

**Progetto Gruppo T46**

**Chad Gym**

**Documento D4-T46**

**Documento di sviluppo dell’applicazione**

Membri: Andrea Perrone, Michael Mattiolo e Mario Vaccaro.

# Sommario

[**Sommario**](#_cbmpteq2qimd) **2**

[**0. Scopo del documento**](#_q5egflog6d8t) **3**

[**1. User Stories e features**](#_6w27lwz2t0n9) **3**

[User Stories](#_b33cxrknl7z) 3

[Features](#_56vhsd2oufhu) 4

[**2. User Flow**](#_x0eh2vlbhdte) **4**

[**3. Application Implementation and Documentation**](#_e149a79s8g5) **5**

[3.1 Project Structure](#_f955qnxigh84) 5

[3.2 Project Dependencies](#_qxkej9rhtc0e) 7

[3.2.1 Backend dependencies](#_4j1zuht1k9ll) 7

[3.2.2 Frontend dependencies](#_yth5fy44h4q6) 9

[3.3 Project Database](#_d8es7z7967cq) 10

[3.4 Project APIs](#_2x0pgpjj35ak) 11

[**4 Documentazione API**](#_l2x7svxj8xkt) **24**

[**5 Implementazione Front End**](#_853gxfla0jyl) **24**

[Pagine](#_1dialx25xesu) 24

[Componenti](#_rqgy4g2vtjmk) 26

[Servizi](#_rqcaught1nvg) 29

[Pipes](#_sm97ikzhyp3d) 29

[**6 Repository GitHub**](#_vzhabue8l3bf) **30**

[**7 Testing**](#_alrq6r7nbdt6) **30**

# 0. Scopo del documento

Nel seguente documento verrà descritto come è stato sviluppato il progetto, infatti verrà mostrato e descritto parte del codice utilizzato, il modo in cui è stato realizzato il front-end e la repository in cui sarà possibile accedere al codice completo.

# 1. User Stories e features

Nel seguente capitolo è presente una descrizione informale e generica di una caratteristica del software presentata dal punto di vista dell’utente finale. Ogni User Story viene definita con un linguaggio non tecnico, volta a fornire allo sviluppatore un’idea del chi, cosa e del perché si sta sviluppando una specifica funzionalità del software.

Le User Stories legate all’utente generale dell’applicazione sono riportate nella tabella seguente.

## User Stories:

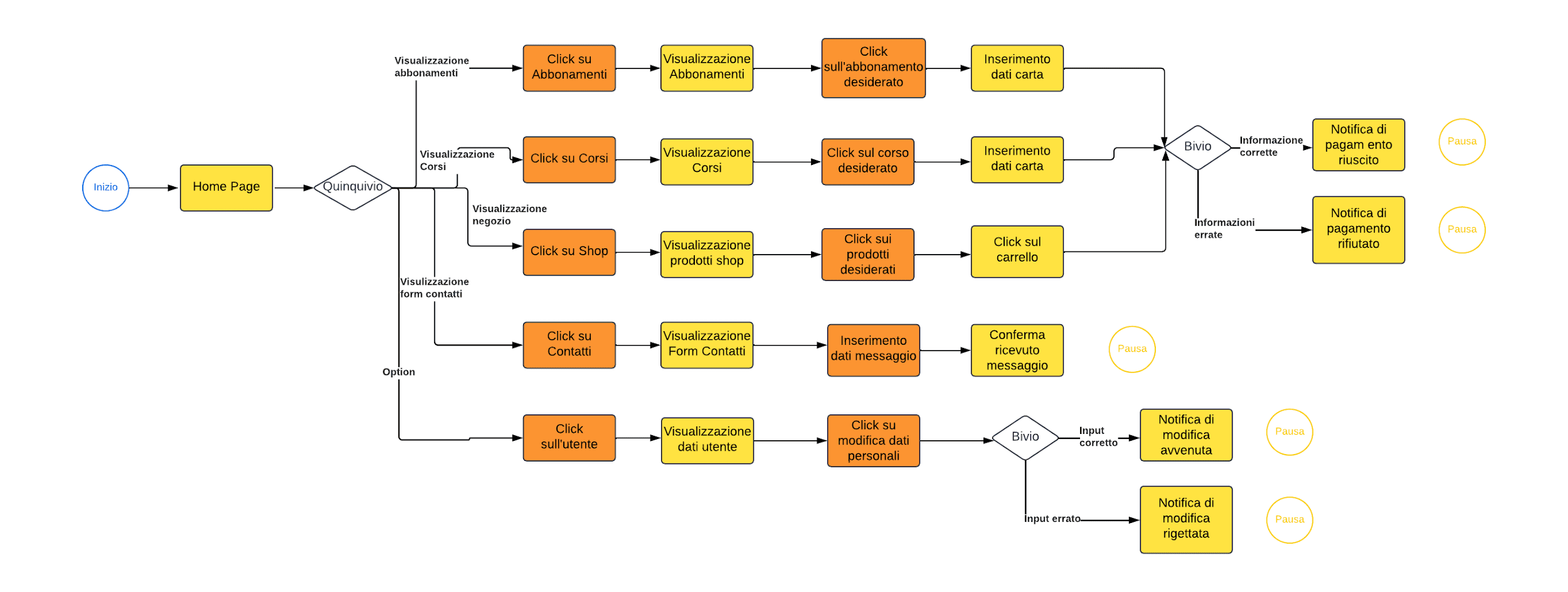
| US1 | Come utente voglio poter acquistare dei prodotti sportivi a cui sono interessato dallo shop della palestra. |
| --- | --- |
| US2 | Come utente voglio poter iscrivermi alla palestra scegliendo uno degli abbonamenti proposti dalla palestra. |
| US3 | Come utente voglio poter iscrivermi ai corsi speciali proposti dalla palestra |
| US4 | Come utente voglio avere la possibilità di contattare i membri dello staff della palestra. |
| US5 | Come utente voglio poter visualizzare e/o modificare i miei dati personali per tenerli aggiornati o correggere degli errori. |

## Features:

| F1 | US1 | Visualizzare tutti i prodotti nello shop |
| --- | --- | --- |
| F2 | US1 | Aggiungere i prodotti desiderati nel carrello |
| F3 | US2 | Visualizzare gli abbonamenti messi a disposizione dalla palestra |
| F4 | US2 | Acquistare un abbonamento solo. |
| F5 | US3 | Visualizzare i corsi proposti dalla palestra con tutte le relative informazioni riguardanti il corso, il prezzo e l’orario. |
| F6 | US3 | Iscriversi ai corsi desiderati. |
| F7 | US4 | Inserire i dati relativi al messaggio che si vuole inviare. |
| F8 | US5 | Visualizzare i dati relativi all’utente |
| F9 | US5 | Modificare i dati relativi all’utente. |

# 2. User Flow

In questa sezione del documento di sviluppo riportiamo gli “user flows” per il ruolo del cliente della nostra applicazione. Il cliente in ogni momento può visualizzare lo shop della palestra con tutti i prodotti, gli abbonamenti messi a disposizione e i corsi proposti dallo staff. Allo stesso momento puo’ decidere di modificare le sue informazioni personali. Inoltre il cliente avrà la possibilità di contattare lo staff della palestra per qualsiasi motivo.

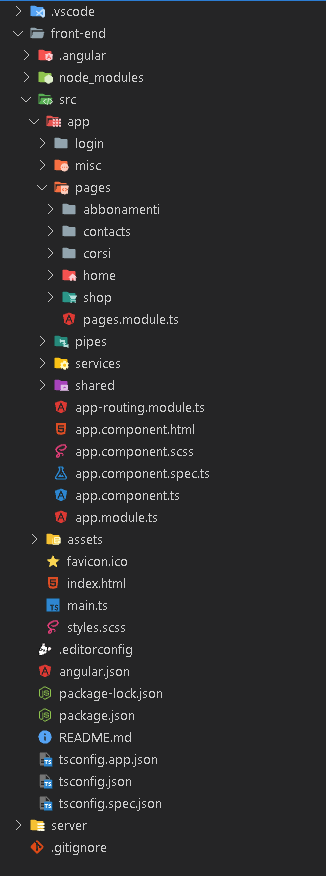


# 3. Application Implementation and Documentation

Sono state individuate le features da implementare nella nostra applicazione web per far si che tutti gli utenti del sito possano utilizzare quest'ultimo nel migliore dei modi in base alle necessità. Per sviluppare l’applicazione sono stati utilizzati il framework Angular, il framework Express.js e il DBMS MongoDB. Quest'ultimo è utilizzato per la gestione di tutti i dati che passano nel sito.

## 3.1 Project Structure

Nella figura sottostante è possibile vedere la struttura e la suddivisione del progetto. E’ presente una cartella front-end che contiene tutte le pagine e i componenti presenti all’interno dell’applicazione web.



## 3.2 Project Dependencies

Sono state usate dipendenze diverse tra frontend e backend, entrambe si dividono in dipendenze di sviluppo, e dipendenze necessarie.

### 

### 3.2.1 Backend dependencies



**express**: Express è il framework web usato per gestire le richieste http necessarie allo sviluppo dell’api backend.

**cors**: La dependency cors è stata usata per soddisfare l’header “Cross-Origin Resource Sharing”. È stato inserito in express tramite l’istruzione app.use(cors).

**express-fileupload**: Per permettere al servizio di accettare e gestire le immagini nell’aggiunta di prodotti, corsi e abbonamenti è stato necessario aggiungere la dependency che permette l’uso dei file ricevuti nella richiesta.

**mongoose**: Mongoose è la dependency su cai abbiamo fatto riferimento per la connessione e l’uso del database MongoDB.

**@types/\***: Le dependencies precedute da “@Types/” sono le definizioni dei tipi necessarie a typescript per utilizzare librerie sviluppate solo per JavaScript e successivamente adattate al linguaggio TypeScript.

**typescript**: Il linguaggio usato nella programmazione del progetto è stato TypeScript. A differenza del linguaggio JavaScript questo linguaggio introduce l’uso dei tipi oltre che ad una sintassi più rigida che semplifica la stesura di un codice privo di errori. Una volta terminato il linguaggio viene compilato in JavaScript dal compilatore tsc eseguibile tramite il comando npx tsc, permettendo quindi all’applicazione di essere eseguita tramite NodeJS.

**jest**: Jest è utilizzata per eseguire i test definiti nella cartella tests.

**ts-jest**: Ts-Jest è la dependency usata per permettere a Jest di eseguire i test scritti in linguaggio TypeScript. Questa permette di creare un file di configurazione per Jest già pronto per l’esecuzione.

**supertest**: Per eseguire il testing dell’api tramite express usiamo la dependency “Supertest” che ci permette di simulare il funzionamento dell’applicazione express inserita come suo parametro, così da verificare il funzionamento dei percorsi dell’api.

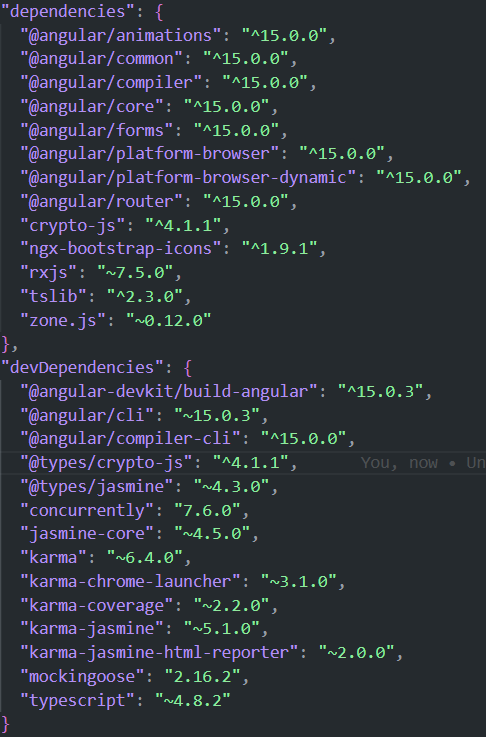
**mockingoose**: Per eseguire i test senza andare a modificare i dati contenuti all’interno del database ci siamo affidati alla dependency “Mockingoose” che permette di sostituirsi all’uso di mongoose replicandone gli effetti.

È possibile replicare le risposte di mongoose semplicemente esplicitando i dati che si vuole vengano ritornati dalle query, così da simulare ogni possibile risposta del database.

**Ts-Node**: La dependency Ts-Node permette di eseguire codice TypeScript senza doverlo prima compilare, permettendo quindi un’avvio più veloce dell’applicazione in fase di sviluppo.

**Node-Dev**: Questa dependency esegue un compito molto simile a quello della precedente permettendo l’esecuzione dell’applicazione durante lo sviluppo, permettendo il riavvio automatico dell’applicazione qualora questa venisse aggiornata (ad esempio modificando un file).

### 3.2.2 Frontend dependencies



**@angular/\***: Le dependencies precedute da “@angular/” sono le dependencies usate dal framework angular e che quindi permettono il funzionamento del framework.

**crypto-js**: Per eseguire l’hashing delle password abbiamo fatto uso della dependency Crypto JS che offre dei metodi pronti per usare diversi algoritmi.

**ngx-bootstrap-icons**: Come dice il nome questa dependency offre l’uso di icone ad Angular senza il bisogno di importare interamente bootstrap.

**rxjs**: La dependency base usata da Angular per offrire observables di eventi. Viene usata per gestire i messaggi tra i vari componenti e nelle richieste http ai servizi.

**tslib**: La dependency è una libreria di funzioni usate in typescript per eseguire diversi compiti base. È necessaria ad angular per assicurare il corretto funzionamento.

**zone.js**: Zone è una libreria di funzioni asincrone anch’esse usate da angular per il corretto funzionamento.

**concurrently**: Dependency che permette l’esecuzione di più script allo stesso momento.

**karma\*, Jasmine\***: Test runner di angular. Permette di eseguire il testing dei vari moduli.

## 3.3 Project Database

Per la gestione dei dati utili all’applicazione è stato usato MongoDB che ci ha permesso di gestire i dati in modo più flessibile rispetto ad un database di tipo relazionale permettendoci di aggiornare gli schemi di dati con l’avanzare delle fasi del progetto.

Gli schemi usati sono stati 5:

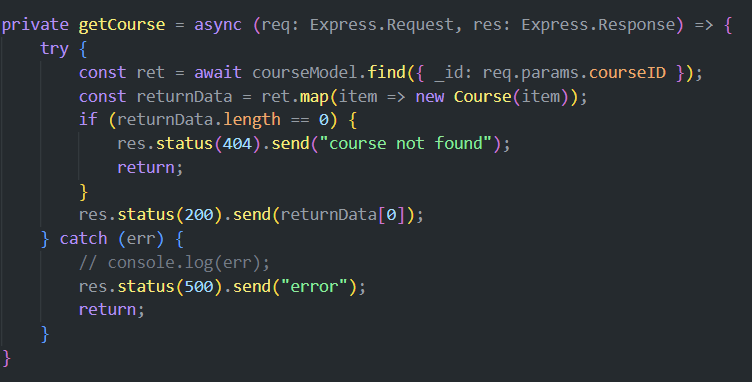
* courses: L’elenco dei corsi, contenente i dettagli del necessari al funzionamento della piattaforma. Gli attributi sono:
  + description;
  + name;
  + price.
* memberships: L’elenco degli abbonamenti disponibili nella piattaforma, contenente i dettagli necessari al funzionamento. Gli attributi sono:
  + description;
  + name;
  + price.
* products: L’elenco dei prodotti disponibili nella piattaforma. Gli attributi sono:
  + description;
  + name;
  + price;
  + styles.
* users: L’elenco degli utenti registrati alla piattaforma. Gli attributi sono:
  + address;
  + birthDate;
  + cf;
  + email;
  + isAdmin;
  + name;
  + password;
  + phoneNumber;
  + subscriptionDate;
  + surname.
* logintokens: L’elenco dei token di accesso relativi a tutti gli utenti loggati nella piattaforma. Gli attributi sono:
  + expires;
  + userID.

## 3.4 Project APIs

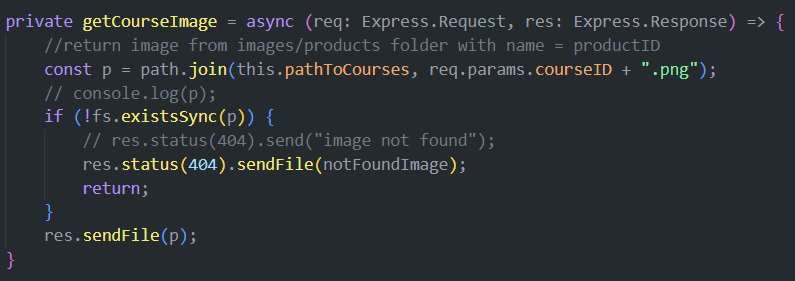
Vengono elencate le varie API implementate per soddisfare i vari requisiti dell’applicazione dal punto di vista dell’utente.



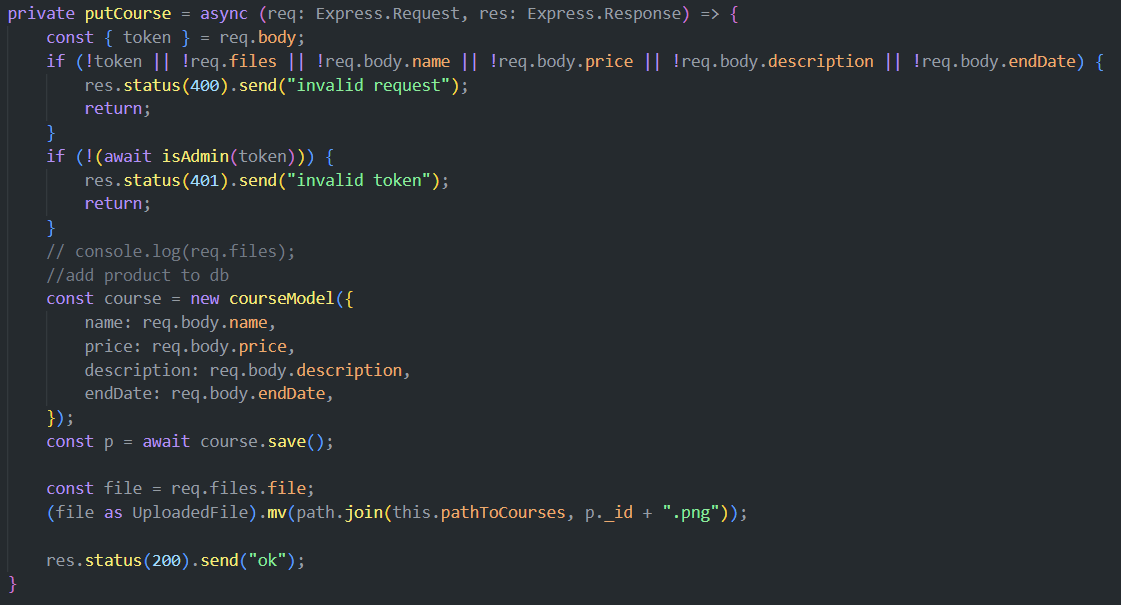
Questa funzione viene eseguita in risposta a una richiesta HTTP di tipo GET all'indirizzo '/api/courses'. La funzione cerca tutti gli elementi nel modello dei corsi, mappa gli elementi trovati a oggetti di tipo "Course" e invia gli oggetti come risposta alla richiesta. Se si verifica un errore durante la ricerca o durante l'elaborazione dei dati, viene inviata una risposta con stato 500 e il testo "error".



Questo codice rappresenta la funzione che gestisce la richiesta GET al percorso /api/course/:courseID. Il percorso specificato contiene una variabile chiamata "courseID", che rappresenta l'ID del corso che si desidera recuperare. Quando viene eseguita questa funzione, il codice tenta di recuperare un singolo documento dal modello di corsi che corrisponde all'ID del corso specificato nella richiesta. Poi, mappa l'elemento su un'istanza della classe Corso e invia il risultato al client che ha effettuato la richiesta. Se non viene trovato alcun corso con l'ID specificato, viene inviato uno stato di errore 404 (non trovato) al client. Se si verifica un altro tipo di errore durante il processo, viene inviato uno stato di errore 500 (errore interno del server) al client.



Questo codice rappresenta la funzione che gestisce la richiesta GET al percorso /api/course/:courseID/image. Il percorso specificato contiene una variabile chiamata "courseID", che rappresenta l'ID del corso per il quale si desidera recuperare l'immagine. Quando viene eseguita questa funzione, il codice tenta di recuperare l'immagine del corso specificato dalla cartella "images/products" con il nome pari all'ID del corso. Se l'immagine non viene trovata, viene inviato uno stato di errore 404 (non trovato) al client insieme a un'immagine di default. Se l'immagine viene trovata, viene inviata al client.



Questo codice rappresenta la funzione che gestisce la richiesta PUT al percorso /api/course. In questo caso, la richiesta viene utilizzata per aggiungere un nuovo corso al database.

Quando viene eseguita questa funzione, il codice esegue le seguenti operazioni:

1. Verifica che siano presenti tutti i dati necessari per creare un nuovo corso (token di autenticazione, file dell'immagine del corso, nome del corso, prezzo del corso, descrizione del corso e data di fine del corso). Se qualcuno di questi dati manca, viene inviato uno stato di errore 400 (richiesta non valida) al client.
2. Verifica che il token di autenticazione sia valido e appartenga a un amministratore. Se il token non è valido o non appartiene a un amministratore, viene inviato uno stato di errore 401 (non autorizzato) al client.
3. Crea un nuovo documento di corso nel modello di corsi utilizzando i dati forniti nella richiesta.
4. Salva il nuovo documento nel database e recupera l'ID del corso assegnato dal database.
5. Salva il file dell'immagine del corso nella cartella "images/products" con il nome pari all'ID del corso.
6. Invia uno stato di successo 200 (OK) al client.

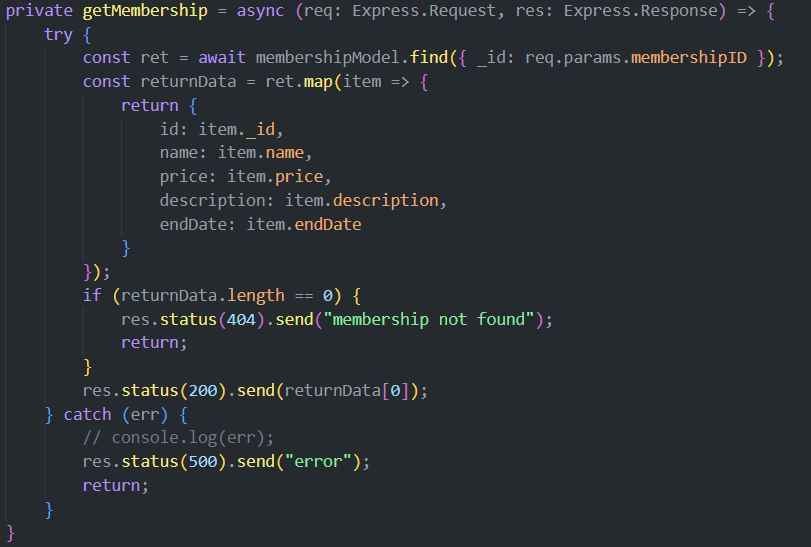


Questo codice rappresenta la funzione che gestisce la richiesta DELETE al percorso /api/course. In questo caso, la richiesta viene utilizzata per eliminare un corso esistente dal database.

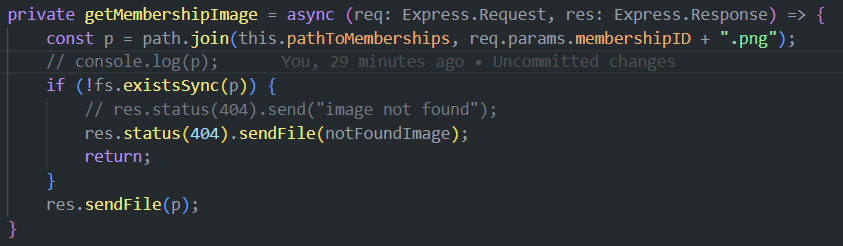
Quando viene eseguita questa funzione, il codice esegue le seguenti operazioni:

1. Verifica che siano presenti tutti i dati necessari per eliminare il corso (token di autenticazione e ID del corso). Se qualcuno di questi dati manca, viene inviato uno stato di errore 400 (richiesta non valida) al client.
2. Verifica che il token di autenticazione sia valido e appartenga a un amministratore. Se il token non è valido o non appartiene a un amministratore, viene inviato uno stato di errore 401 (non autorizzato) al client.
3. Elimina il documento del corso dal modello di corsi utilizzando l'ID del corso fornito nella richiesta.
4. Elimina l'immagine del corso dalla cartella "images/products" utilizzando l'ID del corso.
5. Invia uno stato di successo 200 (OK) al client.

Questo codice rappresenta la funzione che gestisce la richiesta GET al percorso /api/memberships. Quando viene eseguita, il codice tenta di recuperare tutti i documenti dal modello di membership. Poi, mappa ogni elemento su un oggetto con i campi "id", "name", "price", "description" e "endDate" e invia il risultato al client che ha effettuato la richiesta. Se si verifica un errore durante il processo, viene inviato uno stato di errore 500 (errore interno del server) al client.



Questo codice rappresenta la funzione che gestisce la richiesta GET al percorso /api/membership/:membershipID. Il percorso specificato contiene una variabile chiamata "membershipID", che rappresenta l'ID dell'iscrizione o abbonamento per il quale si desidera recuperare i dati. Quando viene eseguita, il codice tenta di recuperare il documento dell'iscrizione o abbonamento dal modello di membership utilizzando l'ID dell'iscrizione o abbonamento fornito nella richiesta. Poi, mappa il documento su un oggetto con i campi "id", "name", "price", "description" e "endDate" e invia il risultato al client che ha effettuato la richiesta. Se il documento non viene trovato, viene inviato uno stato di errore 404 (non trovato) al client. Se si verifica un errore durante il processo, viene inviato uno stato di errore 500 (errore interno del server) al client.



Questo codice rappresenta la funzione che gestisce la richiesta GET al percorso /api/membership/:membershipID/image. Il percorso specificato contiene una variabile chiamata "membershipID", che rappresenta l'ID dell'iscrizione o abbonamento per il quale si desidera recuperare l'immagine. Quando viene eseguita, il codice tenta di recuperare l'immagine dell'iscrizione o abbonamento dalla cartella "images/memberships" utilizzando l'ID dell'iscrizione o abbonamento fornito nella richiesta. Se l'immagine non viene trovata, viene inviato uno stato di errore 404 (non trovato) al client insieme all'immagine di errore "not found". Se l'immagine viene trovata, viene inviata al client.



Questo codice rappresenta la funzione che gestisce la richiesta PUT al percorso /api/memberships. In questo caso, la funzione viene utilizzata per aggiungere una nuova iscrizione o abbonamento al sistema.

Quando viene eseguita, il codice controlla se sono stati forniti tutti i dati necessari nella richiesta, ovvero il token di autenticazione, il file (immagine) dell'iscrizione o abbonamento, il nome dell'iscrizione o abbonamento, il prezzo, la descrizione e la data di fine validità. Se questi dati non sono presenti, viene inviato uno stato di errore 400 (richiesta non valida) al client. Se tutti i dati sono presenti, il codice verifica se il token di autenticazione è valido. Se il token non è valido, viene inviato uno stato di errore 401 (non autorizzato) al client. Se il token è valido, il codice crea un nuovo documento di iscrizione o abbonamento utilizzando i dati forniti nella richiesta e lo salva nel modello di membership. Quindi, il codice salva il file (immagine) dell'iscrizione o abbonamento nella cartella "images/memberships" utilizzando l'ID dell'iscrizione o abbonamento appena creato. Infine, viene inviato uno stato di successo 200 (ok) al client.



Questo codice rappresenta la funzione che gestisce la richiesta DELETE al percorso /api/memberships. In questo caso, la funzione viene utilizzata per eliminare un'iscrizione o abbonamento esistente dal sistema.

Quando viene eseguita, il codice controlla se sono stati forniti tutti i dati necessari nella richiesta, ovvero il token di autenticazione e l'ID dell'iscrizione o abbonamento da eliminare. Se questi dati non sono presenti, viene inviato uno stato di errore 400 (richiesta non valida) al client. Se tutti i dati sono presenti, il codice verifica se il token di autenticazione è valido. Se il token non è valido, viene inviato uno stato di errore 401 (non autorizzato) al client. Se il token è valido, il codice elimina il documento dell'iscrizione o abbonamento dal modello di membership utilizzando l'ID dell'iscrizione o abbonamento fornito nella richiesta. Quindi, il codice elimina l'immagine dell'iscrizione o abbonamento dalla cartella "images/memberships" utilizzando l'ID dell'iscrizione o abbonamento fornito nella richiesta. Infine, il codice invia uno stato di successo 200 al client come risposta alla richiesta.



Questo codice gestisce la richiesta GET al percorso /api/products. In questo caso, la funzione viene utilizzata per ottenere un elenco di prodotti disponibili.

Quando viene eseguita, il codice utilizza il modello di prodotto per cercare tutti i documenti di prodotto nel database. Quindi, il codice mappa i risultati della ricerca in un formato più semplice da gestire, estraendo solo alcuni dei campi di ogni documento di prodotto, come l'ID, il nome, il prezzo e la descrizione. Infine, il codice invia i dati del prodotto al client come risposta alla richiesta. Se si verifica un errore durante l'esecuzione del codice, viene inviato uno stato di errore 500 (errore interno del server) al client.



Questo codice la richiesta GET al percorso /api/product/:productID. In questo caso, la funzione viene utilizzata per ottenere le informazioni di un singolo prodotto specifico, identificato dall'ID del prodotto presente nell'URL della richiesta.

Quando viene eseguita, il codice utilizza il modello di prodotto per cercare il documento di prodotto con l'ID del prodotto fornito nella richiesta. Quindi, il codice mappa i risultati della ricerca in un formato più semplice da gestire, estraendo solo alcuni dei campi del documento di prodotto, come l'ID, il nome e il prezzo. Se il documento di prodotto non viene trovato, il codice invia uno stato di errore 404 (risorsa non trovata) al client. Se il documento viene trovato, il codice invia le informazioni del prodotto al client come risposta alla richiesta. Se si verifica un errore durante l'esecuzione del codice, viene inviato uno stato di errore 500 (errore interno del server) al client.

Questo codice rappresenta la funzione che gestisce la richiesta GET al percorso /api/products/:productID/image. In questo caso, la funzione viene utilizzata per ottenere l'immagine associata a un singolo prodotto specifico, identificato dall'ID del prodotto presente nell'URL della richiesta.

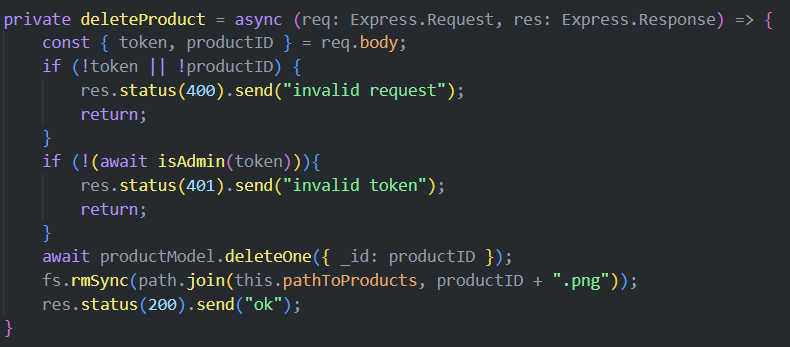
Quando viene eseguita, il codice costruisce il percorso completo dell'immagine del prodotto specifico richiesto utilizzando il percorso della cartella dei prodotti e l'ID del prodotto fornito nella richiesta. Quindi, il codice controlla se l'immagine esiste effettivamente nella cartella dei prodotti utilizzando il modulo "fs" di Node.js. Se l'immagine non esiste, il codice invia un'immagine di errore predefinita al client insieme a uno stato di errore 404 (risorsa non trovata). Se l'immagine esiste, il codice invia l'immagine al client come risposta alla richiesta. Se si verifica un errore durante l'esecuzione del codice, viene inviato uno stato di errore 500 (errore interno del server) al client.



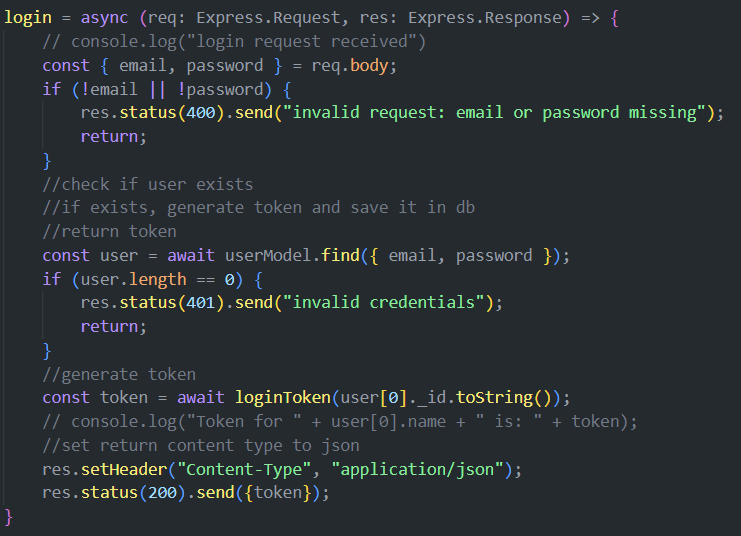
Questo codice gestisce una richiesta HTTP PUT al percorso /api/product. La richiesta viene gestita solo se è stata fornita una proprietà "token" nel corpo della richiesta, se è stato allegato almeno un file con il nome "file" nella richiesta, e se sono state fornite le proprietà "name", "price" e "description" nel corpo della richiesta. Se queste condizioni non sono soddisfatte, il codice invia una risposta con stato HTTP 400 ("invalid request").

Se la proprietà "token" è presente, il codice verifica se il token è valido per un amministratore chiamando la funzione isAdmin(). Se il token non è valido, il codice invia una risposta con stato HTTP 401 ("invalid token").

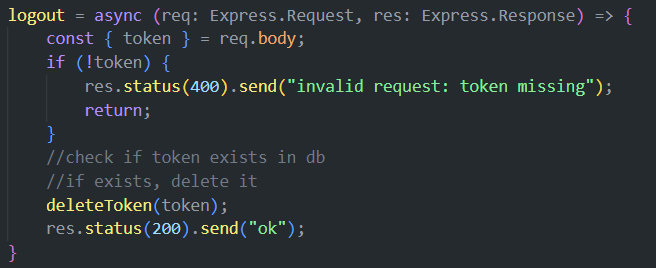
Se tutte le condizioni sono soddisfatte, il codice crea un nuovo oggetto del modello productModel con le proprietà "name", "price" e "description" fornite nella richiesta, e lo salva nel database. Quindi, il codice prende il file allegato nella richiesta e lo sposta nella cartella images/products con il nome corrispondente all'ID del prodotto appena creato. Infine, il codice invia una risposta con stato HTTP 200 e un oggetto JSON contenente l'ID del prodotto appena creato.



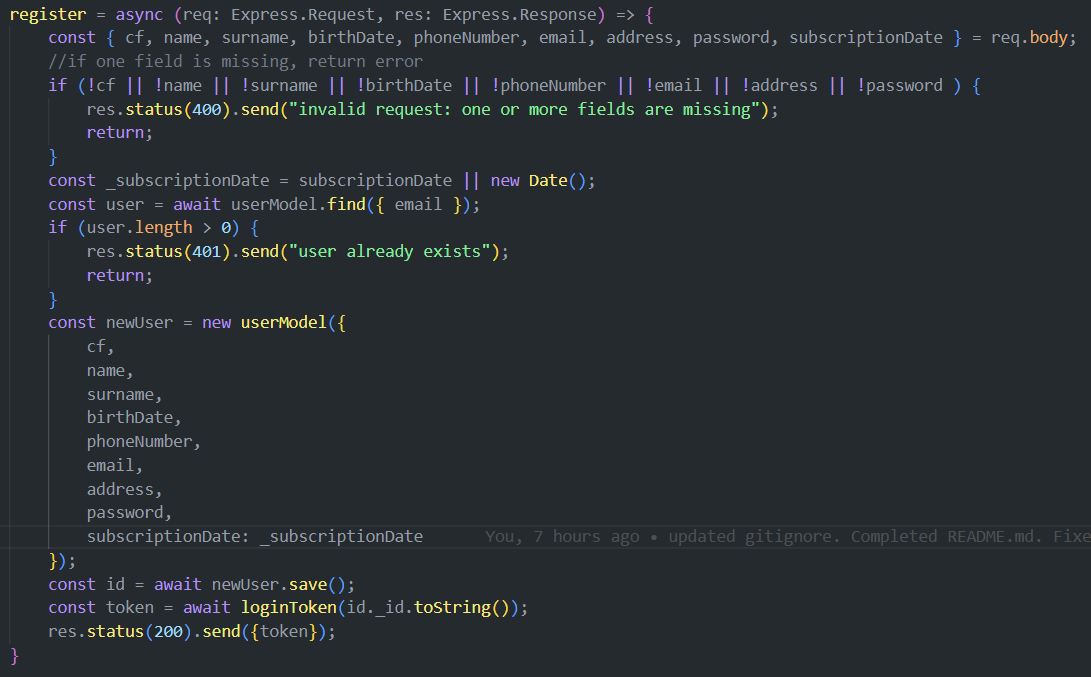
Questo codice fa una richiesta DELETE a /api/product con un corpo di richiesta che include un token di autenticazione e un identificatore di prodotto (productID). Se il token non è presente o l'identificatore del prodotto non è presente, viene inviata una risposta con uno stato 400 "richiesta non valida". Se il token non è valido o non appartiene a un amministratore, viene inviata una risposta con uno stato 401 "token non valido". Se tutti i controlli di validità sono superati, viene eliminato il prodotto dal database con l'identificatore specificato e viene eliminata l'immagine del prodotto dalla cartella dei prodotti. Infine, viene inviata una risposta con uno stato 200 "ok".



Questo codice implementa una funzione di login per l'utente che invia una richiesta POST al percorso /api/login. Quando viene ricevuta la richiesta, viene estratto dalla richiesta il campo "email" e "password" e viene verificato se sono presenti. Se mancano uno o entrambi i campi, viene inviata una risposta con status code 400 (richiesta non valida) e il messaggio "invalid request: email or password missing". Se entrambi i campi sono presenti, viene cercato nella base dati un utente con l'email e la password specificati. Se l'utente non viene trovato, viene inviata una risposta con status code 401 (autorizzazione negata) e il messaggio "invalid credentials". Se l'utente viene trovato, viene generato un token di login utilizzando l'ID dell'utente come input e viene inviata una risposta con status code 200 (successo) e il token come corpo della risposta.



Il codice implementa una funzionalità di logout per l'utente che ha effettuato il login. Quando viene inviato un richiesta POST al percorso "/api/logout", il codice preleva il token dal corpo della richiesta. Se il token non è presente, viene inviato un messaggio di errore con stato HTTP 400 ("invalid request: token missing"). Se invece il token è presente, il codice chiama la funzione "deleteToken" passando il token come parametro. Questa funzione elimina il token dal database. Infine, viene inviato un messaggio di risposta con stato HTTP 200 ("ok").



Questo codice gestisce la richiesta di registrazione di un nuovo utente al percorso PUT /api/login.

Viene controllato che tutti i campi richiesti siano presenti nella richiesta e che l'indirizzo email non sia già stato utilizzato da un altro utente. In caso di successo, viene creato un nuovo utente nel database e viene generato un token di accesso per l'utente appena registrato. Il token viene poi inviato come risposta alla richiesta. In caso di errore, viene inviata una risposta con il codice di stato appropriato e un messaggio di errore.

# 4 Documentazione API

È possibile trovare la documentazione relativa all’api leggendo il file README.md al seguente link:

<https://github.com/ProgettoIngegneria-T46/progettoT46>

# 5 Implementazione Front End

L’implementazione del front-end è stata realizzata tramite il framework Angular.   
Il FrontEnd fornisce le funzionalità dell’applicazione della Chad Gym.

## Pagine:

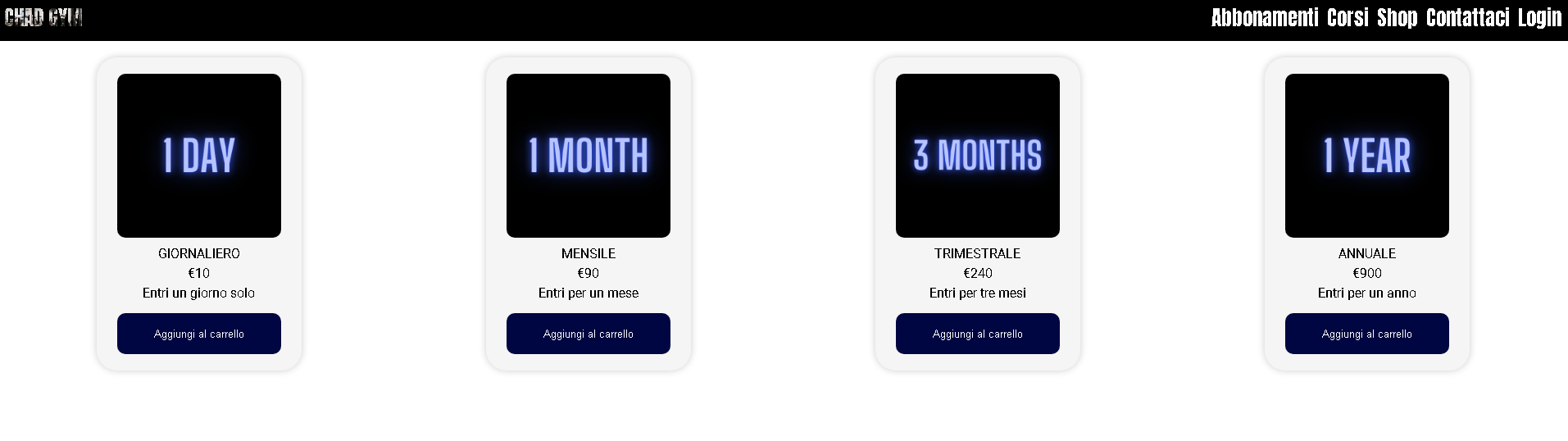
L’applicazione è composta da 5 pagine: Home, Abbonamenti, Corsi, Shop, Contattaci.

**Home:**



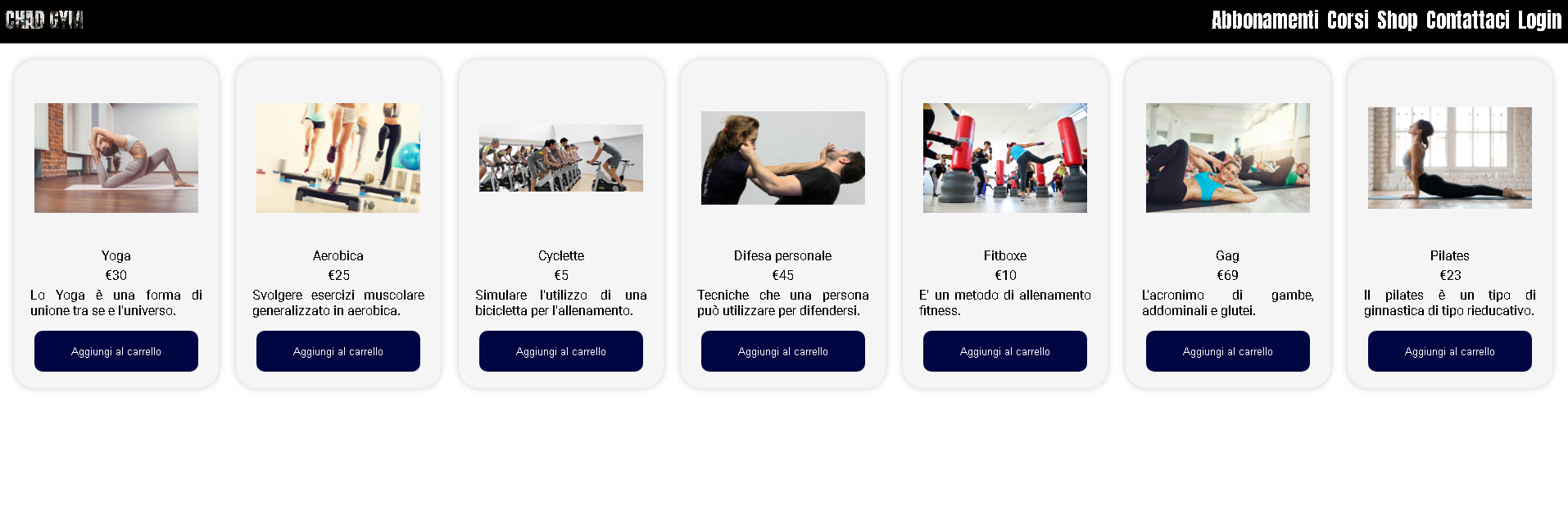
**Abbonamenti:**

In questa pagina sono visibili le varie opzioni di abbonamenti alla palestra con relativa descrizione. Tramite il pulsante aggiungi al carrello, sarà possibile (se autenticati e non in possesso di un abbonamento attivo) abbonarsi al piano desiderato.



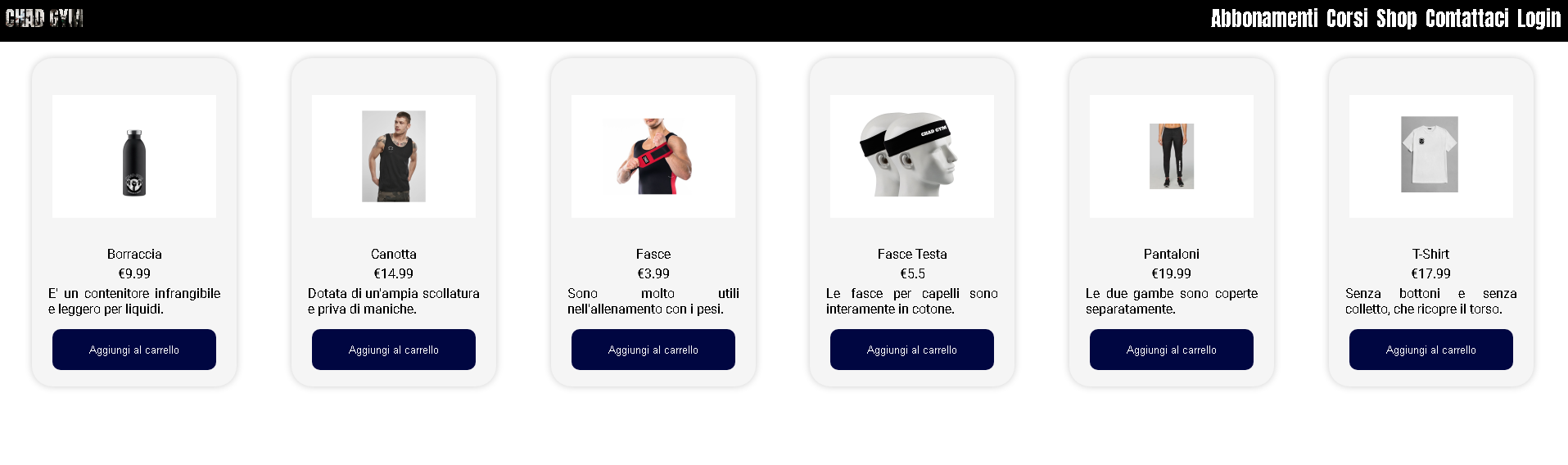
**Corsi:**

In questa pagina, vengono mostrati tutti i corsi disponibili e offerti dalla palestra. E’ possibile selezionare i corsi da seguire aggiungendoli al carrello ed effettuando l'acquisto. Ogni corso ha il prezzo e una descrizione dell’attività che si svolgerà.



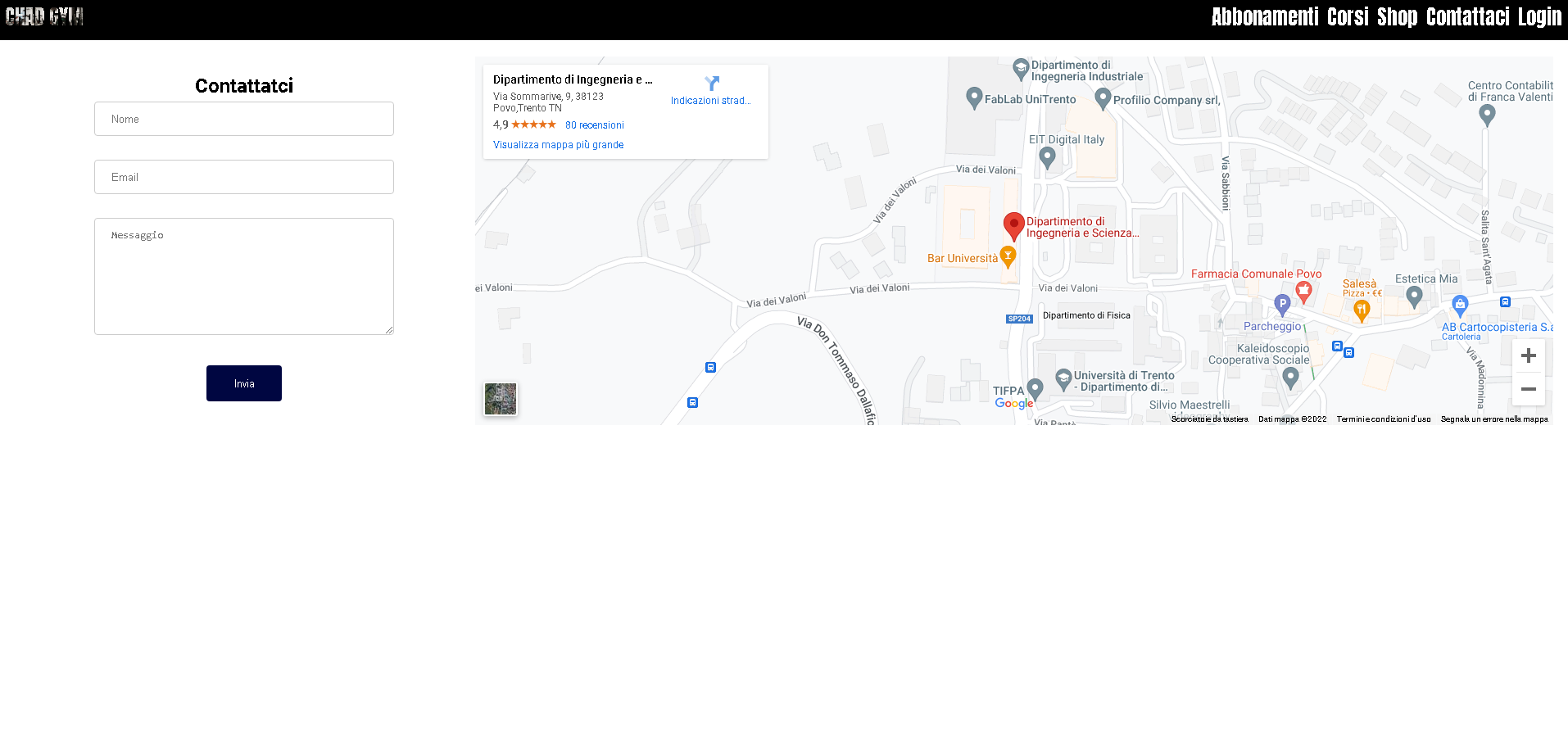
**Shop:**

In questa pagina vengono mostrati i prodotti messi in vendita dalla palestra. Sarà possibile selezionare i prodotti e aggiungerli al carrello tramite i bottoni specifici ed acquistare il merchandising marchiato Chad Gym.



**Contattaci:**

Nella Pagina in questione è presente un form per inviare un messaggio direttamente alla direzione della palestra. Basterà inserire il nome e l’email del mittente e il messaggio da comunicare, successivamente cliccare sul pulsante invia. E’ presente anche una mappa collegata a google maps dove viene mostrata la posizione della palestra.



## Componenti

Le varie pagine sono composte da diversi componenti ognuno di questi necessario a soddisfare determinate esigenze dell’applicazione.

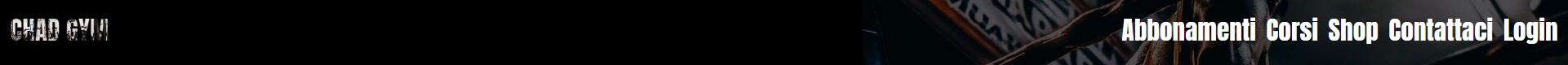
Header:

In tutte le pagine è presente il componente header. Questo componente presenta i link relativi a tutte le pagine del sito, e il bottone di apertura del form di accesso e registrazione.

I 4 link sono:

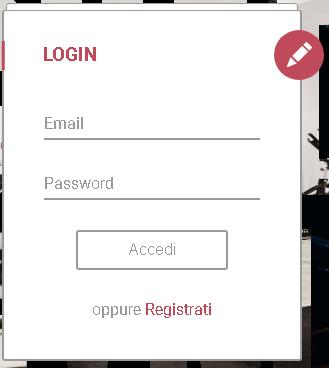
* Home: Per entrare nella pagina home basta cliccare sul pulsante “Chad gym”;
* Abbonamenti: Per entrare nella pagina abbonamenti basta cliccare sul pulsante “Abbonamenti”;
* Corsi: Per entrare nella pagina corsi basta cliccare sul pulsante “Corsi”;
* Shop: Per entrare nella pagina shop basta cliccare sul pulsante “Shop”;

Cliccando il bottone Login si aprirà il form di login tramite il quale sarà possibile effettuare il login o la registrazione alla piattaforma



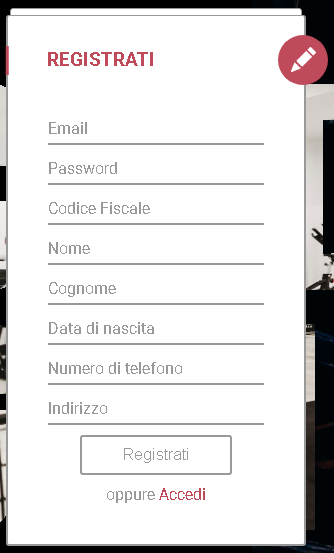
Login:

La schermata del login presenta il form di accesso in cui vengono richiesti nome utente e password con il relativo tasto di accesso. E’ presente anche un collegamento al form di registrazione.

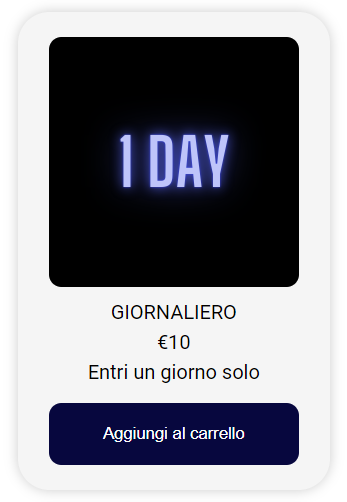


Registrati:

Il form in questione permette ad un nuovo utente di registrarsi al sito della palestra. Basterà inserire di dati richiesti e premere il bottone “registrati”. E’ presente il campo “password” nel quale dovrà per l’appunto essere inserita la parola d’ordine che verrà utilizzata per effettuare l’accesso. Essa verrà criptata per tutelare la privacy degli utenti.



Display:



Il componente display viene usato nelle pagine shop, abbonamenti e corsi per mostrare i dettagli relativi alle selezioni ricevute dal database

## Servizi

Login:

L’applicazione fa uso di un servizio per implementare le funzioni e la gestione del login.

Presenta i metodi di login, logout e registrazione e l’attributo token usato per verificare lo stato di login e per inviare le richieste che necessitano di aver effettuato l’accesso.

## Pipes

SafePipe:

Per richiedere le immagini in modo dinamico è necessario accertarsi che i link siano sicuri, di conseguenza è stato necesario creare una pipe che li prendesse in input e che li rensesse sicuri all’uso.

# 6 Repository GitHub

Il progetto Chad Gym è disponibile al seguente link:https://github.com/ProgettoIngegneria-T46. Per poter eseguire il progetto è necessario avviare separatamente il backend e il frontend.

Per avviare il backend sarà necessario spostarsi nella cartella server e eseguire il comando “node .”.

Per avviare il frontend è necessario entrare nella cartella frontend ed eseguire il comando npm start.

# 7 Testing

È possibile lanciare il testing posizionandosi con il terminale alla cartella “server” e lanciare il comando “npm test”.

Lanciando dunque il comando, se tutti i test andranno a buon fine, il risultato prodotto sarà il seguente:

