Back to the roots! Electron bringt Vue.js auf den Desktop

Ein Webframework geht fremd

Jeder UI-Entwickler, der etwas auf sich hält, entwickelt heutzutage Anwendungen in JavaScript im Browser. Dort sind sie einfach am besten aufgehoben, und wir brauchen uns keine Gedanken um lästige Dinge wie die Softwareverteilung oder das Einspielen von Updates zu machen. Und doch – manchmal wollen unsere Anwender Features, die der Browser einfach nicht liefert. Klingt das abwegig? Wer jetzt nickt, möge die Anzahl der Apps auf seinem Smartphone zählen. Lokal installierte Anwendungen haben auch im Jahr 2018 ihre Vorteile. Dabei können Sie Ihr bewährtes Webframework sogar weiterhin verwenden, denn Electron bringt Vue.js auf den Desktop.

von Stephan Rauh

In den 1980er Jahren kam der PC auf. Der Name war Programm: Jeder „Personal Computer“ gehörte einem einzelnen Anwender, der genau die Software installieren konnte, die er brauchte. Und spätestens in den 1990er Jahren waren die Tage der Terminals mit 25 Zeilen à 80 grüne oder bernsteinfarbene Zeichen gezählt. Der PC bot den Anwendern eine viel bessere User Experience. Zuerst kam Farbe, dann kamen Fenster und grafische Elemente, Multitasking und vieles mehr. Die Produktivität der Anwender stieg enorm.

Die IT der großen Firmen war weniger begeistert. Die Software wurde komplexer und damit teurer. Und wie installiert man ein Softwareupdate auf tausend PCs? Dafür wurden Lösungen entwickelt, die seinerzeit eher schlecht als recht liefen. Also begann um das Jahr 2000 herum die Rolle rückwärts. Anwendungen werden seitdem im Browser geschrieben. Die große Ära der serverseitigen Webanwendungen begann. Aus Sicht der Usability war das eine Katastrophe. In den ersten Jahren waren die neuen Anwendungen langsam, flackerten bei jedem Maskenwechsel und boten viel weniger Möglichkeiten als die alten Desktopanwendungen. Aber die Produktivität der Betriebsabteilung und der Softwareentwickler war sehr viel höher. Das gab den Ausschlag.

Im Laufe der Jahre besserten die Browserhersteller nach. Heute sind die Browser gut genug, um fast alle Anforderungen abzudecken. Ein Beispiel: Wenn der Unternehmensstandard „Internet Explorer“ heißt, müssen große Abstriche gemacht werden. Weiter unten nenne ich noch weitere Beispiele. Mit Electron können Sie aber die Vorteile beider Welten nutzen.

Was ist Electron?

Electron ist das Framework der Wahl, um eine moderne HTML5-Anwendung auf den Desktop zu bringen. Genauer gesagt: auf den Desktop von Windows, Linux oder macOS. Andere Zielplattformen werden nicht unterstützt, für mobile Geräte ist Electron nicht gedacht.

Im Wesentlichen ist Electron ein abgespeckter Google-Chrome-Browser zusammen mit einem Node.js-Server, der die Vue.js-Anwendung hostet. Server und Browser bilden eine Einheit. Aus Anwendersicht ist das Gesamtpaket von einer normalen Anwendung nicht zu unterscheiden. Electron-Anwendungen können Dateien lesen und schreiben, die Zwischenablage verwenden und vieles mehr. Sie können auch drucken – ohne den bei Browseranwendungen üblichen Umweg über PDF-Dateien. Das ist zum Beispiel bei Etikettendruckern mit proprietärer Schnittstelle wichtig, falls der Hersteller keinen Druckertreiber für Windows anbietet.

Für uns Programmierer ändert sich durch Electron wenig. Wir können das gewohnte, komfortable Programmiermodell von Vue.js verwenden und haben zusätzlich ein paar neue APIs zur Verfügung.

Wie kann ich Electron zusammen mit Vue.js verwenden?

Es gibt mindestens zwei Möglichkeiten, unsere Vue.js-Anwendung als Electron-Anwendung zu verpacken. Die erste Möglichkeit ist, Nativefier [1] zu verwenden. Das ist eine gute Wahl, wenn es die Vue.js-Anwendung bereits

Abb. 1: Die Projektstruktur kommt ohne Überraschungen daher

gibt. Nativefier macht aus einer beliebigen HTML5-Anwendung eine Desktopanwendung. Falls Sie einen hybriden Ansatz fahren – sprich sowohl die HTML-Version als auch die native Anwendung anbieten – ist Nativefier eine gute Wahl. Ich habe Nativefier noch nicht selbst verwendet, aber der Anzahl der Sterne auf GitHub nach zu urteilen, handelt es sich um eine gute Wahl.

Der Hybridansatz hat aber einen großen Nachteil: Er bietet nur die APIs, die auch im Browser zur Verfügung stehen. Dateizugriff und der direkte Zugriff auf den Drucker gehören nicht dazu. Der Rest dieses Artikels betrachtet daher nur die zweite Alternative: electron-vue [2]. Diese Variante bietet sich an, wenn der Desktop das alleinige oder wenigstens das primäre Installationsziel ist.

Leinen los!

Es wird Zeit, mit dem Programmieren zu beginnen! Falls Sie es nicht ohnehin schon getan haben, installieren Sie das vue-cli: *npm install -g vue-cli*. Als Nächstes legen Sie ein Projekt mit *vue init* an. Electron-vue ist ein Template für das vue-cli, das während der Installation aus dem Internet heruntergeladen wird. Geben Sie also auf der Kommandozeile diese Anweisung ein: *vue init simulatedgreg/electron-vue my-project*. Das Installationsskript stellt Ihnen einige Fragen. Unter anderem können Sie je nach Bedarf axios, den vue-router, vuex, die Linter-Unterstützung sowie ein Testframework auswählen (Listing 1). Die Anwendung liegt jetzt auf der Festplatte. Sie müssen nur noch *npm install* ausführen und die Anwendung starten:

```
cd my-project
npm install
npm run dev
```

Listing 1

```
> vue init simulatedgreg/electron-vue my-project 2
? Application Name my-project
? Project description An electron-vue project
? Select which Vue plugins to install axios, vue-electron, vue-router, vuex
? Use linting with ESLint? Yes
? Which ESLint config would you like to use? Standard
? Set up unit testing with Karma + Mocha? Yes
? Set up end-to-end testing with Spectron + Mocha? Yes
? What build tool would you like to use? builder
? author Stephan
```

Das war es auch schon. Die Anwendung ist jetzt im einem eigenen Fenster gestartet, so wie jede andere native Anwendung auch, es sei denn, Sie verwenden Windows. Dann fehlen Ihnen wahrscheinlich noch Python und Visual Studio, um das Modul node-gyp zu kompilieren. Am besten schauen Sie in die Dokumentation von electron-vue, in der einige Hinweise für Windows-Anwender gesammelt sind [3].

Wie ist das Projekt aufgebaut?

Wenn Sie einfach alle Defaultoptionen akzeptiert haben, sieht Ihr Projekt wie in **Abbildung 1** aus. Wie Sie sehen, birgt die Projektstruktur keine großen Überraschungen. Es ist ein ganz normales Vue.js-Projekt. Sollten Sie noch nicht mit Vue.js vertraut sein oder einfach mehr über die Entwicklung mit Electron und Vue.js lesen wollen, empfehle ich Ihnen zur Einführung den Artikel von Kirsten Swanson [4] und das Tutorial von Ogundipe Samuel Ayo [5]. Das Handbuch von electron-vue [2] bietet ebenfalls einen guten Einstieg.

Die „main"-Klasse von Electron

Der größte Teil der Magie ist im Verzeichnis *main* versteckt. Dort liegen die Dateien, die das Anwendungsfenster und den URL *http://localhost:9080* öffnen. Die Anwendung besteht aus zwei unabhängigen Prozessen: dem UI-Teil und dem Node.js-Server. Sie können sogar Ihren eigenen Express-Server in diesen Node.js-Server einhängen (Listing 2, [6]). Es spricht also nichts dagegen, sowohl den Clientcode als auch die Serverquelltexte im gleichen Installationspaket zu verteilen.

Halt! Ist das nicht ein Antipattern?

Viele Entwickler halten es für einen besonderen Vorteil von Vue.js, dass es die Clientlogik sauber von der Serverlogik trennt. Das wird vor allem im Vergleich zu serverseitigen Webframeworks wie Spring oder JSF genannt. Bei genauerem Hinsehen ist das Argument jedoch auf Sand gebaut. JSF und Spring (und praktisch alle anderen Webframeworks) propagieren die Trennung von Client und Server. Der Unterschied zu Vue.js ist nur, dass es bei Vue.js physisch unmöglich ist, versehentlich direkt auf eine Serverkomponente zuzugreifen. Wenn Client und Server auf unterschiedlichen Rechnern laufen, ist es einfach, die Trennung durchzuhalten. Aber die physikalische Trennung ist keine notwendige Voraussetzung für die logische Trennung.

Am Ende des Tages ist es nur eine Frage der Disziplin, mit der die vereinbarte Architektur eingehalten wird. Die meisten JSF-Anwendung folgen dem MVC-Schema, obwohl sie es leicht umgehen könnten. Genauso kann man den Vue.js-Client mit dem Node.js-Server in einem Electron-Installationspaket verpacken, ohne die Schichtentrennung zu verletzen. Es ist, wie gesagt, nur eine Frage der Disziplin. Und wenn die Anwendung auch offline laufen soll, ergibt ein lokaler Node.js-Server eine Menge Sinn. Er stellt die Business-

Anzeige

logik bereit, die ansonsten nur auf dem entfernten und nicht immer verfügbaren Server bereitgestellt wird.

Reality Check: Entwickeln mit electron-vue

Hier gibt es nicht viel zu berichten. Mit electron-vue zu entwickeln, fühlt sich kaum anders an als die normale Vue.js-Entwicklung. Das CLI-Plug-in unterstützt sogar Hot Reloading und sorgt damit für schnelle Turnaround-Zeiten. Ein angenehmes Detail ist, dass die Anwendung im Entwicklungsmodus immer mit geöffnetem Debugger startet. Für die wichtigsten Standardaufgaben sind bereits Skripte in der *package.json* vordefiniert. Unter anderem gibt es Skripte zum Testen und zum Bauen der Installationspakete für Windows, Linux und Mac OS X.

Welche Vorteile hat Electron im Vergleich zur Entwicklung im Browser?

Zurück zur Ausgangsfrage. Seit zwei Jahrzehnten gibt es einen massiven Trend, Desktopanwendungen durch Webanwendungen zu ersetzen. Mittlerweile hat dieser Trend sogar Textverarbeitungen erfasst. Man denke nur an Google Docs, das sich hinter Word und Co. nicht verstecken muss. Wo also liegen die Vorteile einer nativen Desktopanwendung mit Electron?

Electron ist für JavaScript-Entwickler die Chance, aus der Sandbox des Browsers auszubrechen. Für eine Browseranwendung sind viele Low-Level-APIs nicht erreichbar, für Electron schon. Das ist Chance und Risiko zugleich. Daher eine Bitte: Denken Sie immer daran, dass Ihre Electron-Anwendung im Grunde genommen ein Google Chrome ist, auch wenn sie nicht so aussieht. Damit ist Electron ein beliebtes Angriffsziel für Hacker. Am besten lassen Sie keinen Zugriff auf das Internet aus der Anwendung heraus zu. Falls Sie solche Zugriffe doch benötigen, schauen Sie sich bitte das Video „How I Hacked Every Major IDE in 2 Weeks“ von Matt Austin [7] an. So sehr Sie sich selbst wünschen, aus der Sandbox ausbrechen zu können, so wenig wünschen Sie sich, dass ein Hacker einbrechen kann.

Ein großer Vorteil von Electron ist es, Dateien auf dem PC des Anwenders lesen und schreiben zu können. In den letzten fünf Jahren hat dieser Punkt an Bedeutung verloren. Inzwischen ist es möglich, Excel-Dateien mit Drag and Drop in eine Vue.js-Anwendung zu importieren. Mit Electron kann solch eine Datei auch direkt – ohne Benutzerinteraktion – gelesen und geschrieben werden. Wobei Excel nur ein Beispiel ist, das in meiner eigenen Berufspraxis besonders häufig vorkam: Das Gleiche gilt natürlich für alle anderen Dateitypen auch.

Wenn es nur darum geht, Daten zu cachen, hat HTML5 in den letzten Jahren enorm aufgeholt. Wenn wir uns darauf verlassen können, dass unsere Anwender einen hinreichend modernen Browser verwenden, sind wir schon am Ziel. Stichworte sind localStorage, Indexed DataBase und das Sandboxed FileSystem API. Aber es hängt eben sehr vom Browser ab. Ein Internet Explorer 11 – der in vielerlei Hinsicht schon als HTML5-kompatibler Browser durchgehen kann – bietet im besten Fall 50 Megabyte Speicher. In einem meiner aktuellen Projekte war das zu wenig. Electron bietet für solche Fälle einen Ausweg.

Falls Sie das Thema vertiefen wollen: auf www.html5rocks.com gibt es einen Artikel mit vielen weiteren Details und einem Test, mit dem Sie die Grenzen der Browserdatenbanken selbst erkunden können [8].

Electron bietet noch viele weitere Vorteile. Im Prinzip bietet es Ihnen Zugriff auf jede Ressource des PCs, auf dem es läuft. Neben den standardmäßig ausgelieferten Adaptern haben Sie auch die Möglichkeit, selbst Adapter zu schreiben. Meine Liste bleibt daher zwangsläufig unvollständig:

- Sie können auf den Drucker zugreifen: natürlich via PDF und über den Windows-Druckerdialog, und auch auf Spezialdrucker mit proprietären Schnittstellen.
- Sie können die Zwischenablage lesen und in sie schreiben.
- Sie können Icons und Menüs zur Taskleiste hinzufügen.
- Für High-End-MacBooks gibt es ein API für die Touchbar.
- Sie können sogar auf die Events des Energiemanagements reagieren. Beispielsweise können Sie rechenintensive Prozesse stoppen, sobald der Laptop in den Batteriemodus wechselt.
- Sie können die Anzeige maximieren, indem Sie den Frameless Mode aktivieren. Wenn Sie kein Menü brauchen, lassen Sie es weg. Wenn Sie keine Scrollbars brauchen, schalten Sie sie aus. Sie können sogar Fenster mit beliebiger Form verwenden und sind nicht auf die rechteckigen Standardfenster angewiesen [9].

Listing 2

```
// src/main/index.js:
import { app, BrowserWindow } from 'electron';
require('./server');  // <<< add this line
...
app.on('window-all-closed', () => {
  app.removeAllListeners();  // <<< add this line
  if (process.platform !== 'darwin') {
    app.quit();
  }
}); ...
// src/main/server.js:
const express = require('express');
const app = express();
app.get('/', function(req, res) {
  res.send('Hello World!');
});
app.listen(3000, function() {
  console.log('Example app listening on port 3000!');
});
```

Womöglich liegt der größte Vorteil von Electron aber ganz woanders. Sie haben als Softwarelieferant die volle Kontrolle über Ihre Laufzeitumgebung. Sie können Ihre Anwendung auf Electron optimieren, statt sie wie bisher eher auf den kleinsten gemeinsamen Nenner auszurichten. Selbst wenn der Unternehmensstandard noch „Internet Explorer 9“ heißt, können Sie mit Electron viele Features von ECMAScript 2016 nutzen. Besser noch, es ist Ihnen überlassen, wann Sie den (unsichtbaren) Browser aktualisieren. Das bedeutet zusätzlich Verantwortung für Sie: Sie müssen aktiv werden, wenn gravierende Sicherheitslücken bekannt werden. Aber es bedeutet eben auch die Sicherheit, dass Ihre Anwendung nicht durch ein unangekündigtes Securityupdate auf die Nase fällt.

Zu guter Letzt ist ein großer Vorteil von Electron, dass die Anwendung lokal installiert ist. Das hat die gleichen Vorteile wie das lokale Installieren einer App auf dem Smartphone: Die Anwendung startet schneller, weil der Download wegfällt, und sie ist auch im Funkloch verfügbar. Aus der Sicht der für den Betrieb zuständigen IT-Abteilung ist das allerdings auch ein Nachteil. Wie werden Updates verteilt?

Welche Nachteile hat Electron?

Auch wenn die Softwareverteilung der Hauptgrund war, warum sich die Browseranwendungen durchgesetzt haben: für dieses Problem gibt es heutzutage Lösungen. Für Electron gibt es (mindestens) zwei Autoupdatesysteme. Das Prinzip kennen Sie schon von vielen anderen Anwendungen: Die Anwendung schaut regelmäßig auf einem Server nach, ob es ein Softwareupdate gibt, und installiert es von selbst. Weitere Einzelheiten finden Sie im Artikel von Philipp Langhans [10].

Unangenehmer ist die Tatsache, dass die Installationspakete plattformspezifisch sind. Im besten Fall müssen Sie immer drei unterschiedliche Installationspakete bauen und ausliefern: für Windows, für Linux und für macOS. In der Praxis stellt sich oft heraus, dass dieser Ansatz nicht funktioniert. Der große Vorteil von Electron – der einfache Zugriff auf Betriebssystemressourcen – stellt sich hier als Nachteil heraus. Entweder Sie investieren viel Aufwand, um sicherzustellen, dass die Anwendung keine Besonderheiten eines einzelnen Betriebssystems verwendet, oder Sie machen aus der Not eine Tugend und legen sich von vorneherein auf ein bestimmtes Betriebssystem fest. Da es bei Electron darum geht, die Vorteile des Desktop-PCs zu nutzen, ist das auch meine Empfehlung: Optimieren Sie Ihre Anwendung für das Betriebssystem Ihres Kunden.

Chromium bringt rund 30 MB Dateigröße mit. Auf der Festplatte eines modernen PCs spielt das keine Rolle. Es kann problematisch werden, wenn ein Update verteilt werden soll. Starten alle Anwender das Update gleichzeitig, ist das Netzwerk schnell überlastet. Auch dafür bietet der Artikel von Philipp Langhans [10] eine Lösung.

Anfangs erwähnte ich die Komplexität der Software, die in den 2000er Jahren ein K.-o.-Kriterium für die Desktopanwendungen war. Häufig verhedderten sich die Programmierer damals bei dem Versuch, auf Ereignisse wie Focus Lost zu reagieren. Mit Vue.js sieht das viel entspannter aus. Der Programmierer braucht sich nicht mehr selbst um das Data Binding zu kümmern, sondern das macht unser Framework für uns. Wir können diesen Punkt von der Liste der Nachteile streichen.

Das Thema Security ist so wichtig, dass ich es nochmal erwähnen möchte. Es ist leicht zu vergessen, dass ein vollwertiger Browser hinter Electron steckt. Sie können mit Electron beliebige Anwendungen auf dem Desktop starten. Seien Sie bitte entsprechend vorsichtig, falls Ihre Anwendung einen URL im Internet bereitstellt. Sie wollen schließlich nicht, dass ein Angreifer Anwendungen auf dem lokalen PC startet! Denkbare Angriffsvektoren sind der Autoupdater und der zuvor erwähnte Express-Server.

Zusammenfassung

Meines Erachtens nach ist die Kombination aus Electron und electron-vue eine sehr attraktive Möglichkeit, die komfortablen Features eines Desktop-PCs zu nutzen, ohne das angenehme Programmiermodell von Vue.js aufzugeben. Sie müssen nur ein paar Besonderheiten wie Security und den Autoupdater im Hinterkopf behalten. Es ist schade, dass die meisten IT-Abteilungen Desktopanwendungen von vornherein ausschließen. Gibt es doch keine bessere Möglichkeit, Ihre Anwendung mit den restlichen Anwendungen Ihres Kunden zu integrieren!

Stephan Rauh arbeitet als Senior Consultant bei der OPITZ CONSULTING Deutschland GmbH und befasst sich seit Jahren mit Angular, JSF und generell mit UI-Technologien. Er ist durch seinen Blog (http://www.beyondjava.net) und sein Open-Source-Framework BootsFaces bekannt geworden.

Links & Literatur

[1] Nativefier: https://github.com/jiahaog/nativefier/

[2] electron-vue: https://legacy.gitbook.com/book/simulatedgreg/electron-vue/details/

[3] Hinweise für Windows-Anwender: https://simulatedgreg.gitbooks.io/electron-vue/content/en/getting_started.html#a-note-for-windows-users

[4] Swanson, Kirsten: „Electron + Vue.js“: https://medium.com/@kswanie21/electron-vue-js-f6c40abeb625

[5] Ayo, Ogundipe Samuel: “Create a Desktop Quiz Application Using Vue.js and Electron”: https://scotch.io/tutorials/create-a-desktop-quiz-application-using-vue-js-and-electron/

[6] Express-Server in Electron starten: https://gist.github.com/maximilian-lindsey/a446a7ee87838a62099d/

[7] Austin, Matt: „How I Hacked Every Major IDE in 2 Weeks“: https://www.youtube.com/watch?v=_IE6U34pXnE

[8] Kitamura, Eiji: „Working with quota on mobile browsers”: https://www.html5rocks.com/en/tutorials/offline/quota-research/

[9] Fenster mit beliebiger Form: https://github.com/electron/electron/blob/master/docs/api/frameless-window.md/

[10] Langhans, Philipp: „Electron: The Bad Parts“: https://hackernoon.com/electron-the-bad-parts-2b710c491547/