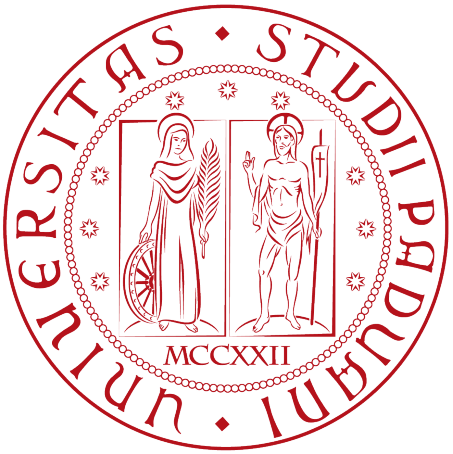
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Una piattaforma serverless basata su canvas per il supporto dei processi agile

Tesi di laurea

Relatore

Prof. Paolo Baldan

Laureando

Marko Vukovic

Anno Accademico 2021-2022

Sommario

Il progetto Agile Wall si propone di digitalizzare alcuni strumenti a supporto delle metodologie agile adottate da Zero12, con particolare attenzione ai meeting Scrum of Scrums svolti dai project manager. Lo strumento principale è un Wall, chiamato internamente "Muro del Pianto", su cui vengono incollati dei post-it per gestire milestone di progetti attivi nell'azienda.

Il progetto ha avuto l'obiettivo di sviluppare un'applicazione web utilizzando il paradigma serverless. I servizi sono stati rilasciati su infrastrutture AWS e sono accessibili solamente a utenti interni all'azienda mediante login con Google. L'interfaccia grafica è stata sviluppata con Angular e Fabric.js come astrazione a più alto livello delle API del canvas HTML. Questo per cercare di rispondere alle esigenze del proponente di mantenere il "feel" del Wall fisico, con la possibilità di effettuare zoom in / out, pan "infinito" e muovere le note trascinandole. È stato utilizzato anche il protocollo WebSocket per permettere di visualizzare in tempo reale gli altri utenti collegati all'applicazione.

Il tutto è stato svolto applicando gli stessi processi agile e best practices utilizzati internamente dall'azienda, tra cui: sviluppo in diversi ambienti (locale, develop, production), utilizzo di una pipeline CI/CD, testing e documentazione.

"Strong opinions are often based on ignorance"

— John Green

Ringraziamenti

Ringrazio profondamente tutti i miei cari.

Padova, Luglio 2022 Marko Vukovic

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Azienda

Zero12 S.r.l. è un'azienda italiana che si occupa di innovazione e diffusione della cultura digitale in ambito business. I servizi principali offerti da Zero12 sono:

- Supporto nel percorso di **cloud adoption** attraverso la migrazione e/o design di infrastrutture cloud;
- **Sviluppo di soluzioni software** cloud native web e mobile totalmente personalizzate;
- Progettazione e sviluppo di soluzioni di **machine learning**.

Il tutto guidati da metodologia Agile e una collaborazione che parte da sessioni di design thinking. Zero12 ha attualmente due sedi operative: una a Padova, dove è stato svolto lo stage curricolare, l'altra a Empoli.

Figura 1.1: Logo di Zero12

Zero12 è fin dalla sua nascita partner AWS. **Amazon Web Services Inc.** è una sussidiaria di Amazon che si occupa di fornire piattaforme di cloud computing on-demand con modalità pay-as-you-go. Zero12 si prende in carico il compito di formare il proprio personale anche mediante le certificazioni ufficiali AWS. Al 2021 AWS offre più di 200 prodotti e servizi, tra cui: computing, storage, networking, database, analytics, deployment, machine learning, developer tools, IoT. In particolare i più popolari sono: Elastic Compute Cloud (EC2), Relational Database Services (RDS), Simple Storage Service (S3), Lambda, Simple Notification Service (SNS), Identity and Access

Management (IAM), DynamoDB, Cognito, CodeCommit, CodePipeline, Amplify, Lex, etc...

Figura 1.2: Logo di Amazon Web Services

L'esperienza da stagista è stata notevolmente positiva: un altissimo livello di empatia da parte di tutto il team permette di abbattere velocemente la barriera di riserbo creata dalla disparità di competenze. Questo porta ad avere una comunicazione molto diretta e trasparente che, lato stagista, permette di chiarire velocemente dubbi e problemi che possono emergere durante lo svolgimento del progetto. Tale approccio predispone una rapida crescita di conoscenze e abilità. Il clima all'interno dell'azienda è professionale ma al contempo rilassato. In ufficio si alternano in modo bilanciato: momenti formali, come stand up meeting, discussioni tecniche e momenti di confronto professionale; momenti informali di ricreazione, conversazione e gioco. Oltre alle attività strettamente legate al business dell'azienda, vengono organizzati momenti di crescita e confronto come:

- Trasferte e partecipazione a conferenze (e.g. AWS Summit Milano 21-22 Giugno 2022);
- Talk aziendali di membri esperti o ospiti;
- Attività di team building.

Zero12 è parte di **Var Group**, società che si dedica all'innovazione del settore ICT nei più svariati ambiti: fashion, manufacturing, retail & GDO, food & wine, automotive, furniture, pharma, banking & insurance, etc. Var Group conta 23 sedi in Italia, 8 sedi all'estero e oltre 2700 collaboratori. Tra i principali ambiti d'interesse ci sono: digital cloud, digital security, data science, industry 4.0 e IoT. La società si evolve continuamente anche grazie alla ricerca continua e alla stretta collaborazione con start up e università.

A sua volta Var Group appartiene al gruppo **Sesa S.p.A.**, leader italiano di soluzioni IT per il segmento business, quotata negli indici FTSE Italia Mid Cap e FTSE Italia STAR della Borsa Italiana.

Figura 1.3: Logo di Var Group e logo di Sesa

1.2 Progetto

Il progetto proposto da Zero12 è stato chiamato **Agile Wall** e riguarda la digitalizzazione di uno strumento utilizzato dai project manager durante i loro meeting Scrum

of Scrums. In pratica, negli uffici di Padova dell'azienda, su un muro, è presente una tabella di circa 2 metri di altezza per 4 di larghezza. Le colonne della tabella rappresentano settimane, riportando nell'intestazione la data del lunedì della settimana in questione. Nelle righe invece sono riportati tutti i progetti attivi, seguiti dai relativi clienti e dai project manager di riferimento. Questo strumento è chiamato internamente anche *"Muro del Pianto"*. In questo documento ci riferiremo alla suddetta matrice come **Wall**. Su essa, i project manager incollano dei post-it per evidenziare Milestone importanti e commenti da condividere. Prima dei meeting Scrum of Scrums, ciascun PM può cambiare lo stato del Wall, aggiungendo, spostando, modificando o rimuovendo post-it.

Il principale problema è che non tutti i PM sono sempre in sede per poter fisicamente modificare il Wall. Da qui la necessità di avere uno strumento digitale accessibile anche da remoto. Tra i bisogni esplicitamente richiesti dal proponente ci sono:

- Mantenere il "feel" del Wall, con post-it che possono essere spostati trascinandoli. Senza entrare in riflessioni sulla psicologia, l'azione di spostare fisicamente, a mano, una nota da un posto a un altro è più potente e intenzionale di premere un bottone o un checkbox su uno schermo. Anche se non è possibile riprodurre completamente quest'esperienza fisica, è quantomeno possibile ispirarsi a essa per dare un minimo di peso ad azioni che, nel nostro caso, devono corrispondere a progressi reali di progetti e devono essere eseguite con un certo livello di consapevolezza;
- Sul Wall deve essere possibile effettuare zoom in / out e spostarsi mediante pan;
- Automazione: gli oggetti del Wall devono avere un comportamento "smart". Ad esempio: i post-it devono gestire il proprio stato di ritardo in automatico in base alla settimana in cui si trovano e quella in cui sono stati creati; le settimane devono aggiornarsi automaticamente, indicando quella corrente e visualizzando sempre un numero adeguato di settimane;
- Definire una color convention per poter distinguere velocemente note di tipo diverso, stato di completamento e ritardo;
- L'applicazione deve essere disponibile solo a utenti di Zero12 e lo stato del Wall modificabile solo dai PM;
- I progetti devono essere collegati ai rispettivi progetti Jira.

Oltre al Wall i PM utilizzano un'altra tabella per la gestione del **Backlog** delle discussioni dei meeting Scrum of Scrums. In particolare questa tabella ha tre colonne: "Todo", "Doing", "Done". Le righe sono invece formate dai nomi dei project manager. I post-it inseriti sono relativi ad attività o commenti il cui progresso è utile da discutere durante i meeting. È stato richiesto di ricreare anche questa tabella, in modo simile a quanto fatto per il Wall. Tuttavia in questo caso non è richiesto alcun comportamento "smart" della tabella o delle note associate.

Sono state segnalate come opzionali anche le seguenti feature più avanzate:

- Collaborazione Real Time: permettere a utenti connessi di lavorare insieme, vedendo in tempo reale i rispettivi cursori, note selezionate e modifiche in modo simile a quanto accade sulla piattaforma Mirò.
- Replay Meeting Wall: la possibilità di tracciare le modifiche dello stato del Wall per poter rivedere come si è evoluta la situazione durante un particolare meeting;

Il progetto è stato diviso in due e affidato a stagisti diversi. Quanto descritto finora (Wall e Backlog) è stato affidato a me. La parte complementare, affidata a un altro tirocinante, può essere riassunta come segue:

- Gestione del Work in Progress (WIP): dato che il gruppo di cui Zero12 fa parte è quotato in borsa, è richiesto di avere un tracciamento preciso dei progressi di tutti i progetti dell'azienda. Si necessita quindi di collegare i progetti attivi a dei parametri di progresso. Applicare un certo livello di automazione nella gestione di questi dati può essere di grande valore;
- OAuth: gestire la parte di autenticazione e autorizzazione mediante login con Google.

Le due parti devono comunque essere periodicamente integrate e funzionare correttamente insieme.

Alcune delle parti di questo progetto contengono informazioni sui processi interni di Zero12. Essendo questo un documento pubblicamente accessibile, quelle più sensibili verranno omesse.

1.3 Struttura della Relazione

Il seguente documento è strutturato come segue:

1. **Introduzione**: informazioni iniziali su azienda e progetto;
2. **Organizzazione e Supporto**: descrizione delle principali attività relative ai processi organizzativi e di supporto;
3. **Analisi dei Requisiti**: definizione dei requisiti del progetto individuati;
4. **Progettazione**: introduzione delle tecnologie utilizzate e loro applicazione nelle parti di backend e frontend del progetto;
5. **Codifica**: implementazione del design definito nella parte di progettazione;
6. **Testing**: descrizione e applicazione dei test effettuati;
7. **Conclusioni**: esposizione del risultato prodotto, bilancio retrospettivo del periodo di stage e miglioramenti ed estensioni del progetto.

Seguono in coda un **glossario** dei termini utilizzati e la **bibliografia** completa.

Capitolo 2

Organizzazione e Supporto

2.1 Way of Working

Per affrontare un progetto impegnativo come quello richiesto è necessario costruire una collaborazione adeguata tra tutti gli stakeholder. Per raggiungere gli obiettivi di efficacia ed efficienza è necessario tenere alto il livello di coinvolgimento di tutti, attraverso connessione e fiducia.

Durante lo stage si è cercato di adattarsi quanto più possibile al metodo di lavoro utilizzato internamente da Zero12, in particolare: metodologia Agile Scrum, sviluppo di Minimum Viable Products (MVP).

2.1.1 Scrum

Scrum è un metodo di gestione di progetto. Particolarmente adatto allo sviluppo software, può trovare adozione anche in campi come la ricerca, il marketing e nell'industria high tech. È pensata per team composti da al massimo 10 persone. Il lavoro viene diviso in obiettivi da completare in determinati intervalli di tempo, tipicamente di due settimane e mai superiori ad un mese. Durante questi periodi, il team si allinea ogni giorno in meeting di massimo 15 minuti. Alla fine del periodo il team di sviluppo espone agli stakeholder il lavoro svolto per ricavare feedback rapidamente. Viene infine fatta una retrospettiva che permetta al team di riflettere e migliorare.

Il nome "Scrum" è stato utilizzato per la prima volta nel 1986, in una pubblicazione del Harvard Business Review di Hirotaka Takeuchi e Ikujiro Nonaka. Il termine deriva dal rugby ed è utilizzato per enfatizzare il lavoro di squadra.

L'idea di base del metodo Scrum è il miglioramento continuo, sia del team, che nel tempo migliora le sue conoscenze e abilità, sia del prodotto, che si evolve e si adatta a diversi fattori fluttuanti. Tiene in considerazione sia il fatto che il team potrebbe non conoscere tutto all'inizio del progetto, sia che i requisiti del prodotto possano cambiare. Tale metodo è strutturato per adattarsi naturalmente al cambiamento di priorità e condizioni. Inoltre lo Scrum è strutturato ma non completamente rigido. La sua esecuzione può essere su misura dei bisogni di chi lo utilizza.

I ruoli principali nello Scrum sono:

- **Product Owner**: si occupa di avere un comprensione del business, del mercato e del cliente per poter decidere dove assegnare priorità. Si concentra sull'assicurare che il team di sviluppo porti il massimo valore all'attività. Per avere una guida chiara, il product owner è spesso un individuo;

- **Sviluppatori**: si occupano di portare a termine il lavoro che incrementa il valore del prodotto a ogni sprint. Il termine sviluppatore è inteso in senso lato e include non solo i programmatori ma anche: ricercatori, designers, data specialists, esperti di statistica, analisti, ingegneri, tester, etc. Il team di sviluppo si autogestisce il lavoro assegnato durante lo sprint;

- **Scrum Master**: è la figura responsabile di assicurare che la metodologia Scrum sia rispettata, educando il team di sviluppo e gli stakeholder chiave. Si occupa di rimuovere gli ostacoli che limitano le abilità del team a portare a termine gli obiettivi, facendo da barriera tra il team e distrazioni esterne. A differenza di un più tradizionale Project Manager, non si occupa della gestione del personale.

Il workflow Scrum prevede i seguenti momenti principali:

- **Sprint Planning**: è l'evento in cui il team concorda gli obiettivi del prossimo sprint, seleziona dal product backlog le attività che contribuiscono a tali obiettivi, costruendo così lo sprint backlog. La durata massima è di otto ore per uno sprint di quattro settimane;

- **Sprint**: è il momento in cui il team di sviluppo investe risorse per produrre l'incremento del prodotto. Lo Scrum enfatizza la produzione di output potenzialmente rilasciabile, quindi: completamente integrato, testato e documentato. La durata dello sprint è compresa tra una settimana e un mese;

- **Daily Scrum**: è una riunione in cui tutti gli sviluppatori del team si aggiornano velocemente sullo stato di avanzamento dello sprint e dove vengono identificati problemi bloccanti. Non è il momento di discussione e soluzione di tali problemi che, se necessario, verranno ripresi esclusivamente dagli interessati e approfonditi in sessioni apposite. Detto anche Standup Meeting dato che può essere condotto con tutti i partecipanti in piedi, per agevolare la rapidità dell'incontro. La durata non dovrebbe superare i 15 minuti;

- **Sprint Review**: il team presenta il lavoro completato agli stakeholder, tipicamente mediante una demo. Vengono ascoltati i feedback, discusso il lavoro non terminato e raccolti suggerimenti sulle prossime priorità. La durata raccomandata è di due ore per un sprint di due settimane;

- **Sprint Retrospective**: il team riflette sullo sprint passato, in particolare su: cosa è andato bene, cosa è andato male e come si può migliorare concretamente dal prossimo sprint. La durata massima è di tre ore per uno sprint di quattro settimane.

I principali artefatti prodotti dallo Scrum sono:

- **Product Backlog**: è la "TODO List" principale del progetto e viene mantenuta dal Product Owner. È una lista dinamica di feature, requisiti, miglioramenti, bug fix del prodotto. Viene costantemente rivista è le priorità cambiano in base a molti fattori;

- **Sprint Backlog**: è la lista di elementi selezionati dal team che verranno trattati durante lo sprint. Può essere flessibile ed evolversi durante il corso dello sprint, tuttavia non ci devono essere compromessi su obiettivi base dello sprint;

- **Incremento**: è l'output potenzialmente rilasciabile di uno sprint che ha soddisfatto gli obiettivi. È formato dagli elementi completati dello sprint backlog, il cui lavoro è integrato nel prodotto esistente. Gli elementi del backlog sono completati secondo una ben predefinita "Definition of Done", funzionanti ed utilizzabili indipendentemente dalla volontà del Product Owner di effettuare il rilascio.

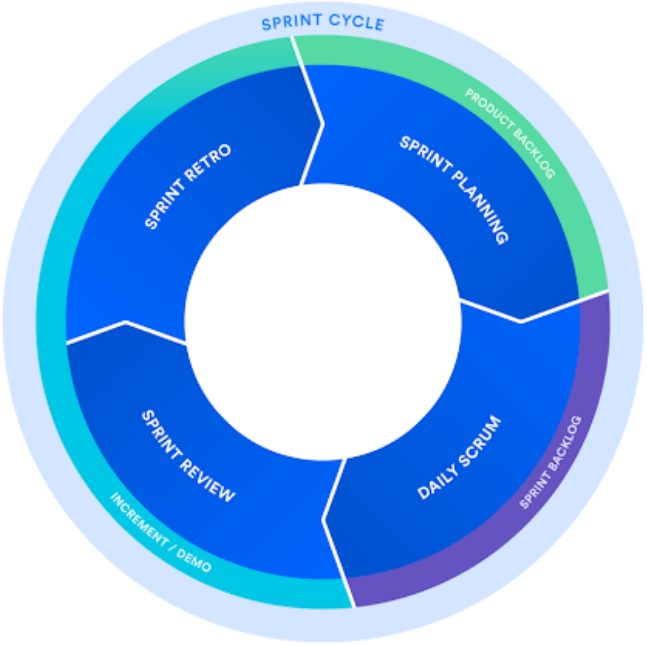


Figura 2.1: Scrum: schema di uno sprint con eventi e artefatti

Concretamente, durante il periodo di stage, l'esperienza di Scrum è stata leggermente divergente da quella teorica. Questo sia per l'applicazione interna all'azienda delle metodologie agili, sia per il velo di protezione offerta dal tirocinio, che spesso ci ha agevolato e tutelato rispetto ai rischi che ci possono effettivamente essere nel mondo del lavoro. Nello specifico:

- Il ruolo di product owner è stato assunto dal CEO di Zero12 *Stefano Dindo*. Il team di sviluppo era composto da noi 2 stagisti (*Sara Nanni* e *Marko Vukovic*). I tutor interni *Francesco Meneguzzo* e *Federico Pinton* possono essere visti come scrum master, anche se concretamente hanno avuto un ruolo più esteso, educandoci non solo nelle metodologie e best practice ma anche sotto l'aspetto tecnico e offrendoci supporto in caso di necessità;
- Essendo un tirocinio curricolare della durata di due mesi, i tre sprint di due settimane ciascuno erano già definiti in modo generico. Ad ogni sprint si raffinavano gli obiettivi sulla base delle competenze dimostrate e dal lavoro fatto. Più che

un backlog, le attività venivano gestite mediante una Kanban board su Trello, con elementi che potevano essere nello stato di "Todo", "Doing" e "Done";

- Gli standup meeting si sono svolti quotidianamente, permettendo di avere un feedback molto rapido da parte dei tutor. Avere periodicamente conferma che la direzione seguita è corretta ha avuto un grande impatto sulla serenità del team nell'affrontare il progetto di stage;

- Alla fine di ogni sprint si è svolta una demo con stagisti, tutor e product owner. Inoltre alla fine del progetto si è tenuta una presentazione del prodotto "finale" davanti a tutti i collaboratori di Zero12, sia in presenza che in remoto.

Infine il progetto di per sé era fortemente legato ai bisogni dell'azienda di agevolare l'adozione di processi Agile in generale. In particolare i meeting Scrum of Scrums, in cui i PM si allineano sull'avanzamento di diversi progetti.

2.1.2 MVP: Minimum Viable Product

Un **Minimum Viable Product (MVP)** è una versione di un prodotto con solamente le funzionalità sufficienti per poter effettuare il rilascio. Questo approccio allo sviluppo può essere collegato alla metodologia Scrum discussa prima. Infatti ogni incremento del prodotto dovrebbe essere potenzialmente rilasciabile. Questo permette di ricevere feedback rapidamente e guidare l'attenzione degli sviluppatori verso le necessità effettive dell'utente finale. Inoltre questo approccio permette di sperimentare investendo la minima quantità di risorse. La velocità di sviluppo consente anche di inserirsi velocemente nel mercato. Durante il tirocinio, questo sarebbe il risultato ideale da proporre alla fine di ogni sprint pianificato.

Questo approccio è contrapposto a un tipo di sviluppo che, anche se incrementale (nei casi migliori, in realtà non è un traguardo così semplice da raggiungere), prevede il rilascio di un prodotto solo al completamento dello sviluppo del prodotto finale.

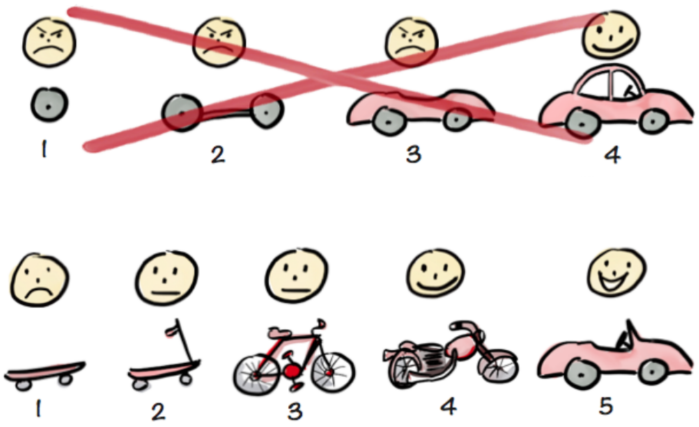


Figura 2.2: Illustrazione delle differenze tra sviluppo incrementale di un prodotto con rilascio finale e di MVP

2.2 Gestione della Configurazione

Per poter utilizzare i servizi AWS, ci sono state fornite delle credenziali di accesso, con permessi limitati e utilizzabili solamente attraverso CLI. Questo ci ha permesso comunque di utilizzare il nostro ambiente di sviluppo preferito (e.g. sistema operativo, shell, editor) per effettuare tutte le operazioni necessarie di configurazione e sviluppo. In caso di mancanza di permessi, le risorse ci venivano fornite direttamente dai tutor.

2.2.1 Versionamento

Per il progetto sono state predisposte 2 Repository vuote, `agile-wall-api` per la parte backend e `agile-wall-web` per la parte di frontend. È stato compito nostro inizializzarle e mantenerle nello stato migliore possibile. Il servizio di hosting delle Repository è **AWS CodeCommit** e utilizzabile mediante comandi `git`.

Il workflow utilizzato per il versionamento è il **Git Flow**. Di norma prevede i seguenti branch:

- `develop`: contiene la versione del software attualmente in sviluppo;
- `feature/...`: è un insieme di rami, uno per ciascuna feature in corso di sviluppo;
- `release/vX.X.X`: è un insieme di rami, uno per release. Permette di prepararsi al rilascio, correggendo bug minori e aggiungendo i metadati necessari;
- `main` (o `master`): contiene la più recente versione rilasciata del software;
- `hotfix/...`: è un insieme di rami, uno per correzione. Si apre nel caso sia indispensabile correggere un errore nella versione attualmente rilasciata e non sia possibile aspettare il prossimo ciclo di release.

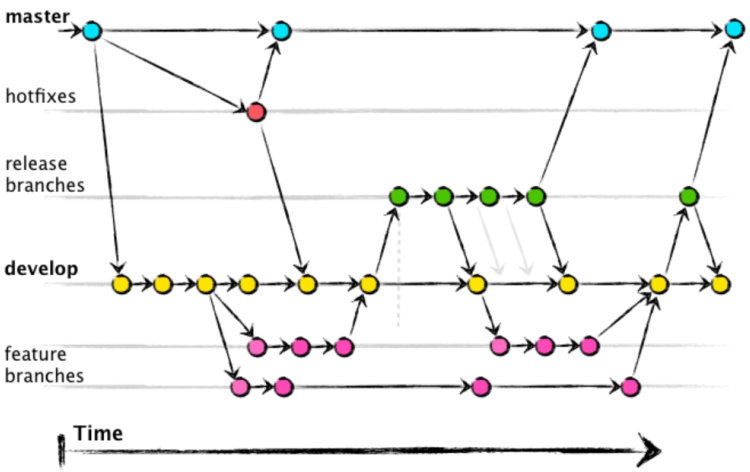


Figura 2.3: Workflow Git Flow - di Vincent Driessen

Questo workflow è utilizzabile mediante CLI `git flow` o mediante client GUI come *Sorucetree*.

2.2.2 Continuous Integration Pipeline

Durante lo stage ogni piccola feature sviluppata veniva integrata nell'applicazione. Questo frequente allineamento ci ha permesso di evitare seri momenti di *"merge hell"* in cui il codice sviluppato in parallelo è così differente da essere difficile incorporarlo nell'applicazione. Questo è stato reso possibile da una pipeline di CI offerta dal servizio **AWS CodePipeline**, installata sia nel Repository backend che frontend. In pratica si sono definiti due ambienti distinti: `dev` (Sviluppo) e `prod` (Produzione). Ciascun ambiente permette di avere una copia indipendente di tutte le risorse rilasciate. Questa pipeline è strettamente collegata al sistema di versionamento: un push nel ramo `develop` innesca un rilascio nell'ambiente di `dev`; un push nel ramo `master` innesca un rilascio nell'ambiente di `prod`. A ciascun rilascio sono associati delle fasi, in cui è possibile specificare i comandi da eseguire sulla macchina remota, necessari affinché la release avvenga correttamente. Vediamo in dettaglio il file che specifica come la CI effettua un rilascio della parte backend:

```
phases:
  install:
    runtime-versions:
      nodejs: 14
  pre_build:
    commands:
      - echo Build started on `date`
      - npm install -s
      - npm install -g serverless@3.16.0 -s
  build:
    commands:
      - echo Entered the build phase...
      - serverless deploy --stage ${ENV}
  post_build:
    commands:
      - echo Entered the post build phase...
      - echo Build completed on `date`
```

- `install`: viene specificato il runtime utilizzato e la versione;
- `pre_build`: vengono installate le dipendenze necessarie per effettuare il deploy;
- `build`: viene effettuato il deploy dell'applicazione;
- `post_build`: se la build è avvenuta con successo, vengono stampate delle informazioni utili.

Possiamo notare i comandi `echo` che permettono di avere nei log, consultabili attraverso la console AWS, gli orari ed eventualmente altre informazioni vantaggiose. Questo è specialmente utile in caso di errori per capire dove cercare eventuali bug.

2.3 Coordinamento

2.3.1 Comunicazioni

Svolgere un tirocinio **in presenza** è stato estremamente vantaggioso per noi studenti. Non solo per la rapidità con cui si possono apprendere vari processi aziendali ma anche

per la disponibilità di comunicazione interna ed esterna al team. Infatti, in caso di dubbi o problematiche, era sempre possibile rivolgersi a qualcuno per ricevere supporto.

Siamo inoltre stati inseriti come ospiti nei canali **Slack** di Zero12, per poter comunicare sia in asincrono, sia con eventuali collaboratori in remoto.

2.3.2 Meeting

Nel periodo di stage i momenti di riunione formale sono stati fondamentalmente due:

- **Daily Stand Up**: come imposto dall'approccio Scrum, ogni giorno il team si allineava sui progressi del progetto. Occasionalmente si utilizzava questo tempo anche per cercare di risolvere problematiche;
- **Demo** con product owner: a fine sprint (ogni 2 settimane) si incontrava il product owner per mostrare la versione più recente del prodotto e discutere feedback.

2.4 Pianificazione

Nonostante sia stato stilato un iniziale piano di lavoro, all'inizio di ciascuno sprint, si definivano obiettivi più specifici. Nel caso si concludesse prima o dopo il previsto, si riorganizzava il tempo in modo che fosse quantomeno produttivo per lo stagista. Sotto questo aspetto, l'approccio Scrum e specialmente gli stand up giornalieri, hanno contribuito molto a rendere sereno tutto il periodo dello stage, garantendo sempre di essere su una strada produttiva e non perdersi.

2.5 Formazione

Tipicamente l'acquisizione di nuove competenze seguiva questo schema:

1. Veniva fornita dai tutor della documentazione, spesso ufficiale, su quanto necessario apprendere;
2. Seguiva un periodo di studio e ricerca personale da parte di noi tirocinanti;
3. Si concludeva con la verifica di quanto appreso ed eventuali chiarificazioni dei tutor o di figure esperte dell'azienda.

La formazione è stata concentrata in buona parte nella prima settimana di tirocinio. Tuttavia, vista la complessità dei temi e delle tecnologie trattate, per raggiungere un livello di competenze adeguato, l'apprendimento è comunque rimasto una costante durante tutto lo stage. Alcune tecnologie richiedono anche molto più dei due mesi a disposizione dello stage per essere utilizzate in modo esperto.

2.6 Organizzazione Personale

Infine è stato soggettivamente utile avere degli strumenti di organizzazione personale come:

- **Calendario**: in cui appuntare eventi e scadenze da rispettare;
- **Backlog**: utilizzata come todo list personale;

- **Diario**: per documentare ciascun giorno di tirocinio;
- **Lista Q&A**: per tenere traccia di domande e risposte da fare ai professionisti presenti in azienda.

2.7 Organizzazione dell'Attività di Stage

Lo stage curricolare ha avuto una durata di 8 settimane, precisamente dal *27/04/2022* al *22/06/2022*. Durante il periodo di stage i tutor aziendali che hanno seguito il progetto *Agile Wall* sono stati *Francesco Meneguzzo* e *Federico Pinton*. Lo stage è stato svolto negli uffici di Padova di Zero12, con sede in via Salboro 22B. Viste le dimensioni del progetto, Zero12 ha da subito diviso il progetto in due parti complementari. Quella descritta precedentemente è stata affidata a me, l'altra, accennata nella parte finale della sezione precedente, è stata affidata alla studentessa *Sara Nanni*, con cui si è collaborato durante l'intero periodo di stage per garantire coesione e uniformità.

Lo scopo dello stage è stato di permettere allo studente di interfacciarsi al mondo professionale dello sviluppo software, acquisendo una conoscenza full stack e operando secondo i processi e le metodologie aziendali di Zero12. I principali obiettivi formativi inizialmente pianificati sono stati:

- Ampliare e approfondire le proprie conoscenze tecniche e tecnologiche;
- Apprendere le dinamiche di sviluppo serverless;
- Lavorare in un team secondo metodologie di lavoro Agile.

La pianificazione iniziale del periodo di stage è stata la seguente:

- Periodo 1 (40 ore): ricerca e studio del dominio e delle tecnologie utili allo sviluppo del progetto;
- Periodo 2 (80 ore), *Sprint 1*: progettazione, sviluppo e testing di API e User Interface del Wall di progetti;
- Periodo 3 (80 ore), *Sprint 2*: progettazione, sviluppo e testing di API e User Interface del Backlog delle attività dei PM;
- Periodo 4 (80 ore), *Sprint 3*: progettazione, sviluppo e testing di API e User Interface della parte di collaborazione real time sul Wall di progetti;
- Periodo 5 (40 ore): finalizzazione del progetto.

I prodotti inizialmente attesi erano:

- Infrastruttura Cloud AWS e sistemi CI/CD dei servizi realizzati;
- Sviluppare e documentare API relative alle funzioni;
- Sviluppare e documentare User Interface associata alle funzioni;
- Documentazione dell'intero progetto di stage;
- Articolo per il blog di Zero12 dove raccontare l'esperienza vissuta in azienda.

I principali rischi previsti dal piano di stage erano:

- Apprendimento dei processi agile di Zero12;
- Complessità dello sviluppo full stack;
- Sviluppo di un applicativo serverless;
- Integrazione con servizi sviluppati da terzi.

Capitolo 3

Analisi dei Requisiti

Nel seguente capitolo verranno evidenziati i requisiti individuati. Verranno ripresi poi nelle conclusioni per avere una misura del livello di successo del progetto. Gli obiettivi del progetto sono stati esposti all'inizio del tirocinio. I requisiti principali sono stati determinati subito insieme al product owner e quando necessario raffinati a inizio sprint. La seguente lista contiene i requisiti relativi a obiettivi soggetti ai tre sprint effettuati. Non sono stati approfonditi gli obiettivi di feature desiderabili ma non pianificate. Inoltre sotto sono riportati solo i requisiti relativi alla parte del progetto da me trattata e non la parte complementare svolta dalla mia collega *Sara Nanni*.

Per assicurare che la comprensione dei bisogni del proponente fosse correttamente compresa da tutto il team, nella prima settimana di tirocinio sono stati creati dei mockup dell'applicazione mediante **Balsamiq Wireframes**. Oltre a confermare i requisiti iniziali, questo strumento ha permesso al product owner di essere molto più specifico nel esprimere i suoi bisogni e integrare alcuni requisiti non emersi inizialmente.

L'identificatore dei requisiti seguirà la seguente convenzione:

- **R**: requisito;
- Tipologia:
  - **F**: requisito funzionale;
  - **Q**: requisito di qualità;
  - **V**: requisito di vincolo.
- Importanza:
  - **1**: requisito obbligatorio;
  - **2**: requisito desiderabile ma non obbligatorio;
  - **3**: requisito opzionale.
- Codice:
  - Identificativo univoco del requisito in base alla tipologia. In alcuni casi sono presenti dei "sotto-codici" preceduti da un "." (punto) per evidenziare un requisito strettamente legato a un altro ma con un livello di dettaglio maggiore.

3.1 Lista dei Requisiti

Identificativo	Definizione
RF1.1	L'utente deve poter accedere all'applicazione con le sue credenziali Google di dominio Zero12
RF1.2	L'utente deve poter effettuare il logout dall'applicazione
RF1.3	L'utente deve poter visualizzare il Wall di progetti
RF1.4	L'utente deve poter muovere liberamente la viewport nel Wall di progetti
RF1.5	L'utente deve poter effettuare zoom in / out nel Wall di progetti
RF1.6	Nel Wall deve essere evidenziata la settimana corrente
RF1.7	Nel Wall devono essere visualizzate almeno 3 settimane passate rispetto alla settimana corrente
RF1.8	Nel Wall devono essere visualizzate un numero sufficiente di settimane future rispetto alla settimana corrente
RF1.9.1	Le note visualizzate sul Wall devono avere un titolo
RF1.9.2	Le note visualizzate sul Wall devono avere un testo
RF1.9.3	Le note visualizzate sul Wall devono poter essere di tipo "milestone"
RF1.9.4	Le note visualizzate sul Wall devono poter essere di tipo "commento"
RF1.9.5	Le note visualizzate sul Wall di tipo "milestone" devono avere uno stato binario di "completamento"
RF1.9.6	Le note visualizzate sul Wall di tipo "milestone" ancora attive (non completate) devono avere uno stato binario di "ritardo", automaticamente impostato in base alla settimana in cui si trova la nota e la settimana in cui è stata creata inizialmente
RF1.9.7	Il tipo delle note visualizzate sul Wall devono essere distinguibili in base a una color convention

RF1.9.8	Lo stato delle note di tipo "milestone" deve essere evidenziato graficamente
RF1.10	L'utente deve poter visualizzare il numero totale di note di tipo "milestone" attualmente in ritardo
RF1.11.1	L'utente PM deve poter inserire il titolo di una nuova nota del Wall
RF1.11.2	L'utente PM deve poter inserire il testo di una nuova nota del Wall
RF1.11.3	L'utente PM deve poter selezionare il tipo di una nuova nota del Wall
RF1.11.4	L'utente PM deve poter selezionare il progetto a cui la nuova nota appartiene
RF1.11.5	L'utente PM deve poter selezionare la settimana iniziale a cui la nuova nota appartiene
RF1.12.1	L'utente PM deve poter spostare una nota del Wall tra le settimane all'interno della riga del progetto corrispondente trascinandola. Ciò può modificare lo stato di "ritardo" della nota corrispondente
RF1.12.2	L'utente PM deve poter modificare il titolo di una nota esistente del Wall
RF1.12.3	L'utente PM deve poter modificare il testo di una nota esistente del Wall
RF1.12.4	L'utente PM deve poter modificare lo stato di completamento di una nota di tipo "milestone" esistente del Wall
RF1.13	l'utente PM deve poter eliminare una nota esistente dal Wall di progetti
RF1.14.1	I progetti visualizzati sul Wall devono avere un nome
RF1.14.2	I progetti visualizzati sul Wall devono avere un cliente associato
RF1.14.3	I progetti visualizzati sul Wall devono avere un project manager di riferimento

RF1.14.4	I progetti visualizzati sul Wall devono avere un link alla pagina Jira corrispondente
RF1.15.1	L'utente PM deve poter inserire il nome di un nuovo progetto
RF1.15.2	L'utente PM deve poter inserire il cliente associato a un nuovo progetto
RF1.15.3	L'utente PM deve poter inserire il project manager di riferimento di un nuovo progetto
RF1.15.4	L'utente PM deve poter inserire il link alla pagina Jira corrispondente a un nuovo progetto
RF1.16.1	L'utente PM deve poter modificare il nome di un progetto esistente
RF1.16.2	L'utente PM deve poter modificare il cliente associato a un progetto esistente
RF1.16.3	L'utente PM deve poter modificare il project manager di riferimento di un progetto esistente
RF1.16.4	L'utente PM deve poter modificare il link alla pagina Jira corrispondente a un progetto esistente
RF1.16.5	L'utente PM deve poter archiviare un progetto esistente e tutte le note associate, nascondendoli dal Wall
RF1.16.6	L'utente PM deve poter riaprire un progetto archiviato e tutte le note associate, riportandoli nel Wall
RF1.17	L'utente PM deve poter eliminare un progetto esistente e tutte le note associate
RF1.18	L'utente PM deve poter visualizzare il Backlog delle attività dei PM
RF1.19	L'utente PM deve poter muovere liberamente la viewport nel Backlog delle attività dei PM
RF1.20	L'utente PM deve poter effettuare zoom in / out nel Backlog delle attività dei PM
RF1.21	Nel Backlog delle attività dei PM devono essere presenti le colonne "Todo", "Doing", "Done"

RF1.22.1	Le note visualizzate sul Backlog delle attività dei PM devono avere un testo
RF1.22.2	Le note visualizzate sul Backlog devono poter essere di tipo "attività"
RF1.22.3	Le note visualizzate sul Backlog devono poter essere di tipo "commento"
RF1.22.4	Il tipo delle note del Backlog deve essere distinguibile in base a una color convention
RF1.23.1	L'utente PM deve poter inserire il testo di una nuova nota del Backlog
RF1.23.2	L'utente PM deve poter selezionare il tipo di una nuova nota del Backlog
RF1.23.3	L'utente PM deve poter selezionare il PM a cui la nuova nota del Backlog è assegnata
RF1.23.4	L'utente PM deve poter selezionare la colonna iniziale a cui la nuova nota del Backlog appartiene
RF1.24.1	L'utente PM deve poter spostare una nota del Backlog tra le colonne all'interno della riga del PM corrispondente trascinandola
RF1.24.2	L'utente PM deve poter modificare il testo di una nota del Backlog esistente
RF1.25	L'utente PM deve poter eliminare una nota esistente dal Backlog delle attività dei PM
RF1.26	L'utente PM deve poter visualizzare tutti i project manager nel Backlog delle attività dei PM
RF1.27	L'utente PM deve poter creare un nuovo project manager nel Backlog delle attività dei PM
RF1.28	L'utente PM deve poter modificare un project manager esistente nel Backlog delle attività dei PM
RF1.29	L'utente PM deve poter eliminare un project manager esistente e tutte le note associate

RF1.30.1	L'utente autenticato deve poter visualizzare il nome relativo al proprio account Google di dominio Zero12
RF1.30.2	L'utente autenticato deve poter visualizzare l'immagine profilo relativa al proprio account Google di dominio Zero12
RF1.31	L'utente deve poter visualizzare in real time gli altri utenti connessi all'applicazione
RQ1.1	Deve essere fornita la documentazione sufficiente per la manutenzione ed estensione del prodotto
RQ1.2	Il codice sorgente del prodotto deve essere gestito utilizzando il workflow "Git Flow"
RQ2.3	Il codice sorgente deve essere accompagnato da Unit Testing
RQ1.4	L'applicazione deve essere sviluppata seguendo le norme e le best practices di Zero12
RV1.1	L'applicazione deve essere accessibile solo a utenti zero12
RV1.2	Il codice sorgente del prodotto deve essere disponibile nei Repository git di Amazon CodeCommit forniti
RV2.3	L'applicazione deve essere rilasciata su infrastrutture AWS in un ambiente di develop
RV2.4	L'applicazione deve essere rilasciata su infrastrutture AWS in un ambiente di production
RV1.5	Le API dell'applicazione devono essere sviluppate in TypeScript

Tabella 3.1: Requisiti del progetto

Capitolo 4

Progettazione

4.1 Tecnologie

Lo stack tecnologico utilizzato nel progetto è un sottoinsieme delle tecnologie studiate durante il periodo di stage. Infatti, una volta chiariti gli obiettivi dell'applicazione, si è utilizzato del tempo per la ricerca e lo studio, guidato dai tutor, di possibili tecnologie che potevano essere utili alle nostre esigenze. Di seguito vengono riportate solamente le tecnologie strettamente utilizzate nel progetto, con alcune spiegazioni ed esempi dove necessario, per giustificare la loro adeguatezza. Verranno prima discusse le tecnologie generali in comune tra frontend e backend, per poi inoltrarsi in quelle specifiche per ciascuna categoria.

4.1.1 Generali

Node.js & npm

Figura 4.1: Logo di Node.js e npm

Node.js è un runtime **JavaScript** open-source e cross-platform che utilizza **V8** per eseguire codice JavaScript fuori da un web browser. È progettato per costruire applicazioni di rete facilmente scalabili. Permette alle applicazioni web di aderire al paradigma *"JavaScript everywhere"*, nel bene e nel male, potendo utilizzare il linguaggio sia lato client che lato server. Sono presenti moduli per gestire: I/O file system, networking, dati binari, crittografia, e molto altro.

La libreria standard di Node.js offre un ampio insieme di **funzioni asincrone** che impediscono al codice di essere bloccante. A questo paradigma **non-blocking** aderiscono anche moltissime librerie, rendendolo la norma in questo ambiente. Esistono comunque API per eseguire operazioni in sincrono. Tutto questo permette a Node.js di gestire connessioni concorrenti con un singolo server senza però dover affrontare il problema della gestione concorrente di thread, che può essere una significativa fonte di bug.

npm (Node Package Manager) è il gestore di pacchetti di default di Node.js. Consiste di un client CLI e un database online di pacchetti pubblici e privati. I pacchetti possono essere installati mediante il client e ricercati dal sito web di npm. Le dipendenze di un particolare progetto vengono specificate in un file chiamato `package.json` e installate localmente in una cartella chiamata `node_modules`. Per capire meglio come funziona la gestione delle dipendenze, si porta come esempio il `package.json` della parte backend del progetto:

```
{
  "name": "agile-wall-api",
  "version": "1.0.0",
  "description": "Setup Serverless con plugin: \"serverless-offline\"
  \"serverless-plugin-typescript\", \"serverless-dynamodb-local\"",
  "author": "Marko Vukovic",
  "scripts": {
    "db": "serverless dynamodb start",
    "serverless": "serverless offline",
    "test": "jest",
    "test-coverage": "jest --coverage --verbose"
  },
  "devDependencies": {
    "@types/jest": "^27.5.1",
    "@types/jsonwebtoken": "^8.5.8",
    "@types/node": "^17.0.31",
    "@types/uuid": "^8.3.4",
    "jest": "^28.1.0",
    "serverless-dynamodb-local": "^0.2.40",
    "serverless-offline": "^8.7.0",
    "serverless-plugin-typescript": "^2.1.2",
    "ts-jest": "^28.0.3",
    "typescript": "^4.6.4"
  },
  "dependencies": {
    "@types/aws-lambda": "^8.10.97",
    "ajv": "^8.11.0",
    "aws-lambda": "^1.0.7",
    "aws-sdk": "^2.1127.0",
    "axios": "^0.27.2",
    "jsonwebtoken": "^8.5.1",
    "jwk-to-pem": "^2.0.5",
    "jwt-decode": "^3.1.2",
    "util.promisify": "^1.1.1",
    "uuid": "^8.3.2"
  }
}
```

Si possono osservare i seguenti campi:

- `name`: indica il nome del progetto;
- `version`: indica la versione corrente;

- `description`: breve descrizione;
- `author`: nome dell'autore;
- `scripts`: definizione di comandi che possono essere invocati mediante `npm run <nome_script>`
- `devDependencies`: indica i pacchetti esterni utilizzati per lo sviluppo dell'applicazione ma non sono strettamente necessari per l'esecuzione della stessa;
- `dependencies`: indica i pacchetti esterni all'applicazione necessari al suo funzionamento.

TypeScript

TypeScript è un linguaggio di programmazione sviluppato e mantenuto da Microsoft. Sintatticamente è un'estensione di JavaScript, sviluppata per compensare alcune mancanze del linguaggio. JavaScript infatti è ad oggi uno dei linguaggi di programmazione più popolari[1][2]. Non è questa la sede più adeguata per discutere del perchè, né di analizzare in profondità pro e contro. Tuttavia è importante conoscere questo fatto poiché giustifica la nascita e l'adozione così estesa di TypeScript. JavaScript infatti nasce come linguaggio di scripting per rendere le pagine web più dinamiche. Oggi invece è possibile utilizzarlo come linguaggio per il web sia lato client che lato server, per sviluppare CLI, applicazioni desktop e mobile. Nel 2007 *Jeff Atwood*, co-fondatore di *Stack Overflow*, cita quella che venne poi battezzata scherzosamente *"Atwood's Law"*: *"Any application that can be written in JavaScript, will eventually be written in JavaScript"*. Tuttavia il fatto che tutto questo sia possibile non indica necessariamente che sia una buona idea. JavaScript si è evoluto molto negli anni, tuttavia continua a sentirsi il peso delle sue origini che talvolta lo rendono poco adatto allo sviluppo di applicazioni di dimensioni importanti. In quest'ottica, TypeScript cerca di sopperire ad alcune lacune di JavaScript, rendendolo più adatto al ruolo che occupa nel mondo dello sviluppo software.

Figura 4.2: Logo di TypeScript

Il codice sorgente TypeScript (TS) viene "transpilato" in codice JavaScript (JS), rendendo possibile utilizzarlo in qualsiasi contesto in cui è utilizzabile JS. Questo rende anche possibile utilizzare in TS tutte le librerie JS già esistenti. Dall'altra parte, essendo TypeScript un'estensione di JavaScript, qualsiasi programma JS è anche un valido programma TS. In questo modo, TypeScript può essere adottato progressivamente anche da progetti JS già esistenti. Un grandissimo vantaggio di TS è che la transpilazione a JS può supportare anche versioni passate, rendendo possibile scrivere un'applicazione utilizzando le ultimissime feature offerte dal linguaggio ma al contempo garantendo retrocompatibilità con browser più datati.

Vediamo le principali caratteristiche di TypeScript:

- **Tipizzazione** e **Compile-Time Type Checking**:

[1] *Stack Overflow Developer Survey 2022*. URL: https://survey.stackoverflow.co/2022.
[2] *Github State of the Octoverse 2021*. URL: https://octoverse.github.com/.

```
function sum(x: number, y: number): number {
  return x + y;
}
```

In questo esempio possiamo vedere che i tipi di `x`, `y` e il tipo di ritorno della funzione `sum` sono ben definiti e non possono cambiare. Questo perché vengono controllati dal compilatore che, in caso, solleva un errore di compilazione. La cattura di errori a compile-time e non a run-time rende molto più facile individuare e gestire bug che potrebbero trovarsi nel codice;

- **Type Inference**: il compilatore può inferire automaticamente i tipi se provvisto di sufficienti informazioni. Nell'esempio di prima è infatti possibile omettere il tipo di ritorno della funzione `sum` dato che può essere inferita dal fatto che sia `x` che `y` sono `number` e che la somma di `number` resta un `number`. Annotare il codice resta comunque un potente strumento di verifica della correttezza. Se per mancanza di dichiarazioni un tipo non può essere inferito, di default assume il tipo dinamico `any`. Il comportamento di una variabile di tipo `any` è lo stesso di una qualsiasi variabile JavaScript. Si vanno quindi a perdere tutti i vantaggi di TypeScript;
- **Interfacce**: è possibile descrivere la "forma" di oggetti e funzioni.

```
// Esempio 1
interface Account {
  id: number
  name: string
  version: 1
}

function welcome(user: Account) {
  console.log(user.name)
}
```

```
// Esempio 2
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

Questo rende possibile non solo avere una (parziale) conferma a compile-time della correttezza ma anche di avere del codice più documentato e leggibile, il che si traduce in una più facile manutenibilità;

- Sono presenti altre feature, alcune proprie di TS, altre già introdotte in *ECMAScript 2015*: **generics**, **namespaces**, **async/await**, **classi**, **moduli**, **anonymous functions**, **parametri di default**, ...

4.1.2 Backend

Serverless

Innanzitutto è importante chiarire il **doppio significato** con cui il termine ***"serverless"*** verrà utilizzato in questo documento:

- Serverless è un **paradigma di cloud computing** in cui un provider alloca risorse on demand, gestendo l'infrastruttura per conto del cliente. Il termine può essere fuorviante dato che i server sono utilizzati dal cloud provider per eseguire il codice. Tuttavia viene astratto allo sviluppatore tutta la parte di pianificazione, configurazione, gestione, manutenzione, scalabilità delle infrastrutture coinvolte. Tipicamente nel serverless computing le risorse non vengono tenute in memoria volatile ma le computazioni avvengono in brevi esecuzioni e i risultati vengono salvati in storage. Questo significa che quando un'applicazione non viene utilizzata, le risorse possono essere deallocate. In questo modo si può pagare solo per le risorse che vengono effettivamente utilizzate. Tra i provider più noti ci sono: AWS, Microsoft Azure e Google Cloud. Pro e contro del serverless computing vanno analizzati caso per caso e non è possibile esprimere un giudizio universale. Tuttavia le motivazioni più frequenti di adozione del paradigma serverless sono:
  - Costi: che sono più flessibili e proporzionali all'utilizzo effettivo delle risorse;
  - Scalabilità: il cloud provider è tipicamente responsabile di scalare la capacità delle risorse necessarie;
  - Produttività: spesso si utilizza modello "function as a service", per cui l'effettivo codice che gestisce le richieste è tipicamente contenuto in una funzione.

  Tra gli svantaggi invece ci possono essere:
  - Performance: dato che le risorse di una particolare applicazione non sono sempre allocate, ci può essere una latenza maggiore rispetto ad avere un server dedicato che è sempre attivo;
  - Vendor Lock-In: applicazioni software che eseguono in ambienti serverless sono tipicamente legate al provider e la migrazione non è sempre triviale.
- Serverless è anche un **Framework** open source scritto in Node.js. Permette di utilizzare il codice sia per configurare le risorse necessarie all'applicazione (e.g. database, web storage) sia per definire le funzioni che andranno effettivamente a gestire il backend. Applicazioni sviluppate con Serverless possono essere rilasciate su infrastrutture AWS e anche altri provider.

Figura 4.3: Logo del Serverless Framework

Ci riferiamo ora al Serverless Framework. Nel caso del nostro progetto, il provider è AWS. Analizzeremo in dettaglio le tecnologie AWS a cui ci siamo appoggiati dopo aver introdotto alcune idee di Serverless. I concetti principali sono:

- **Funzione**: ogni funzione è un'unità di esecuzione indipendente. Una funzione può ad esempio: salvare dati su un database, processare un file, eseguire un'attività schedulata;
- **Eventi**: sono dei trigger per l'esecuzione di funzioni. Esempi di eventi possono essere: richieste HTTP, upload di file su web storage, un qualche tipo di notifica;
- **Risorse**: sono componenti legati a servizi e/o infrastrutture (di AWS nel nostro caso). Ne sono un esempio: bucket S3, tabelle DynamoDB, topic SNS;
- **Servizi**: sono unità organizzative di un progetto. Eseguendo il comando `serverless deploy` vengono rilasciate tutte le funzioni e le risorse definite in un file di configurazione. Vediamo un esempio:

```yaml
service: users

functions:
  usersCreate: # Funzione che crea un nuovo utente
    events: # Evento che triggera la funzione
      - httpApi: 'POST /users/create'
  usersDelete: # Funzione che crea un utente
    events:
      - httpApi: 'DELETE /users/delete'

resources: # Risorse necessarie
  userTable: # Storage per gli utenti
  Type: AWS::DynamoDB::Table
  Properties:
    TableName: userTable
    AttributeDefinitions:
      - AttributeName: uuid
        AttributeType: S
    KeySchema:
      - AttributeName: uuid
        KeyType: HASH
    ProvisionedThroughput:
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
```

  I file di configurazione possono essere scritti in: YAML, JSON, JavaScript, TypeScript. Il file principale di configurazione è `serverless.yml`. In base alle dimensioni del progetto, questo file può essere frammentato;
- **Plugin**: le funzionalità del Framework possono essere estese e/o sovrascritte mediate plugin. Per il nostro progetto sono stati utilizzati i seguenti plugin:
  - `serverless-plugin-typescript`: permette di utilizzare TypeScript senza bisogno di installare e configurare un compilatore;
  - `serverless-offline`: permette di emulare in locale chiamate a eventi che triggerano le funzioni utilizzando un server HTTP locale;
  - `serverless-dynamodb-local`: permette di avere un'istanza locale di DynamoDB per facilitare lo sviluppo.

Per il nostro progetto ci siamo appoggiati ai servizi AWS, in particolare:

- **AWS Lambda**: piattaforma serverless event-driven, ovvero un modello di cloud computing in cui il provider si occupa di allocare risorse macchina on-demand. Questo permette agli sviluppatori di non doversi preoccupare di configurare, gestire, manutenere o scalare l'infrastruttura sottostante, sia essa un insieme di container, macchine virtuali o fisiche. Il codice viene eseguito in risposta ad eventi come, ad esempio: click di un utente su una web application, upload di un oggetto su S3, aggiornamento di un database, lettura da un sensore di un dispositivo IoT. Ufficialmente sono supportati Node.js, Python, Java, Go, Ruby, C#. Dal 2018 sono supportati anche runtime personalizzati;
- **AWS DynamoDB**: servizio di database NoSQL che supporta coppie chiave-valore e document data. Non è necessario definire uno schema per gli oggetti salvati ad eccezione di una chiave primaria. DynamoDB supporta operazioni di Put, Get, Update e Delete. Tra i linguaggi per cui sono offerti dei binding ci sono: Java, Javascript, Go, PHP, Perl, Python, Ruby, Haskell, Erlang, Rust;
- **API Gateway**: è un servizio che permette di creare, pubblicare, rendere sicure, monitorare API di tipo RESTful e WebSocket. Permette di gestire migliaia di chiamate concorrenti, supporto per CORS, autorizzazione e controllo dell'accesso;
- **AWS Simple Storage Service (S3)**: offre un servizio di storage in grado di memorizzare qualsiasi tipo di oggetto, da applicazioni web a backups.

Figura 4.4: Logo di AWS: Lambda, DynamoDB, API Gateway, S3

Ajv

*"**Ajv** JSON Schema Validator"* è uno strumento di validazione. Permette di implementare complesse logiche di validazione di dati JSON in modo dichiarativo. In questo caso un semplice esempio è più significativo di una lunga descrizione:

```
const Ajv = require("ajv")
const ajv = new Ajv()

const schema = {
  type: "object",
  properties: {
    foo: {type: "integer"},
    bar: {type: "string"}
  },
  required: ["foo"],
  additionalProperties: false
}

const data = {foo: 1, bar: "abc"}
```

```
const valid = ajv.validate(schema, data)
if (!valid) console.log(ajv.errors)
```

WebSocket

WebSocket è un protocollo che permette la comunicazione full-duplex utilizzando una singola connessione TCP. Nel nostro progetto questa tecnologia viene utilizzata per la gestione di attività real-time, in particolare per gestire gli utenti connessi contemporaneamente all'applicazione. Il Framework Serverless permette di utilizzare connessioni WebSocket grazie a **API Gateway** che espone un evento `websocket`. Sono disponibili 4 tipi di route, legate al ciclo di vita della connessione WebSocket: `$connect`, `$disconnect`, `$default` e custom. Un esempio di `serverless.yml` è il seguente:

```
service: serverless-ws-test

provider: # ...

functions:
  connectionHandler:
    handler: handler.connectionHandler
    events:
      - websocket:
          route: $connect
      - websocket:
          route: $disconnect
  defaultHandler:
    handler: handler.defaultHandler
    events:
      - websocket: $default
  customHandler:
    handler: handler.customHandler
    events:
      - websocket:
          route: foo
```

Jest

Jest è un Framework utilizzato per il testing di applicazioni JavaScript. Verrà utilizzato per effettuare Unit Testing delle funzioni handler Serverless e di tutte le funzioni ausiliarie utilizzate. In seguito è riportato un semplice esempio che ne illustra il funzionamento:

```
function add(x, y) {
  return x + y;
}

test('2 + 2 = 4', () => {
  expect(add(2, 2)).toBe(4);
});
```

```
// Questo test restituisce in output:
//     PASS  ./sum.test.js
//     adds 1 + 2 to equal 3 (5ms)
```

Jest permette di eseguire test in parallelo per migliorare la velocità di testing. Offre un servizio di generazione del code coverage. Permette di definire dei mock per sovrascrivere il comportamento di funzioni e oggetti fuori dalla nostra applicazione.

4.1.3 Frontend

Angular

Angular è un Framework frontend open-source basato su TypeScript e sviluppato da un team di Google e da una community di individui e corporazioni. Angular permette di sviluppare applicazioni web scalabili mediante componenti definiti in modo dichiarativo. Angular include inoltre un vastissimo insieme di librerie e strumenti a supporto dello sviluppo.

Figura 4.5: Logo di Angular

I componenti sono l'elemento di sviluppo principale utilizzato per comporre un'applicazione. Sono formati da:

- **Classe Typescript**: che definisce il comportamento del componente;
- **Template HTML**: che definisce il contenuto che verrà renderizzato sulla pagina web;
- **Stile CSS**: che definisce l'aspetto del template HTML;
- Un insieme di **metadati** (e.g. un selettore che permette al componente di essere utilizzato come tag HTML in altre parti dell'applicazione).

```
// File: hello-world.component.ts
import { Component } from '@angular/core';

@Component ({
  selector: 'hello-world',
  templateUrl: './hello-world.component.html'
})
export class HelloWorldBindingsComponent {
  fontColor = 'blue';
  sayHelloId = 1;
  canClick = false;
  message = 'Hello, World';

  sayMessage() {
    alert(this.message);
  }
}
```

```
<!-- File: hello-world.component.html -->
<button
  type="button"
  [disabled]="canClick"
  (click)="sayMessage()">
  Trigger alert message
</button>
<p
  [id]="sayHelloId"
  [style.color]="fontColor">
  You can set my color in the component!
</p>
<p>My color is {{ fontColor }}</p>
```

Nell'esempio sopra riportato, nella parte di template HTML:

- Vengono utilizzati i **"property bindings"** per collegare attributi HTML (definiti nel file `hello-world.component.html`) alla logica di presentazione (definita nel file `hello-world.component.ts`) utilizzando le parentesi quadre `[...]`;
- Viene dichiarato un **event listener** `click` collegandolo alla funzione TypeScript utilizzando le parentesi tonde `(...)`;
- Viene **interpolato il valore della variabile** `fontColor` utilizzando le parentesi graffe `{...}`.

In Angular è possibile utilizzare delle direttive per aggiungere funzionalità al template. Ne sono un esempio le direttive:

- `*ngIf`: utilizzata per effettuare il rendering condizionale di un elemento HTML;
- `*ngFor`: utilizzata per effettuare il rendering di più elementi presenti in un array.

Un design pattern molto utilizzato in Angular è il **Dependency Injection**. Permette di dichiarare le dipendenze di una classe senza doversi preoccupare del'istanziazione delle stesse. A differenza dei componenti, le dipendenze vengono definite come servizi. Vediamo un esempio semplicissimo per capire meglio:

```
// File: hello.service.ts
import { Injectable } from '@angular/core';

@Injectable({providedIn: 'root'})
export class HelloService {
  sayHello() {
    console.log("Hello!");
  }
}
```

```
// File: hello.component.ts
import { Component } from '@angular/core';
import { HelloService } from './hello.service.ts';
```

```
@Component({
selector: 'hello',
templateUrl: './hello.component.html'
})
export class HelloComponent  {
  constructor(private helloService: HelloService) { } // Injection

  hello() {
    this.helloService.sayHello();
  }
}
```

La CLI di angular offre molti comandi di supporto allo sviluppo, tra cui:

- `ng build`: compila l'applicazione Angular;
- `ng serve`: compila l'applicazione e esegue un server locale per testarla;
- `ng generate`: utilizzata per generare componenti, servizi, classi, interfacce e molto altro.

Infine per facilitare lo sviluppo e avere un'applicazione finale che ha una design uniforme e moderno, si utilizza **Angular Material** come libreria di componenti UI. Tutti seguono strettamente le specifiche imposte da **Google Material Design**, che definisce linee guide per quanto riguarda: layout, navigazione, colori, tipografia, suoni, icone, forme, movimento e interazione. Grazie a questo strumento l'applicazione finale avrà un "look & feel" molto simile a tutte quelle della suite di Google e di Android, proponendo una user experience familiare. I componenti più interessanti per il progetto sono:

- **Autocomplete**: casella di testo che suggerisce all'utente un insieme di opzioni coerenti con quello che si è iniziato a scrivere;
- **Button**: pulsanti interattivi;
- **Card**: container stilizzato;
- **Datepicker**: calendario per selezionare date;
- **Dialog**: popup che può contenere qualsiasi altro componente;
- **Input**: campi per diversi tipi di input;
- **Select**: permette di selezionare un'opzione da una lista dropdown;
- **Form Field**: per raggruppare più Input;
- **Progress Spinner**: indicatore circolare per indicare il progresso o il processo di un'attività;
- **Tooltip**: mostra una serie di informazioni di un componente on hover.

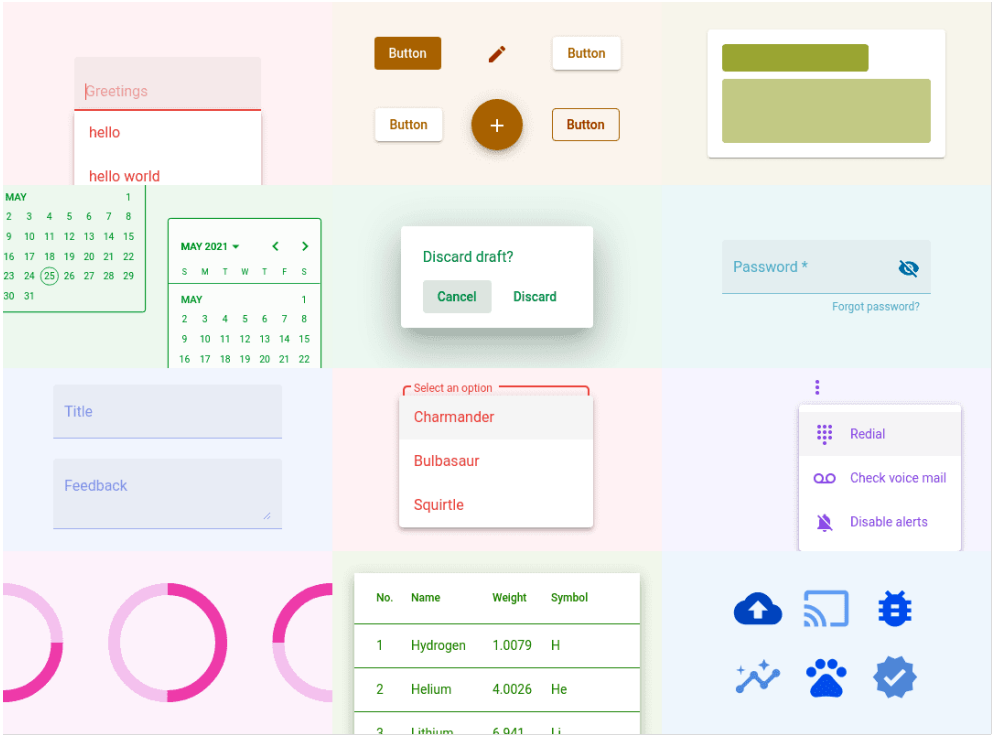


Figura 4.6: Showcase di componenti UI di Angular Material

Fabric.js

Per soddisfare le esigenze di avere un Wall su cui poter fare zoom in / out, panning e dragging di note, tra le tecnologie ricercate, la più adeguata si è dimostrata l' HTML `<canvas>`. Tuttavia l'interazione diretta con le API JavaScript del canvas è di livello molto basso. Vanno bene per disegnare qualche forma geometrica semplice ma appena le cose si fanno un'attimo più complicate o si introduce dell'interattività il tutto diventa estremamente complesso da gestire. **Fabric.js** è una libreria JavaScript che offre delle API per interagire più semplicemente con il canvas. Vediamo un esempio molto semplice per confrontare le due sintassi. Diciamo che vogliamo disegnare questo quadratino:

Figura 4.7: Quadrato rosso, lato di 20px, ruotato di 45 gradi

Con le **API del canvas** si ha:

```
const canvasEl = document.getElementById('c');
const ctx = canvasEl.getContext('2d');
ctx.fillStyle = 'red';
ctx.rotate(Math.PI / 180 * 45);
ctx.fillRect(-10, -10, 20, 20);
```

Con **Fabric.js**:

```
const canvas = new fabric.Canvas('c');
const rect = new fabric.Rect({fill: 'red',width: 20,height: 20,angle: 45});
canvas.add(rect);
```

Ma non è solo per la semplicità che si è scelto di usare Fabric. Infatti la libreria offre molte altre funzionalità, tra cui:

- **Interattività**: qualsiasi oggetto Fabric aggiunto al canvas, di default può essere selezionato, spostato, ruotato o raggruppato con altri oggetti mediante mouse (o il touch se disponibile). Questo semplifica molto l'implementazione del trascinamento di note nel Wall e nel Backlog;
- **Forme**: Fabric offre di default le seguenti forme geometriche: cerchio, ellisse, linea, poligono, polyline, rettangolo, triangolo;
- **Immagini e Filtri**: Fabric permette di inserire e manipolare immagini con la stessa facilità delle forme geometriche;
- **Paths**: funzionano in modo simile ai path SVG, permettendo di creare forme non triviali mediante comandi di movimento;
- **Animazioni**: ogni oggetto Fabric ha un metodo `animate` che permette di costruire animazioni più o meno complesse. Ad esempio aggiungere un'animazione al rettangolo di prima, spostandolo verso destra con un easeout di tipo "bounce":

  ```
  rect.animate('left', 500, {
    onChange: canvas.renderAll.bind(canvas),
    duration: 1000,
    easing: fabric.util.ease.easeOutBounce
  });
  ```

- **Testo**: con funzionalità che il testo del canvas non ha di default, come supporto multiline, allineamento, background, decorazioni, line height, spaziatura dei caratteri e editing direttamente su canvas;
- **Gruppi**: permette di raggruppare e gestire oggetti come se fossero uno solo;
- **Free Drawing**: semplicemente impostando la proprietà `isDrawingMode` del canvas Fabric, ogni click successivo verrà interpretato come pennellata e trasformato in un'istanza di `fabric.Path`.

WebSocket

Come visto nel backend, il WebSocket è utilizzato per la gestione della parte real-time dell'applicazione. Lato frontend si utilizzano le **API standard dei browser**. Un esempio di utilizzo è il seguente:

```
let socket = new WebSocket("wss://example.websocket.com/");

socket.onopen = function(e) {
  socket.send("Hello from Client!");
};

socket.onmessage = function(event) {
```

```
  console.log(`[message] Data received from server: ${event.data}`);
};

socket.onclose = function(event) {
  if (event.wasClean) {
    console.log('[close] Connection closed cleanly');
  } else {
    console.log('[close] Connection died');
  }
};

socket.onerror = function(error) {
  console.log(`[error] ${error.message}`);
};
```

4.2 Progettazione Backend

4.2.1 DynamoDB

Per quanto riguarda la persistenza dei dati, sarà necessario avere le seguenti tabelle:

- **Note Wall**: per memorizzare le note del Wall. Gli oggetti salvati in questa tabella sono definiti dalla seguente interfaccia:

  ```
  interface Note {
    uuid: string,
    title: string,
    text: string,
    project: string,
    type: string,
    dateInitial: string,
    dateCurrent: string,
    relativePosition: number[],
    completed: boolean
    isArchived?: boolean,
  }
  ```

  In questo caso il campo `isArchived` è pensato per essere usato all'archiviazione di un progetto, per poter permettere l'eventuale ripristino futuro delle note se il progetto viene riaperto.

- **Progetti**: per memorizzare i progetti. Gli oggetti salvati in questa tabella sono definiti dalla seguente interfaccia:

  ```
  interface Project{
    projectId: string,
    projectName: string,
    customer: string,
    pm: string,
    jiraLink: string,
    dateInitial: string,
  ```

```
  completed: boolean
}
```

- **Note Backlog**: per memorizzare le note del Backlog. Gli oggetti salvati in questa tabella sono definiti dalla seguente interfaccia:

```
interface BacklogNote {
  uuid: string,
  text: string,
  type: string,
  position: number[],
  idPM: string,
  isArchived?: boolean
}
```

  In questo caso il campo `isArchived` è pensato per effettuare una "soft delete", in caso si decidesse di costruire uno storico delle azioni effettuate.

- **PM**: per memorizzare i project manager del Backlog. Gli oggetti salvati in questa tabella sono definiti dalla seguente interfaccia:

```
interface BacklogPM {
  uuid: string,
  name: string,
  isArchived?: boolean
}
```

  In questo caso il campo `isArchived` è pensato per effettuare una "soft delete", in caso si decidesse di costruire uno storico delle azioni effettuate.

- **WebSocket User**: per permettere al WebSocket di gestire gli utenti connessi contemporaneamente. Gli oggetti salvati in questa tabella sono definiti dalla seguente interfaccia:

```
interface WebsocketUser {
  connectionId: string,
  name: string,
  urlPicture: string
}
```

4.2.2 Funzioni Lambda

Note Wall

Per la gestione delle note del Wall si prevede la necessità delle seguenti funzioni:

- `createNote`: per aggiungere una nuova nota;
- `deleteNote`: per eliminare una nota;
- `getNotes`: per recuperare dal database tutte le note dei progetti ancora attivi;
- `updateNote`: per aggiornare i campi di una nota quando vengono modificati.

Progetti

Per la gestione dei progetti si prevede la necessità delle seguenti funzioni:

- `createProject`: per aggiungere un nuovo progetto;
- `deleteProject`: per eliminare un progetto e tutte le note a esso associate;
- `getProjects`: per recuperare dal database tutti i progetti ancora attivi;
- `updateProject`: per aggiornare i campi di un progetto quando vengono modificati;
- `archiveProject`: per archiviare un progetto e tutte le note a esso associate;
- `restoreProject`: per riaprire un progetto e tutte le note a esso associate.

Note Backlog

Per la gestione delle note del Backlog si prevede la necessità delle seguenti funzioni:

- `createBacklogNote`: per aggiungere una nuova nota;
- `deleteBacklogNote`: per eliminare una nota;
- `getBacklogNotes`: per recuperare dal database tutte le note dei PM esistenti;
- `updateBacklogNote`: per aggiornare i campi di una nota quando vengono modificati.

Project Manager

Per la gestione dei PM del Backlog si prevede la necessità delle seguenti funzioni:

- `createBacklogPM`: per aggiungere un nuovo project manager;
- `deleteBacklogPM`: per rimuovere un project manager e tutte le note a esso associate;
- `getBacklogPM`: per recuperare dal database tutti i project manager;
- `updateBacklogPM`: per aggiornare i campi di un project manager quando vengono modificati.

WebSocket User

Per la gestione degli utenti connessi al WebSocket si prevede la necessità di una funzione che gestisce le seguenti route:

- `connect`: per aggiungere un nuovo utente alla lista degli utenti connessi e notificare a tutti gli altri le sue informazioni condivisibili (principalmente nome e immagine profilo);
- `disconnect`: per rimuovere un utente dalla lista degli utenti connessi e notificare la sua disconnessione a tutti gli altri.

4.3 Progettazione Frontend

In questa sezione si discuterà della progettazione della parte di frontend, suddivisa per ciascuna pagina web che dovrà essere sviluppata.

4.3.1 Wall

Per quanto riguarda la pagina del Wall si prevede la necessità di avere i seguenti componenti Angular:

- `WallComponent`: per la gestione di tutti i componenti e servizi che verranno illustrati a seguire;
- `DialogAddPostit`: dialog che permette di aggiungere una nuova nota al Wall;
- `DialogModifyPostit`: dialog che permette di modificare una nota esistente del Wall;
- `DialogAddProject`: dialog che permette di aggiungere un nuovo progetto;
- `DialogModifyProject`: dialog che permette di modificare un progetto esistente.

Saranno necessari i seguenti servizi:

- `PostitService`: per gestire la lista di note lato client e comunicare con il backend;
- `ProjectService`: per gestire la lista di progetti lato client e comunicare con il backend;
- `CanvasService`: per la di una singola istanza del canvas Fabric, per lo zoom in / out, per il panning;
- `DrawerGridService`: per gestire il disegno le settimane e i progetti esistenti in forma di tabella;
- `DrawerPostitService`: per gestire il disegno dei post-it sul Wall;
- `WeekService`: per gestire le settimane presenti sul Wall;
- `DateService`: per facilitare e uniformare la gestione delle date.

4.3.2 Backlog

Per quanto riguarda la pagina del Backlog, molto simile se non più semplice di quella del Wall, si prevede la necessità di avere i seguenti componenti Angular:

- `BacklogComponent`: per la gestione di tutti i componenti e servizi che verranno illustrati a seguire;
- `DialogAddNote`: dialog che permette di aggiungere una nuova nota al Backlog;
- `DialogModifyNote`: dialog che permette di modificare una nota esistente del Backlog;
- `DialogAddPM`: dialog che permette di aggiungere un nuovo project manager;
- `DialogModifyPM`: dialog che permette di modificare un project manager esistente.

Saranno necessari i seguenti servizi:

- `NoteService`: per gestire la lista di note lato client e comunicare con il backend;
- `PmService`: per gestire la lista di project manager lato client e comunicare con il backend;
- `CanvasService`: per la di una singola istanza del canvas Fabric, per lo zoom in / out, per il panning;
- `DrawerGridService`: per gestire il disegno delle colonne e dei project manager esistenti in forma di tabella;
- `DrawerNoteService`: per gestire il disegno delle note sul Backlog.

4.3.3 Utenti Connessi

Per la visualizzazione degli utenti connessi invece, si è pensato di aggiungere una lista dropdown a partire dalle informazioni dell'utente mostrate nell'header. L'elemento più rilevante per questa parte è un servizio `WebsocketUserService` che si occuperà della gestione lato client degli utenti connessi e di comunicare aggiornamenti in real time mediante WebSocket.

Capitolo 5

Codifica

In questo capitolo verranno discussi alcuni dettagli dell'implementazione del progetto. Vista la forte somiglianza tra Wall e Backlog in questa parte verrà principalmente esposto il primo, visto che più articolato e complesso. L'ordine di esposizione coincide con quello cronologico delle attività. Generalmente venivano prima implementate le API lato backend, tipicamente più rapide da sviluppare, per poi concentrarsi sulla parte Web del frontend che ha richiesto maggiori risorse.

5.1 Wall

5.1.1 API

Prima di tutto si è definito un template Serverless che dichiara e configura tutte le risorse necessarie per poter utilizzare funzioni Lambda e tabelle DynamoDB. In particolare:

- Si definisce il provider utilizzato `aws`, la regione `eu-central-1` e il runtime necessario `nodejs14.x`;
- Si definisce le variabili d'ambiente che possono essere utilizzate sia nel template Serverless che nel codice;
- Si definisce un authorizer per la gestione degli accessi;
- Si dichiarano gli handler delle funzioni Lambda, i loro endpoint e i loro verbi HTTP;
- Si definiscono le tabelle DynamoDB come risorse;
- Si dichiarano i plugin utilizzati e delle impostazioni custom per lo sviluppo in locale.

Si riporta un estratto del file `serverless.yml` prodotto (ripulito da informazioni sensibili):

```
service: agile-wall-api
frameworkVersion: '3'
useDotenv: true
```

```yaml
provider:
  name: aws
  region: eu-central-1
  runtime: nodejs14.x
  stage: ${opt:stage, 'dev'}
  deploymentBucket:
    name: ${self:service}-deploy-${self:provider.stage}
  environment:
    DYNAMO_NOTES_TABLE_NAME: ${self:service}-notes-${self:provider.stage}
    DYNAMO_PROJECTS_TABLE_NAME: ${self:service}-projects-${self:provider.stage}
    # ...
  httpApi:
    cors: true
    authorizers:
      agile-wall-cognito-authorizer:
        identitySource: $request.header.Authorization
        issuerUrl: # ...
        audience:
          # ...
  iamRoleStatements:
    - Effect: Allow
      Action:
        - dynamodb:Query
        - dynamodb:Scan
        - dynamodb:GetItem
        - dynamodb:PutItem
        - dynamodb:UpdateItem
        - dynamodb:DeleteItem
      Resource: "*"

functions:
  getNotes:
    handler: src/notes/handler/getNotes.getNotes
    events:
      - httpApi:
          path: /note
          method: get
          authorizer:
            name: agile-wall-cognito-authorizer
  createNote:
    handler: src/notes/handler/createNote.createNote
    events:
      - httpApi:
          path: /note
          method: post
          authorizer:
            name: agile-wall-cognito-authorizer
  updateNote:
    handler: src/notes/handler/updateNote.updateNote
    events:
```

```
      - httpApi:
          path: /note
          method: put
          authorizer:
            name: agile-wall-cognito-authorizer
  deleteNote:
    handler: src/notes/handler/deleteNote.deleteNote
    events:
      - httpApi:
          path: /note
          method: delete
          authorizer:
            name: agile-wall-cognito-authorizer
  # ... Funzioni per gestire i Progetti

resources:
  Resources:
    notesTable:
      Type: AWS::DynamoDB::Table
      Properties:
        TableName: ${self:provider.environment.DYNAMO_NOTES_TABLE_NAME}
        AttributeDefinitions:
          - AttributeName: uuid
            AttributeType: S
        KeySchema:
          - AttributeName: uuid
            KeyType: HASH
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1
    # ... Tabelle dei progetti

custom:
  dynamodb:
    stages:
      - dev
    start:
      port: 8000
      inMemory: true
      heapInitial: 200m
      heapMax: 1g
      migrate: true
      seed: true
      convertEmptyValues: true
  # ...

plugins:
  - serverless-plugin-typescript
  - serverless-dynamodb-local
  - serverless-offline
```

Per quanto riguarda gli handler delle funzioni Lambda, è stato implementato quanto dichiarato in Progettazione Backend. In particolare la gestione delle chiamate API contiene:

- La verifica se l'utente è PM, e quindi autorizzato a effettuare l'operazione richiesta, mediante un JWT inviato con la richiesta;
- La validazione del `body` della richiesta;
- La logica che permette di accedere alle tabelle DynamoDB in lettura o scrittura in base alla richiesta;
- Il ritorno di una risposta al client.

Alcune attenzioni che sono state poste durante lo sviluppo delle API sono:

- La generazione server-side di uno UUID per ciascun oggetto esistente;
- L'utilizzo di tutte le interfacce necessarie per avere una tipizzazione forte;
- L'utilizzo di schemi per la validazione degli oggetti;
- La separazione della logica in un numero adeguato di funzioni, per permettere all'handler di essere il più chiaro possibile.

5.1.2 Web

Anche per quanto riguarda questa parte si è implementato quanto dichiarato in Progettazione Frontend. Analizziamo alcuni dettagli della codifica di componenti e servizi.

`WallComponent`

Questo componente effettua il rendering di: header, canvas, un indicatore del totale delle Milestone in ritardo, un bottone per creare un nuovo progetto e un spinner per segnalare lo stato di comunicazione con il backend.

Questo componente è responsabile di:

- Gestire l'inizializzazione del canvas Fabric, del disegno iniziale della tabella e dei post-it esistenti;
- Ridimensionare il canvas in modo che occupi sempre pienamente la finestra del browser;
- Gestire i componenti sopra citati di cui effettua il rendering.

`DialogAddPostit`

Questo componente effettua il rendering di un dialog popup con un form contenente: input per il titolo della nota, input per il testo della nota, un selettore dropdown per il tipo della nota, un pulsante di creazione della nota. Progetto, settimana di creazione della nota e posizione relativa vengono passati al componente.

Questo componente è responsabile di:

- Aggiungere un nuovo post-it al Wall con le informazioni fornite dall'utente nel form;

- Assicurarsi che il form contenga tutte le informazioni necessarie per creare correttamente una nuova nota e in caso avvisare l'utente di eventuali input mancanti.

`DialogModifyPostit`

Questo componente effettua il rendering di un dialog popup con un form contenente: input per il titolo della nota, input per il testo della nota, informazioni sul progetto a cui la nota appartiene, informazioni sul tipo della nota, informazioni sulla settimana iniziale e corrente della nota, un pulsante per modificare la nota, un pulsante per marcare la nota come completata se ancora attiva, un pulsante per riaprire una nota se completata, un pulsante per eliminare la nota.

Questo componente è responsabile di:

- Permettere la modifica di titolo e testo di una nota;
- Permettere la cancellazione di una nota;
- Permettere di marcare una nota come completata o di riaprirla se completata.

`DialogAddProject`

Questo componente effettua il rendering di un dialog popup con un form contenente: input per il nome del progetto, input per il cliente associato, input per il project manager di riferimento, input per il link Jira del progetto, un pulsante di creazione del progetto.

Questo componente è responsabile di:

- Aggiungere un nuovo progetto con le informazioni fornite dall'utente nel form;
- Assicurarsi che il form contenga tutte le informazioni necessarie per creare correttamente un nuovo progetto e in caso avvisare l'utente di eventuali input mancanti.

`DialogModifyProject`

Questo componente effettua il rendering di un dialog popup con un form contenente: input per modificare il nome del progetto, input per modificare il cliente associato, input per modificare il project manager di riferimento, input per modificare il link Jira del progetto, un pulsante di modifica del progetto, un pulsante di archiviazione del progetto, un pulsante di cancellazione del progetto.

Questo componente è responsabile di:

- Permettere la modifica di un progetto;
- Permettere l'archiviazione di un progetto;
- Permettere la cancellazione di un progetto.

`PostitService`

Questo servizio è responsabile di:

- Mantenere una lista lato client di tutte le note esistenti sul Wall;

- Esporre dei metodi di tipo CRUD per le note presenti lato frontend;
- Esporre dei metodi di tipo CRUD per le note presenti lato backend.

`ProjectService`

Questo servizio è responsabile di:

- Mantenere una lista lato client di tutti i progetti;
- Esporre dei metodi di tipo CRUD per i progetti presenti lato frontend;
- Esporre dei metodi di tipo CRUD per i progetti presenti lato backend;
- Mantenere i progetti in ordine mediante un semplice algoritmo di sorting;
- Esporre dei metodi di utilità legati ai progetti per altre classi, come: restituire l'index di un progetto dato il suo ID, restituire un progetto dato il suo index.

`CanvasService`

Questo servizio è responsabile di:

- Inizializzare la singola istanza del canvas Fabric;
- Gestire la funzionalità di zoom in / out mediante eventi di mouse scroll;
- Gestire la funzionalità di spostamento mediante pan.

`DrawerGridService`

Questo servizio è responsabile di:

- Gestire parametri legati alla tabella del Wall come: numero di settimane visualizzate, altezza e larghezza delle celle, altezza e larghezza degli header;
- Esporre un metodo per il rendering della tabella che: disegni lo sfondo, disegni le linee, disegni le informazioni delle settimane, disegni le informazioni dei progetti, evidenzi la settimana corrente;
- In caso l'utente sia un PM, permettergli di effettuare un doppio click per inserire una nota nella posizione scelta, determinando in questo modo progetto e settimana di creazione;
- In caso l'utente sia un PM, permettergli di effettuare un doppio click per modificare un progetto;
- Disegnare dei "smart header" nel caso gli header di settimane e progetto non fossero visibili a causa dello zoom e/o della posizione della viewport;
- Esponga dei getter per i parametri legati alla tabella.

`DrawerPostitService`

Questo servizio è responsabile di:

- Esporre un metodo per il rendering dei post-it un post-it;
- In caso l'utente sia un PM, permettergli di modificare un post-it aprendo il rispettivo dialog mediante un doppio click sul post-it stesso;
- Rimuovere un post-it dal Wall;
- Integrare nel rendering dei post-it una serie di comportamenti "smart" come: trascinamento solo nei limiti della riga di progetto corretta, aggiornamento automatico della data corrente in base alla posizione, visualizzazione del cambio di stato del post-it.

`WeekService`

Questo servizio è responsabile di:

- Definire il numero di settimane da visualizzare sul Wall;
- Esporre dei metodi per la gestione delle settimane come: ritorno della settimana corrente, ritorno di una settimana dato il suo index e viceversa.

`DateService`

Questo servizio che espone dei metodi statici per:

- Ritornare la data corrente;
- Ritornare la data del lunedì della settimana a partire da una particolare data;
- Ritornare la data della settimana successiva / precedente a partire da una particolare data;
- Ritornare una stringa formattata a partire da una particolare data.

Rispetto alla progettazione tuttavia, ogni volta che viene effettuata una cancellazione si è implementato un dialog di conferma. Se i dati da eliminare sono molti, come per un progetto (che implica la cancellazione di tutte le sue note), il dialog impone di inserire una particolare stringa in input (e.g. il nome del progetto) prima di poter confermare l'eliminazione.

Infine è stato fornito un esempio di design da cui si è ricavata una palette di colori per rendere il progetto più uniforme all'interno della suite di strumenti utilizzati in azienda.

5.2 Backlog

Il Backlog risulta una versione semplificata del Wall. La sua implementazione segue quella definita nella Progettazione Frontend. Si invita a confrontarla con il Wall per avere un'idea di come sia stata codificata.

5.3 WebSocket

5.3.1 API

Anche per la parte di WebSocket la codifica ha seguito quanto dichiarato nella Progettazione.

Si riporta la dichiarazione della funzione Lambda definita in Serverless, diversa da tutte le altre API di tipo HTTP

```
  websocketConnect:
    handler: src/websocket/handler/websocket.handler
    events:
      - websocket:
          route: $connect
          authorizer:
            name: authorizerWS
            identitySource:
              - 'route.request.querystring.Auth'
      - websocket:
          route: newConnection
      - websocket:
          route: $disconnect
      - websocket:
          route: $default

  authorizerWS:
    handler: src/websocket/handler/auth.handler
```

Si può notare che è stato necessario definire un authorizer ad-hoc diverso rispetto a quello per le altre API, dato che non supportava il protocollo WebSocket. L'authorizer è installato solo sulla route `$connect`, che è il punto d'accesso.

Si riporta in seguito la parte essenziale dell'handler del WebSocket.

```
exports.handler = async function (event: APIGatewayProxyEvent)
  : Promise<APIGatewayProxyResult> {
  // ...
  switch (routeKey) {
    // ...
    case 'newConnection':
      const user: WebsocketUser = {
        connectionId: connectionId!,
        name: JSON.parse(body!).data.name,
        urlPicture: JSON.parse(body!).data.urlPicture
      }
      await createWebsocketUserDB(user);
      await updateAllCurrentlyOnlineUsers();
      break;

    case '$disconnect':
      await deleteWebsocketUserDB(connectionId!);
      await updateAllCurrentlyOnlineUsers();
      break;
```

```
    // ...
  }
  // ...
}
```

Si può osservare che sono definite le route:

- `newConnection`: per la memorizzazione di un nuovo utente collegato e la notifica a tutti gli altri utenti online della nuova connessione;
- `$disconnect`: per rimuovere un utente dalla lista degli utenti connessi e notificarlo agli altri;

5.3.2 Web

Lato frontend il WebSocket è stato implementato in un servizio. Si riporta la parte saliente della codifica.

```
this.socket = new WebSocket(`${environment.endpoint_websocket}?Auth=${token}`);
// ...
this.socket.onopen = () => {
  const user = {
    name: name,
    urlPicture: picture
  }
  this.socket.send(JSON.stringify({ action: 'newConnection', data: user }));
};

this.socket.onmessage = (event) => {
  const data = JSON.parse(event.data)
  if (data.action === "newConnection") {
    this.onlineUsers = data.users;
  }
};
// ...
```

Si osserva che dopo l'apertura della connessione WebSocket, i dati dell'utente vengono inviati ed il backend provvede a notificare tutti gli altri utenti. La notifica avviene mediante `this.socket.onmessage` che aggiorna la lista di utenti attualmente online.

Capitolo 6

Testing

6.1 Backend

Lo Unit Testing sulla parte di backend è stato effettuato con **Jest**. L'idea di base è quella di isolare ciascuna parte del codice, che in questo caso è già comodamente divisa in funzioni (che gestiscono le chiamate API), e dimostrarne la correttezza. Un test di unità è come un contratto per confermare che in una determinata situazione, il codice deve comportarsi in un certo modo. Si è cercato quindi di coprire il maggior numero possibile di casi. Chiaramente non è possibile prevedere tutto, quindi i test non garantiscono l'assoluta assenza di errori. Sono però estremamente utili:

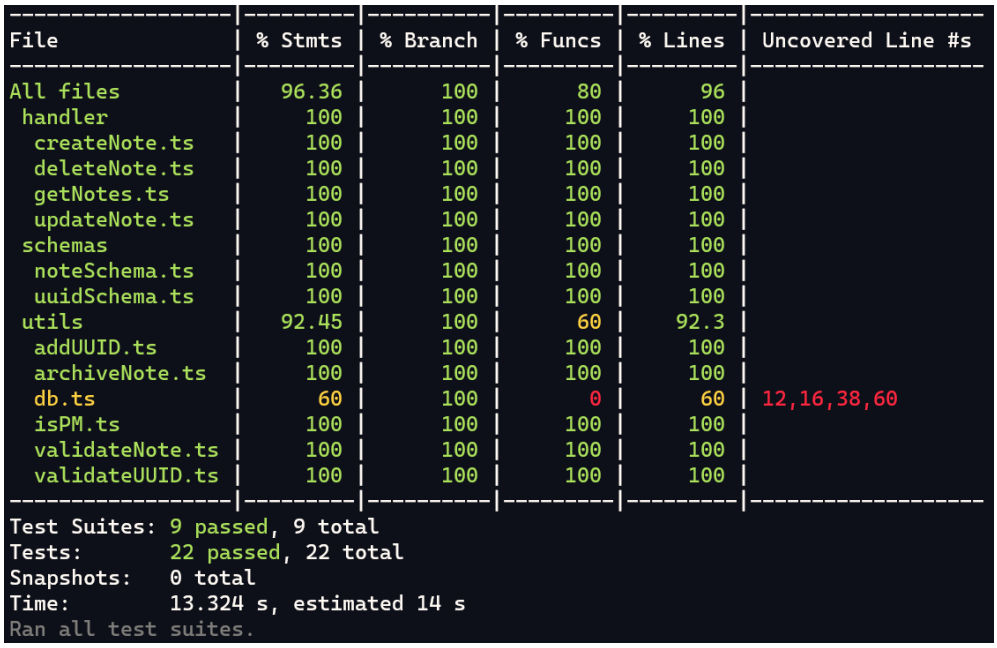
- Nell'immediato: per catturare eventuali errori sul nascere;
- Nel lungo periodo: per assicurare che future evoluzioni del codice non rompano parti esistenti e funzionanti (regression testing).

Jest in particolare permette di spiare e creare dei **mock** di funzioni, utile sia per avere un miglior controllo su funzioni del codebase già testate, sia per testare parti di codice che utilizzano librerie esterne, sulle quali non si ha controllo. Nel nostro caso, per accedere a tali funzionalità, si sono utilizzate le funzioni Jest: `jest.spyOn(object, methodName)`, `mockFn.mockReturnValue(value)` e `mockFn.mockImplementation(fn)`.

Per verificare la correttezza invece, si sono utilizzate le funzioni Jest:

- `.toEqual(value)`: controlla ricorsivamente tutte le proprietà dell'istanza dell'oggetto;
- `.toHaveBeenCalledTimes(number)`: verifica quante volte un mock di una funzione è stato chiamato;
- `.toHaveBeenCalledWith(arg1, arg2, ...)`: verifica che un mock di una funzione sia stato chiamato con specifici parametri. I parametri sono verificati mediante lo stesso algoritmo utilizzato da `.toEqual`.

Infine Jest permette anche di avere delle statistiche sul **code coverage**, oltre a quelle sui test. Si riporta l'output del comando che esegue i test:



```
------------------|---------|----------|---------|---------|-------------------
File              | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------------|---------|----------|---------|---------|-------------------
All files         |   96.36 |      100 |      80 |      96 |
 handler          |     100 |      100 |     100 |     100 |
  createNote.ts   |     100 |      100 |     100 |     100 |
  deleteNote.ts   |     100 |      100 |     100 |     100 |
  getNotes.ts     |     100 |      100 |     100 |     100 |
  updateNote.ts   |     100 |      100 |     100 |     100 |
 schemas          |     100 |      100 |     100 |     100 |
  noteSchema.ts   |     100 |      100 |     100 |     100 |
  uuidSchema.ts   |     100 |      100 |     100 |     100 |
 utils            |   92.45 |      100 |      60 |    92.3 |
  addUUID.ts      |     100 |      100 |     100 |     100 |
  archiveNote.ts  |     100 |      100 |     100 |     100 |
  db.ts           |      60 |      100 |       0 |      60 | 12,16,38,60
  isPM.ts         |     100 |      100 |     100 |     100 |
  validateNote.ts |     100 |      100 |     100 |     100 |
  validateUUID.ts |     100 |      100 |     100 |     100 |
------------------|---------|----------|---------|---------|-------------------
Test Suites: 9 passed, 9 total
Tests:       22 passed, 22 total
Snapshots:   0 total
Time:        13.324 s, estimated 14 s
Ran all test suites.
```

Figura 6.1: Output dei test Jest

Si può notare che l'unico modulo che non è stato coperto dai test al 100% è `db.ts` che contiene le funzioni per la persistenza in DynamoDB. Questo perché le tutte le funzioni definite in tale modulo non sono altro che dei wrapper attorno alla funzioni di DynamoDB `dynamo.scan`, `dynamo.put`, `dynamo.delete`. La correttezza di queste funzionalità è stata testata nel modo esposto in seguito.

Visto che le API venivano tipicamente sviluppate prima di avere un frontend, per verificarne il corretto funzionamento si è utilizzato **Postman**. Attraverso tale piattaforma è possibile creare collezioni di chiamate HTTP e WebSocket, specificando parametri come Headers e Body e catturando poi la risposta del backend. In questo modo è stato possibile accertarsi che tutte le parti del backend funzionassero correttamente.

6.2 Frontend

Per quanto riguarda il frontend, non sono stati effettuati test d'unità. Questo per i seguenti motivi:

- Canvas: la maggior parte dell'applicazione è un insieme di oggetti Fabric renderizzati su canvas. Effettuare Unit Testing in questo ambiente è una pratica difficile e poco documentata;
- Risorse: scrivere dei test per questa parte non sarebbe stato impossibile ma avrebbe sicuramente richiesto un consumo di tempo non indifferente che avrebbe compromesso lo sviluppo dell'applicazione;
- Processi Aziendali: tipicamente i processi aziendali interni impongono un serio sviluppo dei test della parte di backend ma per il frontend non ci sono linee guida particolari.

Questa non è stata una scelta diretta del tirocinante ma è stata una discussa decisione collettiva.

6.3 Resoconto e Riflessioni

In un mondo ideale si ha tempo in abbondanza per implementare tutte le feature di un'applicazione e contemporaneamente scrivere esaustivi test per tutte le parti. Ma in un contesto come quello dello stage, in cui il tempo è una risorsa scarsa, la questione è molto diversa. Bisogna infatti scegliere a cosa dare priorità. Per quanto riguarda i test, la decisione è stata di andare più in profondità, che però ha significato dover coprire meno moduli. Infatti, come sopra riportato, si sono eseguiti dei test dettagliati della parte di gestione delle note del Wall ma non sono stati coperti né i progetti, né la parte di Backlog e WebSocket. Questo in particolare per permettere a noi stagisti di vedere come si coprono differenti parti del codice, dagli handler Serverless a funzioni ausiliarie e di validazione. Per testare le parti mancanti sarebbe stato sufficiente seguire gli stessi passi fatti per la gestione delle note. Al contrario, coprire solo gli handler di tutti i moduli avrebbe lasciato una lacuna su come affrontare i test delle parti mancanti.

Per quanto riguarda il frontend invece, avere almeno qualche tipo di test avrebbe sicuramente contribuito a eseguire integrazioni di nuove parti del software con più serenità. È capitato di chiudere in `dev` delle parti che nello sviluppo in locale funzionavano correttamente e si integravano bene con quanto preesistente, per vedere poi che altre parti non funzionavano più come previsto (comunque in un ambiente "sicuro" come quello di `dev`). Si andava quindi spesso a ricontrollare a mano che tutto funzionasse correttamente. Sarebbe quindi stato utile avere qualche tipo di testing, come ad esempio quello "End to End" con strumenti come Cypress.

In tutto questo bisogna però anche considerare le dimensioni, lo scopo, l'utilità e la criticità del progetto. Si tratta infatti di un progetto principalmente didattico. Quindi permettere al tirocinante di vedere più cose diverse possibili è sicuramente importante. In secondo luogo è un modo per esplorare dei concept per strumenti che possono essere utili all'azienda. Non avere una suite di test completa ed esaustiva, soprattutto a discapito di funzionalità primarie, è quindi accettabile in questo contesto. Tuttavia è molto chiaro che la mancanza di test è pericolosa e rischia seriamente di compromettere lo sviluppo futuro di un progetto.

Capitolo 7

Conclusioni

In Questo capitolo verrà prima esposto il risultato del progetto, con una breve descrizione e degli screenshot, contenenti però dati fittizi per non divulgare informazioni sensibili. Per finire si riporterà un'analisi critica dell'esperienza di stage.

7.1 Risultato

L'applicazione **Agile Wall** sviluppata si divide in 3 parti: **Wall**, **WIP (Work in Progress)** e **Backlog**. L'applicazione è protetta da una schermata di **login con Google**, che rende possibile l'accesso solo a utenti di Zero12. Una volta effettuato l'accesso, il sistema distingue utenti semplici e project manager (PM). I PM hanno completo accesso a tutte le parti dell'applicazione. Un utente semplice invece può solo visualizzare il Wall ma senza effettuare modifiche.

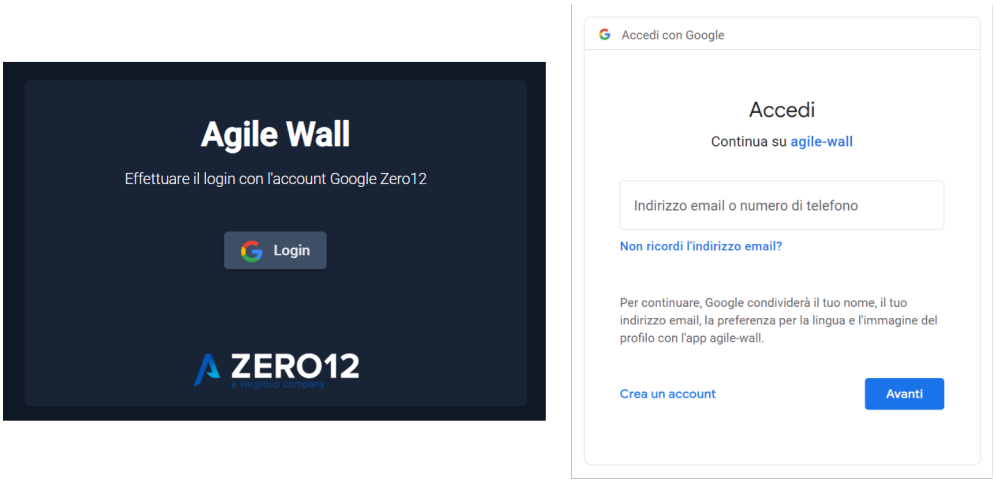


Figura 7.1: Schermate di login all'applicazione

Il **Wall** per la gestione dei progetti è una tabella le cui colonne sono i lunedì di ciascuna settimana e le righe progetti attivi all'interno di Zero12. All'interno di questo Wall sono contenute **note** di tipo "Milestone" (in amaranto quelle attive, in turchese quelle completate) e "Commento" (in ocra). Una milestone attiva viene tracciata da

un indicatore, posto in alto a destra, che evidenzia il numero di note in ritardo rispetto al totale di milestone. Un hover su tale indicatore colora temporaneamente le note in ritardo in arancione. Una milestone è considerata in ritardo se viene spostata dopo la sua data di creazione originale oppure se contenuta in una settimana precedente a quella corrente. In questo modo è possibile vedere a colpo d'occhio quali scadenze e priorità dover gestire urgentemente. Le note sono trascinabili e restano confinate nei limiti delle righe del proprio progetto. Il Wall evidenzia sempre la settimana corrente e mostra 3 settimane precedenti e 52 settimane a venire. Se una nota di tipo milestone ancora attiva esce dal range di settimane visualizzabili (e.g. in ritardo di 1 mese), viene automaticamente portata avanti fino a essere visibile (quindi alla terza settimana precedente quella corrente).

Figura 7.2: Schermate del Wall per la gestione dei progetti

All'interno del Wall è possibile effettuare **zoom in / out e panning**. Se i livelli di zoom o la posizione della viewport non permette di vedere gli header di settimane e progetti, viene renderizzata una versione ridotta non invasiva, che scala anche a grandi livelli di zoom.

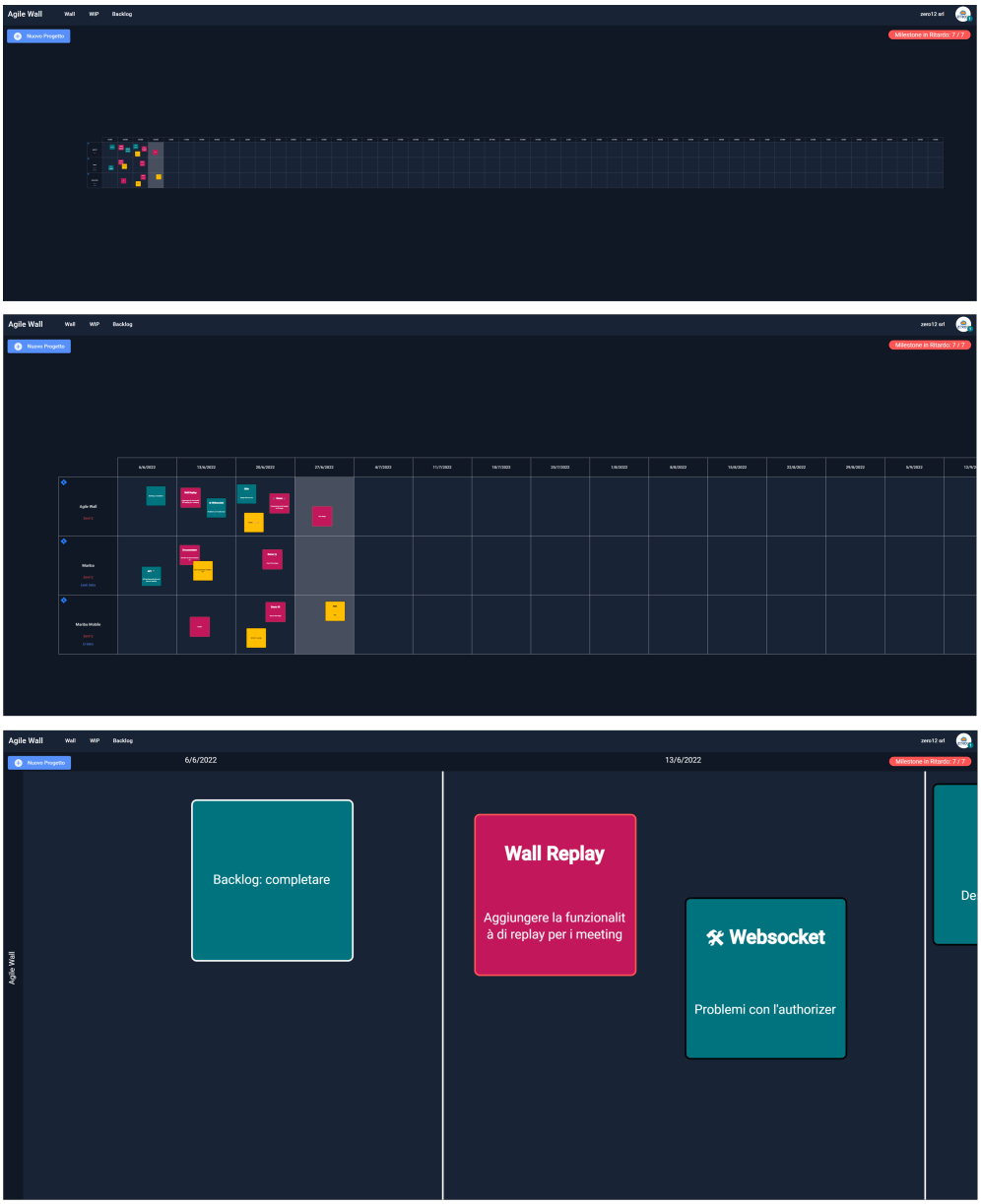


Figura 7.3: Diversi livelli di zoom e posizione della viewport del Wall

Con un doppio click è possibile **creare una nuova** nota nel punto specificato. Si aprirà un dialog popup per aggiungere tutte le informazioni necessarie. Un doppio click su una nota esistente invece, ne permette la **modifica e la cancellazione**.

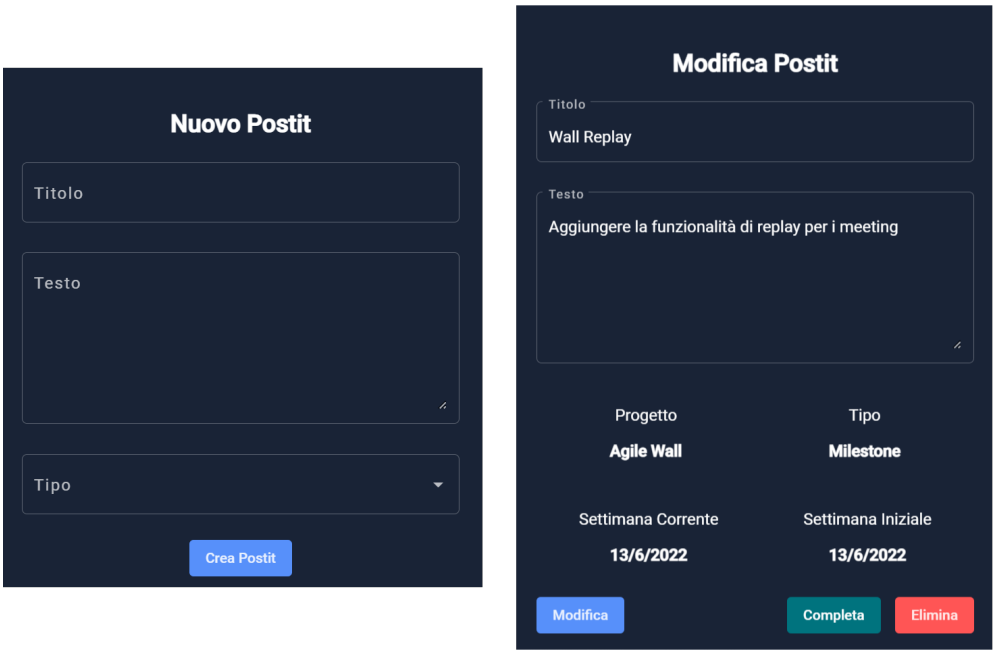


Figura 7.4: Dialog di creazione e modifica di una nota del Wall

Un pulsante in alto a sinistra permette di **creare un nuovo progetto**, che verrà inserito nel Wall in ordine alfabetico. Un doppio click sul progetto permette di **modificarlo, eliminarlo o archiviarlo**. Un progetto archiviato non è visibile nel Wall e può essere riaperto dal WIP.

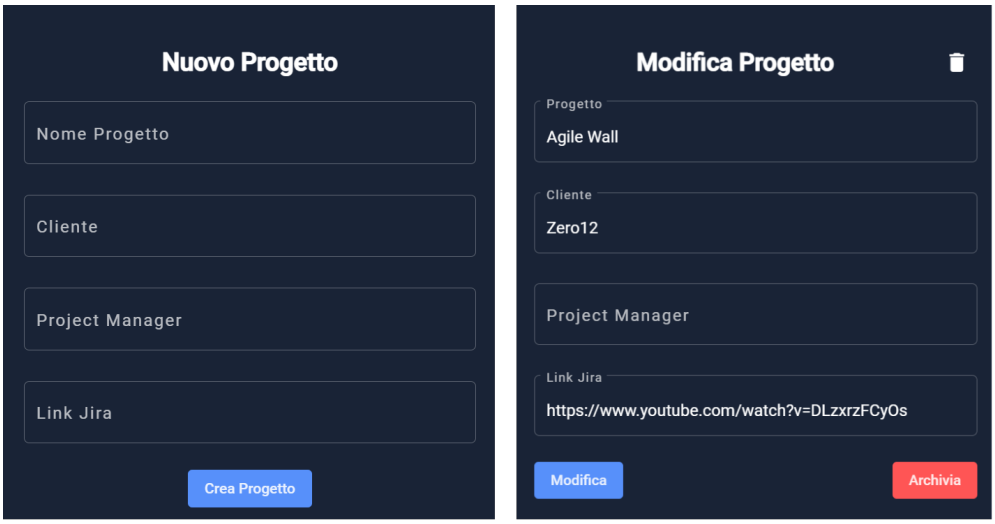


Figura 7.5: Dialog di creazione e modifica di un progetto del Wall

Il **Backlog** di progetti ha un design simile a quello del Wall. Le righe sono i project manager di Zero12 mentre le colonne rappresentano lo stato di un'attività, che può

essere in "Todo", "Doing", "Done". Le note presenti possono essere di tipo "Attività" o "Commento". La user experience è la medesima del Wall.

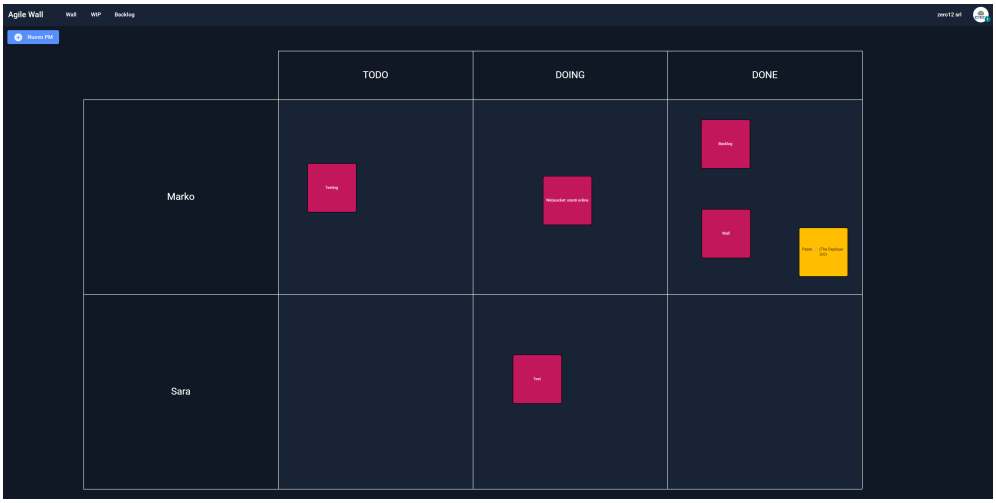


Figura 7.6: Schermate del Backlog per le attività dei project manager

In tutte le schermate dell'applicazione, nell'**header** in alto a destra sono presenti nome e immagine profilo Zero12 dell'utente collegato. Premendo su tali informazioni è possibile effettuare il logout oppure vedere gli utenti connessi all'applicazione in tempo reale. Un piccolo numero sotto l'immagine del profilo indica il numero di **utenti attualmente connessi**.

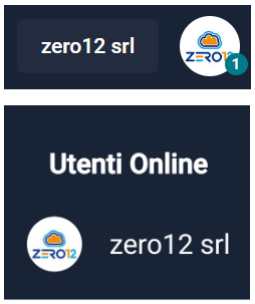


Figura 7.7: WebSocket utilizzato per vedere gli utenti connessi in tempo reale

Infine, in tutti i casi in cui si effettua un'eliminazione di dati, è stato inserito un **dialog popup di conferma**. Se i dati sono particolarmente importanti, oltre a premere un pulsante è necessario inserire una stringa di conferma.

Figura 7.8: Esempi di dialog di conferma per l'eliminazione di dati

Alla fine del periodo di stage, sono stati soddisfatti i seguenti requisiti, definiti in Analisi:

RF1.1	RF1.2	RF1.3	RF1.4
RF1.5	RF1.6	RF1.7	RF1.8
RF1.9.1	RF1.9.2	RF1.9.3	RF1.9.4
RF1.9.5	RF1.9.6	RF1.9.7	RF1.9.8
RF1.10	RF1.11.1	RF1.11.2	RF1.11.3
RF1.11.4	RF1.11.5	RF1.12.1	RF1.12.2
RF1.12.3	RF1.12.4	RF1.13	RF1.14.1
RF1.14.2	RF1.14.3	RF1.14.4	RF1.15.1
RF1.15.2	RF1.15.3	RF1.15.4	RF1.16.1
RF1.16.2	RF1.16.3	RF1.16.4	RF1.16.5
RF1.16.6	RF1.17	RF1.18	RF1.19
RF1.20	RF1.21	RF1.22.1	RF1.22.2
RF1.22.3	RF1.22.4	RF1.23.1	RF1.23.2
RF1.23.3	RF1.23.4	RF1.24.1	RF1.24.2
RF1.25	RF1.26	RF1.27	RF1.28
RF1.29	RF1.30.1	RF1.30.2	RF1.31

RQ1.1	RQ1.2	RQ1.4	RV1.1
RV1.2	RV2.3	RV2.4	RV1.5

Tabella 7.1: Requisiti soddisfatti

L'unico requisito che è stato solo parzialmente soddisfatto è: RQ2.3.

7.2 Analisi Retrospettiva

7.2.1 Resoconto dello Stage

Lo stage è stata sicuramente una delle attività più formative del corso di laurea. Mettersi alla prova in un contesto professionale e con un progetto reale è stato molto stimolante. Gli obiettivi formativi pianificati, definiti nella sezione di introduzione allo stage, sono stati pienamente raggiunti. La pianificazione iniziale è stata indicativamente seguita ma lo *Sprint 1* è sicuramente stato più impegnativo degli altri e alcuni suoi compiti sono stati ripartiti negli altri due sprint. I prodotti attesi sono stati tutti consegnati a meno dell'articolo per il blog Zero12, la cui scrittura è stata rinviata per offrire l'opportunità di utilizzare più tempo per necessità legate al progetto.

Durante il periodo di tirocinio, lato azienda si è espressa una certa curiosità e interesse verso lo stagista. Questo ha portato, al termine dello stage, a una proposta di collaborazione futura dopo la conclusione del ciclo di studi.

A prescindere dalle considerazioni critiche che verranno fatte di seguito, nel complesso è stato un periodo altamente positivo e mi ritengo personalmente soddisfatto. Il progetto proposto è stato tanto interessante quanto impegnativo. La disponibilità dei tutor *Francesco Meneguzzo* e *Federico Pinton*, del CEO di Zero12 *Stefano Dindo* e anche di tutto resto del team di Zero12 è stata sempre molto alta. Anche la collaborazione con la mia collega studentessa *Sara Nanni* è stata molto utile e sempre piacevole. Per tutto il periodo di stage ho personalmente cercato di essere proattivo e mantenere alta la mia motivazione. Spero che questo possa trasparire dal risultato ottenuto. Le comunicazioni con tutti sono sempre state molto chiare e trasparenti. Tutti i problemi emersi sono stati individuati e quando possibile risolti rapidamente. Comunque non ci sono mai state urgenze. Infine ho apprezzato quando ci si prendeva il tempo di esprimere dei rallegramenti quando le cose andavano bene.

7.2.2 Miglioramenti

Durante il corso del progetto, frammenti di codice problematico sono spesso stati fatti visionare ai tutor. In aggiunta sono state fatte due sessioni esaustive di code review, una per il frontend e una per il backend. Tutto questo, insieme a considerazioni personali, fa emergere le seguenti problematiche:

- Il testing sarebbe potuto essere più esaustivo. Ciò avrebbe portato a rilasci più tranquilli e meno possibilità di introdurre regressioni;
- Utilizzando il canvas, tutte le operazioni di rendering che normalmente vengono gestite dal DOM o in qualche modo ancora più furbo dai Framework, sono dovute

essere replicate a mano. Ciò sicuramente introduce un certo grado di inefficienza che, avendo più tempo a disposizione, potrebbe sicuramente essere migliorato, portando ad un uso più vantaggioso delle risorse;

- La gestione degli errori, specialmente lato frontend, sarebbe dovuta essere sviluppata meglio. Capita infatti che l'applicazione "fallisca silenziosamente" (e.g. connessione, scadenza token autenticazione) e l'utente non venga avvisato, rischiando di perdere parte della sessione di lavoro;
- In alcuni casi si è abusato della libertà offerta dal sistema di tipizzazione di TypeScript, ricadendo nei controlli molto deboli di JavaScript;
- Lato backend, uno studio più accurato della documentazione di DynamoDB, avrebbe permesso l'utilizzo di API più specifiche e appropriate in certi contesti, invece di utilizzare operazioni generiche;
- Avere uno strumento di analisi statica del codice, come un linter, avrebbe portato a codice più facilmente manutenibile e con uno stile più uniforme.

7.2.3 Possibili Estensioni

Tra le prime estensioni possibili, sicuramente si potrebbe riflettere su come sviluppare alcune delle idee inizialmente proposte, tra cui:

- Funzionalità di replay dei meeting, mediante un log delle attività del Wall;
- Funzionalità di collaborazione real-time, predisposte con il WebSocket ma mai implementate. Ad esempio vedere in tempo reale i postit selezionati e spostati dai colleghi, senza dover effettuare un refresh, potrebbe essere interessante. Questa funzionalità è stata proposta inizialmente ma non è stata ritenuta critica visto che, tipicamente, l'aggiornamente del Wall dei progetti e delle attività del Backlog avvengono autonomamente da parte di ciascun PM prima di un meeting Scrum of Scrums e vengono visualizzate complessivamente solo durante suddetto incontro.

Infine è possibile estendere l'applicazione con altre piccole feature come:

- Progress bar dei progetti sul Wall legata ai parametri del WIP, in modo da poter confrontare l'avanzamento dei progetti a colpo d'occhio già dal Wall;
- Aggiungere altre tipologie di note;
- Aggiungere una sezione con preferenze e personalizzazioni.

Glossario

Agile Metodologia di sviluppo software che, a differenza di modelli più tradizionali, propone un approccio focalizzato a consegnare al cliente un software funzionante e di qualità in tempi brevi. Fra le pratiche promosse dai metodi agili ci sono: automazione, coinvolgimento diretto e continuo del cliente, consegne frequenti, formazione di team di sviluppo piccoli e auto-organizzati, pianificazione adattiva, miglioramento continuo, testing. Tra i framework agili più conosciuti ci sono: scrum, extreme programming, lean, feature driven development. 1, 5, 8, 12

AWS Amazon Web Services è un'azienda statunitense di proprietà del gruppo Amazon, che fornisce servizi di cloud computing on demand su un'omonima piattaforma. Tra le caratteristiche dei servizi ci sono: high availability, ridondanza e sicurezza. Il costo finale deriva dalla combinazione di tipo e quantità di risorse utilizzate, caratteristiche scelte dall'utente, tempo di utilizzo e performance desiderate. A partire dai dati pubblicati nel quarto trimestre del 2018, AWS rappresenta per Amazon il 58% dei guadagni totali, rendendola la sua più grande fonte di incassi. 1, 2, 9, 10, 12, 20, 25–27

CI La Continuous Integration è la pratica di allineare frequentemente il codice locale di ciascun sviluppatore con quello remoto condiviso. Una Pipeline CI (o CI/CD: Continuous Integration e Continuous Delivery) è un processo automatico che si occupa di effettuare la build del progetto ogni volta che il codice viene allineato nella repository remota. In questo modo si può avere conferma tangibile dell'integrazione di nuove parti. 10, 12

CLI Una Command Line Interface o CLI, è un tipo di interfaccia utente caratterizzata da un'interazione testuale con la macchina. Ne è un esempio la shell Unix `bash` o la shell Windows `cmd`. Molti programmi hanno un corrispettivo CLI che, a differenza di un'interfaccia grafica, permette di automatizzare operazioni in modo più semplice. 9, 22, 23, 31

Dependency Injection Design pattern che ha l'obiettivo di avere un sistema lascamente accoppiato astraendo la costruzione di un oggetto (la dipendenza) dal suo utilizzatore. Tra i tipi di DI i più utilizzati di sono *constructor injection* e *setter injection*. 30

DOM Il Document Object Model è un oggetto che rappresenta la struttura e il contenuto di un documento HTML, permettendo di manipolarlo. 59

ECMA-Script 2015 Standard JavaScript introdotto nel 2015, chiamato anche ES6, che introduce importanti feature nel linguaggio come: dichiarazione di classi, importazione di moduli, arrow functions, `let` e `const`, promises, etc. 24

Framework Insieme di codice riutilizzabile che a differenza di una libreria adotta l'inversione del controllo: il codice di una libreria viene utilizzato all'interno del flusso di controllo definito dallo sviluppatore; invece il codice di un framework gestisce il flusso di controllo ed è lo sviluppatore ad adattarsi a esso. 25, 28, 29, 59

git Software di versionamento distribuito sviluppato da Linus Torvalds nel 2005 per supportare lo sviluppo del kernel Linux. Utilizzabile mediante CLI o GUI. 9, 20

HTML <canvas> Elemento HTML che permette di utilizzare diverse API per il rendering di grafiche e animazioni. 32

JWT JSON Web Token è uno standard proposto per il trasferimento di dati JSON. I token possono opzionalmente essere firmati e/o criptati. 42

Milestone In project management è uno specifico punto nella timeline di progetto che denota un certo livello di progresso dimostrabile. 3, 42

Repository Nel ambito del controllo della versione, una repository è una struttura che memorizza metadati per un insieme di file e directory. Può essere sia locale, sul computer dello sviluppatore, sia remota, cioè su un server, utilizzabile da un team per collaborare su uno stesso progetto. 9, 10, 20

transpilazione Processo di compilazione "source-to-source" ovvero in cui il compilatore non traduce il codice sorgente in codice macchina ma in altro codice sorgente. Nel contesto di questo documento è utilizzato per riferirsi al compilatore TypeScript, che traduce codice TypeScript in codice JavaScript. 23

Unit Testing Metodo di testare individualmente le unità di un software per determinare se rispettano particolari caratteristiche volute. 20, 28, 49, 50

UUID "Universally unique identifier" è una stringa di 16 Byte (128 bit) utilizzata per l'identificazione di informazioni. In pratica, l'ampiezza dello spazio delle chiavi e il loro processo di generazione offrono sufficienti garanzie che la stessa chiave non venga assegnata a due entità differenti, anche se teoricamente, il fatto che le chiavi siano in numero finito, implica che due entità potrebbero avere la stessa chiave identificativa. 42

Bibliografia

Siti web consultati

AJV Documentation. URL: https://ajv.js.org/api.html.

Amazon Web Services. URL: https://aws.amazon.com/.

Angular Documentation. URL: https://angular.io/docs.

Angular Material Components Documentation. URL: https://material.angular.io/components/categories.

Atlassian Agile Coach. URL: https://www.atlassian.com/agile.

Atlassian Git Guides. URL: https://www.atlassian.com/it/git/tutorials.

Fabric.js Documentation. URL: http://fabricjs.com/articles/.

Github State of the Octoverse 2021. URL: https://octoverse.github.com/ (cit. a p. 23).

Jest Documentation. URL: https://jestjs.io/docs/getting-started.

MDN WebSocket Documentation. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.

Node.js Documentation. URL: https://nodejs.org/en/docs/.

Serverless Framework Documentation. URL: https://www.serverless.com/framework/docs.

Stack Overflow Developer Survey 2022. URL: https://survey.stackoverflow.co/2022 (cit. a p. 23).

TypeScript Documentation. URL: https://www.typescriptlang.org/docs/.

Unipd - Corso di Ingegneria del Software 2021-2022. URL: https://docs.google.com/spreadsheets/d/11pBURqeGTaU-wAlDtL0KduJBiGgJ8iqe7qAjy1yIEjk/edit#gid=2015829435.

Wikipedia. URL: https://www.wikipedia.org/.