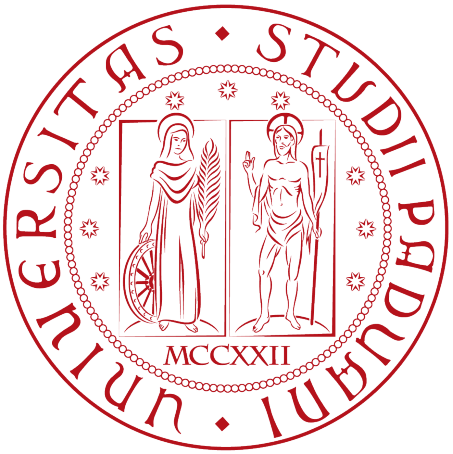
**Università degli Studi di Padova**

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica


# Tecnologie serverless al servizio di un'applicazione di contest d'arte

*Tesi di laurea*


*Relatore*

Prof.ssa Ombretta Gaggi

*Laureando*

Simone Lucato
Matricola 1217322

Anno Accademico 2021-2022

# Sommario

Lo scopo di questa tesi di laurea è presentare il lavoro svolto dal laureando Simone Lucato durante lo stage presso l'azienda FS TECH LAB S.r.l. con sede a Maserà di Padova (PD). Lo stage è stato svolto a conclusione del percorso di studi della Laurea Triennale e ha avuto durata di 320 ore.
Il progetto di stage si è incentrato sullo sviluppo della parte *front-end* di Contest111, un'applicazione di concorsi di opere d'arte, e della sua integrazione con il *back-end*. Partendo dall'analisi dei requisiti, si è proceduti con la progettazione dell'applicazione e la seguente codifica. Durante tutta la parte di sviluppo si è effettuata una verifica costante del codice per terminare con la validazione del prodotto.
Tutte le fasi vengono analizzate nel dettaglio nei capitoli che compongono la presente tesi.

# Organizzazione del testo

Il primo capitolo descrive brevemente l'azienda, analizzandone i processi interni e gli strumenti utilizzati. Viene inoltre descritto il progetto di stage, evidenziando le funzionalità principali da realizzare e le tecnologie che verranno utilizzate.

Il secondo capitolo descrive l'analisi dei requisiti, illustrando gli attori coinvolti nell'interazione con l'applicazione, i principali casi d'uso individuati e i requisiti che si richiede all'applicazione.

Il terzo capitolo descrive la fase di progettazione, l'architettura progettata e i principali flussi di utilizzo individuati.

Il quarto capitolo descrive la fase di codifica, mostrando alcune schermate del prodotto finito e illustrando i dettagli di maggior rilievo realizzati.

Il quinto capitolo descrive le metodologie di verifica e validazione adottate.

Il sesto capitolo illustra le considerazioni finali sugli obiettivi raggiunti e sul progetto di stage.

Per quanto riguarda la stesura del documento, sono state adottate le seguenti convenzioni tipografiche:

- acronimi e termini non di uso comune che verranno menzionati vengono definiti nel glossario presente alla fine del documento;
- la prima occorrenza dei termini riportati nel glossario viene riportata nel seguente modo: termine[g];
- i termini tecnici vengono evidenziati con il carattere *corsivo*;
- il testo che si riferisce a righe di codice viene evidenziato con il seguente `font`.

# Ringraziamenti

*Innanzitutto, vorrei ringraziare di cuore la Professoressa Ombretta Gaggi per la passione trasmessa agli studenti, per la sua ferma dedizione all'insegnamento e per la sua grande umanità.*

*Desidero ringraziare la mia famiglia, in particolare mia nonna Elena, per il costante supporto durante questi anni.*

*Un sentito ringraziamento va a tutti i miei amici, che hanno aggiunto la leggerezza dei momenti di svago a questo cammino.*

*Padova, Luglio 2022* Simone Lucato

# Indice


**1 Introduzione** **1**
- 1.1 L'azienda . . . 1
  - 1.1.1 Processi aziendali . . . 1
  - 1.1.2 Strumenti . . . 1
- 1.2 Il progetto di stage . . . 2
  - 1.2.1 Dominio applicativo . . . 2
  - 1.2.2 Obiettivi . . . 3
  - 1.2.3 Tecnologie utilizzate . . . 4
  - 1.2.4 Strumenti utilizzati . . . 7

**2 Analisi dei requisiti** **11**
- 2.1 Attori . . . 11
- 2.2 Casi d'uso . . . 11
  - 2.2.1 Classificazione . . . 11
  - 2.2.2 Descrizione . . . 12
  - 2.2.3 Elenco casi d'uso . . . 12
- 2.3 Requisiti . . . 18
  - 2.3.1 Classificazione . . . 18
  - 2.3.2 Elenco requisiti . . . 18

**3 Progettazione** **21**
- 3.1 Uso di tecnologie serverless . . . 21
- 3.2 Architettura del prodotto . . . 22
  - 3.2.1 Back-end . . . 22
  - 3.2.2 Front-end . . . 23
  - 3.2.3 Diagrammi di attività . . . 23

**4 Codifica** **27**
- 4.1 Codifica di una componente . . . 27
- 4.2 Home page . . . 28
- 4.3 Artisti . . . 30
- 4.4 Concorsi . . . 30
- 4.5 Opere d'arte . . . 31
- 4.6 Autenticazione . . . 34
- 4.7 Profilo . . . 37
- 4.8 Impostazioni . . . 38
- 4.9 Uso dei context . . . 39
- 4.10 Services . . . 42

4.10.1 Autenticazione . . . . . 43
4.10.2 Contest . . . . . 46
4.10.3 Opere d'arte . . . . . 48

**5 Verifica e validazione** **51**
5.1 Verifica . . . . . 51
5.1.1 Analisi statica . . . . . 51
5.1.2 Analisi dinamica . . . . . 51
5.2 Validazione . . . . . 51

**6 Conclusioni** **53**
6.1 Raggiungimento degli obiettivi . . . . . 53
6.1.1 Completamento degli obiettivi . . . . . 53
6.1.2 Soddisfacimento dei requisiti . . . . . 54
6.2 Conoscenze acquisite . . . . . 55

**Glossario** **57**

**Acronimi** **61**

**Sitografia** **63**

# Elenco delle figure


1.1 Logo di Sveloop . . . 1
1.2 Logo del linguaggio TypeScript . . . 4
1.3 Logo del linguaggio HTML5 . . . 5
1.4 Logo del linguaggio CSS3 . . . 5
1.5 Logo del linguaggio GraphQL . . . 6
1.6 Logo della libreria React . . . 6
1.7 Logo del framework Next.js . . . 7
1.8 Logo del framework TailwindCSS . . . 7
1.9 Logo del sistema operativo Windows 10 . . . 8
1.10 Logo dell'editor di codice sorgente Visual Studio Code . . . 8
1.11 Logo del gestore di pacchetti npm . . . 8
1.12 Logo del sistema di controllo versione distribuito Git . . . 8
1.13 Logo del servizio di hosting di repository Bitbucket . . . 9
1.14 Logo del servizio di issue tracking e project management Jira . . . 9
1.15 Logo del servizio di messaggistica istantanea Slack . . . 9
1.16 Logo del servizio di videoconferenze Zoom . . . 9

2.1 Gerarchia degli attori . . . 11
2.2 Casi d'uso dell'utente non autenticato . . . 12
2.3 Casi d'uso dell'utente autenticato . . . 14
2.4 Casi d'uso dell'amministratore . . . 17

3.1 Architettura ad alto livello dell'applicazione . . . 22
3.2 Diagramma di attività di login e registrazione . . . 24
3.3 Diagramma di attività di creazione di un'opera e di iscrizione a un concorso . . . 25

4.1 Visualizzazione della parte above the fold . . . 29
4.2 Visualizzazione dei contest e delle opere . . . 29
4.3 Visualizzazione degli artisti principali . . . 29
4.4 Visualizzazione della pagina relativa agli artisti . . . 30
4.5 Visualizzazione della pagina del singolo artista . . . 30
4.6 Visualizzazione della pagina relativa ai concorsi . . . 31
4.7 Visualizzazione della pagina del singolo concorso . . . 31
4.8 Visualizzazione della pagina relativa alle opere d'arte . . . 32
4.9 Visualizzazione della pagina di una singola opera (above the fold) . . . 32
4.10 Visualizzazione della pagina di una singola opera (dettagli) . . . 32
4.11 Visualizzazione della pagina di caricamento opera . . . 32

4.12 Visualizzazione della pagina di login in light mode . . . . . 34
4.13 Visualizzazione della pagina di login in dark mode . . . . . 34
4.14 Visualizzazione della pagina di registrazione in light mode . . . . . 35
4.15 Visualizzazione della pagina di registrazione in dark mode . . . . . 35
4.16 Visualizzazione della pagina di conferma email . . . . . 36
4.17 Visualizzazione della pagina di attivazione dell'account . . . . . 36
4.18 Visualizzazione della pagina di richiesta nuova password . . . . . 37
4.19 Visualizzazione della pagina di recupero della password . . . . . 37
4.20 Visualizzazione della pagina del profilo . . . . . 38
4.21 Visualizzazione della pagina di modifica dati personali . . . . . 38
4.22 Visualizzazione della pagina relativa alle impostazioni . . . . . 39

# Elenco delle tabelle

1.1 Obiettivi dello stage . . . . . 3

2.1 Requisiti funzionali dell'applicazione . . . . . 19
2.2 Requisiti di qualità dell'applicazione . . . . . 19

6.1 Stato di completamento degli obiettivi dello stage . . . . . 53
6.2 Stato di soddisfacimento dei requisiti funzionali dell'applicazione . . . 54
6.3 Stato di soddisfacimento dei requisiti di qualità dell'applicazione . . . 55

# Capitolo 1

# Introduzione

## 1.1 L'azienda

FS TECH LAB s.r.l. è un azienda fondata nel 2021 dalla collaborazione di due soci con background pluriennali differenti: uno di sviluppo e ottimizzazione di piattaforme web con ottica *cloud native*, l'altro di web marketing strategico. Nel corso dell'anno è nata l'idea di fondare una software house che si occupasse di sviluppo e gestione di piattaforme web ad alte prestazioni e ad alte conversioni (in ottica marketing). Da qui nasce un nuovo brand: Sveloop (vedasi logo in Figura 1.1), la cui mission principale è quella di sviluppare e gestire piattaforme web *cloud native* in modo da assicurare alte prestazioni e l'utilizzo di tecnologie sempre all'avanguardia. Si è appoggiata ai servizi del cloud[g] AWS[g], di cui sta seguendo tutte le procedure per diventare partner ufficiale. Oltre allo sviluppo di progetti su commissione per terzi, l'azienda ha diversi software proprietari.


**Figura 1.1:** Logo di Sveloop

### 1.1.1 Processi aziendali

L'azienda ha deciso di adottare un approccio Agile nello sviluppo *software* secondo la metodologia Scrum[g]. In essa, il lavoro viene inserito in *sprint* di durata settimanale o bisettimanale e ridotto prima in *epics*, poi in *user stories* e infine in *task* di proporzioni contenute e assegnate ad un individuo. La lista di *task* deriva direttamente dal *backlog* che identifica il lavoro con priorità maggiore secondo i requisiti e le tempistiche del proponente.

### 1.1.2 Strumenti

Tra gli strumenti usati all'interno dell'azienda vi sono:

- **Slack**: comunicazione istantanea con e tra i dipendenti;
- **Jira**: gestione di progetti sviluppati secondo la metodologia Agile e l'*issue tracking*;

- **Confluence**: realizzazione e gestione della documentazione;
- **Bitbucket**: hosting di sistemi di controllo versione di codice Git;
- **Postman**: creazione e gestione di API[g].

Non vengono date indicazioni per l'ambiente di sviluppo, quindi si è lasciati liberi di scegliere l'IDE[g] che si ritiene più adatto allo svolgimento del proprio lavoro.

## 1.2 Il progetto di stage

### 1.2.1 Dominio applicativo

Il progetto di stage proposto riguarda lo sviluppo della parte *front-end* per il progetto Contest111, un'applicazione per la creazione e la gestione di concorsi di opere d'arte, e la gestione delle sue interazioni con il *back-end*. Questo progetto nasce da richieste specifiche di un cliente esterno all'azienda con prospettive di distribuzione dell'applicativo a livello globale. Questo ha portato a una comunicazione continua con il proponente per tenere monitorato un eventuale cambiamento delle condizioni di mercato (e quindi di requisiti) e uno sviluppo orientato all'organizzazione del codice e alla sua scalabilità. Il progetto si propone quindi di coprire tutta la prima fase di sviluppo del codice fino alla sua prima *release*.
Vi sono due funzionalità che formano il nucleo dell'applicazione "Contest111": il caricamento di un'opera d'arte e la possibilità di votarla. A queste si aggiungono altre funzionalità di supporto, pur essendo altrettanto indispensabili: la gestione dei pagamenti e l'autenticazione.

#### Caricamento di un'opera d'arte

Il caricamento di un'opera d'arte dovrà riguardare ogni tipologia di opera (dipinto, scultura, installazione, ecc...). Al fine della realizzazione uniforme per tutte le opere si è deciso di avere una struttura comune semplice per tutte le opere. Di ogni opera infatti sarà necessario sapere il titolo, una descrizione, l'anno e almeno un'immagine. Sarà inoltre possibile inserire la tipologia di opera, le sue dimensioni, il peso, i materiali, i colori ed eventualmente dei tag.
In aggiunta, il caricamento di un'opera potrà essere effettuato solo da utenti registrati alla piattaforma previo pagamento di una quota fissa di iscrizione al concorso.

#### Voto

Il voto di un'opera d'arte consisterà nell'espressione di un giudizio positivo sull'opera (ad esempio un pollice in su) e la possibilità di lasciare un commento. Il voto sarà relativo solo allo specifico concorso e si potrà effettuare a seguito di un contributo forfettario di 1€. Inoltre sarà possibile votare solo a seguito della registrazione sulla piattaforma. Questo viene fatto per associare il voto univocamente all'utente, evitando situazioni di voto multiplo.

#### Gestione dei pagamenti

La gestione dei pagamenti è una funzionalità che è stata implementata con un gestore esterno di pagamenti tramite chiamate API, in modo da semplificare la sua implementazione. I pagamenti gestiti sono:

- L'iscrizione al concorso e quindi il caricamento della prima opera d'arte, con un costo che varia a seconda del concorso;
- Il caricamento di opere aggiuntive alla prima, anche questo costo è variabile a seconda del concorso, sempre inferiore rispetto al primo;
- Il prezzo di voto di un'opera d'arte, ossia 1€.

### Autenticazione

L'autenticazione non è necessaria per la navigazione all'interno dell'applicazione, ma è richiesta per tutte le attività caratterizzanti della stessa (caricamento opera e la possibilità di votarla). Si permette quindi all'utente di registrarsi presso la piattaforma tramite credenziali e dopo una fase di attivazione dell'email, si completa la registrazione inserendo alcuni dati anagrafici (come nome, cognome e codice fiscale) e lo username. Sarà quindi possibile effettuare l'accesso con le stesse credenziali e usufruire delle funzionalità principali.

## 1.2.2 Obiettivi

Vengono riportati in Tabella 1.1 gli obiettivi dello stage, con annesso identificativo e breve descrizione dello stesso.
L'identificativo rispetta la seguente notazione:

**[Importanza][Codice]**

Dove:

- **Importanza**: viene assegnata ad ogni obiettivo nel seguente modo:
  - **Obbligatorio (O)**: importanza assegnata a requisiti che dovranno essere necessariamente soddisfatti, essendo fondamentali per il progetto;
  - **Desiderabile (D)**: importanza assegnata a requisiti non fondamentali, ma la cui realizzazione sarebbe desiderabile.
- **Codice**: Codice identificativo numerico progressivo che parte da 1.

| Identificativo | Descrizione |
|---|---|
| O1 | Stesura documento analisi dei requisiti |
| O2 | Creazione e setting ambiente di sviluppo |
| O3 | Implementazione delle viste relative all'autenticazione |
| O4 | Implementazione delle viste relative alle opere d'arte, ai concorsi e agli artisti |
| O5 | Implementazione del *fetch* di informazioni da *back-end* usando GraphQL |
| O6 | Implementazione della gestione dei pagamenti |
| D1 | Implementazione della piattaforma di admin |

**Tabella 1.1:** Obiettivi dello stage

### 1.2.3 Tecnologie utilizzate

#### 1.2.3.1 Linguaggi

**TypeScript**

TypeScript (vedasi logo in Figura 1.2) è un linguaggio di programmazione *open-source* fortemente tipizzato sviluppato da Microsoft che fornisce un super-set di JavaScript basato su ES6.

**Figura 1.2:** Logo del linguaggio TypeScript

Tra le funzionalità aggiuntive offerte da TypeScript vi sono:

- **Type aliases e interfaces**: modi per registrare la *signature* di una funzione o variabile. Quasi tutte le funzionalità di un `interface` sono presenti su un `type`, la differenza principale è che un `type` non può essere esteso a nuove proprietà, mentre un `interface` è sempre estendibile [40];
- **Classi**: classi vere e proprie per supportare stili di programmazione orientate agli oggetti;
- **Type inference**: TypeScript permette di lasciar dedurre il tipo di una variabile, senza doverlo necessariamente esplicitare. Questo è molto utile al fine di evitare ridondanza di codice [41];
- **Type checking a compile-time**: controllo effettuato durante la compilazione per la verifica di errori di sintassi all'interno del codice. Questa feature è tra le più apprezzate dagli sviluppatori perché permette di risolvere semplici bug in modo molto rapido.

Infine, essendo solo un livello di astrazione aggiuntivo rispetto a JavaScript, ogni programma realizzato in TypeScript viene riconvertito in JavaScript ed elaborato dal *browser* semplicemente rimuovendo i tipi. [39]
La sua robustezza e chiarezza lo rende tra i linguaggi più usati nello sviluppo di grandi applicazioni e per la stessa ragione è stato usato per lo sviluppo della parte *front-end* dell'applicazione.
Viene utilizzata la versione 4.5.4, in quanto versione installata di default dal setup dell'applicazione.

**HTML5**

HTML5 (vedasi logo in Figura 1.3) è un linguaggio di markup usato per creare e strutturare siti web. Rispetto a XHTML introduce nuovi tag (come `<header>`, `<nav>`,

`<section>`) e nuovi attributi (come `placeholder` e `required` nei tag di input) per aumentare il potere espressivo del linguaggio.


**Figura 1.3:** Logo del linguaggio HTML5

### CSS3

CSS3 (vedasi logo in Figura 1.4) è un linguaggio per definire la formattazione di documenti HTML. Rispetto a CSS2 introduce funzionalità importanti come i selettori, nuovi modi per impostare il layout (usando `flex` e `grid`), *media queries* (per cambiare la formattazione solo su dispositivi di specifiche dimensioni) e modi per animare la pagina.


**Figura 1.4:** Logo del linguaggio CSS3

### GraphQL

GraphQL (vedasi logo in Figura 1.5) è un linguaggio di interrogazione e manipolazione dei dati *open-source* sviluppato internamente a Facebook. La peculiarità di questo linguaggio è la sua espressività[1], visto che ritorna solamente i campi richiesti [12]. È inoltre indipendente dal database sottostante o dal modo di memorizzazione dei dati. Questo lo rende usabile sia con database relazionali sia non relazionali, nonostante sia generalmente usato con quest'ultimi (come in questo caso, essendo il database sottostante DynamoDB, il database NoSQL[g]offerto da AWS)

[1] Con espressività si fa riferimento alla possibilità di predirre il risultato, data l'azione (o la richiesta dell'utente). Questo, ad esempio, interrogando un database con MySQL non avviene perché è possibile richiedere tutti i campi senza sapere come questi si chiamino. In GraphQL, invece, è richiesta la conoscenza esatta di ogni campo.

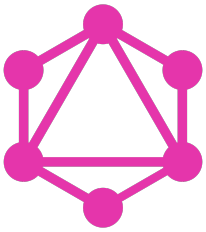


**Figura 1.5:** Logo del linguaggio GraphQL

#### 1.2.3.2 Framework

**React**

React (vedasi logo in Figura 1.6 è una libreria JavaScript *open-source* per la realizzazione di interfacce utente. Sebbene sia possibile realizzare applicazioni mobile usando React Native, generalmente viene usata per la realizzazione di SPA[g]. Lo sviluppo di un'applicazione in React si basa sulla creazione di componenti grafici quanto più riutilizzabili possibili e di procedere alla costruzione della pagina per composizione di componenti.
Un'altra peculiarità è l'uso di un DOM[g]virtuale, che permette l'aggiornamento in modo efficiente del DOM visualizzato dal browser, migliorando le performance [33]. La scrittura di codice in React avviene in JSX[g], un'estensione del linguaggio JavaScript che permette la scrittura di codice simile all'HTML. [29]
Infine, un'ulteriore caratteristica della libreria è l'uso di appositi *hooks*, ossia funzioni che permettono di riferirsi alle caratteristiche del ciclo di vita del componente, permettendo il controllo di stati ed effetti collaterali [28].

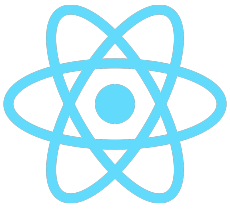


**Figura 1.6:** Logo della libreria React

**Next.js**

Next.js (vedasi logo in Figura 1.7) è un framework *open-source* per lo sviluppo di applicazioni web basate su React costruito su Node.js[g](consigliato dalla stessa documentazione di React [30]). Questo framework estende le funzionalità di React permettendo il *rendering* di applicazioni anche *server-side* (SSR) o la generazione di siti web statici (SSG), ottimizzando le applicazioni anche sotto il punto di vista della SEO[g][17], generalmente un punto critico sviluppando applicazioni in JavaScript (quindi non indicizzate dal motore di ricerca). Implementa numerose features *out-of-the-box*, come:

- Supporto per CSS: compila il CSS con PostCSS, offrendo compatibilità con moduli CSS, preprocessori (come SASS e SCSS) e soluzioni CSS-in-JS (come *inline styles* e `styled-jsx`) [16];

- Routing: il routing[g] è basato sul *file system*, quando un file viene aggiunto alla *directory* `pages`, diventa automaticamente disponibile come percorso. È inoltre possibile avere segmenti dinamici, definendo il nome del file tra parentesi quadre. Infine è anche internazionalizzato, fornendo una lista di *locales* e quello di default [18];
- API Routes: fornisce soluzioni per costruire API endpoint[g] in modo da offrire funzionalità di *back-end*. Per fare ciò, ogni file dentro il percorso `pages/api/*` verrà visto come un endpoint e sarà possibile riferirsi a quel percorso [15].

**Figura 1.7:** Logo del framework Next.js

#### TailwindCSS

TailwindCSS (vedasi logo in Figura 1.8) è un framework di CSS basato sull'uso di classi di utilità, in modo simile a Bootstrap[g]. A differenza sua, però, lavora unicamente con classi [38] (evitando quindi di forzare la priorità massima usando l'attributo `!important`) e ha un supporto di funzionalità molto più esteso. Inoltre, rimuove il CSS non usato durante la *build* per la produzione, arrivando di solito a dimensioni inferiori a 10kB di CSS [37]). In più, gestisce in maniera molto intuitiva la resposiveness (partendo da un'ottica *mobile first*), l'uso di selettori o il supporto per la dark mode. Infine permette di estrarre codice usato di frequente in classi CSS ad-hoc usando la direttiva `@apply`.

**Figura 1.8:** Logo del framework TailwindCSS

### 1.2.4 Strumenti utilizzati

#### 1.2.4.1 Ambiente di lavoro

#### Windows 10

Considerato il fatto che l'azienda ha lasciato libera scelta a dipendenti e stagisti di usare il sistema operativo che preferiscono, è stato utilizzato Windows 10 (vedasi logo in Figura 1.9), in quanto già installato sul dispositivo. Tra le caratteristiche più importanti introdotte vi è il supporto per le app universali, applicazioni progettate per essere eseguite su più famiglie di prodotti Microsoft, cercando un bilanciamento tra interfacce orientate al mouse (per desktop) e orientate all'uso del touchscreen (ad esempio per PC 2 in 1).


**Figura 1.9:** Logo del sistema operativo Windows 10

#### Visual Studio Code

Inoltre considerato che l'azienda ha lasciato libera scelta anche per gli editor di codice, è stato utilizzato Visual Studio Code (vedasi logo in Figura 1.10), essendo l'editor già utilizzato. È un editor di codice sorgente realizzato da Microsoft e tra le sue funzionalità vi è il supporto integrato a Git, la possibilità di debugging, il *syntax highlighting* e il *refactor* del codice. È inoltre possibile aggiungere funzionalità tramite l'installazione di estensioni, in modo da aggiungere supporto per nuovi linguaggi, effettuare analisi statica o aggiungere lint (strumenti per contrassegnare errori di programmazione, errori stilistici, bug e costrutti dubbi).

**Figura 1.10:** Logo dell'editor di codice sorgente Visual Studio Code

#### npm

Npm (vedasi logo in Figura 1.11) è un gestore di pacchetti per il linguaggio di programmazione JavaScript, usato in quanto installato di default con Node.js. Si compone di un client a linea di comando e un database di pacchetti (il cosiddetto *npm registry*). È possibile quindi cercare i pacchetti desiderati dal sito web e installarli tramite il client.


**Figura 1.11:** Logo del gestore di pacchetti npm

### 1.2.4.2 Versionamento

#### Git

Git (vedasi logo in Figura 1.12) è un software usato per il tracciamento di cambiamenti nei file. Generalmente usato per coordinare lo sviluppo in un gruppo di programmatori, supportando flussi di lavoro distribuiti e non lineari tramite *branch*, rappresenta la quasi totalità del mercato dei strumenti di controllo versione.


**Figura 1.12:** Logo del sistema di controllo versione distribuito Git

#### Bitbucket

Bitbucket (vedasi logo in Figura 1.13) è un servizio di hosting di repository realizzato da Atlassian. Viene utilizzato in quanto l'azienda usa diversi strumenti offerti dalla suite di Atlassian.


**Figura 1.13:** Logo del servizio di hosting di repository Bitbucket

#### 1.2.4.3 Issue tracking

Per l'issue tracking si è fatto riferimento a Jira (vedasi logo in Figura 1.14), prodotto realizzato da Atlassian. Viene inoltre utilizzato per la gestione della metodologia Agile impiegata in azienda, quindi per tenere sotto controllo la *roadmap*, monitorare lo stato di avanzamento degli *sprint* e tenere un riferimento *backlog*.


**Figura 1.14:** Logo del servizio di issue tracking e project management Jira

#### 1.2.4.4 Comunicazione

Per la comunicazione interna col tutor aziendale e fra membri del team, si è usato Slack (vedasi logo in Figura 1.15), un programma di messaggistica istantanea realizzato specificamente per l'ambiente lavorativo.


**Figura 1.15:** Logo del servizio di messaggistica istantanea Slack

In caso di riunioni col team o per discutere di problematiche a voce, si è usato Zoom (vedasi logo in Figura 1.16), un software per effettuare videoconferenze sviluppato da Zoom Video Communications dall'interfaccia molto semplice e intuitiva. Permette incontri one-to-one, conferenze di gruppo, condivisione schermo, registrazione degli incontri e fornisce una chat integrata.


**Figura 1.16:** Logo del servizio di videoconferenze Zoom

# Capitolo 2

# Analisi dei requisiti

## 2.1 Attori

Come rappresenta la Figura 2.1, è possibile individuare 3 attori all'interno dell'applicazione:

- **Utente non autenticato**: utente che non ha effettuato l'accesso alla piattaforma;
- **Utente autenticato**: utente che ha effettuato l'accesso alla piattaforma;
- **Admin**: utente con privilegi da amministratore per gestire la piattaforma.

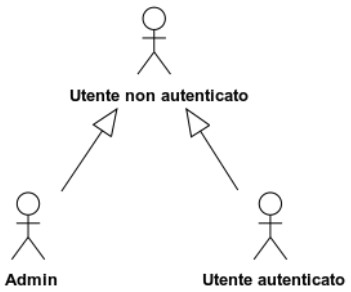



**Figura 2.1:** Gerarchia degli attori

## 2.2 Casi d'uso

### 2.2.1 Classificazione

I casi d'uso avranno la seguente classificazione:

**UC[Codice padre](.[Codice figlio])**

dove:

- **Codice padre** è il codice identificativo numerico del dato caso d'uso;
- **Codice figlio** è il codice identificativo di un eventuale sotto caso d'uso presente solo nell'eventualità di gerarchie tra casi d'uso.

### 2.2.2 Descrizione

Ogni caso d'uso verrà descritto secondo la seguente struttura:

- **Attori**: gli attori del caso d'uso in oggetto. Possono essere l'utente non autenticato, l'utente autenticato e l'amministratore;
- **Precondizione**: specifica le condizioni considerate vere prima del verificarsi degli eventi del caso d'uso;
- **Scenario principale**: rappresenta il flusso principale degli eventi, ossia la situazione più frequente;
- **Estensioni**: eventuali estensioni del caso d'uso;
- **Postcondizione**: specifica le condizioni considerate vere dopo il verificarsi degli eventi del caso d'uso.

### 2.2.3 Elenco casi d'uso

#### Utente non autenticato

La Figura 2.2 riporta i casi d'uso relativi a un utente non autenticato.

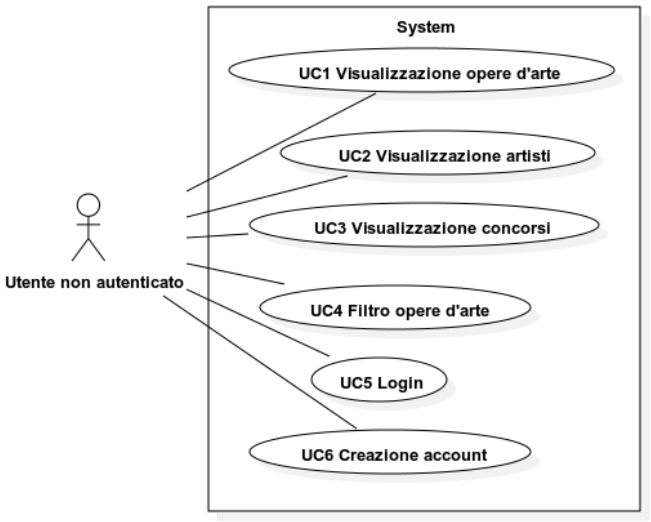



**Figura 2.2:** Casi d'uso dell'utente non autenticato

## UC1: Visualizzazione opere d'arte

- **Attori Principali:** Utente non autenticato;
- **Precondizioni:** L'utente ha accesso al sistema;
- **Flusso principale degli eventi:** L'utente visualizza la lista di opere d'arte e può visualizzare i dettagli di una singola opera;

- **Postcondizioni:** L'utente ha visualizzato le opere d'arte o i dettagli di una singola opera.

### UC2: Visualizzazione artisti

- **Attori Principali:** Utente non autenticato;
- **Precondizioni:** L'utente ha accesso al sistema;
- **Flusso principale degli eventi:** L'utente visualizza la lista di artisti e può visualizzare i dettagli di un singolo artista;
- **Postcondizioni:** L'utente ha visualizzato gli artisti o i dettagli di un singolo artista.

### UC3: Visualizzazione concorsi

- **Attori Principali:** Utente non autenticato;
- **Precondizioni:** L'utente ha accesso al sistema;
- **Flusso principale degli eventi:** L'utente visualizza la lista di concorsi (suddivisi in attivi, futuri e passati, ove presenti) e può vedere i dettagli di un singolo concorso;
- **Postcondizioni:** L'utente ha visualizzato i concorsi o i dettagli di un singolo concorso.

### UC4: Filtro opere d'arte

- **Attori Principali:** Utente non autenticato;
- **Precondizioni:** L'utente ha accesso al sistema;
- **Flusso principale degli eventi:** L'utente filtra le opere d'arte (ad esempio secondo la tipologia di opera o il materiale);
- **Postcondizioni:** L'utente ha filtrato le opere d'arte.

### UC5: Login

- **Attori Principali:** Utente non autenticato;
- **Precondizioni:** L'utente ha accesso al sistema;

- **Flusso principale degli eventi:** L'utente inserisce le credenziali di accesso associate al suo account;
- **Estensione:** L'utente inserisce credenziali non corrette, allora visualizza un messaggio di errore;
- **Postcondizioni:** L'utente ha effettuato l'accesso.

## UC6: Creazione account

- **Attori Principali:** Utente non autenticato;
- **Precondizioni:** L'utente ha accesso al sistema;
- **Flusso principale degli eventi:** L'utente inserisce email e password con cui desidera registrarsi, attiverà la mail tramite un codice ricevuto via e-mail e concluderà la creazione dell'account inserendo alcuni dati anagrafici e lo username;
- **Estensione:** L'utente inserisce email o password non conformi, allora visualizza un messaggio di errore;
- **Postcondizioni:** L'utente ha creato il suo account.

### Utente autenticato

La Figura 2.3 riporta i casi d'uso di un utente autenticato.

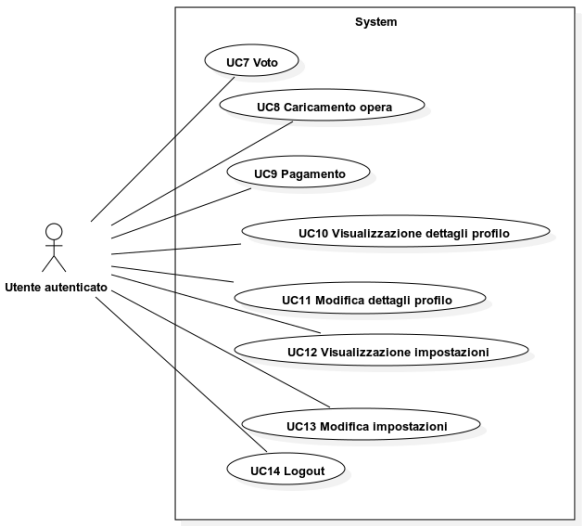



**Figura 2.3:** Casi d'uso dell'utente autenticato

## UC7: Voto

- **Attori Principali:** Utente autenticato;
- **Precondizioni:** L'utente si è autenticato nella piattaforma;
- **Flusso principale degli eventi:** L'utente dà il suo voto a un'opera d'arte all'interno di un concorso, dopo aver pagato 1€ per ogni voto dato;
- **Postcondizioni:** L'utente ha votato l'opera d'arte.

### UC8: Caricamento opera

- **Attori Principali:** Utente autenticato;
- **Precondizioni:** L'utente si è autenticato nella piattaforma;
- **Flusso principale degli eventi:** L'utente inserisce le informazioni relative alla sua opera d'arte e la carica nel sistema;
- **Estensione:** Se l'utente iscrive l'opera ad un concorso, deve anche pagare l'iscrizione al dato concorso;
- **Postcondizioni:** L'utente ha caricato la propria opera d'arte e, se ha voluto iscriverla a un concorso e il pagamento è andato a buon fine, ha iscritto l'opera al concorso scelto.

### UC9: Pagamento

- **Attori Principali:** Utente autenticato;
- **Precondizioni:** L'utente si è autenticato nella piattaforma;
- **Flusso principale degli eventi:** L'utente inserisce i propri dati di pagamento ed effettua un pagamento alla piattaforma;
- **Postcondizioni:** L'utente ha effettuato un pagamento alla piattaforma.

### UC10: Visualizzazione dettagli profilo

- **Attori Principali:** Utente autenticato;
- **Precondizioni:** L'utente si è autenticato nella piattaforma;
- **Flusso principale degli eventi:** L'utente visualizza tutte le informazioni relative al suo profilo (anagrafica, immagine profilo e bio);
- **Postcondizioni:** L'utente ha visualizzato le informazioni relative al suo profilo.

### UC11: Modifica dettagli profilo

- **Attori Principali:** Utente autenticato;
- **Precondizioni:** L'utente si è autenticato nella piattaforma;
- **Flusso principale degli eventi:** L'utente modifica i dettagli del suo profilo che desidera aggiornare;
- **Postcondizioni:** L'utente ha modificato i suoi dettagli.

### UC12: Visualizzazione impostazioni

- **Attori Principali:** Utente autenticato;
- **Precondizioni:** L'utente si è autenticato nella piattaforma;
- **Flusso principale degli eventi:** L'utente visualizza tutte le impostazioni relative al suo profilo (metodi di pagamento, indirizzi, lingua e valuta));
- **Postcondizioni:** L'utente ha visualizzato le impostazioni relative al suo profilo.

### UC13: Modifica impostazioni

- **Attori Principali:** Utente autenticato;
- **Precondizioni:** L'utente si è autenticato nella piattaforma;
- **Flusso principale degli eventi:** L'utente modifica le impostazioni che desidera aggiornare (è possibile inoltre cambiare password);
- **Postcondizioni:** L'utente ha modificato le sue impostazioni.

### UC14: Logout

- **Attori Principali:** Utente autenticato;
- **Precondizioni:** L'utente si è autenticato nella piattaforma;
- **Flusso principale degli eventi:** L'utente disconnette il suo account dalla piattaforma;
- **Postcondizioni:** L'utente non è più autenticato.

### Admin

La Figura 2.4 riporta i casi d'uso di un amministratore.

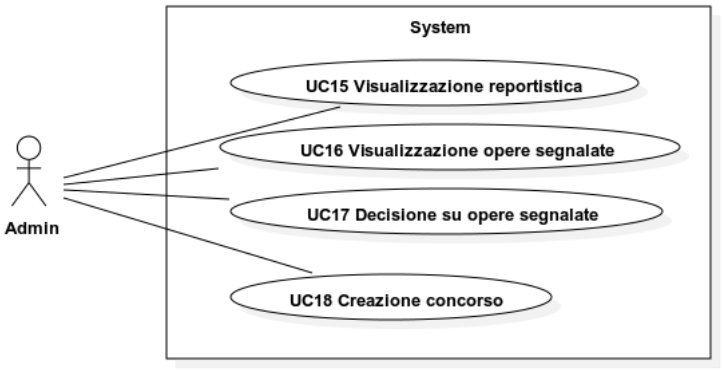



**Figura 2.4:** Casi d'uso dell'amministratore

## UC15: Visualizzazione reportistica

- **Attori Principali:** Admin;
- **Precondizioni:** L'utente ha effettuato l'accesso con privilegi da amministratore;
- **Flusso principale degli eventi:** L'utente visualizza statistiche relative alla piattaforma;
- **Postcondizioni:** L'utente ha visualizzato statistiche relative alla piattaforma.

## UC16: Visualizzazione opere segnalate

- **Attori Principali:** Admin;
- **Precondizioni:** L'utente ha effettuato l'accesso con privilegi da amministratore;
- **Flusso principale degli eventi:** L'utente visualizza l'elenco di opere segnalate;
- **Postcondizioni:** L'utente ha visualizzato l'elenco di opere segnalate.

## UC17: Decisione opere segnalate

- **Attori Principali:** Admin;

- **Precondizioni:** L'utente ha effettuato l'accesso con privilegi da amministratore;
- **Flusso principale degli eventi:** L'utente prende una decisione riguardo un'opera segnalata (ad esempio bloccandone la visualizzazione);
- **Postcondizioni:** L'utente ha preso una decisione riguardo un'opera segnalata.

### UC18: Creazione concorso

- **Attori Principali:** Admin;
- **Precondizioni:** L'utente ha effettuato l'accesso con privilegi da amministratore;
- **Flusso principale degli eventi:** L'utente inserisce le informazioni relative a un nuovo concorso;
- **Postcondizioni:** L'utente ha creato un nuovo concorso.

## 2.3 Requisiti

### 2.3.1 Classificazione

I requisiti seguiranno la seguente classificazione:

**R[Tipologia][Importanza][Codice]**

dove:

- **Tipologia**: può essere uno tra le seguenti:
  - **F**: indica un requisito funzionale;
  - **Q**: indica un requisito di qualità;
- **Importanza**: può essere uno tra le seguenti:
  - **O (Obbligatorio)**: indica un requisito obbligatorio, che dovrà essere necessariamente soddisfatto, di fondamentale importanza per il progetto;
  - **D (Desiderabile)**: indica un requisito desiderabile, ma non fondamentale per il progetto;
- **Codice**: indica un codice identificativo univoco per ogni requisito. In caso di gerarchie è previsto il formato [Codice padre].[Codice figlio].

### 2.3.2 Elenco requisiti

#### 2.3.2.1 Requisiti funzionali

La Tabella 2.1 riporta i requisiti funzionali dell'applicazione.

| Requisito | Descrizione | Fonte |
|---|---|---|
| RFO1 | L'utente deve poter vedere la lista completa di opere d'arte | UC1 |
| RFO2 | L'utente deve poter vedere i dettagli di una singola opera d'arte | UC1 |
| RFO3 | L'utente deve poter vedere la lista completa di artisti | UC2 |
| RFO4 | L'utente deve poter vedere i dettagli di un singolo artista | UC2 |
| RFO5 | L'utente deve poter vedere la lista completa di concorsi (divisi in attivi, futuri e passati) | UC3 |
| RFO6 | L'utente deve poter vedere i dettagli di un singolo concorso | UC3 |
| RFO7 | L'utente deve poter filtrare le opere d'arte | UC4 |
| RFO8 | L'utente deve poter vedere le opere d'arte filtrate | UC1, UC4 |
| RFO9 | L'utente deve poter avere accesso alla piattaforma | UC5 |
| RFO10 | L'utente deve poter creare un account | UC6 |
| RFO11 | L'utente deve poter votare un'opera d'arte, previo pagamento | UC7, UC9 |
| RFO12 | L'utente deve caricare un'opera d'arte, previo pagamento | UC8, UC9 |
| RFO13 | L'utente deve poter effettuare pagamenti verso la piattaforma | UC9 |
| RFO14 | L'utente deve poter visualizzare le informazioni relative al suo profilo | UC10 |
| RFO15 | L'utente deve poter modificare le informazioni relative al suo profilo | UC11 |
| RFO16 | L'utente deve poter visualizzare le impostazioni relative al suo profilo | UC12 |
| RFO17 | L'utente deve poter modificare le impostazioni relative al suo profilo | UC13 |
| RFO18 | L'utente deve poter effettuare il logout | UC14 |
| RFO19 | L'applicazione dev'essere responsive | - |
| RFD1 | L'admin deve poter vedere dei report statistici sulla piattaforma | UC15 |
| RFD2 | L'admin deve poter vedere la lista di opere segnalate | UC16 |
| RFD3 | L'admin deve poter rispondere alla segnalazione di un'opera | UC17 |
| RFD4 | L'admin deve poter creare un nuovo concorso | UC18 |

**Tabella 2.1:** Requisiti funzionali dell'applicazione

#### 2.3.2.2 Requisiti di qualità

La Tabella 2.2 riporta i requisiti funzionali dell'applicazione.

| Requisito | Descrizione |
|---|---|
| RQO1 | L'interfaccia utente deve essere realizzata in lingua inglese |

**Tabella 2.2:** Requisiti di qualità dell'applicazione

# Capitolo 3

# Progettazione

## 3.1 Uso di tecnologie serverless

L'applicazione è stata realizzata secondo le richieste del cliente. In particolare, dagli incontri sono emersi alcuni punti fermi:

- È necessaria una previsione di costo specifica (idealmente un modello pay-as-you-go);
- Non è chiaro il numero di persone collegate simultaneamente che l'applicazione potrebbe dover supportare;
- I costi di gestione devono essere quanto più contenuti possibile (essendo il progetto in fase di start-up);
- Dev'essere possibile (e semplice) poter scalare l'applicazione fino a raggiungere una distribuzione globale.

In generale, è possibile realizzare applicativi usando server locali, server gestiti da terzi oppure tecnologie serverless. Le differenze sono:

- **Server locale**: è necessario comprare dei server e tenerli in azienda. Ha un costo specifico fisso (dato dall'*hardware*), ma è soggetto alla possibile sotto o sovrastima del numero di utenti, aggiungendo complessità. Richiede manutenzione costante (ad esempio l'installazione di patch o la gestione di permessi) ed è necessario assumere qualcuno che lo faccia, aumentando i costi. È possibile scalare verticalmente un server (aumentando memoria, CPU o cambiando *hardware*) od orizzontalmente, duplicando componenti. È tuttavia una gestione onerosa, soprattutto se in un'altra regione, a causa della continua manutenzione richiesta;
- **Server gestiti**: si istanziano macchine virtuali interne a un data center proprietario di terzi (ad esempio Amazon, Google o Microsoft). I costi sono dati dall'istanza scelta, da quando viene accesa a quando viene spenta. La manutenzione è più semplice in quanto non è necessario occuparsi della macchina fisica ma solo di quella virtuale (quindi sono sufficienti gli aggiornamenti software. Inoltre è anche di più semplice scaling perché è sufficiente aggiungere o togliere risorse alla macchina virtuale;

– **Tecnologie serverless**: vi sono servizi pronti all'uso forniti da specifici provider (ad esempio AWS di Amazon, Cloud di Google o Azure di Microsoft). I costi sono dati unicamente dal consumo (su schemi precisi per ogni servizio). Non è necessario effettuare manutenzione, quindi non vi sono costi di gestione. La scaling viene gestito automaticamente dal servizio. Tuttavia, è richiesta la necessità d'uso di quel particolare servizio, non è possibile un uso generico.

A fronte di queste considerazioni, si è scelto di usare tecnologie serverless in quanto risponde nel modo migliore alle esigenze del cliente e tutti i servizi necessari allo sviluppo sono forniti da un unico provider (in questo caso AWS).

## 3.2 Architettura del prodotto

In figura Figura 3.1 è riportata l'architettura ad alto livello del *front-end* dell'applicazione. L'architettura adottata è quella standard per lo sviluppo di una SPA: vi è il package relativo alla `view` viene realizzato tramite la libreria o il framework usato (in questo caso Next.js prevede l'uso di questi due package); è inoltre necessaria la gestione dello stato interno dell'applicazione, per cui si è usato Redux[g]; infine vi sono le chiamate al *back-end* nell'apposito package `services`. In più, dalla figura è possibile capire immediatamente i servizi AWS utilizzati.

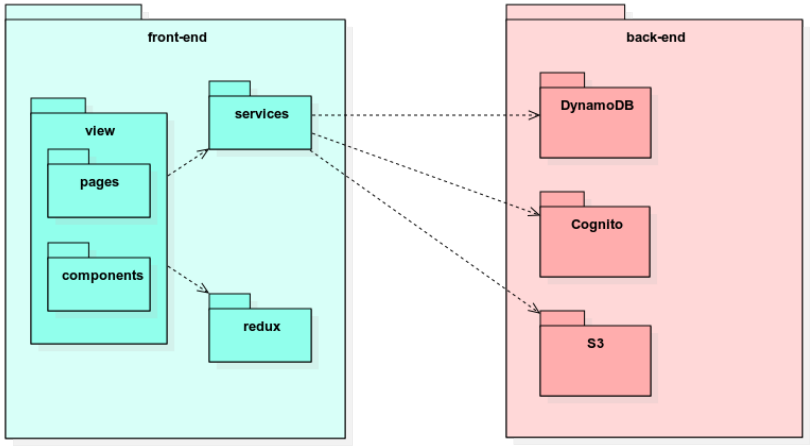



**Figura 3.1:** Architettura ad alto livello dell'applicazione

### 3.2.1 Back-end

Il *back-end* è composto dai servizi AWS utilizzati. In particolare, al momento vengono utilizzati DynamoDB, Cognito e S3 Bucket.
Il primo è un database NoSQL. Questa tipologia di database è stata preferita rispetto a quelli relazionali (pur essendo molto più semplice idealmente) in quanto i dati trattati sono privi di struttura. Non essendo ancora chiara la struttura delle opere d'arte (e per evitare di imporre eccessivi limiti allo sviluppo futuro) si è deciso di adottare un database non relazionale. [5]
Il secondo è un servizio usato per l'autenticazione. Include l'invio di email

di conferma account in fase di registrazione, la possibilità di gestire flussi di autenticazione personalizzati e la gestione di autenticazione a più fattori (MFA[g]). [4]
Il terzo è un servizio usato per l'archiviazione di oggetti. Durante il progetto è stato usato per la memorizzazione delle immagini relative alle opere d'arte.

### 3.2.2 Front-end

Il *front-end* è diviso in 4 package principali: `components`, `pages`, `redux` e `services`.

#### Components

`components` contiene tutti i file di componenti considerate riusabili, quindi non appartenenti a una pagina specifica. Tra questi vi sono componenti UI[g], come bottoni e cards (nella specifica *subdirectory* `elements`), componenti di form, come checkbox, select e input (nella *subdirectory* `forms`), componenti relativi alla navigazione, come le [g]e i menù di navigazione (nella *subdirectory* `navigation`) e componenti relativi al cambio tema (nella *subdirectory* `theme`).

#### Pages

`pages` contiene tutti i file delle pagine visualizzabili dell'applicazione. Next.js infatti ha un sistema di routing basato sul *file system*: ogni file inserito in questa *folder* diventa automaticamente disponibile come route. Il router indirizzerà automaticamente i file chiamati `index.tsx` alla root della directory. È inoltre possibile avere segmenti dinamici usando la sintassi con parentesi quadre (ad esempio chiamando il file `[id].tsx`) [18].

#### Redux

`redux` contiene tutti i file relativi alla gestione di Redux, una libreria di *state management*. Conterrà quindi la configurazione dello *store* dove vengono memorizzate le informazioni e le diverse *slice* che creano le azioni e i *reducer* per interfacciarsi con lo *store* [34] [35].

#### Services

`services` contiene tutti i file con al loro interno funzioni che permettono al *front-end* di interfacciarsi con le API del *back-end*. Verranno analizzati nel dettaglio nella sezione 4.10

### 3.2.3 Diagrammi di attività

Di seguito vengono riportati alcuni diagrammi con i flussi principali di attività dell'utente.

#### 3.2.3.1 Autenticazione

In figura Figura 3.2 si riporta il flusso relativo all'autenticazione di un'utente. In fase di registrazione, si chiede email e password, viene poi inviata un'email di conferma con un codice da inserire per confermare l'e-mail. In seguito, si chiedono alcuni dati anagrafici (come nome, cognome, sesso e codice fiscale) e lo username. Dopo questo passaggio l'account viene definitivamente creato e sarà possibile effettuare il login con le credenziali inserite all'inizio. Se per qualche ragione uno step dovesse essere interrotto, in fase di login si rimanderebbe alla pagina relativa.

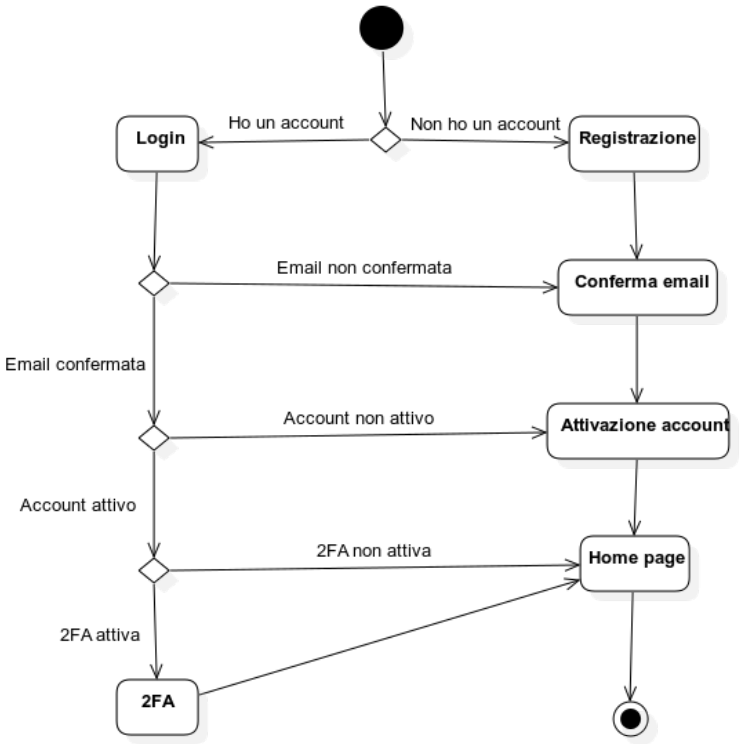



**Figura 3.2:** Diagramma di attività di login e registrazione

#### 3.2.3.2 Creazione opera e iscrizione concorso

In figura Figura 3.3 si riporta il flusso relativo alla creazione di un'opera d'arte e la sua iscrizione a un concorso. In fase di creazione dell'opera è possibile iscriverla o meno a un concorso attivo. In entrambi i casi l'opera viene creata e qualora si scegliesse di farlo, si andrebbe poi al checkout, pagando una quota a seconda di se fosse la prima opera iscritta al concorso o meno. Dopo l'avvenuto pagamento, l'opera risulterebbe iscritta al concorso. Se non venisse iscritta al concorso, l'opera verrebbe solamente creata e sarebbe visualizzabile nella pagina personale dell'artista.
In alternativa, si può iscrivere un'opera a un concorso partendo da quest'ultimo. In questo caso, bisognerebbe scegliere se creare un'opera ex novo (rimandando

alla situazione precedente) oppure scegliendo un'opera già creata, non ancora iscritta. Anche in questo caso si procede al pagamento, a seguito del quale l'operà risulterà iscritta al concorso.

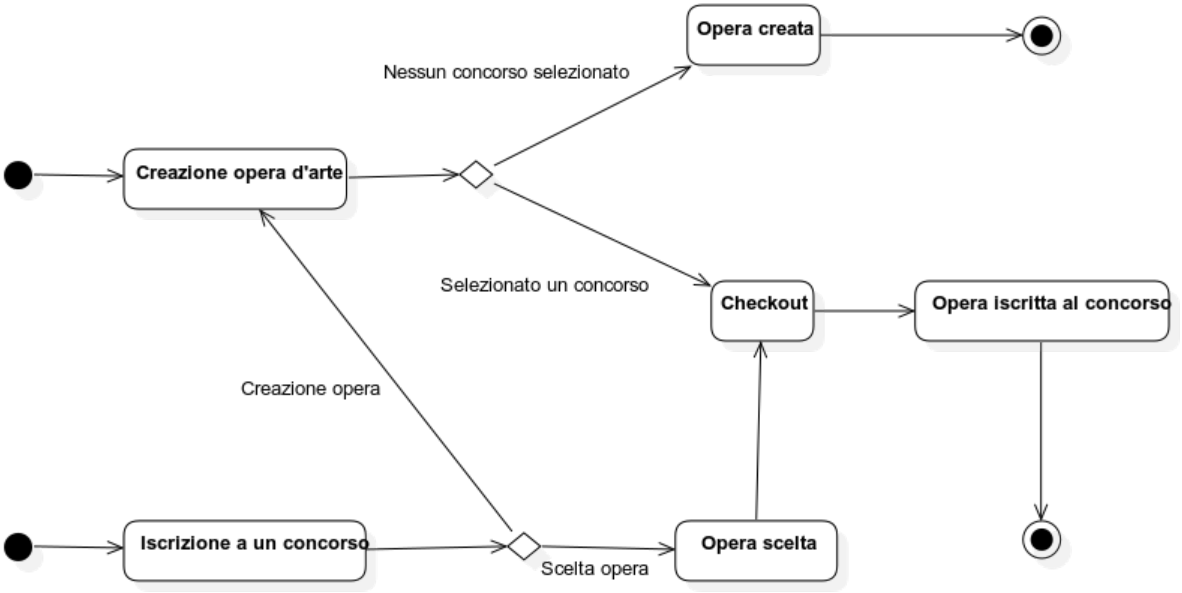



**Figura 3.3:** Diagramma di attività di creazione di un'opera e di iscrizione a un concorso

# Capitolo 4

# Codifica

*Il design dell'applicazione è stato realizzato da un grafico apposito. L'applicazione prevede la possibilità di essere utilizzata sia in light sia in dark mode. Tuttavia, per questioni di tempo non è stato possibile completare tutte le pagine in light mode (per maggiori dettagli si rimanda alla sezione 6.1). Ove possibile, verranno quindi mostrate entrambe le versioni, altrimenti solo la versione scura.*
*Inoltre per questione di comodità verranno riportate preferibilmente le schermate delle pagine in versione mobile, secondo un'ottica* mobile first *e stimando un uso dell'applicazione molto maggiore da questo dispositivo [10] [11].*
*Infine, per ragioni di spazio, non verranno mostrate tutte le schermate dell'applicazione, ma solo quelle considerate più significative.*

## 4.1 Codifica di una componente

Analizziamo di seguito la realizzazione di una componente usando React e TypeScript. In particolare analizziamo `SemitransparentCard.tsx` nello Snippet di codice 4.1, un componente *wrapper*. Come suggerisce il nome, si propone di realizzare una card, ossia un contenitore con bordi arrotondati e uno sfondo diverso, semitrasparente in questo caso. Generalmente un componente non è altro che una funzione che riceve dei parametri in input (definite props[g]) e ritorna del codice JSX, di solito un `React.FC` (*Function Component*, componente funzionale) [21]. Le props in TypeScript possono essere definite nei `type` o nelle `interface` [40] [22]. Nel mio caso ho preferito usare i `type` in quanto consentono funzionalità frequenti non presenti nelle `interface`, come le unioni di tipi [36]. Gli attributi si assumono obbligatori salvo diversamente con un punto di domanda (`?`), come indicato nello Snippet di codice 4.1.

```
1 import classNames from "classnames"
2 import React from "react"
3
4 type Props = {
5     className?: string
6 }
7
8 const SemitransparentCard: React.FC<Props> = ({ className="",
      children }) => {
```

```
9       return (
10          <div className={classNames(className, "semitransparent-card"
                )}>
11        {children}
12          </div>
13      )
14  }
15
16  export default SemitransparentCard
```

**Snippet di codice 4.1:** Esempio di una componente in React e TypeScript

Guardando il codice, `SemitransparentCard` ha come tipo di ritorno esplicito `React.FC<Props>`[1] (il tipo di ritorno è esplicitato con i due punti), ossia un componente funzionale di tipo `Props`. Come parametri di costruzione prende quindi `className` (di default inizializzato a una stringa vuota) e `children` (fornito di default essendo un componente funzionale). La sintassi a graffe viene definita *object destructuring* e viene usata per ottenere proprietà singole dagli oggetti. Di solito è usato in combinazione con la *spread syntax* (`...`) che permette di espandere le proprietà di un oggetto. [13]. Ad esempio:

```
1  const { a, b, ...rest } = obj      // rest contiene tutte le
       proprieta' restanti in obj dopo aver tolto a e b
```

Dentro la funzione `return()` si costruisce il componente che viene ritornato. Nella parte antecedente è possibile definire variabili, funzioni o usare gli *hooks*. Gli *hooks* generalmente più usati sono `useState` e `useEffect`: il primo espone lo stato e un metodo per settarlo e al cambiamento dello stesso viene invocato un *refresh* della componente [32]; il secondo viene usato per fare *side effects* al variare di una o più variabili [31].

## 4.2 Home page

Nelle figure sottostanti viene riportata la visualizzazione della home page. In particolar modo, la Figura 4.1 riporta la parte *above the fold*, ossia la prima parte, senza effettuare nessuno scroll. Il contenuto principale è un video introduttivo di Contest111 che occupa buona parte della pagina. Poi in Figura 4.2 si mostrano i contest attualmente attivi e futuri, oltre alle principali opere. Infine, in Figura 4.3 è possibile vedere i principali artisti.

[1] Viene esplicitato il tipo di ritorno usando `React.FC` unicamente per la necessità di avere il parametro `children`, ossia gli elementi figli del componente. Altrimenti non viene indicato il tipo di ritorno, non essendo considerata una *best pratice* [9]

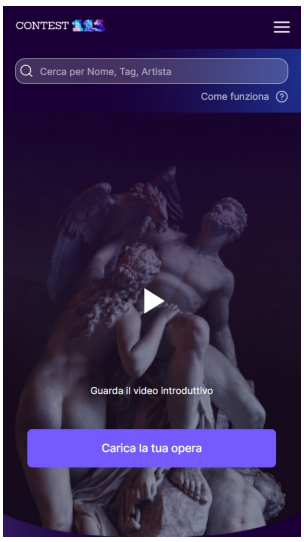



**Figura 4.1:** Visualizzazione della parte above the fold

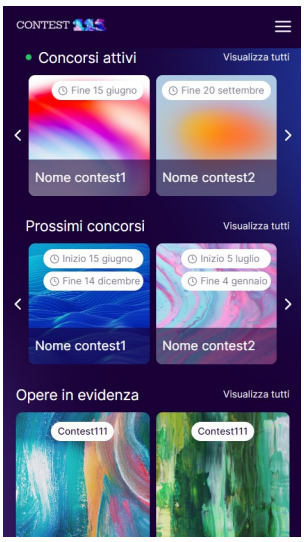



**Figura 4.2:** Visualizzazione dei contest e delle opere

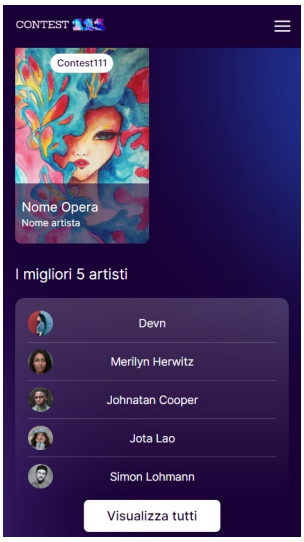



**Figura 4.3:** Visualizzazione degli artisti principali

Per la realizzazione dei caroselli presenti nella home page, si è usata la libreria `pure-react-carousel`, l'unica libreria a realizzare caroselli accessibili secondo gli standard WCAG[g][19]. Per la realizzazione delle componenti relative ai concorsi e alle opere, si è usato un modello ad *Higher Order Component* (HOC[g]). Questo pattern è emerso dalla natura improntata alla composizione di React e consiste in una funzione che prende un componente come parametro e ne ritorna una nuovo, con funzionalità aggiuntive. Per fare ciò, un HOC racchiude il componente originale in un componente contenitore, aggiungendo la funzionalità richiesta [27]. La *naming convention* prevede l'uso di `with` all'inizio del nome della componente. Un esempio di questo può essere visto nell'HOC `withButtonWrapper.tsx` nello Snippet di codice 4.2, in cui si ritorna lo stesso componente *wrappato* in un bottone.

```
1  interface Props {
2    onClick: MouseEventHandler<HTMLButtonElement>
3  }
4
5  export default withButtonWrapper =
6    <P extends object>(Component: React.FC<P>) : React.FC<P & Props>
       =>
7    (props : P & Props) => {
8      const { onClick, ...others } = props
9      return (
10       <button type="button" onClick={onClick}>
11         <Component {...others as P} />
12       </button>
13     )
14   }
```

**Snippet di codice 4.2:** Esempio di HOC

## 4.3 Artisti

Nelle figure sottostanti è possibile vedere le pagine relative agli artisti. In particolar modo, in Figura 4.4 è possibile vedere la pagina in cui sono riportati tutti gli artisti ordinati per numero di voti ricevuti, invece in Figura 4.5 è possibile vedere la pagina del singolo artista, di cui si riporta la foto profilo, il nome, una breve descrizione, le sue opere e i contest in cui ha partecipato.

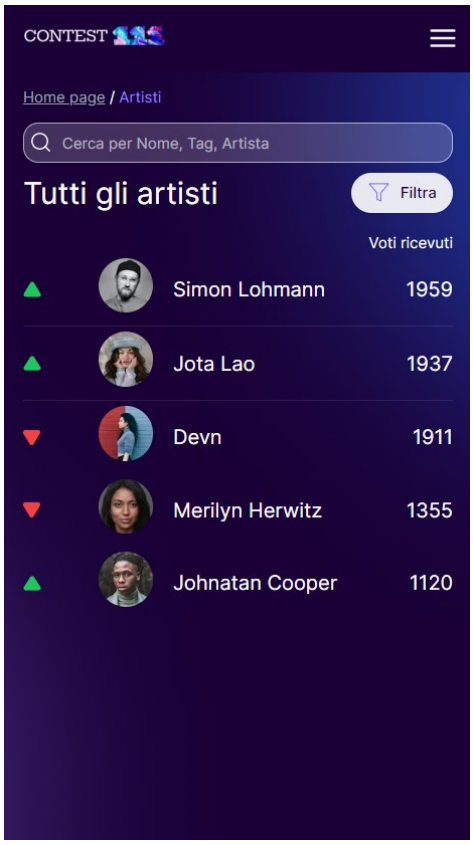



**Figura 4.4:** Visualizzazione della pagina relativa agli artisti

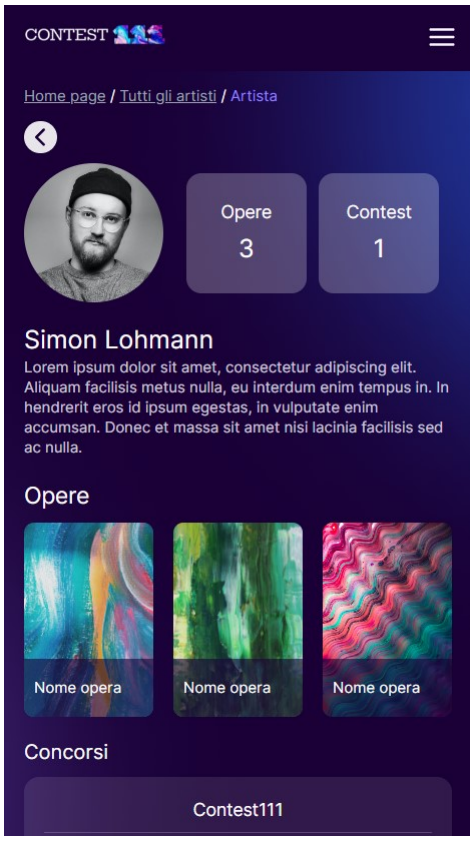



**Figura 4.5:** Visualizzazione della pagina del singolo artista

## 4.4 Concorsi

Nelle figure sottostanti è possibile vedere le pagine relative ai concorsi. In particolare, in Figura 4.6 è possibile vedere la pagina in cui sono riportati tutti i concorsi. Anche in questo caso il componente di ciascun contest è realizzato secondo la logica a HOC, aggiungendo il badge sottostante al titolo. Invece, in Figura 4.7 è possibile vedere la pagina del singolo concorso. A seconda del periodo le iscrizioni potranno essere aperte o chiuse (e il badge sarà aggiornato). È inoltre possibile vedere la classifica attuale del concorso e l'elenco degli artisti iscritti. In questa pagina è possibile è iscriversi ad un concorso, dopo aver effettuato il login, scegliendo un'opera già inserita o caricandone una nuova.

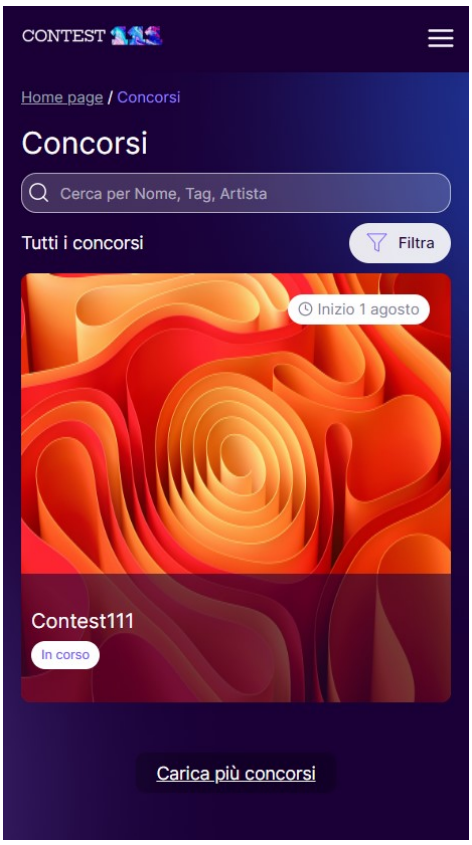



**Figura 4.6:** Visualizzazione della pagina relativa ai concorsi

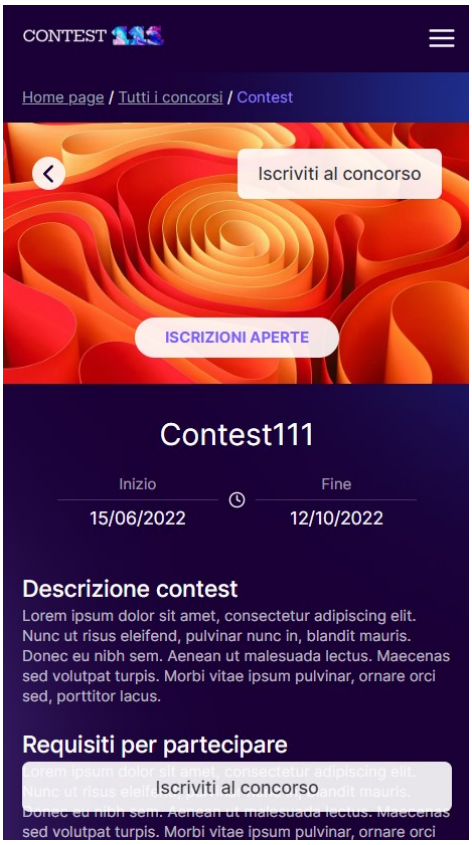



**Figura 4.7:** Visualizzazione della pagina del singolo concorso

## 4.5 Opere d'arte

Nelle figure sottostanti è possibile vedere le pagine relative ai concorsi. In particolare, in Figura 4.8 è possibile vedere la pagina in cui sono riportate tutte le opere d'arte. Invece in Figura 4.9 e in Figura 4.10 si ha la visualizzazione della pagina della singola opera, della quale sono riportati i dettagli principali. Qui sarà possibile votarla, scegliendo in concorso in cui darle il voto.

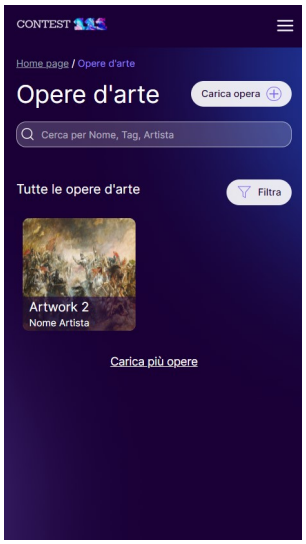



**Figura 4.8:** Visualizzazione della pagina relativa alle opere d'arte

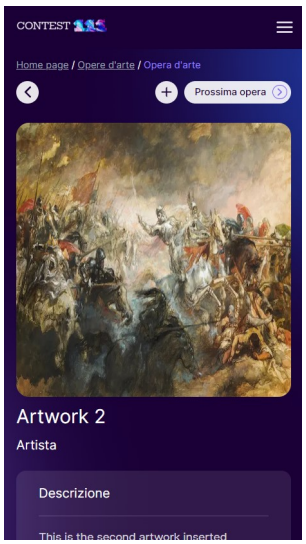



**Figura 4.9:** Visualizzazione della pagina di una singola opera (above the fold)

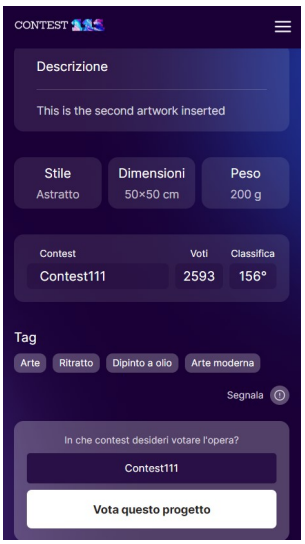



**Figura 4.10:** Visualizzazione della pagina di una singola opera (dettagli)

In Figura 4.11 è possibile vedere la pagina di caricamento dell'opera. Di ogni opera si richiede il titolo, l'anno di produzione, una descrizione e almeno una foto. È inoltre possibile specificare le dimensioni dell'opera, il peso, i materiali, i colori ed eventuali tag. Infine, è possibile decidere se iscrivere o meno l'opera a un concorso. In questo modo, dopo la creazione si procederebbe al pagamento della quota di iscrizione al concorso scelto e dopo l'avvenuto pagamento l'opera sarebbe iscritta.

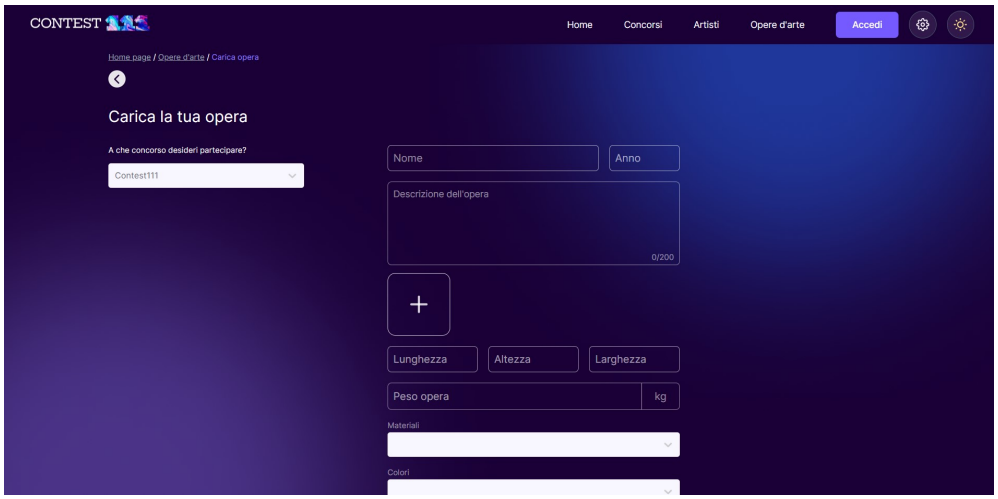



**Figura 4.11:** Visualizzazione della pagina di caricamento opera

Per la realizzazione dei form come questo, si è usata la libreria `React Hook Form`. Questa libreria espone un custom *hook*, `useForm` dal quale si possono ricavare metodi e attributi utili per la gestione del form [24]:

- `submitHandler`: metodo che riceve i dati del form se la validazione è andata a buon fine, prevenendo il submit automatico;
- `formState`: oggetto che contiene informazioni relativo allo stato dell'intero form. Di solito da questo oggetto si ricava la proprietà `errors`, per sapere se vi sono errori di validazione;
- `control`: oggetto che contiene metodi per la registrazione di un field di input all'interno dell'*hook*. Generalmente viene utilizzato per la realizzazione di componenti controllate [26].

In questo caso è sorto il problema di dover riutilizzare lo stesso componente controllato con la stessa logica in form diversi. Questo rappresenta un problema perché `useForm` richiede di essere tipizzato (per sapere che campi si deve aspettare), ma il componente dovrebbe essere generico e riusabile sempre, non vincolato dai campi richiesti dal form. Perciò sono stati realizzati dei componenti templatizzati, come `ControlledTransparentInput.tsx` e `ControlledBadgeCheckbox.tsx` che avvolgono il componente grafico in un `<Controller/>`, fornito da `React Hook Form`, che richiede come parametro di costruzione l'attributo `control` ricevuto da `useForm`. Perciò questo attributo verrà passato al componente templatizzato in fase di costruzione. Si riporta a fine esemplificativo il codice relativo a `ControlledBadgeCheckbox.tsx` nello Snippet di codice 4.3.

```tsx
import React from "react"
import { Control, Controller, Path } from "react-hook-form"
import BadgeCheckbox from "../checkbox/BadgeCheckbox"

interface Props<P> {
  id: string
  control: Control<P, any>
  label: string
  name: Path<P>
  value: string
}

const ControlledBadgeCheckbox = <P extends object>({
  id,
  control,
  label,
  name,
  value,
}: Props<P>) => {
  return (
    <Controller
      control={control}
      name={name}
      render={({ field }) => (
        <BadgeCheckbox
          id={id}
          value={value}
          checked={field.value === value}
          onChange={(event) =>
            field.onChange(event.target.checked ? value : "")
          }
          label={label}
        />
```

```
34          )}
35        />
36      )
37  }
38
39  export default ControlledBadgeCheckbox
```

**Snippet di codice 4.3:** ControlledBadgeCheckbox.tsx

## 4.6 Autenticazione

In Figura 4.12 e Figura 4.13 è possibile vedere la schermata di login rispettivamente in light e dark mode. Per la realizzazione componente della password si è dovuto utilizzare un `<div>` per contenere l'input e l'icona per mostrare la password. Perciò non è stato possibile usare il selettore `:focus` per riportare rendere il bordo rosso in caso di errore, ma è stato necessario usare il selettore `:focus-within`.
L'icona inoltre è stata realizzata come un bottone con attributo `aria-label` apposito per renderla navigabile con la tastiera e in modo che utenti che utilizzano tecnologie assistive siano correttamente guidate.

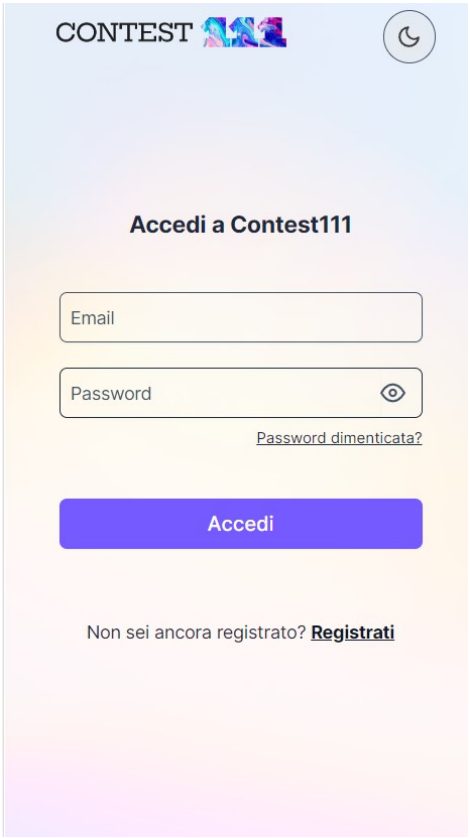



**Figura 4.12:** Visualizzazione della pagina di login in light mode

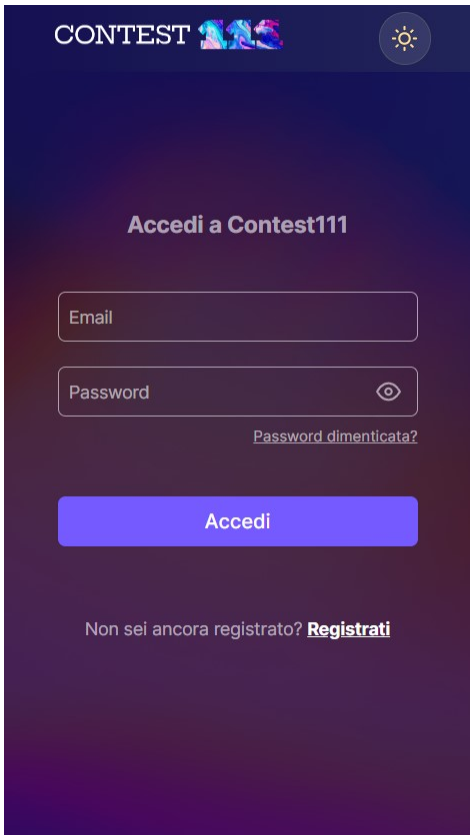



**Figura 4.13:** Visualizzazione della pagina di login in dark mode

In Figura 4.14 e Figura 4.15 è possibile vedere la schermata di registrazione rispettivamente in light e dark mode. La validazione della password viene effettuata in tempo reale, controllandone la lunghezza e validando ciascun requisito con un'espressione regolare apposita.

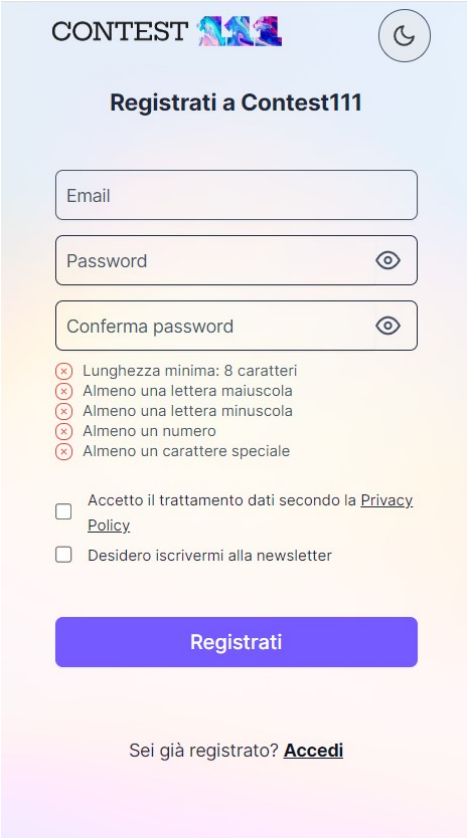



**Figura 4.14:** Visualizzazione della pagina di registrazione in light mode

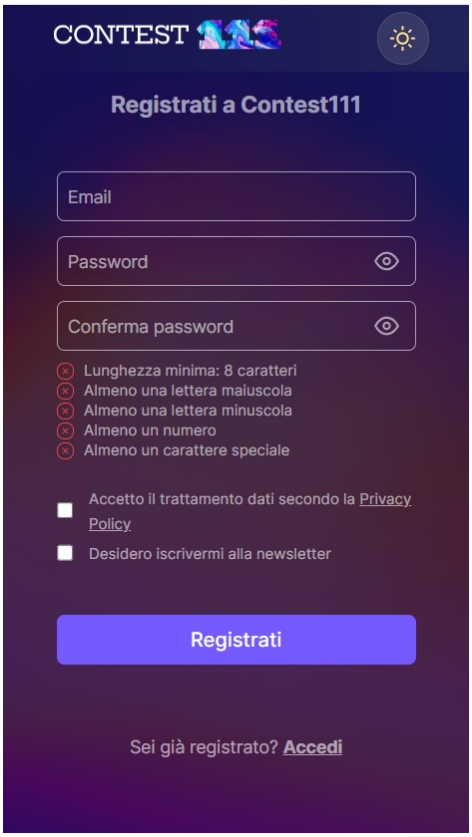



**Figura 4.15:** Visualizzazione della pagina di registrazione in dark mode

In Figura 4.16 e Figura 4.17 è possibile vedere la pagina di conferma email successiva alla registrazione e la pagina di attivazione account, successiva a quest'ultima. In caso il processo di registrazione venisse interrotto, verrebbe effettuato un controllo in fase di login per verificare se l'utente ha confermato l'email e/o attivato l'account e in caso rimandarlo nella pagina corretta. È inoltre possibile richiedere un ulteriore codice di conferma.

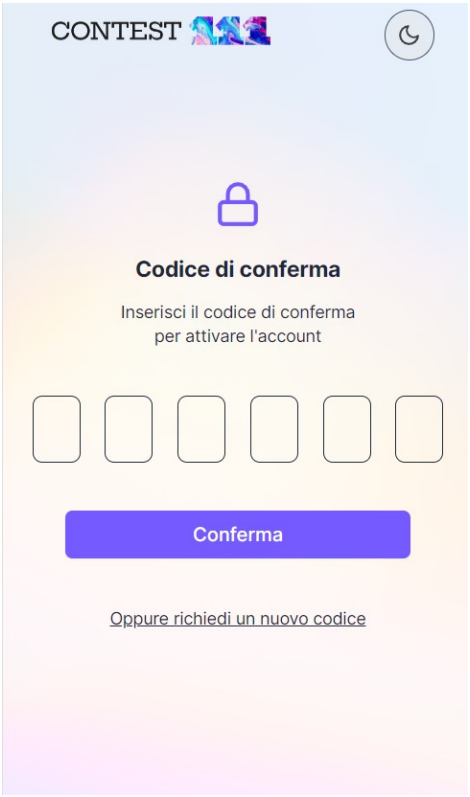



**Figura 4.16:** Visualizzazione della pagina di conferma email

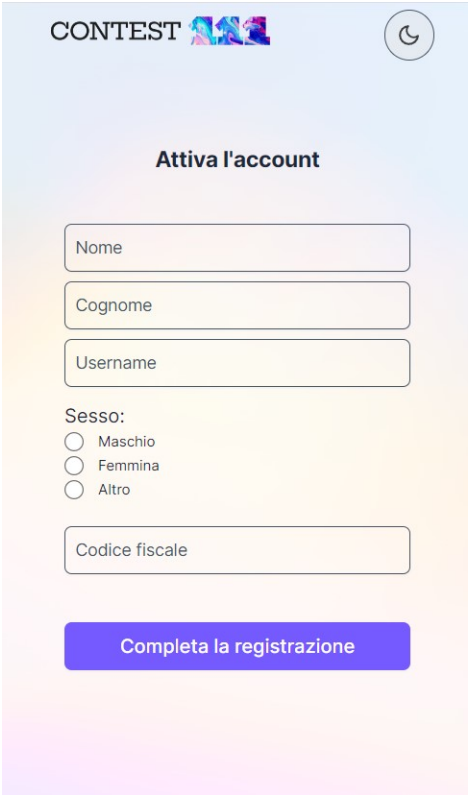



**Figura 4.17:** Visualizzazione della pagina di attivazione dell'account

In Figura 4.18 e Figura 4.19 è possibile vedere le pagine di recupero password. Anche in questo caso vi sono due step: prima viene inviato un codice per il recupero della password alla mail della quale si desidera recuperare la password, poi si inserisce il codice e la nuova password, confermandola.

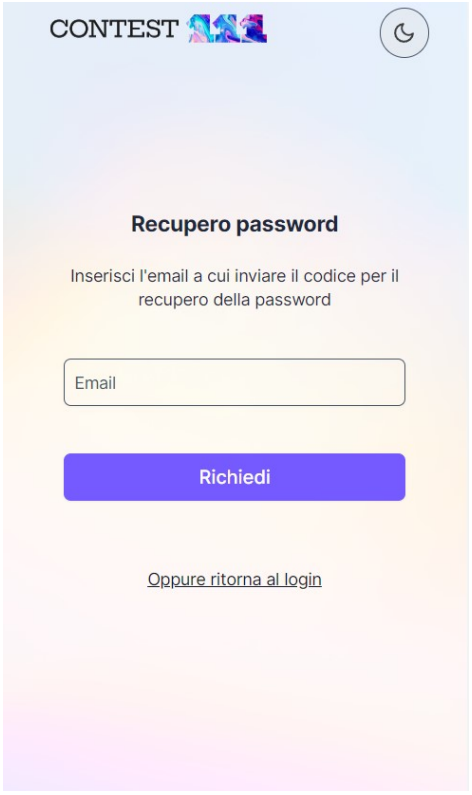



**Figura 4.18:** Visualizzazione della pagina di richiesta nuova password

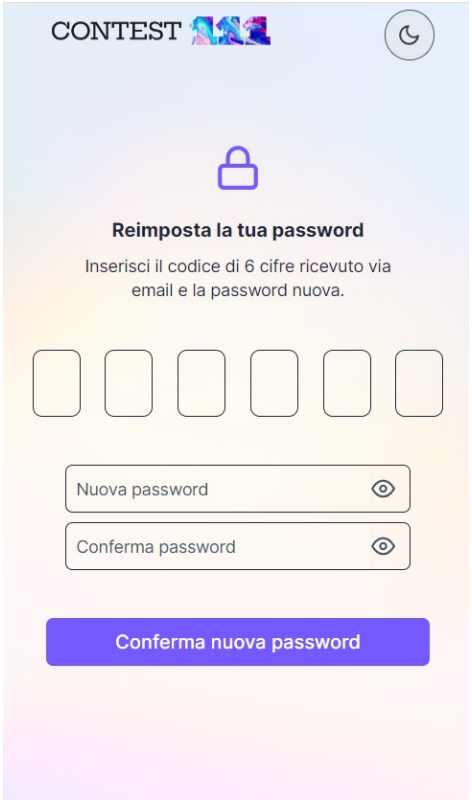



**Figura 4.19:** Visualizzazione della pagina di recupero della password

## 4.7 Profilo

In Figura 4.20 e Figura 4.21 è possibile vedere le pagine relative al profilo personale dell'utente. Ogni utente visualizza e può modificare la propria immagine profilo, il nome impostato, la mail e una breve bio. L'utente poi potrà visualizzare le opere d'arte caricate e i contest a cui ha partecipato.

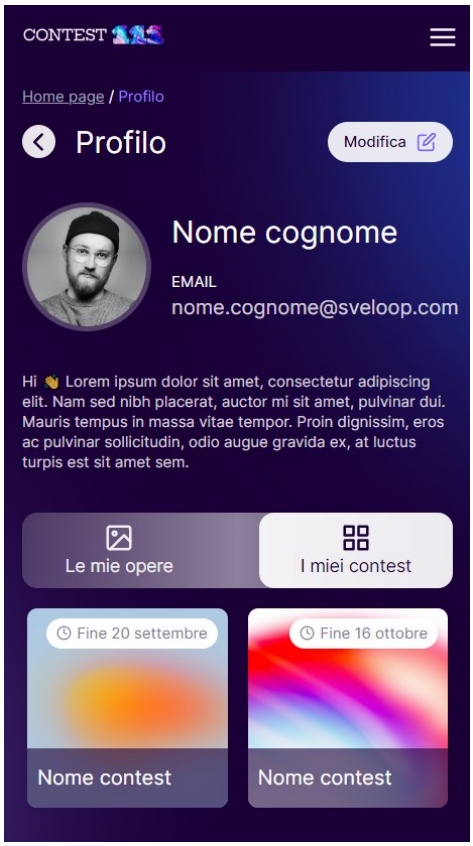



**Figura 4.20:** Visualizzazione della pagina del profilo

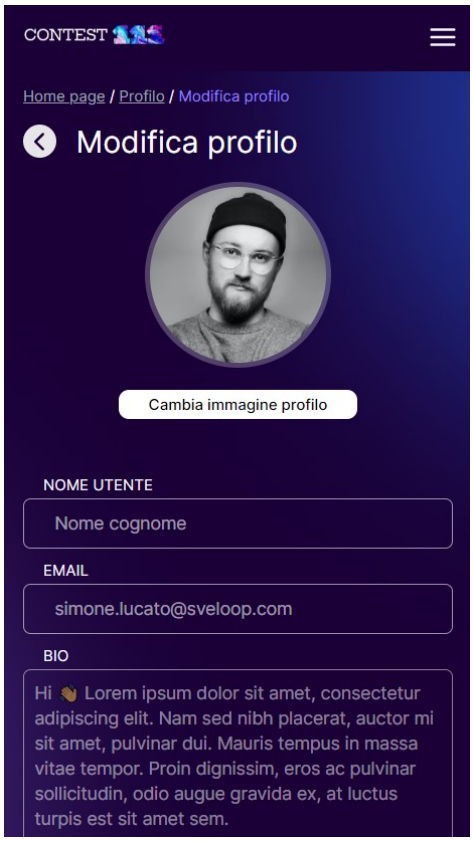



**Figura 4.21:** Visualizzazione della pagina di modifica dati personali

## 4.8 Impostazioni

In Figura 4.22 è possibile modificare le impostazioni relative al proprio account. Sarà possibile gestire i metodi di pagamento e gli indirizzi, effettuando una chiamata all'API di Stripe o cambiare la valuta dell'applicazione. È possibile modificare la password del proprio account o cambiare la lingua in cui visualizzare l'applicazione.
Per la realizzazione del menù a tab è stato assegnato un `role="tablist"` al tag `<ul>` contenente il menù, un `role="tab"` a ogni `<li>` contenente le voci del menù e un `role="tablist"` a ogni `<div>` per il contenuto di ciascuna pagina. Ogni `<li>` ha come attributo `aria-selected="true"` se è l'elemento attivo, altrimenti è `false`; la sezione corrispondente avrà `aria-hidden="false"` e `aria-expanded="true"` se è selezionata, altrimenti il contrario. Ogni `<li>` inoltre ha la propria posizione all'interno della lista marcata da `aria-posinset="index"` e l'attributo `aria-controls="id"`, dove id è l'id della sezione che controllano (la sezione invece avrà `aria-labelledby="id"` dove id corrisponde a quello dell'item che la attiva) [3].

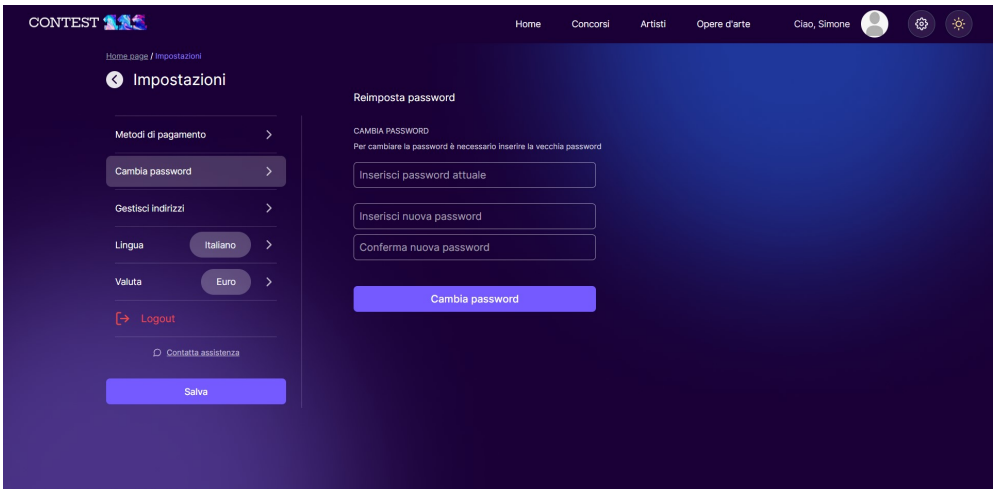



**Figura 4.22:** Visualizzazione della pagina relativa alle impostazioni

## 4.9 Uso dei context

React a partire alla versione v16.3 mette a disposizione diversi strumenti per condividere dati all'interno dell'applicazione, in primis i `context`. Generalmente questi `context` sono oggetti (anche se possono essere una semplice stringa volendo) usati per rendere disponibile a tutta l'applicazione i dati contenuti al loro interno [25]. Vi sono tre step fondamentali per l'uso di un `context`:

- **Creazione**: tramite `createContext(default)`, l'apposito metodo fornito da React, si crea un'istanza di contesto come esemplificato nello Snippet di codice 4.4 ;

```
1  import { createContext } from "react";
2  const Context = createContext("Default Value");
```

**Snippet di codice 4.4:** Esempio di creazione di un context

- **Fornitura**: si procede ad avvolgere l'applicazione nel `Provider` fornito all'interno del `context` stesso, come mostrato nello Snippet di codice 4.5, fornendo il valore che si desidera esporre. Generalmente, si fa uso di un componente *wrapper*, come mostrato nello Snippet di codice 4.7;

```
1  function Main() {
2    const value = "Specific Value";
3    return (
4      <Context.Provider value={value}>
5        <MyComponent />
6      </Context.Provider>
7    );
8  }
```

**Snippet di codice 4.5:** Esempio di fornitura di un context

– **Consumo**: nonostante sia possibile procedere a consumare il valore fornito tramite un apposito `Context.Consumer`, generalmente l'approccio seguito è quello di utilizzare `useContext(context)`, l'*hook* fornito da React. Per fare ciò si può ricevere direttamente il valore (o fare *object destructuring* nel caso di oggetti) usando l'*hook*, come mostrato nello Snippet di codice 4.6

```
1       import { useContext } from "react";
2
3       function MyComponent() {
4         const value = useContext(Context);
5
6         return <span>{value}</span>;
7       }
```

**Snippet di codice 4.6:** Esempio di consumo di un context

Durante lo sviluppo sono stati usati tre `context`: uno per l'autenticazione, uno per l'internazionalizzazione e uno per il tema. Ognuno è stato realizzato come *wrapper* ossia sono componenti funzionali che wrappano i `children`, gli elementi figli del componente, come riportato nello Snippet di codice 4.7

```
1   import { createContext, FC } from "react";
2
3   const defaultValue = { ... }
4   export const Context = createContext(defaultValue);
5
6   export const ContextWrapper : FC = ({ children }) => {
7
8     return (
9       <Context.Provider value={...}>
10        {children}
11      </Context.Provider};
12    )
13  }
```

**Snippet di codice 4.7:** Esempio di wrapper

## AuthContextWrapper

`AuthContextWrapper.tsx` è il `context` usato per la gestione dell'autenticazione. Nello Snippet di codice 4.8 è illustrato l'oggetto che rappresenta il `context`:

```
1     {
2         isLoggedIn: boolean,
3         login: () => void,
4         logout: () => void,
5         userData?: UserAttributes,
6     }
```

**Snippet di codice 4.8:** Context di AuthContextWrapper

`isLoggedIn` e `userData` sono stati all'interno del *wrapper* che vengono aggiornati al login e al logout. `isLoggedIn` è il valore usato per controllare se un utente è loggato o meno nel momento cerca di accedere a una pagina che richiede di essere loggato, come ad esempio la pagina di caricamento di un'opera d'arte.

### IntlContextWrapper

`IntlContextWrapper` è il `context` usato per la gestione dell'internazionalizzazione. Nello Snippet di codice 4.9 è illustrato l'oggetto che rappresenta il `context`:

```
1    {
2        locale: LanguageType,
3        switchLanguage: (language: LanguageType) => void,
4    }
```

**Snippet di codice 4.9:** Context di IntlContextWrapper

dove `type LanguageType = "it" | "en"` è un tipo contenente tutte le lingue supportate. Per la gestione delle traduzioni si è fatto riferimento alla libreria `react-intl` che si basa sull'uso di coppie chiave-valore in un file JSON per la traduzione. Per aggiungere del testo è sufficiente seguire i seguenti passaggi:

- Aggiungere un testo nel file JSON relativo alla lingua appropriata, assegnandogli un id univoco;
- Fornire le traduzioni sugli altri file relativi alle altre lingue, usando lo stesso id;
- Quando è necessario usare il testo, riportarlo nel seguente modo

  `<FormattedMessage id=id_testo/>`

  dove `id_testo` corrisponde all'id univoco assegnato.

Tuttavia, ogni tanto questo può dare problemi perché può essere richiesto come paramentro specificamente una stringa (e `<FormattedMessage />` è un componente (come nel caso delle `aria-label`). In questo caso si può usare un *hook* fornito dalla libreria, `useIntl`, come riportato nello Snippet di codice 4.10:

```
1    const intl = useIntl()
2    const value = intl.formatMessage({ id: "Identificativo" }) //
         value e' di tipo stringa
```

**Snippet di codice 4.10:** Esempio d'uso di useIntl

All'interno del *wrapper*, non appena viene avviata l'applicazione si recupera la lingua usata dall'utente analizzando `navigator.language` [14], dove navigator rappresenta lo stato e l'identità dello user agent (in questo caso il *browser*). Il valore ritornato sarà del tipo `it-IT` o `it` secondo la raccomandazione BCP 47 [20]. Si verificherà quindi se questa lingua rientra in quelle di cui è disponibile la traduzione. Se è disponibile si cambierà lingua tramite il metodo `switchLanguage`, altrimenti si resterà con la lingua predefinita, ossia l'inglese. La stessa libreria verrà inoltre utilizzata per la gestione di diverse valute, essendo presenti compoenenti analoghe come `<FormattedNumer />`.

### ThemeContextWrapper

`ThemeContextWrapper` è il `context` usato per la gestione del tema. Nello Snippet di codice 4.11 è illustrato l'oggetto che rappresenta il `context`:

```
{
    theme: Theme,
    changeTheme: (theme: Theme) => void,
}
```

**Snippet di codice 4.11:** Context di IntlContextWrapper

dove `type Theme = "light" | "dark"` è un tipo che identifica il tema, chiaro o scuro. Il tema di default è chiaro, tuttavia all'avvio dell'applicazione si controlla con `window.matchMedia("(prefers-color-scheme: dark)").matches` se l'utente ha la dark mode inserita nel *browser*. Qui si prende la `window`, ossia la finestra che contiene il DOM e si cerca se il `document` corrisponde la *media query string* passata per parametro (in questo caso `(prefers-color-scheme: dark)`, che analizza le preferenze dell'utente sul tema). A questo punto la proprietà `matches` ritorna `true` se il `document` corrisponde la lista di media query ritornata dal metodo.
Per cambiare il tema si è aggiunto al tag `<html>` la classe `"dark"` e si è impostato il parametro `darkMode:"class"` nel file di configurazione di Tailwind, il framework di CSS. Per impostare una proprietà CSS per il tema scuro sarà sufficiente anteporre alla proprietà `dark:`, ad esempio `dark:text-white` imposta il testo a bianco se è inserita la modalità scura.

## 4.10 Services

Le chiamate di `service` comprendono chiamate a Cognito per l'autenticazione e chiamate GraphQL per interrogare direttamente il database. Per quanto riguarda Cognito, fornisce un SDK[g]per agevolare l'integrazione con le applicazioni. Il modello usato all'interno dell'SDK è abbastanza semplice: si crea un client (in questo caso un `CognitoIdentityProvider`) con la configurazione richiesta; si crea poi il comando che si desidera eseguire e lo si invia. Un esempio viene mostrato nello Snippet di codice 4.12 [6]:

```
// a client can be shared by different commands.
const client = new CognitoIdentityProviderClient({ region: "REGION" });

const params = {
  /** input parameters */
};
const command = new AddCustomAttributesCommand(params);
try {
  const response = await client.send(command);
  // process data
}
catch (error){
  // error handling
}
```

**Snippet di codice 4.12:** Esempio tratto dalla documentazione di Cognito Identity Provider Client

Per quanto riguarda GraphQL, le chiamate vengono gestite con `ApolloClient`, una libreria di *state management* per la gestione di dati con GraphQL. La gestione è molto semplice: si crea un client passandogli l'URL dell'endpoint GraphQL e un'istanza di `InMemoryCache`, usata dal client per salvare i dati dopo averli

recuperati [2]. È possibile aggiungere dei middleware che aggiungano particolari header alla richiesta, ad esempio viene usato un middleware per aggiungere un header di autorizzazione passando l'`AccessToken` per le chiamate che richiedono di essere loggati (ad esempio `me()` o `createUser()`. [1]
Le chiamate al *back-end* effettuate possono riguardare l'autenticazione, i contest e le opere d'arte. Di seguito vengono analizzate nel dettaglio le principali:

### 4.10.1 Autenticazione

L'autenticazione viene gestita con l'ausilio del servizio AWS Cognito, in quanto fornisce numerosi strumenti per agevolare lo sviluppo (ad esempio il supporto per fare l'accesso con provider d'identità social come Google, Facebook e Apple, l'uso di bacini d'utenza facilmente configurabili e standard di sicurezza *out-of-the-box* molto elevati). Viene inoltre usato GraphQL per recuperare le informazioni relative all'utente o agli artisti (con costi e latenza inferiore rispetto a Cognito). Di seguito vengono analizzate le principali chiamate.

#### Login

- **Tipologia di chiamata**: Cognito;
- **Parametri di input**: email e password, entrambe stringhe;
- **Contenuto della funzione**: creazione di `InitiateAuthCommand` passando come parametri in un oggetto `AuthFlow` (la tipologia di flusso di autenticazione, `USER_PASSWORD_AUTH` in questo caso), il ClientId di AWS e i parametri di autenticazione (email e password) [7]. Dopo aver inviato il comando, se la risposta ritorna l'`AccessToken` questo viene salvato nel `localStorage`. Viene poi effettuata la chiamata `me()` (analizzata nell'apposita sezione) e a seconda della risposta si ritorna un oggetto diverso;
- **Oggetto ritornato**:
  - Se la chiamata `me()` ritorna un oggetto si ritorna

```
{
  success: true,
  isAccountActive: true,
}
```

  - Se la chiamata `me()` ritorna `null` si ritorna

```
{
  success: true,
  isAccountActive: false,
}
```

  - Se viene sollevata un'eccezione si ritorna

```
{
  success: false,
  error: error,
}
```

  Nel primo caso si verrà rimandati alla home page, nel secondo si verrà rimandati alla pagina di conferma account e se l'eccezione sollevata è di tipo `"UserNotConfirmedException"`, allora si verrà rimandati alla pagina di conferma email. Altrimenti verrà semplicemente mostrato un messaggio d'errore.

#### Register

- **Tipologia di chiamata**: Cognito;
- **Parametri di input**: email e password, entrambe stringhe;
- **Contenuto della funzione**: creazione di `SignUpCommand` passando come parametri in un oggetto il ClientId di AWS, l'email e la password e invio del comando tramite il client [8].
- **Oggetto ritornato**:
  - Se l'invio del comando non dà errori, si ritorna

```
{
  success: true,
}
```

  - Se viene sollevata un'eccezione si ritorna

```
{
  success: false,
  error: error,
}
```

#### Me

La chiamata `me()` viene usata per ritornare tutte le informazioni relative all'utente attualmente collegato.

- **Tipologia di chiamata**: GraphQL;
- **Parametri di input**: nessuno;
- **Contenuto della funzione**: invio della seguente query tramite client autorizzato [1]:

```
query {
  me {
    id                  // id univoco
    name                // nome
    surname             // cognome
    username            // username
    email               // email
    fiscal_code         // codice fiscale
    gender              // sesso
    profile_image {     // immagine profilo
        URL             // URL dell'immagine profilo
        title           // testo alternativo
    }
    is_dark_mode        // preferenza sulla dark mode
    language            // lingua selezionata
    currency            // valuta selezionata
  }
}
```

- **Oggetto ritornato**:
  - Se la query non solleva eccezioni, si ritorna

```
{
  success: true,
  data,             // l'oggetto contiene tutte le
      informazioni richieste sopra
}
```

* Se la query ritorna errori, si ritorna

```
{
  success: false,
  error: errors,
}
```

**createUser**

- **Tipologia di chiamata**: GraphQL;
- **Parametri di input**: name, surname, username, email, fiscal_code, gender, is_dark_mode. Sono tutte stringhe tranne is_dark_mode che è booleano;
- **Contenuto della funzione**: invio della seguente query tramite client autorizzato [1]:

```
mutation {
    createUser(
        name: "${name}",
        surname: "${surname}",
        username: "${username}",
        email: "${email}",
        fiscal_code: "${fiscal_code}",
        gender: "${gender}",
        is_dark_mode: ${is_dark_mode},
        language: "${language}",
        currency: "EUR",
    ) {
        id                  // id univoco
        name                // nome
        surname             // cognome
        username            // username
        email               // email
        fiscal_code         // codice fiscale
        gender              // sesso
        profile_image {     // immagine profilo
            URL             // URL dell'immagine profilo
            title           // testo alternativo
        }
        is_dark_mode        // preferenza sulla dark mode
        language            // lingua selezionata
        currency            // valuta selezionata
    }
 }
```

- **Oggetto ritornato**:

  * Se la query non solleva eccezioni, si ritorna

```
{
  success: true,
  data,         // l'oggetto contiene tutte le informazioni
      richieste sopra
}
```

  * Se la query ritorna errori, si ritorna

```
{
  success: false,
  error: errors,
}
```

### 4.10.2 Contest

#### getContests

I contest sono paginati, quindi viene utilizzato un token generato da GraphQL per gestire la paginazione. Non passando nessun token vengono ritornati i primi `limit` risultati (dove `limit` è settato di default a 10 ma può essere cambiato). Con ogni chiamata viene ritornato un token (se vi sono ancora risultati disponibili) e passando quel token vengono ritornati i risultati successivi.

- **Tipologia di chiamata**: GraphQL;
- **Parametri di input**: token (di default a stringa vuota) e limit (di default a 10);
- **Contenuto della funzione**: invio della seguente query:

```
query {
  getContests(limit: ${limit}, nextToken: "${token}") {
      nextToken                       // token per paginazione
      contests {                      // elenco di contest
          id                          // id univoco
          title                       // titolo dell'opera
          description                 // descrizione
          image {                     // array di immagini
              URL                     // URL dell'immagine
              title                   // testo alternativo
          }
          cover_image {               // immagine di copertina
              URL                     // URL dell'immagine
              title                   // testo alternativo
          }
          subscription_date_start     // data di inizio
              iscrizioni
          subscription_date_end       // data di fine
              iscrizioni
          contest_date_start          // data di inizio contest
          contest_date_end            // data di fine contest
          price                       // prezzo di iscrizione
          additional_price            // prezzo per l'
              iscrizione di opere aggiuntive
          stripe_price_id             // riferimento di Stripe
              per il prezzo di iscrizione
          stripe_additional_price_id  // riferimento di Stripe
              per il prezzo di iscrizione di opere aggiuntive
      }
  }
}
```

- **Oggetto ritornato**:
  - Se la query non solleva eccezioni, si ritorna

```
{
  success: true,
  contests,     // un array contenente le informazioni
      indicate sopra
  nextToken,    // token per la paginazione
}
```

  - Se la query ritorna errori, si ritorna

```
{
  success: false,
  error: errors,
}
```

**getContestById**

- **Tipologia di chiamata**: GraphQL;
- **Parametri di input**: id del contest;
- **Contenuto della funzione**: invio della seguente query:

```
query {
    getContest(contest_id: "${id}") {
        id                          // id univoco
        title                       // titolo dell'opera
        description                 // descrizione
        image {                     // array di immagini
            URL                     // URL dell'immagine
            title                   // testo alternativo
        }
        cover_image {               // immagine di copertina
            URL                     // URL dell'immagine
            title                   // testo alternativo
        }
        subscription_date_start     // data di inizio
            iscrizioni
        subscription_date_end       // data di fine iscrizioni
        contest_date_start          // data di inizio contest
        contest_date_end            // data di fine contest
        price                       // prezzo di iscrizione
        additional_price            // prezzo per l'iscrizione
            di opere aggiuntive
        stripe_price_id             // riferimento di Stripe
            per il prezzo di iscrizione
        stripe_additional_price_id  // riferimento di Stripe
            per il prezzo di iscrizione di opere aggiuntive
    }
}
```

- **Oggetto ritornato**:
  - Se la query non solleva eccezioni, si ritorna

```
{
  success: true,
  contest,          // un oggetto contenente le informazioni
      indicate sopra
}
```

  - Se la query ritorna errori, si ritorna

```
{
  success: false,
  error: errors,
}
```

### 4.10.3 Opere d'arte

#### getArtworks

Anche le opere d'arte vengono paginate allo stesso modo dei contest, quindi la gestione è analoga.

- **Tipologia di chiamata**: GraphQL;
- **Parametri di input**: token (di default a stringa vuota) e limit (di default a 10);
- **Contenuto della funzione**: invio della seguente query:

```
query {
  getArtworks(limit: ${limit}, nextToken: "${token}") {
    nextToken           // token per la paginazione
    artworks {          // array di opere
      id                // id univoco
      title             // titolo dell'opera
      images {          // array di immagini
          URL           // URL dell'immagine
          title         // testo alternativo
          is_default    // se e' l'immagine di copertina o meno
      }
    }
  }
}
```

- **Oggetto ritornato**:
  - Se la query non solleva eccezioni, si ritorna

```

{
  success: true,
  artworks,     // un array contenente le informazioni
      indicate sopra
  nextToken,    // token per la paginazione
}
```

  - Se la query ritorna errori, si ritorna

```
{
  success: false,
  error: errors,
}
```

#### getArtworksById

- **Tipologia di chiamata**: GraphQL;
- **Parametri di input**: id dell'opera d'arte;
- **Contenuto della funzione**: invio della seguente query:

```
query {
  getArtwork(artwork_id: "${id}") {
    id                // id univoco
    user_id           // id dell'artista
    title             // titolo dell'opera
    description       // descrizione
    images {          // array di immagini
```

```
        URL             // URL dell'immagine
        title           // testo alternativo
        is_default      // se e' l'immagine di copertina o meno
    }
    width               // lunghezza dell'opera
    height              // altezza dell'opera
    depth               // larghezza dell'opera
    weight                // peso dell'opera
    materials           // array di materiali
    colors              // array di colori
    tags                // array di tags
  }
}
```

- **Oggetto ritornato**:
  - Se la query non solleva eccezioni, si ritorna

```

{
  success: true,
  artwork,        // oggetto contenente le informazioni
      indicate sopra
}
```

  - Se la query ritorna errori, si ritorna

```
{
  success: false,
  error: errors,
}
```

#### getArtworksByUserId

- **Tipologia di chiamata**: GraphQL;
- **Parametri di input**: id dell'artista, token (di default a stringa vuota) e limit (di default a 10);
- **Contenuto della funzione**: invio della seguente query:

```
query {
  getArtworksByUserId(user_id: "${id}", limit: ${limit},
      nextToken: "${token}") {
    nextToken           // token per la paginazione
    artworks {          // array di opere
      id                // id univoco
      title             // titolo dell'opera
      images {          // array di immagini
          URL           // URL dell'immagine
          title         // testo alternativo
          is_default    // se e' l'immagine di copertina o meno
      }
    }
  }
}
```

- **Oggetto ritornato**:
  - Se la query non solleva eccezioni, si ritorna

```
1
2 {
3   success: true,
4   artworks,     // un array contenente le informazioni
        indicate sopra
5   nextToken,    // token per la paginazione
6 }
```

* Se la query ritorna errori, si ritorna

```
1 {
2   success: false,
3   error: errors,
4 }
```

**createArtwork**

- **Tipologia di chiamata**: GraphQL;
- **Parametri di input**: di obbligatori vi è il titolo, la descrizione, l'anno dell'opera e almeno un'immagine (all'interno di un array). Poi vi sono le dimensioni dell'opera (`width`, `height` e `depth`) e il peso (`weight`), un array di materiali, uno di colori e uno di tag relativi;
- **Contenuto della funzione**: invio della seguente query:

```
1  mutation {
2      createArtwork(
3          title: "${title}"
4          description: "${description}"
5          year: ${year}
6          images: ${images}
7          width: ${width}
8          height: ${height}
9          depth: ${depth}
10         weight: ${weight}
11         materials: ${materials}
12         colors: ${colors}
13         tags: ${tags}
14     ) {
15         artwork_id    // id dell'opera
16     }
17 }
```

- **Oggetto ritornato**:
  * Se la query non solleva eccezioni, si ritorna

```
1
2 {
3   success: true,
4   artwork_id,     // id dell'opera
5 }
```

  * Se la query ritorna errori, si ritorna

```
1 {
2   success: false,
3   error: errors,
4 }
```

# Capitolo 5

# Verifica e validazione

## 5.1 Verifica

### 5.1.1 Analisi statica

L'analisi statica fa uso di tecniche che non richiedono l'esecuzione del *software*, basandosi per la maggior parte sulla lettura del codice stesso. Quest'attività è stata svolta per tutta la durata del periodo di codifica. I modi in cui questa attività sono stati effettuati sono:

- leggendo attentamente più volte il codice scritto, facendo uso di walkthrough[g]e inspection[g];
- monitorando costantemente la qualità del codice grazie a strumenti di analisi statica del codice come ESLint (installato di default insieme a Next.js). La configurazione di questo file è stata definita seguendo le best pratice per la scrittura del codice React su TypeScript.

### 5.1.2 Analisi dinamica

L'analisi dinamica prevede l'esecuzione del prodotto *software*. Non essendovi stato molto tempo per la stesura di test (per ragioni spiegate nella sezione 6.1), quest'attività è stata svolta principalmente eseguendo il prodotto sull'ambiente di `development` impostato. Qui si è verificato il corretto flusso degli eventi previsto durante l'esecuzione dei diversi casi d'uso, sia simulando il comportamento atteso dall'utente sia cercando appositamente di far fallire l'applicazione, in modo da verificare che non siano presenti *bug* o comportamenti imprevisti dal parte dell'applicazione.

## 5.2 Validazione

Lo scopo del processo di validazione è accertare che il prodotto finale corrisponda alle attese e che soddisfi tutti i requisiti previsti. Non essendo l'applicazione ancora terminata, la validazione è stata principalmente interna, simulando l'uso del

prodotto da parte dell'utente e verificando che tutte le funzionalità implementate funzionassero correttamente.
La validazione esterna, effettuata dal cliente e dalla sua utenza, verrà effettuata non appena l'applicazione verrà rilasciata in produzione (si approfondirà questo aspetto nella sezione 6.1). Nel mentre, il cliente viene costantemente aggiornato delle modifiche e dello stato di avanzamento dei lavori e ha già avuto modo di vedere lo stato attuale raggiunto.

# Capitolo 6

# Conclusioni

## 6.1 Raggiungimento degli obiettivi

Lo scopo del progetto di stage era quello di realizzare la parte *front-end* di Contest111, un'applicazione di concorsi di opere d'arte, in cui un utente può registrarsi, caricare la propria opera d'arte, iscrivendola ad un concorso e altri utenti possono votarla. Era prevista la realizzazione di diverse schermate, l'implementazione del recupero di informazioni da *back-end*, della logica di autenticazione e della gestione dei pagamenti (necessari all'iscrizione di un'opera d'arte a un concorso e al voto). In questa sezione si analizza lo stato attuale raggiunto dall'applicazione rispetto alle previsioni di inizio stage.

### 6.1.1 Completamento degli obiettivi

Con riferimento alla Tabella 1.1 riportante gli obiettivi di inizio stage, si riportano in Tabella 6.1 gli stessi obiettivi con aggiunto lo stato di completamento:

| Identificativo | Descrizione | Stato |
|---|---|---|
| O1 | Stesura documento analisi dei requisiti | Completato |
| O2 | Creazione e setting ambiente di sviluppo | Completato |
| O3 | Implementazione delle viste relative all'autenticazione | Completato |
| O4 | Implementazione delle viste relative alle opere d'arte, ai concorsi e agli artisti | Completato |
| O5 | Implementazione del *fetch* di informazioni da *back-end* usando GraphQL | Completato |
| O6 | Implementazione della gestione dei pagamenti | Completato |
| D1 | Implementazione della piattaforma di admin | Non completato |

**Tabella 6.1:** Stato di completamento degli obiettivi dello stage

Perciò sono stati completati tutti gli obiettivi obbligatori previsti per lo stage, ma non è stato possibile completare l'obiettivo desiderabile.

### 6.1.2 Soddisfacimento dei requisiti

Con riferimento alla Tabella 2.1 e alla Tabella 2.2 riportanti rispettivamente gli obiettivi funzionali e di qualità, si riportano in Tabella 6.2 e in Tabella 6.3 gli stessi requisiti con aggiunto lo stato di soddisfacimento.

| Requisito | Descrizione | Stato |
|---|---|---|
| RFO1 | L'utente deve poter vedere la lista completa di opere d'arte | Soddisfatto |
| RFO2 | L'utente deve poter vedere i dettagli di una singola opera d'arte | Soddisfatto |
| RFO3 | L'utente deve poter vedere la lista completa di artisti | Soddisfatto |
| RFO4 | L'utente deve poter vedere i dettagli di un singolo artista | Soddisfatto |
| RFO5 | L'utente deve poter vedere la lista completa di concorsi (divisi in attivi, futuri e passati) | Soddisfatto |
| RFO6 | L'utente deve poter vedere i dettagli di un singolo concorso | Soddisfatto |
| RFO7 | L'utente deve poter filtrare le opere d'arte | Non soddisfatto |
| RFO8 | L'utente deve poter vedere le opere d'arte filtrate | Non soddisfatto |
| RFO9 | L'utente deve poter avere accesso alla piattaforma | Soddisfatto |
| RFO10 | L'utente deve poter creare un account | Soddisfatto |
| RFO11 | L'utente deve poter votare un'opera d'arte, previo pagamento | Non soddisfatto |
| RFO12 | L'utente deve caricare un'opera d'arte, previo pagamento | Non soddisfatto |
| RFO13 | L'utente deve poter effettuare pagamenti verso la piattaforma | Soddisfatto |
| RFO14 | L'utente deve poter visualizzare le informazioni relative al suo profilo | Soddisfatto |
| RFO15 | L'utente deve poter modificare le informazioni relative al suo profilo | Soddisfatto |
| RFO16 | L'utente deve poter visualizzare le impostazioni relative al suo profilo | Soddisfatto |
| RFO17 | L'utente deve poter modificare le impostazioni relative al suo profilo | Soddisfatto |
| RFO18 | L'utente deve poter effettuare il logout | Soddisfatto |
| RFO19 | L'applicazione dev'essere responsive | Soddisfatto |
| RFD1 | L'admin deve poter vedere dei report statistici sulla piattaforma | Non soddisfatto |
| RFD2 | L'admin deve poter vedere la lista di opere segnalate | Non soddisfatto |
| RFD2 | L'admin deve poter rispondere alla segnalazione di un'opera | Non soddisfatto |
| RFD4 | L'admin deve poter creare un nuovo concorso | Non soddisfatto |

**Tabella 6.2:** Stato di soddisfacimento dei requisiti funzionali dell'applicazione

| Requisito | Descrizione | Stato |
|---|---|---|
| RQO1 | L'interfaccia utente deve essere realizzata in lingua inglese | Soddisfatto |

**Tabella 6.3:** Stato di soddisfacimento dei requisiti di qualità dell'applicazione

Come si può notare, al termine dello stage sono stati soddisfatti 16 requisiti obbligatori su 20, ossia l'80% e 0 su 4 requisiti desiderabili. In particolare non sono stati soddisfatti i requisiti relativi alla possibilità di filtrare le opere d'arte (RFO7 e RFO8), quelli relativi al voto (RFO11), quelli relativi al caricamento di un'opera (RFO12) e quelli relativi alla piattaforma di admin (RFD1, RFD2, RFD3, RFD4).
Le ragioni di questo sono principalmente due:

- Inizialmente era previsto che la stesura della parte grafica del codice fosse delegata a un *software*, Amplify (un servizio offerto da AWS), che doveva prendere il design dell'applicazione realizzato in Figma e tradurlo in codice React. Sono poi sorti problemi nell'utilizzo di questo *software*, in primis di qualità del codice prodotto, giudicata assolutamente non sufficiente. Si è quindi preferito procedere con la codifica anche della parte grafica senza l'ausilio di *software* esterni. Questo ha portato a una qualità molto maggiore del codice prodotto (anche in un'ottica di scalabilità dell'applicazione), ma ha comportato un dispendio temporale non indifferente;
- Una possibile sopravvalutazione del carico di lavoro gestibile nelle tempistiche di stage. Questa eventualità è dovuta al fatto che lo stagista era già lavoratore presso l'azienda presso cui è stato effettuato lo stage. Di conseguenza metodologie e tecnologie erano già in parte conosciute. Essendo già inserito nell'ambiente di lavoro, è possibile che si sia adattato un progetto che richiedesse tempi leggermente maggiori ai tempi previsti dallo stage.

Ciononostante, l'azienda si è comunque detta molto soddisfatta dello stato di avanzamento raggiunto e si prevede di raggiungere un livello compatibile con il rilascio in produzione in meno di tre settimane lavorative. Molto del lavoro che non è stato possibile portare a termine è comunque a un livello molto avanzato e non richiede molto tempo per essere ultimato. Ad esempio il filtro delle opere d'arte è stato impostato da *back-end*, è sufficiente realizzare il componente relativo alla schermata di filtro ed effettuare la chiamata in *back-end* corrispondente; la creazione dell'opera d'arte è pronta, richiede l'aggiunta dello step di pagamento successivo nel caso sia prevista l'iscrizione a un concorso (anche il pagamento è stato implementato correttamente).

## 6.2 Conoscenze acquisite

Nonostante parte delle tecnologie fossero già conosciute al momento di inizio dello stage (essendo già viste nei mesi di lavoro precedenti), quelle che più importanti e che hanno avuto impatto maggiore sono state apprese da zero.
In primis TypeScript, già visto in piccola parte durante il progetto di Ingegneria del Software, è stato usato ben più estensivamente ed è stato particolarmente

apprezzato per i vincoli che possono essere imposti, garantendo maggiore rigore durante lo sviluppo.
Inoltre Next.js, il framework usato per lo sviluppo in React si è rivelato particolarmente comodo per la costruzione da zero di un'applicazione, soprattutto per quanto riguarda la gestione del routing, che risparmia del tempo durante le fasi iniziali della codifica.
Infine Tailwind, il framework usato per il CSS si è rivelato lo strumento più utile con cui affrontare la scrittura delle componenti grafiche da zero. Dispone di una libreria estremamente chiara e l'uso è incredibilmente intuitivo. Tutto questo senza scendere a compromessi con le funzionalità offerte, anzi offrendo compatibilità anche con quelle non direttamente supportate dal framework.
Un altro aspetto molto importante riguarda la gestione di progetto avendo un proponente esterno all'azienda: fino a questo momento si era lavorato solo su progetti proprietari, quindi soggetti a tempistiche molto più flessibili rispetto a un progetto commissionato da un cliente esterno. In questo modo, è stato possibile vedere concretamente aspetti finora dati per scontati o non approfonditi così a lungo, come un rapporto più diretto con il proponente (e il conseguente cambiamento di requisiti durante lo sviluppo), i tempi più ristretti rispetto a quanto visto finora e la necessità di portare costantemente modifiche tangibili all'applicazione, in modo che possano essere viste e commentate dal cliente e sia possibile adattarle in corso d'opera.

# Glossario

**API** API (Application Program Interface) indica ogni insieme di procedure disponibili al programmatore, generalmente volte alla comunicazione tra diverse componenti del software (ad esempio la ricezioni di informazioni da *back-end* a *front-end*). 2, 7, 23, 38, 57, 59, 61

**AWS** AWS (Amazon Web Services) indica una piattaforma che offre servizi di cloud computing su richiesta, ossia pagando solo quando necessario, senza abbonamenti o licenze. Offre più di 200 servizi disponibili in 26 regioni geografiche in tutto il mondo. 1, 5, 22, 43, 44, 55, 61

**Bootstrap** Bootstrap è un framework CSS *open source* per lo sviluppo di applicazioni web *mobile-first*. Fornisce templates HTML, CSS e JavaScript per diversi componenti (come form e bottoni). 7

**Cloud** Il cloud è generalmente considerato come uno spazio di archiviazione personale. Tuttavia fa anche riferimento al cloud computing, ossia un modo di erogazione di servizi (tra cui la memorizzazione dati, da cui deriva l'accezione comune) on-demand, ossia basato sulla richiesta secondo un modello pay-as-you-go. Di solito questi servizi vengono forniti da aziende con grandi disponibilità di data center in tutto il mondo. 1

**DOM** DOM (Document Object Model) è un'interfaccia di programmazione per documenti web. Rappresenta la pagina in modo che sia possibile cambiare la struttura, il contenuto o lo stile del documento. 6, 42, 58, 61

**Endpoint** Un endpoint di un API è un URL che fornisce la posizione di una risorsa sul server dell'API. In questo "luogo", l'API riceve la richiesta e invia la risposta adeguata. 7

**HOC** Un HOC, ossia un componente d'ordine superiore, è una tecnica usata in React per riutilizzare la logica dei componenti. A differenza dei componenti normali che prendono delle props e ritornano un componente, gli HOC prendono come parametri di costruzione dei componenti e ritornano un nuovo componente con più funzionalità rispetto al precedente. 29, 30, 61

**IDE** IDE (Integrated Development Environment) è un software di supporto alle attività di sviluppo e debugging. Generalmente identifica errori di sintassi e raggruppa più tool in un unico software come un editor, un compilatore e/o interprete, un debugger ecc... 2

**Inspection** Con inspection si intende una tecnica di analisi statica è consiste in una lettura mirata della sezione che è necessario analizzare (solitamente si tratta di codice). La lettura è mirata perché basata su una lista di errori più comuni commessi ottimizzando i tempi. Questa lista nel corso del tempo verrà estesa, aumentando l'esperienza, rendendola ancora più efficace. 51, 59

**JSX** JSX, ossia JavaScript XML è un'estensione della sintassi JavaScript usata per produrre codice che rappresenti elementi React. In questo modo è possibile posizionare questi elementi nel DOM senza chiamare metodi aggiuntivi. 6, 27, 61

**MFA** La MFA, o autenticazione a più fattori, è un metodo di autenticazione usato per garantire accesso a un sito o piattaforma dopo che ha fornito più evidenze della sua identità. Oltre al login, i metodi più comuni per autenticarsi prevedono l'inserimento di un codice generato casualmente e inviato via SMS, via mail o attraverso un'applicazione terza. Questo garantisce maggiore sicurezza per l'utente contro possibili violazioni del proprio account. 23, 61

**Node.js** Node.js è un runtime system (cioé un software che fornisce i servizi necessari all'esecuzione di un programma) *open source* per l'esecuzione di codice JavaScript da eseguire lato server. 6, 8

**NoSQL** NoSQL indica una tipologia di database non relazionali, ossia non basati su uno schema di righe e colonne, ma su altre tipologie di archiviazione dati a seconda di quello che serve memorizzare (il più diffuso è quello chiave-valore). Tra le ragioni di utilizzo vi è una grande flssibilità (rendendolo ideale per dati non strutturati), la scalabilità e le elevate prestazioni. 5, 22

**Props** In React, le componente possono essere personalizzate tramite parametri passati in fase di costruzione, definiti *props*. Questi parametri vengono passati dalla componente padre alla componente figlia e sono immutabili all'interno della componente figlia. 27, 57

**Redux** Redux è una libreria di state management basata su uno store centrale, accessibile a tutta l'applicazione, per memorizzare le informazioni richieste. È possibile accedere ai contenuti memorizzati tramite un selettore ed è possibile modificare questi contenuti facendo il *dispatch* di un'azione. Quest'azione viene processata da una funzione di riduzione che ne analizza il tipo e ritorna lo stato aggiornato dello store centrale. 22, 23

**Routing** Il routing nel contesto dello sviluppo web è un meccanismo per cui a un dato URL corrisponde il nodo di componenti UI che costituiscono una pagina. In questo modo si dà maggiore struttura al design e all'implementazione di un'applicazione web, rendendo più semplice la comprensione per lo sviluppatore e la navigazione per l'utente. 7, 23, 55

**Scrum** Scrum è tra i metodi Agili più diffusi che si fonda sulla divisione del carico di lavoro in *sprints* di tempi brevi che portino alla realizzazione di prodotto software. 1

**SDK** Un SDK (Software Development Kit) è una collezione di strumenti di sviluppo software disponibile in un pacchetto unico. Agevolano la creazione di applicazioni avendo un compilatore, un debugger e volte un framework o delle API. Di solito contengono tutto quello che serve per guidare lo sviluppatore nella programmazione. 42, 61

**SEO** SEO (Search Engine Optimization) è un insieme di procedure volte al miglioramento del posizionamento di un contenuto di un sito web nella pagina dei risultati del motore di ricerca. In questo modo è possibile ricevere un volume maggiore di traffico organico. 6, 59, 61

**SPA** SPA (Single Page Application) indica un'applicazione web che carica un'unico documento e che aggiorna il contenuto tramite chiamate API, generalmente in risposta ad azioni dell'utente. Questo porta a migliori performance e un'esperienza utente migliore, al prezzo di una gestione più onerosa della SEO o la necessità di implementare la navigazione tra pagine. 6, 22, 61

**UI** La UI, o interfaccia utente, fa riferimento all'interfaccia uomo-macchina (generalmente un computer o un telefono) e più in generale all'interazione tra queste due entità. Lo scopo di questa interazione è il raggiungimento dello scopo da parte dell'umano che opera sulla macchina, mentre la macchina dovrebbe aiutare nel processo di raggiungimento di questo obiettivo. Concetti strettamente legati a questo aspetto includono la facilità di utilizzo e l'efficienza in modo da produrre l'output desiderato col minimo input richiesto. 23, 58, 61

**Walkthrough** Con walkthrough (letteralmente "camminare attraverso") si intende una tecnica di analisi statica che consiste nel controllo dell'intera sezione che è necessario analizzare. Generalmente questa tecnica viene usata nelle fasi iniziali e si prosegue poi con metodi analisi più efficaci, come l'inspection, dopo aver redatto una lista di errori più comuni. 51

**WCAG** Le WCAG, ossia le linee guida per l'accessibilità dei contenuti web, sono delle linee guida pubblicate dal Web Accessibility Initiative (che fa parte del World Wide Web Consortium, un'organizzazione che stabilisce standard tecnici nell'uso del web). Queste linee guida servono di supporto agli sviluppatori in modo da creare contenuti accessibili per le persone con disabilità. 29, 61

# Acronimi

**API** Application Program Interface. 57
**AWS** Amazon Web Services. 57

**DOM** Document Object Model. 57

**HOC** Higher Order Component. 57

**IDE** Integrated Development Environment. 57

**JSX** JavaScript XML. 58

**MFA** Multi Factor Authentication. 58

**SDK** Software Development Kit. 59
**SEO** Search Engine Optimization. 59
**SPA** Single Page Application. 59

**UI** User Interface. 59

**WCAG** Web Content Accessibility Guidelines. 59

# Sitografia

## Siti web consultati


[1] *Apollo Client: Advanced HTTP networking*. URL: https://www.apollographql.com/docs/react/networking/advanced-http-networking/ (cit. alle pp. 43–45).

[2] *Apollo Client: Get started*. URL: https://www.apollographql.com/docs/react/get-started (cit. a p. 43).

[3] *ARIA Authoring Practices Guide: Example of Tabs with Automatic Activation*. URL: https://www.w3.org/WAI/ARIA/apg/example-index/tabs/tabs-automatic.html (cit. a p. 38).

[4] *AWS: Cognito*. URL: https://aws.amazon.com/it/cognito/ (cit. a p. 23).

[5] *AWS: Database NoSQL*. URL: https://aws.amazon.com/it/nosql/ (cit. a p. 22).

[6] *Cognito Identity Provider Client: Getting started*. URL: https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-cognito-identity-provider/index.html#getting-started (cit. a p. 42).

[7] *Cognito Identity Provider Client: Initiate Auth Command*. URL: https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-cognito-identity-provider/classes/initiateauthcommand.html (cit. a p. 43).

[8] *Cognito Identity Provider Client: Sign Up Command*. URL: https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-cognito-identity-provider/classes/signupcommand.html (cit. a p. 44).

[9] *GitHub: Remove React.FC from TypeScript template*. URL: https://github.com/facebook/create-react-app/pull/8177 (cit. a p. 28).

[10] *GlobalStats: Desktop vs Mobile vs Tablet Market Share Italy*. URL: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/italy (cit. a p. 27).

[11] *GlobalStats: Desktop vs Mobile vs Tablet Market Share Worldwide*. URL: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet (cit. a p. 27).

[12] *GraphQL: Documentation*. URL: https://graphql.org/ (cit. a p. 5).

[13] *JavaScript: Object Destructuring*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment (cit. a p. 28).

[14] *Navigator.language*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/language (cit. a p. 41).

[15] *Next.js: API Routes*. URL: https://nextjs.org/docs/api-routes/introduction (cit. a p. 7).

[16] *Next.js: Built-In CSS Support*. URL: https://nextjs.org/docs/basic-features/built-in-css-support (cit. a p. 6).

[17] *Next.js: Pre-rendering*. URL: https://nextjs.org/docs/basic-features/pages#pre-rendering (cit. a p. 6).

[18] *Next.js: Routing*. URL: https://nextjs.org/docs/routing/introduction (cit. alle pp. 7, 23).

[19] *Pure React Carousel Repository*. URL: https://github.com/express-labs/pure-react-carousel (cit. a p. 29).

[20] *Raccomandazione BCP-47 per codici di lingua IETF*. URL: https://datatracker.ietf.org/doc/html/rfc5646 (cit. a p. 41).

[21] *React + TypeScript Cheatsheet: Function components*. URL: https://github.com/typescript-cheatsheets/react/blob/main/README.md#function-components (cit. a p. 27).

[22] *React + TypeScript Cheatsheet: Types vs Interfaces*. URL: https://github.com/typescript-cheatsheets/react/blob/main/README.md#useful-table-for-types-vs-interfaces (cit. a p. 27).

[23] *React + TypeScript Cheatsheets*. URL: https://github.com/typescript-cheatsheets/react/blob/main/README.md.

[24] *React Hook Form: useForm documentation*. URL: https://react-hook-form.com/api/useform (cit. a p. 33).

[25] *React: Context documentation*. URL: https://it.reactjs.org/docs/context.html (cit. a p. 39).

[26] *React: Controlled components*. URL: https://reactjs.org/docs/forms.html#controlled-components (cit. a p. 33).

[27] *React: HOC documentation*. URL: https://reactjs.org/docs/higher-order-components.html (cit. a p. 29).

[28] *React: Hooks overview*. URL: https://reactjs.org/docs/hooks-overview.html (cit. a p. 6).

[29] *React: Introducing JSX*. URL: https://reactjs.org/docs/introducing-jsx.html (cit. a p. 6).

[30] *React: Recommended toolchains*. URL: https://reactjs.org/docs/create-a-new-react-app.html#recommended-toolchains (cit. a p. 6).

[31] *React: useEffect documentation*. URL: https://reactjs.org/docs/hooks-effect.html (cit. a p. 28).

[32] *React: useState documentation*. URL: https://reactjs.org/docs/hooks-state.html#declaring-a-state-variable (cit. a p. 28).

[33] *React: Virtual DOM*. URL: https://reactjs.org/docs/faq-internals.html (cit. a p. 6).

[34] *Redux: Core concepts*. URL: https://redux.js.org/introduction/core-concepts (cit. a p. 23).

[35] *Redux: TypeScript quick start*. URL: https://react-redux.js.org/tutorials/typescript-quick-start (cit. a p. 23).

[36] *StackOverflow: Props as "type" or "interface"*. URL: https://stackoverflow.com/questions/49562569/typed-react-props-as-type-or-an-interface (cit. a p. 27).

[37] *Tailwind: Optimizing for production*. URL: https://tailwindcss.com/docs/optimizing-for-production (cit. a p. 7).

[38] *Tailwind: Utility-first fundamentals*. URL: https://tailwindcss.com/docs/utility-first (cit. a p. 7).

[39] *TypeScript: Documentation*. URL: https://www.typescriptlang.org/ (cit. a p. 4).

[40] *TypeScript: Type aliases vs Interfaces*. URL: https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#differences-between-type-aliases-and-interfaces/ (cit. alle pp. 4, 27).

[41] *TypeScript: Type inference*. URL: https://www.typescriptlang.org/docs/handbook/type-inference.html (cit. a p. 4).