Guida Modulo

front-end

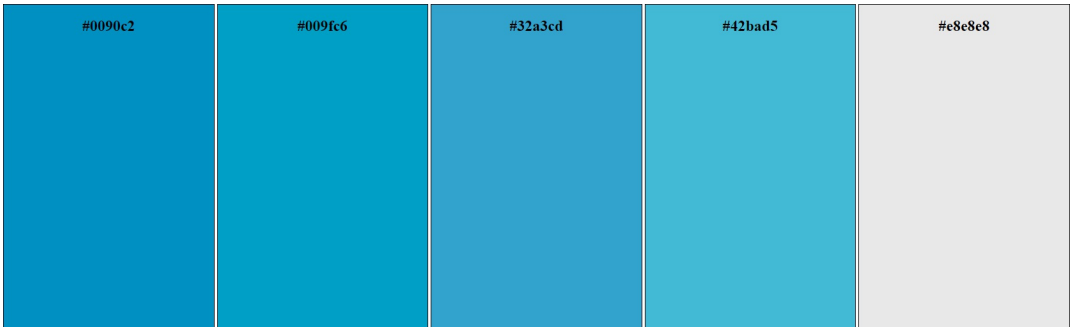
Capitolo Pagina

Indice 2
Design 3
Implementazione 5
WEB-APP 5
moduli presenti 5
AngularJS 5
Creare un controller 5
Creare un service 6
views 5
stile 7

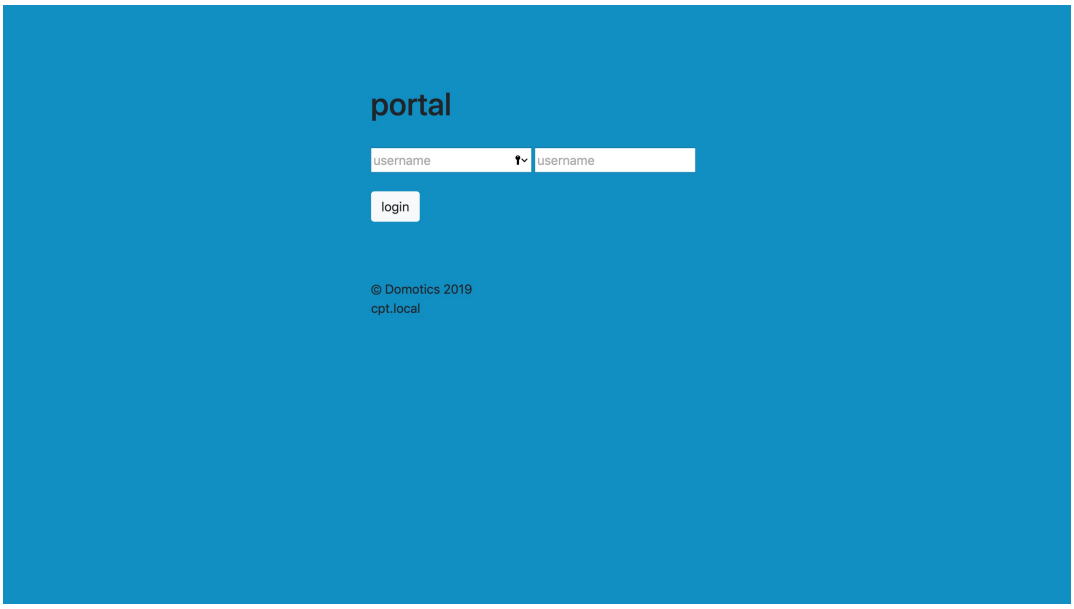
front-end

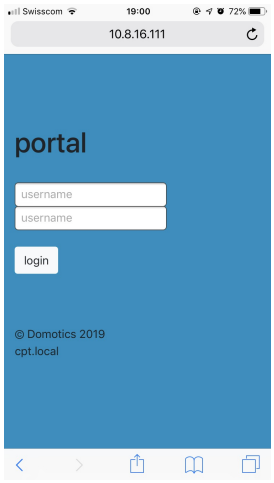
Design

Per il frontend abbiamo scelto una paletta di colori di base per tutti i siti.

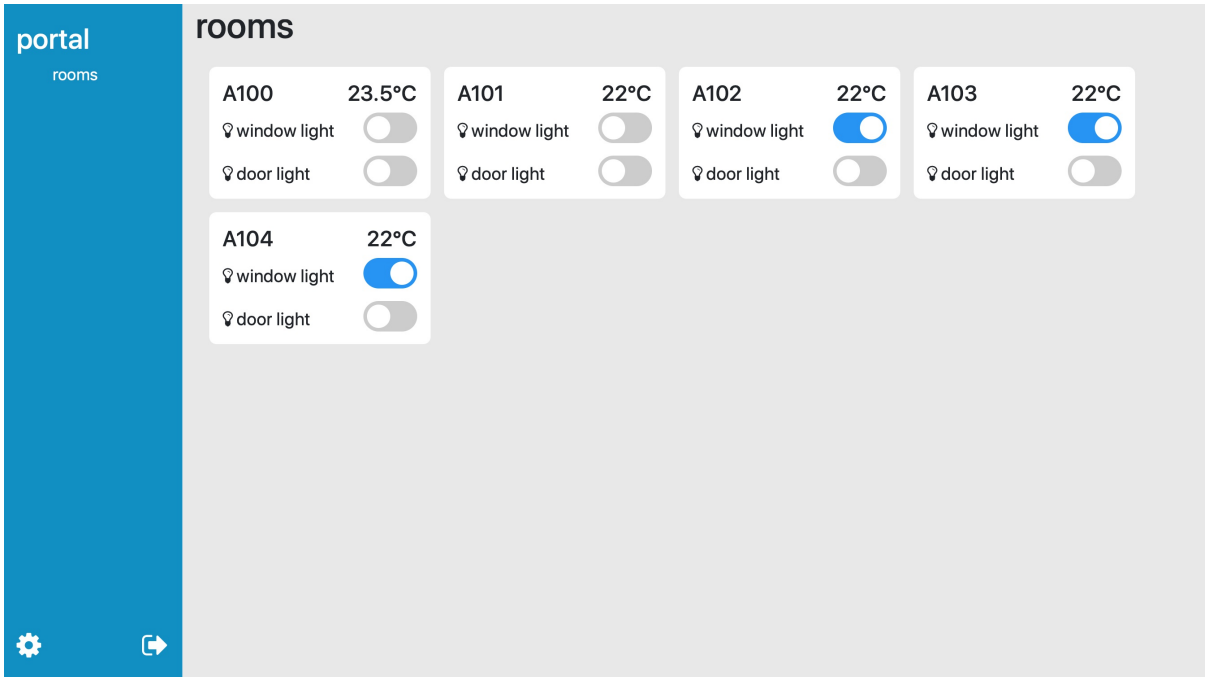


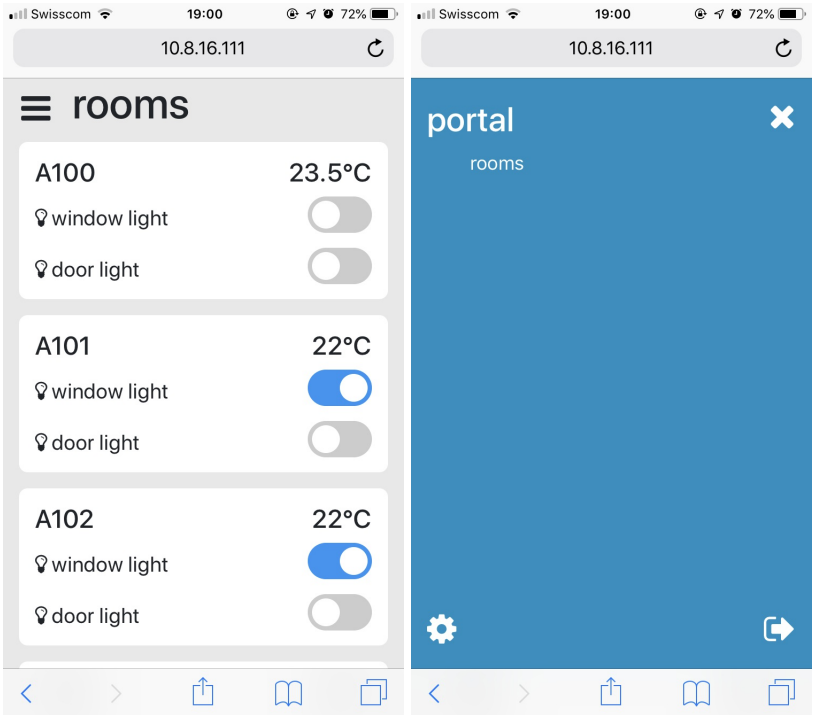
La pagina di login contiene un semplice form composto da un campo testo, un campo password e un pulsante di submit.





La pagina principale è composta da due parti, la dashboard e il corpo principale contenente le aule. La dashboard contiene le pagine del sito (al momento solo una) e la gestione del account. Invece il corpo della pagina permette di gestire le aule, vedere la tempereatura e spegnere/accendere le luci.





Implementazione

WEB-APP

Il front-end di domotics, è stato pensato come una web, app costruita su moduli, quindi per facilitare questo metodo di sviluppo è stato deciso di basarlo su AngularJS, un framework, che permette di aggiungere e togliere moduli indipendenti senza andare ad intaccare gli altri. Questo è stato pensato per poter usare questo sistema come base per un portale al quale si possono aggiungere altri moduli.

moduli presenti

Al momento sono presenti solamente due moduli, quelli di base per il l'interazione con il modulo domotics:

- Login: modulo per il login basato su LDAP.
- Rooms: Controllo delle luci di domotics.

AngularJS

La web-app basata su AngularJS, è formata dalla pagina index.html, la quale carica tutte le librerie e i framework utilizzati dal progetto, per esempio AngularJS, jQuery e bootstrap. In oltre carica l'applicazione Angular, configura le routes delle pagine ed inifine carica i controller ed i services.

Per ogni pagina bisogna creare un controller, il quale richieda i services di cui necessita ed inserisca i valori nella view.

Per inizializzare la web app:

```
var app = angular.module('ViewsAPP', ['ngRoute','ngSanitize']);
```

Questa stringa di codice inizializza l'applicazione angular, con il nome `ViewsAPP` e le configura le librerie `ngRoute`, che serve per caricare le giuste routes, richiede al server la giusta view, dato l'url. Mentre `ngSanitize`, serve per stampare del codice html scaricato tramite un service, per esempio all'interno di un file JSON.

creare un controller

I controller servono, per trasferire i dati dal servizio alla view (e nel caso in cui necessario eseguire delle operazioni su di essi). Un controller si crea come segue:

```
app.controller('Controller', ['$scope', '$sce', 'Service', function ($scope, $sce, service) {
        service.getFromService().then(function (data) {
                $scope.data = data;
        });
}]);
```

Creando un controller, bisogna inserire il suo nome, poi un array contenente gli elementi che necessita il controller, ed infine la funzione del controller, con i parametri richiesti precedentemente.

Dopo di che eseguire le operazioni che si necessitano nel controller, la variabile `$scope`, viene utilizzata per passare i valori fra i controller e le view.

creare un service

I service servono per eseguire le richieste al server, queste possono essere per esempio richieste HTTP.

```
app.factory('Service', ['$http', function($http) {
        var service = [];
        var urlBase = "/data/rooms";

        service.getFromService = function () {
                return $http({
                        method: 'GET',
                        url: url
                }).then(function (response){
                        return response.data;
                },function (error){
                        return error;
                });
        };

        return service;
}]);
```

Un service richiede il nome, ed un array, con gli oggetti di angular di cui necessita, quindi per esempio `$http`, che sarebbe la libreria, per eseguire le richieste HTTP. In ogni service solitamente si mette una sola richiesta, che può essere eseguita in modalità diverse, quindi per ogni modalità si crea una funzione per eseguire la richiesta.

Tutte le richieste vanno inserite in un oggetto, il quale verrà poi ritornato.

Views

Per gestire le pagine vengono usate delle views che vengono caricate da `app.js` nel body della pagina `index.html`, queste vengono gestite dal modulo angular `ngRoute`.

```
app.config(function ($routeProvider) {
        // index
        $routeProvider.when('/', {
                templateUrl: 'views/index.html'
        });

        // login
        $routeProvider.when('/login', {
                templateUrl: 'views/login.html'
        });

        // rooms
        $routeProvider.when('/rooms', {
                templateUrl: 'views/rooms.html'
        });

        // settings
        $routeProvider.when('/settings', {
                templateUrl: 'views/settings.html'
        });

        // else
        $routeProvider.otherwise({
                redirectTo: '/'
        });
}).run(function ($rootScope, $route) {
        $rootScope.$route = $route;
});
```

Per configuare `ngRoute`, bisogna congiurarlo con una funziona da inserire nel metodo `config()` dell'app. Al quale viene passato l'oggetto `$routeProvider`, il quale ha un metodo `when()`, questo metodo permette di settare ad uno specifico url (dopo il simbolo `#`, per esempio `localhost#!/test`), una view da caricare. Ed utilizzare anche un metodo `otherwise()`, che viene utilizzato nel caso in cui viene inserito un url non specificato prima.

Infine avviare la web app con le routes configurate precedentemente.

Stile

Per lo stile delle pagine usiamo bootstrap (https://getbootstrap.com) e fontawesome (https://fontawesome.com). Abbiamo usato Fontawesome per aggiungere delle icone al sito, abbiamo usato Fontawesome al posto delle immagini perché le icone che contiene sono dei caratteri che quindi sono più leggeri e più facili da usare delle immagini.
Ecco un esempio di view implementata usando bootstrap e fontawesome.

```
<div class="navbar">
        <i class="fa fa-bars fa-2x"></i>
</div>
<div class="sidebar">
        <div class="col-md-12">

                <h2>portal<i class="pull-right fa fa-times"></i></h2>
                <ul class="a-white">
                        <li><a href="#!/rooms">rooms</a></li>
                </ul>
        </div>
        <div class="bottom col-md-12 a-white">
    <a href="#!/settings"><i class="fa fa-cog fa-2x"></i></a>
    <a href="/Logout"><i class="fa fa-sign-out pull-right fa-2x"></i></a>
  </div>
</div>
<script src="assets/js/scripts/sidebar.js"></script>
```

Fontawesome usa il tag *per inserire le icone. Per definire l'icona bisogna inserire nel attributo class: fa che definisce l'uso di fontawesome, fa-[nome icona] che definisce quale icona si vuole usare, e se si vuole la grandezza dell'icona*

```
<i class="fa fa-cog fa-2x">
```