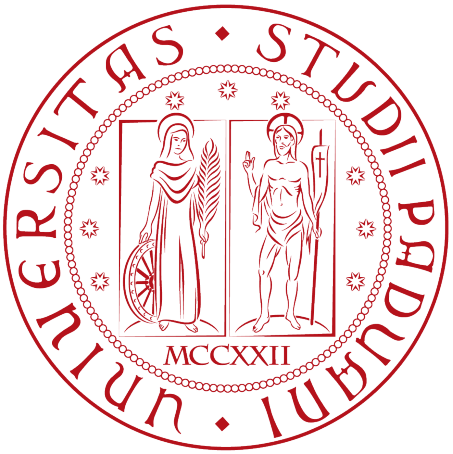
Università degli Studi di Padova

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

Struttura del documento

Il presente documento è articolato in quattro capitoli:

Nel primo capitolo presento i temi su cui ho svolto lo stage, elencando i rischi che ho valutato per le scelte intraprese.

Nel secondo capitolo descrivo con maggior precisione il progetto di stage, elencandone gli obiettivi curricolari, formativi, tecnici e di prodotto.

Nel terzo capitolo approfondisco il lavoro svolto durante lo svolgimento dello stage, esaminando le attività di analisi, progettazione, codifica e test.

Nel quarto capitolo fornisco una valutazione degli obiettivi raggiunti, motivando la presenza di eventuali obiettivi non raggiunti; inoltre esamino le conoscenze acquisite e i rischi descritti nel primo capitolo.

Convenzioni tipografiche

Riguardo la stesura del testo, relativamente al documento ho adottato le seguenti convenzioni tipografiche:

- ho definito un glossario, presente alla fine del presente documento, contenente gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati nel corso del documento;
- ho utilizzato per la prima occorrenza dei termini riportati nel glossario la seguente nomenclatura: parola[g];
- ho evidenziato i termini in lingua straniera o facenti parti del gergo tecnico con il carattere *corsivo*.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'idea

In questa relazione descrivo lo svolgimento dello stage effettuato nel contesto del Corso di Laurea di Informatica dell'Università di Padova.

Ho intrapreso lo stage nella sua forma interna individuale, in cui con il proponente, Prof. Tullio Vardanega, ho redatto un piano di lavoro nel quale lo stage abbia una durata pianificata su 300 ore.

L'obiettivo principale dello stage consiste nello sviluppo e nella realizzazione di un prototipo per la gestione di dispositivi interconnessi (IoT) attraverso un'interfaccia *web*. Questo centro di controllo attraverso cui l'utente del sistema gestisce i dispositivi *smart* presenti nella propria rete domestica dovrebbe permettere operazioni quali:

- avvio/spegnimento di un dispositivo;
- monitoraggio dei dispositivi collegati;
- collegamento all'eventuale interfaccia proprietaria del dispositivo (es. supporto tecnico).

Ho sviluppato e realizzato il progetto di stage con l'idea di 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 confinare 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.
Grazie alla natura prototipale del prodotto sviluppato, ho inoltre potuto sperimentare l'approccio architetturale a microservizi, al fine di garantire scalabilità all'applicazione.

1.2 IoT: definizione e caratteristiche generali

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.

Un dispositivo *smart* è un dispositivo elettronico, generalmente connesso ad altri dispositivi, che può svolgere le proprie funzioni in maniera autonoma. Un esempio lampante di dispositivi *smart* sono gli smartphone, prodotti che hanno arricchito di funzionalità avanzate, interagendo con l'utente in modi precedentemente non possibili, i predecessori telefoni (*phone*).[1]

Studenti e professori del Dipartimento di Informatica dell'Università di Carnegie Mellon presero in considerazione l'idea di una rete di dispositivi *smart* quando modificarono un distributore automatico di bibite del Dipartimento per accedere al suo inventario di bibite. Quel distributore automatico divenne il primo apparecchio collegato ad Internet.[2]

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 ancora pronta a supportare la potenziale domanda e i costi per sostenere l'ampliamento degli impianti non erano facilmente assorbibili in breve tempo.[3]

Kevin Ashton coniò il termine *"Internet of Things"* nel 1999.[4] 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 un numero sempre maggiore di aziende, provenienti dagli ambiti più disparati, ha sviluppato il concetto alla base dell'IoT, estendendolo a settori quali *home automation*, *manufacturing*, *smart agriculture*, etc..[5]

[1] *Smart device*. URL: https://en.wikipedia.org/wiki/Smart_device.

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

[3] 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.

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

[5] 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.

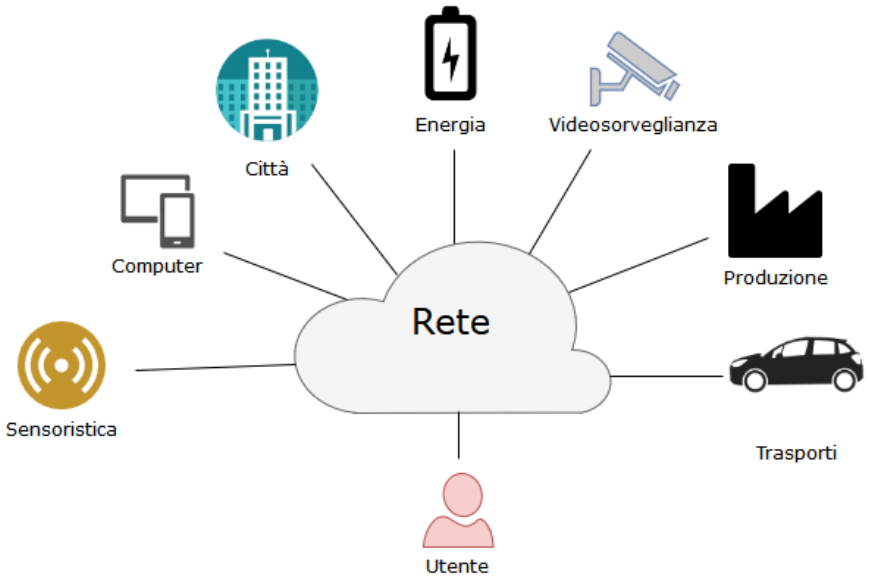


Figura 1.1: Rappresentazione figurata del concetto di *"internet of things"*.

Posso sintetizzare il concetto di *Internet of Things* come quello di 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 supportano tecnologie esistenti, integrandosi nel flusso di informazioni esistente;
- se i dispositivi comunicano in modo M2H, le informazioni aiutano le 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 ([6]) diventerà nel prossimo quinquennio uno 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

[6] 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.

query di ricerca, valutato da 0 (popolarità minima) a 100 (popolarità massima), e nelle ascisse l'arco temporale in analisi.

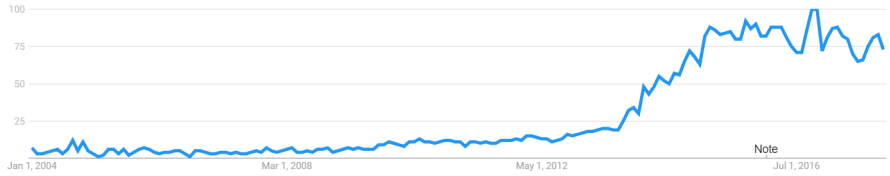


Figura 1.2: Andamento dell'interesse per la stringa di ricerca *"internet of things"*.
URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=internet%20of%20things

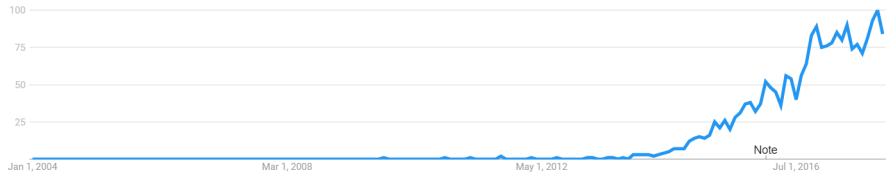


Figura 1.3: Andamento dell'interesse per la stringa di ricerca *"iot devices"*.
URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=iot%20devices

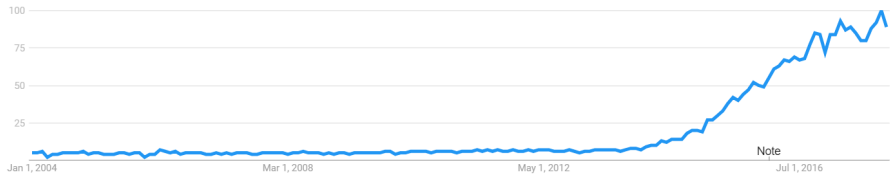


Figura 1.4: Andamento dell'interesse per la stringa di ricerca *"iot"*.
URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=iot

Sintetizzando le previsioni di Forbes con l'andamento dei termini di ricerca legati all'IoT, visibili alle figure 1.2, 1.3 e 1.4, posso evidenziare che l'interesse verso l'argomento IoT stia generalmente aumentando o nel caso peggiore rimanga stabile con l'interesse degli anni precedenti.

1.3 Architettura a microservizi: definizione e caratteristiche

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, che comunicano tra loro grazie a meccanismi di comunicazione "leggeri" (solitamente attraverso Application Program Interface (API)[g] HTTP). Nella concezione originale in cui l'architettura a microservizi è nata, ogni servizio doveva essere progettato per eseguire in un processo indipendente dagli altri; con la nascita e la diffusione dei *container* questo paradigma sta cambiando, associando sempre più l'esecuzione dei microservizi in altrettanti *container*. La *containerization* (containerizzazione) è un metodo di virtualizzazione posto al livello del sistema operativo per la distribuzione ed esecuzione di applicazioni all'interno di *container*.[7] Un *container* è un unità *software* standardizzata, distribuibile in un unico pacchetto composto da:

- l'applicazione da eseguire;
- l'ambiente d'esecuzione configurato correttamente per l'applicazione da eseguire, che a sua volta specifica:
  - le dipendenze dell'applicazione;
  - i file di configurazione dell'applicazione.

Una delle tecnologie di containerizzazione che si è più diffusa è Docker (`https://www.docker.com/what-docker`), sviluppata dall'omonima azienda; Docker ha reso disponibile la propria tecnologia di containerizzazione su molteplici piattaforme, sia locali (*computer* con i sistemi operativi *Windows*, *macOS* e i sistemi operativi basati su *Linux*) sia in *cloud* (con ad es. servizi come *Amazon Web Services* e *Microsoft Azure*). Per implementare il concetto di *container*, la piattaforma di Docker installa un insieme di servizi che comunicano con il Kernel[g] del sistema operativo su cui è in esecuzione (*host*) e con i quali gli utenti interagiscono per creare e gestire *container*.[8] Ho illustrato l'architettura sopra citata in figura 1.5.

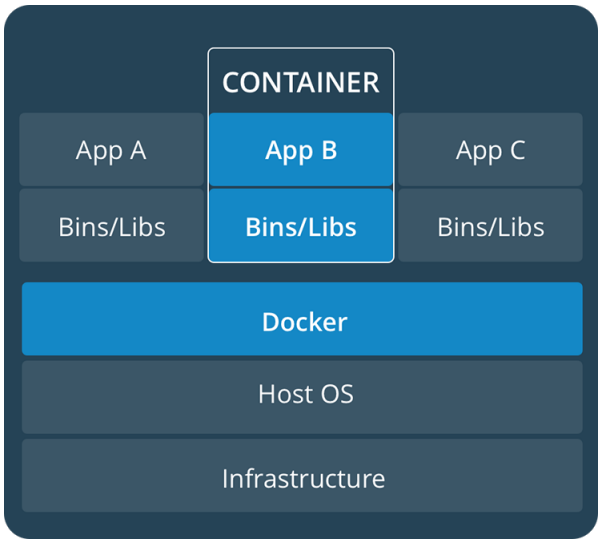


Figura 1.5: Architettura sviluppata da Docker per la sua piattaforma di containerizzazione. *What is a container*. URL: `https://www.docker.com/what-container`

[7] *Containerization*. URL: `https://en.wikipedia.org/wiki/Operating-system-level_virtualization`.

[8] *What is a container*. URL: `https://www.docker.com/what-container`.

Caratteristiche delle architetture monolitiche Un'applicazione monolitica è progettata e costruita per essere una singola unità in esecuzione. L'applicazione sviluppata con architettura monolitica è responsabile della visualizzazione delle informazioni in un'interfaccia utente (pagine web o *software* nativi), del reperimento delle informazioni da una sorgente di dati (solitamente un *database*) e dell'esecuzione delle logiche di business della stessa.[9]

Nelle applicazioni monolitiche la modularità del sistema si ottiene sfruttando i costrutti fondamentali dell'orientamento ad oggetti presente nei linguaggi di programmazione:

- funzioni;
- classi;
- *namespace* o *package*.

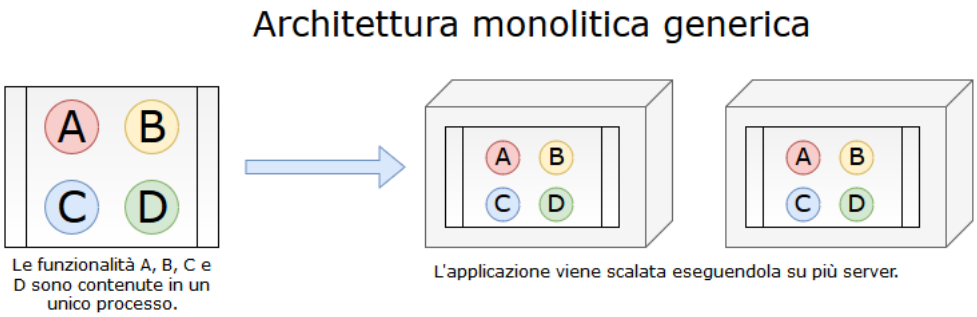


Figura 1.6: Caratteristiche di una generica architettura monolitica.
Martin Fowler. *Microservices, a definition of this new architectural term.* 2014. URL: https://martinfowler.com/articles/microservices.html

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[g]. Tra i difetti delle applicazioni monolitiche posso 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[g]).

Caratteristiche delle architetture a microservizi Per lo stile architetturale a microservizi non esistono definizioni formali, tuttavia gli informatici più esperti in materia, tra i quali annovero Martin Fowler, hanno dedotto 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.[10]

[9] Rod Stephens. *Beginning Software Engineering.* John Wiley & Sons, 2015.
[10] Martin Fowler. *Microservices, a definition of this new architectural term.* 2014. URL: https://martinfowler.com/articles/microservices.html.

L'aspetto cruciale delle architetture a microservizi verte sulla definizione di componente: la definizione comunemente accettata di componente 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 introduco 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).[11]

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.

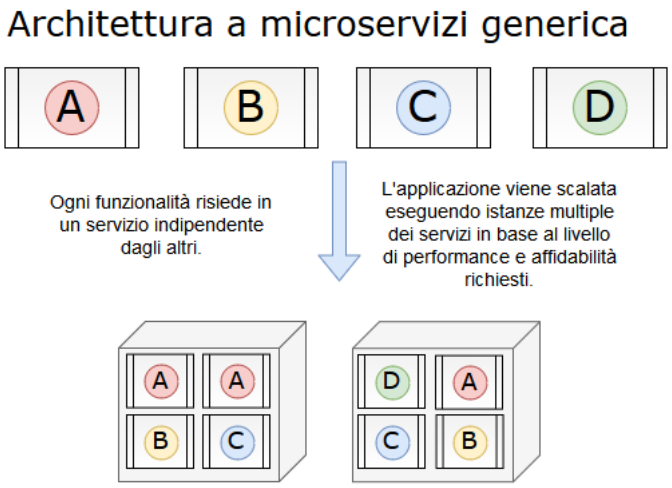


Figura 1.7: Caratteristiche di una generica architettura a microservizi.
Martin Fowler. *Microservices, a definition of this new architectural term.* 2014.
URL: https://martinfowler.com/articles/microservices.html

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 Fowler attribuisce 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.[12]

[11] Ibid.
[12] Ibid.

Nella sua trattazione, Fowler inoltre riscontra ed evidenzia le differenze 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:

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

Ho illustrato graficamente questo ambiente lavorativo in figura 1.8. L'origine di una tale suddivisione risale alla Legge di Conway, enunciata nel 1967 dallo sviluppatore Melvin Conway, la quale afferma:

> "organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."

La legge di Conway ragiona sul fatto che un sistema *software* complesso per funzionare richieda lo sforzo congiunto di più attori; dal momento che questi attori devono comunicare tra loro, la complessità insita nella comunicazione tra gli attori si riflette nelle componenti del sistema.[13]

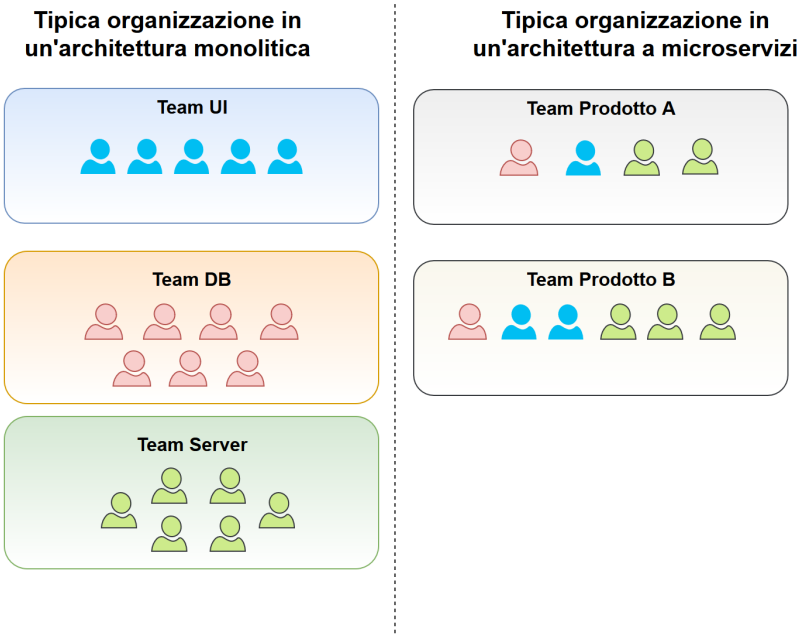


Figura 1.8: Illustrazione che mostra la differente organizzazione aziendale in un ambiente di sviluppo monolitico e in un ambiente di sviluppo a microservizi.
Martin Fowler. *Microservices, a definition of this new architectural term.* 2014. URL: https://martinfowler.com/articles/microservices.html

Quando le persone sono così isolate, anche una semplice modifica può richiedere l'intervento di altre persone in *team* diversi. La maggiore richiesta di pianificazione e organizzazione tra gruppi di sviluppo diversi causa un peggioramento dell'efficienza del processo di sviluppo.

[13] Melvin Conway. *Conway's Law.* URL: http://www.melconway.com/Home/Conways_Law.html.

L'approccio orientato ai microservizi con la 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.[14]

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.[15]

Le comunicazioni tra servizi sono orchestrate usando semplici protocolli basati su REST. REST, acronimo di REpresentational State Transfer, è un tipo di architettura *software* per lo sviluppo di applicazioni distribuite, introdotta nel 2000 nella tesi di dottorato di Roy Fielding. Le architetture basate su REST prevedono che la scalabilità delle applicazioni sia conseguenza di pochi principi di progettazione:

- separazione tra *client* e *server*: i ruoli delle due componenti sono ben distinti utilizzando un insieme di interfacce comuni per la comunicazione, permettendo quindi uno sviluppo indipendente di queste componenti (se l'interfaccia comune non viene alterata);
- *stateless*: la comunicazione *client-server* è vincolata in modo che nessuna informazione sullo stato del *client* venga memorizzata dal *server*;
- *cacheable*: ogni *client* deve poter memorizzare le risposte inviate dal *server* per minimizzare le comunicazioni *client-server*. Ogni risposta deve comunicare al *client* implicitamente o esplicitamente se essa è memorizzabile;
- *layered system*: un *client* non deve poter discernere un *server* di basso livello da uno intermedio, dedicato a migliorare le prestazioni o introdurre politiche di sicurezza;
- *uniform interface*: la comunicazione tra *client* e *server* deve avvenire con un'interfaccia omogenea, disaccoppiando le due componenti ma degradando potenzialmente l'efficienza, dal momento che le informazioni vengono trasferite in una forma standardizzata invece di una più affine alla loro struttura.

[16][17]

L'esperienza di Fowler mostra come siano due le tipologie di protocolli più usati nelle architetture a microservizi:

- protocolli basati su richieste/risposte HTTP secondo API ben dettagliate;
- protocolli basati sullo scambio di messaggi 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 gli analisti astraggono i domini dell'applicazione 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 per

[14] Fowler, *Microservices, a definition of this new architectural term*.

[15] *A Conversation with Werner Vogels - Learning from the Amazon technology platform*. 2006. URL: https://queue.acm.org/detail.cfm?id=1142065.

[16] *REpresentational State Transfer*. URL: https://en.wikipedia.org/wiki/Representational_state_transfer.

[17] Roy Fielding. «Representational State Transfer (REST)». in: (). URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

ogni singolo servizio, utilizzando le tecniche ritenute più appropriate. Questa decentralizzazione dell'astrazione dei modelli si riflette anche sulla possibilità di decentralizzare le decisioni relative a quale *storage* dei dati utilizzare per ciascun servizio. Nell'architettura a microservizi si preferisce che ogni servizio gestisca il proprio *database* in base ai requisiti che il servizio deve soddisfare: il database di un servizio potrebbe essere un'istanza di una stessa piattaforma tecnologica, una piattaforma specifica e ottimizzata per il caso d'uso del servizio oppure potrebbe non essere utilizzato (servizi puramente funzionali). Questo approccio alla gestione della persistenza è chiamato *Polyglot Persistence* ed è utilizzabile anche in architetture monolitiche, malgrado appaia con maggior frequenza in architetture a microservizi.[18]

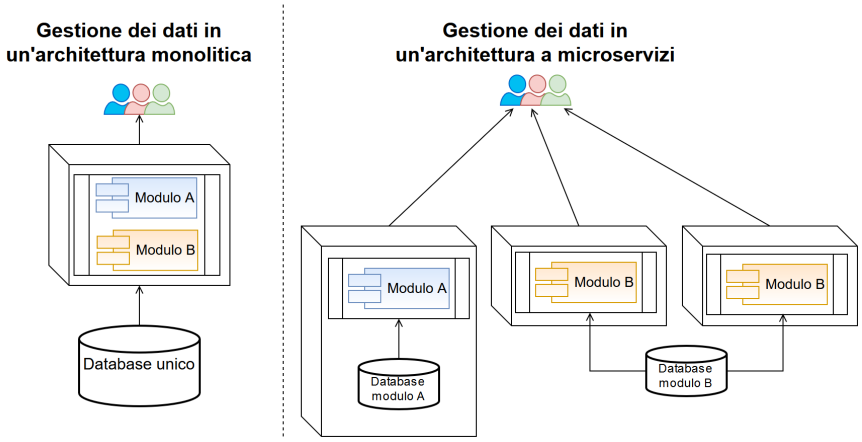


Figura 1.9: Illustrazione che mostra la differente gestione dell'architettura di persistenza dei dati tra prodotti software con architettura monolitica e con architettura a microservizi.
Martin Fowler. *Microservices, a definition of this new architectural term.* 2014. URL: https://martinfowler.com/articles/microservices.html

Le decisioni di *storage* decentralizzate implicano una maggior attenzione verso gli aggiornamenti dei dati. Fowler rileva che 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.[19]

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. É naturale quindi attribuire l'aumento di complessità della gestione dei malfunzionamenti tra i difetti delle architetture a microservizi. Dal momento che i servizi possono malfungere in ogni momento, è fondamentale riuscire a:

- monitorare il servizio,

[18] Fowler, *Microservices, a definition of this new architectural term.*
[19] Ibid.

- 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.).

Uno dei temi cruciali che ho riscontrato nella trattazione di Fowler consiste nel chiedersi quante responsabilità debba avere ciascun servizio: secondo Fowler 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 alle funzionalità offerte). Se due o più servizi vengono aggiornati spesso insieme probabilmente essi dovrebbero essere uniti.[20]

[20] Ibid.

1.4 Valutazione dei rischi dello svolgimento dello stage

1.4.1 Valutazione dei rischi di uno stage interno

Lo stage formativo viene previsto dal CdL triennale di Informatica in due modalità:

- stage aziendali, riferiti anche come stage esterni, per i quali il proponente del progetto di stage è un'azienda;
- stage non aziendali, riferiti anche come stage interni individuali, per i quali il proponente del progetto di stage è un docente dell'Ateneo di Padova.

Sebbene lo stage aziendale sia la modalità preferita per svolgere l'attività, mi sono imbattuto in due ostacoli che mi hanno portato ad avviare uno stage interno.
Il mio stato di studente lavoratore causa il primo ostacolo: l'azienda per cui sono assunto non poteva offrire stage riguardanti il settore IoT. Nel momento in cui ho iniziato a cercare proposte di stage e mi sono quindi informato sulle modalità con cui avrei potuto assentarmi da lavoro, l'ufficio per le risorse umane dell'azienda per cui sono assunto mi ha comunicato che avrei avuto opzioni limitate, dipendenti dal motivo dell'assenza.
Se l'assenza avesse comportato l'inizio di attività lavorative per altre aziende (concorrenti oppure non concorrenti) sarei stato costretto a scegliere tra due opzioni:

- il licenziamento dall'azienda per cui sono assunto;
- l'avvio della procedura di aspettativa[g] del lavoro come prevista dalla legge 53 recante data 8 marzo 2000.[21].

Quindi fin dall'inizio la ricerca di un progetto di stage aziendale è stata fortemente messa in discussione dalle menzionate opzioni.
Malgrado il primo ostacolo, ho proseguito la ricerca dello stage aziendale al fine di ponderare se gli aspetti negativi legati al congedo lavorativo potessero essere in qualche modo bilanciati da eventuali esperienze formative.
Dato il periodo di inizio dello stage (ottobre/novembre 2017), molti degli stage aziendali riguardanti il settore IoT proposti erano già stati svolti da altri studenti del corso. Le poche proposte di stage aziendale legate al settore IoT rimanenti erano interessate alla integrazione con prodotti già esistenti: malgrado l'opportunità offerta da queste aziende fosse interessante, ho riflettuto a lungo sul fatto che l'attività formativa non fosse adeguatamente supportata.
Nei progetti proposti infatti le aziende proponenti hanno enfatizzato l'utilizzo degli strumenti interni da loro sviluppati, da una parte per motivarmi ad essere assunto al termine dello stage, dall'altra per effettivamente semplificarmi il lavoro nel processo di sviluppo.

L'aspetto formativo è stato l'ambito più discusso per effettuare la scelta: malgrado la presenza di esperti nel settore mi interessasse, per l'opportunità di approfondire l'ambito d'uso reale dei dispositivi *smart*, la scelta di legarmi a un singolo prodotto, non particolarmente diffuso e utilizzato, mi ha spinto ad iniziare a valutare il percorso di stage interno.

Ho riassunto i rischi considerati per lo svolgimento di uno stage interno individuale nella tabella 1.1.

[21] *Disposizioni per il sostegno della maternità e della paternità, per il diritto alla cura e alla formazione e per il coordinamento dei tempi delle città*. 2000. URL: http://www.camera.it/parlam/leggi/000531.htm.

Tabella 1.1: Tabella di analisi dei rischi correlati allo svolgimento di uno stage interno

Rischio	**Probabilità di accadimento**	**Gravità del danno potenziale**
Il lavoro svolto individualmente, senza possibilità di essere guidato da esperti nel settore, potrebbe risultare ininfluente. La presenza di una persona con più esperienza in un determinato tema è utile in quanto può focalizzare l'attenzione su problematiche reali, le quali potrebbero non essere correttamente valutate nel momento in cui si dispone di scarsa esperienza in materia.	Molto probabile	Grave
Il lavoro svolto individualmente, senza possibilità di essere guidato da esperti nel settore, potrebbe procedere più lentamente, causando il non raggiungimento degli obiettivi preposti.	Molto probabile	Media

1.4.2 Valutazione dei rischi del tema IoT

Nuove tecnologie contengono sempre una determinata quantità di rischi: mentre la maggior parte degli sviluppatori trovano utilizzi per i dispositivi IoT, altri cercano modi per usarli per scopi meno nobili.
I dispositivi IoT stanno sempre più diffondendosi in molti aspetti su cui basiamo la società moderna:

- trasporti;
- comunicazione;
- settore energetico.

Attacchi informatici contro questi dispositivi possono portare al caos: dalla distruzione di proprietà alla messa in discussione della propria sicurezza, con l'accezione che il termine *safety* possiede nella lingua inglese. A peggiorare la situazione, gli acquirenti di questi dispositivi pretendono che essi continuino a funzionare per una quantità di tempo superiore a quella che il consumismo tecnologico ha abituato.
La quantità di protocolli sviluppati per l'IoT porta ad aumentare la complessità insita in questi dispositivi. Una complessità maggiore implica un maggior costo per le società per aggiornare i prodotti rilasciati, portando quindi i produttori ad abbandonare i dispositivi rilasciati da più tempo, ignorando l'insorgere di nuove vulnerabilità.

Date le suddette premesse, per il settore IoT ho posto numerose osservazioni relative ai rischi collegati allo sviluppo di prodotti software e hardware. I rischi considerati sono riassunti nella tabella 1.2.

Tabella 1.2: Tabella di analisi dei rischi correlati al tema IoT

Rischio	**Probabilità di accadimento**	**Gravità del danno potenziale**
Lo sviluppo di prodotti legati all'IoT espone l'utente degli stessi a possibili rischi di cybersicurezza: il furto di dati ma soprattutto la perdita di controllo dell'utente sui propri dispositivi sono scenari possibili e con conseguenze disastrose per lo sviluppatore di tale prodotto.	Probabile	Molto grave
La mole di dati raccolta dai dispositivi inseriti in un contesto IoT potrebbe essere tale da richiedere investimenti consistenti per la loro elaborazione e per mantenere elevata la loro confidenzialità. Inoltre la diversità dei dispositivi presenti in una rete aumenta la complessità del trattamento delle informazioni.	Probabile	Media
Dal momento che l'IoT è un ambito emergente nel contesto ITC ho osservato la nascita di una multitudine di protocolli per la raccolta e la trasmissione delle informazioni, nessuno dei quali è stato indicato o si è imposto come standard globale.	Molto probabile	Media

1.4.3 Valutazione dei rischi dell'architettura a microservizi

L'architettura a microservizi permette di sviluppare un'applicazione complessa partendo da piccole e relativamente isolate componenti; in questo modo i cambiamenti effettuati sono facilmente verificabili. La frammentazione dell'architettura del prodotto rende tuttavia più complesse le attività di test funzionali, perchè per eseguire un tipico caso d'uso di una funzionalità completa sono richiesti molteplici servizi, correttamente configurati per comunicare tra loro ed eseguire simultaneamente.

Lo sviluppo di servizi indipendenti rende inoltre difficoltosa l'attività di integrazione delle modifiche: nel momento in cui molti *team* di sviluppo lavorano ciascuno nel proprio servizio non è possibile integrare queste modifiche senza una attenta pianificazione, che coinvolga la comunicazione dei cambiamenti effettuati.
Nel mio caso questo non si applica, lavorando individualmente, tuttavia mi sono comunque richieste le attività di allineamento delle funzionalità tra i servizi che necessitano di comunicare tra loro.

Anche in questo caso, la relativa novità dell'argomento causa una generale mancanza di documentazione pratica, lasciando lo spazio ad esempi semplici, che non riflettono applicazioni d'uso reale, oppure documentazione teorica, che non si spinge ad analizzare problematiche reali. La tabella 1.3 riassume e sintetizza i rischi analizzati in precedenza.

Tabella 1.3: Tabella di analisi dei rischi correlati all'utilizzo dell'architettura a microservizi

Rischio	**Probabilità di accadimento**	**Gravità del danno potenziale**
Spostamento di alcune problematiche di progettazione da un livello di modulo a un livello di architettura del sistema.	Probabile	Media
Le performance dell'applicazione sviluppata potrebbero non essere sufficienti passando da un'architettura monolitica a una a microservizi.	Scarsamente probabile	Grave
Difficoltà nel reperimento delle informazioni, data la relativa novità dell'argomento. Assume maggior significato nel momento in cui l'esperienza con un tale paradigma risulti scarsa o nulla.	Molto probabile	Grave

Capitolo 2

Progetto di stage

2.1 Descrizione generale

Lo stage interno è una forma di stage individuale in cui uno studente, in concerto con un docente, redige un piano delle attività da svolgere nell'intervallo di tempo specificato. Nel presente progetto di stage ho iniziato le attività di stage in data 6/11/2017 e le ho terminate in data 2/1/2018, con un monte ore totale di circa 312 ore.
In questo periodo di svolgimento dello stage ho definito, con l'assistenza del Prof. Tullio Vardanega, gli obiettivi dello stage dai seguenti punti di vista:

- obiettivi di prodotto: questi obiettivi coincidono con le caratteristiche e le funzionalità importanti per gli utenti del prodotto;
- obiettivi formativi: questi obiettivi comprendono le conoscenze che mi aspetto di acquisire e le abilità mi aspetto di apprendere durante lo svolgimento dello stage.
- obiettivi tecnici: questi obiettivi comprendono le tecniche e le tecnologie specifiche che mi aspetto di dover padroneggiare al fine di completare il progetto di stage.

L'ambito su cui ho focalizzato l'attenzione dello stage include dispositivi per l'automazione domestica dedicati alla gestione dell'illuminazione e alla gestione termica dell'abitazione.
I dispositivi che ho considerato per l'illuminazione domestica consistono solamente in varianti di un solo dispositivo, la lampada *smart*.
I dispositivi considerati invece per la gestione termica dell'abitazione sono sostanzialmente due:

- sensori, i quali devono inviare informazioni relative alla temperatura di un determinato ambiente;
- termostati, i quali devono interfacciarsi con i sensori da una parte, per ricevere e sintetizzare la distribuzione termica dell'abitazione, e con l'impianto di riscaldamento dall'altra, per applicare le variazioni di temperatura richieste dall'utente.

2.2 Obiettivi di prodotto

L'obiettivo principale del prodotto è quello di fornire un'interfaccia unificata per la gestione dei dispositivi connessi, consentendo all'utente l'accesso all'interfaccia proprietaria di ciascun dispositivo.

Come riporta il titolo della presente relazione, per il progetto di stage ho preferito concentrarmi su funzionalità per certi aspetti innovative restando nell'ambito di sviluppo di un prototipo; come tale, il prototipo non è una soluzione pronta alla distribuzione sul mercato (*production-ready*), quanto un modo per sperimentare con l'automazione domestica introducendo funzionalità non diffuse nei prodotti presenti sul mercato.

Le funzionalità chiave che il prodotto sviluppato vuole offrire all'utente sono riportate in tabella 2.1, che illustra l'identificativo assegnato all'obiettivo, una descrizione dell'obiettivo e la sua importanza, valutata in una scala di importanza crescente da 1 a 5, in cui 1 indica l'importanza minima e 5 l'importanza massima.

Tabella 2.1: Tabella delle funzionalità offerte dal prototipo all'utente

Id obiettivo	**Descrizione obiettivo**	**Importanza funzionalità**
OP1	Visualizzare lo stato generale del sistema.	4
OP2	Visualizzare quali dispositivi sono collegati al sistema.	5
OP3	Visualizzare le informazioni trasmesse dai dispositivi collegati al sistema.	5
OP4	Implementare sistemi di autenticazione dell'utente.	2
OP5	Visualizzare e gestire le preferenze dell'utente.	3

L'obiettivo "OP1" consente all'utente di verificare l'operatività del sistema, mostrando gli eventuali malfunzionamenti che causano un disservizio alla *dashboard*; per questo credo sia una funzionalità relativamente importante per l'utente, che potrebbe voler conoscere se un disservizio della *dashboard* è legato ad un malfunzionamento del sistema sottostante e non ad un errore dell'interfaccia.

Gli obiettivi "OP2" e "OP3" costituiscono funzionalità fondamentali per l'utilizzo della *dashboard* perchè permettono all'utente di conoscere quanti e quali dispositivi sono correttamente riconosciuti dal sistema e per ciascun dispositivo le informazioni messe a disposizione dallo stesso.

Malgrado l'obiettivo "OP4" sia fondamentale per un prodotto distribuibile al pubblico, ho assegnato una importanza medio-bassa a questo obiettivo: data la natura di prototipo del prodotto e considerato che il tema sicurezza richiede una elevata formazione per la sua corretta implementazione, ho preferito dare priorità agli obiettivi riguardanti alle funzionalità che possono dare un valore di innovazione aggiunto rispetto alle soluzioni presenti sul mercato.

L'obiettivo "OP5" aggiunge una componente di personalizzazione alla *dashbaord* che potrebbe essere utile all'utente: l'esempio a cui ho fatto riferimento riguarda la scelta delle unità di misura predefinite con cui il sistema visualizza le informazioni.

2.3 Obiettivi formativi

Per analizzare al meglio gli obiettivi formativi del presente stage, ho deciso di suddividere gli obiettivi formativi legati all'ambito IoT da quelli relativi alle architetture a microservizi.
Dal punto di vista del tema IoT, ho concentrato l'attenzione su due aspetti essenziali:

1. l'analisi e la definizione delle caratteristiche e delle funzionalità di cui sono dotati i dispositivi IoT esistenti;
2. la ricerca e l'implementazione di un protocollo di comunicazione le cui qualità siano adeguate al contesto di utilizzo.

L'aspetto citato in 1 risulta fondamentale per apprendere, in aggiunta alle abilità di analisi imparate durante il corso di studi, abilità specifiche nella comprensione delle funzionalità e caratteristiche richieste dal mercato presente dei dispositivi IoT.
L'aspetto citato in 2 mi permette di acquisire conoscenze specifiche relative a protocolli di comunicazione non approfonditi durante il corso di Reti e Sicurezza del percorso di studi del CdL di Informatica. Il mio obiettivo in questo frangente è analizzare e studiare i protocolli di comunicazione esistenti, utilizzabili liberamente e le cui specifiche siano accessibili pubblicamente e gratuitamente.
Gli obiettivi precedentemente citati corrispondono agli obiettivi "OF1" e "OF2" definiti nella tabella 2.2.
Dal punto di vista delle architetture a microservizi, data la mia totale inesperienza riguardo le architetture *software* orientate ai servizi ho posto come obiettivi formativi l'acquisizione dei concetti alla base di queste architetture, analizzandone:

1. i princìpi generali che definiscono questo insieme di architetture;
2. le tecnologie che consentono di sviluppare sistemi *software* con le caratteristiche richieste da queste architetture;
3. pregi e difetti delle architetture orientate ai servizi, dando particolare risalto ai pregi e difetti delle architetture a microservizi.

Mentre le voci 1 e 3 corrispondono all'obiettivo "OF3" riportato in tabella 2.2, la voce 2 è una generalizzazione dell'obiettivo "OF4" (riportato nella stessa tabella citata precedentemente). Tra i princìpi delle architetture orientate ai servizi e ai microservizi, uno degli aspetti su cui mi sono concentrato con maggior attenzione è l'analisi della corretta dimensione di un servizio tale per cui esso possa essere definito "*micro*". Un altro concetto importante su cui ho dovuto informarmi con attenzione consiste nella scelta delle tecnologie di persistenza dei dati per ciascun servizio, specialmente in relazione alla già citata *Polyglot persistence* (riferimento 1.3).

Tabella 2.2: Tabella degli obiettivi formativi del progetto

Id obiettivo	Descrizione obiettivo
OF1	Apprendere abilità elementari per la comprensione delle funzionalità richieste dal mercato IoT, specialmente nel campo della automazione domestica.
OF2	Acquisire conoscenze adeguate alla scelta e implementazione di un protocollo di comunicazione adeguato al campo di utilizzo del progetto.
OF3	Comprendere il concetto di architettura a microservizi, con i pregi e i difetti caratteristici di una tale architettura.
OF4	Acquisire le nozioni legate alla containerizzazione di un sistema *software* in un contesto architetturale basato su microservizi.

2.4 Obiettivi tecnici

Sin dall'inizio delle attività di stage è stata mia intenzione distribuire i prodotti di queste attività in modo che chiunque potesse consultarli liberamente e pubblicamente: per offrire questa possibilità il primo obiettivo tecnico del progetto consiste nell'adozione di una licenza di distribuzione permissiva, che permetta:

- la visualizzazione,
- l'utilizzazione e
- la modifica

del codice sorgente e della documentazione associata senza vincoli legali. Questo obiettivo corrisponde all'obiettivo "OT1" indicato in tabella 2.3.
Collegato all'obiettivo precedente, il secondo obiettivo tecnico consiste nella pubblicazione delle istruzioni per facilitare l'esecuzione del prototipo in una macchina di sviluppo locale. Dal momento che il prototipo è testabile da un pubblico potenzialmente vasto, questo secondo obiettivo tecnico assicura che il sistema sviluppato possa essere eseguito in maniera ripetibile, semplificando la ricerca e la segnalazione di malfunzionamenti e permettendo a chiunque di valutare le idee sviluppate nel prototipo. Questo obiettivo corrisponde all'obiettivo "OT2" indicato in tabella 2.3.
L'obiettivo "OT3" indicato in tabella 2.3 si riferisce alla possibilità di far funzionare il prototipo in un ambiente realistico: in questo ambiente realistico vi sono dispositivi, con cui l'utente può interagire, che trasmettono le informazioni raccolte a dispositivi in grado di elaborare queste informazioni e metterle a disposizione degli altri dispositivi in maniera strutturata. Dal momento che nella rete questi dispositivi devono poter identificarsi, con l'obiettivo "OT3" evidenzio le caratteristiche di modularità e configurabilità che il prototipo deve possedere.
In maniera complementare a quanto appena detto, il prototipo deve poter essere eseguito in un unico dispositivo che sia in grado di simulare l'esecuzione nell'ambiente realistico precedentemente citato. Questo obiettivo, indicato come "OT4" in tabella 2.3, è fortemente collegato agli obiettivi "OT1" e "OT2", perchè non è assicurato che gli utenti che desiderano provare il prototipo posseggano un insieme di dispositivi che possa eseguire tutte le componenti che formano la *dashboard*.
Da un punto di vista tecnico, gli obiettivi "OT3" e "OT4" vincolano la scelta del protocollo di comunicazione da implementare, perchè è necessario che tale protocollo sia applicabile sia in ambito di esecuzione in un ambiente reale, sia nell'ambito di simulazione, nel quale non ci sono comunicazioni all'esterno della macchina nella quale esegue il prototipo.
L'obiettivo "OT5" in tabella 2.3 si riferisce alla possibilità di conoscere quali componenti del sistema siano in esecuzione e cambiare lo stato delle componenti al fine di aumentare o diminuire la disponibilità di una componente, fino alla completa disattivazione della stessa. Evidenzio la correlazione tra questo obiettivo ("OT5") con l'obiettivo "OF4": l'acquisizione corretta delle conoscenze delle tecnologie di containerizzazione dovrebbe semplificare il soddisfacimento dell'obiettivo "OT5", dato il contesto attinente con cui si sono sviluppate le tecnologie di containerizzazione.

Gli obiettivi tecnici "OT6" e "OT7", indicati in tabella 2.3, impostano dei vincoli tecnologici per l'implementazione del prototipo.
L'obiettivo "OT6" è strettamente correlato con l'obiettivo "OT3", il quale richiede l'utilizzo di tecnologie multipiattaforma per la corretta esecuzione del prototipo. Ho scelto di vincolare lo sviluppo del prototipo adottando *Node.js* come *framework* per l'implementazione della parte *backend* per due motivi:

1. dal momento che gli argomenti trattati nello stage mi sono completamente nuovi, ho voluto appoggiarmi dal punto di vista tecnico a una tecnologia già utilizzata per l'implementazione del progetto formativo del corso di Ingegneria del Software per abbassare la quantità di argomenti nuovi trattati;
2. *Node.js* utilizza JavaScript[g] come linguaggio di programmazione, rendendo lo sviluppo delle applicazioni più veloce, grazie ad una sintassi semplice da imparare.

L'obiettivo "OT7" è il risultato di un altro vincolo tecnologico che ho imposto con lo scopo di semplificare lo sviluppo dell'interfaccia grafica dell'applicazione *web* grazie all'esperienza già acquisita a riguardo con il progetto formativo del corso di Ingegneria del Software.

Tabella 2.3: Tabella degli obiettivi tecnici del progetto

Id obiettivo	Descrizione obiettivo
OT1	Rilascio del codice sorgente del prototipo e della documentazione associata nei termini di una licenza *open source*.
OT2	La documentazione associata al progetto deve includere le istruzioni necessarie all'esecuzione del prototipo.
OT3	Il prototipo deve essere eseguibile su dispositivi presenti in una rete, previa corretta configurazione.
OT4	Il prototipo deve essere eseguibile su un dispositivo di test, che simuli l'esecuzione in un ambiente reale.
OT5	Il prototipo deve prevedere strumenti per gestire la scalabilità del sistema e per monitorarne lo stato.
OT6	Il prototipo deve essere implementato in Node.js (`https://nodejs.org/en/about/`) per il lato server.
OT7	L'interfaccia utente del prototipo deve essere implementata in React (`https://reactjs.org/`).

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

Sistemi operativi, dispositivi e tecnologie utilizzate.

3.2.1 Strumenti di sviluppo

3.3 Analisi dei Requisiti

Approfondire l'Analisi dei Requisiti prodotta. + Analisi dei protocolli

3.4 Progettazione

Approfondire la Specifica Tecnica prodotta.

3.5 Documentazione

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

3.6 Test

Come sono stati svolti i test? Coverage?

3.7 Validazione dei Requisiti

Quali requisiti sono stati rispettati e quali invece sono stati abbandonati.

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

Per ogni obiettivo definito precedentemente valuterei il suo soddisfacimento e descriverei le problematiche rilevate durante lo svolgimento dello stage.

4.2 Conoscenze acquisite

Autovalutazione delle conoscenze acquisite, ragionando in termini di aspettative iniziali e menzionando le parti che hanno causato difficoltà nello svolgimento del progetto.

4.3 Conclusioni

Glossario

API[g] 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. 27

Aspettativa è un periodo di astensione dal lavoro, previsto dalla legge, che il datore di lavoro può concedere ad un proprio lavoratore per motivi familiari o personali, generalmente non retribuito. 12

JavaScript è un linguaggio di programmazione nato per aggiungere dinamicità e interattività alle pagine renderizzate dai browser attraverso l'esecuzione di semplici istruzioni che manipolino la struttura della pagina *web* visualizzata. 21

Kernel è il sottosistema di un sistema operativo che fornisce ai processi in esecuzione sull'elaboratore l'accesso alle risorse fisiche, in modo controllato e sicuro. 5

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*). 6

Throughput indica la capacità di un canale di comunicazione di processare o trasmettere dati in uno specifico periodo di tempo. É una misura di produttività.. 6

Acronimi

API Application Program Interface[g]. 5, 25

Bibliografia

Riferimenti bibliografici

Stephens, Rod. *Beginning Software Engineering*. John Wiley & Sons, 2015 (cit. a p. 6).

Siti web consultati

URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=internet%20of%20things (cit. a p. 4).

URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=iot%20devices (cit. a p. 4).

URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=iot (cit. a p. 4).

A Conversation with Werner Vogels - Learning from the Amazon technology platform. 2006. URL: https://queue.acm.org/detail.cfm?id=1142065 (cit. a p. 9).

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. 3).

Containerization. URL: https://en.wikipedia.org/wiki/Operating-system-level_virtualization (cit. a p. 5).

Conway, Melvin. *Conway's Law*. URL: http://www.melconway.com/Home/Conways_Law.html (cit. a p. 8).