Technische Dokumentation

Teamprojekt 2016

OR-GraphX 1.0

Versionshistorie

Version	Datum	Bemerkung
1.0	28.06.16	Erstellung – Felix Roth

Teammitglieder:

Fox, Alex - 288942

Linse, Matthias - 287019

Pupel, Robert - 289118

Roth, Felix - 289043

Inhaltsverzeichnis

1. Einrichtung der Entwicklungsumgebung

Als Entwicklungsumgebung wird ein Unix Betriebssystem mit mindestens 2GB Arbeitsspeicher empfohlen.

1.1 Bitbucket Zugangsdaten

Für die Weiterentwicklung am OR-GraphX steht ein Bitbucket Account zur Verfügung von dem das aktuelle Repository geklont werden kann:

Zugangsemail:	orgraphx@gmail.com
Zugangspasswort:	orgraphx2016

1.2 IDE Empfehlung

Als IDE kann „Webstorm“ von der Firma Jetbrains kostenlos mit einer Studenten Lizenz genutzt werden. Diese eignet sich hervorragend für die Entwicklung von JavaScript Projekten. Es unterstützt darüber hinaus die Entwicklung mit AngularJS, nodeJS und TypeScript.[1]

Das zuvor von Bitbucket geklonte Repository kann im Webstorm einfach importiert und somit das Programm weiterentwickelt werden.

[1] Vgl. https://www.jetbrains.com/webstorm/ - abgerufen am 26.06.2016

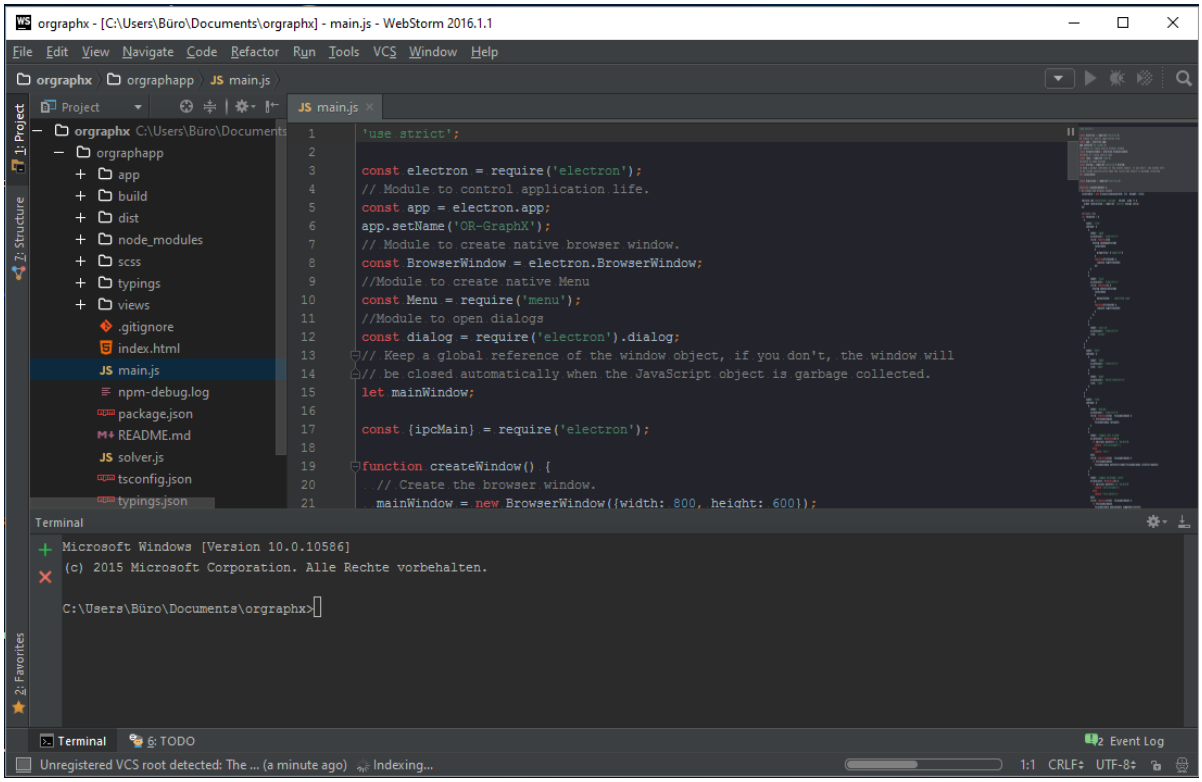

Abbildung 1 - Screenshot: Webstorm + Konsole

1.3 Node.js

Eine aktuelle node.js Version wird zusätzlich benötigt zum Testen und Builden der Anwendung. Diese kann von folgender Webseite geladen werden: https://nodejs.org/en/

Innerhalb unseres Teamprojektes wurde die Version 5.12 genutzt.

2. Projektstruktur

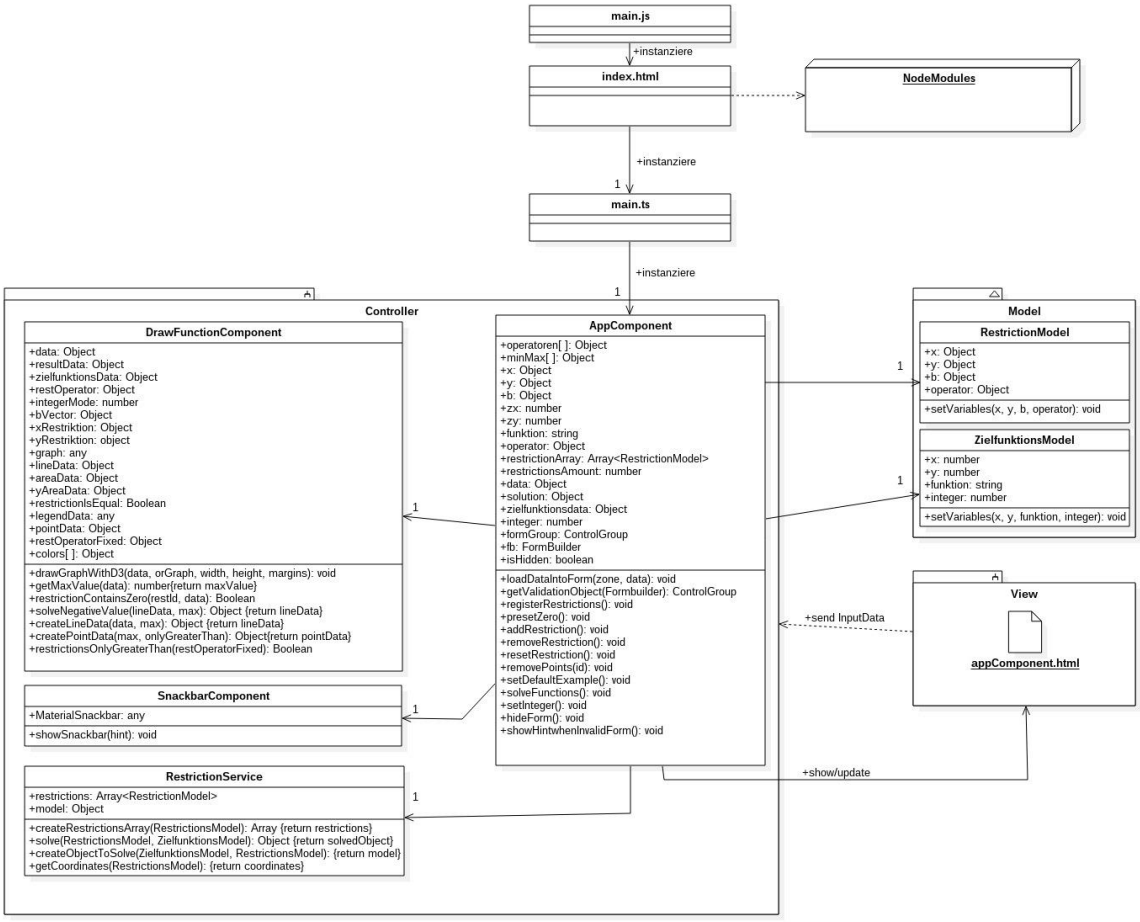

Abbildung 2 - UML Diagramm OR-GraphX

Das Projekt ist mit Hilfe des OpenSource Frameworks Electron von Github entwickelt worden. Das Framework erlaubt ein plattformübergreifendes Entwickeln von Desktop-Apps mit Hilfe von Webtechnologien. So stellt Electron sozusagen den nativen Rahmen der Applikation dar, welches einen Browser zur Darstellung der eigentlichen Webbausteine nutzt.[2]

Über die main.js des Electron-Frameworks werden hierbei die benötigten Node-Module wie beispielsweise, Angular2 und Angular Material, sowie Webpack zum automatischen kompilieren der TypeScript-Dateien und dem komprimieren des daraus entstanden JavaScript-Codes, geladen.

Des Weiteren wird aus den Node-Modulen noch der Solver zur Berechnung des Optimums, D3.js für das spätere zeichnen der Grafik, sowie das Designframework MaterialDesignLite, geladen.

Nach dem Model-View-Controller-Prinzip wurde innerhalb von Electron mit Hilfe von Angular2 in Verbindung mit Typescript und HTML die eigentliche Oberfläche der Anwendung entwickelt.

Die „main.ts“ instanziiert hierbei sozusagen den Controller. Der Controller besteht aus AppComponent, DrawFunctiontionComponent, SnackbarComponent, sowie dem RestrictionService. Die AppComponent fungiert hierbei als Dreh- und Angelpunkt, welcher die Inputdaten aus der View empfängt, die Models instanziert und je nach Interaktionen die anderen Komponenten des Controllers ansteuert.

Mit den Daten aus dem Input wird das ZielfunktionsModel, sowie das Restriktionsmodel instanziiert und an den RestrictionService übergeben. Der RestrictionService erstellt aus den RestriktionModels und dem ZielfunktionsModel wiederum ein Object das via Request an den Solver übergeben wird, der das Problem berechnet und ein Objekt mit der entsprechenden Optimal Lösung zurückliefert. Durch zu vor definierte Deklarativen in der AppComponent werden die berechneten Daten anschließend an die DrawfunctionComponent übergeben. Diese enthält sämtliche Logik zur Generierung der

[2] Vgl. http://electron.atom.io/ - abgerufen am 29.06.2016

Zeichnung aus den zuvor berechneten Daten. Mit Hilfe der Zeichenlibrary D3Js entsteht aus den Daten eine SVG-Grafik, die durch ein HTML-Template anschließend in der View dargestellt wird.

Zusätzlich beinhaltet der Controller noch eine SnackbarComponent, welche zur Hinweisausgabe in der View dient. So überprüft die AppComponent über eine Angular2 Funktion beispielsweise zur Laufzeit, ob die Eingabe des Users valide ist. Ist dies nicht der Fall wird ein Hinweis über ein HTML-Template in der View ausgegeben.

Wie oben schon erwähnt stellt die View die grafische Oberfläche, auf der der User interagieren kann, dar. Sie ist durch ein HTML Dokument realisiert. Mit Hilfe von Sass-Dateien wurde dieses dann noch optisch angepasst.

3. Client

Im Folgenden wird erklärt wie ein OR-Graph-X Build erstellt werden und anschließend auf dem Client ausgeführt werden kann.

3.1 Client Build erstellen

Um einen Build für einen HTWG Client zu erstellen ist es notwendig, das npm Modul „electron-packager“[3] zu installieren:

```
# npm install electron-packager -g
```

Anschließend kann das Projekt über folgenden Befehl gebulidet werden:

```
# electron-packager <sourcedir> <appname> --platform=<platform> --
arch=<arch> [optional flags...]
```

[3] https://github.com/electron-userland/electron-packager - abgerufen am 29.06.2016

3.2 Client Installation

Der im Schritt vorher erzeugte Build muss anschießend nur gepackt und auf den jeweiligen Client kopiert werden.

4. Server

Im Folgenden wird erläutert, welche Anpassungen an dem Build getätigt werden müssen um daraus eine webfähige Version der OR-GraphX Anwendung zu erstellen.

4.1 OR Web Server Build erstellen

Um das Electron basierte Programm auf einen WebServer zu übertragen sind folgende Schritte notwendig:

1. Es muss ein Separates Verzeichnis für die Web Anwendung angelegt werden.

2. In das Verzeichnis des Web Projekts müssen alle Dateien, bis auf die Electron relevanten Dateien, des Desktop Programms kopiert werden.

3. Der Angular Code muss in der Komponente app.component.ts und restrictions.service.ts für den WebServer angepasst werden.

4. Die Kommunikation zwischen Electron main Prozess und renderer Prozess (Angular Code) muss z.B. durch HTTP Request/Response ersetzt werden.

5. Die Logik für die Kommunikation der Speicher- und Ladefunktion befindet sich derzeit ausschließlich im Konstruktor der app.component.ts Komponente. Die Schnittstelle für die Berechnung liegt in der restrictions.service.ts.

6. Es muss beachtet werden, dass sich die node_modules auf einem Windows Server nicht installieren lassen. Deshalb muss das Verzeichnis auf den Server inklusive aller node_modules kopiert werden.

4.2 OR Web Server Integration

Bei einer Neuinstallation des Servers muss „node.js“ installiert werden. Dies kann von der Webseite in der aktuellsten Version heruntergeladen werden.

Der zuvor erstellte Build wird anschließend auf den Server kopiert und kann über folgenden Befehl gestartet werden:

```
# npm <Directory>/server.js
```

Anschließend sollte die Anwendung über folgende Adresse + VPN HTWG erreichbar sein:

http://orweb.in.fhkn.de:8000/

4.3 OR Web Server Dienstinstallation

Um die Anwendung als Dienst, welcher bei Systemstart automatisch ausgeführt wird, im Microsoft Server zu hinterlegen muss noch das npm Modul „node-windows“[4] folgendermaßen installiert werden:

```
#npm install -g node-windows

# npm link node-windows
```

[4] https://www.npmjs.com/package/node-windows - abgerufen am 29.06.2016

Damit der Dienst angelegt wird, sollte zusätzlich eine JavaScript Datei erstellt werden:

Service.js

```
var Service = require('node-windows').Service;

// Create a new service object
var svc = new Service({
  name:'ORGraphX',
  description: 'ORGraphX Serverdienst',
  script: 'C:\\Programme\\ORGraphX\\orgraphweb\\server.js'
});

// Listen for the "install" event, which indicates the
// process is available as a service.
svc.on('install',function(){
  svc.start();
});

svc.install();
```

Diese kann über folgenden Befehl ausführt werden:

```
# node service.js
```

Über die Dienste im Servermanager kann überprüft werden ob der Dienst erfolgreich angelegt wurde:

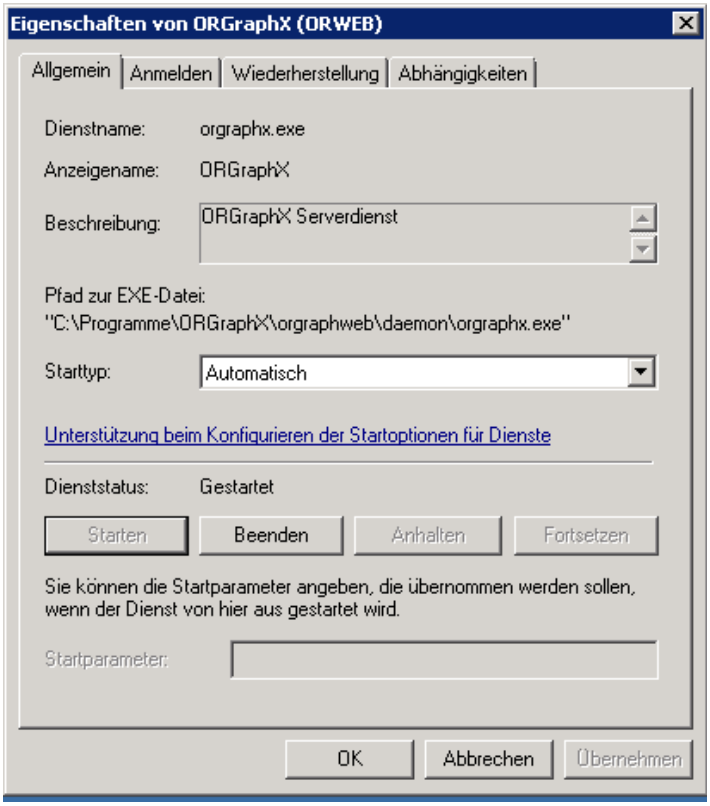

Abbildung 3 - Screenshot: OR-GraphX Dienst auf orwebserver

Abbildungsverzeichnis