**Relazione Progetto Programmazione per Dispositivi Mobili**

E-Tutoring

Flutter Application

**Gruppo**

Bortolotti Simone

Davide De Cenzo

Marignati Luca

# Sommario

[Sommario 2](#_Toc76235493)

[Deviazioni rispetto a quanto specificato nella scheda dell’app 3](#_Toc76235494)

[Metodologia di sviluppo utilizzata: metodologia agile 4](#_Toc76235495)

[Funzionalità dell’app e design 6](#_Toc76235496)

[Architettura dell’app: struttura del codice realizzato e di altre risorse realizzate 20](#_Toc76235497)

[Sicurezza: Basic Authentication 20](#_Toc76235498)

[Database MySql 21](#_Toc76235499)

[Web Service implementati 24](#_Toc76235500)

[Model 26](#_Toc76235501)

[Paradigma MVC 27](#_Toc76235502)

[Principali librerie utilizzate e le possibili alternative 28](#_Toc76235503)

[flutter\_secure\_storage 29](#_Toc76235504)

[Scelta implementativa 29](#_Toc76235505)

[Implementazione 30](#_Toc76235506)

[Alternative: Secure storage / Shared Preferences / SQLite / Local File Storage 31](#_Toc76235507)

[Flutter e supporto per device multipli 32](#_Toc76235508)

[Supporto per le lingue diverse 33](#_Toc76235509)

[Test effettuati 36](#_Toc76235510)

[test/controller - Mockito 36](#_Toc76235511)

[test/screens 38](#_Toc76235512)

# Deviazioni rispetto a quanto specificato nella scheda dell’app

La maggior parte delle funzionalità previste in fase di analisi sono state implementate con successo.

Le funzionalità non implementate sono:

* **ricezione via e-mail dell’iscrizione**: non ritenuta di fondamentale importanza in relazione alla quantità di lavoro richiesto per l’implementazione: occorreva configurare un server SMTP per l’invio e la consegna dei messaggi di posta;
* **chat tra studenti e tutor**: per questioni di tempi e di consegne non è stata implementata la chat in quanto dopo un’analisi delle funzionalità dell’applicazione sarebbe costato troppo in relazione allo scopo dell’app, ovvero dare la possibilità agli studenti di prenotarsi per il tutoraggio dei corsi.

Inoltre, considerando che l’applicazione mostra il numero di telefono e l’e-mail del tutor, è possibile mettersi direttamente in contatto con il tutor stesso, senza aver bisogno di una chat interna all’applicazione.

Ipoteticamente, infatti, si può pensare che ulteriori esigenze dello studente vengano comunicate al tutor mediante chiamata e/o utilizzo di altre applicazioni (es. Skype, social network, WhatsApp, Telegram, ecc.).

Riguardo i vincoli temporali pianificati possiamo ritenerci soddisfatti del lavoro di pianificazione svolto. Per la data 1° luglio, lo sviluppo è stato terminato come previsto.

Immagine che contiene testo

Descrizione generata automaticamente

# Metodologia di sviluppo utilizzata: metodologia agile

È stata utilizzata una metodologia di sviluppo di tipo agile in modo da garantire una distruzione continua di software efficiente creati in modo rapido e iterativo. La metodologia ci ha consentito di **rilasciare rapidamente release piccole, frequenti e modifiche al software in piccole porzioni** con l'obiettivo di migliorare la soddisfazione di un ipotetico cliente (sviluppo puramente accademico).

La metodologia ci ha consentito di adottare un approccio leggero e di integrare le modifiche in qualsiasi fase del ciclo di vita (planning – analisi – implementazione – testing – valutazione), anziché ostacolarle.



Sono state, prima, sviluppate le funzionalità strettamente necessarie:

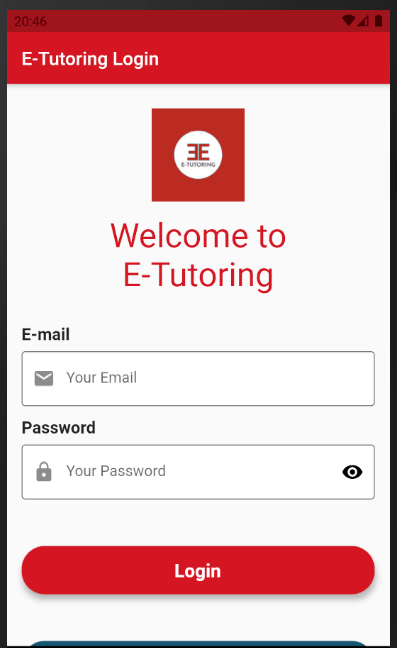
* per esempio, la prima funzionalità implementata è stata quella della prenotazione di una lezione privata da parte di uno studente;
* successivamente sono state, invece, implementate funzionalità aggiuntive come, ad esempio, il calendario, la possibilità di cambiare la password, la possibilità di effettuare una recensione di un tutor, ecc.

In particolare, come team di sviluppo abbiamo lavorato mettendo in risalto i seguenti valori fondamentali:

* + **comunicazione**: come gruppo siamo rimasti quotidianamente in contatto per creare un continuo flusso e scambio di informazioni in modo da velocizzare lo sviluppo e la risoluzione di problemi;
  + **pair programming (programmazione in coppia)**: il codice è stato prodotto dal team (mediante condivisione di schermo) in contemporanea sullo stesso task/funzione. Ci siamo alternati per eseguire la fase di scrittura e di revisione del codice (conducente e osservatore) scambiandoci i ruoli;
  + **semplicità**: abbiamo sviluppato l’applicazione seguendo il principio “**il modo più semplice per raggiungere un risultato è anche quello più efficace**”. Abbiamo, dunque, sviluppato l’applicazione privilegiando la semplicità dei concetti (progettazione della logica e delle view).
  + **modifiche incrementali**: le modifiche sono state eseguite gradualmente: invece di attuare grandi aggiornamenti, sono state implementate le funzionalità e affrontati i problemi uno dopo l’altro fino alla completa risoluzione;
  + **processo continuo** **di refactoring**: il progetto è stato continuamente migliorato (in maniera iterativa) rimuovendo codice duplicato, fattorizzando il codice utilizzato in un classi/widget (es. definendo funzioni di libreria e variabili globali) e rimuovendo gli elementi superflui;
  + **simple design**: è stato seguito questo principio per permettere di avere un codice comprensibile a tutti. Si è cercato di separare le varie componenti seguendo il pattern MVC (Model – View – Controller):
    - le chiamate http sono state implementate in opportune classi/funzioni in modo da separare la logica dalla visualizzazione (view)**;**
    - i modelli che rappresentano i campi di una tabella/oggetto relativi al backend (WS/DB) sono stati implementati in classi che rappresentano i singoli oggetti;
  + **testing**: Test-Driven Development: il team di sviluppo ha scritto i test prima ancora di creare il codice sorgente definitivo: in particolare i test maggiormente implementati sono stati quelli che riguardano l’utilizzo del WS. Prima di procedere all’implementazione dei metodi utilizzati per rappresentare i dati nei vari Widget sono stati ampiamente testate le chiamate http utilizzando Mockito e verificando che gli oggetti JSON restituiti fossero realmente quelli attesi;
  + **feedback continuo:** abbiamo testato il codice e la funzione a livello di view fin dal primo giorno in modo da verificare la corretta implementazione;
  + **software funzionante più che documentazione esaustiva**.

# 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

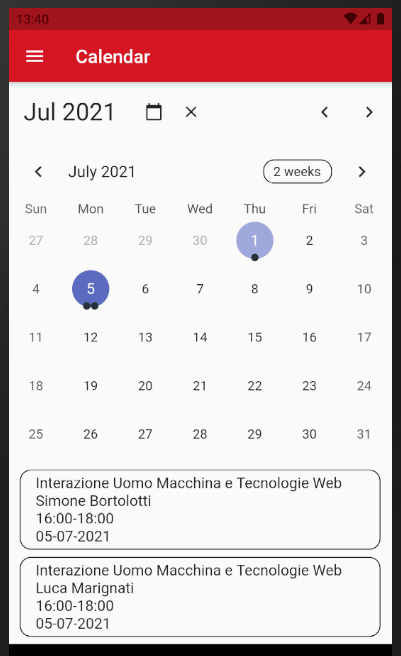
Descrizione generata automaticamenteIn questo Widget è possibile visualizzare la lista (ListView) delle lezioni che il tutor deve sostenere.

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.

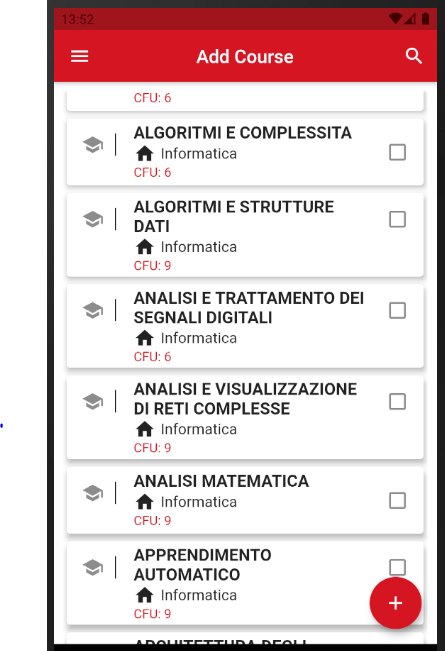
In aggiunta è possibile:

* 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**

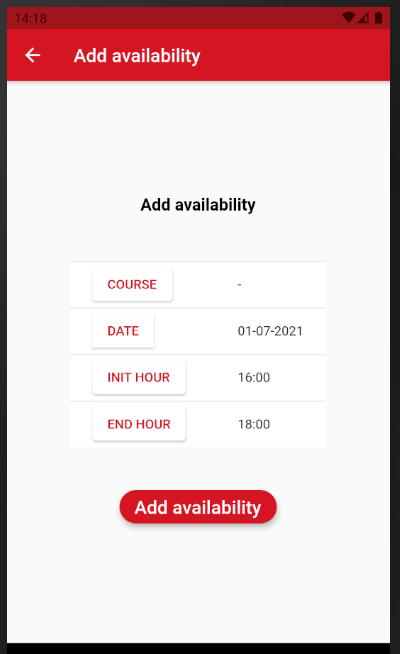
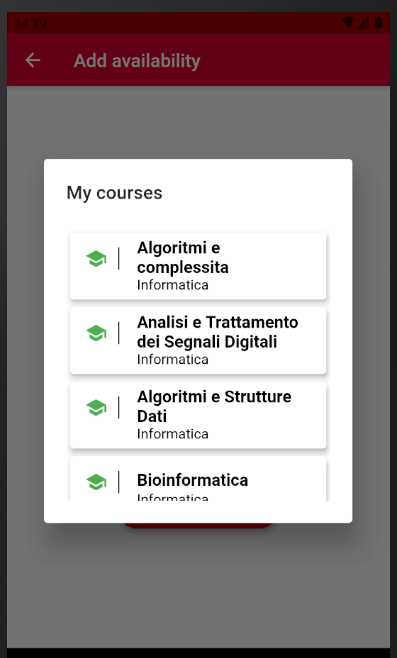
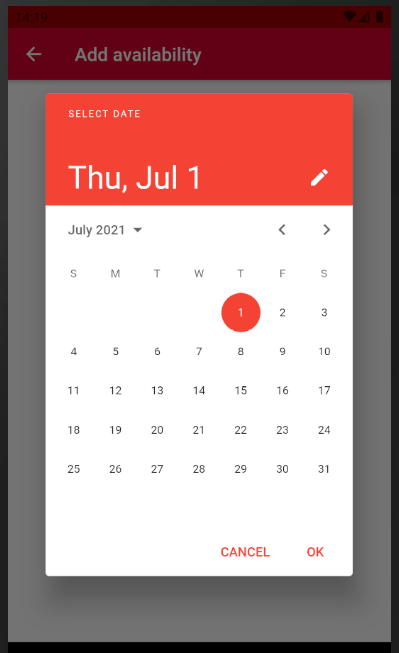
In questo Widget è possibile visualizzare la lista delle disponibilità del tutor.

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 |  |  |

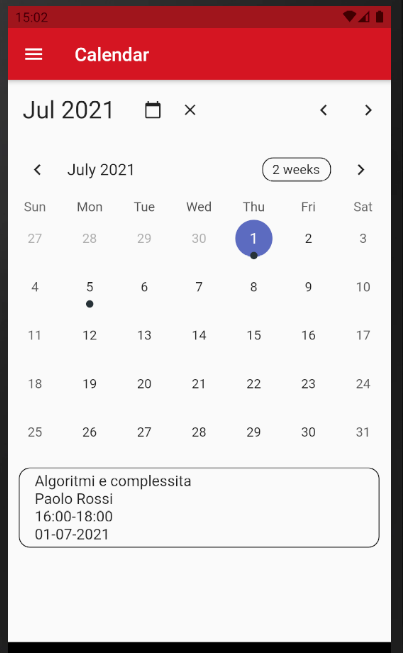
1. **Funzionalità dello Studente: My Reviews/Le mie recensioni**

Immagine che contiene testo

Descrizione generata automaticamente

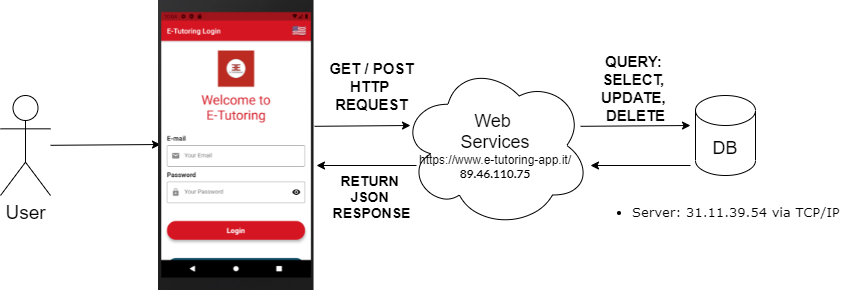
1. **Funzionalità dello Studente: Calendario**

Il Widget mostra le lezioni prenotate dallo studente.



# Architettura dell’app: struttura del codice realizzato e di altre risorse realizzate

**CLIENT 🡪 WEB SERVICES 🡪 DB MYSQL**



1. l’utente interagisce con l’applicazione tramite un client (dispositivo, applicazione IOS o Android);
2. i dati provengono dal back-end:
   1. il client effettua chiamate http (GET/POST method) richiamando i web services implementati;
   2. i WS effettuano query (select) oppure operazioni di write (update o delete) sui dati contenuti nel DB MySQL;
   3. i WS restituiscono, in formato JSON, i dati richiesti oppure l’esito dell’operazione richiesta (es. login, signup, delete profile, ecc.).

Per semplicità abbiamo provveduto a pubblicare in un dominio vero e proprio i WS sviluppati in modo da testare il reale funzionamento anche da dispositivo fisico e non solo mediante emulatore.

Dominio: <https://www.e-tutoring-app.it/>

89.46.110.75

Sicurezza: Basic Authentication

I dati esposti contengono delle informazioni riservate (es. email, indirizzo, ruolo dell’utente, ecc.) e, dunque, per accedere ai WS che comunicano con il database MySQL è stata predisposta una forma di autenticazione (**basic authentication**).



## Database MySql

Immagine che contiene testo

Descrizione generata automaticamente

* **user**: contiene le informazione utilizzate per il login all’applicazione (username, passowrd, email). La password, ovviamente, non viene salvata in chiaro ma viene mantenuta in forma cryptata (md5).

**Esempio**:



* **user\_attribute**: contiene gli attributi dell’utente (es. nome, cognome, numero di matricola, città di residenza ecc.).

**Chiavi esterne**:

* + **user\_id**: si riferisce all’utente, tramite l’identificativo (attributo id), della tabella user;
  + **role\_id**: si riferisce al ruolo dell’utente (Studente o Tutor) tramite l’identificativo role\_id della tabella role;
  + **degree\_path\_id**: si riferisce al curriculum scelto in fase di registrazione dell’utente. L’attributo in questione si riferisce all’attributo degree\_path\_id della tabella degree\_path (chiave esterna);
  + **degree\_id**: si rifersice al corso di laurea a cui l’utente è iscritto.

**Esempio**:



* **role**: contiene i ruoli che gli utenti possono assumere nell’applicativo;



* **course**: contiene le informazioni relative ai corsi (es. nome, CFU, tipo di insegnamento, SSD ecc.);



* **degree**: contiene le informazioni riguardante i corsi di laurea.



**Chiave esterna**:

* + **degree\_type\_id**: indica il tipo di corso di laurea (es. Triennale o Magistrale);
* **degree\_type**: indica il tipo di corso di laurea.



* **degree\_path**: contiene le informazioni riguardanti i curriculum.

Es. il percorso/curriculum “Realtà virtuale e Multimedialità”, tramite chiave esterna degree\_id, è associato al corso di laurea 2 (ovvero Corso di Laurea in Informatica Magistrale).

Immagine che contiene testo

Descrizione generata automaticamente

* **course\_path\_degree**: tabella ternaria che mette insieme le informazioni riguardanti i corsi, le lauree e i percorsi/curriculum.



Es. Analisi Matematica (course\_id = 1) è un corso relativo alla Laurea in Informatica (degree\_id = 1) del curriculum Informazione e conoscenza (degree\_path\_id = 6).

* **tutor\_course**: contiene le informazioni riguardanti i corsi insegnati dai tutor.

Es. il tutor con user\_id = 11 (ovvero Paolo Rossi) effettua tutoraggio del corso con course\_id = 3 (ovvero Agenti Intelligenti).

Immagine che contiene tavolo

Descrizione generata automaticamente

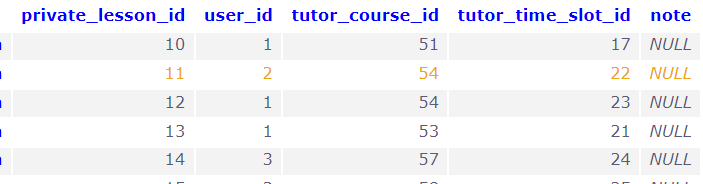
* **tutor\_time\_slot**: contiene le informazioni riguardanti le disponibiiltà dei tutor (giorni e orari). L’attributo user\_id è la chiave esterna che si riferisce all’utente (in questo caso il tutor).



* **review**: contiene le informazioni riguardanti le recensioni che gli utenti (identificati dall’attributo user\_id) effettuano sui tutor (identificati dall’attributo user\_tutor\_id).



* **private\_lesson**: contiene le informazioni sulle lezioni private degli studenti con riferimento al tutor e al corso prenotato.



* **notifications\_tutor**: contiene le notifiche ricevute dai tutor:
  + **private\_lesson\_id** si riferisce alla lezione privata recensita;
  + **date** si riferisce alla data in cui è generata la notifica;
  + **check** specifica se il tutor ha visto o meno la notifica (0 non vista, 1 vista).

Immagine che contiene tavolo

Descrizione generata automaticamente

## Web Service implementati

|  |  |
| --- | --- |
| ADD | |
| add\_private\_lesson.php | Aggiunge una lezione privata insegnato da un tutor. |
| add\_tutor\_course.php | Aggiunge un corso alla lista dei corsi insegnati da un tutor. |
| add\_tutor\_time\_slot.php | Aggiunge una disponibilità (time slot) ad un tutor. |
| add\_user\_review.php | Aggiunge una recensione di un utente (studente) ad un tutor. |
| user\_signup.php | Registrazione di un nuovo utente (studente o tutor) all’applicazione. |
| DELETE | |
| user\_delete.php | Elimina i dati dalle tabelle user e user\_attribute dell’utente richiesto. |
| delete\_tutor\_course.php | Elimina un corso (course\_id) dalla tabella tutor\_course (corsi insegnati da un tutor). |
| delete\_tutor\_time\_slot.php | Elimina una disponibilità (time\_slot) dalla tabella time\_slot. |
| UPDATE | |
| user\_edit.php | Aggiorna i dati dalla tabella user\_attribute dell’utente richiesto; |
| user\_change\_password.php | Prende in input e-mail e password, effettua l’encryption della password (md5) ed effettua l’update dei dati relativi all’utente identificato dall’email. |
| notifications\_tutor\_check.php | Setta l’attributo “check” a 1: indica che la notifica è stata vista dall’utente. |
| SELECT | |
| user\_login.php | Verifica se l’e-mail e la password esistono nella tabella user:   * se esistono, l’utente è abilitato e il login ha successo: restituisce “Login Matched”; * l’utente non è registrato nell’applicazione: restituisce “Invalid Username or Password Please Try Again”. |
| notifications\_tutor.php | Restituisce le notifiche relative alle prenotazioni degli studenti alle lezioni private offerte dal tutor.  <https://www.e-tutoring-app.it/ws/notifications_tutor.php?email=paolo.rossi@edu.unito.it> |
| get\_user\_role.php | Restituisce il ruolo dell’utente (email) passato come parametro al WS.  <https://www.e-tutoring-app.it/ws/get_user_role.php?email=paolo.rossi@edu.unito.it> |
| course\_list.php | Restituisce la lista di tutti i corsi.  [https://www.e-tutoring-app.it/ws/course\_list.php](https://www.e-tutoring-app.it/ws/course_list.php?course_id=1)  Restituisce il singolo corso richiesto.  <https://www.e-tutoring-app.it/ws/course_list.php?course_id=1> |
| course\_search\_private\_lesson.php | [https://www.e-tutoring-app.it/ws/course\_search\_private\_lesson.php? email=luca.marignati@edu.unito.it](https://www.e-tutoring-app.it/ws/course_search_private_lesson.php?%20email=luca.marignati@edu.unito.it)  Restituisce un array di lezioni private di uno specifico utente (identificato dall’email): es. Luca (user\_id = 1) ha ricevuto una lezione di tutoraggio di Agenti Intelligenti da Paolo (user\_tutor\_id = 11) |
| course\_search.php | [https://www.e-tutoring-app.it/ws/course\_search.php? email=luca.marignati@edu.unito.it](https://www.e-tutoring-app.it/ws/course_search.php?%20email=luca.marignati@edu.unito.it)  Restituisce un array di corsi a cui l’utente è iscritto (corsi relativi al percorso di Laurea in cui l’utente è iscritto). |
| course\_user\_list.php | Restituisce un array di corsi relativi al percorso di Laurea in cui l’utente è iscritto.  <https://www.e-tutoring-app.it/ws/course_user_list.php>?[email=luca.marignati@edu.unito.it](mailto:email=luca.marignati@edu.unito.it) |
| curriculum\_path\_by\_degree.php | [https://www.e-tutoring-app.it/ws/curriculum\_path\_by\_degree.php? degree\_name=informatica&degree\_type\_note=Laurea%20Triennale](https://www.e-tutoring-app.it/ws/curriculum_path_by_degree.php?%20degree_name=informatica&degree_type_note=Laurea%20Triennale)  Input:   * nome del corso di laurea (es. informatica); * tipo del corso di laurea (es. triennale o magistrale);   Output: array dei percorsi/curriculum disponibili per il corso di laurea richiesto. |
| degree\_list.php | Restituisce un array che contiene la lista dei Corsi di Laurea disponibili.  <https://www.e-tutoring-app.it/ws/degree_list.php> |
| degree\_path\_list.php | Array dei percorsi/curriculum dei corsi di laurea.  <https://www.e-tutoring-app.it/ws/degree_path_list.php>  Input: id del corso di laurea;  Output:  array dei percorsi/curriculum disponibili per il corso di laurea richiesto.  <https://www.e-tutoring-app.it/ws/degree_path_list.php?degree_id=1> |
| degree\_type\_list.php | Restituisce un array contenente le tipologie di lauree disponibili (triennale o magistrale).  <https://www.e-tutoring-app.it/ws/degree_type_list.php> |
| private\_lesson\_list.php | Restituisce un array contenente le lezioni private (tutoraggi) ricevute da un utente e sostenute da un tutor.  <https://www.e-tutoring-app.it/ws/private_lesson_list.php> |
| reviews\_list.php | Restituisce un array contenente le recensioni ricevute dal tutor passato come parametro (user\_tutor\_id).  <https://www.e-tutoring-app.it/ws/reviews_list.php?user_tutor_id=11> |
| role\_list.php | Restituisce un array contenente la lista dei ruoli.  <https://www.e-tutoring-app.it/ws/role_list.php> |
| tutor\_list.php | Restituisce un array contenente la lista dei tutor (ovvero gli utente aventi ruolo “tutor”).  <https://www.e-tutoring-app.it/ws/tutor_list.php>  Restituisce un json contenente i dati del singolo tutor passato come parametro (id).  <https://www.e-tutoring-app.it/ws/tutor_list.php?id=11>  Restituisce un json contenente i dati del singolo tutor passato come parametro (email).  [https://www.e-tutoring-app.it/ws/tutor\_list.php? email=paolo.rossi@edu.unito.it](https://www.e-tutoring-app.it/ws/tutor_list.php?%20email=paolo.rossi@edu.unito.it) |
| users\_list.php | Restituisce un array contenente la lista degli utenti.  <https://www.e-tutoring-app.it/ws/users_list.php>  Restituisce un json contenente i dati di un singolo utente.  <https://www.e-tutoring-app.it/ws/users_list.php?id=1>  Restituisce un json contenente i dati di un singolo utente.  [https://www.e-tutoring-app.it/ws/users\_list.php? email=luca.marignati@edu.unito.it](https://www.e-tutoring-app.it/ws/users_list.php?%20email=luca.marignati@edu.unito.it) |

## Model

Sono rappresentate le classi con i relativi campi corrispondenti a quelli restituiti dal WS (attributi delle tabelle del DB).

Immagine che contiene testo

Descrizione generata automaticamente

Es. couseModel.dart

Immagine che contiene testo

Descrizione generata automaticamente

## Paradigma MVC

Abbiamo sviluppato l’applicazione adottando il paradigma MVC:

* **Model**: modello dei dati;
* **Controller**:
  + si occupa di effettuare la chiamata http andando a chiamare i WS;
  + mette a disposizione i dati al Widget (View);
* **View**: si occupa di rappresentare i dati forniti dal controller.

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

Esempio:

* view: login.dart
  + TextFormField email
  + TextFormField password
  + ElevatedButton login
  + ElevatedButton signup
* Quando l’utente clicca (onClicked) sul bottone di login si scatena l’evento chiamando la funzione “**userLogin()**” che gestisce la logica della funzionalità:
  + entra in gioco il **login\_controllerWS.dart** che fornisce la funzione “login()”: si occupa di chiamare il WS user\_login.php e ritorna true o false in caso di successo o fallimento.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

# 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**

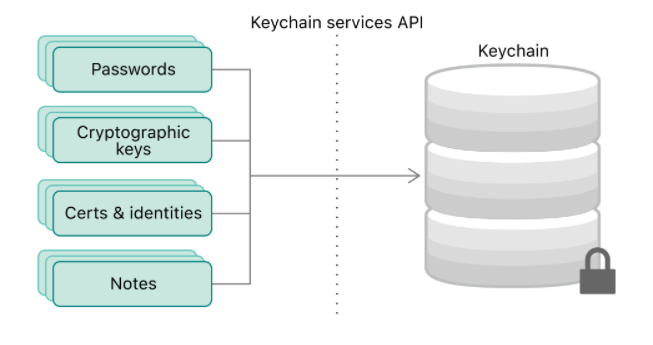
Un plug-in Flutter per utilizzare l'API Firebase Cloud Messaging.

<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.** |

# Flutter e supporto per device multipli

L’applicazione è stata sviluppando mediante l’utilizzo di Flutter e del linguaggio Dart fruttandone i numerosi vantaggi che forniscono:

* **cross-platform e indipendenza dal sistema operativo** (un codice, più piattaforme): le applicazioni sviluppate in Flutter implementano uno strato intermedio definito Bridge che permette di comunicare con le API native dei vari dispositivi (Android e iOS). È, dunque, possibile scrivere il codice una sola volta e non differenziare per le diverse tipologie di SO;
* **performance simili a quelle native**: nonostante la presenza dello strato di Bridge che comunica con le API native, le performance non vengono impattante negativamente e sono simili a quelle delle App native;
* **hot reloading – impatto positivo sulle tempistiche di sviluppo**: la funzione hot reload aiuta a sperimentare rapidamente e facilmente, creare interfacce utente, aggiungere funzionalità e correggere bug più velocemente. Stimola il processo di sviluppo fornendo a uno sviluppatore un record del codice sorgente direttamente all’interno dell’applicazione funzionante. Aiuta a riflettere sulle modifiche apportate al codice in meno di 2-3 secondi e senza ripristinare lo stato dell’applicazione. Pertanto, Hot Reload consente di monitorare rapidamente lo sviluppo dell’applicazione. A seconda della complessità e della natura del progetto, è possibile risparmiare dal 20 al 50% in termini di tempo, che può essere speso per lo sviluppo di altre funzioni utili.
* **semplicità nell’implementazione della logica**: Flutter fornisce funzionalità del sistema operativo avanzate come coordinate GPS, raccolta dei dati dei sensori, gestione delle autorizzazioni, Bluetooth, credenziali e altre funzionalità in **plug-in pronti per l’uso supportati da Google**. Se la app che progettate di sviluppare fa affidamento su una funzionalità a livello di sistema operativo non disponibile come plug-in, Flutter può stabilire la comunicazione tra il suo linguaggio di programmazione Dart e il codice nativo utilizzando i canali della piattaforma.
* **riduzione dei costi di sviluppo**: con Flutter si riducono notevolmente i costi di sviluppo dell’app, visto che si utilizza lo stesso set di librerie per l’interfaccia utente e il medesimo framework. Flutter è in grado di gestire ogni singolo pixel del display, in questo modo si raggiungeranno altissimi livelli di personalizzazione e si potranno sviluppare interfacce utente con grafiche innovative e dettagli visivi di altissima qualità;
* **framework open source**: tempo si è creata una vera e propria community di sviluppatori, che pubblicano esempi di codice e che aiutano e supportano gli utenti nella creazione di nuove app cross-platform.

# 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

# Test effettuati

Abbiamo sviluppato i test suddividendo in cartelle rispettando la struttura del codice in modo da avere un’esatta corrispondenza tra implementazione e test:

* nella cartella di “*test/controller*” abbiamo testato le funzioni relative a “*lib/controller*” (in particolare le chiamate http);
* nella cartella “*test/screens*” abbiamo testato le funzioni relative a “*lib/screen*”;
* nella cartella “test/utils” abbiamo testato le funzioni relative a “*lib/utiils*”;
* nella cartella “*test/widgets*” abbiamo testato le funzioni relative a “*lib/widgets*”.

Immagine che contiene testo

Descrizione generata automaticamente

## test/controller - Mockito

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

<https://flutter.dev/docs/cookbook/testing/unit/mocking>

Per ogni metodo del controller abbiamo implementato un test contenente 2 casi:

* risposta HTTP con status 200: operazione va a buon fine;
* risposta HTTP con status 404 (NOT FOUND): operazione fallita;

Mostriamo di seguito un esempio di test spiegando i passi implementativi del test preso in esame: per esempio, vogliamo testare il metodo getCurriculumListFromWS che va chiama il WS specifico per fornire la lista dei curriculum di un corso di laurea (es. informatica, laurea triennale).

Immagine che contiene testo

Descrizione generata automaticamente

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

Immagine che contiene testo

Descrizione generata automaticamente

TEST: **inizializzazione del MockClient()** che permette di simulare il comportamento di un client http (browser);

1. **definizione dei parametri richiesti** **dal WS**: in questo caso *degree\_name* e *degree\_type\_note*;
2. simulazione della chiamata http utilizzando il mock client: abbiamo simulato la chiamata http;
3. trattandosi di una chiamata fittizia abbiamo dovuto definito la risposta fornita dal WS in formato JSON: *'[{"degree\_path\_name": "Immagini, Visione e Realtà Virtuale"}]'* e definito lo status (es. 200 o 404);
4. abbiamo chiamato la funzione implementata (*getCurriculumFromWS*);
5. infine, abbiamo eseguito i vari test (es. test sul tipo ritorno) in modo da verificare che ci aspettiamo come risposta sia quello ottenuto dalla funzione e dal WS.

Per esempio, ci aspettiamo che la funzione *getCurriculumFromWS* restituisca una lista di oggetti di tipo CurriculumModel.

Di seguito mostriamo il codice relativo al test.

Immagine che contiene testo

Descrizione generata automaticamente

Gli altri test di questo tipo sono del tutto similari.

Per i dettagli rimandiamo al codice.

## test/screens

Questi test riguardano gli screens ChangePassword, Login e Signup.

Come si nota dalle figure sotto, oltre ai test relative alle chiamate http (mockito), abbiamo implementato degli UNIT TEST:

* per esempio, abbiamo testato delle funzioni a livello unitario che riguardano la validazione dell’e-mail inserita dall’utente;
* abbiamo testato le funzioni che lavorano sui tipi di date. Per esempio la funzione formatDate prende in input una stringa, effettua il parsing in un oggetto di tipo DateTime ed rielabora la data nel formato dd-mm-aaa;

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Mockito

Es. testing del metodo di login (WS: user\_login.php). Nel test riportato ci aspettiamo che il metodo di login restituisca il booleano “false” in quanto username ([simone.bortolotti@edu.unito.it](mailto:simone.bortolotti@edu.unito.it)) e password (“errore”) non sono corrette, ovvero non esiste un match nel DB (nella tabella “user”);

Immagine che contiene testo

Descrizione generata automaticamente