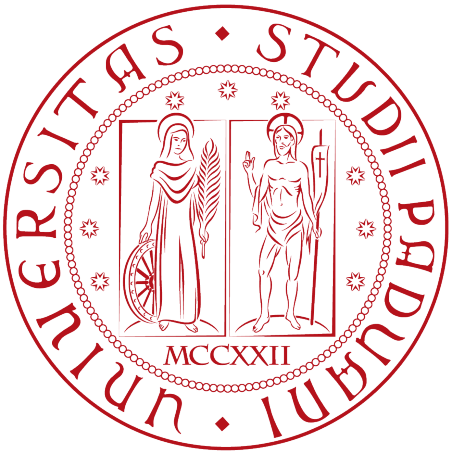
Università degli Studi di Padova

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:

Il primo capitolo presenta i temi su cui lo stage è stato svolto, elencando i rischi valutati per le scelte intraprese.

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

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

Il quarto capitolo fornisce una valutazione degli obiettivi raggiunti, motivando la presenza di eventuali obiettivi non raggiunti; inoltre vengono esaminate le conoscenze acquisite e i rischi descritti nel primo capitolo.

Convenzioni tipografiche

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*.

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

1.1.1 IoT

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. 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.

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]

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.[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

[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.

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

Il termine *"Internet of Things"* fu coniato da Kevin Ashton 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 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.).[5]

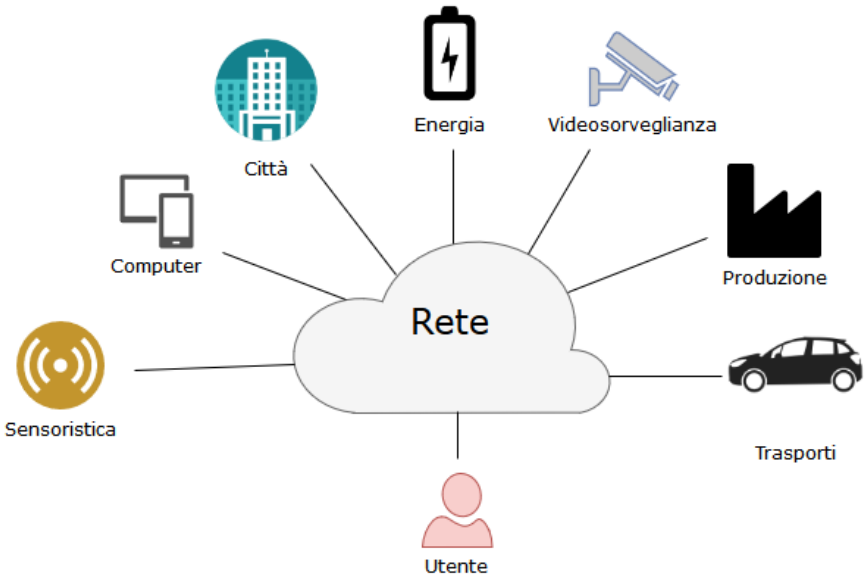


Figura 1.1: Rappresentazione figurata del concetto di *"internet of things"*.

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:

[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.

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

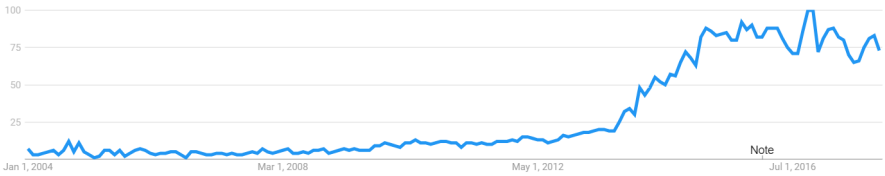


Figura 1.2: Andamento dell'interesse per la stringa di ricerca *"internet of things"*.
URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=internet%20of%20things

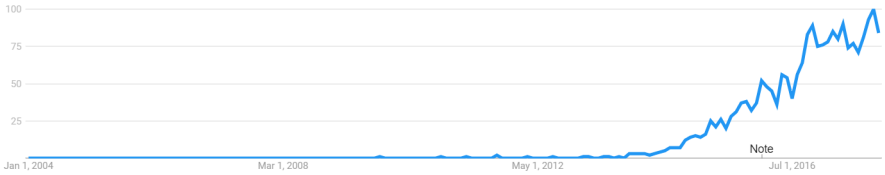


Figura 1.3: Andamento dell'interesse per la stringa di ricerca *"iot devices"*.
URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=iot%20devices

[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.

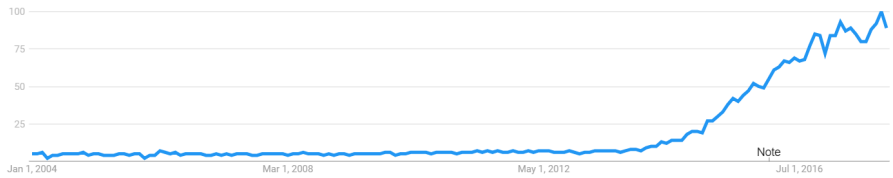


Figura 1.4: Andamento dell'interesse per la stringa di ricerca *"iot"*.
URL: https://trends.google.com/trends/explore?date=2004-01-01%202017-12-31&q=iot

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, 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, 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, che è resa disponibile 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 è possibile interagire a livello utente per creare e gestire *container*.[8] L'architettura sopra citata è illustrata in figura 1.5.

[7] *Containerization*. URL: https://en.wikipedia.org/wiki/Operating-system-level_virtualization.
[8] *What is a container*. URL: https://www.docker.com/what-container.

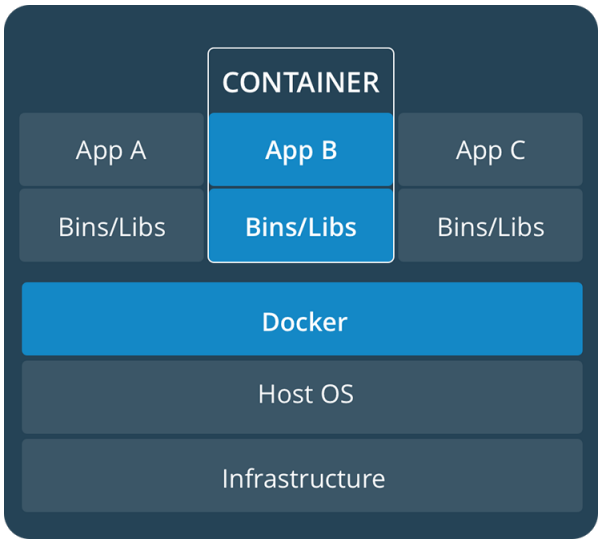


Figura 1.5: Architettura sviluppata da Docker per la sua piattaforma di containerizzazione. *What is a container*. URL: https://www.docker.com/what-container

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.

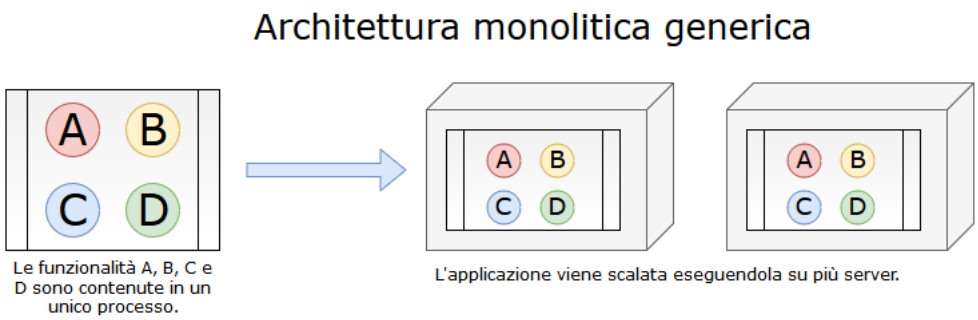


Figura 1.6: Caratteristiche di una generica architettura monolitica. Martin Fowler. *Microservices, a definition of this new architectural term*. 2014. URL: https://martinfowler.com/articles/microservices.html

[9] Rod Stephens. *Beginning Software Engineering*. John Wiley & Sons, 2015.

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 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[g]).

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.[10]

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).[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.

[10] Martin Fowler. *Microservices, a definition of this new architectural term*. 2014. URL: https://martinfowler.com/articles/microservices.html.

[11] Ibid.

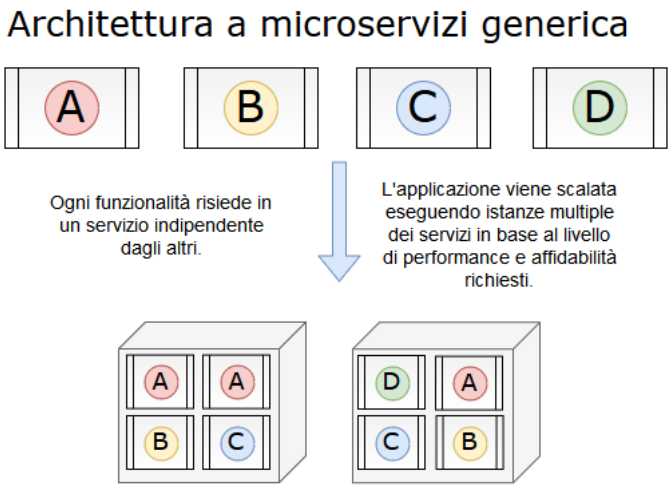


Figura 1.7: Caratteristiche di una generica architettura a microservizi.
Martin Fowler. *Microservices, a definition of this new architectural term.* 2014. URL: https://martinfowler.com/articles/microservices.html

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 si 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]

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:

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

Questo ambiente lavorativo è illustrato graficamente 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]

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

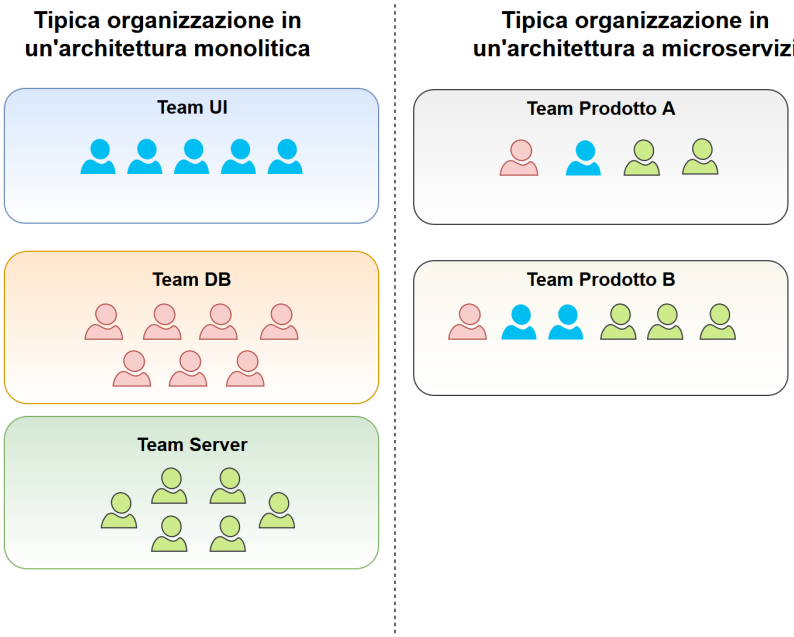


Figura 1.8: Illustrazione che mostra la differente organizzazione aziendale in un ambiente di sviluppo monolitico e in un ambiente di sviluppo a microservizi.
Martin Fowler. *Microservices, a definition of this new architectural term.* 2014. URL: https://martinfowler.com/articles/microservices.html

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.

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]

Applicazioni assemblate con microservizi mirano ad essere più disaccoppiate e più coese possibile: ricevendo una richiesta, applicando la propria logica e producendo una risposta.

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

[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.

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]

I due protocolli più usati nelle architetture a microservizi 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 per 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]

[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.

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

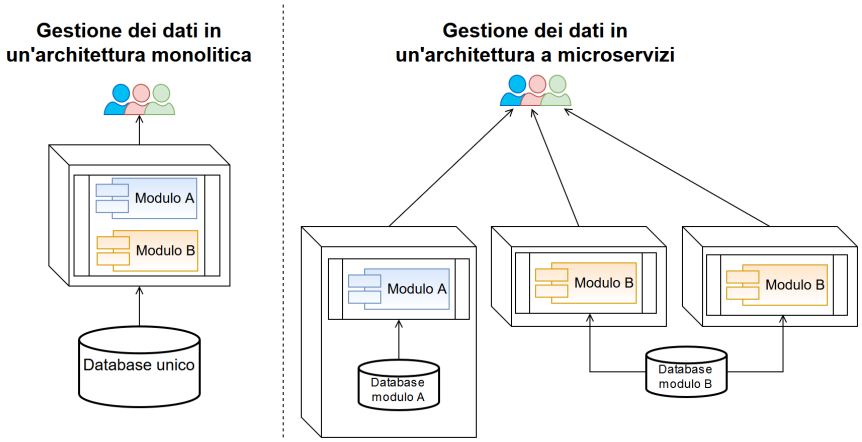


Figura 1.9: Illustrazione che mostra la differente gestione dell'architettura di persistenza dei dati tra prodotti software con architettura monolitica e con architettura a microservizi.
Martin Fowler. *Microservices, a definition of this new architectural term.* 2014. URL: https://martinfowler.com/articles/microservices.html

Le decisioni di *storage* decentralizzate implicano una maggior attenzione verso gli aggiornamenti dei dati. 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. Dato l'aumento di complessità, questo è da attribuire 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 due o più servizi vengono aggiornati spesso insieme probabilmente essi dovrebbero essere uniti.[20]

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

1.2 Valutazione dei rischi dello svolgimento dello stage

1.2.1 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 primo ostacolo è causato dal mio stato di studente lavoratore: 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.

1.2.2 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

[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

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, il settore IoT si pone a numerose osservazioni relative ai rischi collegati allo sviluppo di prodotti software e hardware. I rischi considerati sono riassunti nella tabella 1.2.

1.2.3 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. Anche in questo caso, la relativa novità dell'argomento porta a una generale mancanza di documentazione pratica, lasciando

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 è possibile osservare la nascita di una multitudine di protocolli per la raccolta e la trasmissione delle informazioni, nessuno dei quali è stato indicato come standard globale.	Molto probabile	Media

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.

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 di prodotto

Gli obiettivi di prodotto consistono in quelle caratteristiche e funzionalità importanti per gli utenti del sistema.

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

L'obiettivo principale del prodotto sarà quello di fornire un'interfaccia unificata per la gestione dei dispositivi connessi, consentendo all'utente l'accesso all'interfaccia proprietaria di ciascun dispositivo.
Date le limitate risorse (di tempo, di persone e di conoscenza pregressa) non mi aspetterò che il prodotto sia lo stato dell'arte in questo settore, soprattutto per quanto riguarda l'ambito della sicurezza; allo stesso modo non svilupperò un nuovo protocollo di comunicazione tra dispositivi, bensì mi limiterò a studiare le possibili scelte e a motivare la particolare scelta del protocollo che utilizzerò.

Come riporta il titolo della presente relazione, per il progetto di stage preferirò concentrarmi su funzionalità per certi aspetti innovative restando nell'ambito di sviluppo di un prototipo; come tale, il prototipo non vorrà essere 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.

Un secondo nodo cruciale su cui punterò sin dalle prime attività di sviluppo è la scalabilità della soluzione rispetto all'obiettivo principale citato precedentemente. Ho concepito l'obiettivo di scalabilità del prodotto come una possibilità di decentralizzare le funzionalità che il sistema dovrà fornire; l'idea, a mio modo di vedere innovativa, consiste nel realizzare un sistema *software* che potrà essere eseguito su dispositivi diversi e che potrà gestire funzionalità "dinamiche" per i dispositivi con cui l'utente può interagire. Un esempio che mi ha aiutato a chiarire questo obiettivo è il seguente: supponiamo di voler realizzare un sistema che gestisca l'automazione domestica per quanto riguarda l'illuminazione e per quanto riguarda la gestione termica dell'abitazione.

I dispositivi per l'illuminazione domestica consistono solamente in varianti di un solo dispositivo, la lampada.
I dispositivi per la gestione termica dell'abitazione sono invece 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.

Concepirò la scalabilità del prodotto al fine di permettere che le funzionalità di gestione dei dispositivi legati alla temperatura possano essere responsabilità di uno o più dispositivi che eseguano *software* dedicato allo scopo. Proseguendo l'esempio iniziato precedentemente, la componente del sistema legata alla temperatura potrà essere eseguita su un termostato dotato di capacità di elaborazione adeguate, oppure potrà essere eseguita su un dispositivo *ad-hoc* oppure ancora su un dispositivo utilizzato in comune con gli apparati di illuminazione senza che le due componenti entrino in conflitto. Lo stesso principio sarà applicabile alla gestione dell'illuminazione (lampada *master* con elettronica in grado di eseguire il *software* o dispositivo dedicato).

2.2 Obiettivi curricolari

Gli obiettivi curricolari sono quelle caratteristiche dimostrabili che sono rilevanti sul mercato del lavoro.

Gli obiettivi curricolari su cui punterò per l'aspetto IoT servono a dimostrare le soluzioni che svilupperò . Dal momento che il prodotto sviluppato per il progetto di stage sarà completamente *open source* e rilasciato con la licenza MIT[g], le aziende interessate ad alcuni aspetti del prodotto potranno visualizzare, migliorare ed eventualmente contribuire al codice esistente, tenendo bene in considerazione che il prodotto sarà sviluppato da un solo sviluppatore al primo approccio con il tema.

Gli obiettivi curricolari relativi al paradigma dell'architettura a microservizi invece dimostrano la mia volontà di sperimentare con approcci non convenzionali allo sviluppo di sistemi *software* che hanno particolari esigenze di scalabilità.
A mio avviso, il mercato del lavoro apprezzerà l'esperienza fatta in questo ambito grazie al fatto che la documentazione del prodotto e il codice sorgente del prodotto stesso saranno visualizzabili liberamente da tutti gli attori interessati, permettendo loro di capire secondo quali princìpi avrò sviluppato le componenti del sistema.
In particolare, le aziende interessate potranno valutare le mie effettive competenze nello sviluppo di architetture distribuite e nell'implementazione di un protocollo di comunicazione esistente.

2.3 Obiettivi formativi

Gli obiettivi formativi indicano le conoscenze e le abilità che dovrebbero essere studiate e imparate durante lo svolgimento dello stage.

Anche in questo caso, separo gli obiettivi formativi legati all'ambito IoT da quelli relativi al tema architettura a microservizi.

Dal punto di vista del tema IoT, dovrò studiare le caratteristiche essenziali e le funzionalità che i dispositivi IoT esistenti possiedono per permetterne una corretta rappresentazione all'interno del sistema. Inoltre lo studio dei protocolli di comunicazione esistenti sarà fondamentale per permettere lo sviluppo di un sistema affidabile e performante. In particolare dovrò analizzare protocolli di comunicazione esistenti, utilizzabili liberamente e le cui specifiche siano accessibili pubblicamente e gratuitamente. Anche se il prodotto sviluppato sarà un prototipo in cui la sicurezza non sarà ben implementata, studierò e valuterò l'aspetto relativo alla sicurezza dei protocolli

utilizzabili, considerando la diffusione del protocollo, le effettive vulnerabilità note e il rapporto sicurezza del protocollo rispetto alla sua facilità di utilizzo.

A mio modo di vedere, il lato formativo relativo all'architettura a microservizi sarà quello più impegnativo. Essendo alla mia prima esperienza con la progettazione di un sistema distribuito e con un paradigma architetturale relativamente nuovo, mi aspetto di dover impiegare un tempo maggiore, rispetto alla parte relativa all'IoT, per imparare i concetti che permettono lo sviluppo delle architetture a microservizi. In particolare, uno degli aspetti su cui concentrerò 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 dovrò informarmi con attenzione consiste nella scelta delle tecnologie di persistenza dei dati per ciascun servizio, specialmente in relazione alla già citata *Polyglot persistence* (riferimento 1.1.2). Mi aspetto quindi di dover imparare ad analizzare la struttura delle informazioni che ciascun dispositivo deve fornire al fine di sviluppare una componente che riesca a memorizzare ed elaborare tali informazioni in efficienza.

2.4 Obiettivi tecnici

Gli obiettivi tecnici consistono nelle tecniche e nelle tecnologie specifiche che andrò ad utilizzare durante lo svolgimento dello stage.

Il primo obiettivo tecnico su cui punterò è quello di distribuire l'insieme delle funzionalità del prodotto come *software open source*: esso sarà pubblicato con una licenza permissiva che permetterà a chi è interessato di visualizzare, modificare ed utilizzare il prodotto senza condizioni vincolanti.

Dal momento che il prodotto sarà testabile da un pubblico potenzialmente vasto, il secondo obiettivo tecnico generale consiste nella distribuzione delle istruzioni per provare il sistema prodotto in una macchina locale.

Da un punto di vista tecnico, per l'ambito IoT mi aspetto di dover padroneggiare un protocollo di comunicazione, scelto tra quelli in esame, per permettere ai dispositivi di comunicare tra loro nelle modalità consentite dal protocollo. I dispositivi dovranno comunicare tra loro sia con comunicazioni uno a uno, sia con comunicazioni uno a molti (Broadcast[g]) e gestire eventuali errori di trasmissione in base all'importanza dell'informazione che essi trasmettono.

Per permettere a chiunque di provare il sistema, mi aspetto di dover fornire un modo per simulare la presenza di dispositivi IoT nella propria rete domestica.

Nell'ambito delle architetture a microservizi dovrò padroneggiare alcuni dei Pattern[g] peculiari di queste architetture *software*, come per esempio il Gateway Pattern[g], e dovrò studiare le tecniche e le tecnologie che si sono diffuse per monitorare lo stato del sistema, per orchestrare i servizi e per gestire la scalabilità degli stessi.

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 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

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. 21

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. 11

Broadcast nell'ambito delle telecomunicazione, indica una trasmissione delle informazioni da un sistema trasmittente ad un insieme di apparati riceventi non conosciuto a priori (uno a molti). 16

Gateway Pattern è un Pattern[g] che propone una soluzione al problema "Come un *client* può interagire con un sistema a microservizi?". Questo è un peculiare problema delle architetture a microservizi e la soluzione proposta consiste nel sviluppare un servizio la cui funzionalità principale consiste nel gestire le richieste dei *client* in un singolo *entrypoint* pubblico, che si occupa di redirezionare la richiesta al servizio adeguato o assembla una risposta complessa a partire dalle informazioni messe a disposizione da più servizi[1]. 16

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. 4

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

MIT[g] nell'ambito delle licenze *software*, la licenza MIT è una licenza creata dal *Massachusetts Institute of Technology* per il rilascio di codice sorgente secondo condizioni permissive da parte dell'utilizzatore del codice, tutelando il *copyright* dell'autore. 15

Pattern in informatica è una soluzione testata e dimostrata ad un problema ricorrente. La sua accezione principale consiste in quella di *Design Pattern* (soluzioni progettuali). 16, 20

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

[1] Mike Wasson Masashi Narumoto. *Gateway Aggregation pattern*. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/gateway-aggregation.

Acronimi

API Application Program Interface[g]. 4, 19

MIT Massachusetts Institute of Technology. 20

Bibliografia

Riferimenti bibliografici

Stephens, Rod. *Beginning Software Engineering*. John Wiley & Sons, 2015 (cit. a p. 5).

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. 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. 8).

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. 4).