RELAZIONE PROGETTO PROGRAMMAZIONE MOBILE

INDICE

1. Descrizione delle necessità ed obiettivi dell'applicazione

1.1 – Introduzione

La nostra applicazione nasce con l'esigenza di voler avere più comodità nel campo dei servizi di ristorazione per quanto riguarda l'interazione tra clienti, dipendenti e ristoratori. Lo scopo della nostra applicazione è, quindi, fare in modo di utilizzare uno smartphone, che ai tempi d'oggi è diventato una vera e propria estensione del nostro corpo, per facilitare e velocizzare le ordinazioni e la produttività di un esercizio, riducendo i tempi di attesa e ottimizzando il lavoro dei dipendenti, portando soddisfazione dal lato cliente.
I dati per effettuare il login con tre diversi livelli sono scritti nella documentazione dei metodi e delle classi.
L'applicazione sarà divisa in tre livelli principali, uno per ogni tipologia di utente che potrà accedervi, descritti di seguito.

1.2 – Livello Utente

Le funzionalità di un utente (livello 1) sono:

- Effettuare la registrazione per accedere all'applicazione, inserendo i propri dati personali;
- Effettuare l'accesso attraverso i propri dati personali;
- Effettuare l'accesso in automatico se lo si è già fatto in precedenza;
- Modificare i propri dati o eliminare l'account all'interno dell'applicazione;
- Mandare un'e-mail di recupero se si dimentica la password;
- Vedere le informazioni ed il menù tutte le attività iscritte all'interno dell'applicazione;
- Effettuare dei filtri per cercare delle attività specifiche all'interno dell'applicazione;
- Creare un carrello provvisorio all'interno dell'applicazione in cui è possibile inserire i prodotti nel menù di un'attività;
- Effettuare il "rating" di un'attività, la cui recensione apparirà sotto forma di media fra tutti i rating che i diversi utenti hanno lasciato;
- Generare un QR-code dal carrello, al fine di farlo scannerizzare dai dipendenti dell'attività e quindi creare effettivamente l'ordine;
- Effettuare l'upgrade dell'account da utente a dipendente/ristoratore sotto opportune verifiche;
- Effettuare il logout dall'applicazione;

1.3 – Livello Dipendente

Le funzionalità di un dipendente (livello 2) sono:

- Scannerizzare il QR-code di un utente per confermare la presa in carica dell'ordine;
- Visualizzare l'ordine e spuntare i prodotti per la conferma del completamento;
- Visualizzare lo stato degli ordini;
- Tutte le funzionalità di un utente di livello 1 (tranne che per l'upgrade dell'account);

1.4 – Livello Ristoratore

Le funzionalità di un ristoratore (livello 3) sono:

- Creare una nuova attività, inserendo tutte le eventuali informazioni;
- Gestire le proprie attività (modifica dei dati o eliminazione);
- Creare un menù;
- Gestire il menù (modifica o eliminazione dei prodotti);
- Tutte le funzionalità di un utente di livello 1 (tranne che per l'upgrade dell'account);

2. Scelta del linguaggio di programmazione e framework

2.1 – Android Studio

Come ambiente di sviluppo abbiamo scelto il framework visto e studiato durante le lezioni, ovvero "Android Studio". È basato sul software di JetBrains IntelliJ IDEA, ed è l'IDE primario di Google per lo sviluppo nativo di applicazioni Android. La versatilità rispetto alle versioni del sistema operativo ed alla scelta di vari linguaggi di programmazione, oltre che avere la possibilità di effettuare il debug e l'opzione di avere un'anteprima sulle interfacce grafiche attraverso l'utilizzo di file "xml", secondo noi, fanno di Android Studio il software più comodo ed efficace per organizzare progetti che riguardano applicazioni mobile.

2.2 – Firebase

Per effettuare operazioni di creazione, modifica ed eliminazione permanenti di dati inerenti all'applicazione, ci siamo appoggiati ad un servizio online chiamato "Firebase". Si tratta di un database NoSQL che permette di salvare e sincronizzare i dati elaborati da applicazioni web e mobile. Per utilizzare Firebase in Android Studio, è bastato iscriversi al sito e collegare un nuovo progetto alla nostra applicazione direttamente dall'ambiente di sviluppo. Per le autenticazioni, sono stati utilizzati i metodi standard che Firebase ci offre, effettuando quindi la registrazione e l'autenticazione attraverso E-mail e password e salvandoli nel campo "Authentication" in Firebase. Abbiamo utilizzato la funzionalità "Real-time Database" per salvare dati in maniera permanente, potendoli recuperare attraverso delle query grazie alle API di Firebase e mostrarli nella nostra app. Infine, per quanto riguarda tipi di dati particolari, come le immagini, abbiamo utilizzato lo "Storage", in cui ogni file potrà essere recuperato attraverso il suo path univoco.

2.3 – Kotlin (Android)

Per quanto riguarda il linguaggio di programmazione abbiamo scelto "Kotlin", anch'esso studiato durante il corso. È un linguaggio che si basa sulla JVM, general purpose, multi-paradigma ed open source, sviluppato anch'esso dall'azienda JetBrains. Particolarmente orientato verso la programmazione ad oggetti, è riconosciuto tutt'oggi come il miglior linguaggio per lo sviluppo di applicazioni mobile. Rispetto a Java, lo abbiamo trovato molto più sintetico e semplice da apprendere, con un'ampia varietà di librerie standard ed importabili attraverso le "dependencies", oltre la sua capacità di essere pienamente interoperabile con Java.

2.4 – Flutter (iOS)

Abbiamo adattato parte della nostra applicazione Android per gli utenti che hanno un dispositivo iOS grazie a "Flutter". Si tratta di un framework open source, affrontato a lezione, creato da Google per la creazione di interfacce native per iOS, Android, Linux, macOS e Windows che utilizza due linguaggi di programmazione per due macro-strati: C/C++ e Dart. È stato possibile implementarlo ed utilizzarlo direttamente in "Android Studio", potendo così non avere bisogno di due ambienti di sviluppo diversi. I vantaggi principali che abbiamo riscontrato rispetto a Kotlin sono: la funzionalità "hot reload", che non richiede di ricompilare il codice, ad esempio, per vedere modifiche visive; i "widgets", componenti visive grafiche che possono anche mutare nel tempo, in cui è possibile adattare le funzionalità volute. Il passaggio da Kotlin a Flutter è sembrato un po' scomodo agli inizi, dato che il primo permette una divisione tra interfaccia grafica e funzioni, potendo avere un codice più pulito, mentre il secondo a primo impatto è sembrato più disordinato e confusionario, ma ci siamo adattati abbastanza in fretta.

2.5 – GitHub

Entrambi i progetti (Android ed iOS) sono stati eseguiti in parallelo da tutti e tre i membri del gruppo

grazie all'utilizzo di GitHub, una piattaforma di hosting per il controllo delle versioni e la collaborazione allo sviluppo di software. Potendo condividere e tenere traccia delle modifiche apportate al codice sorgente nel corso del tempo, è stato semplice aggiornare il progetto nel proprio dispositivo non appena un altro membro avesse effettuato delle modifiche. Il sistema di commit, push e pull è stato fondamentale per apportare nuove funzionalità nelle stesse classi contemporaneamente, attraverso l'utilizzo di "unione" o "riassegnazione".

3. Sviluppo dell'applicazione (Android)

3.1 – Requisiti

I requisiti funzionali sono tutte le specifiche di ciascun livello di utente che potrà accedere all'applicazione. Il punto centrale è quello di avere accesso, indipendentemente dal livello utente, ad un proprio profilo personale, dove poter creare o modificare un carrello, inserendo momentaneamente prodotti dal menu di un'attività, come se stessimo facendo un ordine. Dopo di ciò, è possibile salvare momentaneamente quel carrello all'interno del proprio account sotto forma di QR-code, nell'attesa che il dipendente lo scannerizzi per confermare e poter eseguire l'ordine. La scelta dell'utilizzo di QR-codes è nata dall'idea di poter creare l'ordine anche da casa, o durante il viaggio verso l'attività. La concretizzazione effettiva dell'ordine avverrà tramite la scansione da parte del dipendente una volta che si è sul posto, evitando l'invio di ordini da parte di utenti non presenti nel ristorante.
I requisiti non funzionali che abbiamo ricercato sono l'usabilità dell'applicazione da qualsiasi tipologia d'utente che la scaricherà, da chi è più ferrato con l'utilizzo di uno smartphone a chi non lo è affatto, cercando di rendere il più semplice possibile l'interazione con l'interfaccia grafica e con la navigazione. Inoltre, utilizzando Firebase, è stato possibile adattare il sistema di sicurezza integrato nell'Authentication per quanto riguarda la protezione di dati personali, come E-mail e password. Per quanto riguarda alcuni casi particolari, come la verifica della veridicità delle informazioni o per effettuare l'upgrade del proprio account, non sono state implementate con completezza, poiché il progetto è a scopo scolastico.
I casi d'uso da parte degli utenti di diverso livello possono essere approfonditi nello specifico consultando la documentazione del codice.

3.2 – Architettura

L'architettura che abbiamo deciso di considerare comprende quattro principali activity, caratterizzati sia da fragment comuni a tutti i livelli, sia da alcuni che vanno ad implementare nello specifico le funzionalità desiderate.
La lista seguente, suddivisa per activity, ne descrive l'interezza dell'app:

- Intro Activity:
  Activity iniziale dell'applicazione. Chi apre l'applicazione verrà proiettata verso il primo fragment della seguente lista, e potrà navigare nei restanti:
    - Fragment Intro
    - Fragment Login
    - Fragment Registrati

  Chi utilizza questa activity può effettuare la registrazione, il login ed il login automatico.

- User Activity:
  Activity iniziale per chi effettua il login ed il suo livello corrisponde ad 1. Il primo fragment mostrato sarà il primo della lista, ma l'utente potrà effettuare la navigazione per il resto dei fragment:
    - Fragment Ristoranti

    - Fragment Profilo
    - Fragment Impostazioni
    - Fragment Dettagli ristorante
    - Fragment Cerca ristoranti
    - Fragment Carrello
    - Fragment QR-Code
    - Intro Activity (attraverso il logout)

  Chi utilizza quest'activity può accedere alla lista dei ristoranti, che saranno inizialmente suddivisi per varie categorie. La lista potrà essere filtrata attraverso la possibilità di ricerca. Cliccando su un ristorante specifico, verranno visualizzati tutte le informazioni, compreso il menù, ed è possibile effettuare una recensione. Proprio da qui è possibile inserire i prodotti desiderati al carrello. Inoltre, accedendo al profilo si potranno visualizzare e modificare le informazioni personali principali. Dalle impostazioni del profilo, inoltre, è possibile effettuare l'upgrade del livello utente a dipendente o ristoratore, oltre che poter eliminare il proprio profilo. È possibile accedere alla pagina del carrello per creare il QR-Code e visualizzarlo nell'apposito fragment. Infine, è possibile effettuare il logout, che ci riporterà alla pagina Intro.

- Employee Activity:
  Activity iniziale per chi effettua il login ed il suo livello corrisponde a 2. Come per la User Activity, il fragment iniziale sarà lo stesso:
    - Tutti i fragment della User Activity
    - Fragment Lavoro
    - Fragment Dettagli ordine

  Chi utilizza quest'activity può effettuare tutte le funzionalità disponibili della User Activity (tranne che per l'upgrade dell'account), oltre che poter visualizzare la lista degli ordini in corso dalla pagina lavoro. Infine, per ogni singolo ordine si possono visualizzare in dettaglio i prodotti e le relative quantità.

- Restaurateur Activity:
  Activity iniziale per chi effettua il login ed il suo livello corrisponde a 2. Come per la User Activity, il fragment iniziale sarà lo stesso:
    - Fragment Gestione ristoranti
    - Fragment Crea ristorante
    - Fragment Crea menu
    - Fragment Modifica ristorante
    - Fragment Modifica menu

    Chi utilizza quest'activity può effettuare tutte le funzionalità disponibili della User Activity (tranne che per l'upgrade dell'account), oltre che poter avere accesso alla pagina gestione. Da qui, oltre che creare un nuovo ristorante ed il suo relativo menù, è possibile visualizzare e cercare la lista dei propri ristoranti, permettendone la modifica dei campi o l'eliminazione. Infine, anche il menù potrà essere modificato in seguito.

Gli schemi di navigazione verranno riportati nella documentazione del progetto, al fine di avere una visuale più comprensibile.

Per ogni entità utilizzata, è stato creato un model inerente, con attributi e tipologie, metodi getter e setter e costruttori. Sono stati raggruppati all'interno del package "models". Inoltre, funzioni che riguardano l'utilizzo di questi, o funzioni che vengono effettuate da più parti del programma o che ingombrerebbero spazio di codice rendendolo più disordinato, sono state scritte all'interno del

package “utils”. Molte di queste funzioni servono per effettuare operazioni con il database.
Infine, la gestione del carrello è stata effettuata attraverso l’utilizzo di un ViewModel: utilizzando una lista di prodotti del carrello di tipo “MutableLiveData” è stato possibile osservare eventuali aggiornamenti all’interno dell’applicazione. Questi aggiornamenti sono effettuati attraverso determinate funzioni nella classe “CartViewModel”.

3.3 – UI

Ogni interfaccia grafica dei vari fragment è stata creata attraverso un univoco file xml, contenente le componenti necessarie alle funzionalità della pagina stessa.
All’interno del progetto, oltre ai layout xml di ogni fragment, ve ne sono alcuni raffiguranti l’interfaccia grafica di pop-up o cards, sotto il path “res/layout”.
Per quanto riguarda alcune componenti grafiche fisse o file permanenti (come forme per bottoni o immagini standard), sono raccolte nel path “res/drawable”.
Inoltre, ad ogni livello utente sono state attribuite delle bottom navigation bar diverse, raccolte in “res/menu”. Qui, è anche presente la top navigation bar, contenente il Carrello e la pagina del QR-Code, comune a tutti i livelli.
Nel path “res/navigation”, è possibile consultare le mappe di navigazione di ogni livello.
Nel path “res/values”, invece, abbiamo raccolto tutte le stringhe fisse che andranno a comparire in ogni pagina, gli stili standard e personalizzati dei vari componenti, i colori e i temi.
Infine, nella cartella “font” abbiamo importato alcuni font che potrebbero essere utilizzati nell’applicazione, selezionandone uno e con l’idea futura di poterlo cambiare all’interno dell’app.
I colori primari scelti sono il rosso ed il bianco, in modo da ricordare il “rosso” del fuoco della cucina, ed avendo un colore neutro come il bianco per non infastidire l’esperienza dell’utente con colori troppo sgargianti.
PROBLEMI NOTI:
Si è notato che durante la modifica del menù di un ristorante, le liste dei prodotti non vengono mostrate correttamente, limitando i prodotti ad una lista di tre oggetti.
Inoltre, non sono stati effettuati controlli per lunghezze di campi durante l’inserimento di nuove caratteristiche riguardanti modelli come ristoranti o prodotti, tale per cui alcune descrizioni potrebbero essere visualizzate parzialmente.

3.4 – Testing

Durante il processo di testing, sono stati eseguiti diversi tipi di test in JUnit4 di integrazione e test UI, al fine di coprire in modo esaustivo alcuni aspetti critici dell’applicazione. I dettagli in seguito:

1- Intro Navigation Test: in questo test sono stati effettuati controlli innanzitutto per quanto riguarda la verifica dello scenario iniziale all’avvio dell’applicazione: è stato testato che, quando viene lanciata “Intro Activity”, l’interfaccia visualizzata corrisponde a “R.id.intro_activity”. Inoltre, la seconda funzione riguarda l’effettiva visibilità di determinate componenti grafiche all’interno della view, come il tasto di Login o le stringhe Benvenuto e Registrati.
2- Login Navigation Test: in questo test sono state effettuate verifiche che riguardano l’accurata navigazione da un fragment ad un altro quando si clicca su determinati bottoni.
Il primo riguarda l’effettiva visibilità del bottone “Login” nella pagina intro che, dopo averlo cliccato, se non è stato effettuato il login in passato, porterà l’utente alla pagina di Login effettiva, verificando che determinate componenti grafiche vengano renderizzate.
Il secondo invece riguarda la navigazione dalla pagina di Login a quella di Registrati: dopo aver controllato l’effettiva visibilità del bottone con ID “noaccount” e dopo averlo premuto, la navigazione dovrà portare alla pagina “Registrati”, caricando così a schermo le componenti grafiche dell’interfaccia.

3- User Navigation Testing: in questo test creiamo un activity scenario che permette di eseguire action e assertions sull'activity Intro, andando a verificare che alcune componenti grafiche sono state renderizzate a schermo. Dalla splash screen dell'applicazione, cliccando sul bottone Login, andiamo a testare che la nostra pagina di login sia visualizzata correttamente.
Per questo test ci siamo avallati di Espresso, un framework fornito da Android Testing Support Library. Esso fornisce un'API fluida e facile da usare per scrivere test dell'interfaccia utente concisi ed affidabili. Lo scopo principale di Espresso è simulare le interazioni dell'utente con l'interfaccia utente di un'app e verificare il comportamento previsto.

4. Sviluppo dell'applicazione (iOS)

4.1 – Requisiti

Come per l'applicazione Android, i requisiti funzionali e non funzionali sono gli stessi. Lo sviluppo dell'applicazione per iOS è stato implementato coerentemente con lo spazio dedicato a Flutter durante le lezioni. Infatti, non è stata prevista la differenziazione dei livelli utente: questo significa che chiunque entri nell'applicazione tramite dispositivo iOS potrà accedere alle funzionalità principali di un utente di livello 1 (tranne che per l'upgrade dell'account e la modifica delle proprie informazioni).

4.2 – Architettura

L'architettura utilizzata per lo sviluppo dell'applicazione in Flutter è leggermente diversa rispetto a quella in Android. Per quanto riguarda la navigazione, essendo poche le classi, si è optato per una navigazione semplice e lineare rispetto ai navigation graph di Android. Per quanto riguarda l'utilizzo delle activity, infatti, non c'era da fare distinzione fra livelli utente. Per cui, si inizia navigando nelle tre pagine di login che appartengono al package "Intro":

- PageIntro
- PageLogin
- PageRegistrati

Il resto della navigazione è composto dalle seguenti classi:

- PageCarrello
- PageQR_Code
- PageProfilo
- PageRistoranti
- PageSearch
- PageRestaurantDetail

È possibile consultare gli schemi di navigazione nella documentazione.

Per quanto riguarda le entità principali, anche qui sono stati creati dei modelli con la relativa funzione per convertirlo da una mappa iniziale al modello vero e proprio, tutte all'interno del package "models".
Le query per accedere al database sono state controllate attraverso due classi chiamate "FirebaseControllerRestaurants", per recuperare informazioni relative ai dati dei ristoranti, e "FirebaseControllerMenu", per recuperare invece informazioni relative ai menu dei ristoranti. Utilizzando la libreria "Getx", è stato possibile aggiornare il contenuto delle liste di tipo "Stream" restituite dalle query asincrone ed osservare cambiamenti all'interno delle altre classi, in modo da poter effettuare opportuni filtraggi ed inserire il risultato in delle liste strutturate. Inoltre, dato che in entrambi i casi il cambiamento e filtraggio di lista doveva avvenire attraverso il click di una lista di bottoni, abbiamo preferito usare un widget chiamato "Choice Chip". Infatti, nella pagina "ChipController", sono state create due classi "ChipController" (per le tipologie di cibo dei ristoranti) e "ChipControllerMenu" (per le tipologie di prodotti nei menu dei ristoranti), che vengono osservate

all'interno dei controller per le query ed aggiornano la tipologia di filtro scelta.
Infine, la gestione del carrello è stata effettuata attraverso l'utilizzo della classe "Change Notifier" di Flutter. Nella classe "CartProvider", che estende quest'ultima, sono state inserite delle funzioni per la creazione, aggiunta o eliminazione di prodotti carrello in una lista, che può essere osservata nel menu di un ristorante o nel carrello.

4.3 – UI

Per quanto riguarda l'interfaccia grafica, si è cercato di renderla il più possibile uguale a quella creata in Android, con stesse interfacce, dimensioni simili e stessi colori. Alcuni widget sono stati creati direttamente dentro le classi che li utilizzavano, mentre il resto sono stati raccolti nella classe "widgets" per comodità e pulizia del codice, come dei bottoni rossi personalizzati o delle card per le liste di ristoranti/prodotti.
PROBLEMI NOTI:
Per quanto riguarda la renderizzazione di immagini all'interno dell'app iOS, non è stato confermato che funzioni per dispositivi reali, poiché non se ne disponeva di uno. Non è stato possibile risolvere questo problema per quanto riguarda la visualizzazione di immagini nei vari browser (come Edge o Chrome) poiché, dopo molta ricerca, è stata trovata l'unica soluzione di far partire l'applicazione attraverso il comando "flutter run -d chrome –web-renderer html" nel terminale per la corretta renderizzazione delle immagini, dove al posto di "chrome" è possibile inserire il browser prediletto.

5. Conclusioni

In conclusione, l'implementazione di Cookade sia in Kotlin sia in flutter ha fornito una prospettiva interessante sullo sviluppo cross-platform e nativo per Android. Abbiamo potuto valutare le differenze e le similitudini tra le due tecnologie, nonché i relativi vantaggi e sfide descritte anche precedentemente.
Lo sviluppo in Kotlin ha sfruttato le peculiarità e le funzionalità native di Android, consentendo un accesso diretto alle API che sono state implementate. Kotlin ha reso la sintassi molto più semplice ed efficace, rendendo il codice più chiaro e leggibile. D'altro canto, la documentazione fornita e le ricerche online hanno occupato molto tempo, poiché Kotlin è un linguaggio relativamente nuovo.
L'utilizzo di Flutter, invece, ha permesso di creare un'interfaccia utente reattiva e di maggiore qualità, grazie alla struttura di widget dichiarativa e all'ampia personalizzazione. La compilazione rende rapido il ciclo di sviluppo, facilitando il riavvio ed il debug dell'applicazione. Infine, l'hot-reload ha velocizzato moltissimo la realizzazione delle viste.
Complessivamente, l'esperienza di sviluppo con Cookade in Kotlin e Flutter ha arricchito di molto il bagaglio di competenze per lo sviluppo di applicazioni mobile e cross-platform.