POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Scloby: Sviluppo di alcune nuove funzionalità e proposte tecnologiche per la nuova versione

Relatore
prof. Giovanni Malnati

Candidato
Giorgio Romano

POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Scloby: Sviluppo di alcune nuove funzionalità e proposte tecnologiche per la nuova versione

Relatore
prof. Giovanni Malnati

firma del relatore

……

Candidato
Giorgio Romano

firma del candidato

……

Introduzione

Scloby è un'azienda operante nel settore della vendita di "point of sales" e della analisi dei dati. Nata nel 2012 da una idea di Francesco Medda, è oggi uno dei più importanti player italiani del settore. Scloby è presente in centinaia di esercizi, e possiede una rete commerciale con numerosi rivenditori ed una serie di partnership e collaborazioni con grandi imprese del settore IT e commerciale, quali Samsung, Metro, Epson ecc…

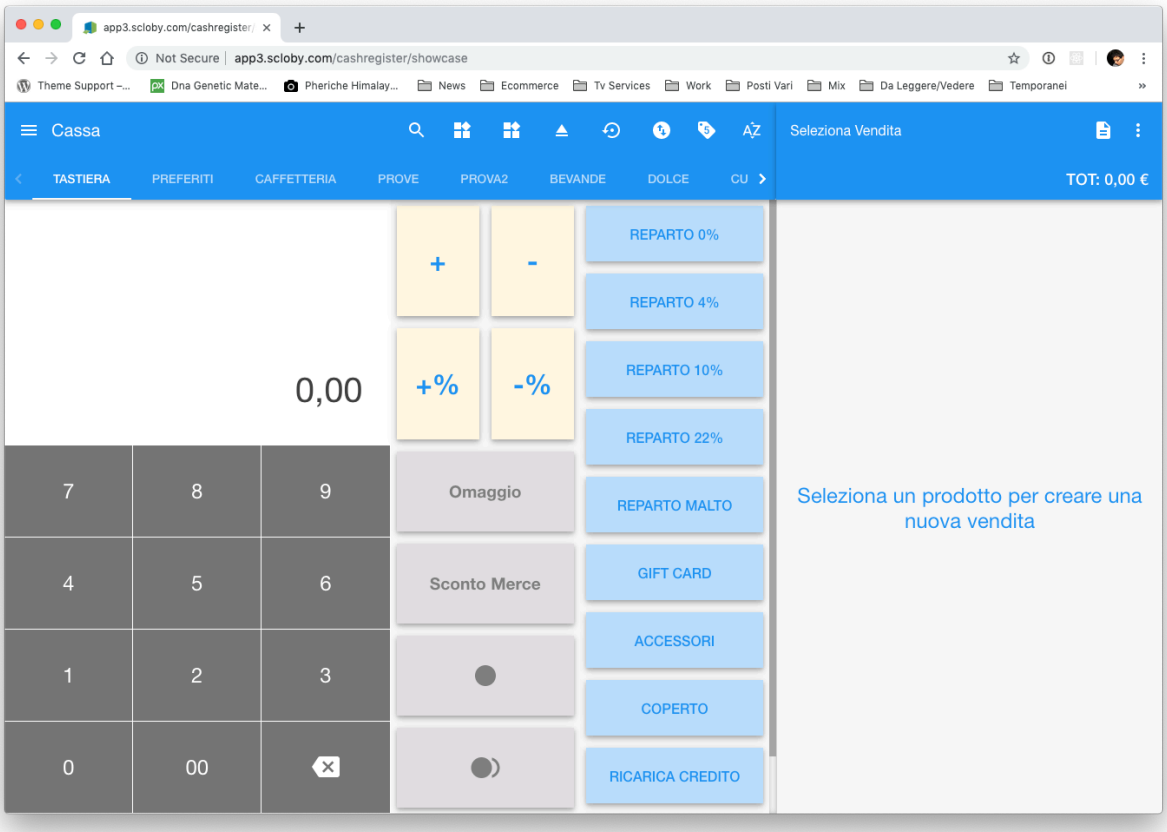
Il core business dell'azienda è il software omonimo, che consente ai professionisti che lo adoperano di poter registrare le vendite dei propri prodotti, di tenere traccia dei movimenti fiscali, delle scorte di magazzino, ed in generale di poter gestire tutti gli aspetti più importanti di cui una attività commerciale ha bisogno lato front-desk.

Da quando sono entrato in azienda nello scorso Novembre, e dopo un adeguato periodo di studio della code-base e dei tool di sviluppo utilizzati nel team, il mio compito è stato di assistere i miei colleghi senior nello sviluppo di nuove funzionalità, sia lato Front-End, sia lato Back-End.

Parallelamente mi è stato assegnato un compito di ricerca, di cui questa tesi è il frutto, con lo scopo di poter fornire al management aziendale una rosa di possibili scelte per le tecnologie future da impiegare nella prossima major release dell'applicazione.

Le tecnologie di Scloby

Scloby è una web-app giunta alla sua terza major-release. La versione 1, con cui è cominciata la fase di startup dell'azienda, era un sito web monolitico in PHP, una metodologia di progettazione ancora molto diffusa nel 2012.



LA PAGINA DI CASSA DI SCLOBY

Con l'aumento del fatturato, e l'emissione dei primi round di finanziamento, è stato possibile l'inserimento in azienda di ulteriori figure di software developer e designer, e successivamente si è proceduto a sviluppare una seconda versione, che utilizzava per la prima volta una architettura front-end + back-end con comunicazione attraverso REST Api.

L'attuale versione, la terza, ha poi rappresentato un deciso step evolutivo con l'utilizzo di Angular.js per realizzare una single-page application, con la riscrittura totale delle api in Node.js, e con l'autorizzazione all'accesso gestita dal protocollo Oauth2, che ha permesso di aprire le api di Scloby a sviluppatori terzi con una serie di integrazioni con numerose applicazioni leader del settore.

Scloby è inoltre disponibile come app per iOS, Android, macOS e Windows, grazie all'utilizzo di Electron e Cordova, degli strumenti che permettono di incapsulare dentro un app nativa una Single Page Application, dando anche la possibilità di usare al-

cune api native del dispositivo, quali notifiche push o accesso alle risorse hardware. Al netto delle limitazioni di performance questo sistema fa si che Scloby sia presente sui maggiori store di applicazioni del mercato, rendendo più facile vendere il software da parte dei colleghi del commerciale.

Angular.js

Nel 2015 gli sviluppatori di Scloby fece la scelta di utilizzare come framework di front-end **Angular.js**, una libreria sviluppata da Google per poter creare Single Page Applications con HTML+CSS+JS, utilizzando una architettura Model-View-Controller o Model-View-ViewModel.

Angular.js era, al momento del suo rilascio, una novità nel panorama di sviluppo web, ed è considerabile il “padre” di tutti i moderni framework javascript per il front-end.

Angular.js è però un prodotto figlio del suo tempo, e porta sulle sue spalle il peso di essere stato il primo player sul mercato; per questo motivo è una libreria molto pesante a livello di occupazione dati, nonostante con le successive release si sia provveduto ad ottimizzare al meglio questo aspetto, e presenta, oltre a funzionalità riprese da molti successori, come l’utilizzo di dichiarazioni e selettori html customizzati, la necessità di utilizzo di molto codice boilerplate ed una lentezza di esecuzione di animazioni e modifiche che sicuramente rendono poco appetibile il suo impegno odierno, se confrontato a molti altri framework su cui andrò a relazionare nel corso della mia tesi.

Gli obiettivi della mia ricerca

Se il problema di Angular.js fosse solo una architettura un pò “antiquata” rispetto agli standard attuali non si porrebbe comunque nessun problema per il suo utilizzo in un ambiente di produzione. I computer sono sempre più performanti, il costo della banda è comunque minimale rispetto ai costi aziendali, e quindi riscrivere da zero l’in-

tera applicazione (parliamo di decine di migliaia di righe di codice) sarebbe una spesa non immediatamente necessaria.
Il vero problema è che questo framework ha raggiunto il suo fine vita, non viene più aggiornato con nuove funzionalità, e la fine delle patch di sicurezza è dietro l'angolo, considerando i tempi che occorrono per realizzare e testare una nuova major release, nel 2021.
Inoltre utilizzare uno stack protocollare vecchio riduce il ventaglio di scelte tra le quali un software engineering può optare nel corso della pianificazione ed implementazione di nuove funzionalità, oltre che continuare ad investire su una libreria che ha già una data di scadenza è un impiego di risorse aziendali che ha sempre meno una giustificazione valida, considerando che ogni nuovo modulo di Scloby implementato in Angular.js sono migliaia di righe di javascript, css, e html da migrare su una nuova tecnologia nel futuro prossimo.

Prima di addentrarmi dentro l'analisi tecnica del futuro di Scloby vorrei parlare della mia esperienza in azienda, poiché parte integrante di questo lavoro finale sono stati i 12 crediti di tirocinio curriculare svolto nello scorso inverno. Nel corso della mia attività ho potuto in primo luogo constatare lo stato dell'arte dell'applicazione, scoprendo i pregi ed i difetti delle tecnologie impiegate, ed ho potuto analizzare quali sono i requisiti tecnologici, legislativi, e amministrativi di Scloby. Ciò mi ha permesso di poter ragionare sulle tecnologie future con cognizione di causa, cercando di trovare tra queste soluzioni che possano compensare i deficit attuali.

I requisiti tecnologici di Scloby

Il particolare mercato in cui opera Scloby obbliga a rispettare dei requisiti stringenti per quanto riguarda non solo l'interfaccia grafica, ma tutti i sottosistemi di input/output che sono necessari per fornire le funzionalità richieste.

Un punto cassa tradizionale, il classico registratore fiscale con 16 tasti accanto ad un display ed ad una stampante termica, è un prodotto totalmente diverso da una web-app che si interfaccia ad un server. Ma Scloby deve fornire esattamente le stesse funzionalità, e deve portare valore aggiunto fornendo una serie di processi per la gestione degli utenti, del magazzino, controllo degli errori (come scontrini dal valore troppo elevato) ed in generale semplificare la vita all'utente. Senza questi strumenti in più un prodotto come Scloby non avrebbe senso di esistere, in quanto è per un commerciante imparare ad utilizzare un dispositivo diverso da quelli con cui è abituato (bisogna pensare al basso tasso di informatizzazione del settore, specialmente tra i piccoli commercianti che rappresentano la più grande fetta del mercato) rappresenta un impegno aggiuntivo non indifferente, da associare ai costi di installazione e ricorrenti.

Per questo qualsiasi valutazione sulle tecnologie da impiegare deve passare da una analisi di queste necessità.

Il principale requisito del punto cassa è proprio la stampante fiscale. L'Italia si pone come un eccezione nel contesto europeo, obbligando a tutti i commercianti di utilizzare specifiche stampanti che devono essere obbligatoriamente, pena un reato amministrativo o penale, validate e fiscalizzate da operatori autorizzati.
Qualsiasi cassa che emetta uno scontrino deve avere il proprio software e scheda logica inaccessibile a qualsiasi possibile manipolazione, per evitare una registrazione delle vendita non equivalente a quella reale (che comporterebbe un minore incasso di IVA con operazioni in nero). Per fortuna non è necessario che tutto il sistema sia isolato, e quindi applicazioni come Scloby possono interfacciarsi con questi dispositivi in molteplici modi.

Poichè di solito le stampanti fiscali sono poste dentro la stessa rete locale dei dispositivi su cui è installato Scloby bisogna fare in modo di creare una connessione

diretta tra di loro. Non è possibile per i server accedere al singolo registratore, ciò implica la necessità di utilizzare il front-end di Scloby per fare da “router” delle richieste, interfacciandosi dapprima con la cassa per richiedere la chiusura di una transazione finanziaria ed i relativi dati fiscali, e poi inviando queste informazioni al server che le immagazzinerà.

La maggior parte dei fornitori di casse fiscali utilizzati da Scloby forniscono, in maniera accessibile tramite indirizzo ip, dei webservices installati sulle loro macchine, che accettano degli specifici file, normalmente xml, per ricevere comandi da eseguire sulla stampante, come la stampa di scontrini con delle righe di vendita, la richiesta di chiusura fiscale (una operazione da effettuare ad ogni chiusura del negozio per “salvare” le vendite giornaliere) oppure semplici modifiche delle impostazioni.

Un ulteriore sistema è l’utilizzo di vere e proprie socket tcp con cui instaurare una connessione tra i due dispositivi, una tipologia di accesso che sui normali browser è impossibile effettuare a causa della mancanza di apposite api javascript. Purtroppo questa modalità di accesso è richiesta su alcune stampanti, come ad esempio le “stampanti comande”, dispositivi da inserire nel contesto di una cucina in un ristorante, che ricevono automaticamente, alla chiusura di un ordine da un tavolo, i vari pasti da preparare, divisi per ordine di uscita e tempo di preparazione.

Un altro strumento spesso presente in contesti commerciali è il lettore di codice a barre, bluetooth o via cavo, che serve a poter facilmente inserire in una vendita i relativi prodotti. Anche in questo caso non esistono api per poter accedere a queste risorse da parte di un browser, ovvero a dispositivi bluetooth o driver usb, e quindi di norma si cerca di utilizzare un lettore in modalità “HID”, per poter mascherare la lettura come un semplice input da tastiera.

Tutte queste necessità dovrebbero essere prese in considerazione da parte di una nuova implementazione di Scloby. Attualmente il software possiede dei driver che consentono l'utilizzo di alcune di queste funzionalità, ma sono pezzi di codice che spesso funzionano tramite l'uso wrapper nativi, come ad esempio l'accesso alle socket, esclusivamente dentro le app basate su cordova, e non sulla versione web; c'è anche da notare l'impossibilità di lettura di codice a barre su dispositivi iOS in quando l'app iOS non può accedere ad un lettore in modalità HID. In questo modo si ha un disaccoppiamento tra le funzionalità offerte delle varie versioni del software, comportando una difficoltà commerciale, e di assistenza.

Questi problemi non sono dovuti in realtà alla tecnologia di front-end utilizzata, sarebbero presenti con una qualsiasi implementazione web di un app, ma i disagi possono essere ridotti migliorando l'interfaccia, oppure scegliendo un approccio architetturale radicalmente differente.

A livello prestazionale Scloby è una applicazione che necessità allo stato attuale di una lunga fase di startup, per poter inizializzare i contenuti con cui popolare le viste. Ho notato come questa fase di inizializzazione dura così a lungo a causa del quantitativo non indifferente di richieste HTTP effettuate ed alla grandezza del peso delle risposte, parliamo di decine e decine di file JSON ed immagini per un totale di 30-40 MB di dati. Questo step è necessario a causa della possibilità offerta dall'applicazione di poter lavorare in locale senza connessione ad internet per un massimo di 5 giorni, per garantire ad un esercizio commerciale che avesse problemi di accesso alla rete di poter comunque lavorare senza problemi, immagazzinando le modifiche in locale, che verranno poi riportate sul database remoto attraverso una sincronizzazione una volta ristabilita la connessione. Durante questo disservizio la maggior parte delle funzionalità

di Scloby, che non necessitano di una business logic remota, possono essere utilizzate, inclusa la comunicazione con la cassa fiscale.

Il lato negativo di questo sistema è il dover scaricare ad ogni reload della pagina tutti quanti i dati. Normalmente questa operazione avviene solo all'avvio del dispositivo o del browser, perchè successivamente dei sistemi di caching locale, come IndexDB, e di sincronizzazione tra risorse locali e remote, permettono di ridurre gli scambi di informazione.

Queste operazioni su connessioni lente, o su tablet dalle poche risorse di calcolo disponibili, possono durare anche 1 minuto, infastidendo notevolmente l'utente. Sarebbe necessaria una soluzione che possa fornire al user-agent una versione già renderizzata e completa dell'applicazione, garantendo lo scaricamento in background delle risorse aggiuntive; le performance sarebbero migliorative perchè anziché fornire tutto il codice javascript e tutte le risposte json prima di visualizzare la schermata iniziale dell'app, il server potrebbe comporre esclusivamente la vista necessaria all'utente al primo avvio con l'utilizzo dei soli dati richiesti da quella schermata, attraverso il rapido accesso diretto che ha con il database, piuttosto che comporre un corposo file json da inviare tramite rete. A livello funzionale non cambierebbe nulla, ma semplicemente riusciremmo a dare l'impressione all'utente di una maggiore velocità di caricamento, che nel contesto della User Experience garantisce una maggiore soddisfazione.

Dentro l'ambito della programmazione web un ulteriore requisito è il poter fornire una applicazione che sia aggiornata con le più recenti novità del W3C, come ad esempio nuovi attributi css, tag html standard, ecc… La versione attuale di Scloby presenta interfacce html costruite solo con l'utilizzo di Flex Layout, un sistema di layouting incluso in css che è sicuramente più recente e facilmente utilizzabile di quello standard a blocchi, ma le più moderne versioni dei moderni browser mettono a disposizione ulteri-

ori strumenti come ad esempio Grid Layout. In alcune viste dell'applicazione utilizziamo alcuni di questi layout alternativi ma solo attraverso l'utilizzo di polyfill, che pongono un ulteriore layer javascript tra il codice angular.js ed il DOM del browser, e sarebbe preferibile ridurre il ricorso a questi strumenti.

Altri requisiti

L'azienda è attualmente in una situazione di espansione senza però risorse adeguate, ed è in attesa di un ulteriore round di finanziamenti od un aumento di capitale, essendo ancora in fase di startup. Queste risorse dovrebbero servire per poter aumentare il numero di impiegati nel settore sviluppo, che allo stato attuale sono solo 2. Preso atto che bisogna sostituire l'attuale stack protocollare con qualcosa di più moderno ed evoluto, bisogna però far conciliare questa necessità con le esigenze finanziarie richieste dal management. Non è quindi possibile assumere ulteriore figure esterne che possano portare competenze non presenti dentro l'azienda, e quindi dovranno essere i dipendenti stessi ad essere formati nell'utilizzo delle nuove tecnologie che verranno decise dopo questa fase di analisi.

Il ridotto numero di sviluppatori è importante pure sotto l'aspetto organizzativo, in quanto non è possibile creare un team che ha come unico scopo il design e la creazione di una nuova versione dell'applicazione; tutti gli sviluppatori dovranno contemporaneamente lavorare sulla vecchia code-base e su quella futura, il che comporta sicuramente un risparmio sul computo totale degli stipendi mensili, ma riduce notevolmente la produttività. La necessità è quindi di trovare un sistema che permetta di ridurre questi rischi.

Bisogna inoltre fornire agli sviluppatori strumenti potenti per poter rapidamente creare e modificare ogni singolo componente del front-end, riducendo al minimo possibile il tasso di bug introdotti. L'azienda non possiede un team di designer, QA, un team

di maintainers o di rilascio, e non utilizza nessun sistema standard di software engineering (come Agile ad esempio). Il gruppo di sviluppo prende direttive dal management e dall'assistenza per decidere su quali task orientarsi e che modifiche effettuare, e quindi occorre usare una libreria grafica il più possibile facile da utilizzare, che garantisca rapide modifiche all'interfaccia e con a disposizione il maggior numero di widget di default, per evitare di dover impiegare ore ad implementare una funzionalità, come magari una barra di caricamento, che poi verrà bocciata prima della release.
La versione attuale utilizza la libreria Angular.js Material, il team è abituato a tutti i widget inclusi, e gli utenti all'interfaccia, e quindi dentro la futura alternativa dovrebbero essere inclusa, oppure implementabile tramite dependencies esterne, una interfaccia similare se non uguale.

A questo riguardo un ulteriore aspetto da non sottovalutare è la possibilità di modularizzare il più possibile una applicazione, cosi da poter riutilizzare in differenti parti dell'applicazione il maggior numero possibile di componenti e widget customizzati. Scloby presenta molte interfacce per la gestione di task similari che allo stato attuale non vengono molto spesso riciclate su più moduli differenti. Questa mancanza di accorpamento del codice è dovuta alla verbosità della dichiarazione di un componente in angular.js, quando in realtà basterebbero poche righe html.

Le tecnologie da me valutate

Per tutta questa serie di motivi mi è stato assegnato il compito di indagare su quali siano le possibili direzioni future, cercando i pro ed i contro, per la realtà aziendale di Scloby, delle più importanti librerie front-end presenti sul mercato.

Per fare ciò ho analizzato in primo luogo le possibilità che offrono le differenti scelte: ci sono librerie molto scarne che offrono solo uno strato di fondo su cui lo sviluppatore può costruire uno stack customizzato e plasmato sulle sue necessità; mentre altre, come lo stesso Angular.js, offrono al proprio interno la maggioranza dei tool necessari alla creazione di una app completa, ma che allo stesso tempo rendono più difficile esprimere la creatività dello sviluppatore e verbosa la creazione di un app.

Successivamente ho valutato, attraverso dei benchmark pubblici ed una lettura di riviste tecniche, le performance delle varie librerie, in quanto nonostante le prestazioni siano un aspetto marginale su un ambiente web desktop, molti clienti possiedono come loro punti di accesso a Scloby vecchi terminali Android, e quindi una nuova release dell'app dovrà fornire quantomeno una esperienza uguale, se non migliore, a questi clienti.

Infine ho indagato su possibili scelte che vadano oltre allo sviluppo di un app web-based, analizzando moderne tecnologie che permettono di compilare app cross-platform native partendo da una code-base unificata in javascript o in linguaggi di programmazione veri e propri.

La mia esperienza in Scloby

La Fatturazione Elettronica

Sono entrato in azienda in uno dei momenti più importanti della sua storia, durante una fase di espansione e crescita dovuta anche ai cambiamenti legislativi in corso in Italia in questi mesi, per quanto riguarda la fiscalità delle vendite di prodotti e servizi. Dal 1 Gennaio 2019 in Italia è obbligatorio, per la quasi totalità delle imprese e dei professionisti, l'utilizzo della Fatturazione Elettronica. Il lavoro aziendale, durante il mio periodo di formazione iniziale, è stato tutto concentro nell'implementare i protocolli dell'agenzia delle entrate per poter rendere il nostro software compatibile.

È stata sicuramente un esperienza stimolante che mi ha subito fatto entrare dentro le dinamiche di lavoro, poiché non sono stato messo in un angolo a fare lavori di contorno, ma mi sono principalmente occupato di realizzare l'interfaccia front-end che era richiesta per il nuovo aggiornamento.

Il sistema di fatturazione elettronica nasce dall'esigenza di poter tracciare tutte le transazioni che avvengono tra persone fisiche e giuridiche in possesso di una partita iva. Prima di questo sistema ogni azienda utilizzava un proprio metodo di notifica e di creazione delle fatture, con un ampio ricorso a sistemi cartacei, il che comportava una notevole facilità nel nascondere transazioni in nero e quindi ad una maggiore difficoltà di controllo dell'entrate dovute al gettito fiscale da parte dello stato italiano.

Sono due gli obblighi di legge da soddisfare nell'ambito della fatturazione eletronica: per prima cosa bisogna inviare un file xml, con un formato standard progettato dall'agenzia delle entrate, ad un "sistema di interscambio", ovvero un webservice fornito dall'agenzia che permette di validare la fattura e far si che il fatturante possa inviarla al ricevente.

In secondo luogo bisogna effettuare la conservazione sostitutiva di queste fatture. Come ogni documento legale redatto da un'azienda esistono infatti degli obblighi per quanto riguarda l'archiviazione degli stessi, per poter permettere, entro un certo lasso di tempo stabilito dalla legge, di poter verificare tutte le operazioni aziendali, da parte di agenzie governative o giudiziarie.

Per conservazione sostitutiva di una fattura, o di un altro documento, si intende una procedura informatica che conferisce valore legale nel tempo a un documento informatico. Questa procedura permette di equiparare la validità di questo file a quella di una normale archiviazione classica, che tutti gli uffici effettuano regolarmente nei propri archivi. Il vantaggio è che non avendo un oggetto fisico e materiale c'è meno rischio di furto, scomparsa, e di perdita dell'informazione. Lo svantaggio è che bisogna stare attenti a salvare un file valido, non cancellarlo, ed essendo un file con una firma digitale bisogna ri-firmare il dato alla scadenza della validità del certificato.

Per poter creare una di queste fatture elettroniche c'è quindi tutta una sequenza di operazioni da effettuare:

1. Bisogna prima creare l'xml vero e proprio della fattura, secondo le specifiche
2. Bisogna apporre una firma digitale qualificata al documento
3. Indicizzare il documento associando i dati del fatturando per semplificare le ricerche in futuro, analogamente alla catalogazione degli archivi cartacei
4. Inserire il documento dentro un lotto di conservazione, che ne garantisce l'integrità sequenziale e l'ordinamento per data
5. Il responsabile della conservazione sostitutiva applica un TDT (Marca temporale) e firma il lotto

I dati cosi elaborati devono essere conservati, per quanto riguardo la fatturazione elettronica, per un tempo di 10 anni.

Tutto questo processo però è complicato da gestire per una azienda o professionista, specialmente se dal basso fatturato, sia per mancanza di risorse, ma sopratutto poiché si prende essa stessa delle responsabilità legali che potrebbe facilmente delegare ad altri.
Per questo motivo la maggioranza delle fatture elettroniche vengono emesse, e conservate, in maniera indiretta attraverso un intermediario, che fa le veci del fatturatore in tutto e per tutto.

Scloby in questo contesto si pone come un ulteriore figura, ovvero come terzo intermediario, un soggetto che per conto del fatturatore genera l'xml e lo invia ad un ulteriore intermediario che sarà colui il quale effettuerà tutte le operazioni necessario per la validazione ed invio del documento.

Abbiamo quindi costruito lato back-end un ulteriore webservice, facente parte della rete privata interna di Scloby su AWS, che permette di accedere a tutti i servizi di Scloby ed alle varie API e Database, che costantemente va a fare un pooling dei dati sui server dell'intermediario per poter ricavare nuove informazioni riguardo alle fatture emesse. Una volta riscontrato un aggiornamento verranno salvati i dati sul database , nei record relativi a quella vendita.

Questo è un sistema totalmente inefficiente, come tutti i sistemi di notifica tramite pool, ma che purtroppo è vincolato alle funzionalità del nostro partner. Ci sono state molte discussioni in azienda sul sostituire, o aggiungere, nuovi fornitori di questi servizi, che potessero offrire api moderne. Un sistema sicuramente migliore potrebbe essere basato su dei Webhooks con uno specifico formato per la callback, un sistema ampiamente utilizzato da fornitori di servizi da integrare dentro un prodotto web, ma purtroppo dopo una analisi delle realtà presenti sul mercato italiano è risultato un prodotto inesistente.

Sono infatti pochi i player nel settore della creazione, crittografia, e storage di dati amministrativi in Italia. I leader del mercato sono Aruba e Infocert, e la maggioranza degli altri provider si appoggiano semplicemente alle api da loro fornite, come un proxy web però applicato a dei servizi, inserendo ovviamente un ricarico sul costo del servizio. Purtroppo la fattura elettronica è un prodotto esclusivamente italiano, perlomeno in questa forma cosi codificata, poiché esistono direttive europee a riguardo ma che l'Italia ha trasferito nella sua legislazione recependole in maniera differente, e quindi ci si può appoggiare solo ad aziende italiane per ottenere queste funzionalità, aziende che spesso tendono a seguire paradigmi vecchi nella creazione dei loro servizi.

Fornire API di qualità, funzionali, e moderne è sicuramente uno degli obiettivi di Scloby, e per questo consiglierei a molte delle aziende partner che utilizziamo in azienda di adeguarsi alle novità del settore.

Il Front-end delle fatture

Non potendo quindi avere una gestione diretta di queste interfacce di back-end è di fondamentale importanza fornire all'utente un front-end che sia il più possibile chiaro ed a prova di errori, poiché tutti i dati del documento inviato devono essere corretti, altrimenti l'agenzia delle entrante non accetterà la fattura. Questi errori possono include valori non validi sia a causa di una formattazione del documento sbagliata, sia a causa di veri e propri errori di battitura da parte dell'utilizzatore del software; inoltre bisogna considerare il fatto che un cliente, di cui l'esercente prende male i dati di fatturazione, potrebbe essere già lontano dall'esercizio commerciale una volta che l'errore viene scoperto e notificato. In tal caso se l'esercente non riesce a modificare il documento commetterà un illecito.

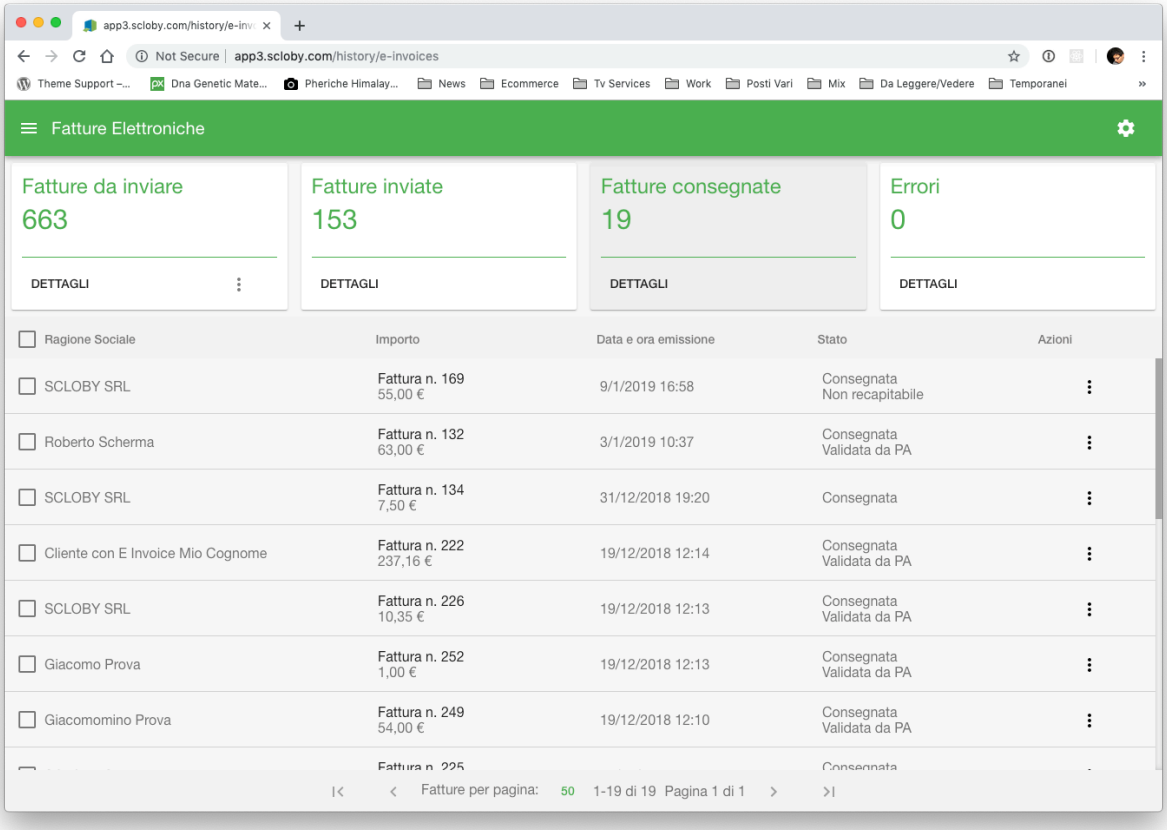
Essendo la Fattura Elettronica un normale xml che è conferme ad un xsl fornito dall'agenzia, un controllo sul documento prima dell'invio potrebbe sembrare di sem-

plice esecuzione. In realtà ci sono una serie di dati di cui non è possibile indicare un pattern preciso nelle regole di validazione xml; degli esempi sono il nome di un comune, un cap, o la partita iva stessa, che possono essere validi a livello "grammaticale" ma non nella realtà dei fatti.

Ogni singolo errore di invio al webservice dell'agenzia non è immediatamente noto, poiché ci sono dei tempi tecnici di elaborazione dei dati da parte dei processi di validazione sui loro server, in aggiunta ai tempi di invio da parte dell'intermediario, che possono rendere le risposte non contestuali alla chiusura di una vendita in cassa.
Inoltre non è detto che il dispositivo sia dotato costantemente di connessione internet, cosi come i server, i disservizi devono sempre essere tenuti in considerazione.
Per questo motivo non è obbligatoria l'elaborazione contestuale del documento all'atto della vendita, ma bisogna tassativamente inviarlo entro 14 giorni al sistema di interscambio, nella forma corretta e definitiva.

Per garantire tutte queste funzionalità e protezioni è necessaria un'ottima User Experience, che cerchi di limitare il numero di errori da parte dell'utilizzatore; ed è stato questo il compito principale da me svolto in questo ambito.

Per prima cosa ho dovuto realizzare, o modificare, tutta una serie schermate che si occupano delle anagrafiche clienti, poiché le versioni presenti in Scloby erano molto improntate o alla gestione lato marketing del cliente (ovvero il poter conoscere i dati di contatto o il suo storico degli acquisti), oppure ai dati necessari all'emissione della tradizionale fattura cartacea; questi dati sono però solo un sub-set di quelli necessari per l'invio telematico delle fatture. Inoltre non era inclusa nessuna validazione di quelle informazioni, poiché per la fattura cartacea non esisteva un formato predefinito e standard.



LA LISTA DELLE FATTURE ELETTRONICHE

Durante questo progetto ho potuto constatare come sia tediosa e ripetitiva la creazione, la gestione, e la validazione dei form in angular.js

Ho provato ad usare un approccio “data-driven” per la generazione di questi form, ma molto spesso il risultato si è rivelato inconsistente e con una serie di eccezioni, di non facile debugging, lanciate dal framework. Nel debuggare questi problemi mi sono reso conto di come era evidente che ci fosse una cancellazione di alcune variabili dello scope, e quindi durante l’iterazione dell’array contenente i dati di generazione del form alcuni valori degli oggetti risultavano null o undefined, quando in realtà erano tutti campi valorizzati partendo da un json remoto corretto.

Ho quindi riscontrato uno dei piu grandi problemi, a mio modo di vedere, di Angular.js ovvero la relazione tra ng-if e lo scope del controllore. Ng-If permette di poter inserire e togliere dall’albero DOM interi pezzi di codice, basandosi sul pattern che

questa direttiva accetta come valore. Il problema sostanziale è che ng-if distrugge anche lo scope relativo a quel elemento se viene rimosso, e uno nuovo viene ricreato quando avviene il restore della vista.

Questo scope viene ricreato partendo da quello padre, quello del controllore stesso, utilizzando “l’ereditarietà dei prototipi”; se si utilizza ng-model all’interno di un ng-if le modifiche effettuate nell’input si ripercuoteranno sullo scope del padre, e quindi bisogna necessariamente utilizzare degli oggetti innestati per operare sullo scope del padre, pena il reset di tutte le modifiche.

Questo problema di conflitto tra gli scope pone problemi nel riutilizzo di piccoli componenti in parti diverse una vista più grande, e può essere abbastanza problematico, anche in presenza di direttive come ng-switch che utilizzato anch’esse lo stesso approccio di ng-if. Serve necessariamente una gestione migliore dello stato e delle variabili, strumenti che mancano nel framework.

Per risolvere la questione relativa al form ho comunque scelto invece di utilizzare ng-show al posto di ng-if, che banalmente al posto di cancellare un intero pezzo di DOM dall’albero pone il suo valore css di display a “none”, nascondendolo dalla vista. Questo è un approccio più semplice ed immediato, e può andare bene in viste dove la sicurezza non è prioritaria, poiché non rimuovendo i campi dal DOM è possibile manipolare comunque gli input, e bisogna stare attenti a non lasciare campi nascosti che non dovrebbero essere accessibili all’utente in un determinato stato. Per fortuna la maggior parte dei form in Scloby non hanno questa esigenza poiché la sicurezza è implementata profondamente lato back-end, ed un utente deve essere autorizzato per effettuare modifiche a campi e risorse specifiche.

Una parte sicuramente più semplice è stata la creazione di un modulo completo di Scloby per la gestione successiva all’emissione delle fatture, poiché queste devono

poter essere visionate, e deve essere possibile recepire lo status attuale di una di esse presso l'elaborazione dell'agenzia delle entrate, che per fortuna risponde con informazioni abbastanza chiare e puntuali su ogni possibile errore.

Un algoritmo da me sviluppato

Successivamente a questo lavoro sulla fatturazione elettronica, mi è stato chiesto di migliorare le prestazioni di alcuni moduli di Scloby. Mi sono immediatamente reso conto che moltissime schermate, ed api lato server, fossero state realizzate senza tenere in considerazione la crescita del volume dei dati da dover visualizzare e fornire. In molti contesti, il software scaricava totalmente in locale, ad ogni caricamento di un modulo, l'intero data-set necessario a visualizzare delle informazioni: un esempio sono le tabelle di riepilogo dei clienti, che venivano pre-caricate con tutti i clienti, ed in contesti con 10.000 tessere fedeltà significava tempi di caricamento notevoli ad ogni utilizzo.

Sicuramente questo approccio rende molto facile poter effettuare ricerche approfondite e complesse in un dataset, poiché esistono innumerevoli plugin javascript che prendendo come parametro un array json, ed una semplice stringa, possono effettuare ricerche approfondite dentro tutto l'albero.
Questo però limita moltissimo le prestazioni, ed è quindi un approccio non scalabile in maniera indefinita. Il metodo più corretto è quello di effettuare una paginazione dei dati lato server, con possibilità di filtro per alcuni campi dei modelli.

Ho quindi implementato queste funzionalità, inserendo nella maggior parte dei moduli di Scloby dei sistemi di paginazione, e di ricerca dei campi più utilizzati.
Mi sono accorto di una particolarità del backend di Scloby, a mio parere abbastanza lungimirante, di avere dei controllori dell'accesso ai modelli del db unici per tutte le di-

verse entità, grazie anche all'utilizzo di un ORM chiamato Sequelize che semplifica di molto alcune procedure.

Abbiamo riscontrato però un problema quando l'assistenza clienti ha cominciato a ricevere un notevole numero di ticket riguardanti la ricerca, ed abbiamo capito come i nostri clienti fossero ormai abituato ad effettuare semplici ricerche testuali, non divise per campi. Normalmente nel nostro software bastava digitare un nome e venivano restituiti lato front-end sia tutti i clienti che avevano un nome contenente quella stringa, sia anche tutte le aziende che avessero un nome similare. In una azienda di servizi come la nostra bisogna spesso adeguarsi alle richieste che provengono dal basso, poiché in un settore altamente competitivo è necessario non scontentare il cliente, che spesso ha un grado di alfabetizzazione informatica non elevato, e che quindi deve essere guidato passo passo nella formazione all'utilizzo del software; cambiamenti cosi radicali all'interfaccia possono essere destabilizzanti.

Dopo uno studio approfondito del problema mi sono reso conto che era necessario implementare due funzionalità nella ricerca di Scloby. In primo luogo bisognava garantire la ricerca nei sotto-modelli di Scloby. Infatti possedendo un albero delle entità abbastanza ramificato, i modelli principali dei dati contenuti nei database contengono molte relazione e sotto-relazioni; la fattura elettronica, per esempio, è modernizzata come un oggetto correlato alla vendita, ma è spesso necessario filtrare tutte le vendite con fatture rigettate dai vari web-service per fare in modo di poter modificare alcuni dati.

In secondo luogo bisognava poter effettuare queste ricerche con l'utilizzo di operatori logici. Un esempio di query necessaria è il dover ricavare tutte le vendite chiuse nel quale nome del cliente sia contenuta una certa stringa.

Questa operazione in linguaggio SQL potrebbe essere cosi scritta (notare come la sotto entità customer sia una tabella dentro una join)

```
status=closed AND (customer.nome LIKE Pippo OR customer.cognome
LIKE Pippo)
```

Un sistema cosi fatto non è di facile implementazione con delle api REST, poichè è possibile includere nella parte di query dell'url solo coppie chiave valore. Bisognerebbe inviare una richiesta post con un json, formattato in una certa maniera, come corpo della richiesta da poter elaborare server-side. Ed è questo l'approccio che per esempio utilizzano GraphQL o gli endpoint di un db mongo. Ma Scloby opera con una architettura rest, quindi non è un approccio viabile, anche a causa del sistema di caching delle richieste get presente nelle nostre api.

Ho quindi avuto l'idea di cercare un sistema di querying complesso per le api rest, ma devo dire che non ho trovato grandi riscontri a riguardo. Ho quindi proposto di creare un "linguaggio" di querying da me sviluppato, forte anche dello studio dei linguaggi regex e dei sistemi di compilazione effettuati nel corso del mio percorso di studi al politecnico. Con l'approvazione del CTO ho cominciato ad approfondire il discorso e sono arrivato ad una implementazione che consente oggi agli utilizzatori delle API di Scloby di poter effettuare lunghe e complesse query.

A livello implementativo ho dapprima creato un parser di questo mio linguaggio da me creato, un sistema semplice che permette di annidare funzioni di AND o OR racchiuse tra parentesi e divise da virgole. La stessa query di prima nel mio linguaggio è trasformata in

```
AND(status=closed,OR(customer.nome_like=Pippo,customer.cognome_-
like=Pippo)
```

Questo albero viene generato partendo dai dati presenti nella query-string per la richiesta HTTP agli endpoint supportati. Questa query-string è composta da due gerarchie di filtri.

La prima è una normale ricerca divisa per campi, che permette di filtrare tutti i record ponendo questi filtri in AND tra di loro. Questi filtri possono contenere i seguenti suffissi: _like _since _max; questi suffissi verranno trasformati in query SQL che al posto di avere una relazione di = useranno LIKE, oppure relazioni numeriche > o <. Senza l'utilizzo di queste variazioni non sarebbe possibile fare una ricerca per nome efficace, le maiuscole e minuscole verrebbero considerate a se stanti, o ricerca per range di date o di valore numerico (molto importante quando si vanno ad analizzare delle risorse come delle vendite).

La seconda gerarchia è composta da una chiave **x_query** (complex_query) della query-string che ha come valore una stringa che verrà poi parsata per poter comporre un similare albero, che però in questo caso non sarà solo tra valori messi in AND tra di loro sullo stesso livello, ma che potrà contenere una serie di N livelli che via via potranno selezionare piu nel dettaglio il data-set richiesto.

Il parser funziona analizzando la stringa, che è un dato lineare e fondamentalmente un array di caratteri, utilizzando delle operazioni regex per identificare l'operazione logica più ad alto livello, e creare un oggetto che contiene un riferimento alla sua tipologia, AND o OR, ed alla stringa estrapolata internamente, delimitata dalle due parentesi più esterne.

Il parser continua la sua esecuzione in maniera ricorsiva andando a creare dentro questo oggetto, partendo dalla stringa estrapolata nello step precedente, tutte quante le stringhe più interne, identificando per ogni livello i token relativi all'attributo da ricercare ed il valore di ricerca. Per ogni livello si possono avere oggetti di tipo AND, OR, o

VALUE. I valori VALUE verranno confrontanti in maniera logica, utilizzando la funzione di quel livello, con quelli provenienti dagli oggetti AND o OR suoi pari.

Una volta completata questa struttura, avviene una analisi bottom-up di questo albero, per poter trovare per ogni livello tutti gli id che soddisfano la relativa relazione, utilizzando poi per salire al livello successivo la relazione AND o OR indicata, fino a arrivare al nodo principale. Ogni livello viene trasformato in operatori che seguendo la sintassi dell'ORM Sequelize vengono dati come input all'entrypoint delle query sql per poter ricavare i dati necessari.

Questo sistema di x_query si integra con i filtri dichiarati dall'altra gerarchia ponendo in AND l'array contenente gli ID trovati dalla query con quelli trovati dai singoli filtri, per poter cosi ricavare l'array finale che verrà utilizzato dal sistema di ricerca per poter fare un ulteriore scrematura utilizzando sistemi di paging o di ordinamento.

I campi da filtrare possono anche essere non facenti parte della risorsa richiesta, ma anche di una sottotentità, sia per le relazioni 1-1 sia per quelle 1-N.

Devo dire che sono stato molto aiutato da quel sistema di controller univoco per tutti gli endpoint delle entità, poichè è bastato implementare solo in un unico middleware questa routine per vederla funzionare ovunque in Scloby, ed a quel punto l'unico step aggiuntivo richiesto è stato modificare le richieste http dentro l'app per ottenere queste funzionalità avanzate ovunque.
Adesso la maggior parte dei campi di ricerca nei moduli di Scloby sono unici ed utilizzano questo sistema, e per ogni modello sono stati stabiliti dei campi dove effettuare queste ricerche in maniera automatica, in base alle esigenze che hanno notato il servizio clienti ed il QA.

Le prestazioni di questo sistema non si sono rivelate terribili, anzi secondo una stima da noi effettuata abbiamo avuto solo un impercettibile leggero rallentamento delle

risposte, ma abbiamo aumentato le performance nella maggior parte delle schermate eliminando il caricamento totale dei dati. Inoltre facendo tuning del db con degli indici, e eliminando dentro le ricerche degli id tutte le colonne in risposta tranne gli id stessi abbiamo visto assieme al CTO come le risposte alle query sql interne siano praticamente immediate.

La manutenzione e creazione di tool interni

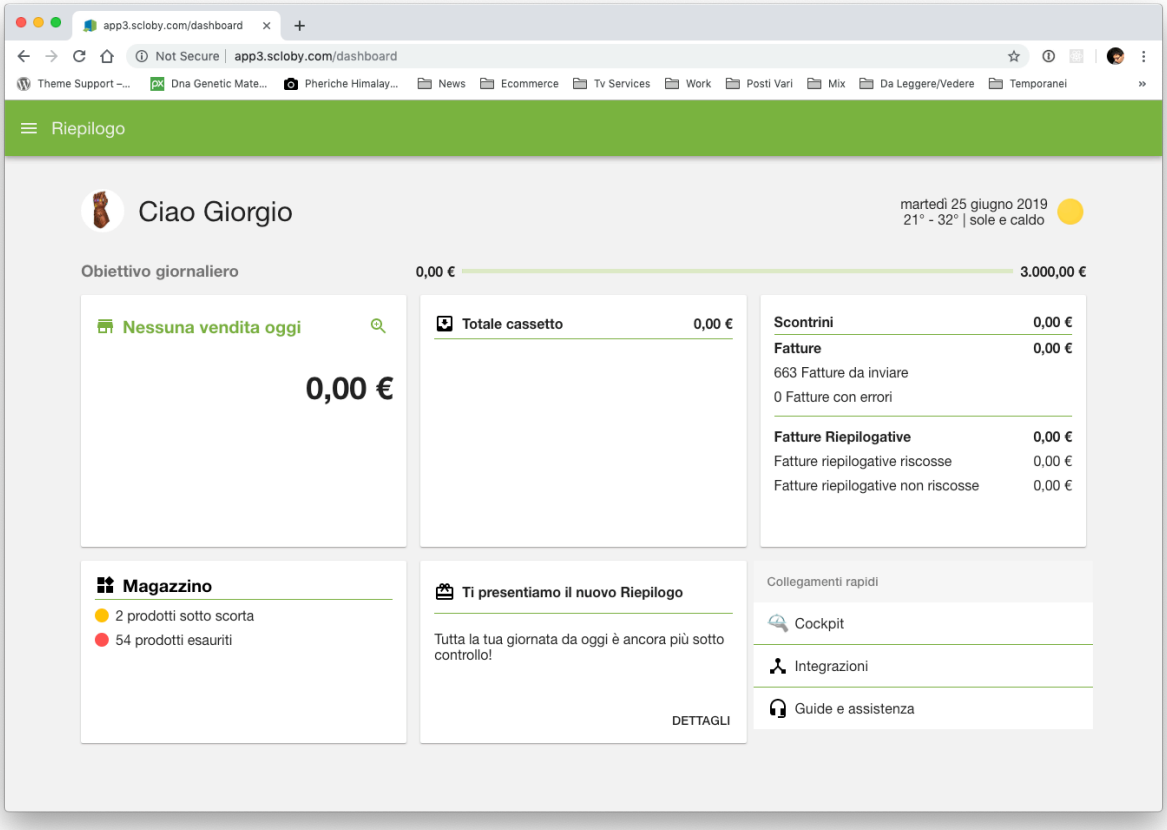
Essendo un team piccolo è necessario che tutti collaborino a manutentore e creare tutti software che garantiscono la continuità aziendale. Non stiamo parlando solo di Scloby l'app, ma anche di una serie di tool interni ed esterni che vanno venduti o che servono a facilitare il lavoro di altre business unit della società.

Scloby non vende solo il suo prodotto principale, esistono infatti ulteriori app per poter visualizzare gli ordini presso una cucina o per fare una analisi più dettagliata delle vendite, ed una serie di integrazioni e connettori per gestionali di terze parti. Sono stati inoltre creati, nel corso degli anni, molti strumenti aggiuntivi per garantire all'assistenza di svolgere una serie di operazioni senza interfacciarsi con il database ed il team di sviluppo, bilanciando il carico di lavoro. Stiamo parlando per esempio di sistemi per attivare nuove funzionalità, algoritmi di conversione di listini da csv al formato interno di Scloby, e cosi via.

La maggior parte di queste app sono anch'esse scritte in Angular.js con l'utilizzo sia delle api pubbliche, sia di api private ed interne per effettuare alcune operazioni critiche.

Sono questi i tool che sicuramente saranno presi in considerazione per effettuare test di conversione o riscrittura con le future tecnologie, poichè sono progetti più piccoli e meno critici dal punto di vista della operatività.

In questi mesi ho quindi anche lavorato in questo settore di sviluppo, e sono riuscito a fare esperienza con richieste “diverse” da quelle di un normale cliente. Sono stato incaricato per esempio di scrivere un intero front-end dedicato alla gestione dei clienti e delle fatturazioni automatiche, cosi da poter ridurre il carico di lavoro che mensilmente l’amministrazione ha sul finire del mese.



UNA PAGINA DI DASHBOARD DA ME REALIZZATA

Mi sono reso conto di come in una azienda sia necessario curare in maniera ottima l’UX sia del proprio prodotto, sia di tutto ciò che sta intorno a quel processo di vendita, poichè un rallentamento delle operazioni, dovuto a bug o UX complicate e poco usabili nelle app a disposizione dei reparti commerciali ed assistenza, si può ripercuotere inevitabilmente su tutta l’operatività aziendale. Scrivere un buon codice, e realizzare qualcosa di gratificante a livello estetico e funzionale non può essere mai messo da parte, non si possono prendere scorciatoie in nessun caso solo perchè apparentemente “non si fanno soldi” su quel prodotto, tutto fa parte di un processo più

complesso, ed ogni utilizzatore di ogni software ha sempre delle necessità da soddisfare.

Angular

Sono tantissimi i framework esistenti, ma ho deciso di restringere la ricerca solamente ai due maggiori player del settore, Angular e React.
Il problema maggiore delle altre soluzioni è il limitato numero di developer ed utilizzatori. Quando si studia una scelta tecnologica bisogna sempre ponderare i possibili costi futuri dovuti ad una mancanza di supporto od a una deprecazione precoce del software scelto. Scegliere un sistema open-source sicuramente riduce i problemi in questi casi, poiché si può sempre accedere al codice e risolvere da se i bug, ma non tutte le aziende possono continuare a mantenere un codice da sole; per questo puntare su una libreria o framework che abbia un notevole numero di contributori al codice, specialmente se tra essi sono incluse grosse aziende del settore IT, è una scelta più ponderata. Per questo motivo ritengo che le altre possibili soluzioni sono a grosso rischio futuro, poiché il mercato del software tende sempre a convergere verso una serie di soluzioni standard invece di avere un grosso numero di servizi poco utilizzati.

La Versione 2 di Angular

Il primo framework da dover analizzare è ovviamente Angular. Successore della versione 1, rinominata poi Angular.js, che è attualmente in uso in Scloby, Angular fornisce alcuni strumenti e pattern che non stravolgono le fondamenta architetturali rispetto alla versione precedente, ma espone delle api diverse, ed è per questo motivo che non è trascendentale poter effettuare una conversione da Angular.js a Angular 2+.
Si può considerare Angular.js come una semplice versione beta del framework attuale, nel quale sono stati testati tutti i paradigmi poi riutilizzati successivamente

In superficie la differenza principale tra i due librerie è il linguaggio con cui devono essere utilizzate. Infatti durante la progettazione del framework Google ha scelto

di optare per un superset di javascript per la sua realizzazione, ed il suo utilizzo come libreria. Questo linguaggio è **Typescript**. Attenzione però, Angular può essere utilizzato anche semplicemente con Javascript puro, ma la maggioranza della documentazione, del supporto, e delle best-practice sono studiate apposta per questo superset, ed è altamente consigliato il suo utilizzo.

Typescript

Typescript è, semplificando, una versione di Javascript ES6 modificata con l'introduzione della tipizzazione delle variabili e delle funzioni. È sviluppato e mantenuto da Microsoft, che ha creato una toolchain di sviluppo perfettamente integrata dentro i suoi prodotti, come Visual Studio Code, l'editor di sviluppo attualmente di maggior successo nel campo della programmazione web[1].

La sua nascita è deriva dalla necessità di applicare alcuni concetti della tradizionale programmazione all'utilizzo di un linguaggio di scripting come Javascript, che è attualmente l'unico standard possibile per realizzare applicazioni interattive web based (WebAssembly è ancora oggi un sistema in sviluppo).

L'errore in javascript è un qualcosa di ubiquito, è difficile realizzare anche solo 20-30 righe di codice senza incappare in un problema di sintassi, od in un typo, errori che possono comportare l'introduzione di profondi bug, con notevole difficoltà di debugging. È possibile ridurre i rischi utilizzando strumenti di linting in tempo reale del codice, come EsLint, ma hanno profonde limitazioni.

Typescript essendo un superset va "compilato" prima di poter funzionare sui comuni browser, ed è proprio questa sua caratteristica che permette anche di effettuare approfonditi controlli sul codice, eliminando tutta quella serie di errori di sintassi che un normale linguaggio di programmazione, come C o Java, non presenta.

---

[1] https://github.com/microsoft/TypeScript

Oltre a ciò il superset di Typescript include diverse funzionalità, ed estensioni, che rendono il codice scritto meno dipendente da boilerplate, hacking, e garantisce la possibilità di usare tutta una serie di paradigmi di programmazione non presenti in ES6 (come una gestione migliorata delle classi).

Migrare una codebase a typescript non è però una operazione semplice, poiché ci sono molte supposizioni e “semplificazioni” di javascript che vengono ovviamente rimosse da una forte tipizzazione.
Si deve considerare ogni var (da dover modificare in let o const, i tipi di assegnazione in typescript, ed incidentalmente di ES6) di un codice javascript come un attributo di tipo “any”. Ma cosi facendo non si possono utilizzare a pieno le funzionalità offerte del set di istruzioni, ed è quindi necessario effettuare un lungo e tedioso processo di rifattorizzazione del codice, anche perché bisogna re-implementare i moduli dentro a delle classi; typescript inoltre non concedete la possibilità di creare View e Controller in un unico file.

Essenzialmente Typescript è uno strumento potente, che però rende più complicato il lavoro di porting di una codebase, un lavoro che può ovviamente introdurre errori e bug di logica dentro le librerie, e quindi va considerato uno step significativo nel processo di selezione (e successivamente di porting) di una tecnologia per una web-app[2].

Le differenze non si fermano a Typescript

Se il problema fosse solo quello del linguaggio da utilizzare non ci sarebbero grandi problemi a passare alla versione 2 di Angular, partendo dall’attuale Angular.js, in un contesto come Scloby. Ma questa migrazione non è solo a livello di sintassi ma anche e sopratutto di organizzazione del codice e di funzionalità fornite[3].

---

[2] https://hashnode.com/post/how-we-migrated-a-200k-loc-project-to-typescript-and-survived-to-tell-the-story-ciyzhikcc0001y253w00n11yb

[3] https://angular.io/guide/upgrade

Come sostengono gli sviluppatori “Angular è una versione reimmaginata di Angular.js, che utilizza le sue parti migliori, preservandole, ed evitando le cattive”.
Angular richiede tassativamente l’inserimento di un componente per singolo file, cosi come di una struttura gerarchica ad albero per i moduli. Questo permetterà, in futuro, di poter effettuare modifiche alla struttura, o anche al linguaggio stesso utilizzato (si pensi a poter usare in futuro una versione di javascript standard senza bisogno di typescript) senza dover effettuare nuovamente una riscrittura totale dell’architettura dell’applicazione. È inoltre caldamente consigliato l’utilizzo di un module loader, come Webpack, per poter gestire meglio l’enorme numero di file e risorse che comporta una struttura del genere. Angular.js dall’altro canto non obbliga a nessuna di queste sottigliezze.

L’utilizzo però in Angular.js di alcune best-practice, consigliate negli ultimi anni da parte del team di sviluppo, e di funzionalità back-ported da Angular 2 alla versione precedente, rende però questo processo di conversione più semplice. Tra queste posso citare “Component Api” che abbiamo largamente utilizzato nel codebase aziendale negli ultimi anni.
È però possibile effettuare una migrazione diretta del codice? No, ma è possibile mischiare componenti in Angular.js e Angular, utilizzando la libreria ngUpgrade.
Bisogna ovviamente partire da un codice Angular.js molto modularizzato, e possibilmente già convertito in Typescript, poichè poter mischiare moduli dei due framework in maniera praticamente intellegibile può essere di grosso aiuto alla conversione del progetto[4].

Questo processo di Hybridazione permette di poter lavorare su un unico branch senza dividere il team in due tra Mantainer della vecchia versione ed i Developer che dovranno riscriverla, e cosi facendo ogni nuovo modulo progettato dal team potrà es-

[4] https://www.monterail.com/blog/angularjs-angular-migration-hybrid

sere scritto in Angular, e nel frattempo ci si potrà anche dedicare alla conversione dei precedenti, fino ad arrivare, in maniera graduale, all'utilizzo di un solo framework.

Questo sistema di upgrade potrebbe essere di fondamentale importanza per Scloby. Come illustrato precedentemente uno dei requisiti più importanti per l'azienda è quello di impiegare in maniera efficiente e funzionale le poche risorse umane a disposizione, e poter lavorare su un unico progetto mentre si sta sviluppando una nuova versione sarebbe un notevole incentivo per l'utilizzo di Angular.

Tutto ciò però presuppone un processo di conversione da pianificare con accuratezza, e necessità una migrazione preliminare di tutto lo stack di sviluppo.
Gulp era il tool standard per al gestione, e l'hot reloading, oltre che al packaging, di un grosso progetto in Angular.js, ma allo stato attuale ci sono strumenti da riga di comando decisamente più supportati e potenti, come ad esempio Webpack.

Questo processo di modifica dei tool può però dei benefit immediati anche senza l'utilizzo di una nuova libreria di front-end, infatti presso Scloby stiamo già effettuando questa conversione, a causa dei quotidiani difetti e limitazioni che riscontriamo in Gulp.

Le differenze principali tra le due versioni

Escludendo la sintassi di Typescript sono tre le principali differenze tra Angular.js ed Angular.
Innanzitutto ogni componente, composto da un template e da una classe controller, è separato logicamente dagli altri. Vige la regola "un controller, un file".
E dentro i controller non esiste il concetto di "scope", che è quella variabile dinamica che permette di effettuare un binding bidirezionale tra template e controllore nella versione 1, che abbiamo visto comporta un costo aggiuntivo di gestione in componenti con la presenza di direttive ng-if e similari. La mancanza dello scope è inoltre sintomo

di un altro profondo cambiamento in Angular, ovvero la mancanza di bidirezionalità delle variabili[5].

Per la realizzazione di un form in Angular.js basta associare ad un modello ad un campo input, facente parte dello scope, ed ad ogni cambiamento dello scope quel campo verrà ricaricato con il nuovo valore, ed ad ogni nuovo input dell'utente sarà lo scope ad essere modificato. Tutto ciò facilità la gestione dell'input output, ed in fondo angular in origine era un framework molto piccolo e limitato che sarebbe dovuto essere appannaggio dei soli designer, non degli sviluppatori; ma dall'altro comporta una frequenza abbastanza elevata di reload dello stato dell'applicazione.
Questo è uno dei motivi per il quale in Angular.js molto spesso le animazioni, e le modifiche dello stato, sono lente, perché teoricamente ogni singola pressione di un tasto della tastiera può scatenare decine e decine di operazioni di apply dello scope. Inoltre ciò comporta uno stretto accoppiamento tra View e Model, mentre in realtà la view dovrebbe essere esclusivamente un prodotto del modello, nell'architettura MVVC.

In Angular di default il binding è sempre one-way, dal controllore-modello al template, questo da un lato aumenta la verbosità il codice per aggiornare il modello, ma dall'altro rende un componente più stabile, performante, ed anche sicuro. Ci sono comunque dei moduli inclusi nel framework, come angular-forms, che permettono di fornire un two-way binding in questi casi specifici, ma son l'eccezione, non la regola.

La terza maggiore novità in Angular è l'utilizzo di un sistema di dependency injection, un pattern molto diffuso che è presente in molti framework di front e backend,

---

5 https://www.quora.com/Why-is-the-two-way-data-binding-being-dropped-in-Angular-2-If-Angular-2-0-only-supports-1-way-data-binding-wasnt-2-way-binding-a-major-feature-of-Angular-1-Does-this-imply-that-somehow-2-way-binding-in-Angular-1-wasnt-a-useful-feature/answer/Aaron-Martin-Colby

come JavaSpring.

In Angular.js era già possibile effettuare l'injection di un servizio dentro un controller, ma la versione 2 permette l'iniezione di un modulo già configurato ed inizializzato, ed introduce anche la possibilità di effettuare uno scoping in modo tale che solo alcuni controllori possano accedere ad un determinato servizio.

Il compilatore di Angular

Ma esattamente quindi come funziona Angular? Al contrario della versione uno questo framework permette funzionalità più avanzate e l'utilizzo di Typescript poiché c'è come detto una fase di compilazione [6].

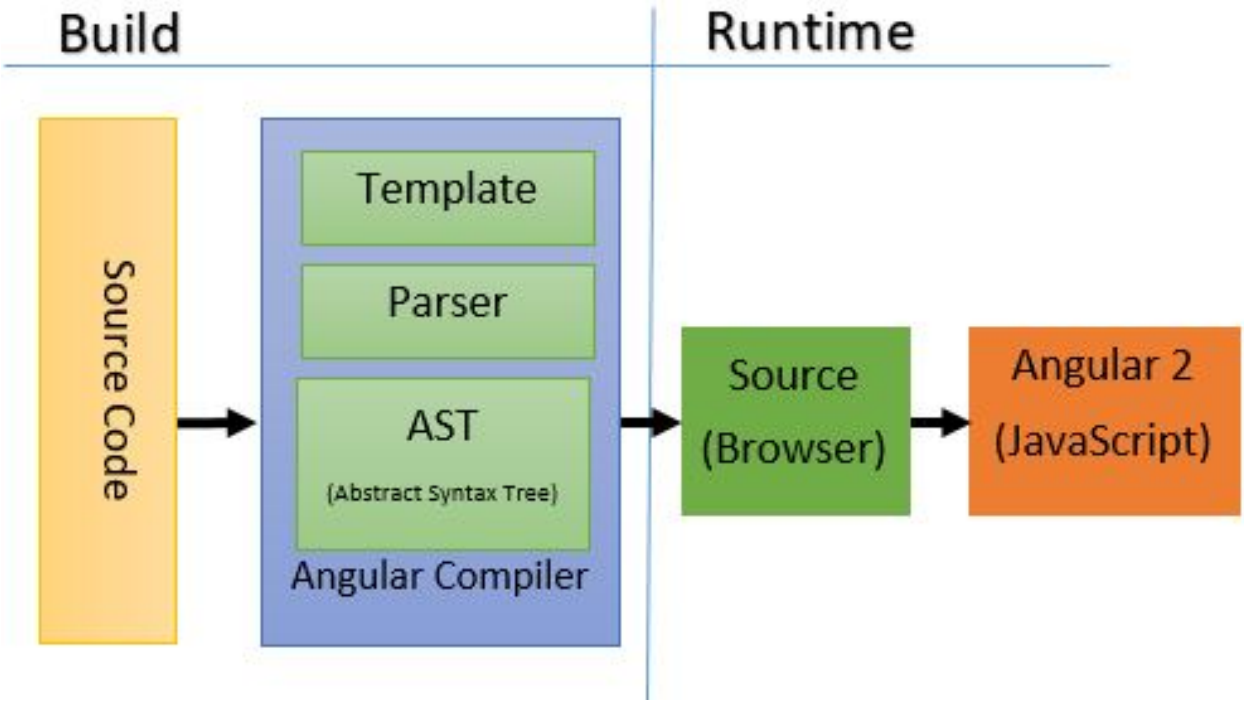
Se si scrive un semplice Hello World, e si fa una build, possiamo notare come la grandezza del file Javascript generato sarà di parecchi megabyte. Ma come può essere? In fondo le righe typescript+html scritte per un programma cosi banale sono pochissime.

Il motivo è che Angular integra nelle proprie build il proprio compilatore e tutti i moduli che possono servire, poiché le prime versioni di Angular2+ utilizzavano esclusivamente un compilatore just-in-time. Questo è molto utile durante lo sviluppo, ma il team di Google si è accorto immediatamente che veniva introdotto un overhead di tempo di cpu, per la compilazione di ogni modulo caricato, e di spazio, che aumentava i costi di banda.

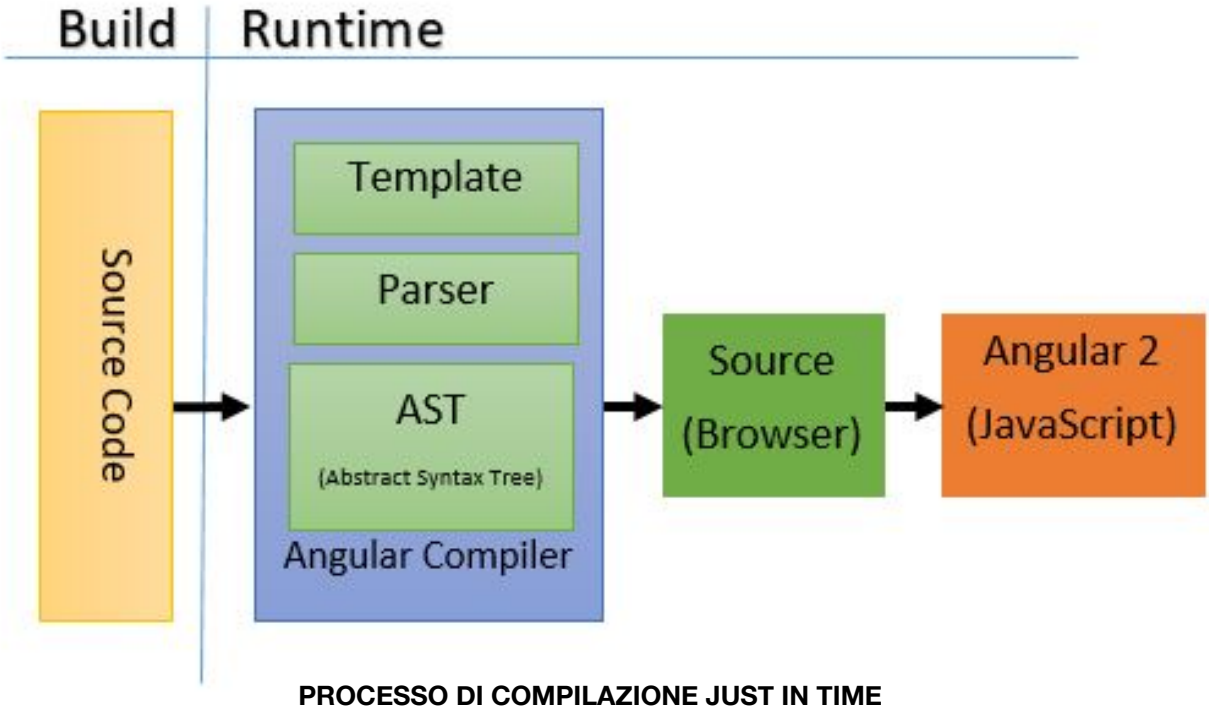
Dalla versione 4 è stato introdotto la possibilità, da poter utilizzare ovviamente esclusivamente nel caso di rilascio in produzione del codice, di usare un sistema di compilazione Ahead-of-time. Al termine della fase di build il codice javascript è immediatamente eseguibile su tutti i browser, o su piattaforme di esecuzione non visual, quali headless-browser, o sistemi di server side rendering. Il codice generato da questi com-

[6] https://blog.angularindepth.com/a-deep-deep-deep-deep-deep-dive-into-the-angular-compiler-5379171ffb7a

pilatori è comunque equivalente, cambia solo l'istante di compilazione ed il numero di file compilati.



PROCESSO DI COMPILAZIONE AHEAD OF TIME



PROCESSO DI COMPILAZIONE JUST IN TIME

Per quanto riguarda i sistemi di server side rendering (SSR), Angular CLI, il sistema per accedere ad un app Angular ed alle sue dipendenze attraverso riga di comando, offre delle funzionalità che permettono di soddisfare i requisiti operazionali di Scloby per quanto riguarda la visualizzazione rapida della prima pagina.

Angular Universal, questo è il nome del plugin che garantisce l'SSR, si possono generare delle pagine html che possono ad uno sguardo distratto sembrare l'app vera e propria. Queste pagine sono pagine html pure, e possono essere visualizzate anche nel caso in cui javascript sia disabilitato (non è questo il caso di Scloby che necessità sempre di avere JS attivo). In pratica con l'uso di Angular Universal si invia dapprima una pagina statica, che è collegata all'app completa, da cui verrà sostituita una volta completato il download.

Per poter configurare correttamente le funzionalità di SSR bisogna andare a personalizzare leggermente il modulo di bootstrap dell'applicazione, sostituendo renderModuleFactory, la funzione che permette di generare automaticamente l'entrypoint dell'applicazione nell'html, con NgExpressEngine.

Il DOM di Angular

In Angular.js la creazione della struttura DOM era delegata al browser, che prendeva i template HTML e creava un albero, in maniera similare a come normalmente un browser compone questa struttura partendo da un singolo file html, ma in questo caso prendendo i vari pezzi dai vari controllori.

Nelle più recenti versioni di Angular invece questo compito è delegato a questi due sistemi di compilazione, che fondamentalmente prendono i file html dei template, e generano un relativo file typescript che in maniera dichiarativa elenca tutti quanti i tag html, ed i loro valori, che devono essere mostrati per quel dato componente.

Questo cambio di impostazione è dovuto alle limitazioni che gli sviluppatori del framework hanno trovato nei browser durante il supporto alla versione 1.
Ogni browser ha delle sue specifiche tecniche di creazione del DOM, delle sue ottimizzazioni, e reagisce diversamente agli errori nei file html (un esempio sono la mancanza di tag chiusi, un errore molto comune). Inoltre HTML è un sistema case-insensitive, e quindi scrivere “h1” o “H1” è equivalente, e da ciò nasce la necessità di una sintassi “particolare”, utilizzata dalle direttive di Angular.js (ng-model, ng-if ecc…).

Un altro aspetto è la totale mancanza di flessibilità per l’ambito del SEO, che è di fondamentale importanza nel web moderno, con i motori di ricerca che stabiliscono loro stessi la qualità e le keyword di una pagina analizzando il contenuto testuale, o con i social che utilizzano tag html dinamici per poter ottenere le informazioni di anteprima da mostrare nelle timeline. Con Angular.js era praticamente impossibile fornire a questi crawler una pagina compilata e statica di ogni route dell’app, poiché era necessario un browser che componesse il DOM, mentre utilizzando javascript (e quindi anche un banale motore js v8 incluso dentro Node.js) si possono creare ambienti di server-side rendering, come Angular Universal. Questi problemi sono stati totalmente risolti demandando a javascript la creazione di tutta la struttura.

Le Prestazioni di Angular

Sicuramente Angular 2+ è un notevole balzo in avanti rispetto ai rudimentali tool presenti nella prima versione, ed è possibile applicare molti concetti già compresi dal team di sviluppo in un porting o nuova app verso questo framework.

Al netto di alcune differenze strutturali possiamo paragonare il concetto di “Componente” a quello di “Controllore”, cosi come l’utilizzo di servizi, filtri, e direttive è pressoché invariato.

Sicuramente il “costo” di tempo più importante da sostenere è quello di creare un nuovo sistema di tooling e formare il team all’utilizzo di typescript, ma la possibile scelta di creare un app ibrida con vecchio e nuovo codebase potrebbe compensare notevolmente questi costi.

È ovviamente inutile parlare di quale tra i due framework sia migliore nel fattore prestazione, ma possiamo estrapolare alcune analisi dai dati raccolti attraverso un benchmark open source, JS Framework Benchmark [7].

Come si può facilmente notare tutte le prestazioni per quanto riguarda la manipolazione del DOM, e le prestazioni di una normale View di un applicazione (come ad esempio una lista di componenti ripetuti N volte) sono migliori con l’utilizzo di Angular 2+, mentre le prestazioni della libreria in se, come l’occupazione di memoria o il peso del download dei file js sono meno efficienti, ma non è un aspetto negativo considerando la maggiore complessità e malleabilità fornita da Angular. Bisogna comunque notare che questo benchmark utilizza il sistema di compilazione di default JIT, invece di una compilazione Ahead-of-time che come abbiamo visto è più efficiente della punto di vista computazionale e fisico.

[7] https://github.com/krausest/js-framework-benchmark

Name Duration for...	angular-v7.1.4-keyed	angularjs-v1.7.2-keyed
create rows creating 1,000 rows	167.0 ± 6.9 (1.00)	200.8 ± 6.9 (1.20)
replace all rows updating all 1,000 rows (5 warmup runs).	153.5 ± 2.4 (1.00)	209.5 ± 6.0 (1.36)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	206.6 ± 7.8 (1.00)	215.5 ± 4.0 (1.04)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	46.5 ± 5.1 (1.00)	68.2 ± 6.7 (1.47)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	541.2 ± 6.7 (1.00)	547.7 ± 5.1 (1.01)
remove row removing one row. (5 warmup runs).	48.9 ± 1.4 (1.00)	51.6 ± 0.7 (1.06)
create many rows creating 10,000 rows	1,472.7 ± 29.6 (1.00)	1,954.3 ± 27.5 (1.33)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown	379.6 ± 18.5 (1.00)	536.4 ± 175.6 (1.41)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown	396.1 ± 6.4 (1.00)	536.5 ± 26.6 (1.35)

Name	angular-v7.1.4-keyed	angularjs-v1.7.2-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	3,121.3 ± 0.8 (1.15)	2,703.0 ± 0.3 (1.00)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	167.3 ± 1.6 (1.52)	109.8 ± 2.1 (1.00)
main thread work cost total amount of time spent doing work on the main thread. includes style/layout/etc.	595.3 ± 25.5 (1.00)	615.0 ± 13.1 (1.03)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	358.8 ± 0.0 (1.12)	320.9 ± 0.0 (1.00)

Name	angular-v7.1.4-keyed	angularjs-v1.7.2-keyed
ready memory Memory usage after page load.	5.1 ± 0.1 (1.82)	2.8 ± 0.0 (1.00)
run memory Memory usage after adding 1000 rows.	9.4 ± 0.0 (1.00)	10.7 ± 0.0 (1.15)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	9.7 ± 0.0 (1.00)	11.1 ± 0.0 (1.14)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	10.1 ± 0.0 (1.00)	11.7 ± 0.0 (1.15)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	6.7 ± 0.0 (1.48)	4.6 ± 0.0 (1.00)

React

Stando alle ultime statistiche di Stack Overflow[8] e di Github[9] il framework front-end basato su javascript con la crescita annua maggiore, e con più sviluppatori che ne portano avanti lo sviluppo, è attualmente React.js

React è una libreria open-source sviluppata da Facebook che nasce dall'esigenza dell'azienda di poter avere un toolkit unico per tutti i suoi prodotti, su tutte le piattaforme. Infatti una notevole caratteristica di React è che esistono dei compilatori che permettono di poter tramutare il codice html+css+javascript di un app scritta con essa in un software nativo per iOS ed Android. Esistono strumenti simili anche per Angular, e tratterò questo argomento successivamente.

React presenta sostanziali differenze rispetto ad Angular, e da ciò derivano i suoi pro e contro rispetto alla soluzione di Google.
Fondamentalmente questa libreria non fornisce un set completo di strumenti per soddisfare tutte le esigenze di uno sviluppatore, ma punta moltissimo sull'essere la più snella possibile. Non esistono infatti concetti come "servizi" o "filtri", e quindi per esempio non sono presenti dei moduli per la gestione delle connessioni http, o dei manipolatori dei contenuti per rendere automatica la conversione di data e ora nei diversi formati.

Lo scopo principale della libreria React è quello di fornire un modo facile, e veloce, per poter realizzare una completa interfaccia grafica, utilizzando il più possibile gli strumenti base di HTML5 e di javascript, modularizzandola il più possibile. Esistono tanti moduli che possono essere importati dentro una app così progettata, ma questi saranno sempre moduli standard javascript, preferibilmente scritti in ES6, e ciò perme-

[8] https://insights.stackoverflow.com/survey/2019#technology-_-web-frameworks

[9] https://octoverse.github.com/

tte di creare anche servizi e utility che possano andare bene sia integrati con React, sia utilizzati in maniera tradizionale in javascript/jquery o dentro un backend in Node.

Queste caratteristiche rispecchiano moltissimo le esigenze che Scloby necessità per lo sviluppo iterativo dell'applicazione. Protipizzare velocemente nuove funzionalità, garantendo la modularità del codice è imperativo per le scarse risorse disponibili.

Il virtual DOM

La maggiore differenza con Angular è presente nel sistema di manipolazione del DOM. Mentre il primo si basava essenzialmente sul generare una struttura HTML reale, sul quale poi innestare dei listener, per reagire alle azioni dell'utente, React crea una struttura totalmente virtuale che si interpone come un layer aggiuntivo tra il codice da noi sviluppato, e il DOM che il browser elabora per mostrarlo all'utente finale.

A che serve questa struttura, chiamata Virtual DOM? È efficiente oppure è solamente un vezzo per gestire al meglio la struttura di un app?
Innanzitutto dobbiamo considerare che React è una libreria puramente dichiarativa. Se in Angular il programmatore ha la possibilità di gestire il sottostrato javascript in maniera più personalizzabile (ad esempio inserendo query per selettori html e listener), in React tutto è gestito per noi dal framework. Il programmatore può esclusivamente creare dei componenti che, in maniera descrittiva, con l'utilizzo di variabili di stato o di binding, forniscono indicazioni su come la vista finale deve essere renderizzata. Sta poi alle routine sottostanti effettuare tutti i passaggi intermedi che portano al fornire al motore grafico del browser un albero di tag da mostrare all'utente.

La caratteristica principale del Virtual DOM è che limita il numero di interazioni con l'albero DOM reale, poiché ad ogni cambiamento di stato non è detto che venga aggiornata la vista lato utente.

Quando si crea un app React ogni componente possiede obbligatoriamente una funzione chiamata “render”, che non effettua la renderizzazione del componente in se, ma fornisce queste indicazioni, sotto forma di una funzione che verrà eseguita dal motore all’atto del refresh del componente. Avendo due alberi, virtuale e reale, ciò che fondamentalmente deve fare la libreria è un matching tra ogni foglia dei due, cosi da poterli sincronizzare. Questa operazione è detta “Riconciliazione”[10].

Non esistono però algoritmi efficienti che possano soddisfare questa necessità, i migliori hanno una complessità di $O(n^3)$, che se pensiamo alla sincronizzazione di una lista di 1000 righe significa un tempo computazionale enorme ed ingestibile.

Per questo motivo in React avviene una operazione di confronto tra i due alberi che prende in considerazione principalmente solo due elementi.
Il primo elemento è la differenza di tipo del componente, se si passa da un div ad un span, o da un componente chiamato Button ad uno Link, il framework deciderà che è meglio ri-elaborare tutto lo stack di quello specifico componente. Ugualmente ogni componente può avere associata una chiave univoca, che darà informazioni sul suo inserimento o cancellazione. Di norma questa chiave è un ID del modello a livello di base di dati, se stiamo manipolando tabelle o viste dinamiche.

Se avviene uno di questi due cambiamenti nella funzione allora il Virtual DOM si occuperà di sostituire integralmente i rami relativi sull’albero del browser, con i nuovi componenti.
Al contrario ci sono alcune modifiche per il quale non occorre questa costosa operazione, come ad esempio il cambiamento di degli attributi di supporto come class, style, ecc... In questi casi verranno solo modificati i relativi tag con le nuove specifiche, per una operazione immediata.

---

[10] https://reactjs.org/docs/reconciliation.html

Bisogna comunque sfruttare degli accorgimenti per fare in modo che questa struttura sia efficiente, come ad esempio avere delle chiavi univoche stabili, per evitare che un componente venga etichettato come da sostituire anche se in realtà nulla è cambiato nella sua struttura interna. Inoltre l'algoritmo non può effettuare confronti di similitudine su due sub-tree, e quindi se abbiamo dei componenti similiari, per i quali cambiano solo degli attributi standard html, o dei binding, è piu conveniente riunificarli in un solo componente poiché ciò riduce la necessità di rerendering. Un esempio può essere un sistema di Tabbing con diverse tab con liste di dati simili, nel quale solo una alla volta viene mostrata all'utente. Ad ogni switch del menu si ha una sincronizzazione tra gli alberi, a meno che queste componenti non siano uniche con solo i campi valorizzati in maniera differente.

Il motore di rendering

Attualmente gli sviluppatori di React stanno procedendo ad una revisione di tutte le componenti interne del motore di rendering, per renderlo ancora più efficiente e pronto alla release della versione 1.0. Infatti nonostante il vasto utilizzo ufficialmente ancora la libreria non ha raggiunto la sua prima versione stabile, poiché gli sviluppatori si tengono in riserva la possibilità di introdurre **breaking changes** al codice (anche se nella realtà dei fatti questo non avviene da parecchie major releases).
Questa revisione è chiamata React Fiber[11] ed il suo scopo è quello di migliorare le performance, e la facilità di creazione, di molti aspetti della composizione grafica di un app, come ad esempio animazioni o gestures. Questo nuovo motore di rendering, cui alcuni pezzi dell'implementazione sono già inseriti entro le più recenti versioni della libreria, è un lavoro incrementale, approccia il problema della reattività usando un sistema di "Scheduling" delle operazioni.

[11] https://github.com/acdlite/react-fiber-architecture

Mentre normalmente il rendering di una schermata avviene in maniera procedurale e contemporanea, Fiber da la possibilità di gestire autonomamente la composizione di ogni componente. Poiché ogni componente in javascript è in realtà una funzione che richiama ricorsivamente tutte le altre funzioni di rendering dei componenti inclusi dentro il padre, possiamo dire che tutto il rendering fa parte di una immensa call stack. I browser moderni implementano delle API che permettono di ridurre la grandezza di questa call stack, come per esempio requestIdleCallback o requestAnimationFrame, che permettono di fornire delle priorità a specifiche funzioni. Il problema è che bisogna cercare di rifattorizzare tutto il sistema in unità di rendering incrementale, per le quali poi si possono sospendere e manipolare gli stati.
Chiamiamo questi blocchetti di operazioni **Fiber**, e quindi React crea una collezione di queste Fiber che non sono altro che frame virtuali di tutto lo stack del programma. I vantaggi di questo sistema sono che in questa maniera è possibile tenere in memoria questi pezzi di codice ed eseguirli come e quando si vuole, e React Fiber è proprio l'algoritmo che schedula tutto questo sistema.

Al contrario di Angular, React non offre un sistema completo di Server Side Rendering, permette solo l'utilizzo della libreria ReactDOMServer, che fornisce dei tool per poter generare una stringa html partendo da un componente. Sotto questo aspetto il framework deficita un pò, e necessita di ulteriori librerie a supporto che permettano di encapsulare una intera app React in maniera tale da poter servire pagine statiche. La modifica del motore di rendering che sta avvenendo con la transazione a Fiber permetterà, utilizzando sempre lo stesso concetto di Task e Fibre, di poter fornire dentro la libreria stessa tutte quelle funzionalità utili al SSR, ma questi sono solo delle pianificazioni a lungo termine da parte del team di sviluppo, che potrebbero non essere con-

fermate. Sotto questo aspetto quindi alcuni requisiti operazionali individuati in Scloby potrebbero non essere soddisfatti.

Come è strutturato un programma in React

A differenza di Angular le applicazioni scritte in React non presentano una struttura particolare, nella quale esistono differenti tipologie di classi (Componenti, Servizi, Filtri ecc..), poiché stiamo parlando di una libreria per sole interfacce grafiche, non di un framework completo.

Quindi c'è solo una tipologia di oggetti manipolati da React, e si tratta di delle classi che contengono alcuni metodi di Lifecycle, chiamati anche in questo caso "Componenti."

Questi componenti sono in tutto e per tutto delle classi standard ES6, e quindi possono avere attributi e metodi aggiuntivi, ma devono tutte estendere React.Component, che contiene tutte le routine ed i metodi che servono al motore grafico per renderizzare la vista.

Il cuore di un componente è composto dal suo stato. Ad ogni cambiamento dello stato interno corrisponde un segnale di re-rendering inviato alla libreria, che, in base alle politiche da me illustrate precedentemente, decide come effettuare la riconciliazione del nuovo virtual dom con il componente modificato, e lo stato del browser.

Bisogna però notare come lo stato in se non sia un normale attributo, ma è un valore più complesso che non va mai manipolato direttamente dopo la prima inizializzazione (che deve avvenire nel costruttore), ma solo attraverso il metodo setState.

Oltre allo stato, ed agli attributi diretti della classe, c'è un altro sistema per immagazzinare e manipolare i dati di una classe in React, ed è il concetto di Prop, che non è altro che un binding tra l'oggetto padre ed il figlio, che per esempio permette di passare alcuni dati di inizializzazione, e che permette anche di avere una classe più

snella, magari senza stato, e che semplicemente reagisce a cambiamenti a livello superiore.

I prop non sono per forza degli oggetti del padre passati al figlio, ma è possibile inserire qualsiasi tipo di valore javascript, che siano variabili ,costanti, o addirittura funzioni, anche facenti parte della classe padre. Poichè la relazione è solo one-way passare una funzione del padre ad un figlio è un modo per poter reagire a delle azioni effettuate a livello inferiore, a patto di inserire un relativo scope o callback.

Se non abbiamo bisogno di uno stato, o di una logica particolare, React ci offre anche la possibilità di dichiarare un componente esclusivamente come una funzione. Questo è possibile perché, come illustrato precedentemente, ogni parte di una View in React non è altro che una funzione, un frame dentro uno stack più grande, e quindi la classe ES6 con cui è normalmente costruito un componente non è altro che un Syntatic-Sugar per questa stessa funzione a basso livello, così come in generale vale per tutte le classi in javascript, dove non esiste un vero e proprio concetto di oggetto.

È quindi possibile fornire alla libreria di rendering di React dei componenti costruiti in maniera complessa senza l'utilizzo di una classe ES6, utilizzando la funzione "createReactClass", che prende in ingresso come parametro un oggetto, in cui le coppie chiave-valore sono le "funzioni" della nostra ipotetica classe.

JSX

Fin qui ho parlato della struttura di fondo di React, trascurando l'aspetto direi fondamentale di ogni strumento utilizzato per creare un front-end: Esattamente come viene progettata la vista visualizzata dall'utente?

React offre, rispetto ad Angular, ed alla maggioranza dei framework JS, un approccio molto semplice ed immediato per delineare l'html da visualizzare. Come detto è la funzione render che si occupa di generarlo, ma cosa può essere inserito dentro di essa?

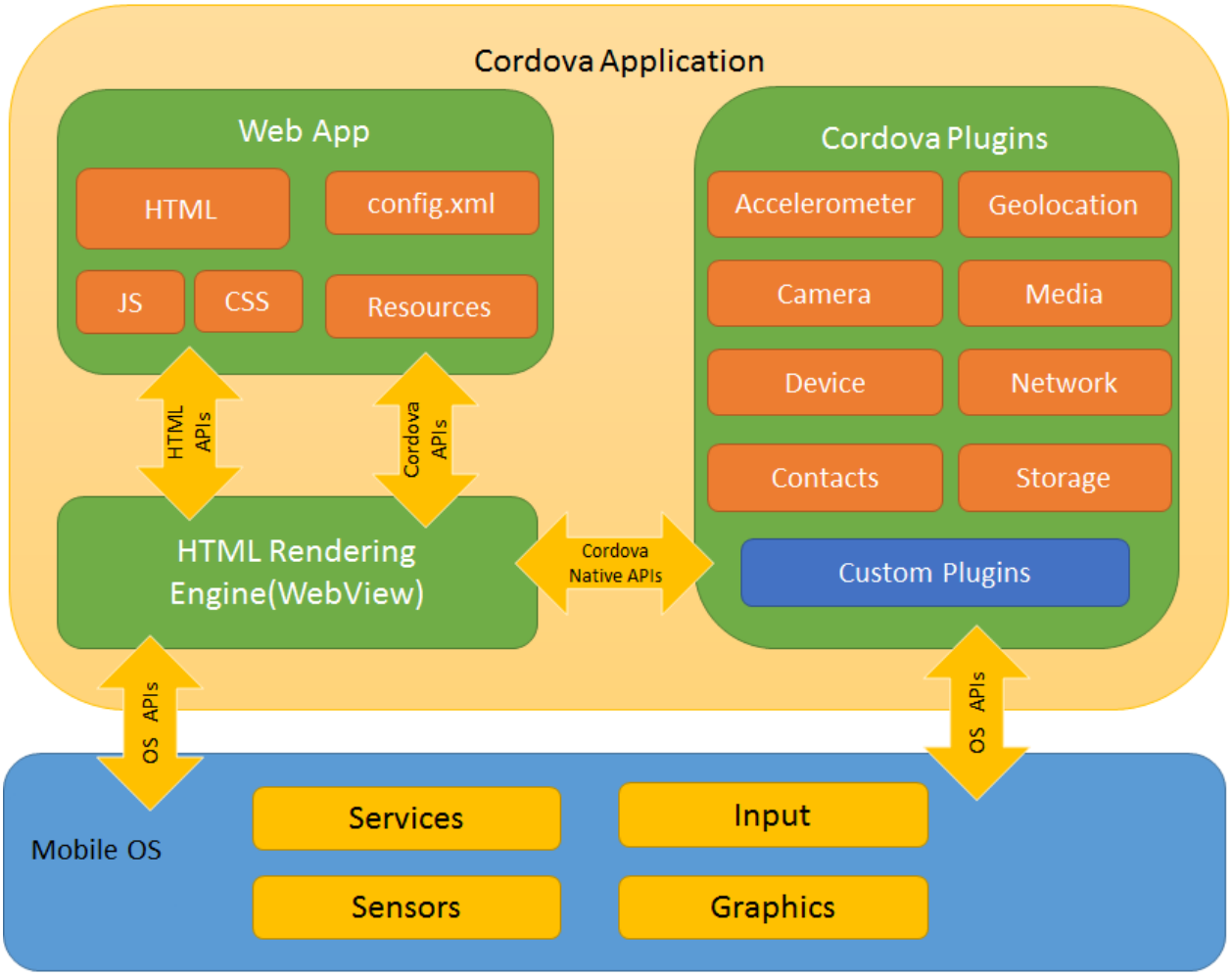
In Angular bisogna associare ad un componente un template, che sarà un normale codice html con però degli attributi in più per ogni tag, cosi da poter manipolare il codice in base a cosa mostrare, che classi inserire, creare liste e così via. Per effettuare controlli di logica è però necessario usare delle direttive aggiuntive, sporcando il codice HTML stesso, rendendolo un pò meno leggibile.
In React non si utilizza un html puro, ma invece si fa uso di un formato chiamato JSX, che possiamo considerare un sua estensione, e che permette di salvare dentro una variabile questo codice.

Essendo una libreria puramente dichiarativa si ritiene che l'UI sia intrinsecamente legata alla logica che la genera. Possiamo immaginare la composizione grafica di un componente React come un lontano parente della stessa operazione in PHP. Dentro la funzione render si possono usare controlli di stato, variabili dinamiche e tutte le normali operazioni di logica di javascript, l'unica prerogativa è che il valore ritornato da questa funzione deve rispettare le specifiche di JSX, poiché sarà il layout della vista.

E' possibile evitare di usare JSX, anche perchè è necessario avere un tooling di compilazione del JSX in javascript per poterlo usare, ma come abbiamo visto con l'utilizzo di Typescript in Angular, questi sono consigli da rispettare quasi obbligatoriamente, poichè sono strettamente legati alla libreria.
Babel è uno dei compilatori più utilizzati per questo processo, ed è spesso usare per poter generare un file javascript finale compatibile anche con ES5, utilizzando però come codice sorgente un file ES6 o superiore.

Tendenzialmente JSX rispetta lo standard html, ma presente alcune modifiche. Ad esempio molti attributi standard, come class o style, presentano una sintassi, o un nome differente, cosi da rendere più facile la loro manipolazione nel virtual DOM ed evitare bug, oppure trarre in inganno gli sviluppatori.

JSX è un sistema che rende più semplice creare le funzioni di generazione del dom, poiché durante la compilazione il codice viene estratto dalle classi, e viene sostituito da



una equivalente funzione *React.createElement(component, props, …children)*. Questo è vero per molti comportamenti di React, che al posto di usare Typescript con i decoratori, o listener del dom del browser, fornisce durante la fase di compilazioni queste traduzioni che rendono il codice più leggibile ed immediato, riducendo anche le righe di codice, cosa che tende a far diminuire la percentuale di bug.

In Angular per poter generare un semplice componente che mostra in maniera dinamica uno od un altro tag bisogna per prima cosa dichiarare il componente con tutta la fase di Dependency injection. Poi bisogna scrivere un template e riempirlo con le direttive di control-flow, e successivamente scrivere tutta la parte di logica interna per

manipolare gli attributi che pilotano questi controlli. In React tutto ciò è possibile anche all'interno di una singola funzione, manipolando direttamente il valore restituito usando i props del Parent.

Questo paradigma di composizione grafica similare al PHP è uno dei maggiori pregi che potrebbe avere React dentro Scloby. Il poter gestire in un unico file sia la manipolazione del modello, sia la sua vista comporta una velocità e facilità di uso immediato, e React infatti, secondo gli stessi report che parlano della sua diffusione, è considerato da molti una libreria molto facile da imparare e da gestire. E per un team ridotto come quello dell'azienda queste caratteristiche sono sicuramente un valore aggiunto.

Un componente JSX può contenere inoltre anche ulteriori componenti, esattamente come avviene normalmente con div dentro ad altri div in html, e la classe può accedere a questi figli diretti attraverso i props.
Se la funzione render di un componente ritorna un valore true/false o null/undefined semplicemente quel componente sarà considerato come "vuoto" e quindi non verrà innestato dentro il dom. Questo è un modo semplice ed efficace per nascondere in maniera dinamica parti della vista.

Librerie Javascript dentro un componente React

Sorge però un problema in questo sistema di composizione della vista, è possibile integrare funzionalità javascript esterne alla libreria?
Un esempio molto utilizzato è quello dell'introdurre widget che normalmente vengono innestati attraverso javascript a specifici elementi del DOM attraverso l'utilizzo del tag id, come ad esempio una mappa di Google Maps.

Ci sono due-tre strade che si possono affrontare: la prima, la più semplice ed immediata è quello di controllare se quello specifico widget è stato re-implementato sfruttando le potenzialità di React, cosi facendo basterà importare il componente instal-

lato con il package manager prescelto e lo si potrà usare esattamente come se fosse un proprio componente customizzato.

La seconda via è quella di sfruttare una possibilità di integrazione con il DOM reale che ci offre React, ovvero l'utilizzo dei ref.
Dentro la fase di startup del componente si può creare un attributo della classe di tipo ref, usando la funzione React.createRef() ; a questo punto questo attributo potrà essere passato ad un normale tag html dentro la funzione di render, cosi da poter fare sempre riferimento in maniera univoca a quel tag dentro le nostre operazioni di logica.

Dentro i metodi di lifecycle del componente si può usare questo ref al post dei vari document.findTagBy, findId ecc… di javascript per poter montare i widget in quella posizione. Bisogna solo fare attenzione a caricare prima dello startup del componente, o di tutta l'app, la libreria necessaria tramite il package manager o i tag script di html. Se usiamo webpack per gestire tutta la fase di compilazione e testing del nostro prodotto è sicuramente più facile evitare qualsiasi tipo di errore.

Un ultima possibilità deriva da questo sistema di ref-ing; è infatti comprensibile come nel caso in cui si voglia creare un componente più completo ed affidabile, che utilizzi widget e librerie esterne a React (in maniera anche da poterlo riutilizzare in altri contesti) si può progettare un componente wrapper intorno a questi strumenti cosi da poter effettuare questo lavoro una sola volta. La maggior parte delle conversioni di widget javascript in componenti react sono proprio soluzioni che adottano questo sistema di wrapping invece di creare da zero il componente.

A differenza di Angular quindi React ha una maggiore difficoltà di gestione di risorse reali presenti nel browser. Usare document, jQuery, o altre librerie caricate tramite file script non è facile, e può portare ad errori. Scloby purtroppo ha un notevole accoppiamento con alcune api di questo tipo per poter dialogare con strumenti esterni.

I lettori bluetooth che forniscono dei codici a barre come testo in input hanno necessità di utilizzare dei listener del DOM; alcune richieste da inviare alle casse devono essere generate partendo utilizzando il modulo “document” di javascript per poter generare un file XML.

È possibile ovviare a questi inconvenienti, però i vantaggi di velocità di creazione di una interfaccia in React possono essere vanificati dagli hack necessari per far dialogare molte di queste componenti esterne.

High Order Components

Recenti novità nell’ecosistema sono le api che permettono di creare degli High-OrderComponent e React Hooks.

L’architettura basata sui primi è una funzionalità piuttosto avanzata che permette riutilizzare parte della logica su componenti diversi senza che essi siano gerarchicamente legati da una relazione padre-figlio.

Possiamo fare l’esempio di componenti che devono essere mostrati dinamicamente in base allo stato attuale di autorizzazione dell’utente (login, logout, admin ecc…). La soluzione più banale sarebbe quella di controllare in ogni componente questo status, ma se parliamo di decine di componenti questa soluzione non è flessibile, e comporterebbe un notevole rischio di inserimento di bug o lunghezza di refactoring in caso di cambio di paradigma di autenticazione. Si potrebbe migliorare la soluzione astraendo il tutto ad una classe di servizio, ma rimarrebbe la necessità di fornire alcuni accorgimenti o personalizzazioni ad ogni componente, utilizzando comunque quegli strumenti forniti dalla classe di servizio.

Usare un High Order Component permette di incapsulare questi elementi dentro un ulteriore componente che si occupa per noi di alcune operazioni, per poi includere il componente originale con il passaggio dei relativi prop da noi definiti. Bisogna co-

munque fare attenzione ad allegare anche tutti i prop sconosciuti al high order component, poiché è probabile che questi siano indirizzati al componente originale, di cui il wrapper è totalmente senza conoscenza del comportamento interno.

Sono tanti gli esempi tra i moduli aggiuntivi e core di React che utilizzano questa funzionalità avanzata, e possiamo fare uno tra gli esempi piu importanti:

L'inizializzazione di un app scritta in React avviene con un entry point javascript ReactDOM.render, che accetta in ingresso un componente ed il tag html su cui si deve innestare dentro la pagina. Ovviamente possono essere presenti più entrypoint per ogni app, se magari vogliamo tenere le logiche delle sezioni separate, analogamente a quanto accade in altre librerie.
Questo componente potrà essere il top-most di tutta la gerarchia della nostra app, ma in realtà la maggior parte delle volte il componente che andiamo ad includere è un router.

React Router è una libreria che va ad analizzare il path dell'url attualmente presente nel browser per poter scegliere il giusto entry-point dell'app, attraverso l'utilizzo di funzionalità regex. Ad esempio, se non è presente alcun path potremmo essere dentro la home; se è '/posts/1343-hello-world' probabilmente il router avrà una regola regex che deciderà quale è la tipologia di view per quel dato post, e innestarà nel dom il relativo componente.

I componenti che andiamo ad includere però non sono a conoscenza dell'utilizzo del routing, e probabilmente nel caso di una applicazione dinamica c'è l'esigenza di conoscere lo slug attuale, o tutta la storia di navigazione, per effettuare alcune operazioni. Per questo motivo React-Router offre una serie di funzioni high-order il cui scopo è passare ai componenti manipolati le informazioni che vogliamo esporre, withRouter, withHistory ecc…

È inoltre importante sapere che possiamo combinare più funzioni HighOrder una dentro l'altra, cosi da poter creare componenti con maggior funzionalità.

React Hooks

L'ultima novità in React sono gli Hook. Come illustrato precedentemente un componente può essere creato o attraverso una classe, o attraverso una funzione; solamente se viene generato da una classe esso può avere, oltre ad attributi e metodi, uno stato. In realtà dall'ultima versione, 0.16.8, non è più così, poiché sono stati introdotti gli Hooks.

Mentre il normale concetto di stato in React è un oggetto corposo che contiene tutte le informazioni necessarie, lo stato attraverso gli hook possiamo considerarlo come un insieme di variabili indipendenti con ognuna che manipola un pezzo differente del nostro componente.

let [stato, modificaStato] = useState('topolino');

Attraverso la funzione useState() si ottiene un array di due soli elementi. Il primo è il nostro stato vero e proprio, il secondo è la funzione che dobbiamo utilizzare per modificarlo; lo stato può essere inizializzato passando un parametro alla funzione di creazione. È possibile creare un numero arbitrario di stati, poiché ogni chiamata alla funzione fornirà due nuovi hook totalmente indipendenti da quelli precedenti.

Oltre alle funzionalità di stato nei tradizionali componenti basati su classi esistono una serie di metodi di lifecycle che permettono di instanziare connessioni e/o manipolare i dati prima del rendering. Nel caso di utilizzo degli hook queste funzionalità ci vengono ovviamente precluse, ed è per questo che assieme ad useState esiste la funzione useEffect.

useEffect ha come parametro una funzione, che viene eseguita ogni volta che avviene un cambiamento di uno qualsiasi degli stati che abbiamo instanziato. Questo

permette di effettuare delle manipolazioni prima che venga mandato in esecuzione il re-rendering del codice.
Dentro gli effect vorremo poter creare magari dei listener, per poter aspettare una response http per esempio, ma questi listener hanno bisogno di essere eliminati quando il componente viene smontato dal DOM; è possibile fare ciò poiché questa funzione può avere un ulteriore funzione come valore di ritorno, che verrà eseguita ogni volta che il motore di React deciderà di avere necessità di clean-up. Bisogna però fare attenzione poichè il clean-up avverrà anche ad ogni esecuzione di useEffect, e non è un comportamento sbagliato in quanto ci potremmo ritrovare con più listener uguali, creando possibili conflitti.

In caso di un componente con decine di stati sarebbe poco conveniente a livello di performance andare a ri-eseguire tutti le manipolazioni al cambiamento di anche solo uno stato, per questo motivo si possono creare più effetti, magari tutti coerenti internamente, che potranno essere differenziati, con un parametro aggiuntivo, per quando essere eseguiti, in base al relativo stato aggiornato.

I developer di React puntano a sostituire molti use-case dove vengono usate le funzioni high-order con gli Hook, questo poiché è possibile crearne di customizzati che possono reagire a cambiamenti di contesto a più alto livello, utilizzando i paradigmi di subscriber e consumer; ad esempio i manteiner del precedentemente illustrato react-router già stanno lavorando su un hook useRouter() per poter le esporre medesime funzionalità. Tuttò ciò porta ad una semplificazione di scrittura e maggior leggibilità del codice senza però perdere in potenza espressiva del linguaggio.

Bisogna però avere delle accortezze durante la scrittura del codice, la maggiore delle quali è quella di chiamare sempre gli hook nel livello piu esterno del nostro componente, di non includerli mai dentro cicli o dentro operatori di control-flow, poiché in-

ternamente React non sa riferirsi in maniera univoca al singolo stato o effetto, ma usa l'ordine in cui sono eseguiti nel codice.
Se volessimo avere degli hook condizionali, ad esempio un effetto attivo solo quando siamo in un determinato stato, si può inserire il controllo di logica all'interno dell'effetto stesso.

Le prestazioni di React

React quindi sfrutta un paradigma radicalmente differente da quello di Angular, e per quanto si tratti sempre di un sistema di frontend che usa un linguaggio js-like, classi, e i normali tool html+css siamo comunque di fronte ad un approccio diverso. Sarebbe conveniente scegliere questa libreria rispetto ad altre? Dipende essenzialmente da come vogliamo strutturare il nostro progetto.

La facilità d'uso della composizione grafica di React permette anche a semplici designer non programmatori, a patto di avere una conoscenza di html, di poter rapidamente fornire ai developer informazioni e pezzi di codice necessari ad una messa in testing e produzione di un componente o di un modulo.

Se c'è però una cosa che la libreria di Facebook rende difficoltoso è la macrogestione del progetto. Non avendo dependancy injection, e non avendo come requisiti consigliati l'utilizzo di un paradigma observer-subscriber, React soffre terribilmente nella composizione di app complesse. È necessario posizionare sopra il layer della libreria grafica tutta una struttura di sincronizzazione e reperimento dei dati, funzionalità che Angular offre già inclusi o per potenzialità del linguaggio e del framework (la possibilità di creare servizi), o perché possiede nelle proprie api vari strumenti utili.

L'utilizzo però di queste librerie aggiuntive permette di imbrigliare dentro un paradigma subscriber/observer la maggioranza delle interazioni che avvengono dentro il front-end. Attualmente quando c'è da dialogare con un componente esterno all'appli-

cazione, come per l'appunto i sistemi di cassa fiscale, oppure quando c'è da aggiornare uno stato remoto, Scloby utilizza un sistema di notifiche customizzato, poiché in Angular.js non erano presenti anche in quel caso metodi efficienti per la gestione di queste funzionalità. Con l'incremento del numero di moduli, di utenti, e di routine remote da effettuare questa architettura non va più bene, e quindi è tassativo l'utilizzo di sistemi similari che in ulteriori framework, come Angular, ci vengono fornite direttamente.

Ci sono tanti modi per offrire queste funzionalità in React, come ad esempio le librerie Redux o mob-x, la scelta sta allo sviluppatore, ma non essendo integrati nel core di React sono tutti sistemi da imparare a parte, codice in più da mantenere e da rifattorizzare ad ogni breaking change. È quindi necessario per Scloby ponderare con accuratezza se optare o meno per una di queste librerie.

A livello di prestazioni possiamo facilmente notare l'impatto di queste differenze con Angular e predecessore.
Angular.js a livello di composizione grafica rimane sempre il sistema meno performante, come era logico aspettarsi viste le premesse. Dall'analisi dei tempi di esecuzione dei vari benchmark possiamo notare come la leggerezza di React sia al tempo stesso il suo punto di forza e di debolezza.
Nei tempi di caricamento il confronto con Angular è impietoso, essendo React una libreria molto snella viene scaricata in memoria in 1/3 del tempo rispetto al concorrente, quasi 100ms, e si ha un notevole risparmio di banda, che in contesti da traffico consistente potrebbe garantire un risparmio dei costi non indifferente.
D'altro canto React da solo non è usabile per applicazioni complesse, ma ha bisogno di quella serie di librerie aggiuntive, che fanno lievitare i suoi tempi di download e carica-

mento fino a raggiungere quasi i livelli del prodotto di Google, rendendo così inutile questa comparazione.

Quello che si può sostenere è che React sembra avere un lieve vantaggio competitivo dal punto di vista prestazione, un vantaggio che però va ben ponderato nell'ottica dei suoi costi di formazione e di produttività con il dover inserire tutta quella serie di librerie esterne necessarie.

Name	react-v16.8.3-keyed	react-hooks-v16.8.3-keyed	angular-v7.1.4-keyed	react-redux-v16.8.3 + 6.0.0-keyed	angularjs-v1.7.2-keyed	react-mobX-v16.4.1 + 5.0.3-keyed
ready memory Memory usage after page load.	2.4 ±0.2 (1.00)	2.4 ±0.0 (1.00)	5.1 ±0.1 (2.14)	2.5 ±0.0 (1.05)	2.8 ±0.0 (1.18)	2.7 ±0.2 (1.15)
run memory Memory usage after adding 1000 rows.	6.8 ±0.0 (1.00)	7.1 ±0.0 (1.05)	9.4 ±0.0 (1.37)	9.0 ±0.0 (1.33)	10.7 ±0.0 (1.57)	10.1 ±0.0 (1.49)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	8.0 ±0.0 (1.00)	8.3 ±0.0 (1.04)	9.7 ±0.0 (1.22)	10.6 ±0.0 (1.33)	11.1 ±0.0 (1.39)	11.1 ±0.0 (1.39)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	8.8 ±0.0 (1.03)	8.6 ±0.0 (1.00)	10.1 ±0.0 (1.18)	13.4 ±0.0 (1.56)	11.7 ±0.0 (1.36)	18.2 ±0.0 (2.13)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	4.6 ±0.1 (1.04)	4.4 ±0.1 (1.00)	6.7 ±0.0 (1.54)	6.1 ±0.0 (1.38)	4.6 ±0.0 (1.04)	8.4 ±0.1 (1.91)

Name Duration for...	react-v16.8.3-keyed	react-hooks-v16.8.3-keyed	angular-v7.1.4-keyed	react-redux-v16.8.3 + 6.0.0-keyed	angularjs-v1.7.2-keyed	react-mobX-v16.4.1 + 5.0.3-keyed
create rows creating 1,000 rows	170.0 ±6.1 (1.03)	165.8 ±4.9 (1.00)	167.0 ±6.9 (1.01)	189.9 ±8.3 (1.15)	200.8 ±6.9 (1.21)	233.0 ±7.0 (1.41)
replace all rows updating all 1,000 rows (5 warmup runs).	147.0 ±1.5 (1.00)	147.2 ±1.9 (1.00)	153.5 ±2.4 (1.04)	154.0 ±1.4 (1.05)	209.5 ±6.0 (1.42)	177.4 ±2.4 (1.21)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	255.3 ±11.0 (1.24)	239.2 ±13.2 (1.16)	206.6 ±7.8 (1.00)	275.0 ±16.1 (1.33)	215.5 ±4.0 (1.04)	327.0 ±14.3 (1.58)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	54.5 ±2.3 (1.17)	59.0 ±3.0 (1.27)	46.5 ±5.1 (1.00)	79.2 ±5.8 (1.70)	68.2 ±6.7 (1.47)	113.0 ±20.7 (2.43)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	534.1 ±4.0 (1.01)	529.2 ±5.3 (1.00)	541.2 ±6.7 (1.02)	537.1 ±6.9 (1.01)	547.7 ±5.1 (1.04)	546.4 ±6.2 (1.03)
remove row removing one row. (5 warmup runs).	49.7 ±0.6 (1.02)	51.9 ±3.1 (1.06)	48.9 ±1.4 (1.00)	58.1 ±0.8 (1.19)	51.6 ±0.7 (1.06)	54.0 ±1.0 (1.10)
create many rows creating 10,000 rows	1,803.7 ±43.4 (1.22)	1,815.6 ±65.2 (1.23)	1,472.7 ±29.6 (1.00)	1,955.6 ±28.0 (1.33)	1,954.3 ±27.5 (1.33)	2,107.3 ±43.4 (1.43)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown	376.2 ±7.8 (1.02)	369.4 ±4.4 (1.00)	379.6 ±18.5 (1.03)	421.2 ±10.4 (1.14)	536.4 ±175.6 (1.45)	490.3 ±12.1 (1.33)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown	202.3 ±6.3 (1.00)	203.1 ±5.7 (1.00)	396.1 ±6.4 (1.96)	228.6 ±4.9 (1.13)	536.5 ±26.6 (2.65)	333.6 ±9.9 (1.65)

Name	react-v16.8.3-keyed	react-hooks-v16.8.3-keyed	angular-v7.1.4-keyed	react-redux-v16.8.3 + 6.0.0-keyed	angularjs-v1.7.2-keyed	react-mobX-v16.4.1 + 5.0.3-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	2,360.4 ±0.8 (1.00)	2,360.0 ±1.3 (1.00)	3,121.3 ±0.8 (1.32)	2,518.1 ±2.3 (1.07)	2,703.0 ±0.3 (1.15)	2,703.2 ±0.2 (1.15)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	52.5 ±2.1 (1.23)	42.5 ±17.4 (1.00)	167.3 ±1.6 (3.93)	67.8 ±4.9 (1.59)	109.8 ±2.1 (2.58)	74.3 ±1.9 (1.75)
main thread work cost total amount of time spent doing work on the main thread. includes style/layout/etc.	560.4 ±3.8 (1.00)	573.7 ±8.5 (1.02)	595.3 ±25.5 (1.06)	582.8 ±11.3 (1.04)	615.0 ±13.1 (1.10)	587.3 ±2.3 (1.05)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	260.6 ±0.0 (1.00)	259.4 ±0.0 (1.00)	358.8 ±0.0 (1.38)	276.5 ±0.0 (1.07)	320.9 ±0.0 (1.24)	313.5 ±0.0 (1.21)

Soluzioni native o ibride

Cosa si intende per applicazione nativa? E' un app che è scritta in uno dei molteplici linguaggi di programmazione e che viene compilata in codice binario utilizzando gli SDK offerti dai vari sistemi operativi.
Nonostante la presenza di un app di Scloby sui maggiori store dei sistemi operativi, in realtà non esiste nessuna applicazione nativa, ma sono tutte dei wrapper costruiti intorno al codice sorgente della web app realizzata in angular.js.

Questa soluzione è al giorno d'oggi molto utilizzata dalle aziende del settore, per molteplici ragioni che comunque hanno la loro origine nel desiderio di risparmiare i costi supportando comunque diverse piattaforme.
Per poter avere un app di Scloby su tutti i sistemi più diffusi bisognerebbe realizzare almeno 4 codebase totalmente scollegate tra di loro:

- Applicazione Web in Angular.js
- Applicazione per iOS in Swift/Objective-C
- Applicazione per Android in Java
- Applicazione per Windows in C#/UWP

È chiaro che avere questi 4 prodotti comporterebbe un notevole bisogno di personale in più, per il semplice fatto di dovere scrivere una certa routine o vista in 4 sistemi differenti, con 4 layout Engine diversi, con il rischio di aggiungere 4 volte il numero dei bug, e quindi una necessità di avere un processo di QA più lungo e scrupoloso.
Inoltre bisognerebbe scegliere se avere 4 team differenti, oppure formare tutti gli sviluppatori per poter lavorare su tutti e 4 gli SDK.

Per una piccola azienda del settore come Scloby questi costi, che possono andare da 30.000€/anno a 70.000€/anno per ogni membro del team aggiuntivo, secondo

l'anzianità, possono avere un notevole impatto sul bilancio societario. Lo scopo principale di una startup, o comunque di un'azienda che lancia sul mercato un nuovo prodotto, è quello prima di imporsi nel suo settore e solo successivamente massimizzare gli incassi, poichè la crescita del numero di installazioni non comporta un aumento dei costi di sviluppo con una relazione lineare, e quindi ci si può permettere a quel punto di poter aumentare l'organico.

In una grande azienda le spese extra per poter fornire funzionalità avanzate e personalizzate, tra cui applicazioni puramente native, sono un problema presente, ma comunque più trascurabile rispetto a realtà minori.

È quindi necessario cercare un compromesso per poter fornire il proprio prodotto al mercato più grande possibile con lo sforzo minore possibile. Ed in questo possono venire in aiuto sistemi di cross-compiling o di ibridazione.

Cordova

La soluzione impiegata attualmente da Scloby è quella di usare Apache Cordova per poter inserire un'intera applicazione web dentro un app, cosi da non dover sempre accedere ai server per poter scaricare gli assett. Cordova utilizza le api del sistema per poter fornire all'utente questi documenti esattamente come se stesse navigando dal browser del sistema operativo, mascherando ovviamente gli url e le altre funzionalità proprie di Chrome o Safari.

In Scloby vengono utilizzati come ambienti di deploy iOS, Android, ed Electron, un ulteriore framework sviluppato da Github che contiene Node.js ed una installazione di chrominum, garantendo cosi la perfetta compatibilità su tutti i sistemi desktop.
Un app Cordova è composta dalla webview, dalla webapp, che deve contenere un file config.xml con dentro varie informazioni che servono al sistema operativo ospite (come ad esempio, ma non solo, il nome del bundle iOS, o del package di Android), e

da una serie di plugin che servono alla webview per potersi connettere ad alcune funzionalità avanzate del sistema operativo, come tutte le api per poter leggere dentro le sandbox su iOS, accesso all'hardware come la fotocamera integrata, usare i motori di rendering video e cosi via.
Esistono tutta una serie di plugin ufficiali mantenuti dal progetto, e le terze parti, o lo sviluppatore stesso, possono crearne di personalizzati per poter fornire ulteriori funzionalità.

Una differenza sostanziale con altri sistemi che illustrerò più avanti è che Cordova non fornisce nessun widget grafico o framework in generale, è esclusivamente un "browser avanzato".

Nel nostro utilizzo di Cordova siamo però incappati in un problema di fondamentale importanza, ci costringe a dover utilizzare una versione di html+css+js molto vecchia, senza la presenza di avanzate funzionalità come fetch, promise, grid layout ecc…

Il problema è che il plugin di Cordova per poter inserire l'engine di un browser dentro l'app è deprecato, e non viene aggiornato da parecchi anni, e su Android questo è un notevole ostacolo poichè la maggioranza dei dispositivi non riceve nessun aggiornamento del sistema operativo e quindi abbiamo una notevole installed-base di Android 4.4, che è l'ultima versione che non contiene al suo interno una versione auto-aggiornante di Chrome.

Mentre con la versione 5 è possibile aggiornare il browser all'ultima release disponibile su tutte le piattoforme, con conseguente presenza di tutte le api aggiornate, la 4.4 di Android rimane ancorata al browser engine interno del momento delle release, e quindi il mancato aggiornamento del plugin che fornisce un motore completo sta notevolmente limitando le possibilità in fase di sviluppo. Inoltre da fine 2019 sarà obbli-

gatorio fornire una versione dell'app in ARM-64 Bit su Google Play Store, e questo plugin deprecato è fornito solo a 32 bit, quindi siamo alla ricerca di un sostituto.

La mancanza di questi strumenti di web moderno è uno dei più grossi limiti attuali dell'applicazione, obbligando gli sviluppatori a dover costruire hack da se, o usare polyfill per ovviare a queste mancanze.

Flutter

Flutter è un framework che permette di poter generare applicazioni native, e web, per la maggior parte dei sistemi operativi. Al contrario di soluzioni ibride come Cordova, Flutter compila il codice sorgente scritto dallo sviluppatore in codice nativo per le piattaforme, ed in Javascript per le versioni da rilasciare su web. Il suo utilizzo permette di poter scrivere il codice una singola volta e realizzare un interfaccia unica, che verrà estesa in base alla piattaforma ed alla dimensione dello schermo. Il sostanziale svantaggio è che un applicazione Flutter viene scritta in un nuovo linguaggio di programmazione, chiamato Dart.

Flutter e Dart sono anch'essi stati creati da Google, per quanto siano in diretta competizione con Angular. Il linguaggio è abbastanza simile a livello di sintassi a Java, ma le similitudini si fermano li. La differenza maggiore tra i due è che Dart è molto improntato all'utilizzo di operazioni asincrone, con non tante differenze da come Javascript cerca di approcciare questo dilemma architetturale, come ad esempio l'utilizzo della semplice sintassi async/await.

L'impronta asincrona di Flutter nascono dalla necessità di una applicazione moderna di dover dialogare costantemente con delle Rest API o con un database remoto.

Flutter fornisce inoltre una completa libreria di widget UI, e tutta la struttura della view di un app in Flutter è semplice una combinazione di più widget uni con gli altri, in

un approccio dichiarativo che ha più similitudini con React piuttosto che con il lontano parente Angular. Essendo un prodotto Google questa libreria è stata progettata secondo i dettami del Material Design, ed infatti sono presenti anche ulteriori funzionalità che la maggior parte delle librerie material forniscono, come un avanzato sistema di Theming.

Fino alla versione 1.5, rilasciata qualche mese fa, Flutter supportava solo ambienti mobile; da qualche mese è disponibile una tech preview di Flutter per il web, un SDK a se stante che ha come obiettivo quello di permettere alla stessa identica app di essere compilata sia in codice javascript, ed essere raggiungibile da un comune browser, sia in app nativa. Non ci sarà quindi nessun bisogno di creare una nuova struttura delle view, con HTML e CSS, ma sarà il compilatore a generarla per noi partendo dai widget delle libreria.

Flutter però non può essere oggi una soluzione viabile per Scloby. Bisogna considerare come detto che la versione web è solo una beta, non è un sistema ancora utilizzato granché nella comunità degli sviluppatori di front-end, e ciò comporta il rischio di ritrovarsi a dover utilizzare per anni una versione sostanzialmente di test di un framework.

Nel contesto di Scloby questi sono rischi operativi che non possono correrси, vista anche l'esperienza abbastanza negativa nell'utilizzo di angular.js, scelto solo pochi mesi prima della sua sostituzione con il nuovo più moderno e funzionale Angular 2.

Va comunque tenuto presente che Flutter è una piattaforma su cui Google sta ponendo molto interesse e risorse, e considerando la potenza di questa società, e il controllo che possiede sulle piattaforme tecnologiche maggiori, Android e Web, e sui relativi tool di sviluppo, Android SDK e lo stesso Angular, l'impatto di un nuovo paradigma di creazione di app, che sia unificato per tutti questi ambienti non è da trascu-

rare, ed è pensabile secondo i maggiori analisti del mercato che in un futuro prossimo la società americana spinga maggiormente verso l'integrazione tra tutte le sue piattaforme, ed in questo contesto Flutter e Dart saranno centrali. Ma stiamo parlando di prospettive a medio-lungo termine che non possono inficiare la scelta del futuro stack tecnologico di Scloby.

React Native e Native Script

È possibile realizzare una versione totalmente nativa di un app, in maniera cross-platform, anche utilizzando gli stessi framework web illustrati precedentemente, senza quindi dover far imparare ai dipendenti ulteriori sistemi.

React Native è un set di tool per effettuare questa operazione per il framework di Facebook, mentre Native Script è utilizzabile partendo da codebase sia Angular, ma anche Vue.js oppure custom con applicazioni realizzate semplicemente in HTML+CSS+JS.

```
<ActionBar title="Home" class="action-bar">
</ActionBar>

<GridLayout>
   <ScrollView class="page">
      <StackLayout class="home-panel">
         <!--Add your page content here—>
         <MapViewComponent></MapViewComponent>
         <Label textWrap="true" text="On iOS, a React Native ScrollView uses a native UIS-
crollView.
        On Android, it uses a native ScrollView.

        On iOS, a React Native Image uses a native UIImageView.
        On Android, it uses a native ImageView.

        React Native wraps the fundamental native components, giving you
        the performance of a native app using the APIs of React.”>
         </Label>
         <Slider value="80"></Slider>
      </StackLayout>
   </ScrollView>
</GridLayout>
```

ESEMPIO DI CODICE NATIVE SCRIPT

```
import React, {Component} from 'react';
import {Image, ScrollView, Text} from ‘react-native';
import {MapViewComponent} from './your-native-map-implementation';

class ScrollingImageWithText extends Component {
  render() {
    return (
      <ScrollView>
      <MapViewComponent />
        <Text>
          On iOS, a React Native ScrollView uses a native UIScrollView.
          On Android, it uses a native ScrollView.

          On iOS, a React Native Image uses a native UIImageView.
          On Android, it uses a native ImageView.

          React Native wraps the fundamental native components, giving you
          the performance of a native app using the APIs of React.
        </Text>
      </ScrollView>
    );
  }
}
```

ESEMPIO DI CODICE REACT NATIVE

Sono tanti i vantaggi di usare questo approccio, ma l’utilità maggiore proviene dalle ovvie potenzialità garantite da performance native, la possibilità di integrare codice nativo scritto per il sistema operativo di esecuzione, direttamente usando plugin o api mischiati al proprio codice javascript, e l’utilizzo di interfacce perfettamente integrate con il sistema ospite, realizzando il front-end con i widget messi a disposizione da React Native o Native Script che in base al sistema verso il cui fare il deploy vengono automaticamente sostituiti dalle UI standard. Ovviamente Material su Android e UIKit in iOS.

Concettualmente le due tecnologie sono molto similari, possiamo prende in esempio il codice di un componente in React Native, e banalmente l’unica cosa che cambia da una normale View di React è proprio l’utilizzo di questo layer di widget che già dal nome si capisce siano una trasposizione di componenti grafici di Android o iOS (ScrollView, Text ecc…).

Per poter utilizzare API native, di cui non è disponibile una implementazione diretta in React Native, basta creare un modulo in codice nativo per ogni piattaforma di

```
#import "CalendarManager.h"
#import <React/RCTLog.h>

@implementation CalendarManager

RCT_EXPORT_MODULE();

RCT_EXPORT_METHOD(addEvent:(NSString *)name location:(NSString *)location)
{
  RCTLogInfo(@"Pretending to create an event %@ at %@", name, location);
}

@end
```

```
module CalendarManager {
    header "CalendarManager.h"
    export *
}
```

```
import {NativeModules} from 'react-native';
var CalendarManager = NativeModules.CalendarManager;
CalendarManager.addEvent('Birthday Party', '4 Privet Drive, Surrey');
```

deploy, che faccia da bridge tra le api degli SDK dei sistemi operativi e il codice javascript, a quel punto si importa questo modulo, che utilizzando i tool CLI di React Native viene compilato esponendo delle api in javascript che permettono la sua importazione dentro il codice dell'app React (MapViewComponent nell'esempio, un bridge con MKMapView in iOS e GoogleMap in Android).
CalendarManager è una implementazione per React delle API del calendario su iOS, utilizzando Objective-C vengono esportate le singole funzionalità creando un metodo dentro il modulo esportato, che poi verrà utilizzato come un normale import in Javascript

La stessa implementazione in Native Script viene scritta, piuttosto che con delle funzione RCT_EXPORT_* dentro il codice, con un file aggiuntivo da aggiungere nello

stesso path del codice sorgente del modulo che semplicemente fornirà indicazione al compilatore di dove prendere le api esposte
Dentro il file TS del componente potremo usare CalendarManager.alloc().addEvent(… , …) per accendere alla funzionalità.

Essenzialmente React Native e Native Script non possono, e non devono essere, una discriminante per poter scegliere o meno per uno dei due framework, sono solo dei tool che ci permettono di raggiungere entrambi lo stesso scopo, ed infatti entrambi i sistemi posseggono gli stessi pregi e difetti dei loro genitori.

Ad esempio a livello di performance React Native riesce a fornire applicazioni compilate meno voluminose in termini di spazio fisico e di memoria occupata, rendendo animazioni ed app complesse molto performanti. E la community della versione native script è di dimensioni inferiori, cosi come lo era quella di angular.

C'è anche da considerare che React Native è gestito ed utilizzato da Facebook in prima persona, che ne è anche il maggior utilizzatore; infatti tutte le app mobili del colosso IT sono realizzate con questa tecnologia, ciò permette di avere un sistema testato in ambienti di utilizzo massivo, oltre che un perfetto accoppiamento con il framework UI da cui ha origine.

Dall'altro lato invece Native Script è un prodotto open source fornito da una azienda terza, Progress, e dalla community, che Google appoggia e supporta, ma nello sviluppo del quale non ha un ruolo prominente, ponendo dubbi sulle prospettive a lungo termine.

L'utilizzo di queste due tecnologie ha comunque uno svantaggio per Scloby: i clienti sono stati formati ad utilizzare un app che utilizzi un interfaccia grafica material su tutti i dispositivi, mentre questi tool forniscono un app nativa che possieda l'interfaccia grafica del sistema operativo di destinazione, e che quindi utilizzi UIKit su iOS.

Potrebbe essere necessario dover formare di nuovo tutti i clienti all'utilizzo di questa nuova major release, mentre utilizzare un approccio tradizionale web consente di poter scegliere una libreria grafica a proprio piacimento.

Abbandonare il Web: Xamarin

Creare una applicazione web permette quindi la sua fruibilità su una vasta gamma di dispositivi, ma per quanto i sistemi di trasposizione di codice javascript in nativo possano essere efficienti e potenti un applicazione progettata interamente usando gli SDK dei sistemi operativi sarà sempre preferibile. Ovviamente nella maggior parte dei casi per un rapporto costi/benefici non è una soluzione conveniente, poiché dovremmo realizzare differenti codebase, ma vorrei porre comunque attenzione su alcune tecnologie che permettono di ridurre il numero di ambienti differenti su cui progettare un applicazione.

Xamarin è un'azienda di proprietà di Microsoft che ha progettato nel corso degli anni, prima e dopo la sua acquisizione, una serie di bridge, ed i relativi tool di sviluppo, per poter usare le api .NET, il linguaggio C#, e tutto lo stack di programmazione di ambienti Windows su iOS e Android. Si può pensare a Xamarin come un sistema che utilizza un approccio posto a metà tra React Native/Native Script ed soluzioni quali Cordova.

L'applicazione è composta da un nucleo che utilizza API fornite da una serie di librerie tra le più importanti disponibili per Windows, che sono state adattate agli ambienti di target supportati da Xamarin. Sopra questo layer, che ricorda vagamente i plugin e le api fornite dagli ambienti ibridi quali Cordova, viene posto uno strato di api univoche per ogni ambiente; infatti per quanto sia possibile cercare di creare delle api standard, molte funzionalità di iOS o Android sono mancanti nell'altro sistema (un esempio pos-

sono essere framework secondari quali HealthKit su iOS oppure Google Maps su Android).

Infine viene creata l'interfaccia grafica partendo da codice condiviso con tutte le implementazioni che durante la compilazione dell'app per un dato ambiente viene sostituito dal framework grafico disponibile sulla piattaforma, UIKit su iOS e Material su Android.

Queste interfaccia grafiche possono essere composte usando la libreria Xamarin Forms, che fornisce una sintassi XAML (xml-like) per poter comporre e disporre i controlli ia piacimento, con l'utilizzo anche, negli ide di sviluppo supportati, di un sistema WYSIWYG.

Ci sono però alcuni difetti che rendono Xamarin non perfettamente utilizzabile in tutti i contesti, e specialmente in una impresa come Scloby non è possibile soprassedere ad alcune limitazioni.

Innanzitutto il supporto a release recenti di iOS e Android non è immediato, in quanto gli sviluppatori del framework necessitano di maggiori tempistiche per poter adattare tutte le librerie a possibili novità e bug introdotti ad ogni release, e per un prodotto, come Scloby, dove non si ha un processo di deploy di nuovi aggiornamenti dei sistemi operativi controllato, poiché i clienti possono a loro piacimenti aggiornare immediatamente i loro dispositivi, è un rischio notevole.

Inoltre Xamarin presenta dei costi non indifferenti per il suo utilizzo, poiché se escludiamo la community edition, che però nella licenza prevede l'impossibilità di utilizzo commerciale, sono richiesti migliaia di euro di licenza per usare Xamarin, e visto le limitazioni di budget sono costi non sono immediatamente giustificabili, specialmente visto il lavoro di adattamento da fare a monte.

Xamarin è però un sistema molto utilizzato in ambienti corporate dove queste due pecche non hanno un grosso impatto sulle scelte operative, e dove spesso si utilizzano già molti servizi offerti da Microsoft, quali ambienti cloud, office, o server windows, e dentro queste aziende si possono quindi già trovare dei team di sviluppo che hanno conoscenza con le tecnologie impiegate da questo framework.

Abbandonare il Web: Catalyst e SwiftUI

Il 60%-70% della base installata di Scloby utilizza come punto di accesso iOS oppure macOS, attraverso le applicazioni disponibili su App Store o scaricabili dal sito web. In un ottica di una nuova applicazione non sarebbe possibile abbandonare totalmente quella consistente fetta di clienti che non possiedono una di queste piattaforme, ma sono di recente annuncio alcune novità per i sistemi operativi di Apple che sono molto interessanti e che in prospettiva futura potrebbero intaccare notevolmente la necessità di utilizzo di sistemi di cross-compiling o di creazione di app partendo dal web. Sto parlando SwiftUI e Project Catalyst, due tecnologie introdotte in iOS 13 e macOS Catalina.

SwiftUI è un nuovo linguaggio DSL per poter progettare un frontend per tutti i sistemi operativi forniti da Apple: iOS, macOS, iPadOS (un fork di iOS annunciato in contemporanea al framework), watchOS e tvOS. Lo stesso file genera interfaccia grafiche equivalenti in tutti gli ambienti di deploy (con le dovute limitazioni, alcuni widget non sono presenti ovunque), e ciò permette di gestire un unica codebase per tutte le interazioni utente. Considerando anche che la maggior parte dei framework di sistema sono disponibili ormai in tutte le piattaforme apple in maniera condivisa c'è quindi la possibilità di unificare tutte le applicazioni.

SwiftUI essendo un sistema DSL sfrutta il paradigma dei linguaggi dichiarativi, questo renderà lo sviluppo di una interfaccia molto più semplice rispetto agli attuali

framework che invece sono puramente imperativi, e ci sono molte somiglianze con React, da cui gli sviluppatori Apple hanno ammesso in alcune dichiarazioni di essersi ispirati, poichè saranno presenti concetti come stato, binding, routing, che ho già trattato in precedenza.

SwiftUI è comunque un sistema acerbo, e molte delle sue componenti interne sono ancora basate, e quindi limitate, dai framework grafici attualmente di default per questi OS, ovvero UIKit, AppKit, WatchKit e TVKit. Ma realizzando le proprie interfaccia con questo nuovo linguaggio si supporteranno già automaticamente tutte le novità future per questi sistemi, in quanto gli sviluppatori del produttore potranno automaticamente cambiare le componenti interne delle api, e le viste renderizzate dai widget in maniera indipendente, avendo un layer di mediazione tra il nostro codice e quello di rendering del sistema operativo

Project Catalyst è invece un sistema, già pronto per poter essere utilizzato in ambiente di produzione, che permette di far eseguire le applicazioni progettate per iPad sul sistema operativo macOS, ed automaticamente saranno le api fornite dal sistema a adattare tutta l'interfaccia, progettata nativamente per il touch, ad un ambiente che utilizza mouse e tastiera. Per esempio verranno adattata le dimensioni del testo e delle immagine, in quando un ambiente desktop necessità di dimensioni minori per poter effettuare una operazioni di input, mentre un dito è un dispositivo meno preciso rispetto ad un mouse.

Tutte le librerie disponibili su iOS, ad eccezione di quelle prettamente mobile come i sensori del dispositivo, l'interfaccia telefonica ecc…, sono state unificate con quelle di macOS, e questo non solo permetterà l'utilizzo di una serie di funzionalità avanzate prima non presenti in ambiente desktop, ma garantirà con risorse limitatissime la presenza l'app da noi realizzata per tablet anche su desktop.

Conclusioni

Dopo tutte queste considerazioni ed analisi, e dopo anche aver discusso di alcune proposte sicuramente non praticabili, ma interessanti per capire l'andamento futuro delle realizzazioni di interfacce grafiche, è giunto il momento di tirare la fila. Cosa consiglio alla mia azienda come direzione futura? Quale opzione garantisce la soddisfazione di tutti i requisiti richiesti? La scelta come detto sta sicuramente al management aziendale, ma dalla mia esperienza posso cercare di fornire un consiglio, una indicazione, che poi potrà essere seguita o meno.

L'app dovrebbe rimanere un applicazione web, questo perchè permette una maggiore penetrazione su tutte le diverse piattaforme, esponendo delle interfacce con cui l'utilizzatore può trovarsi a suo agio, specialmente utilizzando le tecniche di compilazione nativa di codice javascript presenti in entrambi i framework proposti in questa tesi.

I costi di formazione per poter scegliere un framework alternativo sarebbero notevoli, il team ha sempre e solo avuto esperienza lavorativa e professionale con Angular.js ed il web, e cambiare radicalmente l'architettura e l'approccio non è facile. Servirebbero corsi di aggiornamento professionale ed un periodo di adattamento che l'azienda non può permettersi.

Inoltre bisogna considerare come un codice javascript sia perfettamente compatibile con qualsiasi browser prodotto ora o in futuro, raramente ci sono funzionalità che vengono deprecate, e quindi parecchie parti del codice di una applicazione web necessitano di bassa manutenzione, e questo codice può essere eseguito su miliardi di dispositivi. Scegliere una soluzione totalmente nativa ci costringerebbe a dover costan-

temente cercare soluzioni alternative per sistemi operativi deprecati, dispositivi non compatibili, e cambiamenti radicali dell'architettura.

C'è però una pecca grave per l'utilizzo di un framework web che è il sistema di debugging. Nessuna delle soluzione proposte, React o Angular, garantisce dei sistemi di debugging avanzati tanto quelli di un codice nativo. Questo è un costo nascosto per questa tipologia di prodotti, e bisogna tenere in considerazione le possibili ore lavorative perse per risolvere insidiosi bug.

Per questi motivi la scelta è essenzialmente tra React ed Angular. Scloby è attualmente sviluppato, come detto, in Angular.js. Io ed i miei colleghi siamo stati formati nell'utilizzo di questo framework in mesi, o anni, di esperienza sul campo. Ed Angular 2+ fornisce la possibilità di cominciare ad essere usato velocemente in maniera avanzata ed approfondita se si parte dal background di sviluppatore della versione 1. Ed esistono tool, come ho illustrato, per potersi integrare con versioni precedenti, riducendo in maniera drastica le complicazioni di un passaggio di tecnologia.

React è un ottimo prodotto, ho sviluppato personalmente moltissime applicazioni con questa libreria, ma non consiglio il suo utilizzo futuro in Scloby allo stato attuale, per quanto lo ritenga più facile da utilizzare ed in base ai benchmark più performante. Ma questa differenza non è cosi enorme da dare giustificazione al suo utilizzo presso l'azienda.

Ritengo che Angular sia la scelta più sicura, in quanto il settore tecnico potrebbe facilmente applicare concetti già interiorizzati negli scorsi 4-5 anni di lavoro con la versione 3 di Scloby in un nuovo prodotto basato su Angular 2+; la possibilità di realizzare un app ibrido è un fattore non da trascurare in questa scelta.

L'unico caso in cui sia conveniente l'impiego di React è quello in cui si cominci a sviluppare progetti collaterali all'app principale utilizzandolo, cosi da poter formare il

team di sviluppo, senza contemporaneamente sprecare risorse aziendali. Ma nel contesto di Scloby ritengo che questo non sia possibile, visti gli elevati costi di context-switch che il lavoro parallelo su molte applicazioni con tecnologie differenti comporterebbe, ed il relativo calo di produttività.

È però tassativo l'utilizzo di Angular accoppiato a Native Script; i benefici di avere un app nativa di Scloby sono notevoli, e l'aumento di prestazioni dovuto a questa nuova versione potrebbe essere anche un notevole strumento a disposizione del settore marketing.

Le limitazioni dovute ai requisiti di accesso a dispositivi esterni, come le stampanti o lettori di codici a barre possono essere superati esattamente come attualmente viene fatto con l'applicazione in Angular.js . Certo, non è la soluzione migliore fornire supporti differenti per piattaforme differenti, ma considerando come l'utilizzo di un framework web con React Native o Native Script permetta l'acceso diretto ad api del sistema Scloby potrebbe cercare di puntare a vendere esclusivamente un prodotto per le piattaforme mobile utilizzando questi tool. Si può immaginare un futuro dove tutta l'interazione con il processo di vendita sia eseguita solo su tablet e telefoni, con la versione web esclusivamente delegata fare da macchina di visualizzazione di documenti di riepilogo, o di configurazione delle impostazioni, listini, o prodotti.

Inoltre con le api dei sistemi operativi si possono utilizzare la fotocamera del dispositivo al posto di un lettore ottico, o l'accesso a vere e proprie connessioni tcp verso i vari dispositivi esterni, ed inoltre ci viene garantita una maggiore integrazione con lo strumento in mano al cliente, potendo sfruttare le potenzialità di un hardware che altrimenti con il solo browser web sarebbe solo un schermo che risponde a dei touch.

Con l'utilizzo di Native Script c'è però da considerare un aggravio di costi di assistenza per la formazione dei clienti, che si ritroverebbero con una interfaccia grafica

diversa da quella originale. Sicuramente ci sarebbe un aumento dei ticket da gestire, e delle problematiche da risolvere.

La maggior parte delle applicazioni hanno comunque dei cicli di refresh della propria interfaccia, gli stessi sistemi operativi più importanti come iOS, Windows o Android ogni 5-6 anni introducono novità che mettono a disagio l'utente finale, riducendo la sua produttività durante la fase di apprendimento dei nuovi pattern di interazione.

Questa continua evoluzione è comunque necessaria per mantenere il prodotto interessante a livello marketing ed aggiornato con le più recenti novità dell'ambito del design e della UX; il settore dei punti cassa ed in genere quello delle applicazioni per il commercio al dettaglio presenta un notevole deficit in questo ambito, e sarebbe quindi una occasione da poter sfruttare.

Un interfaccia che sia il più nativa possibile è un approccio comunque valido poichè garantisce il riuso di pattern di utilizzo che l'utente può apprendere durante l'uso di ulteriori applicazioni sui suoi dispositivi. Per quanto ciò comporti un maggiore carico di assistenza nel primo periodo dopo la release di una nuova interfaccia, sul lungo periodo i vantaggi di integrazione e usabilità vanno a superare gli svantaggi introdotti da un cambio cosi radicale.

Ho provato in questa tesi a fare un riassunto dei miei mesi in Scloby e fornire un mio ulteriore contributo allo sviluppo aziendale prima di terminare la mia esperienza. Sono stati mesi formativi, mesi che mi hanno consentito di imparare come si lavora in una azienda, come si risolvono problemi non di facile interpretazione, ed è stato un periodo interessante, bello, ed impegnativo.

Sono “entrato dentro gli ingranaggi” di una piccola realtà italiana, ed internazionale. Spero che Scloby abbia il successo che si merita e che possa continuare a modificare il rapporto che gli esercenti hanno con il proprio business.

Ringrazio con cuore chi mi ha dato questa possibilità, chi mi ha aiutato a crescere, e chi mi ha dato una mano nei momenti difficili di questi mesi, ma anche di questi anni universitari, che giungono con questa tesi alla loro conclusione. Un capitolo della mia vita si chiude, ma mi si spalancano le porte del mondo del lavoro al suo completo, e spero di poter dare anche io un contributo a portare avanti software che abbiano un impatto concreto su come interagiamo con il mondo circostante, cercando di non accontentarmi mai.

Indice