



# Urban Shirt

Vendita e personalizzazione t-shirt con spazio per la community

3/12/2022

## D4

Team 32

Minatel Davide

Hu Angela

Zanoni Devis



# Indice

<b>User Flows</b>	<b>3</b>
<b>Application Implementation and Documentation</b>	<b>4</b>
Project Structure	4
Project Dependencies	4
Project Database	5
Project APIs	7
Resources extraction from the class diagram	7
Resources models	8
Sviluppo API	11
Proposte dell'azienda	11
Proposte dell'azienda filtrate	11
Magliette della community	11
Magliette della community filtrate	12
Like alle magliette	12
Maglietta della settimana	13
Maglietta dell'azienda per l'homepage	13
Maglietta della community per l'homepage	14
Area utente	14
Registrazione utente	14
Login	15
<b>API documentation</b>	<b>16</b>
<b>FrontEnd Implementation</b>	<b>17</b>
Home Page	17
Autenticazione e registrazione	18
Area Utente	18
Magliette della community	19
Magliette dell'azienda	20
<b>GitHub Repository and Deployment Info</b>	<b>22</b>
<b>Testing</b>	<b>22</b>

## User Flows

Nei seguenti paragrafi viene descritto il funzionamento della parte di applicazione implementata. Ciò viene rappresentato sotto forma di user flow.

In seguito viene riportata la legenda che descrive i simboli utilizzati negli user flow e che ne permettono la lettura.



L'utente può accedere alla pagina di registrazione e autenticazione cliccando sull'icona apposita, situata in alto a destra nella Home Page.

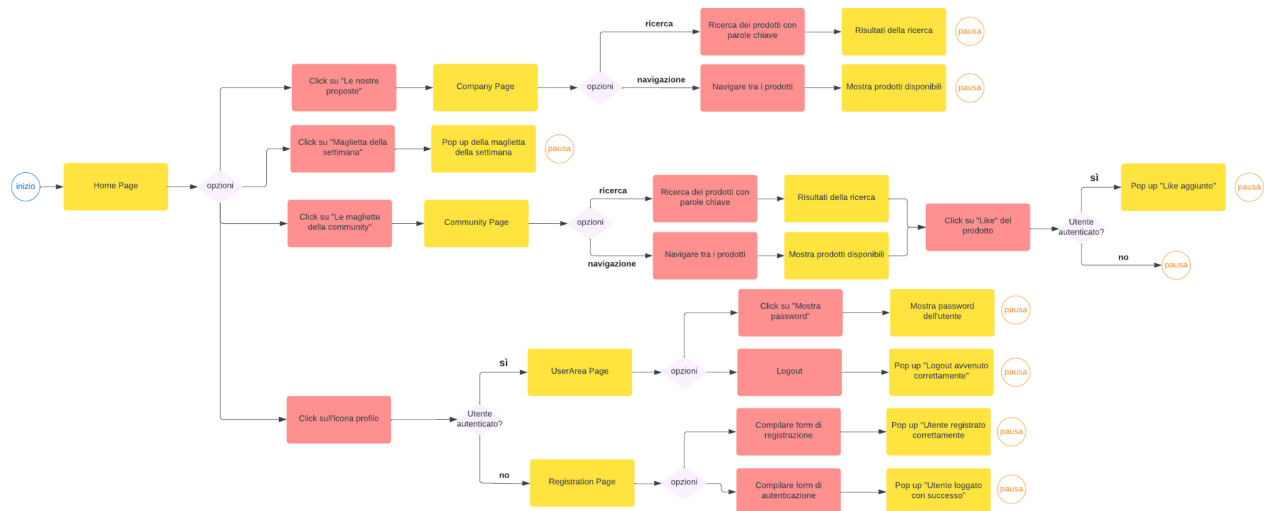
L'utente può scegliere se registrarsi nel sistema, se non l'ha mai fatto prima, oppure autenticarsi, se possiede già credenziali valide. In base a ciò, sceglie se compilare i campi del form di registrazione o di autenticazione.

Il completamento della registrazione non comporta l'autenticazione nel sistema, pertanto, una volta conclusa con successo la registrazione, è necessario compilare il form di autenticazione per poter effettivamente accedere come utente autenticato.

L'utente può visualizzare tutti i prodotti messi a disposizione dal sistema, scegliendo tra quelli proposti dall'azienda o tra quelli ideati dalla community, cliccando sui rispettivi bottoni, *"Le nostre proposte"* e *"Le magliette della community"*. Le magliette possono essere cercate tramite normale navigazione, scorrendo a mano tra le opzioni, oppure possono essere filtrate secondo parole chiave determinate dall'utente, grazie all'utilizzo della barra di ricerca.

Se autenticato, può anche mettere un *"Like"* alle magliette della community.

L'icona del profilo, se cliccata, rimanda ad un'altra pagina. La pagina di destinazione varia in base all'utente; se autenticato sarà *Area Utente*, in caso contrario, sarà la pagina di registrazione e autenticazione.



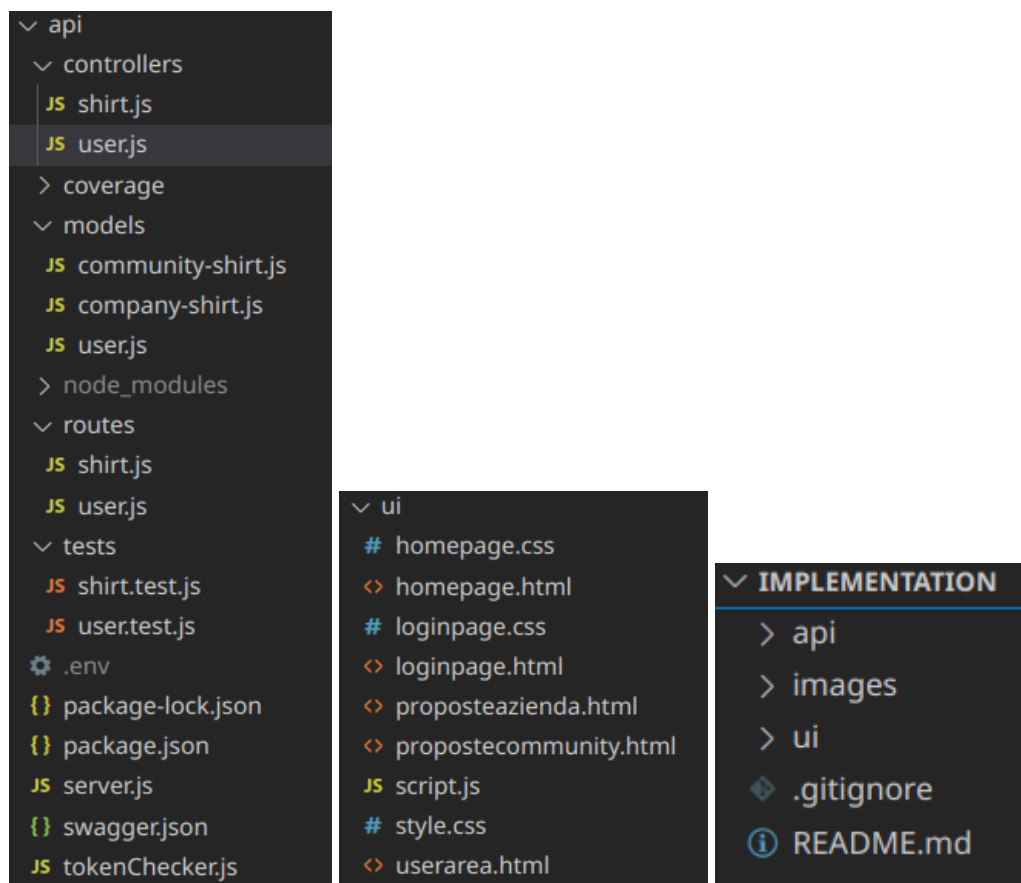
# Application Implementation and Documentation

## Project Structure

La struttura del progetto, visibile negli screenshot sottostanti, è composta da una cartella api contenente la gestione delle API locali (con controllers, models e routes), il file server.js per il funzionamento dell'applicazione, il package.json per i moduli node, il tokenChecker per le operazioni dell'utente, il file di swagger per la documentazione delle API e la cartella tests per il testing dell'applicazione.

Troviamo poi la cartella ui contenente le pagine e lo scripting del frontend.

Nell'ultimo screenshot è visibile la struttura per intero, con la cartella images contenente immagini e loghi utili al frontend del sito oltre che alle immagini delle magliette.



## Project Dependencies

Per lo sviluppo del progetto sono stati utilizzati i seguenti moduli:

- dotenv
- express
- html

- jsonwebtoken
- mongoose
- swagger-ui-express

Per la parte di testing in aggiunta sono stati utilizzati i seguenti moduli:

- jest
- supertest

## Project Database

Per il salvataggio e la gestione dei dati utili allo sviluppo dell'applicazione abbiamo definito tre modelli riportati di seguito:

**UrbanShirt**

LOGICAL DATA SIZE: 4.93KB   STORAGE SIZE: 108KB   INDEX SIZE: 240KB   TOTAL COLLECTIONS: 3

[CREATE COLLECTION](#)

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
<a href="#">communityshirts</a>	5	1.67KB	341B	36KB	2	72KB	36KB
<a href="#">companyshirts</a>	6	1.43KB	245B	36KB	2	96KB	48KB
<a href="#">users</a>	9	1.83KB	209B	36KB	2	72KB	36KB

I tre modelli sono stati rappresentati nelle seguenti modalità:

Magliette dell'azienda

```
_id: ObjectId('63b05850139ac3b7ccb19786')
name: "maglietta bianca c-style"
creationDate: 2022-12-27T08:52:42.207+00:00
image: "https://github.com/UrbanShirt/implementation/blob/main/images/shirts/w..."
color: "white"
material: "rayon"
__v: 0
```

Magliette della community

```
_id: ObjectId('63b05851139ac3b7ccb19805')
name: "maglietta bianca RODOLFO"
creationDate: 2022-12-31T15:42:09.071+00:00
image: "https://github.com/UrbanShirt/implementation/blob/main/images/shirts/w..."
creator: "Rodolfo"
color: "white"
material: "polyester"
likes: 1
voters: Array
  0: "rodolfo"
isPublic: true
isMostLiked: false
__v: 0
```



## Utenti

```
_id: ObjectId('63ab5af86491ffe2fca5b6bc')
username: "rodolfo"
email: "a@b.c"
firstName: "Rodolfo"
lastName: "Sterchi"
address: "Via del capitano, 2"
birthDate: 2022-12-27T21:39:49.647+00:00
password: "TestBello2!"
__v: 0
```

## Project APIs

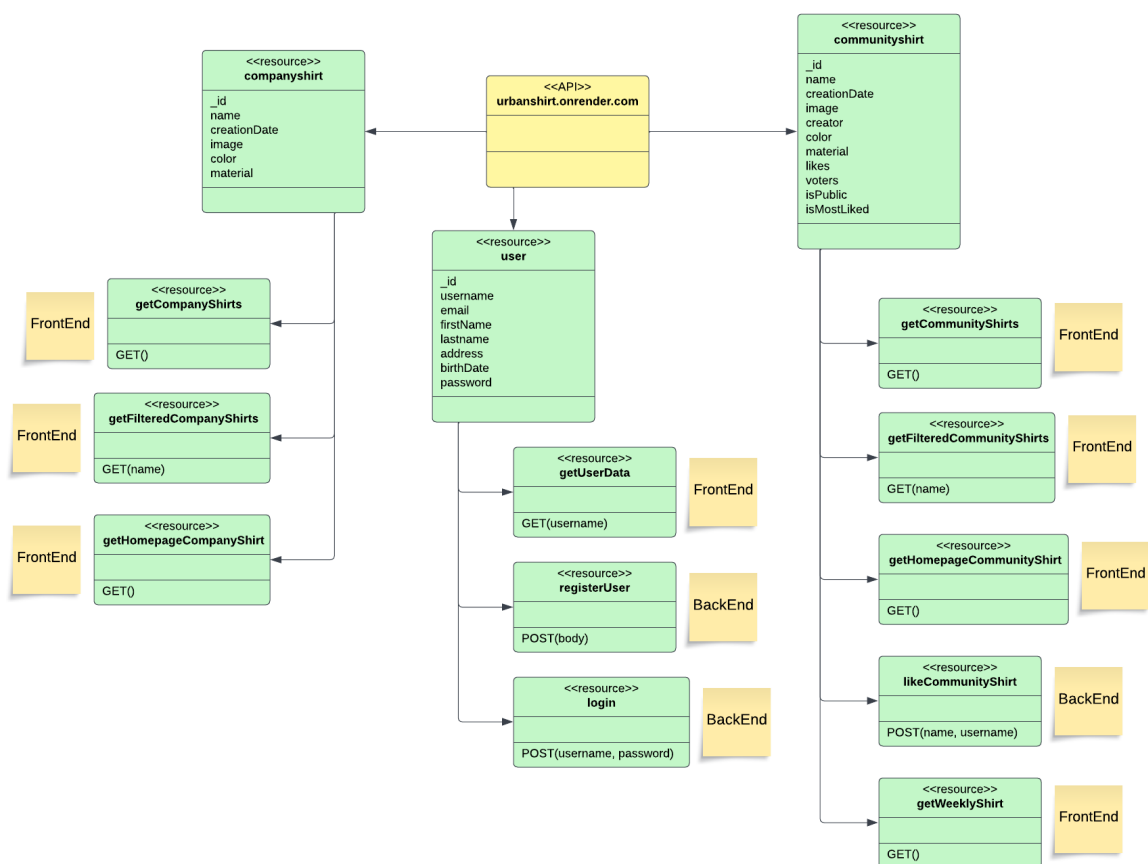
### Resources extraction from the class diagram

Nel seguente diagramma vengono rappresentate le risorse estratte dal class diagram del documento precedente.

In particolare, abbiamo ricavato tre risorse principali: *user*, *communityShirt* e *companyShirt*.

Alla radice del diagramma è presente l'url del sito, ovvero [urbanshirt.onrender.com](http://urbanshirt.onrender.com), dal quale partono le varie API.

(Il parametro *\_id* è un identificatore generato automaticamente da *MongoDB*)

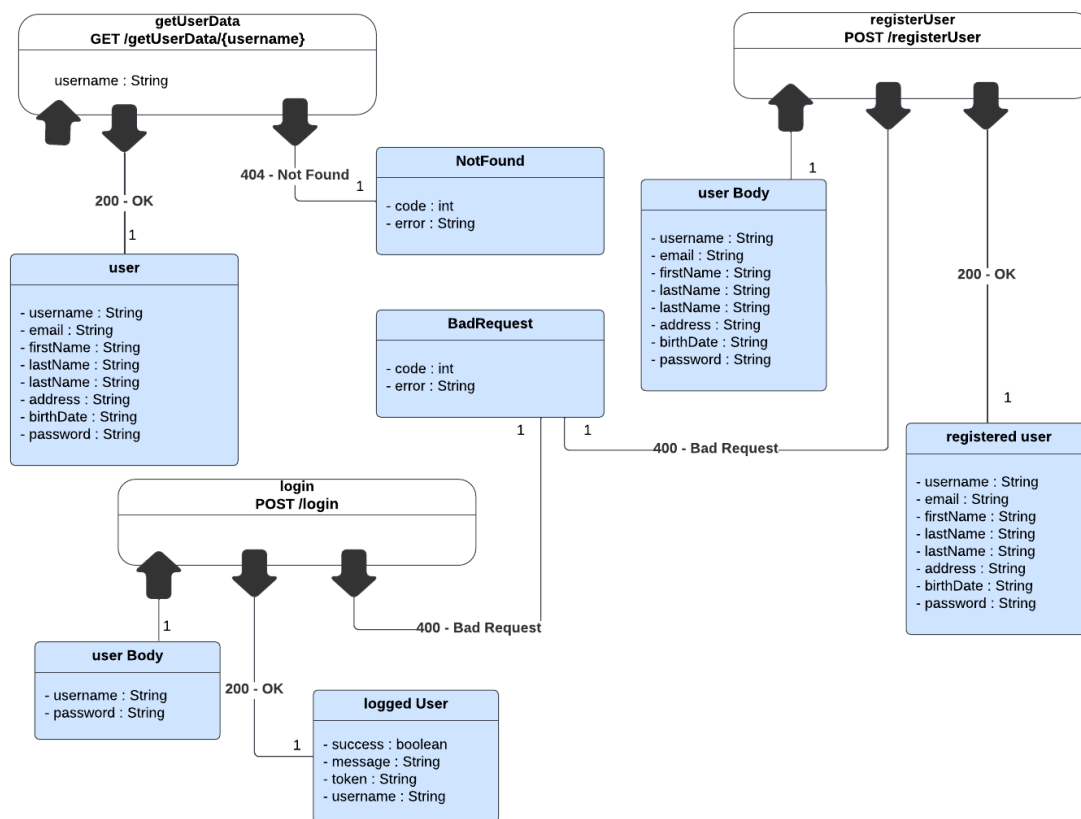


## Resources models

In questa sezione vengono rappresentati i diagrammi Resources Models, uno per ogni risorsa, ricavati dal diagramma precedente.

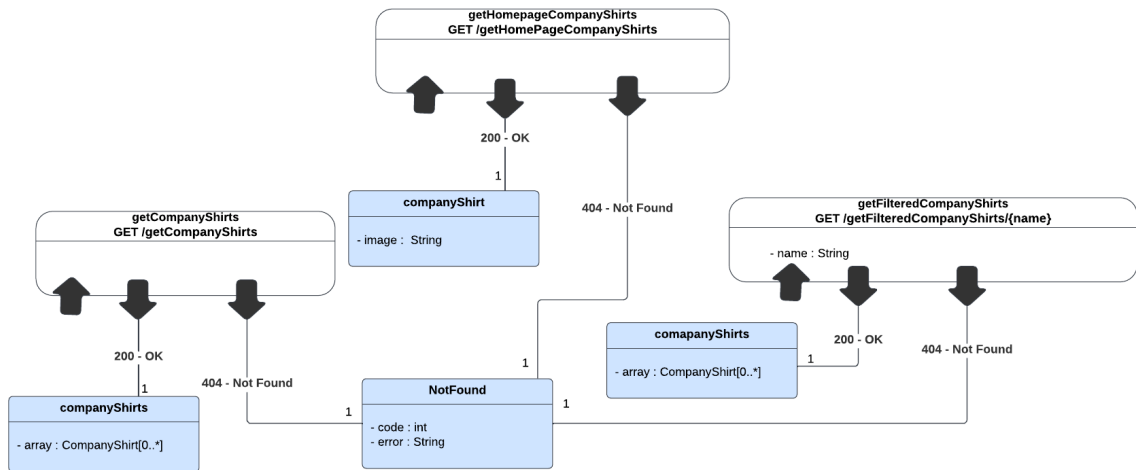
In ogni API resource è presente l'URI per accedere al servizio rappresentato.

### USER

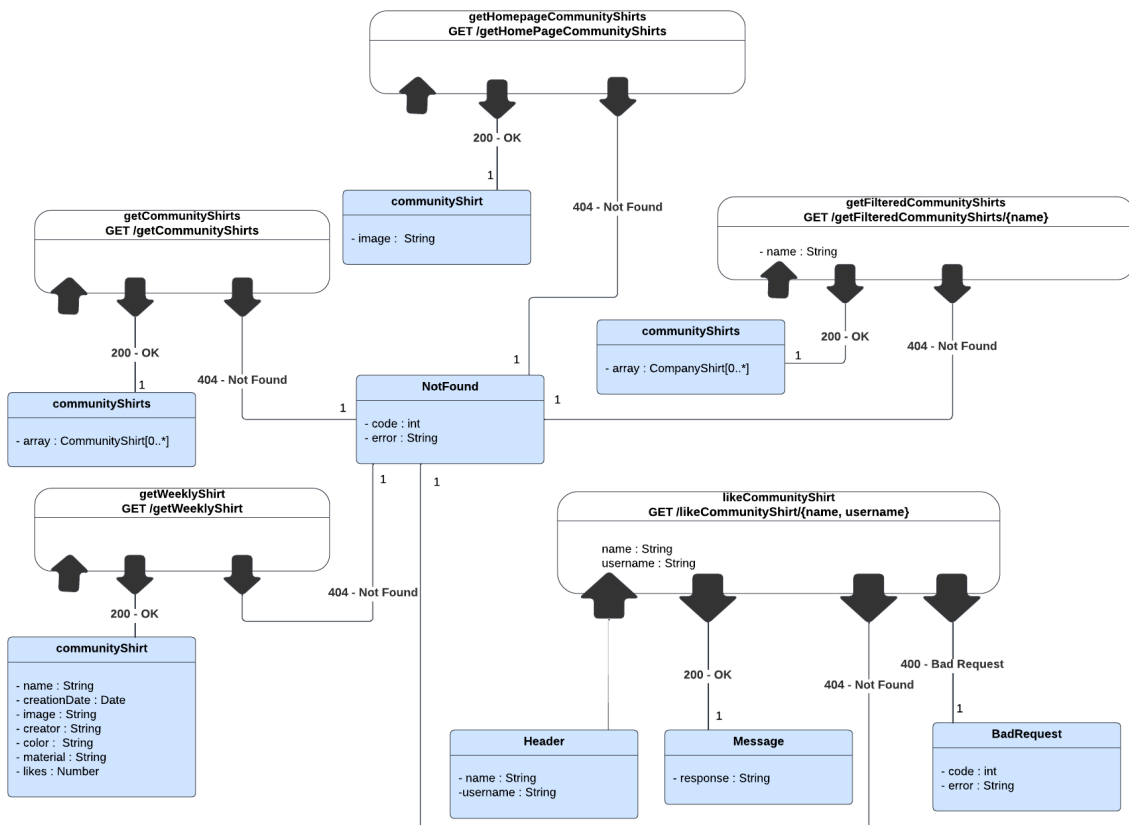





## COMPANY SHIRT



## COMMUNITY SHIRT





In seguito viene mostrata una tabella riassuntiva delle varie API per ogni risorsa:

<b>USER</b>	
/getUserData/{username}	GET
/login	POST
/registerUser	POST
<b>COMPANY SHIRT</b>	
/getComanyShirts	GET
/getFilteredCompanyShirts/{name}	GET
/getHomepageCompanyShirt	GET
<b>COMMUNITY SHIRT</b>	
/getCommunityShirts	GET
/getFilteredCommunityShirts/{name}	GET
/getHomepageCommunityShirt	GET
/getWeeklyShirt	GET
/likeCommunityShirt/{name, username}	GET

## Sviluppo API

In questa sezione sono descritte le varie API sviluppate a partire dal precedente resources model.

### Proposte dell'azienda

Recupera tutte le magliette dell'azienda per mostrarle a frontend.

```
// GET '/getCompanyShirts'
const getCompanyShirts = (req, res) => {
  CompanyShirt.find({}, (err, data) => {
    if (err || !data || data.length == 0) {
      return res.status(404).json({error: "Impossible to find shirts"});
    }
    return res.status(200).json(data);
  })
};
```

### Proposte dell'azienda filtrate

Recupera tutte le magliette dell'azienda per mostrarle a frontend con un filtro in like sul nome scelto dall'utente.

```
// GET '/getFilteredCompanyShirts/:name'
const getFilteredCompanyShirts = (req, res) => {
  let filterName = req.params.filterName;

  CompanyShirt.find({name: {$regex: '.*'+filterName+'.*', $options: 'i'}}, (err, data) => {
    if (err || !data || data.length == 0) {
      return res.status(404).json({error: "Impossible to find shirts"});
    } else {
      return res.status(200).json(data);
    }
  })
};
```

### Magliette della community

Recupera tutte le magliette pubbliche create dagli utenti. Queste magliette si identificano come pubbliche tramite l'apposito flag isPublic, in quanto è previsto che un utente possa mantenere private le sue creazioni.

```
// GET '/getCommunityShirts'
const getCommunityShirts = (req, res) => {
  CommunityShirt.find({isPublic: true}, (err, data) => {
    if (err || !data || data.length == 0) {
      return res.status(404).json({error: "Impossible to find shirts"});
    }
    return res.status(200).json(data);
  })
};
```

## Magliette della community filtrate

Recupera le magliette pubbliche create dagli utenti con un filtro in like scelto dall'utente sul nome della maglietta e sul nome del creatore della maglietta.

```
// GET '/getFilteredCommunityShirts/:name'
const getFilteredCommunityShirts = (req, res) => {
  let filterName = req.params.filterName;

  CommunityShirt.find({$or:[
    {name: {$regex: '.*'+filterName+'.*', $options: 'i'}},
    {creator: {$regex: '.*'+filterName+'.*', $options: 'i'}}
  ]}, (err, data) => {
    if (err || !data || data.length == 0) {
      return res.status(404).json({error: "Impossible to find shirts"});
    } else {
      return res.status(200).json(data);
    }
  })
};
```

## Like alle magliette

Gli utenti possono mettere like alle magliette della community che preferiscono. Questa API, ricevendo in input il nome della maglietta scelta e l'username di chi vota procede con l'inserimento del like: oltre ad incrementare l'attributo likes aggiunge lo username nella lista dei "votanti" della maglietta. In questo modo un utente non può votare una stessa maglietta più volte perchè viene trovato in questa lista.

```
// POST '/likeCommunityShirt'
const likeCommunityShirt = (req, res) => {
  var likedShirt = req.body.name;
  var user = req.body.username;

  CommunityShirt.findOne({name: likedShirt}, (err, data) => {
    if (err || !data || data.length == 0) {
      return res.status(404).json({error: "Impossible to find shirt to like"});
    } else if (data.voters.includes(user)) {
      return res.status(400).json({error: "You already liked this shirt"});
    } else {
      CommunityShirt.updateOne({name: likedShirt}, {$inc: {likes: 1}, $push: {voters: user}}, (err, data) => {
        if (err || !data) {
          return res.status(400).json({error: "Impossible to like shirt: an error occurred"});
        } else {
          return res.status(200).json({message: "Like added"});
        }
      });
    }
  });
};
```

## Maglietta della settimana

Recupera la maglietta con l'attributo `isMostLiked` a `true`. Questo attributo indica che la maglietta in questione è quella che ha ricevuto più voti durante la scorsa settimana.

```
// GET '/getWeeklyShirt'
const getWeeklyShirt = (req, res) => {
  CommunityShirt.findOne({isMostLiked: true}, (err, data) => {
    if (err || !data || data.length == 0) {
      return res.status(404).json({error: "Impossible to find Weekly Shirt"});
    } else {
      return res.status(200).json(data);
    }
  })
};
```

## Maglietta dell'azienda per l'homepage

Recupera l'indirizzo della maglietta con nome `"homepage-company-shirt"`. Di questa maglietta non ci interessano gli altri dati in quanto è utilizzata solo per la visualizzazione di un'immagine di esempio nella homepage del nostro sito, per rappresentare la sezione delle magliette proposte dall'azienda.

```
// GET '/getHomepageCompanyShirt'
const getHomepageCompanyShirt = (req, res) => {
  CompanyShirt.findOne({name: 'homepage-company-shirt'}, (err, data) => {
    if (err || !data || data.length == 0) {
      return res.status(404).json({error: "Impossible to find homepage company Shirt"});
    } else {
      return res.status(200).json(data.image);
    }
  })
};
```

## Maglietta della community per l'homepage

Recupera l'indirizzo della maglietta con nome "homepage-community-shirt". Di questa maglietta non ci interessano gli altri dati in quanto è utilizzata solo per la visualizzazione di un'immagine di esempio nella homepage del nostro sito, per rappresentare la sezione delle magliette della community.

```
// GET '/getHomepageCommunityShirt'
const getHomepageCommunityShirt = (req, res) => {
  CompanyShirt.findOne({name: 'homepage-community-shirt'}, (err, data) => {
    if (err || !data || data.length == 0) {
      return res.status(404).json({error: "Impossible to find homepage community Shirt"});
    } else {
      return res.status(200).json(data.image);
    }
  })
};
```

## Area utente

Recupera le informazioni dell'utente che accede alla propria area utente per la visualizzazione. Viene utilizzato lo username per trovare i dati corretti.

```
// GET '/getUserData'
const getUserData = (req, res) => {
  User.findOne({username: req.params.username}, async (err, data) => {
    if (err || !data || data.length == 0) {
      return res.status(404).json({error: "Impossible to find User"});
    } else {
      return res.status(200).json(data);
    }
  })
};
```

## Registrazione utente

L'API di registrazione utente riceve in input tutti i dati richiesti dal frontend all'utente al momento della registrazione, che sono username, email, nome, cognome, indirizzo, data di nascita e password. In caso di username già in uso o di password non rispettante la formattazione richiesta (vedi **RNF 4**) la registrazione non avverrà e l'API restituirà il relativo messaggio di errore.

```
// POST '/registerUser'
const registerUser = (req, res) => {
  var pswRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/;

  User.findOne({ username: req.body.username }, (err, data) => {
    if (err) {
      return res.status(400).json({error: "Registration failed"});
    } else if (data) {
      return res.status(400).json({error: "User already exists"});
    } else if (!pswRegex.test(req.body.password)) {
      return res.status(400).json({error: "The password is not 8 characters long, containing a lowerca"});
    } else {
      const newUser = new User({
        username: req.body.username,
        email: req.body.email,
        firstName: req.body.firstName,
        lastName: req.body.lastName,
        address: req.body.address,
        birthDate: req.body.birthDate,
        password: req.body.password,
      });

      newUser.save((err, data) => {
        if (err) {
          return res.status(400).json({error: "Registration failed"});
        }
        return res.status(200).json(data);
      });
    }
  });
};
```

## Login

Permette ad un utente registrato di accedere al suo account. Questa API riceve in input username e password, controllando che siano entrambi corretti (in caso contrario il login fallisce e restituisce il messaggio di errore "Bad credentials"), per poi generare un token da restituire in uscita insieme al messaggio "Logged in successfully" e allo username dell'utente loggato. Tale token verrà utilizzato per verificare la validità della sessione dell'utente.

```
// POST '/login'
const login = (req, res) => {
  User.findOne({ username: req.body.username }, (err, user) => {
    if (!user || user.password !== req.body.password) {
      return res.status(400).json({success: false, error: "Bad credentials"});
    }

    var payload = {username: user.username, email: user.email, address: user.address, time: Date()};
    var token = jwt.sign(payload, process.env.SUEG_SECRET);

    return res.status(200).json({success: true, message: 'Logged in successfully',
      token: token, username: user.username});
  });
};
```

## API documentation

Le API fornite dall'applicazione UrbanShirt e descritte nella sezione precedente sono state documentate utilizzando il modulo NodeJS chiamato Swagger UI Express.

In questo modo la documentazione relativa alle API è direttamente disponibile a chiunque veda il codice sorgente.

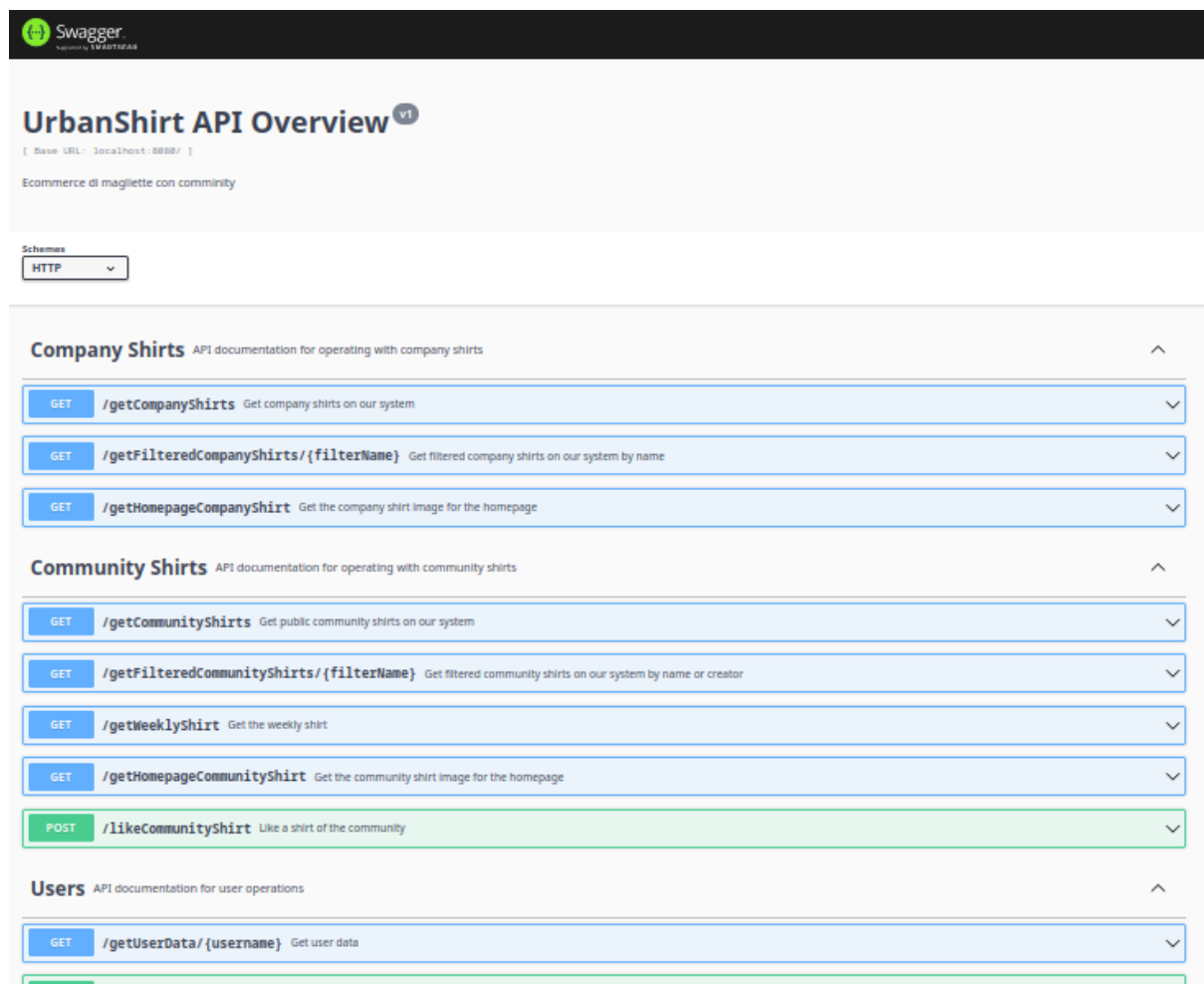
Per poter generare l'endpoint dedicato alla presentazione delle API abbiamo utilizzato Swagger UI in quanto crea una pagina web dalle definizioni delle specifiche OpenAPI.

In particolare, di seguito mostriamo la pagina web relativa alla documentazione che presenta le 2 API (GET, POST) per la gestione dei dati della nostra applicazione:

- La GET viene utilizzata per visualizzare i dati relativi sia alle magliette che a singolo utente. Per le magliette sono previste diverse modalità di recupero dati, mentre per l'utente si vanno a recuperare i dati solo per l'utente loggato che ne richiede la visualizzazione.
- La POST relativamente alle magliette è utilizzata per aggiungere un like alla maglietta della community scelta, mentre nel caso degli utente si usa sia per la registrazione che per il login.

L'url per raggiungere la seguente documentazione e':

<https://urbanshirt.onrender.com/api-docs/>





## FrontEnd Implementation

Il front-end si compone di 5 pagine, tra cui una principale, la Home Page, che connette tutte le altre. Esso fornisce le funzionalità, descritte precedentemente ed erogate dalle corrispondenti API.

### Home Page

Il focus principale della Home Page è fare da vetrina, mettere in evidenza alcuni prodotti che l'utente troverà nelle pagine specifiche. In particolare, al centro della pagina, è situata la maglietta della settimana.



Cliccando sulla scritta *"Maglietta della settimana"*, situata sulla barra colorata in fondo alla pagina, viene visualizzata una finestra che mostra la maglietta della settimana corrente e fornisce le varie informazioni.



## Autenticazione e registrazione

Cliccando nella Home Page sull'icona profilo con accanto la scritta *"Accedi o Registrati"* si viene indirizzati sulla pagina di autenticazione e registrazione.

La pagina presenta due form, una di registrazione e una di autenticazione. Se non si possiedono già delle credenziali valide per l'autenticazione, è necessario compilare il form di registrazione. Per accedere al sistema, invece, bisogna compilare il secondo form, a sinistra, dove vengono richieste le credenziali richieste durante la registrazione. Questa pagina non è più visibile una volta autenticati e ritornati nella Home Page.

The screenshot shows a login and registration interface with a purple header and a light purple background. The title "Entra anche tu nella nostra Community!" is prominently displayed. On the left, a rounded rectangle contains the login form titled "Sei già registrato?". It includes input fields for "username" and "password", and a purple "Accedi" button. On the right, the registration form is divided into two sections. The top section, for new users, includes fields for "nome", "cognome", a date of birth (labeled "gg/mm/aaaa" with a calendar icon), "e-mail", "indirizzo di spedizione", "username", "password", and a "registra" button. The bottom section, for existing users, includes a "conferma password" field and a "Registrati" button.

## Area Utente

Se autenticati nel sistema, nella Home Page, cliccando sull'icona profilo accanto alla scritta *"Area Utente"* si accede alla pagina dedicata alle informazioni dell'utente.

Da qui, è possibile visualizzare le varie informazioni inserite durante la registrazione oppure effettuare la disconnessione dal sistema.



## Area Utente

username:  
mariorossi

email:  
mariorossi@gmail.com

firstName:  
Mario

lastName:  
Rossi

Indirizzo:  
via Verdi 10, Trento, TN, Italia

Data di nascita:  
2001-01-01T00:00:00.000Z

[Visualizza password](#)   [Logout](#)

Cliccando su “*visualizza password*” è possibile vedere la propria password. Cliccando nuovamente sullo stesso tasto, viene nascosta.

Visualizza password

Logout

MarioRossi01@

## Magliette della community

Cliccando nella Home Page sulla scritta “*Le magliette della Community*” si viene indirizzati sulla pagina dedicata alle magliette ideate dalla community.

Accanto ad ogni prodotto sono presenti le loro informazioni e il bottone “*Like*”. Se si è un utente autenticato, cliccandoci sopra è possibile incrementare il numero di “*Like*” complessivi che la maglietta ha ricevuto finora. Per aggiornare il numero di “*Like*”, l’ultimo elemento tra le informazioni della maglietta, è necessario aggiornare la pagina. Per effettuare ricerche più veloci ed efficaci, situata in alto a sinistra c’è la barra di ricerca. L’utente può digitare le parole chiave che identificano il prodotto desiderato e filtrare la lista delle magliette.



## Magliette dell'azienda

Come per le magliette della community, c'è anche una pagina dedicata alle magliette dell'azienda. Per accedervi basta cliccare la scritta *"Le nostre proposte"* presente sulla barra colorata.

La pagina mostra le magliette ideate dall'azienda. Accanto ad ogni prodotto è presente il suo nome.

Per facilitare la ricerca del prodotto ideale, l'utente può ricorrere all'utilizzo della barra di ricerca situata in alto a sinistra. L'inserimento di determinate parole chiave, filtrerà la lista delle magliette presenti sulla pagina.



# Proposte dell'azienda

Cerca  cerca

homepage-company-shirt



maglietta blu c-style



## GitHub Repository and Deployment Info

Il nostro github repository contenente la documentazione e il codice del progetto si trova al seguente indirizzo: <https://github.com/UrbanShirt>.

Presenta una semplice struttura composta da due repositories:

- Implementation, contenente il codice del progetto sia frontend che backend, che è a sua volta suddiviso in:
  - api, contenente tutto il codice per il backend, il file swagger.json per la documentazione delle API, i file di testing sempre relativi alle API, e i fondamentali package.json, per l'installazione dei moduli node, e server.js, file per avviare effettivamente l'applicazione.
  - ui, contenente tutto il codice per il frontend, compreso lo scripting per la gestione delle chiamate a backend.
  - images, contenente le immagini delle magliette usate come esempio.
- Deliverable, contenente i cinque file PDF definitivi dei cinque deliverable.

L'applicazione è disponibile online sulla piattaforma render.com, raggiungibile al seguente url:

<https://urbanshirt.onrender.com>

## Testing

Nella fase di testing sono stati riscontrati diversi problemi. Inizialmente si può notare come la percentuale di codice coperto risulti relativamente bassa:

All files  
67.05% Statements 334/378 41.42% Branches 28/78 37.83% Functions 34/97 67.05% Lines 334/378

Press n or j to go to the next uncovered block, b, p or k for the previous block.

Filter:

File	Statements	Branches	Functions	Lines
api	62.22%	28/45	0%	2/13
api/controllers	52.43%	43/82	46.03%	12/24
api/models	100%	12/12	100%	0/0
api/routes	100%	31/31	100%	0/0

Andando nello specifico notiamo dei valori un po' anomali riguardo la copertura di api/controllers. Alcune funzioni, seppur testate, non risultano coperte (come si può notare in modo chiaro nella sezione user che segnala 0/7 funzioni coperte)


All files api/controllers  
52.43% Statements 43/82 46.03% Branches 28/82 50% Functions 32/74 52.43% Lines 43/82

Press n or j to go to the next uncovered block, b, p or k for the previous block.

Filter:

File	Statements	Branches	Functions	Lines
shirt.js	69.23%	36/52	63.04%	12/17
user.js	23.33%	7/30	0%	0/7

Abbiamo riscontrato ulteriori problemi poi al momento dell'esecuzione dei test, notando una certa incoerenza nei risultati dei test. Gli stessi casi di test testati per più volte di seguito a volte falliscono, come si può notare dai risultati differenti nei due screenshot:



```
Test Suites: 1 failed, 1 passed, 2 total
Tests:      6 failed, 23 passed, 29 total
Snapshots:  0 total
Time:       7.952 s
```

```
Test Suites: 1 failed, 1 passed, 2 total
Tests:      7 failed, 22 passed, 29 total
Snapshots:  0 total
Time:       8.102 s
```

Indagando è risultato essere un problema dovuto dall'asincronicità delle operazioni a database, le quali sono appunto asincrone per definizione. Nel caso quindi in cui un test si aspettasse di trovare il database vuoto spesso lo trova invece popolato in quanto una insertMany avviata in un test precedente poteva essere ancora in corso.

Per tentare di risolvere questa problematica abbiamo provato innanzitutto a sfruttare al meglio la logica async/await offerta da javascript, senza però notare alcuna differenza, e in secondo luogo abbiamo aggiunto il tag `-runInBand` al lancio di jest, il quale dovrebbe far eseguire i test in sequenza anzichè utilizzando il normale pool, ma nemmeno questo ha risolto il problema.

Ci siamo allora assicurati che i singoli test non fossero il problema, testandoli per verificarne l'esito positivo, ed effettivamente non causano errori, riportando l'esito corretto.