2024

Plant-E Based Display & Apps

Software Architecture

PLANT-E GLOW
TADRAŁA,PIOTR P.P.

O-PP-CMK

INHOUDSOPGAVE

1.0 VERSIEBEHEER

Versie	Datum	Aanpassing
0.1	16-04-2024	Eerste opzet van het document
0.2	11-06-2024	C4 model, onderbouwing, bijlagen & Communicatie onderzoek

2.0 INLEIDING

2.1 CONTEXT

Het doel van ons project is het bedenken en realiseren van een kunstinstallatie voor Glow, die gebruik maakt van het Plant-E technologie. Ons concept bestaat uit het creëren van een "display" van planten. Hierbij functioneert elke plant als een pixel door de door bijbehorende LED's aan en uit te zetten. Daarnaast ontwikkelen we een portal/hub met applicaties die gebruik zullen maken van het display via een display controller. Dit document zal het ontwerp van de software architectuur toelichten en onderbouwen.

2.2 SCOPE

Het project bestaat uit de ontwikkeling van twee modules. Ten eerste wordt er een hub ontwikkeld waarop games en apps gedraaid kunnen worden, het voorlopige plan is om dit te met het Electron framework en Angular als frontend te realiseren. Deze applicatie zal communiceren met de tweede module, een 'display controller'. Deze controller, een microcontroller zoals ESP32 of Arduino, ontvangt commando's van de portal over de weer te geven inhoud en zorgt vervolgens voor de visualisatie op het e-planten display.

2.3 HOOFDVRAGEN

- Wat voor modules dinenen er ontwikkeld te worden?
- Welke architectuur is het meest geschikt voor ons project?
- Wat voor communicatie protocol tussen de software en de hardware kunnen we het beste gebruiken?

2.4 DEFINITIES

- **Electron**: Framework voor het ontwikkelen van desktop-applicaties.
- **Angular**: Framework voor het ontiwkkelen van client-sided applicaties.
- **ESP32:** Een microcontroller zoals een Raspberry pi of Arduino
- **Portal**: Desktop-applicatie waarop plant-e apps zullen draaien
- **Dsiplay controller:** ESP32 controller die als proxy tussen de planten en de portal zal functioneren.

- **Plant-E Technologie**: Het concept die het mogelijk maakt om elektriciteit te genereren uit planten.

3. ARCHITECTUUR

3.1 ARCHITECTUUR TYPE

Ons project zal geen database gebruiken en is daarmee volledig client-sided. Daarom heb ik gekozen voor een N-Layer Architectuur, die een duidelijk onderscheid maakt tussen de presentatie die de gebruikers en administrators zien, de logica en communicatie binnen de apps die in de portal draaien, en het hardware dat het display beheert.

3.2 OVERZICHT

PRESENTATION LAYER

Een gebruikersinterface platform ontwikkeld met Electron en Angular. Functineert als de interactieve hub voor de beheerders van het systeem om applicaties te starten en beheren.

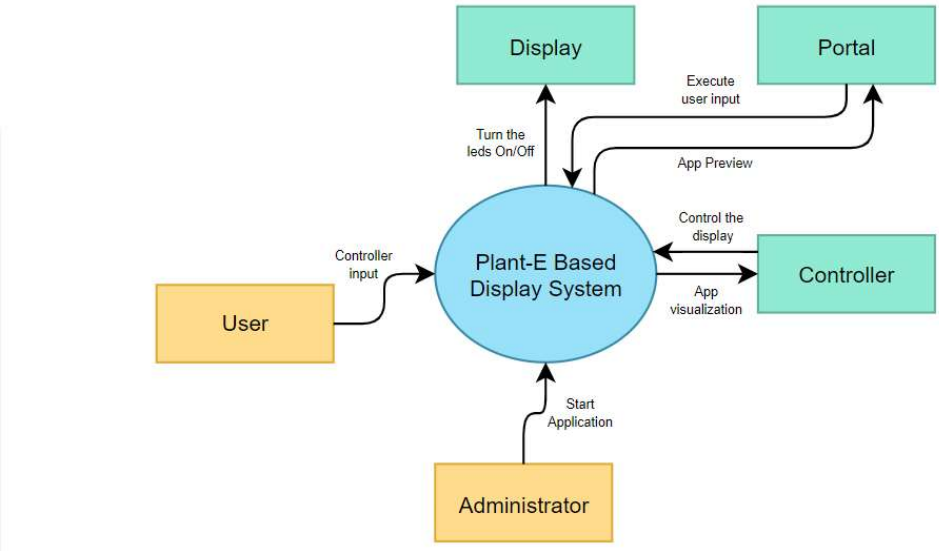
LOGIC & COMMUNICATION LAYER

Deze laag bevat de logica voor applicaties die binnen het systeem functioneren, zoals het spel Pong. Daarnaast vindt in deze laag communicatie met de hardwarecontroller plaats. De verbinding tussen de hub en de display link controller verloopt via USB serial ports, Wi-Fi of (Low Energy) Bluetooth.

HARDWARE LAYER

De microcontroller die als de display link controller functioneer. Deze laag vertaalt ontvangen commando's naar acties op het e-planten display door individuele planten-LED's aan en uit te schakelen, overeenkomstig met de visuele input vanuit de Logic Layer.

3.3 SYSTEM CONTEXT DIAGRAM



Het diagram toont de algemene architectuur van het Plant-E Based Display System en illustreert de interacties tussen gebruikers, beheerders, de portal en de controller, en hun connectie.

USER

Users zijn spelers die games gaan spelen op het display. Zij leveren input aan de portal, die vervolgens in de logische layer wordt verwerkt. Deze input wordt uiteindelijk naar de controller gecommuniceerd en gevisualiseerd op het display.

ADMINISTRATOR

Administrators zijn de beheerders van het systeem. Hun kunnen in de portal tussen de applicaties switchen die op het display getoond worden.

DISPLAY

Het display is een grid aan E-Planten die als pixels functioneren op het ‘display’.

PORTAL

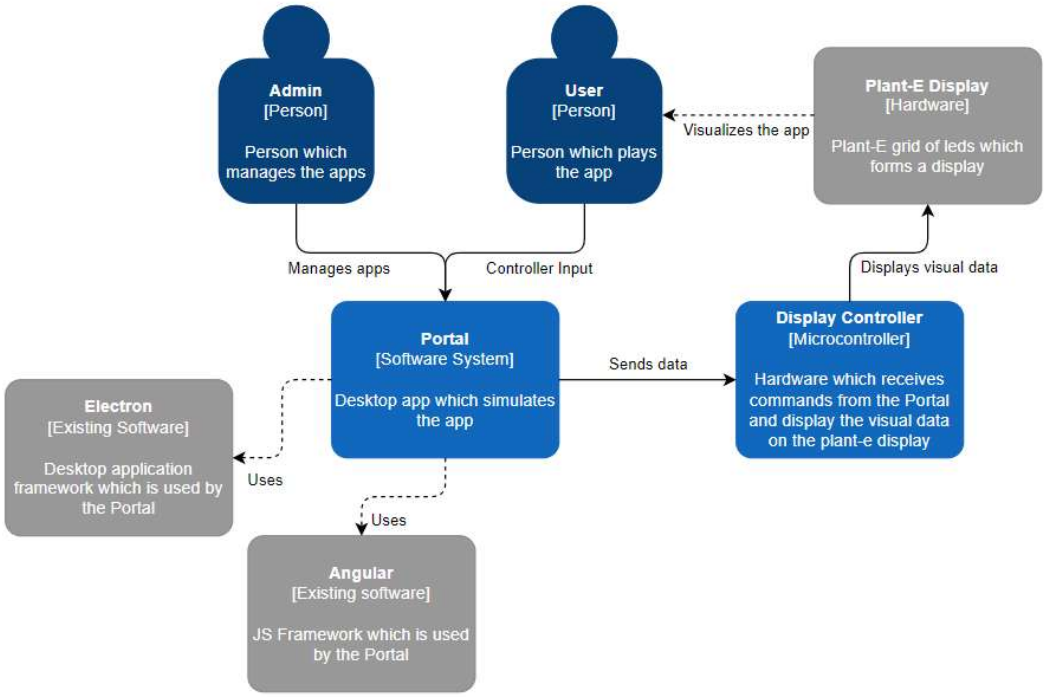
Desktopapplicatie voor het aanzetten en beheren van de apps die om het display weergegeven worden. Een app/game wodt in de portal gesimuleerd en wordt er vervolgens visuele data naar de controller gestuurd.

CONTROLLER

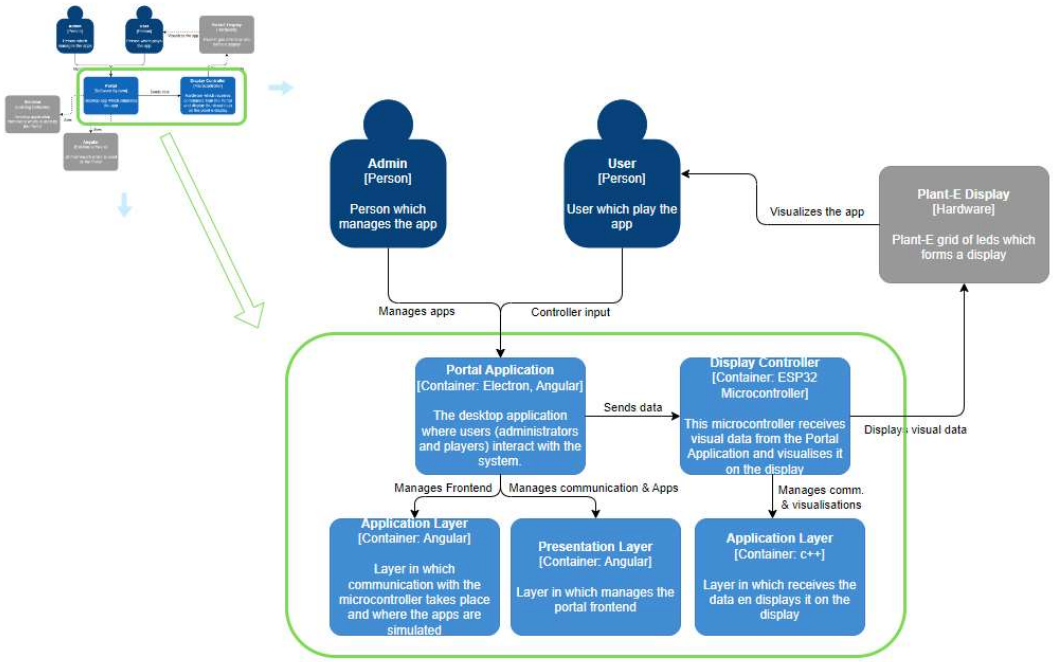
De hardware controller ontvangt visuele data vanuit de portal ontvangt. Vervolgens zorgt de controller ervoor, door middel van een algoritme, dat het zichtbaar wordt op het display.

3.4 C4 MODEL

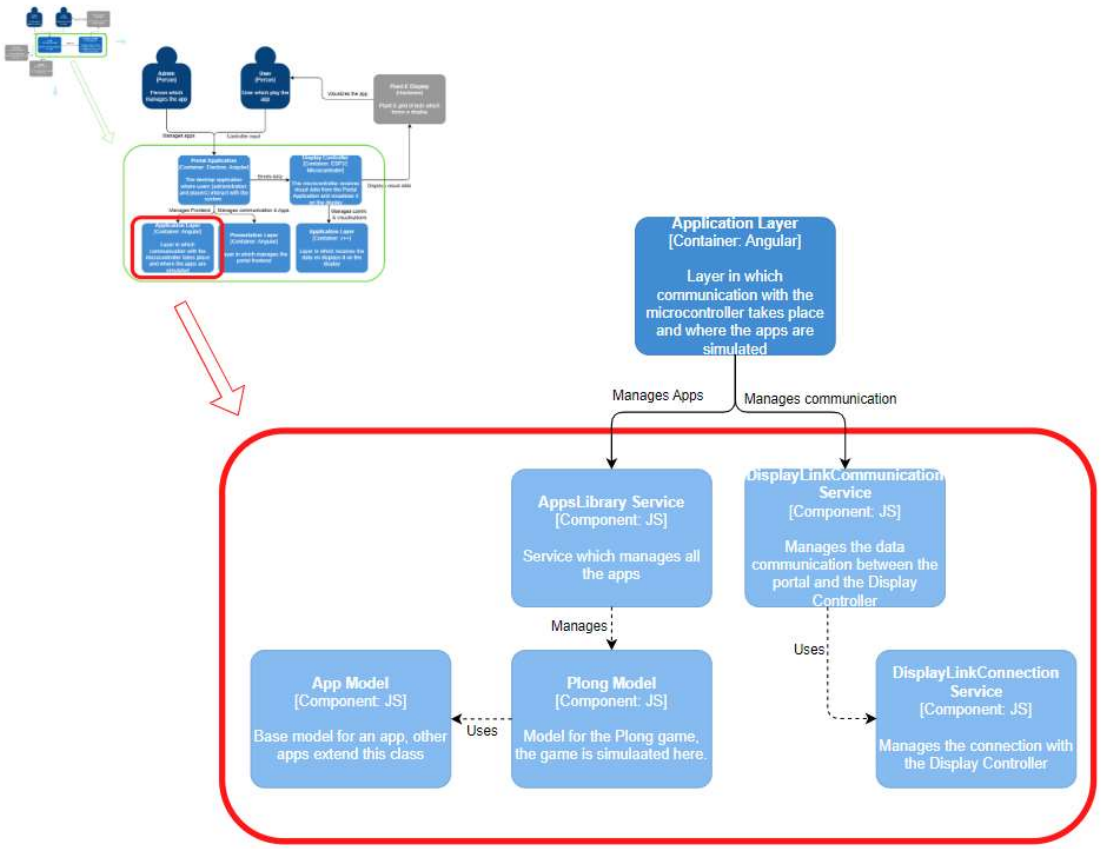
SYSTEM CONTEXT



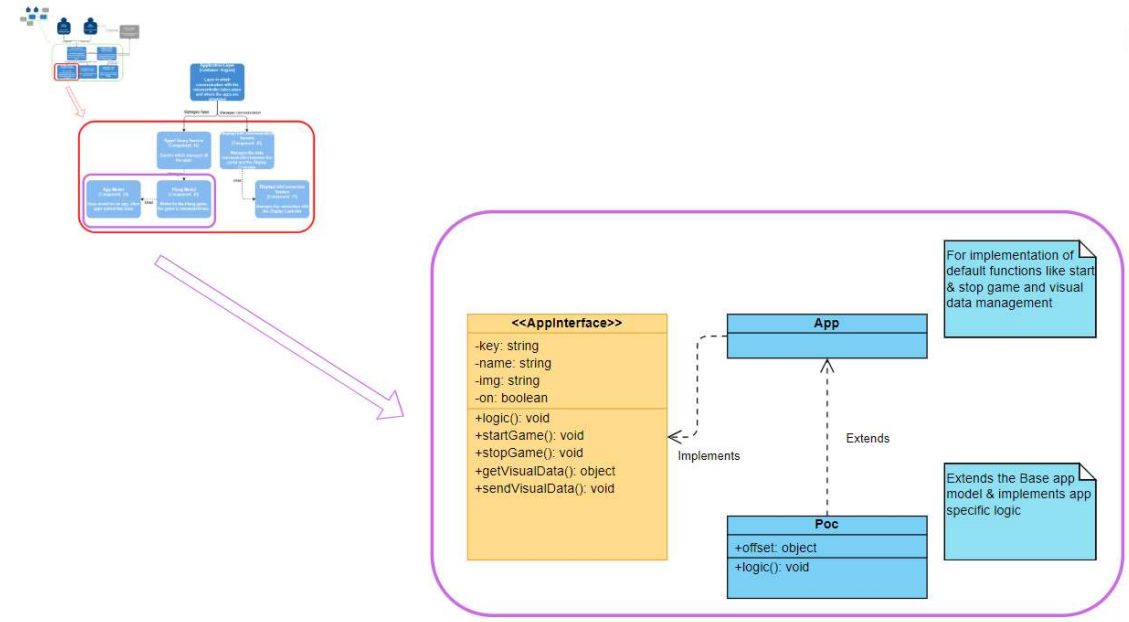
CONTAINERS



COMPONENTS



CODE


<<AppInterface>>
-key: string
-name: string
-img: string
-on: boolean
+logic(): void
+startGame(): void
+stopGame(): void
+getVisualData(): object
+sendVisualData(): void
Implements
App
Extends
Poc
+offset: object
+logic(): void
For implementation of default functions like start & stop game and visual data management
Extends the Base app model & implements app specific logic

4.0 PRESENTATION LAYER

4.1 ELECTRON

WAT IS ELECTRON

Electron is een open-source framework die gebruik maakt van Chromium en Node.js waarin desktopapplicaties gebouwd kunnen worden. Electron functioneert als een shell waarin HTML, JavaScript en TypeScript webapplicaties worden omgezet in desktopapplicaties. Electron kun je ook goed combinere met JS frameworks zoals Angular, React en Vue. Bekende applicaties zoals Microsoft Teams, Discord, GitHub Desktop en WhatsApp zijn met Electron ontwikkeld.

- *Marcin Dryka, What is Electron JS*

WAAROM ELECTRON

Electron maakt het mogelijk om webapplicaties als desktopapplicatie te runnen. Aangezien Electron gebruik maakt van Node.js is het mogelijk om alle NPM-modules te gebruiken, wat de mogelijkheden van applicaties uitbreidt. Electron ondersteunt ook cross-platform builds, waardoor je met enkele aanpassingen apps kunt builden naar zowel Windows, macOS als Linux, en eventueel, na enkele extra tweaks, ook naar mobiele platformen.

4.2 ANGULAR

WAT IS ANGULAR

Angular is een open-soure framework, beheerd door Google, voor het ontwikkelen van dynamische web-applicaties. Angular maak gebruikt van TypeScript en ondersteunt NPM libraries.

- *Beeproger, Angular: wat is het en hoe werkt het.*

WAAROM ANGULAR

Na onderzoek bleek dat het weinig uitmaakt welk JS-framework wordt gebruikt. Daarom heb ik gekozen voor het framework waar ik het meest bekend mee ben, namelijk Angular.

• KilcoDer, Best front-end framework to pair with electron...

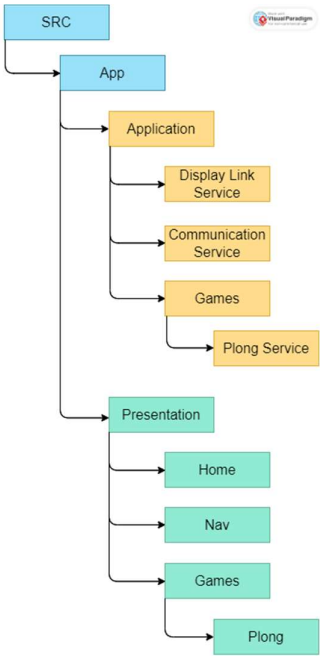
5.0 LOGIC & COMMUNICATION LAYER

5.1 ANGULAR SERVICES

Voor het onderdeel logic & communication zal gebruik worden gemaakt van Angular Services. Angular Services zijn Classes met functionaliteiten die geïnjecteerd kunnen worden in andere componenten. Dit maakt het mogelijk om alle logica te scheiden, waardoor deze niet afhankelijk is van andere layers.

Voor een voorbeeld van de implementatie zie **6.1 Angular Serivces Implementatie**

Presentation & Logic layers zullen als volgt opgesplitst worden binnen Angular:



- *Angular, Architecture Services*

5.2 COMMUNICATION

Voor de communicatie tussen de portal (logica) en de display controller (hardware) overwegen we de volgende opties:

- **HTTP Requests**: Gebruik van een REST API om direct te communiceren met de display link controller, dit zal het stroomverbruik van de controller verhogen.
- **Serial Ports**: Communicatie via een USB-kabel, nadeel hiervan is dat de controller dicht bij het apparaat moet liggen waarop de portal applicatie draait.

- **Bluetooth:** Communicatie via (Low Energy) Bluetooth, nadeel hiervan is dat de hardware over Bluetooth moet beschikken.

Na een uitgebreid onderzoek (zie **Communicatieprotocol Onderzoek & Advies**), heb ik de volgende conclusie kunnen trekken:

Door eerst naar de potentiële opties te kijken, vervolgens alle requirements in kaart te brengen, en tot slot test implementaties voor alle genoemde opties te creëren, is er een overzichtelijke matrix ontstaan. Hieruit kunnen we duidelijk concluderen dat Serial ports het meest optimale protocol is voor ons concept.

- *Feteh Ali Shahrukh Khan, Communication between ESP32 and Postman*
- *Lastminuteengineers, ESP32 Basics, Bluetoth Low Energy (BLE)*
- *Vincent Lab, How to communicate with Serial Port in Node.js*

6.0 HARDWARE LAYER

6.1 CONFIGURATIE

Onze setup zal een microcontroller gebruiken, zoals een ESP32 of een Arduino, die geprogrammeerd wordt in C++. De microcontroller functioneert als een proxy die visuele data van de portal ontvangt, deze vervolgens schaalt naar de ingestelde resolutie van de controller en weergeeft op het display. Ter controle stuurt de controller de aangepaste visuele data terug naar de portal, waar het als preview gebruikt kan worden.

6.2 DATA VISUALISATIE

De microcontroller ontvangt een array met x- en y-coördinaten van de leds die moeten worden aangezet. Op basis van deze coördinaten stuurt de microcontroller een PWM-signaal naar het led-grid om de juiste leds aan en uit te zetten.

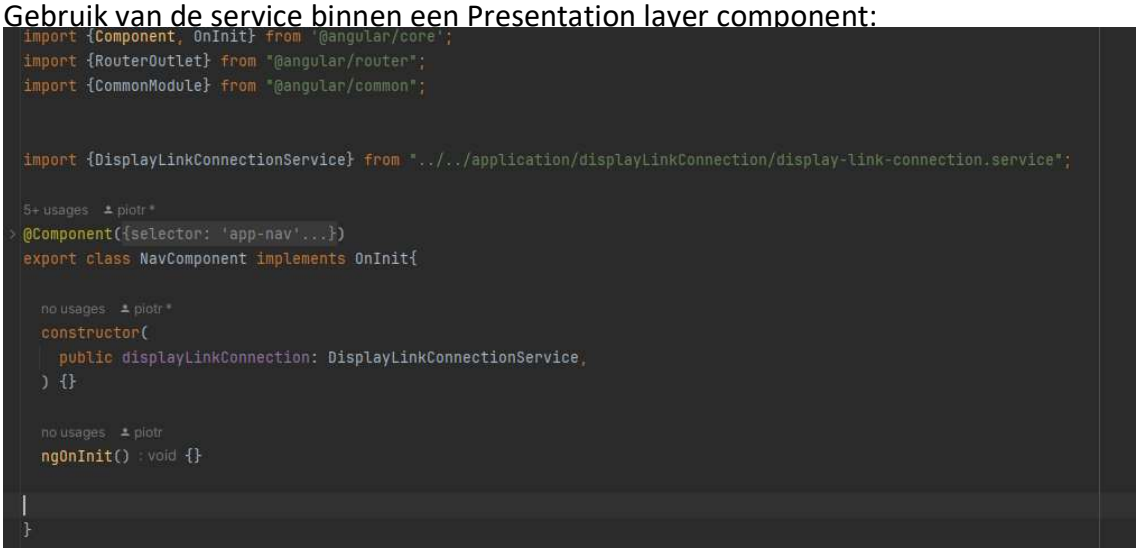
BIJLAGEN

ANGULAR SERVICES IMPLEMENTATIE

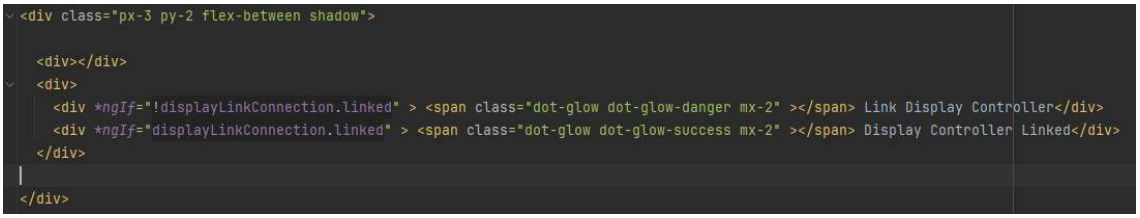
Voorbeeld van een Angular Service Class:

```
import { Injectable, NgZone } from '@angular/core';

5+ usages  piotr *
@Injectable({
  providedIn: 'root'
})
export class DisplayLinkConnectionService {

  public linked : boolean  =  false;
  public device : null  = null;

  no usages  piotr
  constructor(
    private zone: NgZone,
  ) {
    this.listenToUsb();
  }

  1 usage  piotr *
  listenToUsb() : void {
    window.electronAPI.receiveDataFromElectron( channel: 'display-controller-connected',  func: (device: any) : void  => {
      this.zone.run( fn: () : void  => {
        this.linked = true;
        this.device = device;
      })
    })
    window.electronAPI.receiveDataFromElectron( channel: 'display-controller-disconnected',  func: () : void  => {
      this.zone.run( fn: () : void  => {
        this.linked = false;
        this.device = null;
      })
    })
  }
```

Gebruik van de service binnen een Presentation layer component:



```
import {Component, OnInit} from '@angular/core';
import {RouterOutlet} from "@angular/router";
import {CommonModule} from "@angular/common";


import {DisplayLinkConnectionService} from "../../application/displayLinkConnection/display-link-connection.service";

5+ usages  piotr *
@Component({selector: 'app-nav'...})
export class NavComponent implements OnInit{

  no usages  piotr *
  constructor(
    public displayLinkConnection: DisplayLinkConnectionService,
  ) {}

  no usages  piotr
  ngOnInit() : void {}


}
```



```
<div class="px-3 py-2 flex-between shadow">

  <div></div>
  <div>
    <div *ngIf="!displayLinkConnection.linked" > <span class="dot-glow dot-glow-danger mx-2" ></span> Link Display Controller</div>
    <div *ngIf="displayLinkConnection.linked" > <span class="dot-glow dot-glow-success mx-2" ></span> Display Controller Linked</div>
  </div>

</div>
```