POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Laurea Magistrale

Progettazione e sviluppo di una applicazione per il monitoraggio del coinvolgimento emotivo nei processi didattici interattivi

Relatore
Prof. Giovanni Malnati

Candidato
Francesco Politano
matricola 231747

Ottobre 2019

Sommario

Questa tesi vuole fornire ai docenti uno strumento moderno per raccogliere dagli studenti dei feedback in maniera puntuale. Questa necessità affonda le sue radici negli studi sulla didattica dello psicologo di origini ungheresi *Mihaly Csikszentmihaly*. Negli anni '70 lo psicologo ungherese ha dato alla luce la sua teoria più brillante che teorizza l'insorgere sotto alcune condizioni di uno stato psicofisico di estrema concentrazione e produttività detto esperienza ottimale o stato di Flow. Lo psicologo scoprì che questo era un momento di massima efficacia, sicuramente desiderabile per ottenere performance superiori. Scoprì quindi che tutti possono entrare in uno stato di Flow se sono rispettate alcune semplici condizioni ambientali.

La teoria del Flow applicata alla didattica prevede che sia possibile mantenere gli studenti in uno stato tale in cui le performance sono al top ed è incentivata la capacità d'apprendimento. Questo può avvenire solo quando è mantenuto un perfetto bilanciamento tra sfida e capacità degli stessi, con un lento ma costante aumento della difficoltà. Per poter stabilire con precisione questi parametri si rende necessario un costante puntuale monitoraggio dello stato d'animo e dei progressi didattici degli studenti.

Questa tesi si propone di illustrare, progettare ed implementare un prototipo di una applicazione atta a fornire ai docenti queste funzionalità. Didactic, il nome che ho dato a questo applicativo, implementa un sistema di messaggistica istantanea e delle funzionalità di creazione e somministrazione di sondaggi e di consultazione dei risultati statistici.

Per questo sviluppo ho usato strumenti moderni ed un approccio contemporaneo allo sviluppo software per il web. L'applicativo è scritto in JavaScript su runtime Node.js, la persistenza dei dati è affidata ad un database MongoDB e lo sviluppo front end è stato sviluppato con Angular 2+. Ho inoltre implementato un basilare meccanismo di CI/CD usando Jenkins.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

In questa introduzione si effettua una panoramica sul concetto psicologico di Flow e di come gli studi dello psicologo ungherese *Mihaly Csikszentmihalyi* possono essere trasportati nell'ambito della didattica. Si noterà inoltre che per una applicazione pratica delle teorie di Csikszentmihalyi è necessaria una forte interazione tra studenti e docenti ed un sistema capace di raccogliere feedback costanti ed onesti. L'obbiettivo di questa tesi sarà di realizzare un'applicativo che supporti il docente nell'applicazione di una metodologia di didattica che incentivi gli studenti e li motivi migliorando il processo di apprendimento.

Oltre a questa introduzione la tesi consta di altre due sezioni principali. Nel secondo capitolo si affronterà la progettazione del software dalla definizione dei requisiti alla architettura software, passando per una breve informativa sulle tecnologie utilizzate. Nel terzo capitolo, quello riguardante l'implementazione della soluzione software in esame, ci si concentra sulla realizzazione pratica del prodotto per quanto riguarda il codice e le configurazioni necessarie all'operatività.

1.1 L'esperienza ottimale: il Flow

Le teorie scientifiche sul Flow nacquero negli anni settanta quando lo psicologo *Mihaly Csikszentmihalyi* si appassionò per quegli artisti che si perdevano nel loro lavoro, dimenticando lo scorrere del tempo ed i loro bisogni primari.

Scoprì che questa cosa era comune anche tra sportivi, musicisti, professionisti specializzati. Questo momento di massima efficacia viene anche definito "esperienza ottimale".

Mihaly definì queste figure come "persone auteliche": individui che perseguono i loro obiettivi per il gusto di farlo, non per una ricompensa o uno scopo.

Per lo psicologo ungherese queste persone sono naturalmente curiose, perseveranti e disinteressate. L'attività in cui si cimentano trova in se stessa e nel proprio svolgimento lo scopo del suo realizzarsi.

Ma l'umanità era al corrente di questo stato fin dall'antichità: in alcune culture infatti il lo stato di Flow era conosciuto come "estasi". Nella esperienza di flow tutta la nostra attenzione è rivolta all'obbiettivo, perfino la percezione di noi stessi è interrotta. Le neuroscienze ci dicono che il nostro cervello non può processare più di 110 bit di informazioni al secondo, ecco che in uno stato di flow liberiamo parte

di questa capacità alienandoci e raggiungendo quindi uno stato di concentrazione profonda, galvanizzante.

Quali sono le sensazioni legate allo stato di Flow?

- Diminuzione dell'autoconsapevolezza.
- Unione del pensiero e dell'azione.
- Cancellazione della paura di fallire.
- Distorsione del tempo.
- Eliminazione delle distrazioni.

1.1.1 Come entrare in uno stato di Flow

Tutti possono entrare in uno stato di Flow e tutti l'hanno sperimentato più e più volte nel corso della propria vita. Non c'è un modello principale riconosciuto dalla comunità scientifica ma esistono dei punti chiave accomunanti le varie teorie:

- L'obiettivo deve essere ben definito e deve essere chiaro il senso del progresso.
- Il task deve essere intrinsecamente appagante, dobbiamo portarlo a termine per il puro gusto di farlo. Si parla in questo caso di attività *autoteliche*.
- Il feedback deve essere chiaro ed immediato. Questo accade facilmente quando l'attività riguarda competizioni, punteggi e classifiche.
- Il task deve rappresentare una sfida ma non deve risultare invalicabile, o sfocerebbe in frustrazione. L'obbiettivo deve quindi pareggiare le capacità dell'individuo o porsi poco al di sopra di esse.

Nei suoi studi *Csikszentmihalyi* ha ridotto a due gli elementi condizionanti per raggiungere uno stato di Flow: questi sono il livello della sfida e le capacità in possesso per affrontarla. Gli studi hanno permesso di calcolare per ciascun individuo una media; si tratta del livello medio di sfida ed abilità, che risulta diverso da persona a persona.

Conoscendo questo valore è possibile prevedere quando l'individuo entrerà in Flow e agire sul suo stato agendo sul livello di sfida o, più inverosimilmente, sul suo livello di abilità.

Bisogna notare che più ci si allontana dallo stato di Flow più gli stati acquisiscono connotazione negativa. Tipicamente un individuo a suo agio e che sta eseguendo dei task vicini alla sua zona di comfort si troverà più spesso negli stati direttamente adiacenti a quello di Flow, sono due aree complementari ed ideali:

- **Arousal** (eccitazione): tipicamente è lo stato in cui avvengono la crescita ed il miglioramento. Si è infatti sotto sfida e si esce dalla comfort zone, pur senza cadere nell'ansia. Da qui è semplice ricadere nuovamente in uno stato di Flow agendo sulle proprie abilità ed affinandole.
- **Control** (controllo): in questa zona l'attività non è più molto stimolante, ci si trova a proprio agio perché si ha il pieno controllo sull'azione. Anche in questo caso è possibile muoversi verso una situazione di Flow agendo sulla sfida.

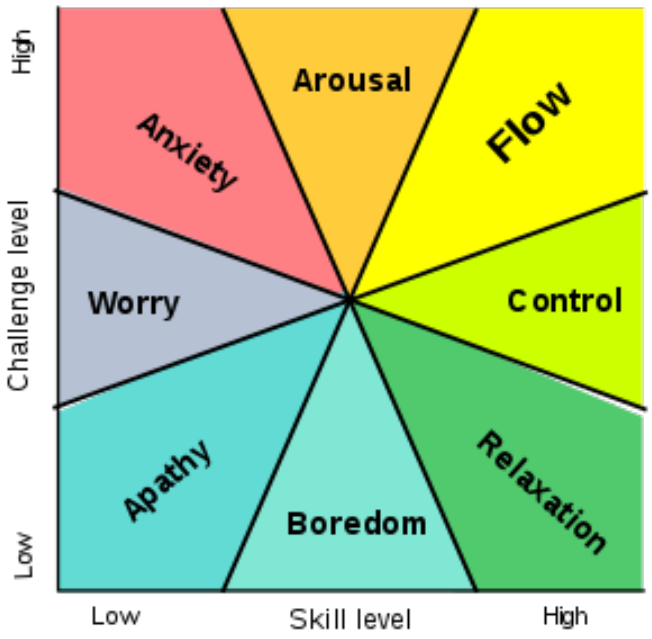

Figura 1.1. Diagramma di Flow

1.2 La didattica ed il Flow

Viene naturale chiedersi come è possibile sfruttare questi stati dell'animo umano nel mondo della didattica. Risulta palese come la condizione ottimale per uno studente sia quella di oscillare tra i tre stati che abbiamo indicato come positivi:l'eccitazione, il flow, il controllo.

Questo è reso possibile da un ascolto continuo dei feedback degli studenti e da un costante adeguamento del livello di sfida imposto durante il corso.

1.2.1 Problemi nella didattica oggi

In un noto studio del 2003 il professore *David J Shernoff* e *Mihaly Csikszentmihalyi* hanno provato a mettere in relazione la teoria del flow con il livello di impegno e concentrazione degli studenti coinvolti in percorsi formativi di alto livello. Lo studio ha analizzato il comportamento e gli stati d'animo di 536 studenti.

Quello che ne risulta è che lo studio individuale e le lezioni frontali non sono abbastanza coinvolgenti. Mentre sussistono alcune condizioni nel rapporto tra la sfida e la capacità, altre vengono a mancare:

- Feedback chiari e frequenti sul proprio operato.
- Task brevi ed auto-consistenti.
- Sensazione di libertà di scelta in merito alle dinamiche del lavoro.

In compenso lo studio rivela anche alcuni dati molto importanti su quali sono invece gli ambienti in cui il flow si manifesta:

- Classi non accademiche: teatro, musica, arte.

- Lavori pratici e collaborativi.

- Apprendimento attivo con un buon grado di autonomia.

Lo scenario ideale quindi risulta essere quello in cui alle classiche lezioni frontali e laboratori viene affiancata una serie di attività stimolanti e pratiche, da svolgere da soli o in autonomia e con un certo livello di difficoltà intrinseca nonché competizione tra gli studenti. Questo motiva gli studenti e stimola l'apprendimento attivo.

1.2.2 Un metodo per una didattica intelligente

Il compito del docente è quello di riuscire a portare più studenti possibili nella zona positiva del diagramma di flusso. Per fare questo deve mantenere costantemente equilibrato il rapporto tra il livello raggiunto dagli studenti e la sfida rappresentata dal corso.

Vediamo adesso quali sono i principi su cui è possibile fare leva per aumentare o diminuire il senso di sfida:

- Difficoltà del compito o degli obbiettivi da raggiungere: il professore può variare la complessità del lavoro da svolgere. Questo può avvenire richiedendo la risoluzione di compiti complessi, aumentando il carico o diminuendo il tempo a disposizione.

- Competizione: mettere gli studenti, o gruppi di studenti, in competizione tra loro aumenta in modo sano il livello di sfida. Bisogna stare comunque attenti a non esagerare perché è molto semplice in questo modo causare ansia e stress ed ottenere risultati opposti a quelli sperati.

- Interesse: questo dovrebbe tendere sempre a salire durante il proseguire del corso.

Questi invece i passi da compiere perché gli studenti vengano indirizzati nell'area giusta e ci rimangano:

- Identificare la posizione dello studente lungo l'asse delle abilità. Questo risultato si può ottenere somministrando dei questionari di auto valutazione o dei test su conoscenze propedeutiche ai argomenti su cui verte il corso.

- Monitorare il livello di motivazione ed interesse: è importante che questi fattori siano sufficientemente alti per portare lo studente fuori dalla zona di apatia.

- Agire costantemente sulla sfida, nello specifico sulla difficoltà dei task, per tenere gli studenti nella zona positiva del diagramma del flow.

Fissare obbiettivi ben definiti e fornire feedback continui e puntuali è molto importante.

1.2.3 Raccogliere e fornire feedback puntuali: lo scopo di questa tesi

Abbiamo fin qui constatato come il dialogo tra docenti e studenti sia alla base di una buona didattica improntata al flow. Il docente deve sempre avere ben chiaro in quale stato d'animo versano gli studenti e quali sono le concause in gioco. Magari non si è riusciti a stimolare l'interesse degli studenti durante le lezioni frontali, o il carico di lavoro e la difficoltà del task superano le conoscenze acquisite.

Questo purtroppo non è sempre possibile nella didattica tradizionale; il dialogo è spesso relegato a brevi momenti tra le lezioni o a poche ore settimanali di ricevimento, di solito dedicate a chiarimenti e correzioni. Non è da sottovalutare inoltre la soggezione a cui sono sottoposti gli studenti, i quali hanno spesso timore di esprimere le proprie opinioni o le proprie difficoltà.

Anche la non omogeneità dei dati raccolti rende difficile definire una strategia e riuscire a correggere il tiro quando occorre; ecco perché non ci si può affidare a feedback generici e disomogenei ma bisogna richiedere che tutti gli studenti forniscano dati aderenti ad uno stesso modello.

Lo scopo di questa tesi è quindi quello di fornire a docenti e studenti uno strumento intuitivo e sempre accessibile per la raccolta di feedback e per favorire il dialogo tra le parti.

Capitolo 2

Analisi del problema

Quelli che vado a presentare in questa sezione sono i requisiti alla base dello sviluppo dell'applicativo. Questi sono stati definiti a partire dal confronto col Prof. Giovanni Malnati.

Nella prima sezione vedremo qual è la modalità di utilizzo che abbiamo pensato per l'applicativo e quali sono i casi in cui questo può portare dei vantaggi nella didattica. Abbiamo infatti visto in precedenza che per poter applicare la teoria del Flow alla didattica è necessario rispettare determinati requisiti. Le attività svolte e le condizioni di studio devono essere tali da realizzare le condizioni necessarie a mettere in moto il meccanismo di oscillazione tra i tre stati considerati "buoni" del diagramma del Flow: eccitazione, flusso, controllo.

Successivamente a questo viene presentata una specifica dei requisiti per l'applicativo; non si tratta stesura conforme allo standard. Il documento di specifica dei requisiti è infatti soggetto allo standard **IEEE Std 830-1993** ma in questo caso ho preferito presentare un modello più snello ed esplicativo per il lettore di questa tesi.

Lo scopo del progetto è quello di realizzare una applicazione di messaggistica totalmente funzionante e che preveda la possibilità di somministrare e compilare questionari. Dovrà inoltre fornire delle funzionalità extra lato Docente per la creazione e programmazione dei questionari e la visione dei risultati.

2.1 Glossario

Di seguito è riportato (tabella 2.1) un glossario con i termini ricorrenti ed eventuali sinonimi.

Tabella 2.1. Glossario.

Termine	**Definizione**
Utente	Generico utilizzatore della piattaforma, può interagire con altri utenti tramite la messaggistica istantanea
Docente	Utente che crea e visiona i questionari ed amministra i gruppi di studenti. **Sinonimi**: professore, admin, amministratore.
Studente	Utente che riceve i questionari a lui indirizzati dal docente
Questionario	Lista di domande a risposta chiusa o aperta creata e programmata tramite l'applicazione
Report	Documento grafico che raccoglie le risposte ad un questionario e disponibile per la consultazione del Docente.
Pubblico	sottoinsieme di Studenti ai quali è indirizzato un Questionario

2.2 Scenario d'uso

Come detto in precedenza vi sono delle condizioni specifiche sotto le quali è possibile sperimentare lo stato di Flow. Per incentivarne la riuscita il docente deve:

- Affiancare alle lezioni frontali delle ore di pratica
- Obiettivi chiari e frequenti feedback sui progressi ottenuti: per ottenere questo è necessario frammentare il lavoro in micro-task a scadenze fisse, ben definite. Per ciascun task inoltre deve esserci una valutazione, sia questa fornita dal docente o dal confronto in aula.
- Stimolo della competizione: altro aspetto fondamentale per instaurare il senso di sfida che è il vero e proprio carburante per raggiungere lo stato di flow. La sfida non è solo competizione diretta, ad esempio tramite classifiche e sfide a punti ma anche interna, tra membri in cooperazione. Per ottenere entrambe l'ideale sarebbe promuovere la realizzazione di un progetto in team.
- Mantenere adeguato il rapporto tra sfida e skill: con il proseguire del corso irrimediabilmente il livello di competenze degli studenti tenderà a salire, sia per via delle lezioni frontali che del lavoro parallelo effettuato per la realizzazione dei task assegnati. Per stimolare la crescita è compito del docente cercare di mantenere gli studenti nella zona di arousal (eccitazione) proponendo sempre un livello di sfida un passo più avanti rispetto alle conoscenze acquisite.

Risulta chiaro fin da subito che le materie scientifiche sono un terreno privilegiato per l'applicazione di questi concetti, essendo già prassi in determinati ambiti STEM affiancare ad una forte conoscenza teorica delle ore di lavoro pratico parzialmente o totalmente autonomo.

Ora è chiaro che quello di cui il docente ha bisogno è guidare gli studenti alla realizzazione di qualcosa, possibilmente cooperando in team cercando di instillare quindi della sana competizione interna, tra compagni di squadra, ed esterna, nei

confronti degli altri gruppi di lavoro. Risulta fondamentale inoltre che agli studenti sia lasciato un certo grado di libertà sul progetto da realizzare; questo stimolerà gli studenti a vedere l'attività come autotelica e volontaria, stimolando il flow. Questo avrà inoltre l'effetto collaterale di sviluppare maggiormente il dibattito tra gli studenti.

Chiaro quindi che quello di cui abbiamo bisogno è una applicazione di messaggistica che permetta agli studenti di comunicare tra loro, sia in gruppi privati che in canali comuni dedicati a tutti gli studenti del corso. Deve essere inoltre possibile comunicare col docente e dargli accesso ai gruppi così che possa monitorare eventuali malumori o difficoltà incontrate. Sarebbe anche raccomandabile che l'ingresso in questo spazio virtuale avvenga quanto prima in modo da permettere agli studenti di familiarizzare col mezzo e di comunicare tra loro fin dalle prime fasi del corso, anche qualora la formazione dei gruppi di lavoro non fosse ancora stata effettuata.

Fissare una sfida adeguata

Ora sappiamo da quanto detto nel capitolo precedente che inevitabilmente gli studenti finiranno per cadere in un punto del grafico del flow, tutto dipenderà dal rapporto che intercorre tra il livello di difficoltà dei task e le motivazioni intrinseche (sfida) sull'asse delle ordinate e dal livello di competenza e padronanza raggiunto, questo valore lo troviamo sull'asse delle ascisse.

Per poter decidere con criterio il livello di difficoltà dei task proposti il docente ha quindi bisogno di conoscere in anticipo una stima del livello medio degli studenti, della loro motivazione e dell' innato entusiasmo per la materia. Questi dati possono essere ottenuti tramite uno o più questionari da sottomettere agli studenti nel momento del loro ingresso nel gruppo o in ogni caso prima dell'inizio delle attività di laboratorio. Un sondaggio prettamente attitudinale ed uno volto a testare le competenze propedeutiche al corso.

A questo punto il software dovrà proporre al docente un risultato, basato unicamente sul test attitudinale, che il docente potrà confermare o ritoccare sulla base del test teorico (qualora questo sia stato effettuato). Questo dovrebbe schiarire le idee al docente su qual è il livello medio dell'aula e su dove fissare l'asticella per fornire agli studenti una didattica efficiente.

Mantenere lo stimolo costante

A questo punto è chiaro che aver fissato un punto di partenza non basta, bisogna monitorare costantemente l'umore ed i sentimenti della classe nonché il loro livello di coinvolgimento e preparazione. Per farlo l'applicativo dovrà somministrare ad intervalli regolari agli studenti dei questionari che vadano costantemente a rivedere l'andamento della platea sul nostro grafico di flow. Questo è un approccio analogo a quello applicato da Mihaly Csikszentmihalyi nei suoi studi sul Flow applicato alla didattica A questi dati il professore potrà sempre affiancare questionari customizzati volti a testare le competenze acquisite, il grado di apprezzamento del corso, la comprensibilità ed attrattiva degli argomenti trattati.

In questo modo non solo il docente potrà continuare ad agire sul livello di sfida ma potrà anche influenzare il valore sulle ordinate, rispiegando un argomento poco chiaro o dedicando più tempo alle esercitazioni. Ma questo miglioramento non

si ferma ad un solo corso, ogni miglioramento alla didattica frontale infatti può portare vantaggi anche negli anni successivi.

2.3 Requisiti

Il lavoro pratico svolto in questa tesi vedrà realizzate solo alcune funzionalità di base dell'applicativo utili a validare la soluzione scelta e a valutarne la fattibilità. Non saranno quindi presenti le funzionalità automatiche di monitoraggio dello stato emotivo degli studenti e quindi il modello matematico tramite il quale posizionare lo studente sul grafo del Flow per valutare eventuali cambiamenti alla didattica.

2.3.1 Requisiti funzionali

- L'applicativo deve permettere agli utenti studenti di comunicare e scambiarsi file multimediali tra di loro e con l'utente docente.
- Gli studenti devono poter essere organizzati in gruppi di dimensioni variabili così da rispecchiare le divisioni in gruppi di laboratorio o progetto.
- Il professore può somministrare agli studenti dei questionari in tempo reale o programmati.
- Il docente, e solo il docente, deve poter visualizzare i dettagli sui questionari creati, inviati o conclusi.

2.3.2 Requisiti non funzionali

- L'applicazione di messaggistica deve essere multi piattaforma e responsiva per incentivarne l'uso tra gli studenti e facilitare la fruizione dei questionari in aula.
- L'applicazione deve fornire un servizio di autenticazione ed autorizzazione delle chiamate per proteggere i dati sensibili degli utenti.
- La creazione e la somministrazione di questionari da parte dei docenti deve essere intuitiva e attuabile in momenti di concitazione come una pausa durante una lezione frontale.
- Eventuali problemi in fase di esecuzione devono essere risolti con un down time minore possibile.

2.4 Casi d'uso

Prendendo in considerazione quanto detto nell'analisi dei requisiti discussa nel paragrafo precedente, è possibile definire la struttura logica dell'applicativo attraverso la definizione dei flussi di operazioni che l'utente deve eseguire per avvalersi delle funzionalità dell'applicazione e delle informazioni necessarie per la loro esecuzione.

Quello illustrato in figura è il diagramma dei casi d'uso il quale permette di descrivere quali sono le funzionalità del sistema, quali gli attori di queste attività e

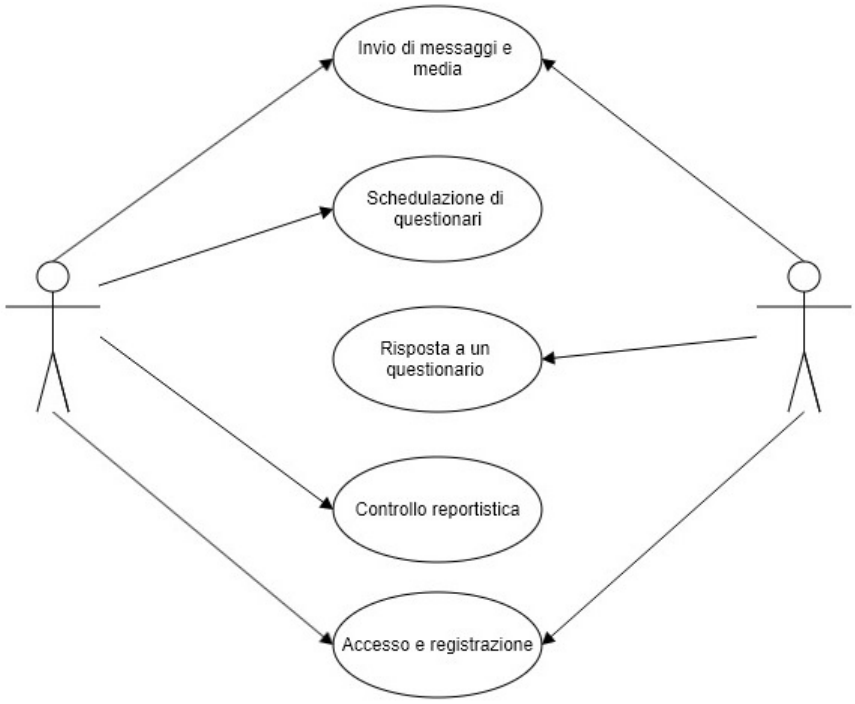

Figura 2.1. Diagramma dei casi d'uso

come eventualmente le attività sono legate logicamente tra loro. Oltre al diagramma generale, per ciascuna funzionalità verrà definito un caso d'uso: verranno illustrate le precondizioni nelle quali sono eseguibili, la sequenza delle azioni svolte dagli attori e dal sistema, nonché le informazioni richieste durante le interazioni.

Questa fase è molto importante perché consente si presenta come transizione tra i requisiti funzionali definiti in linguaggio umano alle specifiche tecniche che portano poi alla effettiva implementazione delle funzionalità del prodotto software.

2.4.1 Registrazione ed accesso

Tabella 2.2. Accesso e registrazione

Nome	Accesso e Registrazione
Attori	Utente
Precondizioni	L'Utente ha installato l'applicazione
Flusso degli eventi	L'Utente si registra al servizio fornendo le informazioni necessarie Il sistema invia all'Utente un'email di conferma L'Utente clicca sul link per l'attivazione dell'account presente nella mail ricevuta Il sistema salva le credenziali dell'Utente

L'utente che intende utilizzare il servizio dovrà eseguire la procedura di registrazione fornendo il proprio nome e cognome, un indirizzo email valido e la password di accesso. La procedura secondo il metodo di *double optin* prevede che il sistema invii un link per l'attivazione dell'account all'indirizzo email specificato. L'utente dovrà confermare che é effettivamente in possesso di tale indirizzo cliccando sul link ricevuto

2.4.2 Invio di messaggi

Tabella 2.3. Invio di messaggi

Nome	Invio di messaggi
Attori	Utente
Precondizioni	L'Utente ha installato l'applicazione L'Utente ha effettuato l'accesso ad un corso
Flusso degli eventi	L'Utente seleziona uno tra i canali a cui ha accesso L'Utente digita il messaggio desiderato L'Utente carica eventuali file multimediali da allegare al messaggio L'utente invia il messaggio sul canale scelto

Una funzionalità imprescindibile per una app di messaggistica. Gli studenti ed il docente devono poter comunicare tra loro sfruttando tutte le funzionalità che ci si aspetta da una moderna applicazione di messaggistica istantanea: invio e ricezione di testo e file multimediali in tempo reale, cronologia delle chat, notifiche, eliminazione e modifica di messaggi inviati.

2.4.3 Programmazione questionario

Tabella 2.4. Programmazione di un questionario

Nome	Programmazione di un questionario
Attori	Docente
Precondizioni	Il Docente ha effettuato l'accesso all'applicazione Il Docente ha creato almeno un corso
Flusso degli eventi	Il Docente sceglie uno tra i suoi corsi Il Docente compone il questionario come desiderato Il Docente seleziona una data di scadenza per il questionario Il Docente invia subito il questionario o lo programma per un istante nel futuro

L'applicazione permette ai docenti di sottomettere questionari ai propri studenti e di raccogliere le risposte in modo semplice ed automatizzato. L'applicazione deve

fornire una interfaccia semplice ma funzionale per comporre dei questionari il più possibile personalizzabili.

2.4.4 Controllo della reportistica

Tabella 2.5. Controllo della reportistica

Nome	Controllo della reportistica
Attori	Docente
Precondizioni	Il Docente ha effettuato l'accesso all'applicazione Il Docente ha creato almeno un questionario
Flusso degli eventi	Il Docente sceglie uno tra i suoi corsi Il Docente sceglie tra la lista dei questionari creati un questionario il cui termine massimo di sottomissione delle risposte sia scaduto

Il docente può accedere e consultare i risultati dei sondaggi sottoposti agli studenti. Quello che interessa non è tanto un elenco puntuale delle risposte studente per studente quanto un report facile e veloce da consultare che esponga il risultato complessivo in una forma utile al docente per ricavare un trend e trarre conclusioni valide a livello di corso.

2.4.5 Risposta ad un questionario

Tabella 2.6. Risposta ad un questionario

Nome	Risposta ad un questionario
Attori	Studente
Precondizioni	Lo studente ha installato l'app ed ha effettuato l'accesso a un corso Il docente ha sottoposto un questionario indirizzato ad un pubblico nel quale sia compreso lo Studente La data di termine per sottoporre le risposte non è passata
Flusso degli eventi	Lo studente riceve notifica di un nuovo questionario Il sistema invia allo studente il questionario Lo studente risponde al questionario e conferma l'inserimento delle sue risposte Il sistema memorizza le risposte e comunica la riuscita dell'invio

Uno studente inserito nel pubblico di un questionario riceverà una notifica per informarlo della disponibilità di un nuovo questionario al quale lui potrà o meno rispondere. Il processo di risposta al questionario è integrato nella stessa applicazione di messaggistica così da fornire tutte le funzionalità in uno stesso ambiente

familiare all'utente. Qualora dovesse scadere il termine massimo per la risposta questa non sarà più inoltrabile.

Capitolo 3

Progettazione

3.1 Stato dell'arte dello sviluppo web

Per applicazione web si intende comunemente qualsiasi applicazione accessibile *on demand* remotamente tramite browser web. L'applicazione è raggiungibile tramite la rete internet e non richiede di installare alcun software sul terminale dell'utente o di usarne la potenza computazionale; questo modello è tipico del world-wide-web ed è comunemente detto Client-Server.

Questa tipologia di applicativi è oramai la più diffusa sul mercato e ha relegato le applicazioni desktop all'utilizzo in settori professionali di nicchia, specialmente quelli che richiedono grande consumo di risorse hardware: manipolazione video, foto-ritocco, calcolo matematico, sviluppo software. L'architettura dominante è quella basata sul modello client-server che per sua stessa natura è tipico di internet fin dagli albori della rete.

Il modello client-server si presenta come la naturale evoluzione del modello P2P (Peer 2 Peer): la presenza di un server, infatti, permette ad un certo numero di client di condividerne le risorse, lasciando che sia il server a gestire ed organizzare gli accessi alle risorse per evitare conflitti o problemi di sovraccarico sulla rete e sulle risorse hardware. In questo modo si può scaricare questi oneri dal client e concentrarsi sul server.

3.1.1 Il pattern MVC

Il modello MVC (Model-View-Controller) è un design pattern estremamente diffuso in ambito web. Per tutti i linguaggi esiste una vasta scelta di framework che implementano MVC come Spring per Java, Django per Python, Rails per Ruby ed Express per Node.js. Il pattern prevede tre *layer*, ciascuno dei quali con compiti ben precisi:

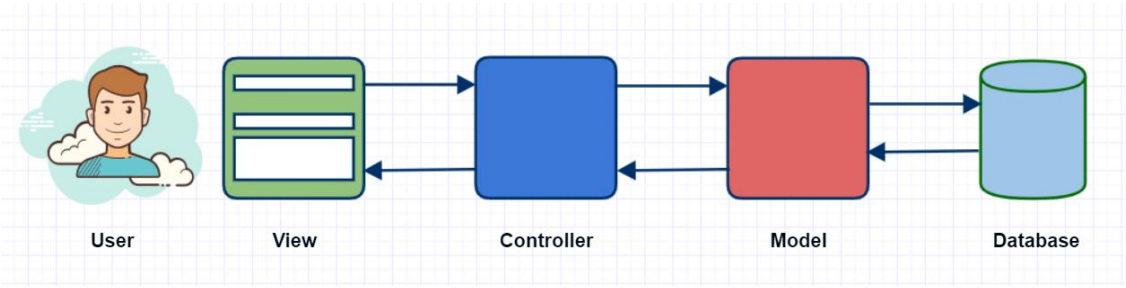

Figura 3.1. Pattern MVC

- **Model**: responsabile dell'accesso e modifica dei dati nonché della modellazione degli stessi. Incapsula lo stato dell'applicazione e notifica alla view eventuali cambiamenti. Si occupa di gestire l'interazione col database dove i dati effettivi sono immagazzinati ed implementa le logiche di creazione, lettura, modifica ed eliminazione.

- **View**: è il livello di presentazione dell'applicazione e si occupa della visualizzazione dei dati per l'utente e della sua interazione col sistema. Questo è l'unico livello su cui l'utente agisce in maniera diretta.

- **Controller**: gestisce i comandi utente ricevuti tramite la view ed implementa la logica applicativa. Ha la responsabilità di trasformare le interazioni dell'utente sulla View in azioni eseguite dal Model. Apporta delle modifiche al modello tramite le API che questo fornisce.

Fino a qualche anno fa il modello MVC era prerogativa del software di back end, l'idea era quella di gestire tutte le logiche sul server per avere dei client più leggeri possibile. Il compito del browser era solo quello di ricevere e visualizzare pagine HTML generate sul back end. Ad oggi questo trend è in declino; le macchine utente non hanno problemi di hardware per cui sono stati creati dei framework front end (come Angular) che implementano il modello MVC. Diventa possibile realizzare applicativi molto complessi poiché molta della logica di business viene spostata sul front end ed anche il carico di lavoro sul server si riduce grazie alle Single Page Application (SPA). Come vedremo a breve le SPA consentono di limitare il carico sul server poiché è il client a generare le viste interagendo col back end solo per piccole richieste AJAX. Questo ci consente inoltre una navigazione più fluida senza ricaricamenti continui.

3.1.2 Single Page Applications

Come detto poc'anzi, tradizionalmente le applicazioni web consistevano di un back end ove era gestita tutta la logica utente e di un front end, costituito prevalentemente da pagine HTML con del CSS e del JavaScript. Il compito di un browser web era quello di fruire di queste pagine web che definiamo "statiche" e di inviare richieste HTTP al server tramite l'interazione con gli elementi della view. Ogni interazione causava quindi un ricaricamento di una nuova vista aggiornata.

Questo approccio offre due vantaggi importanti:

- Tutta la logica risiede lato server poiché il codice sul browser è modificabile e non è sicuro.

- Il carico di elaborazione risiede interamente sul server: necessario quando i terminali utente erano macchine con scarsissime risorse hardware.

Questo ovviamente coincide con un alto carico di rete e computazionale per il server che fatica così a scalare su alti quantitativi di utenti.

Quello che è invece l'approccio moderno allo sviluppo front end è quello di generare le viste sul client. Le comunicazioni col back end servono solo per recuperare o modificare dati, le viste vengono generate di conseguenza dal browser che avrà preventivamente scaricato l'intera applicazione in locale. Questo evita che al server arrivi un enorme carico di richieste e permette di realizzare applicazioni che risultano più veloci e fluide poiché ad ogni interazione viene ricaricato solo il componente della pagina realmente interessato e non l'intera GUI. Da qui il nome di Single Page Application: il codice viene scaricato una sola volta e tutta la navigazione è gestita dal browser agendo sempre sulla stessa pagina web.

Il back end non fornisce più pagine HTML ma dati, solitamente in formato JSON, proprio come farebbe un web server che implementi una API.

3.1.3 Database relazionali e database non relazionali

Le basi di dati (dall'inglese database) sono strumenti imprescindibili nel moderno sviluppo software. Sono necessarie per fornire *memoria* agli applicativi, persistenza del dato e dello stato dell'applicazione stessa in un determinato momento e per un determinato utente. Il modello che si è fatto strada diventando lo standard de facto per la maggior parte degli applicativi è quello relazionale.

In un database relazionale (Relational DBMS) i dati sono organizzati in tabelle le quali contengono dati omogenei tra loro organizzati in tabelle. Queste tabelle sono messe in relazione tra loro tramite riferimenti univoci a dati contenuti in altre tabelle: queste sono dette per l'appunto relazioni.

I dati non sono solitamente ordinati ma sono identificati da codici identificativi univoci che possono essere o meno incrementali; è possibile inoltre definire un qualsivoglia numero di indici che identifichino le tuple permettendo così un accesso più veloce al dato. Questo modello è anche detto ERM (Entity Relationship Model) ed il linguaggio standard per l'interrogazione di questi database è il SQL (Structured Query Language).

Il concetto di transazione e proprietà ACID

Nel mondo delle basi di dati il concetto di transazione è quello di una unità atomica ed indipendente di lavoro; una transazione è una sequenza di operazioni lecite che portano il database da uno stato lecito ad un altro. Le transazioni devono possedere le proprietà ACID. Uno dei motivi dello stradominio dei database relazionali nel mondo pre-cloud è il rispetto *by design* delle regole ACID. ACID è un acronimo per Atomicity, Consistency, Isolation, e Durability (Atomicità, Coerenza, Isolamento e Durabilità) e sono le 4 caratteristiche che le transazioni devono avere per garantire un funzionamento sicuro ed affidabile anche in condizioni d'errore. L'acronimo

nasce nel 1983 per mano dei ricercatori tedeschi Andreas Reuter e Theo Härder ed è valido ancora oggi.

Vediamo nel dettaglio il significato di queste 4 caratteristiche:

- Atomicity: ogni transazione va trattata come una unica operazione. Questa deve sempre essere eseguita interamente in caso di successo o fallire completamente in caso di errore senza lasciare la base di dati in uno stato intermedio.
- Consistency: una transazione deve sempre portare il database da uno stato valido ad un altro stato valido.
- Isolation: in ambienti concorrenti il risultato di operazioni consegutive in modo parallelo deve comunque ritornare lo stesso risultato che avrebbe generato in caso di esecuzione ordinata delle transazioni.
- Durability: una transazione andata a buon fine e correttamente terminata deve essere persistente; non è accettabile perdita di dati da transazioni terminate.

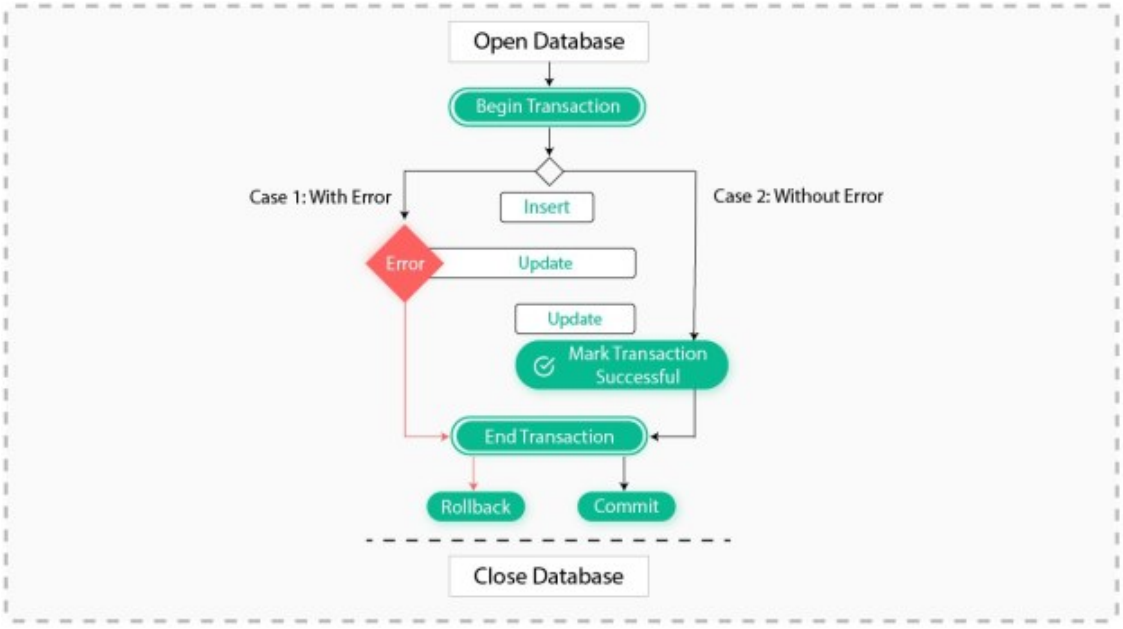

Figura 3.2. Flussi di una transazione.

[1] Tutte queste caratteristiche rendono un database relazionale molto robusto ed affidabile, praticamente *fault tolerant*. Una database con un modello/struttura dati complesso è lento perché deve eseguire tutte le procedure necessarie al fine di rispettare l'ACID per ogni transazione da eseguire. Questo diventa problematico sopratutto in presenza di trigger, ovvero particolari operazioni che il DBMS esegue quando riceve notifica di uno specifico evento sulla base di dati. Spesso i trigger sono utilizzati per propagare delle modifiche sulla base di dati in modo da mantenere i record coerenti tra le diverse tabelle.

[1] Figura 3.2 di proprietà di blogs.innovationm.com

I limiti del modello relazionale

Il problema con i database relazionali è che scalano poco e tendono a diventare lentissimi quando le tabelle diventano enormi e si ramifica eccessivamente la struttura di relazioni, questo perché una singola modifica deve spesso essere propagata su molte tabelle per mantenere coerenti i dati e questo può bloccare il database per molto tempo impedendo magari altri accessi concorrenti. Proprio per questo motivo nascono i database non relazionali come MongoDB che è stato usato come unico database per questo sviluppo (se si esclude il local storage integrato nel browser).

I database non relazionali gestiscono dati meno strutturati per avere maggiore flessibilità e performance, non conoscono spesso il concetto di schema e prevedono una forte denormalizzazione dei dati, in totale controtendenza rispetto ai DBMS Relazionali. Sono inoltre nativamente progettati per il cloud ed i sistemi distribuiti fornendo quindi nativamente soluzioni di scalabilità e ridondanza. MongoDB ad esempio permette di realizzare dei replica-set in modo facile, gestiti *out-of-the-box* senza bisogno di appoggiarsi a software di terze parti.

Tali sistemi in realtà hanno qualche svantaggio rispetto ai sistemi RDBMS, poiché nei sistemi distribuiti non è possibile garantire contemporaneamente le proprietà di consistenza dei dati, tolleranza alla partizione (aree del sistema scollegate tra loro) e disponibilità. Bisogna anche considerare che la consistenza non è automaticamente garantita in ogni momento, è possibile piuttosto negoziare un certo grado di visibilità.

3.1.4 ORM ed ONRM

Il modello ad oggetti è esploso ormai da tempo e domina l'ambiente dello sviluppo web: Java e C# sono nativamente Object-oriented, Python e Ruby supportano il paradigma e JavaScript lo implementa a modo suo essendo in realtà un linguaggio di programmazione orientato ai prototipi ma conoscendo il concetto di oggetto. Una delle difficoltà incontrate nello sviluppo orientato agli oggetti è quella di ottenere la persistenza dei dati senza doverne stravolgere la struttura. Il passaggio infatti da un modello ad oggetti ad uno ad entità-relazioni e viceversa comporta la scrittura di molto codice ripetuto e l'inserimento del fattore umano in uno strato di astrazione che non vorremmo toccare. Quello che vogliamo è poter lavorare con degli oggetti, e poterli persistere in un database relazionale senza curarci di cosa accade nel mentre. Inoltre ci interessa che per le operazioni meno delicate sia il software ad occuparsi di mantenere allineate le due versioni dei dati e mantenere coerenti gli oggetti con lo schema del database. Per questo ci vengono in soccorso le librerie ed i framework di Object/Relational Mapping.

Ovviamente questa necessità è rimasta valida con l'avvento dei database non relazionali. Utilizzare uno strato d'astrazione come questo inoltre consente di cambiare radicalmente struttura o logica di database senza dover riscrivere gran parte del codice. Ne sono un esempio tecnologie prerogativa del mondo Java come JPA, Spring Data ed Hibernate che supportano diversi tipi di database non relazionali senza richiedere particolari modifiche applicative. Si è reso presto necessario coniare un nuovo acronimo per questi moduli quindi oggi si parla di Object/non relational mapping (ONRM). In questa tesi ho fatto uso di Mongoose, un diffuso ONRM per MongoDB e JavaScript.

ORM ed ONRM hanno il vantaggio di velocizzare di molto la scrittura di codice liberando lo sviluppatore da attività ripetitive e tediose, sono inoltre generalmente più sicuri poiché meno codice significa meno bachi. Come già detto inoltre permettono di astrarsi dallo specifico database utilizzato.

Ovviamente queste soluzioni presentano anche degli svantaggi da considerare con cautela. In primis spesso si tratta di software complessi ed il cui funzionamento è difficilmente prevedibile a pieno poiché effettuano tutta una serie di scelte e di ottimizzazioni in automatico; questo porta irrimediabilmente a cedere parte del controllo sul codice. In determinate occasioni ad esempio potrebbe necessario un livello di performance irraggiungibile tramite un ORM.

3.1.5 Architetture RESTFUL

Representational State Transfer (REST) è uno stile architetturale (di architettura software) per i sistemi distribuiti ideato per la prima volta da Roy Fielding nel 2000. REST si presenta come uno stile architetturale per una comunicazione nativamente stateless e che si appoggia sulle funzionalità di HTTP. Questo lo differenzia dalle alternative del tempo come SOAP, tutt'oggi presente sul mercato anche se in misura di molto minore. SOAP infatti è a tutti gli effetti un protocollo di scambio messaggi indipendente dal protocollo di livello applicativo. REST supera SOAP poiché non richiede di scrivere codice aggiuntivo ed è possibile realizzare chiamate HTTP o creare un servizio in pochi minuti e poche righe di codice.

REST è fondato su alcuni principi fondamentali, oltre all'essere stateless organizza le risorse tramite una struttura gerarchica ad albero e le identifica tramite lo Uniform Resource Identifier (in acronimo URI); fa inoltre utilizzo esplicito dei metodi HTTP per definire l'azione da compiere sulla risorsa richiesta. Tra i vari metodi HTTP solitamente ne bastano quattro per realizzare tutte le possibili operazioni su una risorsa: GET, POST, PUT e DELETE.

Tutte queste scelte progettuali hanno contribuito alla progressiva diffusione di REST; questo principio stabilisce un mapping uno a uno tra le tipiche operazioni CRUD (creazione, lettura, aggiornamento, eliminazione di una risorsa) e i metodi HTTP. Il paradigma REST è molto rigido sull'utilizzo corretto dei metodi:

- GET: utilizzato esclusivamente per accedere alla rappresentazione di una risorsa
- POST: utilizzato per sottomettere dati relativi alla URI fornita, solitamente quindi utilizzato per effettuare degli update.
- PUT: utilizzato per l'inserimento su una particolare URI.
- DELETE: utilizzato per l'eliminazione di una risorsa

Queste non sono regole stringenti, HTTP non vieta assolutamente di caricare dati tramite GET o di richiederne in POST ma le linee guida servono per rendere più standardizzate possibili l'API e permetterne un utilizzo più agevole. La differenza tra POST e PUT è labile, spesso gli sviluppatori preferiscono usare il metodo POST sia per inserimenti che per aggiornamenti.

Le risorse sono concettualmente separate dalle rappresentazioni scambiate col client; un servizio web non invia al client oggetti JAVA o record SQL ma una

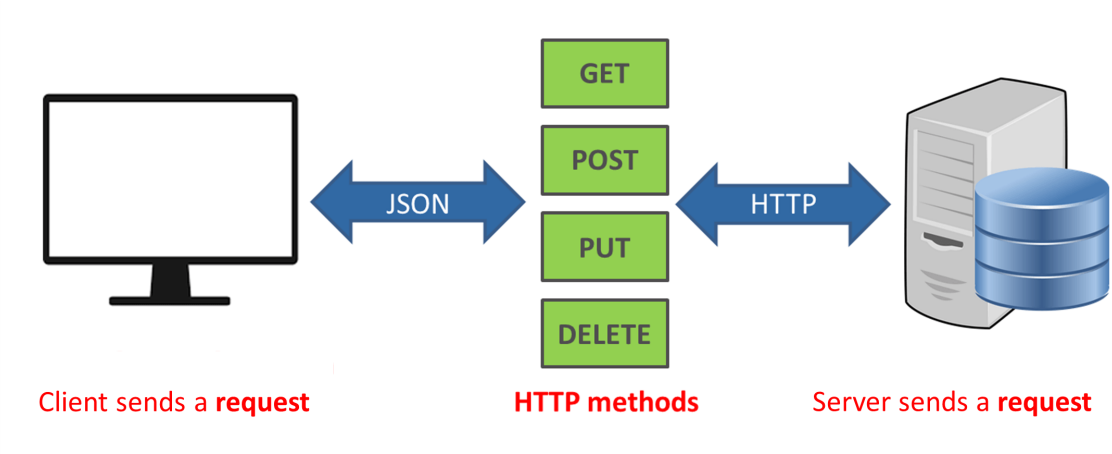

Figura 3.3. Architettura del sistema

codifica dipendente dalla richiesta e dall'implementazione del servizio stesso. Non vi sono vincoli sul formato da usare ma generalmente il formato di trasferimento dati preferito è il JSON. Il tipo di rappresentazione inviata dal Web Service al client è indicato nella stessa risposta HTTP tramite un tipo MIME, così come avviene nella classica comunicazione tra Web server e browser. A sua volta un client può specificare il formato dati richiesto tramite l'attributo *Accept*; se il web service lo supporta la risorsa verrà fornita nel formato richiesto. Questo permette di essere il più possibili indipendenti dalla natura del client.

3.2 Architettura della soluzione

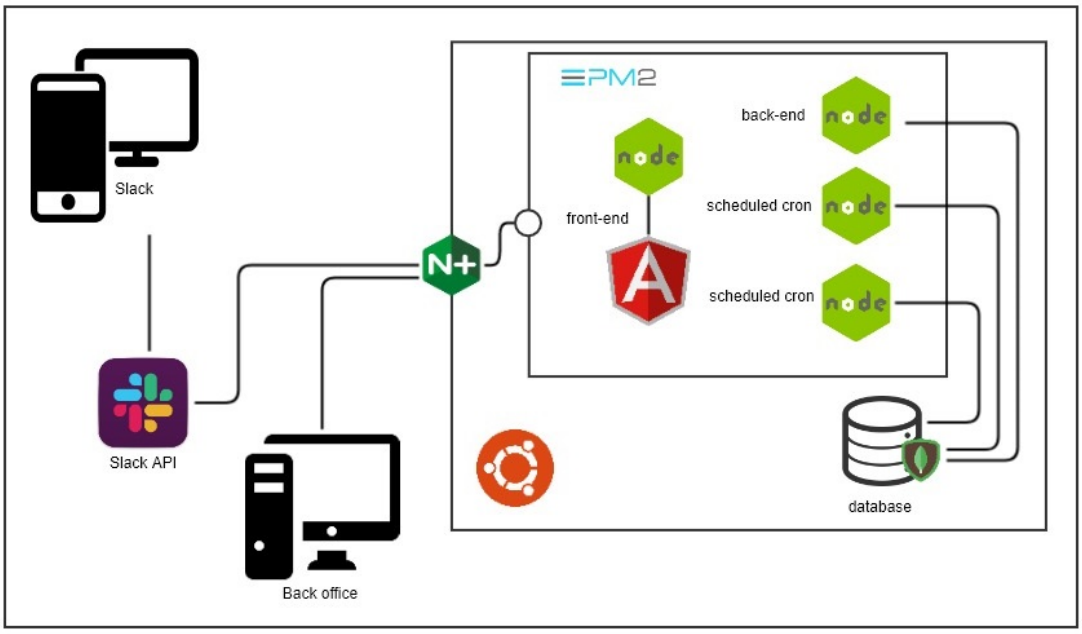

Figura 3.4. Architettura del sistema

La figura 3.4 mostra l'architettura del sistema implementato e i suoi componenti che vanno a comporre la soluzione:

- **Slack**: l'unica interfaccia con cui l'utente studente utilizza l'applicativo. Fornisce tutte le funzionalità di messaggistica istantanea, condivisione di file, creazione e gestione di gruppi. Tramite Slack lo studente risponde ai questionari inviati da Didactic. Anche il docente ha a disposizione tutte le funzionalità offerte da Slack ed un menù integrato in app per interagire con didactic, consultare la lista dei questionari o crearne di nuovi.

- **Slack API**: servizio fornito da Slack per lo sviluppo di applicazioni e bot dedicati alla piattaforma. Slack si comporta da server nei confronti dell'applicazione mobile mentre si comporta da client e server nei confronti del back end a seconda del fatto che sia il back end a richiedere i suoi servizi o sia Slack a contattare il back end in seguito ad un evento. Le comunicazioni da e verso la Slack API avvengono su protocollo HTTP con architettura REST. La Slack API fornisce inoltre l'utente bot didactic, tramite il quale l'applicazione interagisce privatamente con gli utenti umani.

- **VPS (Virtual Private Server)**: istanza Ubuntu Server virtuale fornita da un provider come servizio ad abbonamento. La macchina ospita le componenti dell'applicazione scritte ad hoc che non sono fornite da terze parti: principalmente il front end, il back end ed il database. Il codice viene scaricato da una repository git remota offerta dal provider Github ed il deploy è automatizzato da Jenkins.

- **Reverse Proxy**: indirizza le richieste del client al server responsabile della specifica risorsa richiesta. Si rende necessario dal momento che la stessa macchina espone più servizi sullo stesso dominio. Per questo sviluppo si è utilizzato NGINX.

- **Applicazione di back office**: il back office è una SPA scritta in Angular e fornita su richiesta da un semplice server Node.js. Le funzionalità dell'applicativo sono:

  - Accesso tramite *social identity* Slack.
  - Creazione questionari: è possibile comporre questionari col numero di domande che si preferisce, impostare una scadenza e inviarli puntualmente o programmarli in una data nel futuro. Inoltre è possibile scegliere se creare questionari una tantum o ricorrenti.
  - Visualizzare i report dei questionari chiusi: quando termina la finestra temporale atta a rispondere al questionario viene reso disponibile al docente un report grafico sulle risposte ottenute.
  - gestione dei workflow, flussi personalizzati di questionari basati su eventi specifici come inattività prolungata o primo accesso al workspace.

- **Applicazione back end**: fornisce le logiche al back office, interroga la Slack API per ottenere informazioni ed eseguire azioni sul workspace. Espone inoltre degli endpoint su cui ascolta eventi provenienti da Slack. Si compone di elementi assimilabili a diverse aree funzionali:

  - Autenticazione ed autorizzazione: il web service prevede delle rotte per il processo di autenticazione **Oauth 2.0** ed un middleware per l'autorizzazione delle risorse.
  - Creazione ed invio messaggi: il servizio crea i messaggi Slack da inviare agli utenti basandosi su dei template JSON su cui interpolare i dati reali.
  - Ascolto di eventi Slack: l'applicazione Slack comunica col back end tramite degli endpoint dedicati all'ascolto di eventi. Questi endpoint sono due: uno riguarda gli eventi sul workspace, l'altro le azioni provenienti da messaggi interattivi.
  - back end area privata: il web service fornisce le logiche di back end all'applicativo Angular dedicato ai docenti: interroga la API di Slack per ottenere informazioni sul workspace attivo, accetta in post i nuovi messaggi e serve i dati per la reportistica.

  Il servizio è stato sviluppato in JavaScript su runtime Node.js con Express framework.

- **Cron job**: piccoli eseguibili JavaScript che si occupano di effettuare operazioni ricorrenti. Nello specifico Didactic fa uso di CRON per due operazioni:
  - Invio dei questionari: con una certa periodicità il job controlla che non vi siano questionari da inviare, in caso contrario esegue l'invio e li marca come inviati.
  - Calcolo delle statistiche: periodicamente il cron interroga la base di dati per dei questionari il cui termine massimo per l'invio di risposte è scaduto. Qualora ne trovi calcola le statistiche sulle risposte così da rendere disponibile il report all'utente.

- **Database**: istanza MongoDB che si occupa di fornire persistenza dei dati all'applicazione. Ospita i dati degli utenti registrati, i template dei messaggi Slack ed i questionari.

3.3 Strumenti e tecnologie utilizzati

Lo stato dell'arte per lo sviluppo web si presenta come un ecosistema variegato, pieno di tecnologie e strumenti tutti ugualmente validi ma caratterizzati da peculiarità proprie.

In questo capitolo mi accingo a presentare in maniera più dettagliata gli strumenti scelti per l'implementazione dell'applicativo, ponendo particolare attenzione sulle necessità che mi hanno fatto preferire queste ad altre tecnologie.

3.3.1 JavaScript

JavaScript è un linguaggio di scripting interpretato, ad eventi ed prototype-oriented. JavaScript è ad oggi il linguaggio più utilizzato al mondo ed il più in voga nello sviluppo web, complice il fatto di essere ad oggi praticamente l'unico linguaggio eseguibile sui browser web.

JavaScript nasce nel 1995 per mano di *Brendan Eich*, incaricato da Netscape della realizzazione di un linguaggio che permettesse di interagire con gli elementi statici delle pagine HTML e di rendere dinamici i siti internet. Il nome JavaScript venne utilizzato per sfruttare la popolarità di Java, del quale JavaScript si proponeva come complementare essendo pensato principalmente per l'interazione con le Java Applet.

Il linguaggio ebbe un successo inimmaginabile per Eich e Netscape e si separò ben presto dalla tecnologia Java applet: gli sviluppatori preferivano usarlo per rendere dinamiche le pagine web basate su HTML. Questo successo inatteso portò con se non pochi problemi; JavaScript ha infatti alla base alcune lacune di progettazione dovute alla fretta di uscire sul mercato. Pare infatti che Brendan Eich abbia ideato JavaScript in sole due settimane.

La standardizzazione è arrivata nel 1997 per opera della European Computer Manufacturers Association (ECMA) e porta il nome di ECMAScript, di cui siamo ad oggi alla versione 8.

JavaScript è stato usato per anni come linguaggio di Scripting lato front end ma la vera esplosione di questo linguaggio è avvenuta negli ultimi anni ed è da attribuire a due principali concause:

- Framework front end: La prima versione di Angular, seguita poi da un rapido proliferare di framework per lo sviluppo di app front end standalone ha contribuito all'esplosione di JavaScript per come la vediamo oggi. Strumenti come Angular, Vue.js e React hanno portato una ventata di novità e di nuove potenzialità nello sviluppo web ma anche una necessità di maggiore esperienza e formazione del programmatore. Questo ha dato nuova dignità al ruolo di sviluppatore front end che ad oggi è una delle figure più richieste dal mondo del lavoro.

- Node.js: runtime che permette di sviluppare codice lato back end in JavaScript, dopo una fase di trazione esplosiva si è ridimensionato ma ha trovato una sua nicchia. Ha permesso inoltre lo sviluppo di una serie di prodotti che hanno portato JavaScript in settori nuovi come il game design ed il machine learning.

3.3.2 Node.js

Node.js è una runtime Javascript basato su V8, un motore Javascript open source sviluppato da Google per essere incluso nel browser sviluppato dal colosso americano, Google Chrome. La v8 è attualmente alla base anche delle nuove versioni di Edge, il nuovo browser targato Microsoft. Node.js vede la luce nel 2019 per mano di un team di sviluppatori guidati da Ryan Dahl.

L'obbiettivo di Dahl era semplice quanto ambizioso: migliorare i comuni server web considerati limitati nella capacità di sostenere un gran numero di connessioni concorrenti.

Quello che rende Node così interessante è il suo essere nativamente asincrono ed event driven poiché basato su Javascript, il ché lo rende molto performante nel servire richieste di operazioni caratterizzate da lunghi tempi di I/O. Questo è reso possibile da una API di input/output di basso livello che risponde ai segnali, inviati

dall'hardware alla ricezione e all'invio dei pacchetti di rete, invocando direttamente il corrispondente event handler.

Ciò significa che nonostante il software venga eseguito con un singolo thread è capace di gestire un numero molto elevato di richieste concorrenti. Differentemente, i tradizionali server generano un nuovo thread per ciascuna richiesta pervenuta che rimane attivo fin quando non termina l'invio della risposta, andando a pesare maggiormente sulla RAM disponibile. In un modello event driven invece il thread di esecuzione, in corrispondenza di una operazione di I/O, prende in carico un nuovo task dalla coda. Non appena si rende disponibile il risultato di una operazione asincrona questa viene messa in cima alla coda e verrà eseguita appena possibile.

I colossi dell'informatica si sono accorti presto delle potenzialità della piattaforma ed hanno introdotto con successo codice Javascript back end nei loro ambienti di produzione: due casi meritevoli di menzione sono Paypal e Netflix. Paypal ha da sempre fama di essere una *early adopter* di nuove tecnologie. Il colosso americano ha adottato Node.js nel lontano 2013, inizialmente come strumento per realizzare prototipi molto velocemente. La cosa che più interessava era fornire agli sviluppatori uno strumento universale, una lingua franca che gli permettesse di spostarsi agilmente in modo verticale lungo lo stack tecnologico. Convinti delle possibilità della piattaforma hanno deciso di introdurre Node in produzione e su una delle funzionalità *business critical* della piattaforma di pagamento: la dashboard utente.

Quello che hanno scoperto è stato incredibile: un team di due persone ha completato lo sviluppo nella metà del tempo richiesto al gruppo principale di cinque ingegneri che stava lavorando in Java, in parallelo, sul medesimo sviluppo.

Avevano scritto inoltre il 33% di righe di codice in meno ed avevano ridotto i file sul progetto del 60% per le stesse identiche funzionalità. L'applicazione scritta in Node riusciva inoltre a servire il doppio delle richieste di quella basata su Java ed il framework Spring, con un miglioramento dei tempi di risposta del 35%. L'esperienza di Netflix ricalca a grandi linee quella di Paypal: il colosso dello streaming ha registrato un drastico miglioramento delle performace e della velocità di sviluppo.

Queste sono poi le caratteristiche che mi hanno fatto preferire Node.js e Javascript ad altre alternative per lo sviluppo back end.Node.js con un framework MVC minimale come Express mi ha permesso di realizzare API velocemente perché per niente verboso, con pochissimo *boilerplate* ed un ottimo ecosistema di librerie open surce: NPM.

3.3.3 Express

Express è un framework per applicazioni web Node.js flessibile e leggero che fornisce una serie di funzioni avanzate per le applicazioni web e per dispositivi mobili. Express è considerato un micro-framework ed in quanto tale si discosta molto da altri noti framework MVC per lo sviluppo web come Django per Python o Spring per Java. Questi infatti offrono una suite di strumenti completa e molto varia e per loro stessa natura guidano l'utente nello sviluppare applicativi seguendo una struttura peculiare.

Express invece ha il solo scopo di astrarre alcuni concetti di basso livello di Node e fornisce due componenti essenziali per lo sviluppo di servizi web:

- Router : le rotte definiscono gli endpoint su cui ascolta il servizio ed Express verifica l'aderenza della URI chiamata con le rotte in ordine di definizione; è quindi necessario definire le rotte dalla più specifica alla più generica. Express prevede che per ogni rotta si definisca una callback nella quale il controller si occupa di eseguire logiche o interrogare il modello dei dati.

- Middleware: una funzione che viene eseguita prima che il controller prenda in esame la richiesta. Una funzione di middleware ha accesso alla request ed alla response HTTP e può leggerle e modificarle. Questo è utile per realizzare degli interceptor per l'applicazione di regole di business ortogonali. Dei casi d'uso dei middleware in questo applicativo sono stati la verifica dei token di autenticazione ed il pre-processing della request.

Essendo un micro-framework molto leggero e poco strutturato Express si presta bene ad essere esteso. Diversi noti framework MVC per Node sono infatti costruiti su Express, tra i tanti vale la pena menzionare per la loro diffusione Sails e Nestjs.

3.3.4 JSON

JSON (JavaScript Object Notation) è un formato di testo language-independent per lo scambio di dati. La semplicità di utilizzo ed il suo essere nativamente "human readable" hanno decretato il suo successo, si è infatti imposto come uno standard e sta superando alcuni standard concorrenti come XML.

Attualmente tutti i linguaggi di programmazione supportano serializzare e deserializzare in json. I dati possono essere rappresentati come oggetto (array non ordinato di coppie chiave-valore) oppure array ordinato di valori. Un oggetto in json è rappresentato come un insieme di coppie chiave-valore, separate dalla virgola, all'interno di parentesi graffe. Un array di valori è una sequenza di dati, separati dalla virgola, delimitata da parentesi quadre. Ogni coppia chiave-valore è separata da due punti.

I tipi base supportati sono:

- Stringhe
- Numeri
- Boleani
- Array
- Null

La possibilità di serializzare oggetti complessi in puro testo, rende json un formato particolarmente adatto per il trasferimento di dati perché in grado di preservare la struttura originale delle informazioni inviate. L'altro utilizzo è nella persistenza dei dati in formato facilmente accessibile, leggibile per l'uomo e di facile riconversione. Sono questi i due scenari in cui è stato sfruttato questo formato testuale.

Il JSON si configura un po' come la "lingua franca" nello scambio dati sul web: è il formato scelto da MongoDb per lo storage dei dati ed è la convenzione con cui effettuare richieste tramite HTTP.

3.3.5 MongoDB

MongoDB è un DBMS open source non relazionale basato su documenti, raggruppati semanticamente in collezioni. I database non relazionali vengono anche detti NoSQL. Rispetto alla struttura di un database relazionale basata su tabelle ed uno schema statico, Mongo struttura i record in documenti in formato JSON senza schema.

Mongo predilige un modello di dati fortemente de-normalizzato e ridondante, questo perché un documento è visto come un oggetto auto-consistente e l'architettura stessa del DBMS garantisce ottimi tempi d'accesso su singolo documento ed una gestione nativa thread-safe degli accessi in scrittura e lettura.

MongoDb persiste i documenti JSON in un formato binario chiamato BSON (Binary JSON). BSON estende il formato dei dati JSON per risultare più efficiente in fase di codifica e decodifica e per impiegare meno spazio su disco e meno tempo di I/O in fase di lettura e scrittura. BSON inoltre fornisce tipi di dati non presenti nello standard JSON e ordinamento dei campi, nonché la possibilità di accedere ai record tramite indici, cosa altrimenti non possibile con un documento in puro testo.

Oltre alla sopraccitata comodità nel manipolare e persistere i dati come oggetti Javascript, MongoDB è risultato una scelta vincente per il templating di messaggi Slack. La Slack Messaging API infatti permette di comporre messaggi complessi utilizzando blocchi logici personalizzabili in formato JSON. Ho persistito queste strutture personalizzate su Mongo e le ho utilizzate come template per tutte le tipologie di messaggi inviati dall'applicativo Slack.

3.3.6 TypeScript

TypeScript è un linguaggio open source sviluppato da MicroSoft e si presenta come un super-set di JavaScript fortemente tipizzato. Queste le principali caratteristiche che lo contraddistinguono da JavaScript:

- Tipizzazione forte: una delle principali critiche mosse a JavaScript è infatti quella di essere tipizzato dinamicamente, quindi considerato poco affidabile in ambito enterprise poiché la tipizzazione dinamica può risultare in bug molto ostici da individuare.

- Classi ed ereditarietà: la programmazione orientata ai prototipi è considerata ostica e troppo lontana dai canoni della programmazione ad oggetti. TypeScript maschera l'eredità da prototipo con una gestione ad oggetti gerarchica molto vicina al C# e a Java.

- Interfacce: proprio come per il C# e Java, TypeScript non supporta l'ereditarietà multipla ma fornisce un sistema di interfacce con le quali implementare logiche orizzontali, realizzare il polimorfismo e pattern come la *Dependency Injection* e l'*Inversion Of Control*.

Il linguaggio targato MicroSoft fa tutto questo continuando a compilare in puro JavaScript, mantenendo quindi piena compatibilità con l'ecosistema esistente. TypeScript è il linguaggio scelto da Google per sviluppare Angular 2 ed è il linguaggio scelto per lo sviluppo di applicazioni col noto framework.

3.3.7 Angular

Angular è un framework per lo sviluppo di applicativi front end in HTML, CSS e Typescript. Angular è scritto in Typescript e fornisce una serie di strumenti completa ed una struttura di progetto ben definita da seguire.

Il blocco fondamentale nella costruzione di una applicazione in Angular è l'NgModule che funge da contesto di compilazione per i vari componenti che definiscono ciascuno una view o parte di una view. Un NgModule serve da raccoglitore logico che accorpa del codice logicamente e funzionalmente connesso; una applicazione Angular è definita da uno o più NgModule. E' sempre presente un modulo root che fornisce informazioni di build e definisce parametri e dipendenze trasversali.

Elementi principali

Angular definisce degli elementi funzionali con responsabilità specifiche, tra questi i principali sono:

- **Component**: definisce aspetto e comportamento di uno o più elementi grafici presenti a schermo. Un componente ha un ciclo di vita gestito da Angular. Angular lo crea, lo renderizza, crea e renderizza i suoi componenti figli.

  Angular fornisce inoltre allo sviluppatore una serie di metodi tramite l'implementazione di interfacce chiamati lifecycle hooks. Questi hook forniscono allo sviluppatore degli "agganci" per eseguire specifiche logiche in corrispondenza di questi momenti nel ciclo di vita del coomponente. Gli Hook previsti da Angular sono:

  - constructor()

    È il metodo base del componente Angular e non fa parte del ciclo di vita di un componente ma lo trattiamo come un hook. Infatti il constructor è la prima parte di codice che viene eseguita da un componente quando viene istanziato.

  - onChanges()

    Viene eseguito come primo evento del componente Angular e anche ogni qualvolta che cambiano i valori delle propietà di input. Il metodo implementato usando l'argomento SimpleChanges può essere usato per leggere il vecchio e il nuovo valore di una proprietà.

  - onInit()

    Come presuppone il nome, questo evento dichiara che Angular è pronto alla creazione del componente. Questa fase si verifica subito dopo il primo OnChanges e viene eseguita una sola volta. Il che significa che tutte le dipendenza iniettate verranno risolte e tutti i membri della classe verranno definiti. questo lo rende il posto perfetto per eseguire qualsiasi operazione/logica di inizializzazione del componente.

  - doCheck()

    La fase di check viene eseguita durante il check interno di Angular per valutare le modifiche ai componenti e ai dati. Questo evento a differenza dell'OnChanges che si attiva a ogni cambiamento di una proprietà di

input, viene lanciato a qualsiasi cambiamento accada al componente e quindi viene lanciato molto spesso da Angualr (ci basta pensare che ogni volta che il mouse si muove si genera un evento docheck).

- afterContentInit()

  Questo evento si riferisce al completamento degli eventi legati al componente, in specifico alla fine della creazione dell'albero dei componenti figli.

- afterContentChecked()

  Viene invocato ogni volta che terminano i doCheck().

- afterViewInit()

  Dopo il render della view associata al componente e delle sue viste secondarie, viene innestato questo evento e quindi il DOM è visibile.

- afterViewvChecked()

  Invocato ogni volta che termina il rilevamento di afterContentChecked().

- onDestroy()

  Questo hook è legato all'ultima fase del ciclo di vita del componente, che si verifica appena prima che Angular lo distrugga. Anche questa come OnInit viene eseguita una sola volta.

Alcuni hook sono di uso comune, altri si prestano solo a casi d'uso specifici; in questo progetto è stato fatto uso, oltre che ovviamente dei costruttori, dell' onInit() e dell' afterViewInit().

- **Service** : implementa funzionalità ortogonali o comunque non direttamente legate ad uno specifico componente. L'idea è di separare le funzionalità delle view dal resto della logica, questa è un'altra delle scelte degli sviluppatori di Angular che spinge verso uno sviluppo modulare e disaccoppiato.

  Un componente dovrebbe solo gestire le logiche di user experience e fare da collegamento tra la view e la logica applicativa. Tutti gli altri task vengono delegati ai servizi: ricevere e inviare dati al back end, validare gli input dell'utente o gestire i log applicativi.

  I servizi vengono instanziati dal modulo e gestiti come dei singleton; vengono resi disponibili a tutti i componenti tramite dependency injection.

- **interceptor**: presente dalla versione 4.3 del framework, un Interceptor consente di preprocessare le response e le request delle chiamate HTTP. In modo analogo a quello dei middleware di Express è possibile definire una catena di interceptor. Per creare un interceptor in Angular è necessario definire un servizio che implementa l'interfaccia HttpInterceptor, importandola da @angular/common/http. La classe implementerà il metodo intercept, che verrà invocato per intercettare una richiesta o una risposta HTTP. L'ordine in cui vengono chiamati gli interceptor corrisponde all'ordine in cui viene effettuato il providing dei relativi servizi nell'NgModule.

Principi DRY

L'estrema modularità di Angular, il suo essere naturalmente disaccoppiato e l'incentivo non troppo velato al riuso di componenti permette senza troppo sforzo l'applicazione di un principio caro all'ingegneria del software: DRY (Don't Repeat Yourself). Il principio DRY suggerisce di evitare ogni forma di ripetizione ed è stato teorizzato per la prima volta nel Libro culto "The Pragmatic Programmer" da Andy Hunt e Dave Thomas. L'ambito più immediato ed intuitivo è quello della duplicazione del codice ma è un concetto che si allarga ad ogni componente dei sistemi software.

La presenza di codice duplicato può facilmente portare ad errori e malfunzionamenti ogniqualvolta si renda necessaria una modifica. Questo perché bisognerà portare lo sforzo di andare a modificare il codice in più punti senza dimenticanze o errori.

Per questo durante lo sviluppo Angular è conveniente pensare a dei componenti il più possibile generici e riutilizzabili, questo può portare via del tempo in fase di prima scrittura ma abbatte drasticamente i tempi successivi di sviluppo, modifica e debug.

Dependency Injection in Angular

Come detto precedentemente Angular supporta in modo nativo i paradigmi di *Dependency Injection* e *Inversion Of Control*. Questo permette di non codificare le dipendenze direttamente nel codice ma di lasciare che se ne occupi un componente detto *injector*. Da qui il termine *Inversione del controllo*, la responsabilità di instanziare le dipendenze viene spostata dallo sviluppatore ad un componente software terzo.

Qusto permette di rendere il codice molto più modulare e riusabile nonché permette la scrittura di unit test che risulterebbero altrimenti impossibili. In Angular le dipendenze devono essere dichiarate come parametri nel costruttore della classe, Angular provvede al recupero e ne gestisce autonomamente il ciclo di vita. Una entità che si voglia fornire per l'iniezione deve essere etichettata col decoratore `Injectable()` e registrata tra i provider nel modulo nel quale si vuole importare.

3.3.8 Slack

In fase di analisi sono risultate due le strade percorribili per lo sviluppo dell'applicativo

- Realizzare da zero una applicazione di messaggistica : soluzione che avrebbe permesso di non scendere a compromessi ed avrebbe offerto la possibilità di personalizzare al massimo l'applicativo. D'altro canto questa alternativa avrebbe richiesto tempi di sviluppo molto lunghi per realizzare un applicativo cross-platform, responsivo e con tutte le funzionalità che ci si aspetterebbe da una app di messaggistica moderna.

- Estendere un applicativo esistente: è questa la soluzione che ho adottato e la scelta è ricaduta su Slack.

Slack è la più nota applicazione di collaborazione aziendale cloud based e permette di scambiarsi messaggi e condividere file tra membri di uno stesso team. Originariamente sviluppato nel 2009 da Stewart Butterfield per la sua azienda, *Tiny Speck*. L'intento originale era di sviluppare uno strumento di chat integrato per Glitch, un videogioco all'epoca in sviluppo presso l'azienda. La mission di Slack è quella di fornire un singolo strumento per la comunicazione, la gestione e la produttività. Per fare questo Slack si integra con tutti i maggiori strumenti più in uso nelle aziende ed è disponibile per tutti i sistemi e dispositivi fissi e mobili.

Tra le funzionalità principali dell'applicazione abbiamo messaggistica istantanea, condivisione di documenti e chat di gruppo. Tra le estensioni più scaricate vi sono i principali fornitori di servizi di versioning control quali GitHub e GitLab; spazi di archiviazione cloud come Dropbox e Google Drive e piattaforme di gestione Agile dei team e Kanban Board come Jira e Trello.

Non ultima l'applicazione è sempre più adottata negli ambienti lavorativi di tutto il mondo, conoscerla può risultare in un ulteriore valore aggiunto per gli studenti.

Slack mette a disposizione del tutto gratuitamente, per chiunque voglia realizzare plugin e bot per la piattaforma, degli strumenti ed una API documentata per reagire ad eventi, inviare messaggi e raccogliere dati dai workspace.

3.3.9 PM2

PM2 è il più noto manager di processi per Node.js, permette di eseguire gli applicativi come servizi e farli ripartire automaticamente qualora dovesse interrompersi l'esecuzione. Questo è molto importante nella gestione di applicativi Node.js poiché una eccezione non gestita interromperebbe l'esecuzione del software e richiederebbe altrimenti un riavvio manuale.

PM2 inoltre permette di eseguire una applicazione in cluster e di gestire il carico delle diverse istanze tramite un load balancer integrato. Chiudono il quadro una lunga serie di funzionalità che facilitano l'amministrazione, il debug e la configurazione degli ambienti.

Quando si avvia un'applicazione utilizzando il comando `pm2`, l'applicazione viene immediatamente inviata all'applicazione di background. È possibile controllare l'applicazione di background dalla riga comandi utilizzando diversi comandi PM2.

Dopo che un'applicazione viene avviata tramite il comando `pm2`, viene registrata nell'elenco di processi PM2 con un ID. È possibile quindi gestire le applicazioni con lo stesso nome da directory differenti sul sistema, utilizzando i relativi ID.

3.3.10 Nginx

Nginx è un HTTP web server che può essere usato anche come reverse proxy, mail server, load balancer e server TCP/UDP generico ed è arrivato nel Maggio del 2019 a essere sfruttato dal 42% dei siti web stando ai dati forniti da W3techs, compagnia che si occupa di monitorare l'ultilizzo delle tecnologie per il web.

Tra le sue caratteristiche spiccano una ottimizzata gestione delle richieste verso file statici e un supporto per applicazioni FastCGI, Fast Common Gateway Interface. Un'altra caratteristica che mi ha fatto optare per questa soluzione è l'estrema semplicità che contraddistingue il suo processo di configurazione: si tratta di agire

su un file testuale diviso in moduli, in ciascuno dei quali possono essere specificate delle direttive espresse sotto forma di coppie chiave/valore per abilitare o disabilitare determinati comportamenti.

Nginx è stato utilizzato esclusivamente come reverse proxy. Un reverse proxy si occupa di indirizzare le richieste provenienti da un client su diverse URI verso i server dedicati a fornire la risorsa richiesta. Nel mio caso Nginx si occupa di indirizzare le richieste al server che fornisce i file statici generati da Angular ed al server che fornisce l'API di back end.

3.3.11 Jenkins

Jenkins è un server di automazione open source e rappresenta uno dei principali strumenti per l'implementazione di automatismi. Esso permette di definire dei project, una serie di direttive e script tramite i quali automatizzare compiti complessi, quali ad esempio la compilazione e l'esecuzione di un programma o l'esecuzione di suite di unit test.

Jenkins è un prodotto disponibile per tutti i principali sistemi operativi, è autoconsistente ed è configurabile tramite la GUI integrata; presenta inoltre un ricco ecosistema di plugin messi a disposizione dalla community.

Ho usato Jenkins per implementare un basilare sistema di integrazione continua basato sui webhook forniti da GitHub. Tramite un webhook è possibile chiedere a GitHub di notificarci, ad un indirizzo da noi specificato, eventi che avvengono su specifici branch.

Nel mio caso Jenkins espone un endpoint sul quale viene contattato ogniqualvolta GitHub rileva una modifica alla codebase presente sul branch Master. Quando contattato sull'endpoint richiesto Jenkins scarica il codice aggiornato dal repository ed esegue le routine di rilascio dell'applicativo.

3.4 Slack: un approfondimento

Slack è la soluzione scelta per l'implementazione di questo prototipo. Non si può dire che sia stata in senso assoluto la migliore soluzione possibile o che ne consiglierei l'utilizzo anche per un eventuale prodotto finale. Slack è stata una delle strade vagliate e quella che è poi stata effettivamente percorsa in questo lavoro di tesi con i suoi pregi ed i suoi difetti.

3.4.1 Perché Slack

Le motivazioni dietro la scelta di usare Slack sono diverse:

- **Velocità di sviluppo**: lo sviluppo di un prototipo o di un MVP serve quale studio di fattibilità poiché permette allo sviluppatore di scontrarsi con i problemi tecnici tipici dello sviluppo applicativo e agli stakeholder di provare il software, testarlo e validare l'idea alla base dello sviluppo. C'è una sottile seppur sostanziale differenza tra prototipo ed MVP; un prototipo è solitamente un software che andrà gettato e riscritto da zero dopo la fase di validazione. Un minimum viable product invece è scritto per essere poi ampliato fino a diventare il prodotto finale. L'approccio che ho provato a seguire è quello

dello sviluppo di MVP, scrivendo codice da subito funzionante, modulare ed ottimizzato.

Ad ogni modo, quale sia il nostro caso, è fondamentale ridurre la quantità di codice da scrivere per abbattere i tempi di sviluppo impegnati in attività che non hanno a che fare con il core dell'idea da validare. Utilizzando framework, librerie e software di terze parti non limitiamo solo il tempo di scrittura; meno codice significa anche meno bachi quindi meno tempo speso in attività di debugging.

- **Software collaudato**: Slack è leader nel settore della messaggistica istantanea per team, nel primo quarto del 2019 l'app ha potuto contare su 9 milioni di utenti unici a settimana di cui 3 milioni su account premium. Questa fortuna è dovuta anche al ricco sistema di integrazioni che Slack offre con tutti i principali servizi di gestione team, organizzazione, sviluppo software e monitoraggio. Oltre il 90% degli utenti utilizza almeno una Slack App di terze parti tra le oltre 1500 disponibili nel repository ufficiale. Vi sono applicazioni per la gestione di appuntamenti, l'organizzazione aziendale, il teambuilding, la condivisione di file in cloud, perfino per organizzare la pausa pranzo!

  Questo fa sì che Slack sia un software sicuro, performante, seguito da una ampia community che collabora segnalando bug e realizzando plugin e bot. Appoggiarsi ad un software terzo di successo significa ereditare tutta l'expertise del team Slack e garantirsi tutte le migliorie introdotte nei futuri aggiornamenti.

3.4.2 Una alternativa open source a Slack

Volendo mantenere l'approccio integrato, che prevede una app di messaggistica completa che integri funzionalità personalizzate per la creazione e somministrazione di questionari, la comunità open source offre una alternativa libera a Slack. Il suo nome è **Mattermost** e si presenta come un clone perfetto di Slack in quanto a funzionalità e design; è scritto in Golang e React ed ha alle spalle circa 400 sviluppatori. Mattermost offre davvero tutte le funzionalità di Slack dai comandi ai messaggi interattivi alla gestione degli eventi. Per stessa scelta degli sviluppatori questo applicativo offre una API molto simile a quella di Slack, rendendo possibile un passaggio quasi indolore di una app o di un bot dall'uno all'altro. Mattermost non è un servizio ma un software, necessita quindi di un server sul quale essere eseguito, contrariamente a Slack che ci permette di utilizzare le sue risorse per veicolare le funzionalità di messaggistica. Slack è stato preferito per la facilità di setup, l'opportunità di sfruttare un servizio di terze parti hostato indipendentemente e la maggiore diffusione in ambito accademico e professionale.

3.4.3 Applicazioni e bot per Slack con la Slack API

Le integrazioni, nome che Slack utilizza per fare riferimento alle Slack app ed ai bot, sono fondamentali per gli utenti di Slack e per spiegare il successo planetario dell'applicazione dell'azienda di San Francisco.

Le applicazioni Slack permettono di integrare tutti i nostri strumenti preferiti con Slack e ci permettono di utilizzarli senza abbandonare l'applicazione. Si possono

caricare file su uno spazio di archiviazione in cloud come Dropbox o Google Drive, si può tenere d'occhio il flusso di CI/CD del software ricevendo aggiornamenti da Jenkins o GitHub direttamente in chat e così via.

Si evince come il posto naturale di Slack siano le aziende tech ma esistono applicativi dedicati al business ed alla didattica come fogli ore, calendari condivisi, sondaggi e note. L'ideale sarebbe di integrare tutto il proprio workflow, si parli di una azienda, un corso universitario o un singolo individuo, su Slack. Per fare questo il team di Slack ha messo a disposizione una API aperta, la Slack API.

Le Slack app possono aggregare e mostrare dati provenienti da tool e servizi esterni, per fare questo ogni Slack app ha accesso a degli strumenti che permettono di leggere, scrivere e modificare dati su un workspace. Un workspace non è che un ambiente virtuale composto da un insieme di canali dove i membri di un team possono comunicare tra loro. Un workspace ha inoltre uno o più amministratori che hanno la possibilità di creare e gestire canali (o conversazioni) e di invitare o revocare membri.

Queste sono solo alcune delle funzionalità permesse dalla API:

- Invio di messaggi su qualunque tipo di conversazione tramite webhook e web API. Questa funzionalità è alla base dei bot ma è fondamentale anche per notificare eventi o fornire pseudo interfacce grafiche tramite i messaggi interattivi.

- lettura di messaggi ed altri dati tramite web API ed events API.

- Creazione, modifica, archiviazione ed eliminazione di conversazioni. Utile per la creazione di conversazioni temporanee, della durata di una lezione o di un laboratorio.

- Ascolto di eventi interni al workspace per creare punti di accesso a diversi flussi decisionali. Possiamo citare con gli eventi anche l'invocazione di comandi Slack, messaggi caratterizzati dall'avere come primo carattere un backslash. Questi non vengono stampati in chat ma inviati a Slack come comandi da gestire.

- Invio di dialog, altrimenti impossibile all'utente comune. Permette di raccogliere informazioni strutturate tramite la realizzazione di semplici form compresi di controlli di validazione sui campi.

Componenti principali della Slack API

Nella sezione precedente abbiamo accennato ad alcuni componenti della Slack API: events API, web API, comandi. Nel capitolo sull'implementazione vedremo i dettagli tecnici riguardanti l'utilizzo di questi strumenti. In questa sezione vedremo invece una veloce carrellata di strumenti e della loro utilità nello sviluppo di integrazioni per Slack.

Alcuni degli strumenti messi a disposizione dalla API sono:

- Web API

- Events API

- Conversations API
- Slack Commands
- Endpoint OAUTH 2.0

La console dello sviluppatore

La console dello sviluppatore è un portale web a disposizione di chiunque voglia creare e gestire una integrazione per Slack. Permette di configurare la comunicazione tra Slack ed il nostro software registrando gli endpoint realativi ai comandi, alla event API, all'accesso tramite OAUTH 2.0 e alle azioni effettuate sui messaggi interattivi. Permette inoltre di configurare caratteristiche come l'icona dell'app e dei bot, i permessi richiesti in fase di installazione ed altri.

L'accesso alla console per una specifica Slack App non è limitato al solo creatore, è possibile infatti definire uno o più collaboratori che avranno quindi accesso alle configurazioni dell'integrazione sulle loro console.

Pubblicazione di una integrazione Slack

Durante lo sviluppo l'applicazione sarà installata solo sul workspace che le abbiamo collegato in fase di creazione dalla web console. La possibilità di distribuire l'applicazione ad altri team è attivabile dalla console spuntando la voce relativa. Per permettere questo viene fornita la funzionalità per la condivisione dell'integrazione tramite link e tramite il bottone "Add to Slack".

Diverso invece l'iter per rendere disponibile l'applicazione nella directory ufficiale Slack perché sia quindi indicizzata e più facilmente reperibile per gli utenti interessati. Per poter pubblicare un'app sulla directory ufficiale è infatti necessario rispettare una serie di requisiti:

- Dati necessari al censimento dell'app: categorie di indicizzazione, una descrizione breve ed una completa.
- Link utili quali l'indirizzo della landing page, l'indirizzo alla pagina delle policy della privacy ed alla pagina di supporto.
- Una mail di supporto clienti
- Una icona e delle immagini esplicative sullo scopo dell'applicazione
- Bisogna inoltre accertarsi di non violare alcuna norma di copyright, di non diffondere contenuti inappropriati e di rispettare alcune semplici linee guida di usabilità.

Una volta rispettati questi punti la richiesta viene valutata dal team di Slack ed eventualmente accettata o respinta.

Capitolo 4

Implementazione

In questo capitolo vado ad esporre i dettagli dell'implementazione della soluzione progettata nelle sezioni precedenti. Non verrà esaminato lo sviluppo nella sua interezza ma verrà fornita una panoramica di insieme sulla struttura del codice sorgente e su alcune delle scelte implementative prese. In seguito maggiore spazio sarà concesso ad alcuni casi rilevanti. Ad esempio a quelle che sono state le soluzioni più creative o l'implementazione di standard consolidati in ambito web, come nel caso della autenticazione Slack tramite Oauth 2.0 e l'autenticazione di API tramite Json Web Token.

4.1 Back End

4.1.1 Struttura di un progetto Node Express

Express è un framework minimale per lo sviluppo di applicazioni web in Node.js ed è basato su due componenti fondamentali, come già visto in precedenza in questa tesi: rotte e middleware.

Le rotte definiscono le URI sulle quali il servizio fornisce risorse; queste possono essere registrate direttamente sull'istanza della applicazione Express ma possono essere raggruppate logicamente con l'ausilio dei Router. I Router sono oggetti Express che possono registrare delle rotte e sui quali è possibile definire dei middleware. Tramite il metodo use() una applicazione Express può registrare il router ereditandone le rotte. prevede che per ogni rotta si definisca una callback nella quale il controller sioccupa di eseguire logiche o interrogare il modello dei dati.

Analogamente è possibile usare il metodo use() per registrare un middleware. Un middleware è una funzione che viene eseguita prima che la richiesta venga instradata alla rotta corretta e viene usato per effettuare operazioni sulla request. Si può utilizzare per instradare la richiesta ad una cache, per filtrare determinate richieste o per autenticare l'API. Nel caso di Didactic ho usato i middleware per autenticare le rotte tramite Json Web Token o per gestire logiche orizzontali, comuni quindi a più chiamate.

Più middleware vengono eseguiti in cascata in ordine di dichiarazione. Per continuare la catena è necessario che la funzione invochi il metodo next(). Oltre che definirne di propri è possibile utilizzare middleware di terze parti scaricati tramite NPM. Un esempio lampante è quello del modulo body-parser che fornisce

un middleware che parsa il body delle chiamate in ingresso in un leggibile formato JSON.

Un server Node.js basato su Express ha un punto d'ingresso unico, è qui che si instanzia l'applicazione Express e si registrano router, middleware o rotte. Questi è il file JavaScript contenente la chiamata al metodo listen(). Tramite il metodo listen il server si mette in ascolto su una porta scelta ed accetta richieste web HTTP. Qualora riceva una richiesta l'applicativo effettuerà un matching tra le URI registrate ed il path della chiamata, così facendo la chiamata viene servita alla prima rotta utile in ordine di dichiarazione.

Di seguito uno spaccato del file init.js, entry-point del back end di Didactic.

```
1  //sub-routers as middlewares
2  app.use('/didactic/api/', events);
3  app.use('/didactic/api/', interactiveComponents);
4  app.use('/didactic/api/', accessRoutes);
5  app.use('/didactic/api/', slackOauthRoutes);
6  app.use('/didactic/api/', resourcesRoutes);
7  app.use('/didactic/api/', surveyRoutes);
8
9    mongoose.connect('mongodb://usr:psw@localhost:port/dbname', function
         (err) {
10     console.log("Mongo connection established");
11     app.listen(process.env.PORT, () => {
12       console.log(`App listening on port ${process.env.PORT}!`);
13     });
14   }).catch((error) => {
15     console.log(error);
16     console.log("Was not possible to establish a connection with Mongo");
17   });
```

Figura 4.1. Porzione di codice dal file entry-point del Back end

Il metodo use registra eventuali router e middleware sulla applicazione express corrente, instanziata col metodo express(). Un router è un componente Express sul quale sono state registrate una o più rotte: aiuta a scrivere codice più modulare e disaccoppiato.

La connessione al database verrà affrontata nella sezione successiva.

4.1.2 Connessione a MongoDB con Mongoose

Mongoose è un Object Document Mapper (ODM), ovvero un framework che consente di mappare le entità applicative su documenti MongoDB fornendo una serie di funzionalità che rendono più semplice l'accesso e la manipolazione dei dati sotto forma di oggetti.

Mongoose interagisce con MongoDB per noi e aggiunge delle funzionalità quali la tipizzazione forte delle proprietà ed uno schema immutabile, entrambe caratteristiche non native per Mongo.

Per poter operare su MongoDB per noi Mongoose ha bisogno di stabilire una connessione ad database; è sufficiente effettuarlo una volta solo per questo viene fatto nel file init.js che è il punto d'ingresso della nostra applicazione e dove ci occupiamo di importare dipendenze, definire le rotte Express e avviare il server sulla porta desiderata. Per fare questo il modulo Mongoose mette a disposizione la funzione connect la quale prende come parametro una stringa di connessione ed una funzione di callback.

```
mongoose.connect('mongodb://usr:psw@localhost:port/dbname', function
    (err) {
  console.log("Mongo connection established");
  app.listen(process.env.PORT, () => {
    console.log(`App listening on port ${process.env.PORT}!`);
  });
}).catch((error) => {
  console.log(error);
  console.log("Was not possible to establish a connection with Mongo");
});
```

Essendo la connessione alla base di dati una dipendenza necessaria all'operatività del servizio ci si mette in ascolto sulla porta solo ad operazione terminata e solo se la connessione è stata instaurata con successo.

4.1.3 Modelli Mongoose

Come accennato nella sezione precedente Mongoose offre alcune funzionalità aggiuntive non presenti in MongoDB tra cui uno schema e la tipizzazione forte. Permette inoltre di definire valori di default, l'obbligatorietà dei campi o di modificare il dato in fase di acquisizione ad esempio tramite l'uso di espressioni regolari.

Questo è reso possibile anche dal fatto che Mongoose permette di intervenire sul dato in diversi momenti, ad esempio prima che questo venga ritornato sotto forma di oggetto o durante la fase di persistenza sul database.

Mongoose è basato su due concetto fondamentali ovvero lo schema ed il model. Uno schema mappa una collezione mongoDB e definisce le regole che andranno a dare forma ai documenti presenti nella collezione. Per poter utilizzare la definizione di uno schema bisogna passarlo al costruttore che si occupa di generare il model.

Possiamo vedere il model come il componente che fornisce le funzionalità object oriented legate ad uno schema specifico. Seguendo i familiari concetti della programmazione ad oggetti un model sta ai documenti come una classe sta agli oggetti istanziati. Quello riportato di seguito è il file che contiene la definizione dello schema relativo agli utenti ed esporta il modello realizzato a partire da suddetto schema.

```
var mongoose = require("mongoose");

/*
 *   Model for the teacher user
 */

var workspace_schema = new mongoose.Schema({
    name: 'string',
    team_id: 'string',
    admin: 'string',
    owner_id: 'string'
}, {
    strict: false
});

var teacher_schema = new mongoose.Schema({
    email: {
        type: String,
        index: {
            unique: true,
            dropDups: true
        }
    },
    real_name: 'string',
    picture: "string",
    workspaces: [workspace_schema]
}, {
    strict: false
});

var Teacher = mongoose.model('Teacher', teacher_schema);
module.exports = {
    Teacher: Teacher,
}
```

Figura 4.2. Porzione di codice dal file teacher.js

Lo schema è definito passando il parametro opzionale "strict: false", questo permette a Mongoose di persistere a database anche proprietà che non sono presenti nello schema. Questo mi consente maggiore flessibilità qualora in casi molto specifici e saltuari io abbia bisogno di persistere dati non presenti in schema che altrimenti verrebbero ignorati.

Si può notare inoltre come sia possibile definire più schema innestati per definire strutture ad oggetti complesse.

4.1.4 WebHook Slack

Tutta l'interazione tra un workspace Slack ed una Slack App è realizzata tramite la registrazione a particolari eventi generati all'interno dell'aplicazione di messaggistica. Tramite il portale dedicato agli sviluppatori è possibile fornire a Slack degli end-point da chiamare per comunicare l'occorrenza di specifici eventi e scatenare una qualche forma di interazione con l'applicativo. Alcuni di questi eventi possono essere l'ingresso di un nuovo membro, l'esecuzione di un comando o l'interazione con messaggi interattivi.

Vediamo nel dettaglio come sono state implementate queste funzionalità:

Comandi Slack

Un comando Slack è una qualunque stringa preceduta da un carattere di slash (/). L'esecuzione di un comando deve effettuare una qualche operazione e fornire all'utente un feedback della corretta esecuzione della stessa. Per rendere disponibile un comando personalizzato per la propria applicazione è necessario registrarlo presso il portale dedicato e predisporre l'applicativo per l'ascolto sul path registrato. A questo punto, ogniqualvolta un utente eseguirà il comando un payload in JSON verrà inviato tramite chiamate POST HTTP al servizio sulla URI scelta. L'esempio di seguito riportato riguarda il comando `/survey`, utilizzato dal docente per richiamare un menù con le informazioni in merito agli ultimi questionary rilasciati.

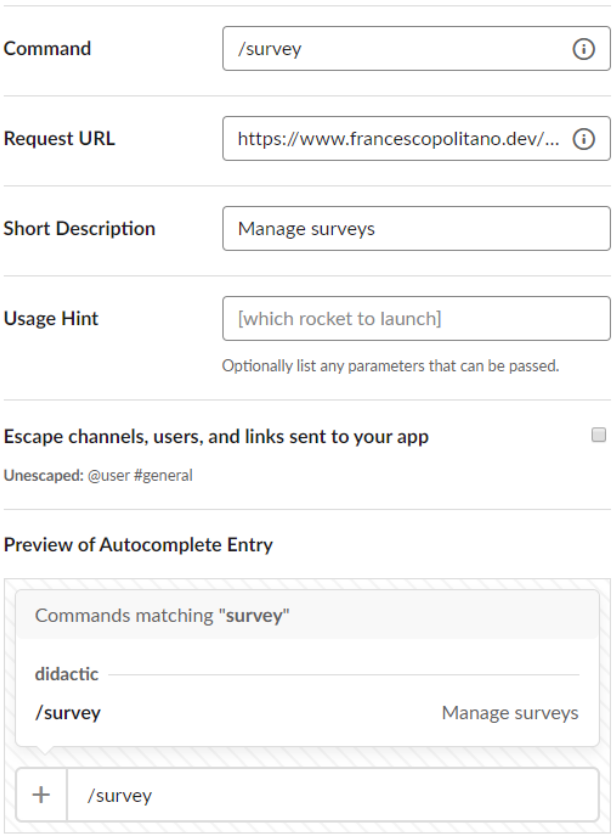

Figura 4.3. Registrazione di un comando Slack

Come si evince dall'immagine 4.3 oltre che il nome del comando ed il path ad esso registrato è possibile definire una descrizione da visualizzare in app, dei suggerimenti per eventuali parametri in input ed un filtro d'accesso.

```
router.post('/commands/survey', function (req, res) {
  if (req.body && req.body.ssl_check) {
    //Connection test by Slack API
    return res.send();
  }
  menuBuilder.buildMenu(req.body.team_id, req.body.user_id, function (err, message) {
    if (err) {
      return res.send(err);
    } else {
      Database.Teacher.aggregate([{
        $match: {
          "workspaces.team_id": req.body.team_id
        }
      }, {
        $unwind: {
          path: "$workspaces"
        }
      }, {
        $match: {
          "workspaces.team_id": req.body.team_id
        }
      }], function (err, users) {
        if (err) {
          console.log("something went wrong in eventsRouter");
          next(err);
        }
        if (users && users.length != 0) {
          var actualUser = users[0];
          if (actualUser) {
            if (actualUser.workspaces.owner_id != req.body.user_id) {
              //The caller is not the workspace owner
              return res.send("This feature is reserved for admin.");
            }
            access_token = actualUser.workspaces.access_token
            directChannel.openIm(access_token, req.body.user_id, true,
              function (err, direct_id) {
                directChannel.sendOnIm(access_token, direct_id, message, function () {
                  res.send();
                });
              });
          };
        }
      })
    }
  })
});
```

Figura 4.4. Gestione back end di un comando Slack

La chiamata contiene tutte le informazioni utili ad identificare il contesto: il workspace, l'ID del chiamante, il canale dal quale è stato lanciato. Questo comando nello specifico è destinato all'amministratore, lato applicativo viene fatto un controllo sull'ID del chiamante che viene confrontato con quello dell'admin registrato su database.

Eventi

Similmente alla gestione dei comandi vi è la gestione degli eventi. Contrariamente però ai comandi non si registra un end-point per ciascun evento bensì un unico path; sarà po il payload della richiesta a permetterci di discriminare quale degli eventi registrati è stato invocato.

Per utilizzare la Events API di Slack è necessario recarsi sul portale dello sviluppatore e cercare la tab `Event Subscriptions`. Qui bisogna attivare la sottoscrizione agli eventi, questo è un passaggio assolutamente necessario.

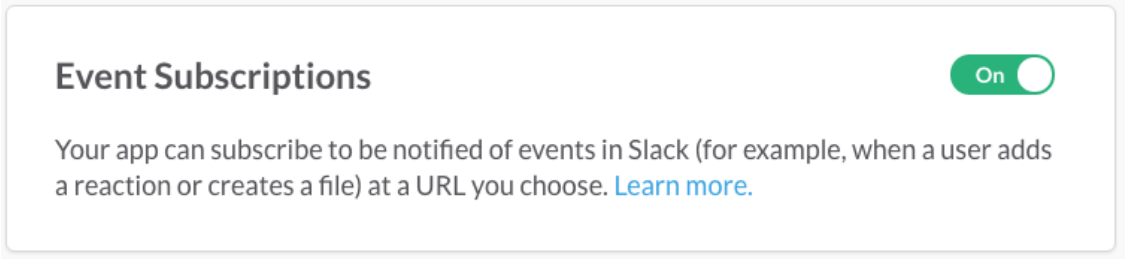

Figura 4.5. Attivare la sottoscrizione agli Eventi.

A questo punto dalla stessa pagina è possibile selezionare tutti gli eventi ai quale si vuole sottoscriversi e registrale la URL sulla quale si vuole riceverli. La registrazione della URL è sottoposta ad una challange: in questo modo Slack si assicura che la URL sia corretta, sia sotto il nostro controllo e protetta da HTTPS.

Essendo l'end-point unico per l'applicativo tutte le operazioni di dispatching vengono lasciate allo sviluppatore. Di seguito un listato che mostra la funzione realizzata per gestire gli eventi in Didactic:

```
/*
 * Endpoint to receive events from Slack's Events API.
 * Handles:
 *    - url_verification: Returns challenge token sent when present.
 *    - event_callback: Confirm verification token & handle 'team_join' event.
 */
router.post('/events', (req, res, next) => {
  console.log(req.body.type);
  switch (req.body.type) {
    case 'url_verification':
        // verify Events API endpoint by returning challenge if present
        res.send({ challenge: req.body.challenge });
        break;
    case 'event_callback':
          const event = req.body.event;
          if (event.type === 'team_join' && !event.is_bot) {
                MessageBroker.newMember(req, res);
          } else {
            res.sendStatus(500);
            console.log('Unknown callback event type received');
          }
          break;
    default:
        res.sendStatus(500);
        console.log('Unknown event received');
  }
});
```

Figura 4.6. Endpoint per la gestione degli eventi

Come possiamo vedere la sfida lanciata da Slack in fase di registrazione dell'end-point richiede soltanto di reinviare al mittente un token alfanumerico. La gestione delle diverse logiche per le diverse tipologie di eventi è gestita tramite uno Switch sul campo type del corpo della richiesta.

Componenti interattivi

In Slack con "componente interattivo" intendiamo tutti gli elementi grafici che consentono di effettuare delle operazioni all'interno di messaggi, detti a loro volta interattivi. Interagire con un componente interattivo stabilisce una connessione tra l'utente e la Slack App.

Per poter gestire questo tipo di interazione Slack richiede delle configurazioni sulla nostra applicazione personalizzata le quali possono essere effettuate, come per i casi precedenti, presso il pannello dedicato allo sviluppatore sul sito delle Slack API. Nell'immagine 4.7 è raffigurato ciò che appare selezionando la voce "Componenti interattivi" nella dashboard della nostra applicazione Slack.

Figura 4.7. Attivazione dell'interattività.

In questa stessa pagina è possibile definire azioni personalizzate, richiamabili dal menù a tendina delle opzioni del messaggio; non verranno approfondite perché non ne è stata definita alcuna in questo lavoro di tesi.

Analogamente a quanto succede nell'event loop anche questo end-point riceve indiscriminatamente tutti gli eventi di interazione scatenati all'interno dell'applicazione, è lasciato quindi allo sviluppatore l'onere di discernere quale tipo di azione è stato invocato e che tipo di reazione fornire all'utente. Per fare questo la chiamata fornisce diverse informazioni:

- Informazioni in merito all'utente chiamante ed al canale.
- Informazioni in merito al componente sorgente dell'interazioni. Il parametro `type` permette di identificare il tipo di componente; il `trigger_id` è un identificativo univoco dell'azione, a corta scadenza, che permette di aprire un dialog o di eliminare il messaggio scatenante. Queste informazioni includono altri valori parametrizzabili utili per distinguere il risultato di azioni specifiche. Un buon esempio è il caso del parametro `action` caratterizzato da un nome ed un valore definiti in fase di invio del messaggio interattivo o di definizione dell'azione personalizzata. Ho fatto largo uso di questa proprietà per definire il workflow dell'interazione.

4.1.5 Oauth 2.0: social login e installazione

Essendo l'applicativo di back office una funzionalità complementare alla Slack App ed imprescindibile da essa, era naturale fornire agli utenti una soluzione di registrazione e login strettamente accoppiata a Slack che non creasse discrepanze tra le informazioni relative all'account Slack e all'aplicazione Didactic.

La Slack login offre diversi vantaggi, primo tra tutti snellisce il processo di registrazione permettendo all'utente di saltare tediosi passaggi in cui inserisce i propri dati anagrafici e la verifica della mail tramite Double opt-in.

Per fare questo Slack mette a disposizione un servizio di autorizzazione basato sullo standard Oauth 2.0.

Oauth è un protocollo di autorizzazione open source che delega l'onere dell'autenticazione ad una terza parte fidata. Sarà Slack infatti a gestire l'infrastruttura di SSO e l'autenticazione a due fattori.

Il protocollo OAuth è stato ideato da Blaine Cook nel 2006, mentre lavorava all'implementazione Twitter di OpenID, proponendosi come alternativa aperta ai molti protocolli proprietari già esistenti. Dalla versione 1.0 (pubblicata nel 2007) OAuth, ha subìto diverse revisioni, recependo via via le RFC (Request For Comments) proposte dai vari esperti. In generale le RFC sono un insieme di documenti di riferimento per la Comunità Internet che riportano informazioni o specifiche riguardanti nuove ricerche, innovazioni e metodologie dell'ambito informatico. I vari principali erogatori di servizi, come ad esempio i social network, Facebook, Twitter, LinkedIn e tanti altri, forniscono un supporto stabile ad oauth 2.0.

Il protocollo prevede interazione tra un service provider, che fornisce le informazioni sullo user e garantisce per l'autenticità delle stesse, ed il consumer cioè l'applicativo che effettua la richiesta per queste informazioni.

Per poter usufruire dei servizi di un provider è necessario registrare l'applicazione; in seguito alla registrazione vengono fornite dal provider una chiave ed un segreto nonché un endpoind sul quale redirigire l'utente perché possa procedere con l'autenticazione sul servizio esterno e possa autorizzare l'invio dei dati sensibili all'applicativo consumer.

Applicazione del protocollo per Slack

Vediamo adesso quali sono i passaggi previsti da Slack per implementare la funzionalità di "login with Slack". Come ormai procedura standard per una web app con autenticazione, al primo accesso al sito di un utente non loggato questo viene indirizzato alla pagina di login.

Vediamo ora nel dettaglio i passaggi che realizzano il processo di login:

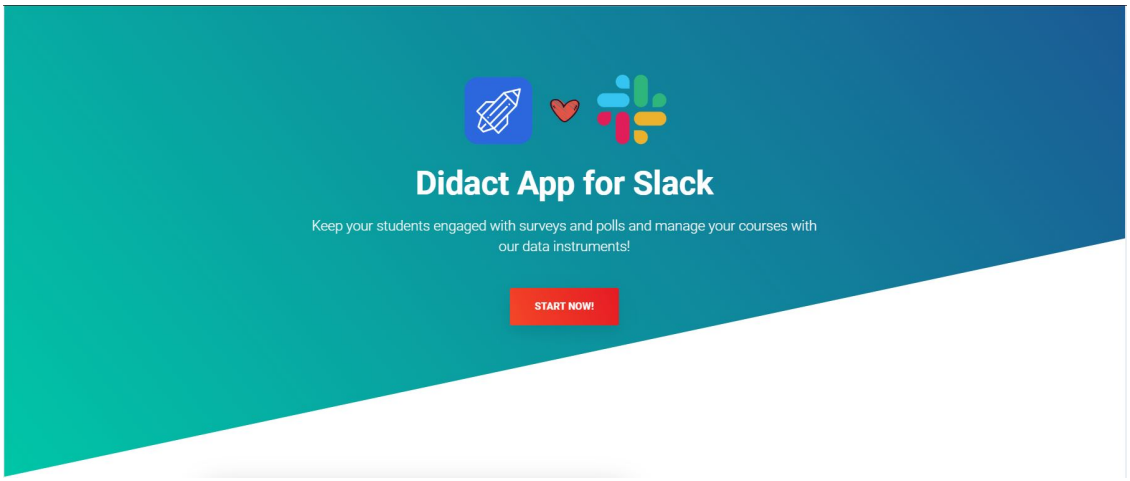

Figura 4.8. Pagina di login dell'applicazione Didactic

- L'applicazione redirige l'utente ad un endpoint di autorizzazione fornito da Slack: https://slack.com/oauth/authorize La richiesta deve passare alcuni parametri obbligatori nella get:

  – client_id: fornito da Slack in fase di creazione di una Slack App

  – scope: una lista di parametri separati da spazi che indica a quali informazioni dell'utente vogliamo accedere, quali la sua mail, il suo username ed il suo avatar.

  – redirect_uri: La URL sulla quale reindirizzare alla fine del processo.

- La chiamata viene reindirizzata sul nostro server con un codice di verifica temporaneo della durata di dieci minuti.

  A questo punto è possibile scambiare il codice di verifica con un access token effettuando una richiesta http get a questo endpoint https://slack.com/api/oauth.access con i seguenti paramentri

  – client_id: fornito da Slack in fase di creazione di una Slack App.

  – client_secret: fornito da Slack in fase di creazione di una Slack App.

  – code: Il codice temporaneo ottenuto dalla chiamata precedente.

  – redirect_uri: Deve essere uguale a quello fornito originariamente.

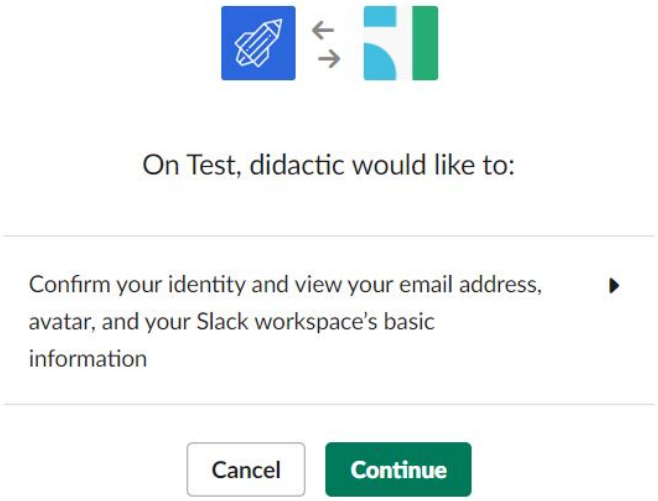

Figura 4.9. Pagina di social login sui sistemi Slack

Se il processo è stato eseguito correttamente Slack risponde con un access token che potrà essere usato per chiamare tutti i metodi API di Slack per conto dell'utente loggato.

- In ultima istanza usiamo l'access token appena ricevuto per interrogare le Slack API ed ottenere i dati utente che ci servono per creare l'utenza e persisterla in database.

Al termine del processo di login il back-end reindirizza la chiamata al front-end con una modalità che vedremo in seguito nella sezione dedicata al token JWT.

Analoga è l'implementazione della funzionalità "add to Slack" che permette di installare l'applicazione Slack su un workspace direttamente dal back office. Mentre per la login si richiedono una serie di permessi per leggere le informazioni utente, è nel processo di Oauth dell'installazione che la nostra app richiede tutti i permessi dei quali necessita per funzionare all'utente che ha il dovere di leggere ed accettare le richieste in blocco per usufruire del servizio.

I permessi definiscono i metodi API che la nostra applicazione è autorizzata a chiamare: questo incide su quali informazioni ed operazioni sono disponibili sul workspace sul quale è installata. Per la distribuzione Slack richiede che ogni permesso richiesto sia accompagnato da una breve motivazione e ha veto sull'accettazione degli stessi.

Una volta che una applicazione Slack viene rilasciata sul Market ufficiale questa potrà richiedere solo i permessi approvati dal team di Slack.

Le risorse per le quali è possibile richiedere un permesso sono divise in diverse categorie, queste le principali:

- Conversations: fornisce tutti i metodi di accesso ai canali pubblici e privati del workspace, permette così di leggere, inviare o modificare messaggi.

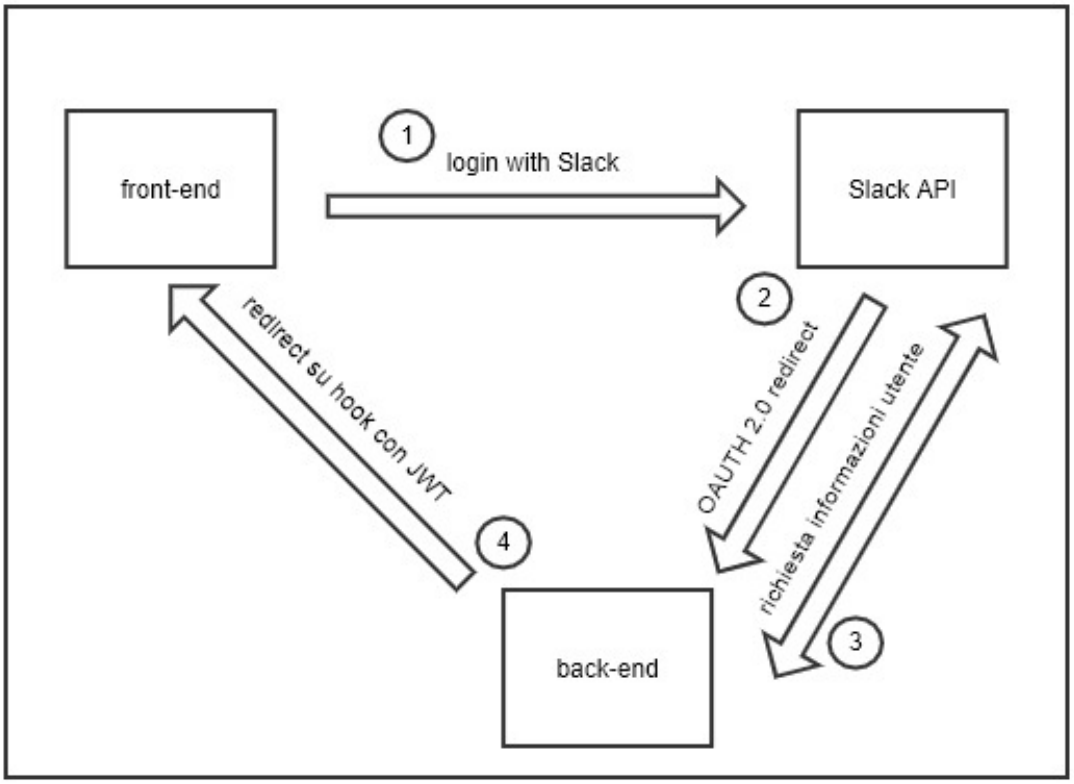

Figura 4.10. Flusso di autenticazione

- Users: metodi che permettono di leggere o modificare le informazioni dei membri e di predisporre dei promemoria per l'utente.
- Interactivity: permette di aggiungere dei bot al workspace, di implementare comandi personalizzati e di inviare messaggi interattivi creati tramite gli strumenti offerti dalla Slack API.

4.1.6 Autenticazione di API Rest con JWT

Una delle esigenze più comuni nello sviluppo di API Rest è quella di limitare l'accesso ad una o più risorse agli utenti autenticati o ad una cerchia ristretta di aventi diritto. Per fare questo ogni richiesta deve contenere, solitamente all'interno dello header, le informazioni per identificare l'utente ed i suoi privilegi.

Tra le soluzioni di autenticazione stateless ricordiamo la Basic authentication, basata su header di autenticazione: ad ogni chiamata effettuata dal client viene passato un token contenente le credenziali di accesso dello user. Questa soluzione prevede che il server autentichi l'utente ad ogni chiamata, aggiungendo un notevole overhead. Inoltre con questa soluzione non è possibile comunicare al server dei dati strutturati secondo le necessità applicative.

Jwt è uno standard utilizzato per regolare le richieste di un client ad un server. Si tratta in sostanza di informazioni in formato Json serializzate e firmate tramite lo standard JSON Web Signature (JWS) per la sola integrità dei dati, o tramite

JSON Web Encryption (JWE) per integrità e riservatezza. Entrambi gli standard prevedono l'ultilizzo di un segreto non condiviso di proprietà del server.

Struttura di un JWT

Vediamo più nel dettaglio le tre componenti peculiari di un JWT:

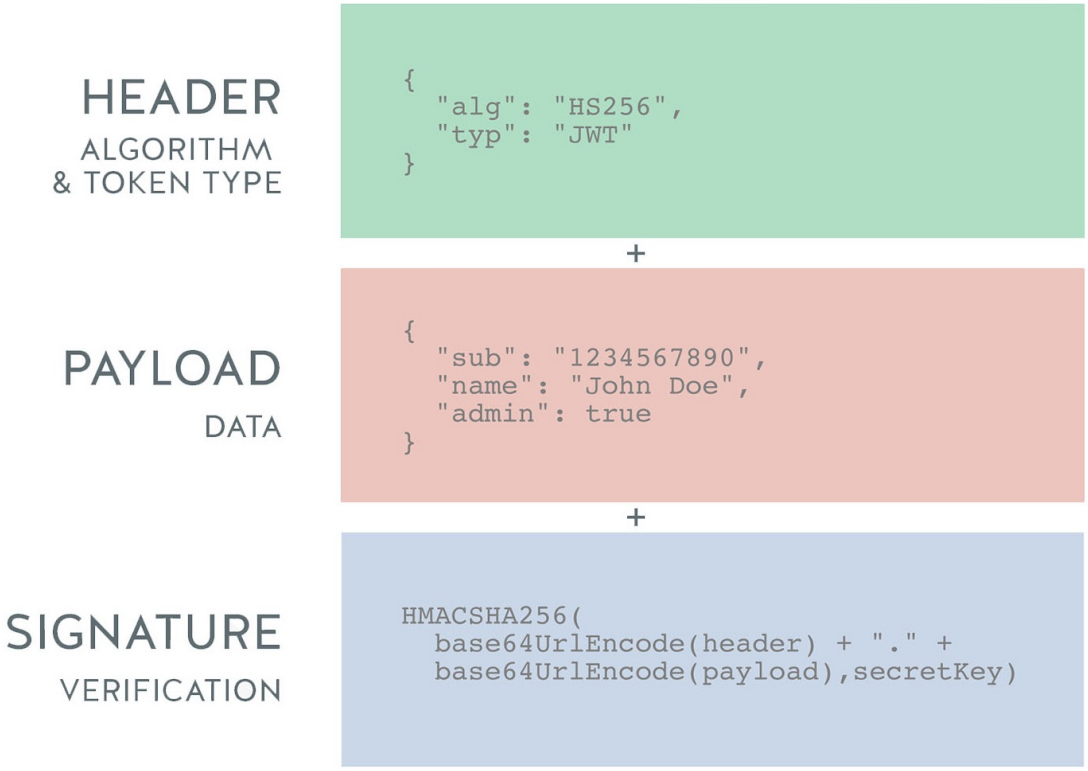

Figura 4.11. Struttura di un Json Web Token

- Header: La prima parte del token è un oggetto in json che contiene le due key "typ" e "alg". Il primo ha come valore sempre "JWT", mentre il nodo "alg" contiene il nome dell'algoritmo usato per il token.

- Payload: La seconda parte del token è il corpo vero e proprio che contiene dei dati custom in formato json e codificati in base64. Lo standard definisce una serie di nomi "registrati" per le chiavi del payload, nessuno di questi è da considerarsi obbligatorio. Tra questi ricordiamo exp (Expiration Time) per la data di scadenza del token o nbf (Not Before) per la data prima del quale il token è da considerarsi non valido.

- Signature: La terza parte di un token è la firma che non è altro che il risultato dell'applicazione di una funzione di hash che prende in input la codifica base64 dell'header concatenandola con un punto alla codifica base64 del payload, il tutto codificato con la nostra "chiave segreta" di proprietà del server.

Flusso di autenticazione JWT

Ora procederò a descrivere come ho implementato il meccanismo di autenticazione del back-end RESTful tramite JWT.

A valle del flusso di login Oauth 2.0 il server di back-end incapsula in uno JWT le informazioni necessarie ad identificare l'utente all'interno dell'applicazione. A questo punto il back-end reindirizza l'utente alla sua area privata, più precisamente ad una rotta Angular corrispondente ad un componente che ho definito login-hook.

Il login-hook accetta come parametro queryString il JSON Web Token risultante dall'autenticazione registrandosi ad un evento in fase di inizializzazione del componente:

```
  constructor(private route: ActivatedRoute, private admin: AdminService)
     { }

  ngOnInit() {
    this.sub = this.route.queryParams.subscribe(params => {
      this.user = decodeURIComponent(params["user"]);
      this.admin.oauth_login(this.user);
    });
  }

}
```

```
  oauth_login(userData) {
    let user = <User>{};
    let userDataObj = JSON.parse(userData);
    localStorage.setItem("token", userDataObj['token']);
    let userUrl = environment.backendUrl + '/user';
    let headers = new HttpHeaders().set('x-access-token',
        userDataObj['token']);
    let params = new HttpParams();
    this.http.get(userUrl, {
      params: params,
      headers: headers,
      withCredentials: false
    }).subscribe((userInfo) => {
      user.name = userInfo['name'];
      user.picture_url = userInfo['picture'];
      user.workspaces = userInfo["workspaces"];
      localStorage.setItem("user", JSON.stringify(user));
      this.userSignedIn.emit(user);
      this.router.navigate(["/workflows"]);
    });
  }
```

Figura 4.12. Funzione oauth_login di AdminService

La funzione oauth_login di AdminService salva il token in local storage ed effettua una chiamata al back-end per ottenere i dati utente necessari a popolare la vista di area privata: username, foto ed i workspace collegati a Didactic.

L'utente viene poi reindirizzato alla rotta principale che corrisponde alla sua area privata.

4.1.7 CRON job

Nell'implementazione della soluzione in esame sono state due le funzionalità che hanno richiesto lo sviluppo di attività schedulate: l'invio di sondaggi e la creazione delle statistiche. Entrambe le operazioni dipendono da un parametro temporale, nel primo caso il timestamp scelto per l'invio, nel secondo caso quello per la scadenza del questionario.

Queste due funzionalità sono realizzate tramite degli script JavaScript programmati per essere eseguiti con cadenze specifiche e molto ravvicinate. Entrambi i job scrollano i record in database ed eseguono le operazioni stabilite sui questionari eligibili. Il listato sottostante mostra la logica di schedule del CRON di generazione delle statistiche; per programmare lo script secondo la sintassi crontab ho usato il modulo node-schedule, ad ogni avvio la logica scorre i record in database per cercare questionari scaduti e non ancora processati. Si omettono l'effettiva logica di calcolo delle statistiche.

```
1  const schedule = require("node-schedule");
2
3  var calculateStatistics = function () {
4      survey.find({
5          isSubmitted: true,
6          expiration: {
7              $lt: new Date()
8          },
9          statistics: []
10
11     }).then(results => {
12         results.forEach(element => {
13         /*****     Logica di generazione statistiche *****/
14          })
15     })
16 }
17 schedule.scheduleJob("*/2 * * * *", calculateStatistics);
```

Sintassi CRONTAB

Crontab è un'applicativo per sistemi UNIX che consente la pianificazione di attività come l'esecuzione di un comando o uno script. Col tempo la sintassi utilizzata per definire le regole di pianificazione è diventata una standard de facto per tutti gli applicativi che consentono la pianificazione di attività.

La sintassi si basa su due soli elementi fondamentali: campi ed operatori.

- Campi: le regole di pianificazione sono caratterizzate da 5 campi che indicano rispettivamente minuti (0-59), ore (0-23), giorno del mese (1-31), mese (1-12 oppure jan, feb, mar...), giorno della settimana (0-6 oppure sun,mon,tue...).

- Operatori: permettono di specificare valori multipli all'interno di un campo e possono essere virgola (,), trattino (-) ed asterisco (*). Virgola permette di specificare una lista di valori, trattino consente di specificare un intervallo tra due valori mentre infine asterisco considera tutti i possibili valori del campo (ad esempio ogni minuto, ogni ora, ogni giorno). Quando si specifica un intervallo, sia questo finito o definito con * è possibile farlo seguire da uno slash (/) ed un valore che corrisponde allo step.

Nello specifico la sintassi CRON è stata utilizzata per definire due tipi di attività frequenti, pianificate per essere eseguite ogni 2 minuti, la configurazione risulta essere: `[*/2 * * * *]`.

4.2 Front End

Lo sviluppo front end dell'applicazione ha interessato la dashboard amministratore dedicata al docente. Questo strumento permette al docente di gestire i suoi workspace, di creare questionari e consultarne i risultati. Sono presenti alcune shortcut per raggiungere le schermate della web-app anche tramite comandi Slack e l'accesso è gestito tramite la funzionalità di "Login with Slack" che consente al docente di accedere col suo account Slack senza dover effettuare un processo di registrazione e senza dover scegliere ed inserire una password. L'applicazione si presenta come una Single Page Application scritta in Angular.

4.2.1 Struttura di un progetto Angular

Abbiamo già visto nei capitoli precedenti quali sono i concetti alla base di Angular ed in cosa le Single Page Application si distinguono dallo sviluppo web tradizinale, in questa sezione andremo ad esplorare come si presenta un progetto Angular e quali sono i suoi componenti fondamentali.

package.json e package-lock.json

Il package.json si presenta un po' come un manifesto, un documento di riconoscimento per il nostro progetto Angular. Questo file definisce quelle che sono le dipendenze del progetto ed è qui che i gestori di pacchetti come npm o yarn vengono a scrivere qualvolta viene aggiunta una nuova dipendenza. Inoltre è qui che in fase di installazione del progetto il gestore viene per leggere le dipendenze da installare. Le dipendenze possono essere dipendenze generiche quindi sempre richieste o dipendenze di sviluppo quindi non necessarie in ambiente di produzione.

Il package.json contiene anche tutta una serie di importanti metadati riguardanti il progetto: l'autore, la licenza, una descrizione e delle parole chiave ed altri ancora.

Il package-lock.json è un file generato automaticamente ogniqualvolta viene eseguita una operazione che modifica l'alberatura dei node_modules o il package.json.

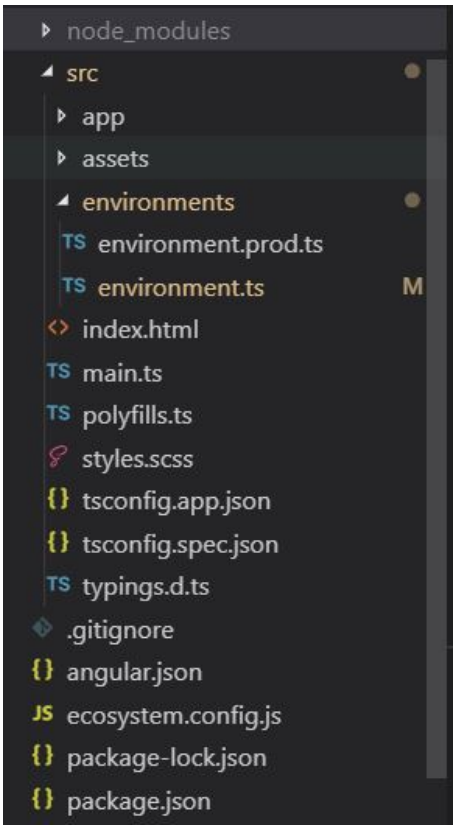

Figura 4.13. Struttura di un progetto Angular da VS Code.

Si tratta di una descrizione precisa, una footprint delle dipendenze atto a ricreare la medesima struttura in ogni ambiente e sulle macchine di ogni contributore. Permette inoltre di navigare indietro nel tempo la cronologia degli aggiornamenti ai moduli. Permette inoltre di velocizzare il proesso di installazione andando ad effettuare il download solo dei componenti di un modulo che sono effettivamente cambiati tra un update e l'altro.

Angular.json

Questo file fornisce le configurazioni per i tool di sviluppo e di build forniti dalla Angular CLI. Le configurazioni possono essere definite su due livelli: a livello di workspace o a livello di progetto. Essendo il workspace un livello superiore al progetto le configurazioni fatte in quest'ultimo risultano più specifiche e quindi vanno a sovrascrivere le generiche.

Agendo su questo file è possibile cambiare le versioni di tool come il builder e il server e modificarne il comportamento, modificare la struttura di base della directory di progetto, includere script JavaScript e fogli di stile.

à

tsconfig.json

La presenza di questo file nel la directory principale di un progetto ci annuncia che siamo davanti ad un progetto TypeScript. Il tsconfig.json contiene le direttive che il

compilatore TypeScript dovrà seguire. Di seguito un esempio di tsconfig.json preso da Didactic.

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "outDir": "./dist/out-tsc",
    "baseUrl": "src",
    "sourceMap": true,
    "moduleResolution": "node",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "es5",
    "typeRoots": [
      "node_modules/@types"
    ],
    "lib": [
      "es2016",
      "dom"
    ]
  }
}
```

Alcune delle opzioni riportate sono:

- target: la versione dello standard cui deve rifarsi l'output.
- moduleResolution: la strategia da usare per risolvere i file di dichiarazione dei moduli. Col parametro node questi verranno caricati da node_modules.
- outDir: il percorso in cui il compilatore andrà a creare il codice compilato.

Directory app

Directory che contiene effettivamente i component ed i servizi che compongono l'applicativo. La SPA è realizzata da un component principale che funge da root e che di default è definito col nome di "app". Altri due file importanti sono app.module.ts ed app.routing.ts. Una applicazione Angular molto grande può essere divisa in moduli, le configurazioni di default prevedono un unico modulo che realizza l'applicazione stessa; l'app.module dichiara tutti gli eventuali altri moduli importati ed esportati e i componenti dichiarati. Anche i servizi vanno registrati in un array di "providers".

Directory assets

Directory di default per file statici.

Directory environments

Directory di default per i file delle variabili d'ambiente. Per ciascun ambiente è possibile dichiarare un file di proprietà che vengono caricate in fase di build. Il file

environment.ts è quello di default ed è usato per l'ambiente di sviluppo mentre per gli altri ambienti è prevista la convenzione: "environment.nome_ambiente.ts.

In fase di build è necessario passare il parametro –env=nome_ambiente per caricare il giusto file di environment.

4.2.2 Gestione della persistenza: il locale storage

Il localestorage è uno spazio di memoria virtuale interno al browser dove è possibile salvare dati sotto forma di coppie chiave/valore e richiamarli, in questo modo:

```
localStorage.setItem("current_workspace_ID", "T8Z7XFVUM");
let currentWorkspaceID = localStorage.getItem("current_workspace_ID")
```

Il browser si occupa di persistere questi dati sul dispositivo dell'utente, in questo modo i dati non vengono persi alla chiusura del browser. Questi inoltre non hanno una scadenza.

Il locale storage deve poter contenere dati strutturati per essere davvero effettivo e permettere di realizzare delle logiche statefull, in questo ci viene in aiuto la serializzazione e deserializzazione da oggetti JavaScript a dati JSON:

```
1
2     let workspaceJS = {
3       name: "myAwesomeWorkspace",
4       team_id: "T8Z7XFVUM",
5       access_token: "this_is_secret_stuff",
6     }
7     localStorage.setItem("currentWorkspace", JSON.stringify(workspaceJS));
8
9     let workspaceJS =
          <Workspace>JSON.parse(localStorage.getItem('currentWorkspace'));
```

Figura 4.14. Lettura e scrittura di oggetti in local storage.

Tutte le funzionalità che richiedevano persistenza di stato e caching lato client sono state realizzate sfruttando lo storage locale in browser.

4.2.3 Guards

Come abbiamo detto in precedenza in una normale applicazione web la navigazione avviene richiedendo le pagine HTML statiche o generate dinamicamente dal server. In questo caso ad ogni richiesta il server può controllare che l'utente abbia i requisiti per accedere ad una specifica pagina o risorsa ed in caso contrario rilanciare un errore della famiglia 4.X.X.

Nel caso di una Single Page Application invece tutto il contenuto viene scaricato dal client in via preventiva e le viste sono generate dinamicamente dal browser, abbiamo quindi bisogno di uno strumento che consenta di verificare qualora lo stato dell'applicativo e/o dell'utente non consenta la navigazione su una specifica rotta. Questa funzionalità viene frequentemente utilizzata per effettuare controlli

di autenticazione e realizzare quindi aree private ma si presta a molti altri utilizzi come vedremo a breve.

Una guard è un servizio che implementa una tra 6 interfacce, ciascuna di queste realizza una logica differente e per tutte queste bisogna implementare il metodo omonimo. Il funzionamento di una guard è molto semplice: allo scatenarsi del trigger la guard viene valutata eseguendo il metodo implementato dell'interfaccia, qualora il metodo torni true la navigazione andrà a buon fine. Oltre a questo è ovviamente possibile effettuare altre operazioni nel mentre come chiamare servizi esterni, mostrare popup, controllare lo storage locale. Questi sono i tipi di Guard ad oggi previsti da Angular:

- *CanActivate*: attivata in fase di accesso su una rotta, in caso di esito positivo permette l'accesso.
- *CanActivateChild*: equivalente alla CanActivate ma per rotte figlio.
- *CanDeactivate*: attivata in fase di abbandono di una rotta. Utile per salvare dei dati o per mostrare all'utente un popup di conferma.
- *Resolve*: attivata in fase di atterraggio su una rotta che ha bisogno di richiedere dati ad un server remoto ma senza sicurezza sulla disponibilità degli stessi. La guard richiede il dato e concede l'accesso solo se questo è effettivamente disponibile.
- *CanLoad*: attivata anch'essa in fase di atterraggio su una rotta, controlla che un modulo *lazy loaded* possa essere caricato o meno.

AuthGuard per Didactic

Questo lavoro di tesi fa uso di diverse guard per le logiche più svariate ma sicuramente tra tutte la più significativa è la AuthGuard, una classe che implementa la guard CanActivate. Ogniqualvola l'utente prova ad accedere ad una rotta autenticata la guard esegue il metodo canActivate(): qualora l'utente dovesse risultare correttamente loggato la navigazione continuerà come previsto, in caso contrario viene caricata la pagina di login. Il metodo isUserLogged() controlla l'esistenza di un token di autenticazione in storage locale. Qualora invece il token dovesse esistessere ma non essere valido (perché scaduto) sarà il server, durante le chiamate API a verificarlo ed a restituire all'utente un errore 401 Unauthorized che sarà compito del front end gestire.

```
1  isUserLogged() {
2    this.isLoggedInUser = !!localStorage.getItem('token');
3    return this.isLoggedInUser;
4  }
```

Nello snippet precedente la notazione !! è utilizzata per effettuare un cast a booleano, il valore ritornato sarà true se l'item con key "token" esiste.

```
1
2  @Injectable()
3  export class AuthGuard implements CanActivate {
4    constructor(private router: Router, private auth: AdminService) { }
5    canActivate() {
6      if (this.auth.isUserLogged()) {
7        return true;
8      }
9      else {
10       this.router.navigate(['login']);
11     }
12   }
13 }
```

Figura 4.15. AuthGuard Angular per accesso area riservata

4.2.4 Interceptors

La versione 4.3 di Angular ha introdotto il package HttpClient, deprecando il predecessore Http. Con HttpClient sono stati introdotti gli HTTP Interceptor, che consentono di intercettare le response e le request delle chiamate HTTP effettuate dalla Single Page Application. Il funzionamento di un interceptor è del tutto comparabile ad un route middleware di un qualsiasi web framework, oppure Express. Per creare un interceptor in Angular è necessario definire un servizio che implementa l'interfaccia HttpInterceptor, importandola da @angular/common/http. La classe implementerà il metodo intercept, che verrà invocato per intercettare una richiesta o una risposta HTTP.

Il primo parametro è un oggetto che modella la richiesta HTTP e implementa l'interfaccia HttpRequest. Tramite questo oggetto è possibile effettuare manipolazioni della richiesta.La request viene passata per valore quindi non viene mai modificato l'oggetto originale. Il secondo parametro implementa l'interfaccia HttpHandler, la quale definisce il metodo handle. Questo prende in input una HttpRequest e restituisce un observable. La chiamata ad handle permette di passare il controllo al prossimo interceptor della catena. L'ordine di esecuzione dipende dall'ordine di provisioning nel file app.module.ts.

4.2.5 Reactive forms

I reactive form sono uno degli approcci allo sviluppo di form di input offerti da Angular. Un form reattivo in un qualsiasi momento è in un particolare stato ed ogni azione effettuata sul form lo muove verso un nuovo stato consistente. Un form reattivo è realizzato come un albero di oggetti di controllo dichiarato nella classe del componente in cui risiede, come vedremo in seguito. Il reactive form si basa su alcuni componenti base:

- FormControl: estende la classe AbstractControl ed implementa la maggior parte delle funzionalità di base per accedere ai valori, validarne lo status, gestire le interazioni utente. Ciascun formControl gestisce un campo in input.

- FormGroup: sostanzialmente funge da contenitore per una serie di formControl e tiene traccia del valore e dello stato del form. Aggrega i valori dei formControl in un unico oggetto dove il nome del controllo funge da chiave per la proprietà; anche il suo stato è dedotto dallo stato dei suoi figli. Corrisponde logicamente al form stesso.

- formArray:tiene traccia del valore e dello stato di un array di formGroup o di altri formArray.

Per gestire e dichiarare questi oggetti nei nostri componenti Angular mette a disposizione un servizio chiamato FormBuilder; altri non è che una classe helper che fornisce un modo più veloce e compatto per dichiarare i formControl ed i loro validatori. Di seguito un listato con il codice più significativo del componente `survey` che fornisce un form dinamico per la creazione di questionari. Si noti l'uso di formGroup innestati tramite l'uso di formArray. Ad un questionario è infatti possibile aggiungere e rimuovere dinamicamente delle domande, così come ad una domanda a risposta multipla è possibile aggiungere un indefinito numero di opzioni.

Il risultato è quello di un form con un variabile numero di form figli di cui il padre "eredita" i validatori. Lato template è necessario passare come variabile di input al componente form l'istanza del formGroup così creato, in questo modo:

```
<form [formGroup]="myForm" (ngSubmit)="submit(formDirective)">
```

Mentre i singoli campi HTML di input fanno riferimento al formControl tramite la proprietà `formControlName` in questo modo:

```
<input formControlName="survey_expiration_time"
    id="authoring-send-hour-picker" class="form-control"
                    type="time" required>
                </div>
```

4.2.6 Grafici con chart.js

Una delle core feature dell'applicativo è la generazione di report sulle risposte ottenute nei questionari. Questi sono utili poiché forniscono al docente uno strumento di rapida consultazione per poter trarre velocemente i dati più significativi dai questionari somministrati agli studenti.

Per la realizzazione di questa funzionalità esistono diverse librerie con un buon grado di interoperabilità con Angular, la scelta è ricaduta sul modulo Chart.js.

Chart.js è una libreria open source per la creazione di grafici HTML fortemente personalizzabili. Fornisce inoltre, in modo totalmente trasparente, supporto a tutta una serie di animazioni per le transizioni sul cambio dei dati, aggiunta di dataset e cambio colori. La libreria supporta nativamente il responsive.

L'utilizzo della libreria è piuttosto semplice, il modulo infatti fornisce un costruttore che accetta diversi parametri: type, data, options.

- type: stringa che definisce la tipologia di grafico da creare; tra i tanti tipi a disposizione ricordiamo bar, pie, line, radar.

```
// inizializzazione del formGroup
      this.myForm = this.formBuilder.group({
       title: ['', [Validators.required, Validators.minLength(5)]],
      audience: [{}, [Validators.required]],
      message: ['', [Validators.required]],
      survey_type: ['', [Validators.required]],
      survey_schedule: '',
      survey_schedule_day: '',
      survey_schedule_time: '',
      survey_expiration_day: ['', [Validators.required]],
      survey_expiration_time: ['', [Validators.required]],
      elements: this.formBuilder.array([this.createQuestion()])
    }, { validator: DatesCrossValidator });

      createQuestion(): FormGroup {
    // add question to the list
    return this.formBuilder.group({
      text: ['', [Validators.required]],
      type: ['', [Validators.required]],
      options: this.formBuilder.array([])

    })
  }

  addQuestion() {
    let quest = this.createQuestion();
    this.questionsArray.push(quest)

  }
    createOption(): FormGroup {
    return this.formBuilder.group({
      label: ['']
    })
  }
      addOption(question: FormGroup) {
    let opt = this.createOption();
    let array = question.get('options') as FormArray
    array.push(opt);

  }
    //custom validator delle date
    const DatesCrossValidator: ValidatorFn = (fg: FormGroup) => {
  const survey_schedule_day = fg.get('survey_schedule_day').value;
  const survey_schedule_time = fg.get('survey_schedule_time').value;
  const survey_expiration_day = fg.get('survey_expiration_day').value;
  const survey_expiration_time = fg.get('survey_expiration_time').value;
  let schedulation, expiration;
  if (survey_schedule_day != null && survey_schedule_time != null) {
    schedulation = new Date(survey_schedule_day + ' ' + survey_schedule_time + ':00');
  } else {
    schedulation = new Date();
  }
  expiration = new Date(survey_expiration_day + ' ' + survey_expiration_time + ':00');
  if (schedulation < new Date()) {
    return { range: true }; //il sondaggio non può essere schedulato nel passato
  }
  if (expiration < schedulation) {
    return { range: true }; //il sondaggio non può scadere prima di essere stato inviato
  } else return null;
}
```

Figura 4.16. Porzione di codice gestione formGroup per il questionario

- data: i dataset da rappresentare; a seconda della tipologia di grafico può prevedere una proprietà labels, un array di stringhe che caratterizza e specifica il significato dei valori in ingresso. Alla modifica dei dati corrisponde dinamicamente una modifica del grafico renderizzato a schermo, in tempo reale.

- options: tutte le possibili configurazioni e personalizzazioni del grafico. Posizione della legenda, titolo del grafico, configurazione degli assi (punto di origine, ordine di grandezza, salto, etichetta) ed ancora effetti di *mouseover*, *tooltip*, colori.

Ogni grafico risiede all'interno di un oggetto HTML Canvas, un elemento utilizzato come contenitore di elementi grafici realizzati usando JavaScript.

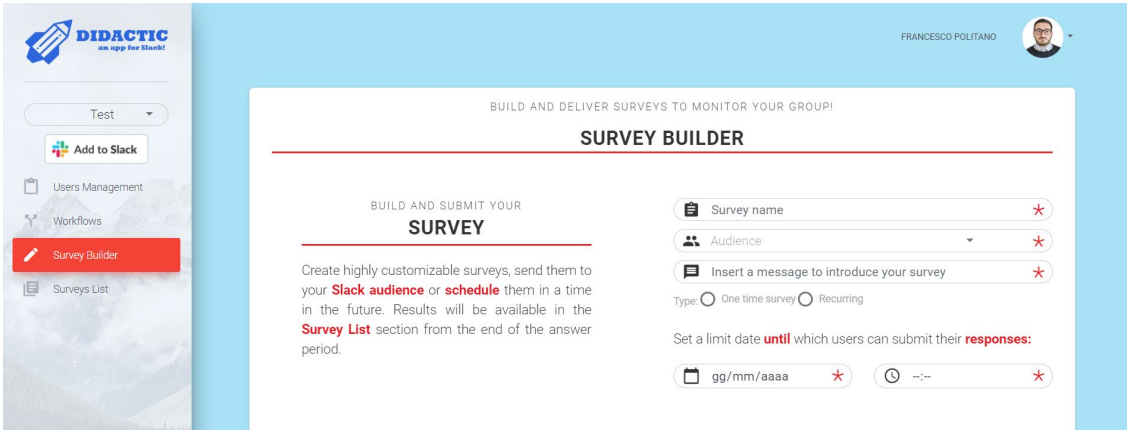

Figura 4.17. Strumento di creazione di un questionario realizzato tramite reactive form.

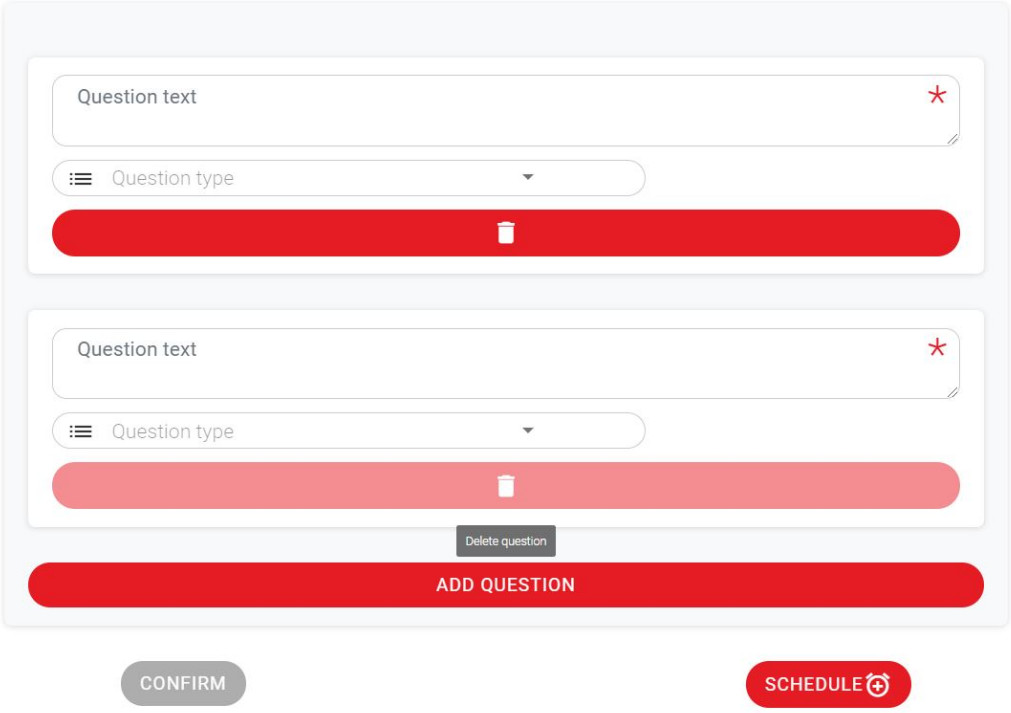

Figura 4.18. Form dinamico per l'aggiunta delle domande al questionario

Implementazione

L'implementazione dei report HTML dei questionari richiedeva la creazione e popolamento dinamici dei canvas contenenti i grafici necessari. Era quindi necessario che Angular potesse referenziare e gestire questi elementi HTML anonimi generati programmaticamente: questo è reso possibile dalla annotazione `@viewChildren()` questa prende come parametro una reference variable e ritorna un `chartElementRef` che possiede, tra le altre, una proprietà array contenente il riferimento a tutti gli elementi del DOM referenziati con la variabile in esame. Possiamo inoltre notare dal listato precedente che il canvas risulta visibile solo qualora esista effettivamente

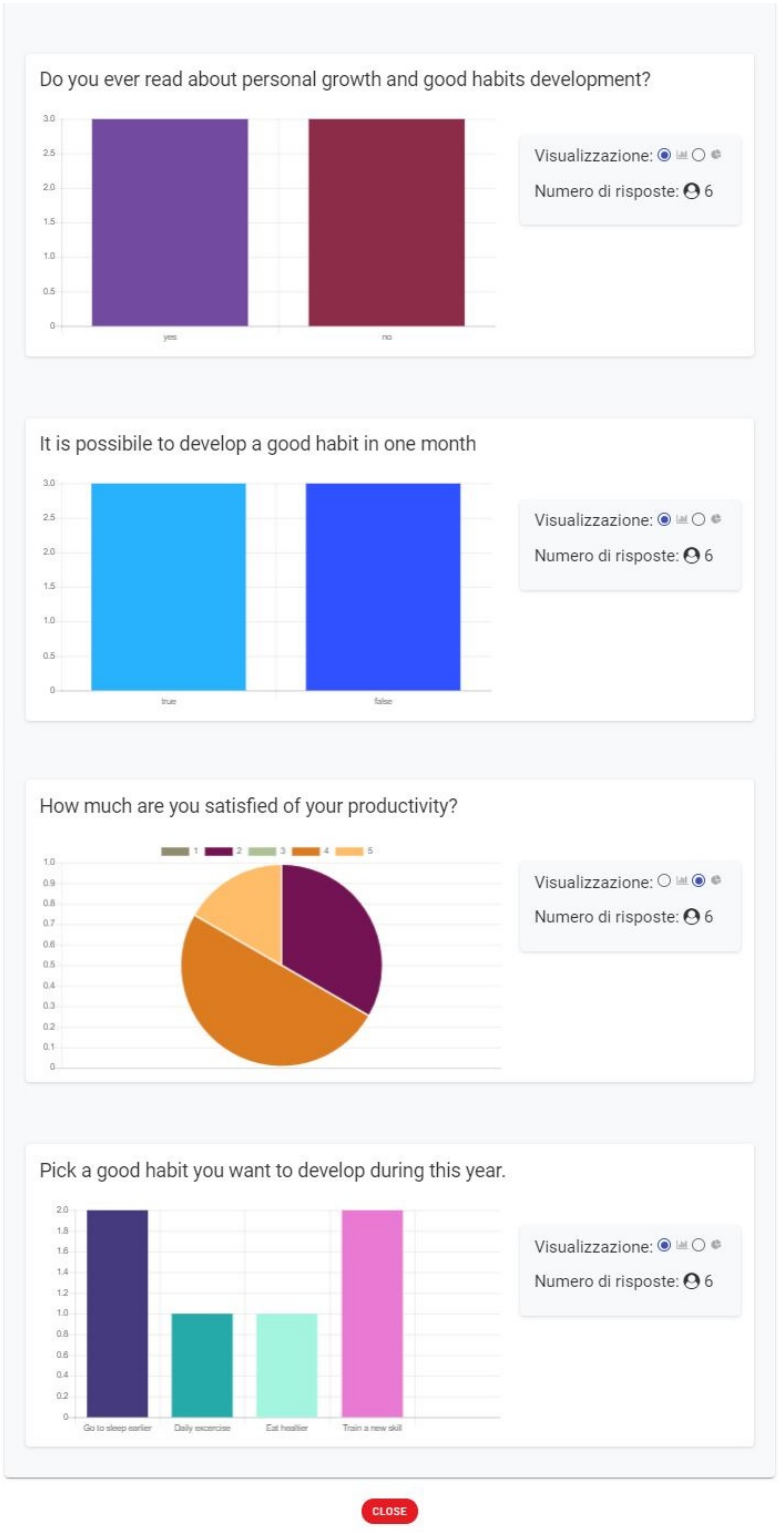

Figura 4.19. Esempio di report realizzato con Chart.js per Didactic

il grafico relativo, questo succede poiché alcune tipologie di domande hanno bisogno di una gestione differente. Ad esempio è il caso delle domande a risposta aperta.

Nel listato in figura 4.20 si vedono i due metodi principali per la gestione dei

```
<div class="card-body" *ngFor="let question of survey.questions; index as i">
    <div class="card">
      <h3 class="question-title">{{question.text}}</h3>
      <div class="card-body question-card">
        <div [hidden]="!graphs[i]" class="row">
          <div class="col-lg-8">
            <div [hidden]="!graphs[i]"><canvas #grafico id="i">{{ graphs[i] }}</canvas></div>
          </div>
          <div class="col-lg-4">
            <div class="card bg-light">
              <div class="card-body">
                <h4>
                  <mat-radio-group *ngIf="!isMobileMenu()">
                    <span style="font-style:initial">Visualizzazione:</span>
                    <mat-radio-button color="primary" value="1" [checked]="true"
                      (click)="switchGraphType('bar',i)">
                      <fa name=" fa-bar-chart" title="Bar chart"></fa>
                    </mat-radio-button>
                    <mat-radio-button color="primary" value="2" (click)="switchGraphType('pie',i)">
                      <fa name=" fa-pie-chart" title="Pie chart"></fa>
                    </mat-radio-button>
                  </mat-radio-group>
                </h4>
                <h4>
                  <span style="font-style:initial">Numero di risposte:</span>

                  <fa name="user-circle" title="Numero di risposte"></fa>
                  {{survey. statistics [i]. answersCount}}
                </h4>
              </div>
            </div>
          </div>
        </div>
        <div [hidden]="graphs[i]" class="row">
          <app-open-answers class="col-12" [survey]="survey" [questionIndex]=i></app-open-answers>
        </div>

      </div>
    </div>
  </div>
```

Figura 4.20. Template HTML creazione questionario

grafici: il metodo populate, che inizializza tutti i grafici relativi alle domande del questionari ed il metodo switchGraphType che si occupa dello switch di tipologia di grafico da line a pie o viceversa; questo avviene tramite distruzione dell'oggetto graph e reinstanziazione.

```
@ViewChildren('grafico') chartElementRef: any;
    popolate() {
  this.survey = this.modalShareVariables.getSurveyDetailsModalData();
  if (this.survey) {
    this.statusIconProperties = {};
    if (this.survey.schedulation > new Date()) {
      this.statusIconProperties.name = "far fa-clock";
      this.statusIconProperties.animation = "none"
      this.statusIconProperties. title = "programmato"
    }
    else if (this.survey.expiration < new Date()) {
      this.statusIconProperties.name = " fa-spinner";
      this.statusIconProperties.animation = "pulse"
      this.statusIconProperties. title = "in corso"
    }
    else {
      this.statusIconProperties.name = "fas fa-bell";
      this.statusIconProperties.animation = "none"
      this.statusIconProperties. title = "terminato"
    }
    this.graphs = new Array;
    for (let i = 0; i < this.survey.questions.length; i++) {
      if (this.survey.questions[i].type != "open") {
        this.graphs.push(new Chart(this.chartElementRef._results[i].nativeElement, {
          type: this.defaultType,
          data: {
            labels: this.survey. statistics [i]. visualization .labels,
            datasets: [this.survey. statistics [i]. visualization ]
          },
          options: {
            scales: {
              yAxes: [{
                display: true,
                ticks: {
                  beginAtZero: true
                }
              }]
            },
            legend: {
```

```
40                  display: this.defaultType == "pie"
41                }
42              }
43            }));
44          } else {
45            this.graphs.push(undefined);
46          }
47        }
48      }
49    }
50
51    switchGraphType(type: string, position: number) {
52
53      let chart = this.graphs[position];
54      chart.destroy();
55      this.graphs[position] = new Chart(this.chartElementRef._results[position].nativeElement, {
56        type: type,
57        data: {
58          labels: this.survey.statistics[position].visualization.labels,
59          datasets: [this.survey.statistics[position].visualization]
60        }, options: {
61          scales: {
62            yAxes: [{
63              display: true,
64              ticks: {
65                beginAtZero: true
66              }
67            }]
68          },
69          legend: {
70            display: type == "pie"
71          }
72        }
73      })
74    }
```

4.3 Continuous Integration e Continuous Deployment

La Continuous Integration ed il Continuous Deployment sono due pratiche moderne per lo sviluppo di software. La Continuous Integration (CI) è il processo di automazione della build e del testing del codice prodotto a seguito di una modifica alla code base nel sistema di version control.

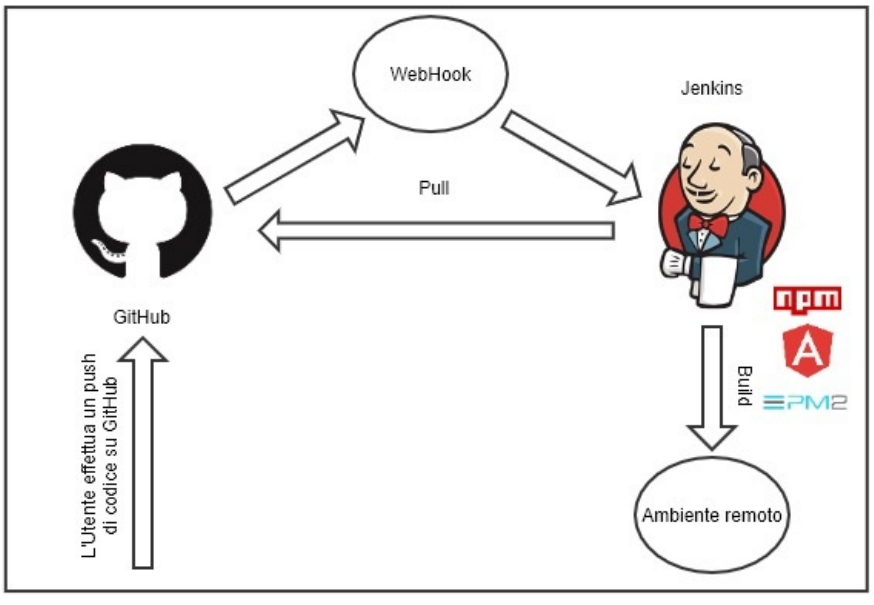

Figura 4.21. Flusso di CI/CD

Il continuous Development (CD) è invece diretta conseguenza del CI e consiste nell'automatizzazione del processo di deploy dell'applicativo a seguito di un CI andato a buon fine.

L'obiettivo è quello di minimizzare i tempi di consegna e ridurre al minimo il tempo da dedicare alle operazioni meccaniche e a scarso valore aggiunto, automatizzandole.

Questi processi chiaramente risultano molto vantaggiosi in ambiti di produzione e con codice sviluppato in team con metodologie Agile, non mancano però i vantaggi anche per lo sviluppo in solo. Ho introdotto la CI/CD, oltre che per motivi di studio, per automatizzare e velocizzare i deploy degli applicativi sull'ambiente remoto.

4.3.1 Installazione di Jenkins

Get Jenkins:

```
//add the repository key to the system
wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo
    apt-key add -
//append the Debian package repository address to the server's echo deb
    https://pkg.jenkins.io/debian-stable binary/ | sudo tee
    /etc/apt/sources.list.d/jenkins.list
//update
sudo apt-get update
```

Install Jenkins:

```
sudo apt-get install jenkins
```

Start Jenkins:

```
sudo systemctl status jenkins
```

Jenkins di dafault è in ascolto sulla porta 8080. Trattandosi di una GUI esposta all'esterno è sempre preferibile cambiare le impostazioni di base, per cambiare la porta sulla quale ascolta Jenkins è necessario recarsi nella directory di installazione, aprire in modifica il file jenkins.xml ed intervenire sulla proprietà httpPort nei JENKINS_ARGS. Perché la modifica sia effettiva è necessario riavviare Jenkins.

A questo punto è possibile accedere alla interfaccia grafica di Jenkins sulla porta esposta e completare la configurazione guidata.

4.3.2 Configurazione di processi Jenkins

Per creare i processi di deploy per le applicazioni di back-end e front-end bisogna agire su queste aree semantiche nel form di configurazione:

- Source Code Management: Selezionando git come fonte è possibile inserire il link relativo alla repository che ospita il codice sorgente.

- Build Triggers: selezionando l'opzione GitHub hook trigger for GITScm polling il job Jenkins verrà eseguito ad ogni push su Master.
- Add Build Step: selezionando Execute Shell option è possibile scrivere i comandi bash che vogliamo vengano eseguiti. Questo è lo script di deploy dell'app di back-end e dei cron:

```
npm install
pm2 start ecosystem.config.js --env=production
```

L'app di front-end prevede un passaggio in più per la build della SPA dal progetto Angular:

```
npm install
ng build --prod
pm2 start ecosystem.config.js --env=production
```

Abbiamo detto che Jenkins espone un endpoint sul quale ascolta le chiamate effettuate da Github in corrispondenza di operazioni di push e Merge Request su Master, per fare questo bisogna impostare un hook sul repository Github recandosi nelle impostazioni del repository sotto la voce "webhooks". Qui è possibile inserire l'indirizzo su cui ascolta Jenkins e gli eventi dei quali vogliamo essere informati, di dafault la URI su cui ascolta il plugin Jenkins per Github è: `http://mioindirizzo:9999/github-webhook/`.

4.4 Configurazione di NGINX come reverse proxy

Come detto in precedenza in questa tesi, uno dei punti di forza di NGINX è la semplicità di utilizzo e configurazione. Tutte le configurazioni infatti passano per dei file di testo in formato comprensibile all'uomo.

Quello che NGINX va a fare in questo progetto è agire da reverse proxy: si frappone tra i client ed uno o più server ed evade le richieste come fosse esso stesso il server, indirizzandole al servizio giusto in modo totalmente trasparente al client. Questo si è reso necessario dal momento che sotto un solo host name sono in ascolto, su porte diverse, due servizi.

Il discriminante che permette al reverse proxy di decidere a quale server indirizzare una data richiesta è il path della risorsa. I file di configurazione risiedono nella directory `/etc/nginx/conf.d/`. Un file di configurazione di Nginx può contenere più blocchi server, caratterizzati da diverse proprietà, in particolare questa soluzione prevede un solo blocco server così preposto:

```
server{
    server_name www.francescopolitano.dev;

    location /didactic/api {
    proxy_pass http://localhost:3000;
    ...
    }
```

```
    location / {
    proxy_pass http://localhost:8080;
    ...
    }
    ...
}
```

Ciascun server è caratterizzato da una proprietà *server_ name* che indica quale è il dominio registrato per cui ascolta il blocco specifico di configurazione.

La direttiva *proxy_ pass* è quella che realizza il reverse proxy. Questa infatti specifica che tutte le richieste che coincidono con il path del blocco *location* devono essere indirizzate all'argomento della direttiva. Il caso in esame prevede due direttive:

- `/` (root): le chiamate in arrivo su questo path vengono girate sulla porta 8080 che ospita in ascolto il server Node che si occupa di fornire la SPA Angular di back office.

- `/didactic/api` : questo path parziale caratterizza le chiamate al back-end, siano queste provenienti dal back office o da Slack. Il server Nodejs di back-end è in ascolto sulla porta 3000

4.5 Certificato SSL con Let's Encrypt

4.5.1 SSL e HTTPS

SSL (Secure Socket Layer) è stato proposto da Netscape Communications ed è un protocollo di trasporto sicuro per il web. Lo stack TCP/IP su cui ad oggi si basa la maggior parte del web non prevede meccanismi di sicurezza end-to-end nelle comunicazioni tra una sorgente (server) ed un destinatario (client). SSL sopperisce a queste mancanze fornendo autenticazione, integrità dei dati e confidenzialità operando al di sopra del livello di trasporto. SSL si può applicare a tutti i protocolli basati su TCP quali HTTP, FTP, SMTP e TELNET.

SSL è basato sul concetto di canale sicuro: all'avvio di una comunicazione le parti si presentano (spesso solo il server ha questo onere) tramite certificato, negoziano gli algoritmi crittografici da utilizzare e si scambiano una coppia asimmetrica di chiavi tramite algoritmi appositi come il Diffie-Hellman, che è il più diffuso, o RSA. Per evitare di rinegoziare ogni volta i parametri e riaprire e chiudere il canale ad ogni connessione il server presenta al client un id di sessione che questi può utilizzare per connessioni ripetute.

Il protocollo HTTP quando fa uso della tecnologia SSL è detto HTTPS. Per sfruttare il protocollo sicuro HTTPS un server ha bisogno di un certificato rilasciato da una certification autority, una terza parte fidata che garantisce per l'identità del server. I certificati richiesti dal protocollo SSL aderiscono allo standard X.509 e vengono utilizzati per validare l'identità delle controparti (firma digitale) o per la cifratura di stream di dati o documenti (riservatezza dei dati).

4.5.2 Uso di HTTPS in Didactic

Sono due i motivi principali per cui si è rivelato assolutamente mandatorio l'utilizzo di connessioni sicure e l'installazione di un certificato.

- HTTPS è lo standard del web: già da tempo i browser scoraggiavano l'utente dal navigare su siti considerati "non sicuri", specialmente quando questi presentavano form per l'inserimento di dati sensibili. Anche gli utenti hanno imparato a guardare con diffidenza ai siti sprovvisti di certificato. Recentemente Google ha iniziato a penalizzare nei ranking del suo motore di ricerca i siti che non supportano HTTPS.
- Slack richiede HTTPS: come detto in altri punti di questa tesi Slack può contattare il servizio di back end su degli endpoint esposti ad hoc. Questi servono per ascoltare determinati eventi interni al workspace. Slack richiede che tutti questi endpoint siano messi in sicurezza tramite protocollo HTTP.

4.5.3 Configurazione tramite Let's Encrypt

Lets Encrypt è una certification autority open source, automatizzata e che fornisce certificati gratuitamente. La mission del progetto è quella di riuscire a mettere in sicurezza tutte le comunicazioni sul web, è per questo che Let's Encrypt fornisce questo servizio in modo rapido e gratuito. Prima dell'avvento del modello Let's Encrypt richiedere un certificato ad una CA poteva essere un procedimento lungo e tedioso: era infatti necessario che il richiedente fosse fisicamente presente presso gli uffici della CA per il riconoscimento e la richiesta, la quale aveva spesso un costo in denaro non indifferente. Let's Encrypt è stata una vera e propria rivoluzione in questo campo.

Per verificare che il richiedente sia davvero proprietario del dominio per cui si fa richiesta Let's Encrypt si basa su un protocollo di sfida chiamato ACME Automated Certificate Management Environment.

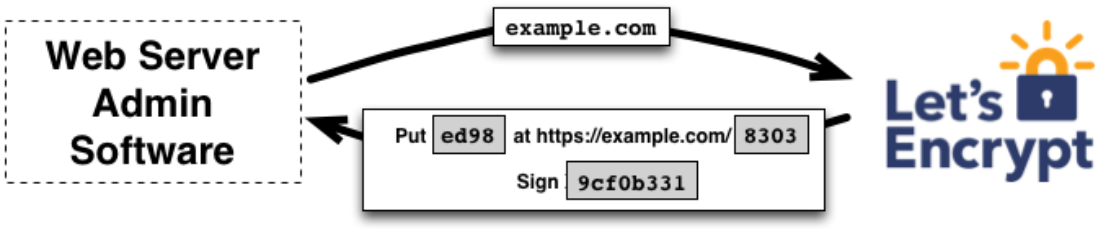

Figura 4.22. Sfida ACME. Immagine dalla documentazione ufficiale Let's Encrypt.

Il richiedente chiede a Let's Encrypt di verificare il nome example.com; la Certification Authority sottometterà al richiedente una o più sfide per dimostrarne la proprietà del dominio. Questa sfida avviene contattando una specifica URI su cui il server deve esporre un file come richiesto da Let's Encrypt. Il server deve inoltre firmare un token con la sua chiave privata e rimandarlo indietro al mittente.

Let's Encrypt fornisce uno strumento di configurazione guidata per la creazione del certificato e per la risoluzione della sfida: il Certbot. Tramite Certbot è possibile

configurare automaticamente Nginx per il supporto a SSL; il programma cerca tra i file di configurazione di Nginx il blocco server legato al dominio per il quale si sta facendo richiesta di certificato, aggiunge automaticamente delle proprietà alla configurazione e riavvia Nginx.

```
server {
    server_name www.francescopolitano.dev;

    ...

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate
        /etc/letsencrypt/live/www.francescopolitano.dev/fullchain.pem; #
        managed by Certbot
    ssl_certificate_key
        /etc/letsencrypt/live/www.francescopolitano.dev/privkey.pem; #
        managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

    ...
}
server {
    if ($host = www.francescopolitano.dev) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80 default_server;
        listen [::]:80 default_server;

        server_name www.francescopolitano.dev;
    return 404; # managed by Certbot


}
```

4.6 Templating JSON per i messaggi Slack

Una delle sfide più interessanti nello sviluppo dell'applicativo è stata quella di realizzare un flusso logico di sottomissione del questionario in un contesto, quello delle Message API di Slack, completamente stateles.

Quello di cui avevo bisogno era una procedura standardizzata per trasformare un questionario creato dall'utente in una sequenza di messaggi interattivi che guidasse lo studente nel completamento del questionario. Il tutto dinamicamente e senza un' espicito supporto dalle API di Slack.

Qui di seguito vado a presentare una veloce introduzione alle API di messaggistica di Slack dopo la quale mi soffermerò sulle funzionalità che ho sfruttato per

realizzare flussi di interazione chiari ed intuitivi.

4.6.1 Messaggi in Slack

Un messaggio Slack non prevede semplice testo. Un messaggio può essere diviso in sezioni, ciascuna di queste può presentare un titolo, del testo, dei media e dei componenti interattivi quali bottoni o drop-down menu.

Le API di Slack permettono di comporre messaggi complessi a piacere tramite una convenzione basata sul formato JSON.

```
{
    "text": "This is a simple text only message"
}
```

Figura 4.23. Un payload JSON per un messaggio di puro testo

Figura 4.24. Un messaggio di solo testo

Prossimamente in questa sezione tratteremo le diverse modalità con cui è possibile inviare messaggi così composti agli utenti del workspace Slack.

4.6.2 Formattare il testo

Per formattare il testo contenuto nei suoi messaggi Slack ha previsto delle apposite regole di markup, simili a quelle del noto linguaggio Markdown. Le regole supportate includono: “‘pre“‘, ‘code‘, _italic_, *bold*, e strike .

Inoltre Slack consente di scrivere particolari direttive per la formattazione di dati specifici; queste vengono preprocessate lato server prima di essere inviate ai dispositivi dei destinatari, eccone due che sono ricorrenti nello sviluppo di questa tesi:

- Tag di canali o utenti: scrivendo tra parentesi angolari l’id univoco di un canale preceduto da # o l’id di un utente preceduto da @ è possibile taggare l’utente o il canale. Slack si occuperà di sostituire il markup con il nome opportuno al quale sarà associato un collegamento ipertestuale.

  L’utilizzo di tag risulta molto utile per personalizzare le interazioni con il chat bot o per fornire all’utente scorciatoie per raggiungere canali o chat private.

- Date: un messaggio contenente una data o un timestamp dovrebbe sempre mostrare i dati secondo la timezone dell’utente che li visualizza; questo è possibile grazie al comando <!date> il quale accetta un numero nello standard di Tempo Unix ed una stringa che determina le regole di formattazione da seguire.

4.6.3 Messaggi complessi: gli allegati

Gli allegati permettono di creare messaggi più complessi e strutturati, quindi più utili ed efficaci. inoltre sono proprio gli allegati ad ospitare gli elementi interattivi, definiti azioni.

Un messaggio può contenere fino a 100 allegati, ma Slack consiglia di non superare i 20 per non inficiare sulla leggibilità del messaggio. Un allegato è definito come un oggetto JSON in un array *attachments* e prevede diverse proprietà, le quali corrispondono solitamente ad elementi visuali che andranno poi a comporre il messaggio.

Essendo la costruzione di messaggi complessi ed interattivi la funzionalità core che ha permesso la realizzazione di questa applicazione, nonché per rendere più chiare le scelte implementative che andrò a presentare in seguito, mi soffermerò adesso sui dettagli di questa funzionalità delle Slack API.

color

Questo valore opzionale è utilizzato per colorare la barra laterale sinistra che contorna uno specifico allegato.Utilissimo per comunicare in modo veloce e diretto le finalità della sezione, accetta dei valori predefiniti come *good*, *warning*, *danger* o un generico codice colore esadecimale.

pretext

Un testo opzionale che appare immediatamente prima del blocco allegato.

Parametri autore

Passare questi parametri permette di mostrare una sezione speciale in cima all'allegato la quale contiene:

- author_name: obbligatorio
- author_link: applica un collegamento ipertestuale al nome dell'autore.
- author_icon: URL che mostra una immagine 16x16 px a sinistra del nome dell' autore.

title e title_link

Il titolo viene mostrato come un testo più grande, in grassetto, in cima all'allegato. Passando una URL valida al parametro title_link il titolo sarà un collegamento ipertestuale.

text

Il testo principale di un allegato.

fields

I field sono definiti come un array nel quale ciascun elemento è definito da una serie di proprietà in formato chiave/valore.

- title: Appare come una intestazione in grassetto sopra il testo del field; non può contenere markup.
- value: il testo del field, può essere formattato e multilinea.
- short: un flag opzionale che qualora valorizzato indica a Slack di mostrare il field sul 50% della riga.

image_url e thumb_url

Un URL valido ad una immagine che verrà mostrata nel corpo dell'allegato.

Parametri footer

Breve testo per contestualizzare e identificare un allegato.Mostrato in pedice e limitato a 300 caratteri.

- footer: il testo del footer
- footer_icon:URL di una immagine che verrà mostrata accanto al footer.
- ts (timestamp): un intero in tempo Unix, utile quando l'allegato fa riferimento a fatti collocabili nel tempo come articoli di giornale o eventi. La precisione del testo risultante dipenderà dalla distanza della data dal presente.

4.6.4 Messaggi interattivi: bottoni, menù, dialog

I messaggi interattivi conservano tutte le caratteristiche viste fino ad ora ma presentano inoltre bottoni, menù e azioni personalizzate che permettono l'interazione con l'utente. I messaggi interattivi inoltre mutano col tempo proprio in base alle interazioni con l'utente.

Ogniqualvolta l'utente esegue una azione su un messaggio interattivo Slack interroga la URL dell'applicazione associata a tale richiesta inviando una richiesta comprendente tutte le informazioni necessarie ad identificare l'origine del messaggio ed il contesto nonché i valori specifici associati all'azione.

La richiesta contiene inoltre un `response\_url` che l'applicativo contatterà per proseguire il flusso legato all'azione. Questa URL apre una "sessione" tra l'app e Slack, della durata di 30 minuti. Ad ogni azione di Didactic corrisponde un feedback grafico per l'utente: accettare un questionario, ad esempio, farà apparire la prima domanda; rispondere ad una domanda porterà automaticamente alla successiva e terminare il questionario farà comparire un messaggio di conferma. Questo è possibile poiché tramite il metodo API `chat.update` si può andare a sovrascrivere il messaggio originale creando un vero e proprio workflow interattivo per l'utente.

4.6.5 Implementazione: templating e parametrizzazione

Per utilizzare lo strumento dei messaggi interattivi per realizzare le funzionalità dell'applicativo è stato necessario partire da una struttura base di cosa e come ciascuno step avrebbe dovuto comunicare e parametrizzarla rendendola indipendente dai contenuti.

Quello che ho fatto è stato definire la struttura dei messaggi e il tipo di informazione da veicolare, ho così realizzato dei template JSON custom contenenti dei placeholder dove necessario. Questi template vengono richiamati da un processo di building dei messaggi che va a selezionare il template adatto all'azione appena effettuata e a sostituire al suo interno i placeholder con i dati reali.

Quello riportato di seguito è il template del messaggio che annuncia la disponibilità di un nuovo questionario. Il messaggio presenta due bottoni che permettono di accettare o declinare il questionario. Il tasto di rifiuto prevede una ulteriore conferma.

```
"type": "new_survey",
"text": "C'è un nuovo questionario per te <user_id>!",
"attachments": [
   {
      "title": "titolo del sondaggio",
      "text": "Optional text that appears within the attachment",
      "callback_id": "<id_high_level_survey>",
      "color": "#3AA3E3",
      "fields": [
         {
            "title": "Domande",
            "value": "<q_num> domande",
            "short": true
         },
         {
            "title": "Data di scadenza",
            "value": "<date>",
            "short": true
         }
      ],
      "actions": [
         {
            "name": "start-survey",
            "text": "Start survey",
            "style": "primary",
            "type": "button",
            "value": "true"
         },
         {
            "name": "start-survey",
            "text": "Reject",
            "style": "danger",
            "type": "button",
            "value": "false",
            "confirm": {
               "title": "Sei sicuro?",
               "text": "Non potrai più rispondere a questo questionario.",
               "ok_text": "Ok",
               "dismiss_text": "No"
            }
         }
      ],
      "footer": "Didactic App",
      "footer_icon": "https://platform.slack-edge.com/img/default_application_icon.png",
      "ts": 123456789
   }
]
```

Figura 4.25. Template JSON nuovo questionario

Questo template prevede i seguenti placeholder:

- <user_id> : id univoco dell'utente. Slack andrà a sostituirlo con un tag allo user composto da nome e cognome.

- <id_high_level_survey> : id del questionario, verrà poi passato tramite la action perché possa mostrare il questionario corretto.
- <q_num>: numero di domande di cui è compsto il questionario.
- <date>: data di termine massimo per l'inserimento di risposte.

Altro template che merita un approfondimento è quello della domanda: Questo

```
"type": "question",
   "text": "<user_id> hai un questionario in corso:",
   "attachments": [
      {
         "title": "<survey-title>Domanda <done/all>",
         "text": "<question_text>",
         "callback_id": "<id_high_level_survey>",
         "color": "#3AA3E3",
         "actions": []
      },
      {
         "title": "",
         "text": "",
         "callback_id": "<id_high_level_survey>",
         "footer": "<symbols> (<done/all>)",
         "actions": [
            {
               "name": "<question_pos>",
               "text": "Previus question!",
               "type": "button",
               "value": "back"
            },
            {
               "name": "<question_pos>",
               "text": "Next question!,
               "type": "button",
               "value": "next"
            },
            {
               "name": "Confirm",
               "text": "conferma",
               "style": "primary",
               "type": "button",
               "value": "Confirm",
               "confirm": {
                  "title": "Sei sicuro?",
                  "text": "Una volta confermato non potrai più modificare o aggiungere risposte.",
                  "ok_text": "Ok",
                  "dismiss_text": "Annulla"
               }
            }
         ]
      }
   ]
```

Figura 4.26. Template JSON domanda.

template prevede i seguenti placeholder:

- <user_id> : id univoco dell'utente.
- <id_high_level_survey> : id del questionario. questionario corretto.
- <survey-title> : titolo del questionario da visualizzare.
- <question_text>: testo della domanda da visualizzare.
- <symbols>: spunte che mostrano il livello di completamento del questionario.
- <done/all>: numero di domande alle quali è stata data una risposta su numero di domande totali.

Così facendo un messaggio interattivo rappresentante un questionario è identificato univocamente tramite gli id dell'utente e del questionario. Questo permette di conservarne lo stato pur essendo di per sé il questionario basato su uno strumento stateless. In database ad ogni questionario corrisponde un array degli utenti che hanno partecipato e relative risposte. Il parametro <question_pos> permette di muoversi lungo il questionario effettuando un passo avanti o uno indietro.

Capitolo 5

Conclusioni

Al termine di questo lavoro di tesi è stato realizzato un applicativo funzionante, hostato in rete e comprendente le funzionalità chiave alla base di questa tesi. Quello che si è cercato di ottenere è uno strumento che metta i docenti nella condizione di offrire agli studenti un servizio efficiente e personalizzato e di incentivare così il loro apprendimento. Monitorare l'andamento dei propri corsi e ricevere costanti feedback sulla qualità dell'insegnamento è la funzionalità principale del sistema e tramite i questionari ed il dialogo instaurato nell'applicazione di messaggistica il docente ha maggiori probabilità di coinvolgere e sfidare gli studenti mettendoli in condizione di sviluppare al meglio le loro capacità secondo le teorie sul Flow di Mihaly Csikszentmihalyi.

Le funzionalità sviluppate vanno a comporre solo il fulcro del prodotto realizzato che potremmo definire come MVP (Minimum Viable Product). Lo scopo è stato quello di valutare una delle possibili strade da seguire nello sviluppo di un'applicazione per il monitoraggio dello stato emotivo e delle competenze degli studenti.

Quello che abbiamo evinto lavorando a questo sviluppo è che la soluzione di integrare lo strumento di raccolta ed elaborazione dei feedback ad un applicativo di messaggistica non è in assoluto la scelta migliore. Da un lato è interessante l'opportunità di integrare in modo stretto il workflow applicativo all'interno di una applicazione di messaggistica completa; d'altro canto disaccoppiare le due funzionalità lascerebbe molta più libertà al docente di sfruttare lo strumento che preferisce per la messaggistica e le comunicazioni (o di non usarne affatto) e lascerebbe allo sviluppatore maggiori libertà nello sviluppo di una interfaccia gradevole ed una user experience migliore.

5.1 Test pratico in aula

In data 5 Giugno 2019 si è svolto un test operativo in aula durante il corso di Programmazione di Sistema per l'anno accademico 2018/2019 tenuto dal Prof. Giovanni Malnati. Un sondaggio riguardante il corso è stato pubblicato su un workspace Slack creato per l'occasione.

Al sondaggio hanno risposto 41 studenti nell'arco di circa 10 minuti. Pur essendo l'applicativo ospitato su una macchina di test dalle basse prestazioni non si sono notati rallentamenti.

Non si è reso inoltre necessario particolare supporto agli studenti durante l'installazione e la login in Slack.

5.2 Sviluppi futuri

Al momento l'applicazione consente al docente di creare e programmare questionari e agli studenti di rispondere ad essi, inoltre sono presenti tutte le funzionalità di messaggistica istantanea offerte da Slack.

Come più volte ricordato nel corso di questa tesi quello che si è andato a realizzare è un prototipo, volendo un MVP (Minimun Viable Product), un prodotto che implementa un ristrettissimo core di funzionalità che funge quindi da studio di fattibilità e da demo per la validazione da parte degli stakeholder (clienti o generici utenti).

Supporto integrato per l'analisi del Flow

Questa funzionalità è pensata per rendere disponibile *out of the box* al docente uno strumento integrato, automatizzato e preciso per il monitoraggio dello stato d'animo degli studenti all'interno del grafico del Flow. L'applicazione dovrebbe fornire una serie di domande preimpostate, realizzate da uno psicologo fedelmente alle metodologie e normative ufficiali per la somministrazione di questionari psicologici. A quel punto il docente deve solo scegliere se attivare o meno la funzionalità, in caso positivo l'applicazione monitorerà costantemente lo stato d'animo degli studenti tramite questionari frequenti e informerà il docente sul generico andamento della classe e fornirà dei consigli su dove intervenire per incentivare l'apprendimento.

Applicazione su lunghi periodi

L'unico interesse non è ovviamente solo quello di rilevare le migliori scelte implementative ma è soprattutto quello di validare il metodo didattico ricavato dagli studi di Csikszentmihalyi. Sarebbe quindi opportuno effettuare un lungo test sul campo del metodo esposto in queste pagine; applicare quindi la teoria del Flow ad una classe di studenti di un corso universitario e realizzare lo scenario esposto nel capitolo sull'analisi del problema. In questo modo sarebbe possibile raccogliere feedback dagli stessi studenti e dai docenti per evidenziare eventuali punti di forza o svantaggi del metodo.

Altri sviluppi

Alcuni sviluppi futuri interessanti potrebbero essere:

- Gestione di workflow personalizzati: tramite hook ed event listener offerti da Slack è possibile definire degli automatismi basati su condizioni predefinite come una specifica azione eseguita dallo studente o un suo comportamento ripetuto. Un esempio può essere la creazione di un questionario di benvenuto, automaticamente somministrato ai nuovi studenti al loro primo accesso al workspace. Una possibilità basata invece sul comportamento è quella di

monitorare il grado di soddisfazione degli studenti che smettono di interagire con l'applicazione, così da poterne indagare le cause. Le possibilità sono pressocché infinite

- Maggiori funzionalità interne a Slack: utilizzare l'applicazione senza mai abbandonare Slack è comodo, per questo sarebbe auspicabile lo sviluppo di nuove funzionalità e menù navigabili da Slack tramite comando. Attualmente è possibile visionare i questionari passati e consultarne alcune statistiche; sarebbe interessante poter costruire e schedulare semplici questionari da Slack, magari realizzando una categoria a parte di sondaggi più snelli e concisi e con scadenza molto ristretta.

Bibliografia

[1] **Csikszentmihalyi M.,** *Flow*, **Harper Perennial (1990).**

[2] **Csikszentmihalyi M.,** *Optimal Experience: Psychological Studies of Flow in Consciousness*, **Harper Perennial (1992).**

[3] **Csikszentmihalyi M.,** *Optimal Applications of Flow in Human Development and Education*, **Springer (2014).**

[4] **Shernoff D., Csikszentmihalyi M.,** *Student Engagement in High School Classrooms from the Perspective of Flow Theory*, **American Psychological Association (2003).**

[5] **AA.VV,** *EEE recommended practice for software requirements specifications IEEE 830-1993*, **Software & Systems Engineering Standards Committee (1994).**

[6] **institute of electrical and electronics engineers,** *EEE recommended practice for software requirements specifications IEEE 830-1993*, **Software & Systems Engineering Standards Committee (1994).**

[7] **Kyle Simpson,** *You don't know JS*, **O'Reilly (2014).**

[8] **Alex Young, Bradley Meck, Mike Cantelon,** *Node.js in action, second edition*, **Manning Pubblications (2017).**

[9] **James Kurose, Ross Keith,** *Computer Networking: A Top-Down Approach, Global Edition* , **Pearson (2016).**

Sitografia

[1] Mihaly Csikszentmihalyi TED Talk 2014, https://youtu.be/I_u-Eh3h7Mo
[2] JavaScript Object Notation, https://www.json.org/
[3] Model-View-Controller, http://www.claudiodesio.com/ooa&d/mvc.htm/
[4] REST,
https://en.wikipedia.org/wiki/Representational_state_transfer
[5] MongoDB Documentation https://docs.mongodb.com/
[6] TypeScript Documentation,
https://www.typescriptlang.org/docs/home.html
[7] Angular Documentation, https://angular.io/api/core/ViewChildren
[8] Bootstrap Documentation,
https://getbootstrap.com/docs/4.3/getting-started/introduction/
[9] Slack Documentation, https://api.slack.com/
[10] Nginx Documentation, https://nginx.org/en/docs/
[11] Jenkins Documentation, https://jenkins.io/doc/
[12] PM2 Documentation,
http://pm2.keymetrics.io/docs/usage/pm2-doc-single-page/
[13] Oauth 2.0, https://oauth.net/2/
[14] Statistiche sull'utilizzo di NGINX sul web,
https://w3techs.com/technologies/details/ws-nginx/all/all
[15] The OpenSSL project, http://www.openssl.org/
[16] Let's encrypt Documentation, https://letsencrypt.org/docs/
[17] DRM Business statistics: Slack statistics and facts,
https://expandedramblings.com/index.php/slack-statistics/