ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

Corso di Studi in Ingegneria e Scienze Informatiche

Tecnologie per la fruizione e rappresentazione di open data

Relazione finale in
Tecnologie Web

Relatore
prof.ssa Paola Salomoni

Correlatore
prof.ssa Silvia Mirri

Candidato
William Di Luigi

Sessione II

Anno accademico 2014 – 2015

Indice

Introduzione

La presente tesi è uno studio sugli strumenti e le tecnologie che caratterizzano l'utilizzo degli open data, in particolare, nello sviluppo di applicazioni web moderne che fanno uso di questo tipo di dati.

Il contesto in cui si colloca la tesi è quindi quello della produzione, rappresentazione e visualizzazione di open data.

Particolare attenzione verrà posta al caso della spesa pubblica delle amministrazioni italiane. Infatti, come vedremo, in Italia sono state avviate diverse iniziative atte a rendere più trasparente la gestione delle risorse pubbliche. Questo ci permette quindi di avere a disposizione dati reali che possiamo usare, aggregare e rielaborare liberamente.

Sarà oggetto di studio anche il portale `http://soldipubblici.gov.it`, che è un'interfaccia realizzata dall'AgID per offrire un accesso comodo e diretto a questi dati, senza avere necessità di doverli scaricare e analizzare autonomamente.

Dal momento che l'obiettivo finale sarà la realizzazione di un'applicazione web, un altro tema cardine del volume di tesi sarà quello dell'usabilità e dell'accessibilità.

L'obiettivo e caso di studio principale della presente tesi è lo sviluppo di un'applicazione web atta ad esporre una visualizzazione dei dati relativi alla spesa pubblica nelle amministrazioni italiane.

Tale visualizzazione sarà sostanzialmente un resoconto degli open data pubblicati dalle amministrazioni e raccolti dal sistema di rilevazione telematica degli incassi SIOPE (Sistema informativo sulle operazioni degli enti pubblici).

Sarà quindi obiettivo del progetto anche quello di decidere come riassumere i dati in modo da semplificare la comunicazione delle caratteristiche più

significative del nostro dataset.

Per fare questo ci avvarremo della possibilità di costruire una nuova banca dati, derivata da quella offerta dal portale soldipubblici. Infatti, l'assenza di restrizioni sull'utilizzo e la rielaborazione di questi dati fa sì che non ci siano limiti su come aggregarli e semplificarli.

La nostra applicazione, tra l'altro, renderà possibile decidere a quale periodo di tempo i dati da considerare debbano far riferimento. In questo modo un'utente può selezionare ad esempio l'anno di interesse e vedere quindi la visualizzazione dei dati relativi a quel particolare anno.

L'applicazione realizzata verrà messa a disposizione attraverso un sito web, raggiungibile tramite l'URL:

http://spesapubblica.in

Il codice sorgente dell'applicazione, oltre a tutti gli script per la riproduzione della banca dati, verrà inoltre pubblicato su Github sotto licenza aperta. Sarà accessibile all'indirizzo:

https://github.com/gabrfarina/spesapubblica

Il lavoro di tesi si suddivide in tre parti, che corrispondono ai capitoli di cui si compone il volume.

- Nel primo capitolo verranno trattate le fondamenta del lavoro di tesi, vale a dire: i concetti di open data e open government, i requisiti che i dati devono rispettare al fine di poter essere utilizzati in modo libero.

  Vedremo anche l'importanza legale dell'accessibilità nelle applicazioni web, oltre che a quali sono alcune tecniche e strumenti che si possono impiegare per migliorare l'usabilità.

  Passeremo quindi in rassegna alcune delle tecnologie comunemente usate nel contesto della visualizzazione e della rappresentazione dei dati, focalizzandoci sul caso dei dati cartografici.

- Il secondo capitolo sarà quello in cui entreremo nel dettaglio del lavoro di tesi. Analizzeremo il portale soldipubblici per capire quali casi d'uso non coperti da questa applicazione si potrebbero implementare nel

nostro progetto. Delineeremo quindi i requisiti precisi del progetto di tesi, e decideremo la struttura dell'interfaccia utente.

- L'ultimo capitolo sarà infine dedicato alla fase di implementazione. Introdurremo quindi tutte quelle tecnologie che abbiamo utilizzato per sviluppare l'applicazione ma che non sono state centrali nello specifico compito della visualizzazione dati. Inoltre descriveremo in dettaglio il procedimento di preparazione della banca dati usata.

Inoltre è degno di nota il fatto che il portale soldipubblici, citato nella sezione 2.1, ha implementato una visualizzazione geografica simile alla nostra. Questa modifica, apparsa sul portale verso la fine di novembre, è il frutto di un lavoro indipendente da quello che ho svolto nel contesto di questa tesi.

Nonostante questa recente aggiunta possa apparire simile nell'aspetto e nelle modalità di interazione a quella esposta in questo lavoro, vale la pena sottolineare alcune differenze chiave:

- Non sembra esserci modo di visualizzare la mappa con una risoluzione maggiore di quella delle province.
- Non è presente una barra di ricerca.
- I codici di spesa selezionabili sono in minor numero.
- È presente una comoda legenda che associa la gradazione del colore alla fascia di spesa.
- La tonalità di colore sembra essere fatta in modo da variare in scala assoluta e non relativa.

1 Open data: visualizzazione e interazione

Questo primo capitolo costituirà le fondamenta sulle quali si baserà il lavoro di tesi. Gli argomenti verranno esposti tramite una suddivisione in tre parti. Le sezioni tratteranno, rispettivamente: open data, human-computer interaction e visualizzazione dati.

1.1 Open data

In questa sezione:

- Definiremo i concetti di open data e open government.
- Delineeremo i passaggi storici che hanno portato all'adozione sempre più ampia di forme di governo aperte.
- Concluderemo con una riflessione sull'importanza di specificare chiaramente la fonte da cui provengono i dati usati.

1.1.1 Definizione di open data

Con open data (o *dati aperti*) intendiamo quei dati che ammettono libero utilizzo, riutilizzo e ridistribuzione con l'eventuale vincolo di attribuzione e/o condivisione soggetta ai medesimi termini [Ope].

Requisiti dell'open data

Il requisito fondamentale dell'open data è l'interoperabilità. Per esempio, una lista di pagamenti scansionati in formato JPG non è interoperabile quanto la

stessa lista in formato CSV. Inoltre, al fine di rendere interoperabile l'open data, è necessario che esso sia:

Accessibile
: I dati devono essere interamente ottenibili ad un costo non superiore di quello necessario per riprodurli. Ad esempio: il costo della banda per scaricarli da Internet [Wha].

Riusabile
: I dati devono godere di una licenza che permetta il riutilizzo, la ridistribuzione, e la creazione di derivati. Ad esempio: mescolare i dati con altri dataset [Wha].

Tipi di dati

I tipi di open data possono essere molteplici, possiamo però cercare di raggrupparli in queste categorie:

Dati strutturati
: Parleremo di dati strutturati quando questi risiedono in un campo a grandezza fissa all'interno di un record o un file. Tra i dati strutturati quindi troviamo quelli contenuti nei database relazionali e nei fogli di calcolo.

  Il linguaggio SQL (da *Structured Query Language*) nasce proprio per gestire dati strutturati nel contesto dei database relazionali.

Dati geolocalizzati
: Parleremo di dati geolocalizzati quando questi fanno riferimento ad una delimitata regione geografica. Per esempio, le spese prodotte dall'amministrazione pubblica del comune di Roma.

Dati aggregati
: Parleremo di dati aggregati quando questi saranno ottenuti aggregando, appunto, altri dati. Per esempio i dati sulle spese relative alla cancelleria possono essere calcolati come dati aggregati, se si sommano le spese relative alla carta e quelle relative alle penne.

Provenienza dei dati

Come mostrato nel 1999, molti siti web contengono informazioni false o imprecise. A causa della diffusione di queste informazioni, gli utenti del Web stanno diventando sempre più scettici quando consultano dei contenuti online [Fur09]. Per questo motivo, nel momento in cui si progetta un'applicazione web che dipende da informazioni — ad esempio dati e statistiche — è opportuno adottare delle misure che permettano all'utente finale di fidarsi di ciò che legge.

Nel caso in cui l'applicazione facesse uso di dati statistici, ad esempio i risultati di un benchmark, è opportuno informare l'utente sulla metodologia di testing e per quanto possibile esplicitarne i passaggi in modo tale che egli possa ripetere e verificare la sperimentazione. Se invece l'applicazione facesse uso di dati ufficiali — ad esempio forniti dal governo — è opportuno fornire chiare indicazioni su come e dove reperire i dati utilizzati.

Nel 2002, lo Stanford Persuasive Technology Lab ha analizzato i commenti di 2440 utenti a proposito dei fattori che promuovono la credibilità di un sito. La ricerca ha portato a stabilire che il font scelto può impattare positivamente o negativamente la percezione della credibilità. Quasi la metà degli utenti (46%) basava la credibilità del sito in parte sul suo design. Tra gli elementi critici c'erano: il layout, i font scelti, la grandezza dei caratteri e gli schemi di colori [Fur09].

È bene quindi, in linea di principio, scegliere accuratamente il tipo di carattere da utilizzare. Un buon criterio per fare questa scelta è considerare il contesto dell'applicazione e valutare qual è il target: non è opportuno, per esempio, utilizzare un font allegro e giocoso per un'applicazione che si occupa di quote finanziarie.

1.1.2 Definizione di open government

Con il termine *open government* ci riferiamo all'idea secondo la quale ai cittadini si dovrebbe garantire il diritto di accedere ai documenti e ai verbali governativi in modo da permettere un efficace vigilanza pubblica [Lat10]. Per questo motivo, la tendenza delle amministrazioni pubbliche a produrre open data rientra nel concetto di open government.

L'idea di un governo aperto al pubblico scrutinio e suscettibile all'opinione pubblica, comunque, non è stata introdotta con l'avvento di Internet. Già ai tempi dell'Illuminismo infatti, molti filosofi attaccarono la dottrina assolutista della segretezza dello stato [Hab62][Kos59]. Si può quindi dire che l'open government faceva parte del progetto intellettuale illuminista.

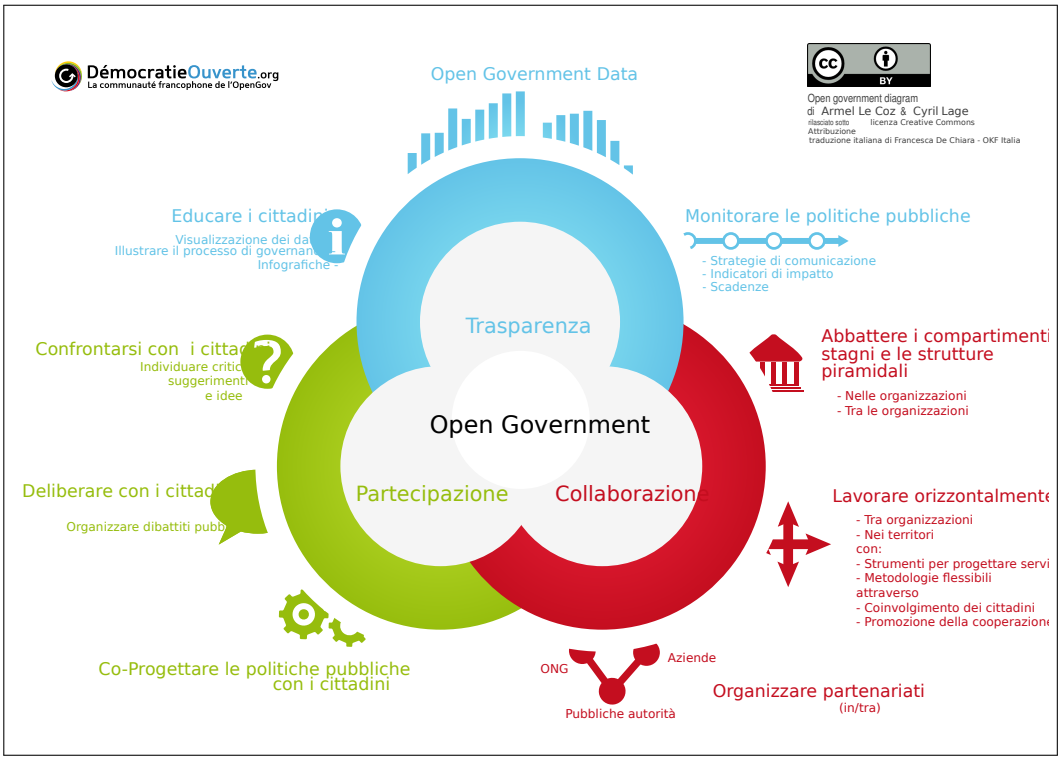


Figura 1: Schema descrittivo degli ingredienti dell'open government
(a cura di Armel Le Coz e Cyril Lage – `flickr.com/photos/46274765@N07/8600245980`)

Nei tempi moderni, l'open government è utilizzato come mezzo per aumentare la trasparenza dei dati e per incoraggiare una maggiore partecipazione cittadina nelle scelte pubbliche.

Open source governance

Con i recenti sviluppi dell'open government è nata la teoria dell'open source governance. La componente che differenzia questa nuova idea dall'open government è la maggiore enfasi che viene data alla componente della collaborazione [Osg].

L'open governance esorta l'applicazione della filosofia open-source ai principi della democrazia, così da dare a tutti i cittadini interessati la possibilità di contribuire alla creazione delle leggi.

In un discorso tenuto nel giugno 2012 per la conferenza TED, lo scrittore Clay Shirky ha descritto come l'avvento dei *distributed version control system* abbia aperto la strada alla possibilità di legiferare in modo trasparente e collaborativo [Shi12].

Come si vede in fig. 2, nel maggio 2015 c'è stata una prova della fattibilità dell'argomentazione di Shirky. Il politico statunitense Gerry Connolly ha aperto una *pull request* sul repository relativo alla gestione e supervisione delle risorse informatiche (all'indirizzo `https://github.com/WhiteHouse/fitara/pull/39`) e questa è stata approvata dopo meno di due ore da Tony Scott, il Federal Chief Information Officer.

1.1.3 Iniziative in direzione dell'*openness*

In Italia sono state avviate diverse iniziative per far in modo che i dati delle pubbliche amministrazioni siano il più facilmente possibile accessibili ai cittadini.

Linee Guida per i siti Web della PA

Con le "Linee Guida per i siti Web della PA" pubblicate ai sensi dell'art. 4 Direttiva del Ministro per la Pubblica Amministrazione e l'Innovazione del 26 novembre 2009, n. 8, sono stati forniti alle amministrazioni pubbliche criteri e prescrizioni da seguire nell'attività di razionalizzazione dei siti web della PA [PA11].

Il documento detta principi in materia di caratteristiche tecniche, alla luce dei quali razionalizzare i siti web dei vari enti, dettando principi comuni in materia di realizzazione, gestione, sviluppo e aggiornamento dei contenuti pubblicati dalle Pubbliche Amministrazioni, nonché in tema di fornitura di servizi online [PA11].

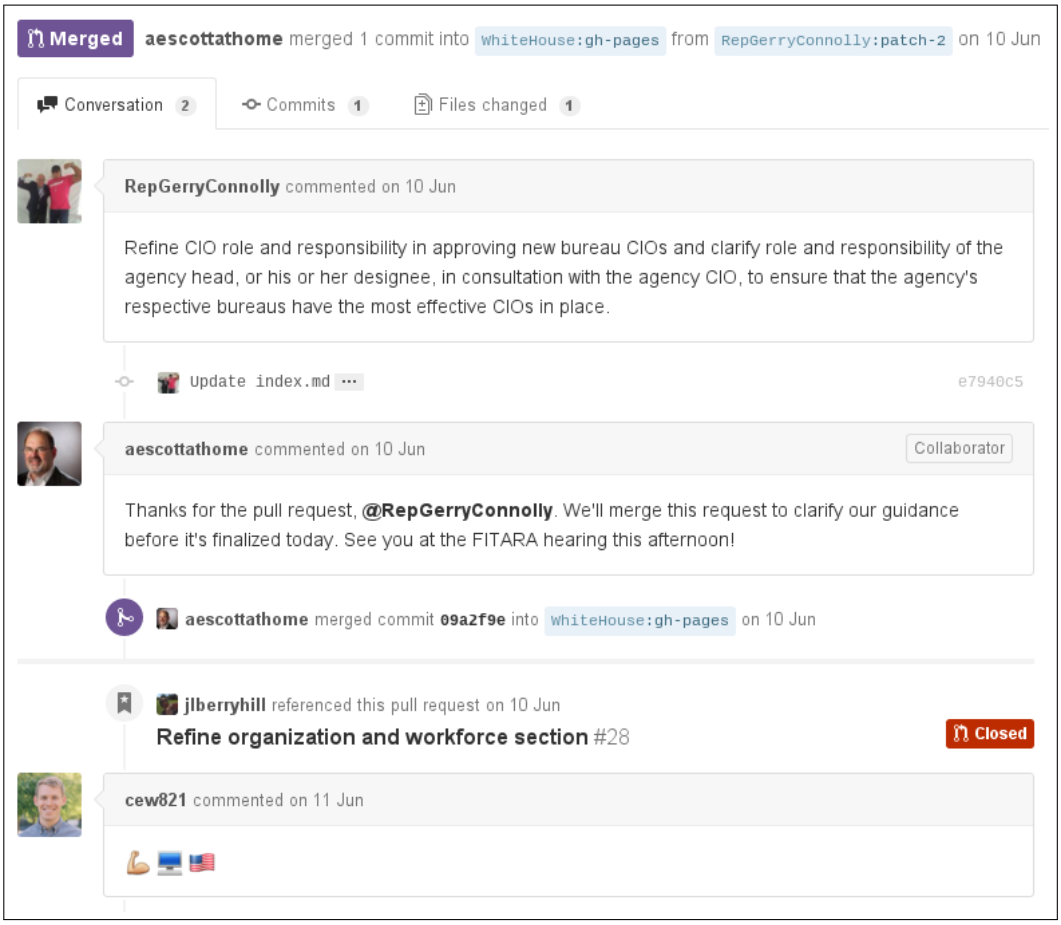


Figura 2: La pull request aperta dal politico Gerry Connolly (**@RepGerryConnolly**) e approvata da Tony Scott (**@aescottathome**)

Portale: dati.gov.it

Il portale dati.gov.it ospita dal 2011 il catalogo degli open data pubblicati da Ministeri, Regioni ed Enti Locali. L'Agenzia per l'Italia Digitale ha revisionato il portale nel 2015 riorganizzando i dataset e aggiornando la veste grafica del sito [AGI15].

Al novembre 2015 sono presenti 10 348 dataset.

Sito web: soldipubblici.gov.it

Il progetto dell'Agenzia per l'Italia Digitale *soldipubblici* (raggiungibile all'indirizzo `soldipubblici.gov.it`) è stato realizzato da una collaborazione

tra il Laboratorio MoSIS (Modelli e sistemi informativi statistici) del PIN, il Polo Universitario "Città di Prato" dell'Università di Firenze, la Banca d'Italia ed il coordinamento di Cristina Martelli dell'Università di Firenze e di Giovanni Menduni del Politecnico di Milano.

Il portale soldipubblici ha l'obiettivo di portare trasparenza nelle spese della pubblica amministrazione. In particolare, è possibile selezionare l'ente pubblico ed una tipologia di spesa. Verrà così mostrata una visualizzazione dei dati relativi a quella spesa, con la possibilità di considerare anche un periodo temporale.

1.2 Human-computer interaction

In questa sezione introdurremo brevemente l'avvento del Web 2.0 e la conseguente crescita della domanda di interfacce usabili e accessibili. Inoltre, ripasseremo gli aspetti legali relativi all'accessibilità. Passeremo poi in rassegna alcune tecniche per l'ottimizzazione delle applicazioni web. Concluderemo con una lista di strumenti che permettono di testare automaticamente un'applicazione web e rilevare possibili margini di miglioramento.

1.2.1 L'avvento del Web 2.0

Fin da sempre il principio fondante del Web è quello di diffondere e condividere informazioni. Tuttavia, prima dell'avvento dei moderni motori di ricerca, era a carico dell'utente dipanare l'enorme matassa dei collegamenti proposti, alla ricerca delle informazioni desiderate.

Inizialmente, il Web consisteva nella ripetizione di click, attese, e caricamenti di pagine. L'utente digitava l'indirizzo di destinazione, e la pagina veniva caricata in memoria. Per interagire con la pagina e ottenere maggiori dettagli, era necessario riempire, direttamente o indirettamente, un *web form*, fare click sul pulsante di inoltro dati, e attendere il caricamento della nuova pagina.

Da qualche anno stiamo assistendo ad un processo di radicale trasformazione del Web, il *Web 2.0*. Grazie agli enormi sforzi e progressi ottenuti nelle tecnologie a supporto del Web, è stato possibile risolvere molti dei pro-

blemi che affliggevano i siti appartenenti alla prima generazione, spostando l'accento sull'interazione, la collaborazione e il *social networking* che ormai fanno parte dell'esperienza quotidiana di tutti noi. Il Web, infatti, è diventato dinamico e fluido, superando le mille barriere dettate dalla staticità delle pagine di informazione caratteristiche del Web 1.0.

Le tecnologie Web 2.0 hanno decisamente cambiato il modo usare e percepire il Web. Da semplice meccanismo di presentazione statica di informazioni, il Web è oggi una vetrina interattiva, in cui è possibile per ognuno esprimere una parte di se stessi, come dimostra l'esplosione del numero di *wikis*, *blogs*, e comunità di cui è popolato. Anche le imprese hanno più volte rivisto la loro presenza in rete, spesso alla ricerca di un rapporto bidirezionale coi propri clienti.

Ovviamente questa rinnovata interazione, complessa e ricca nella sua cangianza, deve essere supportata da interfacce ugualmente ricche e interattive. Non è più sufficiente, nel Web 2.0, fornire lunghe liste di collegamenti per la navigazione per soddisfare i visitatori. La crescente complessità dei siti richiede infatti che le pagine Web possano esprimersi in maniera più sofisticata, attraverso interfacce espressive e intuitive per l'utente. Questo apre naturalmente la strada a due grosse tematiche, centrali nell'ingegneria del Web: l'**usabilità**, ovvero la capacità di una pagina di esprimersi in maniera intuitiva e di facile utilizzo per il visitatore, e l'**accessibilità**, ovvero la possibilità di fruizione dei contenuti da parte di utenti affetti da disabilità.

Questo tipo di problematiche, unite all'enorme successo che ha travolto le applicazioni Web nell'ultima decade, ha reso lo sviluppo di questi contenuti una vera e propria disciplina scientifica. Lo sviluppo di contenuti Web deve infatti essere supportato da una specifica e sempre aggiornata preparazione dal punto di vista scientifico, sia riguardo alle più recenti tecnologie e metodologie di sviluppo, in continua rivoluzione, sia riguardo alle diverse problematiche, sociali ed economiche, che interessano il ciclo di vita di una applicazione Web.

Tutto questo rende quello del *Web engineer* un lavoro complesso e, allo stesso tempo, estremamente ambizioso.

1.2.2 Usabilità ed accessibilità

Quando si parla di usabilità, spesso viene in mente la *facilità di utilizzo* di un certo strumento o software. È vero, spesso e volentieri un sistema complesso risulta non usabile, tuttavia la complessità non è necessariamente indice di una cattiva progettazione: si pensi alla disposizione dei comandi in una cabina di pilotaggio.

Per dare una definizione più precisa al termine usabilità dobbiamo fare ricorso a ciò che sta "alla base" di tutto: il nostro cervello. Infatti, il concetto di usabilità non è una creazione recente, bensì è ciò che potremmo chiamare un sottoprodotto dell'*ergonomia cognitiva*, una scienza che studia l'interazione dell'essere umano con l'ambiente sulla base dei propri limiti fisici e cognitivi.

Dei piccoli miglioramenti — ad esempio usare un sistema di navigazione conciso, chiaro e consistente — possono drasticamente migliorare l'esperienza utente per tutti i visitatori (inclusi quelli che fanno uso di *assistive technology*).

Per riuscire a garantire l'usabilità di un'applicazione è quindi necessario avere una profonda conoscenza dei propri utenti, dei loro bisogni, delle loro abilità, delle loro limitazioni e avere anche un'idea di quali sono le cose che essi ritengono importanti.

1.2.3 Parametri di valutazione dell'accessibilità

Lo standard ISO 9241 definisce l'usabilità come il "grado in cui un prodotto può essere usato da particolari utenti per raggiungere certi obiettivi con efficacia, efficienza, soddisfazione in uno specifico contesto d'uso". Come si può vedere, i parametri delineati sono:

Efficacia

Il sistema deve essere in grado di portare a termine i compiti richiesti da un utente.

Efficienza

Il costo per l'utente — sia in termini di tempo che in termini di fatica, attenzione necessaria, e così via — deve, per quanto possibile, essere minimizzato.

Soddisfazione

L'esperienza dell'utente con il sistema deve risultare piacevole nel suo insieme.

1.2.4 Aspetti legali dell'accessibilità

Le linee guida per l'accessibilità dei contenuti web (Web Content Accessibility Guidelines, WCAG) sono parte di una serie di linee guida sull'accessibilità pubblicate dal World Wide Web Consortium (W3C), la principale organizzazione di standard per internet. Esse consistono di un insieme di linee guida per rendere il contenuto accessibile, in modo particolare nel caso di persone affette da disabilità, ma anche in generale per tutti i possibili user agents, inclusi dispositivi molto limitati come i telefoni cellulari. La versione corrente, WCAG 2.0, è stata pubblicata nel 2008 ed è diventata ufficialmente uno standard ISO nel 2012.

I principi esposti dalle linee guida WCAG 2.0 sono stati diligentemente raggruppati in 4 categorie:

Percepibile

Fornire alternative testuali per qualsiasi contenuto non di testo; fornire alternative per i tipi di media temporizzati; creare contenuti che possano essere rappresentati in modalità differenti (per esempio, con layout più semplici), senza perdite di informazioni o di struttura; rendere semplice per gli utenti la visione e l'ascolto dei contenuti, separando i livelli di primo piano e di sfondo.

Utilizzabile

Rendere disponibili tutte le funzionalità anche tramite tastiera; fornire agli utenti tempo sufficiente per leggere ed utilizzare i contenuti; non sviluppare contenuti che possano causare attacchi epilettici; fornire all'utente funzionalità di supporto per navigare, trovare contenuti e determinare la propria posizione.

Comprensibile

Rendere il testo leggibile e comprensibile; creare pagine Web che appa-

iano e funzionino in modo prevedibile; aiutare gli utenti ad evitare gli errori ed agevolarli nella eventuale correzione.

Robusto

Garantire la massima compatibilità con i programmi utente attuali e futuri, comprese le tecnologie assistive.

Con l'avvento delle applicazioni web, negli ultimi anni si è vista però la necessità di migliorare le tecnologie disponibili a supporto delle persone affette da disabilità. Infatti, una delle colonne portanti di tutte le moderne applicazioni web (si vedano l'interfaccia di Twitter, Facebook, e via discorrendo) è quella di *rinviare* il caricamento dei contenuti fino al momento in cui sono necessari.

Questo tipo di caricamento può essere realizzato ad esempio tramite la tecnologia AJAX, in cui il client manda delle richieste utilizzando JavaScript. Il problema che sorge immediatamente, però, è questo: se a un certo punto una parte del contenuto di una pagina viene modificato, allora questo nuovo contenuto potrebbe risultare non visibile per chi fa uso di tecnologie assistive.

Legge Stanca

La legge Stanca, pubblicata nella Gazzetta Ufficiale il 17 gennaio 2004, prescrive «Disposizioni per favorire l'accesso dei soggetti disabili agli strumenti informatici».

I soggetti destinatari della legge sono gli enti pubblici, tra cui: le pubbliche amministrazioni, le aziende che forniscono servizi pubblici, le aziende di trasporto pubblico.

1.2.5 Progettazione dell'accessibilità

Col tempo sono nate delle tecniche per sviluppare applicazioni accessibili, sia a livello pratico che a livello ingegneristico:

Progettazione basata sull'utente

Come si ha avuto modo di vedere, è molto importante avere ben definito un piano che permetta di progettare applicazioni usabili. Ora verranno passate in rassegna delle tecniche pratiche.

Con progettazione centrata sull'utente (o user-centered design) intendiamo il processo di progettazione di un prodotto o sistema in cui le necessità, le inclinazioni e le limitazioni dell'utente finale vengono prese in attenta considerazione durante tutte le fasi della progettazione.

Lo standard ISO descrive 6 principi chiave che permettono di caratterizzare la progettazione centrata sull'utente:

- Il progetto viene preparato in base ad una conoscenza esplicita degli utenti e dei compiti da svolgere.
- Gli utenti stessi vengono attivamente coinvolti durante la fase di progettazione e di sviluppo.
- La progettazione è guidata e subisce modifiche in base alla valutazione degli utenti.
- Il processo è iterativo.
- La progettazione si occupa dell'intera user experience.
- Il team di design possiede abilità e prospettive multidisciplinari.

ARIA: Accessible Rich Internet Applications

ARIA è una suite nata nell'ambito della Web Accessibility Initiative (WAI), che definisce una tecnica per rendere il contenuto web e in particolare le applicazioni web più accessibili per le persone con disabilità [W3C06].

ARIA affronta le sfide di accessibilità legate alle moderne applicazioni web definendo come le informazioni su una certa funzionalità possono essere fornite al software di assistive technology. Con ARIA, un'applicazione web molto avanzata si può rendere accessible e usabile.

Più precisamente, ARIA espone un framework per l'aggiunta di attributi che identificano delle funzionalità per l'interazione utente, come si relazionano tra di loro, ed il loro stato corrente. ARIA permette di assegnare ad un ipotetico sistema di navigazione innovativo il ruolo di menu, in questo modo gli utenti costretti ad utilizzare sempre la tastiera riescono a muoversi facilmente tra le varie "regioni" dell'applicazione, piuttosto che ritrovarsi a dover premere tante volte il tasto Tab.

ARIA include anche delle tecnologie per legare i controlli, le regioni AJAX e gli eventi di un'applicazione web direttamente alla API di un software di supporto. Inoltre, ARIA fornisce ruoli per definire il tipo di widget, la struttura della pagina, le proprietà dei widget, le proprietà di un'area drag & drop — vitale per chi non può usare il mouse, che in questo modo viene messo in condizione di operare direttamente sulla sorgente (drag) e sulla destinazione (drop).

1.2.6 Tempo di risposta

Dal momento che il nostro livello di attenzione e la nostra memoria sono limitati, è di vitale importanza che le interfacce che progettiamo siano il più possibile veloci e reattive. Inoltre, dobbiamo tenere in considerazione il fatto che l'essere umano tende a voler tenere sotto controllo il sistema che sta utilizzando, non vuole che il computer "detti le regole" costringendolo ad attendere [Nie10].

In uno studio svolto nell'ambito di un lavoro del Nielsen Norman Group, un gruppo di utenti fu intervistato in modo da capire che cosa pensava ognuno di loro di vari siti web che avevano avuto modo di usare: in questo modo, le risposte ottenute erano basate non sull'utilizzo immediato ma su quelle esperienze che erano rimaste impresse al punto da formare dei ricordi. Il risultato fu interessante perché gli utenti facevano generalmente presente la lentezza dei siti visitati. Lo stesso Nielsen Norman Group riporta inoltre che, ogni volta che si svolge un nuovo studio di questo tipo, ci sono sempre utenti che fanno osservazioni sul tempo di risposta; quando un sito web guadagna anche un decimo di secondo, si notano incrementi nel tasso di utenti che decidono poi di usare effettivamente il sito.

La velocità di caricamento di un applicazione web crea un tale impatto sull'utente finale che può addirittura diventare un patrimonio di marca che gli utenti associano con il sito [Nie10].

Limiti di tempo

I limiti di tempo da tenere in considerazione quando si progetta un'applicazione web [Nie93] sono determinati dalle facoltà percettive dell'essere umano e, come tali, sono rimasti gli stessi — accertati con gli anni e validi ancora oggi — che i pionieri del campo di ricerca *Human factors* calcolarono oltre 40 anni fa. In particolare:

0.1 secondi
: La sensazione che l'utente riceve è quella di una risposta istantanea, ovvero, il risultato dell'operazione sembra direttamente causato dall'azione dell'utente, non dal computer.

1 secondo
: Mantiene intatto il flusso di coscienza dell'utente. Sebbene il ritardo sia visibile, c'è ancora la sensazione di tenere sotto controllo il computer.

10 secondi
: Si ha ancora l'attenzione dell'utente, nonostante esso si senta decisamente "sottoposto" al computer, desiderando che fosse più veloce. Superata questa soglia, generalmente l'utente comincia a pensare ad altre cose, rendendo più difficile recuperare l'attenzione una volta che il computer ha concluso l'operazione.

1.2.7 Misure correttive

Se una certa applicazione effettua un download di dimensione significativa, potrebbe essere opportuno suddividere i dati dal lato server e fare in modo che vengano scaricati man mano che l'utente li richiede.

Esistono inoltre delle misure correttive applicabili a priori che, in certi casi, possono ridurre significativamente i tempi di caricamento:

Minificazione

Con minificazione intendiamo il processo di rimozione di tutti i caratteri non necessari da un codice sorgente — generalmente JavaScript o CSS — senza però alterarne le funzionalità. Alcuni programmi di minificazione possono anche accorciare i nomi delle variabili, classi e funzioni in modo da usare meno caratteri. Inoltre, esistono dei tool che eseguono delle vere e proprie ottimizzazioni del codice atte a ridurne la lunghezza, come *Closure Compiler* per il linguaggio JavaScript.

Come facilmente intuibile, il codice sorgente minificato è più adatto per la trasmissione via rete, per questo motivo esistono numerosi tool che svolgono questo compito [Min12]. Per farsi un'idea di quanto la minificazione venga sottovalutata nonostante sia un processo molto semplice, basta considerare questo fatto: al momento della scrittura di questa tesi il sito web *ansa.it* ha un codice CSS del peso di 524946 byte: incollandolo su *csscompressor.com* e premendo il bottone "Compress" viene restituito in output un nuovo codice, del tutto equivalente a quello di partenza, ma la cui dimensione è diminuita del 22.25%.

Gzipping

Ormai dal 1998 si può affermare che praticamente tutti i browser supportano l'elaborazione di contenuto compresso [Sch03]. Assieme alla minificazione, la compressione — generalmente effettuata usando gzip, che si basa sull'algoritmo DEFLATE — è un ottimo modo per risparmiare banda e mandare quindi meno dati in rete, diminuendo così il tempo di risposta.

Ulteriori ottimizzazioni

Esistono ulteriori ottimizzazioni, che però non sono supportate da tutti i browser. Esempi sono il protocollo SPDY (antenato di HTTP/2) ed il formato immagine WebP, entrambi ideati da Google.

Google ha anche avviato un servizio che permette agli utenti di scaricare dati usando i loro server come proxy, sfruttando così HTTP/2 e WebP quando possibile. Come si può vedere in fig. 3, il risultato al quale l'azienda puntava in termini di banda risparmiata è risultato significativo.

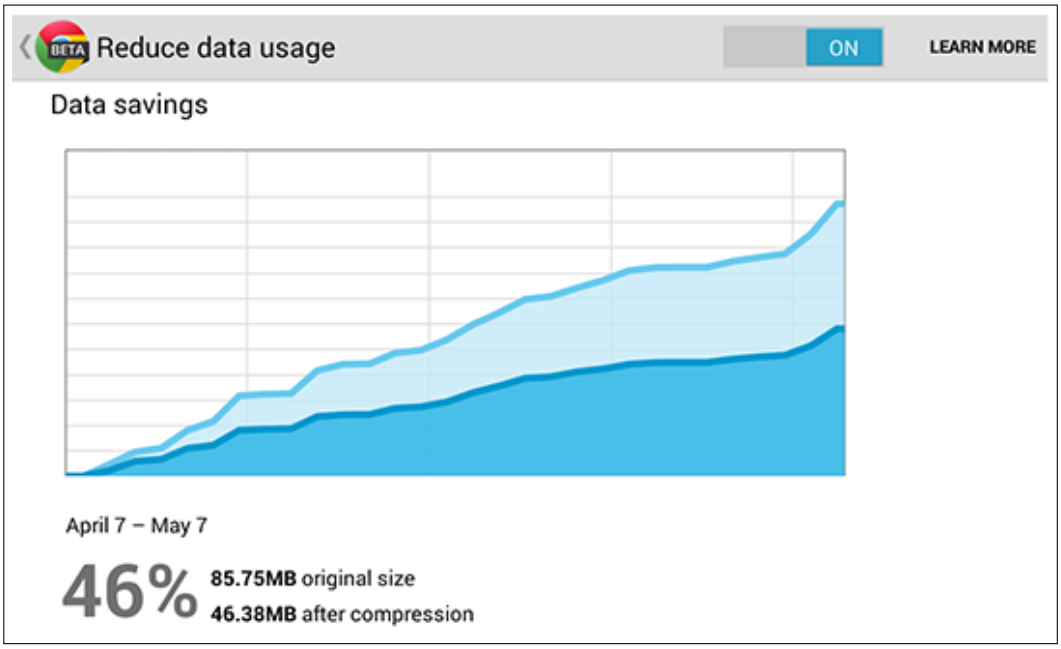


Figura 3: Dati risparmiati in un mese

1.2.8 URL semantici

Gli URL (Uniform Resource Locator) semantici sono degli schemi di URL appositamente studiati in modo da migliorare l'usabilità e l'accessibilità di un'applicazione o servizio web. L'idea alla base è che l'URL di una determinata risorsa offerta dovrebbe essere significativo. L'utente dovrebbe essere in grado di sapere, intuitivamente, che cosa aspettarsi prima ancora di visitare un URL semantico.

Gli URL semantici spesso tendono a riflettere la struttura concettuale delle informazioni memorizzate in un certo sito, e disaccoppiano l'interfaccia utente dalla rappresentazione effettiva delle stesse. Ulteriori ragioni che spingono sempre più web engineer ad utilizzare questa schematica nei propri URL sono:

- L'ottimizzazione nell'ambito dei motori di ricerca (search engine optimization, *SEO*) [Opi06].
- Il volersi conformare allo stile adottato dall'architettura REST (representational state transfer), il cui utilizzo per costruire le API dei servizi

web è in costante crescita.

- Il volersi assicurare che le singole risorse offerte rimangano consistentemente posizionate allo stesso URL. Questo contribuisce a rendere il Web più "stabile" e utile, permettendo inoltre di ridurre il fenomeno dei *dead link*.

Gli URL semantici, inoltre, non contengono dettagli di implementazione dell'applicazione web sottostante [Nym07]. Questo porta diversi benefici, come la sostanziale riduzione delle varie difficoltà legate ad un cambio di tecnologia che per svariati motivi potrebbe rendersi necessario in un secondo momento.

Per esempio, supponiamo che un URL includa il nome di uno script lato server (come `products.php`, `ChangePassword.aspx` o `cgi-bin`). Nel caso in cui l'implementazione di tale risorsa dovesse cambiare (ad esempio da ASP a PHP), l'URL ad essa associato si troverebbe a dover cambiare con essa. Gli effetti di questo cambio "forzato" in certi casi si possono riflettere negativamente sulla propria posizione negli elenchi dei motori di ricerca.

URL non semantico	**URL semantico**
http://example.com/index.php?page=about	http://example.com/about
http://example.com/products?category=c3&pid=25	http://example.com/products/c3/25
http://example.com/cgi-bin/feed.cgi?feed=news&frm=rss	http://example.com/news.rss

Tabella 1: Esempi di URL semantici e non

1.2.9 Responsive design

Con *responsive design* intendiamo un approccio allo sviluppo di applicazioni che ne modifica l'aspetto in base alla grandezza e all'orientamento dello schermo montato sul dispositivo che si sta utilizzando [Sch14].

Il responsive design viene implementato nell'ambito web attraverso le media query del CSS, ad esempio:

Il responsive design ha degli evidenti vantaggi rispetto a sviluppare siti web diversi per i diversi tipi di dispositivo. Infatti, il processo di sviluppo e di manutenzione risultano più economici quando il codice sorgente è uno

```
<style>
@media (max-width: 600px) {
  .sidebar { display: none; }
}
</style>
```

Codice 1: Esempio di media query utile a nascondere `.sidebar` qualora lo schermo abbia una larghezza minore o uguale a 600 pixel.

solo. Inoltre, dal momento che il responsive design si basa sulle dimensioni dello schermo, è facile aggiungere il supporto per i nuovi dispositivi rilasciati nel mercato [Sch14].

Un aspetto negativo è il fatto che, dal momento che tutti i dispositivi ricevono il medesimo HTML, uno smartphone con un piccolo schermo riceverà la stessa quantità di dati che avrebbe ricevuto un PC con uno schermo ad alta risoluzione.

1.2.10 Strumenti per il testing

Esistono degli strumenti automatici (spesso sotto forma di servizi online) per testare l'usabilità e l'accessibilità delle pagine web. Passiamo ora in rassegna alcuni di quelli più interessanti.

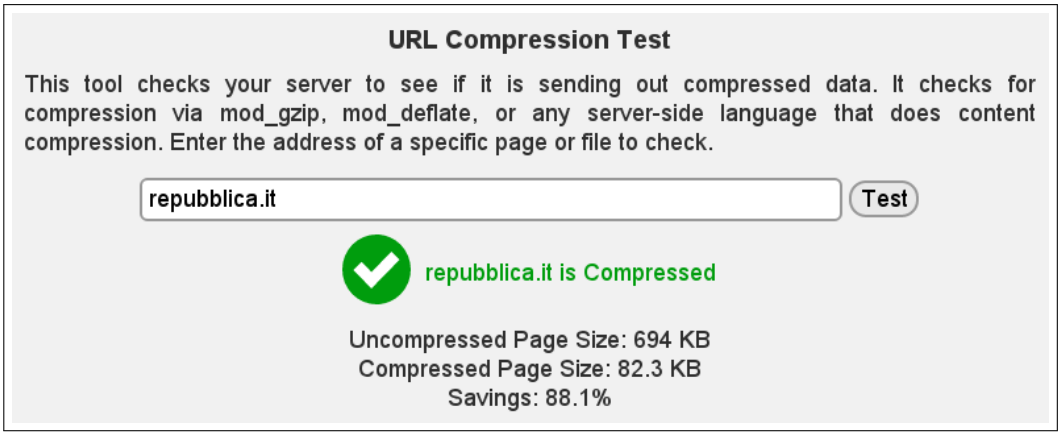


Figura 4: http://www.whatsmyip.org/http-compression-test/

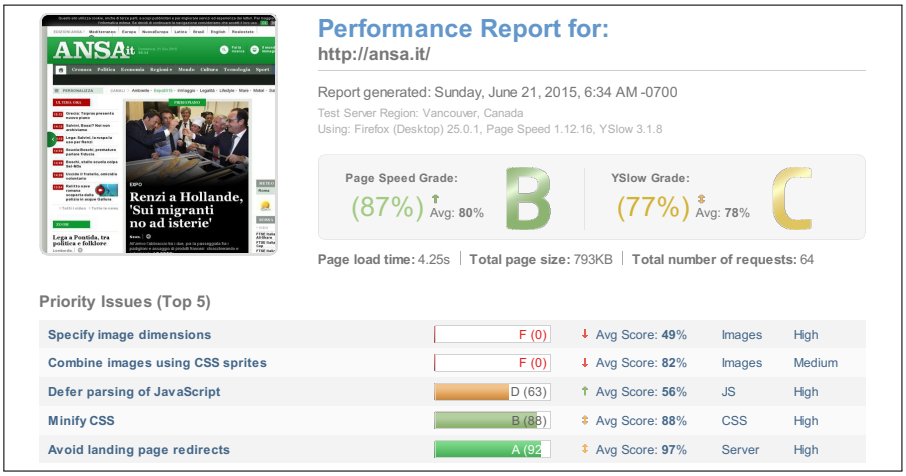


Figura 5: `http://gtmetrix.com/`

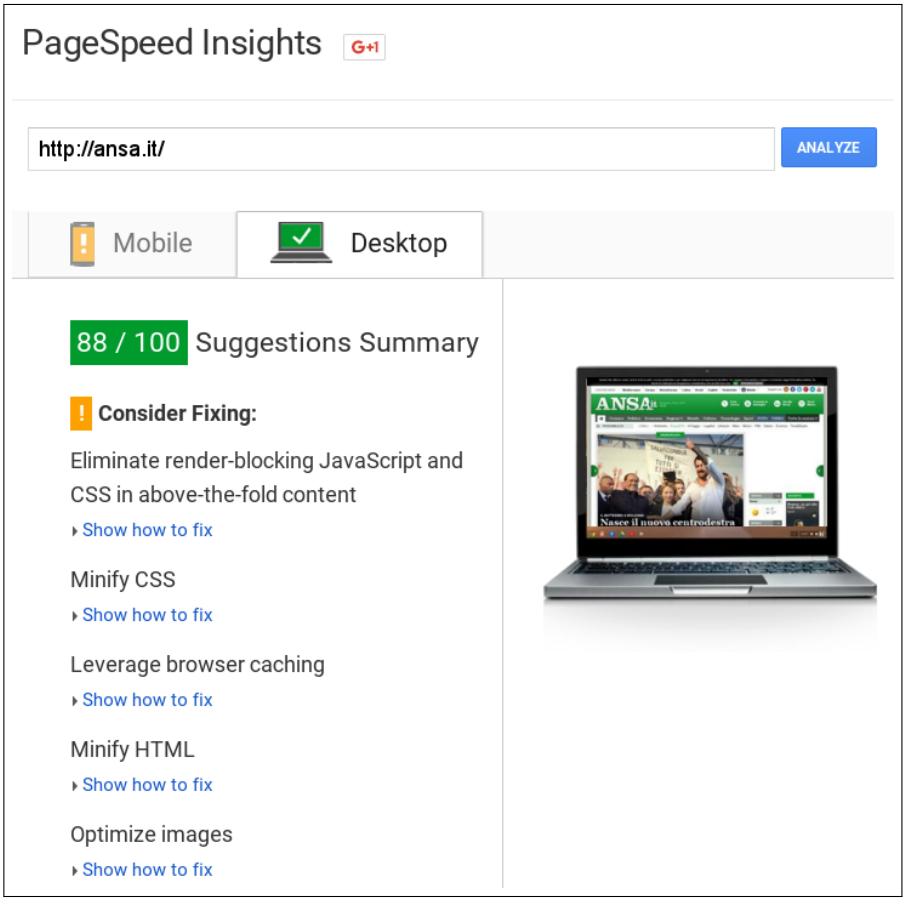


Figura 6: `https://developers.google.com/speed/pagespeed/insights/`

1.3 Caso di studio: visualizzazione dati

Questa sezione tratterà dello sviluppo di applicazioni nel caso specifico della visualizzazione dati. Verranno passate in rassegna le tecniche di visualizzazione e le tecnologie disponibili per visualizzare dati all'interno del browser. Inoltre, verranno descritte le tecnologie di rappresentazione dei dati, con particolare attenzione ai dati cartografici.

1.3.1 Il browser come mezzo di comunicazione

Uno dei tanti possibili utilizzi delle applicazioni web è quello relativo alla visualizzazione dati. Questo tipo di compito spesso richiede che i dati siano costantemente aggiornati e che l'interfaccia di visualizzazione sia particolarmente accessibile. Per questo motivo, il browser si presta bene ad essere lo strumento con il quale eseguire un'applicazione di visualizzazione dati.

Tecniche di visualizzazione

Come indicato da [EH11], le tecniche di visualizzazione dati possono suddividersi in due macrocategorie:

Visual reporting

Il visual reporting fa uso di diagrammi e grafici per visualizzare una descrizione sintetica dei dati, ad esempio il rendimento di un'azienda nel tempo.

Esempi di visual reporting sono le *dashboard* e i *report* utilizzati nell'ambito della Business intelligence [Eck11].

Visual analysis

La visual analysis, d'altro canto, permette agli utenti di esplorare graficamente i dati attraverso dei controlli appositi, rendendo possibile anche la scoperta di nuove chiavi di lettura dei dati. Mentre con il visual reporting la navigazione viene strutturata su delle metriche definite, la visual analysis offre un grado più alto di interattività dei dati.

Con la visual analysis, gli utenti hanno la possibilità di confrontare, filtrare, correlare i dati presentati. In certi casi, vengono anche offerti modelli di previsione, che permettono di proiettare dati futuri [Eck11].

1.3.2 Tecnologie per la visualizzazione dei dati

Nel corso degli anni e in particolare con l'avvento di HTML5, c'è stato un ricambio generazionale nelle tecnologie utilizzate per la visualizzazione dati.

Adobe Flash

Adobe Flash (precedentemente conosciuta con il nome Macromedia Flash) è una piattaforma software multimediale che offre la possibilità di utilizzare grafica raster e vettoriale per produrre applicazioni web, desktop e mobile. È stata utilizzata in modo particolare nei primi anni 2000 per sviluppare le componenti interattive dei siti web.

La tecnologia Flash si presta bene ad essere utilizzata per la visualizzazione dati grazie ai tool di animazione forniti da Adobe e grazie ad ActionScript, il linguaggio utilizzato per programmare l'interazione con l'utente. Come testimoniato da [Gub09], c'è stato un periodo in cui i componenti di visualizzazione dati che facevano uso di questa tecnologia erano molto in voga.

Nonostante l'ottima predisposizione, questa tecnologia presenta dei pesanti svantaggi legati all'accessibilità. Usando Flash, infatti, si rischia di violare le convenzioni delle normali pagine HTML. Ad esempio lo scrolling e la selezione del testo, come indicato in [Nie05], sono spesso fonte di confusione per l'utente quando non funzionano nel modo che ci si aspetta. Il tasto destro del mouse ed i controlli dei form non si comportano allo stesso modo in un componente Flash ed i una normale pagina HTML. L'esperto di usabilità Jakob Nielsen pubblicò in [Nie00] una lista di problemi simili legati alla tecnologia Flash.

Alcuni dei problemi lamentati da [Nie00] sono stati risolti, ma solo parzialmente. Per esempio, dalla versione 6 di Flash Player è possibile includere del testo alternativo, tuttavia questa funzionalità è disponibile solo su Windows e non funziona correttamente con alcuni screen reader.

Alla luce di queste considerazioni, non è sorprendente che al 2015 si stia osservando una sostanziale presa di distanza dalla tecnologia Flash (almeno per quanto riguarda le applicazioni web) anche a causa dell'importanza rivestita dall'accessibilità in questo particolare campo. Per tornare all'articolo citato precedentemente, gli sviluppatori di molti dei componenti elencati in [Gub09] ne hanno ora sospeso lo sviluppo.

Gradualmente, le applicazioni web stanno rimpiazzando le componenti Flash con alternative HTML5. Per esempio [McC15] riporta che nel gennaio 2015 il terzo sito più visitato al mondo, YouTube, ha aggiornato il riproduttore video di default: è ora sviluppato con la tecnologia HTML5.

HTML5

Con l'avvento di HTML5, diventa possibile creare in modo nativo dei componenti che prima erano realizzabili soltanto con Adobe Flash [Htm]. Infatti, tra le varie aggiunte, vediamo il tag `<svg>`, contenitore per grafica vettoriale, ed il tag `<canvas>`, una superficie di disegno ove è possibile controllare i singoli pixel mediante JavaScript.

Tag `<canvas>`

HTML5 non espone modi di rendere accessibile il contenuto di un elemento canvas [Msc]. L'unica cosa che si può fare è utilizzare gli attributi ARIA (trattati nella sezione 1.2.5) per descriverne il ruolo.

```
<!-- La seguente canvas ha ruolo "img" -->
<canvas id="esempio" width="80" height="60" role="img">
  ...
</canvas>
```

Codice 2: Esempio di utilizzo del tag canvas

Questo fatto rende `<canvas>` un tag utile nel caso di applicazioni specifiche come videogiochi o elementi di decorazione, ma non particolarmente indicato per un'istogramma o un grafico a torta.

Tag `<svg>`

SVG prevede delle modalità per indicare alternative equivalenti, utili

per esempio qualora si volesse creare un grafico ed indicarne le componenti in modo accessibile. In particolare, nel documento [W3C00] vengono illustrati gli elementi *title* e *desc*, che sono applicabili ad un qualunque container o elemento grafico di SVG.

```
<svg width="6in" height="4.5in" viewBox="0 0 600 450">
  <title>Network</title>
  <desc>Example of computer network based on a hub</desc>
  ...
</svg>
```

Codice 3: Esempio di utilizzo del tag canvas

L'accessibilità del contenuto e la possibilità di trattare gli elementi di SVG come normali nodi manipolabili mediante JavaScript, fanno sì che il tag `<svg>` risulti adatto per applicazioni di visualizzazione dati.

Strumenti ad alto livello

Esistono diversi strumenti di visualizzazione dati che prendono in ingresso un dataset e una tipologia di diagramma desiderato, restituendo in uscita una rappresentazione grafica.

Google Charts

Charts è un servizio offerto da Google per la visualizzazione di dati attraverso tradizionali diagrammi (torta, barre, e via discorrendo). Charts renderizza i grafici in SVG o VML (deprecato) a seconda delle necessità.

Google Charts

Chart.js è una libreria JavaScript per certi versi simile a Google Charts. La differenza principale è l'utilizzo del tag `<canvas>` per il rendering, piuttosto che `<svg>`.

Strumenti a basso livello

Esistono anche strumenti a basso livello che, invece di permettere di visualizzare direttamente un dataset con il tipo di visualizzazione desiderata, offrono delle primitive per manipolare dati e produrre output visivo.

D3.js

D3, acronimo di *Data-Driven Documents*, è una libreria JavaScript che permette di manipolare documenti in base ai dati che si vogliono rappresentare. È stata sviluppata da Mike Bostock, un informatico e specialista di visualizzazione dati americano.

1.3.3 Tecnologie per la rappresentazione dei dati

Esistono numerosi modi e formati per la rappresentazione dei dati.

CSV

Il CSV, acronimo di *Comma Separated Values* (valori separati da virgola), è un formato aperto utilizzato per memorizzare dati tabellari. Essendo relativamente semplice, risulta facile da consumare in modo programmatico. Nonostante la sua grande diffusione, il formato non ha uno standard ufficiale. Lo standard de-facto è documentato nella RFC 4180 [Sha05].

La struttura di un file CSV, nel caso più semplice, si compone di tante righe quante sono le righe della tabella rappresentata. Di solito la prima riga è riservata per l'header: un record che contiene i nomi dei vari campi separati da virgola:

```
nome,cognome,giorno1,giorno2,totale
mario,rossi,150,200,350
marco,bianchi,100,180,280
franco,verdi,200,200,400
```

Codice 4: Esempio di file in formato CSV contenente 3 record

JSON

Il JSON, acronimo di *JavaScript Object Notation*, è un formato molto semplice da consumare in modo programmatico dato che si compone di chiavi (stringhe) e valori che possono essere: stringhe, numeri, array oppure oggetti.

Questo formato è quindi molto diffuso per lo scambio di informazioni. Molte API, come la Facebook Graph API, restituiscono infatti un output in JSON alle richieste che vengono fatte.

```json
{
  "id": 1,
  "name": "A green door",
  "price": 12.50,
  "tags": ["home", "green"]
}
```

Codice 5: Esempio di codice JSON

RDBMS

Un DBMS, acronimo di *DataBase Management System*, è un sistema che gestisce le interrogazioni e la consistenza/durabilità di un insieme di dati, in modo da tollerare eventuali situazioni di fallimento (come un blackout nel mezzo di un aggiornamento di un record). Inoltre, un DBMS garantisce che le varie interrogazioni e gli aggiornamenti ai dati avvengano in maniera isolata.

Un RDBMS, acronimo di *Relational DBMS*, è un DBMS che fa uso del modello relazionale. I dati sono quindi rappresentati per mezzo di tuple e raggruppati in relazioni. Questo permette di interrogare i dati utilizzando il linguaggio SQL, che permette di specificare in modo dichiarativo quali dati si desiderano, lasciando al RDBMS il compito di capire come ottenere effettivamente la risposta all'interrogazione.

Nel contesto degli open data un RDBMS potrebbe essere utilizzato come mezzo per produrre, ad esempio, un output in formato CSV.

1.3.4 Tecnologie per dati cartografici

I dati cartografici, come per esempio la forma di un'isola, si possono codificare in diversi modi.

GeoJSON

Il formato GeoJSON viene usato per archiviare una collezione di geometrie spaziali i cui attributi sono descritti attraverso JSON. Le geometrie possibili sono *punti* (esempio: l'indirizzo di un'abitazione), *linee spezzate* (percorsi, strade e confini), *poligoni* (paesi, province, laghi), e collezioni multiple di queste tipologie [But08].

```
{
  "type": "Polygon",
  "coordinates": [
    [[30, 10], [40, 40], [20, 40], [10, 20], [30, 10]]
  ]
}
```

Codice 6: Esempio di codice GeoJSON

TopoJSON

TopoJSON è un'estensione di GeoJSON sviluppata da Mike Bostocks, il creatore di D3.js. L'estensione ha l'obiettivo di rendere possibile la codifica delle topologie, oltre alle geometrie. Infatti, invece di rappresentare le varie geometrie spaziali in modo discreto, in TopoJSON esse vengono "incollate" assieme usando segmenti di linee condivisi chiamati *archi* [Bos12].

In TopoJSON quindi si elimina la ridondanza delle informazioni, portando anche a riduzioni dell'80% nella dimensione del file. Inoltre, TopoJSON facilita le applicazioni che usano la topologia (shape simplification, map coloring, cartogrammi) [Bos12].

OpenStreetMap

OpenStreetMap (OSM) è un progetto collaborativo con un'ideale per certi versi simile a quello di Wikipedia. L'obiettivo di OSM è infatti quello di creare una mappa aperta e modificabile del mondo. OSM distribuisce i dati geografici nel formato `.osm`, che è sostanzialmente un file XML contenente le coordinate (latitudine, longitudine) espresse nel sistema di coordinate WGS-84 [Wgs].

Un file `.osm` si può importare in PostgreSQL, in modo da essere elaborato con istruzioni SQL, utilizzando un'apposita estensione di nome PostGIS [Pos].

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
  <bounds minlat="54.08895" minlon="12.24875" .../>
  <node id="298884269" lat="54.09017" lon="12.24826" .../>
  <node id="261728686" lat="54.09063" lon="12.24419" .../>
  <node id="183188121" version="1" changeset="12370172" ...>
    <tag k="name" v="Neu Broderstorf"/>
    <tag k="traffic_sign" v="city_limit"/>
  </node>
  ...
</osm>
```

Codice 7: Esempio di un file `.osm`

Google Maps / Bing Maps

Per quanto riguarda la visualizzazione dei dati cartografici, Google Maps e Bing Maps mettono a disposizione delle API per integrare e manipolare delle mappe all'interno di una pagina web [Gma] [Bma].

Questo tipo di tecnologia permette anche di integrare dati GeoJSON, ma non TopoJSON. Essendo una tecnologia più ad alto livello quindi permette una minore flessibilità. Ad esempio, se tra i casi d'uso ci fosse anche quello di

unire manualmente specifiche aree, allora TopoJSON potrebbe tornare utile grazie alle funzioni `topojson.merge`, `topojson.mergeArcs`, `topojson.mesh`, `topojson.meshArcs`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Map</title>
    <meta name="viewport" content="initial-scale=1.0">
    <meta charset="utf-8">
    <style>
    html, body { height: 100%; margin: 0; padding: 0; }
    #map { height: 100%; }
    </style>
  </head>
  <body>
    <div id="map"></div>
    <script>
    var map;
    function initMap() {
      map = new google.maps.Map(document.querySelector('#map'), {
        center: {lat: -34.397, lng: 150.644},
        zoom: 8
      });
    }
    </script>
    <script src="https://maps.googleapis.com/maps/api/js?key=......">
    </script>
  </body>
  </html>
```

Codice 8: Esempio minimalistico sull'uso delle API di Google Maps

2 Progetto *spesapubblica*

In questo capitolo introdurremo il progetto spesapubblica, un'iniziativa nata per facilitare l'accesso ai dati sulla spesa delle pubbliche amministrazioni italiane. Analizzeremo poi il già citato portale soldipubblici.gov.it, sia perché mantiene la banca dati su cui ci baseremo, e sia perché vogliamo capire quali casi d'uso (non già coperti dal portale) ci interessa implementare.

Nella fase di progettazione studieremo in dettaglio sia l'aspetto dell'interfaccia web che la struttura dell'applicazione. Tratteremo inoltre l'hosting dell'applicazione, e la metodologia di aggiornamento dei dati.

2.1 Iniziativa *soldipubblici*

Come abbiamo accennato nella sezione 1.1.3, l'Agenzia per l'Italia Digitale mette a disposizione un servizio online gratuito per facilitare la consultazione dei dati di spesa prodotti dalla pubblica amministrazione.

In questo portale si possono trovare infatti degli archivi contenenti i dati grezzi dei pagamenti, oltre che un'applicazione web che ne permette la consultazione diretta. Vedremo ora le funzionalità dell'interfaccia web offerta da soldipubblici, in modo da comprendere quali sono i casi d'uso mancanti che verranno implementati da *spesapubblica*.

2.1.1 Interfaccia esistente

Come si può vedere in fig. 7 l'interfaccia web di soldipubblici offre la possibilità di selezionare il *chi* ed il *cosa*, rispettivamente indicativi dell'amministrazione ed il tipo di spesa a cui siamo interessati. Una volta forniti questi dati,

Figura 7: Come si presenta l'interfaccia web di soldipubblici.

il sito è poi in grado di restituire la risposta alla domanda "*Quanto* spende *chi* per *cosa*?".

2.1.2 Da dove provengono i dati

La banca dati utilizzata da soldipubblici è fondata sui dati ufficiali. Tutti i dati sono accessibili e scaricabili dalla pagina `http://soldipubblici.gov.it/it/developers`, dove è anche specificato il formato in cui sono salvati. I dati raccolti da soldipubblici sono composti da incassi e pagamenti effettuati dai tesorieri degli enti della pubblica amministrazione italiana.

Questi dati provengono dal SIOPE (sistema informativo delle operazioni degli enti pubblici), il quale si occupa della loro rilevazione telematica. La gestione del SIOPE è stata affidata alla Banca d'Italia con la convenzione del 31 marzo del 2003 [Ban].

2.1.3 Bollettino dei casi speciali

Una peculiarità di questa specifica banca dati è la propensione alle modifiche presentata dai dati: ogni anno può capitare che alcuni comuni si uniscano ed altri si dividano. Ad esempio, il comune di *Fabbriche di Vallico* si è fuso con *Vergemoli* il 1 gennaio 2014, dando luogo al nuovo comune di *Fabbriche di Vergemoli*. Allo stesso modo è anche possibile che dei nuovi comuni vengano aggiunti a causa di suddivisioni: senza dati per gli anni precedenti, può sembrare quindi che non abbiano mai speso nulla.

Data la complessità della situazione, soldipubblici fornisce un "bollettino dei casi particolari" che elenca tutte le discrepanze e disallineamenti a cui i dati potrebbero essere sottoposti [Ban].

2.2 Iniziativa *spesapubblica*

Il progetto spesapubblica, sviluppato da una collaborazione tra l'autore di questa tesi ed uno studente del Politecnico di Milano, nasce come complemento di soldipubblici. L'obiettivo è quello di offrire all'utente alcune funzionalità di esplorazione dei dati all'interno di un'applicazione web.

2.2.1 Il servizio offerto

L'obiettivo del presente lavoro di tesi è quello di progettare e realizzare un'applicazione web che risulti:

Usabile, accessibile
: Verranno messe in pratica, per quanto possibile, le nozioni introdotte nel primo capitolo al fine di realizzare un prodotto che risulti usabile ed accessibile.

Autosufficiente
: L'applicazione sarà facilmente rilocabile, vale a dire: il server che ospiterà l'applicazione ricevendo le connessioni dagli utenti non avrà bisogno di avere particolari programmi installati a parte un qualsiasi web server che sia in grado di comunicare tramite il protocollo HTTP.

Sociale

Verrà fornita la possibilità di condividere sui social network eventuali "scoperte" o interessanti chiavi di lettura che l'utente troverà nel corso dell'esplorazione dei dati.

Open source

Rimanendo fermamente nel contesto dell'open science, il codice sorgente dell'applicazione realizzata e la banca dati utilizzata saranno resi disponibili online con una licenza che permetta la redistribuzione e la creazione di opere derivate.

2.2.2 La banca dati

La banca dati utilizzata da spesapubblica è un derivato di quella offerta dai creatori di soldipubblici. In particolare il voler realizzare un'applicazione autosufficiente ci impedisce, ad esempio, di collegarsi ad un database. Per questo motivo sarà necessario trovare il modo di elaborare i dati grezzi che verranno scaricati dal browser dell'utente direttamente dall'applicazione.

Questo porta l'applicazione ad aver bisogno di una fase di preparazione in cui i dati vengono trattati con un modello relazionale per poi essere trasformati in una serie di file JSON pronti per l'utilizzo. Nel capitolo 3 tratteremo in dettaglio la preparazione della banca dati.

2.3 Casi d'uso

I casi d'uso da coprire sono stati scelti in modo da permetterci di fare uso delle nozioni e delle tecnologie esposte nel corso dei capitoli precedenti:

2.3.1 Aggregazione di spese comuni

Le categorie di spesa selezionabili dalla pagina principale dell'applicazione soldipubblici sono definite dai *codici gestionali*. In particolare, quando viene cercata una parola chiave (ad esempio: "immondizia"), il sito cerca di associarla a dei codici gestionali per poi chiedere all'utente qual è il codice gestionale del quale desidera esplorare i dati.

Questo tipo di approccio mette l'utente finale a contatto diretto con i codici gestionali. Questi codici costituiscono una classificazione economica delle uscite e delle entrate; quando sono stati decisi, particolare attenzione è stata prestata alle esigenze del sistema europeo dei conti (SET) al fine di fornire informazioni all'ISTAT [Cod]. Come risultato, la stessa macroarea di spesa potrebbe includere diversi codici gestionali.

Esempio: le "spese informatiche"

Supponiamo che un cittadino sia interessato alle spese informatiche del suo comune. Provando a cercare un comune scrivendo "informatica" come categoria di spesa, l'applicazione web informa che la parola chiave cercata può riferirsi a diversi codici gestionali, tra i quali:

- 1203 – Materiale informatico
- 1329 – Assistenza informatica e manutenzione software
- 1404 – Licenze software
- 2507 – Acquisizione o realizzazione software

Alcuni codici gestionali che non sono stati suggeriti sono comunque sufficientemente inerenti all'informatica da far sì che un utente interessato alla macroarea delle spese informatiche possa volerli comunque includere nell'esplorazione, in particolare:

- 1204 – Materiale e strumenti tecnico-specialistici
- 2506 – Hardware

Una possibile soluzione

In un caso come questo, è bene evitare che l'utente si trovi a dover scegliere tra troppe categorie di spesa [Nie09]. Un web engineer potrebbe quindi decidere di rimappare i codici gestionali in una nuova codifica semplificata, pensata per l'utilizzo da parte di un utente medio. Inoltre, è saggio mostrare tutte le categorie di spesa selezionabili, in modo che l'utente possa vederle elencate e

possa quindi decidere a quale è più interessato, piuttosto che essere costretto a pensare a delle categorie interessanti [Kru00].

Per esempio, si potrebbe mostrare una lista di macroaree di spesa tra le quali è possibile selezionarne una denominata *Software, hardware e strumenti tecnici*, la quale include automaticamente nella ricerca tutti i codici gestionali appena citati.

2.3.2 Confronto tra enti geograficamente vicini

L'applicazione soldipubblici non mette ancora a disposizione una funzionalità di confronto della quantità di denaro spesa. Esattamente come pubblicizzato, il sito permette di sapere quanto spende chi e per cosa. Non permette però di conoscere facilmente, ad esempio, quali sono le amministrazioni che spendono di più in una data area.

La funzionalità che più si avvicina a un confronto si realizza cercando, una alla volta, le amministrazioni che si vogliono confrontare (specificando ogni volta la categoria di spesa che ci interessa) e aggiungendole alla selezione. Cliccando su una delle voci presenti nella selezione si può aprire la visualizzazione denominata "Timeline" ed è possibile raggiungere, con un altro click, quella denominata "Sintesi".

La visualizzazione Timeline è utile qualora un utente voglia curiosare nello storico della spesa della propria amministrazione. Tuttavia, sia Timeline che Sintesi fanno riferimento solo alla voce cliccata, e non fanno quindi un confronto diretto con le altre. È necessario perciò aprire la visualizzazione Sintesi per ciascuna voce e fare poi manualmente il confronto.

Esempio: confronto tra Roseto e Pineto

Supponiamo che un cittadino rosetano voglia confrontare la spesa relativa all'immondizia riferita al proprio comune, *Roseto degli Abruzzi* e il comune confinante, *Pineto*, con l'obiettivo di scoprire quale dei due spende di più. C'è un solo codice gestionale che si riferisce all'immondizia: *1303 – Contratti di servizio per smaltimento rifiuti.*

L'intenzione del cittadino è specificatamente quella di fare un confronto tra i due comuni per poi condividere con i suoi amici pinetesi i risultati ottenuti. La selezione fatta dal cittadino sarà quindi quella visibile in fig. 8.

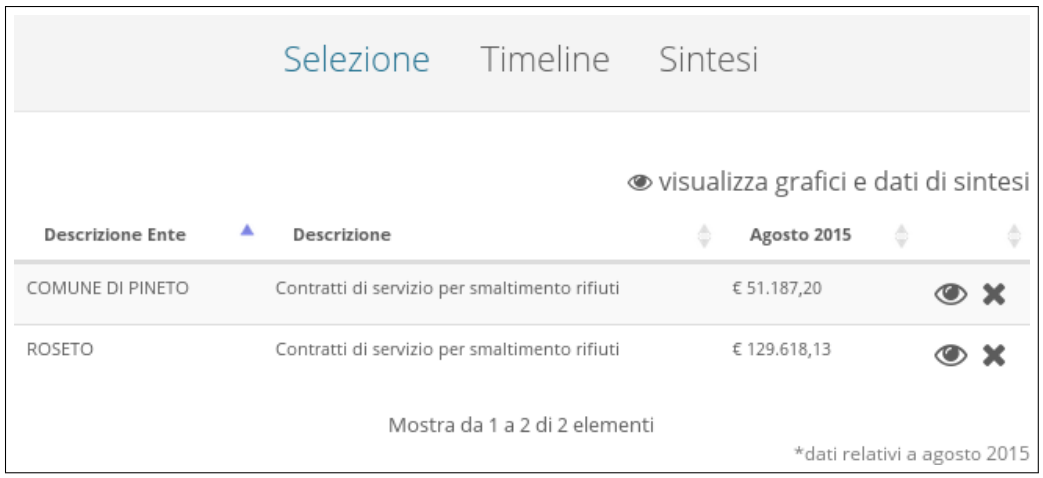


Figura 8: Come si presenta la selezione di due comuni

Al cittadino viene presentata, per ciascuna voce selezionata, la spesa relativa al mese di agosto 2015. Per confrontare la spesa complessiva questo dato non dice molto (se non che probabilmente Pineto spende meno).

Come si può vedere in fig. 9, cliccando sull'icona dell'occhio nella prima riga viene presentata la visualizzazione Timeline della spesa di Pineto.

Le informazioni ricavabili da questa visualizzazione sono molto dettagliate: è possibile inferire che, negli ultimi due anni, il mese di luglio è stato quello in cui si è spesa la maggior quantità di denaro per lo smaltimento dei rifiuti. Questa è un'utile funzionalità, ma non ci permette ancora di effettuare un confronto. Passando da Timeline alla visualizzazione Sintesi otteniamo il risultato visibile in fig. 10.

La visualizzazione Sintesi permette di confrontare la spesa dell'amministrazione selezionata con il totale della spesa di *tutte* le amministrazioni dello stesso tipo. Per confrontare la spesa tra le singole amministrazioni, però, è necessario aprire una per volta le rispettive visualizzazioni Sintesi.

Nell'esempio, il cittadino ha scelto di considerare le spese relative all'immondizia. La parola chiave "immondizia" come abbiamo visto fa riferimento ad un solo codice gestionale, per cui il cittadino deve fare una sola ricerca per

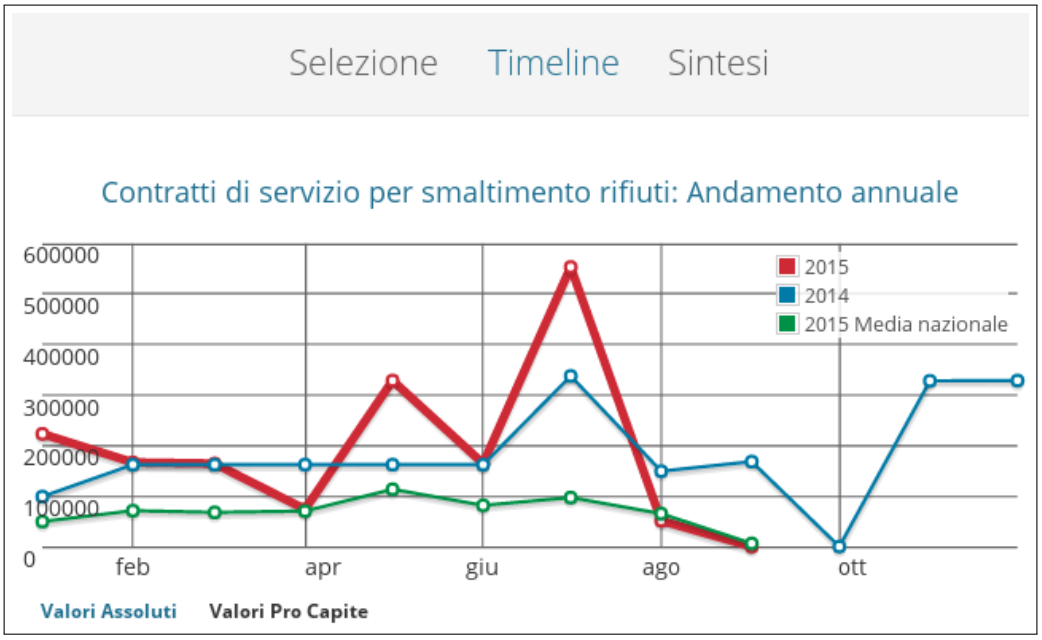


Figura 9: Visualizzazione Timeline della spesa di Pineto

ciascuna voce selezionata. Tuttavia, come abbiamo visto, se avesse cercato "informatica" avrebbe dovuto fare almeno 4 ricerche per ciascuna voce. È importante evitare che l'utente debba sottoporsi ad una tale mole di lavoro [Whi13].

Una possibile soluzione

Un modo per permettere all'utente di eseguire facilmente dei confronti è quello di dargli la possibilità di selezionare un'area geografica (ad esempio: la regione Abruzzo, oppure la provincia di Teramo) ed una certa granularità (ad esempio: visualizzare le province, oppure visualizzare i comuni) ed elencare poi tutte le amministrazioni che ricadono nei filtri impostati. Per ognuna di queste, si può mostrare un indicatore della spesa complessiva.

2.3.3 Condivisione sui social media

Tornando all'esempio di prima, supponiamo che il cittadino voglia condividere con i suoi amici un'osservazione su una sua particolare ricerca. Non vedendo alcun pulsante per condividere la ricerca sui social media o via email,

Figura 10: Visualizzazione Sintesi della spesa di Pineto

la cosa naturale che proverà a fare sarà copiare l'URL della pagina. Non sempre però l'URL di un'applicazione web contiene sufficienti informazioni a ricostruire la pagina che si sta visitando. Questo è vero in modo particolare nelle single page application, dove il click su una risorsa non porta a visitare una nuova pagina [AS10].

Nel caso di soldipubblici, ad esempio, l'URL rimane lo stesso durante tutta l'interazione con l'utente. Il cittadino quindi dovrà scegliere tra fare uno screenshot della pagina web e condividere quello, oppure condividere l'URL dell'applicazione web con delle istruzioni su quali parametri inserire.

Una possibile soluzione

Una possibile soluzione è quella di aggiungere un pulsante di condivisione sui social network che, quando cliccato, salva lo stato della ricerca fatta dall'utente e crea un codice identificativo di questo stato. A creazione completata, l'applicazione mostra all'utente un URL da condividere che somiglierà a: `http://example.com/163527`, dove `163527` è il codice identificativo della ricerca.

Tuttavia, questa soluzione presenta almeno due svantaggi:

1. Per salvare il codice identificativo è necessario avere a disposizione dello spazio di archiviazione (come un database) al quale collegarsi. Peraltro, se qualcun altro ospitasse una copia dell'applicazione, non sarebbe garantito che l'esistenza del codice `12345` nel primo sito implichi l'esistenza di un codice `12345` anche nel secondo sito.

2. Lo schema degli URL non è semantico. Per esempio non c'è modo di sapere quale tra `http://example.com/123` e `http://example.com/456` è una ricerca che fa riferimento alla Provincia di Roma e quale invece fa riferimento alla Regione Toscana.

Una soluzione semantica

Si potrebbe definire uno schema di URL che identifica in modo univoco una ricerca. In particolare, sarebbe necessario prestare attenzione ad includere:

- L'area geografica selezionata.
- La granularità (per regione, per comune, per provincia).
- La categoria di spesa oggetto di interesse.

Facendo attenzione a mantenere un schema semantico per gli URL, ed evitando per quanto possibile di inquinarli con dati ridondanti. Ad esempio, se l'area geografica selezionata è una provincia, è superfluo specificare che la granularità è per comune. Anche nei casi in cui il dato non è determinabile, sarebbe comunque saggio fornire dei valori di default accettabili (ad esempio, per un'area geografica che fa riferimento ad una regione, si può impostare di default la granularità per provincia).

2.3.4 Gestione della banca dati

Oltre alle problematiche legate all'usabilità del sito da parte dell'utente, ne esistono altre legate allo sviluppo e alla manutenzione dell'applicazione stessa da parte dell'amministratore. Ad esempio quelle legate al procedimento di aggiornamento dei dati qualora vengano resi disponibili nella banca dati di soldipubblici.

Aggiornamento dei dati

Come osservato precedentemente, la banca dati di soldipubblici è incline a subire modifiche:

1. Nel corso dei riassestamenti dei comuni, che si uniscono o si dividono.
2. Quando sono disponibili nuovi dati.
3. Quando sono disponibili nuovi tipi di dati (ad esempio se vengono aggiunti nuovi codici gestionali).

È importante quindi avere chiaro in mente un piano che permetta l'aggiornamento dell'applicazione web e dei dati da essa usati in un modo consistente. Un passo in questa direzione è mantenere gli script dell'applicazione e i dati nello stesso repository, aggiornandone di volta in volta il contenuto utilizzando uno strumento software per il controllo versione.

Generazione della banca dati

Dal momento che la banca di dati utilizzata da spesapubblica è un derivato di quella fornita da soldipubblici, non è sufficiente mantenere quest'ultima nel repository. Bisogna definire anche un sistema che, a partire dalla banca di dati di soldipubblici, produca quella usata da spesapubblica.

Per esempio, si potrebbe mantenere una serie di script nel repository adibiti a questa mansione. Sarebbe quindi sufficiente, ogni qualvolta si aggiornano i dati, lanciare gli script di creazione della banca dati.

Esistono anche sistemi appositi denominati *task runner*, che permettono di definire i task ("generazione banca dati", "minificazione CSS", "serving

dell'applicazione web", e così via) e le dipendenze tra di essi ("serving dell'applicazione web" dipende da "minificazione CSS" e da "generazione banca dati", e così via). Tratteremo in dettaglio questo argomento nel capitolo 3.

2.3.5 Ulteriori casi d'uso

Altri casi d'uso che implementeremo sono:

Confronto tra le spese
: Per ciascuna area sarà possibile vedere quali sono le spese più significative. Un modo di implementare questo caso d'uso è utilizzando un grafico a torta.

Confronto temporale
: Per ciascuna area sarà possibile scegliere l'anno solare (ad esempio: 2014) per il quale considerare le spese.

2.4 Specifica dei requisiti e progettazione

La progettazione dell'applicazione web terrà conto di una serie di requisiti tecnici e di usabilità da rispettare. Inoltre il prodotto finito seguirà le nozioni descritte nei capitoli precedenti.

2.4.1 Requisiti

L'applicazione spesapubblica risponderà ai seguenti requisiti:

- Non richiederà elaborazione dal lato server, né di memorizzare dati permanenti sul server.
- Fornirà una mappa interattiva attraverso l'uso della grafica vettoriale in HTML5.
- Darà la possibilità di selezionare, tramite l'utilizzo della mappa interattiva, un'area geografica tra: l'intera penisola italiana, una specifica regione, una specifica provincia oppure uno specifico comune.

- Come alternativa all'uso della mappa, darà la possibilità di cercare l'amministrazione desiderata mediante un'apposita barra di ricerca.
- Darà la possibilità di scegliere il livello di granularità: visualizzazione per regione, per provincia o per comune.
- Darà un feedback visivo all'interno della mappa per indicare l'entità della spesa effettuata dalle amministrazioni che ricadono nell'area selezionata, in modo da confrontarle visivamente. Verrà dato comunque modo di conoscere anche il valore effettivo della spesa.
- Darà la possibilità di considerare la spesa come *totale* o *pro capite*.
- Fornirà una lista di categorie di spesa (meno di 20) selezionabili. Quando l'utente seleziona una categoria, la mappa interattiva si aggiorna riflettendo i dati della spesa relativi alla nuova categoria.
- Darà la possibilità di attivare la selezione di categorie multiple. Quando l'utente seleziona più categorie, la mappa rifletterà i dati relativi alla somma della spesa delle varie categorie.

2.4.2 Progettazione dell'interfaccia web

Con l'obiettivo di offrire un look sobrio e moderno, l'interfaccia web che verrà realizzata seguirà le linee guida definite dal Material Design [Md]. Le funzionalità accessibili all'utente verranno disposte in una barra nella parte superiore della pagina e, qualora lo schermo utilizzato risulti sufficientemente ampio, in una barra laterale posizionata sulla destra.

Valori di default

All'arrivo dell'utente nella pagina principale, le impostazioni dell'applicazione saranno automaticamente regolate a:

- Focus sulla mappa dell'Italia
- Granularità dei dati: regionali

- Modalità spesa: pro capite
- Categoria di spesa da esplorare: “Cancelleria e materiale di ufficio”

Numero di click necessari

Per facilitare l’utente nel portare a termine i casi d’uso descritti nella sezione 2.3, l’interfaccia permetterà di accedere velocemente alle funzioni usate più di frequente:

- Per spostarsi in una regione è sufficiente un click sulla mappa oppure una ricerca.
- Per spostarsi in una provincia sono sufficienti due click sulla mappa (prima nella regione che la contiene, e poi nella provincia) oppure una ricerca.
- Per spostarsi in un comune sono sufficienti tre click sulla mappa (regione, provincia e poi comune) oppure una ricerca.
- Per “risalire” di un livello (da un comune alla provincia che lo contiene, o da una provincia alla regione che la contiene) è sufficiente un click su di un pulsante apposito, indicato con una freccia che punta verso l’alto.

Quantità di informazioni

Per evitare di confondere l’utente [], verranno visualizzate solo informazioni riassuntive. Infatti, a primo impatto, l’utente troverà:

1. Una codifica a colori per confrontare l’entità della spesa per gli enti compresi nel focus.
2. Un grafico a torta che permette di capire quali sono, in proporzione, i tipi di spesa che contribuiscono di più al totale.

Per quanto riguarda la codifica a colori, verranno scelti due colori A e B. Le aree che spendono meno saranno colorate con un colore più vicino ad A, mentre le aree che spendono di più avranno un colore più vicino a B.

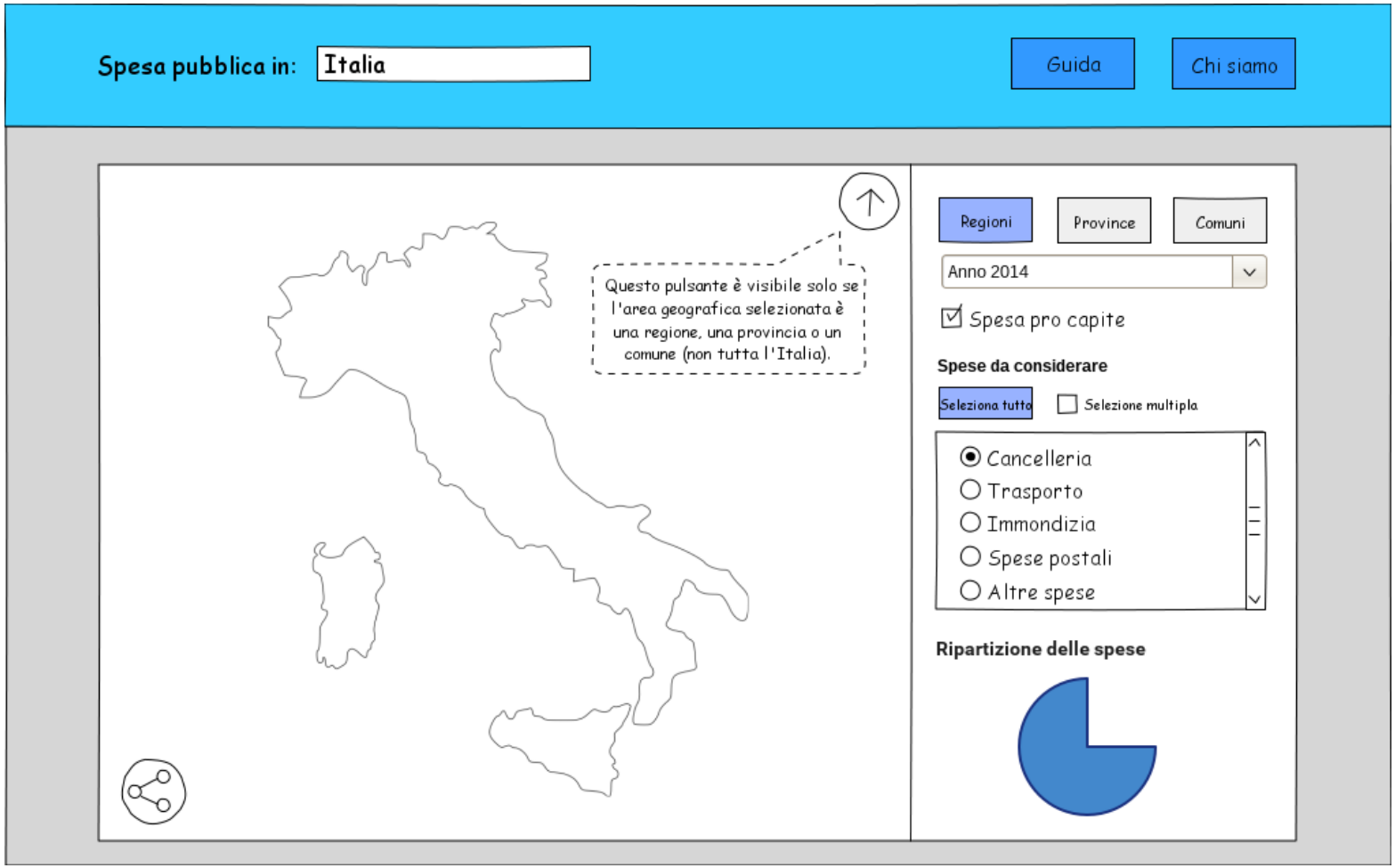
Per ottenere più informazioni dalla mappa l'utente può muovere il mouse su un'area e l'applicazione rivelerà un tooltip con il nome dell'area e il valore della spesa. L'elemento SVG conterrà i dati anche in una maniera accessibile.

Possibilità di annullare le operazioni

Quando l'utente avvia un'operazione costosa, come può essere la visualizzazione di tutti i comuni italiani (cambiando la granularità da "regioni" a "comuni" senza cambiare l'area selezionata), è importante che sia possibile annullare l'operazione senza doverne attendere il completamento.

L'annullamento delle operazioni si può realizzare senza dover aggiungere nuovi controlli: basta ad esempio cambiare nuovamente la granularità e l'operazione precedente verrà interrotta quanto prima.

Mockup dell'interfaccia grafica



2.4.3 Progettazione dell'applicazione

L'applicazione web che gestirà e visualizzerà i dati verrà implementata interamente in linguaggio JavaScript. L'architettura che utilizzeremo sarà modulare e seguirà il pattern *Model-view-controller* (MVC). Per fare questo, utilizzeremo il framework AngularJS. L'applicazione sarà formata dai seguenti componenti:

◇ `info`

Questo componente si occupa di mantenere lo stato dell'applicazione per la durata della visita dell'utente. Se ad esempio l'utente decidesse di modificare la granularità, l'applicazione aggiornerebbe il valore `granularity` contenuto in questo componente.

◇ `dataloader`

Questo componente si occupa di interfacciare il resto dell'applicazione con la banca dati. Se un altro componente avesse bisogno, ad esempio, di ottenere il dizionario delle corrispondenze tra i comuni e le rispettive province che li contengono, lo chiederebbe a questo componente.

◇ `mapchart`

Questo componente si occupa di creare la mappa SVG, assicurandosi di visualizzarne correttamente i confini in base alla granularità scelta, e facendosi carico di colorarla in base ai valori di spesa.

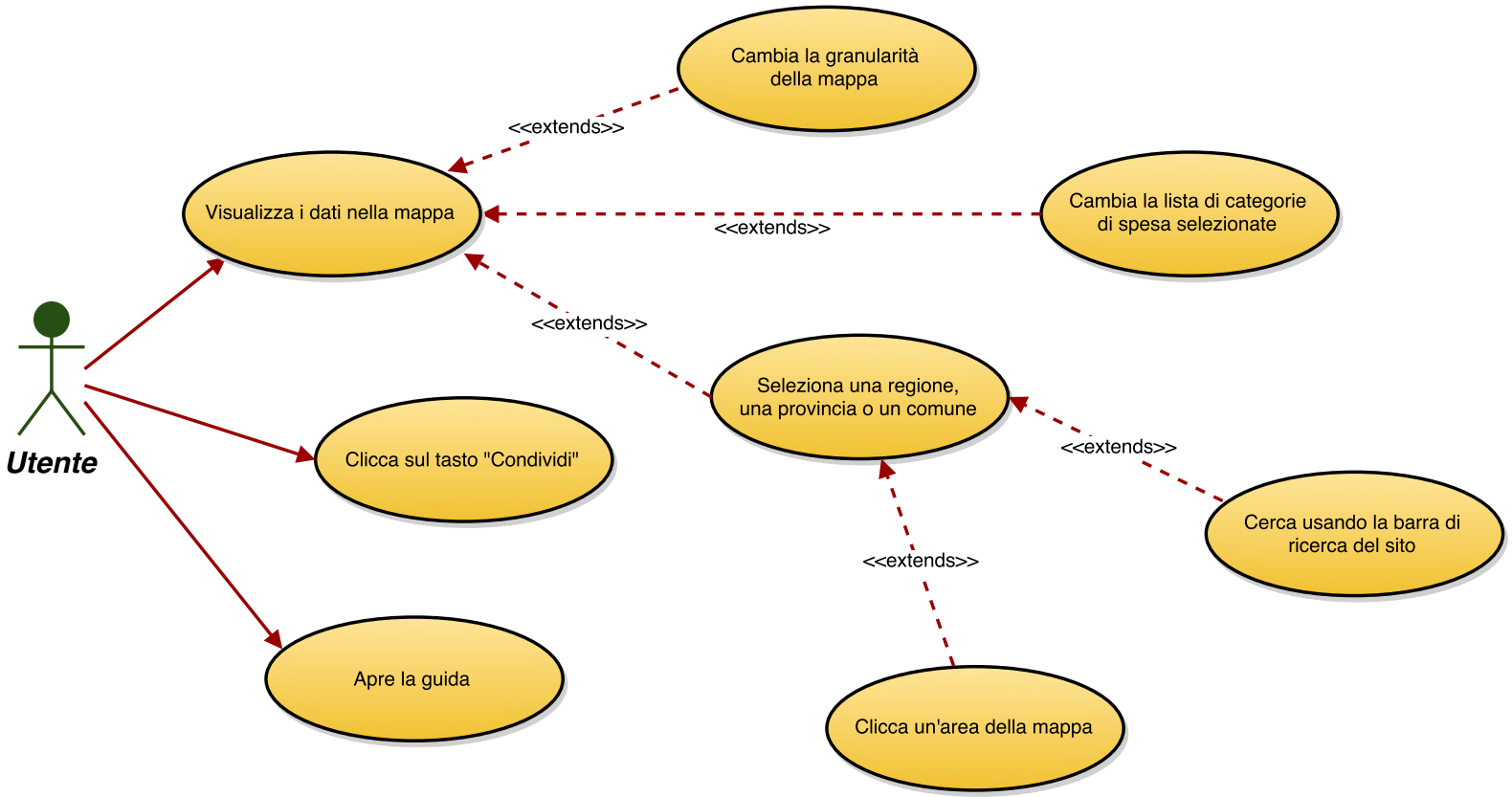
◇ `dashboard_ctrl`

Questo componente è il controller che gestisce l'interazione utente con l'interfaccia web. Qui viene definita, tra le altre cose, la logica dell'applicazione al click sui bottoni dell'interfaccia.

◇ `graph_ctrl`

Questo componente decide che cosa succede in base all'URL che è stato visitato. Per esempio, gestisce l'impostazione della granularità di default in base all'area che si vuole esplorare. Inoltre, gestisce i casi limite in cui vengono specificati dati inconsistenti.

Diagramma dei casi d'uso



2.4.4 Hosting dell'applicazione

Il servizio di Git-repository hosting *GitHub* è uno dei tanti che offrono la possibilità di ospitare gratuitamente un repository, con tanto di issue tracker e pagine wiki. Una delle differenze che lo rendono indicato per questo tipo di applicazione, però, è la possibilità di ospitare siti statici tramite il servizio gratuito denominato *GitHub Pages*.

Github Pages permette di esporre uno specifico branch di un repository su un indirizzo web pubblico [Ghp].

Dal momento che la nostra applicazione non deve fare elaborazione lato server, ospitarla su GitHub Pages risulta quindi la scelta ideale in termini di:

◇ **Costi legati al traffico**

Indipendentemente da quanti utenti visiteranno la nostra applicazione, GitHub sarà in grado di gestire la mole di utenti. A testimonia di questo fatto si può citare un recente evento in cui GitHub ha resistito [New15] ad un attacco DDOS (*distributed denial of service*) ai danni dei suoi server che, a giudicare dalle prove disponibili, si ritiene sia stato eseguito dal governo cinese con l'obiettivo di censurare alcune informazioni [Gra15].

◇ **Costi legati alla gestione**

Per interagire con il server dove è situato il repository è sufficiente avere installato un client Git ed eseguire i comandi:

- `pull` – per ottenere la copia più aggiornata dal server.
- `commit` – per registrare le modifiche locali nel repository.
- `push` – per spedire al server remoto le modifiche registrate col comando `commit`.

Questo significa che se uno dei developer volesse ad esempio modificare il colore di sfondo dell'applicazione ed eseguirne il deploy sul server, gli sarebbe sufficiente modificare la sua copia locale, registrare le modifiche con `commit`, e spedirle al server con `push`. A questo punto agli altri developer basterebbe eseguire il comando `pull` per ottenere la versione aggiornata. Questo rende il processo di deploy semplice ed economico.

2.4.5 Tecnologie utilizzate

Alla luce delle necessità esposte in questo capitolo, e considerate le tecnologie trattate nella sezione 1.3, abbiamo fatto delle scelte su quali tecnologie utilizzare per la rappresentazione e la visualizzazione dei dati nell'applicazione.

HTML5

Abbiamo ampiamente valutato nella sezione 1.3.2 le differenze tra le tecnologie Adobe Flash e HTML5 e, per motivi legati all'accessibilità dell'applicazione, abbiamo optato per visualizzare i dati utilizzando unicamente grafica vettoriale in formato SVG, impiegando la tecnologia HTML5.

D3.js

Questa libreria ci permetterà di manipolare il DOM facilmente in modo da aggiungere/rimuovere nuovi elementi all'interno dei nodi `<svg>`.

CSV

La banca dati offerta da soldipubblici è in formato CSV, quindi impiegheremo questa tecnologia al fine di ottenere i dati grezzi su cui poi faremo ulteriori elaborazioni.

RDBMS

Una volta importati i dati grezzi in un RDBMS, procedimento spiegato in dettaglio nel capitolo 3, procederemo con la produzione della banca dati consumata dall'applicazione utilizzando delle query SQL.

JSON

Essendo un formato molto semplice da gestire tramite il linguaggio JavaScript, lo utilizzeremo per far acquisire all'applicazione le informazioni di cui necessita per produrre le visualizzazioni.

TopoJSON

Dal momento che la nostra banca dati ha una stretta dipendenza con i comuni italiani, è fondamentale che la mappa rifletta correttamente i comuni attuali. Questo ci porta a scartare soluzioni di più alto livello come Google Maps o Bing Maps e a preferire un sistema che ci permetta di alterare facilmente la geometria della mappa.

3 Implementazione

In questo capitolo vedremo più in dettaglio le tecnologie utilizzate per sviluppare l'applicazione ed il contesto in cui sono state inserite. Attraverso l'inclusione di snippet di codice, ripercorreremo i vari passaggi che hanno portato allo sviluppo dell'applicazione *spesapubblica*.

3.1 Altre tecnologie utilizzate

Oltre alle tecnologie per la rappresentazione e la visualizzazione dei dati descritte nella sezione 2.4.5, per lo sviluppo dell'applicazione web e per la preparazione dei dati abbiamo usato diverse altre tecnologie, come illustrato nel seguito.

3.1.1 NodeJS

NodeJS è una runtime che ci permette di eseguire programmi scritti in linguaggio JavaScript. Senza bisogno di aprire un browser è quindi possibile utilizzare NodeJS per eseguire tool e librerie scritte in JavaScript.

hello.js

```
console.log("Hello world!");
```

```
$ node hello.js
Hello world!
```

Codice 9: Esecuzione di Hello world con NodeJS

La community di NodeJS mette a disposizione script liberamente scaricabili utilizzando `npm`, il gestore pacchetti di NodeJS. Per installare un pacchetto, è sufficiente eseguire: `npm install -g nomepacchetto`.

3.1.2 Python

Python è un linguaggio di programmazione multiparadigma. Abbiamo deciso di utilizzarlo nel nostro progetto per via della sua flessibilità e per il fatto che, essendo un linguaggio interattivo, permette di testare facilmente snippet di codice senza bisogno di eseguire un intero script. Inoltre, Python offre direttamente nella sua libreria standard molti strumenti che ci sono tornati utili, come il modulo `csv`.

hello.py

```
print("Hello world!")
```

```
$ python hello.py
Hello world!
```

Codice 10: Esecuzione di Hello world con Python

In modo simile a NodeJS, anche Python mette a disposizione tool e librerie sviluppati dalla community attraverso il gestore di pacchetti `pip`. Per installare un pacchetto Python, è sufficiente eseguire: `pip install nomepacchetto`.

3.1.3 ES2015

Similmente a quanto successo nel caso del formato CSV, di cui abbiamo già parlato nella sezione 1.3.3, anche il linguaggio JavaScript inizialmente non era specificato in modo formale. Nel 1997 però l'organizzazione Ecma standardizzò la specifica ECMAScript, della quale oggi JavaScript è un'implementazione. Ad oggi, la revisione più recente di ECMAScript è la specifica ES2015, anche detta ES6 per via del fatto che è la sesta [Int15].

Il progetto spesapubblica è un'applicazione JavaScript che abbiamo deciso di scrivere utilizzando alcune delle funzionalità della moderna sintassi ES6. I vantaggi rispetto ad ES5 sono infatti:

- Possiamo utilizzare in modo nativo le classi.
- Possiamo scomporre più facilmente la logica in file distinti grazie alle istruzioni `import` ed `export`.
- Possiamo utilizzare il concetto di *promise* senza ricorrere a librerie esterne, grazie all'oggetto `Promise`.

Babel

Per poter utilizzare la sintassi ES6, che non è ancora supportata completamente in tutti i browser, ci avvarremo di Babel. Questo software infatti è un *transpiler*, vale a dire un compilatore che prende in ingresso codice sorgente e stampa in uscita del nuovo codice sorgente.

L'obiettivo di Babel è quello di convertire i nuovi costrutti introdotti da ES6 in forme equivalenti, ma compatibili con ES5. In questo modo possiamo sfruttare le nuove funzionalità del linguaggio senza dover attendere che tutti i browser le supportino.

Babel è un software installabile con il sopracitato `npm`, eseguendo: `npm install -g babel-cli`.

Browserify

Per completare il supporto a ES6 con i moduli, abbiamo usato il tool Browserify, installabile in modo analogo a Babel. Questo tool segue ricorsivamente le istruzioni `import` e produce infine un singolo file con l'applicazione iniziale e tutte le sue dipendenze.

3.1.4 LESS

Un'altra tecnologia di cui abbiamo fatto uso è quella dei transpiler CSS, ovvero quei software che producono codice CSS come risultato della compilazione di un altro codice sorgente. Questo ci ha permesso di scrivere le regole di

classe.js.es6

```
class Polygon {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

```
$ babel --presets=es2015 classe.js.es6 -o classe.js
```

classe.js

```
"use strict";

function _classCallCheck(instance, Constructor) { if (!(instanc...

var Polygon = function Polygon(height, width) {
  _classCallCheck(this, Polygon);

  this.height = height;
  this.width = width;
};
```

Codice 11: Esempio di compilazione da ES6 a ES5 con Babel

stile dell'applicazione in modo più semplice e chiaro evitando anche di dover usare prefissi vendor-specific.

Alcuni tra i più noti transpiler in circolazione sono:

LESS

Scritto in JavaScript. Aggiunge a CSS il supporto a costrutti quali: variabili, regole innestate, funzioni. Può essere eseguito direttamente dal browser, a scapito del tempo richiesto per caricare la pagina.

Sass

Scritto in Ruby. Per certi versi simile a LESS, ma con una sintas-

si diversa. Esiste un'implementazione ottimizzata per la performance chiamata libSass, scritta in linguaggio C.

Nel progetto spesapubblica abbiamo utilizzato LESS, dal momento che nel prototipo iniziale dell'applicazione il CSS veniva compilato direttamente nel browser dell'utente. Con l'utilizzo dei task runner (di cui parleremo nella sezione 3.8.1) non abbiamo più fatto uso di quella funzionalità, ma non avevamo motivo di cambiare e abbiamo quindi continuato a usare LESS.

3.1.5 SQLite

SQLite è un RDBMS scritto in C e molto utilizzato come embedded database all'interno di altre applicazioni. Non segue il modello client-server come PostgreSQL o MySQL, bensì è contenuto interamente nel programma che lo esegue. Un database SQLite può essere mantenuto all'interno di un singolo file.

Abbiamo scelto SQLite come strumento da usare nella preparazione della banca dati per via della sua versatilità e compatibilità.

```
$ sqlite3 datafiles/siope_2015-05-29.sqlite
SQLite version 3.9.2 2015-11-02 18:31:45
Enter ".help" for usage hints.
sqlite> SELECT count(*) FROM comuni;
8136
sqlite>
```

Codice 12: Esempio di utilizzo della linea di comando SQLite

3.1.6 SQLAlchemy

SQLAlchemy è un software di object-relational mapping (ORM) per il linguaggio Python. Questo significa che, da Python, è possibile utilizzare l'API di SQLAlchemy per collegarsi ad un DBMS e da lì gestire le interrogazioni usando normali istanze di classi Python.

Abbiamo scelto di utilizzare un ORM perché, sebbene durante la preparazione della banca dati non venga preso input da parte degli utenti, il mapping permette di astrarre la composizione delle query SQL in modo che i nostri script non abbiano bisogno di fare sanity-checking.

3.2 Preparazione della banca dati

Cominciamo con l'introdurre il modello che abbiamo usato per mappare i record delle tabelle agli oggetti Python.

3.2.1 Modello SQLAlchemy

All'interno della cartella `spesapubblica/model` troviamo i seguenti file:

Comune.py
: Descrive un singolo comune mediante i campi `cod_comune`, `descr_comune`, `cod_provincia`.

Payment.py
: Descrive un singolo pagamento mediante i campi `cod_ente`, `anno`, `periodo`, `codice_gestionale`, `imp_uscite_att`.

SiopeEntity.py
: Descrive un singolo ente/amministrazione mediante i campi `cod_ente`, `data_inc_siope`, `data_esc_siope`, `cod_fiscale`, `descr_ente`, `cod_comune`, `cod_provincia`, `num_abitanti`, `sottocomparto_siope`.

SimpleTable.py
: Questo modello introduce i campi `_created`, `_updated`. Questi campi contengono sempre la data di creazione e aggiornamento del record, rispettivamente. Tutti i modelli sopracitati ereditano da SimpleTable.

Lo schema seguito, eccetto nel caso di SimpleTable, è quello descritto in `http://soldipubblici.gov.it/it/developers`. Nel codice 13 è possibile vedere uno dei file citati.

```python
#-*- encoding: utf8 -*-

from SimpleTable import SimpleTable
from sqlalchemy import Column, Integer, String, DateTime

class Comune(SimpleTable):
    __tablename__ = 'comuni'

    # Column definition taken from:
    # http://soldipubblici.gov.it/it/developers
    cod_comune = Column(Integer)        # Codice Comune
    descr_comune = Column(String)       # Descrizione Comune
    cod_provincia = Column(Integer)     # Codice Provincia
```

Codice 13: Il modello `Comune.py`, uno dei mapping utilizzati

3.2.2 Produzione della banca dati derivata

La preparazione della nostra banca dati JSON a partire dalla banca dati offerta da soldipubblici si compone dei seguenti passaggi:

Download dei dati

Dal sito `http://soldipubblici.gov.it/it/developers` si scaricherà il file `Siope-dati.zip` e lo si posizionerà nella cartella di lavoro.

Creazione del database e importazione dei dati

Qualora non sia già stato fatto, sarà necessario creare il database inizializzandone le tabelle. Per fare questo abbiamo preparato lo script `spesapubblica/create_db.py`. Il programma crea il database, estrae l'archivio ZIP passato come parametro, e infine importa i dati CSV all'interno del database. Questa operazione può impiegare molto tempo.

Produzione dei report in JSON

Al fine di poter esser consumati dall'applicazione web, i dati verranno convertiti in JSON. Per eseguire la conversione abbiamo preparato lo

script `spesapubblica/aggregate_data.py`, che accetta due flag: `-y` e `-m` per l'anno e il mese da considerare, rispettivamente.

Nel codice 14 possiamo vedere come produrre il report per un singolo mese. Omettendo il flag `-m` vengono considerati tutti i mesi dell'anno specificato. In modo analogo, omettendo `-y` vengono considerati tutti gli anni.

```
$ python spesapubblica/aggregate_data.py -y 2015 -m 11
```

Codice 14: Il comando che produce il report in JSON per novembre 2015.

3.3 Sviluppo dell'applicazione web

Per sviluppare l'applicazione web abbiamo adottato il framework AngularJS, che ci permette di separare i ruoli dei vari componenti in modo ordinato.

3.3.1 Single-page application in AngularJS

Una single-page application (SPA) è un'applicazione web che non richiede il ricaricamento della pagina quando avviene un click o i dati vengono aggiornati. AngularJS ci permette di costruire facilmente una SPA grazie a funzionalità come: le route, le view, i controller, i servizi standard (ad esempio `$http`, per acquisire dati via AJAX).

AngularJS si può includere dal file `index.html` come una qualsiasi libreria JavaScript. Per includere la definizione dell'applicazione, dei controller e dei servizi che abbiamo progettato possiamo scegliere tra due modalità: includere tutti i file JavaScript come normali librerie oppure "richiedere" i file secondari da un singolo file principale.

Nella nostra applicazione abbiamo optato per la seconda modalità, usando il già citato Browserify per produrre un bundle da usare in produzione.

3.3.2 Il router

Uno dei punti deboli che abbiamo riscontrato durante l'utilizzo di Angular è il componente router. Infatti, quello fornito di default è risultato piuttosto limitativo:

- Non permette di fare routing a profondità maggiore di 1, vale a dire: non si possono innestare view all'interno di altre view.
- Non permette di spostarsi in modo programmatico da uno stato all'altro: è necessario usare i comandi per l'aggiornamento dell'URL.
- Anche nel caso dei link HTML, per spostarsi da uno stato all'altro è necessario usare l'attributo `href` del tag `<a>` specificando il nuovo URL.

Per questo motivo, abbiamo deciso di utilizzare un componente di terze parti chiamato **ui-router**. Sebbene sia sviluppato in modo indipendente dal progetto AngularJS, questo componente è molto utilizzato[1] ed è stato annunciato che, in un prossimo futuro, Angular 2.0 avrà un nuovo componente router che sarà progettato da capo, basandosi anche su ui-router.

Il componente ui-router tratta i vari route come "stati" che hanno: nome, URL, template, possibilmente dei parametri e, opzionalmente, uno stato padre. Il componente ci mette a disposizione dei servizi chiamati `$stateProvider`, `$stateParams` e `$state`. Il primo è utilizzato per definire gli stati, il secondo viene usato per acquisire i parametri passato allo stato corrente, mentre l'ultimo viene usato per gestire gli stati o cambiare stato.

Nel nostro caso definiremo cinque stati: uno stato principale a cui assegneremo il controller `dashboard_ctrl`, e quattro stati "figli" corrispondenti agli URL specifici per la visualizzazione completa, quella per regioni, quella per province e quella per comuni.

3.3.3 URL semantici

Un estratto del file principale dell'applicazione spesapubblica è illustrato nel codice 15. Come si può notare, ogni stato specifica un URL.

[1] Il repository angular-ui/ui-router, ad oggi, vanta quasi 10 000 stelle su Github.

web/assets/js/app.js.es6

```
angular
  .module("spesapubblica", ["ui.router"])
  .config(function($stateProvider) {
    $stateProvider
      .state("dashboard", {
        templateUrl: "assets/html/dashboard.html",
        controller: "dashboard_ctrl"
      })
      .state("dashboard.italia", {
        url: "/italia?g",
        templateUrl: "assets/html/graph.html",
        controller: "graph_ctrl"
      })
      .state("dashboard.regione", {
        url: "/regione/{regione_id}?g",
        templateUrl: "assets/html/graph.html",
        controller: "graph_ctrl"
      })
      .state("dashboard.provincia", {
        url: "/provincia/{provincia_id}?g",
        templateUrl: "assets/html/graph.html",
        controller: "graph_ctrl"
      })
      .state("dashboard.comune", {
        url: "/comune/{comune_id}?g",
        templateUrl: "assets/html/graph.html",
        controller: "graph_ctrl"
      });
  });
```

Codice 15: Estratto del file `app.js.es6` contenente gli stati definiti per l'applicazione spesapubblica.

Gli URL che abbiamo scelto sono semantici in quanto è chiaro cosa aspet-

tarsi visitando la pagina all'indirizzo `/provincia/milano`, ed è anche possibile sfruttare questo per modificare manualmente l'URL e raggiungere nuove pagine.

3.3.4 I controller

I controller che abbiamo definito per l'applicazione spesapubblica sono:

dashboard_ctrl

Questo controller si occupa di inizializzare la dashboard e di attuare sul model le varie azioni effettuate dall'utente. È una classe che espone i seguenti metodi:

go_to_parent() – viene eseguito quando l'utente clicca sul pulsante per tornare al livello superiore, situato in alto a destra nella mappa.

go_to_searched() – viene eseguito quando l'utente effettua una ricerca tramite l'apposita barra in alto.

get_matches(text) – viene eseguito ogni volta che l'utente digita qualcosa sulla barra di ricerca, e restituisce una lista di risultati per la stringa `text`.

update_granularity() – viene eseguito quando l'utente cambia la granularità (ad esempio da "Regioni" a "Province") e, internamente, utilizza il servizio `mapchart` che descriveremo in seguito.

change_year() – analogo a `update_granularity()`.

Inoltre ci sono altri metodi usati per gestire, ad esempio, il click sui pulsanti "Chi siamo" e "Guida".

graph_ctrl

Questo controller si occupa di inizializzare la mappa. Sostanzialmente, richiama `mapchart` in modo da sincronizzarne la visualizzazione con i parametri specificati nell'URL. Per ottenere tali parametri, il controller fa uso del servizio `$stateParams` descritto precedentemente. Ogni volta che l'URL cambia per via di un aggiornamento dello stato, questo controller verrà richiamato.

3.3.5 I servizi

I servizi che abbiamo definito per l'applicazione spesapubblica sono:

info

Questo servizio si occupa di mantenere lo stato dell'applicazione. È una classe che espone le proprietà: `selected_year`, `num_categories`, e così via.

dataloader

Questo servizio verrà descritto in dettaglio nella sezione 3.5.4.

mapchart

Questo servizio si occupa della mappa dell'Italia, chiedendo opportunamente a `dataloader` i dati di cui ha bisogno. Espone i metodi: `change_state`, `reload_state`, e così via.

3.4 Mappa vettoriale

La mappa da realizzare, in gergo chiamata *mappa coropletica*, è una versione della mappa dell'Italia suddivisa in regioni colorate in base a un criterio. Per realizzarla, abbiamo inizialmente preso in considerazione l'utilizzo dell'API di Google Maps e di Bing. Tuttavia, abbiamo notato che entrambe non supportano dati in TopoJSON. Dal momento che volevamo produrre una mappa fedele alle più recenti divisioni/unioni dei comuni italiani, abbiamo quindi optato per utilizzare direttamente le librerie D3.js e TopoJSON, in modo da usare le funzionalità di quest'ultima (descritte nella sezione 1.3.4) per avere la possibilità di modificare manualmente i dati topografici.

La mappa dell'Italia presente nell'applicazione viene quindi generata tramite D3.js che, attraverso l'uso di elementi SVG, renderizza il codice TopoJSON che noi gli forniamo. Quest'ultimo viene acquisito tramite il modulo dataloader, il quale verrà descritto nella sezione 3.5.4.

3.4.1 Servizio `mapchart`

Il servizio mapchart è il responsabile della corretta visualizzazione della mappa SVG. Nel codice 16 possiamo vedere un estratto del metodo `init()`, molto importante per la creazione e inizializzazione dei vari elementi SVG che compongono la mappa.

web/assets/js/mapchart.js.es6

```
init() {
  this._width = $(this._container).width();
  this._height = $(this._container).height();

  this._svg = d3.select(this._container).append('svg')
    .attr("width", this._width)
    .attr("height", this._height);
  this._g = this._svg.append('g');
  this._graticule_g = this._g.append('g')
    .attr("id", "graticule");
  this._comuni_g = this._g.append('g')
    .attr("id", "comuni");
  ...
  this._projector = d3.geo.mercator().scale(1).translate([0, 0]);
  this._path = d3.geo.path().projection(this._projector);
  this._tooltip.on('mouseenter', this._hide_tooltip);

  // Load regions
  this._wait_for_regioni = this._create_regioni(regions => {
    ...
    this._projector
      .scale(s)
      .translate(t);
  }
}
```

Codice 16: Estratto (semplificato) del metodo `init()` contenuto nel servizio `mapchart`.

3.4.2 Layering dei dati geografici

L'utilizzo di TopoJSON ci ha anche permesso di ideare un metodo per memorizzare in modo più efficiente i dati cartografici. Possiamo infatti notare che, aumentando la granularità da "regioni" a "province", stiamo visualizzando in realtà sia le regioni che le province (perché la regione di cui stiamo visualizzando le province *condivide* i suoi bordi con diverse province).

Abbiamo quindi messo a punto un sistema di layering che ci permette di diminuire la dimensione dei file che l'utente deve scaricare. Lo script che esegue questo compito si trova nel file `spesapubblica/scripts/layers.js`.

L'applicazione web penserà poi ad unire correttamente i dati usando il metodo `merge_objects` di mapchart. È bene notare che `merge_objects` non fa altro che sostituire il primo parametro con l'unione dei due parametri: è quindi equivalente a `_.merge` offerto dalla libreria lodash o a `Object.assign` offerto da ES6. Abbiamo deciso di reimplementare questa procedura per motivi di efficienza.

3.5 Caricamento dei dati

Il caricamento dei dati all'interno dell'applicazione avviene in modo asincrono, in particolare tramite richieste AJAX.

3.5.1 La funzione `d3.json`

Le richieste AJAX effettuate dall'applicazione consistono di chiamate alla funzione `d3.json`, la quale richiama internamente `d3.xhr`.

La funzione `d3.xhr` scarica i dati su richiesta utilizzando l'API standard `XMLHttpRequest` offerta da JavaScript. Usando `d3.json` inoltre richiediamo in modo specifico dati JSON, attraverso il mime type `application/json`.

3.5.2 Le promise in JavaScript

Una *promise* (letteralmente "promessa") è un oggetto che ci permette di gestire in modo semplificato le risorse che richiedono del tempo per essere pronte.

Se ci troviamo in una situazione in cui dobbiamo eseguire del codice al termine di un download, oppure alla fine di un calcolo molto impegnativo, possiamo usare la strategia del callback: passeremo al codice che dovrà preparare i dati una funzione `callback` da chiamare alla fine.

La strategia delle promise è più flessibile dei callback, perché ci permette di specificare funzioni da richiamare anche *dopo* aver iniziato la preparazione dei dati. Infatti, una promise memorizza internamente i callback in attesa e, non appena i dati sono pronti, la promise cambia stato da "pending" a "fulfilled", e comincia a richiamare tutti i callback che ha memorizzato.

Se si aggiunge un callback ad una promise che è "fulfilled", quel callback verrà eseguito immediatamente.

3.5.3 Utilizzo delle promise in spesapubblica

Il modello delle promise risulta molto comodo nel progetto spesapubblica perché in alcuni casi l'applicazione potrebbe causare la richiesta degli stessi dati. Ad esempio, alla prima visita dell'applicazione verranno scaricati i dati della mappa relativi alle regioni. Non verranno però scaricati i dati relativi ai comuni, perché potrebbe rivelarsi uno spreco di tempo (nel caso in cui l'utente non volesse eseguire lo zoom sui comuni). Nel momento in cui viene però eseguito lo zoom sui comuni, è necessario scaricare i dati.

In un caso di questo tipo, in cui diversi punti del codice possono richiedere lo scaricamento degli stessi dati, si può usare una promise che "prometta" di trovare i dati, e aggiungerle ogni volta dei callback con le operazioni da eseguire con i dati.

```
wait_for_comuni.then(function(id_comuni) {
    // esegui operazioni sull'array: id_comuni
}, function(error) {
    // c'è stato un errore e la promise è stata rigettata
});
```

Codice 17: Esempio di una promise di nome `wait_for_comuni` usata per scaricare i dati dei comuni.

3.5.4 Servizio `dataloader`

Il servizio `dataloader` gestisce la logica di richiesta dei dati via AJAX. Contiene metodi e proprietà, le quali sono principalmente delle promise sulle quali eseguire `.then`. Alcune di queste sono:

regione_prepare_urls
: È una promise che, quando viene risolta, può essere usata per lavorare con i dizionari `regione_id_2_url` e `regione_url_2_id`.

provincia_prepare_parents
: È una promise che, quando viene risolta, può essere usata per lavorare con il dizionario `provincia_parent`.

3.6 Interfaccia web

Per l'interfaccia web dell'applicazione abbiamo utilizzato Angular Material, una libreria sviluppata dallo stesso team che ha sviluppato Angular.

3.6.1 Angular Material

Abbiamo scelto questa libreria perché:

- Ci permette di utilizzare componenti (pulsanti, campi di testo, checkbox, e così via) già pronti, riusabili, testati e conformi alla specifica del Material Design proposta da Google.
- È pensata per lavorare in coppia con AngularJS, infatti si interfaccia bene con i suoi moduli. Per esempio espone il servizio `$mdDialog` per creare finestre di dialogo dal codice di un controller o servizio Angular. Inoltre, i vari componenti sono implementati tramite direttive Angular.

Alcune dele direttive di Angular Material che abbiamo utilizzato sono:

md-autocomplete
: Abbiamo usato questa direttiva per la barra di ricerca, perché produce un campo di input dotato di autocompletamento dei dati inseriti.

md-button

Abbiamo usato questa direttiva per i pulsanti “Guida”, “Chi siamo” oltre a quelli presenti sulla barra destra.

md-toolbar

Abbiamo usato questa direttiva per l’header della pagina, che contiene a sua volta una direttiva `md-autocomplete`.

3.7 Accessibilità

Al fine di rendere la nostra applicazione accessibile, abbiamo deciso di prestare particolare attenzione alla scelta dei colori da usare nella mappa, oltre che a fare uso della tecnologia ARIA descritta nella sezione 1.2.5.

3.7.1 Codifica dei colori nella mappa

Come indicato nella sezione 2.4.2 durante la progettazione dell’applicazione, per comunicare in modo immediato all’utente l’informazione dell’entità della spesa sostenuta da ciascuna area abbiamo deciso di interpolare il colore di ogni regione a partire da due colori fissi.

Nella fase di implementazione, abbiamo deciso che i due colori da utilizzare sarebbero stati il blu scuro ed il blu chiaro. In particolare, abbiamo usato il blu scuro per le aree che hanno speso di più e il blu chiaro per quelle che hanno speso di meno. In fig. 11 si può vedere un esempio di tale visualizzazione.

Inizialmente avevamo considerato di utilizzare i colori rosso e verde, come si può vedere in fig. 12, ma abbiamo deciso poi di implementare questa funzionalità usando colori che variassero solo in luminosità in modo da:

- Rendere la mappa più accessibile a persone affette dai diversi tipi di daltonismo.
- Evitare di comunicare involontariamente dei giudizi (rosso: negativo, verde: positivo). Infatti, spendere tanto o poco non è automaticamente un aspetto negativo o positivo.

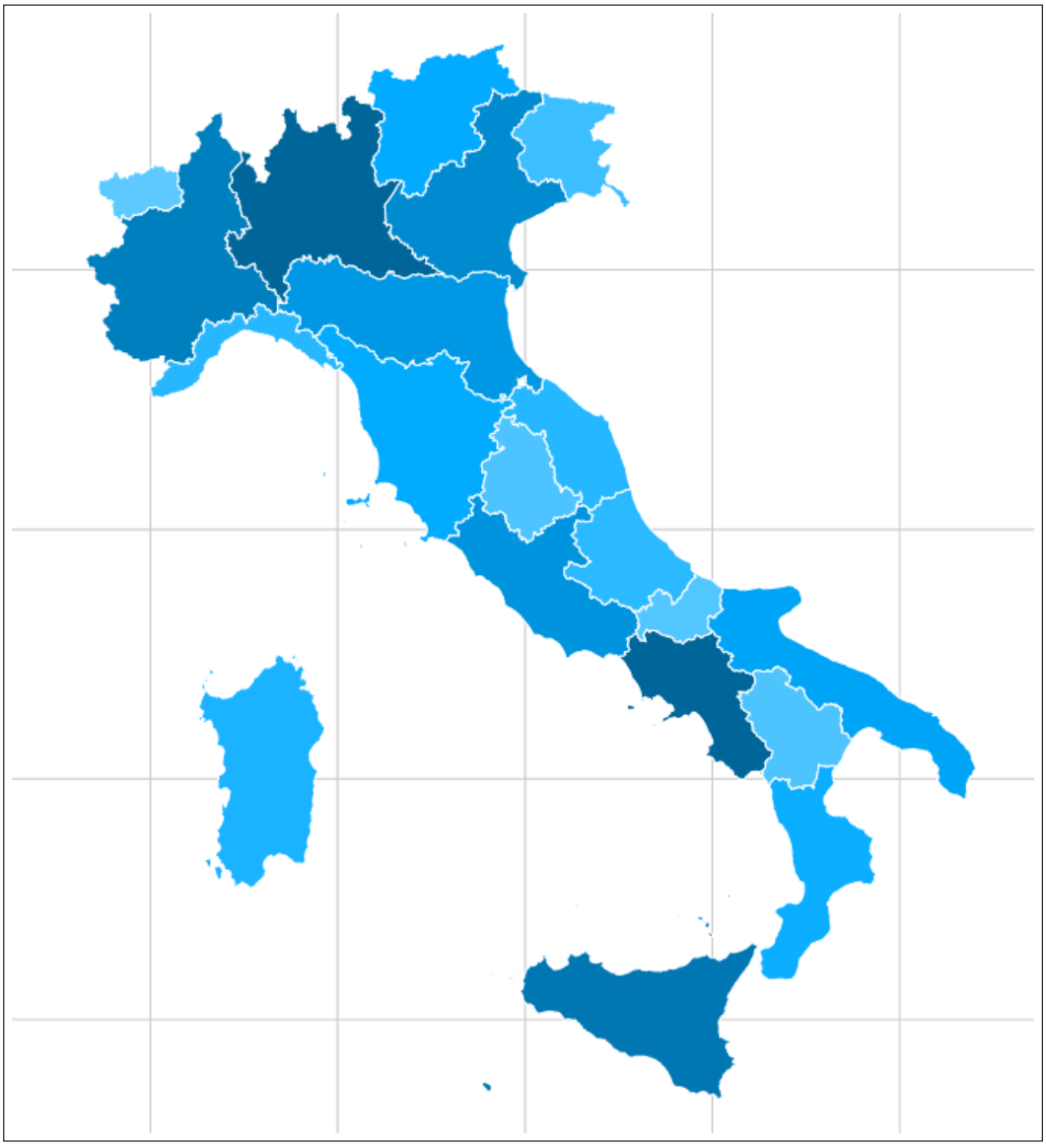


Figura 11: Esempio di visualizzazione con tonalità di blu

La fig. 12 è decisamente poco accessibile. Infatti, le gradazioni tra il rosso ed il verde sono quelle che più comunemente risultano difficili da distinguere per persone affette da daltonismo [Col].

In una futura versione di spesapubblica è possibile che verrà aggiunta la possibilità di confrontare l'aumento/diminuizione della spesa di un area nel tempo (ad esempio, per vedere quanto è variata la spesa tra due mandati

Figura 12: Esempio di visualizzazione con gradazioni di rosso e verde

politici). Per tale funzionalità si potrebbe decidere di usare una colorazione simile alla fig. 11 ma con un diverso colore di base.

3.7.2 ngAria

AngularJS mette a disposizione il modulo ngAria che, una volta incluso come dipendenza della propria applicazione, si occuperà di riempire correttamente

i tag ARIA più comuni, come: `aria-hidden`, `aria-disabled` e via discorrendo. Ad esempio, se avessimo uno `<span ng-hide="check()">Ciao</span>`, allora Angular lo nasconderebbe o visualizzerebbe in base al valore di verità restituito dalla chiamata `check()`. Il modulo ngAria, in un caso del genere, andrebbe anche ad aggiungere il valore corretto dell'attributo `aria-hidden`.

$mdAria

Similmente a quanto fatto da Angular con ngAria, anche la libreria Angular Material utilizza il modulo `$mdAria` per far sì che alcuni attributi vengano riempiti automaticamente. Per esempio, un radio button con testo "Cancelleria" riceverà automaticamente l'attributo `aria-label` con valore `Cancelleria` grazie all'intervento di `$mdAria`.

Come si può vedere in fig. 13 il modulo $mdAria emette anche degli avvisi per elementi a cui mancano attributi importanti.

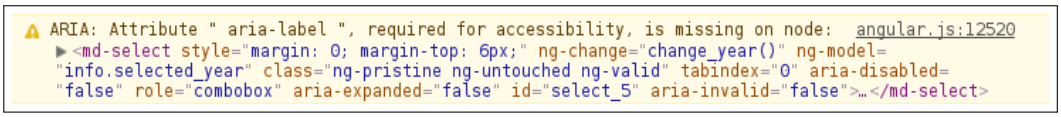


Figura 13: Un avviso comparso nella console del browser Chrome, causato da un attributo `aria-label` mancante.

3.8 Aggiornamento dell'applicazione e dei dati

Per aggiornare l'applicazione e/o i dati abbiamo inizialmente eseguito manualmente i vari script e le procedure necessarie. Durante lo sviluppo, però, abbiamo ritenuto più produttivo cominciare utilizzare un *task runner*.

3.8.1 Task runner

Nello sviluppo web un task runner è uno strumento simile a GNU Make: lo sviluppatore definisce un file con i task da svolgere, i comandi necessari a svolgerli e le dipendenze tra i task (ad esempio: "mettere le scarpe" dipende da "mettere i calzini", mentre "mettere i guanti" non dipende da altri task),

mentre un tool automatico si occupa di leggere quel file ed eseguire i task necessari.

Cercando un task runner per il nostro progetto, abbiamo notato che Grunt e Gulp sono entrambi molto usati per le applicazioni web (entrambi sono scritti in JavaScript). La differenza tra i due è la sintassi usata per definire i task. Gulp ci è sembrato il più facile da usare, per cui abbiamo optato per quello. Nel codice 18 possiamo vedere un esempio di gulpfile.

gulpfile.js

```
gulp.task('scripts', ['clean'], function() {
  // Minify and copy all JavaScript (except vendor scripts)
  // with sourcemaps all the way down
  return gulp.src('client/js/**/*.js.es6')
  .pipe(sourcemaps.init())
  .pipe(babel({
    presets: ['es2015']
  }))
  .pipe(uglify())
  .pipe(concat('all.min.js'))
  .pipe(sourcemaps.write())
  .pipe(gulp.dest('build/js'));
});
```

Codice 18: Estratto di un `gulpfile.js`, con un task **scripts** che dipende dal task **clean** e che esegue compilazione e minificazione di codice JS.

Il gulpfile che abbiamo utilizzato nel progetto spesapubblica comprende i task che eseguivamo più spesso durante lo sviluppo dell'applicazione:

js

Questo task esegue la compilazione del codice ES6 in ES5.

css

Questo task esegue la compilazione del codice LESS in CSS, aggiungendo anche i vendor prefix.

minify

Questo task esegue la minificazione del codice ES5 e di quello CSS.

default

Questo è il task che viene eseguito quando non viene specificato un task (ovvero eseguendo `gulp` senza argomenti). Dipende da **js**, **css**, **minify**, i quali verranno quindi eseguiti automaticamente (se necessario). Inoltre, questo task avvia il webserver usando la libreria BrowserSync, che ci permette di ricaricare automaticamente il browser nel momento in cui avviene un cambiamento in uno dei sorgenti.

In una futura versione di spesapubblica potremmo anche aggiungere nuovi task con le operazioni che vengono eseguite meno spesso, come la generazione della banca dati descritta nella sezione 3.2.2 e la generazione delle mappe descritta nella sezione 3.4.2.

3.9 Deploy dell'applicazione

Il deploy, ovvero la fase in cui l'applicazione web viene messa in funzione sul server al quale era destinata, è stato strutturato nelle fasi descritte di seguito.

3.9.1 Creazione repository e hosting dell'applicazione

Come già descritto nella sezione 2.4.4, abbiamo deciso di usare il servizio Github Pages per l'hosting della nostra applicazione. La prima fase del deploy è stata quindi: creare un repository git sul sito web `https://github.com`.

Dopo aver creato il repository, che si trova all'indirizzo `https://github.com/gabrfarina/spesapubblica`, abbiamo eseguito in ordine le operazioni:

- Generazione della banca dati.
- Generazione dei layer della mappa.
- Preparazione dei file CSS e JS minificati per l'applicazione web.
- Esecuzione di `git push` sul branch `gh-pages`, che è quello che viene automaticamente servito all'indirizzo `http://gabrfarina.github.io/spesapubblica`.

Si può verificare, come indicato nella sezione 1.2.10, che Github supporta la compressione gzip per le richieste HTTP.

3.9.2 Scelta del dominio

Al fine di avere un nome facilmente memorizzabile e per sfruttare al meglio gli URL semantici della nostra applicazione, abbiamo deciso di acquistare il dominio `spesapubblica.in`, sottodominio del TLD indiano.

Questo ci permette di assegnare in modo molto intuitivo un significato preciso a URL di questo tipo:

- `http://spesapubblica.in/#/italia`
- `http://spesapubblica.in/#/regione/abruzzo`
- `http://spesapubblica.in/#/provincia/teramo`
- `http://spesapubblica.in/#/comune/roseto-degli-abruzzi`

Per fare in modo che `spesapubblica.in` venisse redirezionato al dominio assegnato di default da Github Pages, abbiamo utilizzato il pannello online di configurazione offerto dal provider del dominio.

Conclusioni

A conclusione della stesura di questo volume, faremo ora un resoconto del lavoro realizzato nell'ambito del progetto di tesi.

Il lavoro che è stato realizzato per questa tesi consiste in un'applicazione web atta alla fruizione, da parte dei cittadini, degli open data prodotti dalle pubbliche amministrazioni italiane.

L'applicazione sviluppata consiste di una mappa coropletica affiancata da una dashboard con dei controlli che permettono di modificare i parametri di visualizzazione della stessa.

Dalla pagina iniziale è possibile raggiungere qualsiasi posizione (ad esempio: la provincia di Milano, il comune di Cesena, la regione Toscana) utilizzando: la barra dell'URL, la barra di ricerca integrata nell'applicazione, la mappa stessa (tramite click).

Lo sviluppo dell'applicazione è stato portato a termine utilizzando per quanto possibile alcune delle più recenti tecnologie web, come:

- HTML5
- ES2015
- AngularJS
- D3.js
- Le promise JavaScript

Inoltre, sono state adottate diverse tecniche e strumenti che migliorano e semplificano il lavoro di sviluppo, per questo tipo di applicazioni:

- Transpiler CSS

- Transpiler JavaScript
- Task runner
- Software di controllo versione

Per quanto riguarda la rappresentazione degli open data, nella sezione 3.2 è stato illustrato il procedimento che abbiamo impiegato per preparare la banca dati. Nelle sezioni 3.4 e 3.5 invece, sono stati delineati i dettagli su come abbiamo prodotto la visualizzazione di questi open data.

Al termine dello sviluppo ci siamo resi conto che alcune parti del codice si potevano riscrivere in modo migliore, e la logica di memorizzazione e recupero dei dati delle varie tipologie di amministrazione (regione, provincia, comune) si poteva semplificare significativamente.

Inoltre il framework AngularJS, che abbiamo usato per la parte di routing/templating della nostra applicazione, sta per rilasciare la prima beta di Angular 2, la prossima major release. Una delle particolarità, come già accennato nella sezione 3.3.2, sarà la presenza di un rinnovato componente router. L'altra principale modifica fatta al framework però è la completa riscrittura in TypeScript, un linguaggio sviluppato da Microsoft e pensato per essere un estensione di JavaScript.

Alla luce di queste novità, abbiamo deciso che gli sviluppi futuri dell'applicazione spesapubblica saranno:

1. Ristrutturare la logica dell'applicazione. Vogliamo generalizzare la memorizzazione ed il recupero dei dati relativi alle amministrazioni pubbliche (questo avrà come conseguenza la possibilità di supportare un livello arbitrario di nesting dei dati).

2. Effettuare l'aggiornamento da Angular 1.5 ad Angular 2, che non dovrebbe essere particolarmente impegnativo grazie al fatto che abbiamo già usato la sintassi ES6.

3. Adottare TypeScript. Dal momento che questo linguaggio è un superset del JavaScript, tecnicamente tutto il nostro codice è già a tutti gli

effetti TypeScript, senza bisogno di fare alcuna modifica. Vogliamo però provare ad utilizzare le funzionalità introdotte da questo linguaggio (ad esempio i membri privati delle classi).

Bibliografia

[AGI15] AGID. *Dati aperti della PA: on line il nuovo portale dati.gov.it*. 2015. URL: http://www.agid.gov.it/node/1988.

[AS10] Jose Maria Arranz Santamaria. *The Single Page Interface Manifesto*. 2010. URL: http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php.

[Ban] *Sistema informativo delle operazioni degli enti pubblici*. URL: https://www.bancaditalia.it/compiti/tesoreria/siope/index.html.

[Bma] *Bing Maps*. URL: https://msdn.microsoft.com/en-us/library/dd877180.aspx.

[Bos12] Metcalf Calvin Bostock Mike. *The TopoJSON Format Specification*. 2012. URL: https://github.com/mbostock/topojson-specification/blob/master/README.md.

[But08] Daly Martin Doyle Allan Gillies Sean Schaub Tim Schmidt Christopher Butler Howard. *The GeoJSON Format Specification*. 2008. URL: http://geojson.org/geojson-spec.html.

[Cod] *DECRETI SIOPE - CODIFICHE GESTIONALI*. URL: http://www.rgs.mef.gov.it/VERSIONE-I/e-GOVERNME1/SIOPE/Codifica-g/.

[Col] Colblindor. *Deuteranopia – Red-Green Color Blindness*. URL: http://www.color-blindness.com/deuteranopia-red-green-color-blindness/.

[Eck11] Wayne Eckerson. *Visual Reporting and Analysis: Seeing Is Knowing*. TDWI, 2011.

[EH11] Wayne Eckerson e Mark Hammond. *Data Visualization Technology*. 2011. URL: https://tdwi.org/articles/2011/11/09/research-excerpt-data-visualization-technology.aspx.

[Fur09] Susanne Furman. *Credibility*. 2009. URL: http://www.usability.gov/get-involved/blog/2009/10/credibility.html.

[Ghp] *What are GitHub Pages?* URL: https://help.github.com/articles/what-are-github-pages/.

[Gma] *Google Maps Javascript API*. URL: https://developers.google.com/maps/documentation/javascript/.

[Gra15] Robert Graham. *Pin-pointing China's attack against GitHub*. 2015. URL: http://blog.erratasec.com/2015/04/pin-pointing-chinas-attack-against.html.

[Gub09] Jacob Gube. *10 Useful Flash Components for Graphing Data*. 2009. URL: http://sixrevisions.com/flashactionscript/10-useful-flash-components-for-graphing-data/.

[Hab62] Jurgen Habermas. *The Structural Transformation of the Public Sphere*. 1962.

[Htm] *HTML5*. 2014. URL: http://www.w3.org/TR/html5/.

[Int15] Ecma International. *ECMAScript® 2015 Language Specification*. 2015. URL: http://www.ecma-international.org/ecma-262/6.0/.

[Kos59] McCarthy Thomas Koselleck Reinhart. *Critique and Crisis*. 1959.

[Kru00] Steve Krug. *Don't Make Me Think*. Peachpit, 2000.

[Lat10] Ruma Laurel Lathrop Daniel. *Open Government: Transparency, Collaboration and Participation in Practice*. 2010.

[McC15] Rich McCormick. *YouTube drops Flash for HTML5 video as default*. 2015. URL: http://www.theverge.com/2015/1/27/7926001/youtube-drops-flash-for-html5-video-default.

[Md] *Material design*. URL: https://www.google.com/design/spec/material-design/introduction.html.

[Min12] Jim Minatel. *4 JavaScript Minifiers Compared*. 2012.

[Msc] *Making canvas content accessible*. URL: https://msdn.microsoft.com/en-us/library/windows/apps/hh700328.aspx.

[New15] Jesse Newland. *Large Scale DDoS Attack on github.com*. 2015. URL: https://github.com/blog/1981-large-scale-ddos-attack-on-github-com.

[Nie00] Jakob Nielsen. "Flash: 99% Bad". In: *Nielsen Norman Group* (2000).

[Nie05] Jakob Nielsen. *Scrolling and Scrollbars*. 2005. URL: http://www.nngroup.com/articles/scrolling-and-scrollbars/.

[Nie09] Jakob Nielsen. *Top 10 Information Architecture (IA) Mistakes*. 2009. URL: http://www.nngroup.com/articles/top-10-ia-mistakes/.

[Nie10] Jakob Nielsen. *Website Response Times*. 2010. URL: http://www.nngroup.com/articles/website-response-times/.

[Nie93] Jakob Nielsen. *Response Times: The 3 Important Limits*. 1993. URL: http://www.nngroup.com/articles/response-times-3-important-limits/.

[Nym07] Robert Nyman. *The Importance Of A Semantic URL*. 2007. URL: http://robertnyman.com/2007/03/16/the-importance-of-a-semantic-url/.

[Ope] *The Open Definition*. URL: http://opendefinition.org/.

[Opi06] Pascal Opitz. "Clean URLs for better search engine ranking". In: (2006).

[Osg] *Digital Government*. URL: https://www.whitehouse.gov/sites/default/files/omb/egov/digital-government/digital-government.html.

[PA11] Formez PA. *Open Data Come rendere aperti i dati delle pubbliche amministrazioni*. 2011. URL: http://www.funzionepubblica.gov.it/media/982175/vademecumopendata.pdf.

[Pos] *PostGIS*. URL: http://wiki.openstreetmap.org/wiki/PostGIS.

[Sch03] Michael Schröpl. *Which browsers can handle `Content-Encoding: gzip` ?* 2003. URL: http://schroepl.net/projekte/mod_gzip/browser.htm.

[Sch14] Amy Schade. *Responsive Web Design (RWD) and User Experience*. 2014. URL: http://www.nngroup.com/articles/responsive-web-design-definition/.

[Sha05] Yakov Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. 2005. URL: https://tools.ietf.org/html/rfc4180.

[Shi12] Clay Shirky. *How the Internet will (one day) transform government*. 2012. URL: https://www.ted.com/talks/clay_shirky_how_the_internet_will_one_day_transform_government.

[W3C00] W3C. *Accessibility Features of SVG*. 2000. URL: http://www.w3.org/TR/SVG-access/.

[W3C06] W3C. *WAI-ARIA Overview*. 2006. URL: http://www.w3.org/WAI/intro/aria.php.

[Wgs] URL: http://wiki.openstreetmap.org/wiki/Software_comparison/Converting_to_WGS84.

[Wha] *What is Open Data?* URL: http://opendatahandbook.org/guide/en/what-is-open-data/.

[Whi13] Kathryn Whitenton. *Minimize Cognitive Load to Maximize Usability*. 2013. URL: http://www.nngroup.com/articles/minimize-cognitive-load/.