

UNIVERSITÀ
DI TORINO

DotLine

Concezione e sviluppo di una web app per la messagistica istantanea con supporto alla crittografia end-to-end

Il ruolo del dato personale nella società moderna

UNIVERSITÀ DI TORINO



I dati personali al giorno d'oggi hanno assunto un ruolo centrale nella vita digitale, essi vengono ormai prodotti anche inconsapevolmente nel nostro quotidiano.



Il crescere della loro importanza ha accresciuto notevolmente il loro valore, anche pecuniario; di conseguenza molte aziende hanno reso i dati parte della loro revenue.



Questa ascesa ha reso i dati personali un prezioso bersaglio per le attività illecite, soprattutto online, ai danni sia dei soggetti che delle aziende che li trattano.

Il ruolo del dato personale nella società moderna

UNIVERSITÀ DI TORINO



La salvaguardia dei dati personali è ormai diventata una tematica di fondamentale importanza nell'era digitale e di conseguenza riguarda tutti, non soltanto utenti esperti o aziende.

Occorrono soluzioni semplici e accessibili che permettano di dare all'utente comune gli strumenti per essere in grado di salvaguardare i propri dati personali facilmente.



DotLine: una soluzione per la protezione dei dati

UNIVERSITÀ DI TORINO



DotLine è una soluzione semplice e accessibile al problema della salvaguardia delle proprie comunicazioni

dotline.enricosola.dev/presentazione

Pensata per le esigenze dell’utente medio

UNIVERSITÀ DI TORINO



Di semplice utilizzo: DotLine è un applicativo completo, facilmente accessibile e utilizzabile senza sconvolgere le abitudini degli utenti.



Adattabile: realizzata sottoforma di web-app non richiede l’installazione di altro software aggiuntivo e si adatta pressoché a qualsiasi dispositivo, anche mobile.



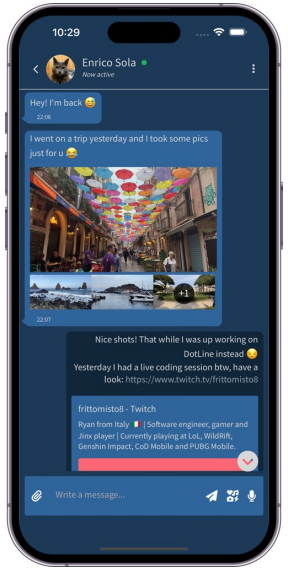
Incentrata sulla privacy: progettata per essere il meno invasiva possibile, il suo utilizzo non richiede l’immissione di alcun recapito tracciabile come e-mail o numero di telefono.

dotline.enricosola.dev/presentazione

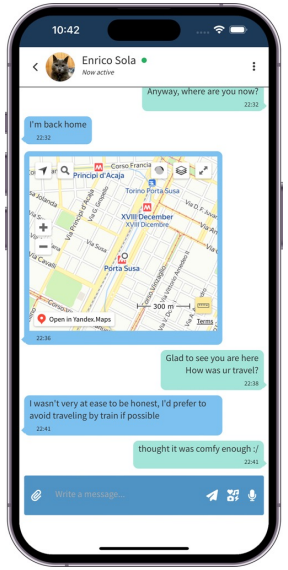
Concepita per dare sfogo alla propria espressività

UNIVERSITÀ DI TORINO

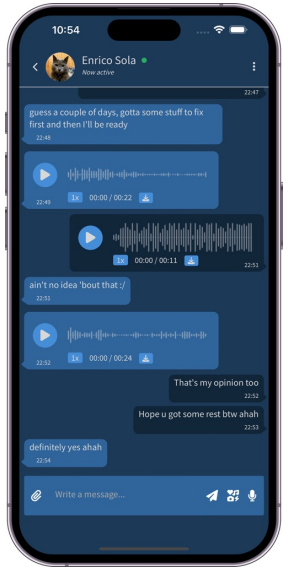
Supporto a un ampio spettro comunicativo con la possibilità di inviare contenuti di vario tipo, fra i quali:



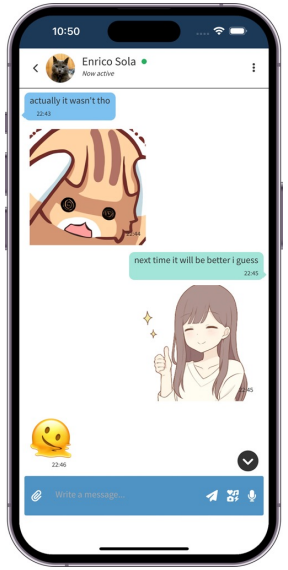
Messaggi di testo e allegati
(immagini, video, audio e file)



Posizioni geografiche



Messaggi vocali



Sticker ed Emoji
(statici o animati)

dotline.enricosola.dev/presentazione

Progettata con la privacy al primo posto

UNIVERSITÀ DI TORINO



Crittografia end-to-end: tutti i messaggi, gli allegati, e ogni altro contenuto è protetto dalla crittografia AES-256 ed è accessibile solamente ai membri della conversazione.



Sincronizzazione: indipendentemente dal dispositivo sul quale si accede a DotLine, ogni messaggio sarà presente e sincronizzato in tempo reale in ogni suo cambiamento.

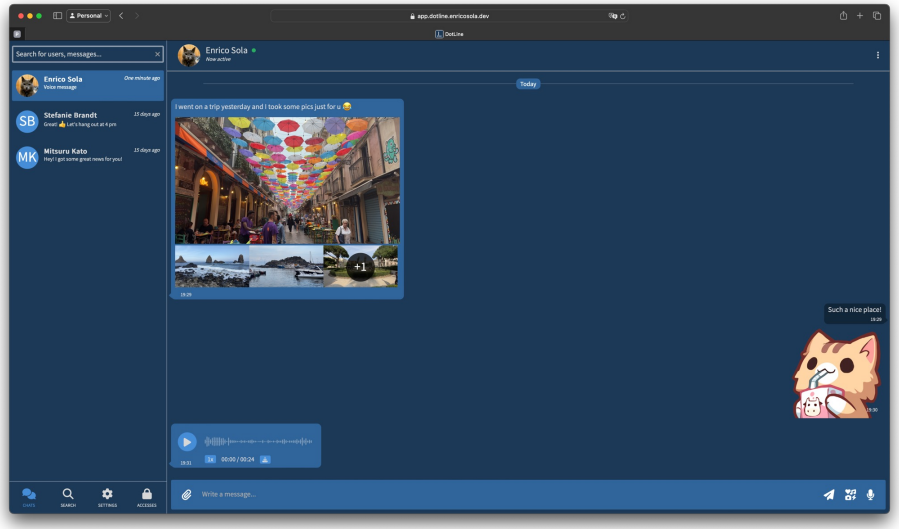


Controllo degli accessi: ogni accesso effettuato viene registrato di modo che l'utente sia in grado di tracciarne la provenienza e, nel caso, intervenire per chiudere la sessione.

dotline.enricosola.dev/presentazione

Architettura di DotLine

UNIVERSITÀ DI TORINO

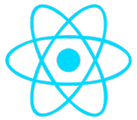




DotLine si compone di due parti: il **client**, ossia la web-app che permette di usufruire dei servizi offerti dalla piattaforma, e il **server**, cioè il cuore dell'applicativo, dove i dati sono archiviati e distribuiti agli utenti del software.

Architettura di DotLine: il client (front-end)

UNIVERSITÀ DI TORINO



L'interfaccia grafica dinamica e interattiva è stata implementata con l'ausilio di **React.js**



Il linguaggio **SASS** ha invece supportato la realizzazione del design dell'applicativo



Il software è stato scritto utilizzando il linguaggio di programmazione **JavaScript**



NPM ha permesso la gestione delle varia dipendenze richieste dal software



Gli assets realizzati sono ottimizzati attraverso l'uso di **WebPack**

Il client di DotLine è offerto sia in lingua Italiana che Inglese ed è inoltre disponibile in due temi: chiaro o scuro

Architettura di DotLine: il server (back-end)

UNIVERSITÀ DI TORINO



Il server di DotLine è scritto in **JavaScript** e la sua esecuzione è affidata al runtime **Node.js**

express

Express.js è il framework sopra il quale è stato sviluppata la piattaforma

DotLine è stato sviluppato seguendo l'approccio **RESTFull** e le linee guida proposte del paradigma **Model-View-Controller**. Alle **API state-less** offerte dal software si affianca la gestione real-time degli eventi attraverso il protocollo **WebSockets**.

Le singole unità software sono sviluppate seguendo il **paradigma a oggetti** in cui le classi si differenziano per le loro **responsabilità e ruolo**; sono organizzate in modo **gerarchico** nonché strutturate con gli opportuni **design pattern** al fine di ottimizzarne la **manutenibilità** e l'**estensione** futura delle funzionalità dell'applicativo.

Architettura di DotLine: persistenza dei dati



La persistenza dei dati è gestita
dal database NoSQL distribuito
Apache Cassandra



Le informazioni più richieste
sono ridondate mediante **Redis**
per una maggiore efficienza

I dati degli utenti sono archiviati su due fronti: localmente sul client, per un accesso rapido e diretto, e sul server, per mantenerne la persistenza e poterli sincronizzare attraverso i molteplici client.

Per ottimizzare la scalabilità e le operazioni di gestione DotLine si avvale di servizi in cloud basati su **AWS** sia per Cassandra che per Redis, rispettivamente **DataStax Astra** e **Redis Enterprise Cloud**.

Architettura di DotLine: deployment

UNIVERSITÀ DI TORINO



Sia il client che il server di DotLine sono distribuiti sottoforma di immagini **Docker**



Le richieste HTTP verso l'app vengono gestite da **NGINX** in modo da scalare i carichi



Il software è distribuito su server **VPS Linux** equipaggiati con **RedHat Enterprise Linux**

DotLine è distribuito attraverso il protocollo **HTTPS** seguendo elevati standard di sicurezza e crittografia viene inoltre fatto uso in via sperimentale del nuovo protocollo **HTTP/3** e della compressione **Brotli**

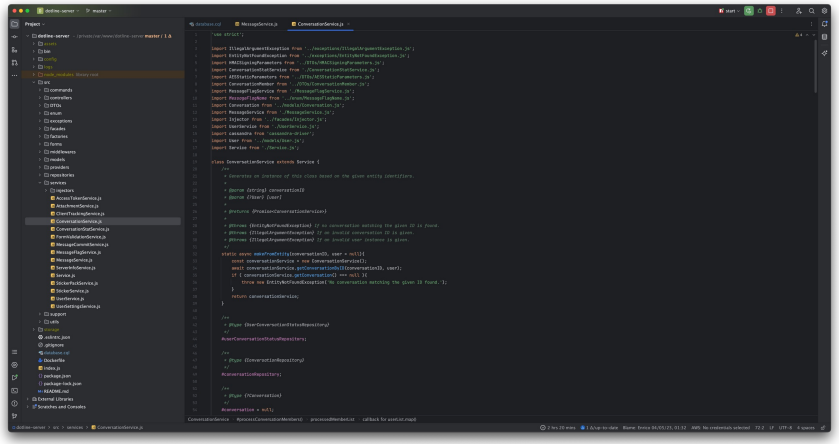


I server utilizzati da DotLine sono stati scelti in modo da ridurre l'impatto ambientale in quanto equipaggiati con processori **Ampere® Altra®** basati su architettura **ARM**

arm

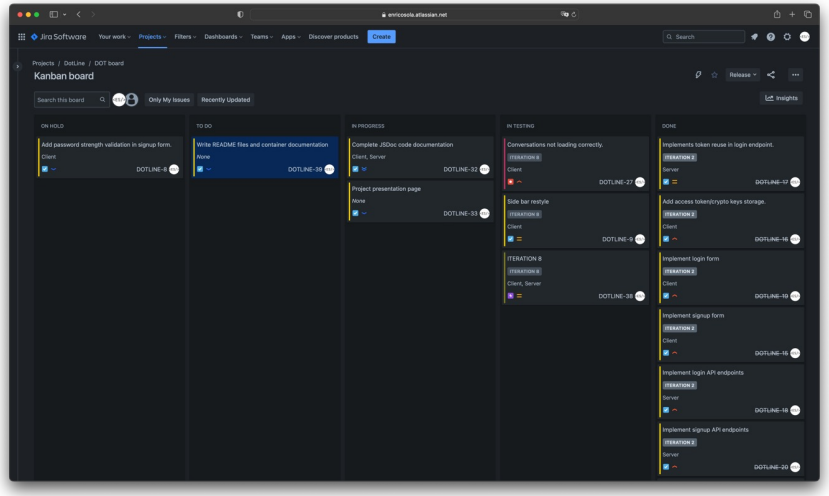
Sviluppo di DotLine

UNIVERSITÀ DI TORINO



DotLine è stato sviluppato su ambiente **macOS** (**ARM**) con il supporto dell'IDE **WebStorm**; il codice sorgente è documentato seguendo lo standard **JSDoc** in lingua inglese mentre la gestione delle risorse prodotte avviene tramite **git** con archiviazione cloud su **GitHub**.

La gestione dello sviluppo si basa sul **modello iterativo**: gli artefatti vengono dunque prodotti per raffinamenti successivi e le singole attività sono a loro volta suddivise in task organizzati attraverso il modello **Kanban** con il supporto del software **Atlassian Jira**.



Considerazioni finali

UNIVERSITÀ DI TORINO

Lo sviluppo di DotLine ha portato con sé sfide e difficoltà interessanti, fra le principali:

```
SMCS_2(C, U)
    # PRE: C è una data conversazione da sincronizzaree e U l'utente autenticato.
    # POST: tutta la lista di commit è stata scaricata e vi sono un SOL a simboleggiare
    # l'inizio della lista e un EOL a guardia della fine della stessa.

    D ← nil, L ← nil, F_SOL ← nil
    while L = nil or L not empty do
        # Carico un segmento di 250 commit dal server in ordine decrescente con data < D
        L ← fetchCommitList(C, U, D)
        if !isEmpty(L) then
            # Creo un checkpoint di tipo SOL usando il primo elemento di L.
            L_SOL ← createCheckpoint(SOL, L[0])
            # Creo un checkpoint di tipo EOL usando l'ultimo elemento di L.
            L_EOL ← createCheckpoint(EOL, L[-1])
            # Estraggo il primo checkpoint precedente alla data dell'EOL creato da L.
            checkpoint ← getFirstByDate(getDateOf(L_EOL))
            if checkpoint = nil or getTypeOf(checkpoint) ≠ EOL then
                # Il checkpoint precedente nell'indice è SOL o non esiste, proseguo.
                addCheckpoint(EOL, L_EOL)
                D ← getDateOf(L_EOL)
            else
                # Il checkpoint è un EOL quindi è stato intersecato un segmento, salta
                alla fine di esso.
                D ← getDateOf(checkpoint)
            if F_SOL = nil then
                # Questa è la prima iterazione, aggiungo il checkpoint SOL a simboleggare
                l'inizio del segmento.
                addCheckpoint(SOL, L_SOL)
                F_SOL ← L_SOL
            # Estraggo tutti i checkpoint successivi alla data dell'EOL creato da L.
            checkpointList ← getCheckpointListByRange(getDateOf(L_EOL))
            foreach CH in checkpointList do
                isCurrentEol ← getTypeOf(CH) = EOL and equals(CH, L_EOL)
                isFirstSOL ← F_SOL ≠ nil and equals(CH, F_SOL)
                if !isCurrentEol and !isFirstSOL then
                    # Il checkpoint CH è diverso dal primo SOL creato o dall'EOL aggiunto,
                    lo rimuovo.
                    removeCheckpoint(CH)
    return
```

La **sincronizzazione dei messaggi**, nel loro intero ciclo di vita, per la quale è stato concepito l'**algoritmo SMCS-2** che permette una **sincronizzazione incrementale** nonché **fault-tollerant** dei messaggi e dei relativi cambiamenti nel tempo

La **gestione del processo crittografico**, partendo dalle **chiavi RSA** degli utenti fino alla distribuzione delle **chiavi AES** ai membri delle conversazioni passando per la verifica dell'integrità dei messaggi mediante **HMAC** e l'hashing **SHA-512**

L'archiviazione locale dei dati al fine di fornire un accesso rapido e implementare alcune funzionalità che, data la natura crittografica del software, non sarebbe possibile offrire delegandole interamente al server

Considerazioni finali

UNIVERSITÀ DI TORINO

Il traguardo finale apre la strada a futuri sviluppi per la piattaforma, ragion per cui l'intero progetto è stato reso **open-source** e quindi pubblicamente disponibile su **GitHub** e su **Docker Hub** nonché liberamente accessibile online all'indirizzo https://dotline.enricosola.dev





UNIVERSITÀ
DI TORINO

Grazie