Unireview

Report Finale

SCOPO DEL DOCUMENTO

In questo documento stiliamo un report complessivo del processo seguito. Partiamo fornendo un riassunto dei seminari seguiti per poi spiegare quale sia stato il nostro approccio lavorativo anche in relazione ad essi. Quindi spieghiamo come abbia funzionato l'organizzazione e la distribuzione del lavoro nel nostro gruppo, specificando anche le criticità incontrate e fornendo un'autovalutazione.

1. APPROCCI ALL'INGEGNERIA DEL SOFTWARE

Nella sezione che segue presentiamo i punti che abbiamo ritenuto più importanti per ciascun seminario presentato nel contesto del corso. Dopodiché, illustriamo in breve come il contenuto di tali seminari abbia influenzato il nostro lavoro sul progetto UniReview.

1.1 I seminari

1.1.1 Metodi di Ingegneria del software applicati allo sviluppo di un progetto di AI

Il primo seminario è stato tenuto da Jonni Malacarne, CEO di BlueTensor. Malacarne ha illustrato come l'ingegneria del software si inserisce nel contesto dello sviluppo di progetti legati che coinvolgono l'intelligenza artificiale. L'approccio per tali progetti è perlopiù simile a quello per lo sviluppo di un progetto software tradizionale, ma sono presenti variazioni specifiche dovute appunto all'uso dell'AI. Lo sviluppo è suddiviso in fasi che includono uno studio di fattibilità iniziale, la creazione di una *proof of concept* (PoC), lo sviluppo di un prodotto minimo funzionante (MVP) e un miglioramento continuo. Ogni fase è supportata da strumenti specifici come UML, diagrammi entità-relazione, mockup, Python, AWS e Docker. Questi facilitano l'analisi, la comunicazione con il cliente, e la gestione del ciclo di vita del software.

Il seminario ha anche messo in evidenza l'importanza di aspetti come la disponibilità e la qualità dei dati, la necessità di un team multidisciplinare e aperto mentalmente, e le considerazioni etiche e di privacy. Sono stati discussi casi studio concreti di applicazioni AI in diversi settori, come il controllo qualità, la sicurezza e videosorveglianza, e la manutenzione predittiva. Il seminario si è concluso mettendo a fuoco ciò che viene utilizzato per supporto al ciclo di vita del progetto, enfatizzando l'importanza di una gestione agile e integrata dei flussi di lavoro, nonché la formazione continua degli utenti e il miglioramento costante

1.1.2 Kanban Simulation

Il seminario, condotto da York Rössler, ha fornito un'introduzione dettagliata al metodo Kanban, evidenziando le sue origini nel sistema di produzione Toyota e la sua evoluzione per la gestione del knowledge work.

La sessione è iniziata con una spiegazione teorica del Kanban, illustrando come visualizzare il flusso di lavoro e controllare il Work In Progress (WIP) per garantire che tale flusso sia fluido e continuo. È stata sottolineata l'importanza della collaborazione del team e della gestione del cambiamento in ambienti incerti e in rapido mutamento.

Successivamente, grazie a una simulazione in tempo reale, i partecipanti hanno avuto l'occasione di vedere in azione quanto illustrato, ricreando un ambiente di lavoro dove implementare e testare le regole del Kanban, come i limiti del WIP e le riunioni standup giornaliere. Durante la simulazione, i partecipanti hanno raccolto dati sui tempi di consegna e osservato gli effetti delle limitazioni del WIP sulla velocità e qualità del lavoro. La sessione si è conclusa con un'analisi statistica dei risultati della simulazione, una discussione sui benefici dell'agilità e delle pratiche Kanban, e sull'importanza di tempi di ciclo ridotti per rispondere rapidamente ai cambiamenti del mercato e migliorare la qualità complessiva del prodotto.

1.1.3 Cloud Computing with IBM Cloud

In questo seminario Ferdinando Gorga, specialista tecnico del cloud pubblico in IBM Technology Sales Italy, ha presentato un approfondimento su IBM Cloud, una piattaforma distribuita che offre sia potenza computazionale che servizi software per supportare la modernizzazione dei sistemi e la trasformazione digitale delle aziende. IBM Cloud è ospitato in numerosi data center sparsi globalmente e permette agli utenti di concentrarsi più sullo sviluppo del codice e meno sull'infrastruttura grazie a tecnologie come bare metal, virtual machine, container e soluzioni serverless come Code Engine. Gorga ha spiegato come questa piattaforma risponde alle sfide della sicurezza informatica, proteggendo le applicazioni da attacchi DDoS e garantendo la segretezza dei dati tramite tecnologie crittografiche avanzate.

Il seminario ha presentato alcuni casi studio reali, come la creazione di un ambiente NodeRed in CodeEngine, illustrando i costi associati e le potenzialità di miglioramento attraverso integrazioni con sistemi di autenticazione e notifiche. È stata inoltre discussa l'importanza di un'architettura cloud flessibile in grado di sfruttare diverse combinazioni di tecnologie come Kubernetes e servizi cloud nativi di database. Per finire, Gorga ha illustrato le possibilità offerte dall'IBM Cloud per la nascita e lo sviluppo di startup innovative, con un esempio di come una nuova idea di business potrebbe essere implementata utilizzando le risorse cloud di IBM.

1.1.4 Software Engineering @Meta

Il seminario di Heorhi Raik, ingegnere del software per Meta, ha illustrato il significato della sua professione in questa grande azienda, passando dalla storia personale, al funzionamento dei processi di software engineering, per finire con il tech stack. Raik ha lavorato per tutti e otto gli anni in *Catalog*, sezione dedicata ad assimilare dati caricati da chi vuole fare advertising su meta. Ha chiarito che la mobilità all'interno di Meta è molto rapida, motivo per cui si è trovato ogni anno sotto un manager diverso.

Raik ha inoltre spiegato come in Meta i software engineer siano coloro che portano avanti i progetti (*software engineer driven company*): il coding è una parte meno importante lasciata perlopiù ai novizi, mentre ingegneri più esperti possono assumere altre responsabilità, sia tecniche, come architecture design e project management, sia non tecniche, come interviewing e team building, il tutto senza essere vincolati da particolari metodologie.

Il seminario è continuato illustrando il tech stack di Meta: molto di ciò che viene utilizzato è costruito *in situ* per ovviare a problemi di scaling, ad esempio il sistema version control basato su Mercurial che rende automatica la pipeline, o l'IDE interno basato su Visual Studio Code. I linguaggi più utilizzati sono Hack (PHP), React (JS), Python e Presto (SQL), tre dei quali sono dialetti sviluppati internamente. Per finire Raik ha presentato degli approfondimenti sul funzionamento del version control, sul background legale, e sull'organizzazione del lavoro.

1.1.5 Lessons learned from the trenches

In questo seminario, Daniele Miorandi di U-Hopper ha condiviso le sue esperienze pratiche nell'ingegneria del software, mettendo in luce le sfide affrontate e le soluzioni implementate per migliorare la gestione delle risorse umane e l'analisi di big data.

Dopo aver presentato tecniche, metodologie e strumenti di ingegneria del software usati da U-Hopper, Miorandi ha spostato il focus su alcuni progetti realizzati dall'azienda, come sistemi di monitoraggio delle presenze dei dipendenti, che includono funzionalità quali richieste di ferie, permessi, malattia, telelavoro e gestione dei rimborsi di viaggio. Questi sistemi non solo migliorano

l'efficienza amministrativa, ma anche la soddisfazione dei dipendenti, offrendo una panoramica completa delle loro richieste e del rispettivo stato.

Un altro progetto di U-Hopper ha fatto uso di Internet of Things (IoT) per il monitoraggio delle fabbriche attraverso l'accesso in tempo reale ai dati dei sensori e la creazione di cruscotti personalizzati per l'analisi e la visualizzazione delle informazioni. Questi strumenti hanno permesso di configurare allarmi e notifiche basati su analisi in tempo reale, rendendo il processo di gestione delle risorse più agile e reattivo. Inoltre, Miorandi ha illustrato l'uso di chatbot intelligenti per le smart city, presentando il caso di Milano, dove un chatbot ha migliorato l'accesso dei cittadini ai servizi e alle informazioni, riducendo i costi di gestione e aumentando l'efficienza del servizio di assistenza.

Nel complesso, il seminario ha sottolineato l'importanza di integrare tecnologie avanzate e soluzioni personalizzate per affrontare le sfide moderne della gestione operativa e dell'interazione con i cittadini, evidenziando come queste innovazioni possano portare a una maggiore efficienza e soddisfazione sia per le aziende che per gli utenti finali.

1.1.6 My journey into the open source world

Il seminario tenuto da Mario Fusco ha raccontato il suo viaggio professionale nell'open source, iniziato nel 1998 come sviluppatore Java. Fusco ha parlato di come il suo primo progetto open source, LambdaJ, lo abbia introdotto a una comunità che valorizza la condivisione della conoscenza e la revisione tra pari. Questi elementi, secondo Fusco, sono fondamentali per l'apprendimento e il miglioramento continuo del codice, garantendo una meritocrazia dove contano solo le idee e la loro implementazione. Fusco ha sottolineato l'importanza della visibilità e del supporto comunitario, che permette agli sviluppatori di farsi conoscere senza dover passare per colloqui di lavoro tradizionali.

Fusco ha illustrato anche le principali categorie di licenze open source: copyleft (come GPL, LGPL, Affero GPL), che obbligano la redistribuzione delle modifiche con la stessa licenza dell'originale, e non-copyleft (come Apache, MIT, BSD), che permettono maggiore flessibilità nella licenza delle opere derivate. Discutendo i benefici dell'open source per le aziende, Fusco ha evidenziato come queste possano ridurre i costi, evitare vincoli con fornitori e ottenere software di qualità superiore. Le aziende possono offrire servizi di supporto, consulenza e

implementazione di funzionalità specifiche come valore aggiunto, trasformando l'adozione dell'open source in un vantaggio competitivo.

Infine, il seminario ha esplorato i vari livelli di partecipazione ai progetti open source, dai principianti, che segnalano bug, ai "maghi", che contribuiscono con nuove funzionalità e revisioni, fino ai membri del core team. Fusco ha sottolineato l'importanza delle pubbliche relazioni, dell'accettazione di contributi esterni e del mentoring per mantenere un progetto open source sano e in crescita. Il seminario si è concluso incoraggiando i professionisti a partecipare al mondo open source per ottenere autonomia, padronanza delle competenze, soddisfazione intellettuale e connessioni significative nella comunità globale

1.1.7 Software Testing

Il seminario è stato condotto da Diego Colombo di Microsoft. Colombo ha illustrato diverse tipologie di software testing inserendole nel loro contesto e fornendo degli esempi, mettendo in guardia che le innovazioni AI potrebbero rendere le presenti pratiche di testing obsolete. La prima parte del seminario ha sottolineato l'importanza di Test-Driven Design (TDD) e Behavior-Driven Design (BDD), presentando diversi tipi di testing (unit, functional, integration…) e il loro scopo, che non è limitato alla ricerca dei bug (definiti come qualunque evento imprevisto che avviene sul computer frustrando l'utente). Colombo ha messo in rilievo quanto siano cruciali gli acceptance test e il convalidare gli acceptance criteria con il cliente per rendere fluido il cash flow, a riconferma dell'importanza di sviluppare software in un contesto di TDD.

Il seminario è continuato presentando alcuni strumenti per il testing: framework, log, Behavior Driven Development Domain Specific Language (BDD DSL). Colombo ha evidenziato ancora una volta l'importanza di un testing continuo e in diversi ambienti e fornendo una "regola d'oro" per il testing: *act* (la funzionalità) e *assert* (il testing) devono avvenire allo stesso livello: servizio, app, libreria…. Per finire sono stati presentati esempi concreti di testing su codice, suggerendo l'utilizzo di framework di asserzione per mantenere il testing allo stesso livello e strumenti come Wallaby.

1.1.8 Sistemi Legacy e Software Engineering: davvero i dinosauri si sono estinti?

Il seminario è stato presentato da Andrea Molinari, specialista di sistemi legacy. Partendo da un inquadramento settore, Molinari ha proseguito facendo del fact-checking attorno alle affermazioni che circolano sui sistemi legacy, esaminando le questioni sollevate dal loro utilizzo, il modo in cui il mondo legacy continua a ritornare e per finire la progettazione e la gestione di progetti in ambienti legacy.

Il mondo legacy è definito da una certa obsolescenza (nell'hardware, nel software…), tuttavia è ancora in uso per una serie di motivi, sintetizzabili nell'affermazione che il rapporto tra costo totale e benefici totali (TCO/TBO) del loro utilizzo è ancora favorevole rispetto a un loro rimpiazzo con un sistema più moderno della stessa scala. Un tratto distintivi dei sistemi legacy è l'hardware mainframe-class con vendor lock-in, vecchi apparecchi con UI basilare e grossi consumi, caratterizzati anche da una significativa mancanza di personale qualificato. Molinari ha spiegato che questi sistemi lavorino su scala molto grande e con high availability.

Continuando, il seminario ha mostrato i principali linguaggi usati nell'ambito (ad esempio RPG, oppure COBOL, molto usato in ambito bancario), spiegando come sia praticamente impossibile trasferirsi su linguaggi più moderni, sia riscrivendo il codice da zero, sia traducendolo, e come quando ciò sia necessario si ricorra a soluzioni ibride. Molinari ha evidenziato i problemi di UX, di persistenza, di costo dei sistemi legacy. Dati tutti queste criticità, è importante spiegare i motivi del loro continuo: rientrare negli investimenti, robustezza, paura di resistenze. Molinari ha evidenziato che man mano che invecchiano le architetture le aziende accumulano *tech* debt. Questo inibisce lo sviluppo, crea insoddisfazione nel cliente e problematica per costi e sicurezza. Però in compenso queste architetture hanno la capacità di gestire grandi numeri di transazioni, enorme potenza di calcolo, garantiscono continuità, e hanno grossa abilità di disaster recovery. Paradossalmente, presentano anche qualche vantaggio in termini di sicurezza – anche a livello fisico, dato che si trovano in posti difesi e custoditi.

Il seminario si è concluso illustrando diversi modi per "modernizzare" questi sistemi legacy, che vanno dalla dismissione (solo trasferimento dei dati) alla migrazione (trasferimento software e dati nel nuovo ambiente) all'interazione

(sistema legacy continua a esistere interagendo con il nuovo) all'inclusione (il sistema legacy viene incorporato nel sistema moderno come una sua componente). Molinari ha messo in rilievo il ruolo cruciale di competenze di software engineering in questo ambiente, chiarendo però i limiti di metodologie moderne come l'agile, e presentando strumenti come UML e diagrammi di distribuzione. Per finire sono stati presentati dei casi studio evidenziando criticità generate dalla densità della documentazione.

1.1.9 Application Lifecycle Management and Modernization

Il seminario di Gerardo Marsiglia ha esplorato in dettaglio il ciclo di vita delle applicazioni software e le pratiche di modernizzazione necessarie per rimanere al passo con le tecnologie emergenti. Marsiglia ha iniziato illustrando le diverse architetture software, inclusi i componenti di dati, applicazioni, tecnologie e business, e come queste si interconnettono per formare un sistema. Ha poi continuato con un approfondimento sul ciclo di vita del software, sulla gestione della configurazione, sulla gestione dei progetti e sul cambiamento, evidenziando l'importanza della collaborazione tra i team e l'uso di strumenti specifici.

L'evoluzione dei processi di sviluppo software è stata analizzata attraverso un confronto tra i metodi tradizionali, iterativi, agili e DevOps. È stata sottolineata l'importanza dell'automazione in tutte le fasi del ciclo di vita del software, dalla costruzione e test all'integrazione continua e al rilascio. Il seminario è proseguito con una discussione sulle tecnologie cloud, descrivendo i vari servizi (SaaS, PaaS, IaaS) e i tipi di cloud (pubblico, privato, ibrido), nonché i benefici della modernizzazione delle applicazioni per sfruttare le potenzialità del cloud. Infine, Marsiglia ha presentato le pratiche e gli strumenti per l'adozione del DevOps, includendo esempi di architetture di riferimento e strumenti specifici, come quelli forniti da Microsoft Azure, IBM e AWS, e evidenziando come l'adozione di microservizi e applicazioni cloud-native sia essenziale per la scalabilità e l'efficienza operativa.

1.10 Apss e il Dipartimento tecnologie

Il seminario dell'APSS di Trento ha delineato la missione e i principi ispiratori dell'ente, focalizzandosi sulla gestione coordinata delle attività sanitarie e sociosanitarie a livello provinciale. Il Dipartimento Tecnologie è stato descritto

come un elemento chiave nel garantire soluzioni digitali e tecnologie sanitarie innovative, essenziali per la programmazione e produzione dei servizi per cittadini, pazienti e operatori. Tra i servizi presentati vi sono l'Ingegneria Clinica, che si occupa della gestione e manutenzione delle attrezzature sanitarie, e le Operazioni e Infrastrutture IT, che è responsabile della gestione delle infrastrutture tecnologiche e della trasformazione digitale.

Durante il seminario, sono stati illustrati vari progetti trasversali come il Fascicolo Sanitario Elettronico, il Patient Summary e la Telemedicina, che mirano a migliorare l'accessibilità e l'efficienza dei servizi sanitari. Sono stati anche forniti dati significativi riguardo l'utenza, il personale e le attrezzature gestite, oltre a una mappa delle applicazioni e dei sistemi informativi utilizzati. Gli obiettivi strategici per il futuro includono la continuazione dell'innovazione tecnologica, la semplificazione dei processi decisionali e l'aumento dell'efficienza operativa tramite l'integrazione di nuove tecnologie e l'ottimizzazione dei sistemi esistenti.

1.2 Considerazioni e approcci adottati

Data l'importanza di partire subito con il progetto, abbiamo assimilato le lezioni contenute in questi seminari mano a mano che lo abbiamo realizzato. Infatti, non siamo partiti con nessuna metodologia specifica, ma, nel corso del lavoro, ci siamo resi sempre più conto della necessità di dividere i compiti e comunicare tra i membri del team, come sottolineato, ad esempio, nel secondo seminario (Kanban Simulation). Ciò è avvenuto in parte attraverso incontri – in particolare all'inizio, quando la divisione dei compiti non era ancora ben definita – ma soprattutto attraverso app di messaggistica che hanno consentito di ottenere feedback immediato sul lavoro svolto da parte degli altri membri del team.

In sostanza, l'approccio è cambiato da una prima suddivisione dei compiti grossolana e per porzioni a una più precisa e per funzioni, integrata da revisioni costanti da parte degli altri membri e dalla possibilità di correggere e rendere coerente il tutto attraverso sistemi di version control via Github.

Per quanto siano state riscontrate diverse criticità nel corso della realizzazione del progetto (di cui sotto), l'esperienza è stata altamente formativa: oltre all'importanza di comunicazione e divisione dei compiti, già sottolineate, ci ha insegnato come la documentazione non sia importante solo per il cliente, ma in primo luogo per gli ingegneri stessi, poiché consente loro di rendere coerente tanto il flusso di lavoro quanto il prodotto finale.

2. ORGANIZZAZIONE DEL LAVORO

Come riportato nella sezione precedente, il livello di organizzazione è aumentato mano a mano che il lavoro procedeva. All'inizio la divisione dei compiti è stata piuttosto grossolana. Ruggero Antonacci ha assunto il ruolo di team leader. Abbiamo selezionato l'idea per il progetto esaminando una serie di possibili proposte, circa tre per membro, optando per quella che ci sembrava più fattibile e interessante.

Abbiamo effettuato una serie di riunioni in presenza mentre lavoravamo ai primi tre deliverable. Ci siamo aiutati con chiamate VoIP di gruppo quando uno dei membri non aveva la possibilità di partecipare fisicamente. Il lavoro su questi documenti è stato spartito più o meno equamente e per porzioni, spesso con più di una persona che si è occupata della stessa porzione.

Dal quarto documento in poi la suddivisione delle mansioni è stata più precisa e funzionale. Matteo Nanni si è occupato del front-end, Sasha Businaro del back-end, mentre Ruggero Antonacci della stesura dei documenti.

Durante tutto il lavoro ci siamo avvalsi di sistemi di messaggistica istantanea per facilitare la comunicazione e il feedback, nonché di strumenti quali GitHub, Google Documents e Google Drive per condividere il nostro lavoro, di VSCode per la stesura del codice. Per il front-end ci siamo avvalsi del framework javascript Next.js con Ant Design per una user experience più fluida e di Vercel per il deployment. Per il back-end di MongoDB Atlas, Google OAuth e di Railway per il deployment. Swagger per la documentazione API e Jest per il testing.

Per alcuni segmenti è stato necessario che i componenti del gruppo lavorassero in modo autonomo, chi per implementare le API, chi per realizzare l'interfaccia grafica o per redigere la struttura dei documenti.
Tuttavia alla fine di ogni traguardo intermedio ci siamo confrontati, chiedendo valutazioni e correzioni da parte degli altri membri, permettendo di avere sempre un quadro generale del progetto rimanendo sincronizzati.

2.1 Ruoli e attività

Di seguito elenchiamo come sono stati suddivisi i ruoli nel team e di quali attività sono state svolte dai membri del gruppo.

Matteo Nanni

Matteo ha lavorato principalmente sul front-end, impostando l'interfaccia grafica e coordinando tutte le azioni eseguite dall'utente su di essa, sulla base anche delle specifiche fornite dallo sviluppo del back-end. Gran parte dello sviluppo è consistito nell'apprendere da zero le conoscenze necessarie per lo sviluppo di un'applicazione in Typescript React. Ha contribuito anche nella scrittura dei documenti e nella realizzazione dei relativi diagrammi, specialmente riguardanti il D2.

Ruggero Antonacci - Team Leader
Ruggero si è impegnato nella stesura dei documenti, ha collaborato alla definizione dei requisiti e alla creazione dei diagrammi e ha partecipato alla revisione di tutti i documenti. I documenti cui ha dedicato più tempo sono stati il D1 e il D5.

Sasha Businaro
Sasha, in quanto ideatore del progetto, si è occupato di definire l'architettura generale dell'applicazione, dichiarando quali servizi esterni e interni utilizzare per lo sviluppo.
Si è occupato della parte di sviluppo del back-end del progetto, lavorando anche sulla documentazione delle API e del testing. Ha inoltre coordinato la stesura del D4, in quanto responsabile dei contenuti relativi al back-end e ha partecipato alla scrittura dei documenti, in particolar modo del D3.
Si è occupato inoltre del deployment dell'applicazione.

2.2 Distribuzione del carico di lavoro

In questa sezione dichiariamo come è avvenuta la distribuzione del carico orario del lavoro.

	D1	**D2**	**D3**	**D4**	**D5**	**TOTALE**
Ruggero Antonacci	25	20	5	2	30	82
Matteo Nanni	22	19	24	145	2	212
Sasha Businaro	23	18	26	150	2	219
TOTALE	70	57	55	297	34	513

Ora descriviamo quali lavori sono stati svolti dai vari componenti del team, divisi per i vari deliverable.

Deliverable 1

Sasha si è occupato di fornire un primo mockup grafico dell'applicazione nel design del front-end e di definire i servizi esterni nel design del back-end.

Tutti i componenti hanno contribuito a redigere i requisiti funzionali e non funzionali in modo completo, sulla base dell'idea iniziale di applicazione di Sasha.

Deliverable 2

Sasha si è occupato di redigere la tabella dei requisiti non funzionali e del disegno del diagramma delle componenti .

Ruggero e Matteo si sono occupati degli use case diagram per i requisiti funzionali.

Deliverable 3

Ruggero ha avuto una parte limitata nella definizione delle classi e si è occupato della revisione del documento.

Deliverable 4

Sasha ha sviluppato la totalità del backend, ha definito la relativa documentazione delle API e ha implementato un meccanismo di testing.

Si è occupato anche del deployment dell'applicazione, permettendo di raggiungerla da internet.

Matteo ha sviluppato la totalità del frontend, definendo l'interfaccia grafica e implementando le componenti responsabili di interrogare le API descritte nel backend.

Ha inoltre disegnato e descritto lo *User flow* relativo al progetto.

Ruggero si è limitato alla revisione del documento.

Deliverable 5

Ruggero ha riassunto e commentato le attività seminariali e si è occupato di redigere una sintesi relativa al lavoro svolto dal team durante tutto il periodo, dichiarando quali aspetti relativi all'ingegneria del software sono stati adottati dal team.

Ogni componente ha contribuito a dichiarare come ha svolto il proprio lavoro, chiarendo le ore spese e le attività svolte. Abbiamo discusso poi collettivamente riguardo alle criticità incontrate e abbiamo rilasciato una autovalutazione.

3. CRITICITÀ

Relativamente allo sviluppo del codice dell'applicazione abbiamo avuto inizialmente delle incertezze, principalmente dovute al fatto che nessuno di noi aveva mai lavorato su un'applicazione web, riuscendo però alla fine dello sviluppo a ottenere dimestichezza con i mezzi utilizzati.
All'inizio sono stati fatti alcuni tentativi per individuare le componenti corrette, ad esempio, per permettere l'autenticazione alla piattaforma, ma dopo diversi tentativi e dopo esserci documentati più approfonditamente in modo autonomo siamo riusciti ad avviarci.
Un'altra criticità avvertita soprattutto all'inizio è stata la comunicazione tra i membri e la coordinazione del lavoro. Nel corso del lavoro siamo riusciti per prova ed errore ad arrivare a una coordinazione più fluida e funzionale e a una comunicazione più efficace, soprattutto durante lo sviluppo della web app.

4. AUTOVALUTAZIONE

in base al lavoro svolto e al risultato ottenuto, ci sentiamo di assegnare a noi stessi la seguente valutazione:

Ruggero Antonacci	28
Matteo Nanni	30
Sasha Businaro	30L