

Szczegóły projektowe - klient

Założenia

- prosty oraz intuicyjny interfejs
- dynamiczne odświeżanie elementów strony (bez konieczności ręcznego odświeżania - Ajax)
- czytelne czcionki, brak elementów rozpraszających użytkownika
- wykorzystanie prostych ikon, ułatwiających odnalezienie konkretnych elementów na stronie
- ograniczenie ilości zapytań kierowanych do serwera
- dobre ustrukturyzowanie kodu (zachowanie zasad Clean Code, zapewnienie łatwej rozszerzalności)
- łatwy dostęp do wszystkich elementów interfejsu (głębokość na maksymalnie trzy kliknięcia)
- spójne formatowanie kodu (zapewnione dzięki IDE - Visual Code)
- podział elementów składowych systemu na małe komponenty (zarządzanie mniejszymi fragmentami jest o wiele łatwiejsze)
- (możliwa obsługa wielu języków)

Podstawowe rozwiązania

Vue.js

Wybrany został framework Vue.js pozwalający tworzyć dynamiczne strony WWW przy wykorzystaniu javascript. Vue jest bardzo dobrze udokumentowany oraz zawiera szereg pomocnych bibliotek. Pozwala w bardzo prosty sposób podzielić widok na mniejsze komponenty, które następnie mogą być używane w wielu miejscach systemu. Ponadto praca nad mniejszymi fragmentami zawsze jest prostsza.

Axios

Axios pozwala w bardzo prosty sposób tworzyć zapytania do API (serwera)

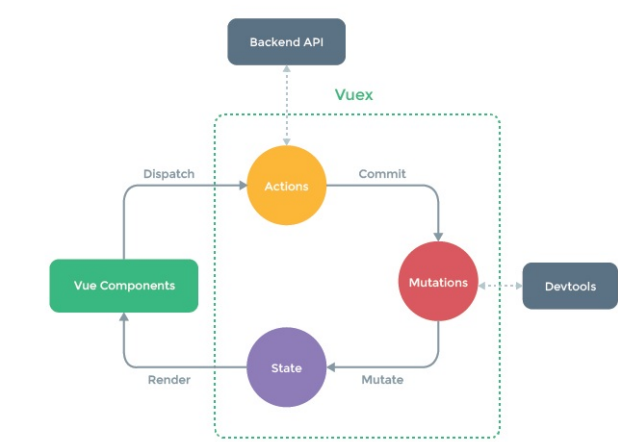
```
const axios = require('axios');

// Make a request for a user with a given ID
axios.get(`/user?ID=12345`)
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
  .then(function () {
    // always executed
  });
```

W celu umożliwienia łatwej zmiany biblioteki, stworzone zostaną specjalne service, wykorzystywane przez klienta do komunikacji z serwerem. Dzięki temu ewentualna zmiana biblioteki służącej do komunikacji z serwerem będzie łatwa - wystarczy wprowadzić nową implementację serwisu.

Vuex

Vuex jest rozszerzeniem Vue.js, funkcjonującym jako centralny magazyn gromadzący dane, z których następnie (w kontrolowany sposób) mogą korzystać wszystkie komponenty widoku.



Jest to bardzo przydatne rozszerzenie, zwłaszcza biorąc pod uwagę to, że klient pobierał będzie od serwera dane, wykorzystywane następnie w wielu miejscach. Klient będzie również zmieniał te dane. W związku z tym nie tylko serwer będzie musiał zostać powiadomiony o zmianie, ale i każdy komponent widoku. Dzięki Vuex utrzymanie spójności jest bardzo proste.

i18n

Rozszerzenie umożliwiające wprowadzenie wielu wersji językowych.

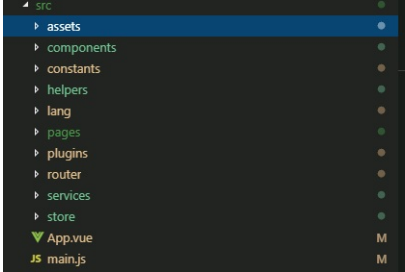
Bootstrap

Bootstrap jest frameworkiem, który pozwala na tworzenie eleganckich, responsywnych widoków, przy użyciu HTML oraz CSS. Posiada szereg zdefiniowanych klas, dzięki czemu tworzenie widoku jest szybkie.

Autoryzacja przy pomocy tokena JWT

Jako że w komunikacji klient-serwer stawiamy na podejście REST, token JWT będzie idealnie pasował do naszego systemu. Klient, w momencie logowania, otrzyma od serwera specjalny token, w którym zakodowana jest tożsamość logującego się użytkownika. Serwer, dzięki mechanizmom wewnętrznym, potrafi ocenić prawdziwość tokena oraz pobrać z niego obiekt użytkownika, który się nim posługuje. Dzięki temu zabezpieczone zostaną zasoby, do których dostęp powinien mieć tylko zalogowany użytkownik oraz zwrócone zostaną tylko i wyłącznie zasoby należące do konkretnego użytkownika.

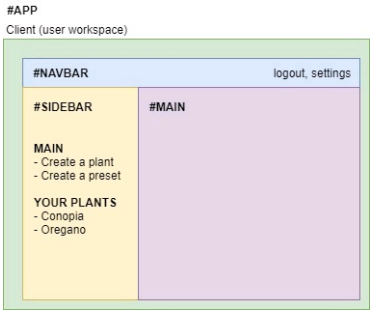
Struktura implementacji



- assets - statyczne pliki .js, .css oraz obrazki
- components - komponenty budujące stronę (np. wyszukiwarka, przełącznik wersji językowej)
- constants - pliki zawierające zmienne stałe (pliki konfiguracyjne)
- lang - pliki z wersjami językowymi
- pages - konkretne fragmenty systemu (komponenty) posiadające kluczową biznesową rolę
- plugins - pliki stanowiące interfejs do wykorzystywanych w systemie pluginów (Toast - elegancka obsługa błędów, i18n - wsparcie dla wielu wersji językowych, etc.)
- router - pliki odpowiedzialne za obsługę routingu w systemie (gdzie skierować użytkownika, gdy wejdzie pod dany adres)
- services - serwisy wykorzystywane przez system do komunikacji z serwerem oraz do innych zadań (weryfikacja adresu email, hasła, etc.)
- store - pliki definiujące magazyn Vuex

Projekt interfejsu

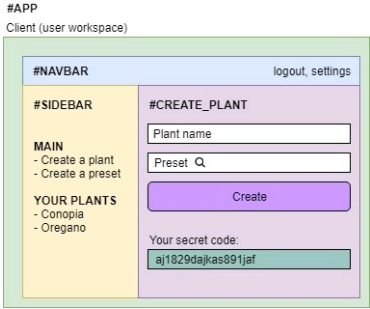
Poniżej przedstawiony został projekt interfejsu (wraz z zaznaczonymi komponentami odpowiedzialnymi za daną część).



- #APP - główny komponent, wrapper dla całej aplikacji
- #NAVBAR - górne menu, zawiera informacje o aktualnie zalogowanym użytkowniku oraz akcje związane stricte z jego kontem (wyloguj się, przejdź do ustawień); wykorzystanie dwóch ikon (mini_user.ico oraz settings.ico)
- #SIDEBAR - menu boczne, może być rozwijane (toggle on / off), dzięki czemu w wersji mobilnej użytkownik będzie miał wciąż dużą ilość miejsca na środku ekranu. Zawiera odnośniki do najważniejszych (pod względem realizacji założeń biznesowych) elementów systemu: zakładanie uprawy, tworzenie presetów, zarządzanie uprawami.
- #MAINBAR - w ten komponent wstrzykiwane będą kolejne komponenty, odpowiedzialne za realizację kluczowej funkcjonalności systemu (formularze służące do dodawania presetów / upraw, szczegóły konkretnej uprawy, panel zarządzania uprawą, etc.)

Dodawanie uprawy

Za pomocą formularza zawartego w komponencie #CREATE_PLANT użytkownik będzie mógł dodać do systemu uprawę. W momencie wysłania formularza, do serwera zostanie skierowane zapytanie POST, zawierające dane dotyczące nowej uprawy (preset oraz nazwę). Serwer spróbuje stworzyć nową uprawę i zwróci odpowiednią informację do klienta (błąd lub wygenerowany kod uprawy).



Patrząc głębiej w działanie klienta: * state - nic innego, jak faktyczne dane (tablice, stringi, inty) * mutations - za pomocą mutacji zmieniamy stany (dzięki temu dostęp do stanów jest kontrolowany). Wszystkie zmiany stanów muszą przechodzić przez mutacje * actions - są asynchroniczne, wywołują mutacje (commitują je). Wewnątrz akcji możemy np. wysłać zapytania do serwera - w naszym przypadku korzystając z odpowiednich serwisów.

Przykład:

```
login({ dispatch, commit }, { username, password }) {
  commit('loginRequest', { username });

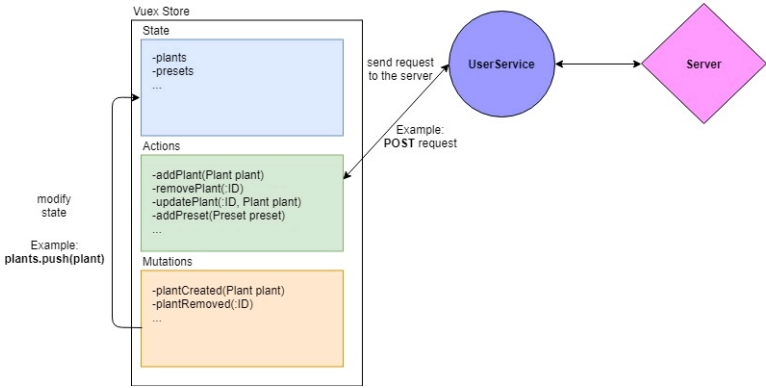
  UserService.login(username, password)
    .then(
      user => {
        commit('loginSuccess', user);
        router.push('/');
      },
      error => {
        commit('loginFailure', error);
        dispatch('alert/error', UserMessagesService.getMessageAfterLogin(error.response), { root: true });
      }
    );
}
```

Akcja login jako argument przyjmuje username oraz password. Na początku wykonuje ona commit, wywołujący mutację:

```
loginRequest(state, user) {
  state.status = { loggingIn: true };
  state.user = user;
}
```

Powyższa mutacja zmienia state ustawiając użytkownika na tego, który próbuje się zalogować oraz parametr loggingIn na true (dzięki temu wiemy, że aktualnie loguje się jakiś użytkownik).

Przepływ jest więc następujący: * AKCJA -> MUTACJA -> ZMIANA STANU



Pliki językowe

Pliki językowe, w formacie .json, przechowywane są w katalogu lang. Fragment przykładowego pliku:

```
{
  "app_title": "Plants",
  "app_description": "Here, you will be able to remotely manage your plants. Create an account and login to the system!",
  "welcome_message": "Welcome to Your Vue.js App",
  "popular_links": "Popular Links",
  "ecosystem": "Ecosystem",
  "not_found": "Page not found",
  "not_found_explanation": "We are sorry, but the page you were looking for does not exist.",
  "loading": "Loading...",
  "pages": {
    "presets": {
      "create_form_send": "Add preset",
      "create_preset": "Create a new preset",
      "presets": "Presets",
      "no_presets": "Currently there are no presets.",
      "modify_preset": "Modify the preset",
      "modify_form_send": "Modify",
      "preset_name": "Name of the preset",
      "min_temperature": "Minimal temperature",
      "max_temperature": "Maximal temperature",
      "min_humidity": "Minimal humidity",
      "max_humidity": "Maximal humidity",
      "min_soil": "Minimal soil",
      "max_soil": "Maximal soil",
      "expected_growth": "Expected weekly growth",
      "preset_color": "Color of the preset",
      "no_preset": "Preset does not exist.",
      "how_often_to_water": "How often to water?",
      "how_long_to_water": "How long to water?"
    },
    (...)
  }
}
```

Nagłówek JWT oraz odnawianie tokena

Wewnątrz serwisu authHeader.js realizowane jest dodawanie do nagłówka zarządzania tokena JWT oraz odświeżanie tokena, tuż przed jego wygaśnięciem:

```
import { UserService } from '@services/api/user/UserService'
import { LanguageRouter } from '@plugins/LanguageRouter';

import {
  MINUTES_TO_REFRESH_TOKEN
} from '@constants/startup';

export function authHeader() {
  let user = JSON.parse(localStorage.getItem('user'));

  if (user && user.token) {
    let token = user.token;

    let lastTokenRequestDate = new Date(user.last_token_request);
    let nextTokenRequest = new Date(lastTokenRequestDate.getTime() + MINUTES_TO_REFRESH_TOKEN * 60000);
    let currentTime = new Date();

    if(currentTime > nextTokenRequest) {
      UserService.refreshToken(user.token).then(newToken => {
        token = newToken;

        localStorage.removeItem('user');
        user.last_token_request = new Date();
        localStorage.setItem('user', JSON.stringify(user));
      }, error => {
        UserService.logout();
        LanguageRouter.pushToPath('/login');
      });
    }
    return { 'Authorization': 'JWT ' + token };
  } else {
    return {};
  }
}
```

Token (a nawet cały zalogowany user), pobierany jest z pamięci lokalnej. Umiejszczony zostaje tam po prawidłowym zalogowaniu się przez użytkownika, dzięki współpracy Vuex-Store oraz serwisu UserService.js.

Przykładowy serwis (PlantService.js)

```
import axios from 'axios';
import Api from '@services/api/Api'
import { PresetService } from '@services/api/preset/PresetService'
import { authHeader } from '@helpers/authHeader'

/*
  Auth-Headers: JWT + $token
*/

const API_URL = 'https://plants.ml/api'

export const PlantService = {
  create,
  getAll,
  getOne,
  add,
  remove,
  update,
  getMeasurements
};
```

```
function getMeasurements(plantId) {
  return Api().get(API_URL + '/measurement/' + plantId + '/',
    {
      headers: authHeader()
    })
  .then(response => {
    return response.data;
  });
}

function update(plant) {
  return Api().put(API_URL + '/plant/' + plant.id + '/',
    plant,
    {
      headers: authHeader()
    })
  .then(response => {
    return response.data;
  });
}

function create(plant) {
  return Api().post(API_URL + '/plant/',
    plant,
    {
      headers: authHeader()
    })
  .then(response => {
    return response.data;
  });
}

function getAll() {
  return Api().get(API_URL + '/plant/',
    {
      headers: authHeader()
    })
  .then(response => {
    return response.data;
  });
}

function getOne(presetId) {
}

function add(preset) {
}

function remove(plantId) {
  return Api().delete(API_URL + '/plant/' + plantId + '/',
    {
      headers: authHeader()
    })
  .then(response => {
    return response.data;
  });
}
```

Przykładowy Vuex-store (plantStore.js)

```
import { Trans } from '@plugins/Translation';
import router from '@router';
import { PlantMessagesService } from '@services/api/plant/PlantMessagesService';
import { PlantService } from '@services/api/plant/PlantService';

const state = { status: {}, plants: [], measurements: [], currentPlant: {} }

const getters = {
  plants: state => state.plants
}

const actions = {
  loadMeasurements({ commit, dispatch }, plantId) {
    commit('getMeasurementsRequest');

    PlantService.getMeasurements(plantId).then(
      measurements => {
        commit('getMeasurementsSuccess', measurements);
      },
      error => {
        commit('getMeasurementsError', error);
        //dispatch('alert/error', PlantMessagesService.getMessageAfterGettingPlants(error.response), { root: true });
      }
    );
  },
  updatePlant({ commit, dispatch }, newPlant) {
    return new Promise((resolve, reject) => {
      commit('updatePlantRequest');

      // API Call to update a plant
      PlantService.update(newPlant).then(
        plant => {
          commit('updatePlantLocally', newPlant);
          resolve(newPlant);
        },
        error => {
          commit('updatePlantError', error);
          dispatch('alert/error', PlantMessagesService.getMessageAfterUpdatingPlantError(error.response), { root: true });
          reject();
        }
      )
    });
  },
  getPlant({ commit, state }, { plantName }) {
    let found = false;

    for (var ia = 0; ia < state.plants.length; ia++) {
      if (state.plants[ia].name == plantName) {
        commit('setPlant', state.plants[ia]);
        found = true;
        break;
      }
    }

    if (!found)
      commit('getPlantRequestError')
  },
  getAllPlants({ dispatch, commit }) {
    commit('getAllPlantsRequest');

    PlantService.getAll().then(
      plants => {
        commit('getAllPlantsSuccess', plants);
      },
      error => {
        commit('getAllPlantsError', error);
        dispatch('alert/error', PlantMessagesService.getMessageAfterGettingPlants(error.response), { root: true });
      }
    );
  }
}
```

```

    },
    createPlant({ dispatch, commit }, newPlant) {
      commit('createPlantRequest');

      // API Call to create a plant
      PlantService.create(newPlant).then(
        plant => {
          commit('createPlantSuccess', plant);
          dispatch('alert/success', PlantMessagesService.getMessageAfterCreatingPlant(), { root: true });
        },
        error => {
          commit('createPlantError', error);
          dispatch('alert/error', PlantMessagesService.getMessageAfterCreatingPlantError(error.response), { root: true });
        }
      )
    },
    checkIfPresetInUse({ commit, state }, preset) {
      return new Promise((resolve, reject) => {
        for (let ia = 0; ia < state.plants.length; ia++) {
          if (state.plants[ia].id_preset == preset.id) {
            reject(plant)
            break;
          }
        }
        resolve();
      })
    },
    deletePlant({ dispatch, commit }, plant) {
      commit('deletePlantRequest');

      PlantService.remove(plant.id).then(
        complete => {
          commit('deletePlantSuccess', plant);
          dispatch('alert/success', PlantMessagesService.getMessageAfterRemovingPlant(), { root: true });
        },
        error => {
          commit('deletePlantError', error);
          dispatch('alert/error', PlantMessagesService.getMessageAfterRemovingPlantError(error.response), { root: true });
        }
      )
    }
  }
};

const mutations = {
  deletePlantRequest(state) {
    state.status = { removingPlant: true }
  },
  deletePlantSuccess(state, plant) {
    state.status = { removingPlant: false }

    for(let ia = 0; ia < state.plants.length; ia++) {
      let plantElement = state.plants[ia];

      if(plantElement.id == plant.id) {
        state.plants.splice(ia, 1)
        break;
      }
    }
  },
  deletePlantError(state) {
    state.status = {};
  },
  getMeasurementsRequest(state) {
    state.status = { gettingMeasurements: true }
  },
  getMeasurementsSuccess(state, measurements) {
    console.log("SUCCESS!")
    console.log(measurements)

    state.measurements = measurements;
    console.log("TO FALSE")
    state.status = { gettingMeasurements: false }
  },
  getMeasurementsError(state) {
    state.measurements = []
    state.status = {}
  },
  updatePlantRequest(state) {
    state.status = { updatingPlant: true }
  },
  updatePlantLocally(state, plant) {
    state.currentPlant.name = plant.name;
    state.currentPlant.temperature = plant.temperature;
    state.currentPlant.color = plant.color;
    state.currentPlant.id_preset = plant.preset.id;

    console.log("Current plant")
    console.log(plant.preset)
    state.currentPlant.preset = plant.preset;

    state.status = { updatingPlant: false }
  },
  updatePlantError(state) {
    state.status = {};
  },
  setPlant(state, plant) {
    state.currentPlant = plant;
  },
  getPlantRequestError(state) {
    state.currentPlant = null;
  },
  createPlantRequest(state) {
    state.status = { creatingPlant: true }
  },
  createPlantSuccess(state, plant) {
    state.status = { creatingPlant: false }
    state.plants.push(plant)
  },
  createPlantError(state) {
    state.status = {};
  },
  getAllPlantsRequest(state) {
    state.status = { gettingPlants: true };
  },
  getAllPlantsSuccess(state, plants) {
    state.status = { gettingPlants: false }

    // Setting plants that were loaded from backend
    state.plants = plants;
  },
  getAllPlantsError(state) {
    state.status = {};
    state.plants = [];
  },
};

export const plant = {
  namespaced: true,
  state,
  actions,
  mutations,
  getters

```

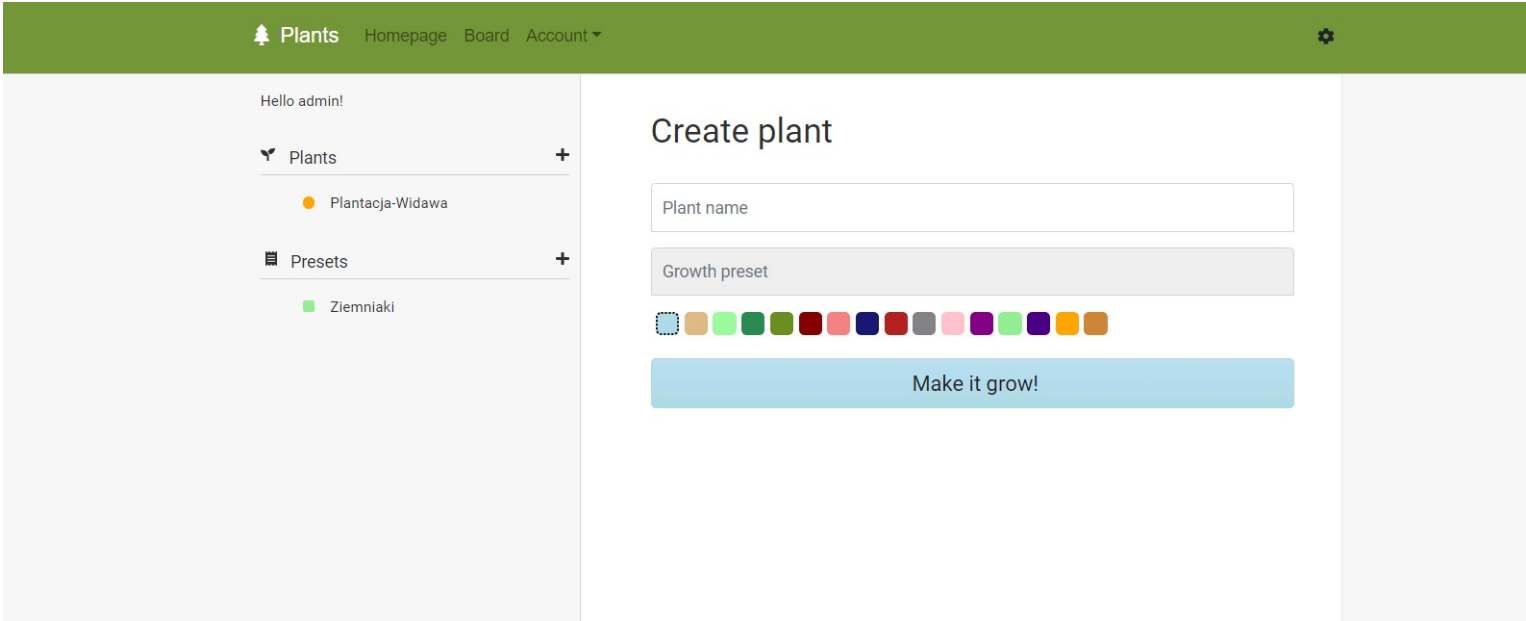
Obsługa błędów API

Ewentualne błędy zwracane przez API obsługiwane są przez Vuex-Store oraz wyświetlane przy użyciu biblioteki iziToast.js. Wiadomości (na podstawie kodów błędów) zwracane są przy pomocy serwisów wiadomości (np. PlantMessageService.js) i definiowane w osobnych plikach językowych .json.

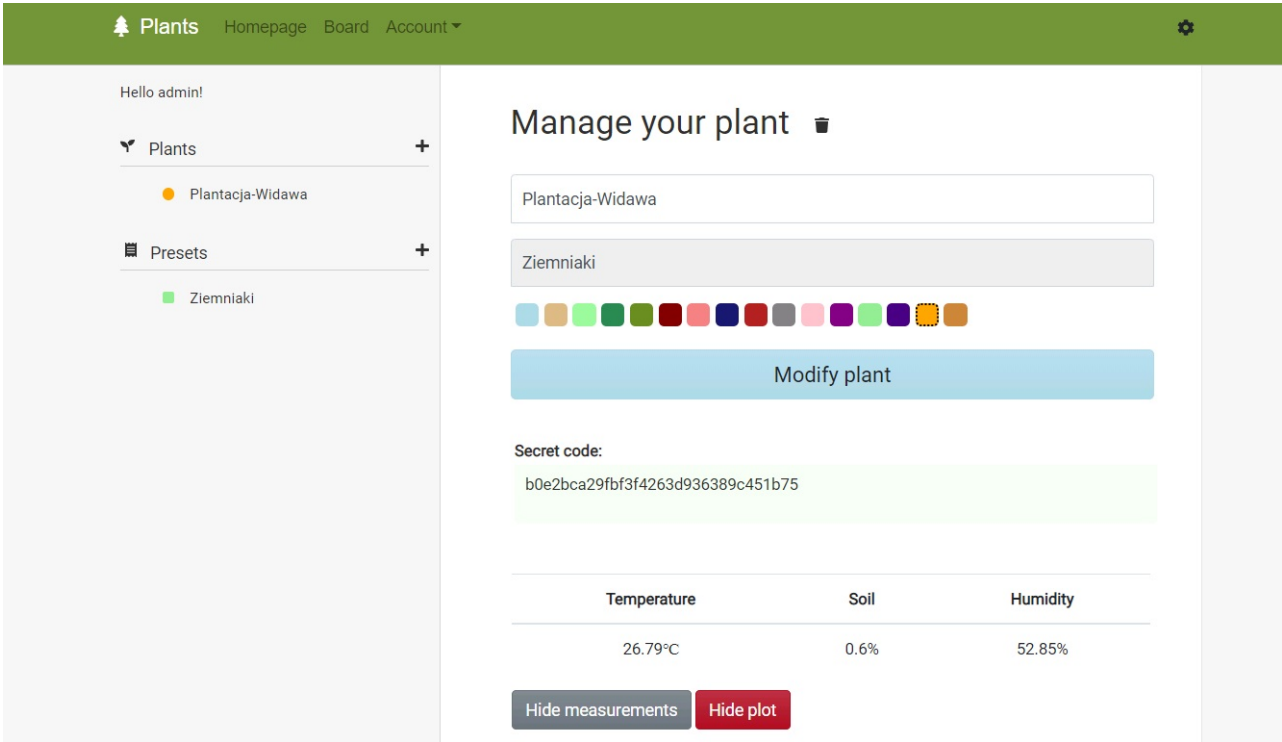
Realizacja interfejsu

Poniżej przedstawione zostały efekty implementacyjne.

Tworzenie plantacji




Zarządzanie plantacją



Analiza pomiarów

Figure 1 is a line graph showing the variation of Temperature, Humidity, and Soil moisture over time. The Y-axis represents the percentage (0 to 60). The X-axis shows dates from 2018-08-29 02:00 to 2019-06-10 02:00. Temperature (orange line) starts at 30%, drops to 26%, and then fluctuates between 26% and 27%. Humidity (blue line) starts at 20%, rises sharply to 54%, and then fluctuates between 52% and 54%. Soil (green line) starts at 5%, rises slightly to 7%, and then fluctuates between 3% and 7%.

 Plants

Homepage

Board

Account

Hello admin!

Plants

Plantacja-Widawa

Presets

Ziemniaki

Create a new preset

Name of the preset

Preset name

How often to water?

How often to water?

h

How long to water?

How long to water?

sec

Minimal temperature

Minimal temperature

°C

Maximal temperature

Maximal temperature

°C

Minimal humidity

Minimal humidity

%

Ustawienia użytkownika

Plants

Homepage

Board

Account

Hello admin!

Plants

Plantacja-Widawa

Presets

Ziemniaki

User settings

Username

admin

First name

Damian

Last name

Cywinski

Phone number

4252352

Email address

admin@slowcode.io

Modify data

Change password

Current password

Obsługa komunikatów

User settings

Username

admin

First name

Damian

Last name

Cywinski

Phone number

4252352

Email address

admin@slowcode.io

Modify data

Change password

✓ Success

User updated successfully!

×

Change password

Current password

.....

New password

.....

Repeat password

.....

Change password

⊗

Error

Wrong password data!

×

Instalacja

Instalacja klienta WWW polega na pobraniu aktualnego kodu aplikacji z repozytorium Git: <https://github.com/Alegres/ziwg-client.git>. Dostęp do repozyterium musi zostać najpierw przyznany (repozytorium nie jest publiczne, w razie pytań: damian.cywinsky@gmail.com).

Na maszynie zainstalowane musi zostać środowisko uruchomieniowe Node, w celu możliwości korzystania z NPM (zarządzanie pakietami, uruchamianie buildu). Następnie możliwe jest przygotowanie klienta do działania:

```
# install dependencies
npm install

# serve with hot reload at localhost:8080
npm run dev

# build for production with minification
npm run build

# build for production and view the bundle analyzer report
npm run build --report

# run unit tests
npm run unit

# run all tests
npm test

# sync down lokalise translation strings. You need to create acc and generate token. Not available for free plan.
npm run lokalise:down
```

W poszczególnych serwisach należy również ustawić adresy URL endpointów serwera - w przeciwnym wypadku moduł nie będzie w stanie komunikować się z serwerem.