**Relazione Progetto Programmazione per Dispositivi Mobili**

E-Tutoring

**Gruppo**

Bortolotti Simone

Davide De Cenzo

Marignati Luca

Sommario

[Tecnologia: scelta implementativa 4](#_Toc75698549)

[Filesystem 5](#_Toc75698550)

[Pubspec.yaml 6](#_Toc75698551)

[Back-end: Database MySQL 8](#_Toc75698552)

[Accesso 8](#_Toc75698553)

[Tabelle 8](#_Toc75698554)

[Vista del DB: tabelle e attributi 11](#_Toc75698555)

[Backend-ws 11](#_Toc75698556)

[Web Service implementati 12](#_Toc75698557)

[ADD 13](#_Toc75698558)

[DELETE 14](#_Toc75698559)

[UPDATE 15](#_Toc75698560)

[LOGIN AND SIGN UP 16](#_Toc75698561)

[GET LIST 17](#_Toc75698562)

[Interazione con il back-end 25](#_Toc75698563)

[Front-end: Flutter Application 25](#_Toc75698564)

[config/config.dart 26](#_Toc75698565)

[constants/Theme.dart 26](#_Toc75698566)

[controller/controllerWS.dart 27](#_Toc75698567)

[Logica del Future 27](#_Toc75698568)

[Lingue supportate: I10n 29](#_Toc75698569)

[Model 30](#_Toc75698570)

[couseModel.dart 30](#_Toc75698571)

[curriculumModel.dart 31](#_Toc75698572)

[MVC (Model – View – Controller) 31](#_Toc75698573)

[Screens descrizione dei widget principali 32](#_Toc75698574)

[login.dart 32](#_Toc75698575)

[Package flutter\_secure\_storage 34](#_Toc75698576)

[Classe UserSecureStorage 34](#_Toc75698577)

[profile.dart 36](#_Toc75698578)

[ACL (Accesso Control List) 37](#_Toc75698579)

[profile.dart e gestione notifiche 39](#_Toc75698580)

[calendar-student.dart/calendar-tutor.dart 42](#_Toc75698581)

[course.dart 43](#_Toc75698582)

[TESTING 44](#_Toc75698583)

# Tecnologia: scelta implementativa

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.

# Filesystem



Come si nota, il filesystem rispecchia la struttura in un progetto sviluppato mediante Flutter. Il codice dell’applicazione è stato sviluppato in linguaggio di programmazione *Dart* e si trova nella cartella *lib*.

I file/cartelle principali sono i seguenti:

* cartella *assets*: contiene i file multimediali, font (OpenSans) e immagini (es. logo) utilizzati dall’applicazione;
* cartella *backend-ws*: contiene i Web Services che comunicano con il DB:
  + sono stati sviluppati mediante linguaggio PHP e comunicano con un database MySql;
  + la logica dei WS è quella tipica: l’applicazione effettua una chiamata http (metodo GET o POST) in cui specifica eventuali parametri (es. username e password);
  + il relativo WS chiamato effettua una query interrogando il DB e fornendo, in output, in formato JSON la risposta all’applicazione (es. lista di tutti i corsi, lista di tutti i tutor);
  + per i dettagli visionare la sezione che ne descrive in maniera più approfondita la struttura;
* cartella *lib*: come detto, contiene il codice dell’applicazione vera e propria (in linguaggio Dart);
* file *pubspec.yaml*: specifica le dipendenze richieste dal progetto, ad esempio pacchetti particolari (e relative versioni), caratteri o file di immagine. Specifica anche altri requisiti, come le dipendenze dai pacchetti per sviluppatori (come pacchetti di test o mocking) o vincoli particolari sulla versione di Flutter SDK.

# Pubspec.yaml

Specifica le dipendenze richieste dal progetto:

* flutter
* **flutter\_localizations**

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

Per impostazione predefinita, Flutter fornisce solo localizzazioni in inglese americano. Per aggiungere il supporto per altre lingue, un'applicazione deve specificare proprietà MaterialApp aggiuntive e includere un pacchetto chiamato flutter\_localizations. A partire da novembre 2020, questo pacchetto supporta 78 lingue.

* **intl**: ^0.17.0

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

Fornisce funzionalità di internazionalizzazione e localizzazione, inclusa la traduzione dei messaggi, i plurali e i generi, la formattazione e l'analisi di data/numero e il testo bidirezionale.

* font\_awesome\_flutter: ^8.8.1

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

Il pacchetto di icone Font Awesome disponibile come set di icone Flutter.

* scrollable\_positioned\_list: ^0.1.7

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

Elemento di View che consente di scorrere fino a un elemento specifico nell'elenco e di determinare quali elementi sono attualmente visibili.

Un ScrollablePositionedList funziona in modo molto simile alla versione builder di ListView, tranne per il fatto che l'elenco può essere fatto scorrere a un elemento specifico.

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

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

È un plug-in Flutter per archiviare i dati in un archivio sicuro:

* + per iOS viene utilizzato 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.

* move\_to\_background: ^1.0.2

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

Plugin Flutter per l'invio di applicazioni mobili in background. Supporta iOS e Android.

* mockito: ^5.0.9

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

* email\_validator: ^2.0.1

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

Plug-in per convalidare gli indirizzi e-mail senza utilizzare RegEx. Può essere utilizzato anche per convalidare le e-mail all'interno delle app Flutter

* table\_calendar: ^3.0.0

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

Widget calendario altamente personalizzabile e ricco di funzionalità per Flutter.

* provider: ^4.3.3

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

* **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

# Back-end: Database MySQL

Accesso:

Server: 31.11.39.54 via TCP/IP

mysql.aruba.it

Username: Sql1558195

Password: \*\*\*\*\*\*\*\*\*\*\*

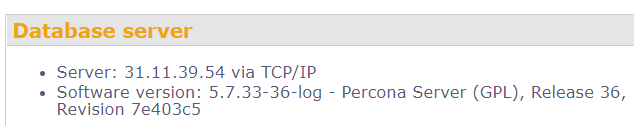


Tabelle:

* **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.



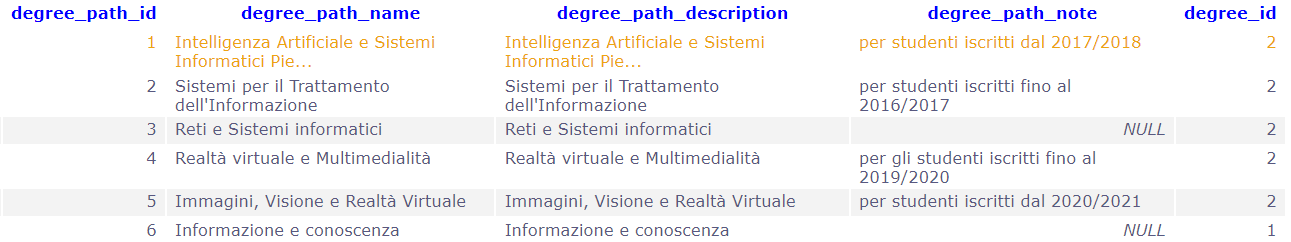
**Chiave esterna**:

* + **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).



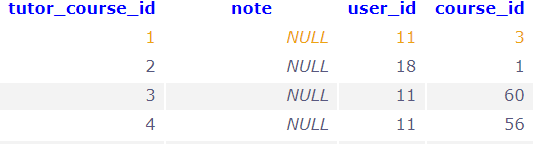
* **course\_path\_degree**: tabella ternaria che mette insieme le informazioni riguardanti i corsi, le lauree e i percorsi/curriculum.



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).



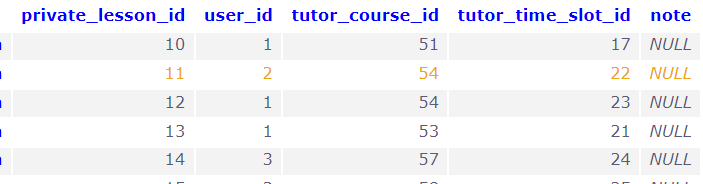
* **tutor\_time\_slot**: contiene le informazioni riguardanti le disponibiiltà dei tutor (giorni e orari). L’attributo user\_id è la chiave esterna che si riferisce all’utente (in questo caso il tutor).



* **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.

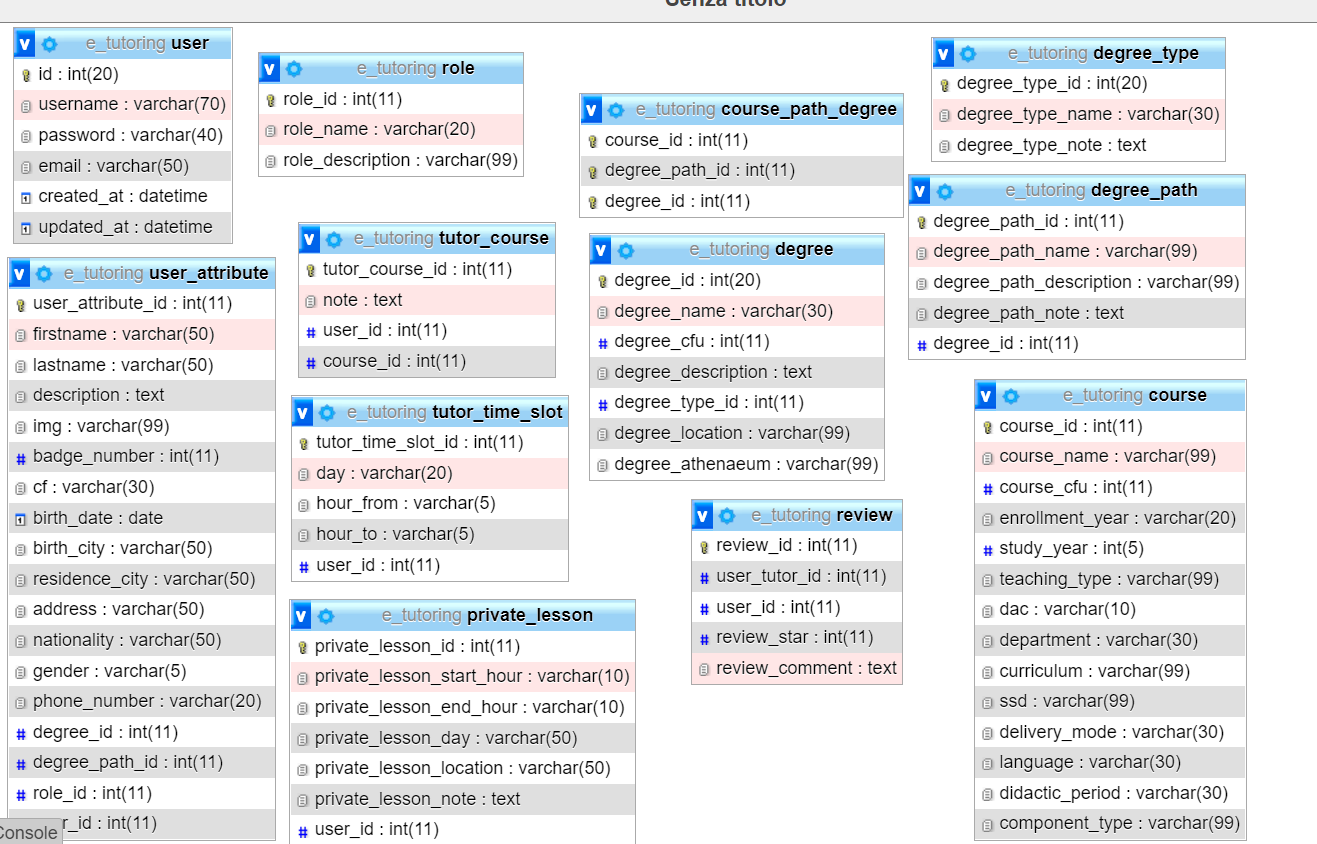


* **notifications\_tutor**: contiene le notifiche ricevute dai tutor:
  + **private\_lesson\_id** si riferisce alla lezione privata recensita;
  + **date** si riferisce alla data in cui è generata la notifica;
  + **check** specifica se il tutor ha visto o meno la notifica (0 non vista, 1 vista).

Immagine che contiene tavolo

Descrizione generata automaticamente

## Vista del DB: tabelle e attributi



# Backend-ws

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**: i dati esposti contengono delle informazioni riservate e, dunque, per accedere ai WS che comunicano con il database MySQL è stata predisposta una forma di autenticazione (**basic authentication**).



## Web Service implementati

Immagine che contiene testo

Descrizione generata automaticamente

### ADD

* add\_private\_lesson.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email  course\_id  tutor\_time\_slot\_id  tutor\_id | Aggiunge una lezione privata insegnato da un tutor.  Aggiorna l’informazione riguardante il timeslot selezionato dall’utente (diventa prenotato/reserved = 1).  Inserisce la nuova lezione private nella tabella relativa alle notifiche. | INSERT INTO private\_lesson (user\_id, tutor\_course\_id, tutor\_time\_slot\_id) VALUES ($user\_id, $tutor\_course\_id, $tutor\_time\_slot\_id);  UPDATE tutor\_time\_slot SET reserved=1 WHERE tutor\_time\_slot\_id='$tutor\_time\_slot\_id'"  INSERT INTO notifications\_tutor (private\_lesson\_id) VALUES ($last\_id) |

* add\_tutor\_course.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email  course\_id | Aggiunge un corso alla lista dei corsi insegnati da un tutor. | INSERT INTO tutor\_course (user\_id, course\_id) VALUES ('$user\_id', '$course\_id') |

* add\_tutor\_time\_slot.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email  couse\_id  day  hour\_from  hour\_to | Aggiunge una disponibilità (time slot) ad un tutor. | INSERT INTO tutor\_time\_slot (course\_id, day, hour\_from, hour\_to, user\_id) VALUES ('$course\_id', '$day', '$hour\_from', '$hour\_to', $user\_id) |

* add\_user\_review.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email  user\_tutor\_id  review\_star  review\_comment | Aggiunge una recensione di un utente (studente) ad un tutor | INSERT INTO review (user\_tutor\_id, user\_id, review\_star, review\_comment) VALUES ('$user\_tutor\_id', '$user\_id', '$review\_star', '$review\_comment') |

### DELETE

* user\_delete.php

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email | Elimina i dati dalle tabelle user e user\_attribute dell’utente richiesto. | SELECT id FROM `user` WHERE email = '" . $email ."'"  DELETE FROM user WHERE email='" . $email ."'  DELETE FROM user\_attribute WHERE user\_id='" . $user\_id ."'" |

* delete\_tutor\_course.php

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email  course\_id | Elimina un corso (course\_id) dalla tabella tutor\_course (corsi insegnati da un tutor) | DELETE FROM tutor\_course WHERE user\_id = '$user\_id' AND course\_id =' $course\_id'" |

* delete\_tutor\_time\_slot.php

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email  time\_slot\_id | Elimina una disponibilità (time\_slot) dalla tabella time\_slot | DELETE FROM tutor\_time\_slot WHERE user\_id = '$user\_id' AND tutor\_time\_slot\_id =' $time\_slot\_id' |

### UPDATE

* user\_edit.php

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email  firstname  lastname  birth\_date  birth\_city  residence\_city  … | Aggiorna i dati dalla tabella user\_attribute dell’utente richiesto (email); | UPDATE user\_attribute JOIN user ON user\_attribute.user\_id = user.id SET firstname = '$firstname', lastname = '$lastname', birth\_date='$birth\_date',  birth\_city ='$birth\_city', residence\_city = '$residence\_city',  nationality = '$nationality', phone\_number = '$phone\_number', description = '$description' WHERE email = '$email'" |

* user\_change\_password.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email  password | Prende in input email e password, effettua l’encryption della password (md5) ed effettua l’update dei dati relativi all’utente identificato dall’email.  Restituisce “Password updated successfully” o “Error updating record” in caso di errore. | UPDATE user SET password = '$hashed\_password' WHERE email = '$email' |

* notifications\_tutor\_check.php

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| notifications\_tutor\_id | Setta l’attributo “check” a 1: indica che la notifica è stata vista dall’utente. | UPDATE notifications\_tutor SET `check` = 1 WHERE notifications\_tutor\_id = $id |

### LOGIN AND SIGN UP

* user\_login.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email  password | Verifica se l’email 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”. | select \* from user where  email = '$email' and password = '$hashed\_password' |

* user\_signup.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| POST  email  password  degree\_name  degree\_type  curriculum  role | Registrazione di un nuovo utente (studente o tutor) all’applicazione. | INSERT INTO user\_attribute (degree\_id, degree\_path\_id, role\_id, user\_id) VALUES ($degree\_id, $degree\_path\_id, $role\_id, $last\_id) |

### GET LIST

* notifications\_tutor.php

<https://www.e-tutoring-app.it/ws/notifications_tutor.php?email=paolo.rossi@edu.unito.it>

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET  email | Restituisce le notifiche relative alle prenotazione degli studenti alle lezioni private offerte dal tutor | … |

* get\_user\_role.php

<https://www.e-tutoring-app.it/ws/get_user_role.php?email=paolo.rossi@edu.unito.it>

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET  email | Restituisce il ruolo dell’utente (email) passato come parametro al WS | SELECT role.\* FROM user left join user\_attribute on user.id = user\_attribute.user\_id  left join role on user\_attribute.role\_id = role.role\_id  where email = '" . $\_GET['email'] ."' |

* course\_list.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET | Restituisce la lista di tutti i corsi | <https://www.e-tutoring-app.it/ws/course_list.php>  SELECT DISTINCT \* FROM course |
| GET  course\_id | Restituisce il singolo corso richiesto | <https://www.e-tutoring-app.it/ws/course_list.php?course_id=1>  SELECT DISTINCT \* FROM course where course\_id = " . $\_GET['course\_id'] |



* course\_search\_private\_lesson.php:

|  |  |
| --- | --- |
| **Metodo http e parametri** | **Descrizione** |
| GET  email | [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) |
| GET  email  query | <https://www.e-tutoring-app.it/ws/course_search_private_lesson.php>? [email=luca.marignati@edu.unito.it&query=agenti](mailto:email=luca.marignati@edu.unito.it&query=agenti)  FUNZIONALITÀ DI RICERCA in base al nome del corso  Query WHERE:  where email = ‘" . $\_GET['email'] ."' AND `course\_name` LIKE '%".$query."%' |



* course\_search.php:

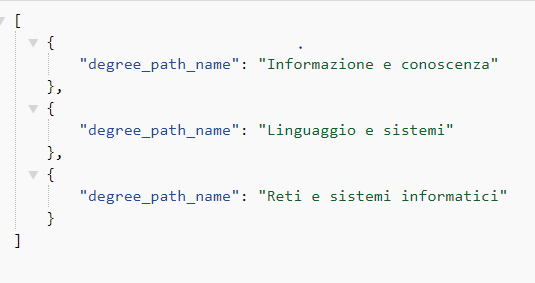
|  |  |
| --- | --- |
| **Metodo http e parametri** | **Descrizione** |
| GET  email | [https://www.e-tutoring-app.it/ws/course\_search.php? email=luca.marignati@edu.unito.it](https://www.e-tutoring-app.it/ws/course_search.php?%20email=luca.marignati@edu.unito.it)  Restituisce un array di corsi a cui l’utente è iscritto (corsi relativi al percorso di Laurea in cui l’utente è iscritto). |
| GET  email  query | FUNZIONALITÀ DI RICERCA in base a course\_name |

* course\_user\_list.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET  email | 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:

|  |  |
| --- | --- |
| **Metodo http e parametri** | **Descrizione** |
| GET  degree\_name  degree\_type\_note | [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:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET | Restituisce un array che contiene la lista dei Corsi di Laurea disponibili. | <https://www.e-tutoring-app.it/ws/degree_list.php>  SELECT \* FROM degree  join degree\_type on degree.degree\_type\_id = degree\_type.degree\_type\_id  ORDER BY degree\_name |



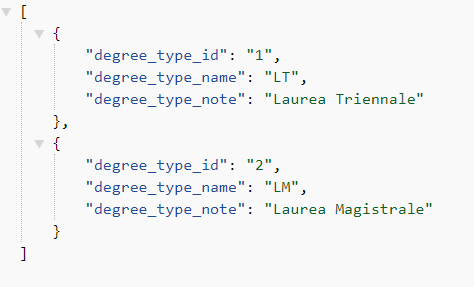
* degree\_path\_list.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET | Array dei percorsi/curriculum dei corsi di laurea. | <https://www.e-tutoring-app.it/ws/degree_path_list.php>  SELECT \* FROM degree\_path join degree on degree.degree\_id = degree\_path.degree\_id |
| GET  degree\_id | 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>  SELECT \* FROM degree\_path join degree on degree.degree\_id = degree\_path.degree\_id  where degree.degree\_id =". $\_GET['degree\_id'] |



* degree\_type\_list.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET | Restituisce un array contenente le tipologie di lauree disponibili (triennale o magistrale) | <https://www.e-tutoring-app.it/ws/degree_type_list.php>  SELECT \* FROM degree\_type |



* private\_lesson\_list.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET | 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>  SELECT \* FROM private\_lesson join user on user.id=private\_lesson.user\_id  left join course on private\_lesson.course\_id = course.course\_id |

* reviews\_list.php:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET  user\_tutor\_id | 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:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET | Restituisce un array contenente la lista dei ruoli. | <https://www.e-tutoring-app.it/ws/role_list.php> |



* tutor\_list.php:

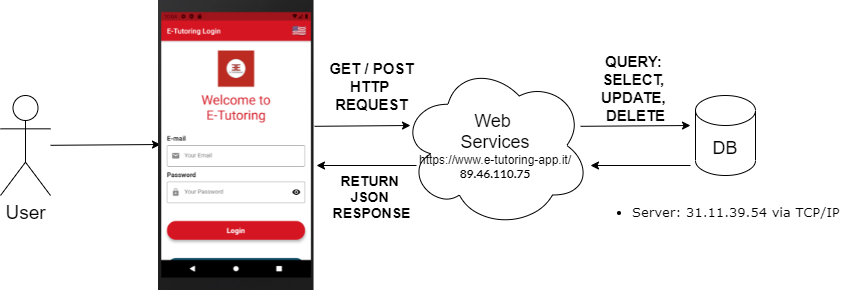
|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET | Restituisce un array contenente la lista dei tutor (ovvero gli utente aventi ruolo “tutor”) | <https://www.e-tutoring-app.it/ws/tutor_list.php>  SELECT \* FROM user join user\_attribute on user.id = user\_attribute.user\_id  left join role on user\_attribute.role\_id = role.role\_id where role.role\_id = 2 |
| GET  id | 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>  SELECT \* FROM user join user\_attribute on user.id = user\_attribute.user\_id  left join role on user\_attribute.role\_id = role.role\_id  where role.role\_id = 2 AND id = " . $\_GET['id'] |
| GET  email | 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:

|  |  |  |
| --- | --- | --- |
| **Metodo http e parametri** | **Descrizione** | **Esempio e Query SQL** |
| GET | Restituisce un array contenente la lista degli utenti | <https://www.e-tutoring-app.it/ws/users_list.php>  SELECT \* FROM user left join user\_attribute on user.id = user\_attribute.user\_id  left join role on user\_attribute.role\_id = role.role\_id  left join degree on user\_attribute.degree\_id = degree.degree\_id  left join degree\_path on user\_attribute.degree\_path\_id = degree\_path.degree\_path\_id  left join degree\_type on degree.degree\_type\_id = degree\_type.degree\_type\_id |
| GET  id | Restituisce un json contenente i dati di un singolo utente | <https://www.e-tutoring-app.it/ws/users_list.php?id=1> |
| GET  email | 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) |



# Interazione con il back-end

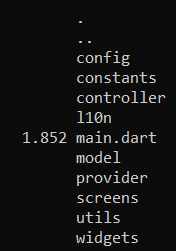


1. l’utente interagisce con l’applicazione tramite un dispositivo;
2. i dati provengono dal back-end:
   1. il client effettua chiamate http (GET/POST method) richiamando i web services implementati;
   2. i WS effettuano query (select) oppure operazioni di write (update o delete) sui dati contenuti 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.).

# Front-end: Flutter Application

Dopo aver descritto in maniera dettagliata sul back-end, ora analizziamo lo sviluppo dell’app lato front-end.

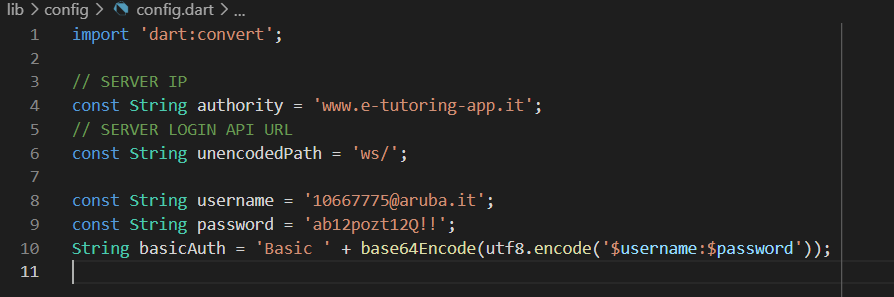
Come detto, trattandosi di un’applicazione sviluppata in Flutter, il software sviluppato è contenuto nella cartella *lib* che ha la seguente struttura:



## config/config.dart

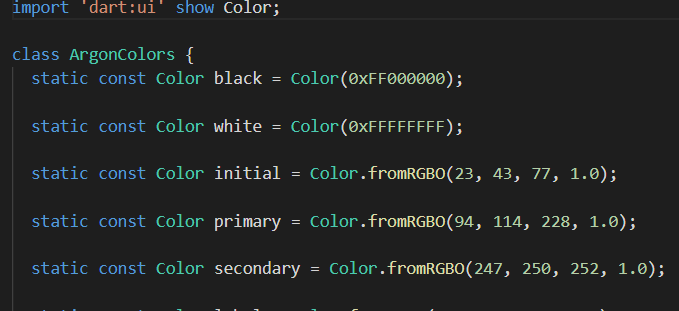
Contiene variabili globali che vengono utilizzate in tutta l’applicazione:

* authority e unencodedPath per l’accesso ai WS;
* username e password per la basic authetication.



## constants/Theme.dart

classe ArgonColors che definisce i colori del tema utilizzato.



## controller/controllerWS.dart

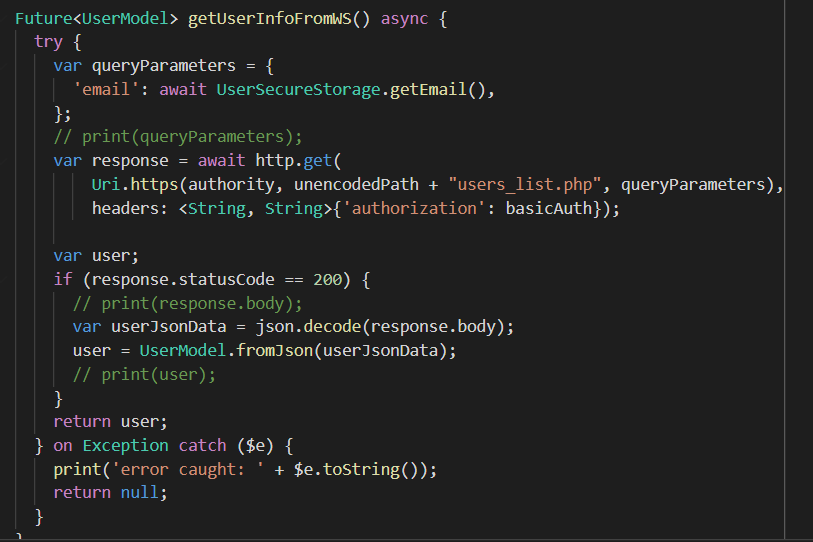
Contiene i metodi che effettuano le chiamate http verso i WS. Per poter comunicare con i Web Services si ha bisogno di un client http: il pacchetto “**https://pub.dev/packages/http”** contiene un insieme di funzioni e classi di alto livello che semplificano l'utilizzo delle risorse HTTP. È multipiattaforma e supporta dispositivi mobili, desktop e browser.

***import 'package:http/http.dart' as http;***

### Logica del Future

Un oggetto Future (futuro) rappresenta il risultato di un'operazione asincrona e può avere due stati: completato con successo o fallito.

Esempio:



* **getUserInfoFromWS**: come si nota nel codice riportato sopra, questo metodo:
  + recupera dallo storage (KeyStore) l’e-mail dell’utente loggato;
  + effettua una chiamata http di tipo GET passando come parametro l’e-mail precedentemente recuperata dallo storage;
  + richiama il WS users\_list.php;
  + nell’headers specifica i parametri per la basic authentication.

In caso di risposta con statusCode = 200 (Successo), viene istanziato un oggetto di tipo UserModel settandolo con i campi (JSON) ricevuti come risposta dal WS e viene restituito come ritorno della funzione.

**Altri metodi**

|  |
| --- |
| getUserCourseSearchFromWS |
| getAllCourseFromWS |
| getDegreeListFromWS |
| getCurriculumListFromWS |
| getRoleListFromWS |
| getCourseDetailFromWS |
| getTutorSearchFromWS |
| getTutorDetailFromWS |
| getReviewFromWS |
| getReviewTutorFromWS |
| getReviewsUserFromWS |
| getRoleFromWS |
| getPrivateLessonFromWS |
| getTutorLessonFromWS |
| getTutorLessonTodayFromWS |
| getTutorLessonTodayFromWS |
| getNotificationsTutorFromWS |
| notificationsUpdateCheck |
| getPrivateLessonTodayFromWS |
| La logica di tutti questi metodi è simile a quanto detto per la funzione **getUserInfoFromWS.**  Per il dettaglio dei singoli metodi si rimanda al codice. |

## Lingue supportate: I10n

Cartella contenente i file di traduzioni: CHIAVE-VALORE.

Lingue supportate: INGLESE e ITALIANO.

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

Formato ARB: le risorse sono codificate come oggetti JSON.

USO:

Immagine che contiene testo

Descrizione generata automaticamente

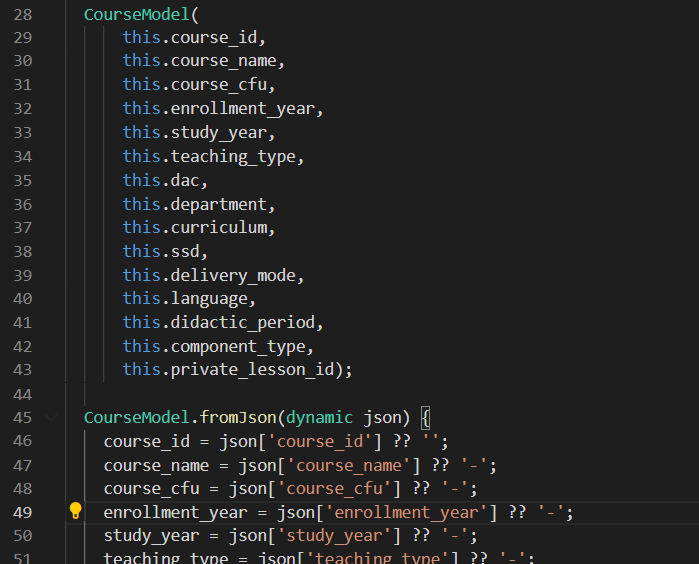
# Model

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

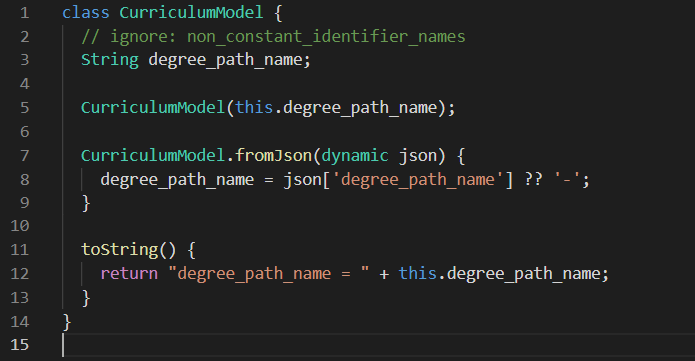
Immagine che contiene testo

Descrizione generata automaticamente

### couseModel.dart



### curriculumModel.dart



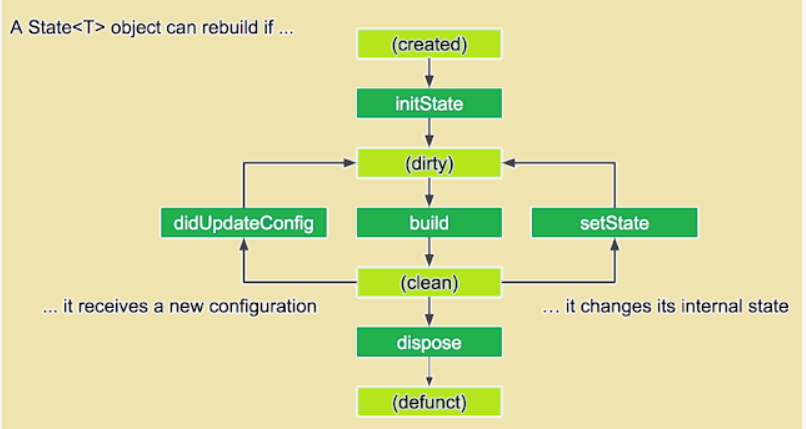
## MVC (Model – View – Controller)

Abbiamo strutturato il codice seguendo il pattern MVC:

|  |  |  |  |
| --- | --- | --- | --- |
| **VIEW** | **CONTROLLER** | **MODEL** | **WS** |
| login.dart | userLogin() |  | user\_login.php |
| signup.dart | userSignup() |  | user\_singup.php |
| profile.dart | controllerWS.dart  metodo:  getUserInfoFromWS() | UserModel | users\_list.php |
| change-password.dart | userChangePassword() |  | user\_change\_password  .php |
| user-edit.dart | controllerWS.dart  metodo:  getUserInfoFromWS()  userEdit() | UserModel | user\_edit.php |
| tutor.dart | controllerWS.dart  metodo:  getTutorSearchFromWS | TutorModel | tutor\_list.php |
| tutorDetail.dart |  | TutorModel |  |
| course.dart | controllerWS.dart  metodo:  getUserCourseSearch  FromWS() | CourseModel | course\_search.php |
| courseModel.dart |  | CourseModel |  |
| reviews.dart | controllerWS.dart  metodo:  getReviewFromWS() | ReviewModel | reviews\_list.php |

## Screens descrizione dei widget principali

Ciclo di vita: INIT STATE 🡪 BUILD 🡪 DISPOSE



### login.dart

#### View

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

#### Widget

Login è uno StatefulWidget in quanto il contenuto è dinamico: l’utente interagisce con il widget.

Uno StatefulWidget può cambiare aspetto in risposta a eventi innescati dalle interazioni dell'utente o quando riceve dati.

Checkbox, Radio, Slider, InkWell, Form e TextField sono esempi di widget stateful.

Lo stato di un widget è memorizzato in un oggetto State. Lo stato è costituito da valori che possono cambiare. Quando lo stato del widget cambia, l'oggetto di stato chiama setState(), dicendo al framework di ridisegnare il widget.

#### Implementazione metodo di login (userLogin)

1. si tratta di un metodo asincrono che restituisce un oggetto di tipo Future in quanto viene effettuata una chiamata al WS per verificare che l’email e la password inserite corrispondano ad un match nella tabella user del DB;
2. setState visible true 🡪 trattandosi di uno Stateful Widget viene modificato lo stato del Widget e viene ridisegnato: mostrato l’indicatore di progressione circolare (rotellina che gira): significa che si è in attesa della risposta del WS;
3. chiamata http
   1. metodo POST;
   2. WS: user\_login.php (vedi sezione riguardante i WS);
   3. negli headers vengono passati i parametri per la basic authentication;
   4. il body della richiesta http contiene un json formato da { email : …, password: …} in modo che il WS possa leggere ed effettuare la query di selezione per verificare se esiste un match dei dati nel DB;

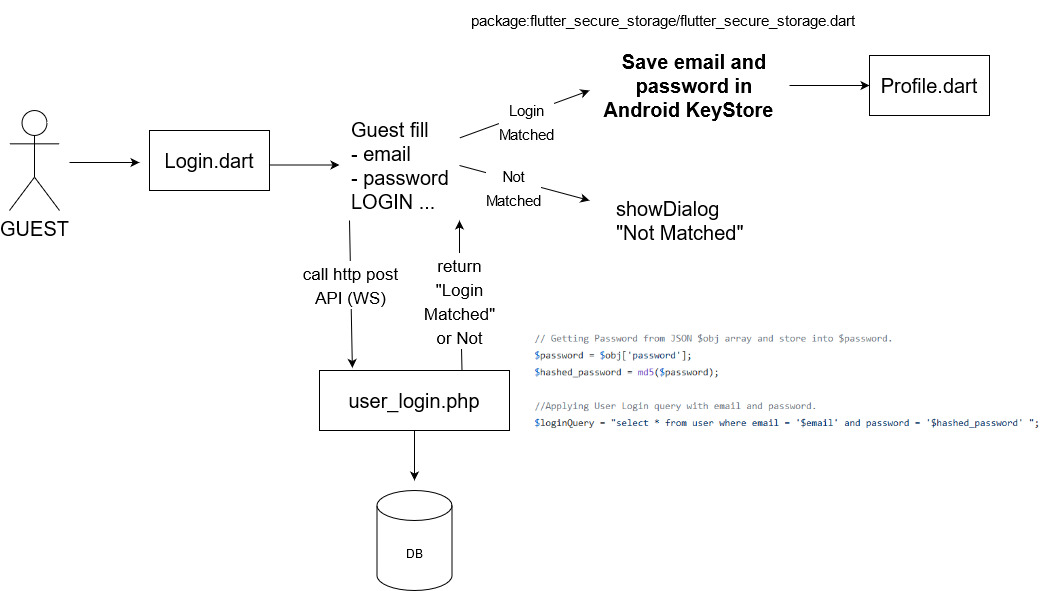
Immagine che contiene testo

Descrizione generata automaticamente

1. Immagine che contiene testo

   Descrizione generata automaticamentese il WS restituisce “Login Matched” allora vuol dire che il login ha avuto successo
2. altrimenti il login non ha avuto successo e viene mostrato un Dialog contenente un messaggio di errore.
3. In caso di successo:
   1. **si salvano le credenziali di accesso nel KeyStore utilizzando la libreria flutter\_secure\_storage** (<https://pub.dev/packages/flutter_secure_storage>);
   2. si viene reindirizzati alla schermata di profilo (profile.dart).

Lo schema seguente riassume quanto detto:



### Package flutter\_secure\_storage

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

È un plug-in Flutter per archiviare i dati in un archivio sicuro:

* per iOS viene utilizzato **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>);

KeyStore di SO Android:

* **consente di archiviare le chiavi crittografiche in un contenitore** per renderne più difficile l'estrazione dal dispositivo. Una volta che le chiavi sono nell'archivio chiavi, possono essere utilizzate per operazioni crittografiche;
* caratteristiche di sicurezza: il sistema Android Keystore protegge il materiale delle chiavi dall'uso non autorizzato. In primo luogo, Android Keystore mitiga l'uso non autorizzato delle chiavi al di fuori del dispositivo Android, impedendo l'estrazione del materiale della chiave dai processi dell'applicazione e dal dispositivo Android nel suo insieme. In secondo luogo, Android KeyStore mitiga l'uso non autorizzato delle chiavi sul dispositivo Android facendo in modo che le app specifichino gli usi autorizzati delle proprie chiavi e quindi applicando queste restrizioni al di fuori dei processi delle app.

Per i dettagli ulteriori si rimanda ai manuali sopra riportati.

### Classe UserSecureStorage

È stata implementata la classe UserSecureStorage che si occupa di comunicare con il KeyStore (per android) o il Keychain (per iOS).

La libreria flutter\_secure\_storage permette di **astrarre dalle primitive native** dei vari dispostivi (Android e iOS) fornendo dei metodi che permettono di scrivere e leggere le chiavi:

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

Per completezza, di seguito, abbiamo riportato il codice della classe.

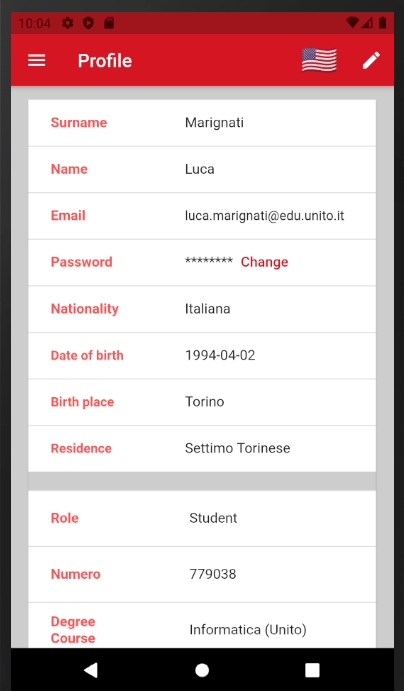
Come si nota abbiamo salvato nello storage del dispositivo i dati che permettono l’identificazione dell’utente:

* email;
* password;
* role. Il ruolo viene usato per ACL (Access Control List): gli utenti aventi ruolo “Studente” hanno un privilegio diverso da quelle dei “Tutor” (ad esempio possono avere accesso soltanto ad alcune risorse).

Immagine che contiene testo

Descrizione generata automaticamente

### profile.dart



Caratteristiche:

* **StatefulWidget**: in quanto viene effettuata una chiamata http asincrona richiamando il WS users\_list.php.

Es. <https://www.e-tutoring-app.it/ws/users_list.php?email=luca.marignati@edu.unito.it>

* **Struttura:** 
  + **FutureBuilder<UserModel> getUserInfoFromWS() 🡪 chiamata http per recuperare i dati dell’utente;**
  + **FutureBuilder<String> setRole() 🡪 per recuperare dallo storage il ruolo dell’utente.**

Immagine che contiene testo

Descrizione generata automaticamente

### ACL (Accesso Control List)

**Scelta implementativa**: flutter\_secure\_storage

Il ruolo viene salvato nello Storage sfruttando il package flutter\_secure\_storage.

Immagine che contiene testo

Descrizione generata automaticamente

In questo modo è possibile avere a disposizione l’informazione del Ruolo (Studente o Tutor) in tutti i Widget dove è necessario.

Ad esempio:Immagine che contiene testo

Descrizione generata automaticamente

Nel codice riportato sopra l’utente avente ruolo “studente” ha il permesso di accedere e visionare l’informazione relativa al suo numero di matricola (per il tutor questa informazione non avrebbe alcun significato: dunque, non viene mostrata).

Esempio:

|  |  |
| --- | --- |
| PROFILO STUDENTE | PROFILO TUTOR |
|  |  |

È possibile, anche, sfruttare il meccanismo implementato anche in fase di composizione del menù (**drawer.dart**) **mostrando e permettendo l’accesso a determinate funzionalità in base al ruolo dell’utente**. Ad esempio, per gli utenti aventi ruolo “tutor” non viene dato accesso ai corsi in quanto non è uno studente e non avrebbe alcun significato dare la possibilità di visionare i corsi relativi al suo percorso di Laurea.

|  |  |
| --- | --- |
| PROFILO STUDENTE | PROFILO TUTOR |
|  |  |

### profile.dart e gestione notifiche

1. in fase di inizializzazione del widget (initState) viene settata la lista iniziale delle notifiche (notificationsList): viene chiamato il WS tramite la funzione ***getNotificationsTutorFromWS*** che va a reperire la lista delle notifiche interrogando (select) la tabella “*notifications\_tutor*”;
2. viene avviato un timer (thread) che esegue ogni 10 secondi chiamato la funzione ***getNotificationsTutorFromWS*** per controllare se ci sono delle nuove notifiche;
3. se la lista delle notifiche iniziali ha lunghezza (lenght) minore della lunghezza della lista delle nuove notifiche vuol dire che vi sono delle nuove notifiche:
   1. viene mostrata una **notifica a livello di sistema** (es. Android) chiamando il metodo **showNotifications**();
   2. viene aggiornato lo stato tramite il metodo setState per aggiornare il contatore lato widget (**notifiche locali all’applicazione**);

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

Immagine che contiene testo

Descrizione generata automaticamente

showNotifications()

Tecnologia: FIREBASE

<https://console.firebase.google.com/u/0/project/flutterpushnotification-1e340/overview>

<https://console.firebase.google.com/u/0/project/flutterpushnotification-1e340/notification>

per la gestione delle notifiche a livello di sistema è stato configurato un progetto Firebase ed è stato integrato il file “**google-services.json**” e le librerie per eseguire la corretta comunicazione (dipendenza **com.google.gms:google-services:4.3.8**)

Immagine che contiene testo

Descrizione generata automaticamente

Metodo showNotifications(): utilizza il plugin flutterLocalNotification per mostrare la notifica mostrando come titolo della notifica il nome dell’applicazione “E-tutoring” e come testo la traduzione corrisponde alla label “message\_motification”:

* IT: Nuova lezione prenotata
* EN: New lesson booked

Immagine che contiene testo

Descrizione generata automaticamente

Metodi per la comunicazione con Firebase

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

### calendar-student.dart/calendar-tutor.dart

Caratteristiche:

* classe Event per l’inizializzazione di un evento (lezione) mostrato sul calendario

Immagine che contiene testo

Descrizione generata automaticamente

* Map<DateTime, List<Event>> eventList = {};

eventList è una Map:

* + chiave: DateTime
  + valore: List<Event>

es. per la data 23/06/2021 sono presenti nel calendario 3 eventi (lista).

* StatefulWidget: la lista degli eventi (lezioni) viene recuperata chiamato il metodo getPrivateLessonFromWS interrogando il web service private\_lesson\_list.php;

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

### course.dart



Caratteristiche:

* Modello: courseModel.dart
* Tramite il metodo **getUserCourseSearchFromWS** chiama il WS course\_search.php che restituisce la lista dei corsi relativi al percorso di laurea a cui l’utente è iscritto (Informatica, Matematica o Fisica);

Immagine che contiene testo

Descrizione generata automaticamente

* funzionalità di ricerca del corso basata sul modello (metodo toString);
* al click su un item della listview si manda al widget di dettaglio del corso.

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

# TESTING