UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

PROGRESSIVE WEB APPLICATION E SISTEMI OPERATIVI WEB

PROGRESSIVE WEB APPLICATIONS AND WEB OPERATING SYSTEMS

CHERUBINI MATTEO

Relatore: *Pugliese Rosario*
Correlatore: *Tiezzi Francesco*

Anno Accademico 2020-2021

INDICE

ELENCO DELLE FIGURE

1

INTRODUZIONE

In questa tesi di laurea presento il moderno concetto di Progressive Web Application (PWA), la loro diffusione e il peso che stanno avendo sui sistemi operativi basati su Cloud Computing e descrivo il lavoro da me svolto nella progettazione di una PWA per la creazione di documenti di testo e nel relativo sviluppo seguendo i canoni che contraddistinguono questa tecnologia.
Infine illustro alcune mie scelte e considerazioni in merito.

In primo luogo introduco quello che è il contesto storico nel quale nascono le PWA e come personalmente mi sono avvicinato a questa tecnologia.

Durante la presentazione della messa in commercio dell'iPhone nell'estate del 2007, Steve Jobs, rivolgendosi agli sviluppatori, accennò alla versatilità che le applicazioni scritte interamente con gli standard della programmazione Web potevano avere con i servizi offerti dal telefono[1].
Da allora iniziò un progressivo miglioramento degli strumenti per lo sviluppo Web, portando l'attenzione anche su stile e design di prodotti quali CSS Grid e FlexBox o Google Web Fonts.

Successivamente, nel 2010 Ethan Marcotte, un Web designer, scrisse un famoso articolo[2] nel quale presentò un modo di gestire in maniera fluida e reattiva i contenuti dei siti Web statici, così da renderli adattabili alle dimensioni degli schermi dei vari dispositivi sul mercato e alla rapida e inarrestabile evoluzione di questi, introducendo il concetto di Responsive Web Design, fondamento dei layout dei siti Web dinamici ed interattivi che vediamo oggi.

Tutte le evoluzioni nel modo di costruire siti Web hanno aiutato l'inserimento in questi di applicazioni Web e widget, ma la vera nascita delle Progressive Web App è da contestualizzare in un mondo ancora più avanzato in cui le rivoluzioni riguardo alle potenzialità dei dispositivi

mobili sono ormai sature e le vere necessità del mercato tendono quasi esclusivamente all'esperienza utente.
Siamo in anni di cambiamento nel modo di sviluppare siti Web, sempre più dinamici e pratici da usare, anni in cui si sono consolidati i framework e le librerie JavaScript e CSS come React nel 2013 o Material Design nel 2014 e si è adottato come standard l'ultima versione del linguaggio di marcatura HTML5.

Con la nascita delle Progressive Web App nel 2015 si arriva allo sbocco naturale del crescere continuo di questi strumenti e delle richieste del mercato portando i siti Web dinamici a fondersi con le applicazioni native, ottenendo fin da subito ottimi risultati sia per l'esperienza nell'utilizzo da parte degli utenti sia per le aziende che registrano maggiori visite e minori costi di sviluppo.

In sintesi, l'obiettivo della tesi è quello di contestualizzare e mostrare l'utilità delle Progressive Web App al giorno d'oggi, spiegando le motivazioni dietro al successo che questa tecnologia sta avendo.
I browser tendono sempre più a comportarsi come sistemi operativi online, offrendo suite di servizi e applicazioni tramite cloud ad un'utenza globale (Chrome OS di Google è esemplificativo in tal senso).
Le PWA nascono per utilizzare queste potenzialità dei browser anche offline, mantenendo velocità, sicurezza e semplicità d'uso, impattando il meno possibile su hardware e memoria.
Il progetto, che conclude la tesi con un'applicazione dimostrativa, vuole presentare la praticità di alcuni moderni strumenti per la progettazione e lo sviluppo di PWA.

Personalmente l'interesse per l'argomento mi è venuto quando, ad inizio 2018, lessi un articolo su Medium, la nota piattaforma di pubblicazione online, scritto da Max Lynch, cofondatore del framework Ionic, intitolato "Building the Progressive Web App OS"[3].

Nell'articolo, Lynch spiega il lavoro svolto dall'azienda in merito alla creazione di componenti Web compatibili per vari dispositivi e sistemi operativi, in quanto eseguiti da browser Web. Si tratta dello sviluppo di una libreria di oggetti ed elementi che vanno a definire l'interfaccia grafica delle Web app, scritti con i linguaggi canonici per la Programmazione Web quali JavaScript, HTML e CSS.

Obiettivo di Ionic è appunto quello di sopperire alle limitazioni dei browser, rendendo funzionali ed efficaci questi componenti e mantenendo

una vasta libreria adatta in particolare a quelle applicazioni Web che tendono a comportarsi come applicazioni native: le Progressive Web App.

Infatti nell'articolo si accenna ad un lavoro in corso, riguardante la creazione di una fotocamera, iniziando il discorso con la seguente frase:

"The browser doesn't give you a "Camera App," it just gives you the low-level primitives that make it possible to build your own."[3]

che marca la sostanziale differenza tra un sistema operativo, il quale definisce utilità e componenti ad alto livello "native", e un browser, il quale mette a disposizione le primitive a basso livello che, anche se implementabili, non risultano efficaci allo stesso modo.

Mi ha quindi stupito ed affascinato vedere come negli anni, mentre seguivo le evoluzioni e gli sviluppi da parte della comunità attorno a tale argomento, i browser venissero sviluppati tendendo a diventare veri e propri sistemi operativi "in miniatura".
In particolare mi ha interessato seguire il lavoro svolto in tale direzione da Google per il suo browser e soprattutto per il suo sistema operativo desktop Chrome OS, avviando una progressiva conversione prima di alcuni software, offerti dalla suite Google Workspace come Meet e Maps, e successivamente di alcune applicazioni di sistema, quali ad esempio la Fotocamera e ultimamente la Calcolatrice, in PWA.

I restanti contenuti trattati all'interno della tesi sono suddivisi nei seguenti capitoli:

- Capitolo 2 "Web e architettura CC":
  una descrizione dell'ambiente Web e Cloud Computing e dei sistemi operativi che si sono sviluppati negli ultimi anni grazie a queste tecnologie, con una descrizione del progetto Chrome OS;
- Capitolo 3 "Progressive Web App":
  la definizione di Progressive Web App e un'analisi dei vari framework e strumenti che ne permettono lo sviluppo;
- Capitolo 4 "Il Progetto":
  la descrizione del progetto di sviluppo di una PWA al quale ho lavorato, degli strumenti che ho scelto di usare ed infine il link al progetto terminato;
- Capitolo 5 "Considerazioni Finali":
  mie considerazioni finali riguardo allo sviluppo e al futuro del progetto.

2

WEB E ARCHITETTURA CLOUD COMPUTING

In questo capitolo introduco l'ambiente Web moderno e gli strumenti utilizzati quotidianamente dalla maggior parte dell'utenza media dei servizi Internet, in particolare le infrastrutture Cloud "as-a-Service", che hanno portato in questi ultimi decenni importanti rivoluzioni nel modo di fruire o fare business con Internet, discutendo infine, come caso di studio, il sistema operativo Chrome OS di Google.

2.1 GLI STRUMENTI DEL WEB

Svariate imprese basano il loro modello di business sulle capacità della moderna **architettura Cloud Computing**[4] per le loro applicazioni, la quale permette interessanti caratteristiche per il servizio utente: accesso sempre e ovunque praticamente illimitato a risorse quali Network, Server, Archiviazione e Servizi.
Ciò è dovuto alle moderne reti di comunicazione grazie alle loro caratteristiche di elevati throughput e velocità di trasmissione e al concetto di **archiviazione dei dati**, che permette di garantire elevata affidabilità e disponibilità di questi.

La **virtualizzazione**[5], altro concetto fondamentale, è alla base del cloud computing, in quanto permette una rappresentazione logica delle risorse fisiche rendendole disponibili al sistema operativo sottostante.

- La **virtualizzazione server** è un processo che permette di suddividere un server fisico in più server virtuali. Ogni server virtuale può eseguire le proprie applicazioni e il proprio sistema operativo, incrementando così sia la disponibilità del server che le prestazioni delle applicazioni.

- La **virtualizzazione desktop** è un metodo per simulare un ambiente di lavoro rendendolo accessibile da un qualsiasi dispositivo connesso in remoto.

- La **virtualizzazione delle applicazioni** isola la singola applicazione dal sistema, non astraendo l'intero desktop (sistema operativo e applicazioni), semplificando così gli aggiornamenti e la disinstallazione, e rendendo l'applicazione eseguibile da qualsiasi endpoint, sia esso Windows, iOS o Linux/Android, seguendo il concetto di **Bring Your Own Device** (BYOD).

Le ripercussioni a cascata comportano:

- dal **lato utente** una enorme semplificazione della manutenzione dei singoli dispositivi, come appunto mantenere aggiornato il sistema operativo e quindi le applicazioni;

- a **livello hardware** un rallentato invecchiamento dei pezzi, in quanto l'utente non avrà bisogno di nuovo hardware dedicato dato che il lavoro verrà eseguito principalmente dal server;

- a **livello di produzione industriale** il relativo cambiamento del modello di business, in merito alla costruzione e all'utilizzo di Datacenter.

I servizi di cloud computing offrono quindi un modo per delegare la gestione dell'infrastruttura al fornitore stesso, e non più interamente al cliente.
La **delega della gestione** comporta una suddivisione in tre tipologie principali di cloud computing: IaaS, Paas e SaaS[6].

- **Infrastructure-as-a-Service**, o **IaaS**, è il tipo di servizio più flessibile e permette all'utente di avere il massimo controllo sulle funzionalità e sulla componentistica dell'infrastruttura, mentre al provider è delegata la gestione di accesso, la virtualizzazione e l'archiviazione sicura dei dati;

- **Platform-as-a-Service**, o **PaaS**, permette all'utente di lavorare su una piattaforma già montata e completa, interamente mantenuta dal provider, dove può dedicarsi esclusivamente allo sviluppo e gestione di software;

- **Software-as-a-Service**, o **SaaS**, è il tipo più completo riguardo al servizio proposto e anche il più utilizzato. Consiste nella fornitura di una applicazione che il provider manterrà aggiornata, integra e sicura. All'utente resterà l'accesso all'applicazione e, ovviamente, l'utilizzo. Rispetto alle altre tipologie, l'utente non deve installare alcun software sul proprio dispositivo.

La centralità che Internet ha assunto grazie anche a questi servizi ha contribuito allo sviluppo di un nuovo mercato: quello dei **Sistemi Operativi Web-Based**[7] o Web OS.
In questa definizione si ritrovano tutti quei servizi che offrono un'interfaccia grafica all'utente permettendogli di avere accesso alle applicazioni completamente, o parzialmente, mantenute sul Web, mimando l'operabilità di un sistema operativo classico. Salvare informazioni in una applicazione comporta archiviarle direttamente in un database connesso a Internet. La perdita di dati per qualsiasi motivo è sempre più infrequente. Il peso sulle capacità dei componenti del dispositivo è ridotto al minimo, permettendo di utilizzare dispositivi vecchi (per gli standard moderni) per molti più anni, mantenendo buone velocità ed efficienze della batteria.

Questo mercato ha promosso lo sviluppo di applicazioni Web, applicazioni scritte con i linguaggi canonici del Web e comunicanti con il cloud, alleggerendo quindi il peso sull'Hardware del dispositivo ma risultando alternative "economiche" con usi specifici e limitati.

2.2 CHROME OS

Un Web Operating System è una User Interface che, per funzionare, si appoggia e comunica su un sistema operativo tradizionale installato sul dispositivo.
Accedere al Web OS può avvenire o tramite un software, Client del servizio offerto dal provider, o tramite un browser Web.
Al provider è delegata la sicurezza, sia del database che dell'accesso dell'utente, e gli aggiornamenti delle applicazioni e del sistema.

Chrome OS è un sistema operativo Linux-based annunciato da Google nel 2009, sviluppato dal progetto open source **Chromium OS**[A] e poi reso proprietario dalla stessa Google, che basa il suo modello di business sulla migliore esperienza per un utente medio moderno, ottimizzando il workflow, quindi il modo di lavorare e di approcciarsi al sistema operativo, sfruttando i servizi cloud e la suite di applicazioni Web raggiungibili tramite il browser **Google Chrome**.

Parole chiave del progetto sono **Velocità** e **Sicurezza**, tutte componenti che appartengono come abbiamo visto ai sistemi operativi centralizzati al Web, che qui vengono sviluppate puntando all'utenza e quindi sulla **Semplicità d'Uso**.
Il successo in quegli anni dei Netbook, ovvero computer portatili minimali nelle componenti per l'uso esclusivo di Internet, ha inoltre portato l'azienda a investire su quella manifattura, creando un prodotto (conosciuto come **ChromeBook**) che valorizzasse il sistema operativo, creando un imponente ecosistema di servizi.

2.2.1 *Architettura del Sistema Operativo*

L'architettura di Chrome OS[8] si basa su tre componenti principali che vengono avviati sequenzialmente partendo dal **Firmware**, passando dai vari **Software di Servizio** e terminando con l'esecuzione del **Browser** e del gestore di finestre del Desktop.

> La **fase di avvio** [fig. 1], intesa dall'azienda come il periodo di tempo che occorre da quando clicchiamo il tasto di avvio sul computer a quando viene eseguito il browser Chrome, in un dispositivo Chromebook riesce, nelle versioni più moderne, in circa 6 secondi.

Questo è dato da una accurata selezione di strumenti di avvio per eseguire il **Kernel Linux** (progetti moderni e veloci come CoreBoot e U-Boot) caricando poi esclusivamente risorse e servizi essenziali all'avvio del sistema e quindi avviando in parallelo, in un secondo momento, le ulteriori applicazioni.

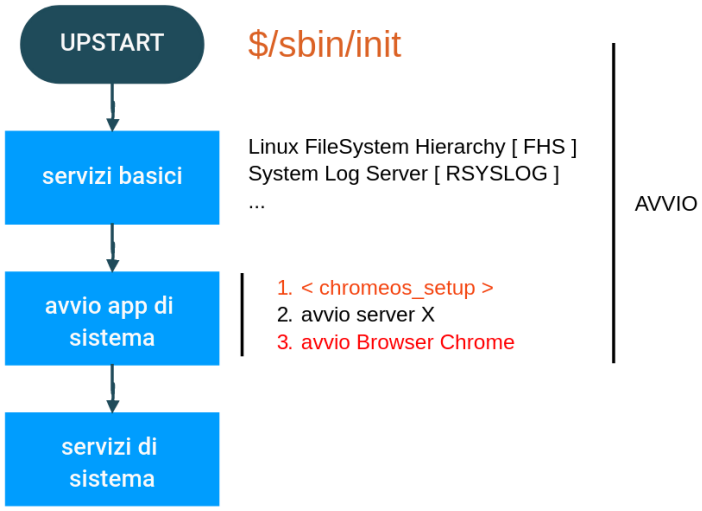

Figura 1: Avvio dei Servizi di Sistema in un Chromebook.

CoreBoot[B] è un firmware Open Source moderno per avviare (operazione di boot) le componenti minimali per assicurarsi che l'Hardware sia utilizzabile dal sistema operativo [fig. 2].

In caso positivo viene eseguito il così detto payload, che carica effettivamente il Kernel del sistema operativo. Nei ChromeBook viene gestito dal programma di Boot Loading soprannominato DepthCharge, un programma di Boot Loading basato sul progetto open source Das U-Boot.

A questo punto viene caricato il **Window Manager**, ovvero il responsabile delle interazioni Desktop-Utente, che permette la gestione e le animazioni delle finestre e degli spazi di lavoro aprendo infine la finestra del **Browser**.

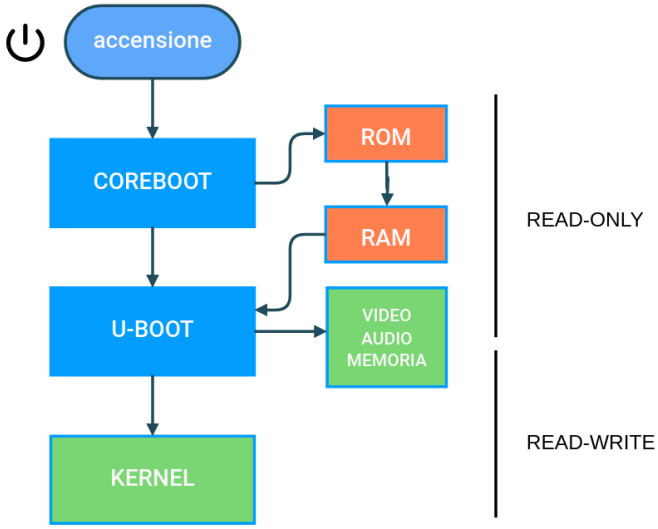

Figura 2: Firmware in un Chromebook.

2.2.2 *Chrome OS e le PWA*

Tra il 2015 e il 2016 Google definisce le Progressive Web App, portando alla nascita di tale standard di sviluppo che nei due anni successivi è stato ampiamente supportato da molte aziende del settore, da Microsoft ad Apple, dando a tale tecnologia sempre più notorietà, soprattutto nel mercato delle applicazioni per dispositivi mobili.

Aziende come Twitter e Pinterest sono state tra le prime a convertire i propri Social Network in PWA misurando fin da subito un notevole incremento nell'utilizzo da parte dell'utenza.

Nel 2019 Google annuncia ufficialmente[9] che la prossima versione del suo browser avrà supporto completo alle Progressive Web App su qualsiasi dispositivo, inclusi i ChromeBook, affiancando così allo sviluppo Mobile anche lo sviluppo Desktop di PWA.

Tra il 2020 e il 2021 Google annuncia[9] che le PWA possono essere aggiunte e listate sul Play Store, il negozio online di applicazioni per i dispositivi Google, rendendo le PWA più visibili grazie alla notorietà e capaci di accettare pagamenti interni.

Da questo momento nascono decine di progetti di Progressive Web App di dimensioni sempre più grandi. La tecnologia che inizialmente aveva reso più pratico ed efficace l'utilizzo di siti Web, come ad esempio i Social Network, sotto forma di applicazioni ora permette lo sviluppo di software completi come app di Disegno o di Video Editing, integrandosi maggiormente alle potenzialità del browser e alle capacità del sistema operativo.

Negli ultimi anni la stessa Google si sta distaccando dalle applicazioni native preinstallate nei Chromebook preferendo invece reinterpretarle sotto forma di PWA, portando così un solo codice per svariati dispositivi: basta avere un browser che le supporta.
La calcolatrice dei Chromebook ad esempio, preinstallata con il sistema operativo, è diventata dalla versione di Chrome 97 interamente una Progressive Web App, scaricabile quindi anche come estensione del browser. Benché non diversa dalle altre calcolatrici preinstallate nei nostri dispositivi, rimane un esempio interessante per capire la versatilità delle PWA.

Un dato altrettanto interessante rilasciato da Google Internal Data[10] è che da inizio 2021 fino ad oggi l'installazione e quindi l'utilizzo di PWA ha visto una crescita di oltre il 270%, dovuta anche alle elevate vendite di Chromebook durante tutto il 2020.

Tutto ciò denota quanto Google abbia egemonia in questo nuovo mercato e interesse nel mantenerla nel lungo periodo.

3

PROGRESSIVE WEB APP

Nel **2015**, durante l'annuale conferenza rivolta agli sviluppatori Web e Mobile "Google I/O" questo termine, coniato dai coniugi **Alex Russell** e **Frances Berriman**[11], fu usato per la prima volta per spiegare in poche parole un primordiale progetto ibrido tra un sito Web e una applicazione nativa chiamato "IOWA"[12].
IOWA era una **Single-Page Application**, cioè una pagina Web che si aggiorna internamente senza caricare altre pagine, progettata per essere scaricata e usata su dispositivi mobili, con la particolarità di funzionare anche offline, interamente sviluppata con linguaggi Web.

Anche se ufficialmente non esiste una vera e propria definizione di Progressive Web App, possiamo dire come introduzione che, con il termine PWA, ci si riferisce ad una moderna rivisitazione del concetto di applicazione Web che tende a comportarsi come una applicazione nativa, risultando sostanzialmente un sito Web con la capacità di essere installabile su diversi sistemi operativi.

Su una rappresentazione cartesiana, le applicazioni native tendono ad avere maggiori **funzionalità** in quanto si integrano al meglio con le potenzialità del sistema operativo, ma hanno spesso un peso notevole sul dispositivo, sia per memoria occupata che per processi in uso e devono essere mantenute dall'utente tramite aggiornamenti costanti. Un sito Web, invece, tende ad una maggiore **portabilità** e **semplicità**. [fig. 3]

Figura 3: Confronto PWA-Web-App.

3.1 STRUTTURA DI UN'APP NATIVA

Un'applicazione si definisce nativa quando viene sviluppata appositamente per un determinato sistema operativo, sul quale avrà pieno accesso alle funzionalità del medesimo, facendo sì che l'applicazione si comporti nel miglior modo possibile in termini di Reattività e Performance.

Gli strumenti di sviluppo di tali applicazioni vengono chiamati SDK, o **Software Development Kit**[13], forniti principalmente dal produttore del sistema operativo o di un linguaggio di programmazione specifico.
Un SDK include un Compilatore, un Debugger e ovviamente le API per lo sviluppo dell'applicazione, il tutto "limitato" al sistema operativo per il quale vogliamo sviluppare.
Questa dipendenza comporta che l'applicazione, una volta sviluppata, non sia Cross-Platform, e quindi non possa essere usata su altri sistemi e dispositivi finché non si ripeta lo sviluppo su un altro SDK, comportando un notevole peso tra costi e tempi di sviluppo.

La pubblicazione di una applicazione nativa avviene tramite "negozi" specializzati virtuali che permettono agli utenti, tramite un ambiente ormai diventato familiare, di cercare, comprare, scaricare e installare applicazioni.
Resta comunque un ambiente specifico al sistema operativo, basti pensare ad App Store di Apple, Play Store di Google e al Microsoft Store, ognuno con svariate applicazioni esclusive da un lato e altrettante in comune dall'altro.

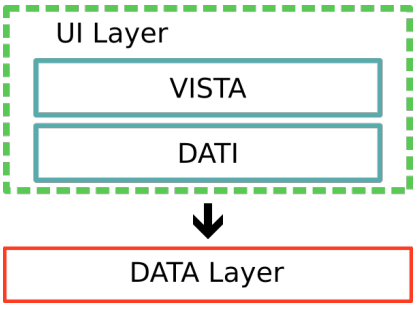

Figura 4: Applicazione Nativa.

Un'applicazione nativa, in particolare sviluppata per dispositivi mobili, si compone di due strati fondamentali: [fig. 4]

- uno strato di Presentazione o "**UI layer**"[14];
- uno strato "**Data Layer**"[14] che definisce la Business Logic dell'applicazione per la gestione dei dati.

3.1.1 *UI Layer*

Descrive come i dati verranno visualizzati sul display del dispositivo e sotto quale forma di interazione, ovvero definisce quegli elementi visuali di design che ritroviamo e che caratterizzano l'esperienza utente nell'applicazione.

Questi elementi, o più correttamente **componenti**, appartengono solitamente ad enormi librerie ben strutturate di oggetti riutilizzabili infinite volte, comportando la necessità di definire linee guida nei design e nell'utilizzo, con il risultato di ottenere sì applicazioni tutte molto simili tra loro, ma indubbiamente **navigabili**, ovvero facili da usare, e **abitudinarie**, essendo ormai abituati a questi canoni di stile.
Tra le guide e le librerie più dettagliate e più utilizzate in tale senso troviamo **Material Design**[C], sviluppata e mantenuta da Google.

Come esempio riporto un collegamento ad una pagina nella documentazione di Material Design che descrive ampiamente l'utilizzo dei cosiddetti FAB (Floating Action Buttons) dettando il corretto modo di utilizzare questi bottoni per azioni primarie rispetto ad un normalissimo bottone e descrivendo le varie configurazioni di interazione con l'utente:

```
https://material.io/components/buttons-floating-action-button
```

Lo User-Interface Layer e quindi i componenti appena definiti si compongono di due livelli:

- un livello di **resa grafica** tramite il concetto di "vista". Le "viste" sono reattive, modulari, dinamiche e componibili tra loro;

- la logica nell'**esposizione dei dati**, ovvero i dati essenziali che il componente si mantiene nel suo stato e come questi interagiscono con la parte grafica.

3.1.2 *Data Layer*

Definisce il modo in cui l'app crea, archivia e modifica i dati[14]. In generale si compone di **Classi di Dati**, suddivisi in più livelli, mantenuti in un database. Tali dati, come ad esempio i dati di accesso, devono essere mantenuti aggiornati in sicurezza.

Altro concetto fondamentale è definire il **Ciclo di Vita** di questi, in quanto una classe di dati può essere inizializzata, ridefinita e mutata durante il tempo di esecuzione e soprattutto può a sua volta inizializzare altre classi andando ad occupare memoria indefinitamente.
A tale scopo, per convenienza, si è soliti utilizzare linguaggi di programmazione ad oggetti, per lavorare con le classi di dati utilizzando principi di design pattern e ottenerne una migliore manutenzione del codice stesso.

3.2 STRUTTURA DI UN SITO WEB

La struttura di un generico sito Web si può schematizzare visivamente come un grafico ad albero che, partendo da una pagina iniziale, si dirama in pagine sempre più interne tramite dei collegamenti tra queste.
In programmazione Web si definisce **Routing** il meccanismo di associazione di una pagina al contenuto che verrà visualizzato.
Un esempio di ciò potrebbe essere passare dalla pagina */home* alla pagina */home/about* in un normalissimo sito Web: nella prima viene esposto il contenuto primario del sito e tramite un link si passa alla seconda pagina, più interna, contenente informazioni aggiuntive.
Elemento fondamentale del concetto di pagina Web è il **DOM**, un'interfaccia standardizzata da W3C che va a definire la strutturazione logica di come accedere al Documento e come modificarlo, compatibile con qualsiasi linguaggio di programmazione.

Come evoluzione dei siti Web statici si è arrivati al concetto di applicazioni Web, cioè si è passati da siti Web il cui contenuto è aggiornato esclusivamente da chi ne ha i privilegi a siti Web generati dinamicamente dalle interazioni con chi ne modifica i contenuti.
L'entry-point di una Web app, ovvero la prima pagina di accesso, per convenzione viene salvata come ***index.html***, ed è solitamente incaricata di definire i metadati dell'applicazione: dati e informazioni che non verranno visualizzati dall'utente ma elaborati dal browser.

Una Web app si sviluppa su due unità principali che comunicano tra di loro. [fig. 5]

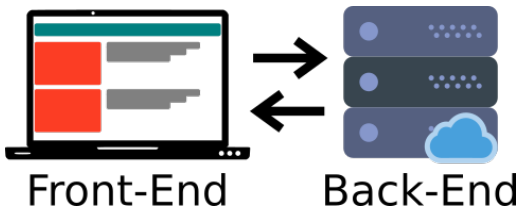

Figura 5: Sito Web.

3.2.1 *Il livello Front-End*

Lo sviluppo Front-End[15] dell'applicazione Web equivale a definire quella parte del sito che interagisce con l'utenza direttamente, il **livello Client-Side**, analogamente al livello UI nelle applicazioni native. Definisce quindi layout, design e comportamento di elementi visivi con i quali interagire [bottoni, menu di navigazione...].
Il livello front-end solitamente è scritto e sviluppato con i linguaggi fondamentali del Web: HTML, CSS e JavaScript.

3.2.2 *Il livello Back-End*

Lo sviluppo Back-End[15] invece equivale al **livello Server-Side** della Web app, ovvero a come questa gestisce e salva i dati e mantiene le librerie di funzioni dell'applicazione stessa, invisibili all'utente finale, analogo al Data Layer delle applicazioni native.
Molti linguaggi ad alto livello sono utilizzati per la programmazione back-end: PHP, Java o C per dirne alcuni.

Ultimamente sta avendo molto successo nello Sviluppo back-end una tecnologia chiamata **NodeJs**[16] che permette di utilizzare la potenza del linguaggio basato su eventi JavaScript anche su lato server, risultando decisamente adatta per qualsiasi applicazione che deve rispondere a numerose richieste in modo rapido ed efficiente.

Per concludere, ci sono sostanzialmente due modi di restituire graficamente un sito Web:

- **Server-Side Render** (**SSR**[17]) implica che sia il server a renderizzare l'intero sito Web, risultando istantaneo sulla prima pagina, ma perdendo di efficienza per caricare pagine più interne (in quanto si eseguono ulteriori chiamate al server);

- **Client-Side Render** (**CSR**[17]) implica che il sito Web venga "scaricato" sul client (operazione di **Caching**), risultando rallentato al primo avvio, ma istantaneo nel caricare le pagine successive più interne.

3.3 STRUTTURA PWA

Ad un sito Web, quindi ad una struttura composta di due livelli front-end e back-end, possiamo aggiungere funzionalità e tecnologie, oggi supportate dalla maggior parte dei browser, che permettono sostanzialmente di convertire il sito in una applicazione.
Questa pratica di "convertire" un sito Web resta comunque estranea ai benefici e all'ideale dietro le PWA[18][19][20], ecco perché la progettazione di una PWA risulta più simile a quella di una applicazione nativa, sviluppata però senza l'ausilio di SDK, ma tramite gli strumenti per la programmazione Web.

3.3.1 *App Shell – il concetto di "offline first"*

Il funzionamento della shell[21] dell'applicazione si può riassumere come: *caricare subito il minimo essenziale dell'interfaccia grafica e salvarlo sul client.*

Corrisponde allo scheletro della UI, composto quindi da stile grafico, componenti e funzionalità, non contiene dati finché non verranno caricati successivamente. Questo significa avere una applicazione Web resa per metà in CSR, in quanto la shell è salvata nella cache (**Caching**) e viene montata dal service worker, e per metà in SSR, in quanto sarà poi delegato al server la resa del contenuto.

A livello utente significherà avere una applicazione Web estremamente fluida e rapida sia in accensione che durante l'utilizzo, anche offline, tale da poterla considerare una applicazione nativa.

3.3.2 *Service Worker – il gestore dell'applicazione*

I service worker[22], o operatori di servizio, sono una tecnologia che permette un proxy virtuale tra il browser e la rete: *un ponte tra le capacità native e le capacità web.*

La maggior parte dei browser (Chrome, Edge, Firefox, Opera, Safari...) ormai supporta i service worker e le loro potenzialità che possiamo riassumere con:

- opzioni di **controllo delle risorse**: quali risorse salvare nella cache del browser, quali invece richiedere al server o anche opzioni di sostituzione e modifica delle stesse a runtime;

- **associazione alla Web App**, la quale può delegare all'operatore di servizio la sincronizzazione in background dei dati, il supporto offline e persino uno scambio di notifiche e informazioni.

Limitazioni principali sono che un service worker non viene eseguito fintanto che il browser non è stato aperto (risolvibile con il permesso al browser di eseguire processi anche in background), e che un service worker non ha accesso al DOM della Web app.

Questi operatori hanno un cosiddetto **Ciclo di Vita** scandito dal browser stesso.

Inizialmente il browser deve interpretare le informazioni della pagina principale dell'applicazione, per convenzione denominata ***index.html***, la quale, essendo un file HTML, riporterà informazioni sul service worker nella sezione <*head*>, come **metadato**.
Se il browser riesce a scaricare correttamente l'operatore, eseguirà gli eventi di **Installazione** (prima), dove crea la cache locale e carica il necessario per funzionare, e **Attivazione** (poi), punto in cui l'applicazione si avvia.

Il service worker resterà attivo per tutta la durata della sessione sul browser.

Importante concetto è la **sicurezza** dell'applicazione.

Secondo quanto detto, il service worker è semplicemente un blocco di codice scritto in JavaScript, mantenuto all'interno delle cartelle pubbliche del Progetto, che viene installato e avviato dal browser. Ha inoltre pieno controllo della applicazione ed è perennemente attivo in background. Per questo i browser richiedono una **connessione TLS**, una connessione sicura come il protocollo HTTPS per evitare che l'applicazione venga compromessa.

Inoltre un service worker può lavorare solo con i file e con le risorse appartenenti alla stessa cartella o nidificate a questa. La struttura superiore dell'applicazione è completamente ignorata, quindi è importante anche decidere dove posizionare fisicamente il file che descrive l'operatore per assicurare un ulteriore livello di sicurezza sui dati sensibili.

3.3.3 *Manifesto – le direttive della Web App*

Una PWA deve comunicare al browser alcune informazioni basilari per un corretto funzionamento.

Si utilizza a tale scopo un file JSON, un semplice linguaggio sviluppato proprio per la condivisione di informazioni client-server, nel quale definiamo alcune direttive[23] che spiegano quindi al browser come installare l'applicazione, includendo informazioni utili come:

- l'icona, o un **set di icone** variabili in base al dispositivo;
- il **nome dell'applicazione**, che verrà visualizzato all'avvio e una descrizione dell'applicazione e delle sue funzionalità;
- l'**indirizzo iniziale**, sotto forma di URL, dal quale l'applicazione si avvierà in automatico. In questo campo possiamo selezionare indirizzi anche interni;
- e varie informazioni sul dispositivo, come ad esempio l'orientamento standard col quale far partire l'applicazione.

4

IL PROGETTO

Il progetto al quale ho lavorato deriva dalla volontà di completare la tesi con un esempio a "grandezza naturale" che riuscisse ad accennare le potenzialità di tutti gli strumenti che negli anni ho scoperto e di cui ho seguito lo sviluppo.

L'idea dietro al progetto è stata ispirata dal testo di Carlos Rojas - Building Progressive Web Applications with Vue.js[24] - che mi ha accompagnato durante tutto lo sviluppo e che mi ha aiutato nelle scelte di progettazione. Nel libro viene modellata una basica applicazione per dispositivi mobili di generazione di note, dove le singole note sono componenti a sé stanti che, una volta generate tramite un bottone, vengono concatenate al DOM della pagina.

Avendo già accennato al progetto IOWA[12] di Google, posso confermare, dopo questi mesi di sviluppo in proprio, che il modo di progettare una Progressive Web App è rimasto concettualmente pressoché invariato, dettato da quei principi che secondo Alex Russell e Frances Berriman[11], coloro che coniarono il termine, definiscono questa classe di applicazioni.

Alcuni dei seguenti punti fanno già parte di buone prassi di progettazione di Web app standard, ma aggiungono alle PWA, ed in particolare al progetto, complessità nella struttura interna rendendole molto più simili nello sviluppo alle applicazioni native.

Secondo i coniugi Russell le PWA devono essere[11]:

- **Reattive** : per adattarsi a qualsiasi forma e quindi a dispositivi differenti;

- **Indipendenti dalla Connettività** : basate e migliorate dai service worker per consentirgli di lavorare anche offline;

- **Interattive** : adottando un modello shell + contenuto per creare applicazioni navigabili ed interattive (tendente all'esperienza app nativa);

- **Trasparenti** : sempre aggiornate grazie al processo di aggiornamento del service worker;

- **Sicure** : servite tramite TLS (un requisito dei service worker) per prevenire la raccolta di dati da parte di esterni;

- **Rilevabili** : sono identificabili come "applicazioni" grazie ai manifesti W3C e più facilmente visibili a motori di ricerca;

- **Notificabili** : interazione con il sistema operativo e pieno accesso alle sue capacità;

- **Installabili** : tramite i prompt forniti dal browser, gli utenti sono in grado di "conservare" le app che ritengono più utili senza il fastidio di un app store;

4.1 STRUMENTI UTILIZZATI

Prima di spiegare il progetto e il suo sviluppo, descrivo gli strumenti che ho usato, molti per la prima volta, e le mie scelte in merito.

Sono partito dai linguaggi canonici per la programmazione Web quali JavaScript, HTML e CSS che ho imparato ad usare negli anni e specialmente durante la frequentazione al corso Progettazione e Produzione Multimediale di Ingegneria Informatica a Firenze, tenuto dal professor Del Bimbo.

L'HyperText Markup Language, o **HTML**, è un linguaggio di marcatura e di impaginazione che permette di definire la struttura della pagina Web. Lo stile degli elementi e il modo in cui vengono resi a schermo è definito dal **CSS**, mentre le funzionalità e il comportamento dovuto alle interazioni con questi è definito in **JavaScript**, un linguaggio di programmazione orientato agli eventi che lavora in modo sincrono su un singolo thread, ovvero eseguendo sequenzialmente funzioni una dopo l'altra.

Ogni browser fornisce un ambiente di utilizzo runtime[25] per JavaScript composto da un **Motore** che analizza ed interpreta il codice che gli passiamo (su Chrome ad esempio troviamo il motore V8 di Google), un **Ciclo di Eventi** per gestire l'ordine in cui le funzioni si susseguiranno ed **API** ed altri strumenti che amplificano ed estendono le potenzialità del linguaggio, rappresentando le moderne strutture di sviluppo di applicazioni Web.

NodeJS[16][26][D] è un runtime JS, quindi un ambiente che include tutto il necessario per eseguire operazioni JavaScript, basato su V8 di Google, che permette lo sviluppo e l'utilizzo di applicazioni anche al di fuori del browser.
Lavora tramite un unico processo perennemente attivo e non bloccante, ovvero nessuna funzione eseguita bloccherà mai il processo, permettendo inoltre la generazione di processi figlio, bilanciando il carico di lavoro tra i vari processori del dispositivo.
Infine ha introdotto JavaScript nello sviluppo back-end, come già accennato, rendendo NodeJS un importante passo avanti nello sviluppo Web.

Allo sviluppo in JavaScript comunque potrebbe essere dedicato un capitolo a sé stante, in quanto negli anni si sono sviluppati decine di framework che apportano modifiche e migliorie al linguaggio e, di conseguenza, una guerra interna alla comunità JavaScript per quale sia il migliore[27].

Inoltre lo sviluppo da parte di Microsoft di **TypeScript**[28][E], che implementa la normale sintassi JavaScript trasformandolo in un linguaggio di programmazione altamente tipizzato, ha portato significativi cambiamenti come la possibilità di eseguire test sul codice e tutti i benefici di una programmazione Object Oriented come classi, interfacce, ereditarietà... diventando talmente popolare ed essenziale che si sta proponendo ultimamente di abbandonarlo, per implementarlo in JavaScript stesso.

Tra i framework più utilizzati e conosciuti a livello globale, utilizzati per applicazioni Web e che supportano lo sviluppo di PWA, troviamo Angular, React e Vue.js:

- sviluppato e mantenuto da Google, **Angular** è una piattaforma completa, costruita in TypeScript, per lo sviluppo di applicazioni Web di grandi dimensioni. Per quanto ottimo come framework, viene screditato per la sua curva di apprendimento molto lenta e la mole di concetti propedeutici a tale scopo.

- sviluppata e mantenuta da Meta (il nuovo nome assunto da Facebook), **React** è una libreria (non un proprio framework) per creare interfacce utente che ha definito negli anni dei veri e propri standard per la programmazione JavaScript.

- **Vue.js** è stata una risposta agli sviluppatori in cerca di un compromesso tra la potenza di React e una versione più stabile e facile da usare di Angular.

Anche se Angular fu il framework con il quale Google sviluppò IOWA[12], ho deciso di utilizzare Vue.js sia per la curva di apprendimento (a detta di molti) più facile in partenza, che per un mio personale interesse e fiducia in un framework open source che ho scoperto molto tempo prima di iniziare la stesura di questa tesi.

La qualità degli strumenti che ho usato e di conseguenza l'interesse in merito ha avuto e sta avendo una ascesa [fig. 6] di non poco conto:

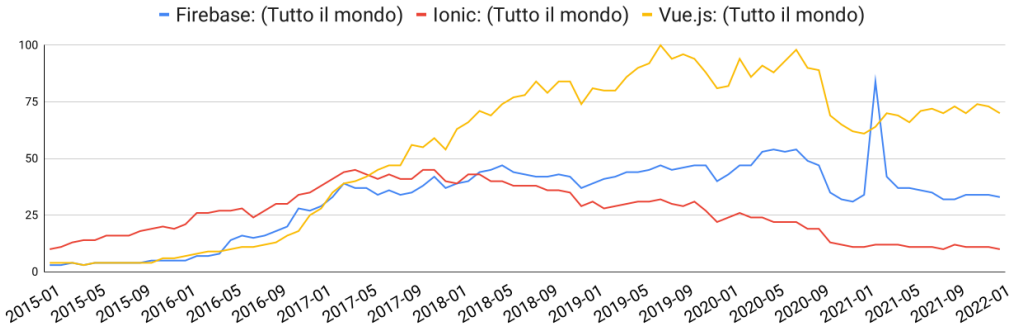

Figura 6: Interesse per i vari Framework dal 2015 al 2022.

da confrontare con l'andamento dell'interesse [fig. 7] riguardo alle PWA:

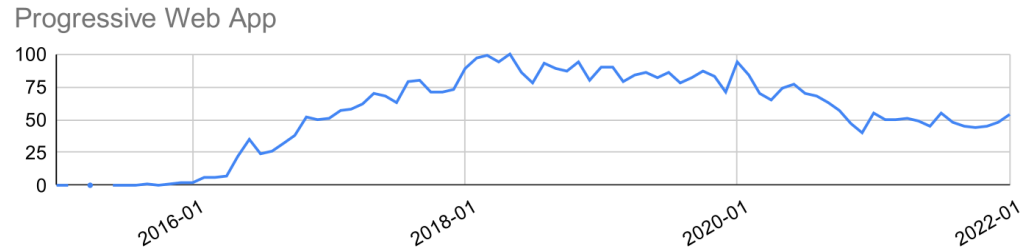

Figura 7: Interesse per le PWA dal 2015 al 2022.

Questi dati, ottenuti tramite Google Trends, dicono quanto l'interesse per questi strumenti sia cresciuto rapidamente dalla nascita del concetto di PWA nel 2015 fino al 2020, divenendo strumenti globalmente utilizzati attualmente e quindi standard per lo sviluppo di Progressive Web App.

4.1.1 *Vue.js*

Vue.js[F] è un moderno framework open source per la realizzazione di UI (Interfacce Utente) con una libreria molto alleggerita e con interessanti caratteristiche per costruire Progressive Web App (PWA) ad alte performance.

Con Vue.js si può gestire una gerarchia complessa di componenti, definiti per nome e comportamento, che hanno la capacità di essere riutilizzabili e mantenuti aggiornati, ed in particolare possono essere usati come tag HTML velocizzando la scrittura e portando il codice ad un livello di programmazione simile al paradigma Object Oriented.
La **comunicazione** tra componenti può avvenire tramite eventi o tramite variabili salvate in modo globale, così da rendere l'intero sistema reattivo a modifiche ed aggiornamenti.

I componenti infine hanno un **Ciclo di Vita** abbastanza complesso, il quale ha la particolarità di essere intervallato da agganci a funzioni esterne che ne permettono lo scambio di informazioni e la modifica a runtime prima e dopo ogni evento.

Quando viene chiamato dal DOM, un componente viene reso a schermo eseguendo la funzione ***setup()*** definita all'interno del componente stesso, che definisce le variabili iniziali e ne importa le funzionalità.
A questo punto viene eseguita la **creazione** così da poter **montare** il componente nel DOM.
Il componente potrà quindi subire interazioni e modifiche dall'utente dell'applicazione, aggiornando la resa grafica dinamicamente. Infine il componente può essere **smontato** e sostituito.

4.1.2 *Vue Router e Pinia*

Il Routing viene gestito in Vue.js tramite **Vue Router**[G], un componente al progetto, che permette una mappatura virtuale delle possibili vie (**Route**) tra le quali ci possiamo indirizzare o essere reindirizzati all'interno del sito. Ad ogni route si associa un componente che verrà visualizzato a schermo solo quando accediamo all'URL corrispondente.

Pinia[H] invece è diventato ultimamente lo **Store** ufficiale di Vue.js, uno spazio di memoria quindi che permette di salvare e organizzare variabili ed oggetti in modo globale all'applicazione, rendendoli raggiungibili durante tutto il periodo di attività.
In particolare Pinia permette di costruire e mantenere aggiornati multipli store, diventando modulare alle necessità dei singoli componenti.

4.1.3 *Ionic*

Come già descritto in precedenza, Ionic[I], aggiornato alla versione 6, è una potente piattaforma open source per la creazione di applicazioni Web. Distribuisce una vasta libreria di componenti Web di alta qualità, scritti seguendo le linee guida dettate dal Material Design di Google.

Come già accennato permette di accedere a svariate funzionalità native del dispositivo e permette una programmazione cross-platform tra sistemi operativi Mobile e browser Web.
Distribuisce inoltre una potente interfaccia da linea di comando (CLI, command-line interface) scritta in Typescript e NodeJs la quale permette la creazione di un progetto, la sua compilazione e altre funzionalità avanzate di manutenzione dello stesso e dei pacchetti in esso utilizzati.

Infine è compatibile con i framework JavaScript più popolari come Angular, React e Vue.js, le quali librerie possono essere facilmente integrate al progetto.

4.1.4 *Firebase*

Firebase di Google è un servizio cloud completo di varie funzionalità lato server che in maniera semplice ed immediata da usare permette uno sviluppo back-end molto solido per applicazioni di piccole e medie dimensioni. Tra gli strumenti messi a disposizione ho usato:

- **Firebase Real-Time DataBase**[L]:
  un servizio che aggiunge un database remoto sicuro per gestire i dati dell'applicazione che si sincronizza su tutti i client connessi in tempo reale, aggiornando quindi il contenuto ad ogni modifica.
  Permette inoltre un utilizzo offline, particolarmente adatto per le PWA, salvando momentaneamente i dati sul disco e caricandoli sul database una volta ristabilita la connessione.

- **Firebase Authentication**[M]:
  un servizio di autenticazione che interagisce direttamente con il database tramite delle regole di accesso per assicurare un livello di sicurezza aggiuntivo all'applicazione.

Risulta inoltre completo di una interfaccia grafica che supporta diversi tipi di autenticazione: dalla identificazione tramite username e password all'accesso tramite servizi di identificazione esterni come quelli offerti da Google o da Facebook.

- **Firebase Hosting**[N]:
  un servizio di Hosting che mette a disposizione un'infrastruttura, un dominio e le funzionalità per mantenere e distribuire una Web app o un sito Web.

4.2 LO SVILUPPO

Basandomi quindi sul progetto spiegato da Rojas, ho deciso di sviluppare una Single-Page Application per la creazione di documenti di testo semplificati, dove il Documento si compone di titolo e capitoli, e ogni Capitolo si compone, analogamente, di titolo e contenuto.
La semplicità dell'istanza Documento mi ha permesso di omettere lo sviluppo di un editor di testo completo, fuori dallo scopo del progetto, mantenendo un'interfaccia pulita, leggera e, soprattutto, limitata nelle componenti.

4.2.1 *L'Ambiente di Sviluppo*

Avviare un progetto di queste dimensioni comporta che l'ambiente di sviluppo sia il più completo e aggiornato possibile [fig. 8].

Inizialmente ho installato Node Version Manager, o **NVM**: un gestore di versioni dell'ambiente runtime NodeJS che include sia l'ultima versione di **NodeJS** sia il gestore di pacchetti e dipendenze predefinito chiamato Node Package Manager, o **NPM**, che mi ha reso possibile l'installazione di **Ionic**.
Tramite Ionic CLI, quindi tramite dei comandi da Shell, ho attivato l'interfaccia grafica per la creazione automatica di una cartella di progetto, con rispettivi nome, icona, framework e struttura di partenza.
In questi ultimi due campi ho potuto scegliere le fondamenta del progetto, inserendo come framework **Vue.js** e lasciando la struttura interna senza implementazione.

```
matteo@pop-os:~$ nvm install --lts
matteo@pop-os:~$ npm install -g @ionic/cli
matteo@pop-os:~$ ionic start

matteo@pop-os:~/pwa$ npm install vue-router
matteo@pop-os:~/pwa$ npm install pinia
matteo@pop-os:~/pwa$ npm install firebase

matteo@pop-os:~/pwa$ ionic build
matteo@pop-os:~/pwa$ ionic serve
```

Figura 8: Comandi da Shell per generare un progetto Ionic, completo di strumenti. Gli ultimi comandi compilano il progetto e lo eseguono su un server locale.

A questo punto Ionic ha creato una cartella, completa di dipendenze JavaScript e struttura interna, praticamente "pronta per l'uso".
Per completare la costruzione dell'ambiente ho ulteriormente installato nel progetto, tramite comandi NPM, **Vue Router**, **Pinia** e i vari strumenti offerti da **Firebase** che avevo intenzione di usare.

Tra di loro, questi strumenti interagiscono secondo uno schema quasi gerarchico.

Il progetto è scritto in Vue.js, un framework con cui è possibile sia montare componenti Ionic, che vengono gestiti e mantenuti da NodeJS, sia importare funzionalità e dipendenze dagli altri strumenti, facendoli interagire tra di loro.

4.2.2 *Le Componenti dell'Applicazione*

Un file con estensione .vue è un componente Web composto da tre elementi:

- **<script>** è un blocco JavaScript (o nel mio caso TypeScript) dove, tramite delle funzioni specifiche, definiamo il componente importando varie dipendenze e definendo varie funzioni. In particolare in questo blocco possiamo importare componenti di qualsiasi tipo, sia Ionic che Vue.js, che potranno essere usati nel blocco <template>;

- **<template>** equivale ad un blocco HTML nel quale possiamo utilizzare tutti i componenti importati, unendoli nella struttura dell'elemento finale;

- **<style>** infine equivale ad un blocco CSS dove possiamo definire stile e layout degli elementi in <template>.

Di seguito espongo le componenti visive che ho sviluppato in Vue.js.

TextEditorComponent.vue

Questo componente corrisponde all'oggetto **Capitolo** composto da titolo e contenuto, due componenti forniti dalla libreria Ionic, indicati rispettivamente dai tag <*ion-input*> e <*ion-textarea*>, ai quali posso aggiungere proprietà e funzionalità.
Una funzionalità interessante che ho sfruttato viene chiamata ***v-model*** e permette di associare un contenuto dinamico del componente, come può essere una frase in input scritta da tastiera, ad una variabile. In questo modo è possibile salvare il contenuto del capitolo ad ogni modifica, che verrà poi caricato sia nella memoria cache del browser che nel cloud.
L'icona di rimozione a destra del titolo permette di rimuovere l'intero capitolo.
Anche questa operazione è direttamente collegata al database realtime offerto da Firebase, permettendo queste operazioni anche ad un utilizzo offline.

CurrentDocument.vue

Questo componente struttura uno spazio dedicato al **Documento**: una volta creato ci troveremo di fronte ad una pagina vuota che rappresenta il documento corrente inizialmente privo di contenuto.
Nello spazio superiore a sinistra troviamo il titolo del documento mentre nello spazio inferiore un bottone che ci permette di aggiungere via via i componenti capitolo.
Su schermi di grandi dimensioni apparirà inoltre un bottone a sinistra del titolo che apre o chiude il menu, di default sempre aperto.

MenuSplitDocument.vue

Il componente visivo principale dell'applicazione che ho sviluppato è un **Menu** per la navigazione tra i documenti, perennemente situato a sinistra se si è su schermi di grandi dimensioni o fatto comparire con uno scorrimento da sinistra a destra se si usano dispositivi di piccole dimensioni.
Anche il menu è un componente Ionic, indicato dal tag <*ion-menu*>, animato e molto più complesso dei precedenti.
Internamente si compone di una lista di titoli che si aggiorna automaticamente ad ogni nuovo documento creato o ad ogni modifica apportata a quelli già esistenti.
Ad ogni titolo nella lista è associato un collegamento al cloud che, se cliccato, ripristinerà il documento con il contenuto originario, sul quale sarà possibile tornare a lavorare. A lato dei singoli titoli si trova un bottone di rimozione che elimina l'intero documento.
Nello spazio inferiore al menu si trova il tasto per la generazione di un nuovo documento. Basterà inserire il titolo, premere sul pulsante e accettare tramite il pop-up che apparirà, per generare, a destra del menu, un nuovo documento vuoto.
Anche qui tutte le funzioni di creazione, modifica o rimozione sono direttamente collegate al database realtime offerto da Firebase, permettendo queste operazioni anche ad un utilizzo offline. Quando la connessione risulterà ripristinata, il database si aggiornerà autonomamente.

ConstructPopUp.vue

Rappresenta lo step intermedio per la creazione di un nuovo documento e ci chiede, tramite un **pop-up** di notifica, se siamo sicuri del titolo che abbiamo digitato.
In caso positivo avvierà la funzione ***createDocument(title)*** la quale crea inizialmente uno spazio nel database, associato ad un identificativo univoco, e successivamente un oggetto documento passandogli l'identificativo. Infine monta il relativo componente CurrentDocument.vue a destra del menu.

Tutti questi componenti non sono file HTML a sé stanti, ma anzi hanno bisogno di essere montati sull'applicazione o su un componente già montato.

Montare un componente sull'applicazione [fig. 9], inizializzata da Vue.js tramite il metodo ***createApp(App)*** nel file *main.ts*, significa interrogare il router dell'applicazione tramite i metodi concatenati ***router.isReady().then()*** ed eseguire la funzione ***mount()*** per agganciare il componente ad un elemento del DOM.
Spostandosi verso quell'indirizzo, o route, nell'applicazione si avvierà un render del componente montato.
È in questo modo che si struttura visivamente l'applicazione.

```
import App from './App.vue';
import router from './router';

const app = createApp(App)
  .use(IonicVue)
  .use(createPinia())
  .use(router);

router.isReady().then(() => {
  app.mount('#app');
});
```

Figura 9: Codice dal file main.ts per montare un componente all'applicazione.

La **comunicazione** tra i vari componenti, montati e attivati nella Web app durante l'utilizzo, è assicurata dallo store di variabili globali Pinia.

Ho creato un file chiamato *PwaBasicStore.ts* nel quale definisco lo store principale, che ho chiamato "LocalStore", costituito da varie informazioni come il documento corrente sul quale stiamo lavorando o la matrice attuale con i contenuti dei capitoli che verranno caricati al salvataggio sul cloud.

Mentre il **Routing** tra questi viene gestito da Vue Router.
Inizialmente l'utente si troverà nella route basica **"/"** alla quale è associata la pagina di login offerta da Firebase. Una volta effettuato l'accesso tramite le credenziali viene sostituita la route di login con la route per lo spazio di lavoro **"/workspace"**.

Da qui l'applicazione si comporterà come una Single-Page Application finché non verrà eseguita la disconnessione tramite il bottone di logout nel menu, che indirizzerà nuovamente alla route di login **"/"**.

Infine tutte le **funzionalità** dei vari componenti e le interazioni che hanno fra di loro e con l'utente dell'applicazione sono state definite in una libreria esterna chiamata *TextEditorLibrary.ts*.

La libreria [fig. 10] è un singolo file interamente scritto in TypeScript che definisce inizialmente le classi astratte e le interfacce degli oggetti dell'applicazione e successivamente le classi concrete, che implementano ed estendono le prime, alle quali delegare la definizione delle funzioni.
Ho cercato di dividere questa parte dalle definizioni dei componenti in quanto risulta la sezione del progetto meno esposta a modifiche, lasciando la possibilità di stravolgere i singoli componenti senza intaccare il comportamento dell'applicazione.

Il diagramma UML [fig. 10] rappresenta la struttura della libreria suddivisa in aree di classi, dove sono state segnate le funzioni principali.
Le classi astratte (in azzurro) ***TextEditorSubject*** e ***TextEditorComponent*** contengono le dichiarazioni astratte delle funzioni rispettivamente *addToRouteList()*, per aggiungere un documento alla lista dei documenti creati, e *addToDom()* per montare un capitolo all'interno del documento.
Le loro rispettive implementazioni (in viola), utilizzate in questo progetto, sono le classi ***Document*** e ***Chapter***. Document in particolare rimane astratta, definendo il metodo *makeChapter()* che alloca uno spazio nel database per un capitolo e restituisce l'istanza di un nuovo oggetto concreto Chapter.
Document si concretizza in due classi (in giallo) che estendono l'interfaccia funzionale (in verde) ***MakeDocStrategy*** per la creazione di un file tramite il metodo *makeDocument()*, così da istanziare il file, in base alla strategia utilizzata, o come un nuovo documento locale (creato tramite input dal menu) o come un documento scaricato dal cloud (scaricato tramite link dal menu).

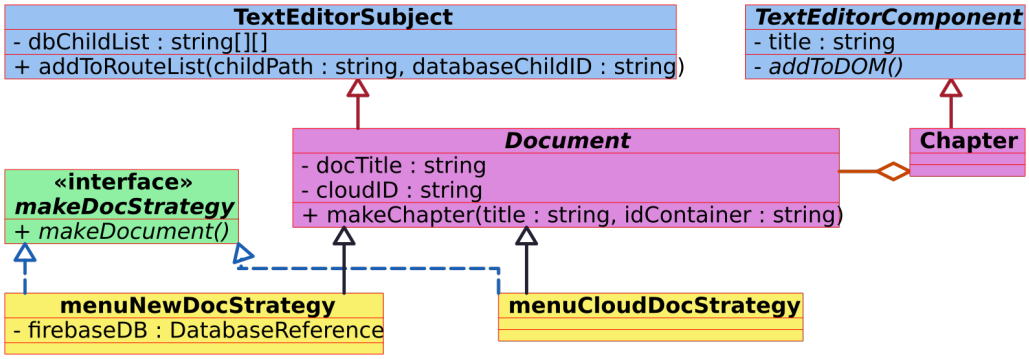

Figura 10: Diagramma delle Classi in formato UML relativo alla libreria TextEditorLibrary.ts.

4.3 L'APPLICAZIONE

Figura 11: Icona dell'applicazione una volta installata sul dispositivo.

Per completare il progetto, descritto nella sezione precedente, e renderlo definitivamente una PWA occorre definire il **Manifesto** nelle cartelle pubbliche, che comunicherà al browser informazioni generiche su come installare e far partire l'applicazione una volta installata sul dispositivo, e il **Service Worker**, che installerà e renderà funzionale la gestione della memoria cache.

Questi due file sono, almeno per progetti piccoli come questo, convenzionali, per cui si trovano modelli già strutturati o strumenti che li generano in automatico nel progetto. Sta al programmatore compilare i parametri per migliorare le prestazioni e la sicurezza.

Il progetto è quindi terminato e può essere provato al seguente URL:

`https://progettotesi-a0499.web.app`

Mentre la cartella del progetto è mantenuta in una Repository di GitHub al seguente link:

`https://github.com/MatteoCherubini/text-editor-pwa`

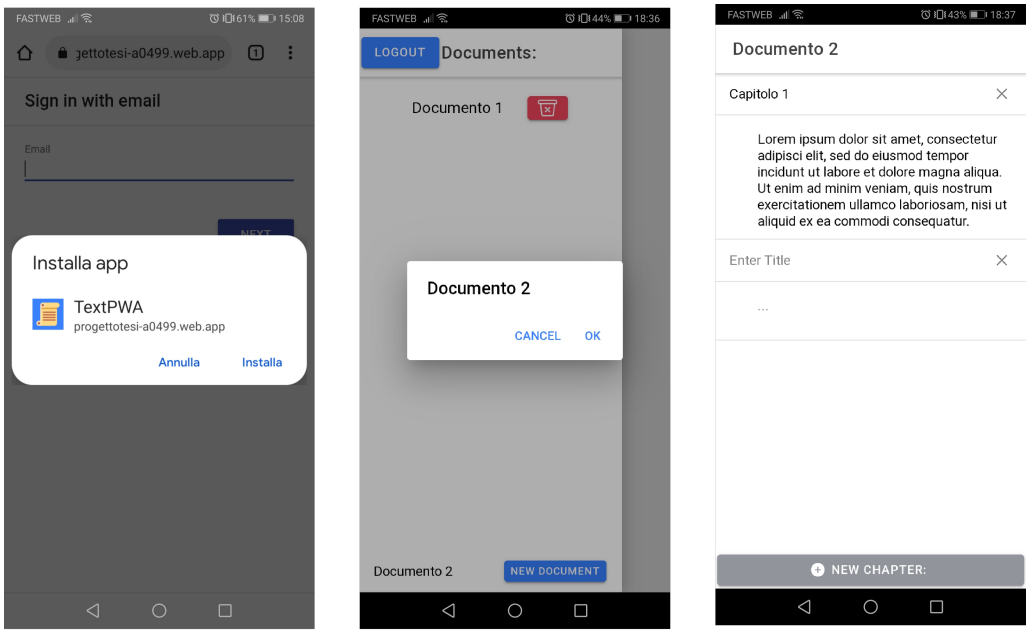

Figura 12: Dimostrazione di utilizzo dell'applicazione.

Una volta aperta la pagina iniziale del sito ufficiale dell'applicazione, il browser, rilevando il manifesto, ci chiederà se vogliamo o meno aggiungere alla home page del dispositivo (sia mobile, come in figura, che desktop) l'applicazione [fig. 12.1]. In entrambi i casi l'applicazione rimarrà utilizzabile, ma la versione sito Web potrebbe comunque perdere di praticità rispetto alla versione scaricata localmente.
Una volta scaricata e aperta tramite la rispettiva icona [fig. 11] nella home page del dispositivo, ci troveremo di fronte alla pagina di login. In questa pagina potremo o inserire una nuova mail, così da completare la registrazione di un nuovo utente, o inserire una mail e una password già esistenti, accedendo al proprio profilo, il quale si presenterà come una pagina vuota.
Per lavorare su un file dovremmo aprire il menu a tendina che compare alla sinistra della pagina, creando così un nuovo documento o aprendo dalla lista un documento già esistente [fig. 12.2]. Una volta nel file possiamo modificarlo aggiungendo o eliminando capitoli [fig. 12.3].

Tutte queste interazioni di creazione, modifica in tempo reale ed eliminazione di componenti nell'applicazione locale sono direttamente connesse ad operazioni nel database fornito da Firebase, il quale ne mantiene

uno storico in memoria cache, permettendo così queste operazioni anche con un dispositivo offline. Una volta ripristinata la connessione verrà aggiornato automaticamente il database.

5

CONSIDERAZIONI FINALI

Termino, con quest'ultimo capitolo, descrivendo quelle che sono le mie considerazioni riguardo agli argomenti esposti in questa tesi, partendo dal progetto sul quale ho lavorato, terminando con le Progressive Web App e il loro futuro.

Le problematiche principali riscontrate durante tutto lo sviluppo sono dovute al lato front-end dell'applicazione, per quanto abbia tentato di mantenerlo più semplice possibile.
Framework open source come Vue.js, Ionic e librerie come Material Design hanno una documentazione estremamente dettagliata, nella quale si trovano i casi di uso, dai più ai meno comuni, e definizioni di concetti, dai semplici ai complessi, con descrizioni approfondite del codice e delle funzionalità.
Elementi che aiutano ad avvicinarsi, ma ai quali non mi ero mai approcciato così tanto, e che sono stati un garbuglio di informazioni che hanno occupato la maggior parte del tempo impiegato nello sviluppo.

Un altro problema è stato capire come usare TypeScript in maniera corretta. Il mio approccio è stato quello di affrontare il linguaggio nella maniera più simile possibile a quanto imparato ai corsi di Programmazione e di Metodologie di Programmazione.
Ho cercato di utilizzare strumenti di buona programmazione come Unit-Testing e Design Pattern, purtroppo fallimentari e non compatibili (o almeno non chiaramente spiegati nella documentazione) con Vue.js e Ionic, framework probabilmente destinati ad un approccio più concreto.

Alcune scelte durante l'intero sviluppo sono state quindi obbligate dalla incompatibilità tra i vari strumenti e dalla mia difficoltà nel reperire informazioni dalle documentazioni ufficiali, ma mi hanno fatto comprendere in modo più pratico cosa sia lo sviluppo front-end.

Lo sviluppo lato back-end invece, essendomi affidato interamente a Firebase, è stato molto più semplice e senza problemi. Le informazioni nella documentazione erano aggiornate e gli esempi erano minimali e suddivisi per i più comuni ambiti di applicazione.

Gli sviluppi futuri del progetto e delle mie conoscenze dovranno essere indirizzati in primo luogo verso la **sicurezza**, che per ora è al minimo indispensabile. Per quanto abbia tentato di mantenere privati e distaccati i dati sensibili, come i collegamenti al database, non è risultato semplice testarne la reale protezione.
In secondo luogo ho necessità di comprendere meglio come utilizzare i **test** su ciò che sviluppo. I test sono strumenti fondamentali in quanto, oltre ad aiutare a controllare l'effettiva efficacia di ciò che è stato scritto, diventano una vera e propria documentazione con esempi di uso che aiutano a mantenere lo sviluppo ordinato e produttivo.
Per quanto riguarda il mio approccio ai test, questo ha prodotto qualche difficoltà in quanto, pur avendo testato immediatamente le classi durante lo sviluppo controllandone la correttezza, queste, quando successivamente hanno iniziato ad interagire con i componenti e quindi con Routing e database, hanno reso insufficienti, se non errati, i test basilari.
Quindi una volta assicurata la correttezza delle classi e delle loro funzionalità basilari, ho rimosso i test. Esistono strumenti aggiuntivi per testare singolarmente i componenti, ma ho evitato di inserirli nel progetto dedicandomi a studiare principalmente gli strumenti descritti nei capitoli precedenti.

Questa esperienza mi ha aiutato a comprendere le potenzialità degli strumenti che oggi vengono utilizzati per lo sviluppo Web e di come sia relativamente facile approcciarsi allo studio di questi e successivamente al loro utilizzo.
Personalmente sono un sostenitore della **comunità Open Source** e mi è piaciuto vedere strumenti così potenti utilizzati in questa economia, dove la comunità controlla e mantiene sicuro e aggiornato il codice.

Sono inoltre entusiasta del coinvolgimento e dell'egemonia di **Google** su questa tecnologia, che per ora non sembra indebolire troppo altri browser concorrenti. Personalmente credo che l'interesse dell'azienda sia nel servizio che propone: le PWA sviluppate da Google finora si ritrovano come estensioni Google Chrome, non come applicazioni a sé stanti, e quindi vengono usate principalmente su telefoni Android o ChromeBook, andando a migliorare la qualità dell'insieme di servizi che

Google propone ai propri clienti.
Restano comunque un visibile e concreto avvicinamento al concetto di "una base di codice per più dispositivi", che potrebbe essere il futuro della programmazione Web, arrivata secondo me (che mi sono approcciato per pochi mesi, con conoscenze generali) ad una saturazione. Infatti, da un lato si trovano decine di framework e librerie simili che si fanno concorrenza tra loro, generando disordine nel reperire informazioni;
dall'altro lato si trovano standard di programmazione, strumenti che in automatico creano modelli da compilare e componenti di alta qualità che sminuiscono il lavoro del programmatore.

In conclusione, le Progressive Web Application sono uno standard nato in un periodo storico in cui approcciarsi allo sviluppo di applicazioni Web non ha mai avuto così tanto potenziale come oggi, e questo potenziale è dato dai Browser che si comportano sempre più come sistemi operativi, offrendo servizi superiori, grazie al Cloud Computing, ad un'utenza ormai abituata a determinati layout e interazioni con i dispositivi che possiede.

BIBLIOGRAFIA

glossario, definizioni, articoli e testi:

[1] Web Design Museum - “Web Design History Timeline” - [articolo]
https://www.webdesignmuseum.org/web-design-history

[2] Ethan Marcotte - “Responsive Web Design” - [articolo]
https://alistapart.com/it/article/web-design-reattivo/

[3] Max Lynch - “Building the Progressive Web App OS” - [articolo]
https://medium.com/@maxlynch/
building-the-progressive-web-app-os-57daebcb69c1

[4] Haohua Qing, Jiali zhang, Hong Cao - “Design and Realization of Webpage Operation Computer System Based on Cloud Computing” - [pubblicazione online]
https:
//iopscience.iop.org/article/10.1088/1742-6596/2037/1/012014

[5] https://www.vmware.com/it/topics/glossary.html\protect\
unhbox\voidb@x\protect\penalty\@M\

[6] https://www.redhat.com/it/topics/cloud-computing/
iaas-vs-paas-vs-saas

[7] d‘wise one - “Web Operating Systems — The Way Of The Future?” - [articolo]
https://medium.com/chip-monks/
web-operating-systems-the-way-of-the-future-c6cc1f203d8a

[8] https://www.chromium.org/chromium-os/
chromiumos-design-docs/software-architecture/

[9] Chromium Blog - “Chrome Dev Summit 2020” - [articolo]
https://blog.chromium.org/2020/12/
chrome-dev-summit-2020-wrap-up.html

[10] ChromeOS Blog - “Powerful apps fueled by the web” - [articolo]
https://chromeos.dev/en/posts/powerful-apps-fueled-by-the-web

[11] Alex Russell - “Progressive Web Apps: Escaping Tabs Without Losing Our Soul” - [articolo]
https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/

[12] Eric Bidelman - “Building the Google I/O 2016 Progressive Web App” - [articolo]
https://developers.google.com/web/showcase/2016/iowa2016

[13]
https://www.redhat.com/it/topics/cloud-native-apps/what-is-SDK

[14] https://developer.android.com/jetpack/guide#recommended-app-arch

[15] https://www.geeksforgeeks.org/frontend-vs-backend/

[16] https://www.nodeacademy.it/differenze-node-js-linguaggi-backend-classici-php-java-c-python-ruby/

[17] Jason Miller, Addy Osmani - "Rendering on the Web” - [articolo]
https://developers.google.com/web/updates/2019/02/rendering-on-the-web

[18] https://web.dev/progressive-web-apps/

[19] https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/

[20] Nancy Davis Kho - “Everything you need to know about Progressive Web Apps” - [pubblicazione online]
https://www.thetilt.com/content/progressive-web-apps

[21] https://developers.google.com/web/fundamentals/architecture/app-shell

[22] John M. Wargo - Learning Progressive Web Apps - [eBook] - © 2020 Pearson Education, Inc.

[23] https://developer.mozilla.org/en-US/docs/Web/Manifest

[24] Carlos Rojas - Building Progressive Web Applications with Vue.js - [eBook] - © Carlos Rojas 2020 Standard Apress

[25] Gemma Stiles - "Understanding the JavaScript runtime environment" - [articolo]
https://medium.com/@gemma.stiles/understanding-the-javascript-runtime-environment-4dd8f52f6fca

[26] Priyesh Patel - "What exactly is Node.js?" - [articolo]
https://medium.com/free-code-camp/what-exactly-is-node-js-ae36e97449f5

[27] David Rodenas - "The JavaScript framework war is over" - [articolo]
https://drpicox.medium.com/the-javascript-framework-war-is-over-bd110ddab732

[28] https://www.tutorialspoint.com/typescript/typescript_overview.htm

siti ufficiali:

[A] Chromium OS - https://www.chromium.org/chromium-os

[B] CoreBoot - https://www.coreboot.org/

[C] Material Design - https://material.io/design

[D] NodeJS - https://nodejs.org/it/

[E] TypeScript - https://www.typescriptlang.org/

[F] Vue.js - https://vuejs.org/

[G] Vue Router - https://router.vuejs.org/

[H] Vue Pinia - https://pinia.vuejs.org/

[I] Ionic - https://ionicframework.com/docs/

[L] Firebase Real-Time DataBase - https://firebase.google.com/docs/database

[M] Firebase Authentication - https://firebase.google.com/docs/auth

[N] Firebase Hosting - https://firebase.google.com/docs/hosting