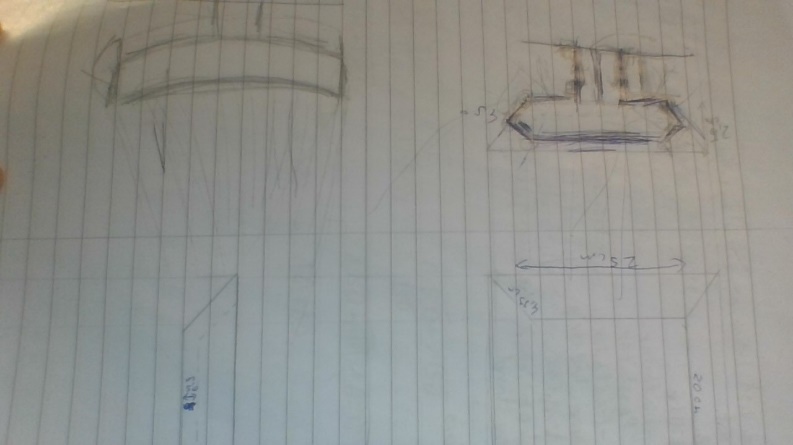
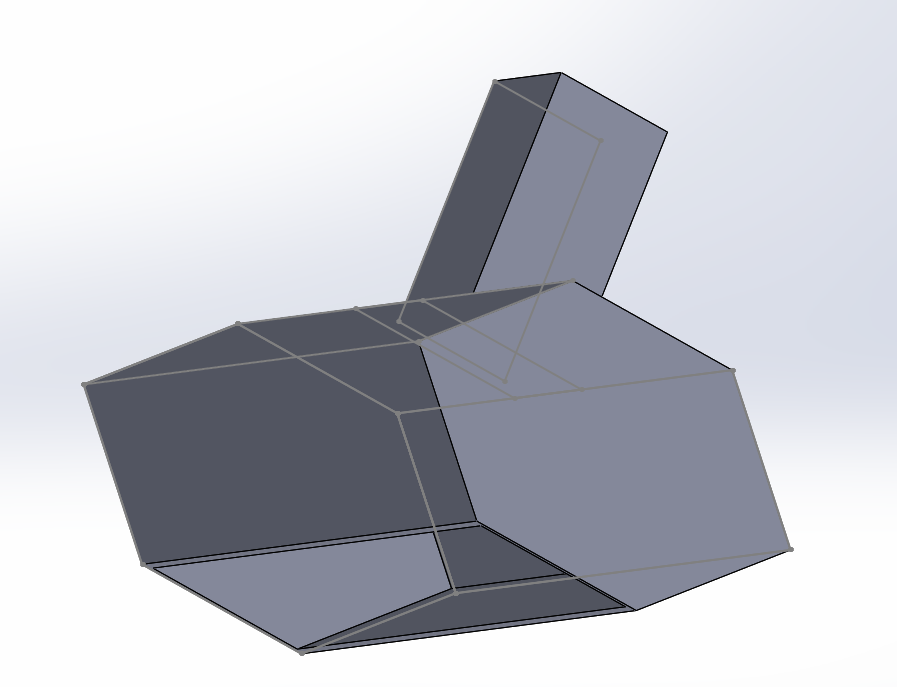
**En Ljus Ide av Grupp 6**

**Funktioner**

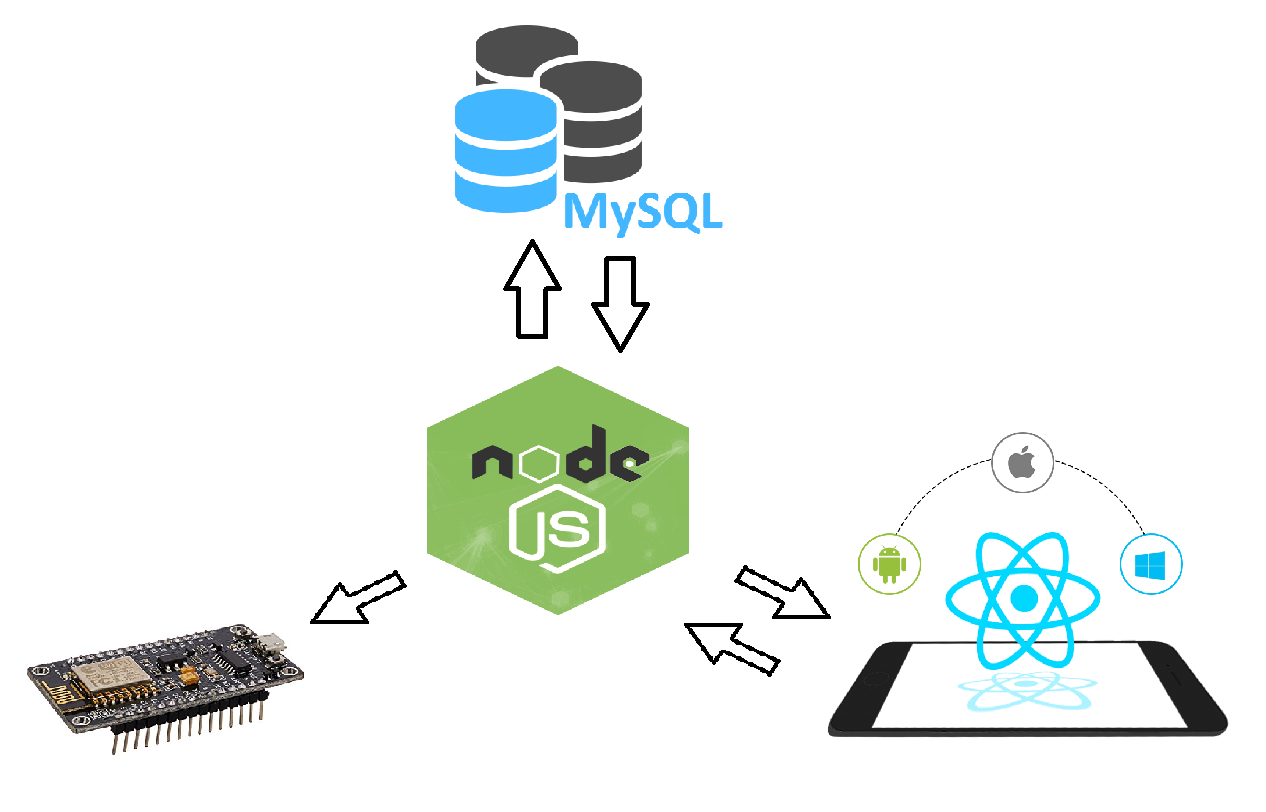
Vi tänkte att vi ville göra det lätt att stänga av lampan, ändra ljusstyrka och färg på ljuset. Vi ville kunna justera hur mycket kallt respektive varmt ljus vi har och justera styrkan på vardera.

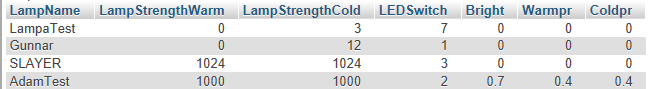
**Modellen**

****I början tänkte vi oss en mer traditionell lampa, d.v.s en triangel iprincip. Vi testade att vända upp och ner på den men vi valde att inte använda den designen, Problemet vi stötte på var dels att vi inte gillade utseendet men främst så strävade vi efter en smartare lösning som lät oss få både direkt och indirekt ljus på ett snyggt sätt. Flera olika designer skapades men den vi fastnade för var en avlång lampa med hexagonformade sidor. Med denna design kunde vi få direkt ljus på ett indirekt sätt. Om LEDstripsen sitter på insidan kan vi få ljuset att reflekteras neråt, vilket gör att man inte behöver oroa sig för att råka kolla rakt in i starka LEDs. Men ifall lampan LEDsen var instängda i lampan skulle inget vi inte få något uppåt mot taket. För att lösa detta tänkte vi först sätta flera strips ovanpå laman för att få ljus, men beslutade att vi inte ville göra det. Dels för att vi trodde att man skulle kunna se LEDstripsen, vilket vi inte ville, och delvis för att det skulle innebära mer och krångligare koppling och fler sladdar. Dock så hade man kunnat ha mer kontroll över ljuset ifall man kunde styra LEDsen inuti och ovanpå separat, men vi bestämde oss för att det inte var värt. För att lösa problemet valde vi istället att använda plexiglas till två av väggarna som var vinklade uppåt. På detta vis skulle vi kunna få ljus både uppåt och neråt med samma LEDstrips. Detta gjorde att vi undvek en sladdhärva, kunde använde färre strips och dessutom så ser det snyggt ut.

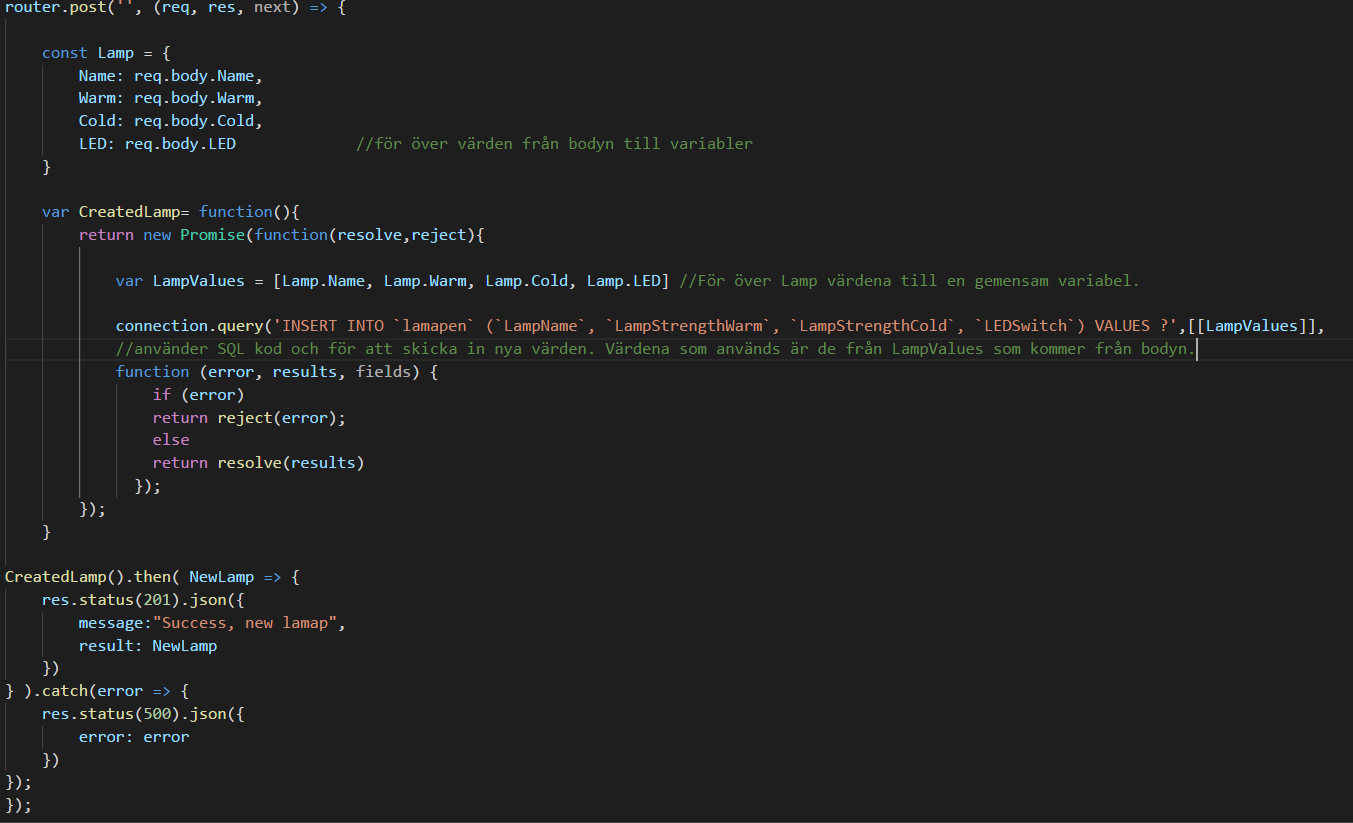
Ovanpå armaturen har vi en avlång hållare som vi tänkte att man skulle använda för att fästa lampan i taket. Den hade även syftet att hålla våran mikroprocessor dold.

**Hur funkar det?**

****

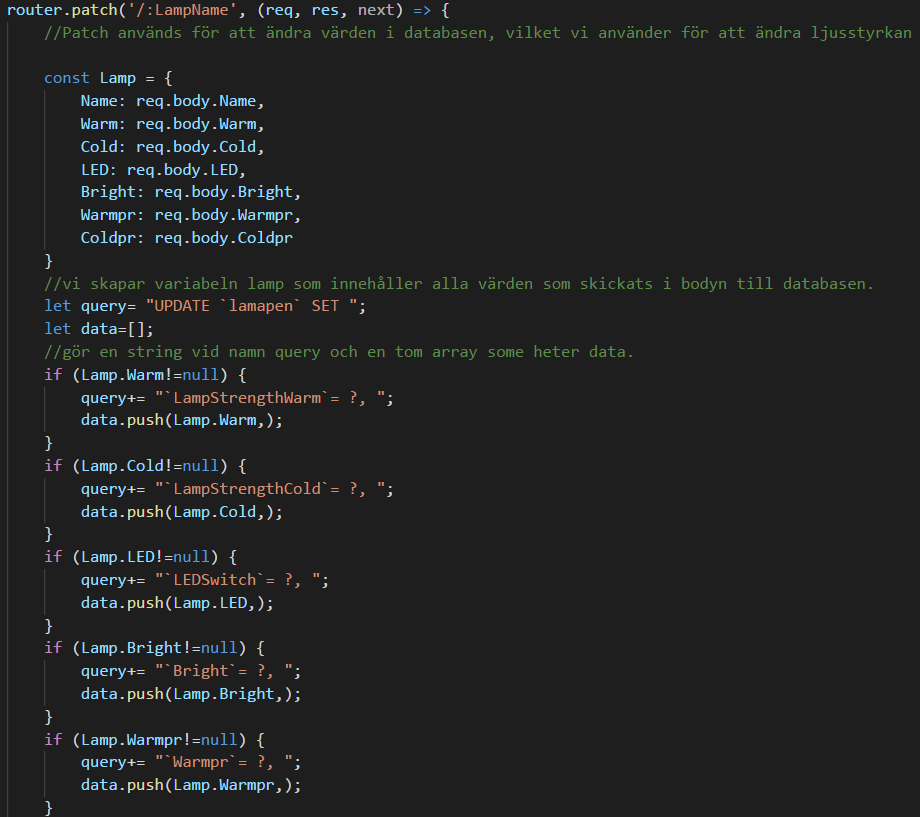
Hela systemet bygger på vår backend. Man kan se det som huvudstationen i vårat system. Alla delar bygger använder backend. En av delarna är våran databas. Det är en MySQL databas och det är där vi har alla våra värden till lampan. Backended kan skicka en förfrågan till databasen om att få värden och databasen skickar värden till backended. Vi kan även använda oss av våran backend för att ändra i databasen för att t.ex ändra värden eller ta bort värden. Så här ser databasen ut:

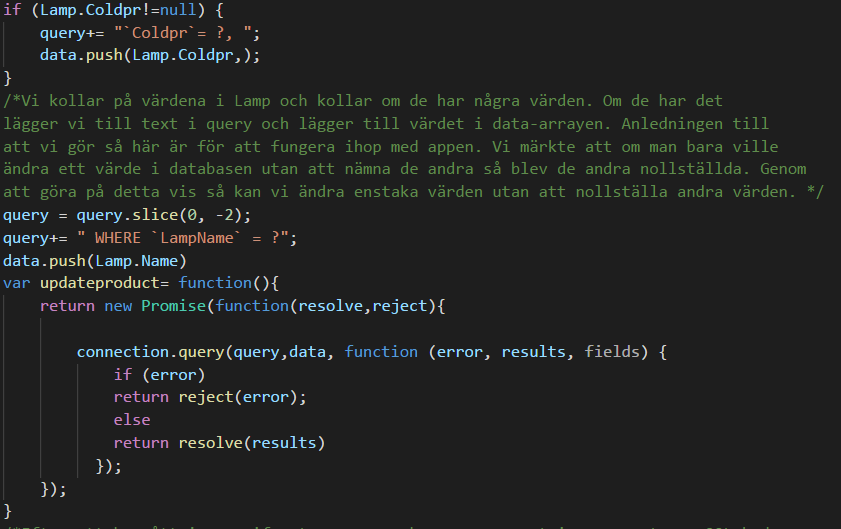
Backenden är som sagt vår huvudstation i projektet. Det använder node.js och använder sig av GET, PATCH, POST och DELETE för att kommunicera med databasen. Den kan sedan vidarebefordra denna information till mobilappen eller mikroprocessorn.

Backenden funkar på så vis att vi skickar en GET request varje sekund. Detta gör att vi hämtar alla värden i databasen. Genom att göra detta varje sekund så ser vi till att backenden alltid har de nyaste värdena. Ifall vi vill ha värden från en specifik rad så får vi skicka namnet på raden vi vill ha. Backenden kommer då leta efter en rad som har samma namn som namnet man skickade bland värdena den har hämtat. Ifall något matchar kommer den skicka tillbaks raderna som matchar. Ifall vi vill ändra värden, antingen genom att ändra värden, lägga till eller ta bort värden. För att göra detta skickas värden som vi själva väljer till backenden som sedan kommunicerar med databasen. För både PATCH och POST så skriver vi in alla värden vi vill ha och skickar det till backenden. Skillnaden är att med PATCH bestämmer namnet vilken rad man ändrar och för POST så är namnet bara ytterligare ett värde. DELETE fungerar på så vis att man väljer ett namn och sedan raderas alla rader med namnet.

Ifall vi kollar på koden så ser vi att det inte är så avancerat. Vi blandar helt enkelt lite SQL-kod (Databasens språk) med värdena som skickas med i våran body. GET funktionen behöver inte ens någon body, då den hämtar allt som finns att hämta. Allt bygger på att vi skapar olika funktioner som utförs beroende på vad man vill göra. Alla funktioner fungerar i princip likadant. Vi ansluter till med databasen med våran connect variabel och använder sedan SQL-kod. Sedan vidarebefordras resultatet till användaren.

Den enda funktionen som är annorlunda är patch funktionen. När vi designade appen märkte vi ett problem. Om man bara ville ändra ett värde och lämnade de andra tomma så blev de nollställda. Detta var ett stort problem eftersom att vi inte vill att allt ska försvinna när vi ändrar en sak. Lösningen på detta var att skapa SQL koden utefter var som behövdes. Vi gör detta genom att skapa en String och en tom array. Vi går sedan igenom alla värden som skickats med i bodyn. Om ett värde finns läggs en rad SQL kod till i Stringen och värdet läggs till i arrayen. Om ett värde saknas eller inte nämns så kommer ingen kod/värde läggas till. I slutändan så kommer vi ha en String med SQL kod och tillhörande värden. Sedan funkar det precis likadant som de andra funktionerna med SQL koden och värdena.





Mobilappen

I mobilappen finns det tre olika komponenter: En On/Off knapp, En slider för ljusstyrka och en slider för färg (Varmt eller kallt).

fungerar genom att när man trycker på knappen så körs en funktion (GetDataFromServer) som tar värdet LEDSwitch från data basen och lägger in det värdet på en variabel. Sen kör jag funktionen UpdateDataToServer som skickar värdet LEDSwitch 3 om värdet var 0 och värdet 0 om värdet var 3

import React from 'react';

import { StyleSheet, Text, View, TouchableOpacity } from 'react-native';

// Här importar jag allt som behövs för att komponenten ska fungera

export default class App extends React.Component {

constructor(props)

{

super(props)

this.state= {

Ledv: 3

// Här säger jag att jag kommer använda variablen Ledv och att den ska vara 3

}

}

GetDataFromServer =() => {

//Här säger jag vad den här funktionen heter

let self= this;

fetch('http://192.168.0.118:1337/lampa/AdamTest',{

// Här säger jag vilken ip-adress backenden som jag ska använda här

method: 'GET'

// Här säger jag vad jag ska göra med backenden (I det här fallet ta data)

}).then((response) => response.json()).then((responseJSON)=>

{

console.log(responseJSON);

self.setState(

{

Ledv: responseJSON.LEDSwitch

// Här säger jag att variablen Ledv ska ändras till värdet LEDSwitch från databasen

});

console.log(this.state.Ledv);

}).then(this.UpdateDataToServer())

// Här säger jag att när GetDataFromServer är klart så ska UpdateDataToServer köras

}

UpdateDataToServer =() => {

//Här säger jag vad den här funktionen heter

const { Ledv } = this.state;

// Här säger jag att Ledv ska användas i funktionen

    fetch('http://192.168.0.118:1337/lampa/',{

// Här säger jag vilken ip-adress backenden som jag ska använda här

method: 'PATCH',

// Här säger jag att jag ska uppdatera värden till databasen

      headers: {

        'Accept': 'application/json',

        'Content-Type': 'application/json',

      },

      body: JSON.stringify({

        Name: "AdamTest",

LED: (3-Ledv)

// Här säger jag vad alla värden ska vara

       })

    }).then((response) => response.json()).then(responseJSON => {

      console.log(responseJSON);

    }).catch((error)=>{

      console.log(error);

    });

}

Sliderna fungerar att varje gång man ändrar värdet på slidern så ändras värdet och GetDataFromServer körs. Där tar brightness slidern värden för färger från databasen och färg slidern tar brightness värden från databasen. Sedan så kör båda slidern UpdateDataToServer där jag både uppdaterar Brightness/Färg och värdena för de varma och kalla ledstrippen. För att få värdena till ledstrippen så tar jag 1024(Maxvärdet)\*Brightnessvärdet(0-1)\*Färgvärdet(0-1)

(Brightness)

import React from "react";

import Slider from "react-native-slider";

import { AppRegistry, StyleSheet, View, Text } from "react-native";

// Här importar jag allt som behövs för att komponenten ska fungera

export default class Component25 extends React.Component {

constructor(props)

{

super(props)

this.state= {

value: 0.5,

Warmpr: 1,

Coldpr: 1,

Warm: 1024,

Cold: 1024,

Bright: 1

// Här sätter jag alla variabler jag behöver i komponenten

}

}

GetDataFromServer =() => {

//Här säger jag vad den här funktionen heter

let self= this;

fetch('http://192.168.0.118:1337/lampa',{

// Här säger jag vilken ip-adress backenden som jag ska använda här

method: 'GET'

// Här säger jag vad jag ska göra med backenden (I det här fallet ta data)

}).then((Response) => response.json()).then((responseJSON)=>

{

console.log(responseJSON);

self.setState({

Warmpr: responseJSON.Warmpr,

Coldpr: responseJSON.Coldpr

// Här säger jag att variablerna Warmpr och Coldpr ska ändras till värdena Warmpr och Coldpr från databasen

});

}).then(this.UpdateDataToServer())

// Här säger jag att när GetDataFromServer är klart så ska UpdateDataToServer köras

}

UpdateDataToServer =() => {S

//Här säger jag vad den här funktionen heter

const {value, Bright, Warm, Cold, Warmpr, Coldpr} = this.state;

// Här säger jag vilka varibler som används i den här funktionen

fetch('http://192.168.0.118:1337/lampa/',{

// Här säger jag vilken ip-adress backenden som jag ska använda här

method: 'PATCH',

// Här säger jag att jag ska uppdatera värden till databasen

headers: {

'Accept': 'application/json',

'Content-Type': 'application/json',

},

body: JSON.stringify({

Name: "AdamTest",

Bright: value,

Warm: (1024\*(value\*Warmpr)),

Cold: (1024\*(value\*Coldpr))

// Här säger jag vad alla värden ska vara

})

}).then((response) => response.json()).then(responseJSON => {

console.log(responseJSON);

}).catch((error)=>{

console.log(error);

});

};

render() {

return (

<View style={styles.container}>

<Slider

value={this.state.value}

onValueChange={value => { this.setState({ value }); this.GetDataFromServer();}}

// Här säger jag att variblen value ska bli värdet på slidern och att GetDataFromServer ska köras

/>

<Text>

Colour

</Text>

{/\* Här säger jag vad som ska stå under slidern \*/}

</View>

);

}

}

(Färg)

import React from "react";

import Slider from "react-native-slider";

import {

StyleSheet,

View,

Text,

} from "react-native";

// Här importar jag allt som behövs för att komponenten ska fungera

export default class Component13 extends React.Component {

constructor(props)

{

super(props)

this.state= {

value: 0.5,

Bright: 1,

Warm: 1024,

Cold: 1024,

Warmpr: 0,

Coldpr: 0

// Här sätter jag alla variabler jag behöver i komponenten

}

}

GetDataFromServer =() => {

//Här säger jag vad den här funktionen heter

let self= this;

fetch('http//192.168.0.118:1337/lampa',{

// Här säger jag vilken ip-adress backenden som jag ska använda här

method: 'GET'

// Här säger jag vad jag ska göra med backenden (I det här fallet ta data)

}).then((Response) => response.json()).then((responseJSON)=>

{

console.log(responseJSON);

self.setState({

Bright: responseJSON.Bright

// Här säger jag att variablen Bright ska ändras till värdet Bright från databasen

});

}).then(this.UpdateDataToServer())

// Här säger jag att när GetDataFromServer är klart så ska UpdateDataToServer köras

}

UpdateDataToServer =() => {

//Här säger jag vad den här funktionen heter

const {value, Bright, Warm, Cold, Warmpr, Coldpr} = this.state;

// Här säger jag vilka varibler som används i den här funktionen

if (value > 0.5) {

this.setState({ Warmpr: 1 });

console.log(Warmpr)

}

else {

this.setState({ Warmpr: value\*2 });

console.log(Warmpr)

// Här säger jag att om value är större än 0,5 så ska Warmpr bli 1 och om det inte är det ska Warmpr bli value\*2

}

if (1-value > 0.5){

this.setState({ Coldpr: 1 });

}

else {

this.setState({ Coldpr: (1-value)\*2 });

// Här säger jag att om 1-value är större än 0,5 så ska Coldpr bli 1 och om det inte är det ska Coldpr bli (1-value)\*2

}

fetch('http://192.168.0.118:1337/lampa/',{

// Här säger jag vilken ip-adress backenden som jag ska använda här

method: 'PATCH',

// Här säger jag att jag ska uppdatera värden till databasen

headers: {

'Accept': 'application/json',

'Content-Type': 'application/json',

},

body: JSON.stringify({

Name: "AdamTest",

Warmpr: Warmpr,

Coldpr: Coldpr,

Warm: (1024\*Bright\*value),

Cold: (1024\*Bright\*(1-value))

// Här säger jag vad alla värden ska vara

})

}).then((response) => response.json()).then(responseJSON => {

S

console.log(responseJSON);

}).catch((error)=>{

console.log(error);

});

};

render() {

return (

<View style={styles.container}>

<Slider

value={this.state.value}

onValueChange={value => { this.setState({ value }); this.GetDataFromServer();}}

// Här säger jag att variblen value ska bli värdet på slidern och att GetDataFromServer ska köras

/>

<Text>

Brigtnes

</Text>

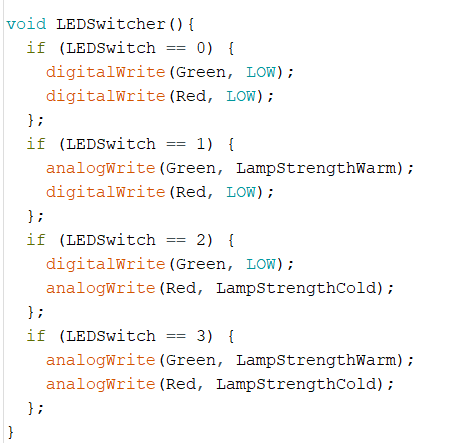
{/\* Här säger jag vad som ska stå under slidern \*/}

</View>

);

}

}

Själva LEDStrippsen i lampan styrs utav en mikroprocessor (en NodeMCU ESP8266) som vi har laddat upp arduinokod till. Denna har internetuppkoppling vilket låter den kommunicera med backenden trådlöst. Det funkar så att mikroprocessorn ber om att få en värden från backenden genom att be den att göra en get request. Backenden kommer då skicka en GET-requst till databasen, få värden och sedan skicka de till mikroprocessorn. Mikroprocessorn läser sedan av värden den får och justerar LEDStrippsen utefter värdena. För att styra våra LEDs använder vi oss våra värden LampStrengthWarm, LampStrengthCold och LEDSwich. Båda LampStrength värdena avgör hur starkt de olika LEDstripparna lyser och LEDSwitch avgör vilka stripps som lyser. Vid 0 lyser ingen, vid 1 lyser en, vid 2 den andra och när värdet är 3 lyser båda sorterna av LEDstrip. Vi kunde göra detta med hjälp av en enkel metod som helt enkelt läser av värdet på LEDSwitch och justerar LEDsen efter det. Vi använder oss av analogWrite istället för digitalWrite eftersom att analogWrite har inbyggd pulsbreddsmodulering, vilket innebär att vi kan styra hur starkt vi vill att lampan ska lysa. Vi sätter styrkan efter värdena vi fått från backenden. Våra värden går mellan 1024 och 0, då 1024 är starkast och 0 är släckt.

Då vi använder node.js till backenden så kommer våra värden vara i json format. Det är viktigt att tänka på eftersom att jsonobjekt är i ett sådant format att vi inte kan använda de som de är i arduinokoden.

**Utvärdering**

Den största lärdomen vi tagit från detta projekt är vikten i att planera. Projektet flöt på ganska bra tack vare att vi hade planerat vem som skulle göra vad, men det hade kunnat förbättras om vi hade planerat bättre. Vi märkte att det blev lite stressigt framåt slutet pågrund av att vi inte hade börjat med appen. Uppskattningen var svår och det vi trodde skulle vara enklast och ta minst tid, att bygga armaturen, visade sig vara det mest tidskrävande och vår planering funkade inte perfekt pågrund av det. På en individuell nivå har vi självklart blivit bättre på det vi jobbat med. Vi såg till att jobba parallellt så att vi var så produktiva som möjligt.

Om vi haft mer tid hade vi velat lägga till en statistiksida på appen. Med en graf och någon tabell och möjlighet att få lampan att justera sig själv efter medelvärden och liknande. Detta var ingen prioritering från början och vi insåg att det skulle bli svårare än vi tänkte oss, men ifall vi haft tiden hade det varit kul att ha åtminstone statistiksidan.

**Lampans Framtid**

Just lampan vi har byggt kanske inte skulle sälja tack vare det billiga materialet och att appen endast fungerar när man kör koden. Dessutom så är den inte helt perfekt, det finns springor och ojämnheter i bygget. Men om man skulle bygga lampan ordentligt och appen förbättrades så att den funkar närsomhelst hade nog lampan kunnat sälja. Den har en effektiv design som ger mycket indirekt ljus och direkt ljus. Det hjälper också att den är lätt att använda med en app som man enkelt kan sätta på lampan, ändra ljusstyrka och färg.