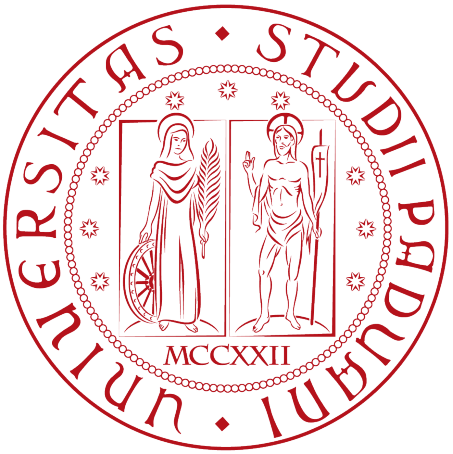
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Un prototipo di sistema di Home Automation basato su microservizi

Tesi di laurea triennale

Relatore

Prof. Tullio Vardanega

Laureando

Nicola Dal Maso

Anno Accademico 2017-2018

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Nelle sezioni di questo capitolo parlerò dell'idea alla base dello svolgimento dello stage e dei rischi derivati dalle scelte effettuate.

1.1 L'idea

Internet Of Things è un paradigma tecnologico diffusosi nell'ultimo decennio. Questa locuzione fa riferimento a un insieme di oggetti, di varia natura e utilizzo, che interagiscono tra loro e che permettono all'utente di interagire con essi. L'idea che mi ha spinto allo sviluppo e alla realizzazione del progetto di stage è stata quella unificare in un unico centro di controllo tutti gli eventuali dispositivi connessi alla rete domestica dell'utente, permettendo tuttavia allo stesso di accedere all'interfaccia proprietaria di ciascun dispositivo. L'obiettivo di una tale dashboard non è quindi *intrappolare* l'utente in un unico ecosistema domotico, bensì quello di facilitare la consultazione delle informazioni più frequentemente richieste dall'utente provenienti da più ecosistemi distinti.

Per assicurare prestazioni, affidabilità (*availability*) e scalabilità del prodotto sviluppato e per aggiungere una natura formativa al progetto di stage, ho scelto di progettare il prodotto secondo l'architettura a microservizi.

1.1.1 IoT

L'idea di una rete di dispositivi smart fu presa in considerazione fin dal 1982, quando un distributore automatico di bibite dell'Università di Carnegie Mellon venne opportunamente modificato per accedere al suo inventario di bibite. Quel distributore automatico divenne il primo apparecchio collegato ad Internet.[1]

Nel corso degli anni '90 il mondo accademico e il mondo dell'industra legata alla produzione continuarono a sperimentare evolvendo il *concept* iniziale, arrivando alla conclusione che l'*Ubiquitous Computing* non si debba riferire solamente ai *computer*, ma debba espandersi agli oggetti di utilizzo quotidiano. Questa visione dovette scontrarsi con i limiti della microelettronica di allora: la produzione di semiconduttori non era

[1] *The Carnegie Mellon University Computer Science Department Coke Machine*. URL: https://www.cs.cmu.edu/~coke/history_long.txt.

ancora pronta a supportare la potenziale domanda e i costi per sostenere l'ampliamento degli impianti non erano facilmente assorbibili in breve tempo.[2]

Il termine *"Internet of Things"* fu coniato da Kevin Ashton nel 1999.[3] L'origine dell'espressione, riassumendo le parole dell'autore, deriva dal fatto che la maggior parte delle informazioni presenti su Internet sono state e sono inserite da utenti "umani", soggetti quindi a concentrazione, precisione e tempo limitate; se queste informazioni fossero invece inserite da macchine senza l'aiuto di un utente umano, la maggior qualità delle stesse garantirebbe

* maggiore capacità di tracciamento delle risorse;
* minor spreco di risorse;
* minor costo per la gestione delle risorse.

Grazie agli avanzamenti nei processi di produzione dei semiconduttori, dovuti alla crescita dei mercati del consumo di massa, allo sviluppo di un numero sempre maggiore di tecnologie volte a migliorare l'efficienza e l'affidabilità dei circuiti integrati e alla sempre maggior diffusione di tecnologie per la trasmissione di informazioni senza fili, dai primi anni 2000 il concetto alla base dell'IoT è stato sviluppato da un numero sempre maggiore di aziende negli ambiti più disparati (*home automation*, *manufacturing*, *smart agriculture*, etc.).[4]

Con Internet of Things si intende una rete di dispositivi interconnessi, individuabili in modo univoco e che possono comunicare informazioni. I dispositivi presenti in una rete possono comunicare con due tipologie di attori diverse:

* se la comunicazione avviene con altri dispositivi si parla di comunicazione M2M (*Machine to Machine*, ovvero comunicazione tra macchine);
* se la comunicazione avviene interagendo con il mondo reale si parla di comunicazione M2H (*Machine to Human*, ovvero comunicazione tra macchina e utente).

Il termine "Things" nel contesto IoT si riferisce a una varietà di dispositivi come ad esempio: videocamere di sorveglianza, automobili a guida autonoma e assistita oppure piccoli e grandi elettrodomestici casalinghi. Questi dispositivi, soprannominati anche *smart object*, raccolgono informazioni utili in base agli attori con cui comunicano:

* se i dispositivi comunicano in modo M2M, le informazioni vengono impiegate al supporto di tecnologie esistenti, integrandosi nel flusso di informazioni esistente;
* se i dispositivi comunicano in modo M2H, le informazioni vengono in aiuto delle persone che interagiscono con essi.

L'interesse verso il tema IoT è cresciuto esponenzialmente sia nel mercato consumer che in quello enterprise e secondo Forbes ([5]) diventerà nel prossimo quinquennio uno

[2] Mark Weiser. *The computer for the 21st century*. New York, NY, USA, 1999. URL: https://web.archive.org/web/20150311220327/http://web.media.mit.edu/~anjchang/ti01/weiser-sciam91-ubicomp.pdf.

[3] Kevin Ashton. *That 'Internet of Things' Thing*. 2009. URL: http://www.rfidjournal.com/articles/view?4986.

[4] Christian Floerkemeier Friedemann Mattern. «From the Internet of Computers to the Internet of Things». In: (2010). URL: http://www.vs.inf.ethz.ch/publ/papers/Internet-of-things.pdf.

[5] Louis Columbus. *2017 Roundup Of Internet Of Things Forecasts*. 10 Dic. 2017. URL: https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts.

dei settori dell'ITC più redditizi. É interessante osservare anche la popolarità dei termini di ricerca correlati all'IoT: i dati sono stati ottenuti interrogando il servizio https://trends.google.com/trends, il quale consente di effettuare analisi sulla popolarità delle stringhe di ricerca immesse nel motore di ricerca di Google. Ciascun grafico a linea (*line chart* in inglese) presenta nelle ordinate il grado di popolarità della *query* di ricerca, valutato da 0 (popolarità minima) a 100 (popolarità massima), e nelle ascisse l'arco temporale in analisi.

Figura 1.1: Andamento dell'interesse per la stringa di ricerca *"internet of things"*.
URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=internet%20of%20things

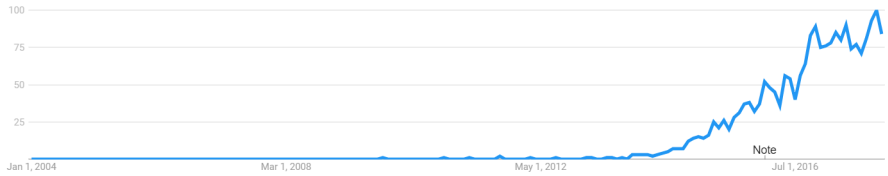


Figura 1.2: Andamento dell'interesse per la stringa di ricerca *"iot devices"*.
URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=iot%20devices

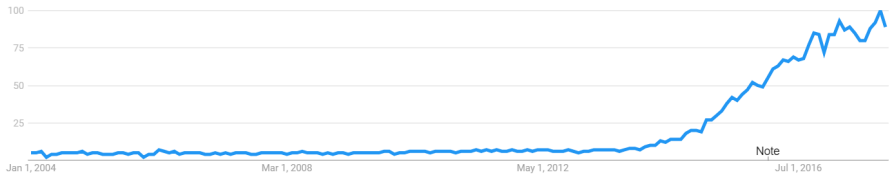


Figura 1.3: Andamento dell'interesse per la stringa di ricerca *"iot"*.
URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=iot

Sintetizzando le previsioni di Forbes con l'andamento dei termini di ricerca legati all'IoT, visibili alle figure 1.1, 1.2 e 1.3, si può notare come l'interesse verso l'argomento IoT stia generalmente aumentando o nel caso peggiore resti stabile con l'interesse degli anni precedenti.

1.1.2 Architettura a microservizi

L'espressione **Architettura a microservizi** è sempre più comune tra gli sviluppatori di applicazioni *enterprise* per descrivere un metodo di progettazione delle applicazioni come insiemi di servizi eseguibili indipendentemente (ogni servizio esegue su un processo indipendente), che comunicano tra loro grazie a meccanismi di comunicazione "leggeri" (solitamente attraverso Application Program Interface (API) HTTP).

Caratteristiche delle architetture monolitiche Un'applicazione monolitica è progettata e costruita per essere una singola unità in esecuzione. Solitamente un'applicazione web monolitica è divisa in 3 parti:

* interfaccia utente (pagine web);
* database;
* applicazione server.

A sua volta, l'applicazione server:

* gestisce le richieste HTTP;
* esegue la business logic dell'applicazione;
* carica e aggiorna dati dal/nel database;
* seleziona e popola le pagine web inviate al browser dell'utente.

Nelle applicazioni monolitiche la modularità del sistema si ottiene sfruttando i costrutti fondamentali dei linguaggi di programmazione:

* funzioni;
* classi;
* namespace o package.

Per aumentare la disponibilità delle applicazioni monolitiche si usa replicare istanze dell'applicazione in molteplici server, bilanciando il traffico verso le applicazioni per mezzo di un Load balancer. Tra i difetti delle applicazioni monolitiche si possono evidenziare:

* modifiche a una piccola parte all'applicazione richiedono la ricompilazione e la ridistribuzione dell'applicazione;
* all'accrescere della complessità dell'applicazione aumenta anche la difficoltà nel mantenere le modifiche isolate ai moduli di competenza;
* scalare l'applicazione richiede l'esecuzione di istanze multiple della stessa applicazione, ignorando di fatto eventuali requisiti di efficienza (solitamente alcune componenti del sistema non richiedono un aumento di throughput).

Caratteristiche delle architetture a microservizi Per lo stile architetturale a microservizi non esistono definizioni formali, tuttavia è possibile dedurre le caratteristiche che hanno accomunato i progetti diventati nel tempo esempi di best-practice. Non tutte le architetture a microservizi hanno tutte le caratteristiche elencate in seguito, ma ci si aspetta che la maggior parte delle architetture esibisca quante più caratteristiche possibili.

L'aspetto cruciale delle architetture a microservizi verte sulla definizione di componente: la definizione comunemente accettata è quella di "unità di software che è indipendentemente aggiornabile e sostituibile in un sistema". Le architetture a microservizi usano i servizi per realizzare tale definizione di componente. A titolo di confronto con gli approcci di sviluppo tradizionali è possibile introdurre la nozione di libreria. Le librerie sono componenti insiti in un'applicazione tanto da risiedere nello stesso spazio di memoria dell'applicazione e che per essere invocate richiedono una chiamata di funzione in memoria. I servizi sono componenti che vivono nel sistema come processi separati, sfruttando vari tipi di comunicazione interprocesso: richieste web, chiamate di funzione remote (RPC).

Il vantaggio principale dei servizi rispetto alle librerie consiste nel fatto che i servizi sono rilasciabili indipendentemente dal sistema. Data la natura dell'architettura a microservizi, modifiche a un singolo servizio comportano il rilascio di una nuova versione solamente per quel servizio e non dell'intera applicazione. Una buona architettura a microservizi quindi mira a progettare e implementare servizi che circoscrivano chiaramente il loro scopo.

L'uso di servizi come componenti consente inoltre di rendere esplicita l'interfaccia dei componenti. Spesso solamente la documentazione e la disciplina prevengono usi impropri di una componente da parte di uno sviluppatore esterno, rischiando di causare un alto accoppiamento tra componenti. I servizi facilitano il rispetto delle interfacce pubblicate attraverso l'uso di meccanismi di chiamate remote esplicite.

Il difetto che si annovera all'uso di servizi come componenti risiede nell'utilizzo di chiamate remote per la comunicazione tra servizi: esse richiedono più risorse rispetto alle chiamate di funzione intraprocesso e quindi è necessario progettare le API di ciascun servizio rivolgendo maggiore attenzione all'aspetto prestazionale delle stesse.

Altre differenze sono riscontrabili dal punto di vista della suddivisione delle persone impegnate nello sviluppo dell'applicazione. Solitamente applicazioni complesse sviluppate seguendo l'architettura monolitica sono divise in team con competenze isolate, illustrata graficamente in figura 1.4:

* team esperto in UI;
* team specializzato in DB Management;
* uno o più team specializzate a realizzare la logica di business.

Quando le persone sono così fortemente isolate, anche una semplice modifica può richiedere l'intervento di altre persone in team diversi, causando inefficienze nel processo di sviluppo.

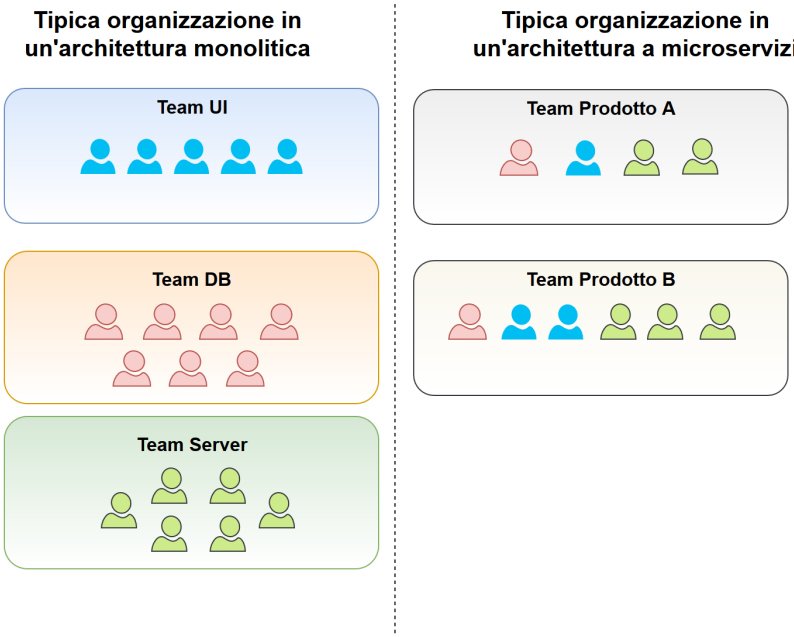


Figura 1.4: Illustrazione che mostra la differente organizzazione aziendale in un ambiente di sviluppo monolitico e in un ambiente di sviluppo a microservizi.

L'approccio microservices-oriented alla suddivisione dell'applicazione invece pone l'accento sulle capacità di business: ogni team inerente un particolare settore di business si occupa dell'intero prodotto per quel settore (sviluppando interamente UI, DB, ecc.). I team in questo approccio sono multidisciplinari e gli scambi con altri settori riflettono le effettive dipendenze tra un settore e un altro all'interno dell'azienda.

Un esempio di quest'approccio alla suddivisione lo si ritrova in Amazon, dove vige il motto "you build, you run it" ("tu lo costruisci, tu lo esegui"). In Amazon ogni team ha completa responsabilità del prodotto anche in ambiente di produzione, mettendo in comunicazione diretta sviluppatori e utenti del prodotto per le attività di supporto e manutenzione.

Applicazioni assemblate con microservizi mirano ad essere più disaccoppiate e più coese possibile: ricevendo una richiesta, applicando la propria logica e producendo una risposta. Un diagramma esemplificativo di questo concetto lo si trova in figura 1.5.

Le comunicazioni tra servizi sono orchestrate usando semplici protocolli REST-like. I due protocolli più usati sono:

* richieste/risposte HTTP secondo API ben dettagliate;
* messaggistica in un canale di comunicazione snello. I servizi producono e consumano i messaggi che circolano nel canale di comunicazione, secondo regole di accesso definite.

Quando un'applicazione è suddivisa in molteplici componenti sorgono naturalmente dubbi sulla gestione delle informazioni che ciascuna componente gestisce. Solitamente nelle architetture monolitiche i domini dell'applicazione vengono astratti scegliendo una fra le tecniche di modellazione disponibili e applicandola a tutti i domini; i modelli prodotti sono poi veicolati su singoli storage di dati (ad es. unico database). L'architettura a microservizi invece propone di concepire i modelli in autonomia

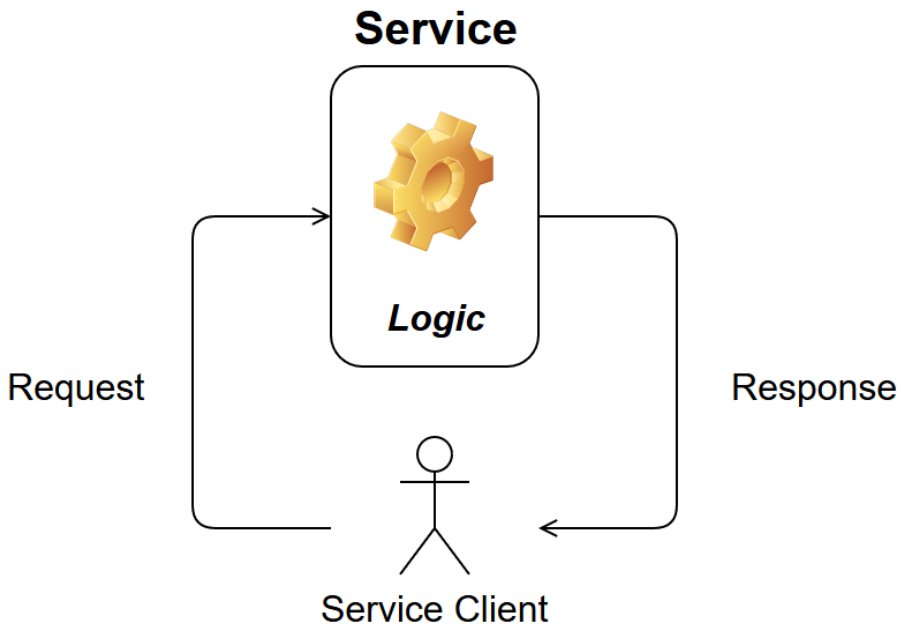


Figura 1.5: Illustrazione che mostra come un utente (inteso sia come umano, sia come altro servizio) di un servizio interagisca con esso.

per ogni singolo servizio, utilizzando le tecniche ritenute più appropriate. Questa decentralizzazione dei modelli si riflette anche sulla decentralizzazione delle decisioni di storage dei dati. Ogni servizio gestisce il proprio database: il database potrebbe essere quindi un'istanza di una stessa piattaforma tecnologica oppure una piattaforma specifica e ottimizzata per il caso d'uso del servizio. Questo approccio alla gestione della persistenza è chiamato Polyglot Persistence.

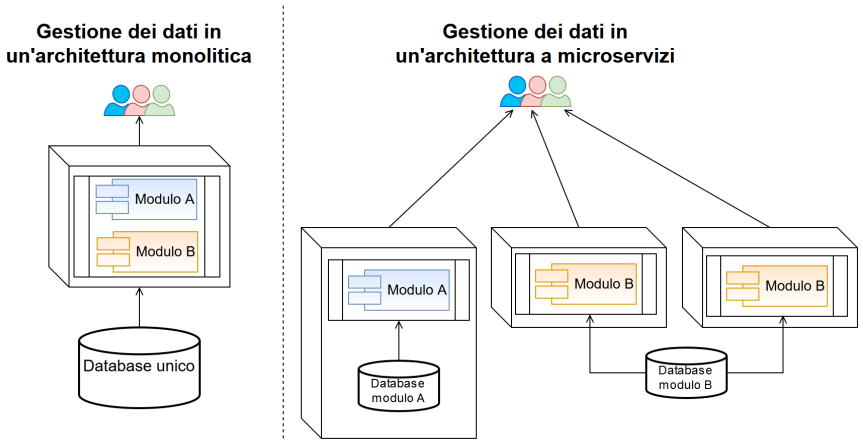


Figura 1.6: Illustrazione che mostra la differente gestione dell'architettura di persistenza dei dati tra prodotti software con architettura monolitica e con architettura a microservizi.

La decentralizzazione delle decisioni di storage implica una maggior attenzione verso gli aggiornamenti dei dati. L'approccio comune agli aggiornamenti in un'architettura monolitica è quello di usare le transazioni per garantire la consistenza dei dati prima e dopo ciascun aggiornamento. L'utilizzo di transazioni è un grave limite per l'architettura a microservizi, in quanto le transazioni impongono un ordine temporale che potrebbe non essere rispettato, causando inconsistenze dei dati salvati. É per questo che le architetture a microservizi enfatizzano l'utilizzo di comunicazioni non vincolanti (transactionless): eventuali inconsistenze vengono segnalate e risolte grazie a operazioni correttive.

Una conseguenza nell'utilizzo dei servizi come componenti è che le applicazioni devono prevedere e tollerare malfunzionamenti nei servizi. L'utilizzatore dei servizi deve quindi rispondere ai malfunzionamenti nel modo più elegante possibile. Dato l'aumento di complessità, questo è da annoverare tra i difetti delle architetture a microservizi. Dal momento che i servizi possono malfungere in ogni momento, è fondamentale riuscire a:

monitorare il servizio, segnalare il malfunzionamento e ripristinare automaticamente il servizio nel più breve tempo possibile. Conseguentemente, ogni servizio deve essere progettato focalizzando l'attenzione sulle attività di monitoring, individuando le metriche rilevanti (ad es. throughput, latenza, ecc.).

Utilizzando i servizi come componenti ci si chiede spesso quante responsabilità debba avere ciascun servizio: la caratteristica fondamentale da osservare è la nozione di sostituzione e aggiornamento indipendenti. Un buon segnale lo si ritrova quando ad ogni modifica di un servizio, questa modifica non richiede adattamenti in altri servizi (a meno di modifiche di funzionalità offerte). Se 2 o più servizi vengono aggiornati spesso insieme probabilmente essi dovrebbero essere uniti.

1.2 Rischi

Illustrare i rischi derivati dalle scelte effettuate.

1.3 Organizzazione del testo

Il secondo capitolo descrive ...

Il terzo capitolo approfondisce ...

Il quarto capitolo approfondisce ...

Il quinto capitolo approfondisce ...

Il sesto capitolo approfondisce ...

Nel settimo capitolo descrive ...

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];

- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Progetto di Stage

Nelle sezioni di questo capitolo parlerò delle aspettative e degli obiettivi posti inizialmente per il progetto di Stage

2.1 Obiettivi curricolari

Questa sezione risponde alla domanda:

Quali valori mi **aspetto** di dedurre dall'esperienza di stage che siano rilevanti sul mercato del lavoro?

2.2 Obiettivi formativi

Questa sezione risponde alla domanda:

Quali informazioni mi **aspetto** di imparare/studiare durante lo svolgimento dello Stage?

2.3 Obiettivi tecnici

Questa sezione risponde alle domande:

Quali tecnologie mi **aspetto** di imparare a padroneggiare durante lo svolgimento dello Stage? Quali competenze mi **aspetto** di apprendere durante lo svolgimento dello Stage?

2.4 Obiettivi di prodotto

Questa sezione risponde alla domanda:

Quali caratteristiche (innovative o non) del prodotto sviluppato mi **aspetto** di poter dimostrare?

Capitolo 3

Svolgimento dello stage

Nelle sezioni di questo capitolo parlerò dell'effettivo svolgimento dello stage: organizzazione dello stage, analisi dei requisiti, progettazione ad alto livello, documentazione prodotta, test sviluppati e validazione dei requisiti.

3.1 Organizzazione dello Stage

Qui posso parlare del tempo impegnato nello svolgimento dello stage e delle milestone raggiunte.

3.2 Ambiente di sviluppo

3.2.1 Strumenti di sviluppo

3.3 Piano di Lavoro

Approfondire il Piano di Lavoro prodotto.

3.4 Analisi dei Requisiti

Approfondire l'Analisi dei Requisiti prodotta.

3.5 Progettazione

Approfondire la Specifica Tecnica prodotta.

3.6 Documentazione

Come è stata prodotta la documentazione? A chi è rivolta? Come è documentato il codice?

3.7 Test

Come sono stati svolti i test? Coverage?

3.8 Validazione dei Requisiti

Capitolo 4

Valutazione retrospettiva

Nelle sezioni di questo capitolo parlerò dell'esperienza avuta durante lo svolgimento dello stage, parlando delle aspettative descritte nel primo capitolo e raffrontandole con le reali attività svolte

4.1 Valutazione raggiungimento degli obiettivi

4.2 Conoscenze acquisite

4.3 Conclusioni

Glossario

API in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 19

Load balancer in informatica, il load balancer è lo strumento hardware o software attraverso cui viene distribuito un carico di lavoro su più risorse di elaborazione (*load balancing*). 4

Acronimi

API Application Program Interface. 4, 17

Bibliografia

Siti web consultati

URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=internet%20of%20things (cit. a p. 3).

URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=iot%20devices (cit. a p. 3).

URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=iot (cit. a p. 3).

Ashton, Kevin. *That 'Internet of Things' Thing*. 2009. URL: http://www.rfidjournal.com/articles/view?4986 (cit. a p. 2).

Columbus, Louis. *2017 Roundup Of Internet Of Things Forecasts*. 10 Dic. 2017. URL: https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts (cit. a p. 2).

Load balancing. URL: https://en.wikipedia.org/wiki/Load_balancing_(computing).

The Carnegie Mellon University Computer Science Department Coke Machine. URL: https://www.cs.cmu.edu/~coke/history_long.txt (cit. a p. 1).