

Relatore

DedicaDedica

SommarioSommario
*Sommario

Ringraziamenti

*Ringraziamenti

,

tableofcontents

lof

lot

Lista degli elementi di codice
lof

Contesto aziendale

Il presente capitolo introduce l'azienda presso cui ho svolto l'attività di *stage*, **SyncLab S.r.l.**, fornendo la base necessaria per comprendere il suo contesto aziendale. Presentazione azienda L'azienda ospitante, *SyncLab S.r.l.*, è stata fondata nel 2002 a Napoli e si è affermata fin da subito come una delle aziende leader nel settore della soluzioni software innovative. Inizialmente nata come *software house*, l'azienda si è dedicata allo sviluppo di soluzioni *software* innovative, progettate per soddisfare le esigenze di diversi settori.

Nel tempo, *SyncLab S.r.l.* ha ampliato il proprio raggio d'azione, assumendo un ruolo rilevante come *system integrator*. La duplice identità coniuga la creatività e la proattività di una *software house* con l'approccio orientato all'efficienza e alla qualità tipico di un'azienda di servizi. *SyncLab S.r.l.* promuove attivamente la collaborazione interna, incoraggiando l'interazione non solo tra i membri della stessa azienda, ma anche con i clienti e i partner.

Rami aziendali e progetti *SyncLab S.r.l.* collabora con un ampio numero di clienti appartenenti a molteplici settori industriali. Alcuni dei *software* che l'azienda ha prodotto sono:

Sobereye (ambito *web*): un'applicazione innovativa progettata per monitorare il rischio di deterioramento neuro-cognitivo.

SynClinic (ambito sanitario): un sistema integrato che supporta la gestione completa dei processi clinici e amministrativi.

DPS 4.0 (ambito *web* e *privacy*): una piattaforma *web* che supporta i titolari, responsabili, *data protection officer* (DPO).

Fast Ride (ambito trasporti): una soluzione per la gestione di servizi di trasporto pubblico a chiamata, in contesti urbani e extraurbani.

Way of working In questa sezione tratterò di alcune tecnologie di cui ho avuto esperienza diretta per lo sviluppo dei progetti.

Dart: linguaggio di programmazione orientato agli oggetti sviluppato da Google, noto per la sua versatilità. Offre numerosi strumenti per lo sviluppo di un applicativo *cross-platform* che consente di creare applicazioni native per diversi sistemi da un'unica base di codice. La compilazione *Ahead-Of-Time (AOT)* che accelerano lo sviluppo.

Flutter: un *framework* basato su Dart per lo sviluppo di applicazioni multi-piattaforma, che offre vantaggi come:

Hot-reload: permette di visualizzare immediatamente le modifiche al codice durante lo sviluppo senza dover aspettare la compilazione.

Compatibilità con material: mette a disposizione un ricco arsenale di *widget* per creare interfacce utente moderne e intuitive.

Firebase: un *database* sviluppato da Google e ben integrato con Flutter che offre una serie di funzionalità di *backend* per applicazioni mobile.

Authentication: servizio che offre i servizi di *backend* e librerie pronte all'uso per autenticare gli utenti nell'applicazione.

Firestore: *database NoSQL* orientato ai documenti, che permette di archiviare i documenti in raccolte, le quali fungono da collezioni.

Per garantire continuità operativa e una efficace gestione dei processi collaborativi, è necessario disporre di strumenti di supporto.

Git: un sistema di controllo versione distribuito che permette di tracciare in modo efficiente le modifiche ai file e di coordinare le attività di sviluppo.

Android Studio: è un ambiente di sviluppo integrato (*IDE*) gratuito, progettato per lo sviluppo di applicazioni **Android**.

Xcode: è un ambiente di sviluppo integrato (*IDE*), sviluppato e mantenuto da Apple che contiene una *suite* di strumenti per lo sviluppo di applicazioni **iOS**.

UMLet: è uno strumento gratuito e *open source* che permette di creare diagrammi UML¹. In particolare diagrammi di:

Smart working e strumenti di comunicazione L'azienda adotta un modello di lavoro prevalentemente da remoto: la
Nel mio caso, trattandosi di un progetto di *scouting* tecnologico, gli incontri con il responsabile designato sono stati

Per garantire una gestione efficace delle attività e tracciare correttamente l'evoluzione del progetto, ho utilizzato diversi

Google Calendar: un calendario condiviso che permette la creazione e la modifica di eventi, specificandone durata e lu-

GitHub Projects: una sezione di GitHub dedicata alla gestione dei *ticket*, preferita in alternativa a **Trello** normalmen-

Per la comunicazione interna durante il lavoro da remoto:

Discord: utilizzato dai dipendenti per lo scambio di informazioni tramite *chat* testuali e vocali. La piattaforma funge a

[!ht] [width=0.8]..../assets/sviluppoComunicazione.png Strumenti di comunicazione

Spirito di innovazione aziendale

L'azienda *SyncLab* si caratterizza per una forte propensione all'innovazione, elemento che guida in modo significativo

Miglioramento delle dinamiche interne: l'azienda investe costantemente nell'ottimizzazione dei processi di gestione

Proposizione di soluzioni d'avanguardia: L'impegno innovativo non si limita all'organizzazione interna, ma si estende

In questo contesto, orientato alla ricerca dell'avanguardia si è collocata la mia esperienza di *stage*. La vocazione aperta a

[!ht] [width=0.75]..../assets/progettoStage.png Progetto di *stage* come intersezione tra le aree di competenze consolidate

Il progetto si è inserito coerentemente nella strategia aziendale, contribuendo su due fronti principali:
Scouting tecnologico: L'utilizzo di Flutter per la realizzazione di un'applicazione, caratterizzata da requisiti in ambito tecnologico.
Adattamento di competenze consolidate: sebbene *SyncLab* possieda già un panorama della sicurezza e dello sviluppo di applicazioni.
L'approccio fondato su posizioni all'avanguardia di *SyncLab* ha rappresentato quindi un motore concreto che ha reso possibile l'adattamento delle competenze consolidate.
Visione aziendale Offerte aziendali Per *SyncLab*, i programmi di *stage* rappresentano uno strumento strategico per il recrutamento di talenti.

[!ht] [width=]..../assets/collab.png Collaborazioni dell'azienda - Fonte: synclab.it

I programmi di *stage* si articolano principalmente in tre aree:

Integrazione: gli stagisti contribuiscono al perfezionamento di *software* già in uso, intervenendo su funzionalità specifiche.

Analisi e ottimizzazione: questa area prevede una valutazione approfondita delle soluzioni *software* esistenti, con l'obiettivo di ottimizzarle.

Innovazione: in questo ambito gli stagisti svolgono analisi teoriche e sperimentazioni su tecnologie emergenti, con l'obiettivo di contribuire alla creazione di nuovi prodotti.

[!ht] [width=0.8]..../assets/stage.png Contributo dei programmi di *stage* in SyncLab.

Stage in azienda Gli *stage* presso *SyncLab* rappresentano un investimento strategico bidirezionale, che riflette la duplice natura dell'esperienza.

Formazione e inserimento di risorse qualificate: Il programma di *stage* funge da canale primario per identificare e formare nuovi talenti.

Acquisizione di nuove competenze: Il programma di *stage* funge da banco di prova per l'innovazione continua, riducendo i rischi di investimento.

[!ht] [width=0.8]..../assets/cap2Stage.png Investimento strategico degli *stage* per l'azienda

Ruolo del *tutor* aziendale Durante il percorso ogni stagista è affidato a un *tutor*. Il *tutor* è una figura professionale che guida il *stage*.

Guida: il *tutor* propone linee guida su come organizzare il lavoro e fornisce allo stagista materiali utili che possono aiutare a superare le difficoltà.

Supporto: il *tutor* offre una serie di suggerimenti o soluzioni possibili ai problemi incontrati, questo per scongiurare l'errore.

Supervisore: il *tutor* offre *feedback* allo stagista nella valutazione del lavoro svolto per verificarne la qualità e supportarne lo sviluppo.

Nel mio caso i dialoghi con il *tutor* assegnato sono stati molto importanti in quanto senza il suo confronto, e i suoi feedback, non avrei potuto ragionare su come procedere.

Motivazione dello *stage* L'obiettivo dello *stage* è stato la valutazione di fattibilità e il potenziale impiego di nuove tecnologie.

[!ht] [width=0.8]..../assets/codebaseUnica.png Confronto tra lo sviluppo nativo e l'approccio *cross-platform* con *Flutter*.

Progetto proposto Nel contesto attuale, i dispositivi mobili rappresentano uno strumento di comunicazione ampiamente utilizzato.

Per tale motivo, l'azienda ha proposto la progettazione e l'implementazione di un'applicazione in grado di generare e gestire dati.

La realizzazione del progetto ha comportato l'identificazione e la risoluzione di alcune sfide progettuali e tecnologiche quali:

*Gestione delle chiavi crittografiche Il recupero della chiave privata usata per decifrare un messaggio rappresenta un problema critico.

[!ht] [width=0.9]..../assets/cap2crittografia.png Esempio di comunicazione critptata

*Integrazione con il lettore NFC L'applicazione deve supportare l'utilizzo del lettore NFC presente sui dispositivi mobili.

Obiettivi e vincoli Obiettivi Il progetto proposto è stato guidato da una serie di obiettivi volti a garantirne il successo.

Studio di fattibilità: effettuare uno studio dettagliato e un'analisi approfondita sulla fattibilità tecnica dell'utilizzo di tecnologie esistenti.

l'integrazione con il *server Firebase*;

il funzionamento del lettore NFC;

la generazione di una coppia di chiavi (pubblica e privata) crittografiche;

Questo per permettere di valutare le potenzialità e i limiti degli strumenti, in modo da individuare i migliori da utilizzare.

Sviluppo di una applicazione: per verificare che i componenti applicativi funzionino bene tra di loro, e che permettano di raggiungere gli obiettivi.

Vincoli tecnologici:

Utilizzo obbligatorio del framework Flutter:

L'intera applicazione deve essere sviluppata interamente tramite *Flutter*, con il conseguente utilizzo del linguaggio *Dart*.

Utilizzo del piano gratuito di Firebase:

L'uso di *Firebase* presenta alcune limitazioni derivanti dal piano gratuito, che esclude funzionalità specifiche presenti solo nel piano premium.

Vincoli architettura di sicurezza:

Protezione della chiave privata:

L'architettura deve impedire la trasmissione *online* della chiave privata, garantendo però un meccanismo sicuro che consente la sua utilizzazione.

Associazione delle chiavi ai documenti:

Ogni coppia di chiavi deve essere associata a un documento, e non a *tag NFC* generici.

Unicità del wallet per documento e account:

Non deve essere consentita la creazione di più *wallet* associati allo stesso documento per un medesimo *account*. Preservando l'unicità del *wallet*.

[!ht] [width=0.8]..../assets/dualitàChiavi.png Schema che rappresenta l'architettura di sistema, la chiave blu corrisponde alla chiave privata.

Scelta dello *stage* Motivazioni della scelta Ho conosciuto *SyncLab* nel mese di aprile attraverso una comunicazione telematica.

Parallelamente, durante il periodo estivo avevo ricevuto ulteriori offerte di stage da altre aziende e stavo valutando le loro proposte.

Alla fine ho scelto di intraprendere il mio percorso in *SyncLab* per i seguenti motivi:

Possibilità di lavorare da remoto, condizione per me fondamentale poiché, dovendomi spostare frequentemente e non potendo lavorare in ufficio.

Elevato livello di autonomia offerto dal progetto, trattandosi di uno *scouting* tecnologico finalizzato alla valutazione delle competenze.

Obiettivi personali prefissati Nell'attuale contesto tecnologico, l'uso dei dispositivi mobili è divenuto parte integrante della vita quotidiana.

L'introduzione della proposta di legge europea relativa al controllo delle *chat* e alla possibilità di analizzare i dati dei messaggi privati.

Da qui è nata la domanda che ha guidato parte della mia ricerca: esiste un modo per conservare le chiavi private in modo sicuro?

Nello sviluppo dell'applicazione mi sono quindi posto i seguenti obiettivi:

Progettare un'applicazione capace di gestire i dati privati degli utenti in modo distinto rispetto a quelli pubblici, incrementando la sicurezza.

Apprendere lo sviluppo di applicazioni per dispositivi mobili attraverso l'utilizzo del framework *Flutter*.

Acquisire i principi di crittografia necessari a consentire una comunicazione sicura tra due dispositivi mobili.

Per quanto qui permesso, lo *stage* presso *SyncLab* costituiva per me un'importante occasione per approfondire i concetti di base.

Attività	Settimane	Ore	
Ripasso costrutti di Java	X	5	
Studio di Dart	XX	30	
Studio di Flutter	XX	40	
Studio algoritmi di criptazione	X X X	30	
Analisi del problema	X	10	
[H] Progettazione della piattaforma	X	25	Pianificazione del lavoro durante le 8 settimane
Sviluppo maschera di <i>login</i>	X	5	
Sviluppo di un prototipo che genera chiavi	X	30	
Sviluppo applicazione finale	XXX	100	
Stesura finale della specifica tecnica	XX	20	
<i>Live demo</i> e presentazione finale	X	5	
totale ore		300	

Durante le prime settimane di *stage*, ho applicato una metodologia a cascata, essenziale per stabilire le fondamenta del progetto.

[H] [width=0.7]..../assets/modelloCascata.png Modello a cascata - Fonte: programmingacademy.it

Una volta completata la prototipazione, il progetto è transitato verso una metodologia agile, in particolare durante la fase di *design thinking*.

[H] [width=0.45]..../assets/agile.png Modello agile - Fonte: smartsheet.com

In definitiva, l'adozione dell'approccio ibrido ha garantito la copertura formativa e analitica iniziale, consentendo di raggiungere gli obiettivi.

Analisi dei requisiti La fase di analisi svolta durante le prime settimane del percorso di *stage*, è stata cruciale per definire i requisiti.

Requisiti funzionali I requisiti funzionali definiscono le funzionalità e i servizi specifici che l'applicazione deve fornire.

Codice Descrizione

[E1] Gestione dei dati privati degli utenti

*Interfacce e classi astratte In Dart sia le interfacce esplicite che le classi astratte sono dichiarate usando la keyword `abstract`. Un'interfaccia definisce un contratto: qualsiasi classe che la derivi deve utilizzare la keyword `implements`, impegnandosi a soddisfare tutti i metodi definiti nell'interfaccia.

[H] Esempio di interfaccia e classe astratta

```
abstract class ContrattoDiConnessione { void connetti(); void disconnect(); }
```

```
class B { void bMethod() { ... } }
```

```
abstract class BaseScreen { void logicaComune(); }
```

```
class GestoreServizio implements ContrattoDiConnessione, B { @override void connetti() /* implementazione */ { ... } }
```

```
class HomePage extends BaseScreen { void init() { super.logicaComune(); } }
```

In Dart, ogni classe definita genera implicitamente un contratto che implementa tutte le interfacce che eredita.

Il *framework* Flutter

La seconda sfida è stata lo studio del *framework* Flutter. Flutter non utilizza componenti di interfaccia nativi del sistema.

[H] [width=0.57]..../assets/flutterWidgets.png Composizione dei *widget* - Fonte: docs.flutter.dev

Per lavorare bene in Flutter è fondamentale comprendere che ogni elemento dell'interfaccia è rappresentato da un *widget*.

**Stateless widget* Uno *stateless widget* è un *widget* immutabile, ovvero un componente la cui configurazione non cambierà.

[H] *Stateless Widget* dart class Titolo extends StatelessWidget final String testo;

const Titolo(super.key, required this.testo);

@override Widget build(BuildContext context) return Text(testo);

**Stateful widget* Uno *stateful widget* è un *widget* mutabile, in grado di mantenere e aggiornare un suo stato interno.

[H] *Stateful Widget* dart class Contatore extends StatefulWidget const Contatore(super.key);

@override State<Contatore> createState() => ContatoreState();

class ContatoreState extends State<Contatore> { int valore = 0;

@override Widget build(BuildContext context) return Column(children: [Text('Valore: \${valore}'), ElevatedButton

Architettura a tre livelli L'applicazione sviluppata adotta il modello di architettura a tre livelli, una struttura che si basa su tre strati distinti:

- Il primo livello di presentazione rappresenta un punto di interazione tra l'utente e il sistema. È composto da tutto ciò che è visibile all'utente.
- Il secondo livello rappresenta la logica di *business*: coordina le operazioni tra il livello di presentazione e il livello dei dati.
- Il terzo livello è responsabile della persistenza e recupero dei dati. Definisce come questi vengono archiviati e resi disponibili per l'intero sistema.

Il *design pattern provider* Per gestire lo stato dei *widget* nell'applicazione realizzata è stato adottato il *design pattern provider*. Permette la separazione delle responsabilità (SoC⁴) eliminando la necessità di propagare le informazioni tramite la gerarchia di componenti. Aumenta le *performance* consentendoci di costruire i *widget* strettamente necessari quando lo stato cambia, evitando i ricaricamenti inutili. Fornisce un meccanismo per accedere e aggiornare lo stato rendendo il codice più facile da mantenere, facilitando quindi la gestione degli stati.

[H] [width=0.9]..../assets/providerPattern.png Architettura a tre livelli

la tecnologia NFC La tecnologia NFC permette a due dispositivi di connettersi, scambiarsi informazioni o attivare funzioni specifiche.

Lettura dati nei *chip* RFID Uno dei requisiti era l'implementazione di un meccanismo di riconoscimento dei documenti.

Per realizzare ciò ho analizzato due librerie Flutter che permettono la scansione e la scrittura attivando il sensore NFC: *nfc_manager* e *nfc*.

Grazie alla documentazione fornita dagli sviluppatori è stato facile eseguire un prototipo che permetta la scansione dei documenti.

[H] [width=0.5]..../assets/PrototipoNFC.png Applicazione prototipale per dimostrare la fattibilità della scansione NFC.

La libreria ci permette di recuperare i campi di ID, lo standard ISO del *chip* RFID (ad esempio la carta d'identità).

Testando l'applicazione con una varietà di documenti e *tag* NFC, ho riscontrato differenze significative nella struttura dei dati.

Nel caso della CIE ho rilevato un comportamento peculiare: il campo identificativo (*id*) letto tramite NFC non rappresenta sempre lo stesso valore.

Infatti il *chip* della Carta d'Identità Digitale (CIE) è progettato per memorizzare dati personali sensibili e proteggerli tramite crittografia.

L'identificatore temporaneo e non persistente rientra nelle misure previste per ridurre i rischi legati al tracciamento e alla truffa.

Questo rappresenta per il progetto un bel problema in quanto senza un identificativo del documento recuperabile tramite NFC non è possibile.

Una prima soluzione valutata è stata quella di consultare la piattaforma dedicata agli sviluppatori che realizzano i software.

A seguito di riunioni interne con il mio responsabile, si è scelto di non procedere su questa strada, sia per ragioni di tempo che di costi.

I documenti quali la CIE, la tessera sanitaria o le carte di credito adottano lo standard ISO/IEC 14443-4, che prevede la lettura dei dati.

[H] Funzione che permette la scansione del tag nfc [font-size=14pt]dart Future<NFCTag>? scanNfcTag(); fetchNfcData() async try if (true) Future<void> scanNfcTag()async if (isScanning) return; setState(() => isScanning = true); tryFinalNfcService =

Studio degli algoritmi di crittografia In questa sezione si espongono e si analizzano i principali algoritmi crittografici

Per analizzare la comunicazione digitale è importante considerare quattro aspetti:

Confidenzialità: assicurare che nessun osservatore non autorizzato sia in grado di leggere il messaggio durante la trasmissione.

Integrità: garantire che il messaggio non sia alterato da nessuno.

Autenticazione: verificare che il mittente sia chi dichiara di essere.

Non ripudio: impedire che il mittente possa negare di aver inviato il messaggio.

Algoritmi a chiave simmetrica Gli algoritmi di crittografia simmetrica utilizzano un'unica chiave segreta condivisa tra i partecipanti. Il funzionamento è descritto come segue:

Alice e Bob concordano o si scambiano in modo sicuro una chiave condivisa k ;

Alice prende il messaggio in chiaro P e applica un algoritmo di cifratura simmetrica S , ottenendo il messaggio cifrato C .

Quando Bob riceve C , applica l'algoritmo di decifratura D , utilizzando k , ricostruendo il messaggio originale P .

Pertanto per adottare questo algoritmo Alice e Bob devono conoscere e adoperare la stessa chiave utilizzando un canale sicuro. Se la chiave venisse rubata l'attaccante può fingere di essere il mittente originale.

Non è possibile distribuire in modo sicuro la chiave privata a un partner remoto senza utilizzare un altro sistema di sicurezza.

Algoritmi a chiave asimmetrica

Gli algoritmi di crittografia asimmetrica superano le limitazioni logistiche degli algoritmi di crittografia simmetrici.

Il funzionamento è descritto come segue:

Bob genera una coppia di chiavi (k_{pub}^B, K_{priv}^B) e rende k^B pubblico a tutti;

Alice ricerca la chiave pubblica di Bob e cifra il messaggio in chiaro P con la chiave pubblica di Bob k_{pub}^B , ottenendo un messaggio cifrato C .

Bob riceve il messaggio e lo decifra con utilizzando la propria chiave privata k_{priv}^B ottenendo P .

L'uso di questi algoritmi risolve il problema dello scambio della chiave presente negli algoritmi di crittografia simmetrica.

Realizzazione di una *chat end-to-end* Uno degli obiettivi del progetto riguarda la generazione di coppie di chiavi pubbliche e private.

```
[H] Generatore di numeri casuali dart SecureRandom getSecureRandom() final secureRandom = FortunaRandom()
Questo generatore viene passato al metodo generateRSAkeyPair() responsabile della creazione delle chiavi RSA. Il risultato è una coppia di chiavi  $(k_{pub}^B, K_{priv}^B)$ 
[H] Generazione della coppia di chiavi dart AsymmetricKeyPair|PublicKey, PrivateKey, generateRSAkeyPair(SecureRandom)
Per rispettare i requisiti e permettere allo stakeholder di visualizzare il risultato ottenuto le chiavi sono mostrate in formato Base64
[H] [width=0.9]..../assets/KeyPairVisualized.png Risultato del processo di creazione delle chiavi
```

Codifica e decodifica del messaggio Una volta generate le coppie di chiavi RSA è possibile utilizzarle per la realizzazione della crittografia.

La crittatura è gestita dal metodo `rsaEncryptBase64()` che elabora il messaggio in chiaro tramite il motore crittografico.

```
[H] Codifica di un messaggio mediante l'uso delle chiave pubblica dart Future<String>; rsaEncryptBase64(String plainText) async final encrypted = engine.process(Uint8List.fromList(utf8.encode(plainText))); return base64Encode(encrypted)
```

Il messaggio cifrato restituito viene poi convertito in un formato *Base64* in modo tale che sia memorizzato e trasmesso.

```
[H] [width=]..../assets/Base64Decoder.png decodifica del testo cifrato in base64 – Strumento: base64decode.org
La decifratura è affidata a rsaDecryptBase64(), che esegue l'operazione inversa decodificando il testo cifrato tramite la chiave privata.
```

```
[H] Decodifica di un messaggio mediante l'uso della chiave privata dart Future<String>; rsaDecryptBase64(String cipherText) async final decrypted = engine.process(base64Decode(cipherText)); return utf8.decode(decrypted); catch (e) return null
```

Problema di questo algoritmo L'implementazione dell'algoritmo mostrato presenta un problema specifico per la realizzazione di una *chat end-to-end*.

È possibile procedere in due modi diversi per la risoluzione del problema:

la prima soluzione consiste nel salvare localmente sul dispositivo del mittente una copia del messaggio in chiaro prima che venga cifrato.

la seconda soluzione adottata nel progetto consiste nell'invio di due copie dello stesso messaggio al *database*: la prima ci viene memorizzata e la seconda viene utilizzata per la decrittazione.

Tuttavia questa scelta espone a un rischio aggiuntivo con: un attaccante potrebbe individuare nel *database* due messaggi diversi.

```
[H] Modifica agli algoritmi di codifica e decodifica aggiungendo OAEP dart Future<String>; rsaEncryptBase64(String plainText) async final encrypted = engine.process(Uint8List.fromList(utf8.encode(plainText))); return base64Encode(encrypted)
```

Future<String>; rsaDecryptBase64(String cipherText, RSAPrivateKey privateKey) async final engine = OAEPEncoder

Gestione sicura delle chiavi pubbliche e private La protezione delle chiavi crittografiche generate sul dispositivo del mittente.

A questo punto avviene le separazione delle due chiavi, tenendo presente che la chiave privata deve essere salvata localmente.

// Inserire Immagine

La chiave pubblica viene inviata al *database* Firestore per essere associata al *wallet* dell'utente insieme ai dati di creazione.

```
DocumentReference docRef = await _firestore.collection('wallets').add(walletDataForFirestore);
```

Invece, per la chiave privata viene chiamata la classe di `SecureStorage` che utilizza la libreria di `flutter_secure_storage`.

```
[H] Salvataggio della chiave privata sul dispositivo dart await _secureStorage.writeSecureData(tempWallet.localKey);
```

La classe `SecureStorage` La classe `SecureStorage` è responsabile del salvataggio, lettura ed eliminazione della chiave privata.

```
[H] La classe SecureStorage dart
```

```
class SecureStorage implements ISecureStorage final FlutterSecureStorage storage;
```

```
SecureStorage(FlutterSecureStorage? storage) : storage = storage ?? const FlutterSecureStorage();
```

```
@override Future<void> writeSecureData(String key, String value) async await storage.write(key: key, value: value);
```

```
@override Future<String?> readSecureData(String key) async final value = await storage.read(key: key); return value;
```

```
@override Future<void> deleteSecureData(String key) async await storage.delete(key: key);
```

Per accedere al dato salvato localmente è quindi necessario che alla creazione della coppia di chiavi venga anche generata una chiave identificativa.

```
[H] Creazione della chiave identificativa dart var uuid = const Uuid(); final newLocalKeyIdentifier = uuid.v4();
```

Tramite il pacchetto `uuid` è possibile generare una chiave unica da associare alla chiave privata per permetterne il recupero.

Procedura di recupero chiave Questa architettura delega all'utente la responsabilità esclusiva sulla propria chiave privata.

Risultati Raggiunti L'applicazione progettata e sviluppata durante il percorso di *stage* implementa tutte le principali funzionalità previste.

*Funzionalità Implementate

Accesso e registrazione: l'utente può creare un account e accedere all'applicazione tramite l'inserimento di *email* e *password*.

Creazione ed eliminazione del *wallet*: un utente può generare la propria coppia di chiavi crittografiche e, se necessario, rimuovere la chiave privata dal dispositivo.

Rimozione della chiave privata dal dispositivo: è stata introdotta la possibilità di rimuovere la chiave privata dal dispositivo.

Recupero del *wallet*: l'utente può ripristinare il proprio *wallet* in caso di smarrimento del dispositivo o migrazione verso un altro dispositivo.

Chat end-to-end: l'applicazione mette a disposizione un servizio di messaggistica sicura basato su chiavi crittografiche.

Ricerca tramite email: l'utente può individuare un altro utente tramite ricerca per indirizzo *email* e avviare con esso una conversazione.

Ricerca tramite scansione NFC: attraverso scansione NFC di un documento fisico, l'applicazione può identificare automaticamente il contenuto.

Risultati Quantitativi

Bibliografia
[heading=subbibliography,title=Siti web consultati,type=online]