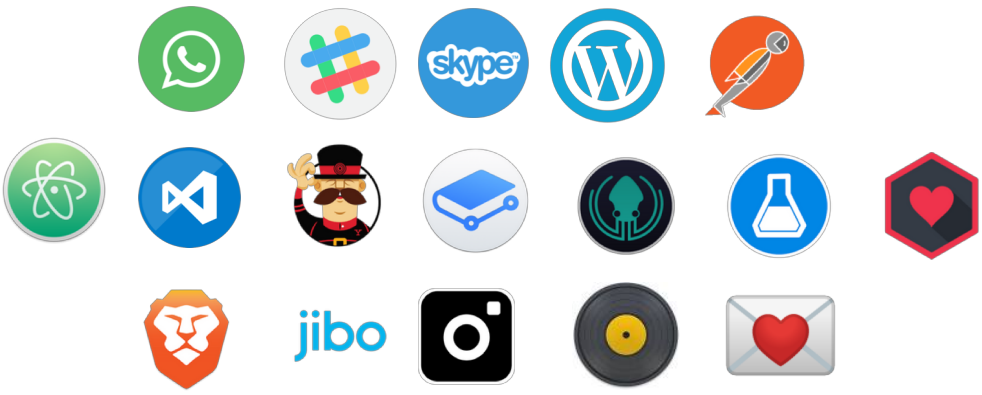
DEV6

WHY USE ELECTRON?

- Single code base and 3 platforms
  - Mac
  - Windows
  - Linux
- Open source
  - Maintained by GitHub team
- Automatic Updates
- Code once, distribute everywhere
- Native Menus & Notifications
- Crash Reporting
- Debugging & Profiling
- Windows Installers
- Deployment and Build Process
  - Electron-Userland

DEV6

APPS BUILT ON ELECTRON



DEV6

INSTALLATION

- Install Electron as a development dependency

```
npm install electron —-save-dev
```

- Global installation

```
npm install electron -g
```

DEV6

EXERCISE

Environment Setup

DEV6

LAB 1: ENVIRONMENT SETUP – GETTING STARTED

In this exercise, you will:

- Install Git
- Install Node.js
- Install Google Chrome
- Install Code Editor Visual Studio Code(open source)/WebStorm(paid)
- Install Electron

Note: These instructions will get your system up and running for Electron development.

Install Git

1. Visit https://git-scm.com/downloads
2. Download and run the installer for your OS.
3. Confirm that Git is installed properly by listing the version number.

```
git --version
```

Install Node.js

For Mac

4. Run the Homebrew installer in a terminal window:

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Note: This will install Node.js as part of the install.

---

5. Confirm your Node.js version number:

```
node -v
```

For Windows:

6. Visit http://nodejs.org
7. Download the latest LTS version for Windows.
8. Install the .msi package
9. Confirm your Node.js version number:

```
node -v
```

Install Google Chrome

10. Visit the http://google.ca/ in your browser and download Chrome.

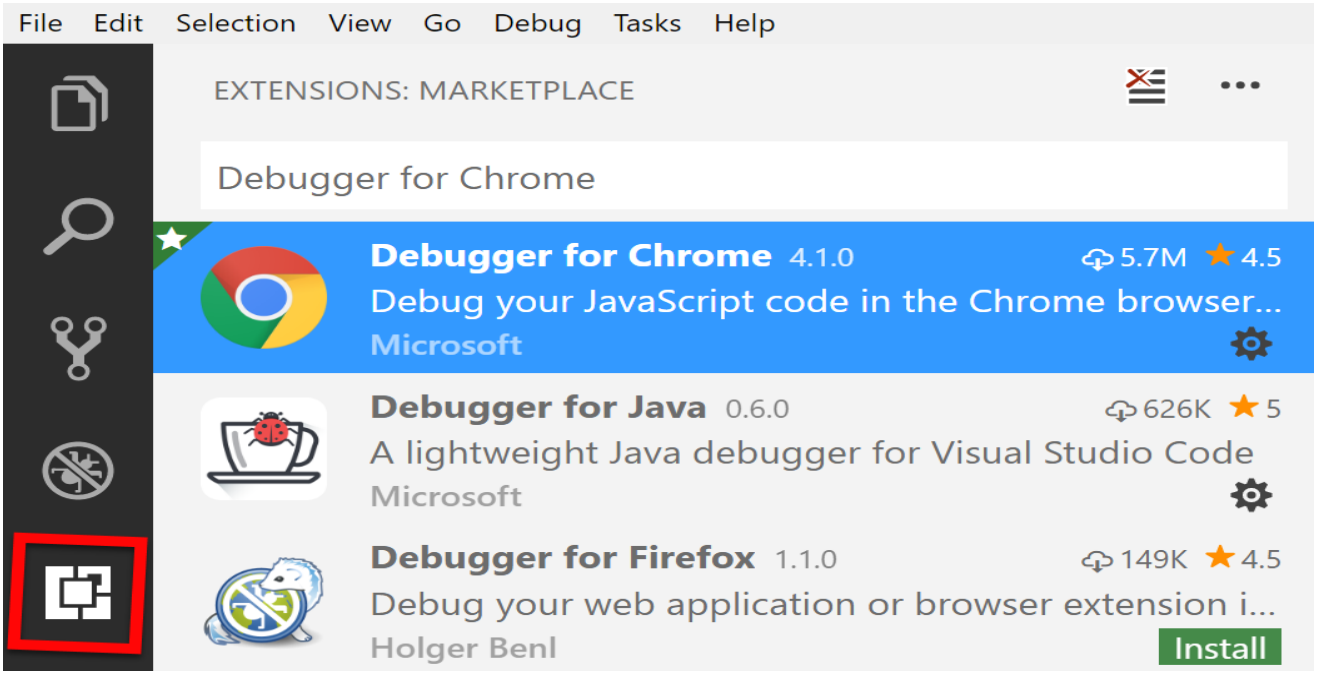
Install Visual Studio Code

11. Visit https://code.visualstudio.com/docs/setup/setup-overview and go to Downloads section and download.

Note: This step will help you set up Visual Studio Code for your system based on OS you are currently using.

---

12. Start Visual Studio Code and open the Extensions viewlet (Ctrl + Shft + X).
13. Search for and install the *Debugger for Chrome* extension.



Note: After the extension is finished installing, you will be prompted to restart the program.

Install Electron

14. Open up a new terminal and type below command.

```
npm install –g electron
```

Install the sample app – Electron Bootcamp

15. In your command line, navigate to the root of your *<course-files>* folder.
16. Use the following git commands to clone the sample app.

```
git clone https://github.com/DEV6hub/Electron-Bootcamp
cd Electron-Bootcamp
git status
```

Note: You should see that your branch is up to date with origin master.

---

17. Install the dependencies that are defined in the project's *package.json* file.

```
npm install
```

18. Start the json-server for serving rest api's from within your project. Open a new terminal window and navigate to JSON Server folder and run below command.

```
json-server --watch signup.json
```

Note: This command will spin up a restful server on default port 3000. You can also verify it by visiting URL http://localhost:3000/userInfo in your browser.

---

19. Similarly, navigate to your project and then JSON Server. Open up new terminal window for each of the below commands. It will spin rest of the api's up and running.

```
json-server --watch tshirts.json --port 4000
json-server –watch contactUs.json –port 5000
```

Note: Verify these URL's in your browser http://localhost:4000/tshirts and http://localhost:5000/contactUs.

---

20. Start the app.

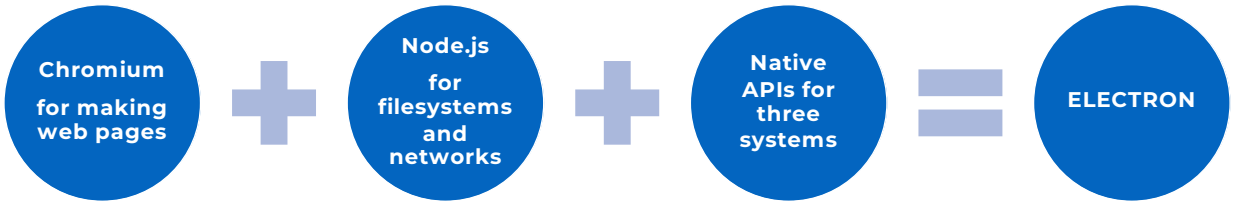
```
npm run start
```

Note: Familiarize with Shirtastic App. Play around, click side navigation links and verify all the links are working fine for you.

MODULE 2: UNDERSTANDING ELECTRON APPLICATION ARCHITECTURE

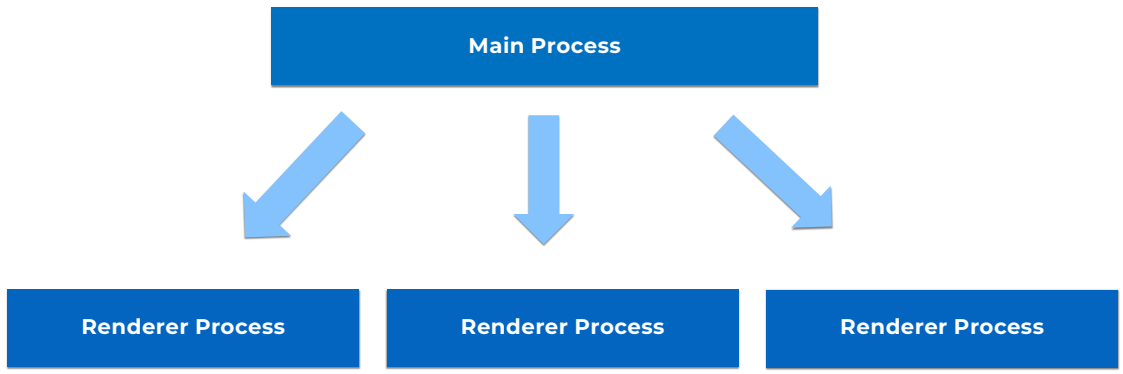
DEV6

ARCHITECTURE



DEV6

ELECTRON UNDER THE HOOD



DEV6

EXERCISE

Converting Existing JavaScript WebApp into Electron Desktop App

DEV6

LAB2: CONVERTING EXISTING JAVASCRIPT WEBAPP INTO ELECTRON DESKTOP APP

In this exercise, you will:

- Set up and run existing web application in the browser.
- Convert web application into working Electron Desktop App.

Set up and run existing web application

1. Check out the `Lab-WebApp-to-ElectronApp` branch from the remote repository.

```
git checkout origin/Lab-WebApp-to-ElectronApp
```

Note: The HEAD of this branch provides the completed code for the entire exercise.

---

2. Run `npm install` to ensure all the dependencies have been installed.
3. Navigate to JSON Server folder in project. Start a new terminal window from within this folder and run below command.

```
json-server --watch --port 4000 tshirts.json
```

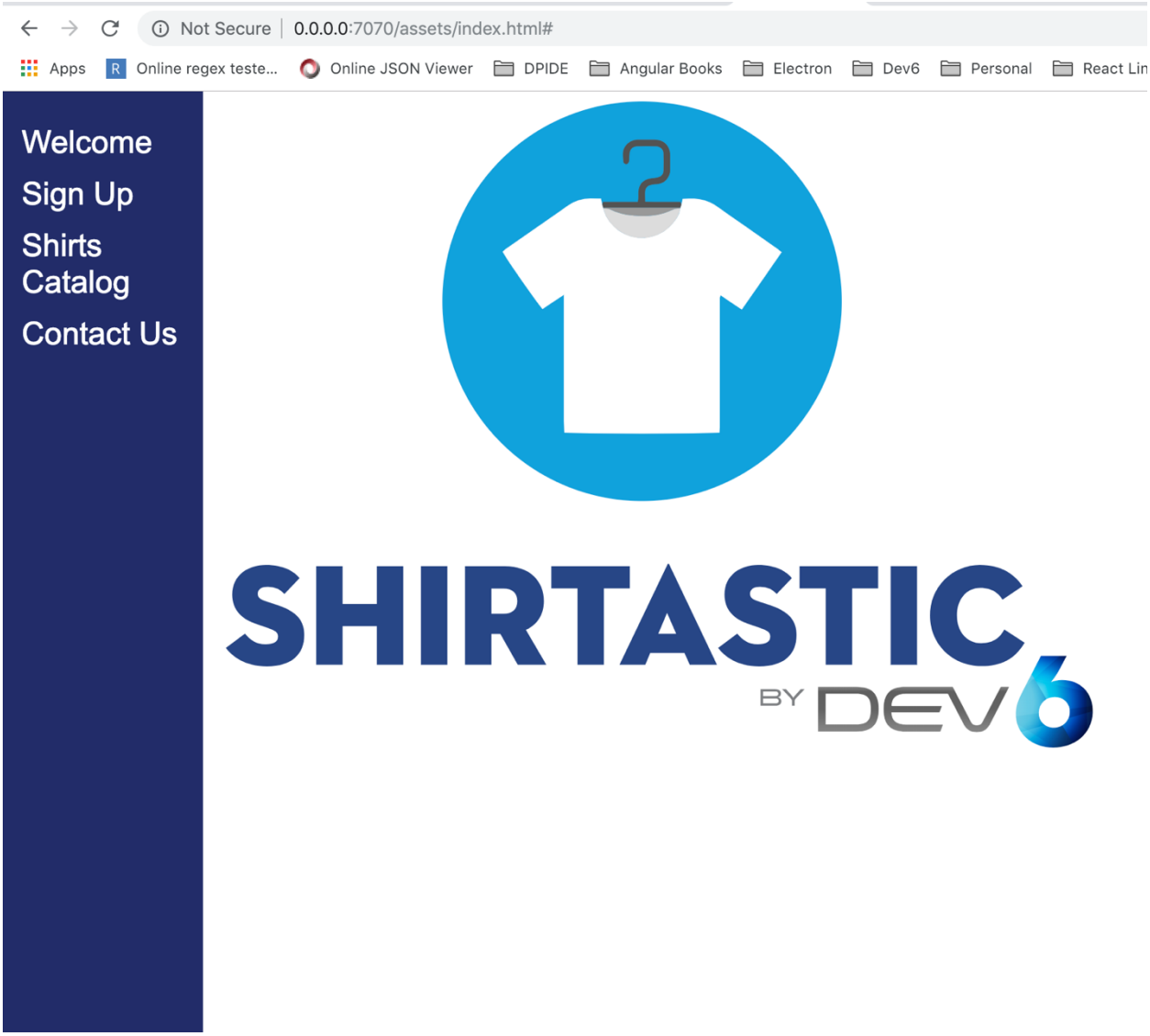
4. Now let's test our web application in the browser. In order to do so you have to run below command in your terminal.

```
npm run start
```

5. Create a new local branch for your work based on the "starting point..." commit.

```
git log --oneline
git checkout <hash for starting point commit>
git checkout -b Lab-WebApp-to-ElectronApp-mine
```

Note: Visit URL http://0.0.0.0:7070/assets/index.html in your browser. You should see the Shirtastic web application. Now, in next series of steps we will convert this web application into working Electron Desktop App.



Convert web application into working Electron Desktop App

6. First step is to install Electron. To do so, run the following command.

```
npm install --save-dev electron
```

```
"dependencies": {
  "light-server": "^2.6.0"
},
"devDependencies": {
  "electron": "^4.0.4"
}
}
```

Note: This command will install Electron in your project repository and save it in dev dependencies section in package.json. There is an interesting thing to note here,

that we have added electron as a dev dependency instead of regular dependency. As we all know, dependencies are required to run, and dev dependency are required to develop. The answer to this variation is that electron is already packaged as part of built output. So, no need of explicitly declaring it as part of dependency. To read more about this you can visit this link.
HTTPS://STACKOVERFLOW.COM/QUESTIONS/50803207/WHY-DOES-ELECTRON-NEED-TO-BE-SAVED-AS-A-DEVELOPER-DEPENDENCY.

---

7. Second step is to create main.js file in the root level of project. You can also refer this link to copy the code for main.js file. Browse to the part where the article talks about main.js content. Just copy paste part which belongs to main.js file into your newly created main.js file.

   https://electronjs.org/docs/tutorial/first-app

8. Once you have pasted the code from this link to main.js file. Now it is time to edit the path for index.html.

```
// and load the index.html of the app.
win.loadFile('assets/index.html')
```

Note: Main.js will be the entry point for application. Main.js will create a browser window which will run the index.html and rest of our old web application. So, it makes sense to tell main.js from where to look up index.html in the project structure.

---

9. Final step is to edit package.json and add a new script to start electron desktop app. Add new entry namely startElectron in existing scripts as shown below in the snippet.

```
"scripts": {
  "startWebApp": "light-server -s . -p 7070 -w \"assets/app/**/*.js, assets/screens/**/*.html, assets/css/**/*.css, assets/index.html, assets/app.js\"",
  "startElectron": "electron ."
},
```

10. And, add main field to package.json.

```
{
  "name": "shirtastic-javascript-app",
  "version": "1.0.0",
  "description": "Shirtastic Javscript App",
  "main": "main.js",
  "scripts": {
    "start": "light-server -s . -p 7070 -w \"assets/app/**/*.js, assets/sc
    "startElectronApp": "electron ."
  },
  "keywords": [],
  "author": "gagank@dev6.com",
  "license": "ISC",
  "dependencies": {
    "light-server": "^2.6.0"
  },
  "devDependencies": {
    "electron": "^4.0.4"
  }
}
```

Note: Electron runtime looks for main field entry in order to start the electron app.

11. Just one last quick fix in the index.html before we can test our brand-new Electron app. Navigate to index.html under assets folder and add below script tag just before the closing of body tag.

```
<script>window.$ = window.jQuery = require('./lib/jquery.min.js');</script>
```

Note: This fix is not related to Electron usage. It is done to resolve the jQuery dependency in electron app, the way project structure is organised. Without this script the electron app gives error related to jQuery usage.

12. Now time for the moment of truth. Yes, let's test the electron app. Go to terminal window and run the command.

```
npm run startElectronApp
```

Note: You should see shiny new Electron Desktop App running on your machine now.

MODULE 3: ELECTRON FEATURES

DEV6

SUPPORTS THREE PLATFORMS

- Code once, distribute everywhere.
- Reusability

DEV6

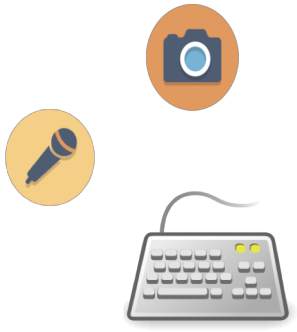
CODE AND APP MANAGEMENT

- Single requirement document
- Single team
- Single code base
- Cost and time

DEV6

LOW-LEVEL/HARDWARE ACCESSIBILITY

- Keyboard Shortcuts
- Low-level accessibility to hardware and operations system components



DEV6

NATIVE MENUS/NOTIFICATIONS

- Native menus
- Native notifications


```
let myNotification = new Notification('Title', {
  body: 'Lorem Ipsum Dolor Sit Amet'
})

myNotification.onclick = () => {
  console.log('Notification clicked')
}
```

https://electronjs.org/docs/tutorial/notifications

DEV6

CRASH REPORTING

- Helps you submit crash reports to a remote server

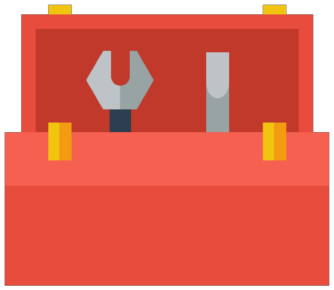
```
const { crashReporter } = require('electron')

crashReporter.start({
  productName: 'YourName',
  companyName: 'YourCompany',
  submitURL: 'https://your-domain.com/url-to-submit',
  uploadToServer: true
})
```

https://electronjs.org/docs/api/crash-reporter

DEV6

DEBUGGING AND PROFILING

- Collects tracing data from Chromium's content module for finding performance blocks and slow operations.
- Debug Renderer Process using Chromium Developer Toolset.
- Debug Main Process using Command Line Switches or External Debuggers.



DEV6

SECURITY AND YOUR RESPONSIBILITY

- Data stays locally in system as it is a desktop application.
- Follow security recommendations by Electron team to ensure your desktop application is safe and protected from attackers.

DEV6

AUTOMATIC UPDATES

- Built-in Squirrel framework and autoUpdater module.
- Electron apps can self update using free and open source webservice update.electronjs.org maintained by GitHub's Electron team.
- Using electron-builder, you can allow updates from any static file host.
- If GitHub is not being used for publishing releases, then a dedicated update server can be used.
- Currently there is no built-in support for auto-updater on Linux. It is recommended to make use of distribution's package manager to update the app.

DEV6

EXERCISE

Electron Development Tools and Features

DEV6

LAB 3: ELECTRON DEVELOPMENT TOOLS

In this exercise, you will:

- Set up electron-reload for live reload.
- Set up devtron for debugging.
- Set up electron-is-dev to check if electron is running in development mode.
- Set up electron-log for logging.

1. Check out the `Lab-Electron-Development-Tools` branch from the remote repository.

```
git checkout origin/Lab-Electron-Development-Tools
```

Note: The HEAD of this branch provides the completed code for the entire exercise.

---

2. Run `npm install` to ensure all the dependencies have been installed.
3. Create a new local branch for your work based on the "starting point..." commit.

```
git log --oneline
git checkout <hash for starting point commit>
git checkout -b Lab-Electron-Development-Tools-mine
```

4. Start the app.

```
npm run start
```

electron-reload

5. Install electron-reload.

```
npm install --save-dev electron-reload
```

6. Navigate to main.js at the root level of project and add below code just under the first line. Save your file.

```
const path = require('path')

require('electron-reload')(__dirname, {

  electron: path.join(__dirname, 'node_modules', '.bin', 'electron')

});
```

7. Restart the app.

```
npm run start
```

Note: Now you can check your app for live reload by changing any of the files under assets folder. And you will observe electron app reloads itself. You can read more about its usage by visiting this URL **HTTPS://WWW.NPMJS.COM/PACKAGE/ELECTRON-RELOAD**

---

devtron

8. Install devtron.

```
npm install --save-dev devtron
```

9. Navigate to main.js at the root level of project. Add below lines of code inside createWindow function.

```
require('devtron').install()
```

Note: You can add above line of code anywhere inside createWindow function.

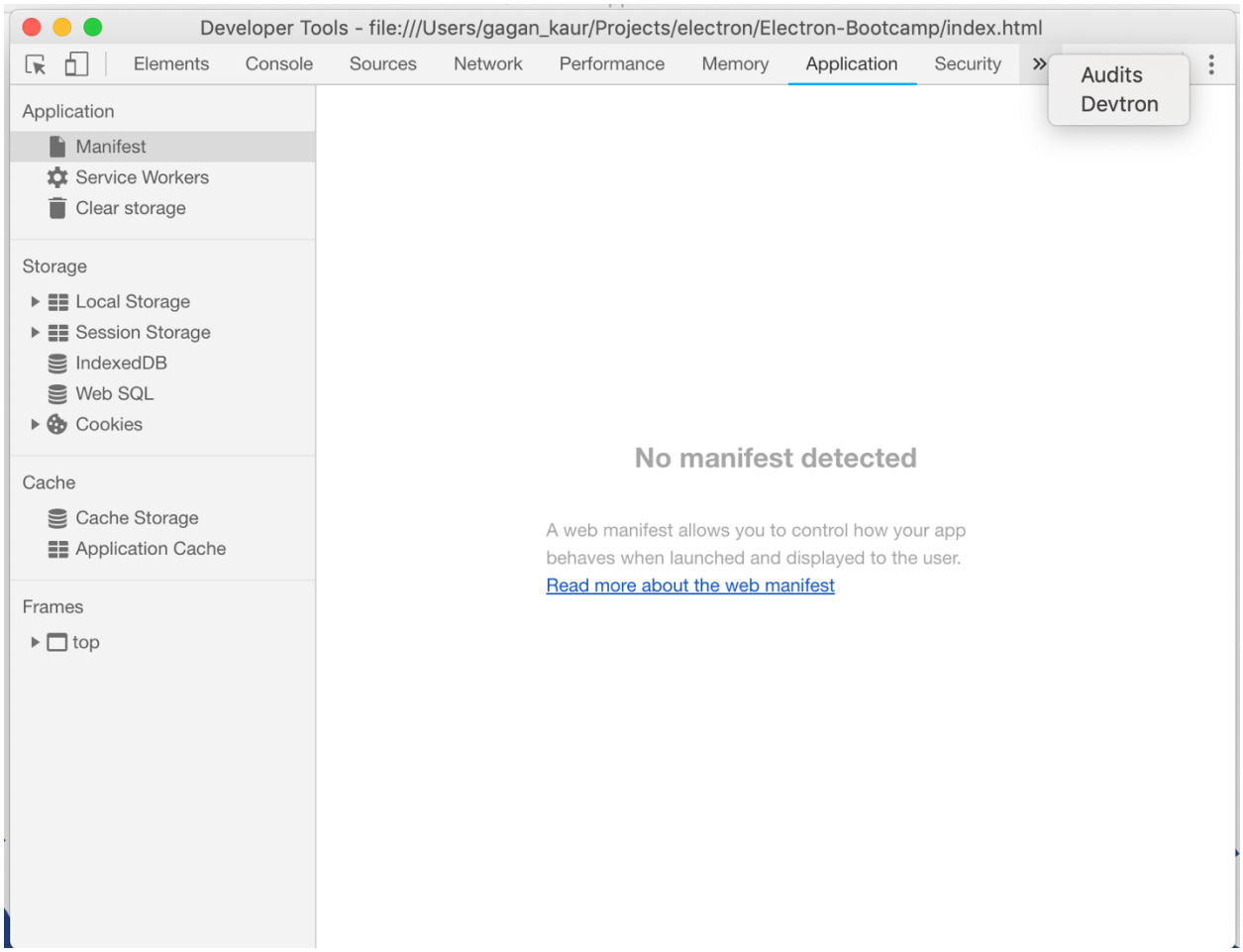
---

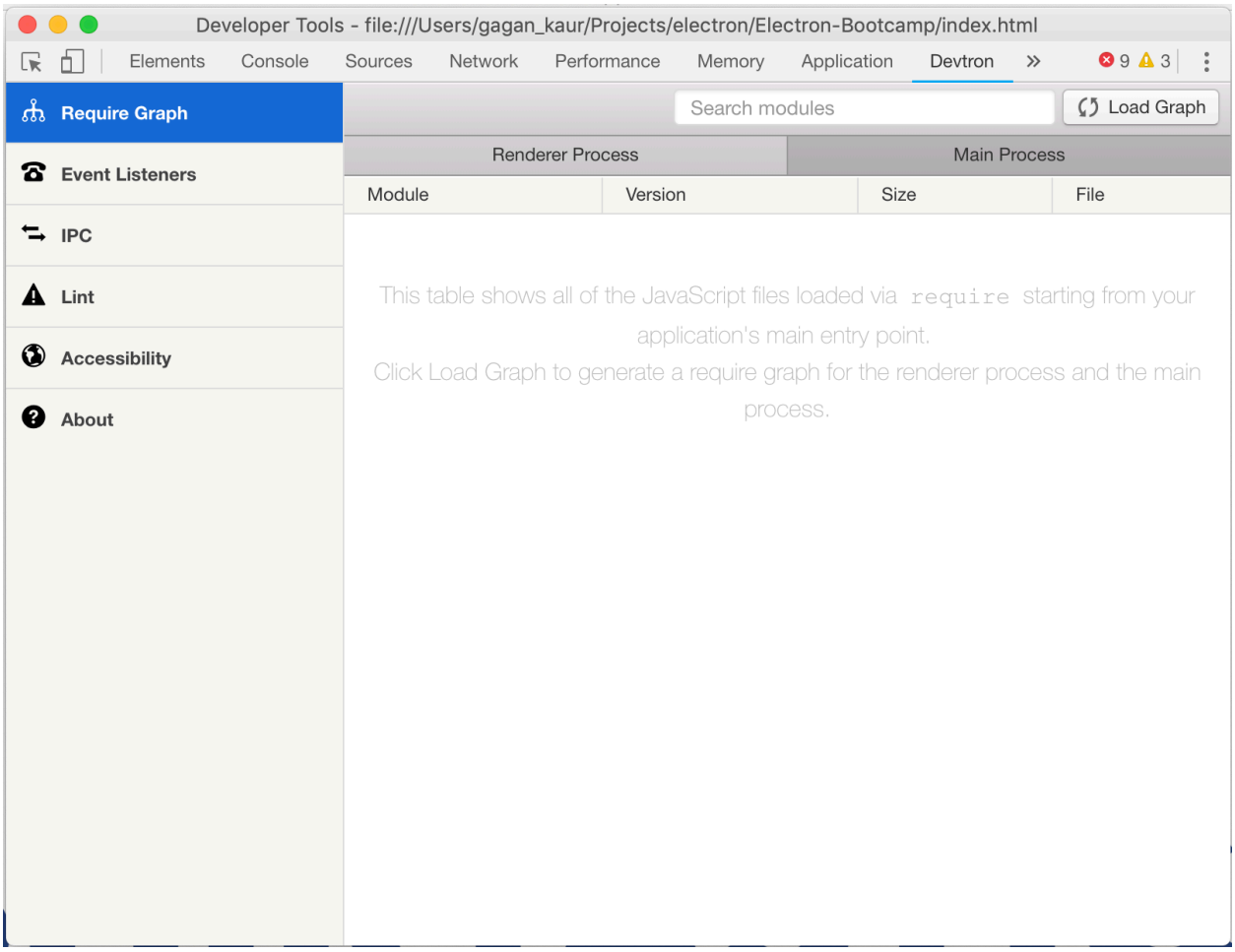
10. Restart the app.

```
npm run start
```

Note: If you notice there is a line of code inside createWindow function win.webContents.openDevTools(). This opens the default developer tools for electron app. Now in addition to this we will get Devtron installed and ready to be used for

more detailed analysis of electron app. You can read more about its usage by visiting this URL **HTTPS://ELECTRONJS.ORG/DEVTRON**





electron-is-dev

11. Install electron-is-dev.

```
npm install --save electron-is-dev
```

12. Navigate to main.js file at the root level of project. Add below line of code at the top level just under first line.

```
const isDev = require('electron-is-dev');
```

13. Now you can use isDev variable in the main.js file. This variable can be used to place conditions for code which needs to be run only in development mode.
14. For example, we can place a check in front of live reload code. So, this code does not run in the production version of the app.

```
if(isDev) {

  require('electron-reload')(__dirname, {

    electron: path.join(__dirname, 'node_modules', '.bin', 'electron')

  });

}
```

15. Another important usage can be placing a check against dev tools and devtron debugger tool. We do not want the end users to see dev tools open in the production version.

```
if(isDev) {

      // Open the DevTools.

      win.webContents.openDevTools()

      // Devtron for debugging application

      require('devtron').install();

    }
```

Note: You can read more about its usage by visiting this URL
HTTPS://WWW.NPMJS.COM/PACKAGE/ELECTRON-IS-DEV

---

electron-log

16. Install electron-log.

```
npm install --save electron-log
```

17. Navigate to main.js file at the root level of project. Add below line of code at the top level just under first line.

```
const log = require('electron-log');
```

18. Now you can make use of log variable to place logging information in various parts of your application. For example, we can place a log inside createWindow function.

```
log.info('window creation');
```

19. Restart the app.

```
npm run start
```

Note: This will generate logs in the terminal where your app is running.

LAB 4: ELECTRON FEATURES

In this exercise, you will:

- Add native menu
- Add native notifications
- Add new browser window
- Perform File Read operation and IPC communication from main process to renderer process
- Use axios library for http requests

1. Check out the `Lab-Electron-Features` branch from the remote repository.

```
git checkout origin/Lab-Electron-Features
```

Note: The HEAD of this branch provides the completed code for the entire exercise.

---

2. Run `npm install` to ensure all the dependencies have been installed.
3. Create a new local branch for your work based on the "starting point..." commit.

```
git log --oneline
git checkout <hash for starting point commit>
git checkout -b Lab-Electron-Features-mine
```

4. Start the app.

```
npm run start
```

Add native menu

5. Create a reference to the Menu class from the electron package. Use below line of code to replace the old first line of code in main.js file (its present at the root level of project structure).

```
const { app, BrowserWindow, Menu } = require('electron')
```

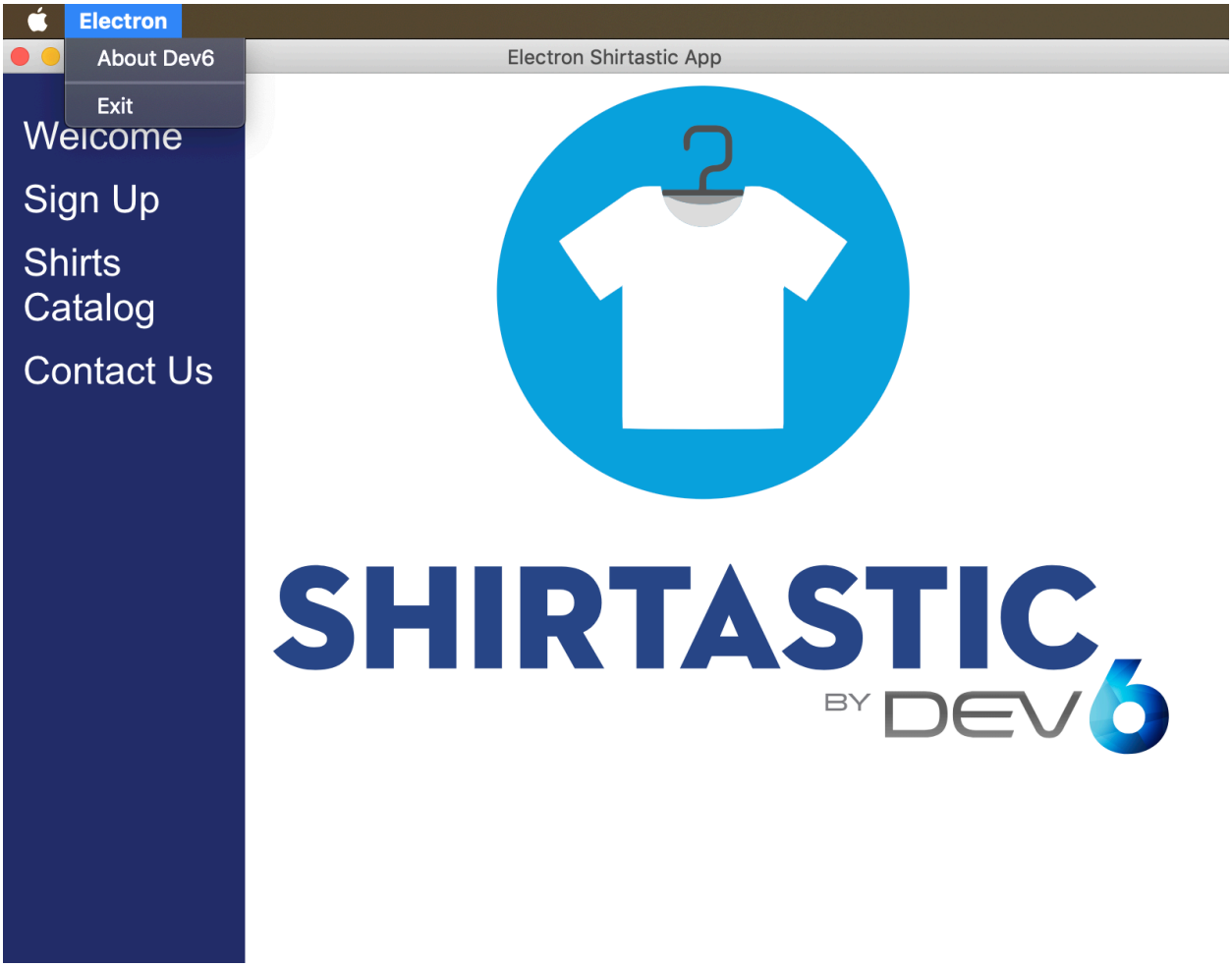
6. Next step is to declare Menu and add it to existing code of createWindow function inside main.js file. Place below code just below win.on('closed') event handler.

```
var menu = Menu.buildFromTemplate([

    {

        label: 'Menu',

        submenu: [

            {

                label: 'About Dev6',

                click() {

                    console.log('clicked');

                }

            },

            {type: 'separator'},

            {

                label: 'Exit',

                click() {

                    app.quit()

                }

            }

        ]

    }

])
```

```
Menu.setApplicationMenu(menu);
```

7. Restart the app.

```
npm run start
```



Note: Now you can check your app has native menus on screen.

8. Now let's add some functionality to About Dev6 menu item. We will use Shell module, to open external links. Place below line of code in main.js file just under previous declarations done.

```
const shell = require('electron').shell
```

9. Next step is to use openExternal function from shell module. Observe in below screenshot how we used openExternal function to open an external link.

```
var menu = Menu.buildFromTemplate([
  {
    label: 'Menu',
    submenu: [
      {
        label: 'About Dev6',
        click() {
          shell.openExternal('https://www.dev6.com/')
        }
      },
      {type: 'separator'},
      {
        label: 'Exit',
        click() {
          app.quit()
        }
      }
    ]
  }
])
  Menu.setApplicationMenu(menu);
}
```

10. Restart the app.

```
npm run start
```

Note: Now you will notice when you click on About Dev6, it opens the URL in user's default browser. Try clicking on Exit menu link and observe how it kills the app.

Add native notifications

11. Let's add native notification for successful signup. Open file signup.js under assets/js folder. Add reference to path module as top line.

```
const path = require('path');
```

12. Then place below line of code in init function.

```
const notification = {
        title: 'Signup Alert',
        body: 'Welcome to Dev6',
        icon: path.join(__dirname, '../images/TShirt.png')
```

```
        }
```

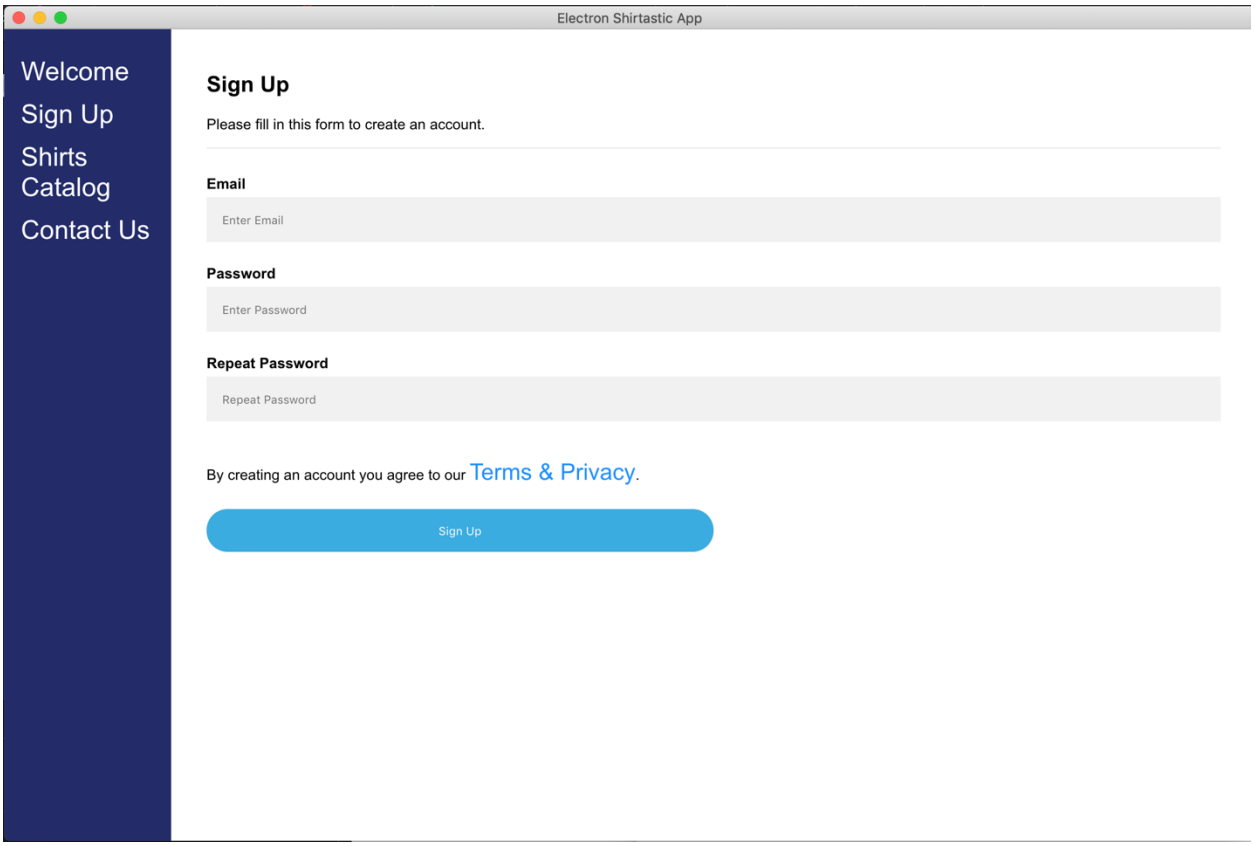
13. Last step is to display notification. This is done by placing below line of code inside success handler of axios call. Please refer to screenshot below to know the exact location for placement of this line of code.

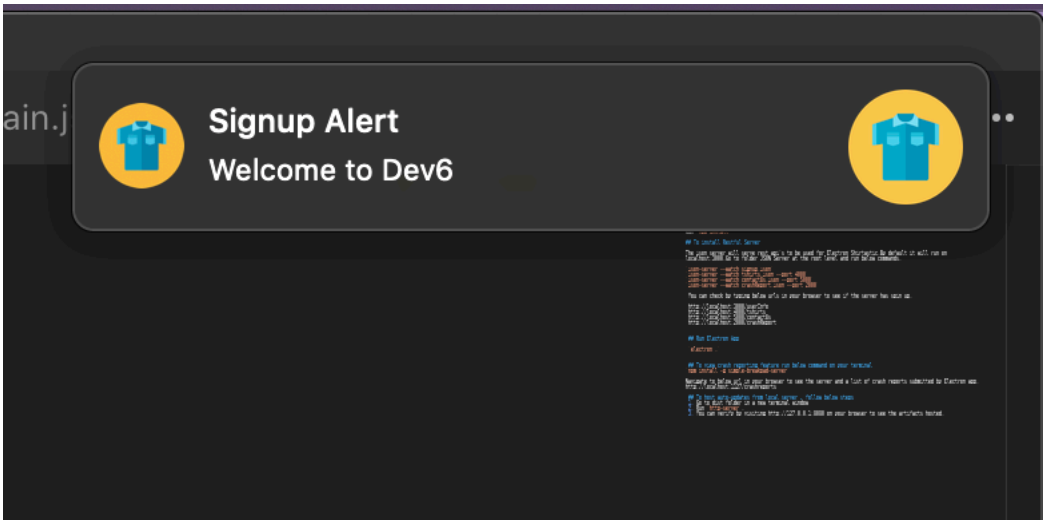
```
const myNotification = new window.Notification(notification.title,
notification);
```

14. Restart the app.

```
npm run start
```

Note: Now as soon you perform the signup in Signup screen, you will see a native alert saying Signup Alert. Please refer below screenshots to know which screen to test and the look of native notification on macOS.





Add new browser window

15. So, in next series of steps, we will add functionality of opening a new browser window for Contact Us link. In order to do so, lets create a new js file under assets/js folder named as new-browser-window.js. Place below line of code in this file. The main hero in this entire set of code is electron.remote.BrowserWindow which is providing the capacity

to open a new browser window for us. You can read more about this by visiting this URL https://electronjs.org/docs/api/browser-window.

```
const path = require('path');

const electron = require('electron');

const BrowserWindow = electron.remote.BrowserWindow;

const customerFeedbackBtn = document.getElementById('customer-feedback');

customerFeedbackBtn.addEventListener('click', function (event) {

    const modalPath = path.join('file://', __dirname, '../screens/customer-feedback.html');

    let win = new BrowserWindow({ width: 800, height: 700, frame: true });

    // // Open the DevTools.

    win.webContents.openDevTools()

    win.loadURL(modalPath);

    win.show();

});
```

Note: Here in above line of code, we created a new browser window on the click on ContactUs link and displayed the content of customer-feedback.html.

Perform File Read Operation and IPC communication from main process to renderer process

16. In order to perform file operations, we will require the help of ipcMain and ipcRenderer modules for communications between main process and renderer process. Let's start with the set up. Open file customer-feedback.js under assets/js folder. Paste below lines of code.

```
selectDirBtn.addEventListener('click', (event) => {

    event.preventDefault();

  ipcRenderer.send('open-file-dialog')

})



ipcRenderer.on('selected-directory', (event, path) => {

  document.getElementById('selected-file').innerHTML = `You selected:
${path}`

})
```

Note: The renderer process is all set to communicate and send messages to main process.

17. Now, open main.js (which is present at the root level). And paste below lines of code to send and receive messages from customer-feedback.js (renderer process). You can paste these lines anywhere in main.js file.

```
const {ipcMain, dialog} = require('electron')

ipcMain.on('open-file-dialog', (event) => {

  dialog.showOpenDialog({

    properties: ['openFile', 'openDirectory']

  }, (files) => {

    if (files) {

      event.sender.send('selected-directory', files)

    }
```

```
  })

})
```

Note: Below are the screenshots for entire code of customer-feedback.js and main.js after performing above steps. Please go through these screens and placement of code, in case any of the instructions is not clear.

Customer-feedback.js

```
const axios =require('axios');

const {ipcRenderer} = require('electron');

const selectDirBtn = document.getElementById('attach-file');

selectDirBtn.addEventListener('click', (event) => {
    event.preventDefault();
  ipcRenderer.send('open-file-dialog')
})

ipcRenderer.on('selected-directory', (event, path) => {
  document.getElementById('selected-file').innerHTML = `You selected: ${path}`
})

submitBtn.addEventListener('click', (event) => {
  event.preventDefault();
  let formData = new FormData(document.querySelector('form'));
  let userInfo = {
    firstname: formData.get('firstname'),
    lastname: formData.get('lastname'),
    attachedFile: formData.get('selected-file'),
    subject: formData.get('subject')
  };
  axios({
    method: 'post',
    url: 'http://localhost:5000/contactUs',
    data: userInfo,
    })
    .then(function (response) {
        //handle success
        console.log(response);
        alert('Thanks for your feedback.We will get in touch soon...');

    })
    .catch(function (response) {
        //handle error
        console.log(response);
    });
})
```

Main.js

```
const { app, BrowserWindow, Menu } = require('electron')
const log = require('electron-log');
const isDev = require('electron-is-dev');
const path = require('path')
const shell = require('electron').shell
const {ipcMain, dialog} = require('electron')

ipcMain.on('open-file-dialog', (event) => {
  dialog.showOpenDialog({
    properties: ['openFile', 'openDirectory']
  }, (files) => {
    if (files) {
      event.sender.send('selected-directory', files)
    }
  })
})
```

Note: You can read more about IPC by visiting this URL https://electronjs.org/docs/api/ipc-main and https://electronjs.org/docs/api/ipc-renderer

---

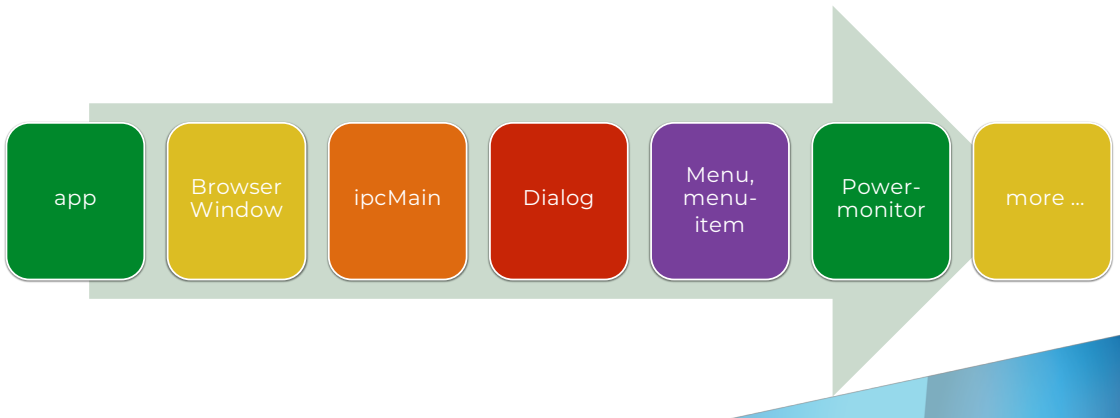
HTTP using axios library

18. For this part of lab, we will just observe how axios is being used in making http get and http post in our electron app.
19. For HTTP Post, please refer to the code used in signup.js under assets/js folder.
20. For HTTP Get, please refer to the code used in catalog.js under assets/js folder.

MODULE 4: ELECTRON API'S

DEV6

MAIN PROCESS - API



DEV6

APP

- Controls application event life cycle.
- Various events available:
  - Ready
  - Window-all-closed
  - Before-quit

```
const { app } = require('electron')
app.on('window-all-closed', () => {
  app.quit()
})
```

https://electronjs.org/docs/api/app

DEV6

BROWSER WINDOW

- Create and control
- Frameless window
- Parent and child windows
- Modal window

```
const {app, BrowserWindow} = require('electron');

let mainWindow;

app.on('ready', () => {
  mainWindow = new BrowserWindow({
      height: 600,
      width: 800
  });
  mainWindow.loadURL('https://github.com');
});
```

DEV6

IPC MAIN

- Helps communicating from main to renderer processes.

```
// In main process.
const { ipcMain } = require('electron')
ipcMain.on('asynchronous-message', (event, arg) => {
  console.log(arg) // prints "ping"
  event.sender.send('asynchronous-reply', 'pong')
})

ipcMain.on('synchronous-message', (event, arg) => {
  console.log(arg) // prints "ping"
  event.returnValue = 'pong'
})
```

https://electronjs.org/docs/api/ipc-main

```
// In renderer process (web page).
const { ipcRenderer } = require('electron')
console.log(ipcRenderer.sendSync('synchronous-message', 'ping'))

ipcRenderer.on('asynchronous-reply', (event, arg) => {
  console.log(arg) // prints "pong"
})
ipcRenderer.send('asynchronous-message', 'ping')
```

DIALOG

- Display native system dialogs for opening and saving files, alerting etc.

```
const { dialog } = require('electron')
console.log(dialog.showOpenDialog({
  properties: ['openFile', 'openDirectory', 'multiSelections']
}))
```

https://electronjs.org/docs/api/dialog

MENU AND MENU ITEM

- Create native application menus and context menus

```
var menu = Menu.buildFromTemplate([
    {
        label: 'Edit',
        submenu: [
            {
                label: 'Edit Sub Menu 1',
                click() {
                    shell.openExternal('http://google.com')
                }
            },
            {
                label: 'Edit Sub Menu 2',
                click() {
                    app.quit()
                }
            }
        ]
    },
    {
        label: 'Info'
    }
])
Menu.setApplicationMenu(menu);
}
```

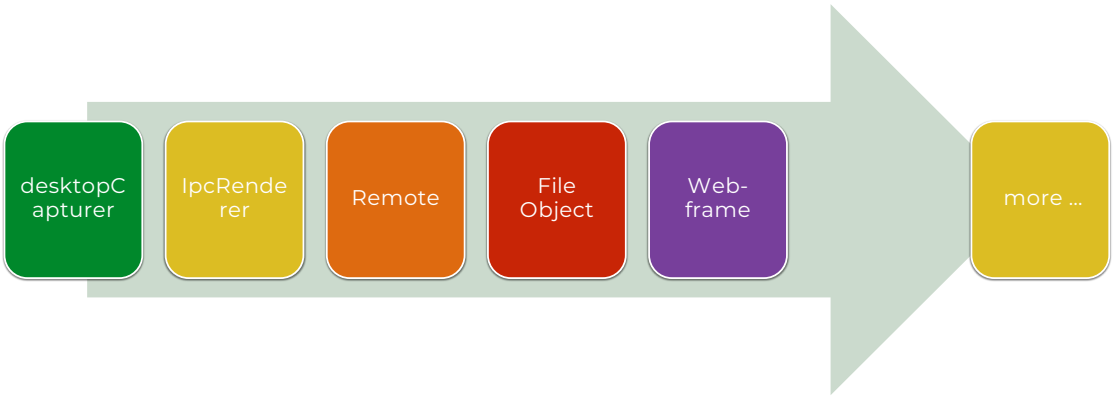
POWER MONITOR

- Monitor power state changes
- Various events available:
  - Suspend
  - Resume
  - On-battery
  - On-ac
  - Shutdown
  - Lock-screen

```
const electron = require('electron')
const { app } = electron

app.on('ready', () => {
  electron.powerMonitor.on('suspend', () => {
    console.log('The system is going to sleep')
  })
})
```

https://electronjs.org/docs/api/power-monitor

RENDER PROCESS - API



DESKTOP CAPTURER

- Access information about media sources that can be used to capture audio and video from the desktop

```
// In the renderer process.
const { desktopCapturer } = require('electron')

desktopCapturer.getSources({ types: ['window', 'screen'] }, (error, sources) => {
  if (error) throw error
  for (let i = 0; i < sources.length; ++i) {
    if (sources[i].name === 'Electron') {
      navigator.mediaDevices.getUserMedia({
        audio: false,
        video: {
          mandatory: {
            chromeMediaSource: 'desktop',
            chromeMediaSourceId: sources[i].id,
            minWidth: 1280,
            maxWidth: 1280,
            minHeight: 720,
            maxHeight: 720
          }
        }
      }).then((stream) => handleStream(stream))
        .catch((e) => handleError(e))
      return
    }
  }
})
```

https://electronjs.org/docs/api/desktop-capturer

IPC RENDERER

- Helps communicating from renderer to main process.

```
// In main process.
const { ipcMain } = require('electron')
ipcMain.on('asynchronous-message', (event, arg) => {
  console.log(arg) // prints "ping"
  event.sender.send('asynchronous-reply', 'pong')
})

ipcMain.on('synchronous-message', (event, arg) => {
  console.log(arg) // prints "ping"
  event.returnValue = 'pong'
})
```

https://electronjs.org/docs/api/ipc-main

```
// In renderer process (web page).
const { ipcRenderer } = require('electron')
console.log(ipcRenderer.sendSync('synchronous-message', 'ping'))

ipcRenderer.on('asynchronous-reply', (event, arg) => {
  console.log(arg) // prints "pong"
})
ipcRenderer.send('asynchronous-message', 'ping')
```

REMOTE

- Helps to use main process modules from renderer process(web page)

```
const { BrowserWindow } = require('electron').remote
let win = new BrowserWindow({ width: 800, height: 600 })
win.loadURL('https://github.com')
```

https://electronjs.org/docs/api/remote

FILE OBJECT

- It provides HTML5 File API to work natively with files on the filesystem

```
<div id="holder">
  Drag your file here
</div>

<script>
  document.addEventListener('drop', function (e) {
    e.preventDefault();
    e.stopPropagation();

    for (let f of e.dataTransfer.files) {
      console.log('File(s) you dragged here: ', f.path)
    }
  });
  document.addEventListener('dragover', function (e) {
    e.preventDefault();
    e.stopPropagation();
  });
</script>
```

https://electronjs.org/docs/api/file-object

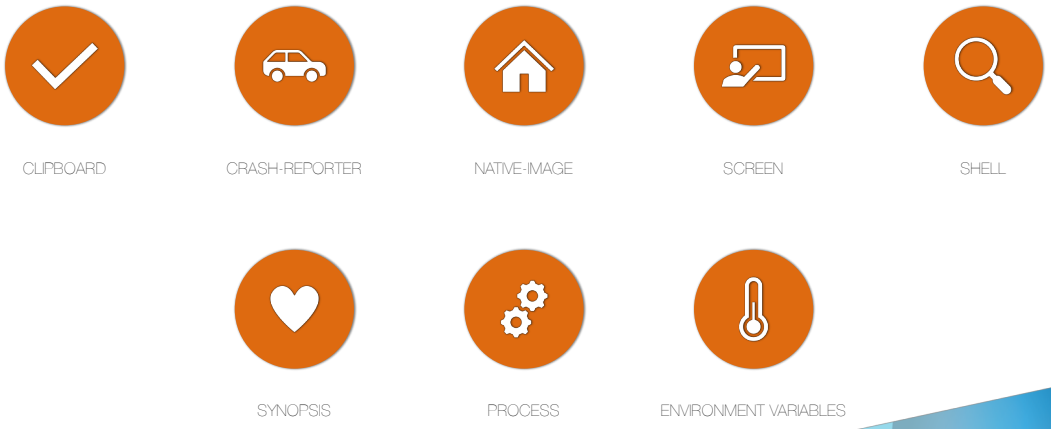
WEB FRAME

- For customizing the rendering of the current web page
- Various methods available:
  - setZoomFactor
  - setZoomLevel
  - setSpellCheckProvider

```
const { webFrame } = require('electron')

webFrame.setZoomFactor(2)
```

https://electronjs.org/docs/api/web-frame

API AVAILABLE TO BOTH PROCESS



DEV6

EXERCISE

Crash Reporting

DEV6

LAB 5: CRASH REPORTING

In this exercise, you will:

- Set up simple-breakpad-server as a server for collecting crash reports.
- Configure main.js for sending crash reports.
- Forcibly crashing the app and visualising the entire process of crash submission.

1. Check out the `Lab-Crash-Reporting` branch from the remote repository.

```
git checkout origin/Lab-Crash-Reporting
```

Note: The HEAD of this branch provides the completed code for the entire exercise.

---

2. Run `npm install` to ensure all the dependencies have been installed.
3. Create a new local branch for your work based on the "starting point..." commit.

```
git log --oneline
git checkout <hash for starting point commit>
git checkout -b Lab-Crash-Reporting-mine
```

4. Start the app.

```
npm run start
```

Start simple-breakpad-server

5. Go to your terminal window from any folder and run below command

```
npm install -g simple-breakpad-server
```

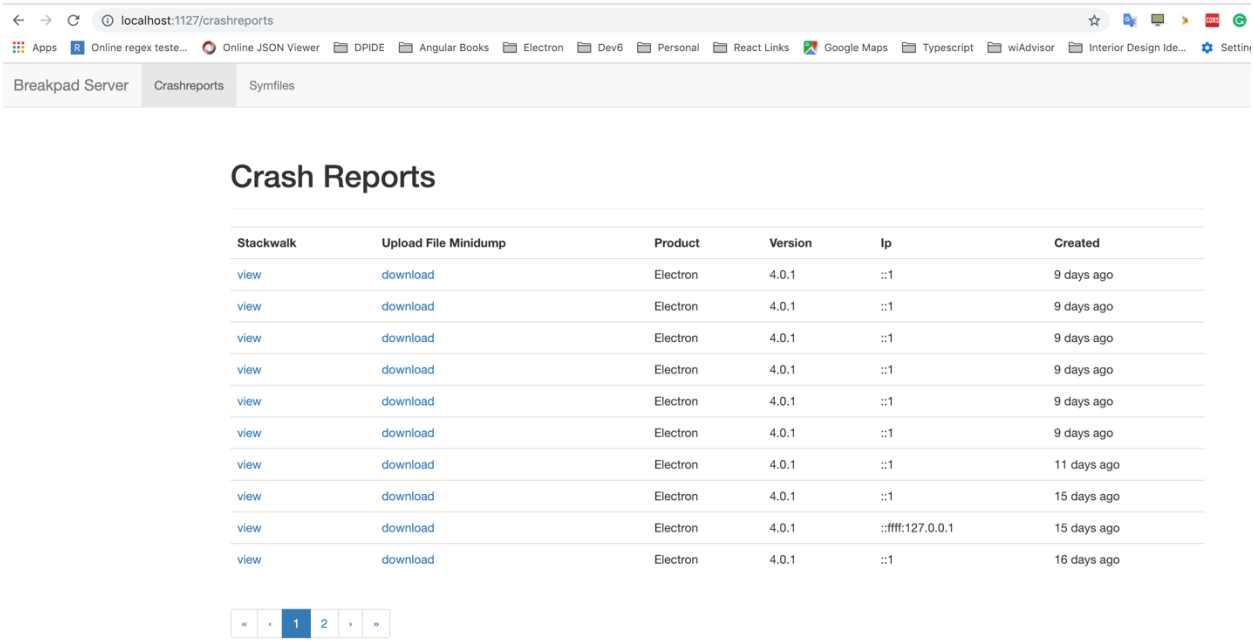
Note: This will install simple-breakpad-server globally on your machine.

---

6. Next step is to start the simple-breakpad-server from any folder outside your project directory. Run below command.

```
Simple-breakpad-server
```

Note: This command will start crash reporting server on port 1127. You can visit this url on your browser to make sure it is working fine. http://localhost:1127/crashreports



Configure main.js for sending crash reports.

7. Open main.js (which is at the root level). Paste below lines of code at the top line.

```
const { crashReporter } = require('electron');

crashReporter.start({

  productName: 'electron-shirtastic',

  companyName: 'Dev6',

  submitURL: 'http://localhost:1127/crashreports',

  uploadToServer: true

})
```

8. Restart the app.

```
npm run start
```

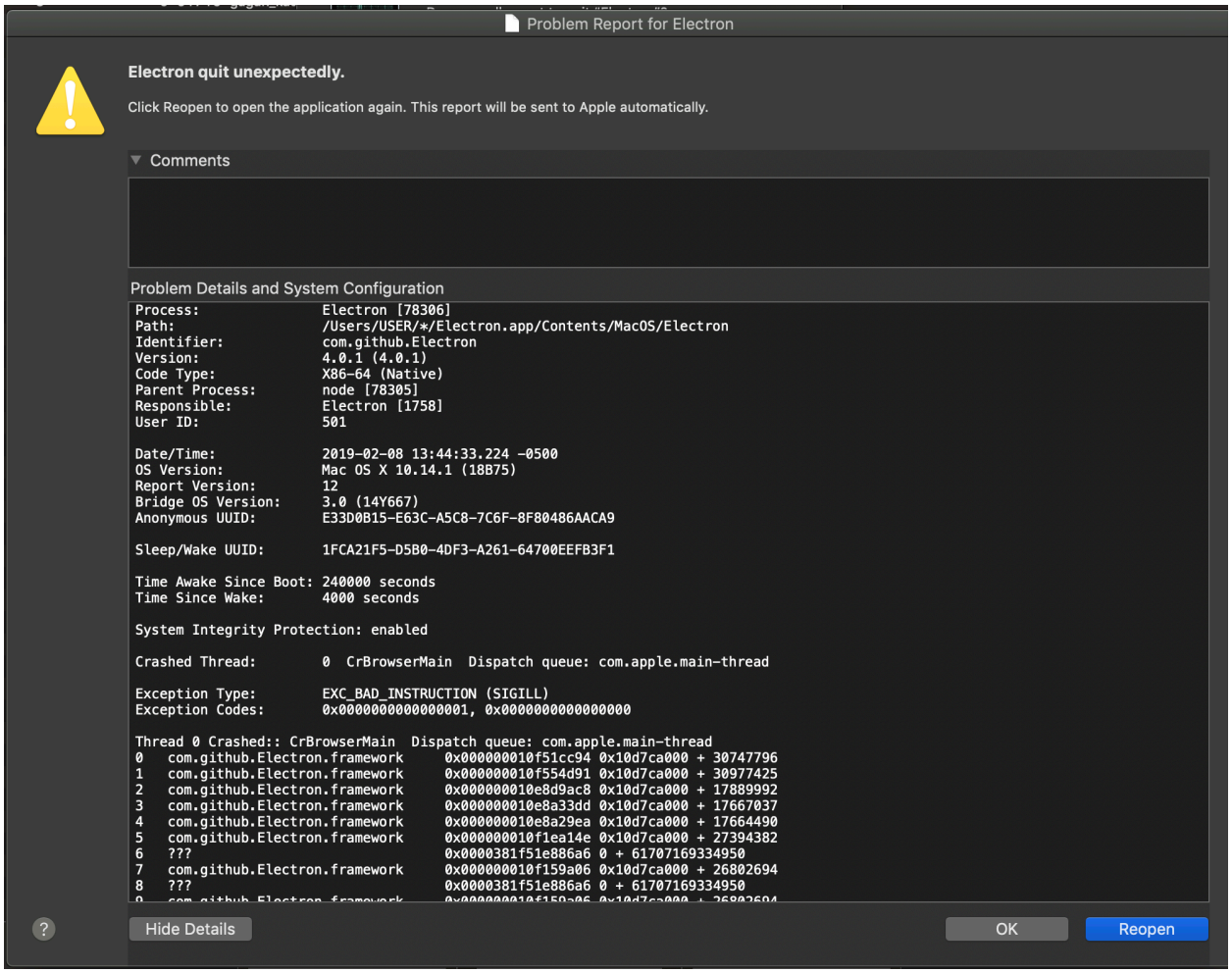
Test crash reporting feature.

9. Navigate to main.js at the root level of project. Add below lines of code anywhere in main.js. Below code is used to forcibly crash the electron application from main process.

```
process.crash()
```

10. Restart the app.

```
npm run start
```

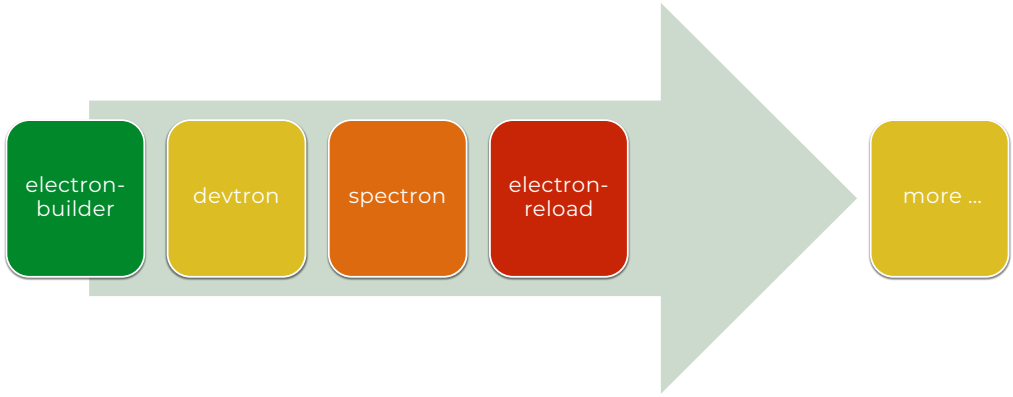


Note: Notice the application will crash as soon you start it and a new crash report will be submitted to server. Go back to localhost:1127 and check a new entry will be present. To read more about crash-reporting and simple-breakpad-server visit below URL's https://github.com/acrisci/simple-breakpad-server and https://electronjs.org/docs/api/crash-reporter

MODULE 5: ELECTRON USERLAND

DEV6

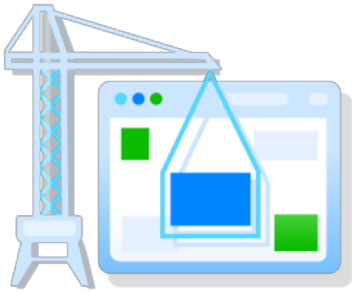
ELECTRON TOOLS



DEV6

ELECTRON BUILDER

- Complete solution to package and build a ready for distribution Electron app for MacOS, Windows and Linux with auto update support out of the box.
- Features like:
  - NPM packages management
  - Code signing
  - Auto update
  - Various target formats
    - macOS: dmg, pkg
    - Linux: AppImage, Debian etc.
    - Windows: nsis, nsis-web Installers and many more…

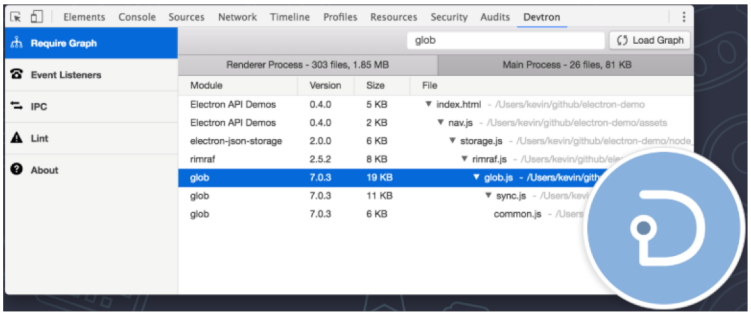


yarn add electron-builder --dev

https://github.com/electron-userland/electron-builder

DEVTRON

- An Electron DevTools extension to help you inspect, monitor and debug
- Features like:
  - Require graph
  - IPC monitor
  - Event Inspector
  - App Linter



npm install --save-dev devtron

https://github.com/electron/devtron

SPECTRON

- Perform Integration Test for Electron apps.
- Its build on top of ChromeDriver and WebdriverIO.
- Spectron supports all testing frameworks.
- npm install –save-dev spectron

```
# Install Spectron
$ npm install --save-dev spectron

// A simple test to verify a visible window is opened with a title
var Application = require('spectron').Application
var assert = require('assert')

var app = new Application({
  path: '/Applications/MyApp.app/Contents/MacOS/MyApp'
})

app.start().then(function () {
  // Check if the window is visible
  return app.browserWindow.isVisible()
}).then(function (isVisible) {
  // Verify the window is visible
  assert.equal(isVisible, true)
}).then(function () {
  // Get the window's title
  return app.client.getTitle()
}).then(function (title) {
  // Verify the window's title
  assert.equal(title, 'My App')
}).then(function () {
  // Stop the application
  return app.stop()
}).catch(function (error) {
  // Log any failures
  console.error('Test failed', error.message)
})
```

https://electronjs.org/spectron

ELECTRON-RELOAD

- Enables live reload contents of all active BrowserWindows when the source files are changed.
- Soft Reset: It refreshes WebContents alone.

```
require('electron-reload')(__dirname);
```

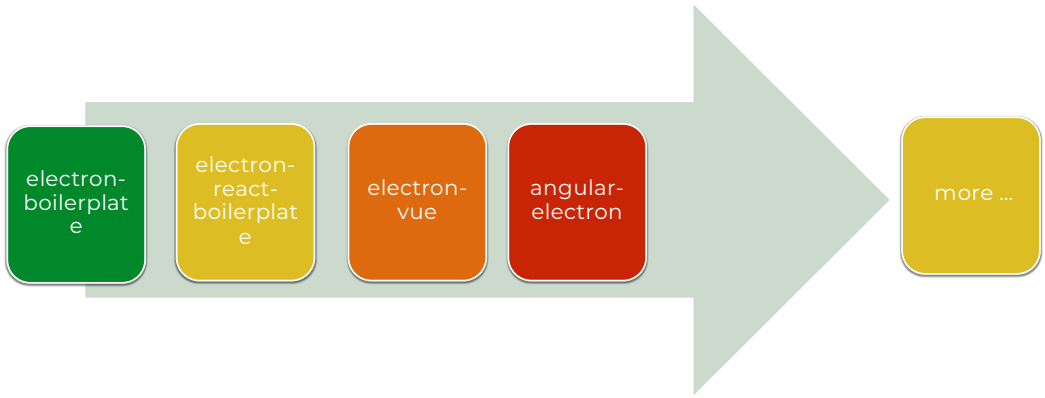
- Hard Reset: It starts a new electron process.

```
const path = require('path')

require('electron-reload')(__dirname, {
  electron: path.join(__dirname, 'node_modules', '.bin', 'electron')
});
```

npm install electron-reload --save-dev

https://www.npmjs.com/package/electron-reload

ELECTRON BOILERPLATES



DEV6

EXERCISE

Generating Installers and Automatic Updates

DEV6

LAB 6: GENERATING INSTALLERS

In this exercise, you will:

- Electron-builder installation and configuration.
- Generate installer for macOS.
- Generate installer for windows.

1. Check out the `Lab-Generating-Installers` branch from the remote repository.

```
git checkout origin/Lab-Generating-Installers
```

Note: The HEAD of this branch provides the completed code for the entire exercise.

---

2. Run `npm install` to ensure all the dependencies have been installed.
3. Create a new local branch for your work based on the "starting point..." commit.

```
git log --oneline
git checkout <hash for starting point commit>
git checkout -b Lab-Electron-Generating-Installers-mine
```

4. Start the app.

```
npm run start
```

Electron-builder Installation and configuration

5. Install electron-builder

```
npm install --save-dev electron-builder
```

6. Create build field in package.json (package.json is present at the root level of project).

```
"build": {
    "appId": "com.dev6.electron",
    "mac": {
```

```
    "category": "public.app-category.educational"
  },
  "dmg": {
    "contents": [
      {
        "x": 110,
        "y": 150
      },
      {
        "x": 240,
        "y": 150,
        "type": "link",
        "path": "/Applications"
      }
    ]
  },
  "linux": {
    "target": [
      "AppImage",
      "deb"
    ]
```

```
  },
  "win": {
    "target": [
      {
        "target": "nsis",
        "arch": [
          "ia32"
        ]
      }
    ],
    "icon": "build/icon.ico"
  }
},
```

Note: To refer the actual placement of build field in package.json file. Please refer to below screenshot.

```
{
  "name": "electron-shirtastic-app",
  "version": "1.0.0",
  "description": "Electron Bootcamp Demo",
  "main": "main.js",
  "scripts": {
    "start": "electron .",
    "pack": "electron-builder --pack",
    "postinstall": "electron-builder install-app-deps",
    "buildWin": "electron-builder --win",
    "buildMac": "electron-builder --mac"
  },
  "keywords": [],
  "author": "gagank@dev6.com",
  "license": "ISC",
  "build": {
    "appId": "com.dev6.electron",
    "mac": {
      "category": "public.app-category.educational"
    },
    "dmg": {…
    },
    "linux": {…
    },
    "win": {…
    }
  },
  "devDependencies": {
    "devtron": "^1.4.0",
    "electron": "4.0.1",
    "electron-builder": "^20.38.5",
    "electron-reload": "^1.4.0"
  },
  "dependencies": {
    "axios": "^0.18.0",
    "electron-is-dev": "^1.0.1",
    "electron-log": "^3.0.1",
    "json-server": "^0.14.2"
  }
}
```

7. Add couple of new entries in scripts field.

pack script only generates the package directory without packaging it.
postinstall script will ensure your native dependencies are always matched electron version.
buildWin script will generate the distributables for window platform.
buildMac script will generate the distributables for macOS platform.

```
"scripts": {

    "start": "electron .",

    "pack": "electron-builder --pack",

    "postinstall": "electron-builder install-app-deps",

    "buildWin": "electron-builder --win",

    "buildMac": "electron-builder -mac -c.mac.identity=null"

  },
```

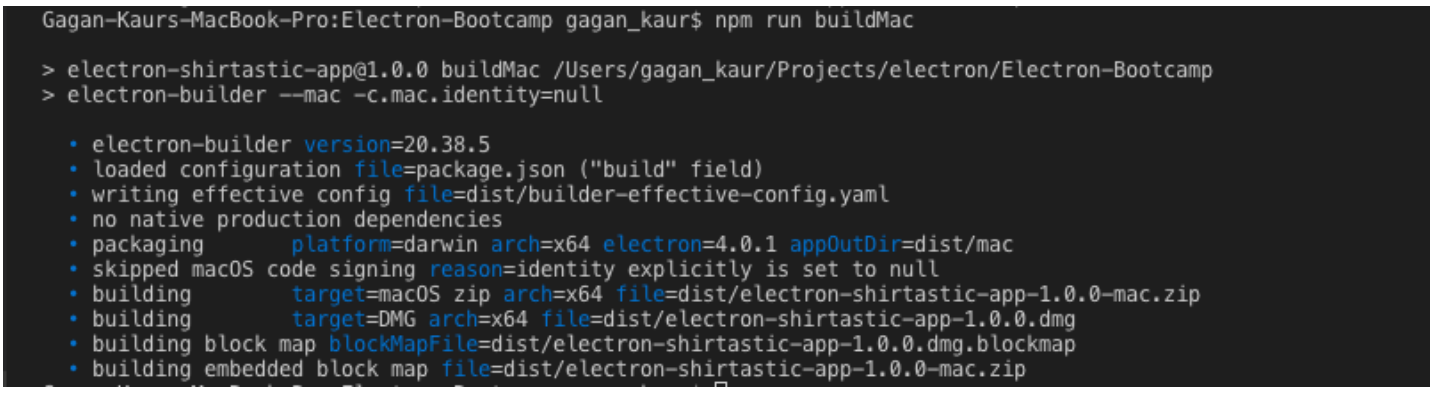
8. Restart the app

```
npm run start
```

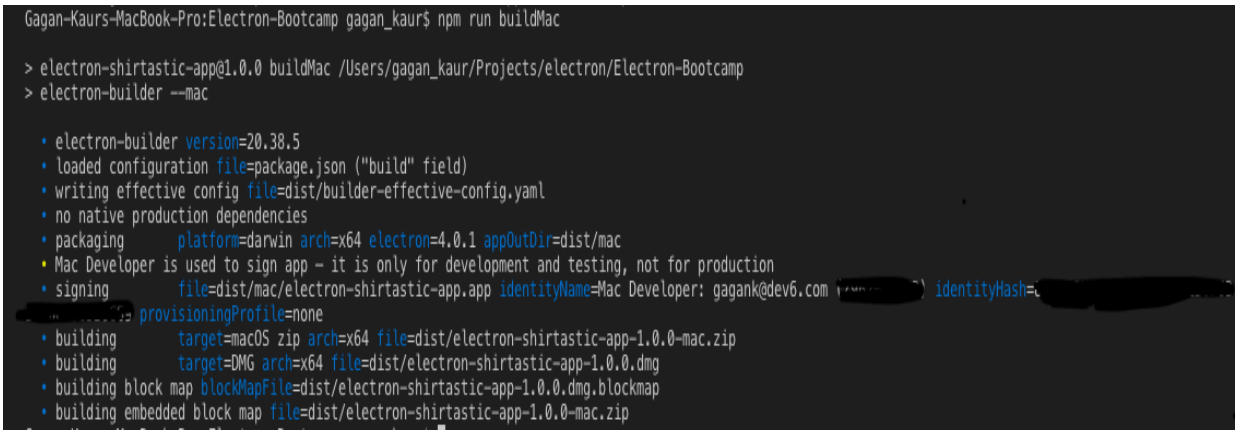
Generate installer for macOS

9. Now it is time to generate distributable (installer) for macOS. Run below command on terminal

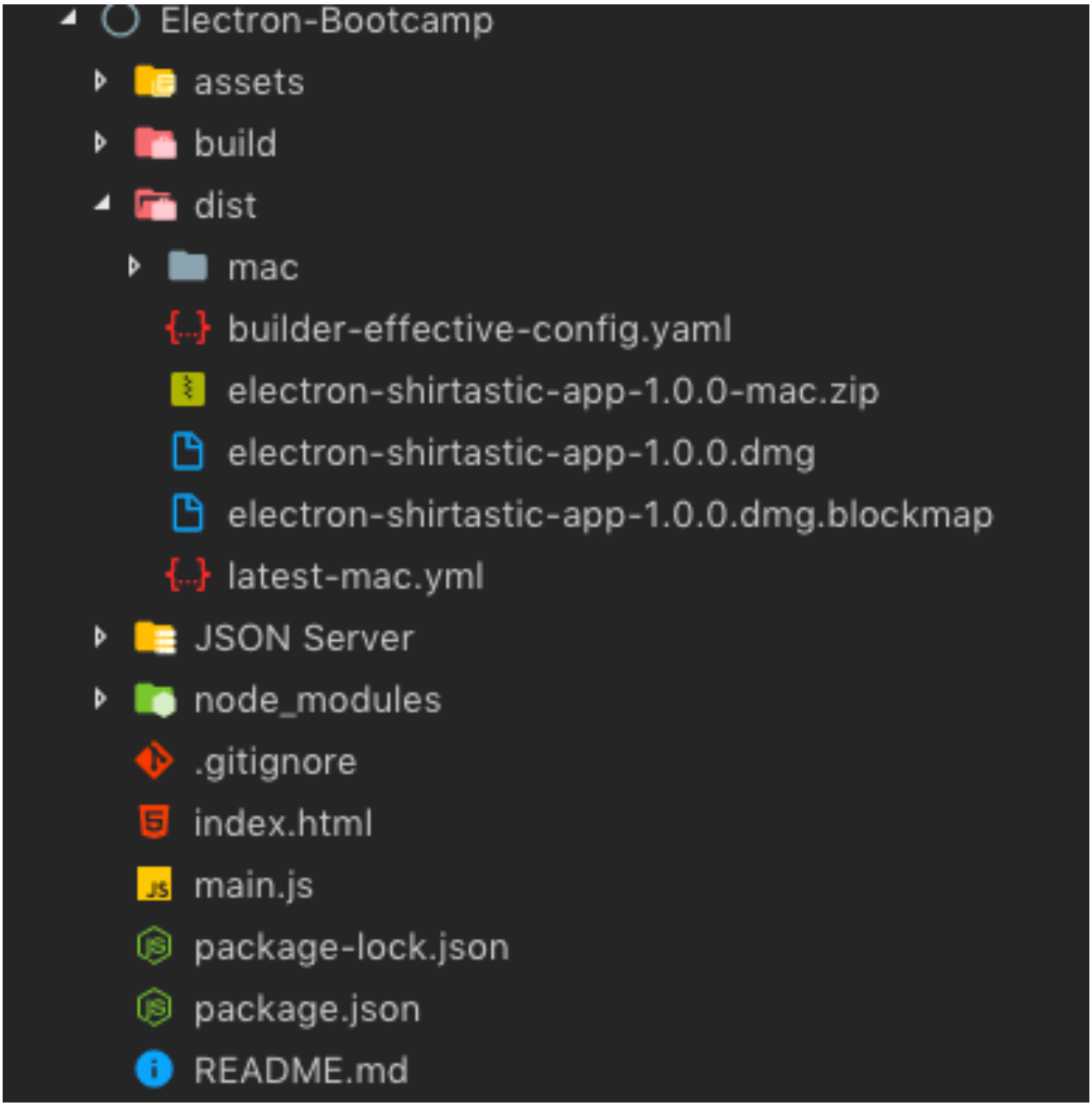
```
npm run buildMac
```



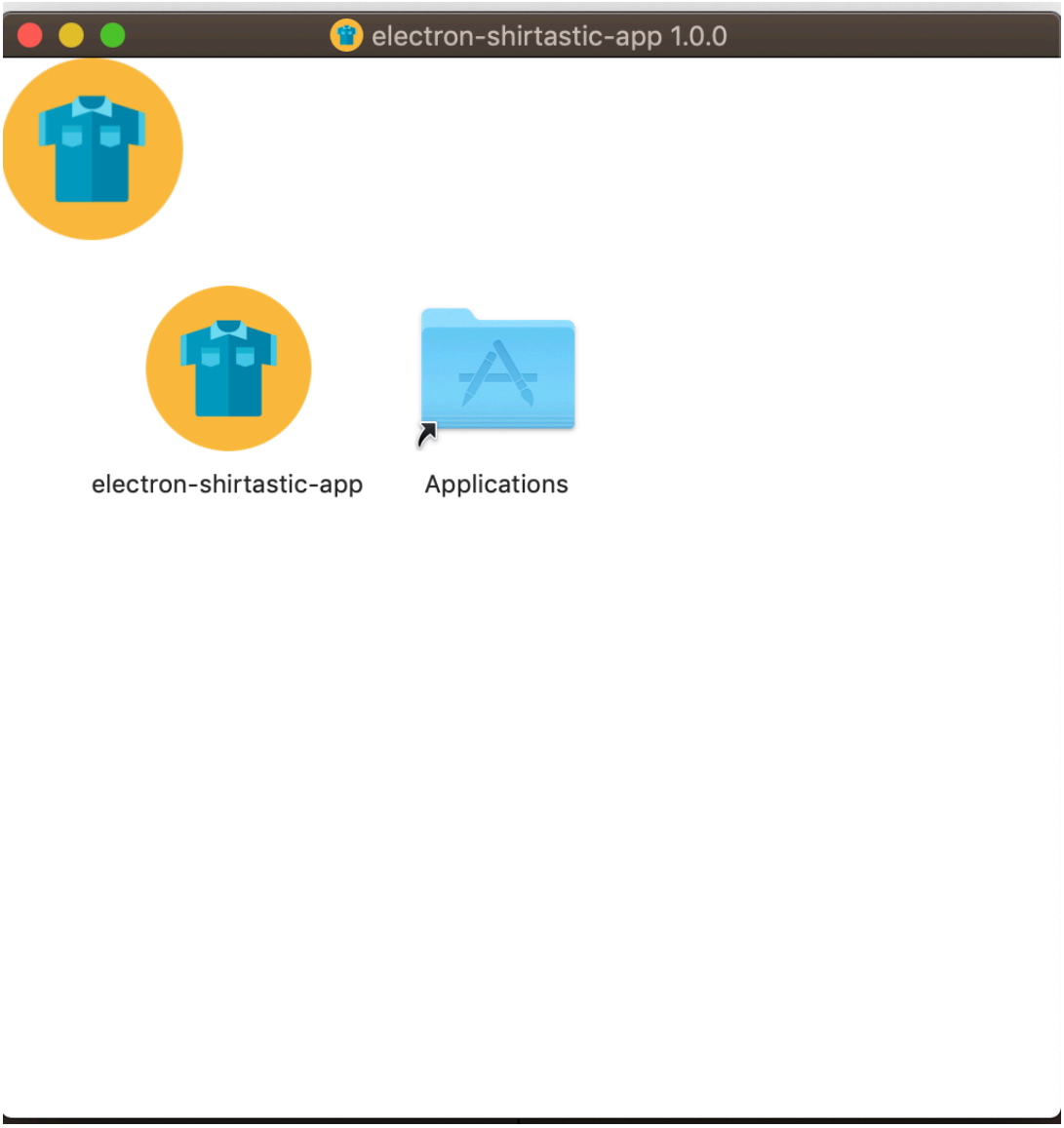
Note: We have passed an extra flag for buildMac script -c.mac.identity=null. By doing so you can skip the code signing process for build purpose on macOS. However, for distribution of electron app on Mac App Store you have to sign the app. Below is the screenshot for code signing build process logs. You can read more about how to generate code certificates by visiting this URL https://www.electron.build/code-signing and https://www.electron.build/code-signing#where-to-buy-code-signing-certificate.



10. The result of above command will be a dist folder is generated at the root level of project which contains the distributable. Below is the screenshot for reference to cross check the location of dist folder.



11. You can test the application by clicking on the dmg file. It will produce below screen to appear. Now you can drag the electron-shirtastic-app to Applications folder. And, it will start appearing in your launchpad. Please verify it by going to Launchpad.

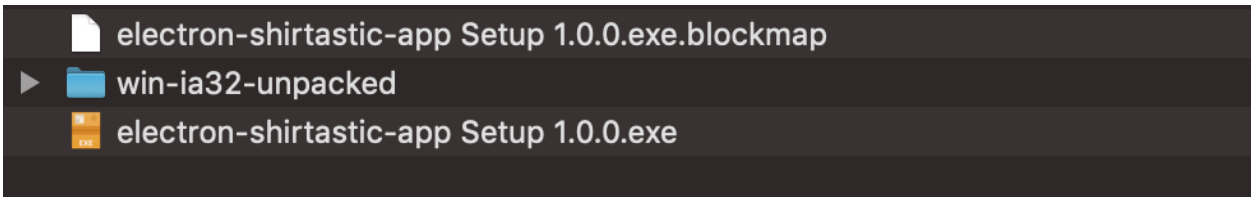


Generate installer for windows

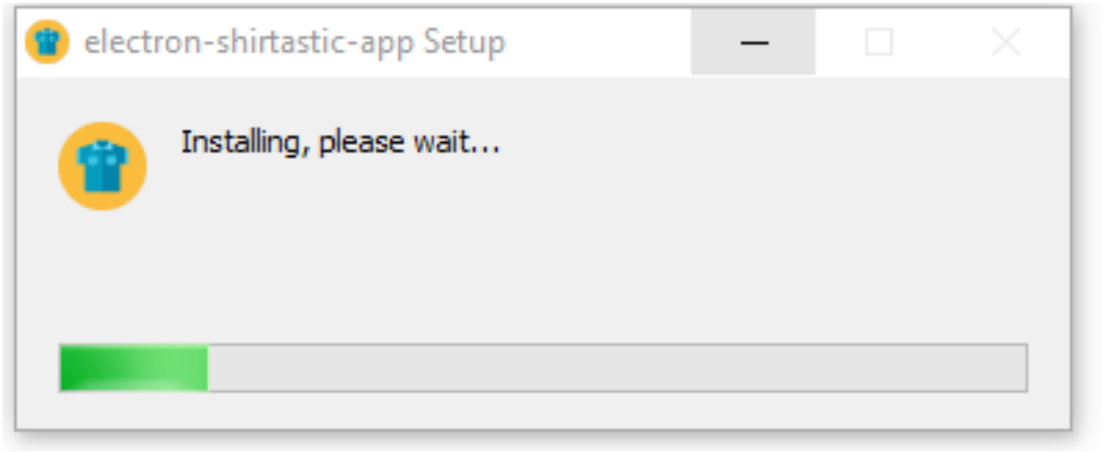
12. To generate the installer for windows, run the following command.

```
npm run buildWin
```

13. It will generate below artifacts inside dist folder.



14. You can test the application by clicking on exe file. As soon you click on it, installation will start for windows.



Note: You can read more about how to do multi-platform build by visiting this URL https://www.electron.build/multi-platform-build.

LAB 7: AUTOMATIC UPDATES

In this exercise, you will:

- Set up http-server for hosting updates.
- Configure build field for making use of electron-builder to consume updates.
- Learn how to test auto-updates on macOS.
- Learn how to test auto-updates on windows.

1. Check out the `Lab-Silent-Updates` branch from the remote repository.

```
git checkout origin/Lab-Silent-Updates
```

Note: The HEAD of this branch provides the completed code for the entire exercise.

---

2. Run `npm install` to ensure all the dependencies have been installed.
3. Create a new local branch for your work based on the "starting point..." commit.

```
git log --oneline
git checkout <hash for starting point commit>
git checkout -b Lab-Silent-Updates-mine
```

4. Start the app.

```
npm run start
```

Set up http-server for hosting updates.

5. Install http-server

```
npm install --save http-server
```

6. Navigate to dist folder in a new terminal window. And run below command.

```
http-server
```

```
[Gagan-Kaurs-MacBook-Pro:dist gagan_kaur$ http-server
Starting up http-server, serving ./
Available on:
  http://127.0.0.1:8080
  http://127.94.0.2:8080
  http://127.94.0.1:8080
  http://10.149.76.3:8080
Hit CTRL-C to stop the server
```

Note: Now you can check http://127.0.0.1.8080 on your browser. It will list the contents of dist folder.

Configure electron-builder's build field in order to consume updates.

7. Open package.json and paste below lines of code inside build field.

```
"publish": [

      {

        "provider": "generic",

        "url": "http://127.0.0.1:8080"

      }

    ],
```

Note: To cross-check the actual placement of this line of code, please refer below screenshot.

```
{
  "name": "electron-shirtastic-app",
  "version": "1.0.0",
  "description": "Electron Bootcamp Demo",
  "main": "main.js",
  "scripts": {
    "start": "electron .",
    "pack": "electron-builder --pack",
    "postinstall": "electron-builder install-app-deps",
    "buildWin": "electron-builder --win",
    "buildMac": "electron-builder --mac -c.mac.identity=null"
  },
  "keywords": [],
  "author": "gagank@dev6.com",
  "license": "ISC",
  "build": {
    "appId": "com.dev6.electron",
    "publish": [
      {
        "provider": "generic",
        "url": "http://127.0.0.1:8080"
      }
    ],
    "mac": {…
    },
    "dmg": {…
    },
    "linux": {…
    },
    "win": {…
    }
  },
  "devDependencies": {
    "devtron": "^1.4.0",
    "electron": "4.0.1",
    "electron-builder": "^20.38.5",
    "electron-reload": "^1.4.0"
  },
```

Note: Please refer to this URL for more configuration options
https://www.electron.build/auto-update

Testing auto-updates on macOS

8. First create a new distributable by running below command by adding version 1.0.0 in package.json

```
npm run buildMac
```

9. Then run the dmg file and install the electron app.
10. Now go back to same terminal window and re-run the buildMac script, but this time change the version to 2.0.0

```
npm run buildMac
```

11. Now, re launch the app from launchpad, and as soon you do so, you will see a pop up like below screen.

12. If you press on Ok, your application will be quit and relaunched with new version installed.

Note: If you closely observe, there is a dev-app-update.yml file at the root level. The purpose of this file is when you test updates in development mode, auto update module requires it internally to function smoothly.

Testing auto updates on Windows

13. Open autoupdater.js under assets/js folder. And uncomment the code mentioned in below screenshot.

```
    //Below code is for windows to view relaunch buttons
    dialog.showMessageBox({
        type: 'question',
        buttons: ['Install and Relaunch', 'Later'],
        defaultId: 0,
        message: 'A new version of ' + app.getName() + ' has been downloaded',
        detail: message
    }, response => {
        if (response === 0) {
            setTimeout(() => autoUpdater.quitAndInstall(), 1);
        }
    });
});
```

14. And, comment this code.

```
    //Below code works for MacOS
    // dialog.showMessageBox({
    //  title: 'Install Updates',
    //  message: 'Updates downloaded, application will be quit for update...'
    //   }, () => {
    //  setImmediate(() => autoUpdater.quitAndInstall())
    //   })
```

s

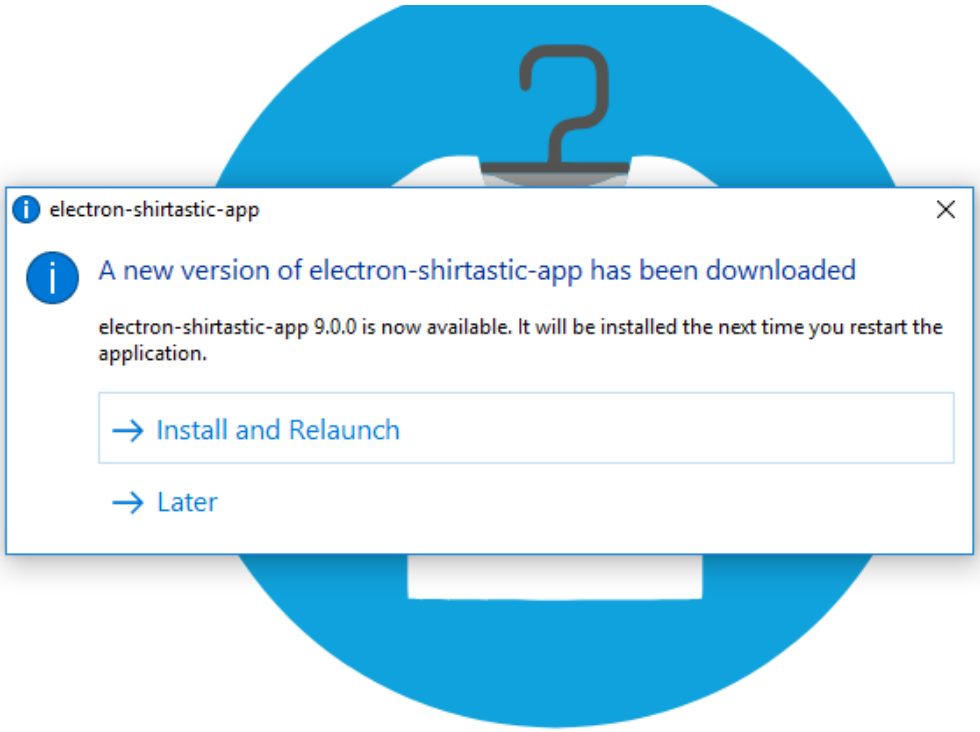
15. First create a new distributable by running below command by adding version 1.0.0 in package.json

```
npm run buildWin
```

16. Then run the exe file and install the electron app.
17. Now go back to same terminal window and re-run the buildWin script, but this time change the version to 2.0.0

```
npm run buildWin
```

18. Now, re launch the app from Application installed, and as soon you do so, you will see a pop up like below.



19. If you select Install and Relaunch, the application will be killed and restart with new version installed. And, if you select Later option, the update will be ignored, but it will download the next time when you restart the app.

Note: You can read more about the various options available by visiting this URL https://www.electron.build/auto-update.

RESOURCES

- Electron Documentation https://electronjs.org/docs
- Electron Userland https://github.com/electron-userland
- Coursetro https://coursetro.com/courses/22/Creating-Desktop-Apps-with-Electron-Tutorial