Università Degli Studi Di Messina

Dipartimento di Scienze Matematiche e Informatiche, Scienze Fisiche e Scienze della Terra

Progetto Tecnologie Web per il giornalismo

News Project

Di Alessandro Russo, matricola 421412
Docente: Prof. Pasquale de Meo

Anno Accademico 2016/2017

Introduzione

Lo sviluppo di questo progetto ha come scopo lo sviluppo di un'applicazione web basata su tecnologie quali: HTML5, CSS3, jQuery, AngularJS, NodeJS, volto ad acquisire esperienza e conoscenza delle tecnologie avanzate per la progettazione di piattaforme comunicative sul web.

Obiettivo e scenario di riferimento

Il progetto che ho sviluppato ha come obiettivo la gestione di un sito web di notizie, di cui sia il layout che i contenuti si adattino al tipo di dispositivo dalla quale vengono visualizzati, sia smartphone, tablet o desktop. Inoltre verrà implementato anche una sezione back-end per poter gestire le notizie, che saranno suddivise in base a determinate categorie.

Progettazione

Strumenti utilizzati

Per la realizzazione dell'applicazione web lato front-end ho utilizzato il framework **AngularJS** (AngularJS 1.5), grazie alla quale è possibile sviluppare applicazioni web in singola pagina e con architettura MVC semplicemente. Grazie all'utilizzo delle direttive two-way binding di Angular è possibile avere dei dati visualizzate nelle viste sincronizzati con il back-end ed è possibile creare viste dinamicamente. Non ho utilizzato AngularJS 2.0 perché ancora in fase Beta e quindi a mio avviso instabile per essere utilizzato.

Per la creazione delle viste utilizzate con Angular ho utilizzato il linguaggio markup **HTML5** e per il layout grafico **CSS3** tramite il framework **Bootstrap** (Bootstrap 2.3.2+) che ha velocizzato e semplificato lo sviluppo delle viste front-end responsive. Inoltre ho utilizzato la libreria JavaScript **jQuery** (jQuery 3.1.0+) per la gestione delle animazioni.

Per lo sviluppo del back-end ho utilizzato il framework **NodeJS** (NodeJS 6.5.0+). Basato sulla modalità di accesso alle risorse event-driven per sfruttare il suo comportamento asincrono in modo da essere più efficiente.

NodeJS è stato accoppiato al modulo **ExpressJS** (ExpressJS 4.0.0+) per la realizzazione di **API REST**.

Ho utilizzato il middleware **Multer** di NodeJS per gestire i form-data multipart per l'upload file e dei dati dal form per la creazione dell'articolo.

Il DBMS utilizzato è **MongoDB**, database non relazionale, orientato ai documenti JSON. Scelto per la semplicità di progettazione e d'implementazione, in base ai pochi dati del progetto da memorizzare e senza la necessità di avere relazioni tra questi.

MongoDB è installato su una istanza **EC2** (**t2.micro***, free tier,* Ubuntu Server 14.04 LTS (64bit) degli **AWS.**

Per semplicizzare la validazione dei campi ed effettuare query semplicemente seguendo la logica degli schemi per la modellazione dei dati, per lo sviluppo delle API ho utilizzato in NodeJS il modulo **Mongoose** per MongoDB.

Le dipendenze dei moduli del back-end sono gestite da **NPM** (Node package manager) tramite il file **packages.json**. Mentre le dipendenze dei moduli del front-end sono gestite tramite **Bower**.

Il software di controllo delle versioni utilizzato è **Git**, il progetto è hostato su GitHub con il nome di **NewsProject**.

Come **IDE** per lo sviluppo ho utilizzato **WebStorm.**

L'applicazione strutturata in questa maniera è scalabile e facilmente mantenibile e il codice completamente riutilizzabile e modulare.

Organizzazione applicazione

Il codice dell'applicazione è stato suddiviso in due sezioni (directory):

1. Client
2. Server

Sviluppo Back-end

L'obiettivo che mi sono posto per lo sviluppo del back-end è stato quello di rendere l'applicazione il più facilmente scalabile e modulare possibile, in quanto nella fase iniziale dello sviluppo di un back-end non si sa a priori quanto complesso possa diventare.

Ho organizzato la struttura del back-end in sezioni autonome, in base alla propria funzione.

Questo ha evitato che l'applicazione diventasse troppo complessa sia da mantenere che da sviluppare ulteriormente.

Innanzitutto ho configurato un file **.gitignore** per la sezione server in modo da ignorare alcune directory per il controllo di versione.

Le directory ignorate sono:

1. node_modules (Contiene i moduli di Node)

2. logs (Contiene i logs del server)
3. .idea (Contiene i file di configurazione di WebStorm per questo progetto)

Ho aggiunto il **package.json** per la gestione delle dipendenze di NodeJS:

```
{
  "name": "NewsProject",
  "private": false,
  "version": "0.0.1",
  "description": "NewsProject",
  "repository": "https://github.com/Alessandroinfo/NewsProject",
  "license": "MIT",
  "dependencies": {
    "express": "^4.14.0",
    "mongoose": "^4.5.10",
    "multer": "^1.2.0",
    "winston": "^2.2.0"
  }
}
```

Ogni modulo è stato installato con l'opzione di NPM **--save** in modo da aggiungersi alle dipendenze.

La configurazione iniziale del server NodeJS è contenuta all'interno del file **server.js.**

Al suo interno ho:

Incluso e configurato Express:

```
var express = require('express');
var app = express();
```

per il Routing delle API:

```
var router = require('./api/v_1');  //Richiedo tutte le rotte
app.use(router);  //Imposto tutte le rotte
```

indicando quali risorse montare su root:

```
app.use(express.static(path.join(__dirname, '../client'))); //Richiede per root tutte le
risorse statiche, punto di montaggio per le risorse statiche
```

Impostando la porta dove risponde il server:

```
app.listen(3030, function () {  //Bootstrap del server
    logger.info("Applicaton listen on port %s", this.address().port);
});
```

Incluso e configurato Mongoose per connettersi all'istanza di MongoDB e in caso di connessione avvia il resto dell'aplicazione:

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://' + config.MONGO_URI + ':' + config.MONGO_PORT + '/' +
config.MONGO_DB);   //Stabilisce i parametri di connessione a Mongo
connection.on('error', function (arg) {   //All'evento on('error') di connection
    logger.error(msg.DB.CONNECTION_FAIL);
```

```
    logger.error.bind(logger, 'connection error:');
});

connection.once('open', function () {   //All'evento once('open') di connection
    logger.info(msg.DB.CONNECTION_OK);

    //ROUTER
    var router = require('./api/v_1');  //Richiedo tutte le rotte
    app.use(router);  //Imposto tutte le rotte

    //STATICS
    app.use(express.static(path.join(__dirname, '../client'))); //Richiede per root tutte
le risorse statiche, punto di montaggio per le risorse statiche

    // LISTEN
    app.listen(3030, function () {  //Bootstrap del server
        logger.info("Applicaton listen on port %s", this.address().port);
    });

});
```

Ho anche aggiunto il modulo **Body-parser** come middleware per il parsing in JSON dei body di tutte le richieste:

```
app.use(bodyParser.json());  //Parsa tutte le richieste
app.use(bodyParser.urlencoded({extended: false}));  //Setto body parser
```

Il modulo **Path** per effettuare operazioni sui percorsi e ho inserito tutti i dati statici come i parametri di connessione a MongoDB in un file **config\index.js** che contiene anche i messaggi globali che utilizzerò nei logs.

Il router è una parte fondamentale per l'organizzazione delle API. Tramite i routers di Express è possibile definire le rotte a cui l'app risponde, che ho separate in file specifici per funzionalità.

Nella cartella **api\v_1** troviamo il file **index.js** (In NodeJS quando effettuiamo il require di una cartella senza specificare il file, automaticamente cerca all'interno il file index.js) che contiene la gestione delle rotte separate per:

1. Redirects
2. Api articles
3. Api admin

Ogni rotta a sua volta potrebbe contenere altre rotte annidate cosi sarebbe possibile costruire una struttura anche complessa ma organizzata in modo semplice.
Nel caso di questo progetto il file index.js delle rotte si presenta cosi:

```
var express = require('express');
var router = express.Router();

var apiArticles = require('./routes/api.articles');
var apiAdmin = require('./routes/api.admin');
var redirects = require('./routes/redirects');

//ROUTES
router.use('/', redirects);
router.use('/api/article', apiArticles);
```

```
router.use('/api/admin', apiAdmin);
module.exports = router;
```

Da notare che per ogni modulo di NodeJS è importante esportare il modulo alla fine.

Nella sezione **Redirects** con rotta **“/”** (api\v_1\routes\redirects.js) sono contenute le rotte che indirizzeranno alle rotte di Angular nelle sezioni di amministrazione **Admin** e nella **Home** page dov’è contenuta la pagina principale dell’applicazione:

```
var express = require('express');
var router = express.Router();


var config = require('../../../config').config();

router.use('/home', function (req, res) {
    res.redirect(config.REDIRECT_ROUTES.home);
});

router.use('/admin', function (req, res) {
    res.redirect(config.REDIRECT_ROUTES.admin);
});

module.exports = router;
```

Nel **file env.json** di config I percorsi sono I seguenti (deve prod nel caso di sviluppo o produzione):

```
{
  "dev": {
    "REDIRECT_ROUTES": {
      "home": "/#/",
      "admin": "/#/admin"
    }
  },
  "prod": {
    "REDIRECT_ROUTES": {
      "home": "/#/",
      "admin": "/#/admin"
  }
}
}
```

Nella sezione **Api articles** con rotta **“/api/article”** (api\v_1\routes\api.articles.js) sono contenute le api per la gestione degli articoli in tutta l’app:

```
var express = require('express');
var multer = require('multer');
var router = express.Router();
var ArticlesModel = require("../../../db/ArticlesModel");  //Importo il modello della
collection


// API ---------------------------------------------------------------------
```

```
//  ------------------------        ARTICOLI       ------------------------

// Configurazione del multer per il caricamento su server delle immagini
var updatedFile;

// Storage memorizza il dato che arriva come file con il nome del file originale e nel
percorso indicato
var storage = multer.diskStorage({…});
var upload = multer({storage: storage}).single('file');

//API per la creazione di un articolo
router.post('/createArticle', upload, function (req, res) {…});

//API per la visualizzazione degli aticoli
router.get('/showArticles', function (req, res) {…});

//API per la cancellazione degli articoli
router.post('/deleteArticle', upload, function (req, res) {…});

//API per la modifica dell'articolo
router.post('/editArticle', upload, function (req, res) {…});


//  ------------------------        ARTICOLI       ------------------------

module.exports = router;
```

Api per la:

1. creazione degli articoli **“/createArticle”**
2. visualizzazione degli articoli **“/showArticle”**
3. cancellazione degli articoli **“/deleteArticle”**
4. modifica degli articoli **“/editArticle”**

Il modello creato per Mongoose viene incluso con:

```
var ArticlesModel = require("../../../db/ArticlesModel");  //Importo il modello della
collection
```

Al suo interno contiene la configurazione dello schema per Mongoose con i relativi tipi per ogni campo:

```
var mongoose = require("mongoose");
var Schema = mongoose.Schema;  //Crea uno schema

var articleSchema = new Schema({
    id: Schema.Types.ObjectId,
    title: String,
    category: String,
    body: String,
    imageUrl: String,
    relevant: Boolean,
    date: Date
}, {collection: 'articles'});
```

```
articleSchema.index({title: 1}); // Indicizzazione campo per veloce ricerca ma più lenti
gli insert

var Articles = mongoose.model('articles', articleSchema);  //Esporto il modello impostato
dallo schema
module.exports = Articles;
```

All'interno del file api.articles viene configurato anche il Multer per il caricamento delle immagini in un determinato path (la directory su client **imagesUploaded**) e mantenendo il nome originale:

```
// Storage memorizza il dato che arriva come file con il nome del file originale e nel
percorso indicato
var storage = multer.diskStorage({
    // Proprieta di multer che dato il file è possibile modificare il percorso dal file
che verrà salvato
    destination: function (req, file, callback) {
        callback(null, '../client/app/imagesUploaded/');
    },
    // Proprieta di multer che dato il file è possibile modificare il nome dal file che
verrà salvato
    filename: function (req, file, callback) {
        callback(null, file.originalname);
    }
});
```

Nella sezione **Api admin** con rotta **"/api/article"** (api\v_1\routes\api.admin.js) sono contenute le api per la gestione della sezione admin:

```
var express = require('express');
var router = express.Router();
var AdminModel = require("../../../db/AdminModel");  //Importo il modello della collection

//  -------------------------        ADMIN       -------------------------
//API per il login
router.post('/tryLogin', function (req, res) {…});

module.exports = router;
```

L'unica api necessaria per l'admin è quella del login.
Viene incluso anche qui il modello per lo schema Mongoose di admin:

```
var mongoose = require("mongoose");
var Schema = mongoose.Schema;  //Crea uno schema

var adminSchema = new Schema({
    id: Schema.Types.ObjectId,
    username: String,
    password: String
}, {collection: 'admin'});

var Admin = mongoose.model('admin', adminSchema);  //Esporto il modello impostato dallo
schema
module.exports = Admin;
```

Sviluppo Front-end

L'organizzazione della sezione front-end segue la modularità implementata nel back-end.
In Angular è buona pratica suddividere i moduli in directory a se stanti in modo da organizzare l'applicazione in blocchi logici indipendenti.

Come per la parte server ho aggiunto al file .gitignore le directory da ignorare per il git, per il progetto ho aggiunto solamente la cartella **bower_component/** che contiene i componenti che utilizzo per il front-end. I componenti utilizzati sono: AngularJS, angular bootstrap, angular bootstrap, angular route, bootstrap, jquery.
Al momento dell'installazione tramite bower dei componenti ho aggiunto l'opzione **--save** per aggiungere i componenti come dipendenze.

Nel file **bower.json** sono contenuti tutte le dipendenze della parte front-end ed è cosi composto:

```
{
  "name": "NewsProject",
  "description": "NewsProject",
  "version": "0.0.0",
  "homepage": "https://github.com/Alessandroinfo/NewsProject",
  "license": "MIT",
  "private": true,
  "dependencies": {
    "angular": "~1.5.0",
    "angular-route": "~1.5.0",
    "angular-bootstrap": "^2.1.3",
    "bootstrap": "^3.3.7",
    "jquery": "^3.1.0"
  }
}
```

Dato che nel file server.js come directory dei file statici da caricare all'avvio dell'applicazione abbiamo indicato **client** viene avviato il file **index.html**, quando ci indirizzeremo alla root della nostra applicazione web.

All'interno di questo file vengono indicati i fogli di stile e i fonts da caricare:

```
<link href="https://fonts.googleapis.com/css?family=Raleway:900" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Raleway:900" rel="stylesheet">
<link rel="icon" href="app/resources/img/favicon.ico" type="image/gif"/>
<link rel="stylesheet" href="app/bower_components/html5-boilerplate/dist/css/main.css">
<link rel="stylesheet" href="app/resources/css/app.css">
```

Sono presenti due Family-font di Google, la favicon dell'applicazione, il foglio di stile di bootstrap, e il foglio contenenti stili personalizzati dell'app.

Contiene il tag **ng-view** dove verrà caricata l'applicazione tramite Angular:

```
<div ng-view></div>
```

Infine i componenti js che utilizza l'applicazione quali jQuery, AngularJS, Angular route, UI bootstrap.

```
<script src="app/bower_components/jquery/dist/jquery.min.js"></script>
<script src="app/bower_components/angular/angular.js"></script>
<script src="app/bower_components/angular-route/angular-route.js"></script>
<script src="app/bower_components/angular-bootstrap/ui-bootstrap-tpls.min.js"></script>
<script src="app/bower_components/bootstrap/dist/js/bootstrap.min.js"></script>
```

Aggiungo il file di una direttiva che utilizzo per l'avvio di una funzione al caricamento dell'immagine nel form.

```
<script src="app/directives/file_selected.directive.js"></script>
```

Di seguito include tutti i file js contenenti i moduli e la sezione principale angular:

```
<script src="app/app.js"></script>
<script src="app/home/home.module.js"></script>
<script src="app/home/home.controller.js"></script>
<script src="app/home/home.service.js"></script>
<script src="app/admin/admin.module.js"></script>
<script src="app/admin/admin.controller.js"></script>
<script src="app/admin/admin.service.js"></script>
```

Includo anche i file dei servizi globali e un file js globale che contiene un oggetto json con le categorie.

```
<script src="app/services/global.service.js"></script>
<script src="app/services/articleUpload.service.js"></script>
<script src="app/config/category.js"></script>
```

Nella parte principale dell'applicazione Angular all'interno del file **app.js** stabilisco le dipendenze dei moduli e dei servizi globali:

```
// Modulo principale di Angular
angular.module('newsApp', [   //Nome modulo principale applicazione

    //Dipendenze applicazione (Moduli)
    'ngRoute',
    'ui.bootstrap',
    'directives.module',
    'newsApp.home',
    'newsApp.admin',
    'newsApp.global.service'

]).config(config);  //Funzione di configurazione
```

Stabilisco le rotte principali dell'applicazione:

```
config.$inject = ['$routeProvider'];  //Inietto i servizi (Provider) di configurazione
all'interno della funzione config
function config($routeProvider) {  //Funzione di configurazione dell'applicazione

    //routeProvider mi permette di configurare le rotte di Angular
    $routeProvider
        .when("/admin", {
            templateUrl: 'app/admin/admin.html', //Template html per la rotta associata
            controller: 'adminCtrl',  //Nome controller
            controllerAs: 'admin'  //Utilizzato per non utilizzare lo $scope e gestire i
controller in modo mnemonico
```

```
        })
        .when("/", {
            templateUrl: 'app/home/home.html',
            controller: 'homeCtrl',
            controllerAs: 'home'
        })
        .otherwise({redirectTo: '/'});

};
```

Le rotte indirizzano alla sezione **Admin** e **Home** con i rispettivi template e sinonimi per il controller.

Seguendo le buone pratiche di Angular ho diviso il componente Admin in controller, **admin.controller.js**, il suo template, **admin.html**, il modulo **admin.module.js** e il servizio, **admin.service.js**. Nella directory login è presente il template per il login.

Il controller Admin gestisce tutte le funzionalità relative alla manipolazione grafica e dei dati degli articoli presenti nella vista admin.html.

Non appena ci si reca all'indirizzo "…/#/admin/" angular ci indirizzerò al controller di admin inizialmente con la vista del login.

Le funzioni sono:

```
// Funzione per visualizzare gli articoli su admin
function showArticles() {
    adminSvc.showArticles().success(function (data) {…});
}

// Funzione per eliminare una o più notizie
function deleteArticles() {
    adminSvc.deleteArticles({"ids": array})
        .success(function (data) {…});
}

// Funzione per il login
function tryLogin() {
    if (vm.username != undefined) {
        globalSvc.tryLogin({username: vm.username, password: vm.password})
            .success(function (data) {…});
    }
}

// Funzione modifica o inserimento articolo tramite il servizio globale articleUpload
function uploadArticle() {

    if (vm.create) {

        // Rotta per l'API
        var route = '/api/article/createArticle/';

        // Invio l'articolo per la creazione tramite servizio upload globale
```

```
        articleUpload.uploadDataToUrl(vm.importedFile, vm.articleOnEditing, route)
            .then(
                function (response) {…});


    } else {
        // Rotta per l'API
        var route = '/api/article/editArticle/';
        // Invio l'articolo per la creazione tramite servizio upload globale
        articleUpload.uploadDataToUrl(vm.importedFile, vm.articleOnEditing, route)
            .then(
                function (response) {…});
    }

};
```

La comunicazione con il server attraverso le Api viene effettuata tramite il service admin.service.js.

Nel controller Admin aggiungo la dipendenza anche del servizio globale **globalSvc** per tenermi memorizzato globalmente l'autenticazione dell'utente.

Per il caricamento, modifica o eliminazione dell'articolo utilizzo il servizio globale **articleUpload** che utilizza l'Api passate attraverso il controller "/api/article/createArticle/" :

```
angular.module('newsApp.global.service')
    .service('articleUpload', articleUpload);

articleUpload.$inject = ['$http'];

// Funzione che simula un form multipart
function articleUpload($http) {

    this.uploadDataToUrl = function (file, data, uploadUrl) {
        // fd è l'oggetto form data
        var fd = new FormData();
        // Append file
        fd.append('file', file);
        fd.append('_id', data._id);
        fd.append('title', data.title);
        fd.append('category', data.category);
        fd.append('body', data.body);
        fd.append('relevant', data.relevant);
        // Chiamo l'API fornita come parametro e i dati del form
        return $http.post(uploadUrl, fd, {
                transformRequest: angular.identity,
                headers: {'Content-Type': undefined}
            })
            .success(function (data) {
                return data;
            })
            .error(function (err) {
                return err;
```

```
            });
    }

}
```

Il template html diviso nei seguenti blocchi:

```
<!-- LOGIN -->
<div ng-include="'app/admin/login/login.html'"></div>
<!-- LOGIN -->

<!-- ADMIN -->
<div ng-if="admin.globalData.isAuthenticated">
    <!-- MENU HEADER -->
    <div ng-include="'app/menu/back.end.menu.html'"></div>
    <!-- MENU HEADER -->

    <!-- MANAGE -->
    <div class="col-lg-12 col-md-12 col-sm-12">

        <div class="row">

            <div class="sub-header"></div>
        </div>

    </div>
    <!-- MANAGE -->

    <!-- ARTICLES -->
    <div ng-if="!admin.editing" class="container-articles">
        <div class="col-lg-12 col-md-12 col-sm-12">
            <div class="row">

                <!-- Repeat per tutti gli articoli presenti filtrati per categoria, la
stringa dentro name confronta le proprietà di article, ordinando per la proprieta date di
articles (date crescente, -date descrescente)-->

            </div>

        </div>
    </div>
    <!-- ARTICLES -->

    <!-- EDIT CREATE ARTICLE -->
    <div ng-if="admin.editing" class="container-articles">
    </div>
    <!-- EDIT ARTICLE -->


</div>
<!-- ADMIN -->

<!-- MESSAGES -->
<div class="messages">
    <div ng-click="admin.fadeOut()" ng-class="{'message-fade': admin.openDeleteMsg}"
class="message-box">
        Cancellazione effettuata!
    </div>
```

```
    <div ng-click="admin.fadeOut()" ng-class="{'message-fade': admin.openErrorAccessMsg}"
class="message-box">
        Dati errati!
    </div>

    <div ng-click="admin.fadeOut()" ng-class="{'message-fade': admin.openEditMsg}"
class="message-box">
        Modifica effettuata!
    </div>

    <div ng-click="admin.fadeOut()" ng-class="{'message-fade': admin.openCreatedtMsg}"
class="message-box">
        Articolo creato!
    </div>
</div>
<!-- MESSAGES -->
```

I tag messages contengono i messaggi di avvenuta cancellazione, salvataggio o modifica dell'articolo.

Nella sezione admin è cosi possibile ordinare tutti gli articoli per data di creazione e filtrarli per ogni categoria. Successivamente è possibile aggiungere nuovi articoli inserendo **Titolo, immagine di copertina, testo dell'articolo**, e impostare un determinato articolo come rilevante, è possibile effettuare la selezione multipla per l'eliminazione o singolarmente per la modifica.

Seguendo la medesima gestione del componente Admin anche per Home ho suddiviso: controller, service, template e modulo:

-- home
----home.controller.js
----home.module.js
----home.service.js
----home.html

All'interno del controller di home ho creato le seguenti funzioni per il cambio di categoria, l'apertura di un singolo articolo, e la visualizzazione di tutti gli articoli:

```
// Funzione per visualizzare gli articoli su home
function showArticles() {
    homeSvc.showArticles().success(function (data) {…});
};

// Funzione per il cambio di categoria
function changeCategory(category) {…}

// Funzione per aprire l'articolo selezionato
function goToArticle(article) {…}

// Funzione per tornare indietro
function backToArticles() {…}
```

che richiamano la funzione per visualizzare gli articoli del proprio service:

```
angular.module('newsApp.home')
    .service('homeSvc', homeSvc);

homeSvc.$inject = ['$http'];
function homeSvc($http) {
    var self = this;
    self.showArticles = showArticles;

    // Funzione per visualizzare gli articoli
    function showArticles() {
        return $http.get('/api/article/showArticles');  //Post torna una promessa
(promise) che è asincrona, le funzioni della promise success e error (o thenin cascata)
vengono scaturiti al ritorno della promise
    };

};
```

Non appena ci recheremo all'indirizzo "…/#/" Angular ci indirizzerà alla pagina home.

Per la sezione home.html cosi come per admin il menu è stato aggiunto staticamente tramite la direttiva **ng-include**:

```
<div ng-include="'app/menu/front.end.menu.html'"></div>
```

Il resto è stato suddiviso nella sezione home che contiene l'home page e nella sezione article che contiene il singolo articolo selezionato.
Le viste si attivano alternativamente controllate dall'ng-if sui propri tag div:

Visualizza sezione home: **ng-if="!home.isOneArticle"**
Visualizza sezione article: **ng-if="home.isOneArticle"**

La sezione home è suddivisa orizzontalmente da un **carosello** che presenta le ultime tre notizie senza filtro per categoria.

Centralmente da tre colonne, a sinistra di 2/12, al centro di 8/12, e a destra di 2/12.

In basso è presente il footer.

Cosi è strutturata il layout per home:

```
<div ng-include="'app/menu/front.end.menu.html'"></div>

<!-- Contenitore front-end-->
<div ng-if="!home.isOneArticle" class="home">

    <!-- Carosello -->
    <div id="myCarousel" class="carousel slide" data-ride="carousel">
        …
    </div>
    <!-- Carosello -->


    <!-- Contenitore centrale notizie -->
    <div class="container-fluid bg-3">
```

```
        <!-- Colonna sinistra -->
        <div class="text-center col-lg-2">
            <h3>Ultime notizie</h3><br>
                …
        </div>

        <!-- Colonna centrale -->
        <div class="center col-lg-8">
            <h3>Tutte le news</h3>
                …
        </div>

        <!-- Colonna destra -->
        <div class="text-center col-lg-2">
            <h3>Notizie rilevanti</h3><br>
                …
    </div>
    <!-- Contenitore centrale notizie -->

</div>

<!-- Singolo articolo -->
<div ng-if="home.isOneArticle" class="article">

    <div class="single-article container-fluid bg-3">


        <!-- Pulsante indietro -->
        <div class="col-lg-1">…</div>

        <!-- Articolo selezionato -->
        <div class="col-lg-10">…</div>


        <div class="col-lg-1">…</div>


    </div>

</div>


<!-- Footer -->
<footer class="container-fluid text-center">
    <div>Copyright News Project 2016</div>
</footer>
<!-- Footer -->
```

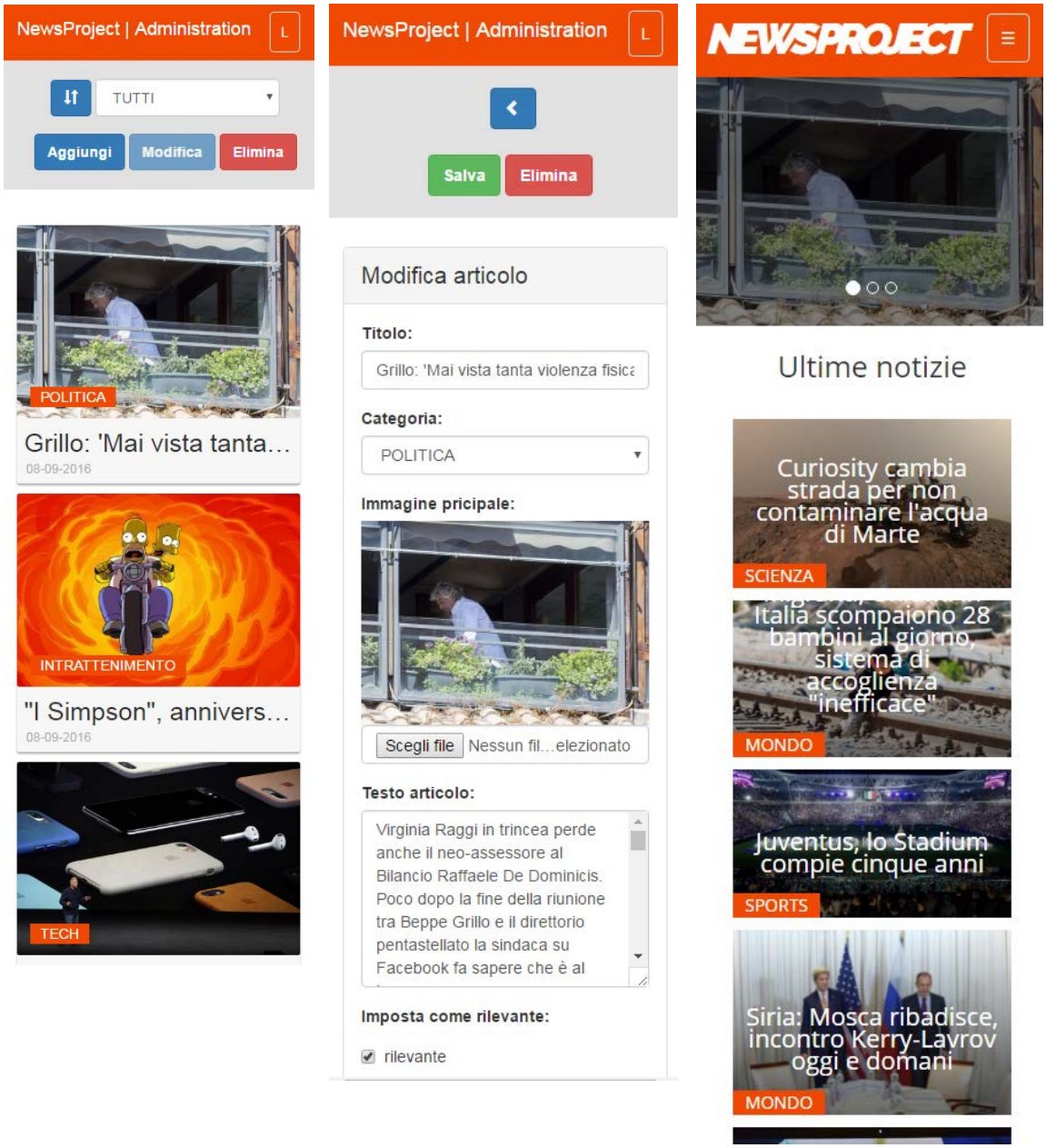
Tutto il layout del front-end è stato progettato secondo le buone pratiche di twitter boostra per la responsività con diverse risoluzioni per diversi dispositiviti.
La scelta dello stile grafico e dei colori si basa sul material design, colori e componenti grafici flat.

Nel menu sono presenti le categorie che è possibile selezionare per filtrare gli articoli.

Nella colonna di sinistra gli articoli sono ordinati dal più nuovo al più vecchio in base alla categoria selezionata.
Nella colonna centrale sono visionati tutti gli articoli in ordine sparso, filtrati in base alla categoria selezionata.
Nella colonna di destra sono visibili solo gli articoli impostati come rilevanti dall'admin.

Selezionando uno qualsiasi degli articoli visioneremo i propri dettagli, ed è possibile anche navigare sulla singola pagina grazie al menu e alla freccia indietro.

Alcuni screenshot dell'applicazione:

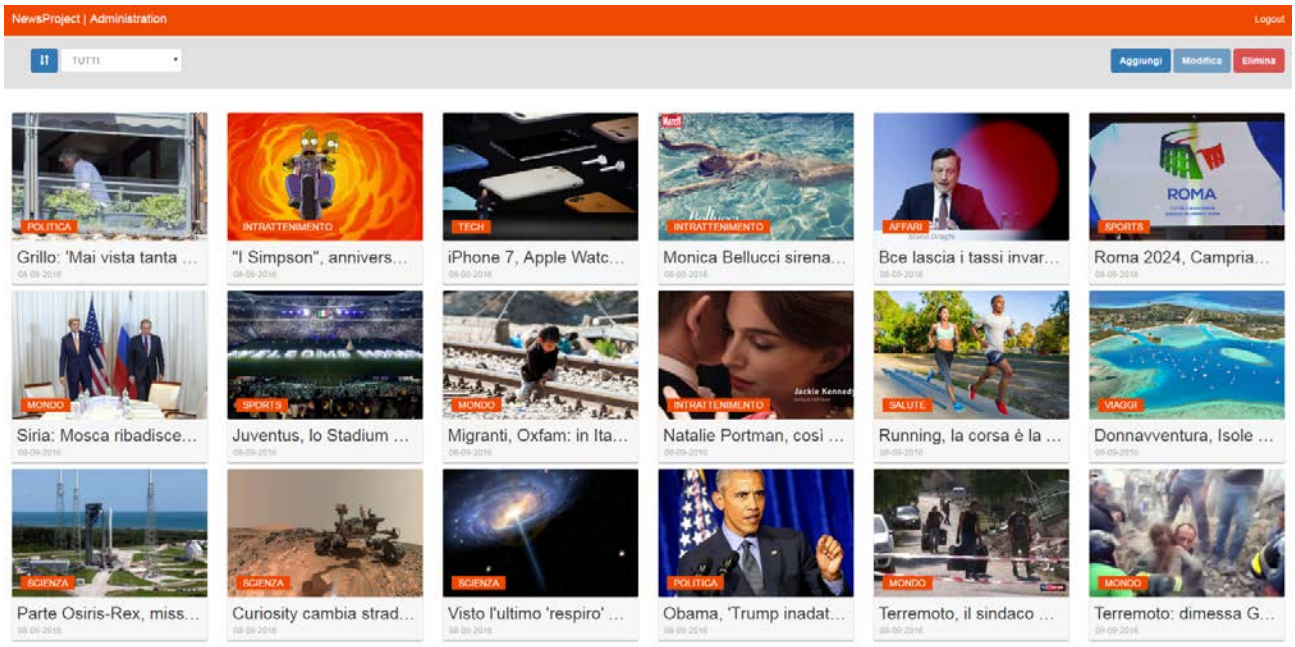


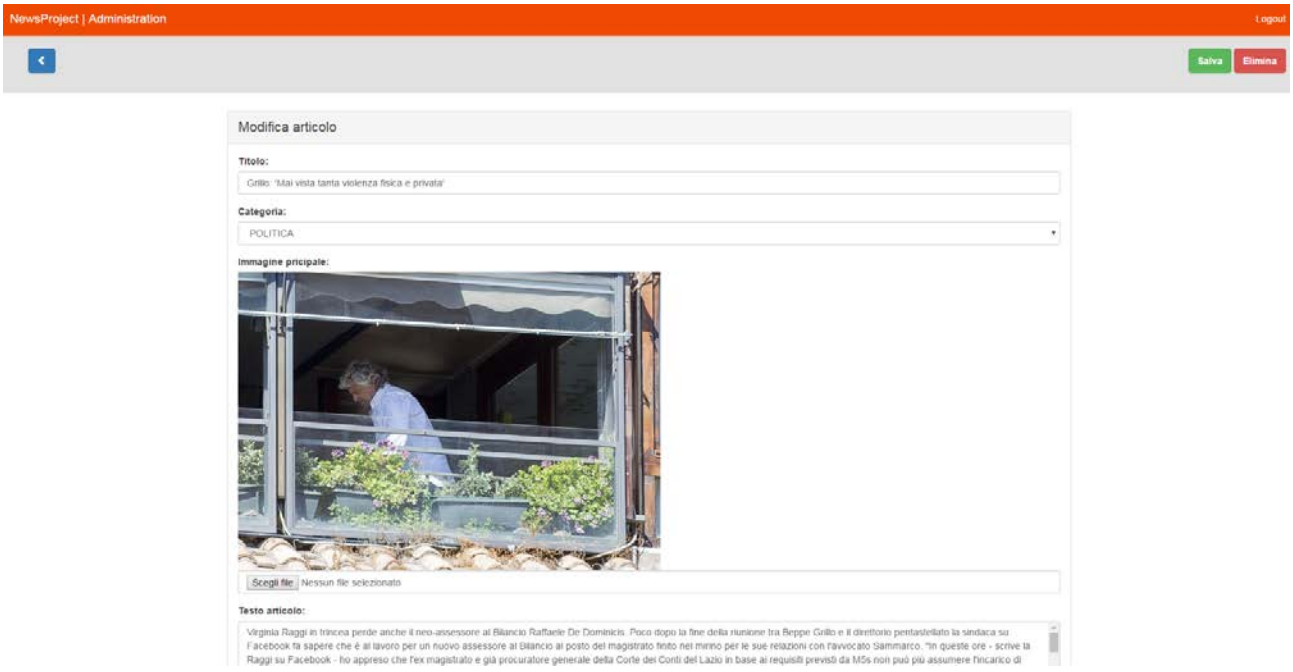
POLITICA

Grillo: 'Mai vista tanta violenza fisica e privata'

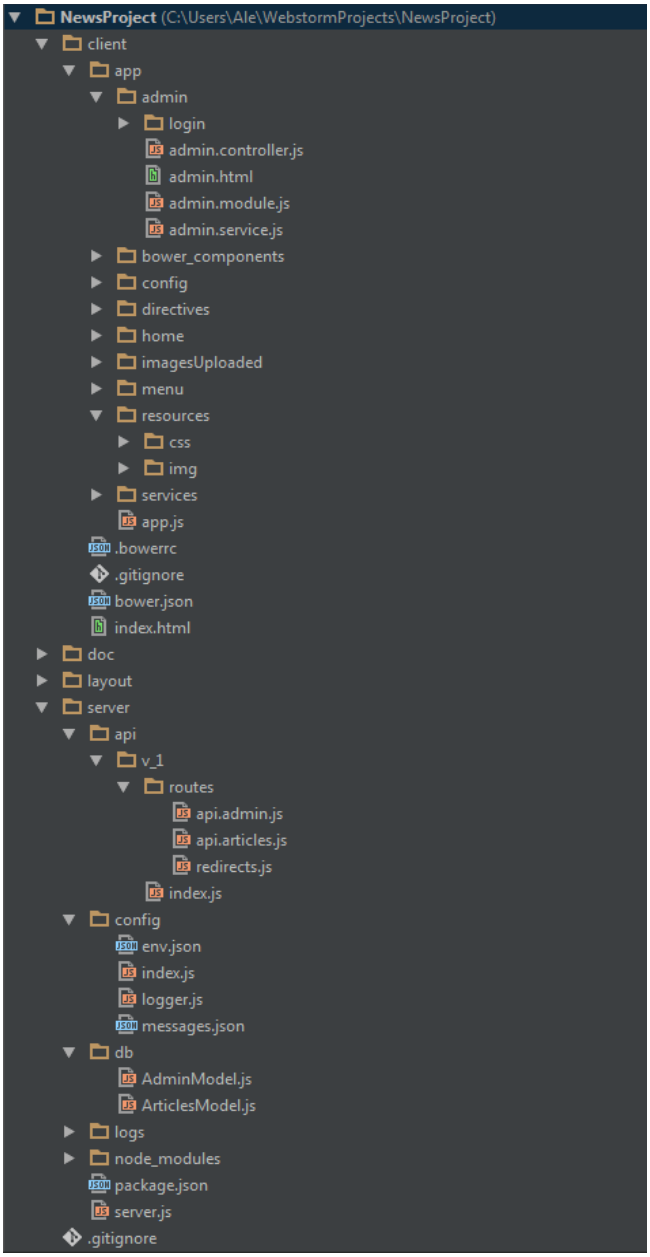
Virginia Raggi in trincea perde anche il neo-assessore al Bilancio Raffaele De Dominicis. Poco dopo la fine della riunione tra Beppe Grillo e il direttorio pentastellato la sindaca su Facebook fa sapere che è al lavoro per un nuovo assessore al Bilancio al posto del magistrato finito nel mirino per le sue relazioni con l'avvocato Sammarco. "In queste ore - scrive la Raggi su Facebook - ho appreso che l'ex magistrato e già procuratore generale della Corte dei Conti del Lazio in base ai requisiti previsti da M5s non può più assumere l'incarico di assessore al bilancio della giunta capitolina, per tanto di comune accordo abbiamo deciso di non proseguire con l'assegnazione dell'incarico". Al termine del vertice del direttorio del M5s. Beppe Grillo ha lasciato l'albergo romano da una uscita secondaria salendo a bordo di un Suv nero con i vetri oscurati. Ai


NEWS PROJECT
HOME SPORTS AFFARI TECH SCIENZA INTRATTENIMENTO SALUTE VIAGGI MONDO POLITICA
POLITICA
Grillo: 'Mai vista tanta violenza fisica e privata'
Ultime notizie
Tutte le news
Notizie rilevanti
Curiosity cambia strada per non contaminare l'acqua di Marte
SCIENZA
Migranti, Oxfam: in Italia scompaiono 28 bambini al giorno, sistema di accoglienza "inefficace"
MONDO
POLITICA
Grillo: 'Mai vista tanta violenza fisica e privata'
Virginia Raggi in trincea perde anche il neo-assessore al Bilancio Raffaele De Dominicis. Poco dopo la fine della riunione tra Beppe Grillo e il direttorio pentastellato la sindaca su Facebook fa sapere che è al lavoro per un nuovo assessore al Bilancio al posto del magistrato finito nel mirino per le sue relazioni con l'avvocato Sammarco. "In queste ore - scrive la Raggi su Facebook - ho appreso che l'ex magistrato e già procuratore generale della Corte dei Conti del
"I Simpson", anniversario d'argento per la serie animata più amata di sempre
Sono ben venticinque quest'anno le candeline sulla torta de "I Simpson". Dopo aver recentemente toccato la quota del 550esimo episodio, la serie animata più celebre e lunga della storia televisiva festeggia l'anniversario d'argento.
Grillo: 'Mai vista tanta violenza fisica e privata'
POLITICA
iPhone 7, Apple Watch 2, AirPods: ecco tutte le novità
TECH


NewsProject | Administration
Logout
TUTTI
Aggiungi
Modifica
Elimina
POLITICA
Grillo: 'Mai vista tanta ...
INTRATTENIMENTO
"I Simpson", annivers...
TECH
iPhone 7, Apple Watc...
INTRATTENIMENTO
Monica Bellucci sirena...
AFFARI
Bce lascia i tassi invar...
SPORTS
Roma 2024, Campria...
MONDO
Siria: Mosca ribadisce...
SPORTS
Juventus, lo Stadium ...
MONDO
Migranti, Oxfam: in Ita...
INTRATTENIMENTO
Natalie Portman, così ...
SALUTE
Running, la corsa è la ...
VIAGGI
Donnavventura, Isole ...
SCIENZA
Parte Osiris-Rex, miss...
SCIENZA
Curiosity cambia strad...
SCIENZA
Visto l'ultimo 'respiro' ...
POLITICA
Obama, 'Trump inadat...
MONDO
Terremoto, il sindaco ...
MONDO
Terremoto: dimessa G...


NewsProject | Administration
Logout
Salva
Elimina
Modifica articolo
Titolo:
Grillo: 'Mai vista tanta violenza fisica e privata'
Categoria:
POLITICA
Immagine pricipale:
Scegli file
Nessun file selezionato
Testo articolo:
Virginia Raggi in trincea perde anche il neo-assessore al Bilancio Raffaele De Dominicis. Poco dopo la fine della riunione tra Beppe Grillo e il direttorio pentastellato la sindaca su Facebook fa sapere che è al lavoro per un nuovo assessore al Bilancio al posto del magistrato finito nel mirino per le sue relazioni con l'avvocato Sammarco. "In queste ore - scrive la Raggi su Facebook - ho appreso che l'ex magistrato e già procuratore generale della Corte dei Conti del Lazio in base ai requisiti previsti da M5s non può più assumere l'incarico di

L’albero finale delle directory e dell’organizzazione scalabile, modulare, scalabile e facilmente gestibile:



Tutto il progetto è hostato su GitHub ed è attualmente visionabile all’ultima versione: [NewsProject]