FILIERA360

DIARIO

29/04/2025

- individuato file json per la memorizzazione dei dati (users)
- creato diagramma E-R
- ricerca per scegliere DBMS : MySql (con libreria pymysql)
- aggiunto container “mysql” modificando il codice in docker-compose.yaml
- creato connessione al db in app.py

30/04/2025

- scrittura delle query per la creazione delle tabelle nel db “filiera360” in file “mysql.sql” (tabelle create: users, otp_codes, liked_products, searches, models, invite_tokens)
- salvati utenti e OTP nel database: modificate le funzioni send_otp, save_users, load_users e le route /signup, /login e /verify-otp”

02/04/2025

- modificato le route /operators, /operators/add, /operators/delete e la funzione required_permissions

06/05/2025

- rimosso psw del db dal codice e inserita in .env (presente in .gitignore)

07/05/2024

- modificate find_producer_by_operator, verify_product_authorization, upload_product, update_product, add_certification_data, load_models, save_models, upload_model, get_model
- modificata tabella models⟶ script LONGTEXT e ID VARCHAR(50)

03/07/2025

- ho introdotto la funzione def get_db_connection()
- modificate route forgot-password, is_valid_invite_token, uploadBatch, updateBatch, addSensorData, addMovementsData, likeProduct, unlikeProduct, getLikedProducts, addRecentlySearched, getRecentlySearched
- modificata tabella searches
- aggiunto per test su scenari frequenti (login, sign up, verifica otp, like product, unliked product, recently searched) (ho installato axios)
- script per stress test con locust (ho installato locust)

07/07/2025

- modifiche per stress test (in app.py e in locust.py)

21/07/2015

- creato cartella database in cui ho definito le query e la connessione al db

WORKFLOW

1. **Analisi dei dati e modellazione del database relazionale**
    - Raccogliere tutti i file per la gestione dei dati
    - Individuare le entità principali e le loro relazioni (uno a molti, uno a uno...)
        - Realizzazione delle tabelle secondo lo schema relazionale definito in precedenza tramite DBMS scelto
    - Realizzare uno schema E-R

2. **Implementazione del database relazionale**
    - Scelta del DBMS da utilizzare (Mysql)
    - Aggiungere il container per il database in Docker modificando il file docker-compose.yaml
    - Effettuare connessione al database in app.py
    - Modificare il codice in app.py per effettuare la migrazione della gestione dei dati con file JSON al database
3. **Verifica e valutazione delle prestazioni**
    - Effettuare test su scenari che si verificano frequentemente
    - Effettuare stress test con una mole di dati crescente
4. **Confronto dei risultati con quelli ottenuti nel 2.2**
    - Effettuare il confronto di entrambi i metodi di gestione dei dati, per analizzare alcuni aspetti come il tempo di accesso ai dati, la scalabilità e la facilità di aggiornare le informazioni.

DEVELOPMENT

Task 01

File per la gestione dei dati:

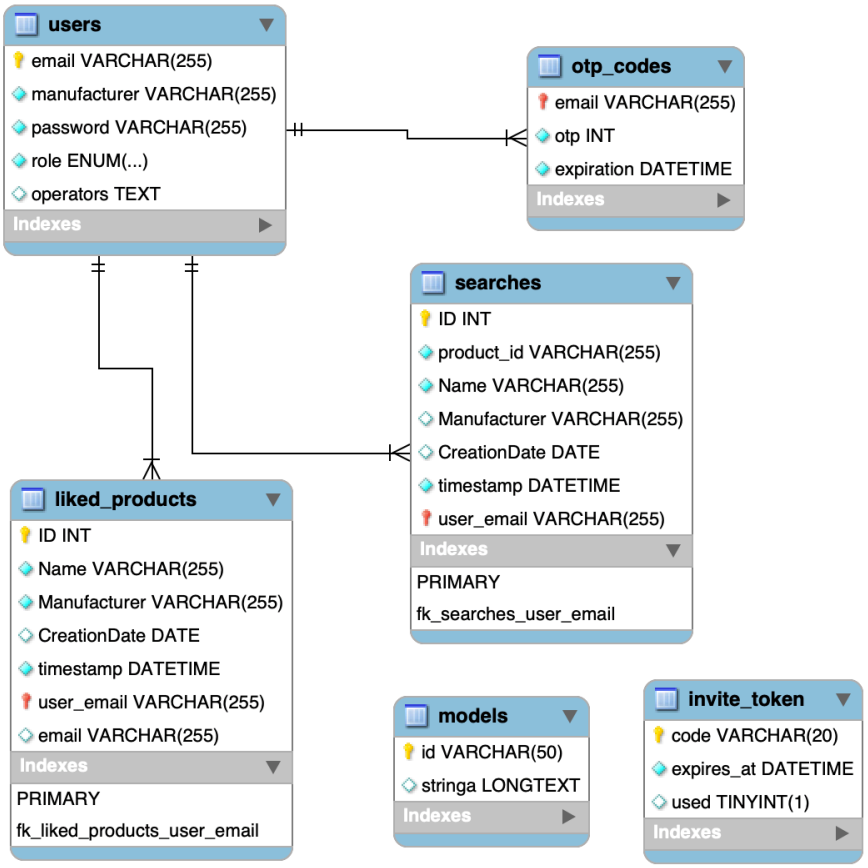
1. users.json
2. users-otp.json
3. searches.json
4. liked-products.json
5. models.json
6. invite_tokens.json

Entità principali e le loro relazioni:

1. Users (email chiave primaria)
2. Otp_codes (email chiave primaria e chiave esterna)
3. Searches (id chiave primaria, user_email chiave esterna)
4. liked_products (id e user_email (chiave esterna) chiave primaria)
5. models (id chiave primaria)
6. invite_tokens (code chiave primaria)

Ogni utente riceve tramite e-mail più codici OTP in quanto sono temporanei (un codice OTP può essere attribuito ad un singolo utente). Ogni utente può cercare prodotti e lotti (i prodotti e i lotti possono essere cercati da un utente). Ogni utente può aggiungere prodotti nei preferiti. Ogni utente producer può caricare un modello3D per ogni prodotto inserito. Poichè il sistema è ancora in fase sperimentale, l'entità invite_tokens e models non presenta delle relazioni con altre entità, attualmente è presente un unico token che viene settato a true quando viene utilizzato dall'utente.

Diagramma E-R con mysql workbench



Task 02

DBMS da utilizzare: MySQL con PyMySQL

- Modificato il file requirements.txt: ho aggiunto le librerie pymysql, cryptography e pytz per ottenere il fuso orario di Roma

Modificato il file docker-compose.yaml:

- Ho aggiunto il container per il database (MySQL) e in particolare ho usato `healthcheck` per controllare se il container fosse funzionante, in quanto quest'ultimo necessitava di qualche secondo in più per avviarsi. Tuttavia, il backend cercava di connettersi a MySQL prima che il container fosse pronto, quindi la connessione falliva.
- Ho definito le variabili d'ambiente nel container del backend
- Ho impostato la dipendenza del backend a MySQL (il backend non può essere avviato se MySQL non è pronto)
- Ho aggiunto il volume mysql_data per far sì che i dati non vengano persi anche se il container dovesse essere interrotto o cancellato.

Creato db_connection.py per stabile la connessione con il db:

- Ho aggiunto le librerie pymysql, time (per generare una pausa tra un tentativo e l'altro) e pytz
- ho usato `pymysql.cursors.DictCursor` è un tipo di cursore che restituisce i **risultati delle query come dizionari** (anziché come tuple). Questo è utile perché, invece di lavorare con indici numerici per accedere ai dati, puoi utilizzare i nomi delle colonne. Questo garantisce maggiore leggibilità, maggiore gestione dei dati e facilità di modifica
- Ho introdotto la funzione `def get_db_connection()` perchè inizialmente la variabile globale connection era aperta una sola volta all'inizio e usata per tutte le rotte con

  `with connection.cursor()` as cursor:

  Questo funziona solo se viene eseguita una operazione per volta (single threaded).

Ho riscontrato questo problema quando dovevo gestire le richieste recenti e i mi piace ai prodotti (multi-thread).

- Aggiunto parti di codice per connettermi al database in Docker e, se la connessione fallisce, ho considerato altri 10 tentativi (questo perché inizialmente la connessione falliva, in particolare il backend non riusciva a collegarsi al container del database perché MySQL necessita di più tempo per avviarsi)

Creazioni entità nel db:

- definisco le tabelle: users, otp_codes, liked_products, searches, models, invite_tokens
- DROP TABLE IF EXISTS nometabella; → **eliminano le tabelle se esistono già**
- ogni record in `otp_codes`, `liked_products`, `searches` deve riferirsi a un `user` esistente; se un utente viene eliminato, allora saranno eliminati anche i record in riferimento a quell'utente nelle tabella sopra citate.

Scrittura delle query:

OTP

- insert_or_update_otp:
  - ottiene il fuso orario di Roma (Italia) usando la libreria `pytz` (senza questo, l'orario di scadenza risultava essere 2 ore indietro rispetto a quello locale)
  - è aperta una connessione nel db tramite cursore per verificare l'esistenza dell'utente nel db attraverso una query. Se l'utente non è presente, restituisce una risposta JSON con messaggio di errore.
  - inserimento/aggiornamento dell'otp: viene eseguita una query per aggiungere una riga oppure se è già presente una riga in corrispondenza di quella email, il codice otp e la data di scadenza vengono aggiornati senza aggiungere un'ulteriore riga (`ON DUPLICATE KEY UPDATE`). infine viene eseguito il commit per salvare le modifiche nel db in maniera permanente
- get_otp_record
  - recupera dal db il codice OTP e la relativa data di scadenza associati a un utente, identificato dal suo indirizzo email.
- get_latest_otp (usato solo per recuperare otp in fase di stress test)
  - serve a **recuperare l'OTP più recente** associato a un determinato indirizzo email dalla tabella `otp_codes`.

USERS

- check_email_exists
  - controlla se esiste un utente registrato con quella email nel db
- check_manufacturer_exists
  - verifica se esiste già un produttore con quel nome (`manufacturer`) nella tabella `users`.
- insert_user
  - inserisce un nuovo utente nel db
- get_user_by_email
  - recupera tutti i dati dell'utente conoscendo l'email
- get_user_operators
  - restituisce la lista degli operatori associati all'utente
- get_user_role
  - restituisce il ruolo dell'utente
- get_raw_operators
  - restituisce l'elenco degli operatori. Il campo operators è salvato nel DB come testo JSON**,** viene convertito in lista Python con json.loads() (se è vuoto
    o None restituisce []).
- update_user_operators
  - aggiorna il campo operators dell'utente.
- get_manufacturer_by_email

    - restituisce il manufacturer associato ad un utente
- update_user_password
    - aggiorna la psw dell'utente

INVITE TOKEN

- fetch_invite_token_data
    - recupera le informazione del token
- mark_invite_token_used
    - segna un token come usato nel db

ACCESS CONTROL

- find_producer_by_operator
    - trova il produttore associato a un operatore (se esiste)
- required_permissions
    - verifica se l'utente ha uno dei ruoli richiesti per svolgere determinate funzioni
- verify_product_authorization
    - verifica se l'utente ha il permesso per gestire un determinato prodotto

MODELS

- save_or_update_model
    - salva o aggiorna un modello 3D associato al prodotto
- get_model_by_product_id
    - recupera il modello 3D in base64 associato a un product_id

LIKES

- has_user_liked_product
    - controlla se l'utente ha già messo mi piace a quel determinato prodotto
- add_product_like
    - aggiunge il prodotto piaciuto nella tabela liked_products
- remove_product_like
    - rimuove il like al prodotto (quindi il prodotto viene rimosso dalla tabella liked_products)
- get_user_liked_products
    - restituisce tutti i prodotti a cui l'utente ha messo mi piace

SEARCHES

- add_recent_search
    - salva una ricerca recente dell'utente e mantiene solo le ultime 5 per ogni utente.
- get_recent_searches
    - recupera le ultime 5 ricerche effettuate dall'utente loggato

Task 03

Test:

Ho introdotto uno script per testare il funzionamento delle API principali. Ho considerato una funzione asincrona che permette di calcolare il tempo di esecuzione di ogni singolo test.

RISULTATI TEST:

🔷 Login utente ruolo 'user' (JWT)
Status: 200
Response: {

access_token:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTc1MTM4NDg4MSwianRpIjoiMTIzN2N
aYPpXaYmxy8Lh9XnHCLJL8',
email: 'user1@gmail.com',
manufacturer: 'user1',
message: 'Login successful',
role: 'user'
}

- Login utente ruolo 'user' completato in 294 ms

🔷 Login utente producer o operator (OTP)
Status: 200
Response: { message: 'OTP sent to your email.' }

- Login utente producer o operator completato in 2688 ms

🔷 Login con password errata
Error: 401 { message: 'Invalid email or password' }

- Login con password errata completato in 279 ms

🔷 Login con email inesistente
Error: 401 { message: 'Invalid email or password' }

- Login con email inesistente completato in 4 ms

🔷 OTP valido
Status: 200
Response: {
access_token:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTc1MTM4NTE5OCwianRpIjoiYzExMTM
IxKzRFs',
email: 'user3@gmail.com',
manufacturer: 'user3',
message: 'OTP validated successfully.',
role: 'producer'
}

- OTP valido completato in 7 ms

🔷 OTP mancante
Error: 400 { message: 'Invalid or expired OTP.' }

- OTP mancante completato in 4 ms

🔷 OTP errato
Error: 400 { message: 'OTP has expired or is invalid.' }

- OTP errato completato in 3 ms

🔷 Email non esistente (OTP)
Error: 400 { message: 'OTP has expired or is invalid.' }

- OTP email non esistente completato in 4 ms

🔷 Signup utente normale (user)
Status: 201
Response: { message: 'User registered successfully' }

- Signup utente normale completato in 250 ms

🔷 Signup con email già registrata
Error: 409 { message: 'Email already exists' }

- Signup email già esistente completato in 3 ms

🔷 Signup producer con token
Status: 201
Response: { message: 'User registered successfully' }

- Signup producer con token completato in 272 ms

🔷 Signup producer senza token
Error: 400 { message: 'The invite token is required for producers.' }

- Signup producer senza token completato in 5 ms

🔷 Like product
Status: 200
Response: { message: 'Product already liked' }

- Like prodotto completato in 5 ms

🔷 Unlike product (con JWT)
Status: 200
Response: { message: 'Product unliked successfully' }

- Unlike prodotto completato in 9 ms

🔷 Add recently searched
Status: 200
Response: { message: 'Product added to recently searched' }

- Add recently searched completato in 6 ms

Tutti i test completati in 3.833s

Stress test:

LOCUST→ test scritti in python; prevede interfaccia UI interattiva in cui è possibile avviare e stoppare il test, scegliere il numero di utenti simultanei e osservare come variano le metriche in tempo reale.

L'interfaccia utente presenta:

- numero di utenti in parallelo (attivi contemporaneamente)
- utenti avviati al secondo (quanti utenti si aggiungono ogni secondo fino ad arrivare al numero massimo definito precedentemente)
- host: indirizzo del sito

Alcune delle metriche che è possibile osservare sono:

— mediana: tempo affinchè il 50% delle richiesta sia elaborato.

— Il **95° percentile** è il tempo **sotto il quale il 95% delle richieste viene completato**.

— Il **99° percentile** è il tempo **sotto il quale il 99% delle richieste viene completato**.

In app.py, nella funzione send_otp è necessario commentare l'invio dell'email, e scrivere un codice per simularne l'invio. Il codice stampa l'otp relativo all'email e viene inviata una risposta positiva all'utente

Ho aggiunto `/get-latest-otp` che permette di ottenere l'ultimo OTP generato per un determinato indirizzo email. Verifico che l'email sia presente. Recupero otp relativo all'email, se l'otp non esiste ritorna errore

FREQUENZA:

- `@task(1)`: login, get_recently_searched, get_liked_products → meno frequente
- `@task(2)`: signup, like_product, add_recently_searched → più frequenti

In Locust, `self.client` è un oggetto che simula un client HTTP (come un browser) e permette di inviare richieste ai tuoi endpoint.

COSA FA QUESTO SCRIPT?

- Recupera un OTP per `user90@gmail.com` (utente già salvato nel db)
- Verifica l'OTP per ottenere un token di accesso
- Esegue una serie di operazioni:

- Login con email/password
- Signup (registra nuovi utenti con email random con il relativo manufacturer→
  - uuid.uuid4() genera un UUID (casuale)
  - .hex lo converte in una stringa esadecimale (senza trattini)
  - [:6] prende solo i primi 6 caratteri
  - Risultato: ogni utente ha un'email diversa e unica)3
- Like su un prodotto
- Aggiunge un prodotto ai "recentemente cercati"
- Recupera prodotti "recentemente cercati"
- Recupera prodotti "piaciuti"

Presenta `wait_time`: attesa casuale tra 1 e 3 secondi tra le richieste.

`on_start()` viene eseguito una sola volta per ogni utente virtuale prima di iniziare ad eseguire i task. L'utente user90 invia una GET a `/get-latest-otp` per ottenere l'otp. Se l'otp non viene recuperato, imposta `self.token= none` e le richieste autenticate per questo utente saranno saltate. Estrae l'OTP ricevuto dalla risposta JSON. Invia una POST a `/verify-otp` per verificare l'OTP. Questo dovrebbe restituire un access token se l'otp è corretto. Se il login OTP va a buon fine, il token viene salvato in `self.token`. Questo token verrà poi usato negli altri task come header di autenticazione.

REPORT TEST

- *PRESTAZIONI DEL SISTEMA*

ENDPOINT	***TEMPO MEDIO***
POST /login	290 ms
POST /login (OTP)	2688 ms (endpoint più lento)
POST /verify-otp	7 ms
GET /getLikedProducts	8 ms
GET /getRecentlySearched	7.7 ms
POST /likeProduct	7.6 ms
POST /addRecentlySearched	12.7 ms
POST /signup	271 ms

- *SCALABILITA'*
  Utenti in parallelo: 100
  Utenti avviati al secondo: 4
  Numero di richieste totali: 3409
  Errori: 5
  RPS (numero medio di richieste al secondo): 41.97

During: 04/07/2025, 20:02:11 - 04/07/2025, 20:03:32 (1 minute and 21 seconds)
Target Host: http://localhost:5001
Script: locustfile.py

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/addRecentlySearched	702	0	22.52	5	140	49	8.64	0
GET	/get-latest-otp?email=user90%40gmail.com	100	0	17.71	8	55	15	1.23	0
GET	/getLikedProducts	329	0	11.85	3	69	347	4.05	0
GET	/getRecentlySearched?userEmail=user90%40gmail.com	337	0	13.12	4	83	387	4.15	0
POST	/likeProduct	714	0	12.67	3	124	36	8.79	0
POST	/login	380	0	340.76	245	747	61	4.68	0
POST	/signup	747	1	336.95	21	820	42.99	9.2	0.01
POST	/verify-otp	100	4	17.25	7	95	456.56	1.23	0.05
	Aggregated	3409	5	122.58	3	820	119.43	41.97	0.06

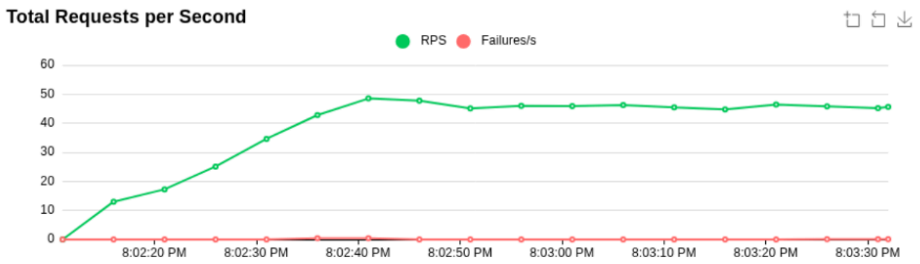
Response Time Statistics

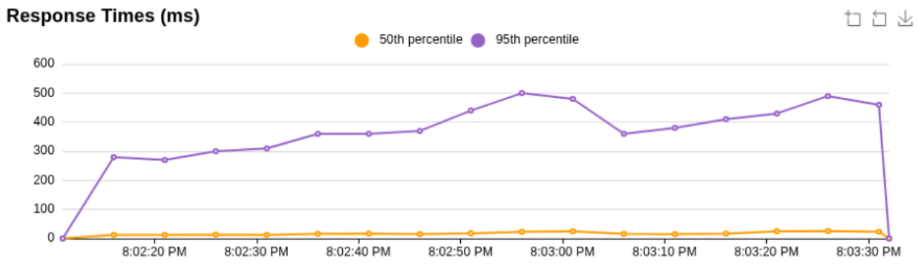
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/addRecentlySearched	14	17	21	29	55	74	100	140
GET	/get-latest-otp?email=user90%40gmail.com	15	16	17	22	27	50	55	55
GET	/getLikedProducts	9	10	11	15	24	31	52	69
GET	/getRecentlySearched?userEmail=user90%40gmail.com	9	11	13	18	28	37	54	83
POST	/likeProduct	9	10	13	16	26	36	57	120
POST	/login	320	330	350	390	460	530	670	750
POST	/signup	310	320	350	380	450	540	680	820
POST	/verify-otp	14	16	17	19	25	30	95	95
	Aggregated	17	30	260	300	350	410	580	820

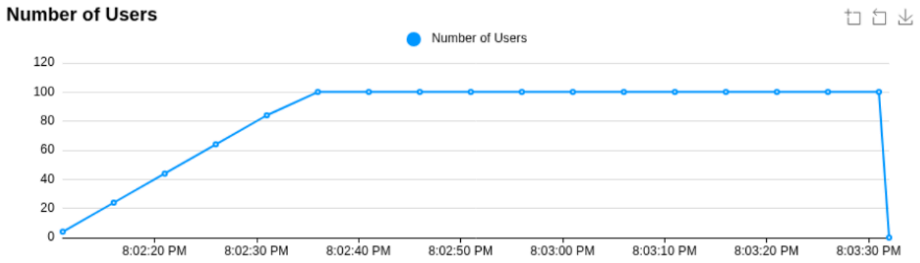
Failures Statistics

# Failures	Method	Name	Message
1	POST	/signup	HTTPError('409 Client Error: CONFLICT for url: /signup')
4	POST	/verify-otp	HTTPError('400 Client Error: BAD REQUEST for url: /verify-otp')

Charts







METRICHE STANDARD PER ANALIZZARE I DATI OTTENUTI:

Nielsen Norman Group - LIMITI DI RISPOSTA

- 100 ms → risposta istantanea
- 1 s → l'utente percepisce un ritardo, ma non vi è perdita d'attenzione dello stesso.
- da 1 a 10 s → l'utente perde l'attenzione ed è difficile recuperarla quando la risposta viene fornita

Google – Core Web Vitals

- LCP – Largest Contentful Paint → misura le prestazioni di caricamento (`/login` , `/signup` , `/getLikedProducts` , `/getRecentlySearched`)
- INP – Interaction to Next Paint → misura l'interattività dell'utente (`/likeProduct` , `/addRecentlySearched` , `/verify-otp`)
- CLS – Cumulative Layout Shift → misura la stabilità visiva

Google Cloud - Throughput (RPS)

https://cloud.google.com/run/docs/about-concurrency

- Numero di richieste gestite al secondo → 5–20 RPS per istanza

K6 BLOG - Error Rate (%)

https://medium.com/qest/load-testing-with-k6-ef17a1f64def

- Percentuale di richieste fallite su totale → <0.1% eccellente, 0.1–1% accettabile, >1% critico.

INTERPRETAZIONE RISULTATI OTTENUTI TRAMITE TEST FUNZIONALI E STRESS TEST:

- `verify-otp`, `getLikedProducts`, `getRecentlySearched`, `likeProduct`: ottimo poichè sotto i 100 ms
- `POST /login`, `POST /signup`, `/addRecentlySearched`: tempi medi entro i limiti raccomandati da Google e NN/g
- `POST /login (OTP)` : **oltre 2,5 secondi**, è *superiore alla soglia ottimale per i limiti Web Vitals*.

Metrica	Valore FILIERA360	Valutazione
RPS	41.97	Buono (adatto per sistemi in fase sperimentale)
Error rate	0.15%	Buono
Tempo medio	7–300 ms	Ottimo (eccetto login con OTP)
OTP time	2688 ms	più lento (da migliorare)