stație de vreme inteligentă cu

conexiune iot

PROIECT SEMESTRIAL COMPUTER NETWORKS

Autori:  **Pietrar Florentin-Mircea**

**Stupinean Larisa-Cristina**

Conducător științific: **SL.dr.ing. Valentin SITA**

Cuprins

[1 Introducere 2](#_Toc89615868)

[1.1 Context general 2](#_Toc89615869)

[1.2 Obiective 2](#_Toc89615870)

[1.3 Specificații 2](#_Toc89615871)

[2 Studiu bibliografic 3](#_Toc89615872)

[3 Analiză, proiectare, implementare 4](#_Toc89615873)

[3.1 Arduino Mega 2560 4](#_Toc89615874)

[3.2 Shield Ethernet W5100 5](#_Toc89615875)

[3.3 Senzorul de temperatură și umiditate DH11 5](#_Toc89615876)

[3.4 Modulul cu senzor de ploaie 6](#_Toc89615877)

[3.5 Implementare hardware 7](#_Toc89615878)

[3.6 Implementare software 8](#_Toc89615879)

[3.7 Testare și validare 9](#_Toc89615880)

[4 Manual de utilizare 10](#_Toc89615881)

[5 Concluzii 11](#_Toc89615882)

[5.1 Rezultate obținute 11](#_Toc89615883)

[5.2 Direcții de dezvoltare 11](#_Toc89615884)

[6 Bibliografie 12](#_Toc89615885)

[Anexa 13](#_Toc89615886)

[Codul sursă 13](#_Toc89615887)

# Introducere

## Context general

Monitorizarea vremii este una dintre principalele preocupări ale oamenilor în zilele moderne. Informațiile legate de temperatură, umiditate și precipitații sunt necesare oricărei persoane în viața de zi cu zi.

Motivația acestei lucrări este de a oferi o soluție pentru această problemă care să fie ușor de folosit pentru a accesa informații legate de starea vremii în timp real. Am ales să dezvoltăm această aplicație pentru a aborda o problemă reală și am ales implementarea ei într-un mod cât mai eficient.

În această lucrare vom discuta aspectele abordate de-a lungul dezvoltării acestui proiect: studiul bibliografic, analiza, proiectarea și implementarea aplicației, manualul de utilizare și concluziile la care am ajuns la finalizarea proiectului.

## Obiective

Obiectivele lucrării sunt:

* afișarea temperaturii în timp real;
* afișarea umidității curente;
* afișarea unui status legat de apariția precipitațiilor;
* folosirea unui indicator luminos pentru vizualizarea rapidă a parametrilor monitorizați.

## Specificații

Specificațiile soluției propuse sunt reprezentate de o aplicație client-server și o configurație hardware corespunzătoare temei alese.

Partea de client a aplicației este implementată în React pentru a putea avea un design modern și o experiență plăcută de utilizare. Serverul este dezvoltat ca un REST API, ceea ce permite accesul la informațiile de la senzorii folosiți într-un mod securizat.

Partea hardware a fost proiectată cu ajutorul unui Arduino Mega 2560, al unui Shield Ethernet W5100, al unui senzor de temperatură și umiditate DH11, al unui modul cu senzor de ploaie și al unui modul cu 3 LED-uri.

# Studiu bibliografic

Conține o analiză a ceea ce s-a realizat/studiat anterior. Arătați că ați studiat materiale bibliografice și că ați înțeles ceea ce ați citit.

Puteți include diferite puncte de vedere asupra problemei pe care o rezolvați în lucrare.

Nu uitați să citați corespunzător autorii oricărei idei extrase dintr-o sursă bibliografică.

# Analiză, proiectare, implementare

## Arduino Mega 2560

Arduino Mega 2560 este o placă de dezvoltare bazată pe microcontrolerul Atmega2560. Dispune de 54 de pini digitali intrări-ieșiri, 16 intrări analogice, 4 UART-uri, un oscilator de 16 MHz, conexiune USB, mufă de alimentare, set de pini ICSP și un buton de resetare.

Caracteristici tehnice:

* Microcontroler ATmega2560;
* Tensiune de alimentare: 5V;
* Tensiune de intrare recomandată: 7-12V;
* Pini Digital Input / Output: 54 (din care 14 oferă ieșire PWM);
* Pini de intrare analogici: 16;
* Curent DC pe Input / Output: 40 mA;
* Curent DC pentru Pinul 3.3V: 50 mA;
* Flash Memory: 256 KB;
* Frecvența: 16MHz.

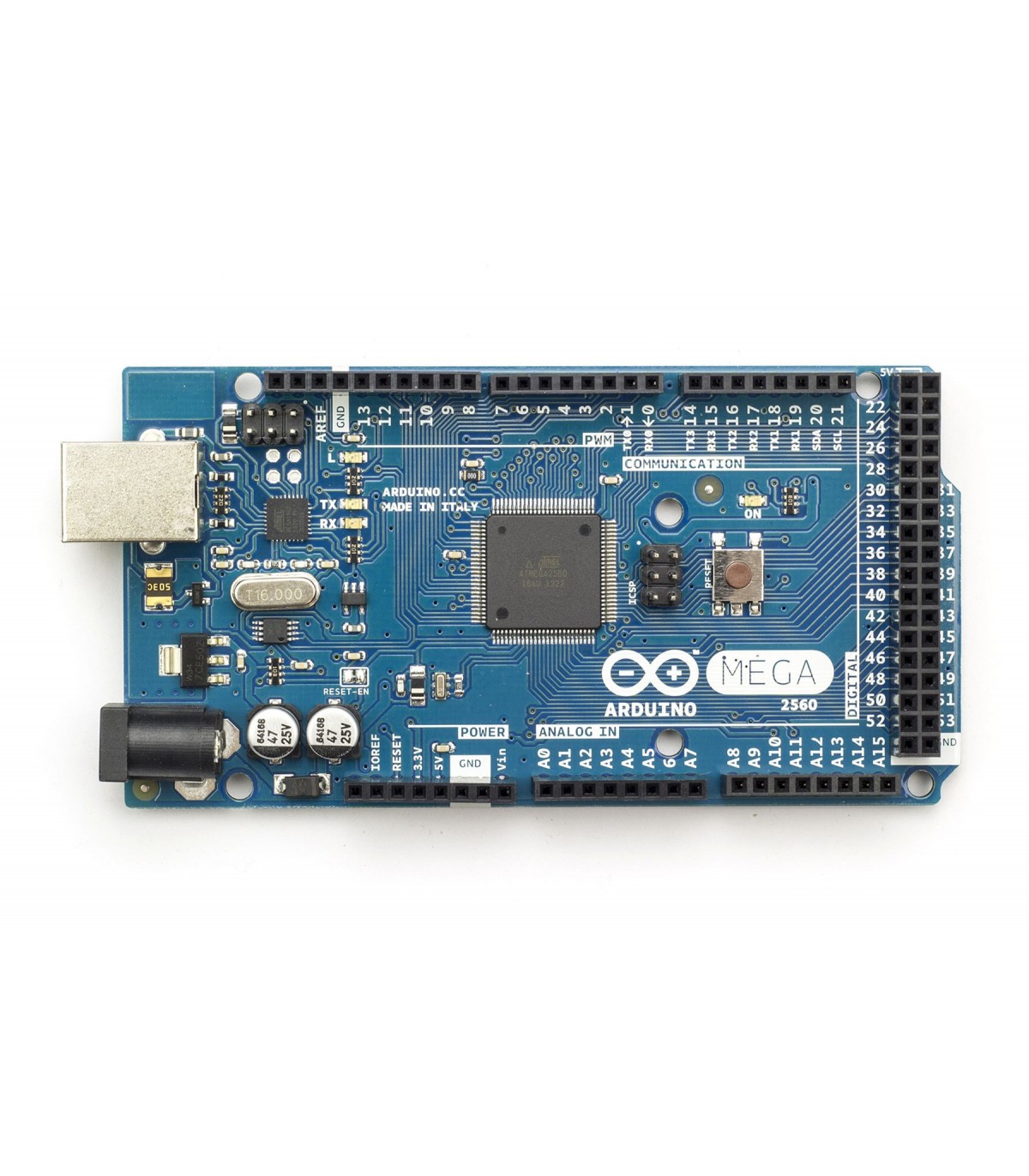


Figura 3.1.1 Placa de dezvoltare Arduino Mega 2560

## Shield Ethernet W5100

Shield-ul Ethernet cu chip Wiznet W5100 face posibilă conectarea la Internet a plăcii Arduino. Chipul oferă o adresa IP de rețea pentru protocoalele TCP si UDP, în timp ce shield-ul suportă până la 4 conexiuni simultane. Placa este compatibilă cu Arduino Mega folosind librăria Ethernet. Arduino comunică prin magistrala SPI (prin header-ul ICSP) cu chipul W5100. Pinii de pe Arduino Mega responsabili sunt 50, 51 și 52. Shieldul dispune de o mufă standard Ethernet RJ45 și suportă carduri TF de până la 16GB.

Caracteristici tehnice:

* Chip Wiznet W5100;
* Suportă protocoalele de rețea TCP si UDP;
* Suportă până la 4 conexiuni simultane;
* Compatibil Arduino Uno și Arduino Mega;
* Card slot microSD;
* Mufă standard Ethernet RJ45;
* Suportă carduri TF de până la 16GB.



Figura 3.2.1 Shield Ethernet W5100

## Senzorul de temperatură și umiditate DH11

DHT11 este un senzor de umiditate și temperatură de înaltă performanță care asigură o bună fiabilitate și stabilitate. Temperatura este măsurată de un termistor NTC, iar umiditatea relativă este măsurată folosind un senzor capacitiv. Ieșirea este oferită ca semnal digital. Senzorul DHT11 oferă precizie bună, simplitate în utilizare și are dimensiuni reduse.

Caracteristici tehnice:

* Poate măsura umiditatea în intervalul 20% - 90% cu o precizie de 5%;
* Poate măsura temperatura în intervalul 0 – 50° C cu o precizie de 2° C;
* Tensiune de alimentare: 3.3V - 5V;
* Curent: 2.5mA (maxim);
* Gama de măsurare a umidității: 20% - 95% RH;
* Acuratețea măsurării umidității: ±5% RH;
* Gama de măsurare a temperaturii: 0° C - 60° C;
* Acuratețea măsurării temperaturii: ±2°C;
* Dimensiuni: 32mm x 14mm.

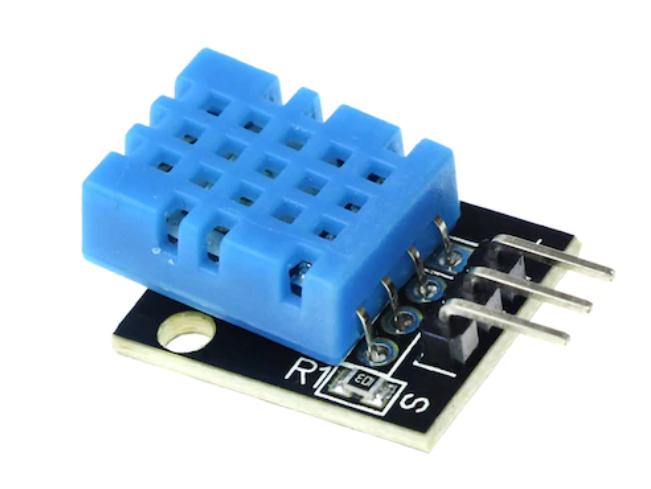


Figura 3.3.1 Senzorul de temperatură și umiditate DHT11

## Modulul cu senzor de ploaie

Modulul cu senzorul de ploaie este folosit pentru a detecta dacă plouă, pentru a da un semnal și a demara o acțiune. Acesta conține o placă de inducție și un modul de control. Senzorul se activează atunci când apa atinge circuitul de pe placa de inducție. Aceasta acționează ca un rezistor variabil care își modifică valoarea în funcție de cantitatea de apă cu care intră în contact: rezistența crește cu cât cantitatea de apă este mai mare și descrește cu cât placa de inducție este mai uscată. Modului de control are 2 ieșiri conectate la senzorul de ploaie: una analogică și una digitală.

Caracteristici tehnice:

* Dimensiune Placă: 5.5 cm x 4.0 cm
* Dimensiune Modul: 3.2cm x 1.4cm
* Anti-oxidare și Anti-Conductivitate pentru o utilizare îndelungată
* Funcționează la 5V

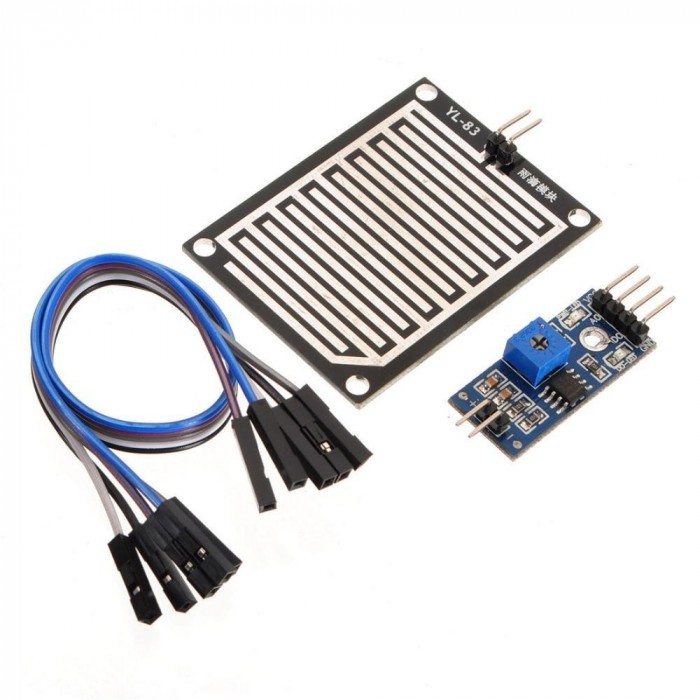


Figura 3.4.1 Modulul cu senzor de ploaie

## Implementare hardware

Arduino Mega 2560 este alimentat prin cablul USB. Shield-ul Ethernet este conectat direct la placa Arduino, iar în mufa RJ45 din shield este conectat un cablu Ethernet.

Senzorul de temperatură și umiditate DH11 este conectat la placa Arduino în modul următor:

* pinul de Ground la unul din pinii de Ground al plăcii;
* pinul de Vcc la pinul de 5V al plăcii;
* pinul de semnal la pinul digital 2 al plăcii.

Modulul cu senzor de ploaie este conectat la placa Arduino după cum urmează:

* pinul de Ground la unul din pinii de Ground al plăcii;
* pinul de Vcc la pinul digital 7 al plăcii pentru ca senzorul să nu fie alimentat constant și astfel să se uzeze prematur ;
* pinul de ieșire digitală la pinul digital 8 al plăcii.

Modulul cu 3 LED-uri este conectat la placa Arduino în felul următor:

* pinul de Ground la unul din pinii de Ground al plăcii;
* pinul pentru LED-ul roșu la pinul digital 3 al plăcii;
* pinul pentru LED-ul galben la pinul digital 4 al plăcii;
* pinul pentru LED-ul verde la pinul digital 5 al plăcii.

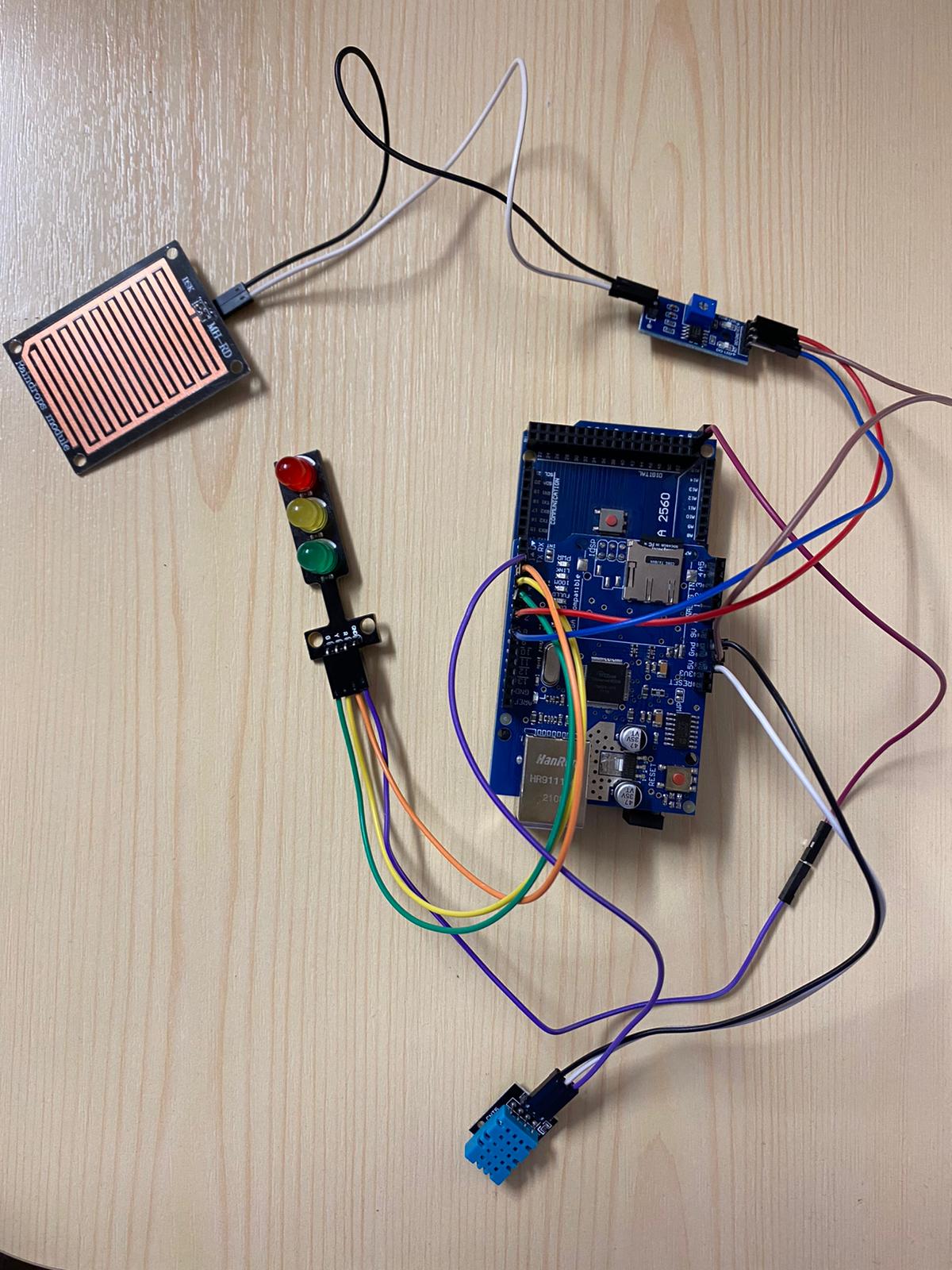


Figura 3.5.1 Configurația hardware

## Implementare software

Partea software constă într-o aplicație web, implementată folosind React pentru client și un REST API pentru server.

Clientul are o arhitectură single-page application care îi permite să fie fluid în utilizare. Acesta are mai multe funcționalități, după cum urmează:

* autentificarea utilizatorului;
* vizualizarea datelor primite de la senzori;
* modificarea intervalului la care se afișează noi date;
* actualizarea contului de utilizator.

Serverul este scris utilizând limbajul de programare C, iar librăriile folosite sunt aWOT (pentru crearea web serverului într-un mod simplu și eficient) și ArduinoJson (pentru transmiterea răspunsurilor în format Json). Web serverul expune path-urile care vor fi accesibile clientului, unde se vor putea efectua request-urile.

Pentru manipularea datelor primite de la senzorul de temperatură și umiditate am folosit librăria SimpleDHT. Citirea datelor de la acest senzor se face prin intermediul funcției *readTempHumid( )*, iar datele de la senzorul de ploaie sunt citite cu ajutorul funcției *readRainSensor( )*.

Indicatorul luminos este programat să funcționeze în felul următor:

* la detecția precipitațiilor va fi activ LED-ul roșu;
* LED-ul galben se va activa atunci când este detectată o temperatură de peste 35°C sau o umiditate de peste 70%;
* LED-ul verde va fi activ când temperatura detectată este sub 35°C sau umiditatea este sub 70%.

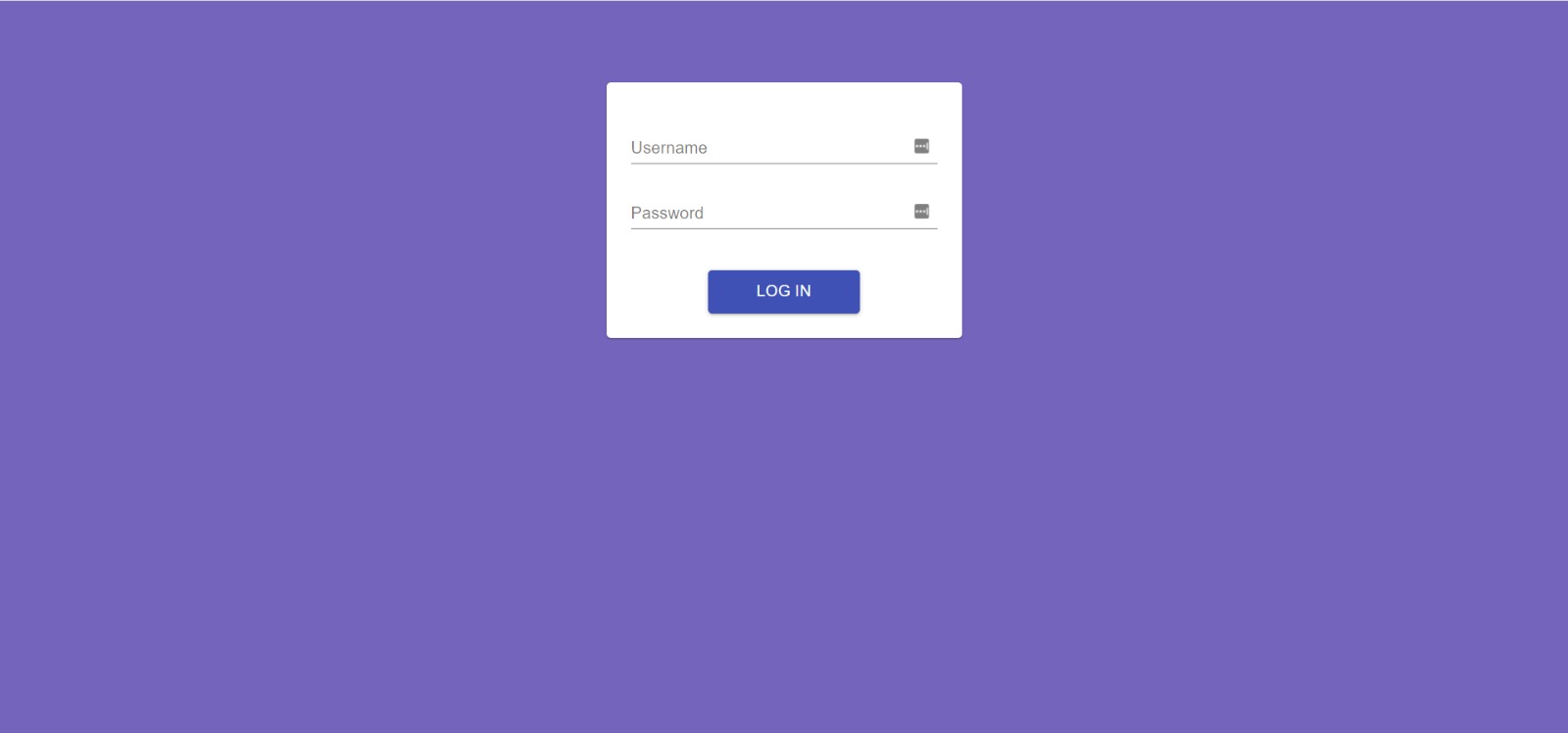


Figura 3.6.1 Pagina de autentificare

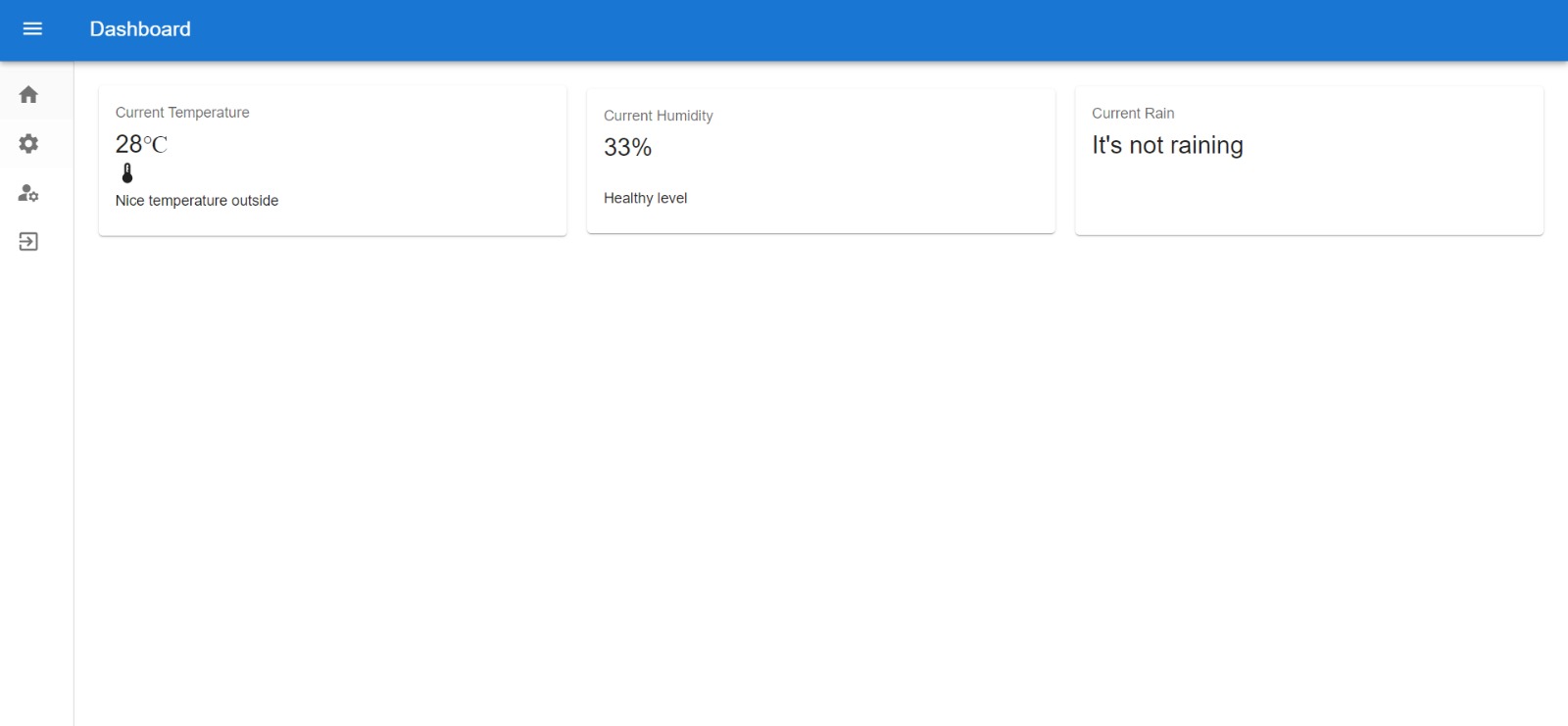


Figura 3.6.2 Vizualizarea datelor legate de starea vremii

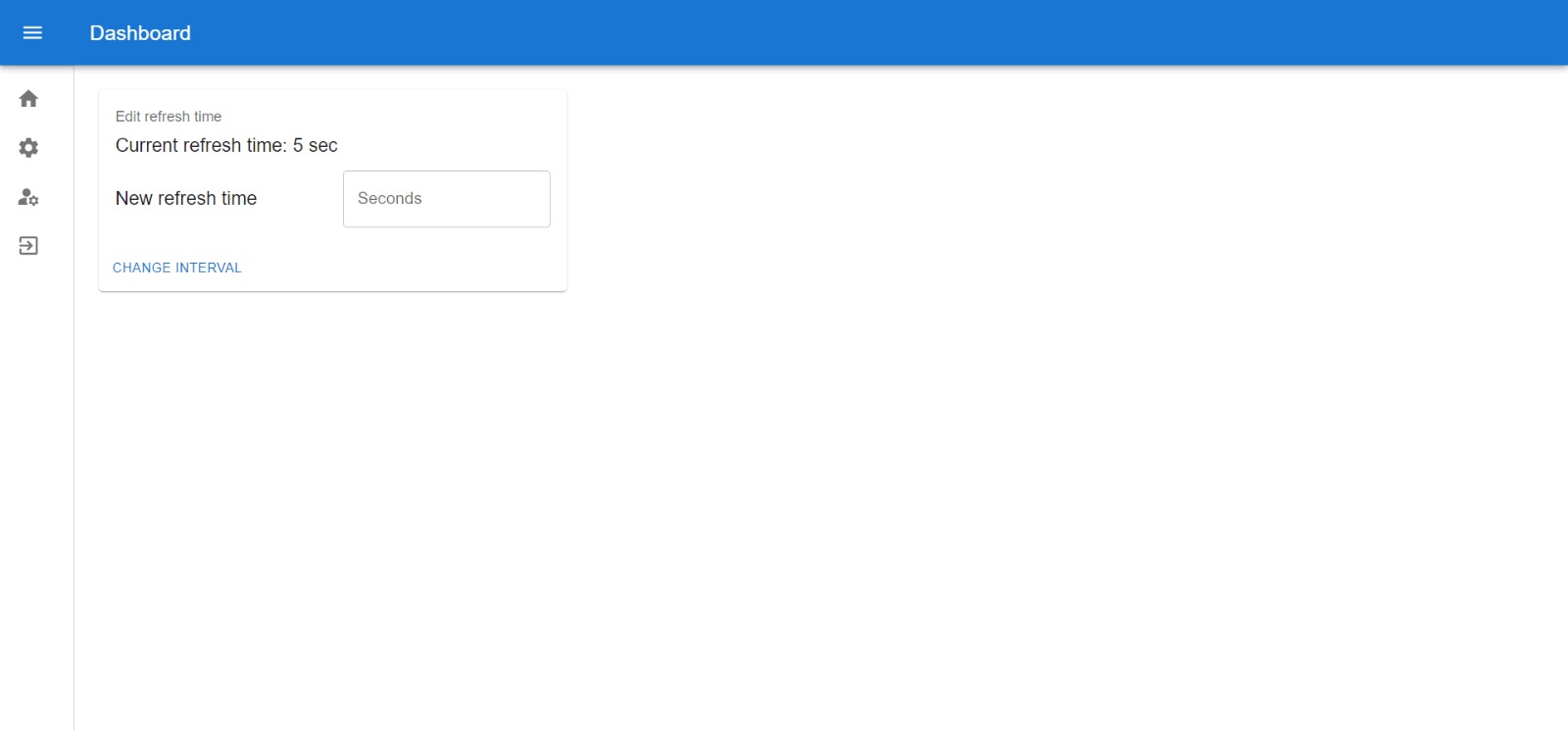


Figura 3.6.3 Modificarea intervalului la care se actualizează datele

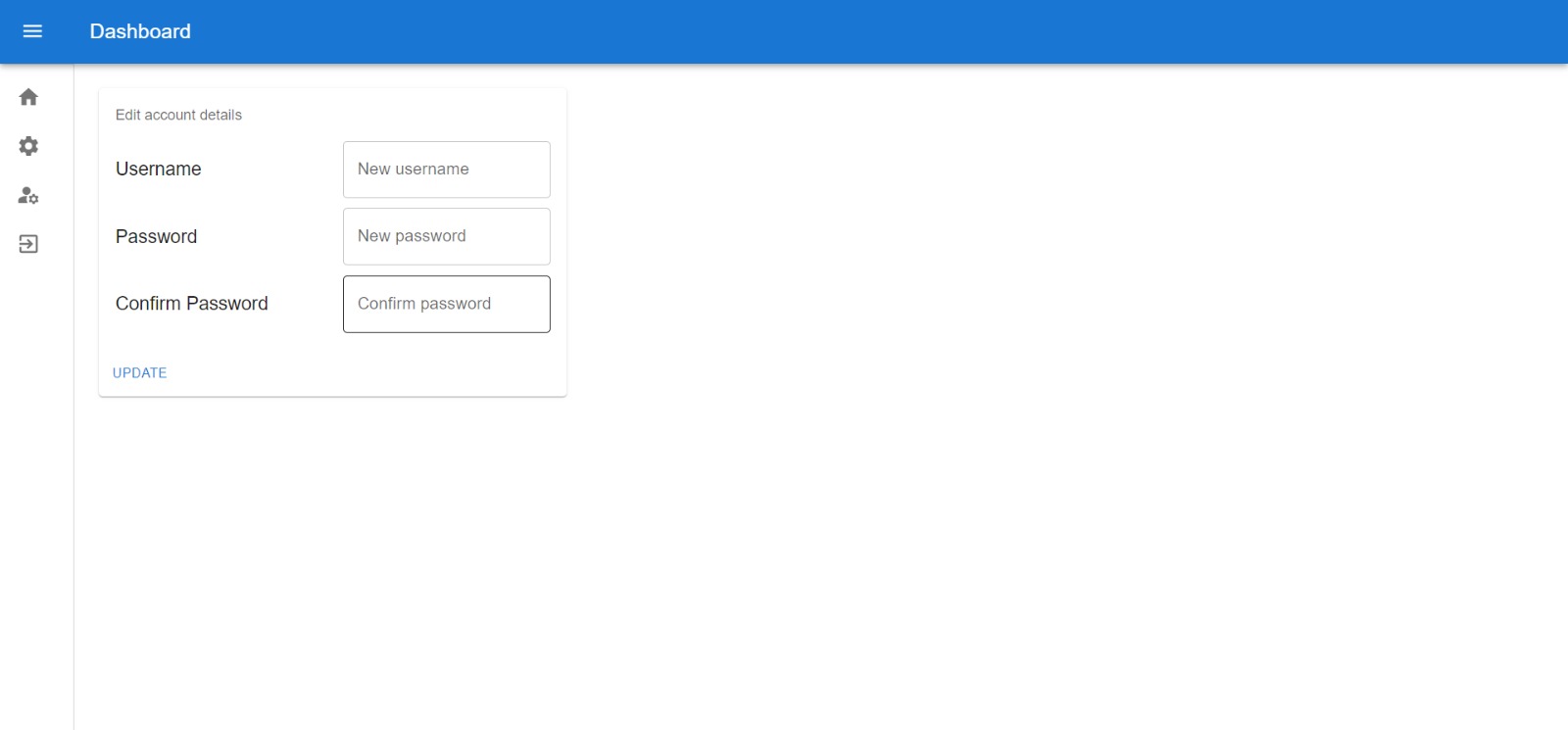
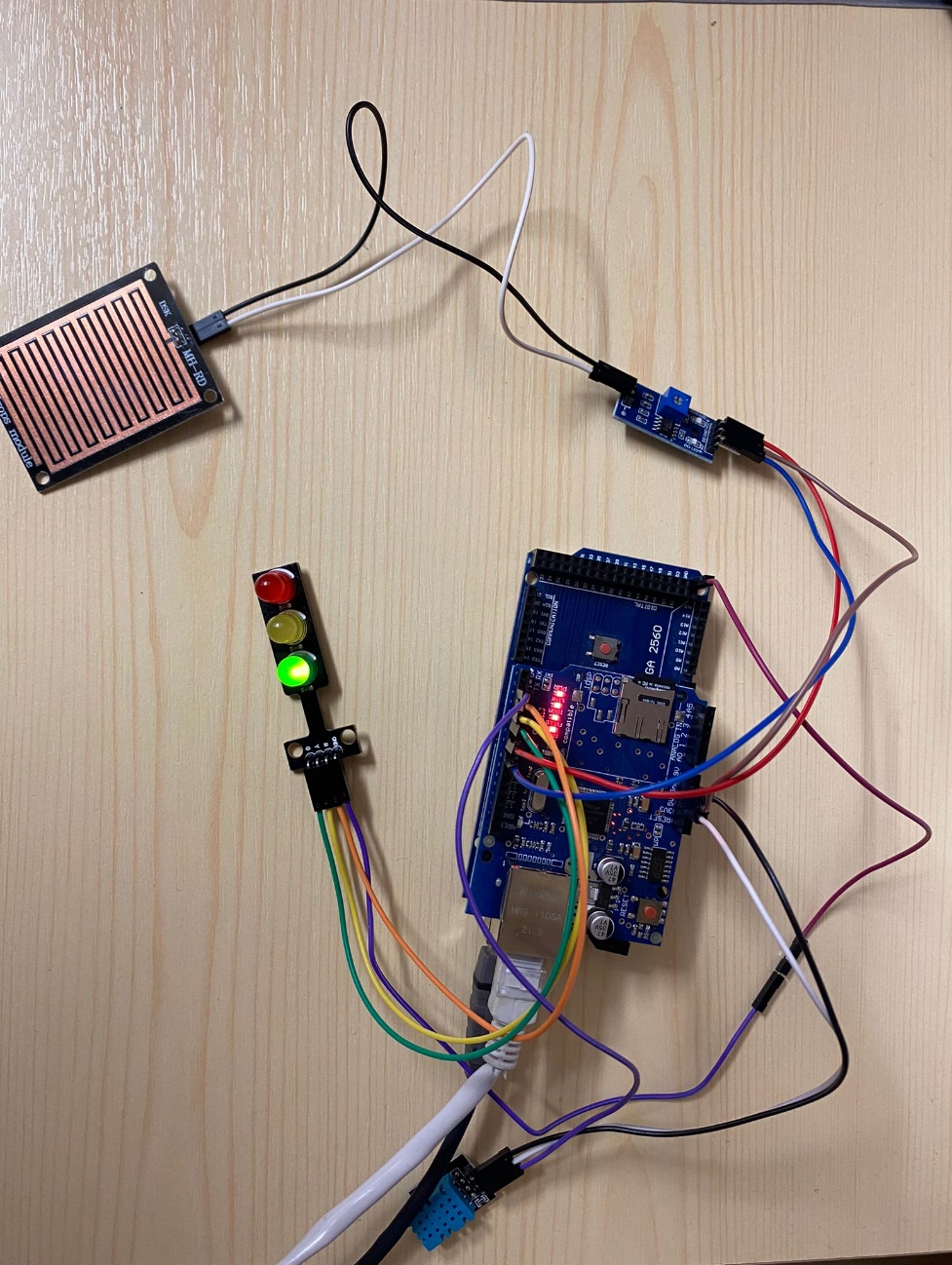


Figura 3.6.4 Actualizarea contului de utilizator

## Testare și validare

După finalizarea procesului de implementare, am testat aplicația în diferite scenarii:

* am simulat creșterea temperaturii prin încălzirea senzorului DHT11;
* am simulat apariția precipitațiilor prin stropirea cu apă a senzorului de ploaie.

În urma acestor teste am costatat ca datele afișate în client corespund cu cele reale, iar indicatorul luminos funcționează în concordanță cu valorile citite.

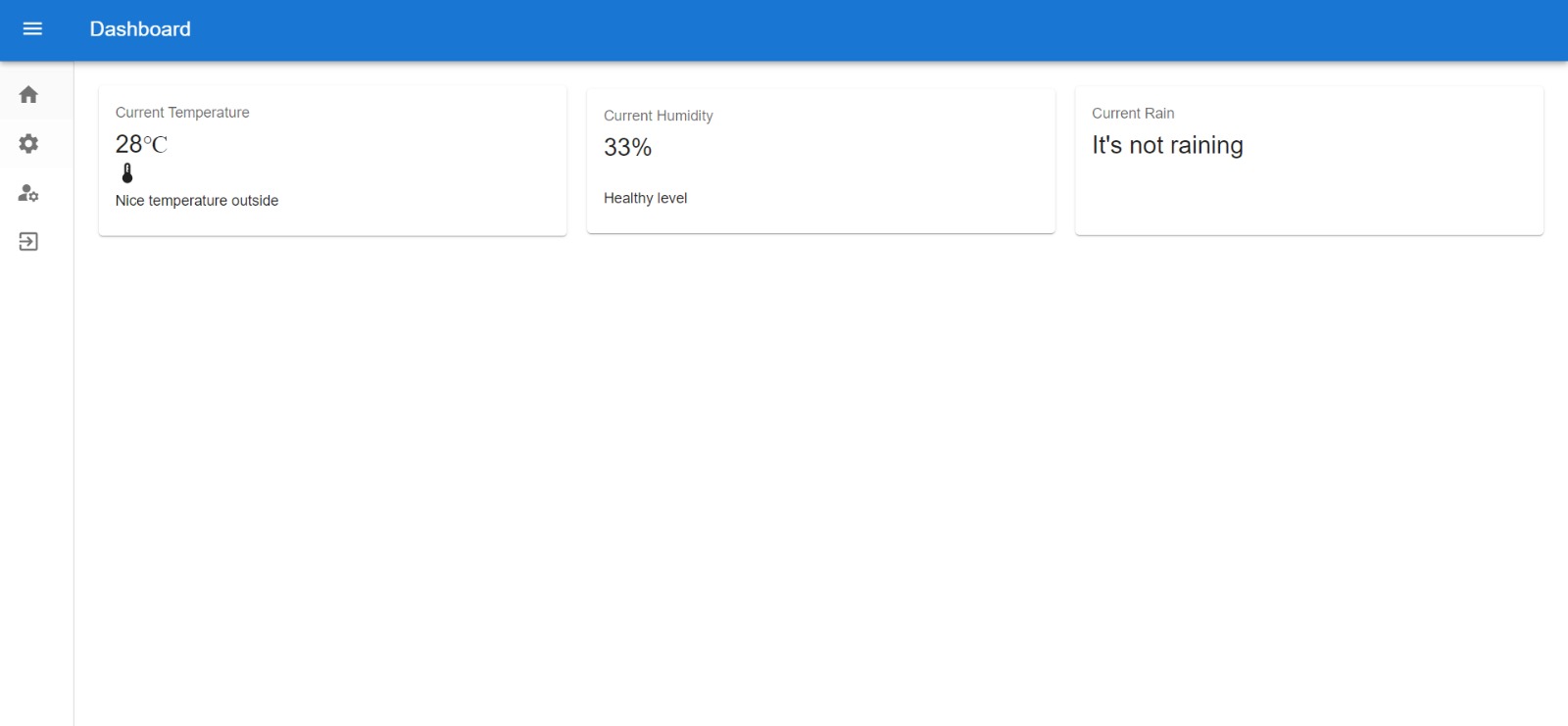
Figura 3.7.1 Aplicația în funcțiune – partea hardware

Figura 3.7.2 Aplicația în funcțiune – partea software

# Manual de utilizare

O precondiție pentru configurarea inițială a aplicației este instalarea programului Arduino IDE, care se poate face accesând acest [link](https://www.arduino.cc/en/software).

Primul pas este conectarea la sursa de alimentare a plăcii Arduino prin cablul USB. După care, se introduce cablul Ethernet în mufa RJ45 a Shield-ului Ethernet. La prima utilizare, este necesară consultarea monitorului serial al plăcii pentru aflarea IP-ului serverului. Înainte de pornirea clientului, se verifică dacă IP-ul setat pentru server corespunde cu cel citit din monitorul serial. În caz contrar, se modifică IP-ul cu cel corespunzător. Se pornește clientul utilizând comanda **npm start** accesând o linie de comandă în folder-ul client. Astfel, clientul se deschide automat în browser. În cazul în care utilizatorul închide clientul și dorește să revină la acesta fără să repornească clientul, acesta este disponibil la adresa **localhost:3000**.

După accesarea clientului, se deschide pagina de autentificare în care utilizatorul își va introduce numele de utilizator și parola pentru a accesa aplicația. După autentificare, se va deschide panoul de control al aplicației, unde sunt disponibile datele referitoare la starea vremii la momentul respectiv. Inițial, datele sunt actualizate la un interval de 5 secunde, care se poate însă modifica de utilizator apăsând butonul de ”Settings”. Datele contului de utilizator pot fi modificate de către acesta prin accesarea paginii ”Account”. Pentru delogare, utilizatorul apasă butonul de ”Logout”.

# Concluzii

## Rezultate obținute

Această lucrare propune o soluție simplă și eficientă pentru monitorizarea vremii în timp real. Obiectivele propuse au fost îndeplinite, rezultatul fiind o aplicație ușor de utilizat care oferă informații relevante legate de temperatură, umiditate și apariția precipitațiilor.

## Direcții de dezvoltare

Posibilele direcții de dezvoltare ar fi adăugarea mai multor senzori pentru colectarea mai multor tipuri de date relevante pentru starea vremii și adăugarea unui ecran LCD pentru afișarea informațiilor colectate de la senzori.

# Bibliografie

|  |  |
| --- | --- |
| [1] | P. Nume, "Titlul capitolului," în *Titlul cartii*, Oras, Editura, 2016, pp. 1-24. |
| [2] | P. Nume, "Titlul articolului," *Titlul revistei,* vol. 1, no. 2, pp. 22-30, 2016. |
| [3] | P. Nume, "Titlul articolului," în *Numele conferintei*, Oras, 2015. |
| [4] | "IEEE Citation Reference," 2009. [Online]. Available: https://www.ieee.org/documents/ieeecitationref.pdf. |
| [5] | "IEEE Editorial Style Manual," 2016. [Online]. Available: https://www.ieee.org/documents/style\_manual.pdf. |

# Anexa

## Codul sursă

* **App.tsx**

import React, { FC, useState } from 'react';

import logo from './logo.svg';

import './App.css';

import LoginPage from './components/LoginPage';

import { AuthInfo, Login } from '@maruware/material-ui-login';

import { useTheme } from '@mui/system';

import MiniDrawer from './components/MiniDrawer';

const App: FC<{}> = props => {

  const [key, setKey] = useState<string | null>(window.sessionStorage.getItem("key"));

  const signIn = async ({ username, password }: AuthInfo) => {

    console.log(username+' '+password)

    window.sessionStorage.setItem("key", password);

    setKey(password);

  }

  const theme = useTheme();

  const [open, setOpen] = React.useState(false);

  return (

    <div>

      {key === null ?

        <Login

        onSubmit={signIn}

        usernameLabel="Username"

        passwordLabel="Password"

        submitButtonLabel="Log in"

        backgroundColor="#7464bc"

        />: null}

      {key !== null ?

      <MiniDrawer setKey={setKey}/> : null

      }

    </div>

  );

}

export default App;

* **MiniDrawer.js**

import React, { useEffect, useState } from "react";

import { styled, useTheme } from "@mui/material/styles";

import Box from "@mui/material/Box";

import MuiDrawer from "@mui/material/Drawer";

import MuiAppBar from "@mui/material/AppBar";

import Toolbar from "@mui/material/Toolbar";

import List from "@mui/material/List";

import CssBaseline from "@mui/material/CssBaseline";

import Typography from "@mui/material/Typography";

import Divider from "@mui/material/Divider";

import IconButton from "@mui/material/IconButton";

import MenuIcon from "@mui/icons-material/Menu";

import ChevronLeftIcon from "@mui/icons-material/ChevronLeft";

import ChevronRightIcon from "@mui/icons-material/ChevronRight";

import ListItem from "@mui/material/ListItem";

import ListItemIcon from "@mui/material/ListItemIcon";

import ListItemText from "@mui/material/ListItemText";

import HomeIcon from "@mui/icons-material/Home";

import ManageAccountsIcon from "@mui/icons-material/ManageAccounts";

import SettingsIcon from "@mui/icons-material/Settings";

import ExitToAppIcon from "@mui/icons-material/ExitToApp";

import axios from "axios";

import Card from "@mui/material/Card";

import CardActions from "@mui/material/CardActions";

import CardContent from "@mui/material/CardContent";

import Button from "@mui/material/Button";

import { makeStyles } from "@material-ui/core";

import ThermostatIcon from "@mui/icons-material/Thermostat";

import { TextField } from "@mui/material";

const drawerWidth = 240;

const openedMixin = (theme) => ({

  width: drawerWidth,

  transition: theme.transitions.create("width", {

    easing: theme.transitions.easing.sharp,

    duration: theme.transitions.duration.enteringScreen,

  }),

  overflowX: "hidden",

});

const closedMixin = (theme) => ({

  transition: theme.transitions.create("width", {

    easing: theme.transitions.easing.sharp,

    duration: theme.transitions.duration.leavingScreen,

  }),

  overflowX: "hidden",

  width: `calc(${theme.spacing(7)} + 1px)`,

  [theme.breakpoints.up("sm")]: {

    width: `calc(${theme.spacing(9)} + 1px)`,

  },

});

const DrawerHeader = styled("div")(({ theme }) => ({

  display: "flex",

  alignItems: "center",

  justifyContent: "flex-end",

  padding: theme.spacing(0, 1),

  // necessary for content to be below app bar

  ...theme.mixins.toolbar,

}));

const AppBar = styled(MuiAppBar, {

  shouldForwardProp: (prop) => prop !== "open",

})(({ theme, open }) => ({

  zIndex: theme.zIndex.drawer + 1,

  transition: theme.transitions.create(["width", "margin"], {

    easing: theme.transitions.easing.sharp,

    duration: theme.transitions.duration.leavingScreen,

  }),

  ...(open && {

    marginLeft: drawerWidth,

    width: `calc(100% - ${drawerWidth}px)`,

    transition: theme.transitions.create(["width", "margin"], {

      easing: theme.transitions.easing.sharp,

      duration: theme.transitions.duration.enteringScreen,

    }),

  }),

}));

const Drawer = styled(MuiDrawer, {

  shouldForwardProp: (prop) => prop !== "open",

})(({ theme, open }) => ({

  width: drawerWidth,

  flexShrink: 0,

  whiteSpace: "nowrap",

  boxSizing: "border-box",

  ...(open && {

    ...openedMixin(theme),

    "& .MuiDrawer-paper": openedMixin(theme),

  }),

  ...(!open && {

    ...closedMixin(theme),

    "& .MuiDrawer-paper": closedMixin(theme),

  }),

}));

export default function MiniDrawer(props) {

  const classes = useStyles();

  const theme = useTheme();

  const [open, setOpen] = React.useState(false);

  const [humidity, setHumidity] = React.useState([]);

  const [temperature, setTemperature] = React.useState([]);

  const [rain, setRain] = React.useState([]);

  const [contentId, setContentId] = useState(0);

  const [refreshTime, setRefreshTime] = useState(5000);

  const [newRefreshTime, setNewRefreshTime] = useState();

  const handleDrawerOpen = () => {

    setOpen(true);

  };

  const handleDrawerClose = () => {

    setOpen(false);

  };

  const handleChange = (e) => {

    setNewRefreshTime(e.target.value)

  }

  const handleClickInterval = () => {

    setRefreshTime(newRefreshTime\*1000);

    setNewRefreshTime();

  }

  useEffect(() => {

    getUserData();

  }, []);

  useEffect(() => {

    const interval = setInterval(() => {

      getUserData();

    }, refreshTime);

    return () => clearInterval(interval);

  }, []);

  const getUserData = async () => {

    try {

      const { data } = await axios.get(`http://192.168.0.101/`);

      setTemperature(data.temperature);

      setHumidity(data.humidity);

      setRain(data.rain);

      // console.log(data);

    } catch (err) {

      console.log(err.message);

    }

  };

  const logout = () => {

    window.sessionStorage.removeItem("key");

    props.setKey(null);

  };

  return (

    <Box sx={{ display: "flex" }}>

      <CssBaseline />

      <AppBar position="fixed" open={open}>

        <Toolbar>

          <IconButton

            color="inherit"

            aria-label="open drawer"

            onClick={handleDrawerOpen}

            edge="start"

            sx={{

              marginRight: "36px",

              ...(open && { display: "none" }),

            }}

          >

            <MenuIcon />

          </IconButton>

          <Typography variant="h6" noWrap component="div">

            Dashboard

          </Typography>

        </Toolbar>

      </AppBar>

      <Drawer variant="permanent" open={open}>

        <DrawerHeader>

          <IconButton onClick={handleDrawerClose}>

            {theme.direction === "rtl" ? (

              <ChevronRightIcon />

            ) : (

              <ChevronLeftIcon />

            )}

          </IconButton>

        </DrawerHeader>

        <Divider />

        <List>

          <ListItem button key="Home" onClick={() => setContentId(0)}>

            <ListItemIcon>

              <HomeIcon />

            </ListItemIcon>

            <ListItemText primary="Home" />

          </ListItem>

          <ListItem button key="Settings" onClick={() => setContentId(1)}>

            <ListItemIcon>

              <SettingsIcon />

            </ListItemIcon>

            <ListItemText primary="Settings" />

          </ListItem>

          <ListItem button key="Account" onClick={() => setContentId(2)}>

            <ListItemIcon>

              <ManageAccountsIcon />

            </ListItemIcon>

            <ListItemText primary="Account" />

          </ListItem>

          <ListItem button key="Logout" onClick={logout}>

            <ListItemIcon>

              <ExitToAppIcon />

            </ListItemIcon>

            <ListItemText primary="Logout" />

          </ListItem>

        </List>

      </Drawer>

      <Box component="main" sx={{ flexGrow: 1, p: 3 }}>

        <DrawerHeader />

        {contentId === 0 ? (

          <div className={classes.columns}>

            <Card>

              <CardContent>

                <Typography

                  sx={{ fontSize: 14 }}

                  color="text.secondary"

                  gutterBottom

                >

                  Current Temperature

                </Typography>

                <Typography variant="h5" component="div">

                  {temperature.length !== 0

                    ? temperature.map((item, index) => {

                        return (

                          <Typography key={1} variant="h5" component="div">

                            {item + "℃"}

                          </Typography>

                        );

                      })

                    : ""}

                </Typography>

                <ThermostatIcon />

                {temperature[0] > 30 ?<Typography variant="body2">Pretty hot outside</Typography> : null}

                {temperature[0] > 25 && temperature[0] < 30  ?<Typography variant="body2">Nice temperature outside</Typography> : null}

                {temperature[0] < 25 && temperature[0] > 0  ?<Typography variant="body2">Cold outside</Typography> : null}

                {temperature[0] < 0  ?<Typography variant="body2">Freezing outside</Typography> : null}

              </CardContent>

            </Card>

            <Card>

              <CardContent>

                <Typography

                  sx={{ fontSize: 14 }}

                  color="text.secondary"

                  gutterBottom

                >

                  Current Humidity

                </Typography>

                {humidity.length !== 0

                    ? humidity.map((item, index) => {

                        return (

                          <Typography key={2} variant="h5" component="div">

                            {item + "%"}

                          </Typography>

                        );

                      })

                    : ""}

                <br />

                {humidity[0] < 25 ?<Typography variant="body2">Poor low humidity</Typography> : null}

                {humidity[0] >= 25 && humidity[0] < 30  ?<Typography variant="body2">Fair humidity</Typography> : null}

                {humidity[0] >= 30 && humidity[0] < 60  ?<Typography variant="body2">Healthy level</Typography> : null}

                {humidity[0] >= 60 && humidity[0] < 70  ?<Typography variant="body2">Fair humidity</Typography> : null}

                {humidity[0] >= 70 ?<Typography variant="body2">Poor high humidity</Typography> : null}

              </CardContent>

            </Card>

            <Card>

              <CardContent>

                <Typography

                  sx={{ fontSize: 14 }}

                  color="text.secondary"

                  gutterBottom

                >

                  Current Rain

                </Typography>

                <Typography variant="h5" component="div">

                {rain.length !== 0

                    ? rain.map((item, index) => {

                        return (

                          <Typography key={3} variant="h5" component="div">

                            {item}

                          </Typography>

                        );

                      })

                    : ""}

                </Typography>

                <br />

                <br />

              </CardContent>

            </Card>

          </div>

        ) : null}

        {contentId === 1 ? (

          <div>

            <div className={classes.columns}>

              <Card>

                <CardContent>

                  <Typography

                    sx={{ fontSize: 14 }}

                    color="text.secondary"

                    gutterBottom

                  >

                    Edit refresh time

                  </Typography>

                  <Typography sx={{ fontSize: 18 }} component="div">

                    Current refresh time: {refreshTime/1000} sec

                  </Typography>

                  <div className={classes.columns2}>

                    <Typography sx={{ fontSize: 18 }} component="div">

                      New refresh time

                    </Typography>

                    <TextField

                      id="outlined-basic"

                      label="Seconds"

                      variant="outlined"

                      value={newRefreshTime}

                      onChange={handleChange}

                    />

                  </div>

                </CardContent>

                <CardActions>

                  <Button size="small" onClick={handleClickInterval}>Change interval</Button>

                </CardActions>

              </Card>

              <div />

              <div />

            </div>

          </div>

        ) : null}

        {contentId === 2 ? (

          <div>

            <div className={classes.columns}>

              <Card>

                <CardContent>

                  <Typography

                    sx={{ fontSize: 14 }}

                    color="text.secondary"

                    gutterBottom

                  >

                    Edit account details

                  </Typography>

                  <div className={classes.columns2}>

                    <Typography sx={{ fontSize: 18 }} component="div">

                      Username

                    </Typography>

                    <TextField

                      id="outlined-basic"

                      label="New username"

                      variant="outlined"

                    />

                  </div>

                  <div className={classes.columns2}>

                    <Typography sx={{ fontSize: 18 }} component="div">

                      Password

                    </Typography>

                    <TextField

                      id="outlined-basic"

                      label="New password"

                      variant="outlined"

                    />

                  </div>

                  <div className={classes.columns2}>

                    <Typography sx={{ fontSize: 18 }} component="div">

                      Confirm Password

                    </Typography>

                    <TextField

                      id="outlined-basic"

                      label="Confirm password"

                      variant="outlined"

                    />

                  </div>

                </CardContent>

                <CardActions>

                  <Button size="small">Update</Button>

                </CardActions>

              </Card>

              <div />

              <div />

            </div>

          </div>

        ) : null}

      </Box>

    </Box>

  );

}

const useStyles = makeStyles((theme) => ({

  columns: {

    display: "grid",

    gridTemplateColumns: "repeat(auto-fit, minmax(240px, 1fr))",

    gridGap: "20px",

    alignItems: "center",

    justifyContent: "center",

  },

  columns2: {

    display: "grid",

    gridTemplateColumns: "0.5fr 0.5fr",

    gridGap: "20px",

    alignItems: "center",

    justifyContent: "center",

    paddingTop: "10px",

  },

}));

* **Webserver.ino**

#include <SPI.h>

#include <Ethernet.h>

#include <aWOT.h>

#include <ArduinoJson.h>

#include <SimpleDHT.h>

#define rainSensorPower 7

#define rainSensorPin 8

#define redLED 3

#define yellowLED 4

#define greenLED 5

#define tempSensor 2

SimpleDHT11 dht11(tempSensor);

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

EthernetServer server(80);

EthernetClient client;

Application app;

int \* readTempHumid() {

  static int  r[10];

  byte temperature = 0;

  byte humidity = 0;

  int err = SimpleDHTErrSuccess;

  if ((err = dht11.read(&temperature, &humidity, NULL)) != SimpleDHTErrSuccess) {

    Serial.print("Read DHT11 failed, err="); Serial.print(SimpleDHTErrCode(err));

    Serial.print(","); Serial.println(SimpleDHTErrDuration(err)); delay(1000);

    return;

  }

  Serial.print("Sample OK: ");

  Serial.print((int)temperature); Serial.print(" \*C, ");

  Serial.print((int)humidity); Serial.println(" H");

  // DHT11 sampling rate is 1HZ.

  delay(1500);

  r[0]=(int)temperature;

  r[1]=(int)humidity;

  return r;

}

int readRainSensor() {

  digitalWrite(rainSensorPower, HIGH);  // Turn the rain sensor ON

  delay(10);              // Allow power to settle

  int val = digitalRead(rainSensorPin); // Read the rain sensor output

  digitalWrite(rainSensorPower, LOW);   // Turn the rain sensor OFF

  return val;             // Return the value

}

// define a handler function

void indexCmd(Request &req, Response &res) {

  int \*p;

  p = readTempHumid();

  int val = readRainSensor();

  // Allocate a temporary JsonDocument

  // Use arduinojson.org/v6/assistant to compute the capacity.

  StaticJsonDocument<500> doc;

  JsonArray temperature = doc.createNestedArray("temperature");

  temperature.add(p[0]);

  JsonArray humidity = doc.createNestedArray("humidity");

  humidity.add(p[1]);

  JsonArray rain = doc.createNestedArray("rain");

  if (val) {

    rain.add("It's not raining");

  } else {

     rain.add("It's raining");

  }

  Serial.print(F("Sending: "));

  serializeJson(doc, Serial);

  Serial.println();

  // Write response headers

  client.println(F("HTTP/1.0 200 OK"));

  client.println("Access-Control-Allow-Origin: \*");

    client.println(

        "Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS");

    client.println("Access-Control-Allow-Headers: "

                   "Origin, X-Requested-With, Content-Type, Accept");

  client.println(F("Content-Type: application/json"));

  client.println(F("Connection: close"));

  client.print(F("Content-Length: "));

  client.println(measureJsonPretty(doc));

  client.println();

 // Write JSON document

  serializeJsonPretty(doc, client);

}

void setup() {

  Serial.begin(115200);

  pinMode(rainSensorPower, OUTPUT);

  pinMode(redLED, OUTPUT);

  pinMode(yellowLED, OUTPUT);

  pinMode(greenLED, OUTPUT);

  digitalWrite(rainSensorPower, LOW);

  if (Ethernet.begin(mac)) {

    Serial.println(Ethernet.localIP());

  } else{

    Serial.println("Ethernet failed");

  }

  // mount the handler to the default router

  app.get("/", &indexCmd);

}

void loop(){

  int \*p;

  p = readTempHumid();

  if(p[0]>35 || p[1] >70)

  {

     digitalWrite(yellowLED, HIGH);

     digitalWrite(greenLED, LOW);

  }

  else if(p[0]<35 || p[1]<70)

  {

    digitalWrite(yellowLED, LOW);

    digitalWrite(greenLED, HIGH);

  }

  int val = readRainSensor();

  if(val)

  {

    digitalWrite(redLED, LOW);

  } else

  {

    digitalWrite(redLED, HIGH);

  }

  client = server.available();

  if (client.connected()) {

    app.process(&client);

    client.stop();

  }

}