En ljus idé

Målgrupp för rapporten: Era spetskompisar (En läsare som har samma kunskaper som er men som inte gjort det ni har gjort.

Omfattning: Läsaren ska kunna göra en fungerande kopia av er konstruktion och förstå er lösning.

Uppkopplad belysning

Teknisk dokumentation  
HT18

Grupp 5  
Jesper Jansson, Zarah Kadhim Lundin, Brian Nguyen

Teknik  
Joakim Flink  
Andreas Jillram  
Daniel Åkerlund

Innehåll

[1 Funktionsbeskrivning 3](#_Toc531169302)

[2 Systembeskrivning 4](#_Toc531169303)

[3 Detaljbeskrivning 5](#_Toc531169304)

[3.1 Backend 5](#_Toc531169305)

[3.2 Armatur 5](#_Toc531169306)

[3.3 Frontend 5](#_Toc531169307)

Bilagor

[Bilaga 1 Kod](#_Toc531168031)

[Bilaga 2](#_Toc531168032) Wireframes och Tillståndsgraf

Figurer

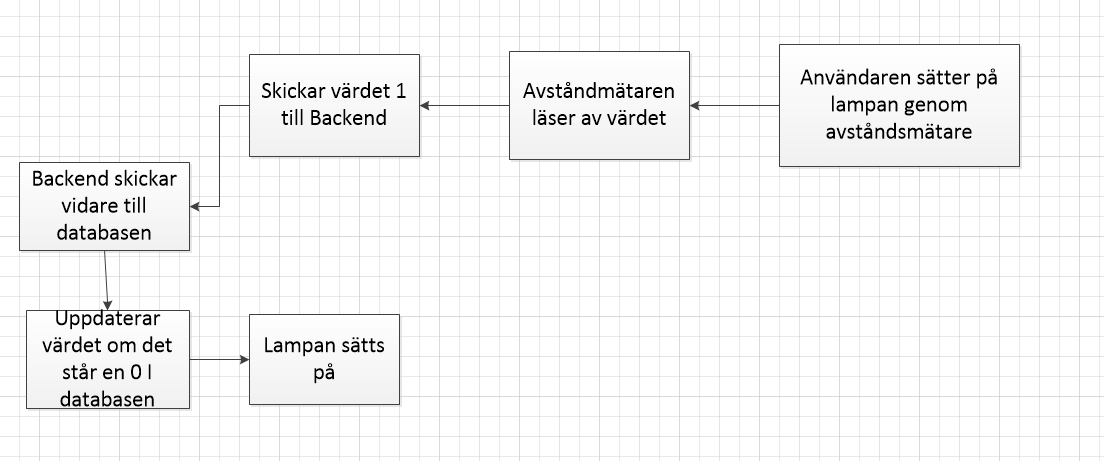
[Figur 1 Figurtext 3](file:///G:\Ämnen\Datorkunskap\Textdokument\Repetition%20Word\Facit\Repetition%20av%20Word%20-%20formaterad.docx#_Toc531167890)

[Figur 2 Figurtext 4](file:///G:\Ämnen\Datorkunskap\Textdokument\Repetition%20Word\Facit\Repetition%20av%20Word%20-%20formaterad.docx#_Toc531167891)

# Funktionsbeskrivning

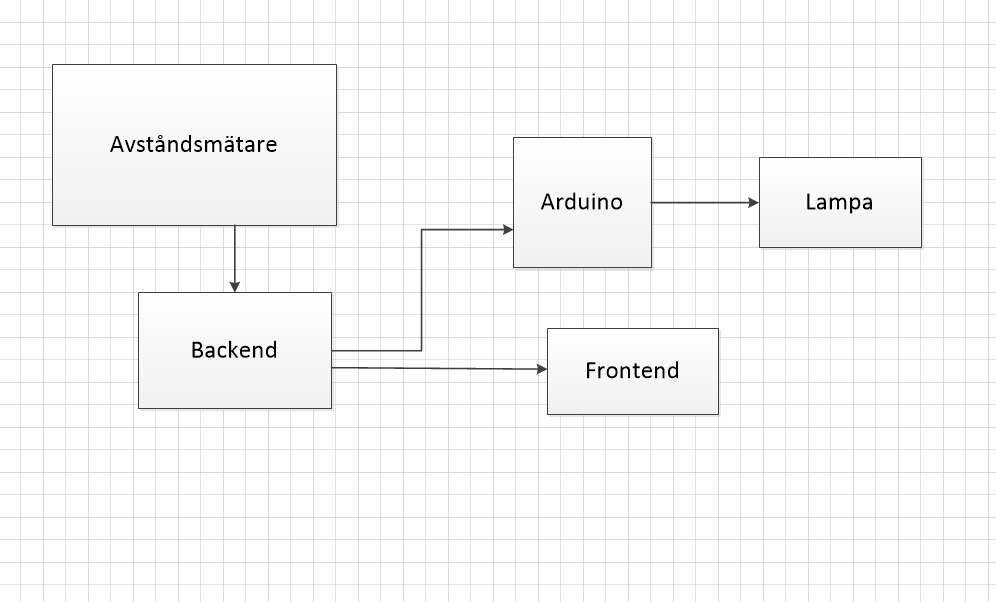
* Appstyrning
* Sensorstyrning
* (Ändra färgtemperatur)

Vi har valt att göra en lampa som går att styra med hjälp av en app och rörelsesensor. I appen ska användaren få tillgång till att modifiera ljusstyrkan med hjälp av en slider, samt sak användaren kunna både sätta på och av lampan, med en av och på knapp i appen. Man ska även manuellt ändra ljusstyrkan efter behov, rörelsesensorn har samma funktion som appen, där du lyfter handen för att få starkare ljusstyrka och sänker den för att sänka styrkan. Du ska även kunna både släcka och tända lampan med en handrörelse. I databasen hittar du rader där det står varm och kall, där hade vi tänkt kunna ändra mellan de två olika färgtemperaturerna. Databasen ger möjlighet till detta, men kopplingen behöver se annorlunda ut för att ge tillgång till detta.



En enkel tillståndsgraf som förklarar hur det går till när användaren sätter på lampan.

# Systembeskrivning

Backend är i princip vad som får hela systemet att stå på två ben. Både arduino och appen är byggt PATCH och GET funktionerna. Så när en användare med hjälp av t.ex slidern i appen sätter på lampan, skickar appen ett värde på antigen 1 om den ska på, eller 0 om den ska stängas av, vidare via backend, där backend sen uppdaterar värdet de får in för att ändra 0 till en 1 för att sätta på den. Backend har också en postfunktion då den kan ta värdena den får och posta nya värden i databasen som istället för att ändra de redan existerande värdena i databasen.

Blockschema

Arduino som har samma funktion som mobil applikationen skickar på likande sätt värdena via backend som skickar vidare och lagrar det i databasen.

# Detaljbeskrivning

## Backend

Backend är vad som ligger i mitten av kommunikationen. Det är där all information lagras och får in värden från Arduino vilket den sen skickar vidare till databasen. Backend innehåller funktionerna GET, POST och PATCH. För GET kod funktionen se bilaga 1.3 för fullständig kod, och bilaga 1.4 för både patch och post. Så den kan ta värden från databasen samt mikrokontrollen. Med de värdena den får in kan den patcha dem, vilket betyder att den uppdaterar värdena den får in och ger den nya värden. Den kan även posta nya värden till databasen.

## Armatur

Vår armatur är en hexagon. Vi har valt att göra den figuren för att det skulle bli en snygg design. Vi har även kommit på en rätt så smart lösning för att gömma hårdvaran så den inte syns. Vi har lagt plast över plexiglaset så att hårdvaran är gömd. Det som var svårast var att räkna ut måtten på bitarna, då måtten måste vara nästan perfekta, eftersom vi hade ont om material. På toppen plattan har vi gjort krokar på kantbitarna som en kedja hänger på. I hexagonen ligger ledstripsarna som sitter på tre bitar cellpannå i lampan. Där har vi tänkt så att fyra kalla ledsrips och en varm strip riktar neråt för att få direkt ljus ner. Medans de finns två varma ledstrips på var sida av cellpannån som lyser indirekt ljus åt sidorna för att få lite av en mysbelysning, nu blev det inte riktigt som vi tänkt oss med att kunna switcha mellan de två färgtemperaturerna.

## Frontend

Frontenden skickar värden för att antingen stänga av eller slå på lampan samt skicka ett av användaren bestämt värde för ljusstyrka. Även en tom skärm finns för att i framtiden bygga på appen med en timerfunktion. Denna finns till viss del redan med tabell i databasen och en påbörjad utökning av backenden, filen timerpatch.js. För att kunna ha dessa funktioner krävs en knapp för av/på funktion, samt en slider för att öka eller sänka ljustyrka. En ytterligare screen behövs för timern och därmed även sätt att navigera mellan dessa. Se wireframe:frontend under bilagor, wireframes och tillståndsgraf. Se även grafisk design av appen som bifogade bilder av moqups.

För koden behövs en funktion som körs när man trycker på av/på knappen. Se bifogad kod 1.1 för exakt kod hur detta görs. Först behövs det hämtas data för om lampan är på eller av i nuläget. Vi har representerat det med siffrorna 0 och 1. Datan hämtas med getfunktion för den lampa i tabellen som vi använder oss av. Sedan sparas värdena från tabellen i de variabler som följande kod kommer att använda sig av. Om lampan är på och värdet 1 finns i databasen vill vi med ett knapptryck ändra det till 0, detsamma gäller fast tvärtom om lampan är av. Det värde som ska patchas beror på det värde som hämtas. Värdena patchas, lägg märke till att värdet för ljusstyrka är detsamma som hämtades, denna vill vi inte ändra på eller nollställa. Namnet är det samma som vi hämtade med ändelsen i adressen. Kod 1.2 visar funktionen för slidern som går att ändra värdet på i appen. Det värdet patchas sedan med ett knapptryck.

## Arduino

Arduino gör så att mjukvara kan kommunicera med hårdvara. Genom Arduino kunde vi få mikrodatorn att ansluta sig till ett nätverk och hämta data. Datan är lagrad i en databas som vi kan komma åt genom vår iot server och vår backend. Skicka, hämta och uppdatera värden i databasen kan vi göra genom en url, där vi skickar och hämtar i datatypen string. Vi använde även arduino för att skapa funktioner ex som att styra lampan genom en avståndsmätare.

Bilaga 1 Kod

* 1. Av/på-knapp Frontend

//Knapp för att slå av och på lampan

import React from 'react';

import {

    StyleSheet,

    View,

    TouchableOpacity,

    Image,

} from 'react-native';

export default class Avpåknapp extends React.Component {

    constructor(props) {

        super(props)

        this.state = {

            avpå:0,

            ljusstyrka:0,

        }

    }

    UpdateDataToServer = () =>{ //patchfunktion

            let self = this;

var adress='http://iot.abbindustrigymnasium.se:3000/grupp5/lampa'; //variabel för addressen backenden ligger på

             fetch('adress', {

                method: 'GET' //anävnder metoden get

             }).then((response) => response.json()) //Gör om resultatet till json

             .then((responseJson) => {

         self.setState( //ändrar till de nya värdena

                {

                avpå = responseJson.påav,

                ljusstyrka = responseJson.ljusstryrka,

             }).catch((error) => { //Fångar fel

                console.error(error);

             });

         });

                if (avpå = 1) { // om lampan är på och värdet i databasen då är 1, vill vi att lampan ska stängas av

                var knappvalue = 0;// knapp value blir 0

                }

                else {

                knappvalue = 1; //annars ska lampan tändas

                }

     fetch(adress, { // adressen för backenden

         method: 'PATCH',   //uppdaterar avpå värdet

         headers: {

         'Accept': 'application/json',

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

         },

         body: JSON.stringify({

         name: "lampa",

         påav: knappvalue,  //skickar med avpå

         kall: 0,

         varm: 0,

         ljusstryrka: ljusstyrka,

         })

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

         .then((responseJson) => {

     console.log(responseJson);

         alert("Lampan slås på eller stängs av");

         }).catch((error) => {

         console.error(error);

         });

         }

render() {

if (!this.props.visible) {

return false;

}

return (

<View

style={styles.component}

>

<View style={styles.layouts}>

                 <View style={styles.layout}>

                        <TouchableOpacity style={styles.item1}

                         onPress={this.UpdateDataToServer}>

                        <Image style={styles.imagestyle} source={require('./Button.png')} />

                        </TouchableOpacity>

                 </View>

</View>

</View>

);

}

}

const styles = StyleSheet.create({

    component: {

     width: '100%',

     flexDirection: 'row',

     paddingLeft: 7.5,

     paddingRight: 7.5,

        paddingTop: 20,

     paddingBottom: 7.5,

    },

    layouts: {

     flexDirection: 'row',

     flexWrap: 'wrap',

    },

    layout: {

        width: '100%',

        position: 'relative',

     alignItems: 'center',

     justifyContent: 'center',

    },

imagestyle: {

        height: 100,

        width: 100

    },

});

* 1. Slider Frontend

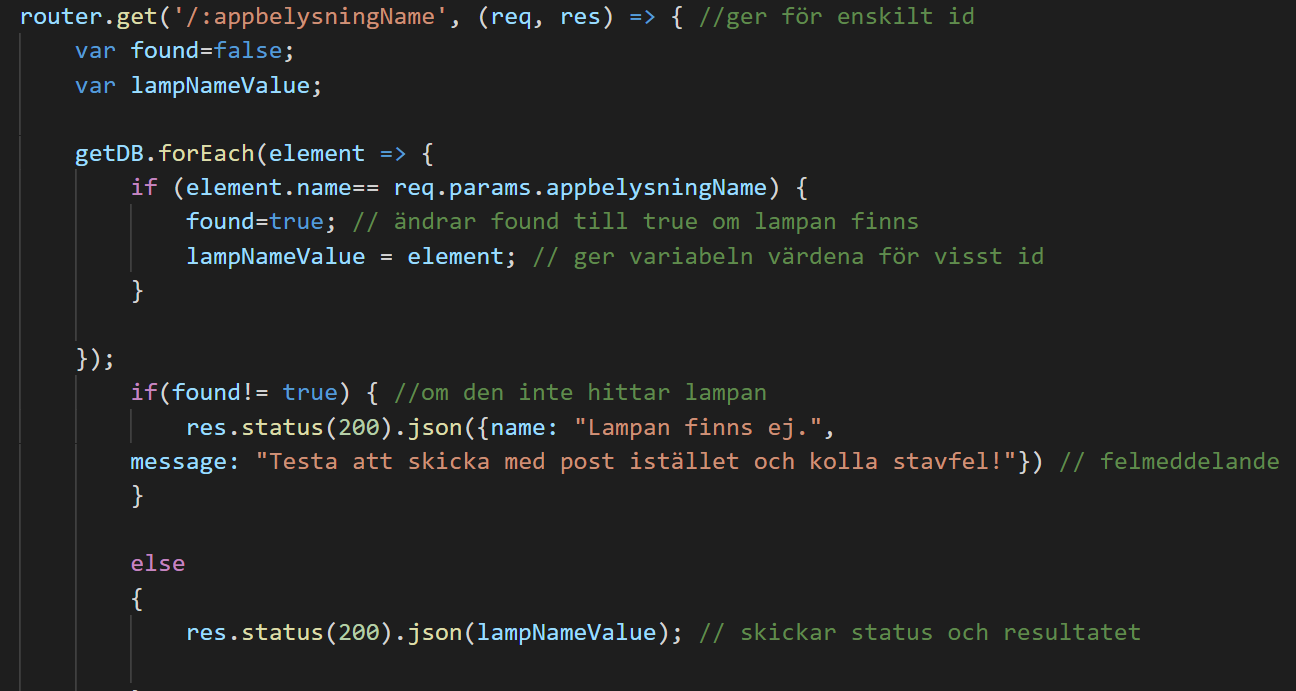
1. import React from 'react';
2. import {
3. StyleSheet,
4. View,
5. Text,
6. TouchableOpacity,
7. } from 'react-native';
8. import Slider from 'react-native-slider';
9. export default class Slidern extends React.Component {
10. DEFAULT\_VALUE = 100; // ursprungsläge för slidern

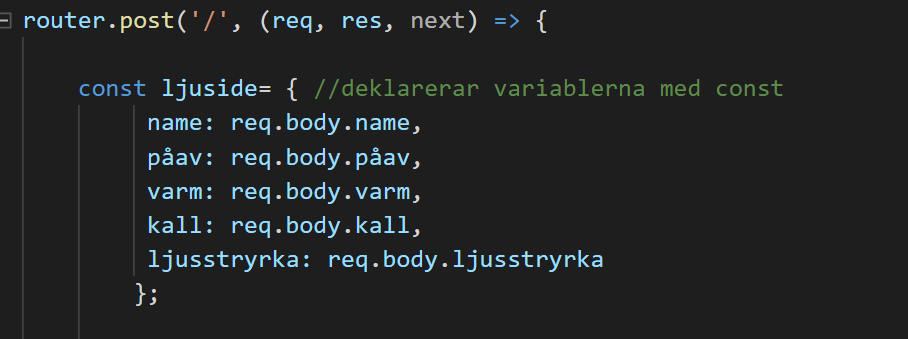
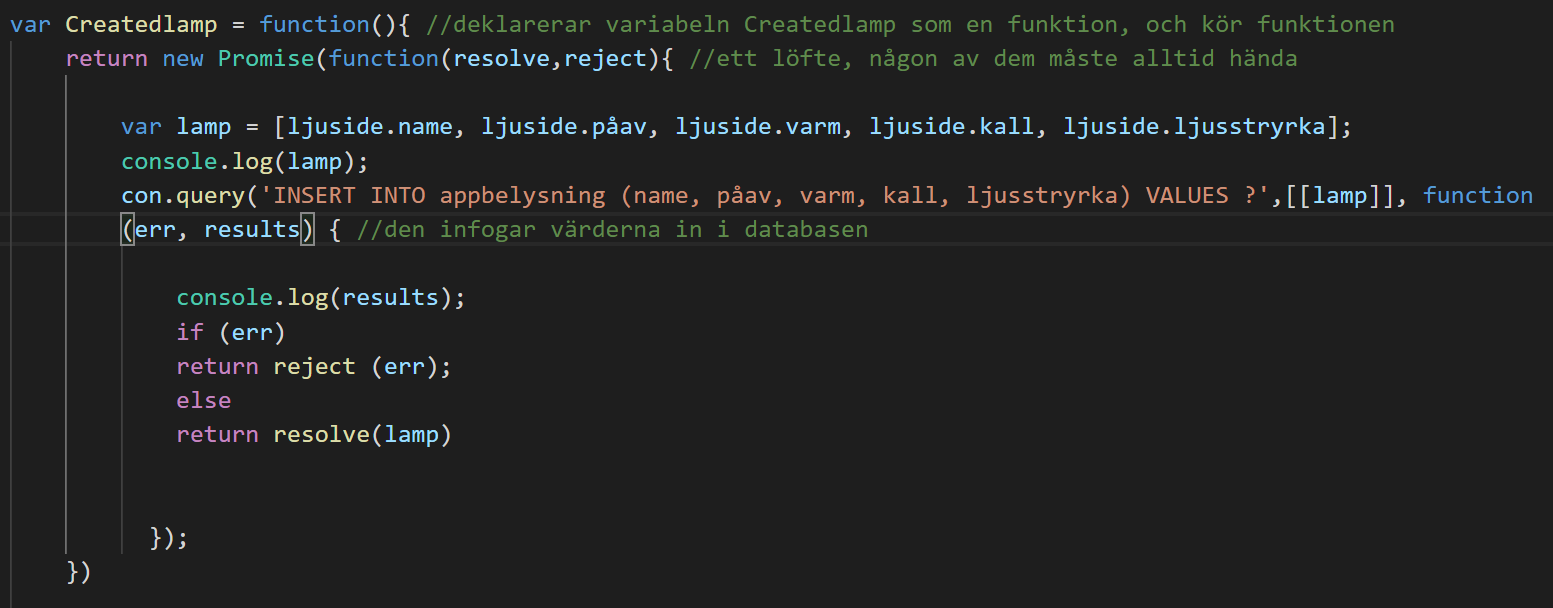
13. UpdateDataToServer = () =>{ // uppdaterar värdet för ljusstyrka


17. const { ljusstyrka } = value ; //ljusstyrka är value som står för sliderns värde
18. if(ljusstyrka!="")
19. {
20. var adress='iot.abbindustrigymnasium.se/ljuside5/lampa';
22. fetch(adress, { //uppdaterar
23. method: 'PATCH',
24. headers: {
25. 'Accept': 'application/json',
26. 'Content-Type': 'application/json',
27. },
28. body: JSON.stringify({
29. ljusstryrka: ljusstyrka,
30. })
32. }).then((response) => response.json())
33. .then((responseJson) => {

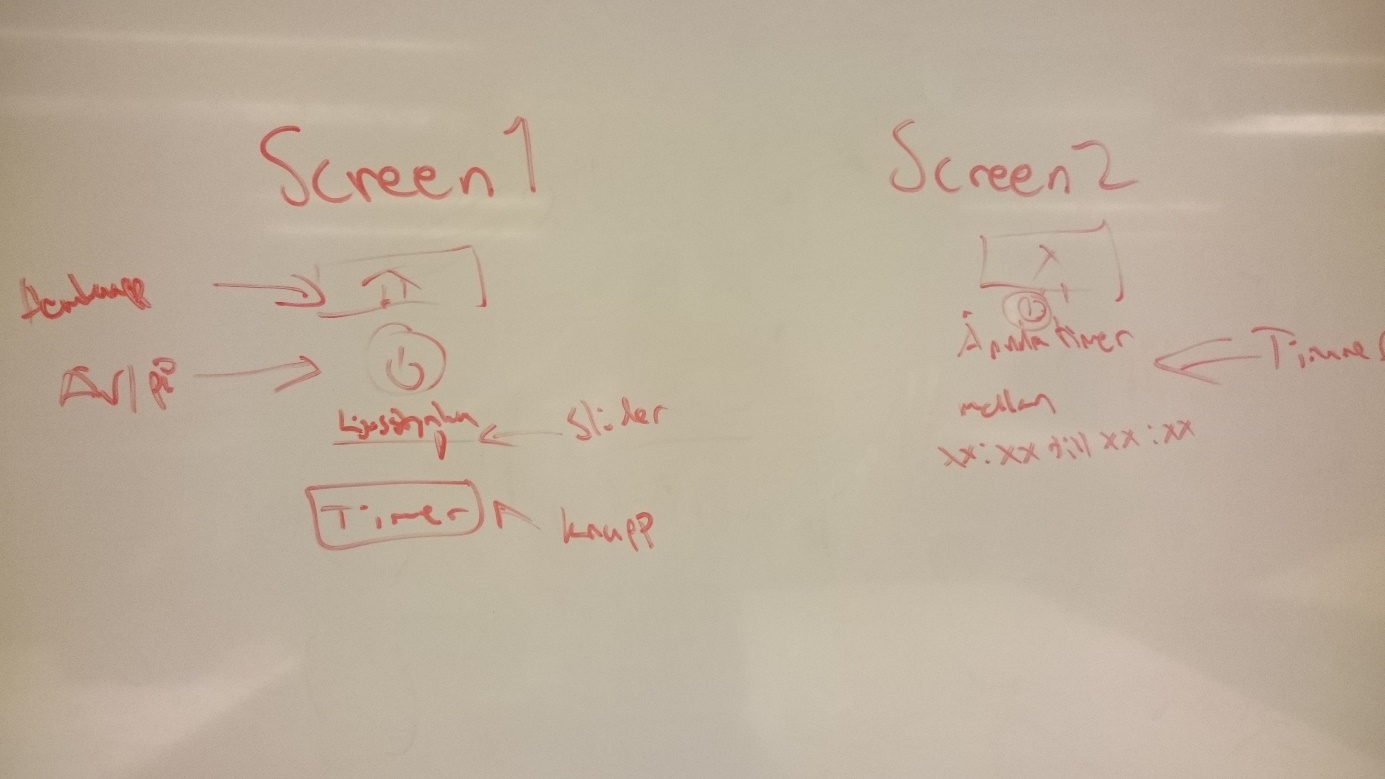

37. console.log(responseJson);
38. alert("Ny ljusstyrka ställs in");
40. }).catch((error) => {
41. console.error(error);
42. });
44. }
45. else
46. alert("Något okänt fel.")
47. }
49. SliderContainer = React.createClass({ // sätter ursprungsvärde för slidern
50. getInitialState() {
51. return {
52. value: DEFAULT\_VALUE,
53. };
55. },
57. render() {
58. if (!this.props.visible) {
59. return false;
60. }
62. return (
63. <View
64. style={styles.component}
65. >
66. <View style={styles.layouts}>
68. <View style={styles.layout1}>
70. <View style={styles.itemcontainer}>
72. <View style={styles.itemcontainerinner}>
74. <View style={styles.item1}>
76. <Text style={styles.itemtext}>
77. Ljusstyrka
78. </Text>
79. </View>
80. <View style={styles.item2}>
81. <Slider
82. minimumvalue={100}
83. maximumvalue={1000}
84. minimumTrackTintColor=''
85. maximumTrackTintColor=''
86. thumbTintColor=''
87. value={this.state.value}
88. onValueChange={(value) => this.setState({value})}
89. />
90. </View>
91. <View style={styles.item3}>
92. <TouchableOpacity style={styles.button}
93. onPress={this.UpdateDataToServer}>
94. <Text style={styles.itemtext2}>
95. Ändra ljusstyrka
96. </Text>
97. </TouchableOpacity>
99. </View>
100. </View>
101. </View>

104. </View>
106. </View>
107. </View>
109. );
110. }
112. });
113. }
114. const styles = StyleSheet.create({
116. component: {
117. width: '100%',
118. flexDirection: 'row',
119. paddingLeft: 7.5,
120. paddingRight: 7.5,
121. paddingTop: 7.5,
122. paddingBottom: 7.5,
123. },
125. layouts: {
126. flexDirection: 'row',
127. flexWrap: 'wrap',
128. },
130. });

Bilaga 1.3

Bilaga 1.4



Wireframe – Frontend

Moqups:

