Clipmaster 9000

Same files structure as last time.

- lib/main.js
- lib/renderer.js
- lib/index.html
- lib/style.css

In this application, we're going to use Max Ogden's excellent *menubar* module.

In `main.js`, we'll get things rolling by including Electron and menubar.

```
const electron = require('electron');
const Menubar = require('menubar');
```

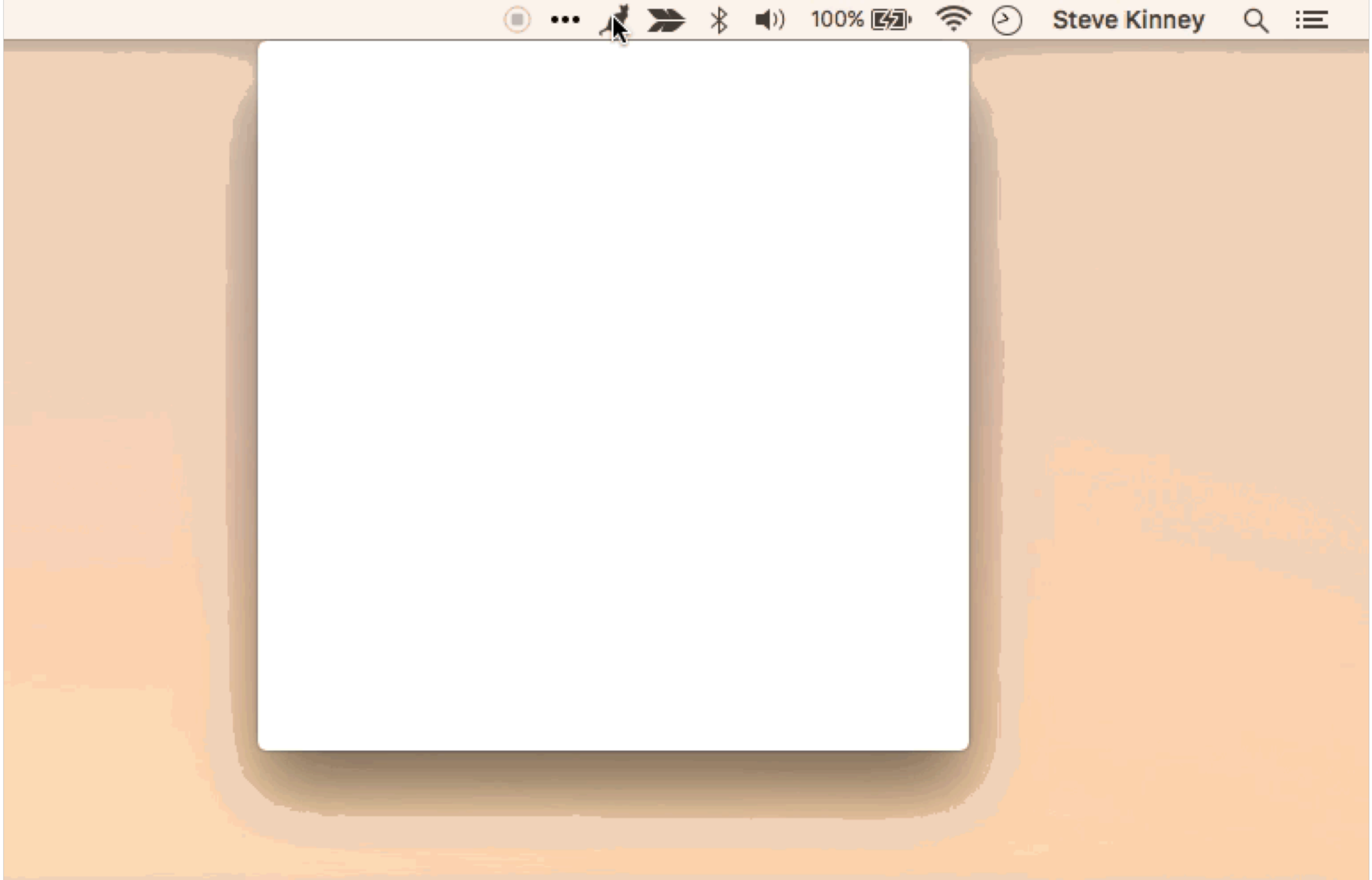
The `Menubar` is a constructor. We'll create an instance to work with.

```
const menubar = Menubar();
```

We'll wait for the application to be fire a `ready` event and then we'll log to the console.

```
menubar.on('ready', function () {
  console.log('Application is ready.');
});
```

Let's use `npm start` to verify that it works correctly. The library gives us a pleasant little cat icon as a default.


100%
Steve Kinney

When it has done so, it will fire a `after-create-window` event. We can listen for this event and load our HTML page accordingly.

```
menubar.on('after-create-window', function () {
  menubar.window.loadURL(`file://${__dirname}/index.html`);
});
```

Implementing The Renderer Functionality

We'll need…

1. Access to Electron's `clipboard` module.
2. A reference to the "Copy From Clipboard" button.
3. A reference to the clippings list.

Let's implement all three in one swift motion:

```
const clipboard = electron.clipboard;

const $clippingsList = $('.clippings-list');
const $copyFromClipboardButton = $('#copy-from-clipboard');
```

I made a function for you:

```
const createClippingElement = require('./support/create-clipping-element');
```

```
function addClippingToList() {
   var text = clipboard.readText();
   var $clipping = createClippingElement(text);
   $clippingsList.append($clipping);
}
```

Now, when a user clicks the "Copy from Clipboard" button, we'll read from the clipboard and add that clipping to the list.

```
$copyFromClipboardButton.on('click', addClippingToList);
```

If all went well, our `renderer.js` looks something like this:

```
const $ = require('jquery');
const electron = require('electron');

const clipboard = electron.clipboard;

const $clippingsList = $('.clippings-list');
const $copyFromClipboardButton = $('#copy-from-clipboard');
const createClippingElement = require('./support/create-clipping-element');

function addClippingToList() {
  var text = clipboard.readText();
  var $clipping = createClippingElement(text);
  $clippingsList.append($clipping);
}

$copyFromClipboardButton.on('click', addClippingToList);
```

Wiring Up Our Actions

We have three buttons on each clipping element.

1. **"→ Clipboard" will write that clipping back to the clipboard.**
2. **"Publish" will send it up to an API that we can share.**
3. **"Remove" will remove it from the list.**

We'll take advantage of event delegation, in order to avoid memory leaks.

Let's implement event listeners for all three. We'll use dummy functionality for "copy" and "publish".

```
$clippingsList.on('click', '.remove-clipping', function () {
  $(this).parents('.clippings-list-item').remove();
});

$clippingsList.on('click', '.copy-clipping', function () {
  var text = $(this).parents('.clippings-list-item').find('.clipping-text').text();
  console.log('COPY', text);
});

$clippingsList.on('click', '.publish-clipping', function () {
  var text = $(this).parents('.clippings-list-item').find('.clipping-text').text();
  console.log('PUBLISH', text);
});
```

You can fire open developer tools using `Command-Option-I` **or** `Control-Option-I` **for OS X and Windows respectively. I like to break them out to their own window.**

Writing Text to the Clipboard.

In the previous code we just wrote, we were just logging the clipping's contents to the console. Let's write it to the clipboard instead.

```
$clippingsList.on('click', '.copy-clipping', function () {
  var text = $(this).parents('.clippings-list-item').find('.clipping-text').text();
  clipboard.writeText(text);
});
```

Publishing to a Gist

Limitations of the Browser Environment

We'll bring in the Request library.

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

We'll need to set three important pieces of information:

1. The URL of the Gist API
2. A User-Agent (the Gist API requires this)
3. A body with the text we'd like to use formatted in a particular way

Our data will look as follows:

```
{
  url: 'https://api.github.com/gists',
  headers: {
    'User-Agent': 'Clipmaster 9000'
  },
  body: JSON.stringify({
    description: "Created with Clipmaster 9000",
    public: "true",
    files:{
      "clipping.txt": {
        content: text
      }
    }
  }
})
```

http://bit.ly/fluent-request

```
$clippingsList.on('click', '.publish-clipping', function () {
  var text = $(this).parents('.clippings-list-item').find('.clipping-text').text();
  request.post({
    url: 'https://api.github.com/gists',
    headers: {
      'User-Agent': 'Clipmaster 9000'
    },
    body: JSON.stringify({
      description: "Created with Clipmaster 9000",
      public: "true",
      files:{
        "clipping.txt": {
          content: text
        }
      }
    })
  }, function (err, response, body) {
    if (err) { return alert(JSON.parse(err).message); }

    var gistUrl = JSON.parse(body).html_url;
    alert(gistUrl);
    clipboard.writeText(gistUrl);
  });
});
```

Using Notifications

- Windows 10 and OS X work out of the box.
- Earlier versions of Windows and Linuxes require additional tweaking as covered in the documentation.

Here's a little snipped form the documentation demonstrating how to use notifications.

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

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

Let's replace our alerts with notifications. We'll be modifying the callback in the Request callback from just a few minutes ago:

```
function (err, response, body) {
  // Error handling…

  var gistUrl = JSON.parse(body).html_url;
  var notification = new Notification('Your Clipping Has Been Published', {
    body: `Click to open ${gistUrl} in your browser.`
  });

  notification.onclick = function () {
    electron.shell.openExternal(gistUrl);
  };

  clipboard.writeText(gistUrl);
})
```

Adding Global Shortcuts

We'll start by creating a reference to Electron `globalShortcut` module in `main.js`.

```
const globalShortcut = electron.globalShortcut;
```

When the `after-create-window` event is fired, we'll register our shortcut.

```
menubar.on('after-create-window', function () {
  menubar.window.loadURL(`file://${__dirname}/index.html`);

  var createClipping = globalShortcut.register('CommandOrControl+!', function () {
    menubar.window.webContents.send('create-new-clipping');
  });

  if (!createClipping) { console.log('Registration failed', 'createClipping'); }
});
```

Let's modify the event listener to send a message to the renderer process.

```
var createClipping = globalShortcut.register('CommandOrControl+!', function () {
  menubar.window.webContents.send('create-new-clipping');
});
```

In `renderer.js`, we'll listen for this message. First, we'll require the `ipcRenderer` module.

```
const ipc = electron.ipcRenderer;
```

We'll then listen for an event on the `create-new-clipping` channel.

```
ipc.on('create-new-clipping', function (event) {
  addClippingToList();
  new Notification('Clipping Added', {
    body: `${clipboard.readText()}`
  });
});
```

We won't do this now, because it's more of the same. But could add additional shortcuts to our application as well.

```
var copyClipping = globalShortcut.register('CmdOrCtrl+Alt+@', function () {
  menubar.window.webContents.send('clipping-to-clipboard');
});

var publishClipping = globalShortcut.register('CmdOrCtrl+Alt+#', function () {
  menubar.window.webContents.send('publish-clipping');
});
```

When the application is ready to quit, we'll unregister our shortcuts.

```
menubar.app.on('will-quit', function () {
  globalShortcut.unregisterAll();
});
```

Congratulations

You built a second application.