

Codemaster
Applicazioni e Servizi Web

Alex Testa - 0001125408 {alex.testa@studio.unibo.it}
Riccardo Rambaldì - matricola {riccardo.rambaldi4@studio.unibo.it}

3 Luglio 2025

Indice

1	Introduzione	3
2	Requisiti	5
2.1	Requisiti utente utilizzatore	5
2.2	Requisiti utente amministratore	6
2.3	Requisiti non funzionali	6
3	Design	8
3.1	Questionario	8
3.2	Analisi dei competitor	11
3.3	Personas	12
3.3.1	User-Personas	12
3.3.2	Proto-Personas	13
3.4	Design System	15
3.5	Mock-Up	18
3.6	Back-end	27
4	Tecnologie	30
4.0.1	Stack MEVN	30
4.1	TypeScript	31
4.2	Fp-ts	32
4.3	Helmet, Jwt, Bycript	32
4.4	TailwindCSS	33
4.5	Lint, Prettier	34
4.6	Vitepress	34
4.7	Mongoose	35
4.8	Cookie Parser	35
4.9	Animate on scroll library (AOS)	36
4.10	Sweetalert2	36
4.11	Docker	36
5	Codice	38
5.1	Back-end	38
5.2	Front-end	39

6	Test	42
6.1	Test del codice	42
6.2	Test visivi	43
7	Deployment	45
8	Conclusioni	47

Capitolo 1

Introduzione

Nel contesto attuale dello sviluppo software, la crescente complessità dei sistemi, la necessità di individuare e correggere bug e la difficoltà nel progettare algoritmi efficienti rendono la programmazione una disciplina sempre più esigente. In questo scenario, emerge il bisogno di strumenti e piattaforme che favoriscano il mantenimento e il potenziamento delle competenze di sviluppo, sia per i programmati esperti che per i principianti.

Da questa esigenza nasce l'idea di una piattaforma educativa pensata per stimolare e consolidare le abilità di programmazione. L'obiettivo è duplice: offrire un ambiente in cui gli utenti possano mettersi alla prova risolvendo problemi (le cosiddette codequest), ma anche permettere loro di crearne di nuovi, condividendo sfide personalizzate con la community. In questo modo, apprendimento e divertimento si uniscono, favorendo un miglioramento continuo attraverso la pratica e la competizione costruttiva.

CodeMaster è una piattaforma interattiva pensata per favorire l'apprendimento e il miglioramento continuo delle competenze di programmazione e di problem-solving. Il suo funzionamento si basa sulla pubblicazione e risoluzione di sfide informatiche, chiamate codequest, che gli utenti possono affrontare direttamente online.

Le soluzioni possono essere scritte in diversi linguaggi – tra cui Java, Kotlin e Scala – senza la necessità di installare alcun software aggiuntivo: tutto avviene all'interno dell'ambiente web, in modo semplice e immediato.

L'obiettivo di CodeMaster è quello di offrire un contesto educativo, dinamico e collaborativo, in cui programmati alle prime armi possano imparare osservando approcci diversi allo stesso problema, mentre i più esperti possono contribuire condividendo soluzioni, commenti e sfide personalizzate.

Inoltre, la piattaforma si propone come uno strumento utile anche per gli insegnanti, che possono usarla per assegnare esercizi, monitorare i progressi degli studenti e stimolare la discussione attraverso il confronto tra strategie diverse. L'interazione tra utenti, la possibilità di commentare le soluzioni altrui e l'esplorazione di stili di programmazione differenti arricchiscono l'esperienza complessiva, trasformando l'apprendimento in un processo attivo e partecipativo.

Capitolo 2

Requisiti

In questa sezione si intende chiarire che tutti i requisiti elencati di seguito, risultano pienamente soddisfatti dal software sviluppato. Rispetto alla proposta iniziale, non vi sono state modifiche, eliminazioni o variazioni sostanziali nei requisiti funzionali o non funzionali.

I requisiti verranno presentati in due categorie distinte, corrispondenti a due tipologie di utenti:

- **Utente utilizzatore:** rappresenta l'utente finale, ovvero una persona che accede e utilizza la piattaforma in modo ordinario, secondo le modalità previste. Questa figura è ispirata alle user personas che verranno descritte nei capitoli successivi.
- **Utente amministratore:** si tratta di un utente interno al sistema che dispone di privilegi elevati, necessari per la gestione, la moderazione e il mantenimento della piattaforma.

2.1 Requisiti utente utilizzatore

- L'utente deve registrarsi e autenticarsi.
- L'utente può effettuare il logout.
- L'utente può visualizzare il proprio profilo e modificarne, l'immagine (selezionando tra quelle presenti sulla piattaforma), la bio, i linguaggi di programmazione preferiti, il proprio curriculum.
- L'utente può creare CodeQuest e specificare quali sono i linguaggi consoni alla soluzione.
- Durante la creazione di una CodeQuest, l'utente deve specificare quali sono i test coerenti con la soluzione.

- L’utente può modificare o rimuovere CodeQuest create in precedenza.
- L’utente può inviare la propria soluzione per le proprie CodeQuest o per quelle condivise da altri utenti.
- L’utente può visualizzare il livello raggiunto (es. Principiante, Novizio, Esperto, ecc.).
- L’utente può visualizzare l’ultimo trofeo ricevuto.
- L’utente può modificare aspetti grafici come la modalità scura o la modalità chiara.
- L’utente può filtrare le CodeQuest in base al livello di difficoltà.
- L’utente può visualizzare se le CodeQuest che ha risolto.
- L’utente può condividere una CodeQuest.

2.2 Requisiti utente amministratore

- L’amministratore deve autenticarsi.
- L’amministratore non può registrarsi ma sarà un utente interno al sistema.
- L’amministratore può bannare gli utenti che non rispettano le norme della community.
- L’amministratore può eliminare CodeQuest che non rispettano le linee guida o problemi non risolvibili attualmente.

2.3 Requisiti non funzionali

- Il sistema deve garantire un livello di sicurezza, non devono essere salvati dati sensibili in un formato leggibile.
- Il sistema deve proteggersi da attacchi agli headers delle richieste.
- Il sistema deve avere un contrasto cromatico che rispetti il livello WCAG AA.
- Il sistema deve essere sempre in grado di fornire feedback all’utente.
- Il sistema deve essere compatibile con i seguenti browser:
 - Firefox
 - Chrome
 - Safari
- Il sistema deve sfruttare le opportunità relative al competitor LeetCode.

- Il sistema deve garantire un'interfaccia responsive.
- Il sistema deve essere reattivo.
- Il sistema deve essere di facile utilizzo anche per utenti novizi al mondo della programmazione.
- Il sistema deve essere facile da installare.
- Lo sviluppo dell'interfaccia si baserà sull'approccio mobile-first.

Capitolo 3

Design

Il design di CodeMaster si basa sui principi dello Human-Centred Design, un approccio che mette al centro del processo progettuale l'utente finale, piuttosto che il punto di vista del progettista. Questo metodo ha permesso di sviluppare un'interfaccia e un'esperienza d'uso realmente orientate ai bisogni, alle aspettative e ai comportamenti concreti delle persone che utilizzeranno la piattaforma.

L'intero processo è stato guidato da una metodologia agile e iterativa, che ha favorito l'evoluzione progressiva del progetto attraverso cicli di test, analisi e miglioramento continuo. Il punto di partenza è stato la somministrazione di un questionario esplorativo, utile per identificare i reali bisogni degli utenti target. Le informazioni raccolte hanno poi guidato la progettazione dei primi mock-up ad alta fedeltà, attraverso cui abbiamo potuto visualizzare e testare l'aspetto e il funzionamento dell'interfaccia ancora prima della fase di sviluppo.

Questo approccio, flessibile e centrato sull'utente, ha garantito che ogni scelta progettuale fosse motivata da dati reali e validata passo dopo passo, fino alla realizzazione finale dell'applicativo.

3.1 Questionario

Prima di avviare lo sviluppo del progetto, è stato somministrato un questionario con l'obiettivo di raccogliere informazioni utili per comprendere meglio le preferenze, le aspettative e le abitudini di un potenziale pubblico. Questo strumento si è rivelato fondamentale non solo per orientare le scelte stilistiche legate al design e alla realizzazione delle interfacce utente, ma anche per gettare le basi per una corretta definizione dei profili utente.

In particolare, i dati raccolti ci hanno permesso di costruire delle proto-personas, ovvero rappresentazioni ipotetiche di utenti basate su intuizioni, osservazioni e

informazioni preliminari, piuttosto che su dati approfonditi e convalidati. A differenza delle user-personas tradizionali, che derivano da ricerche approfondite su utenti reali, le proto-personas sono figure immaginarie, ma costruite in modo credibile, con tratti e comportamenti plausibili che riflettono comunque bisogni e caratteristiche di utenti potenziali.

Queste rappresentazioni ci hanno guidato nelle prime fasi del design, aiutandoci a mantenere il focus sull'esperienza dell'utente finale e a sviluppare un'interfaccia che fosse il più possibile coerente con le sue esigenze.

Il questionario è accessibile tramite questo link

Tra le domande presenti nel questionario, particolare attenzione è stata riservata a quella relativa ai browser più utilizzati dagli utenti. Questa informazione si è rivelata fondamentale durante la fase di sviluppo, poiché ha permesso di identificare i browser su cui concentrare maggiormente i test di compatibilità e ottimizzazione, garantendo così un'esperienza d'uso fluida e coerente per la maggior parte degli utenti, come mostrato in Figura 3.1.

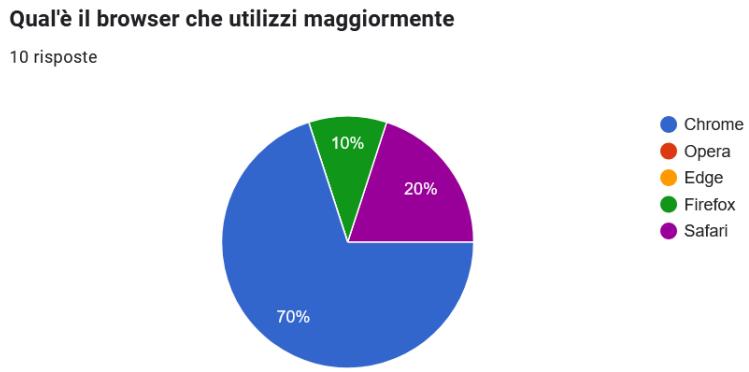


Figura 3.1: L'immagine mostra le risposte alla domanda "Qual'è il browser che utilizzi maggiormente?"

Un'altra domanda di grande interesse riguardava l'ambiente di sviluppo integrato (IDE) preferito dai programmatore. Conoscere gli strumenti più diffusi tra i potenziali utenti ha offerto preziosi spunti per la fase di progettazione dell'interfaccia: prendendo spunto dai design degli IDE più usati, ricreando un aspetto familiare per chi già li utilizza. Questo approccio è utile per attivare la memoria visiva dell'utente, facilitando l'apprendimento e l'interazione all'interno della piattaforma. Le risposte degli utenti sono mostrate in Figura 3.2

Quale IDE utilizzi maggiormente

10 risposte

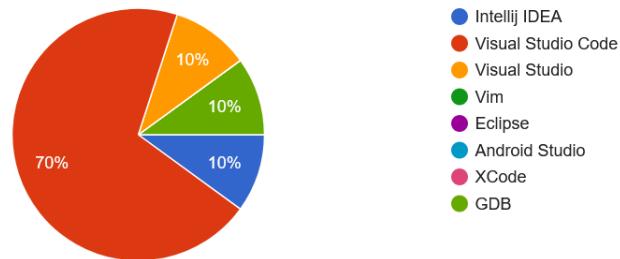


Figura 3.2: L'immagine mostra le risposte alla domanda "Quale IDE utilizzi maggiormente?"

Infine, una particolare attenzione è stata dedicata anche alla domanda relativa ai linguaggi di programmazione più utilizzati. Le risposte hanno fornito un quadro chiaro delle preferenze degli utenti, permettendo di selezionare i linguaggi di programmazione da inserire all'interno dell'applicativo, in modo da rispondere concretamente ai bisogni reali della community di riferimento. La Figura 3.3 mostra le risposte.

Quali sono i linguaggi di programmazione che preferisci

Copia grafico

10 risposte

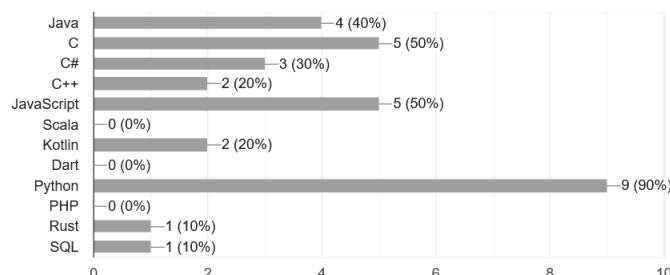


Figura 3.3: L'immagine mostra le risposte alla domanda "Quali sono i linguaggi di programmazione che preferisci?"

3.2 Analisi dei competitor

Un aspetto fondamentale all'interno del processo di progettazione è lo studio dei competitor, attività che ha permesso di analizzare punti di forza, limiti e opportunità nel contesto delle piattaforme dedicate all'apprendimento della programmazione.

In particolare, si è deciso di analizzare l'applicativo LeetCode[16], considerato uno dei principali riferimenti a livello globale nel settore. LeetCode è una piattaforma ampiamente riconosciuta per la qualità delle sfide proposte e per la sua capacità di simulare problemi reali di tipo algoritmico e tecnico, tanto da essere adottata da alcune delle aziende tecnologiche più importanti al mondo – come Google, Apple, Microsoft e molte altre – come strumento di supporto durante i processi di selezione del personale.

Questa osservazione ha permesso di comprendere meglio le aspettative e gli standard di riferimento per una piattaforma educativa orientata alla pratica. Pur riconoscendo l'indiscusso valore di LeetCode, si è colta l'occasione per differenziarsi, cercando di offrire un'esperienza più accessibile, orientata anche ai neofiti, e con una forte componente educativa e collaborativa, dove il focus non è solo sull'esecuzione del codice, ma anche sulla comprensione, il confronto tra approcci e la crescita della community.

La Figura 3.4 riporta l'analisi delle opportunità rispetto al competitor Leetcode.

Svantaggi	Opportunità	Possibili Frustrazioni
L'utente non può creare problemi	Dare la possibilità all'utente di creare problemi	Il processo potrebbe richiedere tanto tempo, in particolare nella fase di scrittura dei test.
L'utente non può debuggare il codice se non nella versione a pagamento	Dare la possibilità di debuggare il codice nella versione gratuita	Il processo di debug, se non guidato da numerosi feedback, potrebbe far disperdere l'utente all'interno del sistema
L'utente non può inserire informazioni utili come il cv	Dare la possibilità di inserire il proprio cv	Se l'utente sbaglia ad inserire il cv, potrebbe esporre un'informazione fittizia
L'interfaccia è minimal	Creare un'interfaccia accattivante, moderna, in stile flat design	Se la UI manca di UX, l'utente potrebbe non saper utilizzare l'applicativo
L'auto-completamento non è gratuito	Dare la possibilità di avere un auto-completamento gratuito	/

Figura 3.4: L'immagine mostra l'analisi del competitor Leetcode

3.3 Personas

Per comprendere meglio il tipo di utenza a cui si rivolge la piattaforma, è stata condotta un'indagine mirata basata su due profili rappresentativi: uno reale, l'altro ipotetico.

Da un lato, è stata analizzata l'esperienza e il comportamento di Elisa, una persona reale con background e interessi compatibili con l'obiettivo della piattaforma. Dall'altro, è stato costruito il profilo di Marco, una figura immaginaria ma plausibile, che rispecchia tratti comuni a un'ampia fascia di utenti potenziali.

Questo approccio, che unisce l'osservazione empirica con la progettazione orientata alle proto-personas, ha permesso di identificare bisogni concreti, aspettative e potenziali criticità dal punto di vista dell'utente. Di conseguenza, ha influenzato in modo significativo le scelte di design, usabilità e comunicazione, rendendo la piattaforma più vicina alle reali esigenze del pubblico a cui si rivolge.

3.3.1 User-Personas

Il profilo reale di Elisa, illustrato in Figura 3.5, ha giocato un ruolo fondamentale nella comprensione concreta delle esigenze, delle abitudini e delle aspettative di un utente reale che si approccia per la prima volta a una piattaforma come CodeMaster.

Nella fase iniziale di progettazione, molte delle funzionalità erano state definite sulla base delle nostre ipotesi e del nostro punto di vista da sviluppatori. Tuttavia, confrontandoci direttamente con Elisa e illustrandole le caratteristiche del sistema, ci siamo resi conto che alcune scelte erano in linea con le sue aspettative, mentre altre richiedevano un ripensamento o una maggiore attenzione all'esperienza utente.

Questo confronto diretto ci ha permesso di valutare criticamente il nostro approccio progettuale, passando da un punto di vista autoreferenziale a uno realmente centrato sull'utente, in linea con i principi dello Human-Centered Design.

Inoltre, Elisa non è stata solo un punto di riferimento teorico: rivestirà un ruolo chiave anche nella fase di user testing, contribuendo attivamente alla valutazione dell'usabilità dell'applicativo e alla raccolta di feedback qualitativi utili per eventuali miglioramenti futuri. La sua partecipazione come tester ci permetterà di validare le scelte progettuali e misurare concretamente l'efficacia dell'interfaccia e delle funzionalità proposte.

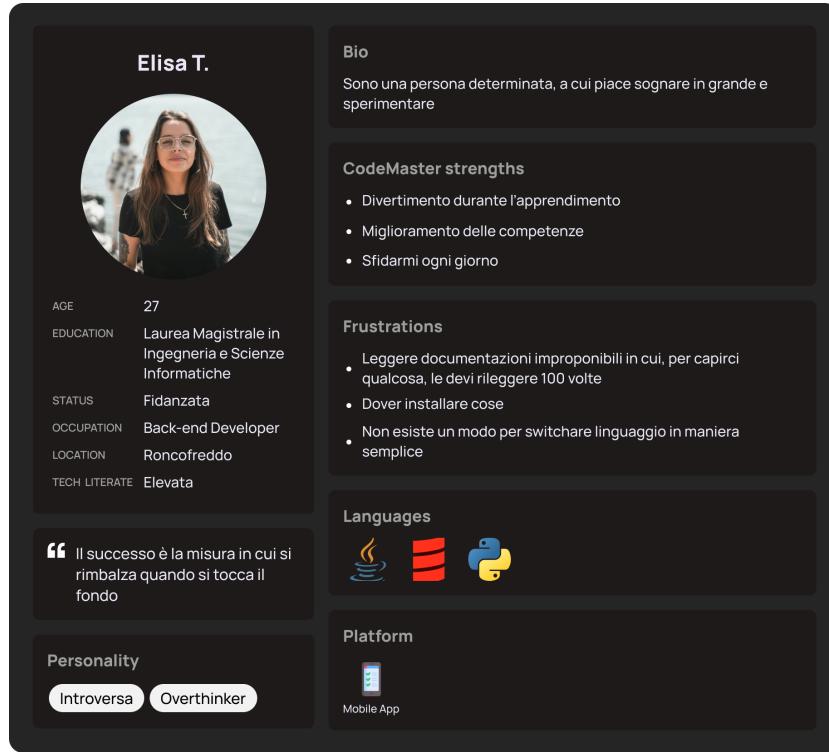


Figura 3.5: L'immagine mostra il profilo di Elisa

3.3.2 Proto-Personas

La seconda figura analizzata, illustrata in Figura 3.6, è una proto-persona di nome Marco. Sebbene si tratti di un profilo immaginario, è stato costruito con cura ispirandosi a caratteristiche e comportamenti comuni riscontrabili all'interno del nostro target di riferimento.

La creazione di Marco non ha lo scopo di descrivere una persona reale, bensì quello di rappresentare in modo sintetico e realistico un potenziale utente della piattaforma, dotato di esigenze, aspettative e obiettivi specifici. Questo processo di idealizzazione si è rivelato estremamente utile per guidare scelte progettuali più consapevoli, orientate non ai gusti soggettivi dei progettisti, ma ai bisogni effettivi dell'utenza che si vuole coinvolgere.

L'uso delle proto-personas, infatti, è una pratica consolidata nello User-Centered Design: permette di conservare la centralità dell'utente durante tutte le fasi dello sviluppo, fornendo una bussola decisionale ogni volta che si devono effettuare scelte di funzionalità, interazione o comunicazione.

Marco B.

AGE 54
EDUCATION Diploma Tecnico Industriale Specializzato In Informatica
STATUS Single
OCCUPATION Software Developer
LOCATION Riccione
TECH LITERATE Moderato

Volere è potere

Personality
Estroverso Thinker

Bio
Sono un marito con due bellissimi bambini. Amo la natura e mi dedico al giardinaggio. La mia più grande passione è l'informatica, ho imparato a scrivere il mio nome prima al computer e poi su carta

CodeMaster strengths

- Mi aspetto una piattaforma in grado di migliorare le mie skills nel tempo
- Mi piacerebbe risolvere problemi sempre più complessi
- Mi piacerebbe risolvere problemi presentati in colloqui importanti ad esempio per aziende come Google o Apple

Frustrations

- Odio non avere l'autocompletamento
- Odio navigare ore e ore su stackoverflow e non ottenere risultati
- Odio la velocità del mio computer, ci mette troppo tempo

Languages

Platform
 Website Mobile App

Figura 3.6: L'immagine mostra il profilo di Marco

Grazie a Marco, è stato possibile simulare scenari realistici di utilizzo, anticipare potenziali difficoltà o aspettative, e definire una direzione coerente per il design dell'interfaccia e delle dinamiche della piattaforma.

3.4 Design System

Un Design System è un insieme strutturato di regole, componenti visivi, linee guida e buone pratiche che guidano la progettazione e lo sviluppo dell'interfaccia utente in modo coerente e scalabile. Si tratta, in sostanza, di una "cassetta degli attrezzi" condivisa, pensata per garantire uniformità grafica e comportamentale in tutte le parti di un'applicazione, migliorando l'esperienza utente.

Per questo progetto, si è scelto di realizzare un Design System proprietario, prendendo ispirazione dallo stile flat design di Apple, noto per la sua pulizia visiva, l'uso equilibrato di spazi e colori, e un'interfaccia moderna, essenziale e funzionale. L'obiettivo era quello di creare un'interfaccia che trasmettesse immediatamente una sensazione di familiarità, fluidità e semplicità, quasi come un gioco, ma sempre mantenendo una forte coerenza e professionalità.

In parallelo, è stato definito un preciso lavoro di costruzione dell'identità visiva del brand CodeMaster. Uno degli elementi centrali di questa identità è l'uso del colore principale #6246EA, una tonalità di viola distintiva e moderna, scelta non solo per motivi estetici, ma soprattutto per il suo potere comunicativo e riconoscibile. Questo colore è stato applicato strategicamente in tutta l'interfaccia, dai pulsanti agli elementi interattivi, fino ai loghi e ai feedback visivi, con l'intento di rafforzare il legame tra l'utente e il brand.

L'utilizzo sistematico di questo colore contribuisce a creare una memoria visiva inconscia, facilitando il riconoscimento immediato dell'applicativo e promuovendo la fidelizzazione dell'utente. Ogni interazione diventa così parte di un'esperienza coerente, che istruisce l'utente a riconoscere, classificare e riutilizzare in modo intuitivo gli elementi della piattaforma.

La Figura 3.7 mostra il Design System realizzato per CodeMaster.

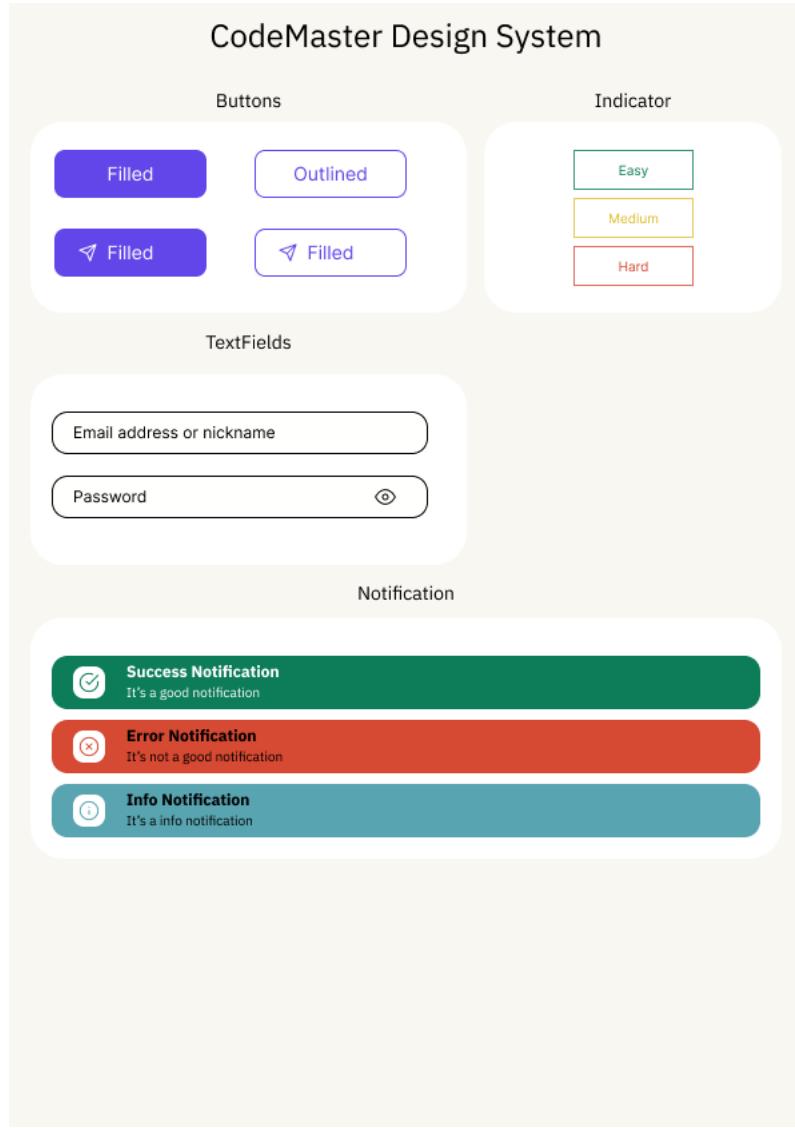
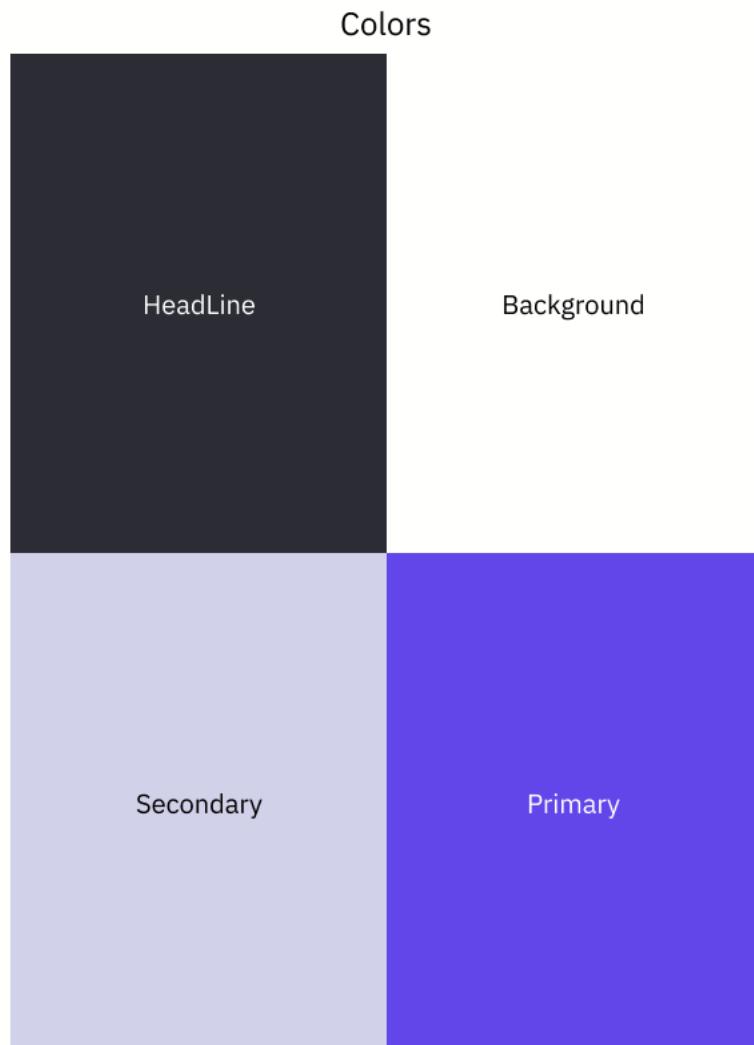


Figura 3.7: L'immagine mostra il Design System di CodeMaster

La Figura 3.8 mostra la scelta dei colori utilizzati.



<https://www.happyhues.co/palettes/6>

Figura 3.8: L'immagine mostra la palette utilizzata

3.5 Mock-Up

Durante la fase di progettazione visiva, si è scelto di realizzare i mockup in versione desktop, poiché dalle risposte raccolte tramite il questionario preliminare è emerso che la maggior parte degli utenti avrebbe fruito della piattaforma da dispositivi fissi o con schermi di grandi dimensioni. Per questo motivo, la priorità iniziale è stata data alla visualizzazione desktop, così da soddisfare fin da subito le esigenze emerse dall'analisi del target.

Tuttavia, nella fase di sviluppo vero e proprio, si è adottata un'impostazione mobile-first, una metodologia che prevede di progettare inizialmente l'interfaccia per dispositivi mobili, per poi estenderla progressivamente a schermi più ampi. Questo approccio ha permesso di garantire una buona responsività e accessibilità su tutti i dispositivi, mantenendo al tempo stesso un'esperienza utente coerente.

I mockup realizzati sono tutti ad alta fedeltà (high-fidelity) e includono dettagli precisi come la scelta dei colori, le dimensioni dei testi e la disposizione degli elementi, al fine di ridurre al minimo la distanza tra la fase di progettazione e quella di implementazione. Questa coerenza ha agevolato lo sviluppo dell'interfaccia, fornendo un riferimento chiaro e condiviso tra designer e sviluppatori.

Per la realizzazione dei mock-up è stato scelto Figma [12], un software moderno e versatile che si è rivelato particolarmente adatto alle esigenze del progetto. Figma consente di creare interfacce grafiche a qualsiasi livello di fedeltà, dai wireframe a bassa definizione fino a prototipi ad alta fedeltà, curati nei minimi dettagli. Uno degli aspetti più apprezzati dello strumento è la possibilità di realizzare prototipi interattivi, che simulano in modo realistico il comportamento dell'interfaccia. Questo ha rappresentato un grande vantaggio, poiché ha permesso di testare e valutare l'esperienza utente già nella fase di progettazione, riducendo in modo significativo il divario tra design e sviluppo.

Inoltre, grazie alla sua natura collaborativa e basata su cloud, Figma ha facilitato il lavoro in team, rendendo possibile la condivisione immediata dei mock-up, la raccolta di feedback e l'adattamento iterativo delle interfacce in modo rapido ed efficace.

La prima schermata progettata è stata quella di login e registrazione, considerata fondamentale per l'accesso alla piattaforma. L'obiettivo era offrire un'interazione semplice, chiara e immediata, sfruttando pattern visivi consolidati e familiari agli utenti, così da garantire un'esperienza fluida sin dal primo utilizzo.

La Figura 3.9 mostra i mockup realizzati, illustrando in dettaglio l'aspetto e la struttura delle principali schermate.

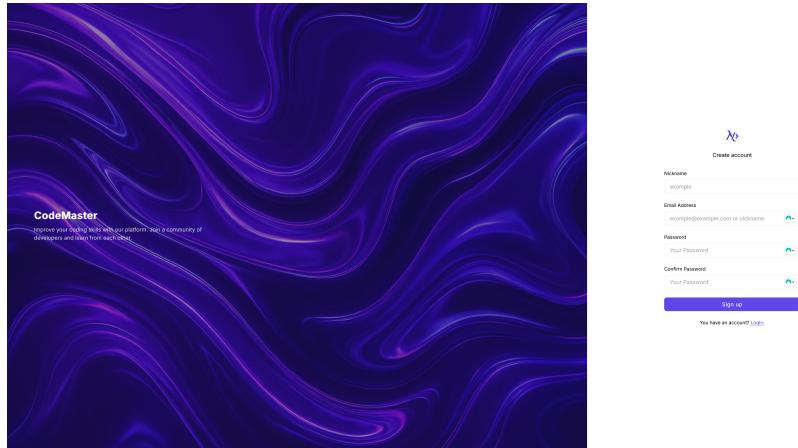


Figura 3.9: L’immagine mostra il mockup realizzato per la schermata di registrazione

La seconda schermata, su cui si è posta maggior attenzione nella fase di progettazione è quella della dashboard, ovvero l’interfaccia principale che l’utente visualizza subito dopo aver effettuato il login. Questa schermata rappresenta il vero e proprio centro nevralgico dell’applicativo, un punto di accesso rapido e intuitivo a tutte le funzionalità essenziali della piattaforma.

La dashboard è pensata per offrire una panoramica chiara e immediata sull’attività dell’utente. Da qui, è possibile visualizzare l’elenco aggiornato delle codequest disponibili, filtrate per difficoltà o stato di completamento, e monitorare i propri progressi personali, come il livello raggiunto, il numero di sfide completate e la distribuzione delle codequest risolte per ciascun livello di difficoltà.

Inoltre, l’utente può consultare i trofei più recenti conquistati, accedere rapidamente al proprio profilo personale per gestire impostazioni e preferenze, oppure entrare in contatto con il team di sviluppo. In caso di problemi tecnici o suggerimenti, è infatti disponibile una sezione dedicata alla segnalazione di bug o al supporto diretto, con l’obiettivo di mantenere attiva e partecipativa la comunicazione tra utenti e sviluppatori.

La dashboard, quindi, non è soltanto una schermata informativa, ma anche uno strumento dinamico che guida l’utente nell’esplorazione della piattaforma e nella gestione della propria esperienza all’interno dell’ambiente CodeMaster.

La Figura 3.10 mostra il mockup realizzato per la dashboard.

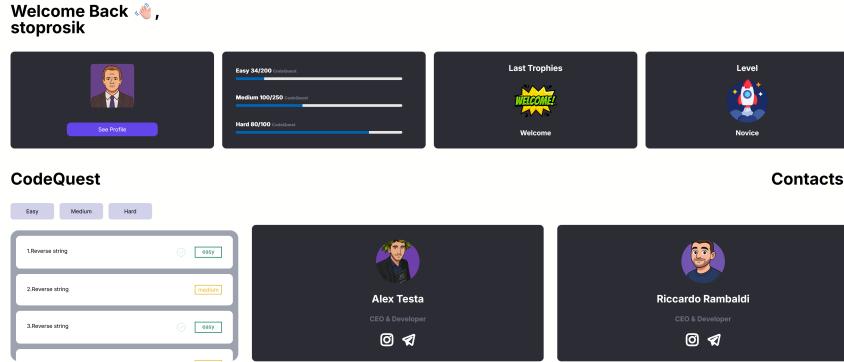


Figura 3.10: L’immagine mostra il mockup realizzato per la schermata dashboard.

Un’altra schermata fondamentale all’interno della piattaforma è quella dedicata al profilo utente, che rappresenta a tutti gli effetti l’identità dell’utente all’interno dell’ecosistema CodeMaster. Attraverso questa sezione, l’utente può gestire e personalizzare il proprio spazio, rendendolo più rappresentativo del proprio stile e delle proprie preferenze.

Nello specifico, è possibile modificare l’immagine del profilo scegliendo tra una serie di avatar predefiniti offerti dalla piattaforma, come mostra la Figura 3.12, pensati per mantenere coerenza grafica e semplicità d’uso. L’utente può inoltre aggiornare la propria biografia personale e il proprio cv, che può servire a raccontare sé stesso alla community, oppure evidenziare interessi e competenze.

Un’altra funzionalità centrale è la possibilità di selezionare i propri linguaggi di programmazione preferiti. L’utente può attingere a una vasta gamma di linguaggi disponibili, così da rendere il profilo più aderente alle sue reali competenze tecniche. Questo permette anche alla piattaforma di offrire contenuti e codequest più pertinenti alle abilità indicate.

Dalla stessa schermata, è possibile eseguire il logout in modo semplice e immediato, oppure visualizzare l’elenco delle codequest create dall’utente stesso. Per ogni sfida, è disponibile l’opzione di modifica o eliminazione, offrendo così pieno controllo sui contenuti generati e condivisi.

In sintesi, la schermata del profilo utente non è solo una raccolta di impostazioni, ma uno spazio personale dinamico e personalizzabile, che valorizza l’identità dell’utente e ne favorisce l’interazione con il resto della community.

La Figura 3.11 mostra la schermata di profilo utente.

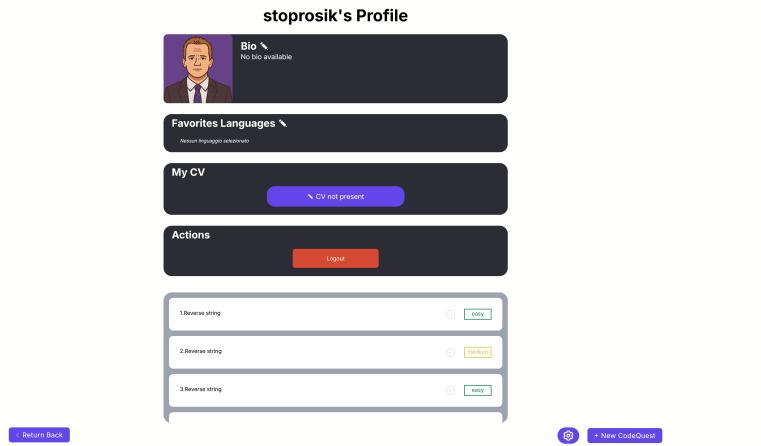


Figura 3.11: L'immagine mostra il mockup realizzato per la schermata di profilo utente

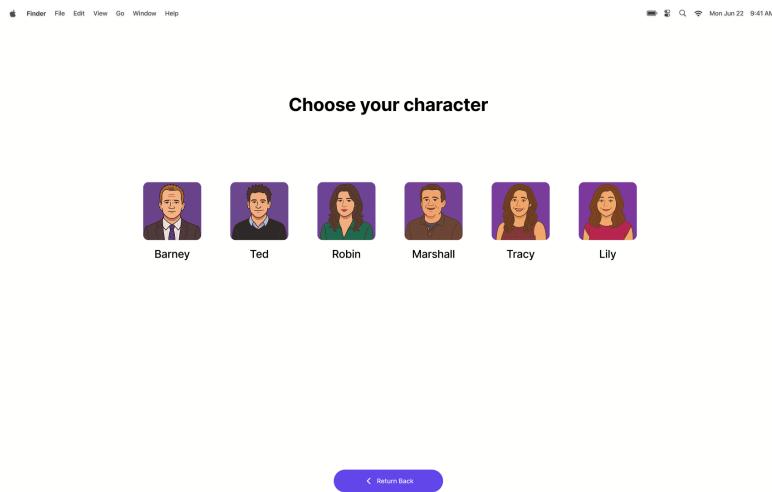


Figura 3.12: L'immagine mostra il mockup realizzato per la schermata di scelta di un avatar

Le ultime due schermate fondamentali della piattaforma sono quelle dedicate alla risoluzione e alla creazione delle codequest, che rappresentano il cuore dell'esperienza interattiva su CodeMaster. Come mostrato in Figura 3.13, la schermata di risoluzione di una codequest è stata progettata per offrire un ambiente il più possibile familiare, funzionale e coinvolgente. L'interfaccia si presenta divisa in due sezioni principali: sulla destra, l'utente può leggere la descrizione della challenge, formattata in Markdown per una maggiore chiarezza e leggibilità; sulla sinistra, è disponibile un editor di codice interattivo, ispirato all'aspetto e al comportamento di ambienti di sviluppo come Visual Studio Code, una scelta motivata dalle preferenze emerse durante la fase di raccolta dati tramite questionario.

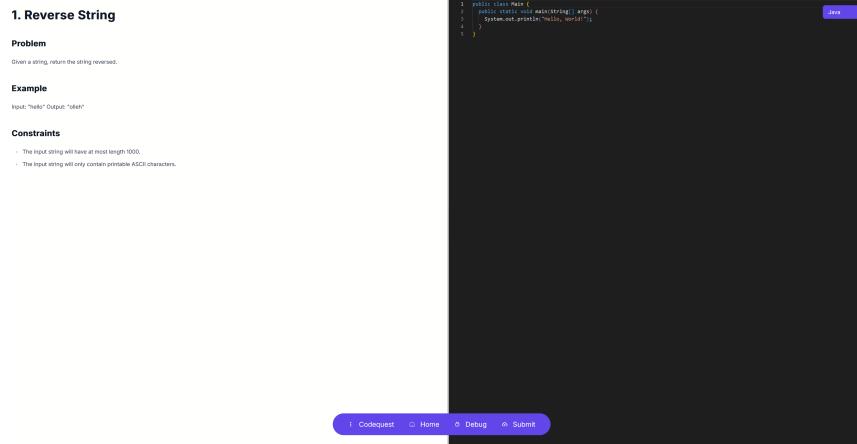


Figura 3.13: L'immagine mostra il mockup realizzato per la schermata di risoluzione di una codequest

L'editor consente all'utente di scrivere codice in diversi linguaggi, con supporto per funzionalità avanzate come l'autocompletamento del codice, solitamente disponibile solo nelle versioni a pagamento di piattaforme concorrenti come LeetCode. Oltre all'editor, sono presenti pulsanti di azione che permettono di effettuare un debug rapido del codice, evidenziando eventuali errori sintattici o lessicali prima di inviare la soluzione.

Una volta completata, la soluzione può essere sottomessa per essere valutata attraverso una serie di test automatici, che devono essere superati per considerare la codequest completata. Per rendere l'esperienza più fluida e ridurre possibili situazioni di frustrazione, ad esempio, in caso di blocco su una sfida particolarmente complessa, è stato introdotto anche un menu a tendina che consente di passare rapidamente da una codequest all'altra, come mostrato in Figura 3.14. Questo consente all'utente di esplorare più sfide senza sentirsi obbligato a completarne una prima di proseguire.

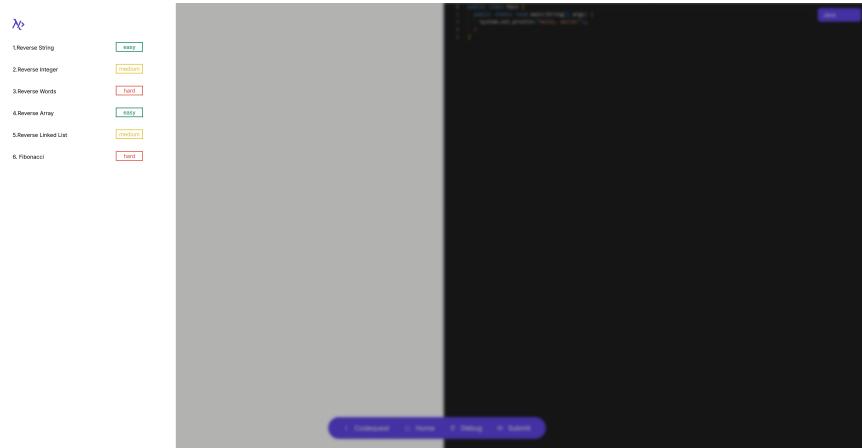


Figura 3.14: L'immagine mostra il mockup realizzato per il menù a tendina

L'altra schermata cruciale è quella dedicata alla creazione di una codequest, come mostrato in Figura 3.15. Anche in questo caso, l'interfaccia è stata pensata per essere chiara, guidata e intuitiva, con l'obiettivo di semplificare l'intero processo di creazione e ridurre al minimo gli errori. L'utente viene accompagnato passo dopo passo, dall'inserimento del titolo e della descrizione del problema fino alla definizione dei test.

Per facilitare ulteriormente la creazione delle challenge, è stato introdotto un sistema semplificato per la definizione dei test, basato su un linguaggio più naturale, in cui l'utente può specificare direttamente input e output attesi per ogni caso. Questo sistema genera automaticamente i test per tutti i linguaggi supportati dalla piattaforma, riducendo in modo significativo il tempo necessario per la configurazione tecnica e, di conseguenza, anche il livello di frustrazione legato a compiti ripetitivi e tecnici.

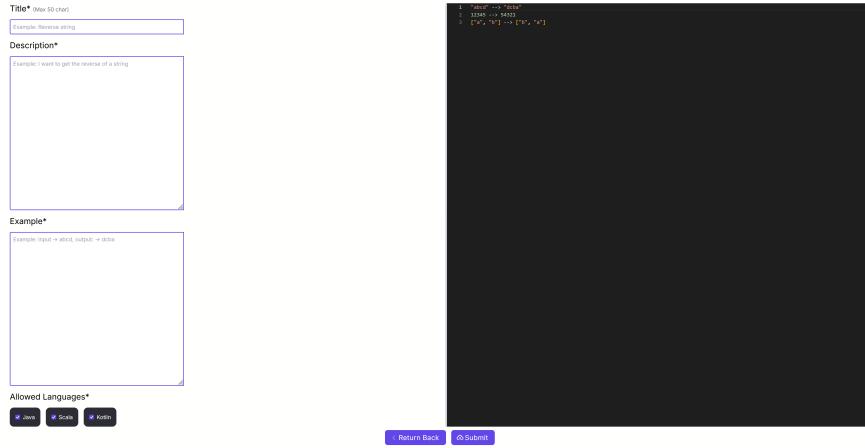


Figura 3.15: L’immagine mostra il mockup realizzato per la schermata di creazione di una CodeQuest

Nella fase di progettazione dell’interfaccia, si è scelto di offrire all’utente la possibilità di personalizzare l’aspetto visivo dell’applicativo, selezionando il tema preferito tra modalità chiara e scura. Questa funzionalità è accessibile all’interno della schermata delle impostazioni, pensata per raccogliere tutte le opzioni di configurazione dell’esperienza utente.

La decisione di introdurre questa possibilità nasce direttamente dai risultati del questionario somministrato in fase preliminare, da cui è emersa una netta preferenza per il tema scuro, soprattutto da parte degli utenti che utilizzano la piattaforma in ambienti di sviluppo prolungati o con scarsa illuminazione. Il tema scuro, infatti, viene spesso percepito come più confortevole per gli occhi, soprattutto durante sessioni di lavoro estese.

Tuttavia, per garantire la massima flessibilità e inclusività, è stato deciso di non forzare una scelta predefinita, ma di lasciare all’utente il controllo sull’aspetto dell’interfaccia, rendendo l’applicativo più accessibile e aderente alle preferenze individuali.

Figura 3.16 mostra un’anteprima della schermata dei settaggi, in cui è possibile selezionare il tema preferito in modo semplice e immediato.

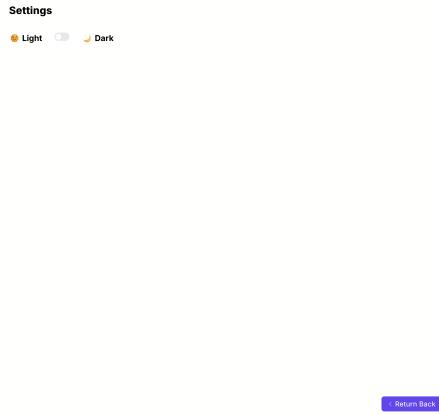


Figura 3.16: L'immagine mostra il mockup realizzato per la schermata dei settaggi

Infine, per rendere l'esperienza d'uso più coinvolgente e intuitiva, si è scelto di integrare numerosi feedback visivi all'interno dell'applicativo. Questi feedback svolgono un ruolo fondamentale nel creare una connessione chiara e continua tra l'utente e il sistema, aiutandolo a comprendere cosa sta accadendo in ogni momento e guidandolo nelle sue interazioni.

Ad esempio, durante operazioni che richiedono attesa, come l'elaborazione di una soluzione o il salvataggio di contenuti, viene mostrata una schermata di caricamento accompagnata da messaggi di conferma, che rassicurano l'utente sull'esito positivo dell'azione intrapresa. Questo tipo di interazione è mostrato in Figura 3.17, dove si può osservare un messaggio di successo a seguito di un'azione completata correttamente.

I feedback visivi sono inoltre essenziali per la prevenzione degli errori. In Figura 3.18, ad esempio, viene mostrato un modale di conferma che compare nel momento in cui l'utente tenta di lasciare una pagina senza aver salvato il proprio lavoro. Il sistema, in questo caso, chiede se si desidera realmente uscire (perdendo le modifiche) oppure rimanere sulla pagina per continuare. Questo semplice accorgimento permette di evitare perdite accidentali di dati e ridurre la frustrazione, offrendo all'utente un maggiore senso di controllo.

Per aumentare il coinvolgimento e motivare l'utente nel lungo periodo, si è scelto inoltre di introdurre una componente di gratificazione visiva, tramite l'uso di trofei. Ogni volta che l'utente raggiunge un obiettivo significativo, ad esempio il completamento di un certo numero di codequest o il superamento di sfide particolarmente complesse, viene assegnato un trofeo visibile nella propria area

personale. Questa dinamica, illustrata in Figura 3.19, contribuisce a rafforzare il senso di progresso e a incentivare la partecipazione attiva, trasformando l'apprendimento in un'esperienza più stimolante e appagante.



Figura 3.17: L'immagine mostra il mockup realizzato per un feedback visivo

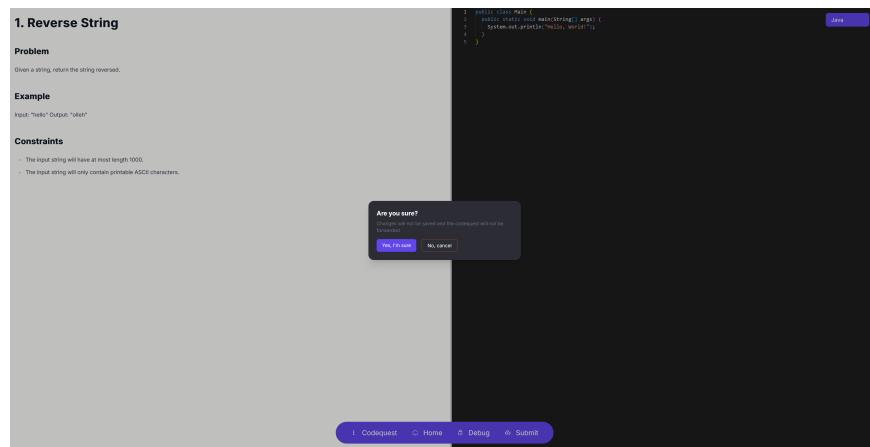


Figura 3.18: L'immagine mostra il mockup realizzato per un modale

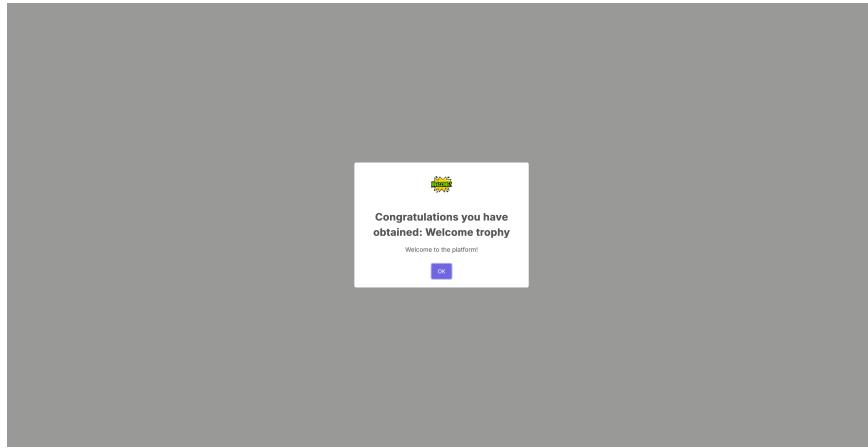


Figura 3.19: L'immagine mostra il mockup realizzato per un trofeo ottenuto

3.6 Back-end

In questo progetto, il back-end verrà presentato solo a livello introduttivo, senza entrare nel dettaglio tecnico, poiché la sua trattazione approfondita rientra nell'ambito di altre discipline del percorso di studi. Tuttavia, è importante sottolineare che l'architettura implementata è basata su un approccio a micro-servizi, una scelta moderna che garantisce modularità, scalabilità e facilità di manutenzione.

Il sistema si compone di diversi microservizi indipendenti, ciascuno con una responsabilità ben definita:

- **Authentication-service:** gestisce i meccanismi di registrazione e autenticazione degli utenti, consentendo loro di accedere in modo sicuro alla piattaforma.
- **User-service:** si occupa della gestione dei dati utente, come la biografia, l'immagine del profilo, i trofei ottenuti e il livello raggiunto. Questo microservizio funge da archivio centrale per tutte le informazioni personali visibili nella sezione profilo.
- **Codequest-service:** è il microservizio incaricato della creazione e gestione delle codequest, ovvero le sfide di programmazione proposte e risolte dagli utenti.
- **Solution-service:** si occupa della validazione delle soluzioni inviate, eseguendo i test automatici associati a ciascuna codequest per verificarne la correttezza e restituire un feedback immediato all'utente.

- **API-gateway-service**: rappresenta il punto di ingresso principale per tutte le richieste del sistema. Questo servizio sfrutta NGINX [20] per fungere da proxy inverso, reindirizzando il traffico in modo efficiente verso i microservizi corretti. Inoltre, è progettato per gestire un elevato carico di richieste, contribuendo alla stabilità e alla reattività complessiva dell'applicativo.
- **CodeGenerator-Service**: si occupa delle generazione dei test per la risoluzione di una Codequest e del template del codice di partenza.

L'adozione di questa struttura distribuita consente a ogni componente del sistema di evolvere in modo indipendente, migliorando la manutenibilità del codice e favorendo una gestione più agile dello sviluppo.

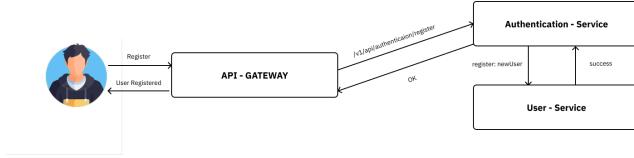


Figura 3.20: L'immagine mostra il sistema di gestione di una richiesta di registrazione sulla piattaforma

Figura 3.20 illustra in modo semplificato il flusso di registrazione di un nuovo utente all'interno dell'architettura basata su microservizi di CodeMaster. Il processo inizia quando un utente invia la richiesta di registrazione attraverso l'interfaccia dell'applicativo. Questa richiesta viene inoltrata all'API Gateway, che agisce come punto di ingresso centralizzato e si occupa di smistare il traffico

verso il microservizio appropriato.

In questo caso, il Gateway reindirizza la richiesta all'authentication-service, responsabile della gestione delle credenziali e della creazione dell'account. Una volta completata correttamente la fase di registrazione, ovvero dopo aver verificato la validità dei dati e creato l'identità digitale dell'utente, il sistema procede a notificare l'avvenuta registrazione ad altri microservizi che ne devono essere a conoscenza.

In particolare, viene inviato un evento al user-service, tramite il message broker RabbitMQ [26], una tecnologia di comunicazione asincrona che consente ai microservizi di comunicare in modo decentrato ed efficiente. Questo passaggio è fondamentale per mantenere la consistenza dei dati tra i servizi, assicurandosi che anche il database degli utenti venga aggiornato con le informazioni necessarie, come la creazione della biografia di default, l'immagine di profilo iniziale o altri attributi personalizzabili.

Al termine dell'intero processo, se tutte le operazioni vanno a buon fine, l'utente riceve un feedback visivo immediato, che lo informa dell'avvenuta registrazione con successo. Questo garantisce trasparenza nel funzionamento del sistema e migliora la qualità dell'esperienza utente.

Capitolo 4

Tecnologie

Di seguito sono elencate le tecnologie principali utilizzate nella realizzazione dei microservizi che compongono l’architettura di CodeMaster. Queste tecnologie sono state impiegate in modo uniforme nella maggior parte dei servizi, ad eccezione del Solution-service, sviluppato separatamente utilizzando il linguaggio Kotlin, per esigenze specifiche legate all’esecuzione e validazione del codice.

Tutti gli altri microservizi condividono una base tecnologica comune, scelta con l’obiettivo di garantire coerenza nello sviluppo, facilitare la manutenzione e rendere più semplice l’integrazione tra le varie componenti del sistema. Le tecnologie adottate spaziano da framework backend consolidati a strumenti per la gestione della comunicazione asincrona e della sicurezza, riflettendo un approccio moderno, modulare e scalabile.

4.0.1 Stack MEVN

Per la realizzazione dell’architettura di CodeMaster si è scelto di adottare lo stack tecnologico MEVN, acronimo che fa riferimento a MongoDB [18], Express.js [11], Vue.js [32] e Node.js [21]. Questo stack rappresenta una soluzione moderna e largamente diffusa nello sviluppo di applicazioni web full-stack, in quanto unisce efficienza, scalabilità e coerenza tecnologica grazie all’utilizzo comune del linguaggio JavaScript sia lato client che lato server.

Per quanto riguarda la persistenza dei dati, si è scelto MongoDB, un database NoSQL orientato ai documenti, che permette di memorizzare informazioni in formato JSON-like (BSON). Questa tecnologia si adatta perfettamente a un’architettura a microservizi grazie alla sua natura distribuita, alla scalabilità orizzontale e alla flessibilità nello schema dei dati, che consente di evolvere facilmente la struttura informativa nel tempo, senza dover affrontare le rigidità tipiche dei database relazionali.

Per la parte server-side, ogni microservizio (ad eccezione del solution-service) utilizza Node.js, un runtime JavaScript basato sul motore V8 di Chrome, noto per la sua efficienza nella gestione delle operazioni asincrone e per il suo modello event-driven, che ben si adatta a sistemi ad alto carico e altamente concorrenti. All'interno di ciascun microservizio è stato utilizzato Express.js, un framework minimalista ma potente che semplifica la creazione di API RESTful, migliorando la leggibilità e la struttura del codice, grazie a una sintassi chiara e a un'organizzazione modulare delle route e dei middleware.

Lato client, l'interfaccia utente è stata sviluppata utilizzando Vue.js, un framework JavaScript progressivo pensato per la creazione di interfacce reactive e component-based. La filosofia a componenti riutilizzabili di Vue ha permesso di organizzare l'interfaccia in blocchi logici facilmente manutenibili, migliorando la coerenza del design e favorendo il riuso del codice in diverse parti dell'applicativo. Sfruttando le Single-Page-Application, si è ridotto il traffico di rete a discapito di una computazione client più lenta. Inoltre, la curva di apprendimento ridotta e la ricca documentazione hanno reso Vue una scelta ideale per un progetto agile e in continua evoluzione.

4.1 Typescript

Per lo sviluppo dell'applicativo si è scelto di utilizzare TypeScript [17], un superset di JavaScript che introduce il concetto di tipizzazione statica all'interno di un linguaggio originariamente dinamico come JavaScript. Questa scelta non è stata casuale: nasce dall'esigenza di rendere il codice più sicuro, robusto e manutenibile, riducendo in modo significativo la probabilità di incorrere in errori a tempo di esecuzione (runtime).

Una delle principali caratteristiche di TypeScript è la possibilità di intercettare gli errori già in fase di compilazione (compile-time), grazie alla dichiarazione esplicita dei tipi. Questo approccio consente di individuare bug e anomalie prima ancora di avviare l'applicazione, migliorando l'affidabilità complessiva del sistema e riducendo drasticamente il tempo dedicato al debugging (bug hunting) nelle fasi successive.

A livello teorico, la tipizzazione statica rappresenta una delle pratiche più raccomandate nei moderni paradigmi di programmazione, poiché rende il comportamento del codice più prevedibile e comprensibile, agevolando il lavoro in team e favorendo l'adozione di pratiche di sviluppo evolute come il refactoring sicuro, la programmazione orientata agli oggetti e la generazione automatica di documentazione tramite strumenti come IntelliSense.

All'interno del progetto, TypeScript è stato integrato perfettamente anche con Vue.js, grazie al supporto nativo offerto dalle versioni più recenti del framework.

Questo ha permesso di mantenere i benefici della componentizzazione di Vue, unendoli alla solidità garantita dalla tipizzazione statica di TypeScript. In altre parole, si è riusciti a unire la flessibilità dello sviluppo front-end con la sicurezza di un linguaggio tipato, migliorando la qualità del codice, la leggibilità e l'esperienza complessiva di sviluppo.

4.2 Fp-ts

All'interno del progetto si è deciso di integrare, insieme a TypeScript, la libreria fp-ts [4], uno strumento che consente di applicare in modo sistematico i principi della programmazione funzionale (FP) in un ambiente JavaScript/TypeScript.

L'adozione di fp-ts nasce dalla volontà di rendere il codice più prevedibile, sicuro e dichiarativo, sfruttando concetti fondamentali della programmazione funzionale come immutabilità, funzioni pure, composizione e gestione esplicita degli effetti collaterali. Questo approccio si discosta dal paradigma imperativo tradizionale, mettendo invece al centro il concetto di dati come valori immutabili e funzioni come unità fondamentali del comportamento.

Una delle caratteristiche più potenti di fp-ts è la sua capacità di rafforzare ulteriormente il sistema di tipi di TypeScript, attraverso l'uso di algebra dei tipi e costrutti come Option, Either, Task, IO, che permettono di modellare in modo esplicito situazioni di errore, asincronia, effetti o assenza di valore. Questo consente di spostare il controllo degli errori dal runtime alla fase di compilazione, rendendo l'applicativo più robusto e resistente ai comportamenti imprevisti.

Inoltre, l'utilizzo di fp-ts favorisce un codice più modulare e riutilizzabile, grazie alla composizione di funzioni tramite operatori come pipe o flow, migliorando al tempo stesso la leggibilità e la testabilità del progetto.

4.3 Helmet, Jwt, Bycript

Per garantire un maggiore livello di sicurezza all'interno del progetto, è stata adottata una serie di pratiche e librerie specifiche, pensate per proteggere l'applicativo da attacchi comuni e vulnerabilità legate al traffico HTTP e alla gestione delle credenziali utente.

In particolare, si è scelto di utilizzare Helmet [13], una libreria middleware per Node.js che si integra facilmente con Express e che ha il compito di rafforzare gli header HTTP delle richieste e risposte. Helmet agisce come una barriera protettiva contro una serie di attacchi noti, come Cross-Site Scripting (XSS) [23], Clickjacking [22] e MIME-sniffing [15], applicando una configurazione sicura e standardizzata degli header. In pratica, la libreria ripulisce e configura gli header HTTP in modo da ridurre la superficie di attacco, rendendo il server

più resistente a intrusioni e manipolazioni malevoli.

Per quanto riguarda l'autenticazione degli utenti, si è optato per l'utilizzo dei JSON Web Token (JWT) [1], una tecnologia ormai consolidata per la gestione sicura di sessioni utente. I JWT consentono di trasportare in modo sicuro informazioni tra client e server, sfruttando una firma digitale che ne garantisce l'integrità. Questo approccio ha permesso di evitare la memorizzazione delle sessioni lato server, rendendo il sistema più scalabile e moderno, soprattutto in un contesto basato su microservizi.

Infine, per proteggere le password degli utenti, è stata utilizzata la libreria bcrypt [2], uno standard di fatto nel campo della sicurezza informatica. Bcrypt permette di generare hash sicuri e non reversibili, utilizzando tecniche di salting automatico per ogni password. Questo significa che anche due password uguali tra utenti diversi produrranno hash differenti, rendendo estremamente difficile qualsiasi tentativo di attacco basato su dizionari o rainbow tables. L'utilizzo del salt casuale, unito al costante aggiornamento dell'algoritmo di hashing, contribuisce a garantire che le credenziali siano trattate in modo sicuro e conforme alle best practices del settore.

4.4 TailwindCSS

Per la gestione dello stile e della parte visuale dell'interfaccia, si è scelto di utilizzare Tailwind CSS [30], un framework utility-first pensato per semplificare e velocizzare la scrittura del codice CSS. A differenza di approcci più tradizionali, Tailwind consente di scrivere direttamente le classi CSS all'interno del markup HTML, rendendo immediata la relazione tra struttura e stile. Questo approccio ha permesso di ridurre drasticamente la quantità di CSS personalizzato scritto a mano, snellendo il processo di sviluppo front-end e aumentando la produttività.

Un aspetto distintivo di Tailwind è la sua flessibilità compositiva: a differenza di framework come Bootstrap [3], che offrono componenti già pronti ma spesso rigidi e difficili da personalizzare, Tailwind non impone alcun stile predefinito. Al contrario, offre un set ampio e modulare di utility CSS che possono essere combinate liberamente per costruire componenti completamente personalizzati. Questo approccio ha permesso di sviluppare un'interfaccia coerente e originale, mantenendo al contempo pieno controllo sul design e sull'identità visiva del progetto.

Inoltre, grazie all'integrazione con PostCSS [24], Tailwind consente di ottimizzare e processare automaticamente il CSS, ad esempio rimuovendo le classi inutilizzate durante la fase di build, migliorando così anche le performance lato client.

Un altro vantaggio importante riguarda la gestione della responsività. Tailwind include classi responsive già pronte all’uso, basate su un sistema di breakpoint ben definito, che ha reso semplice l’adattamento dell’interfaccia ai diversi dispositivi senza dover scrivere media query manuali. Questo ha reso possibile creare layout moderni e reattivi in modo rapido, mantenendo al contempo un alto livello di leggibilità e pulizia del codice.

4.5 Lint, Prettier

Per garantire una maggiore qualità del codice e migliorarne la leggibilità e manutenibilità, si è scelto di adottare due strumenti fondamentali nel moderno sviluppo JavaScript/TypeScript: ESLint e Prettier.

In particolare, ESLint [10] è stato utilizzato per applicare regole e convenzioni di codifica condivise, individuando automaticamente pratiche scorrette, potenziali errori logici e effetti collaterali indesiderati (side effects) nel codice. Questo ha permesso di mantenere uno stile coerente tra i vari moduli del progetto e di prevenire bug difficili da tracciare.

Parallelamente, è stato integrato Prettier [25], uno strumento di formattazione automatica del codice che si occupa di uniformare lo stile (indentazione, spaziature, punteggiatura, ecc.) secondo regole predefinite. A differenza di ESLint, che si concentra sul comportamento e sulla qualità sintattica, Prettier si occupa esclusivamente dell’aspetto estetico del codice.

4.6 Vitepress

Un ruolo centrale nello sviluppo e nella gestione della documentazione del progetto è stato svolto da VitePress [31], uno strumento progettato specificamente per generare siti statici documentativi ad alte prestazioni. La sua integrazione nativa con Vue.js ha rappresentato un enorme vantaggio, soprattutto per un progetto come CodeMaster in cui l’interfaccia è fortemente basata su componenti Vue.

Una delle funzionalità più apprezzate durante la fase di sviluppo è stata sicuramente l’hot reload: ogni modifica apportata ai file Markdown o Vue veniva riflessa in tempo reale, senza necessità di riavviare il server manualmente. Questo ha permesso di lavorare con estrema fluidità, riducendo drasticamente i tempi di attesa.

4.7 Mongoose

Per semplificare l’interazione con il database MongoDB, si è deciso di adottare Mongoose [19], una delle librerie ODM (Object Data Modeling) più diffuse e affidabili nell’ecosistema Node.js.

Mongoose fornisce un’interfaccia strutturata e altamente astratta per lavorare con MongoDB, permettendo di definire schemi fortemente tipizzati per i documenti e di gestire in modo semplice le relazioni tra dati, la validazione, i middleware e molto altro.

Grazie a Mongoose, è stato possibile evitare la scrittura manuale di molte operazioni ripetitive e complesse, come la gestione delle query, dei dati annidati o delle conversioni tra formati. Inoltre, la possibilità di modellare i dati in modo esplicito attraverso schemi ben definiti ha contribuito a rendere il codice più leggibile, robusto e manutenibile nel tempo.

In particolare, in un progetto strutturato su microservizi come CodeMaster, l’utilizzo di Mongoose ha rappresentato una soluzione strategica per standardizzare l’accesso al database, riducendo gli errori e migliorando la coerenza del codice tra i diversi servizi che interagiscono con MongoDB.

4.8 Cookie Parser

Durante il processo di registrazione e autenticazione degli utenti, è stata utilizzata la libreria cookie-parser [7], uno strumento semplice ma estremamente utile per la gestione dei cookie HTTP in ambiente Node.js.

L’integrazione di questa libreria ha permesso di semplificare l’accesso alle informazioni contenute nei cookie, facilitando il tracciamento delle sessioni utente e migliorando l’esperienza di login. In particolare, al momento della registrazione o dell’autenticazione, viene generato un cookie sicuro contenente un token (come un JWT), che viene automaticamente inviato al client.

Questo meccanismo consente all’utente di rimanere autenticato anche nelle sessioni successive, evitando la necessità di reinserire le credenziali ogni volta che accede alla piattaforma. In pratica, il cookie funge da ponte tra le sessioni, rendendo il flusso di login più fluido e trasparente, senza compromettere la sicurezza del sistema.

Inoltre, l’utilizzo di cookie-parser ha reso più intuitiva la gestione lato server di queste informazioni, grazie al parsing automatico dei cookie contenuti nelle richieste HTTP, che vengono resi disponibili in modo immediato all’interno del codice Express.

4.9 Animate on scroll library (AOS)

Per migliorare l’esperienza utente e rendere l’interfaccia grafica più fluida, moderna e gradevole alla vista, si è scelto di integrare la libreria AOS – Animate On Scroll [28].

AOS è una libreria leggera e altamente configurabile che consente di aggiungere animazioni professionali agli elementi HTML quando appaiono nel viewport durante lo scroll, senza dover scrivere codice CSS o JavaScript complesso.

Questo approccio ha permesso di evitare un’interfaccia statica o “macchinosa”, introducendo transizioni morbide e naturali in punti strategici del layout, come la comparsa di sezioni, card o buttoni.

L’uso di animazioni leggere e contestuali ha reso l’interazione utente più coinvolgente, aumentando la percezione di reattività e modernità dell’applicazione, senza compromettere le prestazioni o l’accessibilità.

4.10 SweetAlert2

Per migliorare la comunicazione tra sistema e utente, è stato fondamentale curare l’aspetto dei feedback visivi. In quest’ottica, si è deciso di integrare la libreria SweetAlert2 [6], uno strumento moderno e altamente personalizzabile per la creazione di modali e toast notifiche.

SweetAlert2 si è rivelato particolarmente efficace perché consente di integrare rapidamente popup informativi, di conferma o di errore, semplicemente utilizzando una sola riga di codice JavaScript, senza dover scrivere markup HTML o logica complessa.

Grazie alla sua sintassi intuitiva e all’elevata configurabilità, è stato possibile offrire agli utenti risposte immediate e visivamente accattivanti, migliorando la percezione generale di fluidità e cura del dettaglio.

4.11 Docker

Per la gestione del processo di deployment, si è scelto di adottare Docker [9], una delle tecnologie container più diffuse e affidabili nel panorama dello sviluppo moderno. La scelta non è stata casuale: Docker ha infatti offerto una soluzione elegante e scalabile per isolare e distribuire ogni singolo microservizio del progetto.

Ogni microservizio è stato ”dockerizzato”, ovvero incluso in un’immagine dedi-

cata, che contiene al suo interno tutte le dipendenze, configurazioni e ambienti necessari alla sua esecuzione. Questo approccio ha portato numerosi vantaggi:

- Gli utenti o valutatori del progetto non devono installare manualmente librerie, runtime o strumenti esterni per ogni singola parte del sistema.
- Ogni microservizio può essere eseguito in modo indipendente, semplificando lo sviluppo, il debug e la manutenzione.
- L'intero ecosistema può essere replicato su qualsiasi macchina, garantendo coerenza tra ambienti di sviluppo, testing e produzione.

Un ulteriore vantaggio di Docker è la sua ottimizzazione intelligente delle dipendenze: nel caso in cui più container condividano componenti simili (come versioni di Node.js o pacchetti comuni), Docker evita di duplicare inutilmente lo spazio occupato, gestendo le immagini in modo efficiente tramite un sistema di livelli (layers).

Questo approccio non solo riduce il tempo di download, ma migliora anche le prestazioni generali in fase di build e deploy, rendendo il sistema più veloce da avviare e più leggero da mantenere.

Capitolo 5

Codice

5.1 Back-end

Un aspetto particolarmente significativo dal punto di vista architettonale del back-end è stato l'impiego della libreria fp-ts [4], uno strumento avanzato pensato per portare i principi della programmazione funzionale all'interno del mondo TypeScript.

L'integrazione di fp-ts ha consentito di sviluppare un back-end più robusto, fortemente tipizzato e meno soggetto a errori imprevisti durante l'esecuzione. Uno dei vantaggi principali di questo approccio è proprio la possibilità di spostare l'individuazione degli errori dal runtime al compile-time, riducendo così drasticamente il rischio di bug in fase di produzione.

Tra le molte astrazioni messe a disposizione dalla libreria, una delle più utilizzate e potenti è il costrutto `Either<E, T>`. Questo tipo rappresenta un valore che può essere o un errore (Left) o un risultato valido (Right), forzando lo sviluppatore a gestire esplicitamente i casi d'errore. In altre parole, fp-ts promuove uno stile di codice dichiarativo, predicibile e facilmente testabile, in cui ogni ramo di esecuzione è trattato con la stessa importanza.

Questo paradigma, mutuato dalla programmazione funzionale pura, ha permesso di creare una logica server-side più solida, resiliente e mantenibile nel tempo, riducendo il rischio di comportamento inatteso in caso di input malformati, richieste non valide o errori di sistema.

L'adozione di fp-ts ha inoltre favorito la scrittura di codice più modulare e riutilizzabile, elevando la qualità generale del progetto e contribuendo a minimizzare il cosiddetto debito tecnico, che troppo spesso affligge il software sviluppato in ambienti dinamici.

La Figura 5.1 illustra un esempio concreto dell'utilizzo del costrutto `Either<E, T>`, evidenziando come questo strumento abbia migliorato la gestione degli errori e reso più sicuro il flusso logico delle operazioni backend.



```
export const saveAdvancedUser = async (
  userManager: UserManager
): Promise<Either<Error, UserManager>> =>
  pipe(
    tryCatch(
      async () => {
        const userDocument = toUserManagerModel(userManager)
        await userDocument.save()
        return userManager
      },
      (error) => (error instanceof Error ? error : new UnknownError())
    )
  )()
```

Figura 5.1: L'immagine mostra l'utilizzo del costrutto `Either<E, T>`

5.2 Front-end

L'architettura del front-end del progetto si fonda su due pilastri fondamentali che hanno contribuito in maniera determinante alla qualità, alla manutenzione e alla scalabilità del codice.

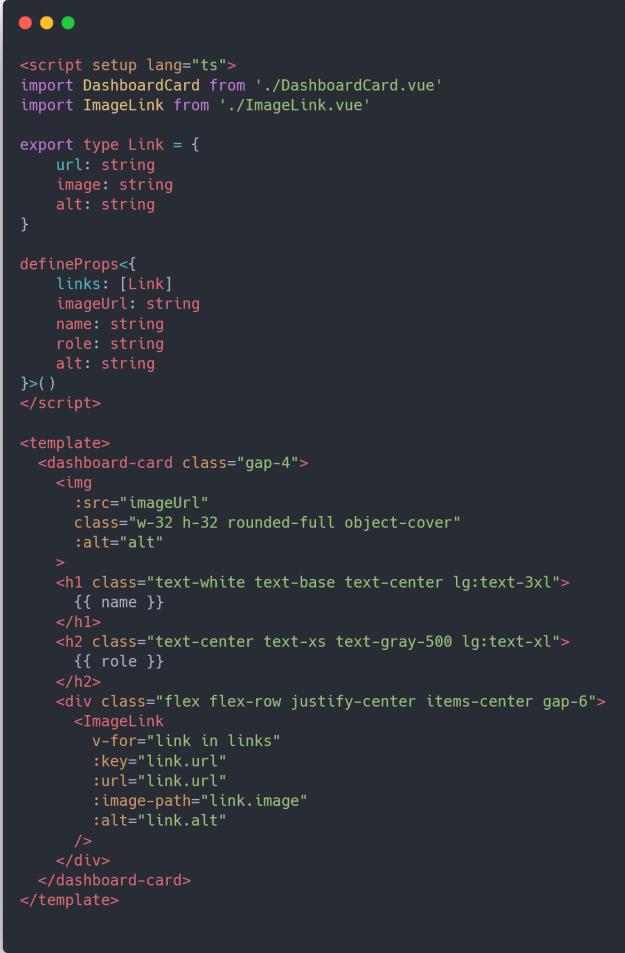
Il primo riguarda l'adozione di Tailwind CSS, un framework utility-first che ha rivoluzionato l'approccio alla scrittura dello stile. Grazie a Tailwind, è stato possibile evitare completamente la scrittura manuale di file CSS tradizionali: tutte le regole di stile sono state applicate direttamente nei file .vue tramite classi predefinite, rendendo l'interfaccia più semplice da costruire e modificare. Questo approccio ha consentito un notevole risparmio di tempo e una maggiore coerenza visiva, soprattutto nella gestione del layout e della responsività. Tuttavia, va sottolineato che l'integrazione dello stile direttamente nel markup HTML può rappresentare, per i più puristi del web development, una violazione del principio di separazione tra struttura e presentazione. In questo contesto, però, i vantaggi in termini di produttività e leggibilità hanno prevalso sulle considerazioni più accademiche.

Il secondo aspetto chiave è stato l'utilizzo congiunto di Vue.js e TypeScript.

TypeScript ha introdotto una tipizzazione statica, che ha migliorato in modo significativo la sicurezza del codice e ha permesso di individuare errori già in fase di compilazione, riducendo drasticamente il rischio di bug in fase di esecuzione. Questa integrazione ha anche favorito una maggiore chiarezza nella progettazione dei componenti, nella definizione dei props e nella gestione dello stato.

A completare il quadro, l'integrazione dello strumento ESLint [10] ha permesso di mantenere alta la qualità del codice, applicando regole di stile coerenti, prevenendo errori comuni e facilitando la collaborazione tra sviluppatori grazie a un codice più uniforme e leggibile.

La Figura 5.2 mostra un estratto del codice che evidenzia chiaramente l'utilizzo delle utility Tailwind e la tipizzazione TypeScript.



```
<script setup lang="ts">
import DashboardCard from './DashboardCard.vue'
import ImageLink from './ImageLink.vue'

export type Link = {
    url: string
    image: string
    alt: string
}

defineProps<{
    links: [Link]
    imageUrl: string
    name: string
    role: string
    alt: string
}>()

</script>

<template>
    <dashboard-card class="gap-4">
        
        <h1 class="text-white text-base text-center lg:text-3xl">
            {{ name }}
        </h1>
        <h2 class="text-center text-xs text-gray-500 lg:text-xl">
            {{ role }}
        </h2>
        <div class="flex flex-row justify-center items-center gap-6">
            <ImageLink
                v-for="link in links"
                :key="link.url"
                :url="link.url"
                :image-path="link.image"
                :alt="link.alt"
            />
        </div>
    </dashboard-card>
</template>
```

Figura 5.2: L'immagine mostra l'utilizzo di Vue + Typescript e le utility di Tailwind

Capitolo 6

Test

6.1 Test del codice

Per garantire l'affidabilità del codice e il corretto funzionamento delle funzionalità sviluppate, all'interno del progetto è stato adottato un approccio sistematico al testing, sfruttando strumenti moderni e consolidati nel panorama JavaScript.

In particolare, è stata utilizzata la libreria Jest [14], una delle soluzioni più popolari per il testing di unità e moduli in ambienti JavaScript e TypeScript. Jest è apprezzata per la sua configurazione minima, la velocità di esecuzione e il supporto nativo alle funzioni di mocking, essenziali per isolare e verificare il comportamento di singole porzioni di codice.

Per quanto riguarda il testing delle API REST esposte dai microservizi, si è scelto di affiancare a Jest la libreria Supertest [29]. Quest'ultima consente di effettuare richieste HTTP simulate verso il server e di verificarne le risposte in modo preciso, coprendo così anche le interazioni tra client e back-end.

Al fine di garantire che ogni microservizio fosse adeguatamente testato, è stato attivato il sistema di code coverage fornito da Jest tramite il comando:

```
Listing 6.1: Generazione della coverage tramite Jest  
jest . --coverage
```

Questo strumento permette di generare un report dettagliato sulla copertura del codice, evidenziando quali parti sono state testate e quali no. I risultati di questa analisi vengono successivamente inviati e visualizzati su Codecov [5], una piattaforma specializzata nella gestione e visualizzazione della coverage.

Grazie a Codecov, si è potuto monitorare con precisione eventuali lacune nella copertura dei test, identificando facilmente le aree del codice meno verificate e

pianificando interventi mirati per migliorare la qualità complessiva del software.

Figura 6.1 mostra lo stato attuale della code coverage dei microservizi, evidenziando visivamente le percentuali di copertura per ciascun modulo.



Figura 6.1: L'immagine mostra la codecoverage attuale di CodeMaster

6.2 Test visivi

Un aspetto fondamentale preso in considerazione durante la fase di design è stato quello dell'accessibilità visiva, con l'obiettivo di offrire un'esperienza inclusiva anche agli utenti con disabilità visive o difficoltà nella percezione dei colori.

Per garantire un buon livello di contrasto cromatico tra testi e sfondi, è stato utilizzato il tool Contrast Checker messo a disposizione da WebAIM [33], uno strumento riconosciuto per la verifica dell'accessibilità secondo gli standard internazionali. Attraverso questo strumento è stato possibile testare le combinazioni di colori adottate nel progetto, evitando accostamenti potenzialmente confusi o non leggibili. L'obiettivo minimo prefissato è stato il livello AA delle linee guida WCAG (Web Content Accessibility Guidelines), che rappresenta un buon compromesso tra accessibilità e flessibilità grafica, risultando adeguato per la maggior parte delle condizioni visive comuni, inclusi daltonismo e ipovisione.

Inoltre, per quanto riguarda la tipografia, è stato scelto il font Inter, un carattere moderno, neutro e ottimizzato per la lettura a schermo. La sua chiarezza, spaziatura equilibrata e leggibilità anche a dimensioni ridotte hanno contribuito a ridurre l'affaticamento visivo durante l'utilizzo prolungato dell'interfaccia, migliorando significativamente la qualità dell'esperienza utente.

Un ulteriore passo verso l'inclusività è stato fatto curando la semantica delle immagini: ogni immagine presente nel sistema è corredata di un attributo alt descrittivo, che consente agli utenti non vedenti o ipovedenti di accedere al contenuto visivo attraverso tecnologie assistive, come i lettori di schermo. Questo semplice accorgimento rende la piattaforma più accessibile, più empatica e più

rispettosa delle diversità degli utenti che la utilizzano.

Per garantire un’interfaccia davvero accessibile e fruibile su dispositivi di ogni tipo, è stato necessario validare attentamente il comportamento del sistema su diverse dimensioni di schermo, come smartphone, tablet e desktop.

A questo scopo è stata utilizzata Responsively App [27], un tool open source progettato per aiutare gli sviluppatori a testare e monitorare il layout responsive di una pagina web in tempo reale. Grazie a questo strumento, è stato possibile caricare simultaneamente la stessa interfaccia su molteplici viewport, visualizzando l’aspetto del sito così come apparirebbe su diversi dispositivi, senza doverli possedere fisicamente.

Questo tipo di test ha consentito di individuare facilmente eventuali problemi di scalabilità degli elementi, ridimensionamento del testo, overflow o posizionamento errato dei componenti, e ha permesso interventi rapidi e mirati durante la fase di sviluppo. In combinazione con l’approccio mobile-first adottato nel progetto, l’uso di Responsively App ha contribuito in modo significativo a realizzare una piattaforma davvero adattiva e consistente in termini di esperienza utente.

Capitolo 7

Deployment

La fase di deploy è stata trattata con particolare attenzione, in quanto pubblicare e avviare un sistema distribuito come CodeMaster non è un'operazione banale. Un'applicazione composta da più microservizi, ciascuno con dipendenze e comportamenti autonomi, richiede una gestione attenta della sua orchestrazione in ambienti di produzione o test.

Per rispondere a questa esigenza, si è scelto di contenere ogni microservizio all'interno di un'immagine Docker dedicata, rendendolo isolato, replicabile e facilmente distribuibile. L'intero sistema è stato poi orchestrato attraverso un unico file `docker-compose.yml`, che consente di avviare in maniera coordinata tutti i servizi coinvolti.

Questa configurazione ha permesso di automatizzare completamente il processo di deploy, riducendo la complessità a un singolo comando:

```
Listing 7.1: Orchestrazione con docker-compose file  
docker-compose up
```

Una volta eseguito, Docker si occupa autonomamente di:

- Costruire le immagini Docker per ciascun microservizio (backend, frontend, RabbitMQ, database, ecc.).
- Creare ed eseguire i container rispettando l'ordine e le dipendenze specificate.
- Garantire l'interoperabilità tra i servizi tramite una rete condivisa.

Un esempio concreto è rappresentato dal servizio di autenticazione, che dipende dal corretto funzionamento di RabbitMQ: grazie alla configurazione fornita in docker-compose, l'avvio del container authentication-service viene eseguito solo quando il container rabbitmq è attivo e pronto, evitando errori di connessione o comportamenti inattesi.

Inoltre, questo approccio ha reso estremamente semplice anche la fase di testing e valutazione del progetto da parte di terzi (come docenti o colleghi). È sufficiente accedere alla directory principale del progetto ed eseguire il comando docker-compose up per avviare l'intera infrastruttura. Una volta completato l'avvio, il frontend risulterà accessibile da un qualsiasi browser web all'indirizzo:

Listing 7.2: Indirizzo per accedere al front-end

```
http://localhost:5157
```

Questo approccio ha garantito non solo un'ottima scalabilità, ma anche portabilità, semplicità di setup e ripetibilità degli ambienti, tutte caratteristiche fondamentali per la gestione professionale di un'applicazione moderna.

Un ulteriore passo verso una distribuzione semplice ed efficace è stato quello di pubblicare ciascuna immagine Docker dei microservizi su Docker Hub [8], la piattaforma ufficiale per la condivisione di immagini containerizzate.

Questa scelta ha permesso di semplificare enormemente il processo di accesso e utilizzo del sistema, anche per chi non ha familiarità con l'ambiente di sviluppo o non vuole installare tutte le dipendenze localmente. Gli utenti possono infatti scaricare ed eseguire ogni microservizio con un semplice comando Docker, senza dover compilare manualmente le immagini o configurare il progetto da zero.

In questo modo, CodeMaster diventa immediatamente eseguibile su qualsiasi macchina dotata di Docker, rendendo il progetto altamente portabile, facilmente testabile e rapidamente distribuibile, anche in ambienti diversi (sviluppo, staging, produzione).

Capitolo 8

Conclusioni

Giunti al termine di questo percorso, possiamo affermare con grande soddisfazione di aver portato a compimento tutte le funzionalità essenziali previste nella fase di ideazione del progetto CodeMaster. L'intero sistema è stato costruito con attenzione, passione e rigore tecnico, e rappresenta per noi non solo un traguardo, ma anche una base solida su cui costruire ulteriori evoluzioni.

Durante lo sviluppo, abbiamo avuto l'opportunità di mettere in pratica numerose competenze apprese nel corso dell'anno, approfondendo al contempo tecnologie nuove e strumenti di lavoro reali, spesso utilizzati nel mondo professionale.

Tuttavia, come in ogni progetto software ben concepito, esistono margini di miglioramento e nuove sfide che ci stimolano a guardare al futuro con curiosità e spirito innovativo.

Tra le possibili evoluzioni, evidenziamo i seguenti ambiti:

- **Test generator per le codequest:** uno degli aspetti su cui desideriamo intervenire riguarda l'automazione nella generazione dei test durante la creazione di nuove sfide. L'idea è quella di progettare un compilatore custom in grado di interpretare il linguaggio semplificato con cui l'utente definisce input/output attesi, traducendolo automaticamente in test reali (ad esempio test scritti in JUnit per Java). Questo automatizzerebbe e standardizzerebbe il processo, riducendo l'errore umano e rendendo la piattaforma più scalabile.
- **Espansione del supporto linguistico:** attualmente il sistema supporta linguaggi basati sulla JVM (come Java, Kotlin e Scala). Un'evoluzione importante sarà quella di estendere la compatibilità ad altri linguaggi ampiamente utilizzati, come Python, Go, Rust, C, C++, C#, e altri. Questo

renderebbe la piattaforma più inclusiva e attrattiva per una community di sviluppatori ancora più ampia.

- **Funzionalità di gestione dell'account:** a livello di backend sono già state implementate API per funzionalità come il cambio della password e dell'indirizzo email. Tuttavia, queste funzionalità non sono ancora state esposte nel frontend. L'integrazione nel pannello utente rappresenta un piccolo ma importante passo verso un'interfaccia più completa.
- **Supporto alla concorrenza nei test:** infine, un'evoluzione significativa sarà quella di introdurre la possibilità di scrivere e valutare test concorrenti, utile in contesti dove la gestione di thread, risorse condivise o operazioni asincrone fa parte del problema. Questa funzionalità, attualmente assente, aprirebbe la strada a nuove tipologie di sfide ancora più realistiche e tecnicamente avanzate.

In conclusione, siamo davvero soddisfatti del lavoro svolto: CodeMaster rappresenta per noi non solo la somma di conoscenze tecniche, ma anche una dimostrazione concreta della nostra capacità di progettare, collaborare, adattarci e imparare lungo il percorso. Ci portiamo dietro nuove competenze, strumenti potenti, e soprattutto una maggiore consapevolezza di cosa significhi costruire software efficace, accessibile e mantenibile nel tempo.

Bibliografia

- [1] Auth0. Json web tokens - jwt.io.
- [2] bcrypt. bcrypt - a library to help you hash passwords.
- [3] Bootstrap. Bootstrap - the most popular html, css, and js library in the world.
- [4] Giulio Canti. fp-ts - functional programming in typescript.
- [5]Codecov. Codecov - code coverage reports for testing.
- [6] SweetAlert2 Contributors. Sweetalert2 - beautiful, responsive, customizable replacement for javascript's popup boxes.
- [7] cookie-parser. cookie-parser - parse cookie header and populate req.cookies.
- [8] Docker Hub. Docker hub - official container image library.
- [9] Docker Inc. Docker - empowering app development for developers.
- [10] ESLint. Eslint - pluggable javascript linter.
- [11] Express.js. Express - web framework for node.js.
- [12] Figma. Figma: The collaborative interface design tool.
- [13] Helmet.js. Helmet - help secure express apps with http headers.
- [14] Jest. Jest - delightful javascript testing framework.
- [15] KeyCDN. What is mime sniffing?
- [16] LeetCode. Leetcode: Online coding platform.
- [17] Microsoft. Typescript - javascript with syntax for types.
- [18] MongoDB, Inc. Mongodb - the developer data platform.
- [19] Mongoose. Mongoose - elegant mongodb object modeling for node.js.
- [20] NGINX. Nginx official website.

- [21] Node.js Foundation. Node.js - javascript runtime built on chrome's v8 engine.
- [22] OWASP Foundation. Clickjacking.
- [23] OWASP Foundation. Cross site scripting (xss).
- [24] PostCSS. Postcss - transforming styles with js plugins.
- [25] Prettier. Prettier - opinionated code formatter.
- [26] RabbitMQ. Rabbitmq - messaging that just works.
- [27] Responsively App. Responsively app - a modified browser for responsive web development.
- [28] Michal Snik. Aos - animate on scroll library.
- [29] Supertest. Supertest - http assertions made easy via superagent.
- [30] Tailwind Labs. Tailwind css - rapidly build modern websites without ever leaving your html.
- [31] VitePress. Vitepress - static site generator powered by vite.
- [32] Vue.js. Vue.js - the progressive javascript framework.
- [33] WebAIM. Contrast checker - webaim.