

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 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;