



UNIVERSITY
OF TRENTO

Dipartimento di Ingegneria e
Scienza dell'Informazione

Drop Zero

Documento D2



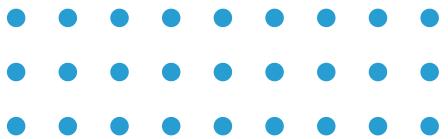
Gruppo 045:

- Sami Facchinelli
- Gabriele Chini
- Tiziano Manfredi



Indice

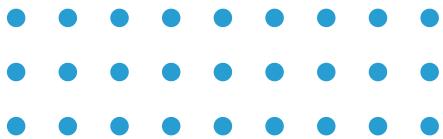
- 01** • Scopo del documento
- 02** • Web APIs
- 03** • Implementation
 - Repository Organization
 - Branching strategy e organizzazione del lavoro
 - Dependencies
 - Database
- 04** • FrontEnd
- 05** • Deployment



01-Scopo del documento

Il presente documento riporta le informazioni necessarie per lo sviluppo dell'applicazione web DropZero, sistema di monitoraggio idrico intelligente per la municipalità di Trento.

Sono descritti le Web API, l'organizzazione del codice, il modello dati MongoDB, le principali attività di testing e le scelte implementative adottate per frontend e deployment.



02-Web API

Le Web API di DropZero espongono, tramite architettura RESTful, le funzionalità principali della piattaforma per utenti privati e amministratori comunali, sotto il prefisso

```
http://localhost:5001/api
```

Le rotte sensibili richiedono autenticazione tramite JSON Web Token (JWT), e alcune sono ulteriormente limitate a utenti con ruolo amministratore.

Autenticazione - Registrazione utente

Crea un nuovo account nel sistema.

URL: **/auth/register**

Metodo: **POST**

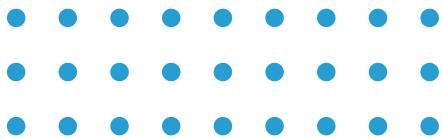
Accesso: Pubblico (non richiede token, perché serve per creare il primo account).

Body di esempio:

```
{
  "email": "user@example.com",
  "password": "password123",
  "firstName": "Mario",
  "lastName": "Rossi",
  "codiceFiscale": "RSSMRA80A01H501Z",
  "userType": "PRIVATE"
}
```

Risposta (201): Restituisce i dati dell'utente creato (senza password) e il token JWT da usare per le chiamate successive.

In caso di email o codice fiscale già presenti, il backend restituisce un errore significativo (es. 400/409) che il frontend mostra come messaggio di "utente già registrato".



Autenticazione - Login standard

Autenticazione tramite email e password.

URL: **/auth/login**

Metodo: **POST**

Accesso: Pubblico.

Body di esempio:

```
{  
  "email": "user@example.com",  
  "password": "password123"  
}
```

Risposta (200): Dati utente (incluso il ruolo: PRIVATE o ADMIN) e token JWT per accedere alle rotte protette.

Se email o password non sono corrette, il sistema risponde con 401 Unauthorized e un messaggio che il frontend visualizza nella schermata di login.

Autenticazione - Google login

Autenticazione sicura tramite Google OAuth 2.0.

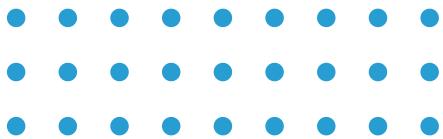
URL: **/auth/google**

Metodo: **POST**

Accesso: Pubblico.

Body di esempio:

```
{  
  "token": "GOOGLE_ID_TOKEN"  
}
```



Risposta (200): Dati utente e token JWT. Se l'utente non esiste ancora nel database, viene creato automaticamente a partire dalle informazioni ricevute da Google (email, nome, cognome).

In caso di token non valido o scaduto, viene restituito 401 Unauthorized e il frontend invita l'utente a ripetere il login con Google.

Readings - Statistiche dashboard

Dati per i widget della home (consumo settimanale, trend, spesa stimata, suggerimenti).

URL: **/readings/dashboard/:userId**

Modo: **GET**

Accesso: Privato (Richiede JWT).

Risposta (200) di esempio:

```
{  
  "hasData": true,  
  "currentConsumption": 850,  
  "trendPercentage": "-5.2",  
  "estimatedCost": 45.30,  
  "costStatus": "In calo",  
  "suggestions": [...]  
}
```

Se hasData è false, la dashboard mostra messaggi introduttivi e invita l'utente a inserire le prime letture.

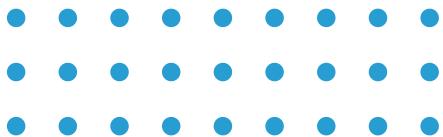
Readings - Storico lettura

Lista paginata di tutte le letture settimanali dell'utente.

URL: **/readings/history/:userId?page=1&limit=10**

Modo: **GET**

Accesso: Privato.



Risposta (200): Array di letture con logica di paginazione (es. campi items, page, totalPages), usato per popolare la tabella “Storico consumi” nella UI.
I filtri lato frontend (per data, costo, volume) sfruttano questi dati per permettere all’utente analisi più dettagliate.

Readings - Dati grafico

Dati ottimizzati per i grafici temporali.

URL: **/readings/chart/:userId?timeframe=90days**

Metodo: **GET**

Accesso: Privato.

Parametri Query: timeframe (valori supportati: 90days, 1year).

La risposta restituisce serie temporali già aggregate che il frontend passa direttamente a Recharts per disegnare line chart e bar chart, evitando calcoli pesanti lato client.

Readings - Aggiungi lettura manuale

Permette all’utente di caricare una lettura dal contatore fisico, utile se non è presente la telelettura automatica.

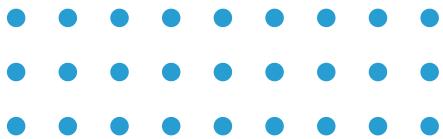
URL: **/readings**

Metodo: **POST**

Accesso: Privato.

Body di esempio:

```
{  
  "userId": "ID_UTENTE",  
  "readingValue": 125.5,  
  "date": "2024-01-31T00:00:00Z"  
}
```



Il backend valida che la lettura sia coerente con la precedente e, se tutto è corretto, aggiorna weekly_readings e l'eventuale storico dei consumi.

In caso di dato incoerente (lettura troppo bassa, formato data errato, campo mancante) viene restituito un errore 400 con messaggi utili per la correzione.

Amministrazione - Statistiche territoriali

Overview aggregata per il comune.

URL: **/admin/stats**

Metodo: **GET**

Accesso: Privato (Solo Admin).

Risposta (200): include almeno totale contatori attivi, consumo totale settimanale e numero di anomalie rilevate, informazioni usate per i KPI della dashboard admin.

Amministrazione - Dati mappa

Stato dei consumi per ogni circoscrizione di Trento.

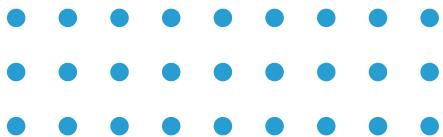
URL: **/admin/map**

Metodo: **GET**

Accesso: Privato (Solo Admin).

La risposta fornisce, per ogni zona, metriche aggregate (es. consumo medio, intensità heatmap) che il frontend mappa su una SVG delle 12 circoscrizioni, colorando l'area in base al livello di criticità.

Amministrazione - Segnalazione anomalie



Lista delle letture anomale rilevate nel territorio.

URL: **/admin/alerts**

Metodo: **GET**

Accesso: Privato (Solo Admin).

Risposta: dati anonimizzati (Zona, Volume, Gravità, Data) che alimentano la tabella "Allerte" della dashboard admin e permettono agli operatori di pianificare verifiche e interventi.

User - Ottieni profilo

Dati personali dell'utente autenticato.

URL: **/users/profile/:userId**

Metodo: **GET**

Accesso: Privato.

La risposta include dati anagrafici, impostazioni di lingua, tema e preferenze di notifica, usati dal frontend per costruire l'area "Profilo".

User - Aggiorna profilo

Modifica dati personali o password.

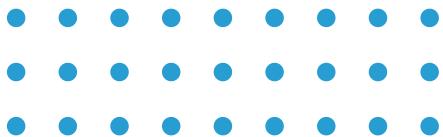
URL: **/users/profile/:userId**

Metodo: **PUT**

Accesso: Privato.

Body: oggetto JSON con i soli campi da aggiornare (pattern "partial update" lato API, anche se implementato via PUT).

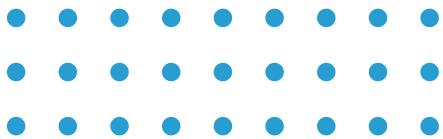
Il backend controlla che vengano rispettati i vincoli di validazione (es. formato email, lunghezza minima password) e che l'utente non possa modificare campi sensibili di altri account.



Gestione errori

Il backend utilizza codici HTTP standard per segnalare l'esito delle richieste e consente al frontend di mostrare messaggi chiari all'utente.

Codice errore	Descrizione
401 Unauthorized	Token JWT mancante, scaduto o non valido; il frontend reindirizza normalmente alla schermata di login
403 Forbidden	L'utente è autenticato ma non ha i permessi necessari (per esempio un utente privato che prova ad accedere a una rotta admin)
404 Not Found	La risorsa richiesta non esiste o non è accessibile nel contesto dell'utente (es. userId inesistente)
500 Server Error	Errore interno imprevisto; un middleware di gestione errori centralizzato logga il dettaglio lato server e restituisce al client una risposta generica per non esporre dettagli sensibili



03-Implementation

DropZero è realizzato con uno stack MERN (MongoDB, Express, React, Node.js) con frontend disaccoppiato che comunica con il backend tramite REST API.

L'architettura supporta due flussi principali: la dashboard utente per il monitoraggio dei consumi e la dashboard amministrativa per l'analisi aggregata e l'individuazione di sprechi idrici sul territorio

03.1-Repository organization

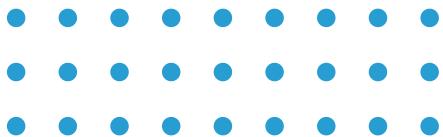
Il codice è pubblicato sul repository GitHub:

<https://github.com/GabrieleChini/DropZero.git>

organizzato in cartelle distinte per backend, frontend e documentazione.

Struttura concettuale:

Posizione	Contenuto
/backend (applicazione Node.js/Express)	Modelli Mongoose per le collezioni MongoDB
	Controller e router per i vari moduli (auth, readings, users, admin)
	Middleware di autenticazione JWT e gestione errori
	Script di seeding dei dati



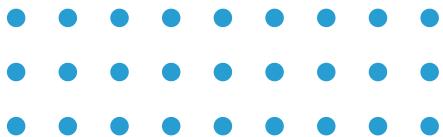
Posizione	Contenuto
/frontend (applicazione React con Vite)	Componenti condivisi (layout, navbar, card, grafici)
	Pagine per dashboard utente e dashboard admin
	Configurazione di React Router e integrazione con TailwindCSS
/docs (documentazione)	Documentazione del progetto (D1, D2, D3, D4 e documenti vari)

Questa separazione favorisce indipendenza tra livelli, semplificando manutenzione e possibili deploy separati di frontend e backend.

03.2-Branching strategy

Nel repository sono presenti i branch **main**, **Develop**, **Documentation** e **Release**, che riflettono una strategia ispirata al GitFlow Workflow.

Branch	Contenuto
main	Contiene le versioni stabili pronte per essere usate in demo o rilasciate

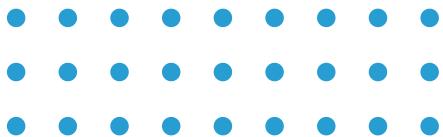


Develop	Viene usato come branch di integrazione sul quale confluiscono le nuove funzionalità una volta sviluppate e testate
Documentation	Raccoglie la documentazione del progetto, separando artefatti testuali dal codice eseguibile
Release	Impiegato per preparare rilasci intermedi, consentendo eventuali correzioni prima del merge definitivo su main

Schema dei branch



NOTA: In questo modello, i nuovi sviluppi dovrebbero nascere da Develop su branch di feature, essere testati e poi reintegrati



Gestione della documentazione

La documentazione progettuale (in particolare le parti grafiche del D1-D4 e le schermate illustrate dell'applicazione) è stata realizzata in modo collaborativo tramite **Canva**, strumento che ha permesso al gruppo di lavorare in condivisione su layout, diagrammi e componenti visuali avanzati.

Poiché il flusso di lavoro con Canva produceva principalmente output grafici e impaginati completi, questi materiali venivano caricati nel branch Documentation solo a valle delle revisioni più importanti, anziché con commit molto frequenti, privilegiando la coerenza del documento finale rispetto alla granularità della storia dei cambiamenti.

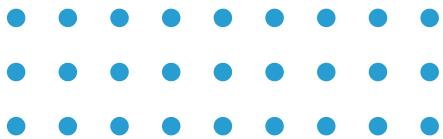
Disclaimer sull'uso effettivo di Git

Nel corso del progetto, l'utilizzo di Git non ha seguito in modo pienamente sistematico il modello di branching sopra descritto. Numerose modifiche al codice sono state sviluppate in locale e committate solo in fasi avanzate, riducendo la tracciabilità fine delle evoluzioni e i benefici tipici dell'integrazione frequente.

Inoltre, una prima versione del progetto era ospitata su un repository Git diverso, che nel tempo è diventato molto disordinato (branch non standardizzati, naming incoerente, commit poco significativi). Per ripristinare una struttura più chiara e presentabile, il gruppo ha deciso di migrare verso l'attuale repository, trasferendo il codice in uno stato ordinato ma sacrificando parte della cronologia dettagliata.

Questa scelta ha consentito di avere un repository più leggibile agli scopi didattici, pur essendo consapevoli che, in un contesto produttivo, sarebbe preferibile preservare l'intera storia e adottare in modo più rigoroso una strategia di branching basata su feature branch, commit frequenti e pipeline di integrazione continua.

03.3-Dependencies



Backend

Il backend di DropZero è realizzato in Node.js ed Express e fa uso di una serie di librerie esterne per gestire autenticazione, persistenza e configurazione.

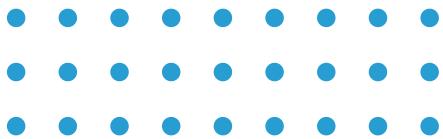
- **express**: framework web utilizzato per definire le rotte REST, i middleware di autenticazione e la gestione centralizzata degli errori.
- **mongoose**: ODM per MongoDB, impiegato per definire gli schemi delle collezioni (users, meters, weekly_readings, bills, ecc.), gli indici e le relazioni logiche tra i dati.
- **jsonwebtoken**: utilizzato per generare e validare i JSON Web Token che proteggono le rotte private del backend.
- **google-auth-library**: si occupa della verifica dei Google ID token nella rotta /auth/google, integrando in modo sicuro l'autenticazione esterna.
- **bcrypt.js**: gestisce l'hashing delle password per gli account con credenziali locali, garantendo che le password non siano mai memorizzate in chiaro nel database.
- **dotenv**: consente di esternalizzare tutti i parametri sensibili (porta, URI MongoDB, segreto JWT, client id Google) in un file .env non versionato, semplificando il passaggio tra ambienti diversi.
- **faker.js**: utilizzato negli script di seeding per generare utenti, contatori e letture di consumo con valori realistici, utili per test e demo dell'applicazione.

Questa scelta di dipendenze permette di costruire rapidamente un backend sicuro e flessibile, mantenendo al tempo stesso una buona leggibilità del codice e una chiara separazione delle responsabilità.

Frontend

Il frontend è sviluppato usando React e Vite, integrando librerie pensate per routing, stile e grafici.

- **react, react-dom**: costituiscono il core della UI, gestendo componenti, stato locale e ciclo di vita.
- **vite**: fornisce un ambiente di sviluppo veloce con hot reload e una build ottimizzata per la produzione, riducendo i tempi di compilazione.



- **react-router-dom**: gestisce la navigazione tra le varie pagine (login, dashboard utente, dashboard admin, profilo), incluse rotte protette che richiedono autenticazione.
- **tailwindcss**: permette di definire lo stile tramite utility class, realizzando interfacce responsive con estetica glassmorphism e coerenza visiva tra le diverse viste.
- **lucide-react**: fornisce un set di icone vettoriali utilizzate per rendere più leggibili card, pulsanti, avvisi e indicatori.
- **@react-oauth/google**: facilita l'integrazione del bottone ufficiale di login con Google, incapsulando la logica di autenticazione lato client.
- **recharts**: consente di costruire grafici a linee e a barre per rappresentare trend di consumo, spese e confronti temporali, con gestione semplificata dei tooltip e delle legende.

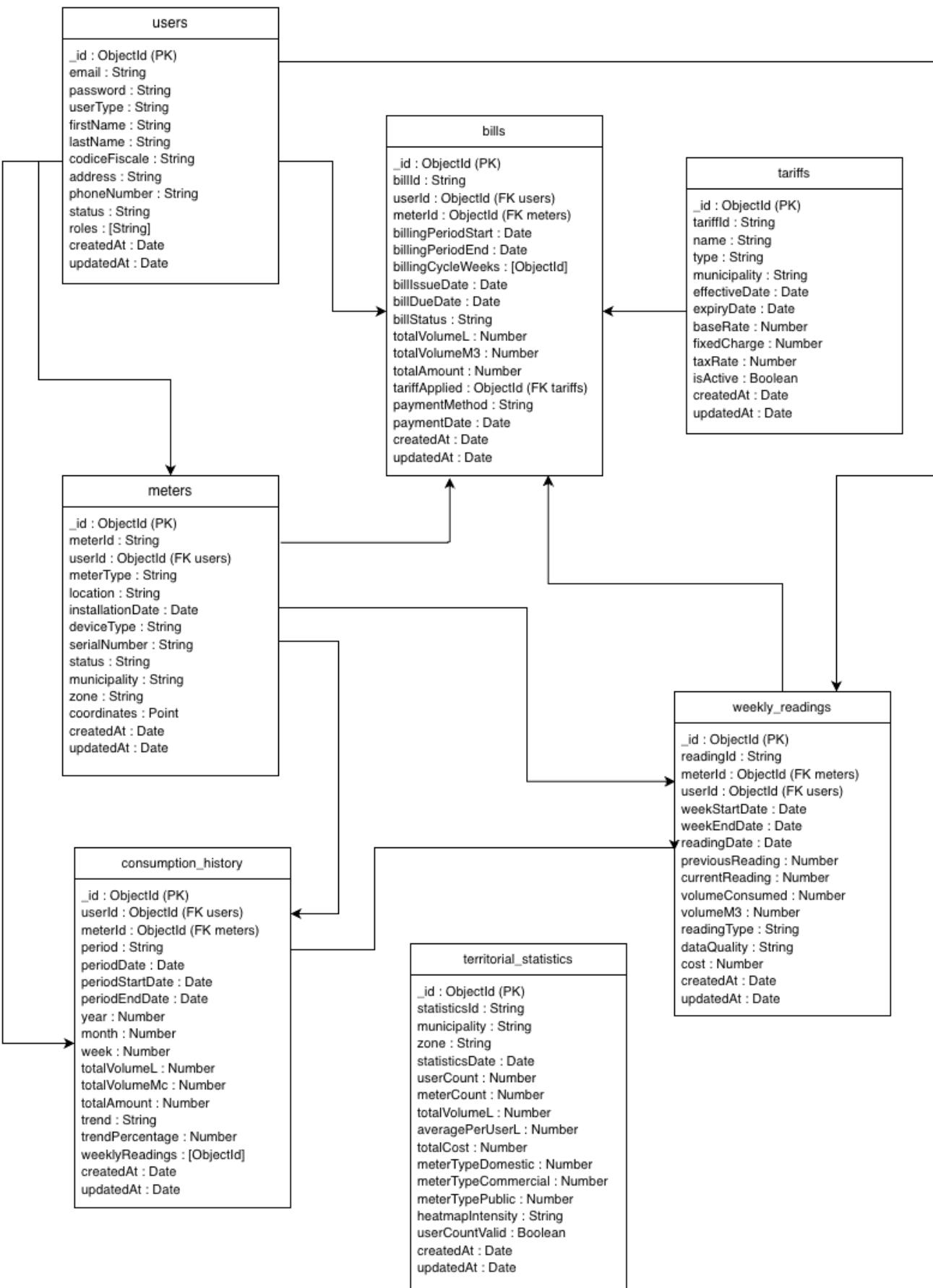
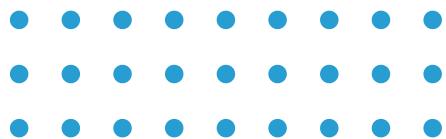
L'insieme di queste dipendenze rende possibile una UX reattiva e centrata sull'utente, mantenendo il codice modulare e facilmente estendibile.

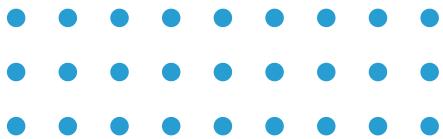
03.4-Database

Il database di DropZero è realizzato in MongoDB (versione minima 5.0) e organizzato in sette collezioni principali, progettate per supportare tracciamento dei consumi, generazione bollette e analisi territoriali anonime.

Le collezioni sono legate tra loro tramite identificativi (userId, meterId, readingId) e arricchite con indici che ottimizzano le query più frequenti (ricerca per utente, periodo temporale, zona geografica).

Schema database



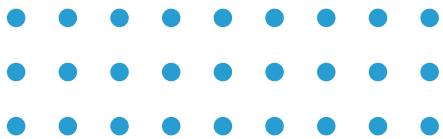


Descrizione elementi principali

Le collezioni principali sono:

- **users**: memorizza gli utenti del sistema (privati e amministratori) con email univoca, password hashata, tipo utente, dati anagrafici, indirizzo, stato, ruoli e preferenze (lingua, tema, canali di notifica).
- **meters**: rappresenta i contatori idrici associati agli utenti, con informazioni su ubicazione, tipo di utenza (domestica, commerciale, pubblica), metodo di comunicazione e coordinate geo-referenziate.
- **weekly_readings**: registra le letture settimanali dei contatori, includendo lettura precedente e corrente, volume consumato, qualità del dato, costo stimato e riferimenti a utente e contatore.
- **consumption_history**: conserva lo storico aggregato per periodi (settimanali, mensili, annuali) con dettagli su volumi totali, suddivisione per fasce tariffarie, breakdown dei costi e statistiche derivate (trend, min/max, anomalie).
- **tariffs**: definisce le tariffe idriche (domestiche, commerciali, pubbliche) con canone fisso, fasce di consumo, aliquote IVA e aggiustamenti stagionali, usate per il calcolo di costi e simulazioni.
- **bills**: memorizza le bollette generate a partire dalle letture settimanali, includendo periodo di fatturazione, consumi aggregati, costi dettagliati, stato della fattura e informazioni di pagamento.
- **territorial_statistics**: gestisce dati aggregati anonimizzati per comune e zona (numero utenti, consumi totali, consumi medi, breakdown per tipo di contatore, livello di intensità della heatmap), alimentando la dashboard amministrativa.

Le "relazioni" logiche tra collezioni (ad esempio users 1-N meters, meters 1-N weekly_readings, weekly_readings → bills e consumption_history) permettono di ricostruire la storia dei consumi dall'utente fino alle bollette e alle statistiche territoriali.



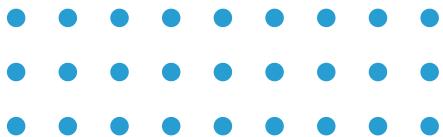
03.5-Testing

Le principali funzionalità del sistema sono state verificate tramite una combinazione di test manuali sull'interfaccia web e chiamate dirette alle API, con lo scopo di validare autenticazione, flusso delle letture, generazione delle bollette e comportamento della dashboard amministrativa.

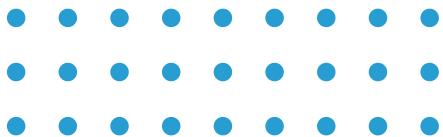
La presenza di script di seeding con dati realistici ha permesso di simulare scenari con utenti domestici reali, contatori attivi e periodi di fatturazione multipli, rendendo i test più vicini a casi d'uso concreti.

Esempi di casi di test documentati:

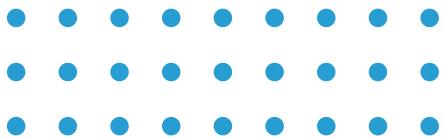
N.	Descrizione	Dati	Precondizioni	Dipendenze	Risultato
1	Login con credenziali locali corrette	email valida, password corretta	Utente registrato nel sistema, account ACTIVE	Endpoint POST /auth/login , collezione users	L'utente effettua il login con successo, riceve un token JWT valido e viene reindirizzato alla dashboard utente.



N.	Descrizione	Dati	Precondizioni	Dipendenze	Risultato
1.1	Login con credenziali locali errate	email valida, password errata	Utente registrato nel sistema	Endpoint POST /auth/login	Il sistema restituisce errore 401 Unauthorized con messaggio "Email o password non corretta", nessun token generato.
2	Login tramite Google con token valido	token Google OAuth 2.0 valido	Applicazione configurata con client ID Google	Endpoint POST /auth/google, collezione users	Verifica corretta del token, eventuale creazione dell'utente se non esiste, restituzione di dati utente e JWT; accesso alla dashboard appropriata (PRIVATE o ADMIN).



N.	Descrizione	Dati	Precondizioni	Dipendenze	Risultato
3	Visualizzazione mappa admin con statistiche territoriali	Dati presenti in territori al_statistics per più zone	Utente admin autenticato	Endpoint GET /admin/stats, GET /admin/map, collezione territorial_statistics	La dashboard admin mostra numero contatori, consumo totale e mappa SVG di Trento con zone colorate in base alla heatmap Intensity.
4	Visualizzazione elenco anomalie territoriali	Record di anomalie presenti	Utente admin autenticato	Endpoint GET /admin/alerts, weekly_readings, territorial_statistics	La tabella mostra elenco di alert anonimizzati, coerenti con i dati di consumo settimanale; nessun dato personale dell'utente viene esposto.



04-Frontend

Il frontend di DropZero è una single-page application sviluppata in React e costruita con Vite, progettata per offrire un'esperienza utente moderna e reattiva sia ai cittadini sia agli amministratori comunali.

L'interfaccia è focalizzata sulla visualizzazione chiara dei consumi idrici tramite grafici interattivi, card informative e viste aggregate, nel rispetto dei vincoli di privacy definiti nel progetto.

Architettura e tecnologie UI

L'applicazione utilizza React Router Dom per definire rotte pubbliche (login, registrazione) e rotte protette, accessibili solo a utenti autenticati e con ruolo corretto (PRIVATE o ADMIN).

Un componente Layout centrale gestisce header, sidebar e contenuto principale, adattando automaticamente le voci di menu e le sezioni disponibili in base al ruolo dell'utente, in modo da separare chiaramente l'esperienza "utente privato" da quella "amministratore".

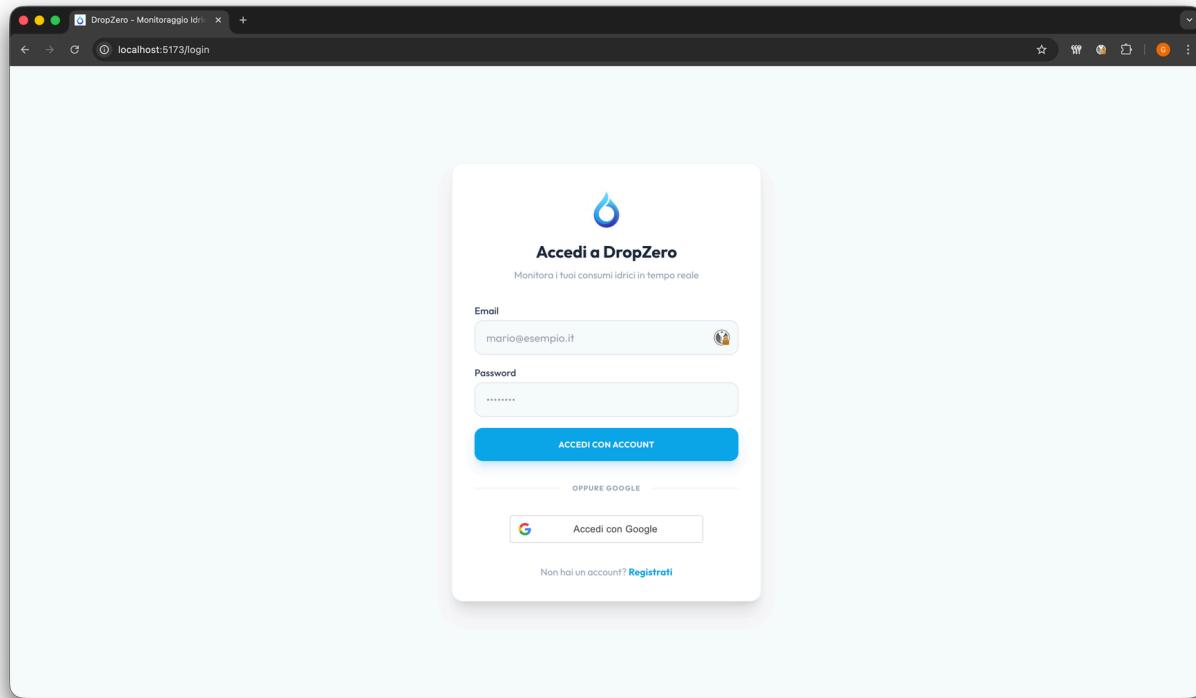
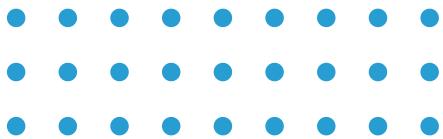
Lo stile è implementato tramite TailwindCSS con estetica di tipo glassmorphism (pannelli semi-trasparenti, gradienti e micro-animazioni), mentre Lucide React viene usato per l'iconografia (indicatori di stato, pulsanti di azione, card informative).

Per la rappresentazione dei dati numerici e temporali vengono utilizzati grafici Recharts (line chart e bar chart), che permettono interazioni come hover, tooltip, selezione intervalli e confronti tra periodi.

Schermata di login e autenticazione

La schermata di login offre sia l'autenticazione tradizionale con email e password sia il login federato tramite Google, in linea con il requisito RF1 del D1.

Il client invia le credenziali agli endpoint **/auth/login** o **/auth/google** e, in caso di successo, memorizza il token JWT e il profilo utente, reindirizzando alla dashboard appropriata.

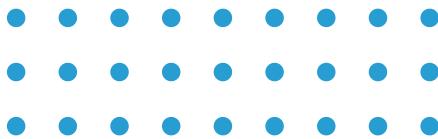


Eventuali errori (credenziali errate, token non valido) vengono mostrati tramite messaggi chiari nella UI, permettendo all'utente di riprovare o, se necessario, passare alla registrazione.

Dashboard utente - monitoraggio consumi

Dopo il login, l'utente privato accede a una dashboard che riassume la situazione corrente dei consumi idrici tramite card informative e grafici interattivi, soddisfacendo i requisiti RF3-RF5 (dashboard, storico, previsioni).

I dati mostrati (consumo settimanale, trend percentuale, spesa stimata, eventuali suggerimenti) sono ottenuti principalmente dalle API **/readings/dashboard/:userId** e **/readings/chart/:userId**, che restituiscono serie temporali già aggregate.



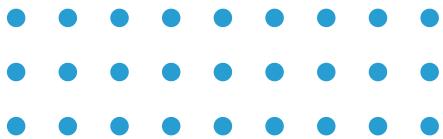
The screenshot shows the DropZero Premium Dashboard interface. On the left, there's a sidebar with the 'DropZero WATER MONITOR' logo, a 'MENU PRINCIPALE' section, and three main navigation items: 'Overview' (selected), 'Storico Consumi', and 'Profilo Utente'. Below these is a 'TUO RISPARMIO -10%' card with a water drop icon. At the bottom of the sidebar is a user profile for 'Gabriele' (Piano Privato). The main dashboard area has a dark header with the greeting 'Ciao, Gabriele' and a message about saving water during showers. It features several cards: one for weekly consumption (2736 L, Ottimo), one for estimated expenses (€ 34.90, In calo), and one for system efficiency (Ottimale, Nessuna perdita rilevata). A central chart titled 'Analisi Andamento' shows consumption over the last 90 days. To the right, there's an 'AI Insights' box with a 'Risparmio Doccia' suggestion. A 'Nuova Lettura →' button is in the top right corner.

La dashboard consente di cambiare intervallo temporale (ad esempio 7 giorni, 30 giorni, 1 anno), di analizzare i dettagli dei punti sul grafico tramite tooltip e di accedere direttamente alle sezioni di storico e suggerimenti.

Storico consumi e bollette

Una sezione dedicata consente all'utente di consultare lo storico dettagliato delle letture e delle bollette, con tabelle paginate e filtri per periodo, costo o volume consumato, in linea con RF4 del D1.

Le informazioni sono ottenute interrogando le API **/readings/history/:userId** e le collezioni `weekly_readings`, `bills` e `tariffs`, che forniscono per ogni riga data, consumo, costo e stato della fattura.



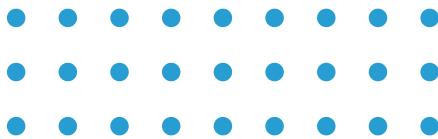
The screenshot shows the 'Storico Letture' (Historical Readings) page. The left sidebar includes a logo, navigation links for 'MENU PRINCIPALE' (Overview, Storico Consumi, Profilo Utente), and a 'TUO RISPARMIO' (Your Savings) section indicating a 10% savings. The main content area has a title 'Storico Letture' and a subtitle 'Consulta il dettaglio di tutte le tue letture settimanali.' A 'Esporta CSV' button is at the top right. Below is a table with columns: PERIODO, LETTURA (M³), CONSUMO, COSTO STIMATO, and STATO. The table lists weekly readings from February 2026 back to December 2025, all marked as 'Reale' (Real).

PERIODO	LETTURA (M ³)	CONSUMO	COSTO STIMATO	STATO
12 febbraio 2026 Settimana 0	1093.39	2736 L	€ 8.06	+ Reale
5 febbraio 2026 Settimana 1	1090.65	3546 L	€ 9.76	+ Reale
29 gennaio 2026 Settimana 2	1087.10	2569 L	€ 7.71	+ Reale
22 gennaio 2026 Settimana 3	1084.54	2879 L	€ 8.36	+ Reale
15 gennaio 2026 Settimana 4	1081.66	3105 L	€ 8.83	+ Reale
8 gennaio 2026 Settimana 5	1078.55	3084 L	€ 8.79	+ Reale
1 gennaio 2026 Settimana 6	1075.47	3088 L	€ 8.79	+ Reale
25 dicembre 2025 Settimana 7	1072.38	2855 L	€ 8.31	+ Reale
18 dicembre 2025				

Questa interfaccia rende trasparente la correlazione tra i dati tecnici salvati in MongoDB e le informazioni che l'utente vede a schermo, permettendo verifiche puntuali e, in prospettiva, esportazioni in formati come CSV o PDF.

Dashboard amministrativa - mappa territoriale

Gli amministratori comunali accedono a una dashboard distinta, che offre una vista di alto livello sui consumi aggregati per circoscrizione del comune di Trento. La componente principale è una mappa SVG delle 12 circoscrizioni, colorata secondo un codice di intensità (ad esempio da blu a rosso) che riflette l'indicatore heatmapIntensity presente nella collezione territorial_statistics.



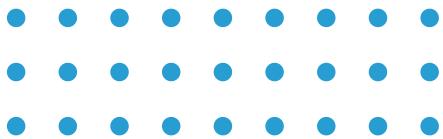
Cliccando su una zona, l'interfaccia mostra un popup o pannello laterale con valori come consumo medio, numero di utenti, numero di contatori e trend di periodo, ottenuti tramite le API /admin/stats e /admin/map.

Tutti i dati usati sono aggregati e anonimizzati, in modo da rispettare i vincoli di privacy delineati nel D1 (niente statistiche su insiemi troppo piccoli di utenti).

Dashboard amministrativa - gestione alert anomalie

Accanto alla mappa è presente una tabella di alert che elenca le anomalie individuate dagli algoritmi di analisi, con informazioni su zona, volume anomalo, livello di gravità e data rilevazione.

Questa tabella è alimentata dall'endpoint /admin/alerts, che ritorna esclusivamente dati anonimizzati e aggregati per area, senza riferimenti diretti ai singoli utenti o contatori.



Screenshot of the DropZero Water Monitor application interface, showing the 'Vista Comune' (Common View) section.

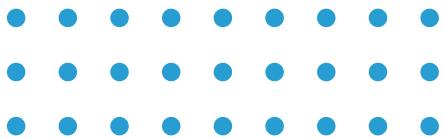
The interface includes:

- Left Sidebar (MENU PRINCIPALE):**
 - Overview
 - Storico Consumi
 - Profilo Utente
 - Vista Comune** (selected)
- Map View:** A schematic map of administrative districts (circoscrizioni) in the area. Districts are color-coded: Bondone (orange), Sardagna (green), Centro (light green), San (light green), Oltreferina (light green), Argentario (green), Povo (green), Meano (green), and Ravina-Romagnano (green). Two specific locations are highlighted with red dots: Bondone and Villazzano.
- Recent Reports:** A section titled "Segnalazioni Recenti" (Recent Reports) showing anomalies in consumption. It lists two entries:

STATO	CIRCOSCRIZIONE	DATA RILEVAZIONE	CONSUMO ANOMALO (M ³)
ATTENZIONE	Bondone	12/02	8.64
ATTENZIONE	Villazzano	12/02	8.15

- User Information:** A box showing "TUO RISPARMIO +0%" and "Consumi in aumento questo mese".
- User Profile:** A box showing "Utente Piano Privato".

La vista consente all'amministratore di ordinare o filtrare gli eventi in base alla gravità o al periodo temporale, supportando le decisioni su eventuali interventi di manutenzione o campagne di sensibilizzazione nelle zone più critiche.



05-Deployment

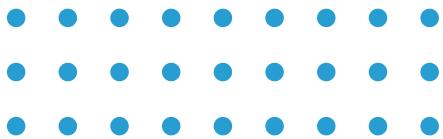
Il deployment di DropZero è stato pensato principalmente per l'ambiente di sviluppo e di demo del corso, con la possibilità di estendere in futuro la soluzione verso un ambiente di produzione più strutturato.

L'applicazione è composta da due componenti principali – backend Node.js/Express e frontend React/Vite – che comunicano tramite REST API, appoggiandosi a un database MongoDB.

Architettura di esecuzione

L'architettura di deployment prevede tre elementi principali:

Elemento	Descrizione
Backend	Applicazione Node.js ed Express, esposta sulla porta configurata nella variabile PORT (nostro caso 5000), con prefisso API <code>http://localhost:5000/api</code> .
	Si connette a un'istanza MongoDB (locale o Atlas) tramite l'URI specificato in <code>MONGODB_URI</code>
Database	Può essere eseguito localmente oppure in container Docker dedicato. Per avviare un'istanza MongoDB tramite Docker con il comando: <code>docker run -d --name mongodb -p 27017:27017 mongo:latest</code>
Frontend	Applicazione React eseguita tramite Vite in modalità sviluppo (tipicamente su <code>http://localhost:5173</code>), configurata per chiamare le API del backend.



Configurazione backend

Per avviare il backend in ambiente locale sono necessari i seguenti passi:

1. Posizionarsi nella directory backend del repository.
2. Installare le dipendenze con il comando **npm install**, che scarica le librerie descritte nel file **package.json** (Express, Mongoose, JsonWebToken, Google Auth Library, ecc.).
3. Creare un file **.env** contenente almeno le variabili:
 - **PORT=5000**
 - **MONGODB_URI=<stringa connessione MongoDB>**
 - **JWT_SECRET=<segreto per firma JWT>**
 - eventuali parametri per Google OAuth (client ID, ecc.).
- 4.(Opzionale) Eseguire lo script di seeding, ad esempio **node seed.js**, per popolare le collezioni del DB con dati realistici utili per test e demo.
5. Avviare il server di sviluppo con **npm run dev**; a questo punto le API risultano raggiungibili all'indirizzo **http://localhost:5000/api**.

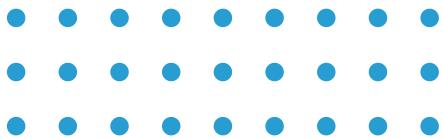
Questa configurazione permette a membri diversi del gruppo di avviare rapidamente un ambiente identico, grazie alla centralizzazione dei parametri in **.env**.

Configurazione frontend

Il frontend viene eseguito separatamente, mantenendo un'architettura a frontend disaccoppiato.

1. Posizionarsi nella directory **frontend** del repository.
2. Installare le dipendenze con **npm install**, recuperando React, Vite, React Router, TailwindCSS, Recharts e le altre librerie UI.
3. Verificare che la base URL delle API sia configurata correttamente nel codice (ad esempio in un file di configurazione o in un client HTTP dedicato), puntando a **http://localhost:5000/api** per l'ambiente di sviluppo.
4. Avviare il server di sviluppo con **npm run dev**; Vite espone l'applicazione su una porta locale (**http://localhost:5173**), indicata anche nel terminale.

In questa versione non è previsto un file **.env** nel frontend: l'URL del backend è impostato direttamente nel codice sorgente, scelta sufficiente per l'uso didattico del progetto ma facilmente estendibile in futuro introducendo variabili d'ambiente specifiche per ambiente.



Gestione ambienti e portabilità

La configurazione basata su file .env e sull'utilizzo di Node.js/MongoDB rende relativamente semplice lo spostamento tra ambienti diversi (sviluppo, test, demo). L'uso opzionale di Docker per l'istanza MongoDB permette di avere un database isolato e ripetibile, riducendo problemi di configurazione locale e facilitando la condivisione dell'ambiente tra i membri del gruppo.

Nel progetto attuale non è ancora stata impostata una pipeline CI/CD completa, ma la struttura del repository e la separazione tra backend, frontend e database sono già compatibili con un futuro deploy su piattaforme come Render/Vercel/Netlify o su macchine virtuali dedicate.

Account demo

Per supportare le attività di test e le dimostrazioni del progetto sono stati definiti alcuni account di esempio:

Tipo utente	Email	Password
Privato cittadino	mario.rossi@email.com	password123
Amministratore comunale	admin@dropzero.com	password123

Questi account consentono di mostrare rapidamente, durante la demo o la correzione del progetto, sia la prospettiva del cittadino (dashboard consumi, storico, suggerimenti) sia quella dell'ente gestore (mappa territoriale, statistiche e alert di anomalia).