**Relazione Progetto Programmazione per Dispositivi Mobili**

E-Tutoring

**Gruppo**

Bortolotti Simone

Davide De Cenzo

Marignati Luca

# Sommario

[Sommario 2](#_Toc76206117)

[Deviazioni rispetto a quanto specificato nella scheda dell’app 3](#_Toc76206118)

[Metodologia di sviluppo utilizzata 4](#_Toc76206119)

[Funzionalità dell’app e design 5](#_Toc76206120)

[Architettura dell’app e le possibili alternative 19](#_Toc76206121)

[Principali librerie utilizzate e le possibili alternative 20](#_Toc76206122)

[flutter\_secure\_storage 21](#_Toc76206123)

[Scelta implementativa 21](#_Toc76206124)

[Implementazione 22](#_Toc76206125)

[Alternative: Secure storage / Shared Preferences / SQLite / Local File Storage 23](#_Toc76206126)

[Supporto per device multipli 24](#_Toc76206127)

[Supporto per le lingue diverse 25](#_Toc76206128)

[Struttura del codice realizzato e di altre risorse realizzate 27](#_Toc76206129)

[Test effettuati 28](#_Toc76206130)

[Controller 28](#_Toc76206131)

[Screens 29](#_Toc76206132)

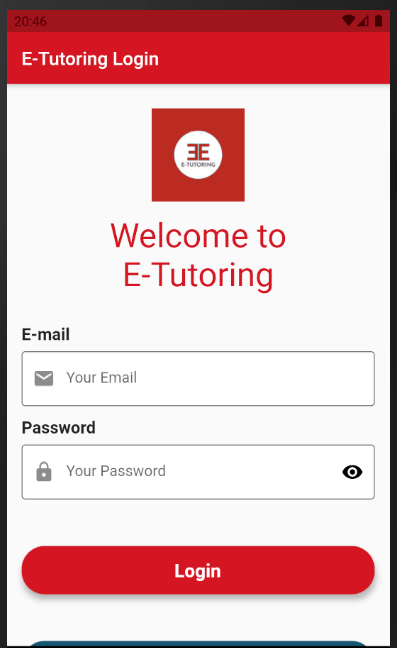
[Totale test effettuati: 64 30](#_Toc76206133)

# Deviazioni rispetto a quanto specificato nella scheda dell’app

# Metodologia di sviluppo utilizzata

# Funzionalità dell’app e design

**Login**: Avviando l’applicazione ci imbattiamo nella schermata iniziale (E-Tutoring Login) dove gli utenti possono eseguire l’accesso all’applicazione inserendo e-mail e password, oppure se utilizzano l’app per la prima volta, hanno la possibilità di iscriversi;



È stata prevista una doppia validazione:

* FRONT-END:
  + e-mail e password non vuoti;
  + e-mail che rispetta il formato “test@email.it”;
* BACK-END: verifica che l’email e la password inserite dall’utente corrispondano a quelle inserite nel DB (chiamando il WS user\_login.php - <https://www.e-tutoring-app.it/ws/user_login.php>);

|  |  |
| --- | --- |
|  |  |

1. **Registrazione di un nuovo utente**: un utente GUEST può registrarsi come studente o tutor all’applicazione compilando i campi relativi all’email, alla password, e selezionando il ruolo, il grado del corso e il curriculum;

Immagine che contiene testo

Descrizione generata automaticamente

Filtri e validazioni lato front-end:

* + e-mail e password non vuoti;
  + e-mail che rispetta il formato “test@email.it”;
  + passowrd e confirm password devono matchare;

Immagine che contiene testo

Descrizione generata automaticamente

1. **Privacy policy**

Nella Schermata seguente vengono descritte all’utente le modalità di trattamento dei dati, come ad esempio le misure di sicurezza utilizzate per impedire la divulgazione di dati personali e a chi è possibile dare l’accesso ad essi. Inoltre informa l’utente sul periodo di conservazione dei dati personali.

Immagine che contiene testo

Descrizione generata automaticamente

1. **Drawer menu**

Tramite una sorta di Access Control List è stata differenziata la visibilità del menù in base al ruolo dell’utente. Alcune funzionalità sono comuni ad entrambi i profili, altre sono specifiche per il tipo di utente (es. il tutor può aggiungere delle disponibilità di orario/giorni, lo studente può effettuare recensioni sui tutor, ecc.).

|  |  |
| --- | --- |
| **Tutor** | **Studente** |
|  |  |

1. **Home differente per profilo studente e tutor**

La schermata di profilo è divisa in due TAB:

* profilo;
* notifiche;

**Tab profilo**

Nelle schermate seguenti vengono mostrate le differenze tra il profilo studente (a destra) ed il profilo tutor (a sinistra). Da notare che la schermata viene “scomposta” in due sezioni separate: la prima riguarda i dati personali dell’utente ed è identica sia per lo studente che per il tutor; la seconda parte identifica i dati dello studente o del tutor e vediamo che differiscono: per il tutor viene indicato il ruolo, mentre per lo studente oltre che al ruolo vengono indicate anche altre informazioni come, ad esempio, il numero di matricola e le informazioni riguardi il corso di Laurea a cui lo studente è iscritto;

|  |  |
| --- | --- |
| **Tutor** | **Studente** |
|  |  |

Inoltre, in questo widget sono presenti le funzionalità per effettuare il Logout (in alto a destra) e tramite il floatingActionButton (Icons.edit) è possibile fare la modifica dei dati inseriti dall’utente.

Immagine che contiene testo

Descrizione generata automaticamente

**Tab notifiche**

In questo tab è possibile visionare le notifiche dei Tutor.

Non abbiamo previsto notifiche per gli studenti.

I tutor ricevono una notifica quando uno studente si iscrive ad uno slot relativo ad un corso.

Es. Davide De Cenzo si iscrive al corso di Programmazione per Dispositivi Mobili tenuto da Paolo Rossi il giorno 20 Luglio 2020 dalle ore 16:00 alle ore 19:00.

**In questo caso, il tutor riceverà la notifica sia a livello globale (utilizzando Firebase per la gestione e l’invio delle notifiche) di sistema operativo (es. Android) sia a livello locale di applicazione**:

* incremento del contatore delle nuove notifiche;
* visualizzazione evidenziata delle nuove notifiche (in giallo);

|  |  |
| --- | --- |
| **Notifica a livello globale (SO)** | **Notifica locale all’applicazione** |
| C:\Users\Borla\Desktop\Screenshot (19).png |  |

1. **Settings/Impostazioni**

Il Widget delle impostazioni è molto semplice e minimale per garantire un corretto e veloce funzionamento da parte dell’utente. È possibile visualizzare la lingua (supporta Italiano ed Inglese), ed è possibile eliminare l’account dopo aver confermato di voler realmente eseguire l’operazione di cancellazione (AlertDialog).

|  |  |
| --- | --- |
| Immagine che contiene testo  Descrizione generata automaticamente |  |

**TUTOR**

1. **Funzionalità del Tutor: Tutoring Lesson/lezioni private**

Immagine che contiene testo

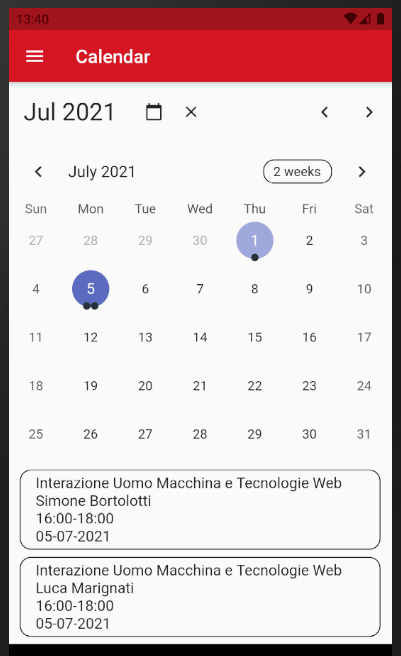
Descrizione generata automaticamenteIn questo Widget è possibile visualizzare la lista (ListView) delle lezioni che il tutor deve sostenere.

Per ogni lezione vengono indicati:

* il nome del corso;
* il nome dello studente;
* il giorno di prenotazione;
* l’orario di prenotazione.

Per facilitare l’usabilità del widget è stato prevista una **funzionalità di ricerca** (in alto a destra) in cui il tutor può ricercare le proprie lezioni private in base al nome del corso.

In basse a destra, tramite il floatingActionButton (TODAY) è possibile visionare le lezioni che il tutor deve sostenere il giorno corrente in modo da sapere sempre se vi sono lezioni prenotate da sostenere oggi ed evitare dimenticanze.

Proprio per questo, si è deciso di implementare un **Widget Calendario** interno all’applicazione che mostra le lezioni private (vedi sotto).

1. **Funzionalità del Tutor: Calendario**

Nella sezione sottostante possiamo visionare il calendario personale per ogni tutor con i relativi impegni, ossia con le relative lezioni private ad una certa data in una certa ora. Per semplificarne la lettura sul calendario attraverso dei punti è possibile capire se in un determinato giorno sono previste lezioni o meno.

Cliccando sul giorno è possibile anche visionare più nel dettaglio le lezioni relative a quel giorno

1. **Funzionalità del Tutor: My Course/I miei corsi**

Questo Widget offre la possibilità di visualizzare la lista (ListView) dei corsi insegnatisi dal tutor.

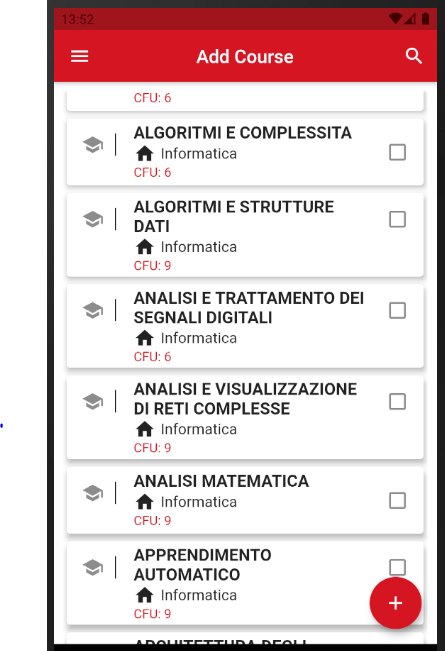
In aggiunta è possibile:

* aggiungere un corso cliccando il floatingActionButton in basso a destra: si rimanda al Widget AddCourse;
* eliminare i corsi su cui offrire le proprie ripetizioni agli studenti: la listView è selezionabile, ovvero è possibile selezionare i corsi che si vogliono eliminare.

Dopo la lezione, occorre cliccare sul bottone in alto a destra (il cestino).

|  |  |
| --- | --- |
|  |  |

1. **Funzionalità del Tutor: Add Course/Aggiungi corso**



Tramite questo widget, il tutor può aggiungere dei corsi a quelli da lui insegnati.

Come si nota dalla figura sopra, è possibile vedere la lista (ListView) di tutti i corsi.

Tale ListView è selezionabile, il tutor può selezionare può corsi è aggiungerli cliccando sul floatingActionButton in basso a destra (Icons.add).

Inoltre, per migliorare l’usabilità è stata prevista la funzionalità di ricerca (in alto a destra).

1. **Funzionalità del Tutor: My Availability/Le mie disponibilità**

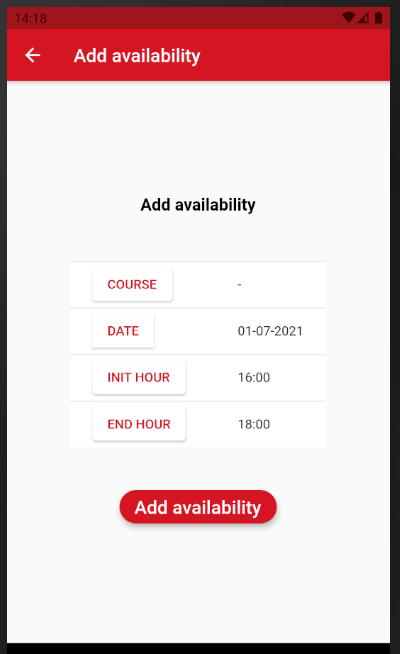
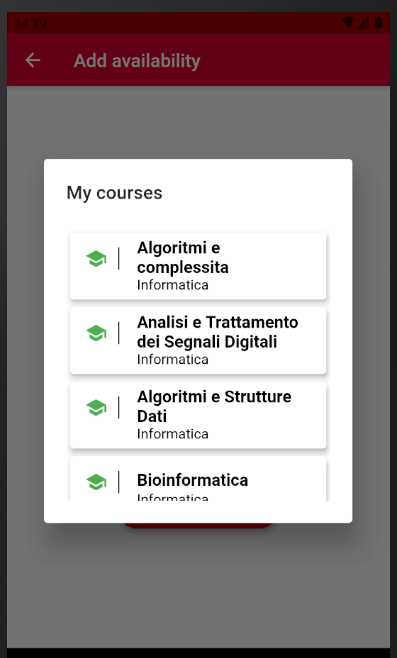
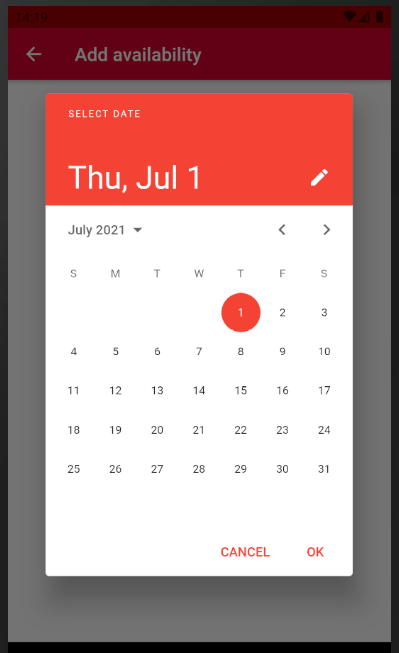
In questo Widget è possibile visualizzare la lista delle disponibilità del tutor.

Come si nota, il singolo item della ListView mostra è selezionabile o bloccato se la lezione è stata già prenotata da uno studente o meno.

Nel caso in cui non è stata prenotata da nessuno ed è selezionabile è possibile rimuovere la disponibilità selezionando lo slot e cliccando sul cestino (in alto a destra).

In basso a destra è presente il floatingActionButton (Icons.add) che permette di aggiungere una disponibilità selezionando corso, data, inizio e fine della disponibilità (vedi figure sotto).

Immagine che contiene testo

Descrizione generata automaticamente    Immagine che contiene testo, orologio

Descrizione generata automaticamente

1. **Funzionalità del Tutor: Review about me/recensioni su di me**

In questo Widget è possibile visualizzare la lista (ListView) delle recensioni fatte dagli studenti. Ogni recensione mostra il nome dello studente, il suo commento sul tutor e un voto (da 1 a 5).

Immagine che contiene testo

Descrizione generata automaticamente

**STUDENTE**

1. **Funzionalità dello Studente: Course/Corsi**

|  |  |
| --- | --- |
|  |  |

Il Widget mostra i corsi che possono essere scelti per poter richiedere una lezione privata. Viene mostrata una ListView contenente tutti i corsi relativi al corso di Laurea a cui l’utente è iscritto (Es. laurea Magistrale di Informatica). La nostra applicazione comprende tutti i corsi relativi all’università di informatica, matematica e fisica (laurea triennale o magistrale).

**Il Widget offre la possibilità di ricercare un corso in base al nome.**

1. **Funzionalità dello Studente: Course Detail/ Dettaglio del corso**

Il click di un item della ListView del Widget sopra citato (Course) permette di visualizzare le informazioni di dettaglio del corso (CFU, Dipartimento, Currisulum).

In questo Widget è possibile procedere alla ricerca del tutor in base al corso selezionato cliccando sul bottone “Search Tutor”.

Immagine che contiene testo

Descrizione generata automaticamente

Esempio di interazione dell’utente:

* Ricerca del corso “Bioinformatica”
* Visualizzazione del dettaglio del corso: ricerca Tutor
* Ricerca dei Tutor che offrono ripetizioni del corso selezionato

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. **Funzionalità dello Studente: My Lesson/Le mie lezioni**

In maniera similare a quanto visto per le lezioni offerte dai Tutor, uno Studente può visualizzare la lista delle lezioni prenotate in ordine di data crescente.

Anche qui, per facilitare l’usabilità, è possibile ricercare una lezione specifica e visualizzare se vi sono lezioni prenotate per il giorno corrente.

Immagine che contiene testo

Descrizione generata automaticamente

1. **Funzionalità dello Studente: Add review/fare una recensione**

Come si nota dalla lista precedente, vi sono 2 tipi di item della ListView:

1. lezioni sostenute che possono essere recensite da parte dello studente (lezioni con data minore a data corrente);
2. lezioni ancora da sostenere (quindi non recensibili).

Le lezioni sostenute possono essere recensite dallo studente cliccando sull’item specifica.

Al click dell’item, si ha il Widget che permette di scrivere una recensione al Tutor (es. Paolo Rossi) definendo un commento e uno score (intero da 1 a 5 con validazione);

|  |  |
| --- | --- |
| Immagine che contiene testo  Descrizione generata automaticamente |  |

1. **Funzionalità dello Studente: Search Tutor/Ricerca Tutor**

Immagine che contiene testo

Descrizione generata automaticamente

Questo Widget fornisce la lista (ListView) dei tutor disponibili.

Ogni elemento della lista visualizza le informazioni relative al singolo tutor come ad esempio il nome, l’e-mail, i suoi insegnamenti e la media degli score delle recensioni fatte dagli studenti.

Anche qui è possibile ricerca i tutor per nome e insegnamenti.

|  |  |
| --- | --- |
| Immagine che contiene testo  Descrizione generata automaticamente |  |

1. **Funzionalità dello Studente: Dettaglio del Tutor**

Questo Widget visualizza le informazioni del tutor scelto: informazioni generali, descrizione, corsi insegnati e media delle recensioni.

Inoltre, sono fornite le segue funzionalità:

* floatingActionsButton icons.calendar che permette di visualizzare le disponibilità del tutor;
* icons.review che permette di visualizzare le recensioni fatte dagli studenti.

|  |  |  |
| --- | --- | --- |
| Immagine che contiene testo  Descrizione generata automaticamente |  |  |

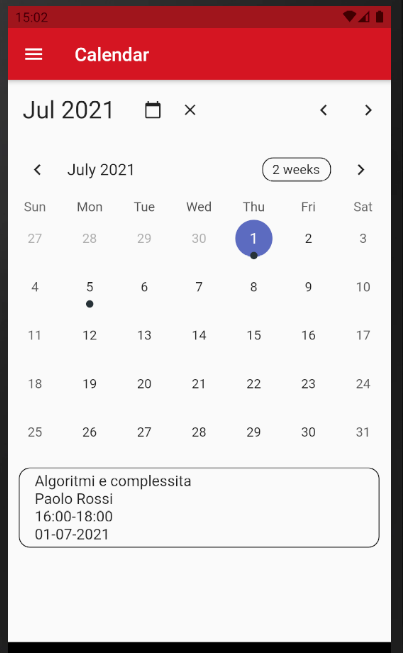
1. **Funzionalità dello Studente: My Reviews/Le mie recensioni**

Immagine che contiene testo

Descrizione generata automaticamente

1. **Funzionalità dello Studente: Calendario**

Il Widget mostra le lezioni prenotate dallo studente.



# Architettura dell’app e le possibili alternative

# Principali librerie utilizzate e le possibili alternative

* **flutter\_secure\_storage: ^4.2.0**

<https://pub.dev/packages/flutter_secure_storage>

È un plug-in Flutter per archiviare i dati in maniera sicura.

* **flutter\_localizations**

<https://flutter.dev/docs/development/accessibility-and-localization/internationalization>

* **intl: ^0.17.0**

<https://pub.dev/packages/intl>

* **http: ^0.13.3**

<https://pub.dev/packages/http>

Questo pacchetto contiene un insieme di funzioni e classi di alto livello che semplificano l'utilizzo delle risorse HTTP. È multipiattaforma e supporta dispositivi mobili, desktop e browser.

* **flutter\_local\_notifications: ^5.0.0+4**

Un plug-in multipiattaforma per la visualizzazione di notifiche locali.

<https://pub.dev/packages/flutter_local_notifications>

* **firebase\_core: ^1.1.0**

Un plug-in Flutter per utilizzare l'API Firebase Core, che consente la connessione a più app Firebase.

<https://pub.dev/packages/firebase_core>

* **firebase\_messaging: ^10.0.2**

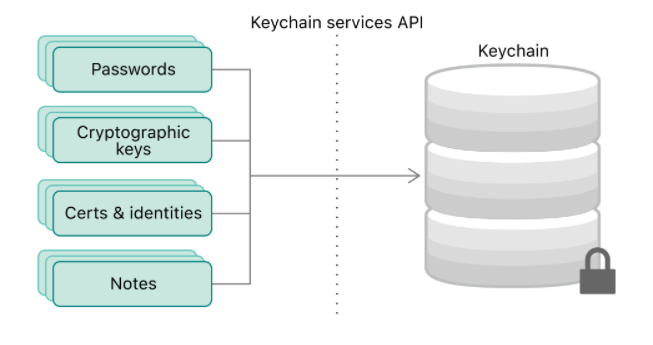
Un plug-in Flutter per utilizzare l'API Firebase Cloud Messaging.

<https://pub.dev/packages/firebase_messaging>

<https://firebase.google.com/docs/cloud-messaging>

## flutter\_secure\_storage

Concentriamoci descrivendo in dettaglio la libreria “flutter\_secure\_storage” discutendone i pro e i contro e analizzando le possibili alternative.



La libreria permette di **astrarre dalle primitive native** dei vari dispostivi fornendo dei metodi che permettono di scrivere (write) e leggere (read) le chiavi:

* **metodo write(key, value);**
* **metodo read(key);**
* **metodo delete(key).**

Nativamente:

* per iOS le chiavi vengono salvate in un archivio chiamato **Keychain** (<https://developer.apple.com/documentation/security/keychain_services#//apple_ref/doc/uid/TP30000897-CH203-TP1>);
* per Android, viene utilizzata la crittografia AES: la chiave segreta AES viene crittografata con RSA e viene archiviata nel **KeyStore** (<https://developer.android.com/training/articles/keystore>);
* per Linux viene utilizzata **libsecret** (<https://wiki.gnome.org/Projects/Libsecret>);

Nota: KeyStore è stato introdotto in Android 4.3 (livello API 18).

Il plugin non funzionerebbe per le versioni precedenti.

Scelta implementativa: **abbiamo utilizzato questa libreria per l’archiviazione sicura delle informazioni dell’utente (username, password e ruolo) nel dispositivo**.

Caratteristiche per Android (per IOS sono equivalenti):

* il Keystore fornisce un contenitore sicuro, che può essere utilizzato dalle applicazioni per memorizzare le chiavi private, in un modo che sia difficile per gli utenti malintenzionati (non autorizzati) recuperare le informazioni private;
* un’applicazione è in grado di memorizzare più chiavi nel Keystore, ma può solo visualizzare, e ricercare, le **sue chiavi**;
* il vantaggio di memorizzare una chiave nel KeyStore è che consente di utilizzare le chiavi senza esporre il contenuto segreto di quella chiave: **i dati chiave non entrano nello spazio dell'app**. Le chiavi sono protette da permessi in modo che solo la tua app possa accedervi;
* vantaggi relativi alla sicurezza dell’accesso al KeyStore e alle chiavi salvate.

Implementazione: è la stata implementata la classe **UserSecureStorage** che si occupa di comunicare con il KeyStore (per Android) o il Keychain (per iOS). Per completezza, di seguito, abbiamo riportato il codice della classe.

Immagine che contiene testo

Descrizione generata automaticamente

### Alternative: Secure storage / Shared Preferences / SQLite / Local File Storage

Le alternative a questo approccio potevano essere molteplici a partire dall’utilizzo delle Shared Preferences o di altre tipologie di Data Storage (es. SQL Database).

Brevemente analizziamo i pro e i contro dei vari approcci:

|  |  |  |
| --- | --- | --- |
| **Shared Preferences** | **SQLite** | **Local File Storage** |
| Si tratta di un archivio chiave/valore in cui è possibile salvare un dato con una determinata chiave.  Per leggere i dati dal negozio è necessario conoscere la chiave dei dati.  **Questo rende la lettura dei dati molto semplice.**  Un utilizzo tipico è la gestione delle **preferenze utente** (es. modalità scura dell’applicazione, ultimo tab usato, ecc.): questi sono piccoli dettagli che **non richiedono oggetti di grandi dimensioni** o database altamente strutturati e sono informazioni che non richiedono un'archiviazione sicura. **Vantaggio principale: velocità nel reperimento del dato**. | Adatto per il salvataggio di grandi quantità di stessi dati strutturati.  Poiché i **dati sono strutturati** e gestiti dal database, possono essere interrogati per ottenere un sottoinsieme dei dati che corrisponde a determinati criteri utilizzando un linguaggio di query come SQL.  Ciò rende possibile la **ricerca nei dati**. | È una modalità **semplice** nella quale andiamo a creare un file, tipicamente un file di testo, nel quale possiamo scrivere e dal quale possiamo leggere alcuni dati. Di default questo tipo di salvataggio è un **salvataggio privato** e ciò implica che il file può essere letto e scritto esclusivamente dall'applicazione che lo ha creato e non è dunque possibile accedervi da un'altra applicazione. |
| **Problemi di spazio**:  i dispositivi hanno dei limiti di memoria e il sistema potrebbe eliminare le shared preferences se occupano molto spazio: questo significa che **nessuna delle preferenze condivise dovrebbe essere una funzionalità vitale dell'app**.  **Dati non sicuri:** è facile recuperare le informazioni da parte di utenti malintenzionati.  Proprio per questo non devono essere memorizzate informazioni private.  **È difficile archiviare e leggere dati strutturati di grandi dimensioni** in quanto è necessario definire la chiave per ogni singolo dato.  Inoltre, **non è possibile eseguire ricerche** all'interno dei dati se non si ha un certo concetto per nominare le chiavi. | Ovviamente la gestione e la ricerca di grandi insiemi di dati influenzano le prestazioni, quindi la **lettura dei dati da un database può essere più lenta rispetto alla lettura dei dati da Shared Preferences**. | **È difficile archiviare e leggere dati strutturati di grandi dimensioni** in quanto è necessario leggere l’intero file.  Inoltre, **non è possibile eseguire ricerche.** |

# Supporto per device multipli

# Supporto per le lingue diverse

È possibile impostare la lingua del dispositivo andando su “Settings” 🡪 “Languages” (es. su Android) e portando la lingua scelta in cima alla lista (vedi figura sotto).

|  |  |
| --- | --- |
|  |  |

**L’applicazione, in base, a questa configurazione adatterà i propri contenuti in base alla lingua scelta sfruttando le librerie *flutter\_localizations* e *intl* che** **forniscono le funzionalità di internazionalizzazione, localizzazione e la traduzione dei messaggi**.

**Lingue supportate dell’applicazione**: **inglese e italiano**.

Sarebbe triviale aggiungere il supporto di un’altra lingua:

* la cartella “I10n” contiene i file di traduzioni in formato ARB in cui le risorse sono codificate come oggetti JSON;
* in caso si voglia aggiungere il supporto di una terza lingua basterebbe aggiungere il relativo file che contiene le traduzioni per la nuova lingua;

In questi file vengono specificate **coppie CHIAVE-VALORE**. Es:

* inglese: chiave “welcome”, valore “welcome to\nE-Tutoring”;
* italiano: chiave “welcome”, valore “Benvenuto in\nE-Tutoring”;

|  |  |
| --- | --- |
| **app\_en.arb** | **app\_it.arb** |
|  |  |

FLUTTER GENERATE TRUE: evidenziamo il comportamento della libreria: in fase di compilazione dell’applicazione, vengono generate le seguenti classi:

* **.dart\_tool/flutter\_gen/genI10\_n/app\_localizations.dart**: classe astratta che contiene le chiavi di traduzione;
* **.dart\_tool/flutter\_gen/genI10\_n/app\_localizationsIt.dart**: classe che espone i metodi get delle chiavi fornendo il valore della relativa traduzione in lingua italiana;
* **.dart\_tool/flutter\_gen/genI10\_n/app\_localizationsEn.dart**: classe che espone i metodi get delle chiavi fornendo il valore della relativa traduzione in lingua inglese;

|  |  |
| --- | --- |
| **app\_localizationsIt.dart** | **app\_localizationsEn.dart** |
|  |  |

CONFIGURAZIONE

Immagine che contiene testo

Descrizione generata automaticamente

UTILIZZO (tramite AppLocalizations): si definisce la chiave della traduzione (es. login):

Immagine che contiene testo

Descrizione generata automaticamente

# Struttura del codice realizzato e di altre risorse realizzate

# Test effettuati

Abbiamo sviluppato i test suddividendo in cartelle rispettando la struttura dell’architettura del codice: per esempio nella cartella di test “*controller*” abbiamo testato le funzioni relative alla cartella “*lib/controller*”.

Immagine che contiene testo

Descrizione generata automaticamente

## Controller

Utilizzando la libreria Mockito abbiamo testato i metodi relativi alle chiamate http.

Principalmente abbiamo testato i casi di successo (risposta http 200) e fallimento (risposta http 404 not found).

Mostriamo di seguito un esempio di test spiegando i passi strutturali:

1. **inizializzazione del MockClient()** che permette di simulare il comportamento di un client http (browser);
2. **definizione dei parametri richiesti** **dal WS**: in questo caso notiamo *degree\_name* e *degree\_type\_note*;
3. simulazione della chiamata http utilizzando il mockclient: in questo caso abbiamo simulato la chiamata http:

<https://www.e-tutoring-app.it/ws/curriculum_path_by_degree.php?degree_name=informatica&degree_type_note=Laurea%20Triennale>

1. abbiamo definito la risposta in formato JSON: *'[{"degree\_path\_name": "Immagini, Visione e Realtà Virtuale"}]'* e definito lo status (es. 200 o 404);
2. abbiamo chiamato la funzione effettiva implementata (es. getCurriculumFromWS);
3. infine, abbiamo eseguito i vari test (es. test sul tipo ritorno);

Di seguito mostriamo il codice relativo al test e alla funzione testata.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Gli altri test relativi alle chiamate http e ai WS sono del tutto similari.

Per i dettagli rimandiamo al codice.

## Screens

Per gli screens, ovvero i widget (Login, ChangePassword e Signup) abbiamo effettuato, anche, degli Unit Test relativi alle funzioni utilizzate.

Es.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Oltre a questi testi unitari, come in precedenza, abbiamo sviluppato i test relativi alle chiamate http utilizzando Mockito.

## Totale test effettuati: 64

Immagine che contiene testo

Descrizione generata automaticamente