Corso di Programmazione Web e Mobile

Progettazione e sviluppo di un'applicazione web

A.A. 2020 - 2021

Ernesto Pastori
https://github.com/ernepasto/news-app

Indice

1 Introduzione

Il progetto propone un'applicazione web il cui obiettivo è quello il fornire un servizio agli utenti: la possibilità di visualizzare una lista delle notizie in trend dei principali quotidiani italiani, con l'opportunità di leggere l'articolo completo tramite un link alla pagina del relativo giornale.

L'idea che ha portato al suo sviluppo è quella di fornire all'utenza la possibilità di informarsi sugli eventi correnti in rapido tempo, con la possibilità di scegliere in base ai propri interessi e tramite notizie accreditate.

Il funzionamento è semplice, una volta registrato, l'utente può effettuare l'accesso con le proprie credenziali e usufruire del servizio.

Tramite una pagina dedicata sarà poi possibile visualizzare la lista di articoli in trend e modificarne il contenuto in relazione alla proprie preferenze, scegliendo una catergoria tra: *generale, buisness, intrattenimento, salute, scienza, sport* e *tecnologia* (tramite apposito elemento select previsto da HTML). Inizialmente è proposta *generale* come categoria di notizie.

Sarà inoltre possibile modificare i dati relativi al profilo creato e anche eliminare quest'ultimo. I dati richiesti in fase di registrazione sono: *nome, cognome, email* e *password*, mentre in fase di login: *email* e *password*.

Per lo sviluppo delle interfacce grafiche è stato utilizzato Sass ossia un'estensione del linguaggio CSS che permette di utilizzare variabili, di creare funzioni e una migliore organizzazione del foglio di stile. Il linguaggio Sass si basa sul concetto di preprocessore CSS, il quale permette di generare file CSS ottimizzati, aggregando le strutture definite anche in modo complesso.

Per la creazione del web server è stato utilizzato Node js e vari moduli fondamentali per l'implementazione di alcune funzioni specifiche quali *express, bcrypt, express-session, mongoose, joi* e altri.

Invece per definire la struttura delle views è stato utilizzato il framework Vue js (un progressive framework per costruire interfacce utente) e alcune librerie per compiti specifici quali *vue-tosted* e *vue2-smooth-scroll*.

1.1 Breve analisi dei requisiti

1.1.1 Destinatari

L'applicazione web è rivolta a qualsiasi tipo di utenza grazie alla semplicità di accesso e al design intuitivo che presenta. Inoltre l'applicazione è fruibile anche da dispositivi mobile per l'approccio responsivo dato tramite codice CSS.

L'obiettivo di questa scelta risiede nel voler permettere al numero maggiore di utenti possibile di utilizzare tale applicazione.

1.1.2 Modello di valore

Il valore dell'applicazione descritta risiede nel servizio offerto e nell'obiettivo di rendere tale software il più diffuso e utilizzato possibile. In questa direzione sono state prese le scelte sul suo sviluppo: design responsivo e intuitivo e la disponibilità per ogni tipo di dispositivo tramite un qualsiasi browser web.

Da un punto di vista dell'investitore il valore di tale progetto si trova nella possibilità di direzionare il pubblico alla visione di un contenuto di natura pubblicitaria (advertising model), garantendone la visibilità.

Potrebbero inoltre essere applicati in futuro, tramite l'aggiunta di nuove funzionalità, i modelli freemium e subscription.

1.1.3 Flusso dei dati

I dati utilizzati dall'applicazione sono acquisiti, mediante chiamate a una API REST, in formato JSON che successivamente vengono utilizzati e posizionati nel layout mediante l'utilizzo di Vue, andando così a comporre pagine dinamiche che si adattano al contenuto ricevuto, permettendone la visualizzazione da parte dell'utenza, anche con contenuti personalizzati in base alle scelta (la scelta della categoria delle notizie visualizzate).

L'API è in grado di fornire notizie da testate giornalistiche differenti e una personalizzazione del contenuto della risposta aggiungendo alle chiamate delle query specifiche. I costi mensili legati all'utilizzo di tali risorse sono di circa euro 500.

Per la gestione degli utenti e del loro accesso il web server fa utilizzo di un database locale (Mongodb - NOSQL) dove sono salvati i relativi dati con l'uso di una funzione hash (e salt) per rendere sicura l'archiviazione delle password.

Tramite l'interrogazione del database si autorizza o meno l'accesso di un utente e si convalida la sua registrazione. Le informazioni sull'utente sono salvate e fruibili in formato JSON. Sono state inoltre implementate tecniche di validazione dell'input ricevuto per evitare attacchi di tipo XSS e Mongo Injection.

Per favorire l'usabilità dell'applicazione stessa è stato implementato un sistema per la gestione delle sessioni tramite l'utilizzo di cookie (della durata di 60 minuti).

All'utente inoltre è data la possibilità in completa autonomia di modificare ed eventualmente eliminare i dati relativi al proprio account tramite la pagina appositamente dedicata.

1.1.4 Aspetti tecnologici

La gestione del codice avviene tramite il modello Model View Controller.

La logica di presentazione del contenuto è affidata a Vue, si occupa inoltre di "catturare" le interazioni degli utenti con l'applicazione stessa e di generare chiamate HTTP (tramite l'API Javascript fetch) al web server. L'interazione con il database, l'autenticazione degli utenti, la validazione dell'input e le chiamate all'API che fornisce le notizie è demandata al server web.

Le tecnologie richieste sono state inserite, con l'aggiunta dell'utilizzo di Scss, fetch e Vue. Tali scelte derivano dalla volontà personale di scoprire nuove tecnologie adatte allo sviluppo di applicazioni web performanti, complesse, scalabili e moderne.

1.1.5 Altro

Per garantire un utilizzo facilitato dell'applicazione è stato introdotto, tramite la libreria *vue-tosted*, l'utilizzo di popup di differenti colori, in base al successo o all'errore dell'operazione.

Tali popup illustrano all'utente possibili informazioni sul servizio, in presenza di malfunzionamenti, e messaggi di errore nel caso di login errato oppure per la validazione dei campi in fase di registrazione (es: la password deve essere minimo di 8 caratteri). Gli errori o i messaggi illustrati dai popup sono generati dal web server, il quale in base all'azione eseguita risponde in maniera appropriata.

2 Interfacce

Nella Figura 1 vi è rappresentata la pagina principale dell'applicazione, il suo ruolo è direzionare l'utente verso altre pagine e di fornire informazioni riguardo al servizio offerto per attrarre l'utente nell'utilizzo dell'applicazione stessa.

Le successive interfacce rappresentano le pagine per il login e la registrazione degli utenti, sono presenti diversi capi input per l'acquisizione di dati che poi verranno inviati e convalidati dal web server tramite il click dell'apposito pulsante.

L'interfaccia che rappresenta la User page permette all'utente loggato di visualizzare i propri dati e di eseguire un aggiornamento di questi: sono

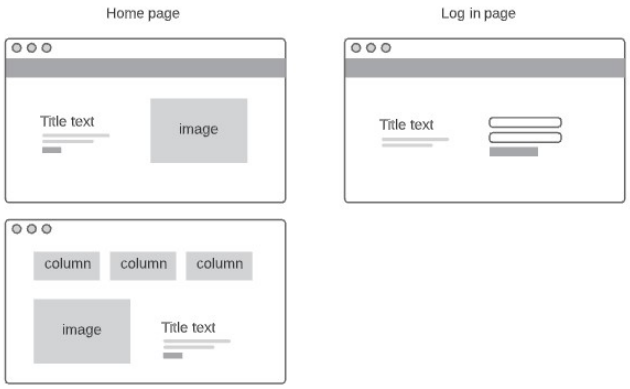

Figura 1: Schema interfacce

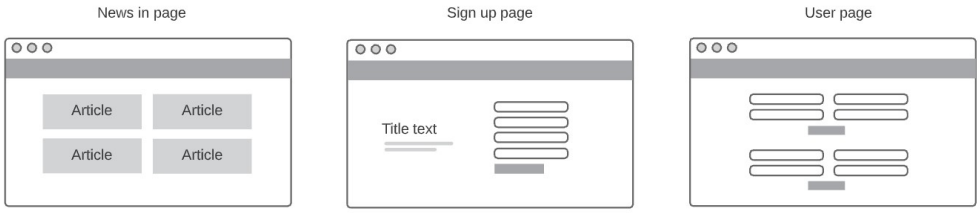

Figura 2: Schema interfacce

presenti campi di input per permettere l'acquisizione di nuovi dati e campi disabilitati per la visualizzazione delle informazioni presenti nel database. Tali dati sono ottenuti grazie ad una chiamata al web server che tramite il cookie (quindi l'id dell'utente) recupera le informazioni che verranno poi mostrate. Inoltre è possibile anche eliminare l'account creato con un apposito bottone, l'utente sarà poi indirizzato verso la pagina principale.

Nella Figura 2 è rappresentata l'interfaccia per la visualizzazione degli articoli, tale pagina ha il compito di illustrare le informazioni ottenute tramite le chiamate all'API. Al punto 4.2.4 vi è una parte del codice sorgente utilizzato per tale interfaccia.

3 Architettura

3.1 Diagramma dell'ordine gerarchico delle risorse

Nello schema in Figura 3 si possono vedere le pagine (views) che l'applicazione mette a disposizione degli utenti. Dalla schermata di home è possibile

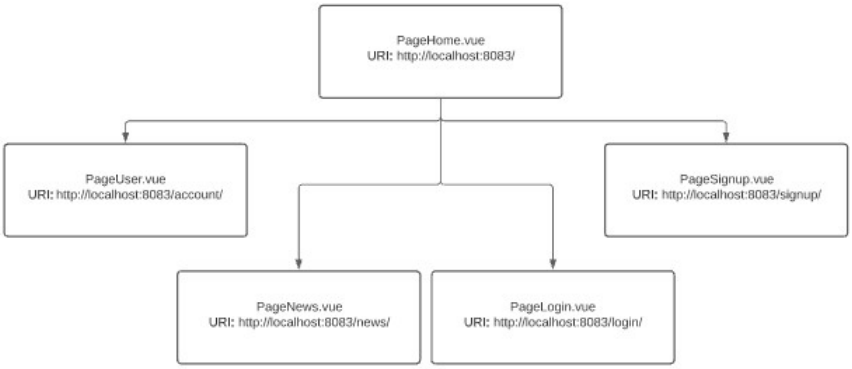

Figura 3: architettura dell'applicazione

raggiungere tramite una nav e alcuni bottoni le varie pagine.

La nav è un componente Vue (Nav.vue) presente in ogni pagina, ma in grado di variare il proprio contenuto in base alla pagine visualizzata, vi è un controllo sul cambiamento dei valori del router.

In caso di inserimento di un path errato il router gestisce tale errore andando a indirizzare l'utente sulla pagina principale dell'applicazione.

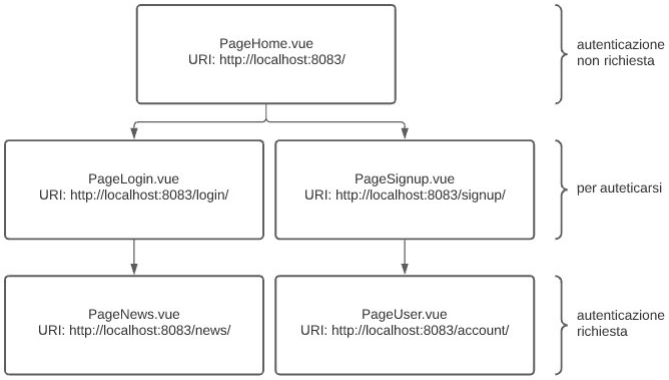

Figura 4: schema dei permessi

L'utente durante la navigazione dell'applicazione potrà accedere alle prime tre pagine in Figura 4. Successivamente alla sua autenticazione potrà invece accedere a tutte le pagine presenti. Tuttavia la visualizzazione della pagina di login e di registrazione sarà possibile solo dopo aver effettuato il logout.

Dopo l'autenticazione non sarà necessario inserire le credenziali di accesso (email e password) per la successiva ora, grazie all'implementazione di un cookie persistente.

La durata di tale cookie è definita tramite il campo maxAge nella configurazione lato server. L'informazione che contiene il cookie è l'id dell'utente. Tale dato viene generato al momento della registrazione, si tratta di un valore alfanumerico univoco per ogni utente.

Tramite questa informazione sono possibili alcune operazioni durante la navigazione dell'utente quali:

- controllo delle credenziali per la visualizzazione dei contenuti
- ricerca dell'utente per la visualizzazione delle credenziali
- ricerca dell'utente per la modifica delle credenziali

Sarà inoltre possibile eseguire la disconnessione tramite l'apposito pulsante "logout", avverrà quindi la rimozione del cookie lato server e l'indirizzamento verso la pagina home lato client.

Nel caso in cui un utente non loggato tenti di accedere a pagine riservate avverrà un indirizzamento verso la pagine di login (PageLogin.vue - vedi Figura 3) con notifica tramite un popup. Tale controllo avviene tramite un'apposita chiamata al web server, il quale controlla la presenza del cookie.

3.2 Descrizione delle risorse

Lo scambio di risorse mediante l'interazione dei componenti del MVC avviene nello stesso modo per tutte le interfacce presenti nell'applicazione, ad eccezione della pagina principale che non è orientata alla raccolta e allo scambio di alcuna risorsa.

Nell'applicazione le varie views costruite con Vue raccolgono le interazioni e i dati che sono inviati dall'utente. Successivamente avvengono delle chiamate http verso il web server (chiamate di tipo POST nel caso di login, signup e modifica dei dati, di tipo GET negli altri casi) il quale si occupa dell'elaborazione delle informazioni e dalla loro validazione, se necessario.

Il web server dopo aver terminato le proprie operazioni risponderà al client con dati in formato JSON che possono essere messaggi diretti all'utente oppure informazioni da visualizzare (nel caso di login e signup avviene anche l'invio del cookie).

3.3 Altri diagrammi

Nella Figura 5 sono mostrati i campi presenti nel database per il salvataggio degli utenti registrati. Nel campo *date* è presente la data e l'ora della registrazione dell'utente. Il campo *password* invece conterrà l'hash della password (e salt). Il campo *_id* viene generato dal database stesso al momento dell'inserimento dei dati nella collection.

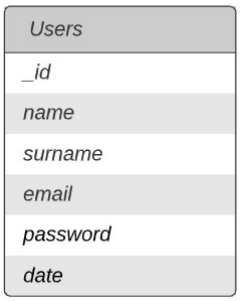

Figura 5: diagramma del database

4 Codice

4.1 Codice lato server

4.1.1 Pagina principale del web server

```
/* Express session configuration */
const session = require('express-session');
server.use(session({
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: false,
    cookie: {
        secure: false, // HTTPS connection
        httpOnly: true,
        maxAge: 1000 * 60 * 60 // 1 hour
    }
}));

```

```
/* Cross-origin resource sharing */
```

```
2 const cors = require('cors');
3 server.use(cors({ credentials: true, origin: ['http://
    localhost:8080'], methods: ['GET', 'PUT', 'POST'],
    allowedHeaders: ['Content-Type', 'Authorization'] }));
4
```

```
1 /* Database connection */
2 const mongoose = require('mongoose');
3 const mongooseOption = { useNewUrlParser: true,
    useUnifiedTopology: true, useCreateIndex: true,
    useFindAndModify: false };
4 mongoose.connect(process.env.DATABASE_ADDRESS, mongooseOption
    );
5
```

```
1 /* Enable posts route */
2 const newsRoute = require('./routes/news.routes');
3 server.use('/api/news/', newsRoute);
4
5 /* Enable users route */
6 const usersRoute = require('./routes/users.routes');
7 server.use('/api/users/', usersRoute);
8
```

Nel punto 4.1.1. è mostrata una parte del codice del file *index.js* ossia il file principale del web server. Avviene la configurazione dei cookie che saranno poi utilizzati durante il login e signup e successivamente per la verifica dell'avvenuta autenticazione dell'utente.

É mostrata la connessione al database tramite l'uso di *mongoose*, l'indirizzo passato come parametro viene acquisito dal file *.env*, è necessario l'utilizzo del modulo *dotenv*.

4.1.2 Validazione degli input

```
1 //user.validation.js
2 const joi = require('@hapi/joi');
3
4 const register = (data) => {
5     const schema = {
6         name: joi
7             .string()
```

```
8               .max(255)
9               .min(2)
10              .required(),
11          surname: joi
12              .string()
13              .max(255)
14              .min(2)
15              .required(),
16          email: joi
17              .string()
18              .email()
19              .max(255)
20              .min(6)
21              .required(),
22          password: joi
23              .string()
24              .max(1024)
25              .min(8)
26              .required(),
27          date: joi
28              .date()
29      };
30      return joi.validate(data, schema);
31  };
32
33  module.exports = { register, login, update, remove };
34
```

Nel codice è mostrata la validazione dei dati che vengono inviati al client tramite richiesta http.

La validazione serve a prevenire attacchi di tipo XSS e Mongo Injection oltre che a permettere un controllo sulla correttezza delle informazioni acquisite. Inoltre è stato introdotto l'obbligo di una password di una lunghezza minima di otto caratteri, in ottica di aumentare la sicurezza del profilo degli utenti.

4.1.3 Gestione delle autorizzazioni

```
1  //index.js
2  const verifyAutetication = (req, res, next) => {
3      if (req.isAuthenticated()) {
4          next();
5      } else {
6          res.status(401).send({ message: 'authentication
       required' });
```

```
    }
};

const verifyNoAutetication = (req, res, next) => {
    if (!req.isAuthenticated()) {
        next();
    } else {
        res.status(406).send({ message: 'you are already logged in' });
    }
};

module.exports = { verifyAutetication, verifyNoAutetication };

```

```
//passport.config.js
const Strategy = require('passport-local');
const bcrypt = require('bcryptjs');
const User = require('../models/user.model');

const strategyOptions = { usernameField: 'email', passwordField: 'password' };

module.exports = (passport) => {
    passport.use(new Strategy(strategyOptions, (email, password, done) => {
        /* Searching user with email */
        User.findOne({ email: email }, (error, user) => {

            if (error) return done(error);
            if (!user) return done(null, false);

            /* If user exists, password check */
            bcrypt.compare(password, user.password, (error, result) => {

                if (error) return done(error);
                if (result == true) return done(null, user);
                return done(null, false);

            });
        });
    }));

    /* Methods to manage session data */
    passport.serializeUser((user, cb) => {
```

```
29          cb(null, user.id);
30      });
31      passport.deserializeUser((id, cb) => {
32          User.findOne({ _id: id }, (error, user) => {
33              cb(error, user);
34          });
35      });
36
37  };
38
```

Sono mostrate le linee di codice necessarie per verificare se l'utente è autenticato o meno (ossia se in possesso di cookie) e per effettuare il login dello stesso: controllo dei dati inseriti e assegnazione del relativo cookie.

4.1.4 Rotte utenti

```
1  //users.routes.js
2  const validation = require('../validation/user.validation');
3  const User = require('../models/user.model');
4  const auth = require('../auth');
5  const bcrypt = require('bcryptjs');
6  const express = require('express');
7  const router = express.Router();
8
9  /* Parse data from client */
10 router.use(express.json());
11
12 /* Route for user registration */
13 router.post('/signup', auth.verifyNoAutetication, async (req,
       res) => {
14     const { error } = validation.register(req.body);
15     if (error) return res.status(400).send({ message: error.
       details[0].message });
16
17     const email = await User.findOne({ email: req.body.email
       });
18     if (email) return res.status(400).send({ message: 'email
       already exists' });
19
20     const salt = await bcrypt.genSalt(10);
21     const password = await bcrypt.hash(req.body.password,
       salt);
22
23     const user = new User({
24         name: req.body.name,
```

```
25          surname: req.body.surname,
26          email: req.body.email,
27          password: password
28      });
29
30      try {
31          await user.save();
32
33          req.login(user, (error) => {
34              if (error) return res.status(400).send({ message:
        'system error' });
35          });
36          res.status(200).send({ message: 'user registered' });
37      } catch (error) {
38          res.status(400).send({ message: 'system error' });
39      }
40  });
41
```

Nel codice viene illustrata la procedura necessaria per la registrazione di un utente. Avvengono diversi controlli e la generazione dell'hash della password.

4.1.5 Rotte per le news

```
1   /* Route to get news by category */
2   router.get('/:category', auth.verifyAutetication, async (req,
        res) => {
3       const dataset = `https://newsapi.org/v2/top-headlines?
        country=it&category=${req.params.category}&apiKey=${
        process.env.API_ACCESS_KEY}`;
4       fetch(dataset, { method: 'GET' })
5           .then(data => data.json())
6           .then(response => {
7               if (response.status === 'ok') {
8                   const news = response.articles;
9                   return res.status(200).send(news);
10              } else res.status(400).send({ message: 'API error
        ' })
11          })
12          .catch(error => { res.status(400).send({ message:
        error }) });
13  });
14
15  module.exports = router;
16
```

Il codice utilizzato per richiedere tramite il web server le notizie fornite dall'API. Vi è un controllo sull'autenticazione dell'utente e successivamente una chiamata http effettuata grazie al modulo *node-fetch*, in caso di successo restituisce al client i dati in formato JSON.

4.2 Codice lato client

4.2.1 Vue Router

```
1  import Vue from 'vue';
2  import VueRouter from 'vue-router';
3
4  import store from '../store';
5
6  Vue.use(VueRouter);
7
8  const lazyLoad = (view) => {
9      return () => import(`@/views/${view}.vue`);
10 }
11
12 const routes = [
13     {
14         path: '/',
15         name: 'PageHome',
16         alias: '/home/',
17         component: lazyLoad('PageHome'),
18         pathToRegexpOptions: { strict: true }
19     },
20     {
21         path: '/login/',
22         name: 'PageLogin',
23         component: lazyLoad('PageLogin'),
24         pathToRegexpOptions: { strict: true }
25     },     {
26         path: '/*',
27         redirect: '/'
28     }
29 ];
30
31 const router = new VueRouter({
32     mode: 'history',
33     //base: process.env.BASE_URL,
34     routes,
35 });
```

```
36
37 router.beforeEach((to, from, next) => {
38     if (store.getters.isStored) return next();
39     store.dispatch('search');
40     next();
41 });
42
43 export default router;
44
```

4.2.2 Vue store

```
1  actions: {
2      signup({ commit }, credentials) {
3        fetch('http://localhost:8083/api/users/signup', {
4            method: 'POST',
5            credentials: 'include',
6            headers: {
7                'Content-Type': 'application/json',
8            },
9            body: credentials
10       })
11       .then(async data => {
12           return { data: await data.json(), status: data.
     status };
13       })
14       .then(response => {
15           if (response.status !== 200) {
16               Vue.$toast.warning(response.data.message);
17           } else {
18               const data = JSON.parse(credentials);
19               commit('insertUser', data);
20               router.replace({ name: 'PageNews' });
21           }
22       })
23       .catch((error) => {
24           Vue.$toast.error('server error, try later');
25           console.error('Registration error:', error);
26       });
27   },
28   login({ commit }, credentials) {
29       fetch('http://localhost:8083/api/users/login', {
30           method: 'POST',
31           credentials: 'include',
32           headers: {
33               'Content-Type': 'application/json'
```

```
34              },
35              body: credentials
36          })
37          .then(async data => {
38              if (data.status === 200) return { data: await data.
    json(), status: data.status };
39              else if (data.status === 406) return { data: await
    data.json(), status: data.status };
40              else return { data: { message: 'invalid email or
    password' }, status: data.status };
41          })
42          .then(response => {
43              if (response.status !== 200) {
44                  Vue.$toast.warning(response.data.message);
45              } else {
46                  router.replace({ name: 'PageNews' });
47              }
48          })
49          .catch((error) => {
50              Vue.$toast.error('server error, try later');
51              console.error('Login error:', error);
52          });
53  },
54
```

Vue store e *Vue router* sono due moduli installati tramite *npm* e successivamente importati, servono rispettivamente per la gestione delle azioni legate all'utente e del salvataggio dei dati di quest'ultimo (i dati presenti nello store sono visibili in tutte le views) e per il routing delle pagine (con la possibilità di eseguire controlli o altre azioni).

4.2.3 Animazione della Nav

```
1  methods: {
2      toggleMenu: function() {
3          this.isVisible ? this.isVisible = false : this.
    isVisible = true;
4      },
5      onScroll: function() {
6          const scrolling = window.scrollY;
7          scrolling > 100 ? this.isScrolled = true : this.
    isScrolled = false;
8      },
9      onResize: function() {
10         const resize = window.innerWidth;
11         if (resize < 768) {
```

```
12              this.isSmall = true;
13              this.isVisible = false;
14          } else {
15              this.isSmall = false;
16              this.isVisible = true;
17          }
18      }
19  },
20
```

4.2.4 Pagine relativa alle notizie

```
1   <template>
2       <div id="PageNews">
3           <div class="articles-container">
4               <div v-bind:class="{ removed: showPage }" class="
    loading-view">
5                   <h1 class="loading-view-title">Wait just a
    second...</h1>
6               </div> <!-- /.loading-view -->
7               <h1 class="articles-title">Scegli, clicca <span
    >&</span> leggi</h1>
8               <div class="articles-category">
9                   <p>Scegli una categoria di notizie:</p>
10                  <select v-model="category">
11                      <option value="general">Generale</option>
12                      <option value="business">Business</option
    >
13                      <option value="entertainment">
    Intrattenimento</option>
14                      <option value="health">Salute</option>
15                      <option value="science">Scienza</option>
16                      <option value="sports">Sport</option>
17                      <option value="technology">Tecnologia</
    option>
18                  </select>
19              </div> <!-- /.article-category -->
20              <div class="articles-grid">
21                  <div v-for="article in news" :key="article.
    title" class="article">
22                      <a v-bind:href="article.url" target="
    _blank">
23                          <div class="article-preview-wrapper">
24                              <div class="article-filer"></div>
```

```
                            <img v-bind:src="article.urlToImage" class="article-image" v-bind:alt="'article-image-'+ article.source.name">
                            <h4 class="article-title">{{ article.title }}</h4>
                        </div> <!-- /.post-preview-wrapper -->
                        <div class="article-info">
                            <span>{{ article.publishedAt | formatDate }}</span> -
                            <span>{{ article.source.name }}</span>
                        </div> <!-- /.post-info -->
                    </a>
                </div> <!-- /.post -->
            </div> <!-- /.posts-grid -->
        </div> <!-- /.posts-container -->
    </div> <!-- /#PageNews -->
</template>

```

```
onLoad: function() {
    fetch('http://localhost:8083/api/news', {
        method: 'GET',
        credentials: 'include',
        headers: {
            'Content-Type': 'application/json',
        }
    })
    .then(async data => {
        return { data: await data.json(), status: data.status
    };
    })
    .then(response => {
        if (response.status !== 200) {
            this.$toast.warning(response.data.message);
            this.$router.replace({ name: 'PageLogin' });
        } else this.news = response.data;
    })
    .catch(error => {
        console.error(error);
        this.$toast.error('System error, try later');
    });
},


watch: {
```

```
26        category: function (selected) {
27            this.category = selected;
28            this.loadByCategory();
29        },
30    },
31
```

```
1  .articles-grid {
2          width: 100%;
3          display: grid;
4          grid-template-columns: repeat(2, 1fr);
5          column-gap: 20px;
6          row-gap: 50px;
7          @media screen and (max-width: 1000px) {
8              grid-template-columns: 100%;
9          }
10         .article {
11             width: 100%;
12             cursor: pointer;
13             justify-self: center;
14             @media screen and (max-width: 1000px) {
15                 max-width: 70%;
16             }
17             @media screen and (max-width: 700px) {
18                 max-width: 100%;
19             }
20             a {
21                 text-decoration: none;
22             }
23             .article-preview-wrapper {
24                 max-height: 250px;
25                 width: 100%;
26                 position: relative;
27                 overflow: hidden;
28                 margin-bottom: 1em;
29                 border: solid 1px #f9f9f9;
30                 border-radius: 6px;
31                 background-size: cover;
32                 background-position: center center;
33                 box-shadow: 8px 8px 20px rgba($color: #000,
     $alpha: .005);
34                 .article-filer {
35                     height: 100%;
36                     width: 100%;
37                     position: absolute;
38                     top: 0;
39                     left: 0;
```

```
40                      z-index: 1;
41                      background: linear-gradient(0deg, rgba
    (0,0,0,.9) 0%,
42                      rgba(0,0,0,0.4) 35%, rgba(0,0,0,0.2) 55%,
     rgba(0,0,0,0) 75%,
43                      rgba(0,0,0,0) 100%);
44                  }
45                  .article-image {
46                      height: auto;
47                      width: 100%;
48                      display: block;
49                  }
50                  .article-title {
51                      position: absolute;
52                      bottom: 0;
53                      left: 0;
54                      z-index: 2;
55                      padding: 25px;
56                      color: #fff;
57                      font-size: 1.3em;
58                      font-weight: 500;
59                  }
60              }
61              .article-info {
62                  color: #999;
63                  font-size: .95em;
64                  span {
65                      display: inline-block;
66                      color: #999;
67                      font-size: .95em;
68                      font-style: italic;
69                  }
70              }
71          }
72      }
73  }
74
```

Il codice mostrato è utilizzato per la presentazione all'utente delle notizie acquisite mediante una chiamata http al web server. La risposta ottenuta sarà poi utilizzata per inserire i dati nel *template* ossia nel codice HTML.

Inoltre è mostrato il codice Scss necessario per definire lo stile della pagina. Sono presenti le *media-query* per la definizione del comportamento della pagine nei dispositivi mobile.