*Relazione progetto  
Laboratorio avanzato di programmazione II  
Simone Guarnuccio X81000143*

**Nome Progetto**: AppQuest

Lo scopo del progetto ha per finalità lo sviluppo di un applicazione mobile cross-platform (Android e IOS) facendo uso del framework React-Native basato su Javascript e delle librerie Expo che permettono l’accesso a funzioni native tra le diverse piattaforme scrivendo una sola volta il codice e testandolo in fase di sviluppo tramite app “Expo” presente nei vari store: <https://play.google.com/store/apps/details?id=host.exp.exponent&hl=it>  
L’applicazione da realizzare deve somministrare e raccogliere risposte di questionari precompilati e disponibili in formato json su un server, che nel nostro caso scriviamo e/o carichiamo sul servizio di hosting Google Firebase. E’ sempre su questo che vengono caricati e aggiornati i file json relativi ai risultati del test scelto e che viene garantita e utilizzata l’autenticazione univoca dell’utente. Il link al progetto firebase è il seguente: <https://console.firebase.google.com/u/0/project/esempio-ae701/database/esempio-ae701/data>

Il link al repository GitHub è invece il seguente:

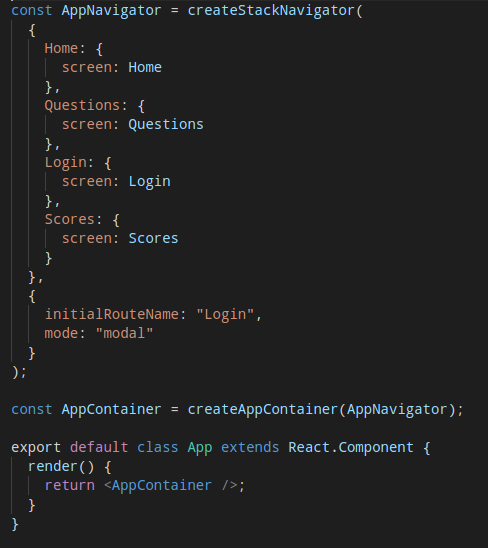
<https://github.com/grnsmn/LAP2Project>

L’applicazione è principalmente costituita da 5 file di tipo javascript:

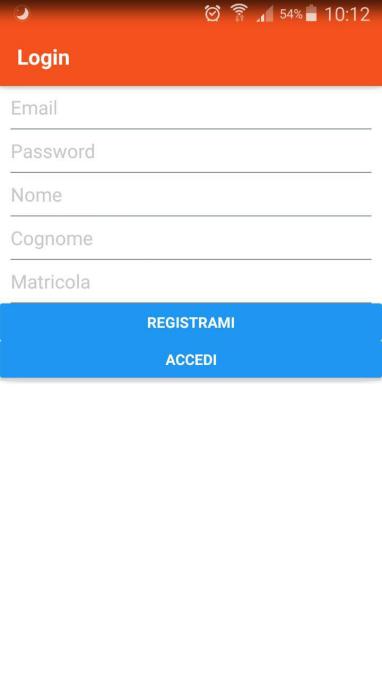
* 4 file corrispondenti a 4 screens (Login, Home, Question, Scores) di cui e’ composta l’app.
* 1 file {App} corrispondente al principale contenitore delle screen precedenti.

Procederò nella descrizione di quanto realizzato partendo dal file App.js e proseguirò poi nella descrizione delle varie screen seguendo la stessa logica di navigazione sequenziale reale dell’applicazione allo stato attuale.

***App***

File javascript che si carica principalmente dell’integrazione della libreria “React-Navigation” utile a permettere la navigazione tra screen diverse basandosi su una logica strutturale di pila. La createStackNavigator fornisce un modo per collegare logicamente le varie screen tra le quali l’applicazione potrà navigare e stabilisce anche quale screen sarà quello iniziale all’avvio dell’app.  
La createAppContainer crea il componente principale per il render dello screen da visualizzare.

***Login***

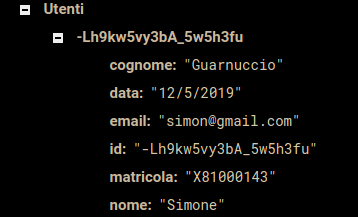


All’avvio dell’applicazione l’utente è indirizzato principalmente alla screen “Login” nella quale potere inserire i dati attraverso l’uso di Input dedicate alla modifica dello stato del componente per salvare dinamicamente i dati inseriti (Email e Password sono campi obbligatori) e successivamente scegliere tra due componenti Button per l’accesso o registrazione. I tre metodi principali di funzionamento di questa screen sono:

* \_signUp();
* \_logIn();
* \_save();

**\_save**

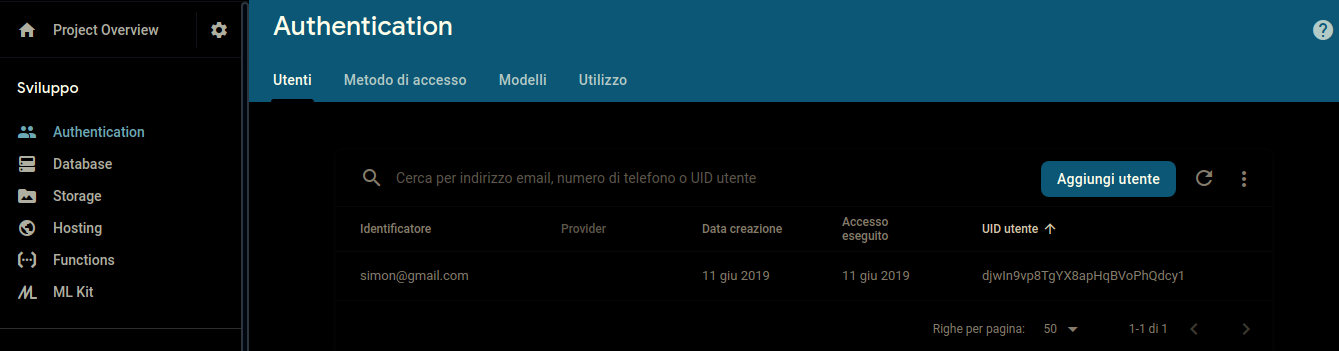
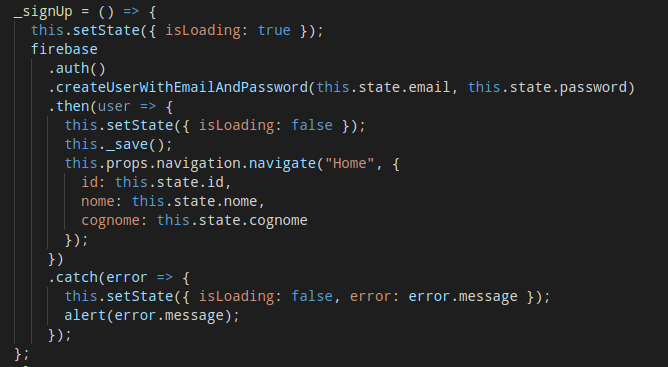
Metodo necessario a creare direttamente un oggetto utente partendo dai valori dello “state”, già modificati all’inserimento sui componenti input della screen.  
L’oggetto “utenti” nello state punta e chiama automaticamente una push di inserimento con id univoco al database di farebase sotto il nodo padre Utenti che se non ancora presente viene creato. La \_save memorizza prima di tutto la chiave univoca creata dalla push e creato l’oggetto newUser questo viene scritto nel nodo relativo alla chiave univoca attraverso la funzione .set() che permette la scrittura su uno specifico nodo del database.



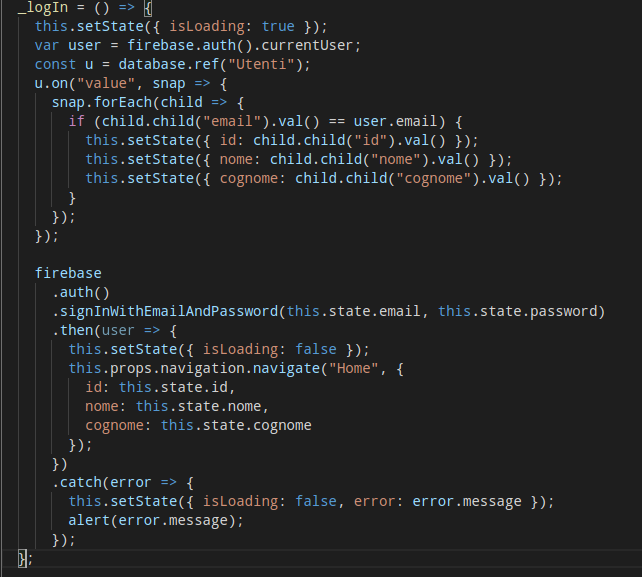
In figura il nodo utenti del database al momento della

creazione di un nuovo utente.

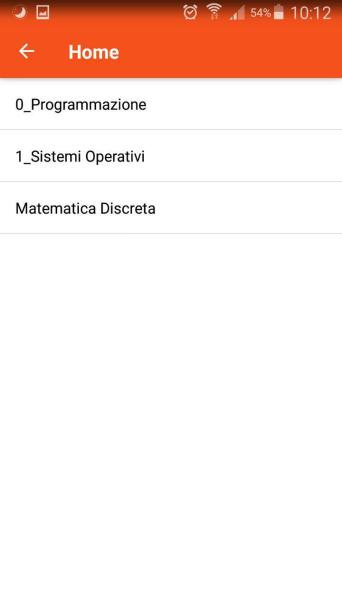
**\_signUp**

Metodo richiamato dal Button “Registrami” per effettuare la creazione di un nuovo utente tramite mail nella sezione Authentication di firebase. Verificata l’esattezza della mail inserita e andata a buon fine verrà richiamata il metodo \_save per la creazione di un nuovo utente nella sezione Database di firebase e successivamente indirizzamento alla screen Home attraverso l’uso delle navigation prop di React-Navigation che usiamo per portare tra screen diverse dati utente utili ad aggiornare il database e in particolar modo il nodo utente relativo a conclusione del test scelto ed eseguito.

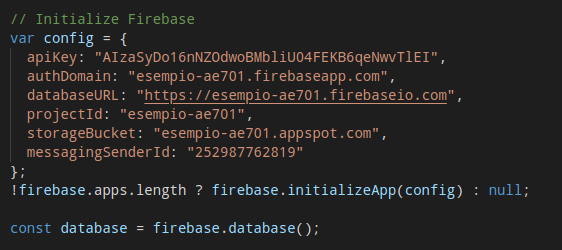
**\_logIn**

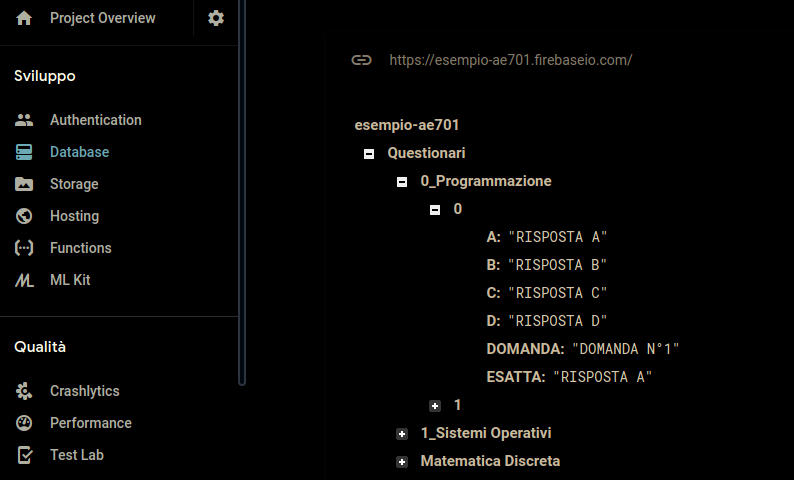
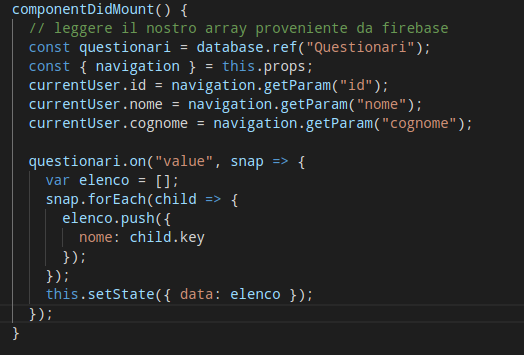


Metodo necessario a far eseguire l’accesso all’utente se già precedentemente registrato.  
Considerato il fatto che di firebase utilizziamo due sezioni (Authentication e Database) divise, quello che il metodo in un primo momento verifica è se esiste un istanza dello stesso utente autenticato nell’albero del file json in database. Se si, aggiorna in state i valori di identificazione relativi all’utente loggato perchè necessari da portare alla screen successiva per il futuro aggiornamento del nodo json.

***Home***

Rappresenta la screen alla quale l’utente visualizzerà la lista di questionari disponibili. E’ qui che viene eseguita l’inizializzazione del database di firebase attraverso il seguente codice:

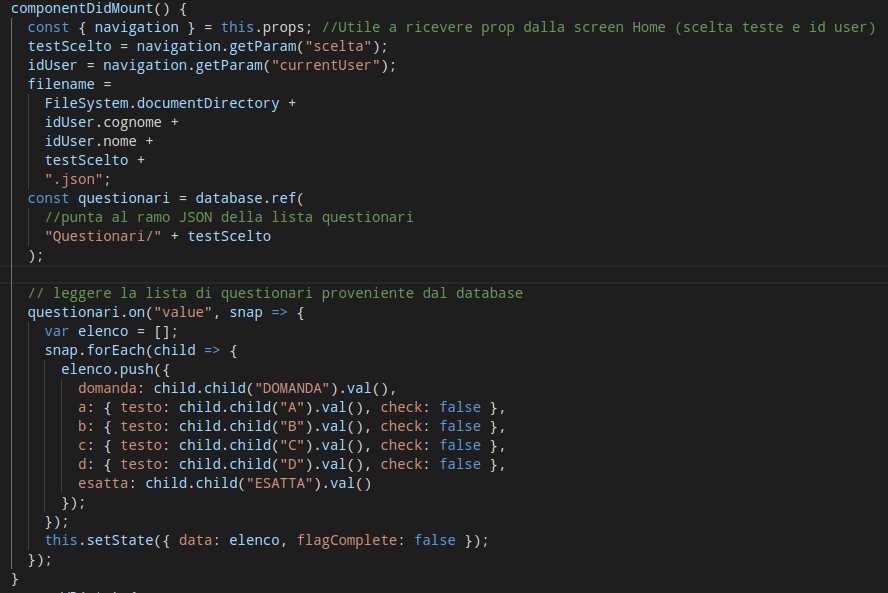


Dopo di ciò, quel che il componente della screen farà sarà spacchettare i dati ricevuti tramite navigate props dalla screen login usando l’istruzione navigation.getParam e in seguito effettua la lettura e upload dei questionari scritti e disponibili nell’albero json del database di firebase, upload eseguito temporaneamente su variabile locale che useremo subito dopo per settare lo state proprio con questo elenco, perchè è da qui che nella fase di render del componente verrà effettuata la lettura dei dati.

Proprio il render è costituito da una scrollView così da prevedere anche liste abbastanza corpose di test che saranno visualizzati e selezionabili grazie al componente ListItem inserito all’interno del metodo map richiamato dall’array di test caricato nel nostro stato.  
Alla selezione del questionario che l’utente vuole svolgere verrà indirizzato sempre con i consueti metodi alla screen Questions per l’effettiva esecuzione del test.

***Questions***

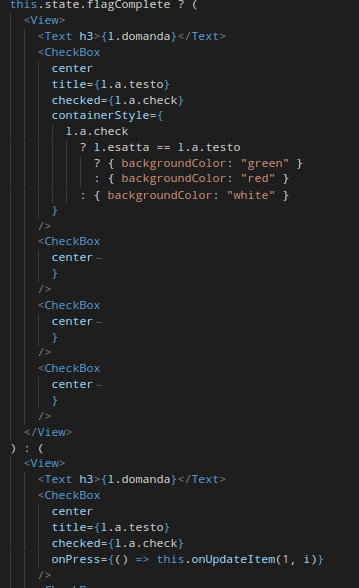
Scelto il test l’utente si ritroverà di fronte alla lista di domande e conseguenti quattro risposte multiple dalle quali poter scegliere.  
Dal punto di vista di codice è sicuramente il file più ricco tra gli altri.



Prima del render il componente spacchetta e salva su variabili globali *testScelto* e *idUser* le prop riceventi dalla screen precedente, dopodiché crea un file .json locale sul quale si scriverà il risultato del test e

infine con le stesse modalità dell’upload dei test nella screen home carichiamo stavolta il listato di domande figlie del nodo *Questionari/testScelto* nel database di firebase.

In aggiunta in questo caso avremo che lo stato presenta anche un flagComplete che è utilizzato nel conditional rendering per la diversa grafica dello screen in fase di compilazione del test e completamento.

  
Al render le risposte sono inserite in componenti checkbox e tramite conditional rendering dipendente appunto dalla flagComplete che cambia all’onPress del button in basso *“completato”* verrà modificato alle checkbox scelte per risposta il containerStyle relativo con un background verde se risposta corretta, rosso viceversa.

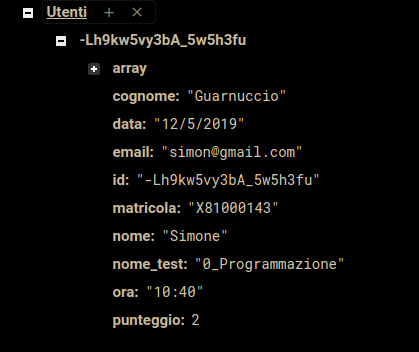
I tre metodi principali di funzionamento di questa screen sono invece i seguenti:

* updateFlag();
* onUpdateItem();

**onUpdateItem()**

Metodo necessario a garantire la scelta univoca della risposta in ogni domanda, l’intero array dell’i-esima domanda memorizzato nello stato viene aggiornato tramite setState() ed in funzione a qual è il valore di *risp* ricevuto come parametro corrispondente alla scelta dell’utente (casi da 1 a 4 in riferimento alle possibili 4 risposte selezionabili), abilita o disabilita il ckeck della checkbox abilitandone/disabilitandone il flag tenendo costantemente conto dello stato precedente del componente.

**updateFlag()**

Metodo richiamato dall’utente attraverso la button “completato” e che in primo luogo setta a ‘true’ il flagComplete nello stato dopodiché esegue la verifica della scelta effettuata dall’utente con la risposta esatta e calcola consecutivamente, memorizzando in un primo momento in variabili locali e strettamente rilegate alle singole risposte possibil, il punteggio finale nella variabile globale *scores.* Prima però viene anche inserito in un array di oggetti la relativa risposta scelta dall’utente con annesso punteggio. Questo array insieme al nome del test eseguito, del punteggio totale e dell’ora di fine saranno usati per effettuare l’aggiornamento del relativo nodo utente nel database di firebase. Fatto ciò viene anche eseguita una stringify dal relativo snap derivante dal database su una stringa che viene passata come paramentro alla funzione asincrona WR(str) che si occupa della scrittura sul file json locale creato nella fase precedente al rendering. Sempre su conditional rendering il button “completato” cambia titolo per divenire “punteggio” un button che all’onPress indirizzerà alla screen finale Scores per la visualizzazione del punteggio totale conseguito.

***Scores***



Questa screen finale serve a visualizzare il punteggio totale raggiunto al completamento del questionario svolto. Il punteggio arriva alla screen tramite le consuete navigation props e viene memorizzato nello state del componente per poi essere visualizzato.

***Conclusioni***

Il progetto è utilizzabile da qualsiasi dispositivo Android o IOS installando l’app expo e scannerizzando il QR code visualizzabile nella sezione “Project” resa disponibile nel sito di Expo.io al seguente link: <https://expo.io/@gscripts/AppQuestionari>

Oppure scaricando ed installando (Android) il file apk generato e disponibile all’interno del repository GitHub riferito all’inizio.