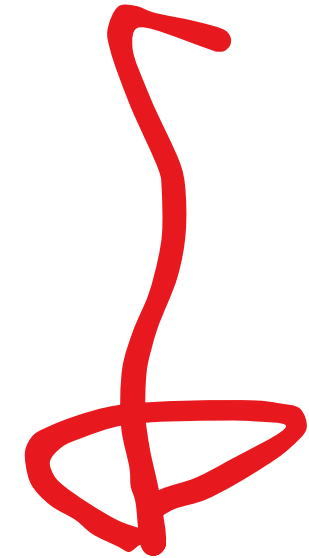


Cartella:FoodMine  **ng new frontend** 

ng g c components/partials/header

import google fonts in style.css



Cartella shared/models  **data.ts con tutti gli alimenti**
Food.ts **+ add immagini in assets**

ng g c

components/pages/home



Crea FoodService 

 npm i RatingModule, per le stelle
Metodi

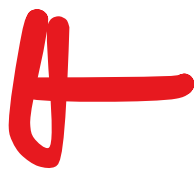
FoodService: GetAll, Search, FoodById 

 route params per filtrare i cibi in home

ng g c components(partials/search 

 ng g c components/pages/food-page => pagina che si
apre cliccando su un alimento 

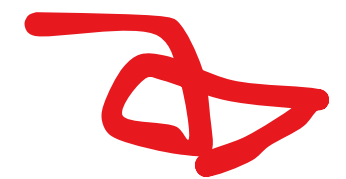
Aggiungi ad home

 FoodService:
getAllTags,
getAllFoodByTag

 crea in Models
Tag.ts 

ng g c components/partials/tags →

CartItem.ts nei Models



Metodi CartService=>

addtoCart

removeFromCart

ChangeQuantity

clearCart

crea CartService



Cart.ts in models

addToCart metodo in food page



ng g c components/pages/cart-page/
Questo è il componente del carrello



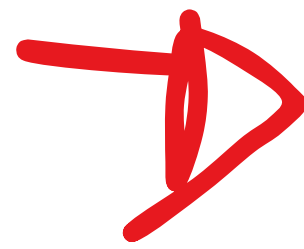
dal CartService metodi:

removeFromCart

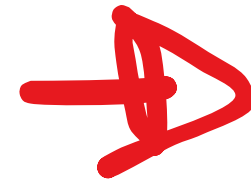
changeQuantity



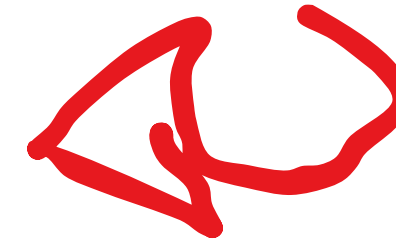
getCartObservable(per restituire una
copia del carrello
setCartToLocalStorage:(aggiorna il
carrello e lo salva nello storage del
browser(utilizziamo JSON.stringify per
convertire un oggetto in stringa)
getCartFromLocalStorage:(recupera i
dati salvati se sono presenti)



create titleComponent



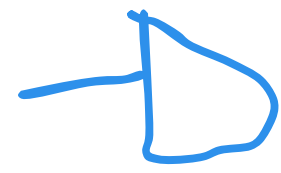
Gli passiamo valori dal padre @input, è più essere utilizzata anche da altri componenti



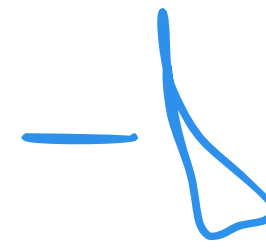
creiamo il componente not-found
con @input visible ecc per farli visualizzare negli altri
componenti in caso di errore

Passiamo al Backend





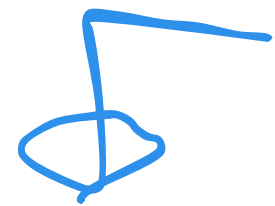
Crea nuova cartella Backend



npm init -y



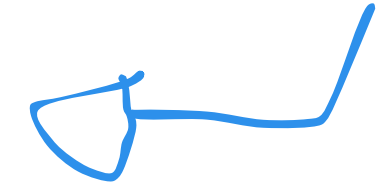
iniziamo a installare i
pacchetti ts ecc



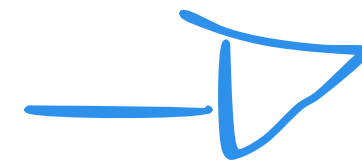
install express and cors



copy data.ts dal front



creiamo un server web con server.js
import { dbConnect } from
'./configs/database.config';
dbConnect(); con questo colleghiamo il
nostro MongoDB

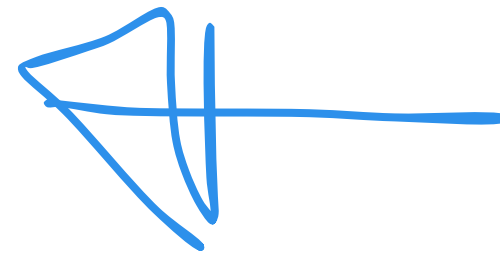


Utilizziamo un web server local host per maggiore
scalabilità, comunicazione tra front e database con la
gestione api, e maggiore sicurezza per il database

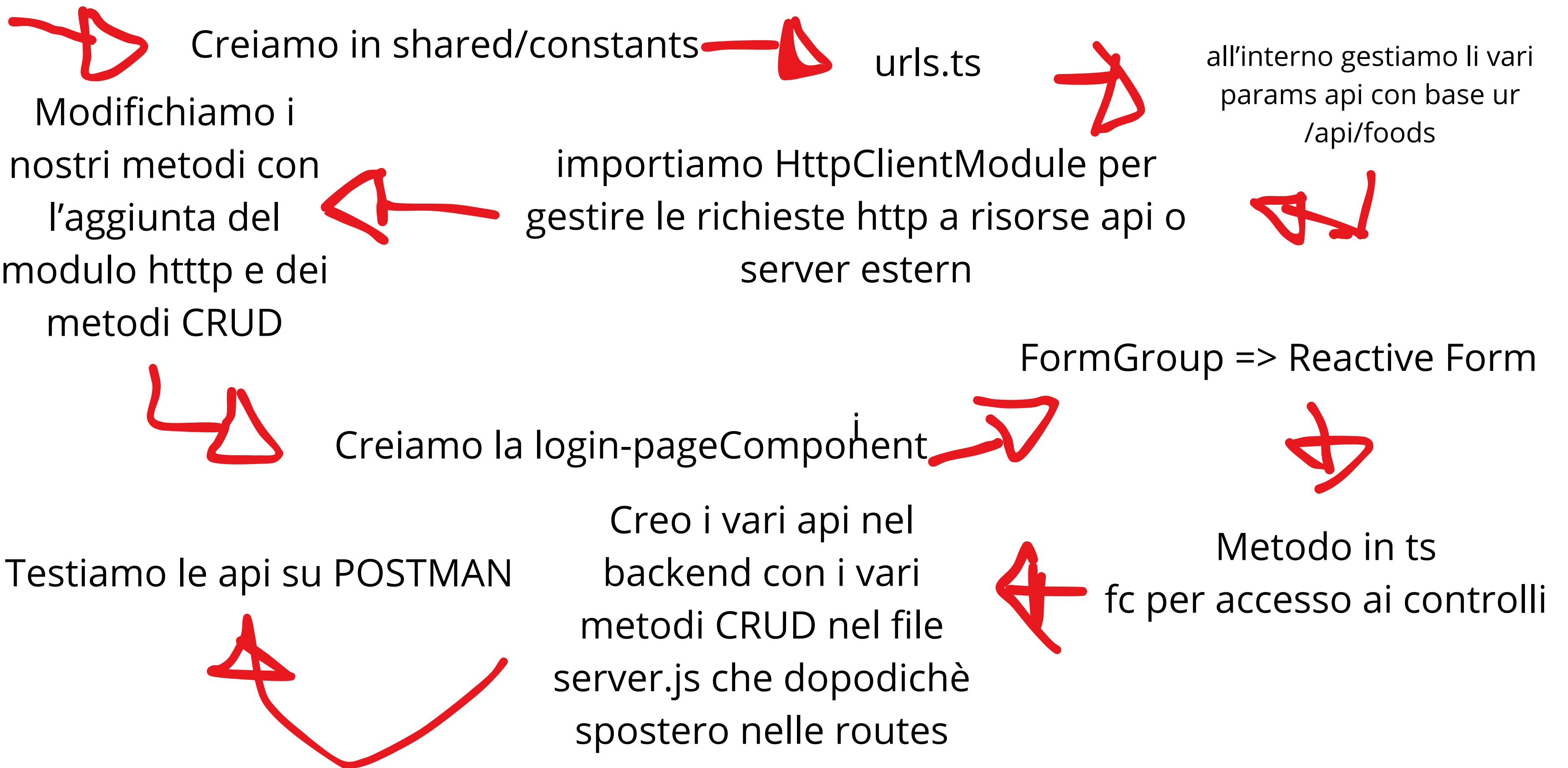
installiamo nodemon per un live
server e



Collegiamo back e front



ts-node ci consente di scrivere
script da riga di comando



Creiamo un nuovo servizio: UserService — Nuovo Modello User.ts

Creiamo nella cartella : Interfaces:
ILogin con i due valori email e passw

Creiamo il metodo login in UserService con il metodo post per l'aggiunta nuovo user

usiamo operatore pipe con tap() metodo che ci consente di non alterare un flusso di
dati generando per esempio un risultato in base al risultato

ngx-toastr

libreria di notifiche popup che useremo in caso di error
o success

Aggiungiamo i metodi dal service al componente login-page

Aggiungiamo in UserService:

setUserToLocalStorage
getUserFromLocalStorage

Gestiscono l'archiviazione dei dati

Implementiamo i
vari buttons corretti
per logout, login e
profile

Torniamo nel HeaderComponent

Lo aggiungiamo
quando vogliamo una
risposta ad un input
inserito nei form o
altro

ng g components/partials/input-container

Utilizzabile sempre
per i form in caso di
input non validi

Creiamo nuovo componente
input-validation

Componente text-input → Lo utilizziamo sempre per i form con vari alert in base all'errore → creiamo un componente default-bottom

Torniamo al backend

Aggiungiamo una serie di stili a questo button, che potremo utilizzare già di default per altri componenti

importiamo nel server.js la rotta food.router.ts contenente tutti i metodi crud con url api

Creiamo l'user.router.ts

→ Aggiungiamo i metodi
che avevamo creato
precedentemente nel
server.js e passiamoli
all'user.router.js

Per
generateTokenRespo
nse installare la
libreria jwt per creare
token sicuri

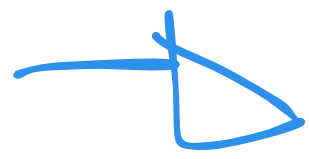
Creiamo il file database.config.js
dove inseriremo il collegamento al
database con la libreria mongoose

→ Creiamo il nostro mongoDB atlas

→ Creiamo il nostro file .env e
inseriamo l'url mongo con nome e
passw

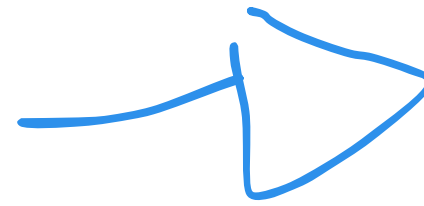
→ Installiamo i pacchetti mongoose,
dotenv,
bcryptjs

→ jsonwebtoken
express-async-handler

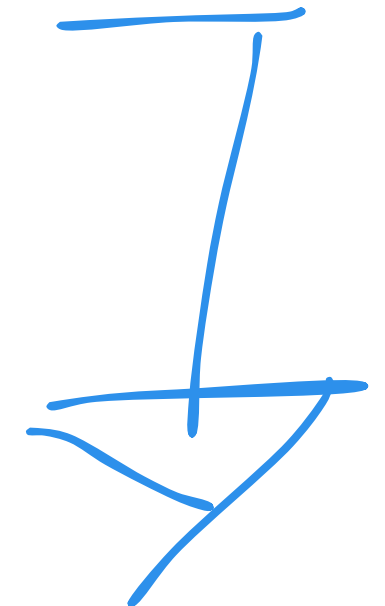


Ora creiamo i modelli,
food.model.ts:
interfaccia Food e
implementiamo
Schema con i valori
timestamps:true
aggiungiamo i campi
temporali

Andiamo nel FrontEnd
e nel UserService
aggiungiamo il metodo



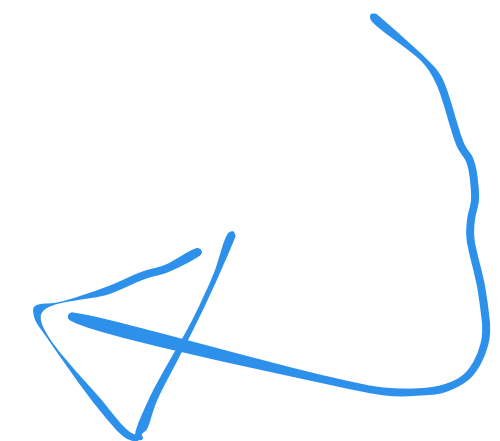
Creiamo User.model.ts
interfaccia user(per
avere un codice più
ordinato)
Schema con i valori

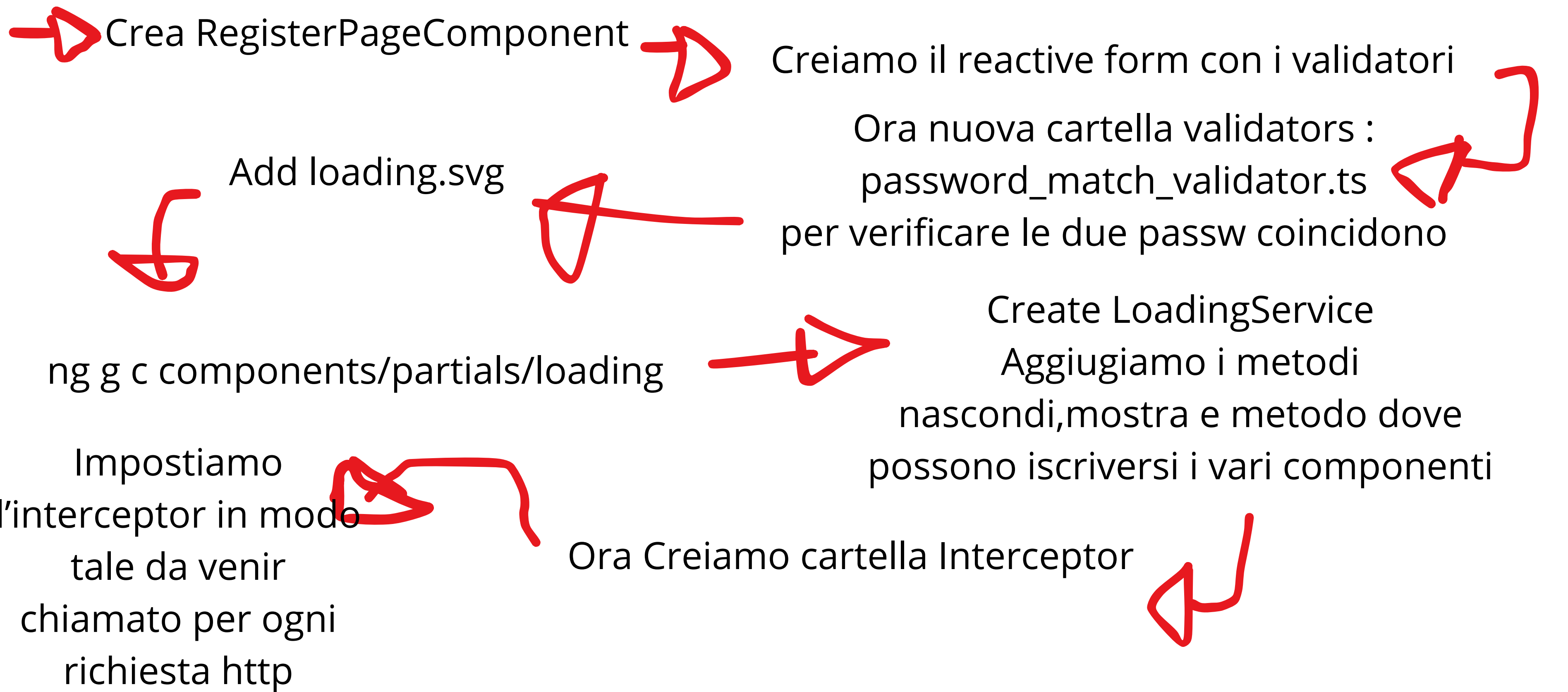


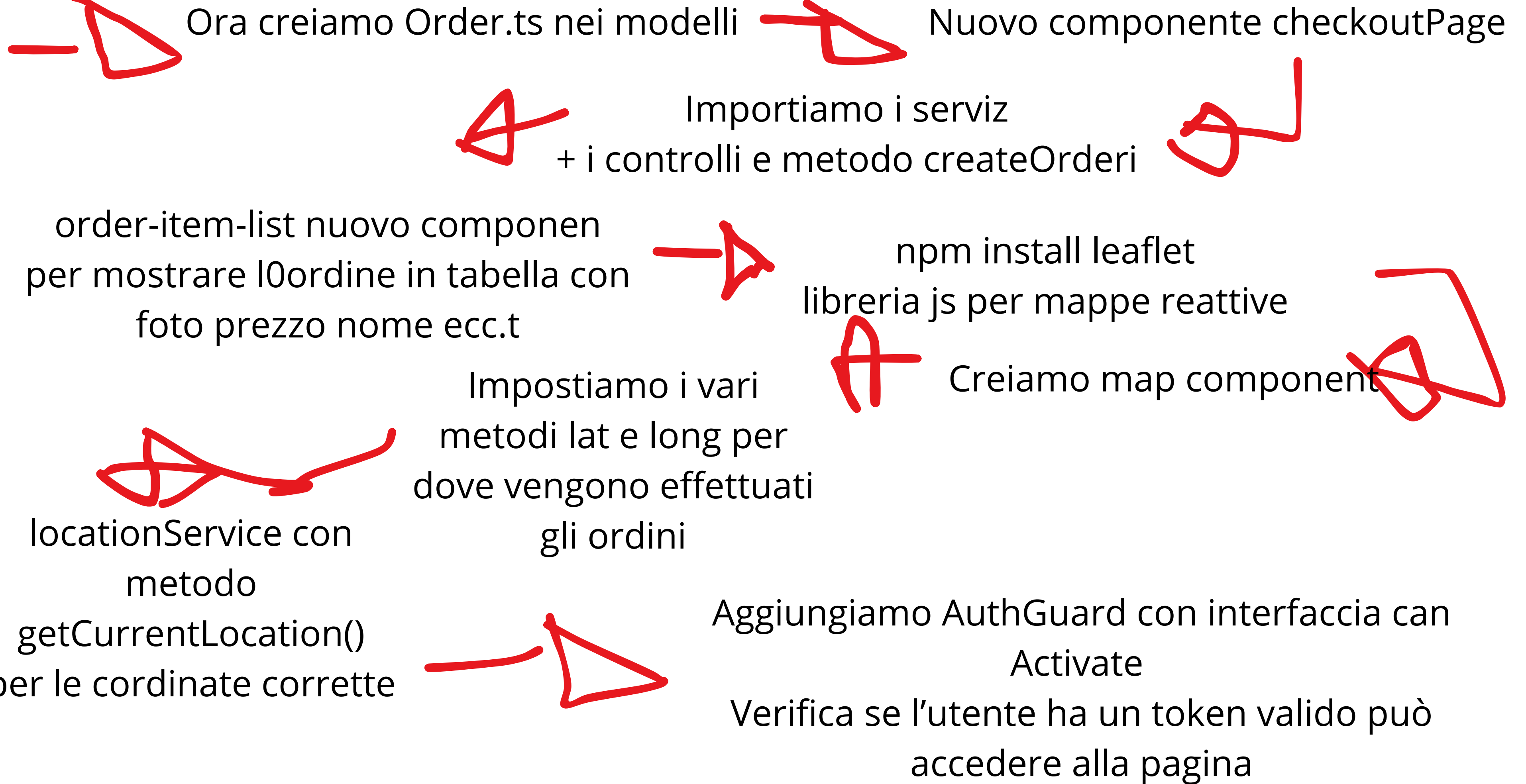
Completiamo i routes con i vari me
aggiornati con i modelli e l'uso d
pacchetti



In User.router.ts
Aggiungiamo
metodo per la
registrazione
+bcrypt per la
password





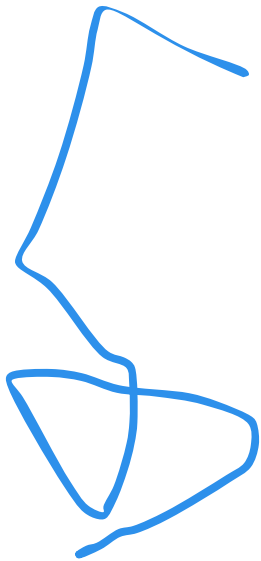




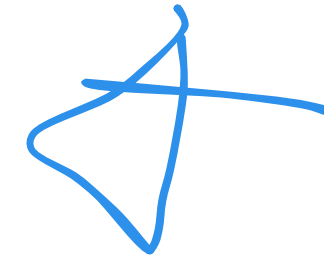
Creiamo nel backend l'orderModel.ts



implementiamo
Schema per latitudine
e longitudine map
+OrderSchema



Creiamo cartella constants
aggiungendo i enumeratori
(enum) per poterli chiamare
quando necessario



Creiamo il middle auth
Viene estratto il token jwt si verifica se
esiste e si decodifica
da errore o uno stato di successo



Lato
FrontEndCreiamo
OrderService

Gestiamo tutti i vari metodi esist

Creiamo order.router.ts con i vari metodi :
CREAZIONE

RECUPERO NUOVO ORDINE PER L'UTENTE

PAGAMENTO

TRACCIAMENTO ORDINE

VISUALIZZAZIONE ORDINE

➤ Creiamo auth.interceptor
intercetta le richieste http e verifica che
l'utente sia autenticato dopo di che crea per
ogni diverso utente un access token e con
next..handle va avanti

➤ payment-page componen
collega metodo tra back end e service e adda
map

➤ Crea paypal-button component
PAYPAL SDK

➤ create order tracker

RENDER.COM

Package.json

start= cd backend/built && node server.js

prebuild:cd backend && npm run build

build: cd frontend && npm run build

+add Environment

npm start