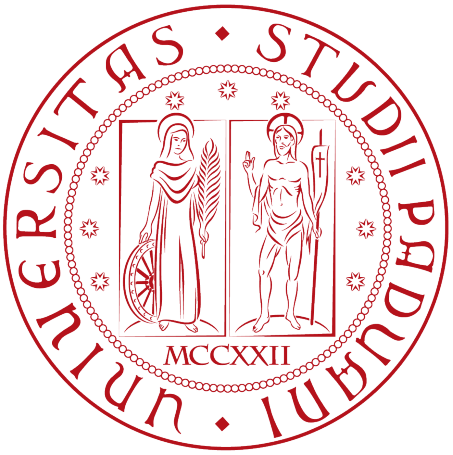
Università degli Studi di Padova

Dipartimento di Matematica 'Tullio Levi-Civita'

Corso di Laurea in Informatica



Sviluppo di una piattaforma di video *streaming* per l'assistenza remota tramite dispositivi wearable

Tesi di laurea triennale

Relatore
Prof. Tullio Vardanega

Laureando
Filippo Berto

Anno Accademico 2016–2017

Filippo Berto: *Sviluppo di una piattaforma di video* streaming *per l'assistenza remota tramite dispositivi wearable,* Tesi di laurea triennale, © Settembre 2017.

"The first precept was never to accept a thing as true until I knew it as such without a single doubt."

René Descartes

Convenzioni

Sono descritte in questa pagina le convenzioni impiegate all'interno del documento.

Immagini e didascalie

Le didascalie delle immagini riportano la sorgente originaria, l'autore dell'oggetto e, se presente, la licenza con cui è stato distribuito. Se la licenza è assente, l'utilizzo dell'immagine è da considerarsi *Fair Use*. Se la sorgente non è presente, l'immagine è stata realizzata dall'autore del documento e viene rilasciata secondo la licenza del documento stesso.

Indice

Elenco delle figure

1 L'azienda

Figura 1.1: Logo di Vision Lab Apps S.r.l.

Vision Lab Apps S.r.l. è una startup nata a New York nel 2011, con sede operativa a Torri di Quartesolo (VI), impegnata nello sviluppo di tecnologie di *ubiquitous computing* per i settori sanitario, manifatturiero e della sicurezza.

1.1 Prodotti e servizi

I prodotti principali di Vision Lab Apps sono *software* personalizzati, siti web e contenuti video. L'azienda, inoltre, offre un servizio pubblicitario per le nuove aziende: costruisce il *brand* del cliente, pone le fondamenta della sua rete di clienti e si occupa di consulenze e di *SEO*.

Con il crescere del *team* e l'acquisizione di nuovo personale più specializzato, Vision Lab Apps si sta espandendo verso servizi *cloud* per le aziende e *software* per dispositivi *wearable* e *IoT*; questi sono i primi approcci al modello di *ubiquitous computing* e permettono ai loro utenti una maggiore integrazione con la rete di informazioni e sensori che li circondano nella vita quotidiana. L'azienda sta sviluppando particolarmente il campo dei visori per realtà aumentata come supporto alle attività lavorative, promettendo grandi innovazioni nel settore manifatturiero.

1.2 Come lavora

1.2.1 Modello di sviluppo

Vision Lab Apps lavora con il modello di sviluppo *Agile* di tipo *Scrum*. Questo modello pone una minore rigidità sulla documentazione e sulle formalità del prodotto, permettendo modifiche in corso d'opera e una collaborazione più rilassata tra cliente e fornitore.

Scrum definisce uno *sprint* come l'unità di misura dello sviluppo di un progetto, un periodo di tempo di lunghezza fissata generalmente tra una settimana e quattro settimane. L'insieme delle attività necessarie per l'avanzamento del progetto sono organizzate nel *backlog* del prodotto. Per ogni *sprint* il *team* pianifica quali di questi *task* dovranno essere svolti e a chi andrà assegnato ciascuno di essi, definendo così il *backlog* dello sprint.

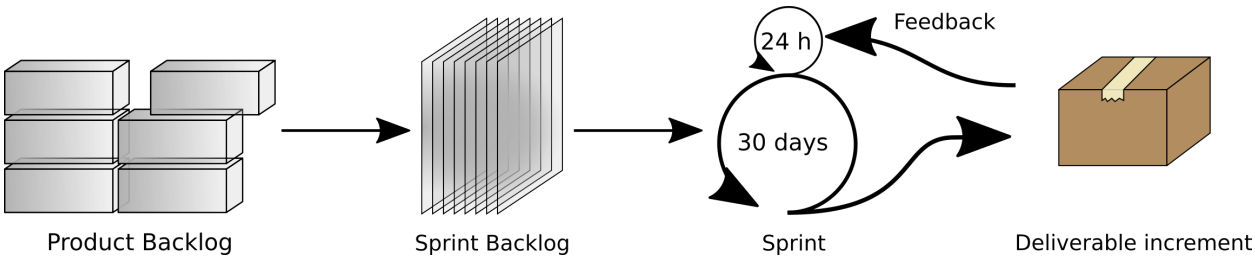


Figura 1.2: Ciclo di vita di Scrum

Ogni giorno il *team* si ritrova con una breve riunione, detta "*daily scrum*", per controllare lo stato dei *task* e degli obiettivi. I *meeting* giornalieri permettono al project manager di avere misure dello stato del progetto con più frequenza, rispetto a altri modelli di sviluppo, così da intervenire più rapidamente alla necessità di correzioni.

Un vantaggio del modello *Scrum*, e in generale dei modelli *Agile*, è quello di poter vedere il risultato del proprio lavoro più in fretta rispetto ai metodi tradizionali: i *daily scrum* servono anche a incentivare gli sviluppatori e a fornire loro una sensazione di progresso, che, invece, viene persa se i tempi tra un aggiornamento e l'altro si dilatano.
Un ulteriore punto di forza di *Scrum* è il legame di cooperazione che si forma tra il fornitore e il cliente: questo si sente parte del *team* ed è più propenso a offrire e a ricevere opinioni costruttive, con minori impuntamenti e risultati migliori per entrambe le parti.

Trattandosi di un modello *Agile* la documentazione è molto più ridotta rispetto ai metodi tradizionali: nasce il concetto di user story, un documento che descrive le richieste del cliente e le decisioni prese assieme a quest'ultimo sul progetto durante incontro faccia a faccia. Il vantaggio principale di questo tipo di documentazione è la snellezza dei documenti, sia quando devono essere consultati, sia quando devono essere scritti. Avere una visione chiara di ciò che il cliente vuole può essere difficile se è necessario scorrere decine di pagine di verbali per ottenere tali informazioni; così, al contrario, è sufficiente controllare le ultime decisioni prese.

Il coordinamento del lavoro viene gestito tramite fogli di calcolo con funzioni automatiche, condivisi all'interno del *team*. Per ogni *task* è segnalato il livello di avanzamento, che deve essere aggiornato da colui a cui è stato assegnato, riportando il tempo impiegato ed eventuali note.
Gli stati in cui un *task* si può trovare sono i seguenti:

- **Analysis**: il *task* richiede analisi
- **Pending**: il *task* è definito ed è in attesa di essere svolto
- **Blocked**: il *task* è bloccato a causa delle sue dipendenze
- **Development**: il *task* è in svolgimento
- **Testing**: il prodotto è in fase di *test*
- **Reworking**: lo sviluppo è fallito e sta venendo rieseguito
- **Refactoring**: il codice prodotto è in fase di pulizia
- **Completed**: lo sviluppo è completato

- **Confirmed**: il *task* è stato validato

Il sistema di tracking del tempo impegnato da ciascun *task* aiuta il project manager a valutare lo stato del progetto, confrontandolo con le stime fatte a preventivo.

1.2.2 Progetti importanti

VisionHealthCare

VisionHealthCare è un *software* prodotto da Vision Lab Apps in collaborazione con Dedalus S.p.a[1], società leader nazionale nel *software* clinico sanitario. L'applicazione, legata a OrmaWeb, suite applicativa web di Dedalus S.p.a, sfrutta gli occhiali per la realtà aumentata di Google, i Google Glass, per automatizzare e semplificare ogni fase del percorso chirurgico, dalla lista d'attesa alla gestione del blocco operatorio, fino alla produzione del registro operatorio e la redazione della cartella anestesiologica pre e intraoperatoria.

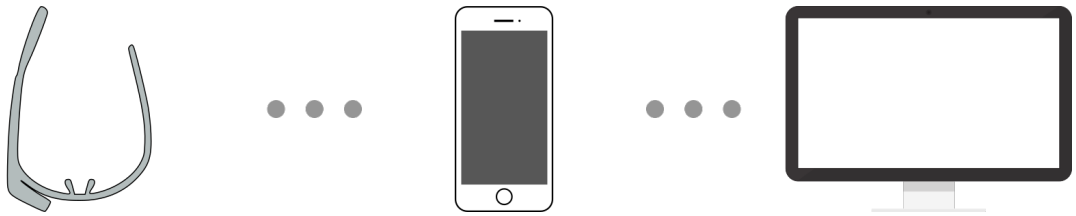


Figura 1.3: Schema di funzionamento di VisionHealthCare

I Google Glass si interfacciano con un'applicazione installata in uno smartphone e si collegano tramite questo ai servizi di OrmaWeb. Gli occhiali permettono di registrare note vocali correlate da video e foto, utili alla documentazione dell'operazione. Il sistema permette di automatizzare buona parte delle procedure di verbalizzazione dell'intervento, trascrivendo il testo registrato e aggiungendolo ai dati salvati su OrmaWeb. Un ulteriore utilizzo dei dati registrati è quello educativo: spesso l'unica persona in sala operatoria ad avere un buon punto di vista sull'operazione è il chirurgo che la sta praticando, ma questo non ha la possibilità di reggere una videocamera. Una soluzione di questo tipo permette di lavorare con le mani libere e allo stesso tempo di ricevere dati aggiuntivi sullo stato del paziente, come il suo battito cardiaco o la quantità di ossigeno nel sangue.

NapkinForever — Paperworld 2017

Vision Lab Apps sta collaborando con NapkinForever[2], azienda italiana di penne e stilo di design, per realizzare una presentazione virtuale dell'impresa per il Paperworld 2017, la più grande fiera al mondo di prodotti per gli uffici e strumenti di scrittura. Il progetto consiste in una simulazione realizzata tramite l'utilizzo dei Google Cardboard, gli occhiali per la realtà virtuale di Google, che accompagnerà i visitatori nel mondo del design di NapkinForever e presenterà loro i prodotti dell'azienda.

[1] Sito web di Dedalus S.p.a: www.dedalus.eu
[2] Sito web di NapkinForever: www.napkinforever.com

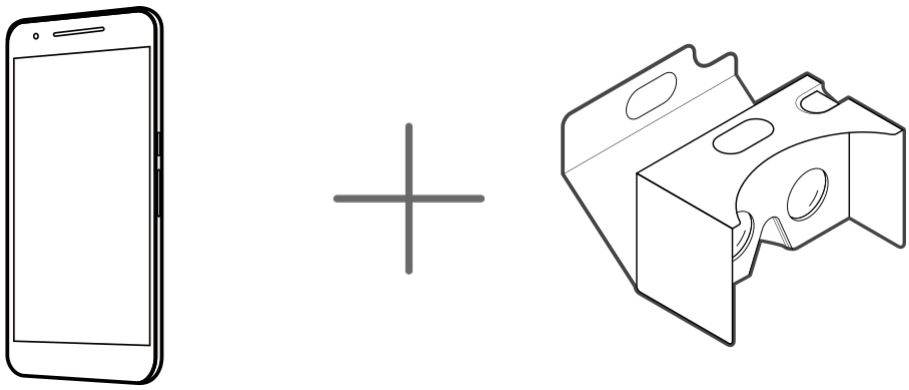


Figura 1.4: Schema di funzionamento di Google Cardboard

I Google Cardboard consistono in un apparato di lenti, da applicare allo schermo di uno smartphone, e di un telaio in cartone. L'utilizzo di *hardware* comune e di materiali poveri permette di mantenere il prezzo del dispositivo molto basso rispetto ai *competitor*, pur fornendo una buona esperienza d'uso. La simulazione permetterà di visualizzare i prodotti proposti dall'azienda in scenografie ad hoc e di ottenere informazioni contestuali su di essi.

1.2.3 Premi e certificazioni

UniCredit Start Lab

Figura 1.5: Logo di UniCredit Start Lab

Vision Lab Apps ha partecipato alla competizione tra startup UniCredit Start Lab[3] 2017, durante la quale le aziende partecipanti hanno proposto i propri progetti innovativi nei campi "Digital", "Clean Tech" e "Innovative Made in Italy". L'azienda si è classificata tra i 10 finalisti, ottenendo un periodo di incubazione e accelerazione da parte di UniCredit a partire da Settembre 2017.

1.3 Tecnologie utilizzate dall'azienda

L'azienda fa uso di un gran numero di tecnologie durante le proprie attività; di seguito analizzerò le più utilizzate.

1.3.1 Rackspace

Rackspace è un *cloud* provider che offre servizi di managed *cloud* computing, basati su Virtual Private Server (VPS) e altri servizi *cloud* come Amazon Web Services (AWS), Microsoft Azure e OpenStack. Questo tipo di servizio permette di gestire facilmente servizi *cloud* utilizzati, mantenendo il pieno controllo di costi e infrastrutture, senza la necessità di conoscere a fondo ogni componente utilizzato.

[3]Sito web UniCredit Start Lab: www.unicreditstartlab.eu

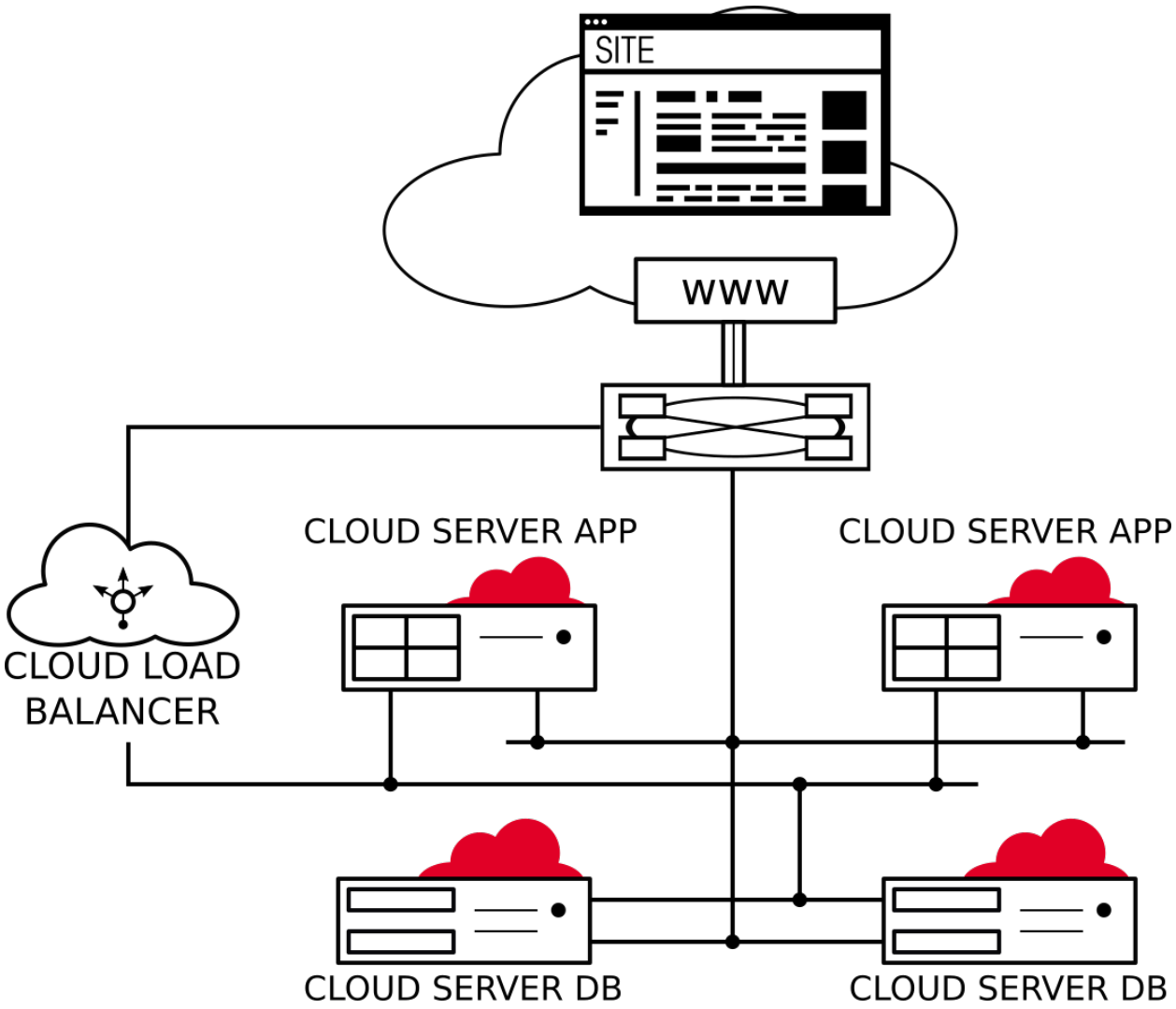


Figura 1.6: Schema di rete generale di un'applicazione su Rackspace

Rackspace permette di gestire in modo semplice nodi per il *load balancing* e la ridondanza dei servizi. In questo modo è possibile garantire *up-time* estremamente elevati e distribuire gli aggiornamenti sui singoli nodi, senza mai fermare il servizio completamente.

Vision Lab Apps usa Rackspace come *hosting provider* nel caso di progetti complessi, quando è necessaria una completa gestione delle risorse.

1.3.2 Firebase

Firebase è una piattaforma di sviluppo per applicazioni web e *mobile*, parte di Google Cloud Platform; fornisce servizi di scambio di messaggi e basi di dati in tempo reale, spazio di archiviazione, sistemi di autenticazione, web hosting e *test* automatici per applicazioni Android. La piattaforma fornisce anche un servizio di analisi e profilazione degli utenti e l'integrazione con il sistema di annunci pubblicitari di Google, AdMob.

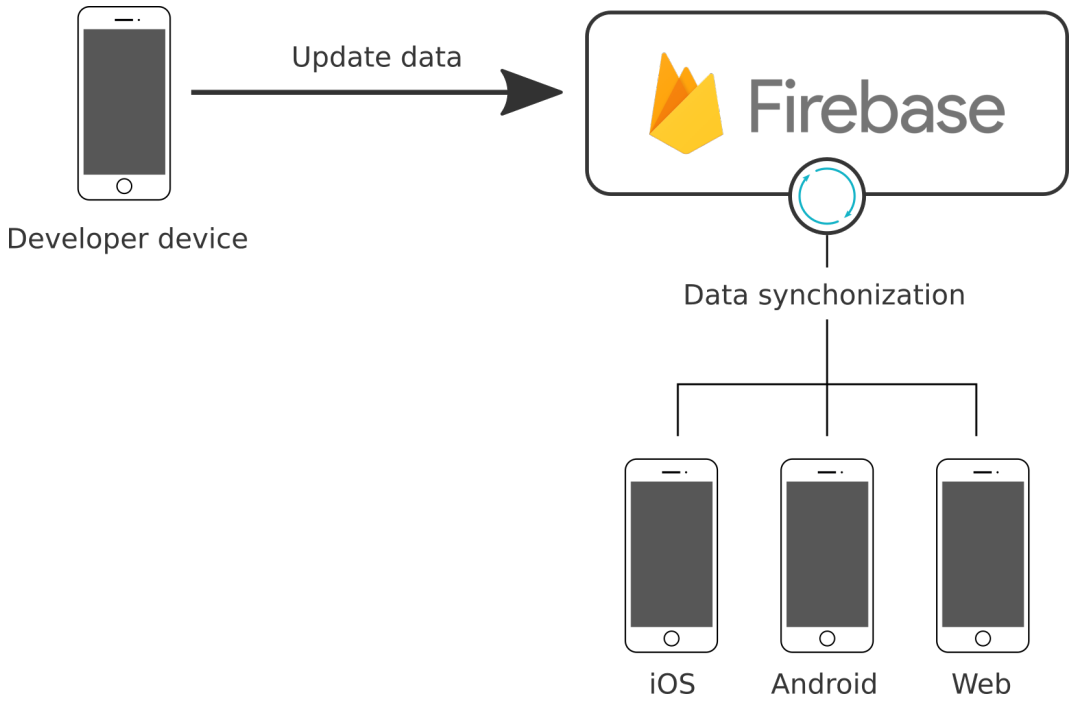


Figura 1.7: Usecase generale di Firebase

Il caso d'uso principale di Firebase è quello di base di dati per la sincronizzazione di dati tra dispositivi, anche su piattaforme diverse, in tempi molto ridotti. A differenza di un *database* non relazionale comune, il servizio che Firebase offre si basa su una CDN molto estesa, che permette di ridurre i tempi di attesa per la sincronizzazione dei dati.

Vision Lab Apps utilizza Firebase quando necessita della creazione di un ambiente di sviluppo completo, veloce e facile da manutenere.

1.3.3 Java

Java è un linguaggio di programmazione ad alto livello orientato agli oggetti pensato per essere il più possibile indipendente dalla piattaforma sulla quale viene eseguito. Java supera questo ostacolo utilizzano una macchina virtuale, la Java Virtual Machine (JVM), che permette di astrarre il sistema sottostante. Il vantaggio di Java sui linguaggi compilati tradizionali è proprio quello di poter essere eseguito su una qualsiasi piattaforma, a patto che esista una JVM per questa; inoltre, dato che il codice viene compilato per un'architettura virtuale *standard*, il compilatore è in grado di applicare una serie di miglioramenti che permettono di ottimizzare il bytecode prodotto.

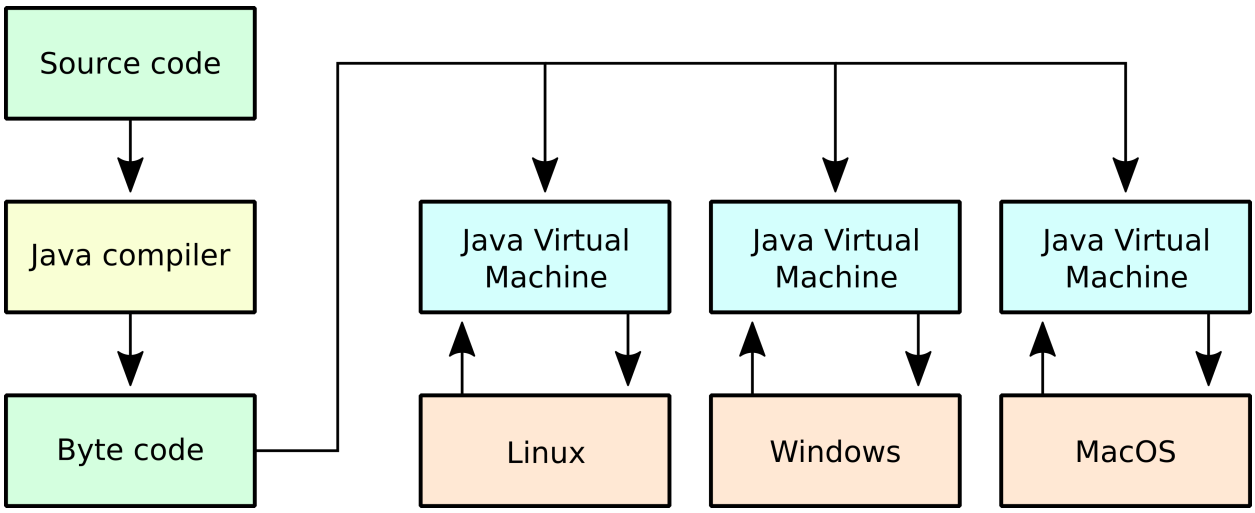


Figura 1.8: Compilazione di Java e utilizzo su più piattaforme

Tra le tecnologie utilizzate da Vision Lab Apps troviamo Android, fortemente basato su Java, e utilizzato per la creazione di applicazioni per dispositivi *mobile*. Molti dei progetti passati dell'azienda sono legati ad applicazioni Android, ma Vision Lab Apps utilizza Java anche nel caso di servizi web ad alto parallelismo e concorrenza.

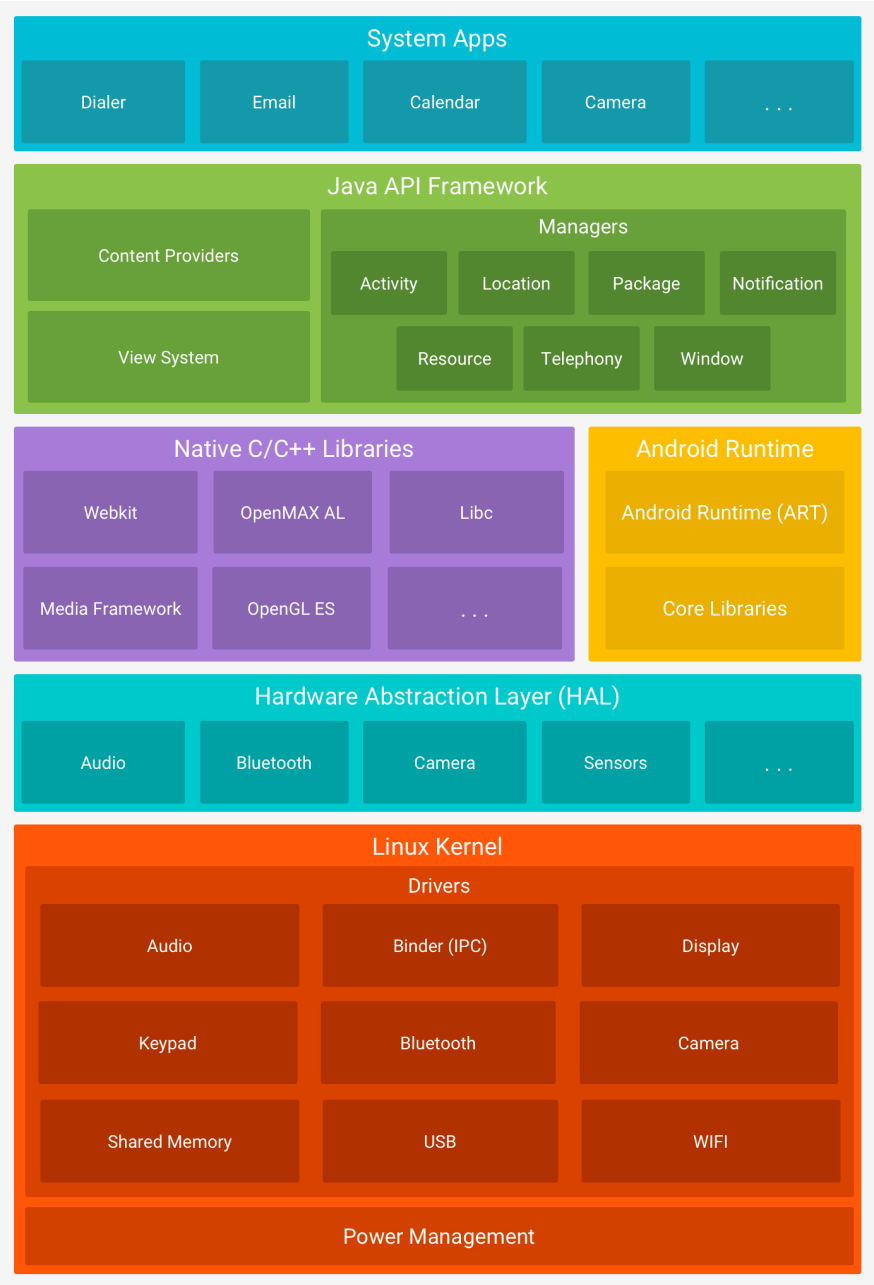


Figura 1.9: Architettura di Android
Source: Documentazione Android Developers

Android consiste di una versione minimale del *kernel* Linux, completo di driver specializzati per l'*hardware* del dispositivo sul quale verrà installato. Al di sopra del *kernel* si trova un layer di astrazione dell'*hardware*. Questo layer permette agli strati superiori di astrarre l'*hardware* sottostante. Il layer superiore contiene librerie native in C o C++, altamente efficienti, e il *runtime* Android, l'equivalente della JVM in Java, ma specializzato per l'esecuzione su dispositivi *mobile* e fortemente integrato con le librerie e l'*hardware* sottostante. Gli strati superiori sono, infine un'API per l'accesso alle risorse del sistema e le applicazioni. Queste sono racchiuse in contenitori e le autorizzazioni di queste possono essere gestite singolarmente.

1.3.4 Git e Bitbucket

Vision Lab Apps utilizza Git come CVS per il versionamento del codice: Git è in grado di gestire progetti anche molto complessi in modo efficiente. Il suo sistema completamente distribuito permette a due persone di lavorare contemporaneamente sullo stesso file, senza necessità di una connessione di rete, e di conservare copie sicure del prodotto in luoghi separati, pur garantendone la consistenza. Git permette di creare degli *snapshot* del proprio progetto in qualisasi momento tramite un *commit*. È possibile eseguire più lavori sullo stesso progetto utilizzando creando nuovi rami di sviluppo, detti *branch*.

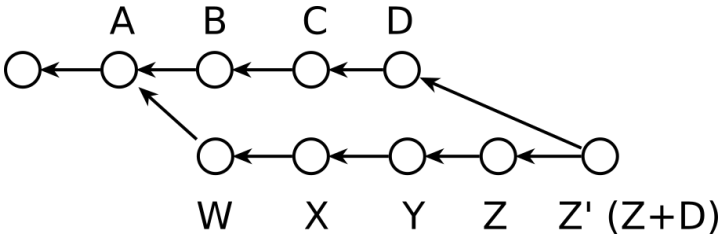


Figura 1.10: Esempio di grafo di lavoro in Git
Source: Wikimedia Commons, Bunyk, CC BY-SA 4.0

Nell'esempio in Fig. 1.10 due sviluppatori hanno lavorato in parallelo in un *repository* Git, ciascuno in un *branch* separato, dando origine ad alcuni *commit* (i nodi A, B, C, D e W, X, Y, Z). Successivamente è stato eseguito un *merge*, un'unione tra i due *branch*, che ha generato il nodo Z'. Nel caso di modifica dello stesso file Git è in grado di valutare quali sono i *chunk* di codice da unire e, in caso di conflitti tra le righe di codice scritte o modificate, lascerà agli sviluppatori il compito di risolverli, evitando così spiacevoli incongruenze.

Per facilitare la gestione del codice e automatizzare alcune attività, l'azienda ha scelto di utilizzare Bitbucket come *hoster* per le proprie *repository*. Bitbucket, oltre ad un pannello di gestione dei permessi, di statistiche e di altri utili servizi, integra il concetto di *pipeline*, una sequenza di task che permette di eseguire degli script in ambienti virtualizzati basati su Docker; in questo modo sono stati automatizzati i *test* di unità e integrazione e i controlli della *quality assurance*. Questo tipo di operazioni fanno risparmiare tempo, dato che non necessitano dell'intervento umano; inoltre, la garanzia che ogni *commit* al *repository* è stato testato conferisce la sicurezza di poter rilasciare una nuova versione senza riserbo.

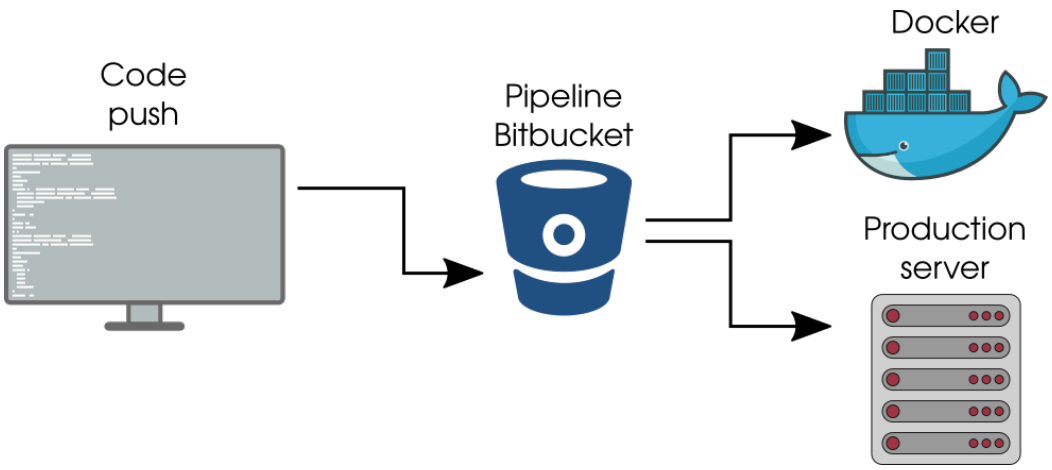


Figura 1.11: Schema generale di una *pipeline* in Bitbucket

La configurazione tipica di una *pipeline* su Bitbucket segue il seguente iter:

1. Uno sviluppatore carica sul server Bitbucket del codice, tramite un push di Git;
2. Il server Bitbucket carica il codice in un contenitore Docker, secondo la configurazione del *repository*;
3. Il contenitore esegue la configurazione dell'applicazione e lancia tutti i *test*, restituendone i risultati al server;
4. Se i *test* vengono superati il *commit* viene ritenuto valido, altrimenti può essere scartato;
5. Se il codice è stato caricato su un ramo di *release*, il nuovo *commit* viene caricato su un nuovo contenitore Docker, che si occuperà della creazione di una build del prodotto;
6. Lo stesso Docker può essere configurato in modo tale da caricare il prodotto compilato sui server di produzione, oppure avvisare un certo gruppo di persone di eseguire ulteriori *test* sulla build ottenuta.

In caso di fallimento di uno dei task della *pipeline*, lo sviluppatore che ha caricato il codice viene avvisato degli errori e gli vengono forniti i *log* delle operazioni eseguite.

1.3.5 G Suite

G Suite, la soluzione per l'ufficio di Google, offre una gestione completa di email commerciali, *editor* di testo, fogli di calcolo, calendario e archivio di dati; il tutto tramite una semplice interfaccia web.

Figura 1.12: I prodotti di G Suite

Tra i prodotti più utilizzati di G Suite si notano Drive, il servizio di *storage*; Docs, Spreadsheet e Slides, rispettivamente per documenti di testo, fogli di calcolo e presentazioni; Gmail, per la gestione delle email aziendali. G Suite include anche un servizio per la gestione di contatti, calendari e un servizio di video chat.

Vision Lab Apps usa questo servizio per le proprie attività, soprattutto per il vantaggio di poter accedere ai dati salvati anche in mobilità, con la massima comodità.

1.3.6 WordPress

WordPress è Content Management System (CMS) *open-source* che offre una piattaforma editoriale personale; nato per gestire semplici *blog*, viene utilizzato come *framework* di sviluppo di siti molto più complessi, sfruttando il sistema a plugin su cui e basato. L'utilizzo di WordPress come base di un sito permette di iniziare a lavorare con un *framework* riutilizzabile, stabile e aggiornato che ne gestisce i contenuti e i dati, permettendo allo sviluppatore di concentrarsi sulla loro presentazione all'utente.

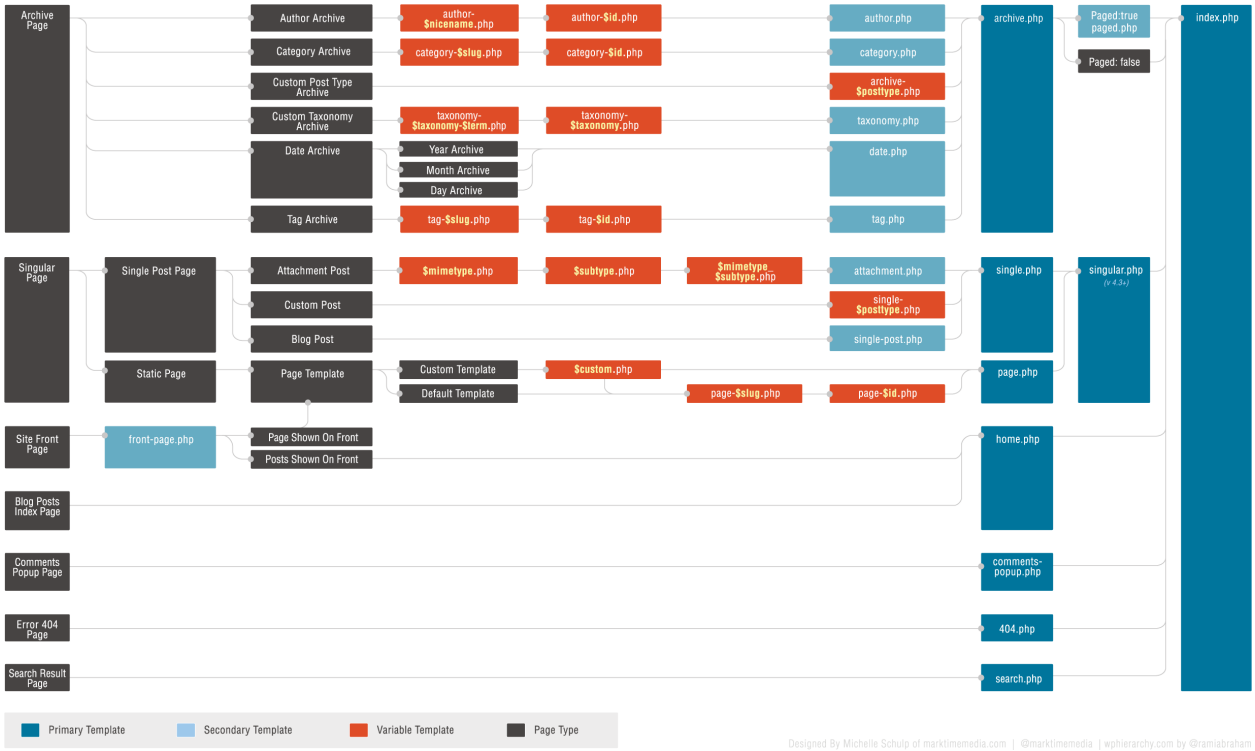


Figura 1.13: Architettura di una pagina WordPress e gerarchia delle pagine
Source: WordPress Developer Documentation

WordPress organizza i contenuti come pagine o articoli; predispone un archivio per organizzarli e un sistema di ricerca basato sui testi e sui metadati legati. La *home* del sito ha il compito di mostrare i contenuti più recenti o più importanti del sito e permette di scorrere tutti gli articoli come una vetrina.

Vision Lab Apps sfrutta WordPress come base dei propri siti anche per rendere la modifica dei contenuti semplice al proprio cliente.

1.4 Rapporto con l'innovazione

Vision Lab Apps è da sempre alla continua ricerca di nuove tecnologie da conoscere e integrare nei propri prodotti, anche in campi sperimentali, come i dispositivi *wearable*, *IoT* e la realtà aumentata. Proprio questi ultimi hanno dato origine ad alcuni dei progetti più all'avanguardia dell'azienda e l'hanno spinta all'acquisizione di personale dedito alla sperimentazione di nuove soluzioni.

Un'ulteriore necessità di innovazione deriva dal settore nel quale Vision Lab Apps si propone: il mercato è in rapida crescita e questo impone un continuo aggiornamento delle conoscenze e delle tecniche per mantenere i propri prodotti validi e restare al passo con i *competitor*.

Testimonianza di questo continuo aggiornamento è la migrazione verso uno sviluppo *cloud based* di molti dei prodotti dell'azienda, che ha portato a una riduzione dei costi di manutenzione e a un maggiore controllo sulla disponibilità dei servizi. L'azienda, inoltre, organizza seminari periodici e

laboratori per informare ed aggiornare i propri componenti sulle ultime frontiere della tecnologia, in ambito di sviluppo e *marketing*.

La proposta di nuove tecnologie è libera all'interno dell'azienda e, se ritenute utili per progetti futuri, viene predisposto un piccolo progetto di prova. In questo modo si riescono a ottenere dati concreti sui vantaggi e gli svantaggi che possono offrire.

2 Scelta dello stage e rapporto con l'azienda

2.1 Lo stage per l'azienda

2.1.1 Necessità dell'azienda

Vision Lab Apps sta sviluppando un sistema di videoconferenza per assistenza remota da applicare ai lavori specialistici e all'interno di aziende manifatturiere. Il sistema è pensato per aiutare un lavoratore inesperto, in situazioni difficili, facendolo guidare da una persona con le conoscenze adeguate a svolgere tali mansioni. Per rendere l'esperienza pratica funzionale, l'azienda ha pensato di utilizzare dei visori per la realtà aumentata, concentrandosi particolarmente sui Google Glass. L'utente esperto dovrà essere in grado di vedere in tempo reale quello che l'altro vede, tramite una videocamera integrata negli occhiali, e di mostrargli dati ed indicazioni su quello che deve fare, tramite il display integrato. Per realizzare tale progetto, Vision Lab Apps necessita di un'intera piattaforma di *streaming*, che sta realizzando a componenti nel corso del tempo.

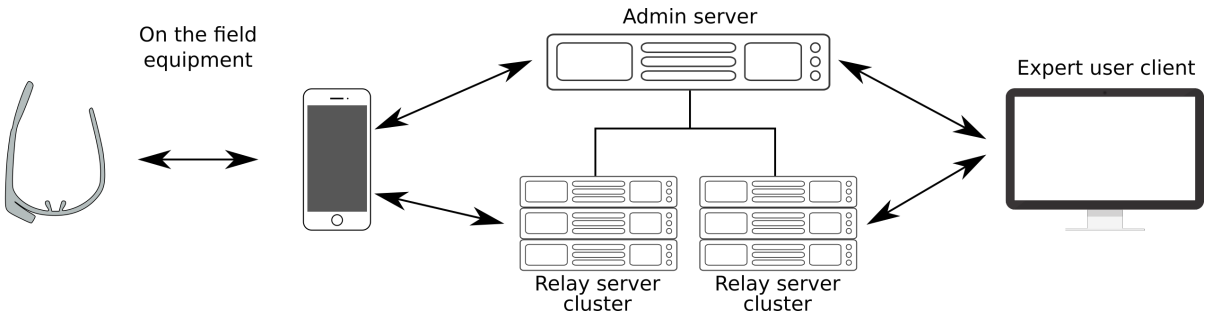


Figura 2.1: Schema generale del funzionamento della piattaforma di *streaming*

Il sistema che l'azienda vuole costruire è composto da tre elementi fondamentali, il *client* dell'utente sul campo, il sistema di *relay* e il *client* dell'utente esperto. Il primo è realizzato con uno smartphone Android e un paio di Google Glass; permetterà di registrare audio e video, inviati al sistema di *relay* tramite la rete del telefono, e di ricevere audio, video o messaggi di testo, inviati dal secondo *client*. Il *client* dell'utente esperto deve permettere di visualizzare e ascoltare i dati ricevuti e di poter rispondere con dati simili, per indicare al primo utente le azioni da compiere. Infine, il sistema di *relay* è composta da due elementi, un cluster di server di *relay* e un server amministratore; il primo è quello che si occupa del reinvio dei messaggi e della continuità della trasmissione tra i *client* connessi, mentre il secondo è adibito alla gestione dei canali, dell'autenticazione degli utenti e del bilanciamento del carico tra i nodi di *relay*.

Vision Lab Apps ha già realizzato un prototipo di *client* Android del servizio e, dopo *test* di trasmissione interni al dispositivo, si sta preparando per il passaggio alla comunicazione tra due dispositivi connessi ad una rete locale. Successivamente l'azienda provvederà a creare un sistema di amministrazione sfruttando Firebase e il suo servizio di autenticazione sicura.

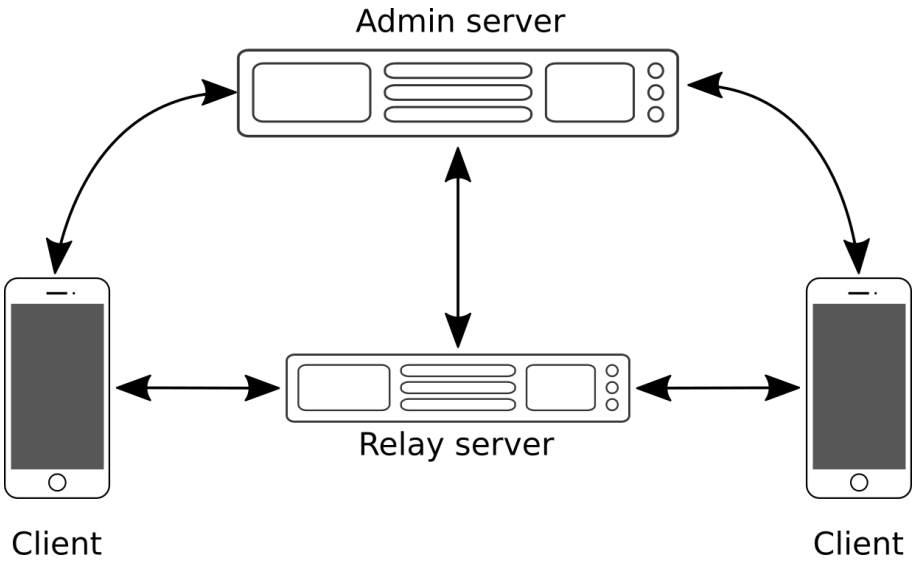


Figura 2.2: Schema delle componenti in fase di sviluppo

L'Obiettivo dello *stage* è quello di sviluppare il *software* del server di *relay*, necessario per far comunicare i due *client*. Il server deve permettere ai due dispositivi di inviarsi messaggi di qualsiasi tipo, senza che uno conosca l'indirizzo di rete dell'altro, e di gestire i canali e gli utenti tramite un API amministratore.

2.1.2 Risultati degli stage precedenti e seguito degli stagisti nell'azienda

Vision Lab Apps ha deciso anche quest'anno di partecipare all'evento "StageIt", organizzato da Confindustria Padova in collaborazione con l'Università degli Studi di Padova, e di proporre agli studenti un tirocinio interno all'azienda. L'impresa è alla ricerca di neolaureati per arricchire il proprio *team* di sviluppatori, dato il recente aumento di clienti e la conseguente espansione. Vision Lab Apps è, in particolare, interessata a studenti che sono appassionati di nuove tecnologie e che hanno interesse a imparare nuove tecniche e a farne conoscere di nuove all'azienda stessa. L'impresa ha già organizzato tirocini con altri studenti negli anni precedenti, ottenuto risultati soddisfacenti, tanto che molti degli ex stagisti sono stati assunti dall'azienda.

2.2 Rapporto con il mio stage e con l'azienda

2.2.1 Ambiti di interesse

Al momento della ricerca di uno *stage* ho prestato particolare attenzione alle aziende che proponevano un percorso legato ai miei interessi di studio. In particolare cercavo tirocini il cui argomento fosse compreso tra i seguenti:

- Sicurezza;
- Sistemi ad alta concorrenza;
- Dispositivi *IoT*;
- Sistemi virtualizzati;
- Servizi *cloud*

- *DevOps*;
- Sistemi multimediali.

2.2.2 Proposte di stage ricevute

Nei giorni immediatamente seguenti all'evento StageIt sono stato contattato da alcune aziende presenti, delle quali ho selezionato le proposte che più mi parevano interessanti.

Uno dei progetti proposti consisteva in un sistema di controllo di risorse e consumi di sistemi virtualizzati tramite Docker: il *software* doveva fornire una chiara visione dello stato di ciascun servizio e riportare eventuali anomalie. Lo *stage* mi avrebbe permesso di conoscere meglio l'utilizzo di Docker, specialmente in un contesto *DevOps*, e di imparare nuove tecnologie di gestione di gruppi di contenitori, in particolare Kubernetes; fondamentali in un contesto di *continuous deployment* e *continuous release*. Le conoscenze che mi avrebbe fornito questo tirocinio erano di mio grande interesse, ma, sfortunatamente, non sono stato scelto, tra i possibili candidati.

Un altro *stage* riguardava la realizzazione di un *software* in grado di unificare la grande mole di dati dell'azienda, sparsa tra diversi sistemi aziendali legacy e i CMS dei loro clienti. Il prodotto avrebbe dovuto raccogliere i dati, salvarli in un formato comune, facilmente accessibile, e aggiornare i sistemi in caso di modifiche. L'utilità di un *software* del genere è molto elevata per un'azienda, ma il prodotto in sé consiste in un certo numero di adattatori per ciascuna delle fonti di dati, e non mi avrebbe fornito particolari conoscenze aggiuntive, rispetto a quelle che già possedevo.

Un altro progetto proposto, invece, prevedeva la realizzazione di un sistema per la gestione delle traduzioni dei testi di un catalogo prodotti, consentendo visualizzazione, modifica e la ricerca di testi ripetuti tramite Elasticsearch, per l'ottimizzazione delle spese di traduzione. Elasticsearch è un motore di analisi del testo il cui utilizzo si sta diffondendo molto in fretta, ma lo *stage* non comprendeva lo studio del suo funzionamento. Il prodotto avrebbe anche dovuto generare una serie di informazioni, come un preventivo dei costi di traduzione e una lista di frequenza di certe espressioni.

Un ulteriore tirocinio prevedeva un progetto sperimentale per lo spostamento di un *software* di rendering CAD per TAC da un sistema locale a uno *client-server*, con elaborazione dei dati su server virtualizzati. Il video elaborato sarebbe poi stato servito a un *thin client*; alleggerendolo del carico di computazione del render. Il progetto comportava la scomposizione del *software* proprietario dell'azienda, lo studio di una soluzione di virtualizzazione e l'integrazione con una libreria per la trasmissione dei dati. Sebbene lo *stage* proposto toccasse molti e vari ambiti, si sarebbe concentrato solo in una parte del progetto, prevedendo la continuazione da parte dello studente dopo la laurea. Considerata la mia scelta di continuare i miei studi, non ho potuto accettare la proposta dell'azienda.

2.2.3 Scelta dello *stage*

Al momento della scelta di quale *stage* accettare, tra quelli proposti, ho preferito optare per quello che proponeva argomenti per me più nuovi e meno conosciuti, che producesse, quindi, il migliore valore aggiunto per mie competenze. Ho scelto il progetto di Vision Lab Apps perché ho trovato il campo di cui si occupa interessante e le mie conoscenze sull'argomento erano solo marginali. Inoltre una più approfondita conoscenza di sistemi ad alta concorrenza, come può essere un servizio di *streaming*, è applicabile a molti altri campi, come sistemi *cloud* distribuiti e dispositivi *IoT*.

2.2.4 Scelta dell'azienda

Durante la scelta dello *stage* ho considerato marginale l'aspetto del futuro in azienda, dato che la mia scelta di continuare a studiare per la laurea magistrale è incompatibile con un lavoro a tempo pieno. Il mio interesse più grande era quello di poter collaborare con persone più esperte di me per imparare cose nuove; per questo motivo mi sono accertato che durante il tirocinio avrei potuto interagire con molte figure dell'azienda che si occupano di sviluppo.

Un altro importante fattore che ho deciso di ignorare è stata la distanza dell'azienda dalla mia residenza: ho preferito dare più importanza all'esperienza dello *stage* in sé rispetto alla comodità di spostamento.

2.3 Obiettivi del progetto di tirocinio

All'inizio del progetto di tirocinio sono stati definiti gli obiettivi, distinti poi in obbligatori e desiderabili, e i vincoli tecnologici ai quali ho dovuto attenermi. Ne segue una lista dei fondamentali.

2.3.1 Obiettivi obbligatori

- Studio dei formati video e dei protocolli di rete per le trasmissioni video in *real-time* e *on-demand*;
- Studio dell'architettura di rete di un servizio di *streaming*;
- Sviluppo di un applicativo server per il *relay* di messaggi tra i suoi *client*.

2.3.2 Obiettivi desiderabili

- Studio delle problematiche della trasmissione di dati in mobilità;
- Sistema di autenticazione dei *client* e organizzazione a canali delle trasmissioni;
- Studio di possibili utilizzi della tecnologia *Blockchain* nel campo dello *streaming*.

2.3.3 Vincoli tecnologici

Mi è stato richiesto di utilizzare Java come linguaggio di programmazione, per consentire una più veloce integrazione con il *client* Android. Inoltre, data la struttura a servizi della piattaforma, ho dovuto utilizzare un server Java come base del prodotto. Data la natura molto libera del progetto, non mi sono stati dati altri vincoli tecnologici, lasciandomi, quindi, piena libertà di scelta sull'architettura, i pattern, i moduli e le librerie da utilizzare.

2.4 Pianificazione del lavoro

La pianificazione del progetto è stata eseguita in accordo con il tutor interno. Nella prima settimana ho avuto modo di studiare gli strumenti, i formati e i protocolli necessari alle funzioni della piattaforma, stilando una relazione sullo stato dell'arte. Ho, poi, proseguito con l'analisi delle funzionalità richieste e la definizione di requisiti e casi d'uso. Ho impiegato la terza settimana nella progettazione del servizio e della componente server, per poi procedere le due settimane successive alla sua realizzazione. Per garantire una buona qualità del codice, ho dedicato una settimana alla

validazione, ai beta *test* e alla riformattazione. L'ultima settimana ho ultimato la documentazione del progetto e dei risultati ottenuti.

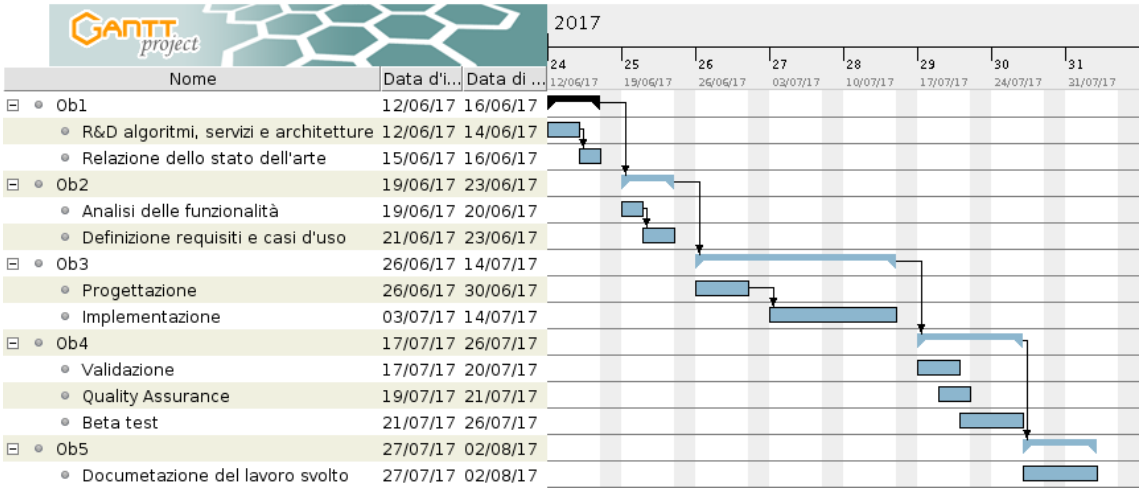


Figura 2.3: Diagramma di Gantt della pianificazione

2.4.1 Strumenti utilizzati

Per ottenere buoni risultati durante la pianificazione Vision Lab Apps utilizza GanttProject[1], un *software open-source* per la realizzazione di diagrammi di Gantt e PERT. Il programma è stato progettato per costruire diagrammi con estrema facilità, gestendo scadenze, risorse, personale e dipendenze; inoltre permette di esportare il risultato in Hypertext Markup Language (HTML) o in formato Portable Network Graphics (PNG), utile per l'integrazione in presentazioni o documentazione.

[1]Sito web del progetto GanttProject: www.ganttproject.biz

3 Il prodotto

In questo capitolo descriverò il progetto, le sue fasi, i suoi prodotti e il lavoro che è stato compiuto durante il tirocinio.

3.1 Acquisizione delle conoscenze settoriali

La prima fase del progetto ha vertito sull'acquisizione delle conoscenze del campo che ancora non avevo o di cui necessitavo approfondimenti. Ho eseguito questi studi in modo autonomo, ricercando gli argomenti in Internet e su libri e appunti dei corsi di laurea.

3.1.1 Codifica video

Un video digitale è definito come la codifica numerica di un segnale video; la codifica è, dunque, l'azione di trasformare un segnale video in un formato digitale. Questa operazione è effettuata da un *software* o da un dispositivo elettronico detto *codec*. Generalmente, i *codec* eseguono anche compressione lossy, ovvero con perdita di informazioni, in modo tale da ridurre i dati necessari per trasmettere o salvare i flussi video. Esistono anche *codec* che mantengono inalterata la quantità di informazione; questi vengono definiti *lossless*.

Esistono due famiglie di *codec*, quelli intraframe e quelli interframe: i primi descrivono i *frame* di un video singolarmente, trattandoli come immagini statiche; i secondi, invece, descrivono i cambiamenti che occorrono tra un fotogramma e l'altro, partendo da *frame* chiave descritti con codifica intraframe. I *codec* intraframe risultano, quindi più adatti alla descrizione di scene molto movimentate, senza perdita di qualità, mentre quelli interframe sono più adatti nel caso di sequenze per lo più statiche, per la loro notevole efficienza nel descrivere i pochi cambiamenti avvenuti.

3.1.2 Formati video comunemente usati

M-JPEG

Motion Joint Photographic Experts Group (JPEG) è un *codec* video che comprime ogni *frame* del video originale in una immagine in formato JPEG, un comune formato di compressione con perdita per immagini digitali. Originariamente creato per applicazioni multimediali, M-JPEG è attualmente utilizzato da dispositivi di videoregistrazione come camere digitali, IPcam e webcam.

Un grande vantaggio dell'utilizzo di M-JPEG per la codifica video, che ha spinto la sua diffusione sulla maggior parte delle prime fotocamere digitali non professionali, è quello della bassa potenza di calcolo necessaria al suo utilizzo: è sufficiente una modifica *software* all'encoder *hardware* per JPEG per poter creare un *codec* M-JPEG. Inoltre, permette la lettura non lineare del file, utile nel caso di sistemi di *video editing*. Inoltre l'*encoding* intraframe che sfrutta limita l'informazione contenuta in ciascun *frame* M-JPEG a quella di un *frame* del video originale, permettendo di cambiare velocemente il contenuto, senza considerare i *frame* precedenti o successivi, come invece accade nelle codifiche interframe come H.264/MPEG-4 AVC.

Con l'accrescere della potenza di calcolo, M-JPEG è stato soppiantato da MPEG-4, per la resa qualitativamente molto migliore a parità di *bitrate*, infatti JPEG è inefficiente nello spazio di archiviazione. Inoltre M-JPEG non sfrutta tecniche di predizione spaziale, come H.264/MPEG-4 AVC, che permetterebbero un ulteriore risparmio di spazio, a discapito di una maggiore potenza di calcolo necessaria.

MPEG-1

MPEG-1 è uno *standard* per la compressione con perdita di video e audio progettato per comprimere video con la qualità variabile tra il video di una VHS e l'audio di un Compact Disc (CD) fino a 1.5Mbit/s senza perdita di qualità eccessive, permettendo la creazione di video CD, trasmissioni TV via satellite e via cavo e DAB. Attualmente, MPEG-1 è il formato *lossy* con più supporto di compatibilità al mondo e uno degli *standard* che ha introdotto più noti è stato il formato audio MP3.

MPEG-1 introduce molte tecnologie, frutto di studi statistici, che gli permettono di risparmiare molti dati per descrivere lo stesso video con poca perdita di qualità: innanzitutto l'utilizzo della trasformata discreta del coseno su blocchi di $8 \cdot 8$ permette di ottenere coefficienti tra loro incorrelati e solo alcuni di questi sono dominanti, permettendo un alto livello di compressione. I coefficienti Discrete Cosine Transform (DCT) ottenuti vengono quantizzati, con una grana più fine nelle frequenze più basse e con una più grossa nelle frequenze più alte. Queste influiscono meno sul risultato finale e il loro annullamento permette di diminuire il *bitrate* complessivo senza pesanti perdite di qualità. Data la grande quantità di zeri dovuti alla quantizzazione, si utilizza una codifica run-length, ovvero si indicano i valori a coppie (a, b), dove a rappresenta il valore di un coefficiente non nullo e b il numero di zeri che lo precedono. Si sfrutta infine una codifica entropica basata su una tabella di valori standardizzati, ottenuti con un approccio basato sulla codifica di Huffman.

MPEG-4

MPEG-4 è basato sugli *standard* MPEG-1, MPEG-2 e QuickTime. Si è osservato che, in caso di video con sezioni statiche o relativamente statiche, molti *frame* sono riproducibili dai precedenti. È stata, quindi, realizzata una tecnica che sfrutta la somiglianza tra *frame* vicini per risparmiare spazio nella codifica. Questa si ottiene indicando alcuni *frame* come *frame* chiave (I-Frame) e generando i *frame* seguenti come trasformazioni del *frame* principale

MPEG-4 permette di creare oggetti multimediali con migliori capacità di adattamento e flessibilità, in grado di migliorare la qualità del risultato finale; inoltre è in grado di adattare la propria qualità sia a *stream* a basso *bitrate*, anche 1 Mbit/s, fino a formati ad alta risoluzione, come 4K e 8K.

Le sue ottime performance di qualità e compressione costano in potenza computazionale, a causa della complessità della fase di quantizzazione; al contrario, la decodifica per la riproduzione è sufficientemente rapida e poco costosa, grazie anche a decodificatori *hardware* specializzati, ormai comunemente integrati nella maggior parte dei processori.

WebM

WebM è un formato video *royalty-free* pensato per il web, rilasciato sotto licenza Berkeley Software Distribution (BSD), il cui sviluppo è sponsorizzato da Google. WebM sfrutta il motore di codifica VP9 e audio Opus ed è supportato nativamente dai maggiori browser, inclusi quelli dei

dispositivi *mobile*. WebM punta ad una migliore efficienza di compressione rispetto a MPEG-4 e sta sostituendo Graphics Interchange Format (GIF) come formato di animazione video. Mentre il formato è libero, i motori VP8 e VP9 sono patentati Google e solo nel 2013 Moving Picture Experts Group (MPEG) LA ha raggiunto un accordo con la multinazionale per la licenza delle componenti essenziali all'implementazione dei *codec*. Le attuali implementazioni libere (libvpx) mostrano un significativo vantaggio nei tempi di codifica e nel risparmio di *bitrate* rispetto a MPEG-4 H.264 e H.265.

3.1.3 Protocolli

Un protocollo di rete è un protocollo di comunicazione utilizzato da in una rete informatica, ovvero la definizione formale a priori delle modalità di interazione tra due o più apparecchiature elettroniche devono utilizzare per poter comunicare tra di loro tramite una rete.

Protocolli per il video *streaming*

Quando parliamo di protocolli di rete possiamo dividerli due categorie in base al tipo di trasmissione che utilizzano, ovvero se *stream-oriented* o *message-oriented*. Questa differenza, riscontrata al *Transport Layer* della pila Open Systems Interconnection (OSI), è la base dei due più famosi protocolli di quel livello: Transmission Control Protocol (TCP) e User Datagram Protocol (UDP).

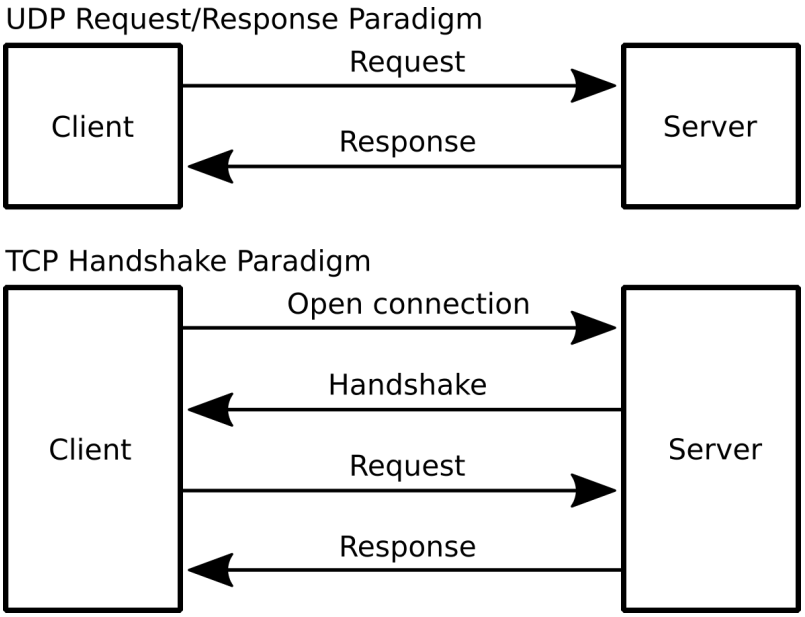


Figura 3.1: Schema di connessione di TCP e UDP
Adattato da: Wikimedia commons, Khandarmaa, CC BY-SA

TCP appartiene alla prima categoria: provvede alla realizzazione dei una connessione affidabile, nella quale i dati vengono suddivisi in pacchetti ordinati e in caso di errori di trasmissione il ricevente può richiederne il reinvio. Questo protocollo è pensato per la consistenza dei dati e assicura che le informazioni richieste arrivino tutte, integre e nell'ordine corretto al ricevente.

UDP, al contrario, appartiene al secondo genere: utilizza un modello *connectionless*, pensato per messaggi atomici, e non garantisce che l'ordine di invio di un certo numero di pacchetti sia lo stesso

con il quale vengono ricevuti. Inoltre, il protocollo non ha particolari misure di prevenzione della corruzione dei dati, se non un checksum dei contenuti del pacchetto, che ne permette la verifica di integrità.

Nel caso di applicazioni di *streaming* è importante nominare anche il protocollo Stream Control Transmission Protocol (SCTP): si tratta di un protocollo *message-oriented* e *connectionless*, simile a UDP, ma che aggiunge alcune feature di TCP, come l'ordinamento dei pacchetti e la richiesta di reinvio. Questo permette al ricevente di mantenere la coerenza dei dati, anche in caso di messaggi lunghi, e di ricevere più *stream* di pacchetti in parallelo.

Tipologie di streaming

Nel caso applicativo dello *streaming* video si riscontrano due possibili situazioni: nel caso di *streaming real-time* l'obiettivo è quello di trasmettere un video con una latenza molto bassa, anche a discapito della qualità video; si pensi, ad esempio, quanto è negativa l'esperienza di un utente che durante una videoconferenza si accorge che un altro utente deve aspettare del tempo prima di ricevere la propria registrazione.

Altro caso è quello dello *streaming on-demand*, nel quale un utente richiede l'accesso ad una risorsa video remota e la sua visualizzazione deve essere riportata con la qualità nativa; in questo caso l'utente è disposto ad aspettare qualche secondo di precaricamento iniziale, ma risulta particolarmente infastidito dalle interruzioni, magari a causa del *buffering*.

Nel primo caso il protocollo ideale è quello che trasmette con il *bitrate* più alto possibile e con la massima frequenza di invio di messaggi; infatti il solo *bitrate* non basta, una frequenza di invio alta permette di limitare la latenza alla sorgente del segnale, rimpicciolendo il tempo di attesa tra un intervallo e il successivo. Per questo tipo di trasmissioni il protocollo il protocollo più adatto è UDP, per la dimensione variabile del messaggio, la piccola dimensione del suo *header* e l'assenza di garanzie di consistenza non fondamentali, come la certezza che tutti i dati siano stati ricevuti. Sono nati quindi numerosi protocolli per l'*Application Layer* OSI basati su UDP, adattati alla trasmissione di *streaming* video, come Real-time Transport Protocol (RTP), Real Time Streaming Protocol (RTSP), Real Time Messaging Protocol (RTMP). A questo tipo di protocolli vengono spesso associati protocolli di controllo, come Real Time Message Control Protocol (RTMCP) e Realtek Remote Control Protocol (RRCP), che permettono a trasmittente e ricevente di accordarsi sul formato dei dati e sul *bitrate* da utilizzare.

Nel secondo caso il protocollo deve garantire la consistenza e l'integrità della trasmissione, pur mantenendo un *bitrate* elevato. Questo tipo di trasmissione può essere ottenuta tramite un protocollo *stream-oriented*, come quello di TCP. Nascono, quindi, anche nel caso di TCP protocolli per l'*Application Layer* OSI progettati per *stream* di video, come Dynamic Adaptive Streaming over HTTP (MPEG-DASH), HTTP Live Streaming (HLS) di Apple e Microsoft Smooth Streaming. Per risolvere il problema delle interruzioni, molti servizi lasciano al *client* la possibilità di caricare dinamicamente più di una versione dello stesso contenuto, in base alla disponibilità di bandwidth.

Esiste infine un ulteriore funzionalità che sta diventando comune nei servizi più grandi: quella dello *streaming* ibrido. Il servizio si basa su *streaming real-time*, ma permette agli utenti di rivedere una scena già trasmessa, comportandosi come uno *streaming on-demand*. Questo tipo di servizio richiede che il contenuto registrato venga salvato e velocemente diffuso all'interno della piattaforma, per poi essere reso disponibile, come per i restanti contenuti *on-demand*.

UDP e i problemi di *firewall*

Sebbene i protocolli basati su UDP siano particolarmente adatti alle comunicazioni *real-time*, soffrono di un grave problema legato alle regole applicate ai *firewall*, specialmente nel caso di reti pubbliche o aziendali: capita molto spesso che per evitare minacce proveniente da reti esterne gli amministratori decidano di bloccare tutti, o quasi, i pacchetti UDP in entrata, a meno di risposta ad una chiamata proveniente dall'interno. Questa pratica è diventata molto comune, tanto che anche servizi di *streaming*, come YouTube e Twitch, o di telecomunicazione, come Skype o Hangouts, siano stati costretti ad incapsulare il contenuto dei propri pacchetti UDP in pacchetti TCP, per superare le protezioni e offrire i propri servizi.

WebRTC

WebRTC è un insieme di protocolli di rete che permettono l'invio di messaggi in *real-time* tramite connessioni *peer-to-peer*. Il sistema è pensato per essere integrato nativamente nei browser, in modo tale da poter essere utilizzato, tramite un'interfaccia, da *webapp* apposite per comunicare con componenti di backend e browser di altri utenti. WebRTC sfrutta il protocollo RTP, per il trasferimento di audio e video, e lo stato attuale del suo supporto da parte dei principali browser è quasi totale: solo vecchie versioni di Internet Explorer e Safari non lo supportano, se non tramite plugin. Sebbene questa tecnologia sia standardizzata e il suo impiego sia sempre più frequente da parte di servizi di *videochat*, le sue implementazioni per le piattaforme Android e iOS sono ancora acerbe e poco documentate; per questo motivo è ancora comune definire un protocollo di comunicazione specifico per l'applicazione, piuttosto di utilizzare questo *standard*.

3.1.4 Architettura di una piattaforma di *streaming*

Struttura di massima

Una piattaforma di video *streaming* si compone di cinque elementi principali:

- La rete di *inbound*, ovvero il sistema di reti che gestisce l'ingresso di dati nella piattaforma, siano questi file video o *stream*;
- Il motore di codifica, che permette di ricodificare i video in ingresso per renderli disponibili nei formati opportuni ai passaggi successivi.
- Il sistema di *storage*, per il salvataggio dei contenuti per la breve, media e lunga conservazione;
- Il sistema di gestione e ricerca dei contenuti, che permette agli utenti di caricare, cercare e riprodurre i contenuti di proprio interesse;
- Il sistema di distribuzione dei contenuti, ovvero la rete che si occupa della trasmissione di dati ai *client*.

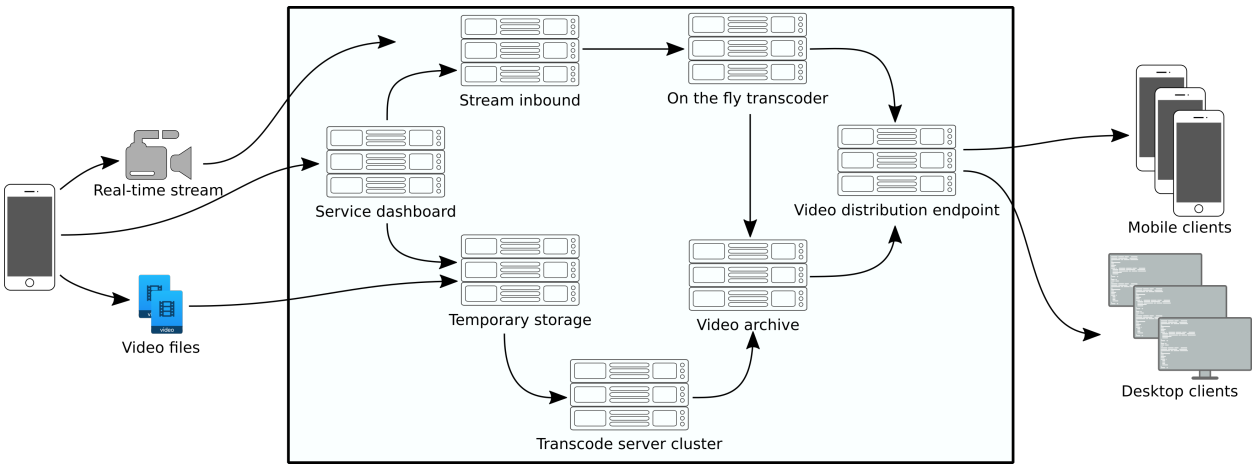


Figura 3.2: Schema generale di una piattaforma di *streaming*

Inbound e transcodifica

La rete di *inbound* è la parte del sistema che si occupa di accettare file e *stream* da parte degli utenti e di inoltrarli ai nodi sottostanti. Questa componente è soggetta ai tipici problemi delle applicazioni web con un numero di utenti rilevante, ovvero la stabilità del servizio, la necessità di reti ad alta disponibilità, il bilanciamento del carico di lavoro tra i nodi sottostanti.

I contenuti caricati tramite il sistema di *inbound* vengono, poi, inviati ai motori di codifica, componenti specializzati nell'*encoding* dei video. A seconda del tipo di input è possibile scegliere se utilizzare un transcodificatore per lo *streaming*, in grado di rielaborare il formato video di uno *stream* "al volo" e con poca latenza aggiuntiva, oppure, nel caso di file video, questi vengono caricati in *storage* temporanei e successivamente elaborati da transcodificatori comuni.

Storage e organizzazione dei contenuti

Il sistema di *storage* si occupa dell'organizzazione dei contenuti, del loro salvataggio e del loro recupero. Al semplice *filesystem* viene associato un *database*, spesso non relazionale, per la gestione di metadati, testi, commenti e dati aggiuntivi. Il sistema deve permettere una veloce ed efficace ricerca dei contenuti, per permettere agli utenti di accedere alle informazioni volute.

Outbound, protocolli e CDN

Nel caso della distribuzione di contenuti verso i *client* sono generalmente utilizzati protocolli basati su TCP, come MPEG-DASH, Apple HLS e Microsoft Smooth Streaming. Questo tipo di protocolli permettono più *stream* paralleli e la possibilità di eseguire caching dei contenuti trasmessi, particolarmente utili nel caso di utilizzo di CDN per estendere la distribuzione. Queste reti, solitamente di proprietà delle aziende di telecomunicazione su cui poggiano gli Internet Service Provider (ISP), offrono un servizio di caching che permette di spostare il carico di rete dovuto all'accesso ai contenuti dai server principali del servizio a server più piccoli e dislocati sul territorio. Questo tipo di distribuzione, non solo diminuisce il carico di rete, limitato quindi alla sola autorizzazione all'accesso, ma permette di avere latenza e tempi di caricamento minori da parte degli utenti.

Message relay

Esistono casi di servizi di *streaming* che devono permettere il reinvio di dati tra due *client*, senza alcuna modifica ai contenuti trasmessi; in questo caso di parla di message *relay*. Questo tipo di servizio è fondamentale se i due *client* non possono comunicare direttamente, magari perché bloccati da *firewall* o nascosti da una rete con Network Address Translation (NAT).

Servizi annessi

Molte piattaforme di *streaming* non si limitano alla sola trasmissione di contenenti ma offrono anche un certo numero di servizi annessi. In particolare le informazioni legate alla demografia degli utenti e alla loro profilazione sono di particolare interesse per i creatori. Inoltre sono comuni servizi di monetizzazione e annunci pubblicitari, cifratura dei contenuti e sistemi Digital Rights Management (DRM).

3.1.5 Trasmissione dei contenuti in mobilità

La quantità di utenti che consumano contenuti multimediali su applicazioni *mobile* sempre più in aumento, tanto che nel Novembre 2016 si è registrato il sorpasso della quantità di utenti su tali piattaforme, rispetto ai *desktop*. È in crescita anche l'utilizzo di questi dispositivi per eseguire live *stream*, specialmente legati a social network, come Twitter, Facebook e Google+. L'utilizzo di dispositivi *mobile* aggiunge un ulteriore strato di complessità alla comunicazione con i servizi web, come anche le piattaforme di *streaming*. Le difficoltà maggiori che si incontrano in questo campo sono la limitata potenza computazionale dei dispositivi e autonomia energetica, i limiti imposti da una rete senza fili e la grande frammentazione dei sistemi, specialmente quando si tratta di Android. La maggior parte dei problemi precedentemente citati viene gestito dal sistema operativo, ma restano le limitazioni fisiche, ovvero la velocità di trasferimento dei dati:

3.1.6 Reti per dispositivi *mobile*

Le principali reti disponibili per dispositivi *mobile* possono essere riassunte nelle seguenti:

LTE — 4G

Long Term Evolution (LTE), lo *standard* commerciale di quarta generazione per le telecomunicazioni *mobile*, supporta *downlink* con velocità di picco di 300 Mbit/s e *uplink* di 75 Mbit/s in caso di posizione statica. Questo tipo di connessione è ampiamente sufficiente alla trasmissione di video in alta risoluzione e alto framerate. LTE ha lo svantaggio di essere soggetto a forti interferenze quando deve attraversare materiali solidi come muri o persone.

Wi-Fi

L'ultimo *standard* Wi-Fi, IEEE 802.11ac, ormai comunemente supportato dai dispositivi *mobile* in commercio, raggiunge una velocità di trasmissione di circa un Gbit/s. Il precedente *standard*, IEEE 802.11n, era limitato a 300 Mbit/s tramite l'utilizzo di sistemi ad antenna multipla. Anche Wi-Fi è soggetto ad interferenze, specialmente quando utilizzato sulle frequenze più alte, a 5GHz. Oltre alla comune connessione a infrastruttura, IEEE 802.11 supporta anche connessioni *peer-to-peer* tramite il protocollo Wi-Fi Direct, che riduce i consumi e massimizza il throughput.

Bluetooth

Bluetooth 5 ha un throughput molto più basso dei precedenti canali di comunicazione; raggiunge, infatti, circa 50 Mbit/s, un *bitrate* appena sufficiente alla trasmissione di video HD, al contrario, però, la distanza tra il trasmittente e il ricevente è molto più ridotto. Il vantaggio principale di Bluetooth è quello di utilizzare una minore quantità di energia elettrica per comunicare con altri dispositivi.

3.2 Nuovi orizzonti

Tra gli obiettivi desiderabili indicati dall'azienda sono presenti lo studio di alcuni campi innovativi e la possibilità di utilizzarli nel contesto dello *streaming* video. Seguono i risultati dei miei studi e delle mie ricerche a riguardo.

3.2.1 Blockchain

Blockchain è una tecnologia che permette di rendere delle informazioni pubbliche virtualmente impossibili da alterare. Il sistema si basa su catene di nodi distribuite in una rete *peer-to-peer*; ogni nodo contiene una certa quantità di informazioni fissata, un numero intero variabile, detto nonce, e l'hash dell'ultimo nodo della catena.

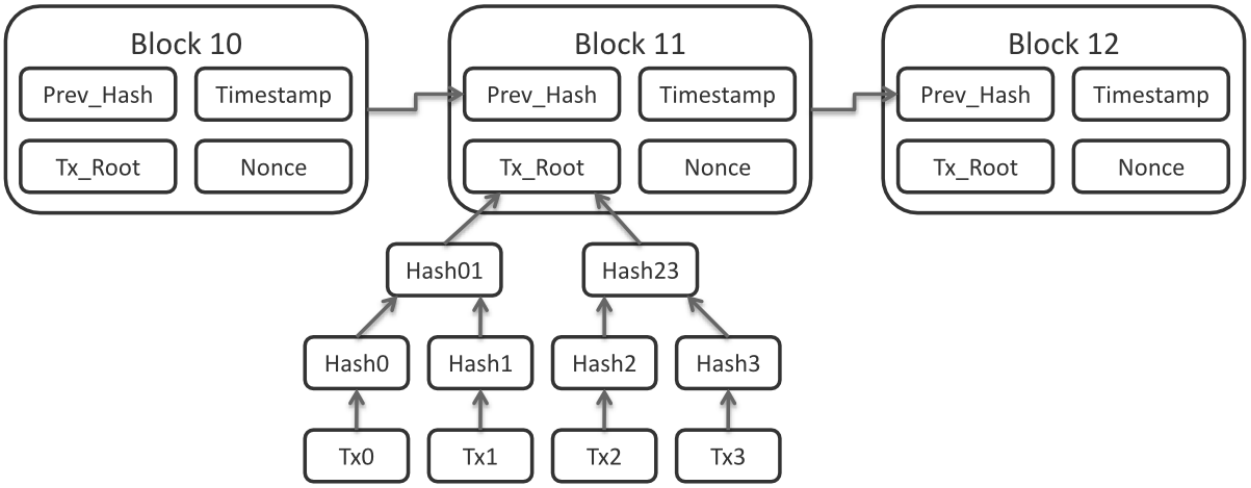


Figura 3.3: Schema di funzionamento di una blockchain
Source: Wikimedia commons, Matthäus Wander, CC BY-SA 3.0

Tutte le informazioni contenute nella catena sono pubblicamente accessibili e vengono condivise tra i *client* che sono connessi alla rete. La catena può essere estesa con nuovi nodi, ma affinché un nodo sia valido il suo hash deve soddisfare una regola nota a tutti i *client*. In generale questa impone che un certo numero di cifre in posizione fissata dell'hash siano uguali tra di loro e l'unico modo per far variare il risultato è cambiare il valore del numero intero all'interno dei dati. Dato che una funzione di hash è difficilmente reversibile, ovvero non è possibile ottenere i dati in input della funzione partendo dall'output, la complessità di questa operazione è molto alta e con l'aumentare del numero di cifre diventa sempre più difficile trovare un valore corretto. La regola della complessità varia in funzione del tempo necessario alla comunità di utenti per generare un nodo: con lo sviluppo di *hardware* in grado di eseguire le operazioni necessarie più velocemente, il sistema si adatterà rendendo più difficile il *mining*.

Quando un *client* riesce a generare un nodo adatto trasmette i valori adatti a dei server detti beacon, che accertano la correttezza del risultato e marcano l'istante temporale in cui il nodo è ritenuto valido, condividendo l'informazione a tutti i *client* vicini. Questi inizieranno a lavorare sul prossimo nodo della catena, mentre l'utente che ha scoperto il nodo ottiene dei vantaggi: ad esempio, nel caso di Bitcoin guadagna una ricompensa economica. In qualsiasi momento è possibile convalidare la creazione dei nodi precedenti applicando la funzione di hash su un nodo; inoltre tanto più la catena di nodi si allunga, tanto più è improbabile che qualcuno sia riuscito ad alterare l'intera sequenza, dato che questo richiederebbe l'aver trovato i valori corretti per generare ogni nodo in essa.

La possibilità di rendere alcuni dati pubblici immodificabili può essere molto utile in molti campi, ma nel caso dello *streaming* affronta dei limiti molto importanti: mentre potrebbe essere possibile salvare interi file video all'interno di un singolo nodo, il throughput di questo metodo è molto basso a causa del tempo necessario al *mining* del nuovo blocco. Si pensi che il tempo medio di generazione di un nodo di Bitcoin si aggira intorno agli otto minuti e trenta secondi. Un altro punto critico è il motivo che dovrebbe spingere gli utenti a eseguire le operazioni di hashing, ovvero la ricompensa: nel caso di un sistema di criptovaluta si può guadagnare denaro, ma parlando di *streaming* video non c'è una ricompensa ovvia.
Una soluzione potrebbe essere quella di ottenere crediti per l'utilizzo del servizio, oppure un certo valore monetario; il problema di questo approccio è che potrebbe non essere autocontenuto nella sola piattaforma, limitandone l'efficacia e l'interesse. Esistono, inoltre, soluzioni meno onerose per la condivisione di file video su grande scala in modo decentralizzato, pur mantenendo la consistenza dei dati, come ad esempio il sistema Torrent, che permette di suddividere delle informazioni in blocchi cifrati univocamente identificati e di condividerli con chiunque ne faccia richiesta; in questo modo è comunque possibile garantire che il contenuto di ciascun blocco non sia stato alterato, senza necessità della complessità aggiunta da una blockchain.

3.2.2 Streaming peer to peer

Un nuovo sistema di distribuzione, particolarmente efficace se accoppiato ad una CDN, è quello di aggiungere al *client* una componente che permetta la condivisione dei chunk di video scaricati tramite una rete *peer-to-peer*. Gli utenti connessi diventano così parte della CDN migliorando aumentandone la portata in modo lineare seguendo la crescita della base di utenza.
Esistono già soluzioni commerciali che adottano questo tipo di soluzione, ottenendo una grande riduzione del carico di rete sui server adibiti alla distribuzione e, di conseguenza, diminuendo i costi di gestione.

Ho avuto modo di conoscere anche soluzioni interamente *peer-to-peer*, come quella ideata dagli sviluppatori di WebTorrent[1]: il sistema sfrutta la rete BitTorrent e permette di condividere i contenuti man mano che vengono scaricati da altri utenti.

Sono state realizzate anche soluzioni *peer-to-peer* per gli *streaming real-time*; una di queste è WebRTC: tra le molteplici funzionalità dello *standard* è inclusa la possibilità di ritrasmettere i contenuti ricevuti agli altri utenti collegati allo stesso *stream*.

3.3 Analisi delle soluzioni esistenti

Propongo di seguito un'analisi delle più famose soluzioni commerciali di *streaming*.

[1] Sito web di WebTorrent: webtorrent.io

3.3.1 YouTube

YouTube utilizza lo *standard* RTSP per l'*inbound* di *stream* video *real-time*, mentre il principale protocollo per la trasmissione è MPEG-DASH; questa scelta permette di ottenere una piattaforma facilmente accessibile, pur perdendo una parte del throughput a causa del protocollo di trasporto. YouTube permette *streaming*-on demand dei video caricati dagli utenti, inoltre ha integrato il servizio con un sistema live-*streaming* ibrido nel Settembre del 2016. Attualmente YouTube supporta video filmati con tecnologie 3D e video 360°, integrati con la piattaforma Google Cardboard. Il formato *standard* utilizzato da YouTube è H.264/MPEG-4 AVC, ma è in corso una migrazione verso il formato WebM.

YouTube associa al suo servizio di *streaming* una serie di servizi paralleli, come il sistema di monetizzazione di AdMob e di pubblicità AdSense e quello di profilazione e analisi Google Analytics.

3.3.2 Netflix

Netflix è un servizio di *streaming on-demand* di serie TV e film ad abbonamento. Il sistema utilizza lo *standard* MPEG-DASH e integra una protezione DRM sui dati inviati. I video, scalati in diverse qualità, vengono spezzati in chunk e richiesti ad uno ad uno dal *client*; la scelta della qualità da visualizzare viene calcolata in base al tempo necessario per scaricare la precedente.

Netflix si avvale dei servizi *cloud* di Amazon per soddisfare le richieste in modo scalabile e distribuito; inoltre, ha stipulato accordi con i principali ISP dei paesi in cui il servizio è attivo, per poter sfruttare le CDN locali e migliorare i tempi di risposta e di caricamento dei contenuti.

3.3.3 Red5 Pro

Red5 Pro è una piattaforma ibrida, che offre servizi di *streaming* facilmente integrabili con applicazioni e siti. Attualmente i protocolli supportati sono RTMP, RTSP, Real Time Messaging Protocol over HTTP (RTMPT), RTSP, HLS, mentre i formati sono FLV e H.264 MP4; inoltre dispone di Software Development Kit (SDK) per Android e iOS. Red5 Pro permette di controllare direttamente il *bitrate* di *outbound* dai server e offre i servizi di *videochat* a due vie o in gruppo. Da poco è stato aggiunto supporto anche a WebRTC.

3.3.4 Contus Vplay

Contus Vplay è una piattaforma di *streaming* ibrida, basata su AWS. Il sistema sfrutta i servizi Simple Storage Service (S3) e Elastic Transcode di Amazon, per la transcodifica dei file video caricati dagli utenti, e il Wowza Streaming Engine[2], il motore di transcodifica leader del mercato e che supporta la conversione "al volo", il quale viene installato in instanze server Elastic Compute Cloud (EC2).

[2]Sito web di Wowza Streaming Engine: www.wowza.com/products/streaming-engine

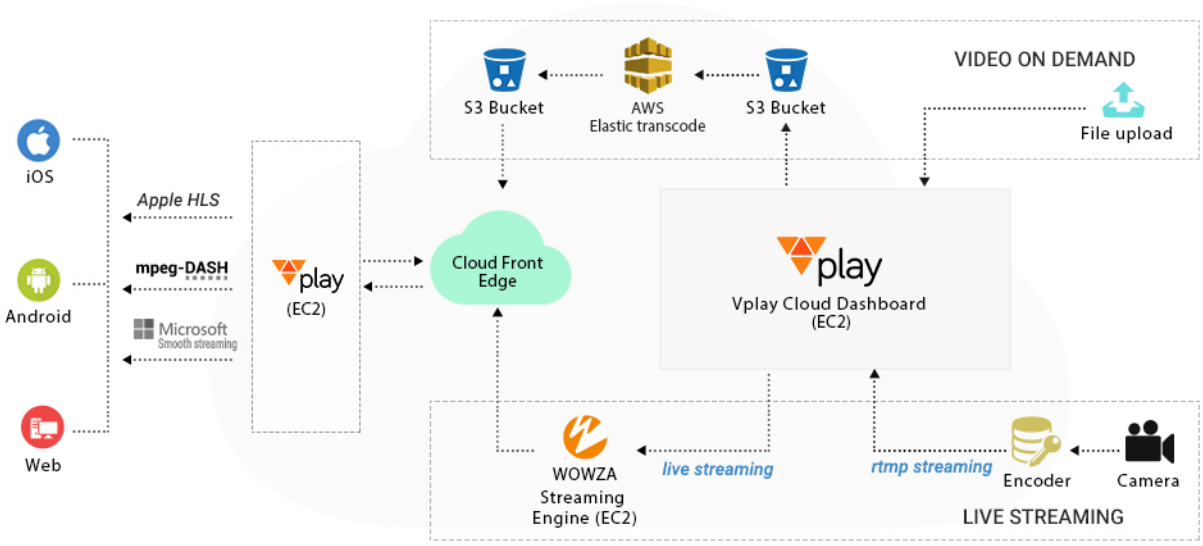


Figura 3.4: Architettura di Contus Vplay
Source: sito web di Contus Vplay www.contus.com/video-on-demand-solution.php

Il servizio è predisposto per soddisfare le richieste dei clienti tramite tre protocolli, Apple HLS, MPEG-DASH e Microsoft Smooth Streaming, sulle principali piattaforme e con multiple qualità del video. La gestione dei contenuti avviene tramite una dashboard che viene mantenuta tramite una combinazione di server EC2 e il servizio CloudFront di AWS. Oltre al sistema di *streaming*, Contus offre una serie di servizi allegati, come analisi sul traffico, profilazione, monetizzazione sugli annunci pubblicitari, cifratura delle connessioni e SDK per le maggiori piattaforme.

3.3.5 Streamroot

Streamroot offre un servizio di *streaming* ibrido. La sua particolarità rispetto alle altre soluzioni è quella di utilizzare un sistema di condivisione *peer-to-peer* dei contenuti scaricati dai clienti. In questo modo ciascun cliente non solo può contare sulla CDN, ma anche su ogni altro utente che sta visualizzando gli stessi contenuti. Secondo i dati riportati dal servizio, questo tipo di distribuzione, unita al sistema di CDN, permette di abbassare il carico sui server di circa il 75%. Streamroot viene utilizzato da molte aziende che hanno necessità di trasmettere ingenti quantità di dati, tra le quali Dailymotion, Eurosport e Canal+.

3.4 Analisi dei requisiti

3.4.1 Analisi preliminare

L'analisi preliminare si è focalizzata sulla definizione degli attori e degli obiettivi e delle funzionalità principali del *software* da produrre.

Il *software* deve gestire l'autenticazione di amministratori e utenti; questi hanno, in entrambi i casi, hanno un'identità univoca e un token di accesso. Gli amministratori hanno la possibilità di creare e rimuovere canali, aggiungere e togliere utenti e gestirne l'autenticazione. Gli utenti possono autenticarsi, inviare messaggi nei canali dove sono registrati o abbandonarne uno.

Ho definito, in collaborazione con gli sviluppatori del *team* che si occupa del progetto, gli attori del sistema, le interazioni che possono intraprendere e i risultati di ciascuna di queste. Per fare ciò ho definito i casi d'uso tramite un diagramma Unified Modeling Language (UML).

3.4.2 Definizione dei casi d'uso

Segue una versione ristretta della definizione dei casi d'uso trovati durante l'analisi dei requisiti.

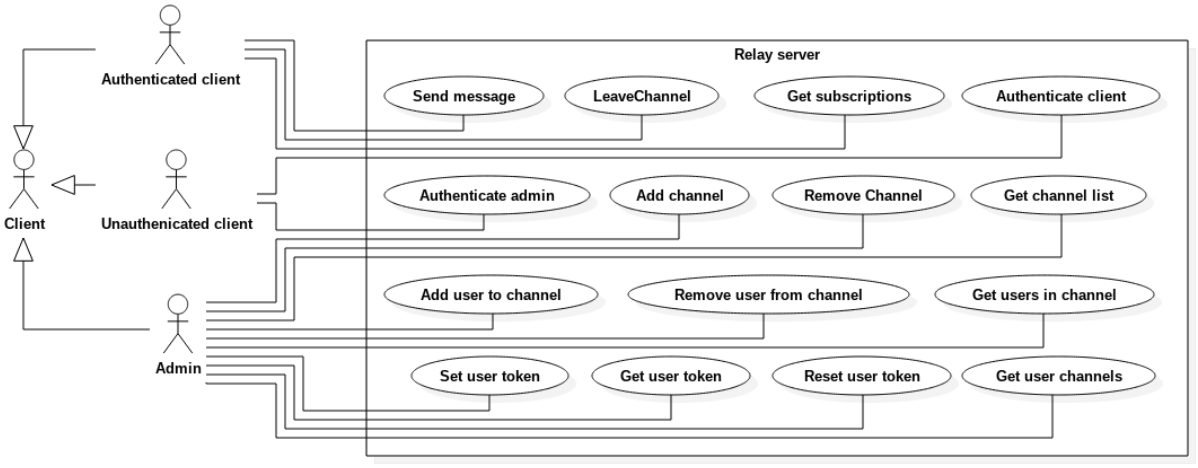


Figura 3.5: Diagramma dei casi d'uso del sistema

Attori

- **Client**: un utente che interagisce con il sistema;
- **Unauthenticated Client**: un *client* non ancora autenticato;
- **Authenticated Client**: un *client* che ha eseguito l'autenticazione ed è registrato all'interno di almeno un canale;
- **Admin**: un utente o un sistema automatico che ha il ruolo di amministratore del servizio offerto dal server.

Casi d'uso di Unauthenticated Client

- **Authenticate client**: L'utente esegue l'autenticazione su sistema come *client*;
- **Authenticate admin**: L'utente esegue l'autenticazione su sistema come amministratore.

Casi d'uso di Authenticated Client

- **Send message**: L'utente invia un messaggio all'interno di un canale in cui è registrato;
- **Leave channel**: L'utente lascia il canale e viene deregistrato;
- **Get subscriptions**: L'utente ottiene la lista dei canali nei quali è registrato;

Casi d'uso di Admin

- **Add channel**: L'Amministratore crea un nuovo canale;
- **Remove channel**: L'amministratore elimina un canale;
- **Get channel list**: L'amministratore ottiene la lista dei canali registrati;
- **Add user to channel**: L'amministratore registra un utente in un certo canale;
- **Remove user from channel**: L'amministratore rimuove un utente da un canale;
- **Get users in channel**: L'amministratore ottiene la lista degli utenti registrati in un canale;
- **Set user token**: L'amministratore imposta il token di autenticazione di un utente;
- **Get user token**: L'amministratore ottiene il token di autenticazione di un utente;
- **Reset user token**: L'amministratore resetta il token di autenticazione di un utente;
- **Get user channels**: L'amministratore ottiene la lista dei canali in cui un certo utente è registrato.

3.5 Struttura della piattaforma

Una volta definiti i casi d'uso, ho proceduto alla scelta dei protocolli di rete da utilizzare e la progettazione strutturale del sistema.

3.5.1 Protocolli

Ho eseguito la scelta dei protocolli in collaborazione con il *team* di sviluppatori che lavorava sul *client* Android, per trovare la migliore soluzione in termini di latenza, throughput e complessità di implementazione.

I protocolli UDP sono stati scartati a causa dei problemi con *firewall* riportati precedentemente. Le specifiche del servizio richiedono che sia possibile inviare messaggi di qualsiasi tipo, sia atomici, sia *stream* di dati. Per semplificare l'invio di dati generici, senza essere necessariamente legati ad un certo protocollo dello strato applicativo, abbiamo deciso di utilizzare dei WebSocket ed esistono implementazioni *open-source* dello *standard* per la maggior parte delle piattaforme, inclusi iOS e Android e Java, rendendo, quindi, l'implementazione di *client* e server più semplice.
Lo standard WebSocket utilizza TCP per trasmettere stream di byte, esattamente come succede sui comuni socket, ma integra una serie di controlli di rete e di sicurezza, fondamentali per l'utilizzo sicuro tramite Internet.

3.5.2 Diagramma delle classi

Segue in Fig. 3.6 il diagramma UML delle classi semplificato.

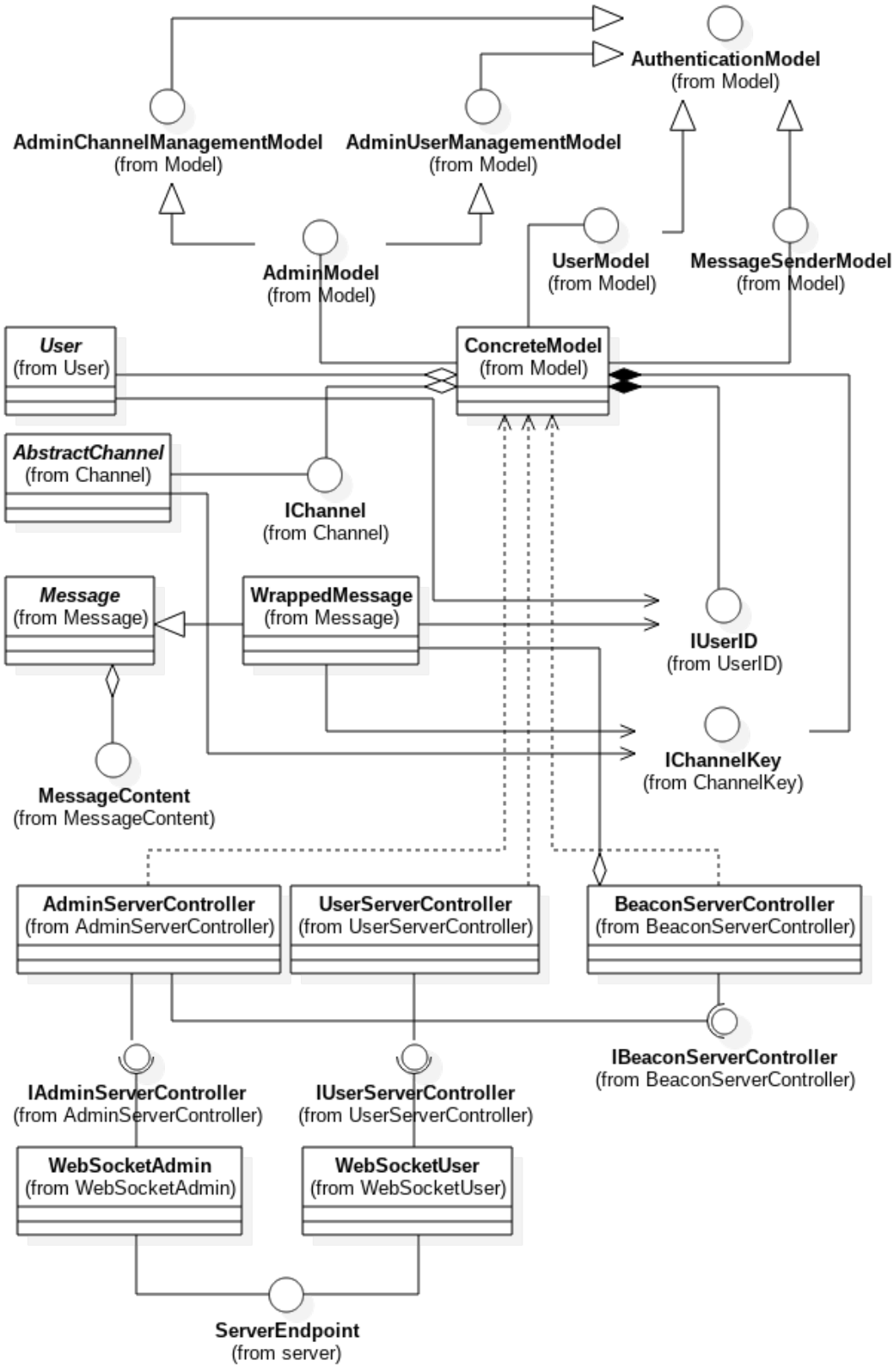


Figura 3.6: Diagramma delle classi del sistema

ConcreteModel e interfacce del modello

Il modello è responsabile del salvataggio e della gestione delle informazioni legate agli utenti, ai canali, alle registrazioni e ai messaggi da inviare. Per organizzare chiaramente le sue funzionalità ho utilizzato interfacce molto specifiche.

IUserID e IChannelKey

Per identificare univocamente un utente o un canale ho designato due interfacce per due classi di oggetti che permettono di ottenere un identificativo univoco sotto forma di stringa; in questo modo è possibile referenziare uno di questi oggetti indicandone il solo ID.

User

La classe astratta User definisce un utente e permette di caricare ed estrarre informazioni sui profili degli utenti.

IChannel e AbstractChannel

IChannel e AbstractChannel individuano le funzionalità e i dati di base di un canale, come il suo nome e il suo scopo.

Message, WrappedMessage e MessageContent

Message rappresenta un messaggio da inviare, il cui contenuto è un insieme di MessageContent. Al fine di garantire la possibilità di inviare qualsiasi tipo di contenuto tramite un messaggio, MessageContent eredita direttamente dalla classe Java Serializable, ma non impone altri limiti. WrappedMessage aggiunge ad un messaggio le informazioni per spedirlo, ovvero la ChannelKey e l'UserID del destinatario.

Controller di Admin, User e Beacon

I tre controller del sistema sono singleton e gestiscono le interazioni e aggiornano il modello. AdminServerController e UserServerController eseguono, rispettivamente, le azioni ricevute da un amministratore e da un utente. BeaconServerController è incaricato della creazione dei messaggi e del loro invio; utilizza un pattern command e una coda intelligente.

3.6 Tecnologie utilizzate

3.6.1 Linguaggi — Java

Il linguaggio Java è stato scelto innanzitutto per la possibilità di eseguire il codice su qualsiasi piattaforma che offra una JVM; in questo modo il sistema può essere eseguito su server con diversi sistemi operativi senza la necessità di riscriverne componenti o di essere ricompilato. Un altro vantaggio è quello di poter riutilizzare parte del codice scritto nel *client* Android, riducendone i tempi di sviluppo e aumentando la qualità del codice.

3.6.2 Piattaforma — Apache Tomcat 8

Un servizio Java che espone dei WebSocket ha bisogno di una piattaforma che ne gestisca gli accessi ovvero un componente che implementi la classe JavaServer. Per questo progetto ho deciso di utilizzare Apache Tomcat 8 per il suo supporto nativo ai WebSocket. Inoltre, Tomcat 8 è il

server Java *open-source* più diffuso, la sua documentazione è sufficientemente chiara e completa e attualmente è l'unico a supportare completamente l'ultima versione di Java.

3.6.3 Tipo di dati scambiati — JSON vs XML

Per poter inviare messaggi tramite un WebSocket è necessario definire una codifica. L'implementazione *standard* di Java permette di utilizzare stringhe di byte, stringhe di testo oppure di definire un codificatore e un decodificatore per gestire oggetti più complessi. Per semplificare lo sviluppo della piattaforma ho preferito utilizzare una codifica testuale, che mi permettesse di eseguire *test* automatici e manuali tramite semplici messaggi testuali. Le due codifiche maggiormente utilizzate che permettono di descrivere oggetti anche molto complessi sono JSON e XML. Entrambe le codifiche soddisfano pienamente i requisiti necessari, ma JSON è molto meno prolisso di XML; per questo motivo ho preferito tale linguaggio.

3.6.4 Strumenti di supporto

Strumenti di codifica

Lo strumento che ho utilizzato per codificare ed eseguire i *test* sul sistema è stato l'IDE IntelliJ Idea, sviluppato da JetBrains, nella versione *open-source* "Community Edition". Questo *editor*, progettato appositamente per Java, è ricco di utili strumenti per velocizzare la codifica e l'analisi, come lo smart code completion, per i suggerimenti contestuali per il completamento del codice, i code template, per la rapida generazione di strutture di codice prestabilite, e sistemi di analisi e di automatizzazione dei task, come l'aggiornamento delle dipendenze o l'esecuzione dei *test*.

Strumenti di versionamento

Lo strumento di versionamento utilizzato durante il progetto è stato quello comunemente utilizzato dall'azienda, ovvero Git, con l'ausilio dei server di Bitbucket per l'automatizzazione di alcuni *test*. Il vantaggio di aver utilizzato Git durante lo sviluppo è stato quello di poter condividere velocemente le ultime modifiche con gli altri sviluppatori, pur mantenendo baseline sicure e testate.

Strumenti di test e verifica

Ho scelto di utilizzare il *framework* JUnit 4 per eseguire *test* di unità e di integrazione dopo un'attenta analisi delle sue funzionalità. Il sistema di *test* si è rivelato semplice ed efficace; mi ha permesso di realizzare *test* significativi con poco sforzo e ha contribuito alla buona qualità del codice. L'integrazione dell'IDE usato con JUnit permette anche di esportare i dati dei *test*, del coverage e delle analisi in formato HTML, facilmente consultabile anche senza strumenti di sviluppo dedicati. JUnit 4 è stato utile anche per eseguire benchmark per valutare l'efficacia del prodotto, tramite la realizzazione di una simulazione di conversazione tra un numero definito di utenti.

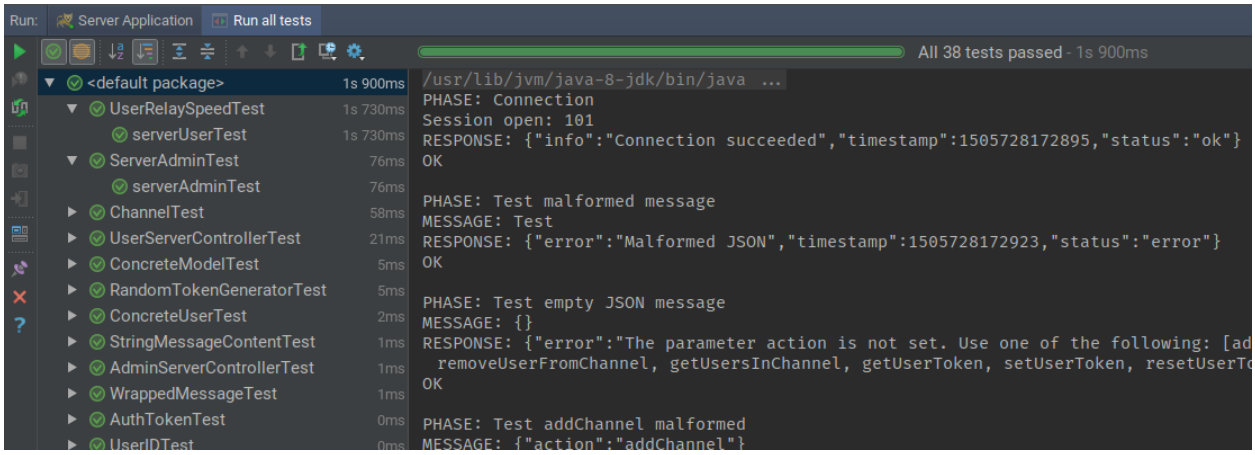


Figura 3.7: Sistema integrato di esecuzione dei *test* in IntelliJ Idea

Strumenti di analisi di rete

Durante i *test* sulla sicurezza delle trasmissioni ho dovuto assicurarmi che il sistema inviasse contenuti sensibili non cifrati; per questo motivo ho avuto necessità di registrare le conversazioni tra alcuni utenti simulati. Per eseguire queste operazioni ho utilizzato Wireshark, un *software open-source* per l'analisi del traffico di rete. Wireshark permette di effettuare sniffing del traffico di rete, di visualizzare ciascun pacchetto e di filtrare facilmente i contenuti d'interesse. Tramite un'analisi eseguita con questo strumento ho potuto garantire che tutti i dati scambiati fossero cifrati end to end tramite Transport Layer Security (TLS).

Strumenti di continuous integration

Per poter garantire l'indipendenza dalla piattaforma di sviluppo, in particolare durante l'esecuzione dei *test*, ho utilizzato un sistema Docker. Questo strumento permette di configurare facilmente l'installazione e la configurazione di un *software* in sviluppo su una macchina virtuale minimale e di eseguire *test* e script su di essa. Tramite l'integrazione con Bitbucket ho potuto automatizzare queste attività, fornendo una maggiore qualità al codice senza dover svolgere compiti ripetitivi e onerosi in termini di tempo.

Tramite le *pipeline*, il sistema di configurazione delle operazioni sui container Docker di Bitbucket, ho potuto automatizzare anche il deploy dell'applicazione, limitando i tempi di manutenzione necessari al suo aggiornamento manuale.

Strumenti di debugging

Durante la codifica, specialmente durante la scrittura delle componenti per la gestione dei messaggi, mi è stato molto utile la possibilità di eseguire parti del programma in modalità debug. Questa permette di interrompere temporaneamente l'esecuzione in punti prestabiliti, semplificando la comprensione della sequenza di azioni avvenute e fornendo maggiori informazioni sulla *stack trace* del programma, ovvero la pila di funzioni che stanno venendo eseguite.
IntelliJ Idea integra questa modalità e raccoglie le informazioni ottenute in un'intergaccia semplice da comprendere. L'editor mostra anche lo stato delle singole variabili, sia su una scheda apposita, sia direttamente sul codice.

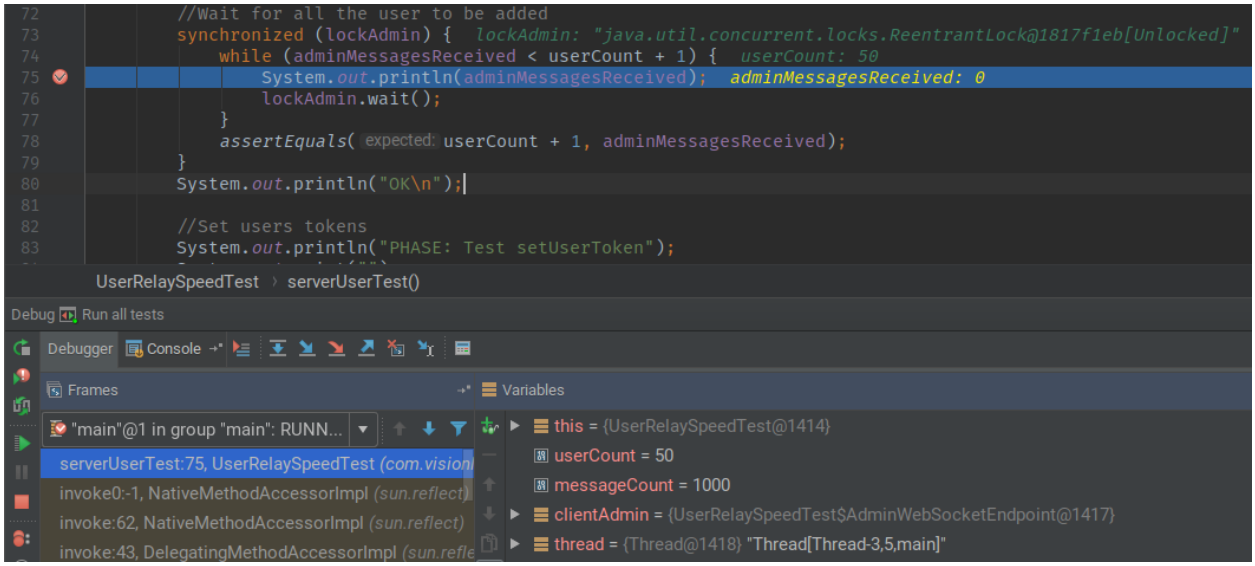


Figura 3.8: Integrazione della modalità di debug in IntelliJ Idea

3.7 Qualifica

3.7.1 Verifica

la verifica di un prodotto è l'insieme delle operazioni atte a garantirne la qualità e il rispetto delle specifiche definite durante la progettazione. Può essere riassunto nella domanda "*did I build the system right?*"

Analisi statica

L'analisi statica consiste nei *test* che possono essere eseguiti sul prodotto senza compilare ed eseguire il codice. In questo tipo di *test* possiamo trovare, quindi, analisi grammaticali del codice, *test* su errori tipici, *test* sui file di configurazione, controlli sulla duplicazione del codice, ecc.

L'analisi statica del prodotto è stata delegata ai *test* interni dell'IDE utilizzato, IntelliJ Idea. Questo esegue molti dei *test* sopra citati in modo automatico, mentre il codice viene scritto, fornendo utili suggerimenti.

Una funzionalità che mi è sembrata particolarmente utile durante la verifica è il controllo automatico sulle classi importate: l'IDE controlla quali classi importate nel file vengono effettivamente utilizzate, segnala le mancanti offrendo suggerimenti e permette di eliminare quelle non necessarie.

Analisi dinamica

Con analisi dinamica si intendono le operazioni di controllo fatte sul codice compilato ed eseguito. Questi *test* includono asserzioni sullo stato dell'esecuzione ed esecuzioni controllate dei componenti, sia singolarmente, sia facendoli interagire l'uno con l'altro.

Ho programmato l'analisi dinamica del prodotto durante l'analisi e la progettazione, per poi essere implementata durante la codifica. Ho realizzato i *test* di unità delle singole componenti e di integrazione, verificando che eseguissero i compiti assegnati dalle interfacce nel modo corretto, per poi passare ai *test* di sistema, eseguiti tramite simulazioni controllate.

Un vantaggio importante nella scelta di eseguire *test* di sistema automatici è stata quella di poter valutare il prodotto nella sua interezza, generando un report chiaro sullo stato di sviluppo e sulle prestazioni ottenute. In particolare, dopo una prima fase di codifica ho scoperto un errore sulla gestione di alcuni task da parte del componente dedicato al reinvio dei messaggi proprio grazie a un *test* di questo tipo; alcuni messaggi non venivano inviati e le sessioni stabilite con i *client* venivano chiuse anticipatamente se il numero di utenti superava una certa soglia. La simulazione ha evidenziato questo problema e mi ha permesso di individuare e correggere il bug in poco tempo.

3.7.2 Validazione

La validazione di un prodotto consiste nel controllare che tutti i requisiti del prodotto vengano effettivamente soddisfatti e può essere riassunta nella domanda "*did I build the right system?*"

Validazione interna

Dato che il prodotto è destinato ad essere un componente di una più grande piattaforma, ancora in sviluppo, la validazione è stata interna. Il project manager e il mio tutor aziendale hanno eseguito tutti i *test*, le simulazioni e, come prova finale, hanno provato a comunicare tramite due *client* WebSocket.

4 Analisi retrospettiva

In questo capitolo realizzerò una breve analisi retrospettiva dello *stage* e del progetto, dei risultati ottenuti, del raggiungimento degli obiettivi, e delle conoscenze che mi sono state necessarie o utili.

4.1 Obiettivi

4.1.1 Obiettivi personali

4.1.2 Obiettivi dell'azienda

A seguito ai *test* e alla validazione finale, è stato confermato dal mio tutor aziendale che tutti gli obiettivi fissati dall'azienda, sia quelli obbligatori, sia quelli desiderabili, sono stati raggiunti.

4.2 Bilancio formativo

Dal mio punto di vista, il tirocinio è stato un'ottima esperienza formativa. Ho avuto la possibilità di espandere le mie conoscenze di seguire un progetto in quasi totale autonomia, un'occasione per accrescere la mia capacità di farmi carico di responsabilità. È stato sicuramente molto importante avere la possibilità di confrontarmi con l'ambiente lavorativo di un'azienda ben avviata e di comprenderne i ritmi e i rapporti con le varie cariche.

Per quanto riguarda le conoscenze acquisite, ho potuto studiare e conoscere più approfonditamente protocolli e formati video; questi ultimi non fanno parte del programma del corso laurea in informatica che mi accingo a concludere e sono stati particolarmente interessanti, data la loro natura prettamente materiale. Altre importanti nozioni che ho appreso durante lo *stage* riguardano le blockchain, delle quali conoscevo solamente le applicazioni pratiche e non il loro funzionamento più teorico, e le reti *peer-to-peer*, in particolare la rete BitTorrent, di cui non conoscevo né le specifiche, né le possibilità nell'ambito dello *streaming* video. Ho anche avuto modo di utilizzare praticamente le conoscenze di programmazione concorrente e di Java, apprese durante i corsi, e di altre tecnologie di supporto, come Git e Docker, migliorando considerevolmente le mie competenze.

4.3 Conoscenze desiderabili

Le nozioni apprese durante questo corso di laurea mi sono state di grande aiuto, non solo come competenze necessario allo svolgimento delle attività all'interno di un'azienda, ma anche come base per un più rapido apprendimento di nuove tecnologie e nuove tecniche, come ho potuto constatare durante il tirocinio all'interno di Vision Lab Apps. Durante il tirocinio ho avuto necessità di conoscere più approfonditamente alcuni aspetti di programmazione e di reti, in particolare le basi di programmazione funzionale in Java, utilizzate per semplificare la codifica del lavoro in parallelo sul server, e l'uso pratico dei protocolli di rete *real-time*.

Tra le conoscenze che più mi sono state utili durante lo *stage* posso sicuramente indicare quelle fornitemi dai corsi di Ingegneria del Software e Programmazione Concorrente e Distribuita: il primo per le competenze necessarie all'organizzazione e alla gestione di un progetto, fondamentali per la buona riuscita di questi, nonché i principali design pattern da utilizzare durante la progettazione dei prodotti. Penso, proprio questi ultimi potrebbero essere più utili se ci fosse la possibilità di applicarli in più di un progetto, durante i corsi, magari durante il corso di Programmazione ad Oggetti oppure tramite progetti opzionali dei corsi del terzo anno, come punti aggiuntivi alla valutazione finale. Altro corso le cui nozioni mi sono state particolarmente utili durante il tirocinio è stato il corso di Programmazione Concorrente e Distribuita; in particolare per lo studio di Java e dell'utilizzo di processi paralleli durante l'esecuzione dei processi, fondamentali nel caso di applicazioni di rete che debbano servire un numero enorme di *client* nel minor tempo possibile. Anche in questo caso penso che il corso ci prepari in modo più che soddisfacente sia sul lato teorico che sul lato pratico, ma che un progetto opzionale potrebbe aiutare a fissare maggiormente le capacità acquisite.

4.4 Futuri sviluppi del progetto di *stage*

Durante lo sviluppo della piattaforma di *streaming* sono nate nuove idee riguardo l'implementazione di altri servizi da aggregare alla piattaforma. Tali ampliamenti porterebbero il prodotto ad una maggiore competitività nel mercato e ad un migliore qualità di servizio ai clienti che lo useranno.

La necessità del prodotto è nata nel momento in cui è stato necessario far comunicare due dispositivi Android, in cui stava venendo testato il *client* in sviluppo, senza incorrere nei problemi legati ad una comunicazione diretta e aggiungendo funzionalità come la struttura a canali del servizio, che permette anche a più di due *client* di comunicare assieme.

Attualmente il servizio permette di trasmettere dati tra più dispositivi, ma non esegue alcuna elaborazione dei contenuti. Questo modo di operare è stato scelto per poter avere un servizio di scambio di dati generico altamente scalabile, agnostico sui contenuti dei messaggi. Il passaggio successivo, un a volta reso stabile il *client* Android, sarà quello di realizzare i tre componenti dell'architettura impegnati, rispettivamente, nella ricezione dei flussi, nella loro transcodifica e successivamente nella loro distribuzione.

Un'ulteriore tecnologia che Vision Lab Apps vuole integrare nel prodotto è quella della condivisione, tramite rete *peer-to-peer*, dei dati. Come visto tramite nella mia relazione, la quantità di lavoro spostato sui *client*, a vantaggio dei server di distribuzione, è tale da rendere molto fruttuoso un investimento nello sviluppo di questa funzionalità in una piattaforma di *streaming*.

4.5 Valutazione personale

Concludo questo Rapporto di fine Stage con una breve valutazione personale.

Mi ritengo soddisfatto del lavoro svolto. Ho avuto modo di studiare, analizzare, progettare e sviluppare il progetto nella sua interezza secondo i requisiti tecnologici, temporali e qualitativi richiesti. Il servizio realizzato è completo e il suo codice verrà utilizzato dall'azienda e integrato nel *client* Android; inoltre, gli studi eseguiti hanno portato nuove idee al *team* di sviluppo e suscitato grande interesse tra il personale.

Posso, infine, essere felice di aver potuto lavorare in un'azienda emergente, che punta molto sull'innovazione e sui giovani, con un gruppo di persone entusiaste ad un argomento interessante e particolarmente sentito negli ultimi tempi.

Glossario

Application Programming Interface (API) In informatica con API 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, in genere tra l'*hardware* e il programmatore o tra *software* a basso e quello ad alto livello, semplificando, così, le attività di programmazione. 16, 23

AWS Amazon Web Services (AWS) è una collezione di servizi di *cloud* computing *on-demand* offerta da Amazon. 12, 36, 37

BSD Berkeley Software Distribution. 28

CD Compact Disc. 28, 48

Computer-Aided Design (CAD) Computer-Aided Design (CAD) indica un *software* volto all'utilizzo di tecnologie per la computer grafica per supportare l'attività di progettazione di modelli, soprattutto 3D. 24

Content Delivery Network (CDN) Una Content Delivery Network (CDN) è una rete di computer collegati tra di loro tramite una rete Internet, che collaborano in maniera trasparente per distribuire contenuti; in genere *streaming* audio e video. 14, 32, 35–37

Content Management System (CMS) Un Content Management System (CMS) è un *software* di supporto alla creazione, modifica e gestione di contenuti digitali. 19, 24

CVS É detto Concurrent Versioning System (CVS) un *software* che implementa un sistema di controllo di versione. Il sistema mantiene organizzati i cambiamenti fatti a un certo numero di file e permette a molti sviluppatori di collaborare accedendo alle stesse risorse. 17

DCT Discrete Cosine Transform. 28, 50

Digital Audio Broadcasting (DAB) Il Digital Audio Broadcasting (DAB) è uno *standard* di radiotrasmissione digitale che permette la trasmissione di un segnale audio, in genere programmi radiofonici, con una qualità paragonabile a quella di un CD audio; supporta il multiplexing, la trasmissione di più tracce contemporaneamente sullo stesso canale e utilizza l'algoritmo di compressione HE-AAC, che prevede resistenza ai disturbi e un'ottima efficienza. 28

Docker Docker è un *software open-source* per la virtualizzazione di sistemi operativi in "container" isolati e controllati. Il metodo utilizzato da Docker sfrutta il sistema di isolamento delle risorse del *kernel* Linux, permettendo la coesistenza di più container sulla stessa macchina e limitando gli sprechi di risorse collegati all'utilizzo di una macchina virtuale completa. 24, 43

DRM Digital Rights Management. 33, 36

EC2 Elastic Compute Cloud. 36, 37

Gantt Un diagramma di Gantt è un diagramma a barre pensato per mostrare su una scala temporale le attività di un processo, le risorse che occupano, il tempo impiegato e le dipendenze di ciascuna. Questo diagramma è molto utile per stimare i tempi di sviluppo di un prodotto e fissare milestone e scadenze adeguate. 26

GIF Graphics Interchange Format. 29

HLS HTTP Live Streaming. 30, 32, 36, 37

HTML Hypertext Markup Language. 26, 42

HTTP HyperText Transfer Protocol. 30, 49

IEEE Institute of Electrical and Electronics Engineers. 33

Internet Service Provider (ISP) Un Internet Service Provider (ISP) è una struttura commerciale o un'organizzazione che fornisce l'accesso a servizi Internet, dietro la stipulazione di un contratto. 32, 36

IoT Per Internet of Things (IoT) ci si riferisce all'estensione di Internet agli oggetti comuni, che diventano intelligenti e comunicano dati su sé stessi e sul mondo che li circonda e allo stesso tempo accedere ad informazioni altrove nella rete. 9, 20, 23, 24, 50

JPEG Joint Photographic Experts Group. 27, 28

JSON JavaScript Object Notation. 7, 42

JVM Java Virtual Machine. 14, 16, 41

LTE Long Term Evolution. 33

MPEG Moving Picture Experts Group. 29

MPEG-DASH Dynamic Adaptive Streaming over HTTP. 30, 32, 36, 37

NAT Network Address Translation. 33

OSI Open Systems Interconnection. 29, 30

PNG Portable Network Graphics. 26

Program Evaluation and Review Technique (PERT) Program Evaluation and Review Technique (PERT) è uno strumento utilizzato per la gestione di un progetto pensato per analizzarne e rappresentarne i task necessari al suo completamento. 26

Project manager Il *project manager* di un progetto è il responsabile dell'organizzazione dei processi e della loro pianificazione all'interno di esso. 10

RRCP Realtek Remote Control Protocol. 30

RTMCP Real Time Message Control Protocol. 30

RTMP Real Time Messaging Protocol. 30, 36

RTMPT Real Time Messaging Protocol over HTTP. 36

RTP Real-time Transport Protocol. 30, 31

RTSP Real Time Streaming Protocol. 30, 36

S3 Simple Storage Service. 36

SCTP Stream Control Transmission Protocol. 30

SDK Software Development Kit. 36, 37

SEO Si definisce Search Engine Optimization (SEO) l'attività di ottimizzazione dei contenuti di una pagina web per l'indicizzazione da parte dei motori di ricerca. 9

TCP Transmission Control Protocol. 29–31

Thin client Un *thin client* è un *client* leggero pensato per connettersi a un server remoto che esegue tutte le operazioni sensibili. Si contrappone al convenzionale *fat client* nel quale è il *client* stesso ad eseguire la maggior parte delle operazioni e può comunicare parte dei dati ad altri dispositivi. 24

TLS Transport Layer Security. 42

Tomografia Assiale Computerizzata (TAC) In medicina la Tomografia Assiale Computerizzata (TAC) è una metodica diagnostica per immagini che consente di riprodurre sezioni o strati ed effettuare elaborazioni tridimensionali dei dati ottenuti. 24

Transcodificatore Descrizione TRANSCODIFICATORE. 32

Trasformata Discreta del Coseno La trasformata discreta del coseno (DCT) è una funzione che provvede alla compressione spaziale, capace di rilevare variazioni tra un area e quella contigua. 28

Ubiquitous computing L'ubiquitous computing è un nuovo modello di interfaccia uomo macchina, secondo il quale ogni persona, nelle sue azioni quotidiane, può entrare in contatto con un enorme numero di dispositivi elettronici, più o meno specializzati, che comunicano tra loro e possono collaborare a uno scopo. Si differenzia dal precedente modello uomo-macchina per la completa integrazione dell'elaborazione delle informazioni all'interno del singolo dispositivo, senza dipendere da un nodo computazionale esterno. 9

UDP User Datagram Protocol. 29–31

UML Unified Modeling Language. 38, 39

Video Home System (VHS) Video Home System è un sistema di registrazione video in formato analogico su supporto magnetico. 28

VPS Un Virtual Private Server (VPS) è un'istanza di un sistema che viene eseguito in un ambiente virtuale. 12

Wearable Si dice *wearable* un dispositivo elettronico indossabile on impiantabile. In generale questi dispositivi offrono delle funzionalità di notifica legate agli smartphone oppure contengono sensori per la rilevazione di attività fisica e sono un esempio di dispositivo IoT. 9, 20

XML Extensible Markup Language. 7, 42

Bibliografia