POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Annual: arrivare in vetta assieme

Relatore

Prof. Giovanni MALNATI

Candidato

Andrea COMINELLI

Dicembre 2023

Sommario

Negli ultimi anni, si è assistito ad un forte aumento del numero di ragazzi che soffrono di solitudine e ansia sociale e la pandemia da COVID19 ha aggravato ulteriormente la situazione, perpetuando questi malesseri anche dopo la cessazione delle restrizioni. Diversi studi hanno riportato che la frustrazione e la solitudine cronica possono causare un basso rendimento universitario, con conseguenti disagi dovuti alla sensazione di inadeguatezza.
Questo è uno dei motivi che ci ha spinti a creare Annual, un'applicazione per Android e iOS progettata per connettere studenti universitari in Italia con interessi simili attraverso annunci di gruppi studio. L'obiettivo principale di Annual è combattere la solitudine nel mondo accademico fornendo agli studenti uno spazio digitale dove possono facilmente trovare compagni di studio, formare gruppi e stringere nuove amicizie. Gli utenti possono creare annunci personalizzati per i loro obiettivi accademici, consentendo loro di trovare velocemente partner di studio compatibili. Grazie ad Annual, gli studenti universitari avranno la possibilità di rendere il loro percorso accademico più sociale, gratificante e meno isolante, promuovendo allo stesso tempo il successo universitario attraverso la condivisione delle conoscenze e la collaborazione.
Nel capitolo 1 si discutono le premesse, riportando alcuni studi. Nel capitolo 2 viene presentata l'applicazione da un punto di vista prettamente informale e nel terzo capitolo si va più nel dettaglio, analizzando e discutendo le tecnologie utilizzate. Infine, nel capitolo 4, si racconta brevemente il processo di conversione da file jar a container Docker e la

procedura di pubblicazione su Amazon AWS.

Ringraziamenti

Voglio ringraziare in primis la mia famiglia che, in un modo o nell'altro, mi ha sempre supportato (e sopportato) in ogni mia decisione. Voglio ringraziare i miei migliori amici, Marcello C., Simone V., Cristiano F., Andrea L., Alessandro L., Lorenzo S. e Riccardo P. per essere sempre stati con me, sia nei momenti belli che in quelli più cupi. Voglio ringraziare Caterina N. e Gianna R. perché, senza di loro, tutto questo non sarebbe stato possibile e il mio relatore, il professor Giovanni M., per aver accettato di seguirmi nella realizzazione di questi progetto.
Infine, voglio ringraziare me per aver reagito al meglio ad ogni sacrificio che ho dovuto fare per conseguire questo obiettivo e per averci sempre creduto nonostante le difficoltà.

Table of Contents

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Vicini ma lontani

1.1 La crescente solitudine

Negli ultimi anni, abbiamo assistito ad una crescita significativa del numero di individui che si trovano a fronteggiare il problema della solitudine. La presenza sempre più invadente di piattaforme social ha ingigantito e accentuato questo pericoloso trend, immergendo molti giovani (e adulti) in una realtà virtuale in cui, ad ogni ora del giorno, si è protagonisti di storie, post, like e messaggi. Il risultato di queste interazioni è un livello di benessere mentale in continua decrescita e la carenza di relazioni sociali, costruite sulla base di interazioni fisiche, ha causato un'impennata di casi di depressione e solitudine. Inoltre, la pandemia di COVID-19 ha ulteriormente marcato questo fenomeno, con numerose restrizioni in tutto il mondo che, da un lato, per contenere l'epidemia, hanno imposto ai cittadini di uscire dalle proprie abitazioni solo in casi sporadici, ma che, dall'altro, hanno incrementato l'isolamento, azzerando (o quasi) il contatto umano.
Nel corso di questo articolo, esploreremo i risvolti di questa crescita e l'influenza subita dal mondo accademico.

1.2 La ricerca delle cause

Con l'avvento delle piattaforme social, si è verificata una crescita del numero di individui che affermano di sentirsi socialmente e/o emotivamente soli. In effetti, è abbastanza frequente incontrare persone che, nonostante si trovino assieme, passano gran parte del loro tempo libero a interagire con lo smartphone, distanti dal mondo reale sebbene si trovino fisicamente nello stesso posto e nello stesso gruppo di amici. Nell'articolo 'Worldwide increases in adolescent loneliness', pubblicato da Jean M. Twenge, Jonathan Haidt, Andrew B. Blake, Cooper McAllister, Hannah Lemon, Astrid Le Roy nel Dicembre 2021, viene condotto uno studio sul livello di solitudine percepito nelle scuole su più di un milione di adolescenti provenienti da 37 paesi diversi, di età compresa tra i 15 e i 16 anni. L'esperimento consiste nel rispondere a 6 domande, ciascuna con un punteggio che va da 1 a 4. Il punteggio finale viene calcolato sommando i punteggi parziali e dividendo per 6. I risultati emersi indicano un forte aumento della sensazione di solitudine nelle scuole dal 2012 in poi (Figura 1.1). La causa di questo andamento non è chiara poiché ci sono molti fattori che determinano il benessere psichico e la capacità di socializzare: come anticipato poco fa, l'avvento dell'era digitale potrebbe essere stato motore di questo fenomeno. Per di più, crisi economiche mondiali (come quella del 2008), il livello di disparità salariale e quello di fertilità di un paese possono aver influito ugualmente sulla serenità degli individui appartenenti a diverse condizioni sociali ed economiche. Per questo motivo, sono stati studiati, e combinati, gli andamenti di diverse variabili (Figura 1.2). Dal 2012 (anno in cui si è registrato un'innalzamento del sentimento di solitudine) in poi si è verificata una forte crescita dell'utilizzo di Internet e dell'accesso a dispositivi mobili come gli smartphones. Al contrario, l'indice di fertilità totale è precipitato ma in quel periodo il livello generale di solitudine nelle scuole non era ancora preoccupante, suggerendo una bassa correlazione tra i 2 fenomeni. Infine, il livello di disoccupazione e la disparità salariale sono scesi mentre il GDP (Gross

Domestic Product) è cresciuto ben prima del 2012 (a partire dal 2003 per poi rallentare qualche anno prima del 2012), facendoci supporre che non abbiano avuto rilevanza nella crescita media della solitudine nelle scuole (Figura 1.1). Questo trend si è ripetuto in 36 dei 37 paesi in cui è stato condotto il test. Dalla Figura 1.3 si nota facilmente che questa crescita è globale: attorno all'anno 2012, con l'esplosione di Internet e degli smarthphones (e dei primi portali social), tutti i territori con aspetti culturali simili hanno registrato un'impennata di casi di solitudine nel mondo accademico (Figura 3). Ciò potrebbe farci supporre che il crescente utilizzo di dispositivi tecnologici sia una delle principali cause dell'aumento di casi di solitudine tra studenti.
Nel prossimo paragrafo, verranno esplorati i risultati di uno studio olandese in cui si analizzano le correlazioni tra il sentimento di solitudine, e più in generale di frustrazione, e il rendimento scolastico nel mondo universitario.

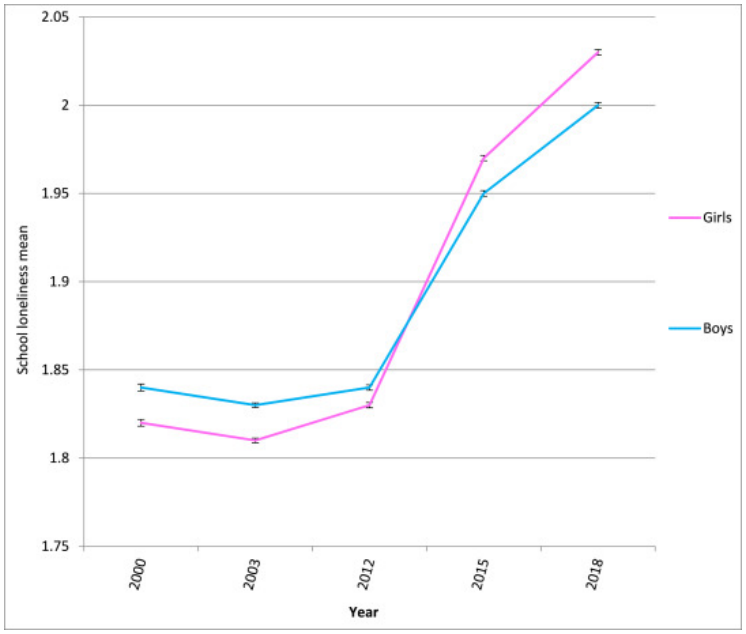

Figura 1.1: Livello medio di solitudine per gender tra il 2000 e il 2018

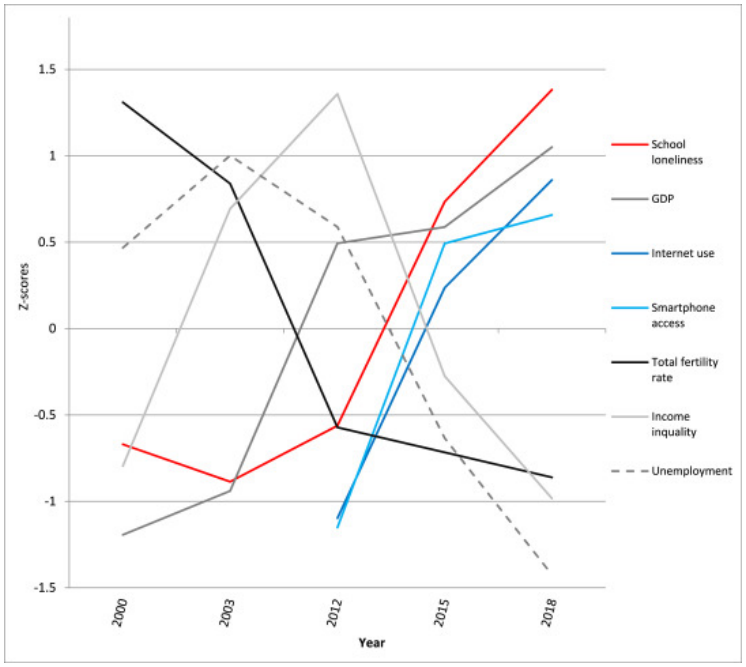

Figura 1.2: Scores dei diversi fattori presi in considerazione tra il 2000 e il 2018

1.3 Malessere e rendimento universitario

In una società in cui la tendenza all'isolamento sta diventando sempre più una realtà per molti di noi, è naturale chiedersi quali siano le conseguenze di tale comportamento. Gli esseri umani sono creature che, fin dall'alba dei tempi, vivono in gruppo e fanno affidamento su altri individui per cercare di emergere in ogni contesto. Quali sono quindi le conseguenze di questa forte lontananza che, come è stato spiegato nel paragrafo precedente, sta prendendo sempre più piede in tutto il mondo?
Nel 2020, l'epidemia di COVID-19 ha messo in ginocchio l'intero globo, colpendo non solo il settore medico, ma ferendo gravemente anche quello economico e psicologico. Infatti, a causa delle sempre più rigide

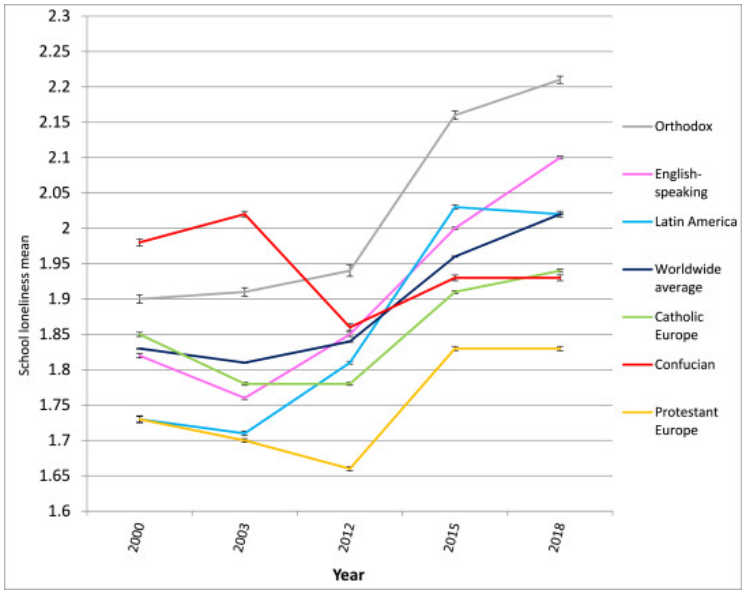

Figura 1.3: Andamento medio della solitudine in ambiente scolastico per regioni geografiche/culturali tra il 2000 e il 2018

restrizioni nate per contenere la piaga, i cittadini di tutto il mondo si sono ritrovati a vivere in una condizione di isolamento forzato che ha fatto crollare l'economia e aggravato il malessere psichico di molti individui. Uno dei tanti gruppi colpiti è stato sicuramente quello degli studenti. Soprattutto nelle fascie di età più basse, è fondamentale stimolare l'interazione con altri studenti: passare l'intervallo assieme, bere un caffè in compagnia davanti ai distributori, darsi appuntamento per attività extra scolastiche sono solo alcuni dei momenti che, prima del 2020, venivano dati per scontato da ognuno di noi ma che poi si sono rilevati fondamentali.
In 'Students' academic engagement during COVID-19 times: a mixed-methods study into relatedness and loneliness during the pandemic' Laura Hendrick, Marie-Christine Opdenakker e Wander Van der Vaart analizzano la correlazione tra il malessere generale degli studenti in Olanda durante la pandemia e il loro livello di impegno scolastico.
Gli autori hanno chiesto ad un gruppo di studenti, appartenenti ad

un'università olandese, di rispondere ad un questionario online relativo al loro grado di socialità, solitudine e impegno universitario. È stato chiesto loro di dare un punteggio a situazioni che suscitano socievolezza e solitudine, in ambito accademico e nella vita di tutti i giorni, così da comprendere meglio la differenza tra il sentirsi parte di un gruppo nel contesto accademico e non.
Gli autori fanno una distinzione tra solitudine emotiva e sociale: con la prima, si intende un sentimento di mancanza di legami forti con altre persone mentre con la seconda si indica una mancanza di integrazione sociale in una comunità o di appartenenza a gruppi di amici.
I risultati emersi da quest' analisi mettono in luce una forte correlazione positiva tra la sensazione di solitudine (emotiva e sociale) e la frustrazione relativa allo studio. La tabella 1.1 mostra la correlazione tra le 7 variabili prese in considerazione.

	1	2	3	4	5	6	7
(1)Acad. engagement	1						
(2)Satisf. related	0.242	1					
(3)Frustr. related life	-0.284	-0.735	1				
(4)Satisf. related study	0.206	0.268	-0.294	1			
(5)Frustr. related study	-0.292	-0.284	0.439	-0.516	1		
(6)Social loneliness	-0.187	-0.674	0.616	-0.240	0.368	1	
(7)Emotional loneliness	-0.316	-0.473	0.503	-0.312	0.470	0.477	1

Tabella 1.1: Correlazione di Pearson per le 7 variabili esaminate. Per una maggiore leggibilità, la tabella mostra solo i valori al di sotto della diagonale

Questo dato fa emergere un'informazione importante: il livello di affiliazione a gruppi o a singoli individui è indice del grado di produttività. Tanto più uno studente si sente parte di una cerchia di persone, per cui da e riceve gratificazione, maggiore è il suo rendimento scolastico.

Al contrario, uno studente emarginato ha una resa più bassa e fa più fatica a mantenere un buon livello di efficienza.

1.4 Il mondo post COVID-19

Nel paragrafo precedente è stato discusso come la pandemia del 2020 ha, direttamente e indirettamente, scosso le fondamente della nostra società, dimostrando una crescita proporzionale tra il grado di soddisfazione sociale e la resa accademica. Purtroppo, le restrizioni dovute alla pandemia, hanno lasciato degli strascichi anche dopo essere state tolte. Nell'articolo 'Tadros O K, Arabiyat S, Jaber D, et al. (August 23, 2023) The Impact of the COVID-19 Pandemic on Loneliness Among University Students: A Cross-Sectional Study in Jordan', pubblicato in Agosto 2023, gli autori eseguono uno studio su un campione di studenti provenienti da un'università in Giordania, rappresentativo della popolazione giordana. Ciò che è emerso è che, sebbene le restrizioni globali siano ormai cessate, il sentimento di isolamento è ancora forte nel mondo scolastico. Il test consiste nel 'University of California, Los Angeles (UCLA) three-item loneliness scale', un questionario basato su 3 domande a cui il tester deve dare un punteggio da 1 a 3. Se la somma dei 3 punteggi parziali è maggiore o uguale a 6, l'individuo si considera "solo" altrimenti "non solo".
Di seguito sono riportate le percentuali delle risposte per ognuna delle 3 domande e il risultato finale del test.

Si nota come in tutte e 3 le domande quasi la metà degli studenti intervistati si senta spesso privo di compagnia, escluso e/o isolato dagli altri. Nella figura 1.5 emerge un dato allarmante: secondo il test riportato, il 48.8% degli studenti ha totalizzato un puteggio compreso tra 3 e 5, il che significa che quasi la metà dei partecipanti si sente solo. Incrociando questo dato con ciò che è emerso dalla tabella 1.1, si può ipotizzare che, anche al termine della situazione di emergenza globale del 2020, gli studenti non siano mai tornati alla realtà pre pandemia,

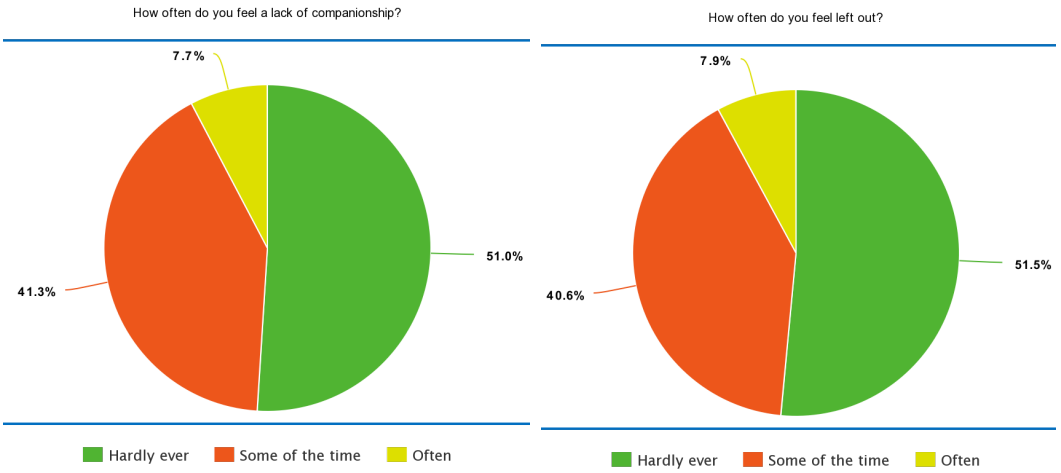

Figura 1.4: Percezione della mancanza di compagnia da parte di studenti universitari

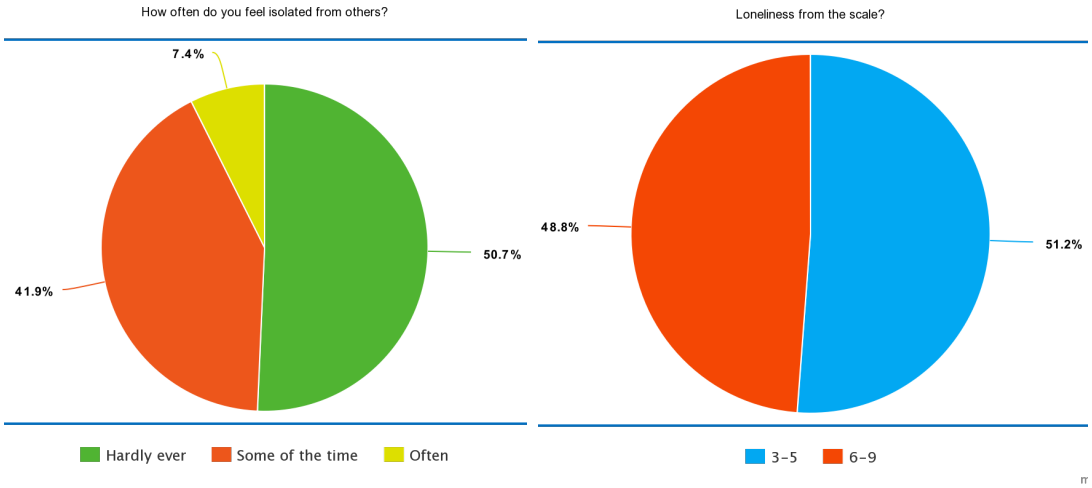

Figura 1.5: Percezione di isolamento da parte di studenti universitari

con conseguenti ripercussioni negative sul loro andamento accademico. Il bisogno di stringere legami e il successo universitario sono i 2 capi saldi sui cui si basa Annual. Nel capitolo successivo, verranno discussi i fondamenti dell'applicazione in questione, nonché le tecnologie usate e le varie implementazioni.

Capitolo 2

La nostra soluzione

2.1 Perché l'ennesimo social?

Nel capitolo precedente, si è visto come il grado medio di solitudine sia stato in rialzo dal 2012 fino al 2018 e come questo influisca negativamente sul rendimento didattico. Inoltre, questa tendenza si è aggravata nel 2020, a causa dei divieti imposti, e non si è mai invertita, mostrando ancora oggi livelli di solitudine tra studenti molto alti. Se però parte della colpa va all'utilizzo sempre più frequente e invadente di dispositivi mobili, perché l'ennesimo social dovrebbe aiutare a invertire questa tendenza?
Annual è fondamentalmente diverso dai social più popolari: in prima battuta, studenti con difficoltà in comune vengono messi in contatto attraverso l'app e, in un momento successivo, potranno organizzarsi per vedersi dal vivo ed eventualmente approfondire la conoscenza. Generalmente, anche le piattaforme social più usate permettono di ampliare la propria rete sociale ma spesso questa resta su un piano virtuale, priva di qualsiasi contatto con il mondo fisico. Con Annual, studenti iscritti a diverse università italiane possono affrontare il percorso accademico più serenamente, ampliando la loro rete di conoscenze, con conseguente successo universitario.

2.2 Home: la sezione "Annunci"

Annual è un'applicazione per Android e iOS che permette, agli utenti bisognosi di approfondire certi argomenti, di creare e partecipare a gruppi studio pubblici o privati. Ciascun annuncio è caraterizzato dal nome della materia per cui ci si vuole preparare e da un messaggio lasciato dal creatore che specifica nel dettaglio quale argomento non gli è chiaro. Inoltre, viene specificato un numero massimo di partecipanti ammissibili al gruppo, la data e l'ora dell'incontro, il tipo di incontro (online o dal vivo), il livello di visibilità (visibile a tutti o solo per la cerchia di amici) e alcune informazioni segrete. Queste ultime denotano tutto ciò che deve essere conosciuto dai membri ammessi al gruppo, come il luogo dell'incontro (fisico o virtuale, nel caso si tratti di un incontro online), materiale da portare e chi ne ha più ne metta.
Dopo la pubblicazione dell'annuncio, esso sarà visibile nella sezione "Nella tua università" oppure in "Nel tuo annual" (con "annual" si intende la lista di tutti gli utenti che rispettano un certo stato di amicizia) e gli studenti potranno richiedere di essere ammessi, a patto che il numero massimo di utenti ammissibili non sia stato raggiunto. L'annuncio resta visibile nella Home fintanto che lo studente che l'ha pubblicato non decida di renderlo ufficiale. A quel punto, solo gli studenti ammessi al gruppo potranno visualizzarne le informazioni segrete e un nuovo impegno verrà aggiunto alla loro "lista degli impegni" (vedere sezione 'Il profilo personale'). Poiché altri studenti esterni al gruppo non potranno più accedervi (a meno che l'amministratore non voglia "riaprire le candidature"), l'annuncio verrà automaticamente rimosso dalla Home. Nella figura 2.1 viene riportato uno screenshot della sezione "Annunci" della Home e del form per la creazione e pubblicazione di essi.

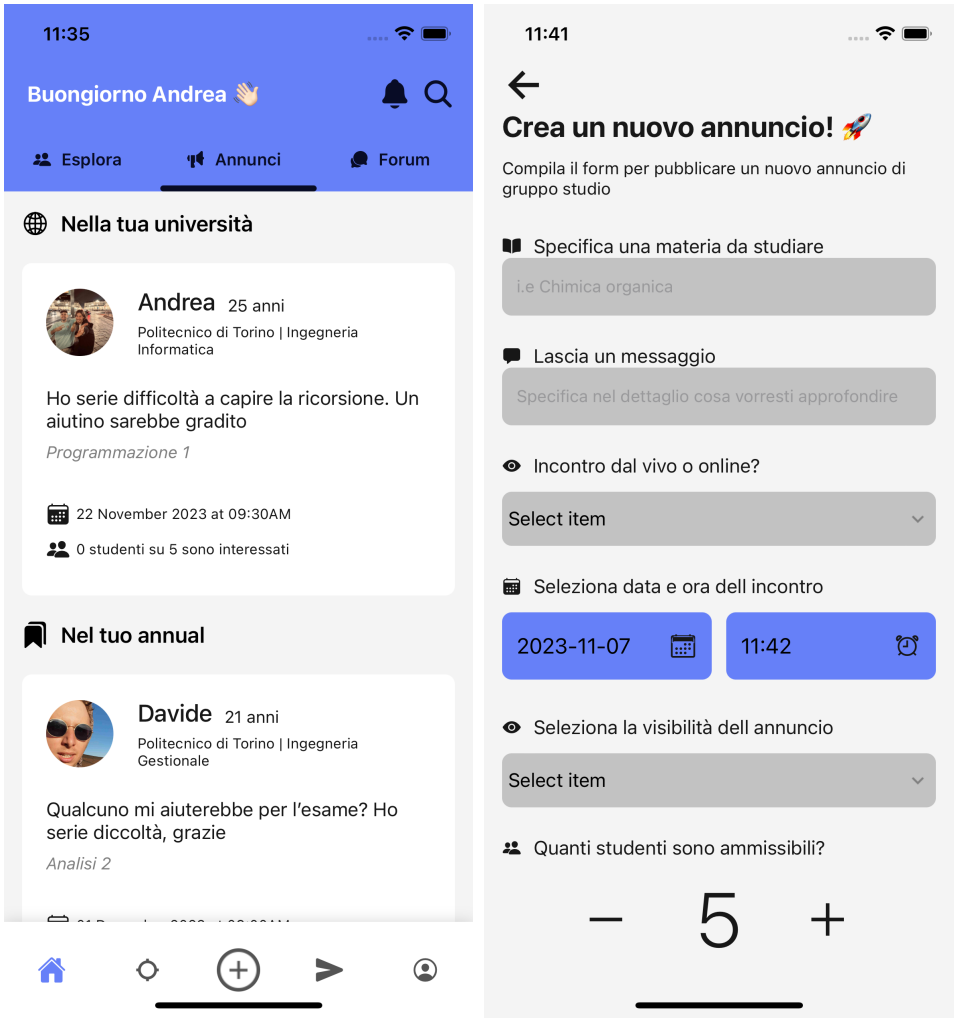

Figura 2.1: A sinistra, screenshot della schermata Home, sezione "Annunci". A destra, screenshot del form per la creazione e pubblicazione di annunci di gruppo studio

2.3 Home: la sezione "Forum"

Accanto alla sezione degli annunci, è possibile accedere al Forum. In questa schermata, l'utente potrà interagire con domande e risposte rilasciate da studenti appartenenti alla propria università. Domande e risposte possono essere pubblicate con nome e cognome visibile oppure

in forma anonima. In questo caso, a meno che non si è l'utente creatore, non sarà possibile visionare il profilo dello studente in questione (la cui identità viene censurata) e la domanda (o risposta) non sarà visibile nella scheda "Forum" nel profilo dell' utente.
Inoltre, al momento della creazione della domanda, lo studente potrà aggiungerci una o più categorie. Ad esempio, nel caso in cui si volesse chiedere informazioni riguardanti un corso in particolare, sarà possibile etichettare la domanda in questione con la categoria "Corsi di studio". Questo consente, in fase di ricerca, di filtrare i risultati per categoria, così da cercare, in maniera più mirata, domande a cui si vuole rispondere o per cui si è interessati alle risposte. In futuro, sarà possibile applicare direttamente questo filtro con un menù a tendina posto in alto a destra sulla schermata. La lista delle categorie comprende "Esami", "Corsi di studio", "Affitti" e "Curiosità".
È possibile interagire con domande e risposte con un up-vote o down-vote. Solo la somma totale di up-votes e down-votes sarà visibile, ma non la lista di utenti che hanno espresso il voto. Sulla card della domanda, in basso a sinistra, viene visualizzata la preview di una lista di utenti che hanno risposto alla domanda, sotto forma di avatar. Chiaramente, nel caso in cui uno o più studenti abbiano risposto in forma anonima, la loro foto profilo verrà censurata con un avatar di default. Alla fine di questa lista, è presente un'icona "+". Alla pressione del pulsante, un piccolo form compare nella parte bassa dello schermo, offrendo uno spazio per digitare la risposta e un pulsante per attivare o disattivare la modalità anonima.
Per mantenere un'esperienza fluida, è stato implementato un sistema automatico di paginazione: vengono scaricate 10 domande alla volta e quando l'utente sta per raggiungere la fine della lista, vengono scaricate altre 10 domande, a patto che vi siano contenuti da scaricare. Nel caso in cui si volesse aggiornare il contenuto della pagina, è stato implementato un comune sistema "pull to refresh": se l'applicazione dovesse trovare nuove domande, queste verrebbero scaricate e aggiute all'inizio della lista, essendo ordinate per timestamp di pubblicazione

decrescente (dalla più recente alla più vecchia).
Nella figura 2.2 vengono mostrati due screenshots, a sinistra la sezione "Forum" e a destra il form per la creazione e pubblicazione di una domanda.

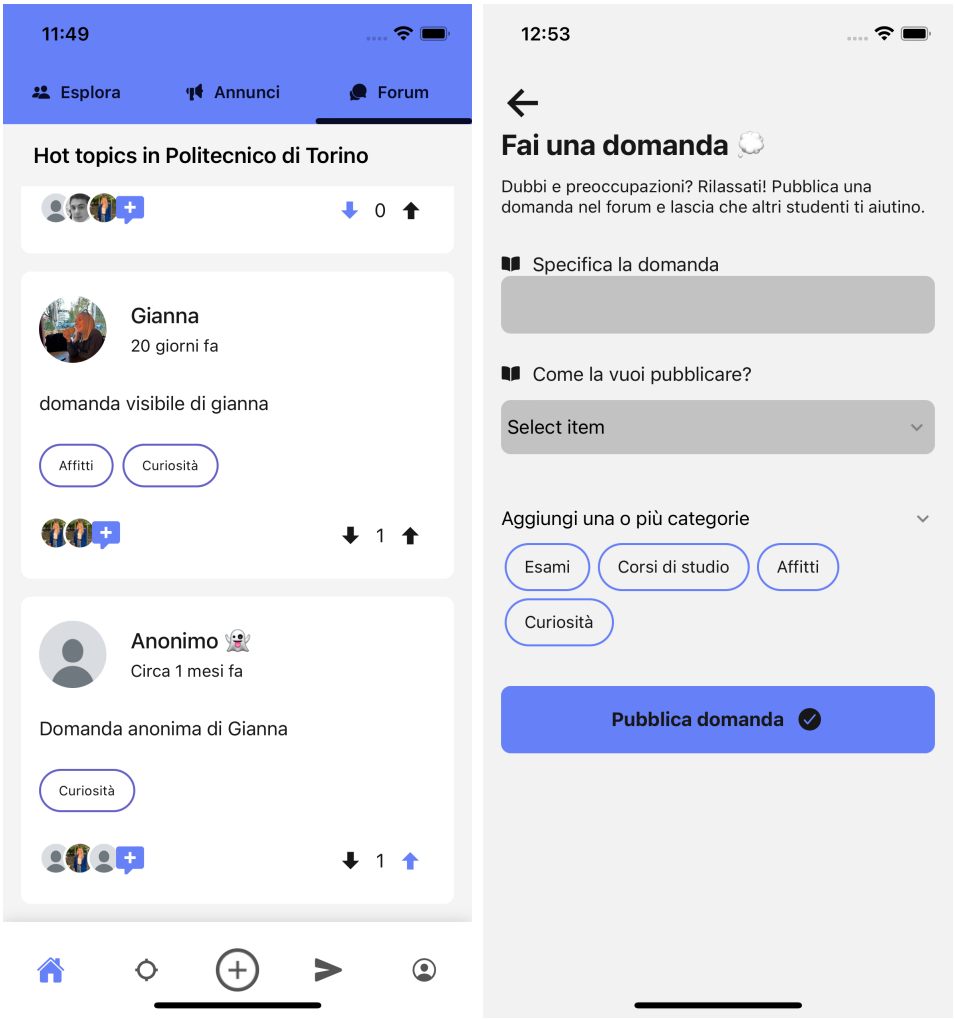

Figura 2.2: A sinistra, lista di domande del forum. A destra, form di creazione e pubblicazione della domanda

2.4 Home: la sezione "Esplora"

L'ultima scheda a cui si può accedere dalla schermata "Home" è la sezione "Esplora" (2.3). In questa schermata, è possibile visionare tutti gli studenti facenti parte della propria università, organizzati in 2 liste: la prima raccoglie tutti gli studenti iscritti allo stesso corso di studi, la seconda tutti quelli appartenenti alla stessa università. Se si volesse visualizzare il profilo di un certo studente, basta premere sulla relativa card. Ciascuna card è caratterizzata da foto profilo, foto di background, nome e cognome e da una preview dei dettagli dell'utente: si può infatti visualizzare la prima riga della biografia (nel caso in cui sia stata inserita) e una lista scorrevole che riassume alcune informazioni dello studente come corso di studi, università di appartenenza e età.

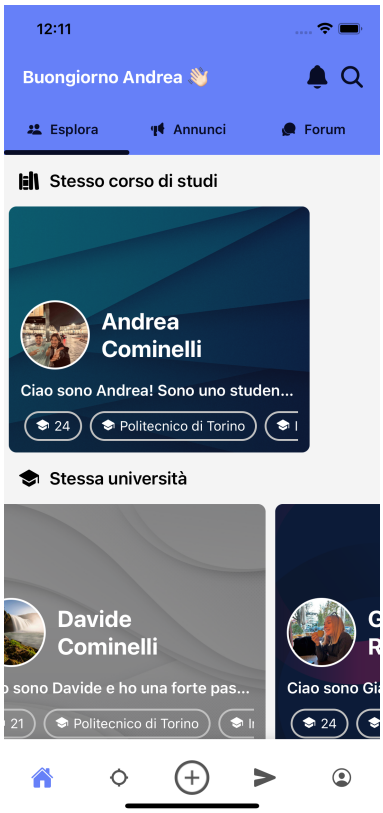

Figura 2.3: Schermata "Esplora"

2.5 Il profilo personale

Attraverso la schermata del profilo personale, lo studente potrà modificare le proprie informazione personali, visuallizzare la lista dei suoi annunci e dei suoi impegni, le domande pubblicate nel forum e i dettagli del proprio account, come l'indirizzo email associato, l'università di appartenenza, il corso di studi a cui è iscritto e altro. Il layout della vista è organizzato principalmente in due porzioni: nella prima metà, viene mostrata in primo piano la foto profilo, posta davanti alla foto di background, con sotto il nome e il cognome. Vicino alla status bar in alto, si trovano la classica icona della freccia per tornare indietro e l'hamburger icon per accedere alle impostazioni dell'account.
Premendo il pulsante "Aggiorna profilo" si avrà accesso ad un form per modificare le informazioni principali del profilo. Potranno essere aggiornate la foto profilo, la foto di background, la materia preferita, e la bio. I dati anagrafici dello studente (nome, cognome e età) non saranno aggiornabili.
Nella metà più bassa dello schermo, possiamo interagire con quattro sottomenù: da sinistra verso destra "Su di me", "Annunci", "Forum", "I miei impegni". La schermata "Su di me" raccoglie le principali caratteristiche dell'utente, come la bio, l'università, il corso a cui è iscritto, l'età e la materia preferita. Se alcune informazioni non fossero presenti, il layout della schermata si adatta di conseguenza, mostrando solo ciò che l'utente ha scelto di specificare.
Nella schermata "Annunci" viene visualizzata una lista degli annunci pubblicati dall'utente. Questi potranno essere filtrati a seconda che siano ancora pubblici (e quindi visibili nella Home) o ufficiali. Premendo sulla relativa card si apre il pannello di controllo dell'annuncio, una scheda che ne elenca le caratteristiche e gli studenti ammessi o ammissibili (a seconda dello stato del suo stato). Da questa schermata, l'utente amministratore può scegliere di modificarne le informazioni, di rimuoverlo, di rimetterlo pubblico o di abbandonarlo, o di promuovere uno studente ammesso ad amministratore.

Premendo il pulsante "Forum", si potrà interagire con le proprie domande pubblicate nel forum ed eventualmente eliminarle. Si possono leggere anche le diverse risposte rilasciate, pubblicarne di nuove e mettere up o down votes.
L'ultima delle quattro sotto sezioni raccoglie tutti gli annunci di gruppo studio a cui l'utente è stato ammesso. Gli annunci vengono ordinati in ordine crescente di data e ora dell'incontro (dal più vicino al più lontano) come una sorta di promemoria. Premendo su una card, si potranno visualizzare i dettagli dell'incontro a cui si è stati ammessi e gli altri studenti del gruppo. È possibile abbandonare il gruppo in qualsiasi momento. A quel punto, l'annuncio di gruppo studio verrà definitivamente rimosso dai propri impegni e l'amministratore verrà notificato di tale cambiamento.

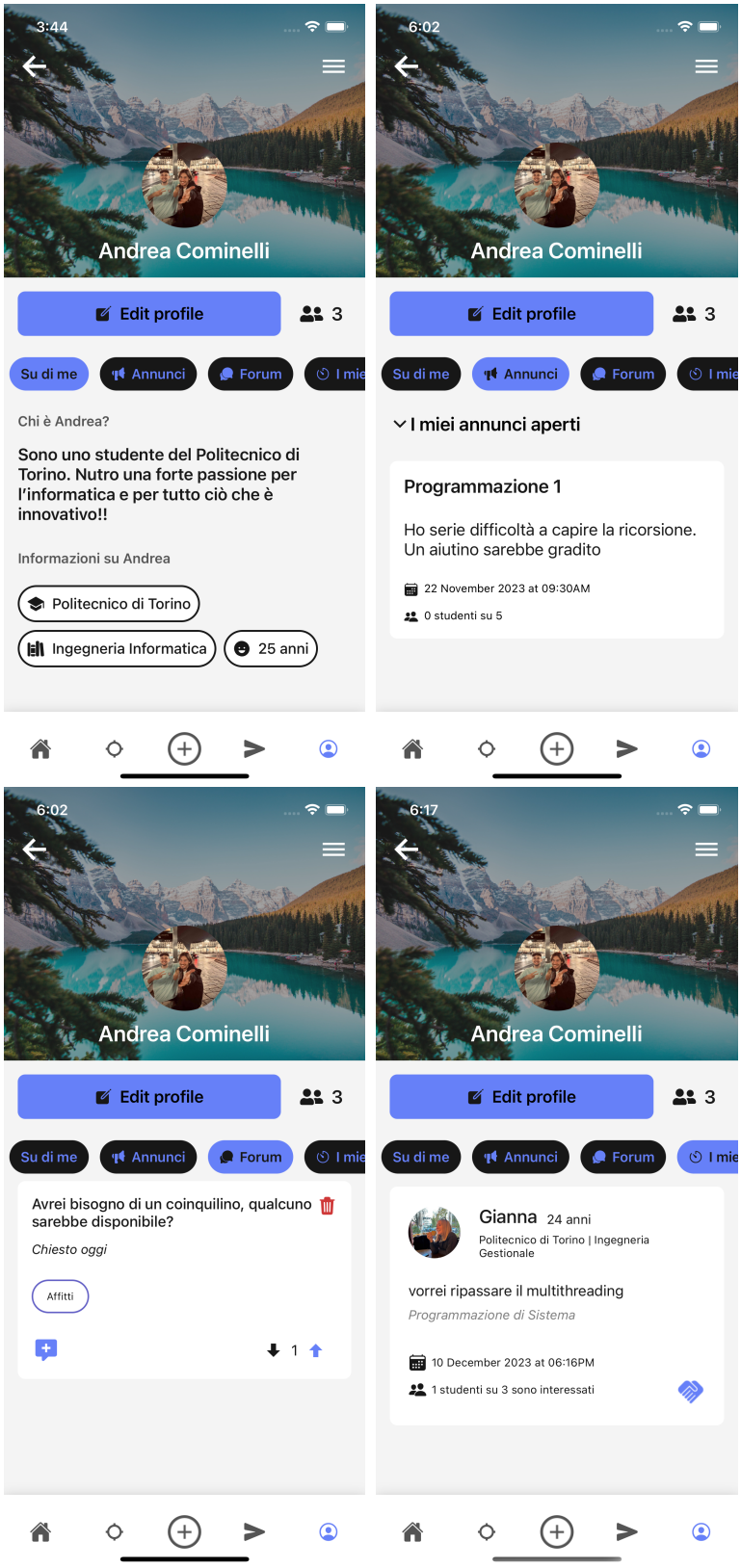

Figura 2.4: Lista di domande pubblicate e promemoria degli impegni presi

Capitolo 3

Le fasi di progettazione

3.1 Stesura dei requisiti e prototipizzazione

A partire dai requisiti raccolti, si è concentrato il lavoro sull'organizzazione delle pagine e sono stati disegnati diversi prototipi usando il software Figma, applicazione che facilita il processo di design e prototipizzazione. Questo ci ha consentito di scegliere, in maniera mirata, un set di API endpoint che ricoprisse le diverse funzionalità offerte dall'applicazione. Ad esempio, avendo disegnato la home affinché mostri gli annunci di gruppo studio di studenti appartenenti alla stessa università dell'utente loggato, sarebbe utile un endpoint che, dato l'identificatore dell'università e il numero di pagina, recuperi tutti gli annunci pubblici sotto forma di materia, messaggio, data e ora dell'incontro, numero massimo di studenti ammissibili e numero di studenti che hanno mostrato interesse. In poche parole, incrociando i requisiti raccolti con i prototipi disegnati, è stato poi più facile identificare le varie funzionalità da implementare nel backend e, per ciascuna di esse, le varie pre e post condizioni.
Durante la fase di prototipizzazione, ci siamo concentrati sul layout di

ciascuna pagina, senza prestare attenzione alla palette di colori. Questa configurazione in bianco e nero ci rende abbastanza sicuri che la user experience e la disposizione degli elementi siano corretti, poiché non ci sono componenti che spiccano in modo più evidente rispetto agli altri, rischiando di influenzare l'orientamento dell'utente.
Di seguito viene riportato il diagramma di flusso dell'esperienza utente.

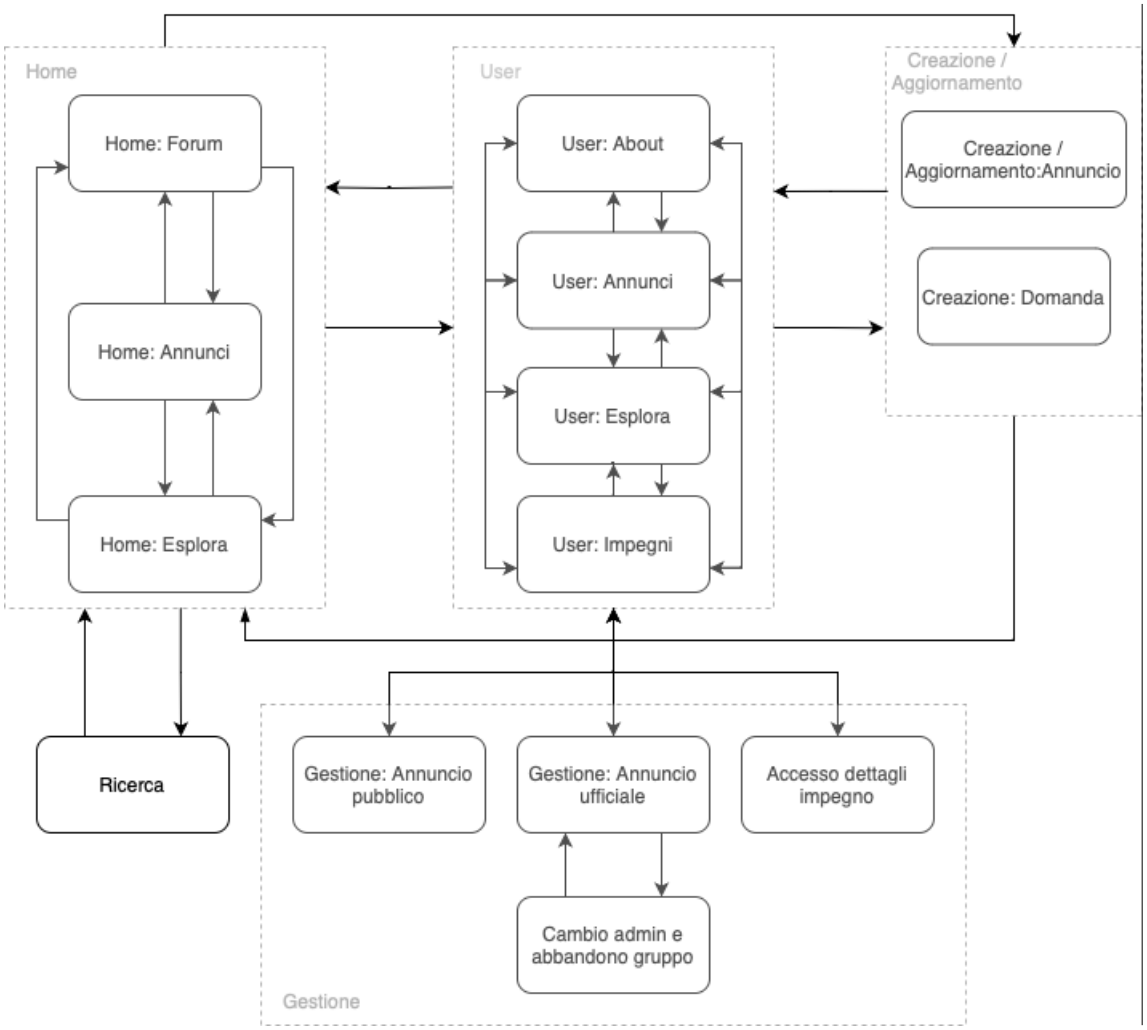

Figura 3.1: Diagramma di flusso dell'esperienza utente

Il diagramma riportato (3.1) mostra i nomi delle principali schermate (organizzate per categoria) e il flusso di navigazione che concatena

le diverse schede. Assumendo che l'utente abbia passato la fase di autenticazione, egli verrà reindirizzato alla schermata principale: la Home. Come discusso in precedenza, la Home è costituita da 3 schermate (Annunci, Forum e Esplora) ciascuna raggiungibile dalle altre due. Dalla home, l'utente può navigare verso la schermata di ricerca (che permette di trovare studenti, annunci e/o domande attraverso parole chiave e filtri) verso il menù di creazione e pubblicazione di un annuncio e verso la scheda del proprio profilo. Proprio da qui, lo studente avrà accesso ad una serie di schermate atte a garantire la completa gestione degli annunci pubblicati e dei gruppi studio a cui è stato ammesso.
Tra le schermate non citate, c'è quella che permette di cedere, per un certo annuncio ufficiale, l'autorità di amministratore ad uno studente ammesso. In questo modo, è possibile sbarazzarsi del gruppo senza doverlo eliminare, così da non arrecare danno agli altri studenti del gruppo che potrebbero già essersi organizzati.
Dopo aver stabilito il flusso di navigazione tra le principali schede, sono stati elaborati diversi prototipi (`https://www.figma.com/file/iyFbQ6KTEcrNwvBcH1Zrvu/Annual-2?type=design&node-id=0%3A1&mode=design&t=SEiRsLlzviNf1KkT-1`). Il layout finale è abbastanza diverso da quello inizialmente immaginato, per certi versi più semplice e con meno elementi a schermo. Infatti, è piuttosto comune modificare diverse volte il design originale prima di trovare quello ottimale.
Le pagine successive andranno a descrivere più nel dettaglio le tecnologie usate e l'architettura disegnata.

3.2 La scelta dei tools

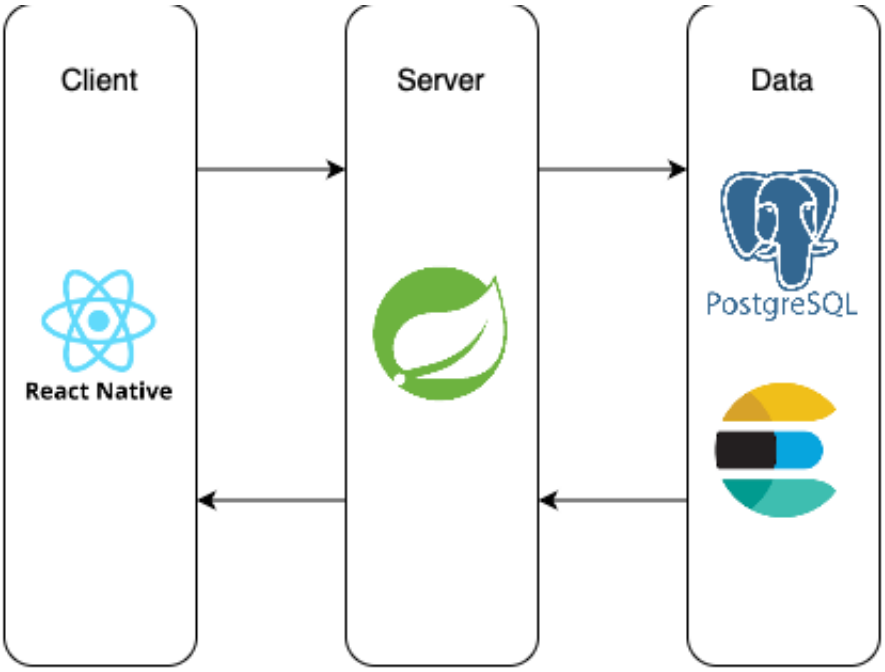

Figura 3.2: Diagramma di flusso dell'esperienza utente

A grandi linee, l'architettura è composta da 3 layers indipendenti tra loro: il client layer, il server layer e il data layer.
Il client layer definisce l'aspetto dell'applicazione e il set di API. Abbiamo scelto di implementarlo sfruttando React Native, un framework Javascript che permette di definire componenti riutilizzabili all'interno dell'applicazione e, al momento di compilazione, di tradurli per la piattaforma target desiderata. In questo modo, non è necessario conoscere i concetti e le librerie di Android e iOS, rendendo lo sviluppo molto più semplice e veloce.
Tutta la business logic è racchiusa nel server layer e viene gestita da Spring. Spring è un framework open source che mette a disposizione degli sviluppatori una marea di librerie per lo sviluppo di applicazioni su piattaforma Java. Sfruttando pesantemente l'uso di annotazioni, consente di scrivere codice in maniera piuttosto intuitiva, liberando

lo sviluppatore dal peso di progettare meccanismi complessi da zero (diminuendo il rischio di incappare in bachi) e semplificando la scrittura di unit e integration tests.
Il server layer scambia informazioni con il data layer attraverso componenti denominate "Repositories". Il data layer si occupa di gestire il recupero, l'inserimento, l'aggiornamento e la rimozione di informazioni, memorizzate in un apposito database. Per questo progetto, si è scelto di usare 2 sistemi di storage: alcune informazioni, come i dettagli dell'account, vengono memorizzate in un'istanza di PostgreSQL, un DBMS relazionale open source. Altri dati vengono gestiti da Elasticsearch, un database NoSQL nato per la ricerca testuale.

3.3 AuthenticationProvider: Come funziona l'aggiornamento delle componenti?

Come anticipato nel paragrafo precedente, per realizzare il client è stato usato React Native, un framework open source creato da Meta che semplifica notevolemente lo sviluppo front-end per diverse piattaforme, tra cui Android e iOS. React Native si basa su React, una libreria Javascript che permette di costruire interfaccie grafiche sfruttando la 'Component-based software engineering', una tecnica che consiste nello scrivere software composto da molteplici componenti debolmente connessi tra loro, garantendone il riuso ed un testing più semplice.
React gestisce automaticamente ed efficientemente l'aggiornamento di tali componenti registrando in memoria un'apposita struttura dati ad albero, dove ciascun nodo identifica un componente e ciascun arco la relazione tra il componente padre e il componente figlio. Durante l'avvio, React inizializza tutte i moduli e i loro stati (render) e nel momento in cui uno di essi cambia (per esempio è arrivata la risposta ad una chiamata remota e l'interfaccia deve aggiornarsi per mostrare

il risultato) viene lanciato un processo ricorsivo atto ad aggiornare solamente il componente il cui stato è cambiato e tutti i relativi figli che dipendono dallo stato stesso. In questo modo, solo una minima porzione della struttura dati deve aggiornarsi, garantendo un'efficiente gestione delle risorse computazionali.
Il sistema di autenticazione è un esempio perfetto di questo meccanismo. L'AuthenticationProvider è il componente che gestisce lo stato di autenticazione dell'utente. Nel caso in cui, al primo render, trovasse in memoria un token di autenticazione, proverebbe a contattare il server per scaricare, e memorizzare, le informazioni dell'utente loggato. Se il token dovesse essere ancora valido, la richiesta andrebbe a buon fine e il componente cambierebbe il suo stato "authenticated" da "false" a "true". A questo punto, scatterebbe un re-render del componente, il quale monterebbe l' HomeStack (lo stack navigator composto da tutte le schermate raggiungibili dopo aver passato lo step di autenticazione) e smonterebbe l'AuthenticationStack (lo stack navigator che controlla il flusso di tutte le schermate adibite alla registrazione di un nuovo account e della login). Se il token non dovesse essere valido, il componente eseguirebbe automaticamente un logout, cancellando il token e settando lo stato "authenticated" a "false". Infine se il token non dovesse essere presente (l'utente ha appena scaricato l'applicazione oppure è stato eseguito un logout) l'AuthenticationProvider non eseguirà mai la richiesta e rimarrebbe montato solo l'AuthenticationStack.

3.4 Una soluzione comune per la gestione delle richieste a API

Gestire e configurare un set di API endpoints può risultare complesso: gli endpoints possono essere parecchi, alcuni necessitano di un token di autenticazione e le risposte che giungono al client possono avere formati diversi.
Per semplificare questo processo, lo sviluppatore può estrarre blocchi

di logica comune e accorparli in un'unica funzione da aggiungere ad un componente piuttosto che ad un altro. Queste funzioni, nel contesto di React, vengono chiamate "custom hooks".

Listing 3.1: Custom hook per il fetch delle informazioni

```
import { useEffect, useState } from 'react'
import axiosInstance from '../../API/interceptor'
import {EXPO_PUBLIC_ANNUAL_HOST_IP} from "@env"

const useFetchData = (url, method = "GET", body, loading = true) => {

    if (url == "")
        throw Error("You must provide a valid URL")

    const [data, setData] = useState(null)
    const [apiCallError, setApiCallError] = useState("")

    useEffect(() => {
        if (loading) {
            axiosInstance({
                baseURL: `http://${EXPO_PUBLIC_ANNUAL_HOST_IP}:9000`,
                method: method,
                url: url,
                body: body ? body : null
            })
                .then(response => {
                    const pageDetails = {
                        pageNumber: response.data?.pageable?.pageNumber,
                        totalNumberPages: response.data?.totalPages
                    }

                    let data = response.data?.content || response.data
                    if (data instanceof Array)
                        data = data.filter(elem => elem != null)

                    setData(() => { return { data, ...pageDetails } })
                })
                .catch(error => {
                    const message = {code: error.response?.status,message: error.
    message}
                    setApiCallError(() => JSON.stringify(message))
                })
        }

    }, [loading, url])

    return [
        data,
        apiCallError,
        setData,
        setApiCallError
    ]
}

export default useFetchData
```

Viene riportato il codice del custom hook "useFetchData" (3.1), usato per interagire con il server in ascolto sulla porta 9000 all'indirizzo "EXPO_PUBLIC_ANNUAL_HOST", una variabile d'ambiente che

viene modificata in base all'indirizzo IP dell'istanza del server (in un ambiente di testing sarà "localhost" o 127.0.0.1).
Il parametro "loading" fa riferimento ad uno stato esterno, il cui cambiamento innesca l'esecuzione del corpo della useEffect (a patto che loading sia uguale a true). A questo punto, un'istanza axios esegue una richiesta verso l'endpoint specificato dal parametro url e ritorna una Promise, un oggetto che rappresenta l'eventaule successo, o fallimento, di un'operazione asincrona, consentendo alla axiosInstance di ritornare immediatamente senza bloccare l'esecuzione del codice.
A seconda della risposta ricevuta dal client, verrà invocata una certa callback, "then" o "catch". Nel caso di esito positivo (then), il contenuto della risposta viene estratto e memorizzato in uno stato interno al custom hook attraverso l'invocazione di setData(). In situazioni avverse, una stringa in formato JSON contenente codice e messaggio di errore viene memorizzata nello stato apiCallError. Infine, la funzione restituisce un array contenente la risposta, il messaggio di errore e le funzioni "set" corrispondenti, consentendo al componente chiamante di avere un controllo completo sulla risposta inviata dal server.

3.5 Intercettare le richieste e le risposte: Axios interceptor

Annual utilizza lo scambio di JSON Web Token (JWT) per autenticare le richieste provenienti da utenti diversi. Un JWT è un token di autenticazione codificato in Base64 e composto da tre parti separate da un punto: l'header, il payload e la firma. L'header specifica l'algoritmo utilizzato per calcolare la firma (ad esempio, HMAC512) e il tipo di token (ad esempio, JWT). Il payload raccoglie le informazioni principali del proprietario del token, come l'identificatore, il set di autorizzazioni e, per ragioni di sicurezza, il timestamp di massima validità del token. In questo modo, se il token venisse intercettato, diventerebbe invalido dopo il raggiungimento del timestamp specificato, impedendo l'accesso

all'utente non autorizzato. Infine, la firma impedisce la manipolazione del token e viene calcolata alimentando l'algoritmo specificato nell'header con la stringa ottenuta dalla concatenazione dell'header codificato, del payload codificato e di una chiave segreta custodita nel server e mai condivisa all'esterno.
Per inviare e memorizzare i token, Annual sfrutta un Axios Interceptor, un meccanismo in grado di intercettare richieste e risposte da e verso una certa Axios Instance.

Listing 3.2: Request interceptor

```
1  import axios from "axios";
2  import * as SecureStore from 'expo-secure-store';
3  import {EXPO_PUBLIC_ANNUAL_HOST_IP} from "@env"
4
5  let lastRequestConfig = []
6
7  const authenticatedURLs = [
8      "/api/v1/announce",
9      "/api/v1/forum",
10     "/api/v1/user",
11     "/api/v1/account/refresh",
12     "/api/v1/friends",
13     "/api/v1/university",
14     "/api/v1/search"
15 ]
16
17 const urlNeedsAuthentication = (url) => {
18     return authenticatedURLs
19     .filter(baseUrl => url.includes(baseUrl))
20     .length > 0
21 }
22
23 const axiosInstance = axios.create()
24
25 axiosInstance.interceptors.request.use(async config => {
26     const path = config.url.replace(/\?.*/, "")
27     lastRequestConfig[path] = config
28     if(urlNeedsAuthentication(config.url)){
29         let token = null
30         try{
31             token = await SecureStore.getItemAsync("access")
32             config.headers.Authorization = `Bearer ${token}`
33         }catch(error){
34             throw Error("Access jwt not found")
35         }
36     }
37     return config
38 }, error => {
39     throw error
40 })
```

Viene mostrato, nel frammento di codice 3.3, la logica del request interceptor (da riga 25 in poi). Il metodo "use" accetta una funzione asincrona che permette di accedere e modificare il parametro "config",

un oggetto contenente le varie proprietà della richiesta, come l'header e il corpo, prima che giunga al server. Per prima cosa, viene salvato nell'array "lastRequestConfig" il config stesso. Questo consentirà ad Annual di rieseguire la richiesta nel caso in cui il server dovesse rispondere con codice 401 (Unauthorized). Dopodiché, verifica che la richiesta sia da autenticare, cercando l'url di destinazione nella lista "authenticatedURLs". In tal caso, con l'aiuto della libreria "SecureStore" di Expo, recupera il token da una porzione di memoria di storage cifrata e lo aggiunge alla richiesta, modificando il campo Authorization con "config.headers.Authorization = 'Bearer $token'" (linea 32). Infine, indipendentemente dal fatto che il token venga incluso o meno, l'oggetto config viene ritornato, consentendo alla richiesta di proseguire verso il servizio remoto.
Precedentemente, è stata discussa la struttura di un comune JWT. Tra le varie proprietà, per ragioni di sicurezza, vi è un timestamp di massima validità che indica la durata di vita massima del token. Cosa succederebbe se l'utente dovesse continuare a mandare richieste dopo lo scadere della validità del token? Il server risponderebbe con uno status code 401, indicando al client che dovrebbe autenticarsi di nuovo. Chiaramente, per motivi di user experience, è impensabile chiedere all'utente di ri-eseguire il processo di login ogni volta che il token memorizzato risulti invalido. Per ovviare a questa problematica, è stato implementato nel server un sistema di refresh dei token di accesso: superata la fase di autenticazione, il server genera e firma due token, uno di accesso, valido per un ora, e uno di refresh, valido per 24 ore. Allo scadere del token di accesso, il client deve eseguire un refresh dei token inviando, ad uno specifico endpoint, il refresh token. In caso di successo, il server risponde con un nuovo token di accesso e un nuovo token di refresh.
Con questa strategia, se il token di accesso dovesse scadere, il client riceverebbe un nuovo set di JWT. Chiaramente, se anche il token di refresh non dovesse essere più valido (l'utente non interagisce con l'applicazione per almeno 24 ore), l'utente è costretto ad eseguire nuovamente

il processo di login.
In 3.3 viene mostrato il funzionamento di questo sistema.

Listing 3.3: Response interceptor

```
axiosInstance.interceptors.response.use(async config => {
    lastRequestConfig = []
    return config
}, async error => {
    if(error.response){
        // richiesta eseguita ma il server ha risposto con un codice errore
        if(error.response.status == 401){
            try{
                const refreshResponse = await axiosRefreshTokensInstance.post()
                if(refreshResponse.status == 200){
                    const body = refreshResponse.data
                    await SecureStore.setItemAsync("access", body.accessJwt)
                    await SecureStore.setItemAsync("refresh", body.refreshJwt)

                    // ri-eseguo la richiesta andata male
                    const request = lastRequestConfig[error.response.data.path]
                    return await axiosInstance.request(request)
                }

                if(refreshResponse.status == 401){
                    // rilancio il 401 al chiamante -> così sa che deve settare
    authenticated a false
                    throw error
                }

            }catch(exception){
                throw exception
            }
        }
    }
    throw error
})

const axiosRefreshTokensInstance = axios.create({
    baseURL: `http://${EXPO_PUBLIC_ANNUAL_HOST_IP}:9000/api/v1/account/refresh`,
    withCredentials: true,
    headers:{
        "Content-Type": "application-json"
    }
})

axiosRefreshTokensInstance.interceptors.request.use(async config => {
    const refreshToken = await SecureStore.getItemAsync("refresh")
    config.headers.Authorization = `Bearer ${refreshToken}`
    return config
}, async error => {
    throw error
})
```

Se il codice di stato della risposta è 401, significa che l'access token usato non è più valido e deve essere refreshato. In caso contrario, l'errore verrebbe semplicemente rilanciato e gestito dal componente.
Il meccanismo di rigenerazione dei nuovi token è gestito attraverso l'istanza "axiosRefreshTokensInstance". A questa istanza è collegato un interceptor che, ad ogni richiesta, aggiunge il token di refresh. Se la risposta HTTP presenta uno status code 200, l'operazione di refresh è stata completata con successo e i vecchi token vengono sostituiti con quelli appena generati. Successivamente, la richiesta originale viene eseguita nuovamente.
Se il codice di ritorno dovesse essere 401, anche il token di refresh è scaduto, e l'utente deve effettuare nuovamente l'autenticazione con le proprie credenziali.

3.6 Microservizi con Spring

Avendo come obiettivo quello di rilasciare l'applicazione sui principali store, è necessario costruire un'architettura solida, capace di gestire, eventualmente, molto traffico e facilmente aggiornabile. Per questo motivo, abbiamo deciso di implementare le fondamenta della business logic con Spring Boot.
Spring Boot è un framework open-source progettato per semplificare lo sviluppo, la configurazione e l'implementazione di applicazioni Java. Creato come parte del progetto Spring Framework, Spring Boot offre un approccio convenzionale sulla configurazione, eliminando la necessità di molte attività di routine e di configurazione manuale. Questo framework facilita la creazione di applicazioni robuste, scalabili e altamente performanti, fornendo un ambiente preconfigurato per lo sviluppo e l'esecuzione di applicazioni basate su Spring. Grazie alle sue caratteristiche di automazione e alle convenzioni intelligenti, Spring Boot permette agli sviluppatori di concentrarsi maggiormente sulla logica dell'applicazione, accelerando il processo di sviluppo e semplificando

la gestione delle dipendenze. Inoltre, Spring Boot supporta l'integrazione con una vasta gamma di tecnologie e offre un ecosistema ricco di strumenti che agevolano la creazione di microservizi e applicazioni enterprise moderne.
Qualora Annual dovesse essere molto usata, è necessario che gli utenti abbiano un'esperienza fluida e piacevole, il più possibile priva di rallentamenti e delay. Di conseguenza, la business logic è stata separata in più microservizi che cooperano assieme per fornire all'utente ciò che ha richiesto.
La strategia a microservizi rappresenta un approccio architetturale innovativo nel mondo dello sviluppo software, focalizzato sulla decomposizione di un'applicazione complessa in servizi modulari e autonomi. In contrasto con l'approccio monolitico, dove un'applicazione è sviluppata come un'unica entità, i microservizi consentono la realizzazione e la gestione di componenti indipendenti che svolgono funzioni specifiche. Ogni microservizio è sviluppato, distribuito e scalato autonomamente, spesso utilizzando tecnologie e stack tecnologici diversi. Questa architettura favorisce la flessibilità, la scalabilità e la manutenibilità delle applicazioni, permettendo agli sviluppatori di lavorare in modo più efficiente e alle organizzazioni di adattarsi rapidamente alle mutevoli esigenze del mercato. Tuttavia, la transizione verso un'architettura a microservizi richiede una pianificazione attenta e una gestione avanzata delle operazioni, poiché coinvolge la coordinazione di numerosi servizi distribuiti. Inoltre, in fase di deployment, bisogna tener conto di fattori "manageriali" che, normalmente, scivolano in secondo piano durante la fase di progettazione e scrittura del codice. Ad esempio, se si decidesse di fare affidamento a servizi di hosting come Amazon AWS o Google Cloud, è necessario tener conto dei costi richiesti per mantenere l'applicazione accesa. Generalmente, questi servizi specificano un costo mensile basato sulla quantità di risorse utilizzate (banda, cicli macchina, RAM eccetera). Chiaramente, maggiore è il numero di componenti che compongono il nostro sistema, maggiore sarà la quantità di risorse richieste dall'applicazione e, di conseguenza, più alti saranno i costi.

Dopo aver attentamente analizzato i requisiti, siamo passati alla fase di progettazione. In questa fase, abbiamo realizzato diversi diagrammi per semplificare la scrittura e l'organizzazione del codice. L'immagine 3.3 riassume questo processo, mettendo in luce le principali componenti della core logic e loro relazioni.

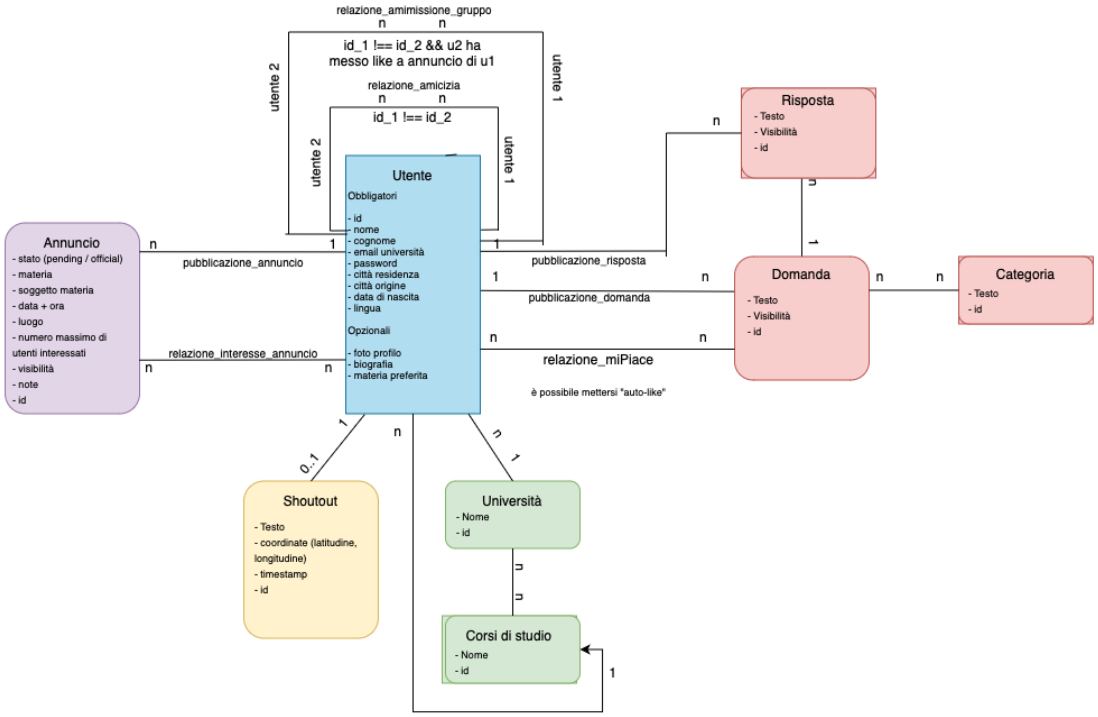

Figura 3.3: Separazione logica delle diverse parti dell'applicazione. I colori categorizzano un diverso microservizio

Ad alto livello, l'architettura si compone di molteplici bounding context, ovvero di aree di business logic separate l'una dall'altra. I bounding context fungono da fondamenta per la costruzione dei microservizi. La figura 3.3 raccoglie i bounding context (servizi) individuati. Si nota, in blu, il servizio che gestisce gli utenti, in viola il serizio per gli annunci, in rosso quello per il forum e in verde quello adibito alla gestione delle informazioni accademiche. In basso a sinistra, è presente il servizio di Shoutout (non ancora implementato). L'obiettivo di

Shoutout è facilitare gli incontri tra studenti di una determinata città. Se uno studente desidera organizzare un'attività come una partita a basket, una passeggiata in centro o una tranquilla giornata di shopping, e contemporaneamente desidera fare nuove conoscenze, può condividere uno Shoutout, un breve messaggio in cui viene indicato l'invito all'attività e la posizione per incontrarsi, consentendo agli altri utenti della stessa città di unirsi all'evento. Sfruttando Shoutout, studenti soli appena trasferiti hanno l'opportunità di socializzare e fare nuove esperienze.

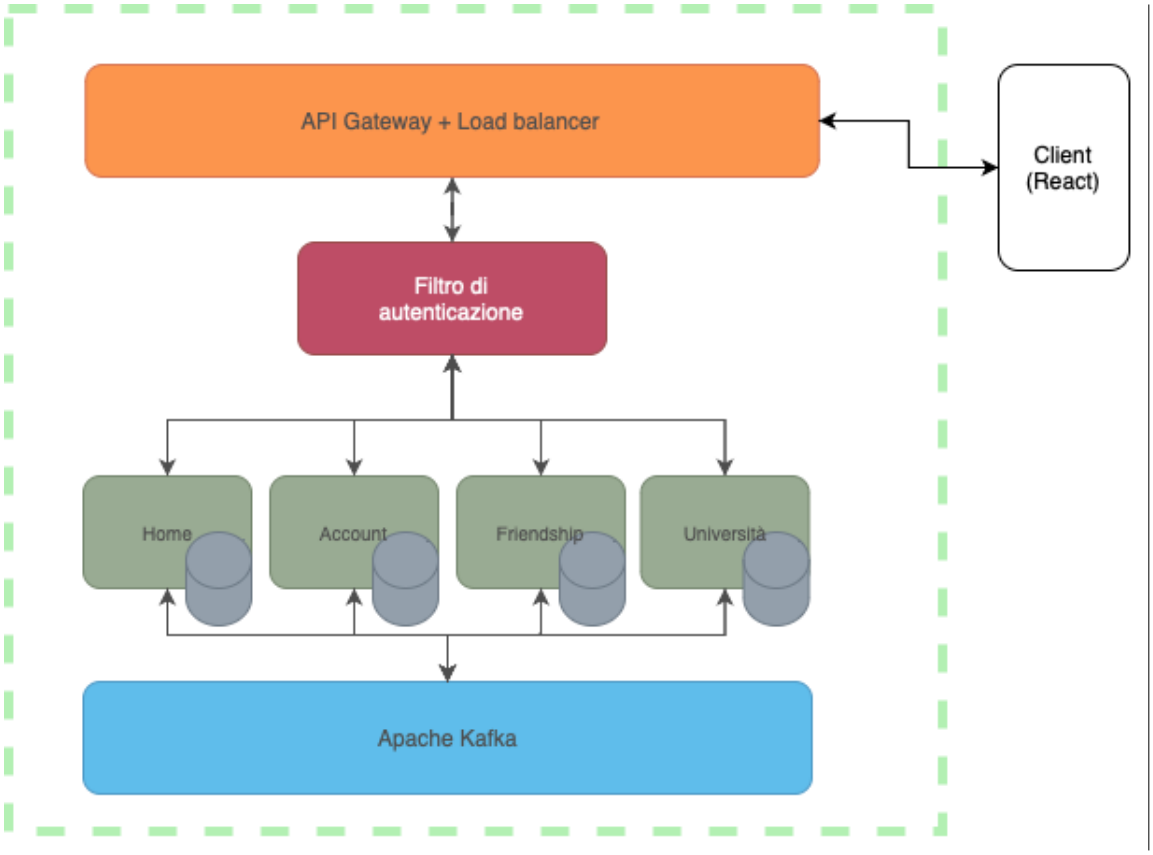

Figura 3.4: Diagramme dell'architettura a microservizi finale

Nella figura 3.4 viene mostrata la versione finale dell'architettura proposta. I servizi per la gestione degli annunci, del forum e degli

utenti sono stati accorpati, per mantenere i costi, nel servizio Home, un servizio monolitico che dialoga con un'istanza di Elasticsearch.
Ciascun servizio resta in ascolto su una porta. Per far sì che il client possa inviare richieste verso un singolo punto di contatto, è stato implementato un servizio di gateway. Il gateway non fa altro che instradare una richiesta verso il servizio corretto, richiedendo l'indirizzo IP dell'istanza a Eureka, un server il cui compito è registrare l'indirizzo IP di ciascun servizio. In questo modo, se due applicazioni dovessero comunicare e una delle due è stata riavviata, l'altra sarebbe in grado di inoltrare la richiesta (ad esempio attraverso WebClient) perché riceverebbe l'indirizzo IP corretto da Eureka. Inoltre, Eureka implementa un sistema di load balancing, permettendo al traffico di essere instradato verso istanze più scariche (se presenti), evitando il sovraccarico del sistema.
Prima di venir consegnata al servizio corretto, la richiesta attraversa una catena di filtri. Nel caso di Annual, viene fatto prima un controllo sulla validità del token di autenticazione (qualora fosse presente). Se dovesse ancora essere valido, header e corpo della richiesta (e della rispettiva risposta) vengono stampati e salvati su file, affinché sia possibile diagnosticare lo stato di ciascun servizio in caso di problemi.
A questo punto, la richiesta è pronta per essere gestita dal servizio a cui è diretta. Ciascun servizio è logicamente separato dagli altri, con il proprio database e la propria business logic. Tuttavia, non è possibile raggiungere l'isolamento completo: tutti i servizi lavorano all unisono per poter gestire le richieste provenienti dal gateway. La comunicazione inter servizio è stata realizzata con due strategie: la prima prevede di interrogare un servizio esterno inviando una richiesta ad un suo specifico endpoint tramite un'istanza dell oggetto WebClient. La seconda consiste nel rimanere in ascolto di messaggi scritti su uno o più topic di un server Apache Kafka.
Nelle pagine successive viene spiegato il funzionamento degli approcci appena elencati, i loro pro e contro e possibili casi d'uso.

3.7 WebClient

WebClient è una potente e flessibile libreria che offre un modo reattivo per effettuare chiamate HTTP in applicazioni basate su Spring. Progettato per integrarsi perfettamente con il paradigma di programmazione reattiva di Spring, il WebClient offre un'interfaccia intuitiva per eseguire operazioni asincrone e non bloccanti, migliorando l'efficienza e la scalabilità delle applicazioni. Con il supporto per la programmazione reattiva basata su Reactor, il WebClient semplifica la gestione delle richieste HTTP, consentendo agli sviluppatori di scrivere codice più conciso ed efficiente. Grazie alla sua versatilità, il WebClient di Spring è ideale per interagire con servizi RESTful, recuperare dati da API esterne e realizzare comunicazioni efficienti e affidabili tra i componenti distribuiti di un'applicazione.
WebClient ha la capacità di convertire le risposte http in Mono o Flux, due implementazioni dell'interfaccia Publisher di Spring WebFlux. Mono rappresenta una sequenza reattiva che può emettere al massimo un elemento, ed è spesso utilizzato per modellare risultati singoli o operazioni asincrone che restituiscono un unico valore. Flux, d'altra parte, rappresenta una sequenza reattiva che può emettere zero o più elementi, risultando particolarmente utile per rappresentare flussi di dati. È possibile manipolare il publisher restituito aggiungendo una catena di operatori che verranno eseguiti al fluire dei dati. Infatti, il vantaggio di usare Mono e Flux sta nel fatto che i dati fluiscono dal Publisher verso il Subscriber solo dopo che il Subscriber ha invocato il metodo subscribe() sul Publisher. A questo punto, le informazioni vengono emesse dal publisher, eventualmente manipolate dagli operatori, e giungeranno al subscriber che ne ha richiesto l'emissione. In questo modo, il Subscriber può ricevere la risposta in maniera asincrona rispetto a quando essa viene effettivamente calcolata e non è più necessario che rimanga in attesa del risultato.
In 3.4 viene mostrato un esempio di tale meccanismo. Il metodo "verifyMailDomain" ha il compito di verificare che un certo indirizzo email

sia stato rilasciato da un'università italiana, andando ad interrogare il servizio "university-service" e restituendo un Mono<Int?>: 1 se l'e-mail è valida, 0 se non valida. Questo metodo viene usato in fase di registrazione per assicurarsi che l'utente sia uno studente universitario.

Listing 3.4: WebClient

```
1  private fun verifyMailDomain(email: String): Mono<Int?>{
2      return webClient
3          .get()
4          .uri("http://university-service/api/v1/university/validDomain?mail=${email}")
5          .retrieve()
6          .onStatus(HttpStatus.NOT_ACCEPTABLE::equals) { response ->
7              (response.bodyToMono(Int::class.java).map {MailNotValidException()})
8          }
9          .bodyToMono(Int::class.java)
10         .transform {
11             circuitBreaker.run(it) { throwable ->
12                 if(throwable is WebClientRequestException) throw
       ServiceNotReachableException("/api/v1/university/validDomain?mail=${email}")
13                 if(throwable is WebClientResponseException) throw ExternalException()
14             throw throwable
15             }
16         }
17 }
```

Per costruire una richiesta con WebClient, vengono specificati metodo e uri della risorsa da interrogare (riga 3 e 4). Opzionalmente, è possibile specificare degli header con il metodo "headers". A questo punto, viene invocata la funzione retrieve() che esegue la richiesta e restituisce una WebClient.ResponseSpec (riga 5). Se lo stato della risposta dovesse essere NOT_ACCEPTABLE, viene invocata la callback a riga 7 che converte il corpo della risposta in un Mono<Int>, successivamente mappato, con l'operatore map, in una MailNotValidException(). In caso contrario, la risposta verrebbe convertita in un Mono<Int> a riga 9. A questo punto, il processo che ha eseguito la richiesta può convertire il Mono in questione in un oggetto CompletableFuture e invocarci il metodo get(), restando in attesa finché il risultato non è pronto.
L'operatore transform alla riga 10 ha il compito importante di eseguire la richiesta proteggendola con un circuit breaker. Il circuit breaker, nel contesto dello sviluppo del software, è un pattern progettato per migliorare la resilienza e la robustezza delle applicazioni distribuite. Analogamente al concetto di interruttore di circuito in elettronica, il circuit breaker software è un meccanismo che protegge un'applicazione

dalle conseguenze di operazioni fallite o di servizi non disponibili. Esso monitora continuamente le chiamate a un servizio remoto e, se rileva un numero significativo di errori o un fallimento prolungato nella risposta del servizio, interrompe temporaneamente le chiamate successive. Questo "interruttore" consente all'applicazione di gestire la situazione in modo più controllato, evitando di continuare a effettuare chiamate costose che potrebbero ulteriormente degradare le prestazioni o portare a problemi più gravi.
Tornando all esempio, la callback specificata nel metodo run dell'istanza del circuit breaker analizza l'eccezione catturata: se dovesse essere di tipo WebClientRequestException rilancerebbe una ServiceNotReachableException, se di tipo WebClientResponseException rilancia una ExternalException (che indica un'errore nel servizio interrogato). Nel caso in cui nessuna delle due condizioni dovesse risultare vera, il throwable verrebbe semplicemente rilanciato verso il chiamante.
Di seguito, viene riportata la configurazione del circuit breaker adibito al servizio di account management(3.5).

Listing 3.5: Configurazione del CircuitBreaker

```
resilience4j:
    circuitbreaker:
        instances:
            AccountManagerCB:
                failure-rate-threshold: 50
                slow-call-duration-threshold: 3
                slow-call-rate-threshold: 50
                ignore-exceptions:
                - com.annual.account_manager.exceptions.UniversityNotFoundException
                - com.annual.account_manager.exceptions.MailNotValidException
```

Il campo "ignore-exceptions" lista un'insieme di eccezioni che il circuit breaker non deve interpretare come errori. In questo caso, le statistiche del circuit breaker non verrebbero aggiornate nel caso un cui venisse lanciata una UniversityNotFoundException o una MailNotValidException.

3.8 Apache Kafka

Apache Kafka è una piattaforma di streaming distribuito progettata per gestire, elaborare e trasmettere flussi di dati in tempo reale su grandi scale. Di proprietà della Apache Software Foundation, Kafka si è affermato come uno degli strumenti chiave nell'ambito dell'elaborazione di eventi e della messaggistica distribuita. La sua architettura scalabile e resiliente consente di gestire volumi massicci di dati in modo affidabile, fornendo al contempo una latenza ridotta. Kafka opera secondo il paradigma publish-subscribe, consentendo a produttori e consumatori di dati di comunicare in modo efficiente e disaccoppiato. È particolarmente adatto per scenari in cui è cruciale garantire la disponibilità e la coerenza dei dati in tempo reale, come nell'analisi dei dati in streaming, nell'elaborazione degli eventi e nell'integrazione di sistemi distribuiti. Con caratteristiche come la persistenza su disco, la replicazione dei dati e l'architettura a log, Apache Kafka si è affermato come una soluzione robusta per le sfide legate alla gestione di flussi di dati in ambienti distribuiti su larga scala.
Nel caso di Annual, Apache Kafka viene utilizzato per comunicare ad alcuni servizi che è avvenuta una certa azione in un altro processo. Di conseguenza, i servizi interessati possono reagire, aggiornando certe informazioni e mantenendo la consistenza dei dati nel sistema.
Si immagini il caso in cui un utente decidesse di cambiare corso. La relazione tra studente e corso di studio è memorizzata nella tabella "student" nel database "university" e quest'informazione viene replicata in un campo del documento "User" in Elasticsearch. Questa configurazione ha il vantaggio di disaccoppiare, durante il fetch delle informazioni, l'utente gestito dal servizio Home e lo studente gestito dal servizio University ma ha lo svantaggio che, durante l'aggiornamento o l'eliminazione dell'informazione, il sistema deve garantire coerenza con i dati salvati nei due servizi. Per di più, se il processo destinatario (Home) dovesse essere momentaneamente spento, bisogna garantire che il messaggio venga recapitato al successivo avvio, così da assicurare la

congruenza dei dati.
Per ovviare a questo problema si potrebbero accorpare i due servizi in un unico processo, affinché le informazioni siano salvate in una sola locazione. Nel nostro caso, abbiamo preferito tenerli separati nell'eventualità in cui ci fosse bisogno di scalarli indipendentemente. Apache Kafka si è quindi rivelata la soluzione più opportuna.
Questo sistema presenta una falla piuttosto importante: un processo deve scrivere un messaggio su un topic Kafka se e solo se l'operazione è andata a buon fine. In caso contrario, il database del servizio produttore e quello del servizio consumatore si ritroverebbero in due stati diversi. Tornando al caso di Annual, supponiamo che l'utente abbia deciso di aggiornare il suo corso di studio e che l'aggiornamento fallisca (il dbms del servizio di università fa rollback della transazione). Se l'invio del messaggio venisse eseguito in parallelo rispetto all'esecuzione della transazione, il servizio home riceverebbe il messaggio e aggiornerebbe erroneamente la sua copia dei dati. Per di più non vi è nessuna garanzia che, se la transazione dovesse andare a buon fine, il messaggio venga inviato sul topic, dato che il sistema potrebbe spegnersi (a causa di un malfunzionamento) proprio nel momento in cui il messaggio verrebbe scritto.
L'Outbox Pattern nasce per rimediare a questo genere di problematiche. La figura 3.5 ne schematizza le componenti principali.

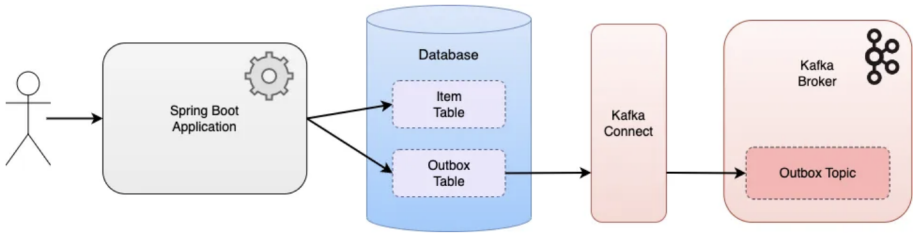

Figura 3.5: Outbox Pattern `https://microservices.io/patterns/data/transactional-outbox.html`

Secondo questa soluzione, l'applicazione demanda il compito di scrivere il messaggio sul topic ad un altro componente: il Message Relay (ruolo ricoperto da Kafka Connect in 3.5). Inoltre, il Database si compone di un'ulteriore tabella, chiamata Outbox Table, in cui vengono salvate le operazioni andate a buon fine eseguite sulla item table (nel caso di Annual, la item table è la tabella "student"). L'Outbox Table viene aggiornata nel contesto della stessa transazione che modifica la item table. In questo modo siamo sicuri che l'outbox table viene aggiornata se e solo se l'aggiornamento si è concluso con successo. Nel frattempo, il message relay resta in ascolto di modifiche sulla outbox table (nel nostro caso di operazioni di UPDATE) e, se dovesse rilevarne qualcuna, scrive un messaggio sull'outbox topic del server Kafka.
Con questa configurazione, siamo certi che il messaggio venga inviato solamente nel caso in cui l'aggiornamento della item table abbia successo. Inoltre, se il sistema dovesse andare fuori uso appena dopo il commit della transazione, Kafka Connect, al successivo avvio, leggerebbe dalla Outbox Table se ci sono aggiornamenti da propagare e, in caso affermativo, tenterebbe nuovamente di scrivere il messaggio sul topic. Infine, se l'architettura dovesse spegnersi durante la propagazione del messaggio (l'informazione è già scritta sul topic), Apache Kafka riuscirebbe a inviarlo in un secondo momento, sfruttando la sua architettura altamente affidabile.
Essendo limitati dalla quantità di risorse disponibili, Annual, al momento della stesura di questo documento, non supporta alcun tipo di implementazione dell'outbox pattern. Assumendo che le operazioni che necessitano di uno scambio di messaggi siano rare (aggiornamento del corso, aggiornamento dell'università e cancellazione dell'account) la probabilità di un malfunzionamento rimane bassa e possiamo assumercene il rischio.

3.9 Upload e download delle immagini

Annual permette di caricare delle immagini per caratterizzare meglio il profilo di ogni utente. In particolare, la foto profilo e l'immagine di background aiutano a dare vita all'intera applicazione, arricchendo annunci, domande e risposte con un tocco di personalità. Sfortunatamente, l'integrazione di questa funzione presenta notevoli sfide, tra cui la compressione delle immagini, la gestione dello spazio di archiviazione e la velocità di download. Per questo progetto, abbiamo scelto di appoggiarci ad Amazon S3, un potente servizio appartenente alla famiglia AWS (Amazon Web Service).
Amazon S3, acronimo di Simple Storage Service, è un servizio di archiviazione cloud. S3 è diventato uno dei servizi di storage più utilizzati e affidabili a livello globale, progettato per offrire scalabilità, affidabilità e accesso veloce ai dati. Amazon S3 consente agli utenti di archiviare e recuperare quantità virtualmente illimitate di informazioni in modo semplice e sicuro. S3 fornisce un'infrastruttura altamente disponibile per memorizzare oggetti, che possono essere immagini, video, file di grandi dimensioni o dati di backup. La sua versatilità e facilità d'uso lo rendono una scelta popolare sia per le piccole imprese che per le grandi aziende alla ricerca di una soluzione di archiviazione affidabile e flessibile nel cloud. Per di più, Amazon S3 risulta vantaggioso anche dal punto di vista dei costi. Infatti, uno dei piani inclusi nella suite di servizi AWS si basa sulla fatturazione delle sole risorse effettivamente impiegate dall'applicazione. Quindi, se AWS dovesse registrare una bassa attività, l'amministratore della piattaforma non sarà tenuto a spendere, ottimizzando al meglio le risorse economiche.
In linea generale, ridurre la quantità di risorse risulta vantaggioso, indipendentemente dal fatto che si registri un elevato o basso tasso di traffico. Per diminuire il numero di richieste verso il servizio S3, Annual implementa Caffeine, un efficiente libreria di caching basata su una struttura dati di tipo key-vaue disegnata per ottimizzare letture e

scritture da e verso la memoria principale. Grazie a questo meccanismo, Annual è in grado di fornire velocemente le immagini richieste, se presenti nella cache. Durante l'aggiornamento dell'immagine, viene eseguita una cancellazione dell'immagine precedentemente inserita nella cache, affinché possa venir caricata quella nuova alla prossima richiesta. In questo modo, ad ogni interazione eseguita, l'utente riceverà sempre l'ultima immagine caricata.

Listing 3.6: Configurazione di Caffeine CacheManager

```
@Configuration
@EnableCaching
class CachingConfig {

    @Bean
    fun caffeineConfiguration(): Caffeine<Any,Any> {
        return Caffeine.newBuilder().expireAfterWrite(Duration.ofMinutes(60))
    }

    @Bean
    fun cacheManager(caffeine: Caffeine<Any, Any>): CacheManager {
        val caffeineCacheManager = CaffeineCacheManager()
        caffeineCacheManager.setCaffeine(caffeine)
        return caffeineCacheManager
    }
}
```

Listing 3.7: Implementazione del cache manager

```
@Component
class UserCachingHelper {

    @Autowired
    private lateinit var imageService: ImageService

    @Cacheable(value = ["profile-image"], key = "#id.toString()", sync = true,
    cacheManager = "cacheManager")
    fun getUserProfileImageDeferred(id: UUID): Deferred<ByteArray> = CoroutineScope(
    Dispatchers.IO).async {
        imageService.getProfileImage(id)
    }

    @CacheEvict(value = ["profile-image"], key = "#principal.name", cacheManager = "
    cacheManager")
    fun uploadUserProfileImageDeferred(principal: Principal, file: FilePart): Deferred<
    Mono<UserDTO>> = CoroutineScope(Dispatchers.IO).async {
        imageService.uploadProfileImage(principal, file)
    }
}
```

In 3.6 viene mostrato il codice per configurare l'istanza del cache manager. Viene creato un cache manager a partire da un'istanza della libreria Caffeine, configurata per cancellare le informazioni memorizzate dopo un'ora dal loro inserimento.

La figura 3.7 presenta un utilizzo del cache manager appena configurato. I metodi "getUserProfileImageDeferred" e "uploadUserprofileImageDeferred" della classe "UserCachingHelper" sono annotati con @Cacheable e @CacheEvict. Il primo impedisce al sistema di eseguire il metodo "getUserprofileImageDeferred" se il suo risultato si trova nella cache "profile-image" alla chiave "#id.toString()", (l'identificatore dell'utente per cui si vuole recuperare l'immagine), evitando di consumare risorse di rete. In caso contrario, il metodo viene eseguito e il risultato salvato nella cache affinché la prossima richiesta possa recuperarne velocemente il contenuto. L'annotazione @CacheEvict segnala a Spring di rimuovere dalla cache "profile-image" alla chiave "#principal.name" (l'identificativo dell'utente correntemente autenticato) la risorsa attualmente memorizzata, se presente, ogni volta che il metodo "uploadUserProfileImageDeferred" viene invocato. Così facendo, ogniqualvolta l'utente voglia aggiornare la sua foto profilo, Annual sarà in grado di fornire la versione più recente della foto caricata.
Di seguito viene riportato uno schema che descrive a grandi linee il caching in Spring.

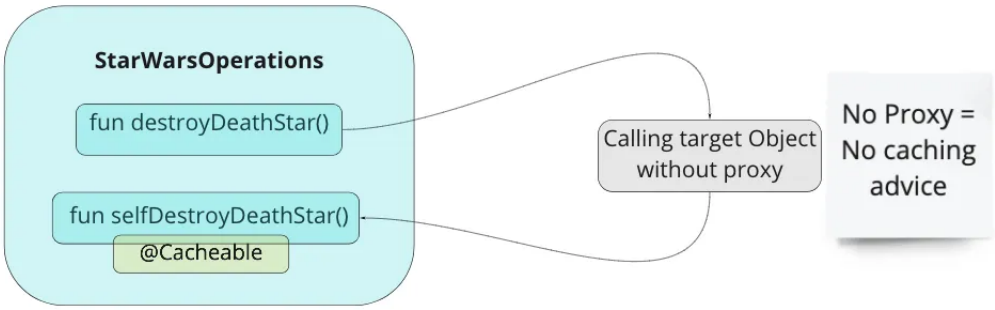

Figura 3.6: `https://medium.com/upday-devs/3-common-mistakes-when-implementing-spring-cache-abstraction-a7`

Nel caso dell'esempio fornito, la classe "StarWarsOperation" è costituita da due metodi: destroyDeathStar e selfDestroyDeathStar annotata con @Cacheable. Si supponga che all'interno di "destroyDeathStar" venga invocato il metodo "selfDestroyDeathStar". In questa situazione,

l'annotazione @Cacheable viene ignorata e il metodo viene eseguito in modo continuo. Questo comportamento è causato dal fatto che Spring genera, per ogni classe in cui si trovino una o più annotazioni di Spring cache, una nuova classe che estende quella originale (in questo caso, "StarWarsOperations"). Per ciascun metodo annotato con @Cacheable, @CacheEvict o @CachePut, vengono generati nuovi metodi che incorporano la logica di gestione della cache nella loro business logic. Pertanto, se un metodo viene chiamato da un altro metodo all'interno della stessa classe, nessuna richiesta sarà inoltrata verso la classe proxy e non verrà invocato il metodo generato da Spring, rendendo la cache inefficace.
Per abilitare le funzionalità di caching di Spring, è necessario racchiudere i metodi che si vogliono annotare con @Cacheable, @CacheEvict o @CachePut in un componente separato dal bean @Service e iniettarlo nel servizio come dipendenza. In questo modo, per ogni richiesta che include il download/upload delle immagini, Spring interrogherà prima la cache e, in caso di miss, inoltra la richiesta verso S3. Un'ulteriore semplice strategia utile alla riduzione del traffico consiste nel comprimere il file che sarà caricato.

Listing 3.8: Image upload

```
export const pickImage = async (compressionQuality = 0.2) => {
    try{
        await ImagePicker.requestCameraPermissionsAsync()
        const result = await ImagePicker.launchImageLibraryAsync({
            mediaTypes: ImagePicker.MediaTypeOptions.Images,
            allowsEditing: true,
            aspect: [4, 3],
            quality: compressionQuality,
        });

        if (!result.canceled) {
            return saveImage(result.assets[0].uri);
        }
        return null

    }catch(error){
        throw error
    }
};
```

La libreria ImagePicker di Expo consente di accedere all'interfaccia utente di sistema per selezionare foto e video dalla galleria o per utilizzare la fotocamera. Nel contesto di Annual, abbiamo configurato ImagePicker per accedere esclusivamente alla galleria e consentire all'utente di apportare modifiche all'immagine prima di caricarla, ad esempio ritagliandola. Il parametro "quality" accetta un valore compreso tra 0 e 1, che determina il livello di compressione da applicare all'immagine, variando da una compressione totale (0) all'immagine originale (1). Poiché le immagini vengono visualizzate in componenti di dimensioni ridotte, è possibile mantenere un livello di qualità relativamente basso senza che eventuali artefatti diventino troppo evidenti. Sperimentando con diversi valori, sembra che 0.2 rappresenti un compromesso adeguato tra compressione e qualità dell'immagine.
Se la variabile result dovesse indicare una corretta esecuzione del processo, l'immagine verrebbe rinominata (generando uno UUID pseudo-randomico) e salvata in "images/", una subdirectory di "Documents". In caso contrario, pickImage ritornerebbe "null" e sarà necessario ritentare.

Listing 3.9: Salvataggio dell'immagine selezionata

```
    const saveImage = async (uri) => {
    try {
        await ensureDirExists()
        const fileName = Crypto.randomUUID() + ".jpeg"
        const dest = imgDir + fileName
        await FileSystem.copyAsync({ from: uri, to: dest
    })

        return dest

    } catch (error) {
        throw error
    }
}
```

3.10 Data layer: PostgreSQL e Elasticsearch

Le informazioni sugli account, gli annunci, le domande e le risposte vengono memorizzate e gestite dal data layer. La nostra applicazione è composta da due soluzioni: alcune informazioni, come lo stato dell'account e della carriera universitaria e le relazioni di amicizia, sono raramente accedute, se non per fare il login/logout o registrazione, per modificare i dettagli della carriera (università o corso) o per inviare una richiesta di amicizia o per rispondere ad essa. L'accesso e la modifica di queste informazioni sono gestite da PostgreSQL, un potente sistema di gestione di database relazionali (RDBMS) open-source. Concepito per essere estensibile e conforme agli standard, PostgreSQL offre una vasta gamma di funzionalità avanzate che lo rendono perfetto per applicazioni di ogni dimensione e complessità. Tra le caratteristiche più significative, vi sono il supporto per transazioni ACID-compliant, la gestione efficiente di grandi quantità di dati, e la flessibilità nell'implementare modelli di dati complessi.
Al contrario, gli annunci, le domande, le risposte e i profili utente sono soggetti ad un fetch più frequente e devono essere memorizzati per garantirne l'efficienza. Abbiamo dunque optato per integrare un'istanza di Elasticsearch. Elasticsearch è un potente motore di ricerca e analisi distribuito, progettato per gestire grandi quantità di dati in modo rapido ed efficiente. Basato su Apache Lucene, Elasticsearch offre una ricerca full-text avanzata, consentendo agli utenti di esplorare e analizzare dati strutturati e non strutturati con estrema facilità. Grazie alla sua architettura distribuita e scalabile, Elasticsearch è in grado di gestire grandi volumi di informazioni in modo altamente disponibile e resiliente. Una delle caratteristiche distintive di Elasticsearch è la sua capacità di indicizzare e analizzare dati in tempo reale, rendendolo ideale per applicazioni che richiedono risposte istantanee e aggiornamenti continui.

Vengono mostrati due esempi di diagrammi ER. Il primo descrive il database "Università" in PostgreSQL mentre il secondo specifica la struttura dell'indice "Forum" in Elasticsearch.

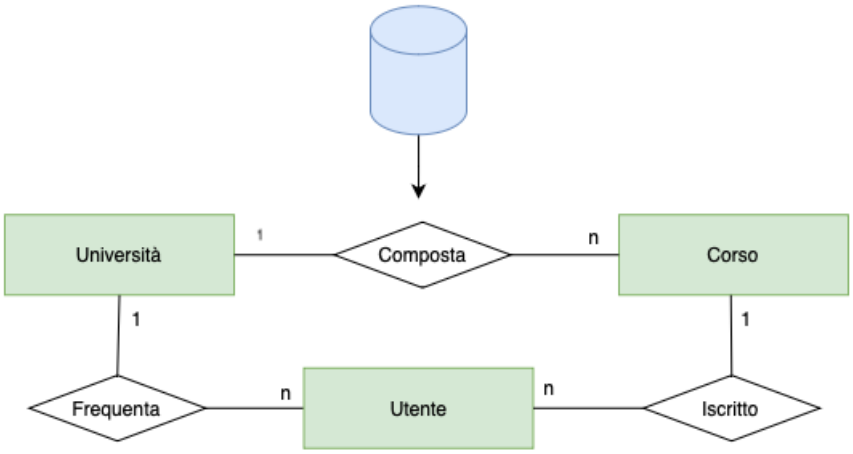

Figura 3.7: Entity-Relationship per il servizio di Università

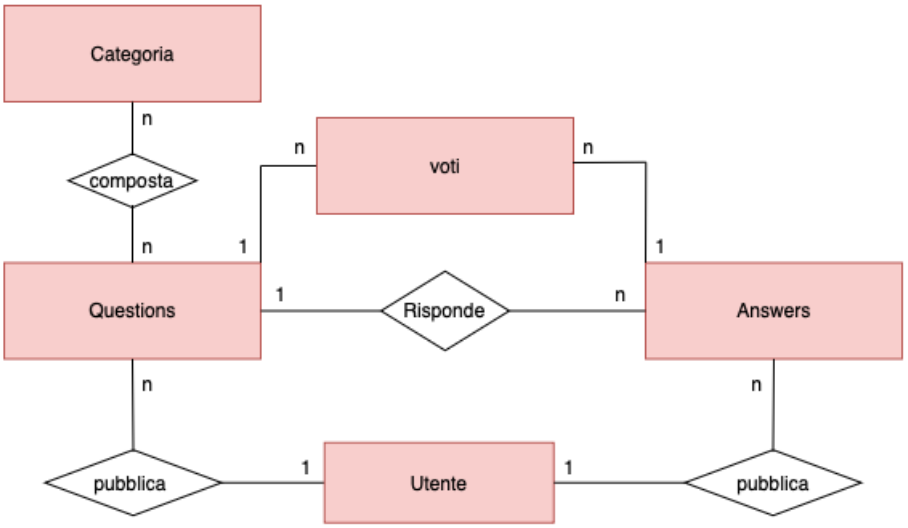

Figura 3.8: Entity-Relationship per il servizio Forum

Ricordiamo che Elasticsearch è un DBMS appartenente alla famiglia NoSQL. Ciò significa che non supporta il meccanismo delle relazioni ed è in grado di indicizzare grandi moli di dati sottoforma di strutture dati simili a oggetti JSON. Per di più, operazioni come la join sono proibitive a causa della gigantesca quantità di informazioni che andrebbero incrociate. Per questi motivi, a partire dal diagramma ER, è necessario riscrivere le entità affinché la struttura dati sia compatibile con la natura di Elasticsearch.
Per convertire il diagramma 3.8 in un formato comprensibile da Elasticsearch è stata inizialmente scelta una relazione che identifichi l'indice (nel gergo di Elasticsearch, l'indice è il corrispettivo della tabella nei DBMS relazionali). La scelta è ricaduta su "Questions" dato che l'obiettivo del Forum è quello di fornire velocemente le domande. Partendo da quest'assunzione, si è deciso che tutte le relazioni 1-n e n-n si tramutassero in una lista nel documento padre. Il documento "Question" sarebbe quindi composto da una lista contenente le categorie scelte al momento della pubblicazione, una lista di voti (up-votes e down-votes) e una di risposte, entrambe inizialmente vuote. Infine, per le relazioni 1-1 e n-1 è stato scelto di aggiungere semplicemente un campo che racchiuda l'identificatore della risorsa. Tornando all'esempio della domanda, essa può appartenere ad un solo utente il cui identificatore viene memorizzato nel campo "creator". Per recuperare tutte le informazioni dell'utente, Annual esegue un fetch del relativo documento nell'indice "User" a partire dal suo identificatore solamente quando è necessario, ad esempio durante il recupero di tutte le domande rilasciate da studenti della stessa università, creandone il DTO.
Un DTO (Data Transfer Object) è un oggetto che generalmente contiene informazioni diverse rispetto all'entità salvata nel data layer. L'obiettivo principale di un DTO è semplificare la comunicazione tra gli strati di un'applicazione o tra un'applicazione client e un server.

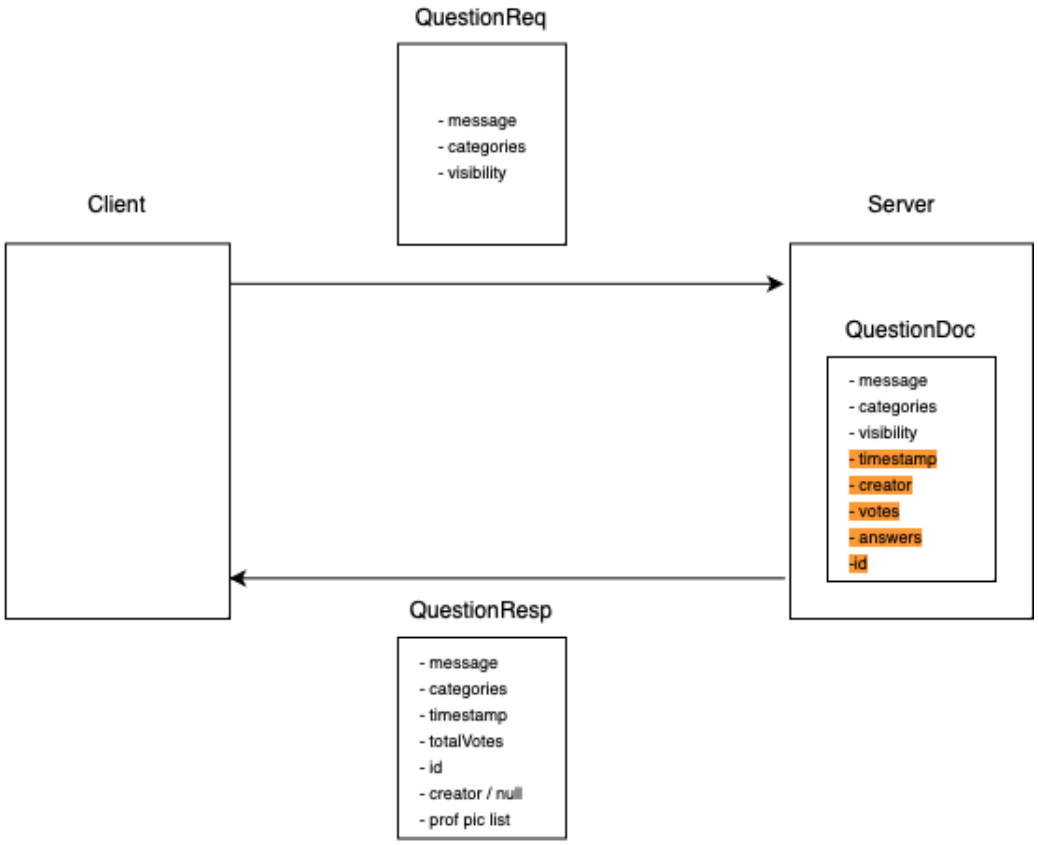

Figura 3.9: Esempio di richiesta/risposta tra client e servizio Forum

Nell'immagine 3.9 viene schematizzata l'interazione tra client e server durante la pubblicazione di una certa domanda. Il client richiede al server la creazione di una nuova domanda comunicandogli 3 parametri: il testo della domanda (message), la lista di categorie e il tipo di visibilità (anonima o con creatore visibile). A questo punto, alla ricezione del messaggio, il server elabora la richiesta creando un documento con le informazioni necessarie. Il DTO generato dal server (QuestionResp) non contiene tutte le informazioni salvate in Elasticsearch: per motivi di privacy, è necessario oscurare l'identità degli utenti che hanno votato una domanda, restituendone solo il voto totale (dato dalla somma di up-votes e down-votes). Inoltre, una domanda può essere pubblicata in forma anonima. In questo caso, se l'utente che recupera la domanda

fosse diverso da quello che l'ha pubblicata, il DTO censura l'identità dello studente settando a "null" il campo creator.
Sfruttado il pattern DTO, il client è completamente ignaro della presenza di eventuali informazioni sensibili, proteggendo la privacy degli studenti nel caso in cui si riuscisse a recuperare la risposta del server.

Capitolo 4

Containers e Hosting

4.1 Virtualizzare i processi con Docker

Docker è una piattaforma open-source che facilita la creazione, distribuzione e gestione di applicazioni containerizzate. Un container è un'istanza in esecuzione di un'immagine, un'unità standardizzata che racchiude l'applicazione insieme a tutte le sue dipendenze, garantendo un ambiente isolato e consistente indipendentemente dal sistema operativo sottostante. Docker permette agli sviluppatori di confezionare le proprie applicazioni insieme a tutte le risorse necessarie in un unico container, semplificando così il processo di sviluppo, distribuzione e scalabilità. Grazie alla sua architettura leggera e portatile, Docker consente agli utenti di eseguire facilmente le applicazioni in diversi ambienti, riducendo i problemi legati alle differenze di configurazione e assicurandone il funzionamento.
Per convertire il file jar compilato con Gradle in un'immagine Docker, è stato scaricato e configurato un plugin (`https://github.com/palantir/gradle-docker`) che, in maniera del tutto trasparente, esegue il comando **docker build** per creare un'immagine a partire da un Dockerfile. Un Dockerfile è un file di testo script che contiene una serie di istruzioni per la costruzione di un'immagine Docker, garantendo riproducibilità e consistenza nell'ambiente di sviluppo e distribuzione.

Viene di seguito riportata la configurazione del plugin in questione (4.1) e successivamente un esempio di Dockerfile per creare e configurare un ambiente isolato su Java 17 (4.2).

Listing 4.1: Configurazione del plugin Gradle

```
docker{
    name = "annual-home"
    setDockerfile(file("src/docker/Dockerfile"))
    copySpec.from(libsDirectory).rename(".*", "app.jar")
    buildArgs(mutableMapOf(Pair("JAR_FILE", "app.jar")))
}
```

Il frammento di codice indicato nel riferimento 4.1 dettaglia la configurazione concisa del plugin menzionato in precedenza. In questa sezione, sono specificati il nome dell'immagine che si vuole generare e il percorso del Dockerfile. Inoltre, viene eseguito il cambio di denominazione del file jar, e all'istruzione di build viene aggiunto l'argomento JAR_FILE, specificando il nome del file appena rinominato.

Listing 4.2: Dockerfile

```
FROM --platform:linux/amd64 eclipse-temurin:17
VOLUME /tmp
ARG JAR_FILE
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Nella porzione di codice 4.2 è riportato il contenuto del Dockerfile usato per convertire un file jar in un'immagine Docker. A grandi linee, verrà creata un'immagine a partire da eclipse-temurin 17 per piattaforma linux/amd64. Eclipse-temurin è un progetto che fornisce codice e processi per la creazione di binari basati su Java (in questo caso, Java 17). Partendo da quest'immagine, viene creato un volume con la parola chiave VOLUME e copiato il file jar (il cui path viene passato come argomento al comando build), compilato con Gradle, dentro l'immagine e rinominadolo "app.jar". Infine, ENTRYPOINT specifica il comando che verrà eseguito all'avvio del container. Nell'esempio riportato, viene

eseguito il comando **java -jar /app.jar** che manderà in esecuzione il jar copiato dall'operazione precedente.
Per gestire al meglio il set di immagini create, è stato scritto un docker compose affinché sia più facile definire la configurazione dei relativi container.
Docker Compose è uno strumento potente e flessibile che semplifica la gestione e l'orchestrazione di applicazioni containerizzate attraverso l'uso di file di configurazione YAML. Questo strumento consente agli sviluppatori di definire, configurare e coordinare diversi servizi, container e le relative interconnessioni all'interno di un'applicazione complessa. Docker Compose elimina la complessità associata alla gestione di più container separatamente, offrendo un approccio dichiarativo alla definizione delle risorse necessarie per l'esecuzione di un'applicazione.
Le righe successive riportano, per brevità, un estratto del docker compose che gestisce la configurazione delle diverse componenti che costituiscono Annual.

```
version: '3'
services:
  postgres:
    image: postgres:13.2
    container_name: postgres
    ports:
      - 5432:5432
    environment:
      - POSTGRES_PASSWORD=postgres
    networks:
      - annual-network
    volumes:
      - postgres-data:/var/lib/postgresql/data

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch
   :8.9.0-arm64
    container_name: elasticsearch
```

```
18      ports:
19        - 9200:9200
20      networks:
21        - annual-network
22      environment:
23        - xpack.security.enabled=false
24        - xpack.security.enrollment.enabled=true
25        - discovery.type=single-node
26
27    annual-eureka:
28      image: annual-eureka-test:latest
29      container_name: eureka
30      ports:
31        - 8761:8761
32      networks:
33        - annual-network
34
35    annual-home:
36      image: annual-home:latest
37      container_name: home
38      ports:
39        - 8090:8090
40      networks:
41        - annual-network
42      depends_on:
43        - elasticsearch
44
45  networks:
46    annual-network:
47      driver: bridge
48
49  volumes:
50    postgres-data:
51      driver: local
```

Ciascuna voce del campo "services" denota la configurazione di un container. Il campo "environment" raccoglie una lista di variabili d'ambiente. Ad esempio, per il container "postgres", viene specificata la password attraverso l'attributo POSTGRES_PASSWORD. Nel

caso in cui più persone dovessero lavorare al file, potrebbe essere necessario proteggere le credenziali, nascondendole ad individui non autorizzati. Docker consente di specificare il percorso di un file .env, in cui memorizzare tutte le variabili d'ambiente necessarie a configurare il container, attraverso l'opzione "env_file". In questo modo, il docker compose resta indipendente dal file di configurazione e può essere gestito in maniera totalmente trasparente.
Il campo "volumes" indica al container un volume su cui mappare una porzione del suo file system in quello dell'host, affinché alcune informazioni restino salvate permanentemente. Il container PostgreSQL sfrutta un volume per registrare i database con il loro contenuto, al fine di ripristinare le informazioni all'avvio.
"Networks" definisce la rete su cui eseguire il container. Docker consente la creazione di nuove reti virtuali, su cui è possibile eseguire uno o più container. La suddivisione in reti virtuali si rivela vantaggiosa quando è necessario isolare ambienti con configurazioni diverse, rendendoli inaccessibili tra loro.
Inoltre, si possono mappare una o più porte di un container ad una o più porte dell'host, così che sia raggiungibile dall'esterno. Infine, l'opzione "depends_on" subordina l'esecuzione del container stesso a quella di altri containers. Ad esempio, Docker lancerà in esecuzione "annual-home" dopo che avrà avviato Elasticsearch.

4.2 Hosting con AWS

Abbiamo già citato la suite AWS nel paragrafo 3.9 discutendo brevemente sul servizio S3. Oltre alle importanti capacità di storage, AWS offre la possibilità di hostare applicazioni web con il servizio EC2 (Amazon Elastic Compute Cloud), un servizio di cloud computing scalabile e flessibile. EC2 consente agli utenti di eseguire virtualmente macchine (istanze) su richiesta, fornendo una capacità di calcolo affidabile e adattabile alle esigenze dell'utente. Questo servizio offre una vasta gamma di

configurazioni hardware, permettendo agli utenti di selezionare risorse computazionali, memoria e capacità di storage in base alle specifiche necessità delle loro applicazioni.
Per prima cosa, le immagini buildate sono state caricate su Amazon ECR (Elastic Container Registry), un registro di container Docker fornito da Amazon. A questo punto, con l'aiuto di FileZilla (un client FTP) è stato caricato il docker compose descritto in precedenza sull'istanza EC2. Attraverso una sessione ssh, il docker compose in questione è stato leggermente modificato, affinché il campo "image" di ciascun servizio puntasse alla relativa immagine caricata su ECR. Prima di lanciare in esecuzione l'applicazione, sono state modificate le regole in entrata dell'istanza EC2 per accettare richieste di connessioni sulla porta 8761 (Eureka). Infine, mediante ssh, dopo aver configurato a dovere la macchina remota, è stato eseguito il comando **docker-compose -f docker-compose.yml up** e tutti i servizi dichiarati nel docker compose sono stati scaricati da ECR e mandati in esecuzione.

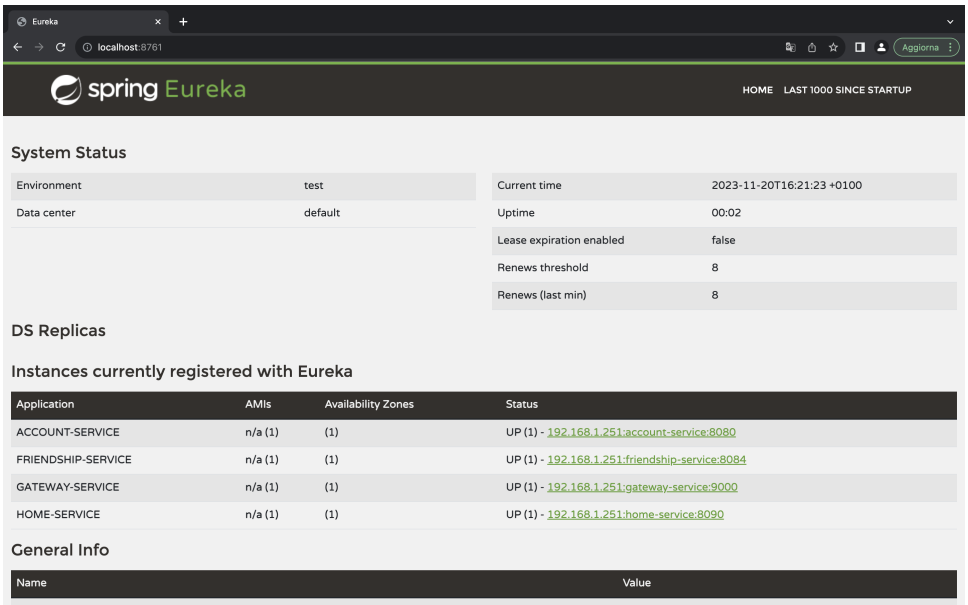

Figura 4.1: Eureka console

4.3 In conclusione...

Il mondo per come l'hanno conosciuto i nostri genitori è ormai un lontano ricordo. In un'epoca in cui la tecnologia avanza a passi da gigante e le dinamiche sociali si trasformano in modo incessante, ci troviamo di fronte ad un panorama in costante evoluzione, dove tutti corrono alla cieca senza mai fermarsi. Diamo per scontato la presenza dei nostri cari senza mai coltivarla perché siamo troppo impegnati a "conquistare il mondo" e le discussioni si trasformano in vuoti dibattiti per il puro gusto di avere ragione, senza un vero ascolto reciproco. Qualsiasi attività o passione viene trasformata in competizione, spesso inutile, per il puro piacere di auto-proclamarsi "il migliore". Scegliamo di farci influenzare e polarizziamo le nostre opinioni, dimenticando l'importanza di ascoltare prima di rispondere. Abbiamo la costante paura che il mondo possa finire da un giorno all'altro ma la verità è che siamo troppo pigri per sbirciare fuori dalla finestra e scoprire che, in realtà, il cielo è sempre blu, il sole caldo e la pioggia bagnata. Non nego di temere per il futuro quando, dinnanzi a barbarie, il nostro governo propone di integrare nel programma scolastico ulteriori lezioni con psicologi e psichiatri per "estirpare il male" dalle menti di giovani ragazzi, marcando, ancora una volta, la più completa incapacità di capire il reale problema. Credo che l'intero pianeta stia sanguinando e non abbiamo davvero intezione di curarlo ma di aggiungere garze e cerotti su una ferita che non si rimargina, lasciando che sia il prossimo a risolvere la questione.
Non ho la presunzione di avere in mano le chiavi di un futuro migliore, ma ho sempre pensato che l'innaturale livello di "socialità" raggiunta negli ultimi 10 anni abbia pesantemente aggravato la situazione, allontanandoci fisicamente ed emotivamente. Sono fortemente convinto che la felicità del singolo individuo possa contagiarne altri, illuminando la loro giornata in una settimana buia.
Questo è il motivo che mi ha spinto, quel giorno, ad abbracciare questo

progetto e a versare ogni goccia di sudore pur di renderlo realtà. Annual è costruito su fondamenta semplici e il suo obiettivo primario è quello di migliorare la giornata di studenti universitari dando loro un vero spazio per conoscersi, integrarsi e addirittura legarsi. Annual non nasce per creare dipendenza o per polarizzare opinioni ma per accollarsi l'arduo compito di far da stampella ad un livello di espansività ormai zoppicante.
Mi auguro vivamente che questo progetto possa fare la differenza nella vita delle persone pur rimanendo una goccia sperduta nel mare.