

WIP

Work Is Progress

Android version documentation



Alfabeticamente, a cura di:
Colleuori Federico, Frisi Emanuele, Giusti Kevin
Versione documento: 1.0

Indice

Indice	2
Introduzione	4
Generalità	5
Raccolta dei requisiti	7
Analisi dei requisiti	15
divisione delle user story in task	15
Analisi dei requisiti	22
glossario dei termini	22
Analisi dei requisiti	23
requisiti utente: requisiti funzionali e non funzionali	23
Schematizzazione generale dei requisiti del sistema	23
Requisiti funzionali e descrizione strutturata	24
Requisiti non funzionali e descrizione strutturata	28
requisiti funzionali sistema: UML use-case diagram	29
Attori use case	30
Diagramma use case: iniziare una nuova storia	31
Diagramma use case: sviluppare e concludere una nuova storia	36
Diagramma use case: navigare nello shop	38
Diagramma use case: consultare i propri progressi	42
Diagramma use case: configurare le preferenze	46
Progettazione database	49
Dizionario dei dati	53
Entità	53
Relationship	54
Vincoli non esprimibili	55
Implementazione del database	56
Mappa dell'architettura	57
Use case banale: apertura dell'app	58
Use case: iniziare una nuova storia	59

Use case: sviluppare e concludere una nuova storia	60
Use case: navigare nello shop	60
Use case: consultare i propri progressi	61
Use case: configurare le preferenze	62
Analisi dell'architettura	63
Intent map	89
Test	90
Contributi	94
Strumenti utilizzati e Bug	95

Introduzione

“Niente è facile come sembra”

“Tutto richiede più tempo di quanto si pensi”

“Ogni soluzione genera nuovi problemi”

- *Corollari della legge di Murphy, nonché leggi fondamentali dell'ingegneria del software*

Il documento che segue è atto ad accompagnare il lettore nella comprensione di tutti gli elementi costitutivi del software “WIP: Work Is Progress”, progettato e sviluppato nell’arco di tre mesi dalle matricole Colleluori Federico, Frisi Emanuele e Giusti Kevin, di cui riportiamo i contatti:

- Colleluori Federico, matr. 1093242, e-mail s1093242@studenti.univpm.it
- Frisi Emanuele, matr. 1092866, e-mail s1092866@studenti.univpm.it
- Giusti Kevin, matr. 1092345, e-mail s1092345@studenti.univpm.it

Università Politecnica delle Marche
Facoltà di Ingegneria Informatica e dell’Automazione
Corso di Programmazione Mobile 2021/22

Generalità

classificazione app, utenti target e software simili

WIP: Work Is Progress è un'applicazione mobile classificabile mediante le seguenti app-categories:

- **Productivity:** categoria di software che aiuta l'utente nella gestione e nell'esecuzione di compiti quotidiani;
- **Time management:** categoria di software che permette all'utente di gestire il tempo;
- **Focus keeping:** categoria di software che impedisce l'uso di alcune funzionalità del device al fine di aumentare la concentrazione dell'utente durante le attività quotidiane;
- **Interactive study:** categoria di software che invoglia l'utente alla produttività mediante l'uso di un device;
- **Pixel art video-game graphic:** categoria di software con UI disegnata in pixel art;

In particolare, l'insieme dei requisiti funzionali e non funzionali derivanti dalla tassonomia appena presentata colloca WIP tra le applicazioni native poiché:

- le native-apps funzionano completamente anche offline; nel nostro caso, garantire che l'app sia affidabile e persistente è un requisito non funzionale critico banalmente esplicabile attraverso la seguente implicazione logica: l'utente si rivolge alla nostra app se è consci di avere una soglia di distrazione molto alta e/o è demotivato ⇒ l'app deve funzionare correttamente e totalmente per l'intero tempo di attività dell'user poiché, se così non fosse, essa favorirebbe perdita di concentrazione;
In conclusione, il software non può dipendere dall'assenza o dalla presenza di connettività nemmeno per le funzionalità di minor rilievo;
- le native-apps si basano su tool del sistema operativo; nel nostro caso, il software interagisce ampiamente e continuamente con il S.O per:
 - calcolo del tempo di attività dell'utente; in breve, il cronometro si basa sul clock che conta il tempo di attività del device;
 - rilevare l'activity corrente in foreground;
 - leggere/scrivere dal/sul database in memoria locale;
 - rilevare il lock e l'unlock dello schermo;

- le native-apps sono molto più veloci ed affidabili rispetto alle altre app, e quindi offrono una User Experience di più alto livello;
Nel nostro caso, la velocità delle operazioni da eseguire è essenziale poiché l'utente deve utilizzare il software solo ed esclusivamente per configurare i parametri della sua sessione di produttività;
il restante tempo di vita dell'app deve essere in background.

Per quanto riguarda gli utenti target, WIP è un'applicazione di produttività rivolta principalmente a studenti ma che, in generale, può essere utilizzata da qualsiasi individuo necessiti di tracking del tempo e di meccanismi di blocco temporaneo del dispositivo che favoriscono benessere digitale e focus keeping;

Per concludere, è importante citare le applicazioni di riferimento che hanno permesso la progettazione di WIP:

- Forest - Stay focused, be present:
 - dev company: Seekrtech
 - descrizione: app di produttività e di smartphone disintoxication
 - riferimenti: è possibile apprezzare tutte le qualità di Forest nella sezione “informazioni su questa app” presente nel fondo della pagina:
<https://play.google.com/store/apps/details?id=cc.forestapp&gl=IT>
- HourBuddy - Monitoraggio del tempo e produttività:
 - dev company: NeuronDigital
 - descrizione: app per il monitoraggio del tempo, per la produttività e per la produzione di statistiche in base alle attività svolte
 - riferimenti: è possibile apprezzare tutte le qualità di HourBuddy nella sezione “informazioni su questa app” presente nel fondo della pagina:
<https://play.google.com/store/search?q=hourbuddy&c=apps&gl=IT>

Raccolta dei requisiti

user story e UI navigation

Nota vocabolario: i vocaboli contrassegnati con “*” compaiono nel glossario dei termini.

Nota grafica: la risoluzione delle immagini aumenta zoomando su di esse.

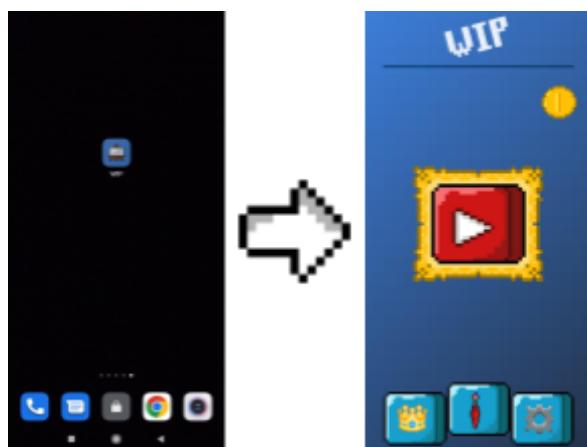
1. Avvio app:

Michelangelo è un ragazzo che desidera cominciare a studiare/lavorare in modo produttivo e senza distrazioni e, per questo motivo, decide di utilizzare WIP.

Cliccando sull’icona dell’applicazione, a Michelangelo viene mostrata la schermata principale che prevede:

- il menù nella parte bassa dello schermo;
- il pulsante “play” necessario all’avvio di una nuova storia*;
- il pulsante “shop” per accedere al negozio;

graficamente:



2. Navigazione mediante menù:

Michelangelo, incuriosito, decide di navigare nell’applicazione mediante l’utilizzo del menù posto nella parte bassa dello schermo;

tale menù è costituito da tre bottoni:

- il bottone “pennello”, che serve a mostrare a Michelangelo la schermata principale se egli sta visualizzando un’altra schermata;
- il bottone “corona”, che serve a mostrare a Michelangelo la schermata “Kingdom”, ovvero lo storico di tutte le sue storie conclusive, se egli sta visualizzando un’altra schermata.
per maggiori info, consulta la user story 9;
- il bottone “impostazioni”: che serve a mostrare a Michelangelo la schermata “Impostazioni” se egli sta visualizzando un’altra schermata.
per maggiori info, consulta la user story 11;

Graficamente:



Per promuovere maggiore chiarezza, avanziamo il seguente esempio: Michelangelo si trova nella schermata principale; dunque:

- se preme il pulsante “pennello”, non accade nulla;
- se preme il pulsante “corona”, viene mostrata la schermata “Kingdom”;
- se preme il pulsante “impostazioni”, viene mostrata la schermata “Impostazioni”;

3. Avvio storia:

Michelangelo ha già avviato l'app e, in questo momento, sta visualizzando la schermata principale;

Per iniziare una nuova storia, ovvero una nuova sessione di produttività, Michelangelo clicca il pulsante “play” e viene reindirizzato alla schermata di impostazione dei parametri della storia;

In questa schermata, Michelangelo :

- scrive, mediante tastiera, il titolo della nuova storia;
- definisce la partizione studio-pausa su una linea temporale di 60 min; ad esempio:
 - selezionando 50 min di studio si impostano automaticamente 10 min di pausa;
 - selezionando 40 min di studio si impostano automaticamente 20 min di pausa, ecc...
- seleziona le impostazioni opzionali:
 - se si opziona “modalità silenziosa”, il telefono va in modalità silenziosa*;
 - se si opziona “modalità hardcore”, il telefono va in modalità silenziosa e se si esce* dall'app la storia termina automaticamente;
 - è possibile non selezionare alcuna opzione;
 - è possibile visualizzare a schermo informazioni relative a “modalità silenziosa” e “modalità hardcore” premendo l'apposito tasto “info”;

- seleziona il suo avatar* mediante pulsanti a forma di frecce direzionali;
- se preme il tasto “indietro”, Michelangelo esce dalla schermata di impostazione dei parametri della storia e torna alla schermata principale dell’app;
- se preme il tasto “start”, Michelangelo inizia ufficialmente la sua sessione di produttività e, di conseguenza, gli viene mostrata la schermata “storia avviata”.

graficamente:



4. Sviluppo storia:

Michelangelo ha ufficialmente iniziato la sua sessione di produttività e, di conseguenza, sta visualizzando la schermata “storia avviata” che contiene:

- il cronometro che indica da quanto tempo la storia è stata avviata;
- il quadro da dipingere durante la sua sessione di produttività di 60 min;
- il suo avatar che, ad ogni sblocco* del cellulare, enuncia una frase di incoraggiamento;
- il pulsante “stop” che, se premuto, termina la storia;

graficamente:



5. Conclusione storia:

Mentre sta visualizzando la schermata “storia avviata”, Michelangelo decide di voler concludere la propria sessione di produttività e, di conseguenza, preme il tasto “stop”; In questo modo:

- il cronometro si ferma;
- vengono calcolate le monete che Michelangelo riceve come ricompensa per la sua produttività;
- viene mostrato un messaggio che mostra il numero di monete guadagnate;
- le informazioni relative alla storia appena conclusa vengono salvate nella schermata “Kingdom” (solo se il tempo segnato dal cronometro è maggiore o uguale a 10 secondi);
- Michelangelo viene riportato nella schermata principale dell’app; graficamente:



6. Accesso al negozio:

Dopo aver guadagnato molte monete attraverso il completamento delle sue storie, Michelangelo vuole acquistare un nuovo avatar/background.

Michelangelo si trova nella schermata principale e, premendo l’icona della moneta, viene reindirizzato alla schermata “Shop”;

(è possibile accedere al negozio solo ed esclusivamente dalla schermata home). graficamente:



L'uscita dalla schermata "Shop" è possibile mediante l'ausilio del bottom menù.

7. Tour del negozio:

Poiché intendo ad effettuare un acquisto, Michelangelo sta visualizzando la schermata "Shop" che gli permette di:

- vedere il numero di monete che ha guadagnato;
- scorrere gli avatar acquistabili e quelli già acquistati mediante dei bottoni a forma di frecce direzionali;
- scorrere i quadri acquistabili e quelli già acquistati mediante dei bottoni a forma di frecce direzionali;

Quando Michelangelo seleziona l'avatar/background che vuole acquistare, si apre la schermata di dettaglio ordine che mostra:

- l'avatar/background selezionato;
- il nome dell'avatar/background selezionato;
- un pulsante "info" che, se premuto, mostra a schermo una breve storia circa l'opera d'arte selezionata;
- un pulsante di acquisto.

graficamente:



se Michelangelo non vuole acquistare l'opera, allora preme il tasto "indietro" che gli mostra nuovamente la schermata "Shop".

8. Acquisti nel negozio:

Michelangelo sta visualizzando la schermata di dettaglio ordine e, poiché intento ad acquistare l'avatar "David", preme il tasto "Buy";

Immediatamente, Michelangelo vede a schermo un messaggio di conferma acquisto; graficamente:



Se Michelangelo opziona "no", il messaggio si chiude e viene mostrata la schermata di dettaglio ordine;

al contrario, se Michelangelo opziona "si", si hanno due esiti possibili:

- l'acquisto viene correttamente effettuato e viene mostrato a schermo un messaggio "ricevuta";
- le monete sono insufficienti e, di conseguenza, viene mostrato un messaggio errore nel checkout;

graficamente:



i messaggi "ricevuta" e "errore nel checkout", una volta chiusi, mostrano a Michelangelo la schermata "Shop";

Per concludere, è importante notare che:

- gli elementi non acquistati presentano il costo nella schermata "Shop" ed il bottone "buy" nella schermata di dettaglio ordine;

- gli elementi acquistati presentano il “tick” nella schermata “Shop” ma non il bottone “buy” nella schermata di dettaglio ordine;

9. Le mie storie:

Michelangelo, dopo aver creato e completato con successo numerose storie, vuole controllare i progressi compiuti;

Per questo motivo, Michelangelo sta visualizzando la schermata “Kingdom” che contiene:

- la scaletta di storie create fino a questo istante;
- una story navbar contenente la sequenza di storie create fino a questo istante; ciascuna degli elementi nella navbar è selezionabile e riporta ad una storia presente nella scaletta;

graficamente:

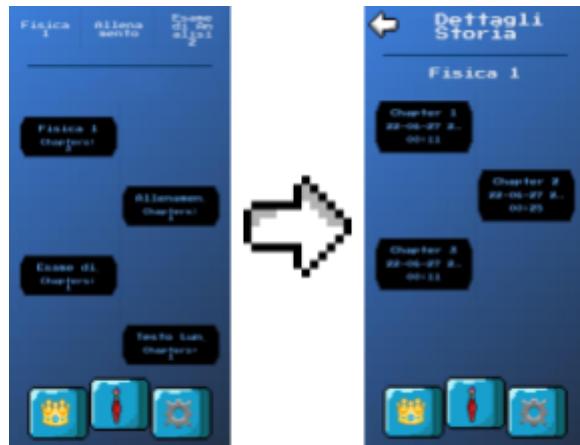


10. I miei capitoli*:

Michelangelo si trova nella schermata “Kingdom” e vuole visualizzare i dettagli di una storia. Dunque, clicca sulla storia di interesse e viene reindirizzato nella schermata “Dettagli storia” che mostra:

- il titolo della storia selezionata;
- tutti i capitoli in cui si articola la storia selezionata;

anche in questo caso, i capitoli sono presentati mediante scaletta;
graficamente:



nella schermata “Dettagli storia”, Michelangelo può:

- tornare alla schermata “Kingdom” premendo il pulsante “indietro”;
 - apprezzare i dettagli di un capitolo cliccando sul capitolo di interesse;
- graficamente:



Nella schermata “Dettagli capitolo”, Michelangelo può:

- tornare indietro mediante il bottone “indietro”;
- visualizzare data ed ora di inizio del capitolo;
- visualizzare il tempo totale della sessione di produttività;
- visualizzare la partizione studio/pausa selezionata;
- visualizzare la modalità opzionale selezionata;
- visualizzare l’avatar selezionato;

11. Impostazioni:

Michelangelo si trova nella schermata “Impostazioni” perché vuole vedere cosa è possibile configurare; gli viene mostrato:

- il riquadro per impostare il nuovo tempo massimo previsto in una sessione studio-pausa;
- uno slider per impostare la partizione studio-pausa di default;
- un pulsante per cambiare tra la modalità mancino e la modalità destroso;
- un pulsante che mostra le info essenziali dell’applicazione;

graficamente:



Analisi dei requisiti

divisione delle user story in task

Nota: non tutte le user story necessitano di essere divise in task; infatti, i task sono approfondimenti dei concetti già espressi nelle user story, utili solo a progettisti e programmatore;

1. User story 3: Avvio storia:

- **Task 1:** autocompletamento: se l'utente vuole inserire, come titolo della storia corrente, un titolo già utilizzato precedentemente, il sistema suggerisce automaticamente il completamento del titolo desiderato;
- **Task 2:** slide seekbar: l'utente definisce la partizione studio-pausa su una linea temporale che, di default, è di 60 min; le partizioni studio/pausa limite configurabili dall'utente sono:

- [10 min studio, 50 min pausa]
- [50 min studio, 10 min pausa]

in altre parole, non è possibile configurare sessioni di produttività che:

- sono costituite da 0 minuti di studio e da 60 min di pausa;
- sono costituite da 60 minuti di studio e da 0 min di pausa;

per concludere, i valori assumibili dai minuti di studio e dai minuti di pausa sono solo ed esclusivamente multipli di 10.

- **Task 3:** modalità silenziosa:

- se si opziona “modalità silenziosa”, il telefono va in modalità silenziosa;

- se si opziona “modalità silenziosa” ed il telefono è già in modalità silenziosa, allora non accade nulla;
- se si deselectiona la “modalità silenziosa”, le suonerie vengono ripristinate;

- **Task 4:** modalità hardcore:

- se si opziona “modalità hardcore”, il telefono va in modalità silenziosa e se si esce dall’app la storia termina automaticamente;
- se si opziona “modalità hardcore” ed il telefono è già in modalità silenziosa, allora viene impostato solo il parametro di terminazione automatica della storia in caso di uscita dall’app;
- se si deselectiona la “modalità hardcore”, le suonerie vengono ripristinate e la storia non termina automaticamente in caso di uscita dall’app;
- se si seleziona “modalità hardcore” e si esce dall’app, l’utente non riceve ricompense in monete a causa della violazione del vincolo comportamentale auto-imposto; in più, i dati della storia non vengono memorizzati;

2. User story 4: Sviluppo storia:

- **Task 1:** selezione casuale: quando si avvia una nuova sessione di produttività, al centro dello schermo, viene mostrato un quadro disegnato in pixel art; in particolare:
 - il quadro è selezionato randomicamente dal sistema;
 - i quadri selezionabili dal sistema sono solo ed esclusivamente quelli sbloccati di default e quelli acquistati dall’utente;
- **Task 2:** algoritmo evolutivo: i quadri evolvono nel tempo; infatti, ogni quadro è costituito da quattro stati; graficamente:



Matematicamente, l’evoluzione dei quadri dipende dal tempo di studio selezionato dall’utente; infatti, tale tempo di studio permette di definire quattro intervalli temporali di evoluzione calcolati nel seguente modo:

- sia x il tempo di studio selezionato;
- sia $n = 4$ il numero degli stati di ogni quadro;

- sia $y = \frac{x}{n}$ il right value del primo intervallo di tempo;
 - il primo intervallo di tempo è $[0, y]$ ed, in esso, viene mostrato lo stato 1 del quadro;
 - il secondo intervallo di tempo è $(y, 2y]$ ed, in esso, viene mostrato lo stato 2 del quadro;
 - il terzo intervallo di tempo è $(2y, 3y]$ ed, in esso, viene mostrato lo stato 3 del quadro;
 - il quarto intervallo di tempo è $(3y, x]$ ed, in esso, viene mostrato lo stato 4 del quadro, ovvero il quadro completo;
 - durante il tempo di pausa viene mostrato il quadro completo.
- **Task 3:** switch avatar-falò: quando il cronometro entra nell'arco di durata della pausa, si ha che:
 - l'avatar selezionato dall'utente viene sostituito da un falò;
 - le frasi di incoraggiamento vengono sostituite da frasi di invito al riposo e da congratulazioni per aver mantenuto la concentrazione per il tempo di studio selezionato;
 - le frasi di invito al riposo e le congratulazioni per aver mantenuto la concentrazione per il tempo di studio selezionato cambiano ogni volta che si sblocca il cellulare;
 - **Task 4:** switch falò-avatar: quando il cronometro esce dall'arco di durata della pausa, si ha che:
 - il falò viene sostituito con l'avatar selezionato dall'utente;
 - le frasi di invito al riposo e le congratulazioni per aver mantenuto la concentrazione per il tempo di studio selezionato vengono sostituite da frasi di incoraggiamento;
- graficamente:



- **Task 5:** ciclicità: la partizione studio-pausa selezionata dall'utente nella schermata di impostazione dei parametri della storia viene ripetuta ciclicamente fino a che non si preme il pulsante “stop” nella schermata

“storia avviata”;

in altre parole, l’algoritmo da implementare è il seguente:

- sia x il tempo di studio selezionato;
- sia y il tempo di pausa selezionato;
- sia $z = x + y$ il tempo di una sessione di studio-pausa;
- sia k il tempo calcolato dal cronometro;

allora, avremo che:

- **passo 1:** se $k < x$, viene mostrato a schermo il quadro da evolvere e l’avatar che esprime frasi di incoraggiamento;
- **passo 2:** se $x < k < z$, viene mostrato a schermo il quadro completo e il falò con frasi di invito al riposo;
- **passo 3:** se $k > z$, allora si ha che:
 - $z += z$
 - $x = x + z$
 - il sistema carica randomicamente un nuovo quadro da dipingere;
 - il nuovo quadro da dipingere selezionato dal sistema deve necessariamente essere diverso da quello selezionato al ciclo precedente.
- ripetiamo i passi 1, 2 e 3 fino a che non si preme il pulsante “stop”;

3. User story 5: Conclusione storia:

- **Task 1:** calcolo coefficiente di merito: il coefficiente di merito è un valore naturale $p \geq 1$ calcolato nel seguente modo:

$$p = f\left(\frac{x}{y}\right) \quad \text{se } \frac{x}{y} > 1$$

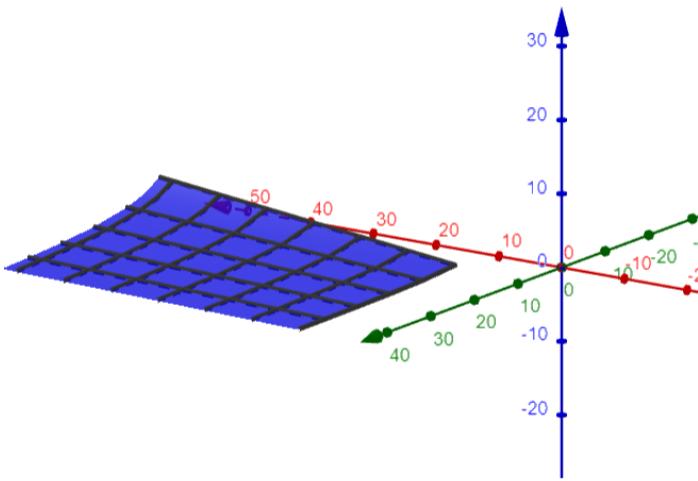
$$p = 1 \quad \text{se } \frac{x}{y} \leq 1$$

$$p \in [1, \infty) \subset N$$

dove:

- f indica il cast ad intero arrotondato per difetto;
- x è il tempo di studio selezionato;
- y è il tempo di pausa selezionato;
- p è il coefficiente di merito

plot:



in particolare, il coefficiente di merito aumenta o diminuisce il numero di monete, guadagnate dall'utente in una sessione di produttività, in base alla seguente logica:

- se il tempo di studio è molto più grande del tempo di pausa, il coefficiente aumenta;
- se il tempo di studio è minore o uguale al tempo di pausa, il coefficiente diminuisce;

in breve, avanziamo un esempio per intendere al meglio quanto appena descritto:

- se $x = 50$, $y = 10$ allora $p = \frac{x}{y} = 5 > 1$; in altre parole, l'utente guadagna 5 monete ogni $x + y = 60$ min;
- se $x = 10$, $y = 50$ allora $p = \frac{x}{y} = 0.2 < 1$; in altre parole, l'utente guadagna 1 moneta ogni $x + y = 60$ min;

- **Task 2:** calcolo delle monete: le monete guadagnate dall'utente in una sessione di produttività, ovvero nel periodo di tempo compreso tra il primo clock del cronometro e la pressione del bottone “stop”, è calcolato mediante la seguente formula conta-monete:

$$m = f(p * \frac{z}{x}) \quad \text{se } z < x + y$$

$$m = f(p * \frac{z}{x+y}) \quad \text{se } z \geq x + y$$

$$m \in N$$

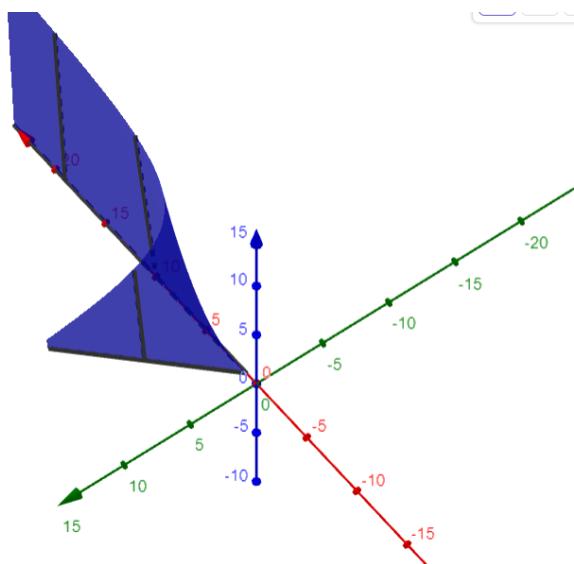
dove:

- f indica il cast ad intero arrotondato per difetto;
- p è il coefficiente di merito;
- z è il tempo calcolato dal cronometro nel momento in cui si preme “stop”;

- x è il tempo di studio selezionato;
- y è il tempo di pausa selezionato;
- m è il numero monete guadagnate in una sessione di produttività;

in particolare, la formula per il calcolo di m cambia a seconda della relazione che lega z ed $x + y$ poiché, in generale, l'utente può ricevere la ricompensa solo quando l'intera sessione di studio-pausa $x + y$ è terminata; tuttavia, poiché la ciclicità delle sessioni studio-pausa è interrotta solo quando si preme il tasto "stop", per evitare di donare monete all'utente anche durante lo studio di pausa si utilizza la formula imposta dalla condizione $z \geq x + y$.

plot:



per intendere al meglio il funzionamento della formula conta-monete, avanziamo il seguente esempio:

- sia $p = 5, z = 30, x = 50$; allora, avremo che

$$m = f(p * \frac{z}{x}) = f(5 * 0.6) = 3$$
 monete;
- sia $p = 5, z = 124, x + y = 60$; allora, avremo che

$$m = f(p * \frac{z}{x+y}) = f(5 * 2.06) = 10$$
 monete;
- **Task 3:** salvataggio dei dati della storia nella schermata "Kingdom": se, nella schermata "Kingdom", è già presente una storia con lo stesso nome di quello inserito dall'user, allora tale nuova storia viene salvata come capitolo della storia già esistente;
 in caso contrario, si crea una nuova storia.

4. User story 9: Le mie storie:

- **Task 1:** struttura delle storie: in relazione alla scaletta delle storie, è importante notare che:

- le storie sono ordinate dalla più recente alla meno recente;
- ogni storia è costituita dal nome e dal numero di capitoli in cui si articola;
- il numero di capitoli di ogni storia è calcolato automaticamente dal sistema; ogni storia ha almeno un capitolo;
- le sessioni di produttività, definite nella schermata di impostazione dei parametri della storia, che hanno un titolo originale costituiscono una nuova storia;
quelle prive di titolo originale, al contrario, costituiscono un capitolo di una storia pre-esistente;
esempio:
 - l'utente avvia una nuova sessione di produttività intitolata “Fisica 1”;
 - quando l'utente termina la sessione di produttività, viene creata una nuova storia nella schermata “Kingdom”;
 - l'utente avvia nuovamente una sessione di produttività intitolata “Fisica 1”;
 - quando l'utente termina la sessione di produttività, viene creato un nuovo capitolo della storia “Fisica 1” nella schermata “Kingdom”; ecc...
- le storie sono composte da capitoli poiché, in questo modo, l'utente può intendere quanto tempo e quanti sforzi dedica ad una singola attività;

5. User story 11: Impostazioni:

- **task 1:** massimo e minimo tempo limite impostabile:
 - il minimo tempo limite impostabile dall'utente è di 60 min;
 - il massimo tempo limite impostabile dall'utente è di 120 min;
- **Task 2:** arrotondamento a multipli di 10: se il massimo tempo limite impostabile dall'utente non è un multiplo di 10, esso:
 - viene arrotondato al multiplo di 10 immediatamente precedente se l'ultima cifra è compresa tra 1 e 4;
ad esempio, 64 viene automaticamente arrotondato a 60;
 - viene arrotondato al multiplo di 10 immediatamente successivo se l'ultima cifra è compresa tra 5 e 9;
ad esempio, 66 viene automaticamente arrotondato a 70;
- **Task 3:** modalità destroso:
 - mostra l'icona della moneta a destra nella schermata principale;
 - il pulsante “Corona” è a sinistra;
 - il pulsante “impostazioni” è a destra;

- **Task 4:** modalità mancino:

- mostra l'icona della moneta a sinistra nella schermata principale;
- il pulsante “Corona” è a destra;
- il pulsante “impostazioni” è a sinistra;

in poche parole, modalità mancino e modalità destroso rendono le regioni dell’UI più accessibili a seconda della mano dominante.

Analisi dei requisiti

glossario dei termini

Termine	Descrizione	Tipo	Sinonimi
Storia	Sessione di produttività compresa tra il primo ciclo di clock del cronometro e la pressione del tasto “stop” da parte dell’utente; ogni storia deve essere almeno di 10 secondi	Termine specifico dell’app	Sessione di produttività
Modalità silenziosa	Modalità che sostituisce il suono delle notifiche e delle chiamate con vibrazione	Smartphone feature	/
Uscire dall’app	Porre l’app in background aprendo applicazioni terze o premendo i tasti home, menù ed indietro; l’uscita dall’app non include la chiusura mediante task-manager	Smartphone feature	/
Avatar	Personaggio selezionabile dall’utente	Videogames	/
Sblocco del cellulare	Wake up del device dopo che esso è stato messo in sleep mode dall’utente o da un tempo prolungato di inattività; wake up ⇒ device diviene interattivo sleep mode ⇒ device perde interattività	Smartphone feature	/
Capitolo	Storia il cui titolo è già stato precedentemente utilizzato dall’utente nel sistema	Termine specifico dell’app	/

Analisi dei requisiti

requisiti utente: requisiti funzionali e non funzionali

Schematizzazione generale dei requisiti del sistema



Requisiti funzionali e descrizione strutturata

Funzionali	
Iniziare una nuova storia <ul style="list-style-type: none"> + RF1: Scrivere il titolo della storia + RF2: Selezionare la partizione studio/pausa + RF3: Possibilità di impostare la modalità silenziosa + RF4: Possibilità di impostare la modalità hardcore + RF5: Selezionare l'avatar + RF6: Avviare la sessione di produttività 	Sviluppare una nuova storia <ul style="list-style-type: none"> + RF7: Calcolare il tempo trascorso mediante cronometro + RF8: Mostrare avatar selezionato dall'utente e falò in relazione al tempo trascorso + RF9: Mostrare frasi di incoraggiamento + RF10: Mostrare frasi di invito al riposo e di congratulazioni + RF11: Selezionare randomicamente il quadro da evolvere + RF12: Evolvere il quadro in relazione al tempo trascorso + RF13: Selezionare randomicamente un nuovo quadro quando un ciclo studio/pausa termina fino a che l'utente non preme il tasto "stop"
Concludere una nuova storia <ul style="list-style-type: none"> + RF14: Concludere la storia + RF15: Calcolare le monete guadagnate + RF16: Memorizzare i dati della storia conclusa 	Navigare nello shop <ul style="list-style-type: none"> + RF17: Scorrere gli avatar acquistabili e quelli da acquistare + RF18: Scorrere i quadri acquistabili e quelli da acquistare + RF19: Calcolare le monete a disposizione + RF20: Visualizzare i dettagli dell'opera d'arte selezionata + RF21: Acquistare l'opera d'arte verificando, prima, che si abbiano monete a sufficienza
Consultare i propri progressi <ul style="list-style-type: none"> + RF22: Navigare tra le storie mediante scorrimento + RF23: Shortcut storia mediante click della navbar + RF24: Navigare tra capitoli mediante scorrimento + RF25: Produrre info sul capitolo selezionato 	Configurare le preferenze <ul style="list-style-type: none"> + RF26: Impostare il massimo tempo limite di una sessione studio-pausa + RF27: Controllare che il massimo tempo limite sia un valore multiplo di 10 + RF28: Impostare la partizione studio-pausa di default + RF29: Possibilità di impostare la modalità mancino + RF30: Possibilità di impostare la modalità destrorso

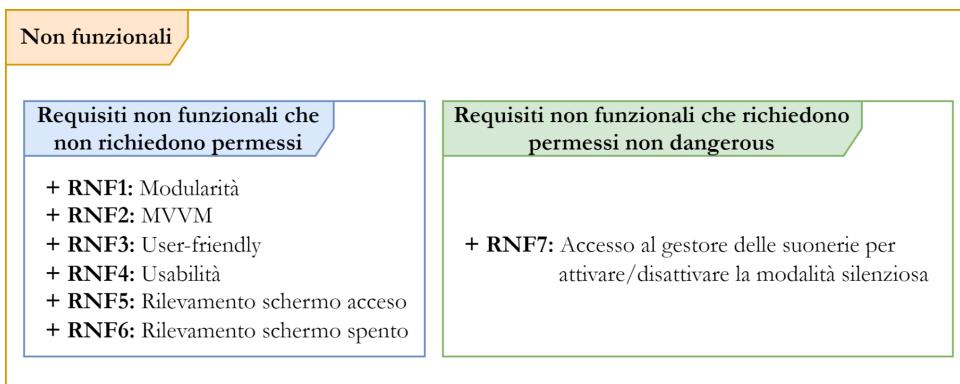
Requisito	Descrizione
RF1: Scrivere il titolo della storia	L'utente deve necessariamente scrivere, mediante tastiera, il titolo della storia che sta avviando; in caso contrario, il sistema impedisce l'avvio della sessione di produttività
RF2: Selezionare la partizione studio/pausa	L'utente seleziona, mediante slide con il dito, la partizione studio-pausa desiderata; se l'user non interagisce con la seekbar, viene impostata la partizione di default
RF3: Possibilità di impostare la modalità silenziosa	L'utente attiva la modalità silenziosa, ovvero disattiva le suonerie e attiva la vibrazione
RF4: Possibilità di impostare la modalità hardcore	L'utente attiva la modalità silenziosa e, se esce dall'app, la sua storia termina
RF5: Selezionare l'avatar	L'utente seleziona, mediante bottoni per lo scorrimento, l'avatar desiderato; se l'utente non seleziona alcun avatar, il sistema considera l'opzione di default
RF6: Avviare la sessione di produttività	L'utente avvia ufficialmente la propria sessione di produttività; a seconda delle impostazioni opzionali selezionate, l'app rileva se viene posta in background
RF7: Calcolare il tempo trascorso mediante cronometro	Mediante l'uso del clock che traccia il tempo di attività totale del device, il cronometro dell'app calcola il tempo della storia
RF8: Mostrare avatar selezionato dall'utente e falò in relazione al tempo trascorso	La scelta dell'avatar da parte dell'utente deve essere memorizzata e mostrata nelle schermate di interesse; se l'istante di tempo calcolato dal cronometro appartiene al tempo di studio, allora viene mostrato a schermo l'avatar; al contrario, se l'istante di tempo calcolato dal cronometro appartiene al tempo di pausa, allora viene mostrato a

	schermo il falò;
RF9: Mostrare frasi di incoraggiamento	Durante il tempo di studio, ogni volta che l'utente sblocca il telefono viene selezionata e mostrata randomicamente una nuova frase motivazionale
RF10: Mostrare frasi di invito al riposo e di congratulazioni	Durante il tempo di pausa, ogni volta che l'utente sblocca il telefono viene selezionata e mostrata randomicamente una nuova frase di invito al riposo o di congratulazioni
RF11: Selezionare randomicamente il quadro da evolvere	Il sistema seleziona randomicamente il quadro al centro dello schermo
RF12: Evolvere il quadro in relazione al tempo trascorso	Il tempo di studio è scisso in 4 intervalli temporali; in base all'intervallo temporale a cui appartiene l'istante corrente calcolato dal cronometro, il quadro evolve; l'utente vede il quadro completo al termine del tempo di studio
RF13: Selezionare randomicamente un nuovo quadro quando un ciclo studio/pausa termina fino a che l'utente non preme il tasto "stop"	Quando il tempo calcolato dal cronometro eccede il tempo studio+pausa, viene selezionato un nuovo quadro (facendo attenzione che sia diverso dal precedente) e si ripete il ciclo evolutivo; tale algoritmo viene ripetuto finché l'utente non termina esplicitamente la propria sessione di produttività
RF14: Concludere la storia	La sessione di produttività corrente può essere chiusa mediante apposito bottone "stop" o, alternativamente, ponendo l'app in background se si è in modalità hardcore
RF15: Calcolare le monete guadagnate	Ogni sessione di produttività, una volta terminata, ricompensa l'utente calcolando un numero di monete pari all'impegno dell'utente stesso; l'impegno dell'utente è determinato in base alla partizione studio-pausa e al tempo effettivo calcolato dal

	cronometro
RF16: Memorizzare i dati della storia conclusa	Una volta eseguito il calcolo delle monete, i dati della storia vanno memorizzati nella schermata “Kingdom”
RF17: Scorrere gli avatar acquistabili e quelli da acquistare	Mediante frecce direzionali, l'utente sfoglia il catalogo degli avatar acquistabili/acquistati
RF18: Scorrere i quadri acquistabili e quelli da acquistare	Mediante frecce direzionali, l'utente sfoglia il catalogo dei quadri acquistabili/acquistati
RF19: Calcolare le monete a disposizione	L'utente vede le monete guadagnate fino a questo istante
RF20: Visualizzare i dettagli dell'opera d'arte selezionata	Ogni opera d'arte prevede una breve storia integrativa
RF21: Acquistare l'opera d'arte verificando, prima, che si abbiano monete a sufficienza	La procedura di acquisto di un'opera d'arte prevede, necessariamente, che le monete possedute siano sufficienti ad effettuare la transazione
RF22: Navigare tra le storie mediante scorrimento	Le lista delle storie dell'utente deve consultabile mediante scorrimento verticale
RF23: Shortcut storia mediante click della navbar	Le storie dell'utente sono riportate in una navbar; per trovare velocemente una storia sita nella recyclerView, è sufficiente cliccare la storia nella navbar
RF24: Navigare tra capitoli mediante scorrimento	Le lista dei capitoli di ogni storia dell'utente deve consultabile mediante scorrimento verticale
RF25: Produrre info sul capitolo selezionato	Quando l'utente accede al capitolo di una storia, vengono mostrate informazioni a riguardo
RF26: Impostare il massimo tempo limite di una sessione studio-pausa	l'utente può impostare il massimo tempo limite di una sessione studio-pausa in modo da rispettare tecniche di studio quali pomodoro, ecc

<p>RF27: Controllare che il massimo tempo limite sia un valore multiplo di 10</p>	<p>Se il massimo tempo limite impostato dall'utente non è un multiplo di 10, esso viene arrotondato: - per eccesso se l'ultima unità è compresa tra 5 e 9; - per difetto altrimenti</p>
<p>RF28: Impostare la partizione studio-pausa di default</p>	<p>Impostare, mediante slide con il dito, la partizione studio-pausa che verrà proposta, come partizione di default, nella schermata di configurazione dei parametri di una nuova storia</p>
<p>RF29: Possibilità di impostare la modalità mancino</p>	<p>La modalità mancino sposta alcuni elementi della UI in modo che siano più raggiungibili per i mancini</p>
<p>RF30: Possibilità di impostare la modalità destrorso</p>	<p>La modalità destrorso sposta alcuni elementi della UI in modo che siano più raggiungibili per i destrorsi</p>

Requisiti non funzionali e descrizione strutturata



Requisito	Descrizione
<p>RNF1: Modularità</p>	<p>Il sistema dovrà essere composto da moduli (classi) e da file che hanno riferimenti ad altri file (classi e .XML)</p>
<p>RNF2: MVVM</p>	<p>Per promuovere un'architettura indipendente dal ciclo di vita delle singole activity, si è deciso di implementare il pattern architettonicale Model-View-ViewModel</p>

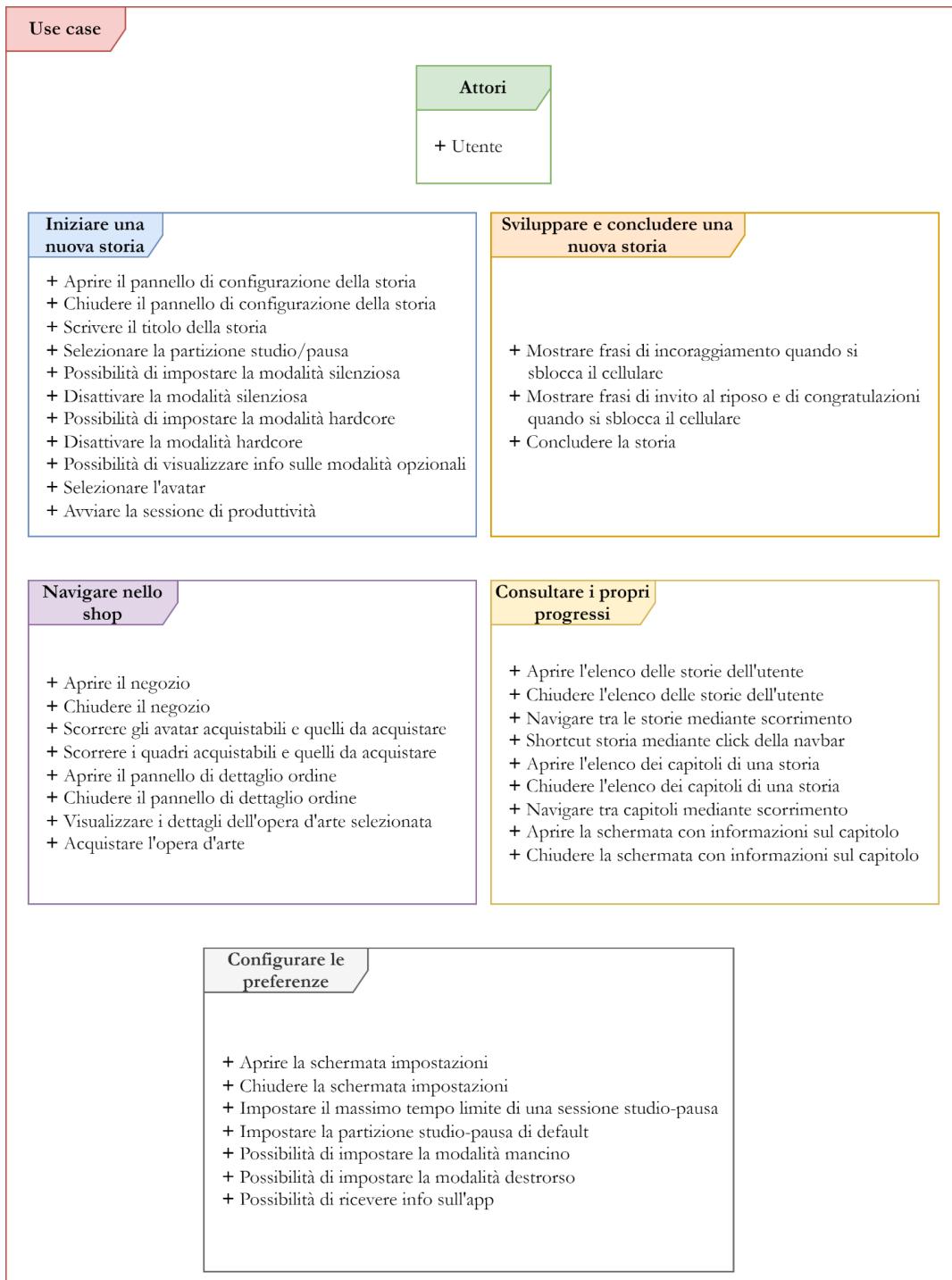
RNF3: User-friendly	L'interfaccia grafica del sistema dovrà essere facilmente utilizzabile da ogni utente
RNF4: Usabilità	Il sistema dovrà essere usabile per il tipo di utenti per i quali è progettato
RNF5: Rilevamento schermo acceso	Il sistema deve essere in grado di rilevare quando l'utente accende lo schermo; infatti, l'accensione dello schermo è un trigger per eseguire alcune funzionalità durante la sessione di produttività corrente
RNF6: Rilevamento schermo spento	Il sistema deve essere in grado di rilevare quando l'utente spegne lo schermo; infatti, lo spegnimento dello schermo è un trigger per eseguire alcune funzionalità durante la sessione di produttività corrente
RNF7: Accesso al gestore delle suonerie per attivare/disattivare la modalità silenziosa	Il sistema deve poter accedere al gestore delle suonerie per poter attivare/disattivare la modalità silenziosa

Analisi dei requisiti

requisiti funzionali sistema: UML use-case diagram

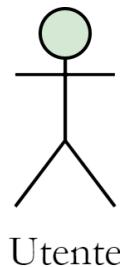
Utilizziamo la seguente notazione insiemistica per dividere, in aree funzionali, i diagrammi dei casi d'uso:

Nota: per rilevare facilmente ciascuna delle aree funzionali nelle sezioni di documento che seguono, è possibile utilizzare il colore.



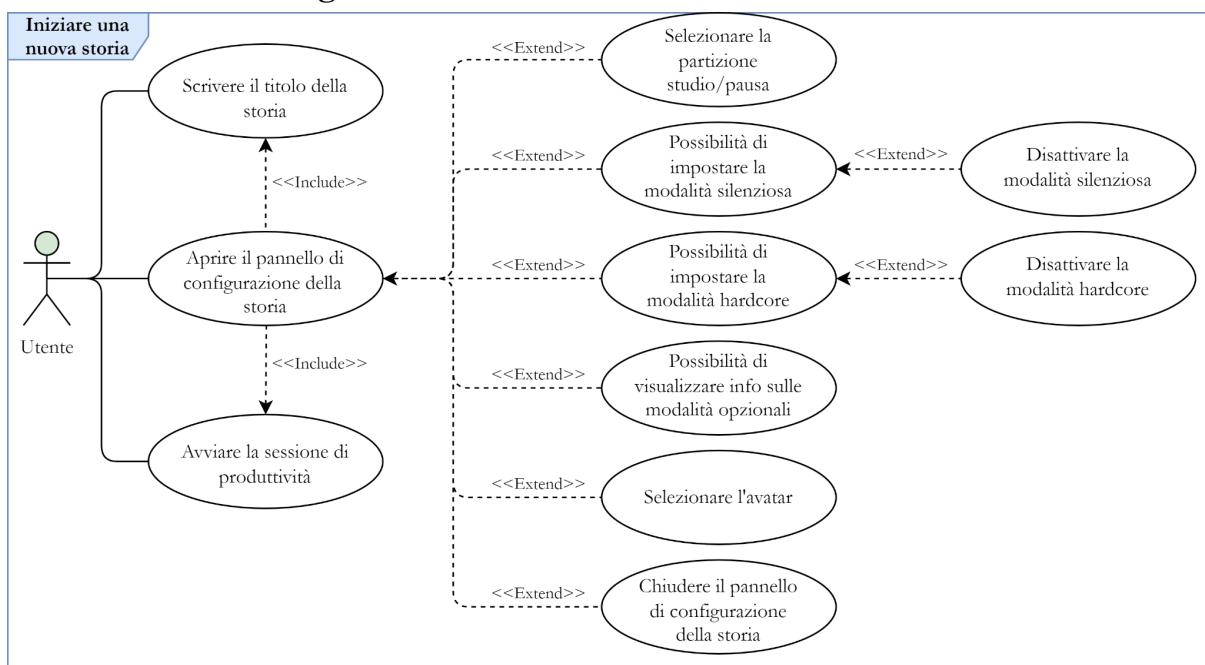
Attori use case

Poiché l'oggetto della trattazione è un software di produttività che non richiede autenticazione per erogare servizi, l'unico attore ad interagire con il sistema è l'utente generico.



Utente

Diagramma use case: iniziare una nuova storia



Descrizione strutturata dei casi d'uso

1. Use case: Aprire il pannello di configurazione della storia

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende avviare una nuova sessione di produttività.
- **Pre-condizioni:** Nessuna.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente vuole avviare una nuova sessione di produttività.

- b.** il sistema mostra il tasto di configurazione dei parametri della storia.
- c.** l'utente preme il tasto di configurazione dei parametri della storia.
- d.** il sistema mostra il pannello di configurazione della storia.

- **Sequenza degli eventi alternativa:** Nessuna.

2. Use case: Scrivere il titolo della storia

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende assegnare un nome alla propria sessione di produttività.
In particolare, il titolo è l'unico parametro ad essere dichiarato <<Include>> poiché non prevede valori di default;
La scelta di non assegnare valori di default al titolo non è dovuta a difficoltà implementative, ma è dovuta alla volontà dei progettisti di costringere l'utente ad assegnare identità alle attività che lo formano.
- **Pre-condizioni:** il pannello di configurazione della storia deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a.** il caso d'uso inizia quando un utente intende assegnare un nome alla propria sessione di produttività.
 - b.** il sistema mostra la view “editText” per ricevere il testo in input.
 - c.** l'utente preme la view “editText” per immettere il testo in input.
 - d.** il sistema mostra la tastiera all'utente.
 - e.** l'utente immette il titolo mediante tastiera.
 - f.** il sistema mostra il titolo della storia appena immesso.
- **Sequenza degli eventi alternativa:** Nessuna.

3. Use case: Avviare la sessione di produttività

- **Descrizione:** Questo caso d'uso si verifica quando un utente ha configurato tutti i parametri della sua storia ed intende avviare la propria sessione di produttività.
- **Pre-condizioni:** il pannello di configurazione della storia deve essere aperto ed il titolo della storia deve essere stato immesso.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a.** il caso d'uso inizia quando un utente, dopo aver configurato tutti i parametri della sua storia, intende avviare la propria sessione di produttività mediante apposito bottone.
 - b.** il sistema mostra il bottone per avviare la storia.
 - c.** l'utente preme il bottone per avviare la storia.
 - d.** il sistema mostra la schermata relativa allo svolgimento della storia.

- **Sequenza degli eventi alternativa:** Nessuna.

4. Use case: Selezionare la partizione studio-pausa

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende selezionare la partizione studio-pausa.
- **Pre-condizioni:** il pannello di configurazione della storia deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende selezionare la partizione studio-pausa.
 - b. il sistema mostra la seekbar che consente di impostare la partizione studio-pausa desiderata.
 - c. l'utente interagisce con la seekbar mediante slide con il dito.
 - d. il sistema legge l'input da sliding.
 - e. il sistema mostra a schermo il tempo di studio ed il tempo di pausa a seconda della posizione del dito nella seekbar.
- **Sequenza degli eventi alternativa:** l'utente non interagisce con la seekbar e, di conseguenza, il sistema considera la partizione di default.

5. Use case: Possibilità di impostare la modalità silenziosa

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende attivare la modalità silenziosa.
- **Pre-condizioni:** il pannello di configurazione della storia deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende attivare la modalità silenziosa.
 - b. il sistema mostra lo switch che consente di attivare la modalità silenziosa.
 - c. l'utente interagisce con lo switch mediante tap con il dito.
 - d. il sistema legge l'input da tocco.
 - e. il sistema notifica all'utente, mediante vibrazione, che la modalità silenziosa è stata attivata.
- **Prima sequenza degli eventi alternativa:** se l'utente ha già attivato la modalità silenziosa, l'attivazione dello switch non genera alcuna callback.
- **Seconda sequenza degli eventi alternativa:** l'utente non interagisce con lo switch per l'attivazione della modalità silenziosa.

6. Use case: Disattivare la modalità silenziosa

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende disattivare la modalità silenziosa.
- **Pre-condizioni:** il pannello di configurazione della storia deve essere aperto e l'utente deve aver attivato la modalità silenziosa mediante switch.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende disattivare la modalità silenziosa.
 - b. il sistema mostra lo switch che consente di disattivare la modalità silenziosa.
 - c. l'utente interagisce con lo switch mediante tap con il dito.
 - d. il sistema legge l'input da tocco.
 - e. il sistema notifica all'utente che la modalità silenziosa è stata disattivata e che le suonerie sono state ripristinate.
- **Sequenza degli eventi alternativa:** Nessuna.

7. Use case: Possibilità di impostare la modalità hardcore

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende attivare la modalità hardcore.
- **Pre-condizioni:** il pannello di configurazione della storia deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende attivare la modalità hardcore.
 - b. il sistema mostra lo switch che consente di attivare la modalità hardcore.
 - c. l'utente interagisce con lo switch mediante tap con il dito.
 - d. il sistema legge l'input da tocco.
 - e. il sistema attiva la modalità silenziosa.
 - f. il sistema notifica all'utente, mediante vibrazione, che la modalità hardcore è stata attivata.
- **Sequenza degli eventi alternativa:** l'utente non interagisce con lo switch per l'attivazione della modalità hardcore.

8. Use case: Disattivare la modalità hardcore

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende disattivare la modalità hardcore.

- **Pre-condizioni:** il pannello di configurazione della storia deve essere aperto e l'utente deve aver attivato la modalità hardcore mediante switch.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende disattivare la modalità hardcore.
 - b. il sistema mostra lo switch che consente di disattivare la modalità hardcore.
 - c. l'utente interagisce con lo switch mediante tap con il dito.
 - d. il sistema legge l'input da tocco.
 - e. il sistema disattiva la modalità silenziosa.
 - f. il sistema notifica all'utente che la modalità hardcore è stata disattivata e che le suonerie sono state ripristinate.
- **Sequenza degli eventi alternativa:** Nessuna.

9. Use case: Possibilità di visualizzare info sulle modalità opzionali

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende visualizzare info sulle modalità opzionali.
- **Pre-condizioni:** il pannello di configurazione della storia deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende visualizzare info sulle modalità opzionali.
 - b. il sistema mostra il pulsante che consente di visualizzare info sulle modalità opzionali.
 - c. l'utente interagisce con il pulsante mediante tap con il dito.
 - d. il sistema legge l'input da tocco.
 - e. il sistema disattiva mostra le info sulle modalità opzionali.
- **Sequenza degli eventi alternativa:** l'utente non interagisce con il pulsante che consente di visualizzare info sulle modalità opzionali.

10. Use case: Selezionare l'avatar

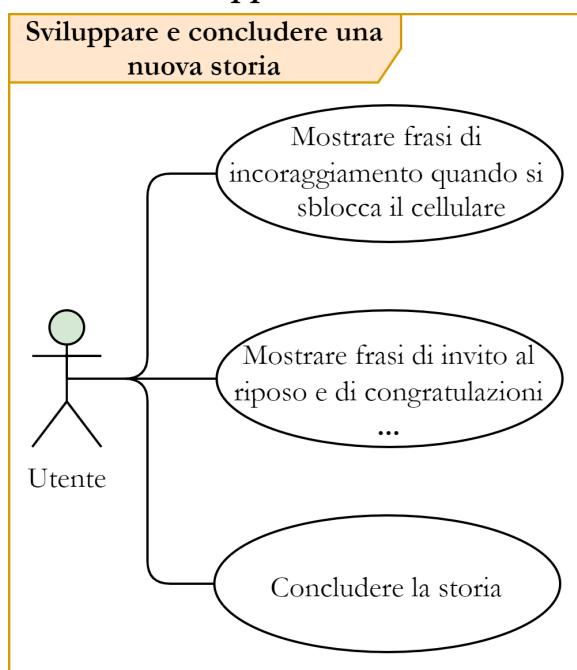
- **Descrizione:** Questo caso d'uso si verifica quando un utente intende selezionare un avatar.
- **Pre-condizioni:** il pannello di configurazione della storia deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende selezionare un avatar.

- b. il sistema mostra i pulsanti direzionali che consentono di scorrere gli avatar disponibili.
- c. l'utente interagisce con i pulsanti mediante tap con il dito.
- d. il sistema legge l'input da tocco.
- e. il sistema mostra un nuovo avatar ogni volta che l'utente preme i pulsanti direzionali.
- **Sequenza degli eventi alternativa:** l'utente non interagisce con i pulsanti e viene automaticamente selezionato l'avatar di default.

11. Use case: Chiudere il pannello di configurazione della storia

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende chiudere il pannello di configurazione della storia.
- **Pre-condizioni:** il pannello di configurazione della storia deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende chiudere il pannello di configurazione della storia.
 - b. il sistema mostra il pulsante “indietro”.
 - c. l'utente interagisce con il pulsante “indietro”.
 - d. il sistema legge l'input da tocco.
 - e. il sistema chiude il pannello di configurazione della storia.
- **Sequenza degli eventi alternativa:** Nessuna.

Diagramma use case: sviluppare e concludere una nuova storia

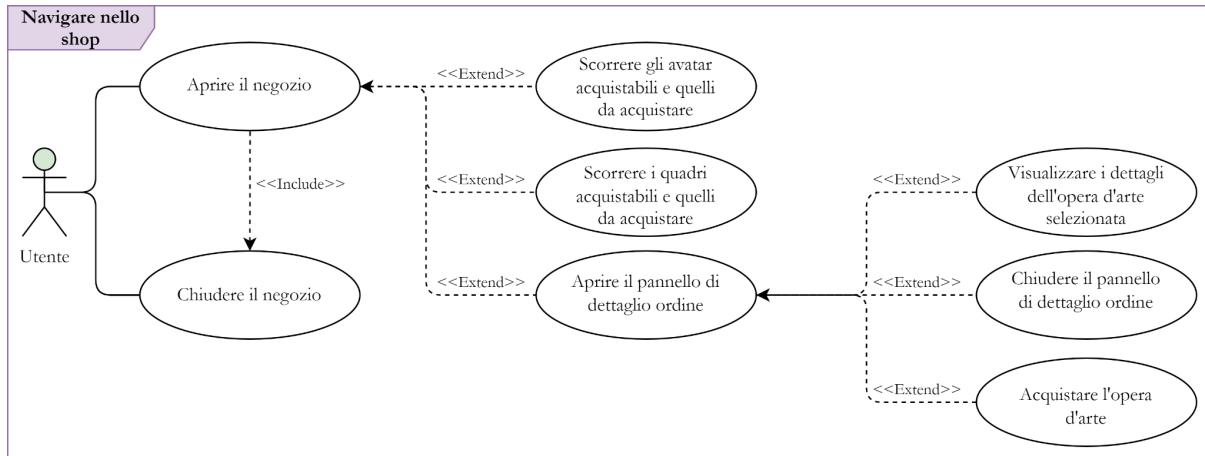


Descrizione strutturata dei casi d'uso

1. **Use case:** Mostrare frasi di incoraggiamento quando si sblocca il cellulare
 - **Descrizione:** Questo caso d'uso si verifica quando un utente sblocca il cellulare mentre è in corso una sessione di produttività.
 - **Pre-condizioni:** Nessuna.
 - **Post-condizioni:** Nessuna.
 - **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente sblocca il cellulare mentre è in corso una sessione di produttività.
 - b. l'utente sblocca il cellulare durante il tempo di studio.
 - c. il sistema rileva l'unlock del device.
 - d. il sistema mostra frasi di incoraggiamento all'utente.
 - **Sequenza degli eventi alternativa:** Use case: Mostrare frasi di invito al riposo e di congratulazioni quando si sblocca il cellulare.
2. **Use case:** Mostrare frasi di invito al riposo e di congratulazioni quando si sblocca il cellulare
 - **Descrizione:** Questo caso d'uso si verifica quando un utente sblocca il cellulare mentre è in corso una sessione di produttività.
 - **Pre-condizioni:** Nessuna.
 - **Post-condizioni:** Nessuna.
 - **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente sblocca il cellulare mentre è in corso una sessione di produttività.
 - b. l'utente sblocca il cellulare durante il tempo di pausa.
 - c. il sistema rileva l'unlock del device.
 - d. il sistema mostra frasi di invito al riposo e di congratulazioni all'utente.
 - **Sequenza degli eventi alternativa:** Use case: Mostrare frasi di incoraggiamento quando si sblocca il cellulare.
3. **Use case:** Concludere la storia
 - **Descrizione:** Questo caso d'uso si verifica quando un utente intende concludere la storia
 - **Pre-condizioni:** Nessuna.
 - **Post-condizioni:** Nessuna.
 - **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende concludere la storia.
 - b. il sistema mostra il pulsante per concludere la storia.

- c. l'utente interagisce con il pulsante mediante input touch.
- d. il sistema rileva la pressione del pulsante.
- e. il sistema conclude la sessione di produttività.
- **Sequenza degli eventi alternativa:** se la modalità hardcore è attiva e l'utente pone WIP in background, il sistema conclude la sessione di produttività corrente.

Diagramma use case: navigare nello shop



Descrizione strutturata dei casi d'uso

1. Use case: Aprire il negozio

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende aprire il negozio
- **Pre-condizioni:** Nessuna.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende aprire il negozio.
 - b. il sistema mostra il pulsante d'accesso allo shop.
 - c. l'utente interagisce con il pulsante mediante input touch.
 - d. il sistema rileva la pressione del pulsante.
 - e. il sistema mostra a schermo lo shop.
- **Sequenza degli eventi alternativa:** Nessuna.

2. Use case: Chiudere il negozio:

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende chiudere il negozio
- **Pre-condizioni:** lo shop deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**

- a. il caso d'uso inizia quando un utente intende chiudere il negozio.
- b. il sistema mostra il bottom menù che consente di uscire dallo shop.
- c. l'utente interagisce con il bottom menù.
- d. il sistema rileva l'interazione.
- e. il sistema esce dallo shop.

- **Sequenza degli eventi alternativa:** Nessuna.

3. Use case: Scorrere gli avatar acquistabili e quelli da acquistare

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende visualizzare gli avatar acquistabili e quelli già acquistati
- **Pre-condizioni:** lo shop deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende visualizzare gli avatar acquistabili e quelli già acquistati.
 - b. il sistema mostra i pulsanti di scorrimento degli avatar.
 - c. l'utente interagisce con i pulsanti.
 - d. il sistema rileva l'interazione.
 - e. il sistema mostra a schermo un nuovo avatar ogni qualvolta si verifica un'interazione.
- **Sequenza degli eventi alternativa:** se l'utente non preme alcun pulsante di navigazione, viene mostrato l'avatar di default, indipendentemente dal fatto che sia stato acquistato o meno.

4. Use case: Scorrere i quadri acquistabili e quelli da acquistare

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende visualizzare i quadri acquistabili e quelli già acquistati
- **Pre-condizioni:** lo shop deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende visualizzare i quadri acquistabili e quelli già acquistati.
 - b. il sistema mostra i pulsanti di scorrimento dei quadri.
 - c. l'utente interagisce con i pulsanti.
 - d. il sistema rileva l'interazione.
 - e. il sistema mostra a schermo un nuovo quadro ogni qualvolta si verifica un'interazione.
- **Sequenza degli eventi alternativa:** se l'utente non preme alcun pulsante di navigazione, viene mostrato il quadro di default, indipendentemente dal fatto che sia stato acquistato o meno.

5. Use case: Aprire il pannello di dettaglio ordine

- **Descrizione:** Questo caso d'uso si verifica quando un utente tocca un avatar o un quadro nello shop
- **Pre-condizioni:** lo shop deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente tocca un avatar o un quadro nello shop.
 - b. il sistema mostra avatar e quadri.
 - c. l'utente interagisce con l'avatar o con il quadro desiderato mediante input touch.
 - d. il sistema rileva l'interazione.
 - e. il sistema mostra a schermo il pannello di dettaglio ordine.
- **Sequenza degli eventi alternativa:** Nessuna.

6. Use case: Visualizzare i dettagli dell'opera d'arte selezionata

- **Descrizione:** Questo caso d'uso si verifica quando un utente vuole conoscere la storia dell'opera d'arte selezionata
- **Pre-condizioni:** lo shop deve essere aperto e il pannello di dettaglio ordine deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente vuole conoscere la storia dell'opera d'arte selezionata.
 - b. il sistema mostra il pulsante “info”.
 - c. l'utente interagisce con il pulsante “info”.
 - d. il sistema rileva l'interazione.
 - e. il sistema mostra a schermo la storia dell'opera d'arte selezionata dall'utente.
- **Sequenza degli eventi alternativa:** Nessuna.

7. Use case: Acquistare l'opera d'arte

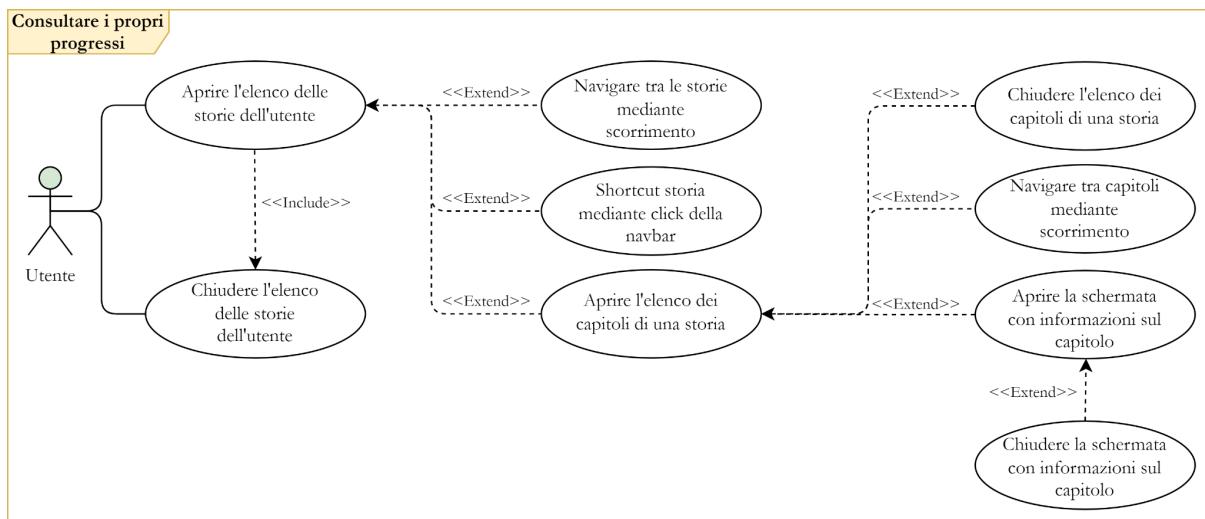
- **Descrizione:** Questo caso d'uso si verifica quando un utente intende acquistare un'opera d'arte
- **Pre-condizioni:** lo shop deve essere aperto e il pannello di dettaglio ordine deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**

- a. il caso d'uso inizia quando un utente intende acquistare un'opera d'arte.
 - b. il sistema mostra il pulsante d'acquisto.
 - c. l'utente interagisce con il pulsante d'acquisto.
 - d. il sistema rileva l'interazione.
 - e. il sistema mostra a schermo un messaggio che chiede, all'utente, se è sicuro di voler effettuare l'acquisto.
 - f. l'utente clicca "si".
 - g. il sistema rileva l'interazione.
 - h. il sistema sottrae le monete richieste all'acquisto dalle monete possedute dall'utente
 - i. il sistema mostra a schermo un messaggio che conferma l'esito positivo della transazione.
- **Prima sequenza degli eventi alternativa:** abbiamo appena superato il punto "e" della sequenza degli eventi principale:
 - h. l'utente clicca "no".
 - i. il sistema rileva l'interazione.
 - j. il sistema annulla l'operazione di checkout.
- **Seconda sequenza degli eventi alternativa:** abbiamo appena superato il punto "g" della sequenza degli eventi principale:
 - h. il sistema osserva che le monete possedute dall'utente non sono sufficienti all'acquisto dell'opera d'arte selezionata.
 - i. il sistema annulla l'operazione di checkout.

8. Use case: Chiudere il pannello di dettaglio ordine

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende uscire dal pannello di dettaglio ordine
- **Pre-condizioni:** lo shop deve essere aperto e il pannello di dettaglio ordine deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende uscire dal pannello di dettaglio ordine.
 - b. il sistema mostra il pulsante "indietro".
 - c. l'utente interagisce con il pulsante "indietro".
 - d. il sistema rileva l'interazione.
 - e. il sistema chiude il pannello di dettaglio ordine.
- **Sequenza degli eventi alternativa:** Nessuna.

Diagramma use case: consultare i propri progressi



Descrizione strutturata dei casi d'uso

1. Use case: Aprire l'elenco delle storie dell'utente

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende visualizzare lo storico delle sessioni di produttività svolte
- **Pre-condizioni:** Nessuna.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende visualizzare lo storico delle sessioni di produttività svolte.
 - b. il sistema mostra il pulsante d'accesso allo storico delle sessioni di produttività svolte.
 - c. l'utente interagisce con il pulsante mediante input touch.
 - d. il sistema rileva la pressione del pulsante.
 - e. il sistema mostra a schermo lo storico delle sessioni di produttività svolte.
- **Sequenza degli eventi alternativa:** Nessuna.

2. Use case: Chiudere l'elenco delle storie dell'utente

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende chiudere lo storico delle sessioni di produttività svolte
- **Pre-condizioni:** lo storico delle sessioni di produttività svolte dall'utente deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende chiudere lo storico delle sessioni di produttività svolte.

- b.** il sistema mostra il bottom menù.
 - c.** l'utente interagisce con il bottom menù.
 - d.** il sistema rileva l'interazione.
 - e.** il sistema chiude lo storico delle sessioni di produttività svolte.
 - **Sequenza degli eventi alternativa:** Nessuna.
- 3. Use case:** Navigare tra le storie mediante scorrimento
- **Descrizione:** Questo caso d'uso si verifica quando un utente intende visualizzare le storie non mostrate a schermo
 - **Pre-condizioni:** lo storico delle sessioni di produttività svolte dall'utente deve essere aperto.
 - **Post-condizioni:** Nessuna.
 - **Sequenza degli eventi principale:**
 - a.** il caso d'uso inizia quando un utente intende visualizzare le storie non mostrate a schermo.
 - b.** in generale, per ragioni legate alla lunghezza dei device, il sistema mostra a schermo un sottoinsieme di storie.
 - c.** l'utente interagisce con lo schermo effettuando scrolling verso il basso o verso l'alto al fine di cercare la storia desiderata.
 - d.** il sistema rileva l'interazione.
 - e.** il sistema permette la navigazione tra le storie mediante scorrimento.
 - **Sequenza degli eventi alternativa:** se tutte le storie entrano nello schermo, la funzionalità di scrolling viene temporaneamente disabilitata.

4. Use case: Shortcut storia mediante click della navbar

- **Descrizione:** Questo caso d'uso si verifica quando un utente vuole essere reindirizzato ad una storia mediante click di un elemento nella navbar
- **Pre-condizioni:** lo storico delle sessioni di produttività svolte dall'utente deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a.** il caso d'uso inizia quando un utente vuole essere reindirizzato ad una storia mediante click di un elemento nella navbar.
 - b.** il sistema mostra la navbar.
 - c.** l'utente interagisce con la navbar scorrendo tra gli elementi disponibili mediante sliding.
 - d.** il sistema rileva l'interazione.
 - e.** il sistema aggiorna il contenuto nella navbar in base allo sliding.
 - f.** l'utente interagisce con la navbar eseguendo un tap sulla storia desiderata.

- g. il sistema rileva l'interazione.
- h. il sistema reindirizza l'utente alla storia desiderata.

- **Sequenza degli eventi alternativa:** Nessuna.

5. Use case: Aprire l'elenco dei capitoli di una storia

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende aprire l'elenco dei capitoli di una storia
- **Pre-condizioni:** lo storico delle sessioni di produttività svolte dall'utente deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende aprire l'elenco dei capitoli di una storia.
 - b. il sistema mostra le storie disponibili a schermo.
 - c. l'utente esegue tap sulla storia desiderata.
 - d. il sistema rileva l'interazione.
 - e. il sistema mostra l'elenco dei capitoli relativi alla storia selezionata.
- **Sequenza degli eventi alternativa:** Nessuna.

6. Use case: Chiudere l'elenco dei capitoli di una storia

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende chiudere l'elenco dei capitoli di una storia
- **Pre-condizioni:** lo storico delle sessioni di produttività svolte dall'utente deve essere aperto; l'elenco dei capitoli di una storia deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende chiudere l'elenco dei capitoli di una storia.
 - b. il sistema mostra il tasto di uscita.
 - c. l'utente preme il tasto di uscita.
 - d. il sistema rileva l'interazione.
 - e. il sistema chiude l'elenco dei capitoli relativi alla storia selezionata.
- **Sequenza degli eventi alternativa:** Nessuna.

7. Use case: Navigare tra capitoli mediante scorrimento

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende visualizzare i capitoli non mostrati a schermo
- **Pre-condizioni:** lo storico delle sessioni di produttività svolte dall'utente deve essere aperto; l'elenco dei capitoli di una storia deve essere aperto.
- **Post-condizioni:** Nessuna.

- **Sequenza degli eventi principale:**
 - il caso d'uso inizia quando un utente intende visualizzare i capitoli non mostrati a schermo.
 - in generale, per ragioni legate alla lunghezza dei device, il sistema mostra a schermo un sottoinsieme di capitoli.
 - l'utente interagisce con lo schermo effettuando scrolling verso il basso o verso l'alto al fine di cercare il capitolo desiderato.
 - il sistema rileva l'interazione.
 - il sistema permette la navigazione tra i capitoli mediante scorrimento.
- **Sequenza degli eventi alternativa:** se tutti i capitoli entrano nello schermo, la funzionalità di scrolling viene temporaneamente disabilitata.

8. Use case: Aprire la schermata con informazioni sul capitolo

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende visualizzare le informazioni di un capitolo
- **Pre-condizioni:** lo storico delle sessioni di produttività svolte dall'utente deve essere aperto; l'elenco dei capitoli di una storia deve essere aperto.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - il caso d'uso inizia quando un utente intende visualizzare le informazioni di un capitolo.
 - il sistema mostra a schermo i capitoli della storia selezionata.
 - l'utente seleziona il capitolo desiderato mediante tap su schermo.
 - il sistema rileva l'interazione.
 - il sistema mostra informazioni sul capitolo selezionato.
- **Sequenza degli eventi alternativa:** Nessuna.

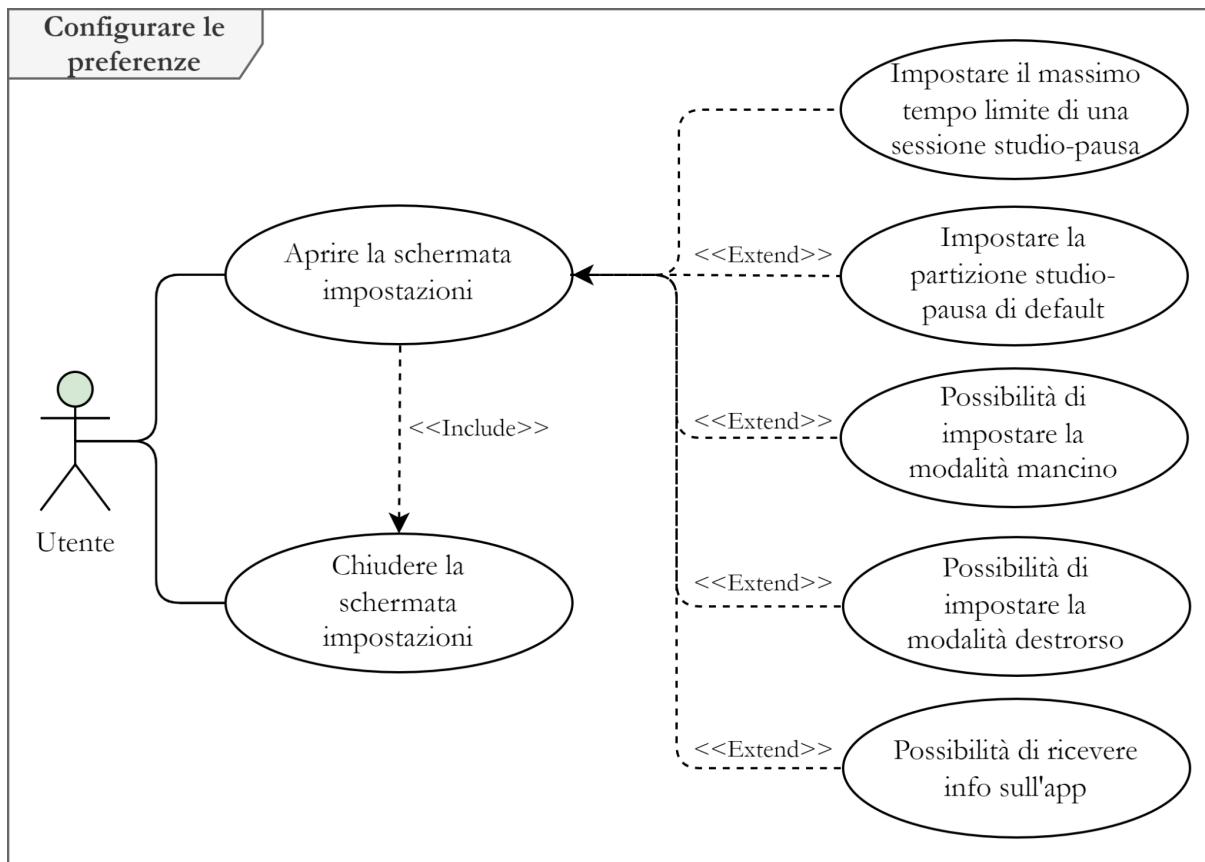
9. Use case: Chiudere la schermata con informazioni sul capitolo

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende nascondere le informazioni di un capitolo
- **Pre-condizioni:** lo storico delle sessioni di produttività svolte dall'utente deve essere aperto; l'elenco dei capitoli di una storia deve essere aperto; la schermata con informazioni sul capitolo deve essere aperta.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - il caso d'uso inizia quando un utente intende nascondere le informazioni di un capitolo.
 - il sistema mostra a schermo le informazioni sul capitolo selezionato ed il tasto indietro.

- c. l'utente seleziona il tasto indietro.
- d. il sistema rileva l'interazione.
- e. il sistema nasconde le informazioni sul capitolo selezionato.

- **Sequenza degli eventi alternativa:** Nessuna.

Diagramma use case: configurare le preferenze



Descrizione strutturata dei casi d'uso

1. Use case: Aprire la schermata impostazioni

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende visualizzare le impostazioni dell'app
- **Pre-condizioni:** Nessuna.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende visualizzare le impostazioni dell'app.
 - b. il sistema mostra il pulsante d'accesso alle impostazioni.
 - c. l'utente interagisce con il pulsante mediante input touch.
 - d. il sistema rileva la pressione del pulsante.
 - e. il sistema mostra a schermo le impostazioni.

- **Sequenza degli eventi alternativa:** Nessuna.

2. Use case: Chiudere la schermata impostazioni

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende chiudere le impostazioni dell'app
- **Pre-condizioni:** La schermata impostazioni deve essere aperta.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende chiudere le impostazioni dell'app.
 - b. il sistema mostra il bottom menù.
 - c. l'utente interagisce con il bottom menù mediante input touch.
 - d. il sistema rileva l'interazione.
 - e. il sistema esce dalle impostazioni.
- **Sequenza degli eventi alternativa:** Nessuna.

3. Use case: Impostare il massimo tempo limite di una sessione studio-pausa

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende impostare il massimo tempo limite di una sessione studio-pausa
- **Pre-condizioni:** La schermata impostazioni deve essere aperta.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende impostare il massimo tempo limite di una sessione studio-pausa.
 - b. il sistema mostra la view “editText” per permettere all'utente di immettere un nuovo valore.
 - c. l'utente interagisce con la view “editText” mediante input touch.
 - d. il sistema rileva l'interazione.
 - e. il sistema mostra a schermo la tastiera.
 - f. l'utente inserisce un valore da tastiera.
 - g. il sistema rileva l'interazione.
 - h. il sistema aggiorna il massimo tempo limite di una sessione studio-pausa.
- **Prima sequenza degli eventi alternativa:** supponiamo di essere al punto “e”:
 - f. l'utente inserisce un valore da tastiera, ma esso non è compreso tra 60 e 120.
 - g. il sistema rileva l'interazione.
 - h. il sistema non aggiorna il massimo tempo limite di una sessione studio-pausa.

- **Seconda sequenza degli eventi alternativa:** supponiamo di essere al punto “e”:
 - f. l’utente inserisce un valore da tastiera, ma esso non è un multiplo di 10.
 - g. il sistema rileva l’interazione.
 - h. il sistema arrotonda il valore immesso dall’utente e aggiorna il massimo tempo limite di una sessione studio-pausa.

4. Use case: Impostare la partizione studio-pausa di default

- **Descrizione:** Questo caso d’uso si verifica quando un utente intende impostare una nuova partizione studio-pausa di default
- **Pre-condizioni:** La schermata impostazioni deve essere aperta.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - il caso d’uso inizia quando un utente intende impostare una nuova partizione studio-pausa di default.
 - il sistema mostra la seekbar.
 - l’utente interagisce con la seekbar mediante sliding del dito.
 - il sistema rileva l’interazione.
 - il sistema aggiorna la partizione studio-pausa in base alla posizione corrente del dito sulla seekbar.
- **Sequenza degli eventi alternativa:** Nessuna.

5. Use case: Possibilità di impostare la modalità mancino

- **Descrizione:** Questo caso d’uso si verifica quando un utente intende attivare la modalità mancino
- **Pre-condizioni:** La schermata impostazioni deve essere aperta.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - il caso d’uso inizia quando un utente intende attivare la modalità mancino.
 - il sistema mostra lo switch di attivazione della modalità mancino.
 - l’utente interagisce con lo switch mediante input touch.
 - il sistema rileva l’interazione.
 - il sistema attiva la modalità mancino che sposta alcuni elementi della UI affinché siano facilmente raggiungibili per gli utenti mancini.
- **Sequenza degli eventi alternativa:** Use case: Possibilità di impostare la modalità destrorso

6. Use case: Possibilità di impostare la modalità destrorso

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende attivare la modalità destrorso
- **Pre-condizioni:** La schermata impostazioni deve essere aperta.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende attivare la modalità destrorso.
 - b. il sistema mostra lo switch di disattivazione della modalità mancino.
 - c. l'utente interagisce con lo switch mediante input touch.
 - d. il sistema rileva l'interazione.
 - e. il sistema disattiva la modalità mancino e sposta alcuni elementi della UI affinché siano facilmente raggiungibili per gli utenti destrorsi.
- **Sequenza degli eventi alternativa:** Use case: Possibilità di impostare la modalità mancino.

7. Use case: Possibilità di ricevere info sull'app

- **Descrizione:** Questo caso d'uso si verifica quando un utente intende visualizzare info sull'app
- **Pre-condizioni:** La schermata impostazioni deve essere aperta.
- **Post-condizioni:** Nessuna.
- **Sequenza degli eventi principale:**
 - a. il caso d'uso inizia quando un utente intende visualizzare info sull'app.
 - b. il sistema mostra il pulsante informativo.
 - c. l'utente interagisce con il pulsante informativo mediante input touch.
 - d. il sistema rileva l'interazione.
 - e. il sistema mostra le info sull'app.
- **Sequenza degli eventi alternativa:** Nessuna.

Progettazione database

schema concettuale, schema logico

La gestione delle informazioni nel software Work Is Progress è delegata ad un database relazionale locale al dispositivo di ciascun utente;

In particolare, la scelta di utilizzare un DB fondato sul modello relazionale dei dati è dovuto alle seguenti motivazioni:

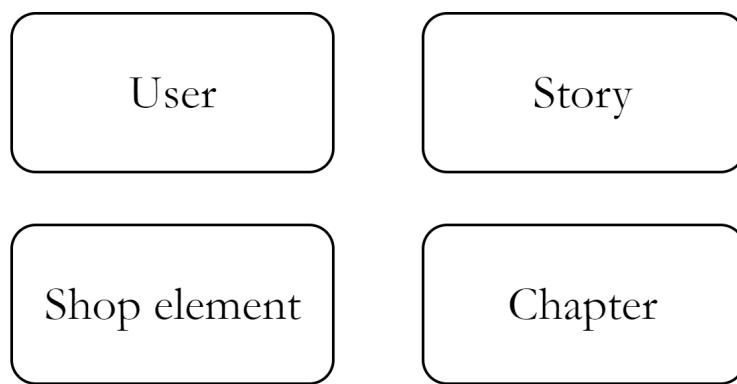
- le informazioni trattate nel software trovano naturale rappresentazione mediante tabelle, ovvero mediante relazioni matematiche;
- la rigidità dello schema facilita il mapping tra le entità nel DB e le classi nel codice sorgente;
- il modello relazionale offre un buon compromesso tra complessità della base di dati e complessità del codice sorgente dell'app;
per intendere al meglio tale affermazione, si osservi che se il DB dell'app fosse stato un file JSON, ovvero un file potenzialmente schema-less, allora la complessità del codice sarebbe stata notevolmente superiore a quella della base di dati poiché si sarebbe dovuto gestire, con logica imperativa, i vincoli di dominio ed i vincoli di integrità;
- garanzia di assenza di occorrenze duplicate grazie alle chiavi primarie;

Per la progettazione della base di dati dell'app WIP, si è optato per l'adozione della strategia Top-Down: lo schema concettuale è stato prodotto mediante raffinamenti successivi, cioè sono stati identificati gli oggetti essenziali e, mediante raffinamenti, sono stati aggiunti dettagli;

in sintesi, lo schema concettuale è stato modellato, analizzando i requisiti, dal generale al particolare.

Il flusso di lavoro che descrive la modellazione del DB è costituito dai seguenti passi:

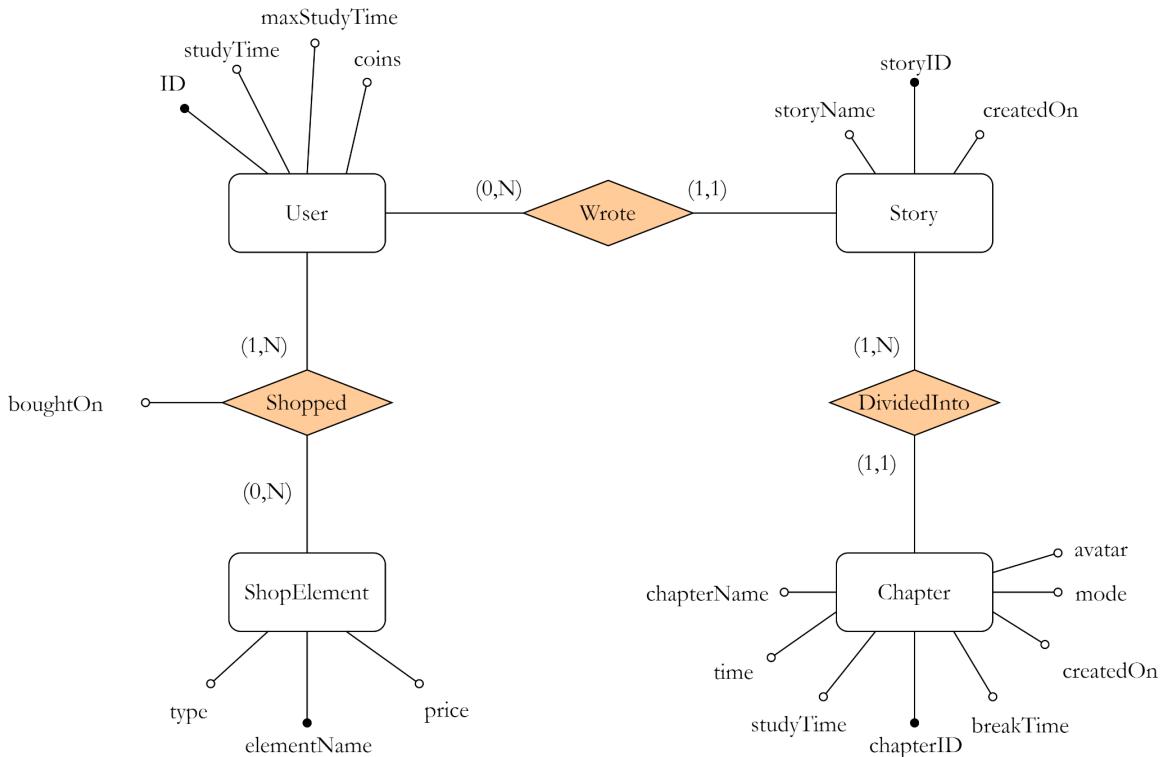
1. Identificazione delle entità essenziali dello schema concettuale:



le informazioni relative a tali entità sono apprezzabili nel dizionario dei dati.

2. Schema concettuale e schema logico:

Poiché la realtà da modellare non si basa su astrazioni particolarmente complesse, in fase di traduzione del modello concettuale in modello logico si è osservato che, in realtà, schema logico e schema concettuale coincidono;



segue l'analisi dello schema logico:

- **Relazione tra le entità User e Shop element:**

- **descrizione:** relationship che assegna all'utente le opere d'arte acquistate nello shop;
- **cardinalità:**
 - User - Shopped (1, N): l'istanza “utente” appartiene almeno una volta alla relationship poiché l'app prevede opere d'arte gratuite, ovvero acquistate di default;
 - ShopElement - Shopped (0, N): la partecipazione opzionale è necessaria poiché non è certo che una specifica opera d'arte venga acquistata dall'utente; la cardinalità massima è N, e non 1, perché anche se ogni utente può acquistare un'opera d'arte una sola volta, nelle versioni successive dell'app, si intende implementare una base di dati online che supporti più utenti e che:
 - salvi online i dati dell'utente in caso il dispositivo abbia accesso alla rete;

- salvi in locale i dati dell’utente in caso il dispositivo non abbia accesso alla rete; segue poi il dump quando la connessione viene ripristinata;
in questo modo, si supporta il device swapping.
- **Relazione tra le entità User e Story:**
 - **descrizione:** relationship che assegna all’utente le sessioni di produttività svolte;
 - **cardinalità:**
 - User - Wrote (0, N): in generale, un utente svolge molte sessioni di produttività; tuttavia, la partecipazione opzionale è necessaria poiché è possibile che l’utente non avvii mai una storia; un esempio è dato dal primo avvio dell’app;
 - Story - Wrote (1, 1): la partecipazione obbligatoria è necessaria poiché una storia può esistere se e solo se è stata creata dall’utente;
- **Relazione tra le entità Story e Chapter:**
 - **descrizione:** relationship che assegna ad ogni storia il numero di capitoli in cui è divisa;
 - **cardinalità:**
 - Story - DividedInto (1, N): ogni storia ha almeno un capitolo;
 - Chapter - DividedInto (1, 1): la partecipazione obbligatoria è necessaria poiché un capitolo può esistere se e solo se appartiene ad una storia;

Dizionario dei dati

dizionario entità e relationships

La descrizione delle entità, delle relationship e degli attributi di entità e relationship è apprezzabile nel seguente dizionario dei dati:

Entità			
Nome	Descrizione	Attributi	Identificatore
User	Struttura dati che memorizza: <ul style="list-style-type: none"> • le preferenze relative al time management dell'utente quando vengono configurati i parametri di una nuova sessione di produttività; • le monete in possesso dell'utente; 	Id Tipo: numerico-intero studyTime Tipo: numerico-intero studyTime: tempo di studio nella partizione studio-pausa maxStudyTime Tipo: numerico-intero maxStudyTime: tempo studio sommato a tempo pausa coins Tipo: numerico-intero	Id
Story	Struttura dati che memorizza le informazioni atte all'identificazione univoca di una storia	storyId Tipo: numerico-intero storyName Tipo: stringa createdOn Tipo: stringa	storyId
Shop element	Struttura dati che memorizza le informazioni atte alla profilazione di ciascuna opera d'arte;	elementName Tipo: stringa type Tipo: stringa type: avatar o quadro price Tipo: numerico-intero	elementName

Chapter	Struttura dati che memorizza tutte le informazioni relative ad una sessione di produttività	chapterId Tipo: numerico-intero	chapterId
		chapterName Tipo: stringa	
		time Tipo: stringa	
		studyTime Tipo: numerico-intero	
		breakTime Tipo: numerico-intero	
		avatar Tipo: stringa	
		mode Tipo: numerico-intero	
		createdOn Tipo: stringa	

Relationship			
Nome	Descrizione	Attributi	Entità coinvolte
Shopped	Associa un elemento dello shop ad un utente	boughtOn Tipo: stringa	User (1,N) ShopElement (0,N)
Wrote	Associa un utente ad una storia	/	User (1,N) Story (1,1)
DividedInto	Associa una storia ai suoi capitoli	/	Story (1,N) Chapter (1,1)

Vincoli non esprimibili

vincoli intra-relazionali

Di seguito, sono riportati i vincoli che non è stato possibile esprimere mediante i costrutti del modello concettuale, ovvero mediante:

- entità
- relationship
- attributo
- cardinalità
- identificatore
- generalizzazione

ma che, comunque, sono di notevole importanza e significatività nel modello progettato:

1. l'attributo “studyTime”, relativo all'entità “User”, può assumere solo ed esclusivamente valori interi multipli di 10;
2. l'attributo “studyTime”, relativo all'entità “User”, può assumere solo ed esclusivamente valori interi al più uguali a $\maxStudyTime - 10$
3. l'attributo “maxStudyTime”, relativo all'entità “User”, può assumere solo ed esclusivamente valori interi multipli di 10;
4. l'attributo “maxStudyTime”, relativo all'entità “User”, può assumere solo ed esclusivamente valori interi compresi tra 60 e 120;
5. l'attributo “type”, relativo all'entità “ShopElement”, può assumere solo ed esclusivamente valori “avatar” e “background”
6. l'attributo “avatar”, relativo all'entità “Chapter”, può assumere solo ed esclusivamente gli identificatori assegnati alle avatar-image-resources;

Implementazione del database

traduzione nel modello relazionale

Note:

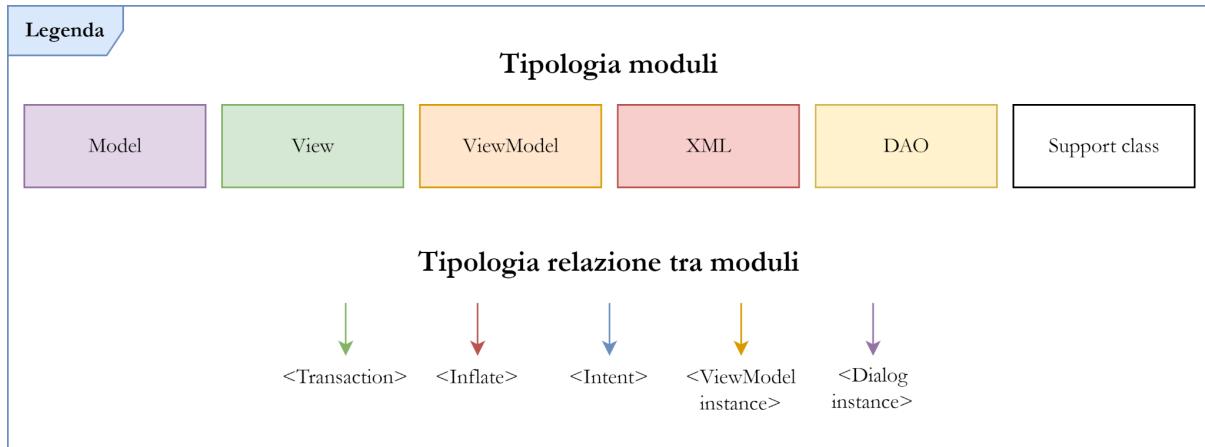
- le primary key sono sottolineate;
- le foreign key sono evidenziate mediante “*”

Entità/relazione	Traduzione
User	User(<u>ID</u> , studyTime, maxStudyTime, coins)
Story	Story(<u>storyID</u> , storyName, createdOn, user*)
Chapter	Chapter(<u>chapterID</u> , chapterName, time, createdOn, studyTime, breakTime, mode, avatar, story*)
ShopElement	ShopElement(<u>elementName</u> , type, description, price)
Shopped	Shopped(user*, <u>ShopElement</u> *, boughtOn)

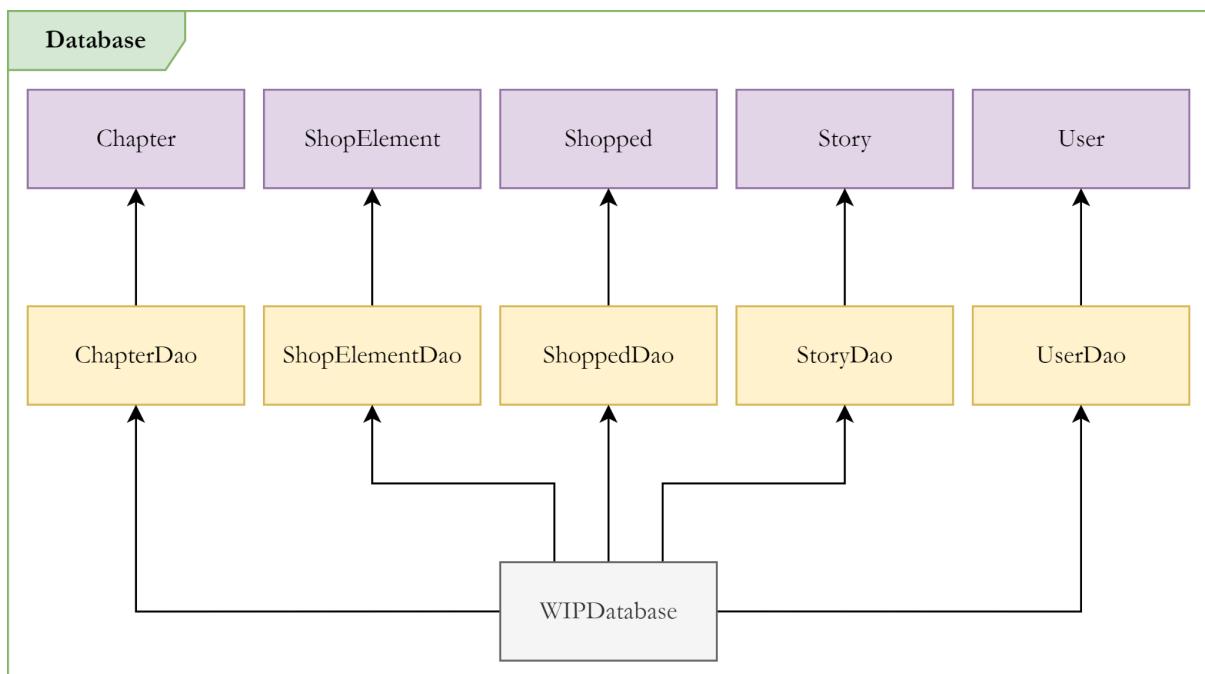
Mappa dell'architettura

architettura database, architettura totale e architettura divisa in aree funzionali

Al fine di aumentare la leggibilità della mappa dell'architettura, il team di progettazione ha ritenuto opportuno fornire, al lettore, la seguente legenda:

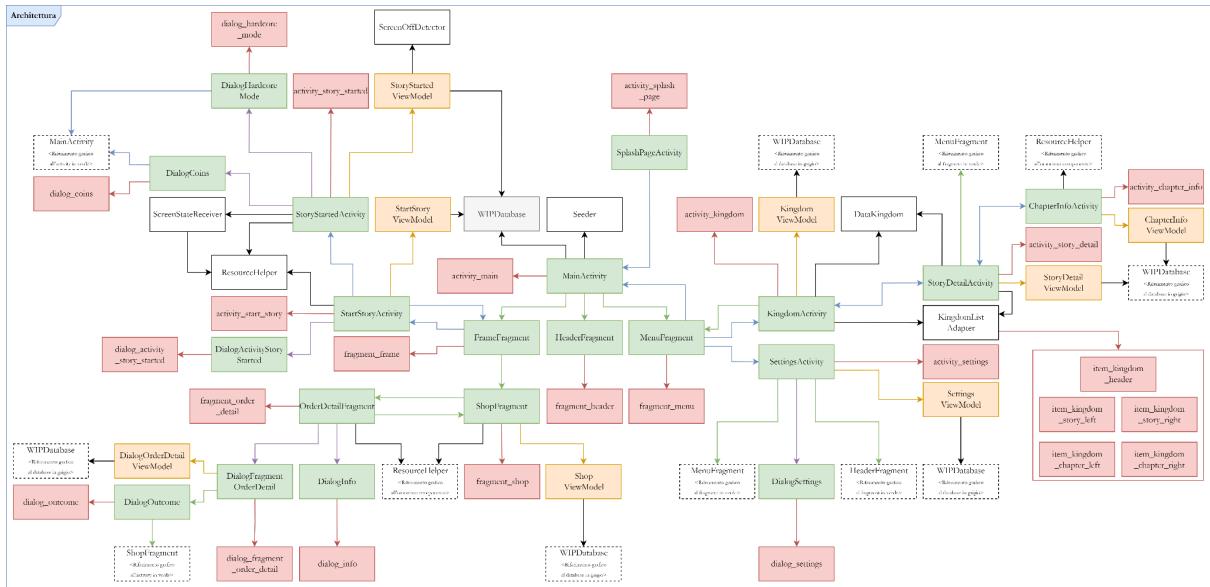


In primo luogo, l'architettura dell'app WIP prevede un modulo dedito alla manipolazione ed alla memorizzazione persistente dei dati, ovvero prevede un database costituito dai seguenti componenti:



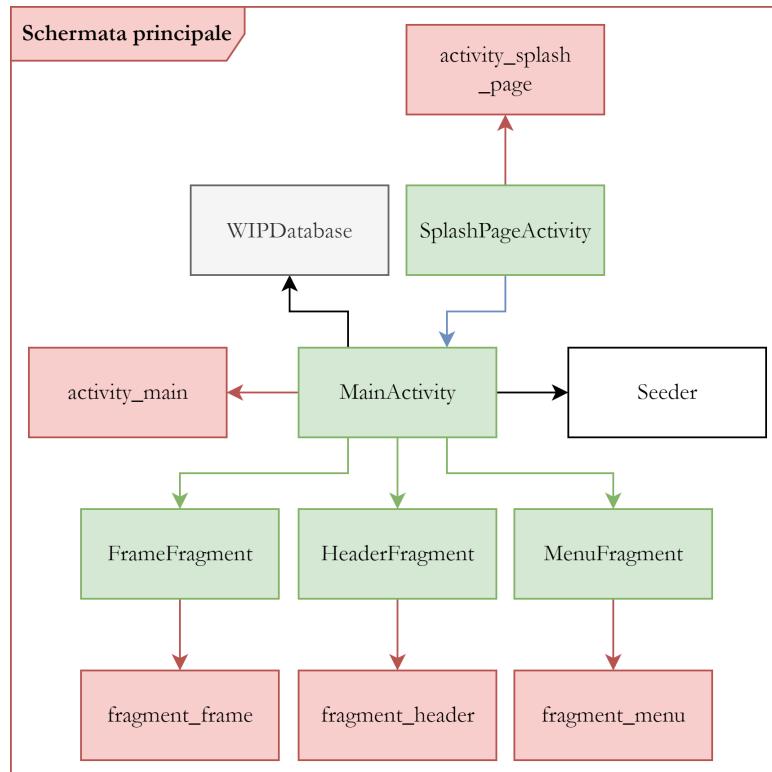
L'interazione tra la base di dati e l'app WIP è gestita mediante libreria ROOM, che fornisce un livello di astrazione su SQLite e che permette il mapping tra entità ed oggetti e tra query e metodi.

Di seguito, presentiamo l'architettura totale dell'applicazione WIP:



In conclusione, scindiamo l'architettura totale del software in aree funzionali; ciascuna area funzionale è basata su uno dei casi d'uso precedentemente illustrati:

Use case banale: apertura dell'app

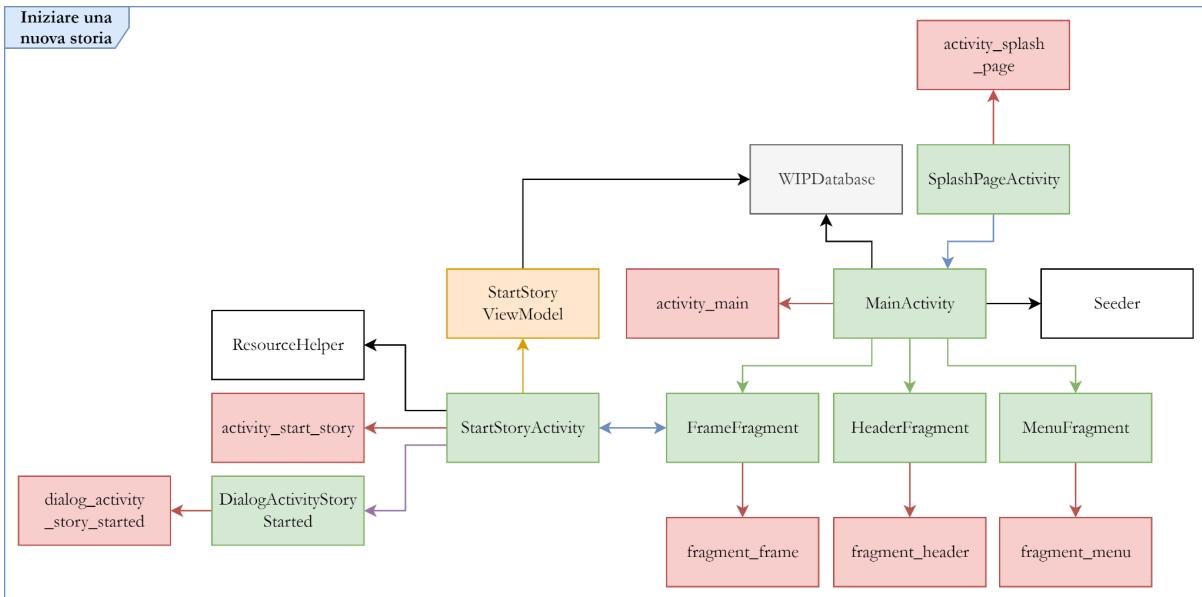


la seguente porzione architetturale racchiude tutti e soli i moduli dediti a:

- generare e gestire graficamente la splash page dell'app;
- generare e gestire graficamente la schermata principale dell'app;
- popolare la base di dati;

- avviare il gestore del database;
- generare e gestire il bottom menù;

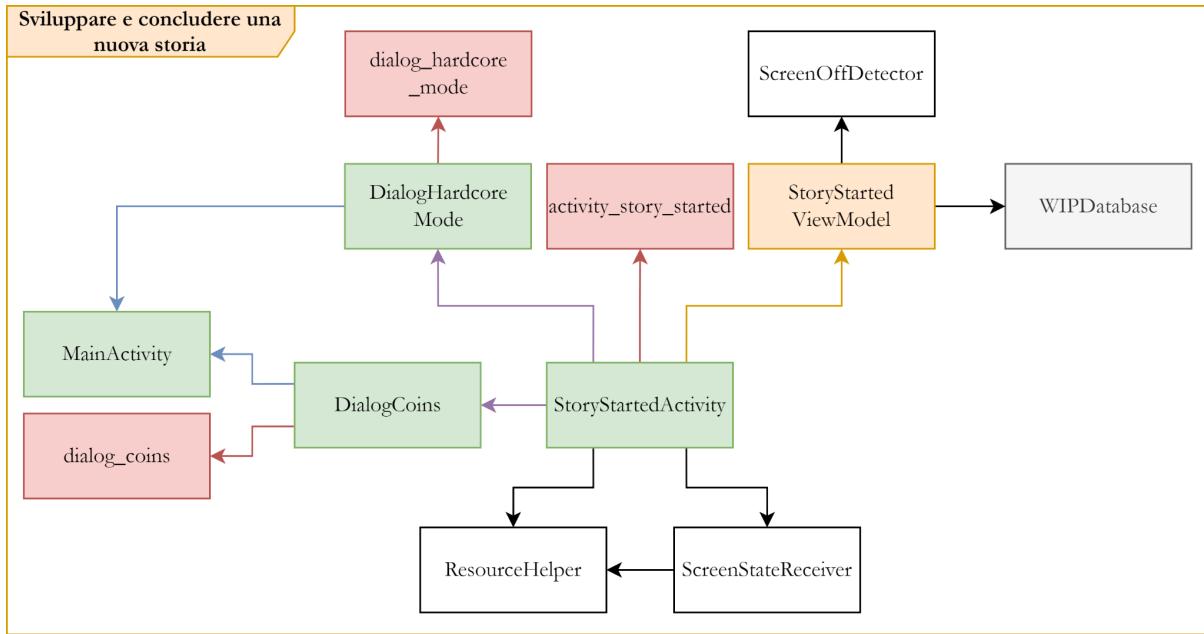
Use case: iniziare una nuova storia



la seguente porzione architetturale racchiude tutti e soli i moduli dediti a:

- avviare l'app;
- generare e gestire graficamente la schermata di impostazione dei parametri della storia;
- gestire la logica e la persistenza dei dati della schermata di impostazione dei parametri della storia;
- generare e gestire graficamente un dialog con contenuto informativo circa le modalità di utilizzo dei componenti dell'app;

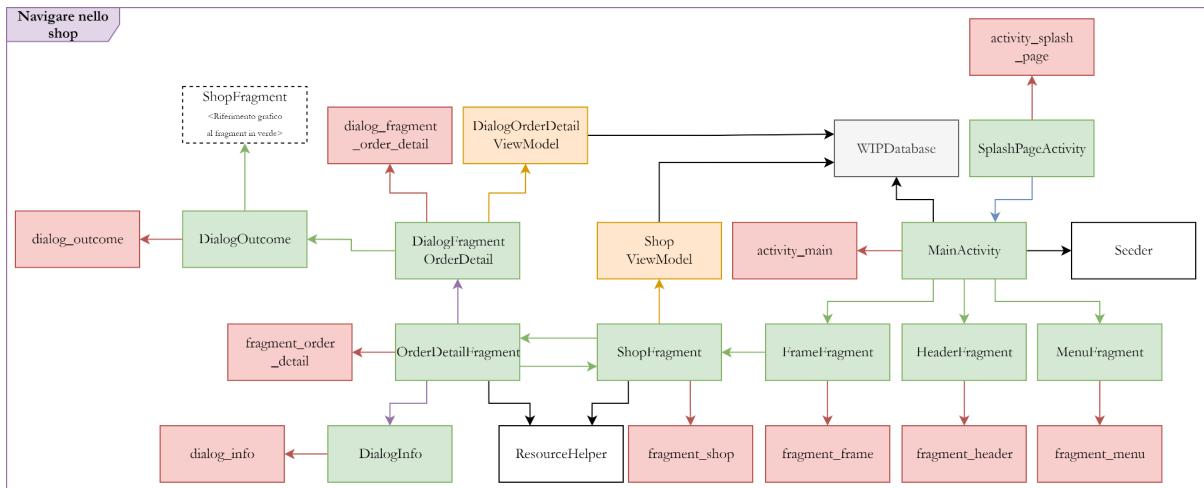
Use case: sviluppare e concludere una nuova storia



la seguente porzione architetturale racchiude tutti e soli i moduli dediti a:

- generare e gestire graficamente la schermata che mostra lo svolgimento della storia;
- rilevare il passaggio in background dell'app, il lock e l'unlock dello schermo;
- gestire la logica e la persistenza dei dati della schermata che mostra lo svolgimento della storia;
- aggiornare la base di dati in base alle monete guadagnate;
- generare e gestire graficamente dialog di conclusione storia;
- generare e gestire graficamente la schermata principale mostrata quando la storia viene terminata;

Use case: navigare nello shop

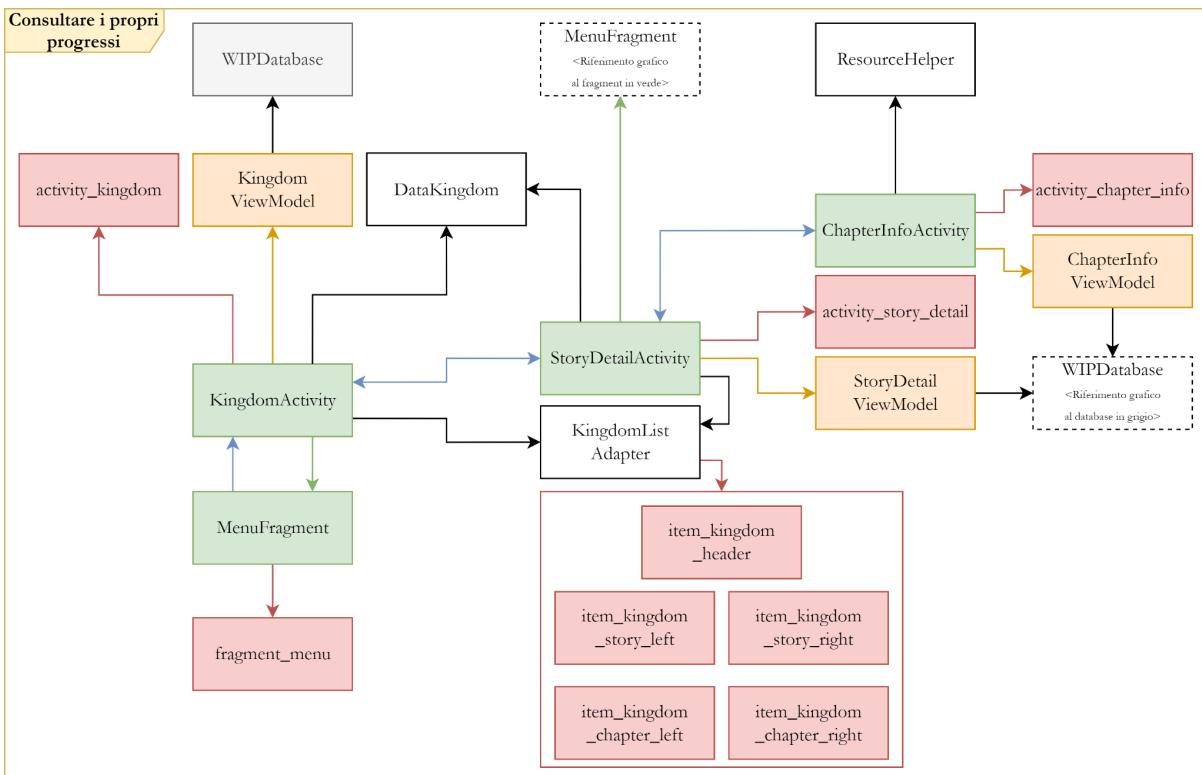


la seguente porzione architetturale racchiude tutti e soli i moduli dediti a:

- generare e gestire graficamente la schermata shop;

- gestire la logica e la persistenza dei dati della schermata shop;
- generare e gestire graficamente la schermata di dettaglio ordine;
- gestire il passaggio tra la schermata shop e la schermata di dettaglio ordine e viceversa;
- generare e gestire graficamente dialog informativi circa le opere d'arte;
- generare e gestire graficamente il dialog di checkout ordine;
- gestire la logica e la persistenza dei dati del dialog di checkout ordine;
- generare e gestire graficamente i dialog che descrivono l'esito dell'acquisto;
- generare e gestire graficamente la schermata shop quando si termina l'acquisto;

Use case: consultare i propri progressi

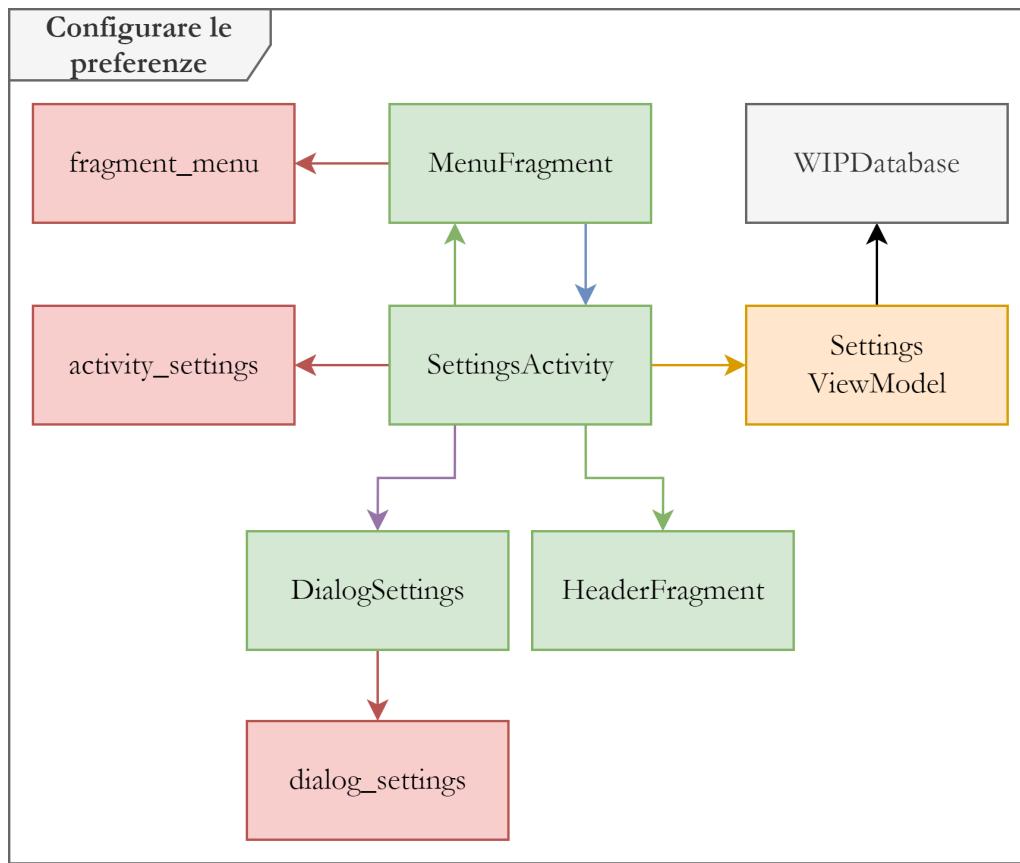


la seguente porzione architetturale racchiude tutti e soli i moduli dediti a:

- generare e gestire graficamente la schermata contenente la lista delle storie dell'utente;
- gestire la logica e la persistenza dei dati della schermata contenente la lista delle storie dell'utente;
- generare e gestire la logica della scrollView con item eterogenei;
- generare e gestire graficamente la schermata contenente la lista dei capitoli di una storia;
- gestire la logica e la persistenza dei dati della schermata contenente la lista dei capitoli di una storia;
- generare e gestire la logica della scrollView contenente i capitoli;
- generare e gestire graficamente la schermata contenente i dettagli di un capitolo;

- gestire la logica e la persistenza dei dati della schermata contenente i dettagli di un capitolo;

Use case: configurare le preferenze



la seguente porzione architetturale racchiude tutti e soli i moduli dediti a:

- generare e gestire graficamente la schermata impostazioni;
- gestire la logica e la persistenza dei dati della schermata impostazioni;
- generare e gestire graficamente il dialog contenente info sull'app;

Analisi dell'architettura

descrizione dei moduli dell'architettura dell'app

Nella sezione che segue, presentiamo ciascuno dei moduli costitutivi dell'architettura dell'app WIP;

Dalla trattazione sono, tuttavia, esclusi i file .XML di layout poiché responsabili esclusivamente dell'aspetto grafico del software.

Avviamo l'analisi dei componenti esaminando gli UI controllers:

UI controllers

Activities

Fragments

Dialogs

UI controllers: Activities

Activities

SplashPageActivity

MainActivity

StartStoryActivity

StoryStartedActivity

KingdomActivity

ChapterInfoActivity

StoryDetailActivity

SettingsActivity

1. SplashPageActivity:

- **tipologia classe:** activity;
- **classe estesa:** AppCompatActivity; viene eseguito l'override del metodo onCreate(...);
- **funzionalità:** la classe SplashPageActivity racchiude le seguenti funzionalità:
 - esegue linflate del file .xml di layout “activity_splash_page”;
 - mostra la splash page dell'app e, dopo un secondo (1000 millisecondi), avvia l'activity MainActivity mediante intent esplicito; l'intent esplicito non prevede passaggio di informazioni tra le activity coinvolte;

- mediante il metodo finish, rimuove la sua istanza dallo stack;

2. MainActivity:

- **tipologia classe:** activity;
- **classe estesa:** AppCompatActivity; viene eseguito l'override del metodo onCreate(...);
- **funzionalità:** la classe MainActivity racchiude le seguenti funzionalità:
 - consulta il file sharedPreference memorizzato localmente sul dispositivo e, a seconda del valore letto, imposta l'UI dell'app in modalità mancino o in modalità destroverso;
 - invoca il costruttore della superclasse estesa;
 - esegue linflate del file .xml di layout “activity_main”;
 - mappa layout e view espresse nel file .xml “activity_main” in oggetti mediante dataBinding;
 - esegue una transaction per mostrare a schermo i fragment:
 - “header_layout”: file .xml che descrive lheader dell'app
 - “menu_layout”: file .xml che descrive il bottom menù;
 - “frame_layout”: file .xml che descrive il contenuto della schermata principale dell'app, ovvero il pulsante per avviare una nuova storia e il pulsante per accedere allo shop;
 - genera un'istanza della base di dati locale al dispositivo che si auto-inizializza;
 - gestisce l'eccezione relativa alla mancata lettura dell'id dell'utente durante linizializzazione del database;

3. StartStoryActivity:

- **tipologia classe:** activity;
- **classe estesa:** AppCompatActivity; viene eseguito l'override del metodo onCreate(...);
- **funzionalità:** la classe StartStoryActivity racchiude le seguenti funzionalità:
 - genera un'istanza del viewModel “StartStoryViewModel” a cui delega la logica dellactivity e la persistenza dei dati mediante componenti lifeCycle-aware;
 - consulta il file sharedPreference memorizzato localmente sul dispositivo e, a seconda del valore letto, imposta l'UI dell'app in modalità mancino o in modalità destroverso;
 - invoca il costruttore della superclasse estesa;
 - esegue linflate del file .xml di layout “activity_start_story”;
 - mappa layout e view espresse nel file .xml “activity_start_story” in oggetti mediante dataBinding;

- lega l'istanza del viewModel al file .xml “activity_start_story”;
- consente all'utente di scrivere il titolo della storia assegnando focus all'apposita editText mediante listener;
- suggerisce all'utente il completamento del titolo della storia in caso essa sia già stata memorizzata precedentemente nel sistema;
- memorizza, in un oggetto observable nel viewModel, il titolo della storia ogni volta che viene cambiato;
- consente all'utente di impostare la partizione studio-pausa assegnando focus all'apposita seekbar mediante listener;
- memorizza, in un oggetto observable nel viewModel, la partizione studio-pausa scelta dall'utente;
- formatta il testo mostrato all'utente quando viene eseguito lo slide della seekbar;
- imposta dinamicamente il font degli switch hardcore mode e silent mode;
infatti, per queste view, il font non viene riconosciuto se definito come attributo nel file .xml di layout;
- gestisce le azioni da eseguire quando lo switch silent mode viene selezionato mediante input touch;
in questa sezione, non approfondiamo l'algoritmo relativo alla gestione della modalità silenziosa poiché esso appartiene al viewModel;
- gestisce le azioni da eseguire quando lo switch hardcore mode viene selezionato mediante input touch;
in questa sezione, non approfondiamo l'algoritmo relativo alla gestione della modalità hardcore poiché esso appartiene al viewModel;
- cambia l'aspetto del bottone “info” mediante TouchListener ogni qualvolta l'utente vi interagisce;
- istanzia un oggetto della classe DialogActivityStoryStarted, che mostra il dialog contenente le informazioni relative alle modalità silent ed hardcore, quando viene rilevato il click del bottone “info” mediante OnClickListener;
- cambia l'aspetto dei bottoni direzionali per la selezione dell'avatar mediante TouchListener ogni qualvolta l'utente vi interagisce;
- mostra a schermo l'avatar di default consultando la lista degli avatar acquistati, ottenuta mediante interazione tra viewModel e model, mappando la risorsa ottenuta dal viewModel in idResource, utilizzando il metodo fromShopElementNameToResource appartenente alla classe ResourceHelper, e passando al metodo

- setBackgroundResource l'idResource appena ottenuto;
 la classe ResourceHelper è stata precedentemente importata;
- gestisce lo switch dell'avatar correntemente mostrato a schermo mediante ClickListener;
 - cambia l'aspetto del bottone “indietro”, utile per tornare alla schermata principale, mediante TouchListener ogni qualvolta l'utente vi interagisce;
 - avvia l'activity MainActivity, mediante intent esplicito, quando viene rilevato il click del bottone “indietro” attraverso un clickListener; l'intent esplicito non prevede passaggio di informazioni tra le activity coinvolte;
 - cambia l'aspetto del bottone “start”, utile per avviare la storia appena configurata, mediante TouchListener ogni qualvolta l'utente vi interagisce;
 - controlla se il titolo della storia è stato inserito quando viene rilevato il click del bottone “start” attraverso un clickListener; in caso di esito negativo, il sistema impedisce l'avvio della storia e mostra all'utente un Toast che lo invita ad inserire il titolo; il caso di esito positivo, si ha l'avvio dell'activity StoryStartedActivity mediante intent esplicito; l'intent esplicito prevede il passaggio delle seguenti informazioni tra le activity coinvolte:
 - nome della storia;
 - tempo di studio;
 - tempo di pausa;
 - avatar selezionato;
 - id dell'avatar selezionato;
 - modalità selezionata (hardcore mode, silent mode o nessuna delle due);

4. StoryStartedActivity:

- **tipologia classe:** activity;
- **classe estesa:** AppCompatActivity; viene eseguito l'override di metodi onCreate(...), onRestart() ed onResume();
- **funzionalità in onCreate:** la classe StoryStartedActivity racchiude le seguenti funzionalità:
 - genera un'istanza del viewModel “StoryStartedViewModel” a cui delega la logica dell'activity;

- consulta il file sharedPreference memorizzato localmente sul dispositivo e, a seconda del valore letto, imposta l'UI dell'app in modalità mancino o in modalità destroso;
- invoca il costruttore della superclasse estesa;
- esegue linflate del file .xml di layout “activity_story_started”;
- mappa layout e view espresse nel file .xml “activity_story_started” in oggetti mediante dataBinding;
- lega l'istanza del viewModel al file .xml “activity_story_started”;
- recupera le informazioni passate come extras mediante “intent.extras”;
- definisce le variabili per il time management (utili alla gestione dell'evoluzione del quadro);
- seleziona randomicamente e mostra la prima frase di incoraggiamento; le frasi di incoraggiamento sono ottenute mediante il metodo encouragementQuotes() della classe ResourceHelper;
la classe ResourceHelper è stata precedentemente importata;
- istanzia, configura e registra un BroadcastReceiver utile a rilevare lunlock dello schermo; il funzionamento del BroadcastReceiver sarà espresso nell'apposita sezione;
- registra un BroadcastReceiver utile a rilevare il lock dello schermo ed il task correntemente in esecuzione; il funzionamento del BroadcastReceiver sarà espresso nell'apposita sezione;
- mostra a schermo il primo stato di un quadro; il quadro è selezionato randomicamente tra quelli disponibili mediante il metodo “backgroundSelector”;
il metodo “backgroundSelector” è dichiarato nell'activity, e non nel viewModel, poiché gestisce solo ed esclusivamente lo swap di imageResources e non la logica dell'activity;
infatti, le activity sono, per definizione, i controller volti alla presentazione dei dati e, coerentemente a quanto appena espresso, il metodo “backgroundSelector” svolge esattamente tale funzionalità;
- mostra a schermo lavatar precedentemente selezionato;
- avvia il cronometro;
- gestisce il tempo calcolato dal cronometro mediante il listener “setOnChronometerTickListener”;
- evolve il quadro corrente mediante algoritmo sito nel blocco di codice del listener “setOnChronometerTickListener”;
il funzionamento dell'algoritmo può essere inteso mediante commenti sul codice e consultando i task dell' User story 4:

Sviluppo storia;

l'evoluzione dello stato dei quadri è gestita mediante il metodo “backGroundEvolution”;

analogamente a quanto espresso per il metodo “backgroundSelector”, il metodo “backGroundEvolution” è dichiarato nell’activity, e non nel viewModel, poiché gestisce solo ed esclusivamente lo swap di imageResources e non la logica dell’activity;

- cambia l’aspetto del bottone “stop”, utile per concludere la storia, mediante TouchListener ogni qualvolta l’utente vi interagisce;
- esegue questi passi quando viene rilevato il click del bottone “stop” attraverso un clickListener;
 - il cronometro si ferma;
 - i BroadcastReceiver vengono rimossi;
 - vengono calcolate le monete guadagnate dall’utente durante la sessione di produttività mediante il metodo “coinCalculator”; il metodo "coinCalculator" è dichiarato nel viewModel e, di conseguenza, non verrà analizzato nella sezione corrente;
 - istanzia un oggetto della classe DialogCoins, che mostra il dialog contenente le monete guadagnate durante la storia;
 - memorizza la storia appena conclusa mediante una coroutine se e solo se la sua durata è maggiore o uguale a 10 secondi; il metodo per la memorizzazione di una nuova storia è dichiarato nel viewModel e, di conseguenza, non è presentato nella sezione corrente;

- **funzionalità in onRestart:**

- recupera le informazioni relative alla modalità opzionale selezionata dall’utente; tali informazioni sono passate come extras mediante “intent.extras”;
- rileva se la modalità hardcore è stata selezionata dall’utente e, in caso di esito positivo, analizza lo stato dei flag per determinare se l’azione compiuta dall’user comporta la terminazione automatica della storia o meno; in particolare, i controlli circa la legittimità delle azioni compiute dall’user in modalità hardcore vengono svolte nei metodi onRestart ed onResume; il motivo è che qualsiasi azione porti l’app in background, o blocchi e sblocchi lo schermo del cellulare, cambia lo stato dell’activity

nell'activity lifecycle e, di conseguenza, servono metodi appropriati alla gestione dello stato corrente dell'activity;

- **funzionalità in onResume:**

- recupera le informazioni relative alla modalità opzionale selezionata dall'utente;
tali informazioni sono passate come extras mediante “intent.extras”;
- mappa un oggetto di tipo Chronometer alla view Chronometer, presente nel file .xml “activity_story_started”, mediante metodo findViewById;
- rileva se la modalità hardcore è stata selezionata dall'utente e, in caso di esito positivo, analizza lo stato del flag di terminazione storia; a seconda del valore assunto dal flag di terminazione storia, si hanno due possibili scenari:
 - la storia continua poiché non sono state eseguite azioni illegittime;
 - la storia termina poiché sono state eseguite azioni illegittime; in questo caso:
 - il cronometro si ferma;
 - i BroadcastReceiver vengono rimossi;
 - si istanzia un oggetto della classe DialogHardcoreMode, che mostra il dialog contenente informazioni circa la terminazione della storia a causa delle azioni non consentite;
 - non si guadagnano monete;
 - la storia/capitolo non viene salvata nella schermata “Kingdom”;

5. KingdomActivity:

- **tipologia classe:** activity;
- **classe estesa:** AppCompatActivity; viene eseguito l'override del metodo onCreate(...);
- **funzionalità:** la classe KingdomActivity racchiude le seguenti funzionalità:
 - genera un'istanza del viewModel “KingdomViewModel” a cui delega la logica dell'activity e la comunicazione con il database;
 - consulta il file sharedPreference memorizzato localmente sul dispositivo e, a seconda del valore letto, imposta l'UI dell'app in modalità mancino o in modalità destroso;
 - invoca il costruttore della superclasse estesa;
 - esegue linflate del file .xml di layout “activity_kingdom”;

- mappa layout e view espresse nel file .xml “activity_kingdom” in oggetti mediante dataBinding;
- lega l’istanza del viewModel al file .xml “activity_kingdom”;
- importa la dataClass DataKingdom;
- genera la lista delle storie presenti nel Kingdom;
- dispone a destra o a sinistra le storie nella recyclerView;
- genera la lista delle storie da inserire nella navbar, ovvero nella scrollView orizzontale;
- avvia l’activity storyDetailActivity mediante intent esplicito quando, mediante listener, viene rilevato il click dell’utente su una storia; l’intent esplicito prevede passaggio di informazioni tra le activity coinvolte; in particolare, le informazioni condivise sono:
 - id della storia;
 - nome della storia;
- reindirizza l’utente alla storia desiderata quando, mediante listener, viene rilevato il click di un elemento nella navbar;
- esegue una transaction per mostrare a schermo il fragment MenuFragment che, a sua volta, esegue l’inflate del file: “menu_layout”;

6. StoryDetailActivity:

- **tipologia classe:** activity;
- **classe estesa:** AppCompatActivity; viene eseguito l’override del metodo onCreate(...);
- **funzionalità:** la classe StoryDetailActivity racchiude le seguenti funzionalità:
 - genera un’istanza del viewModel “StoryDetailViewModel” a cui delega la logica dell’activity e la comunicazione con il database;
 - consulta il file sharedPreference memorizzato localmente sul dispositivo e, a seconda del valore letto, imposta l’UI dell’app in modalità mancino o in modalità destroverso;
 - invoca il costruttore della superclasse estesa;
 - esegue l’inflate del file .xml di layout “activity_story_detail”;
 - mappa layout e view espresse nel file .xml “activity_story_detail” in oggetti mediante dataBinding;
 - lega l’istanza del viewModel al file .xml “activity_story_detail”;
 - recupera le informazioni passate come extras mediante “intent.extras”;
 - importa la dataClass DataKingdom;
 - genera la lista dei capitoli appartenenti alla storia selezionata;

- dispone a destra o a sinistra i capitoli nella recyclerView;
- avvia l'activity ChapterInfoActivity mediante intent esplicito quando, mediante listener, viene rilevato il click dell'utente su un capitolo; l'intent esplicito prevede passaggio di informazioni tra le activity coinvolte; in particolare, le informazioni condivise sono:
 - id della storia;
 - nome della storia;
 - id del capitolo
- esegue una transaction per mostrare a schermo il fragment MenuFragment che, a sua volta, esegue linflate del file: “menu_layout”;
- cambia l'aspetto del bottone “indietro”, utile per tornare alla KingdomActivity, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- avvia l'activity KingdomActivity mediante intent esplicito quando, mediante listener, viene rilevato il click dell'utente sul bottone “indietro”; l'intent esplicito prevede passaggio di informazioni tra le activity coinvolte; in particolare, l'informazione condivisa è l'id della storia;

7. ChapterInfoActivity:

- **tipologia classe:** activity;
- **classe estesa:** AppCompatActivity; viene eseguito l'override del metodo onCreate(...);
- **funzionalità:** la classe ChapterInfoActivity racchiude le seguenti funzionalità:
 - genera un'istanza del viewModel “ChapterInfoViewModel” a cui delega la comunicazione con il database;
 - consulta il file sharedPreference memorizzato localmente sul dispositivo e, a seconda del valore letto, imposta l'UI dell'app in modalità mancino o in modalità destroso;
 - invoca il costruttore della superclasse estesa;
 - esegue linflate del file .xml di layout “activity_chapter_info”;
 - mappa layout e view espresse nel file .xml “activity_chapter_info” in oggetti mediante dataBinding;
 - lega l'istanza del viewModel al file .xml “activity_chapter_info”;
 - recupera le informazioni passate come extras mediante “intent.extras”;
 - recupera le informazioni relative al capitolo selezionato dall'utente;
 - mostra data e ora in cui è stato memorizzato il capitolo;

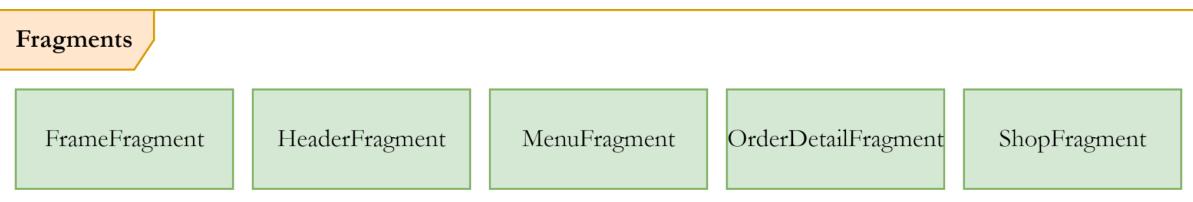
- mostra il tempo totale della sessione di produttività;
- mostra la partizione studio-pausa selezionata dall'utente;
- mostra le modalità opzionali selezionate dall'utente;
- mostra l'avatar selezionato dall'utente;
-
- cambia l'aspetto del bottone “indietro”, utile per tornare alla StoryDetailActivity, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- avvia l'activity StoryDetailActivity mediante intent esplicito quando, mediante listener, viene rilevato il click dell'utente sul bottone “indietro”; l'intent esplicito prevede passaggio di informazioni tra le activity coinvolte; in particolare, le informazione condivisa sono:
 - il nome della storia;
 - l'id della storia;

8. SettingsActivity:

- **tipologia classe:** activity;
- **classe estesa:** AppCompatActivity; viene eseguito l'override del metodo onCreate(...);
- **funzionalità:** la classe SettingsActivity racchiude le seguenti funzionalità:
 - genera un'istanza del viewModel “SettingsViewModel” a cui delega la logica dell'activity, la persistenza dei dati mediante oggetti observable e la comunicazione con il database;
 - consulta il file sharedPreference memorizzato localmente sul dispositivo e, a seconda del valore letto, imposta l'UI dell'app in modalità mancino o in modalità destro;
 - invoca il costruttore della superclasse estesa;
 - esegue linflate del file .xml di layout “activity_settings”;
 - mappa layout e view espresse nel file .xml “activity_settings” in oggetti mediante dataBinding;
 - lega l'istanza del viewModel al file .xml “activity_settings”;
 - consente all'utente di impostare la partizione studio-pausa assegnando focus all'apposita seekbar mediante listener;
 - memorizza, in un oggetto observable nel viewModel, la partizione studio-pausa scelta dall'utente;
 - formatta il testo mostrato all'utente quando viene eseguito lo slide della seekbar;
 - consente all'utente di impostare il massimo tempo limite della storia assegnando focus all'apposita editText mediante listener;
 - aggiorna il testo della editText a seconda dell'input dell'utente;

- se l'utente cancella ogni carattere nell'editText, viene mostrato un testo di default;
- se l'utente inserisce un valore numerico, tale valore numerico viene seguito dalla stringa “min”;
- impedisce all'utente di spostare il cursore nella editText;
- gestisce il tempo minimo impostabile come maxTime;
- gestisce il tempo massimo impostabile come maxTime;
- arrotonda i valori numerici inseriti in input dall'utente a multipli di 10;
- imposta dinamicamente il font dello switch left hand mode; infatti, per questa view, il font non viene riconosciuto se definito come attributo nel file .xml di layout;
- gestisce l'attivazione/disattivazione della modalità mancino mediante listener;
 - ogni qualvolta attiviamo/disattiviamo la modalità mancino:
 - la configurazione della UI viene memorizzata nelle sharedPreferences;
 - l'activity corrente è ricaricata lanciando un intent esplicito;
- cambia l'aspetto del bottone “info”, utile per mostrare dettagli sull'app, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- istanzia un oggetto della classe DialogSettings, che mostra il dialog contenente le informazioni sull'app, quando viene rilevato, mediante listener, il click del bottone “info”;
- esegue una transaction per mostrare a schermo i fragment MenuFragment e HeaderFragment che, a loro volta, eseguono linflate dei file “menu_layout” ed “header_layout”;

UI controllers: Fragments



1. FrameFragment:

- **tipologia classe:** fragment;
- **classe estesa:** Fragment; viene eseguito l'override del metodo onCreateView(...);
- **funzionalità:** la classe FrameFragment racchiude le seguenti funzionalità:

- esegue l'inflate del file .xml di layout “fragment_frame”;
- invoca il costruttore della superclasse;
- mappa le view espresse nel file .xml “fragment_frame” in oggetti mediante il metodo findViewById;
- cambia l'aspetto del bottone “play”, utile per iniziare la configurazione di una nuova storia, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- avvia l'activity StartStoryActivity, mediante intent esplicito, quando viene rilevato il click del bottone “play” attraverso un clickListener; l'intent esplicito non prevede passaggio di informazioni tra le activity coinvolte;
- cambia l'aspetto del bottone “shop”, utile per iniziare la configurazione di una nuova storia, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- avvia l'activity StartStoryActivity, mediante intent esplicito, quando viene rilevato il click del bottone “play” attraverso un clickListener; l'intent esplicito non prevede passaggio di informazioni tra le activity coinvolte;
- esegue una transaction per mostrare a schermo lo ShopFragment, che esegue l'inflate del file “fragment_shop”, quando viene rilevato il click del bottone “shop” attraverso un clickListener;

2. HeaderFragment:

- **tipologia classe:** fragment;
- **classe estesa:** Fragment;
- **funzionalità:** la classe HeaderFragment esegue l'inflate del file .xml di layout “fragment_header”;

3. MenuFragment:

- **tipologia classe:** fragment;
- **classe estesa:** Fragment; viene eseguito l'override del metodo onCreateView(...);
- **funzionalità:** la classe MenuFragment racchiude le seguenti funzionalità:
 - esegue l'inflate del file .xml di layout “fragment_menu”;
 - mappa layout e view espresse nel file .xml “fragment_menu” in oggetti mediante dataBinding;
 - cambia l'aspetto del bottone “home”, utile per tornare alla schermata principale, mediante TouchListener ogni qualvolta l'utente vi interagisce;

- cambia l'aspetto del bottone “Corona”, utile per tornare alla schermata Kingdom, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- cambia l'aspetto del bottone “impostazioni”, utile per tornare alla schermata Impostazioni, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- avvia l'activity MainActivity, mediante intent esplicito, quando viene rilevato il click del bottone “home” attraverso un clickListener; l'intent esplicito non prevede passaggio di informazioni tra le activity coinvolte; se l'utente si trova già nella MainActivity, il click del bottone non genera alcun evento;
- avvia l'activity KingdomActivity, mediante intent esplicito, quando viene rilevato il click del bottone “Corona” attraverso un clickListener; l'intent esplicito non prevede passaggio di informazioni tra le activity coinvolte; se l'utente si trova già nella KingdomActivity, il click del bottone non genera alcun evento;
- avvia l'activity settingsActivity, mediante intent esplicito, quando viene rilevato il click del bottone “impostazioni” attraverso un clickListener; l'intent esplicito non prevede passaggio di informazioni tra le activity coinvolte; se l'utente si trova già nella SettingsActivity, il click del bottone non genera alcun evento;

4. ShopFragment:

- **tipologia classe:** fragment;
- **classe estesa:** Fragment; viene eseguito l'override del metodo onCreateView(...);
- **funzionalità:** la classe ShopFragment racchiude le seguenti funzionalità:
 - crea un'istanza del viewModel “ShopViewModel” che gestisce l'interazione con il database e la persistenza dei dati mediante oggetti observable;
 - esegue linflate del file .xml di layout “fragment_shop”;
 - mappa layout e view espresse nel file .xml “fragment_shop” in oggetti mediante dataBinding;
 - lega l'istanza del viewModel al file .xml “fragment_shop”;
 - consulta il file sharedPreference memorizzato localmente sul dispositivo e, a seconda del valore letto, imposta l'UI dell'app in modalità mancino o in modalità destroverso;
 - cambia l'aspetto dei bottoni direzionali di avatar e quadri, utili per scorrere avatar/quadri acquistati e acquistabili, mediante TouchListener ogni qualvolta l'utente vi interagisce;

- gestisce lo switch dell'avatar/quadro correntemente mostrato a schermo mediante ClickListener;
 l'avatar/quadro viene mostrato a schermo mediante il metodo setBackgroundResource;
 setBackgroundResource prende, come parametro attuale, il percorso dell'imageResource ottenuto mediante il mapping fornito dal metodo fromShopElementNameToResource appartenente alla classe ResourceHelper;
 la classe ResourceHelper è stata precedentemente importata;
- controlla se ogni avatar/quadro correntemente visualizzato dall'utente è stato acquistato o meno;
 - se l'opera d'arte non è stata acquistata, il sistema mostra il costo e l'icona della moneta;
 - se l'opera d'arte è stata acquistata, il sistema mostra un tick; in particolare, le operazioni appena descritte vengono gestite dal metodo setPriceVisibility(...);
- esegue una transaction per mostrare a schermo OrderDetailFragment, che esegue linflate del file “fragment_order_detail”, quando viene rilevato il click di un'avatar o di un quadro attraverso un clickListener;
 poiché la stessa transaction deve essere eseguita sia in caso si esegua tap su un'avatar e sia in caso si esegua tap su un quadro, si è deciso di implementare la logica della transaction nel metodo “fragmentTransactionOrderDetail”;
 il metodo “fragmentTransactionOrderDetail” serve anche a:
 - passare, al costruttore della classe OrderDetailFragment, il valore “true” se l'avatar/quadro selezionato è già stato acquistato;
 in questo modo, in OrderDetailFragment, il bottone “buy” non verrà mostrato a schermo;
 - passare, al costruttore della classe OrderDetailFragment, il valore “false” se l'avatar/quadro selezionato non è stato acquistato;
 in questo modo, in OrderDetailFragment, il bottone “buy” sarà visibile a schermo;
 - tramite un bundle, passare le seguenti informazioni tra fragment:
 - avatar selezionato;
 - id dell'avatar selezionato;
 - prezzo dell'avatar;

- quadro selezionato;
- id del quadro selezionato;
- prezzo del quadro;
- **Nota:** nella classe ShopFragment è presente la seguente condizione:
`< if(this.arguments?.isEmpty == false) {...} else{...}>`
tuttavia, al momento, non disponiamo degli elementi necessari per avviare la trattazione di tale blocco di codice;
il discorso intorno alla condition appena espressa è presentato nell'analisi del fragment OrderDetailFragment;

5. OrderDetailFragment:

- **tipologia classe:** fragment;
- **classe estesa:** Fragment; viene eseguito l'override del metodo onCreateView(...);
- **funzionalità:** la classe ShopFragment racchiude le seguenti funzionalità:
 - esegue linflate del file .xml di layout “fragment_order_detail”;
 - mappa layout e view espresse nel file .xml “fragment_order_detail” in oggetti mediante dataBinding;
 - mostra a schermo lavatar/quadro selezionato mediante il metodo setBackgroundResource;
setBackgroundResource prende, come parametro attuale, il percorso dellimageResource ottenuto mediante il mapping fornito dal metodo fromShopElementNameToResource appartenente alla classe ResourceHelper;
la classe ResourceHelper è stata precedentemente importata;
 - mostra a schermo il titolo dellavatar/quadro selezionato mediante il metodo getString;
getString prende, come parametro attuale, il percorso della stringResource ottenuto mediante il mapping fornito dal metodo fromShopElementNameToLocalizedName appartenente alla classe ResourceHelper;
la classe ResourceHelper è stata precedentemente importata;
 - controlla se lavatar/quadro selezionato dallutente è stato acquistato o meno; in questo modo:
 - mostra a schermo il tasto “buy” se lopera darte non è ancora stata acquistata;
 - non mostra a schermo il tasto “buy” se lopera darte è stata acquistata;

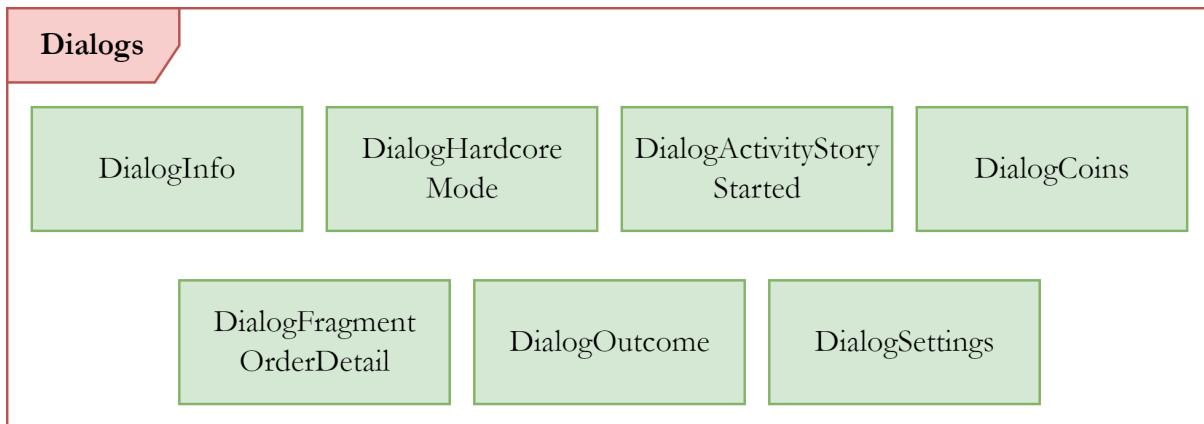
- cambia l'aspetto del bottone “indietro”, utile per tornare alla schermata ShopFragment, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- esegue una transaction per mostrare a schermo lo ShopFragment, che esegue linflate del file “fragment_shop”, quando viene rilevato il click del bottone “indietro” attraverso un clickListener; l'attivazione di questa transaction memorizza, in un bundle, le informazioni relative all'opera d'arte selezionata in modo che, una volta cliccato il tasto indietro e mostrato lo ShopFragment, l'opera d'arte correntemente visualizzata a schermo sia esattamente quella antecedente all'avvio di OrderDetailFragment; se quanto descritto non fosse stato implementato, ad ogni click del tasto “indietro” in OrderDetailFragment, lo ShopFragment mostrerebbe sempre le opere d'arte di default, ovvero:
 - Il Figlio dell'Uomo nella sezione avatar;
 - L'Urlo, di Munch, nella sezione quadri;
 per intendere al meglio quanto appena descritto, avanziamo il seguente esempio;
 - accedendo per la prima volta allo shop, visualizzo le opere di default (L'Urlo e Il Figlio dell'Uomo);
 - supponiamo di scorrere le frecce direzionali e di selezionare l'avatar David;
 - selezionando l'avatar David mediante click, apro OrderDetailFragment;
 - ora, se memorizzo le informazioni in un bundle, quando torno allo shop mediante tasto “indietro”, l'avatar correntemente visualizzato è David; al contrario, se non memorizzo le informazioni in un bundle, quando torno allo shop mediante tasto “indietro” non ho modo di conoscere quale avatar è stato selezionato dall'user e, di conseguenza, l'avatar correntemente visualizzato è quello di default (Il Figlio dell'Uomo) e non David; quanto appena detto viene codificato, in ShopFragment, nella condizione:


```
< if(this.arguments?.isEmpty == false) {...} else{...}>;
```

 in particolare:
 - se la condizione è vera, allora viene mostrata l'opera d'arte selezionata dall'user prima di avviare OrderDetailFragment; in più, sono presenti dei controlli per determinare se l'artwork è stato acquistato o meno in modo da mostrare il costo o il tick;

- se la condizione è falsa, vengono mostrate le opere d'arte di default;
 - le opere d'arte di default, ovviamente, sono già acquistate, ovvero sono possedute dall'utente by design;
- cambia l'aspetto del bottone “info”, utile per apprezzare la storia dell'opera d'arte selezionata, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- istanzia un oggetto della classe DialogInfo, che mostra il dialog contenente la storia dell'artwork selezionato, quando viene rilevato, mediante listener, il click del bottone “info”;
 - al costruttore della classe DialogInfo viene passato, come parametro attuale, il percorso della stringResource ottenuto mediante il mapping fornito dal metodo fromShopElementNameToDescription appartenente alla classe ResourceHelper;
 - in altre parole, ad essere passata, come parametro formale, è la storia dell'opera d'arte che verrà mostrata nel dialog;
 - la classe ResourceHelper è stata precedentemente importata;
- cambia l'aspetto del bottone “buy”, utile per acquistare l'opera d'arte selezionata, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- istanzia un oggetto della classe DialogFragmentOrderDetail, che mostra il dialog di checkout ordine, quando viene rilevato, mediante listener, il click del bottone “buy”;

UI controllers: Dialogs



1. DialogInfo:

- **tipologia classe:** dialog;
- **classe estesa:** DialogFragment; viene eseguito l'override del metodo onCreateView(...);
- **funzionalità:** la classe DialogInfo racchiude le seguenti funzionalità:

- esegue l'inflate del file .xml di layout “dialog_info”;
- mappa le view espresse nel file .xml “dialog_info” in oggetti mediante dataBinding;
- mostra la storia dell’opera d’arte selezionata dall’utente;
- cambia l’aspetto del bottone “okay”, utile a chiudere il dialog, mediante TouchListener ogni qualvolta l’utente vi interagisce;
- chiude il dialog quando viene rilevato il click del bottone “okay” attraverso un clickListener, oppure quando l’utente clicca una qualsiasi parte dello schermo diversa dal dialog;

2. DialogHardcoreMode:

- **tipologia classe:** dialog;
- **classe estesa:** DialogFragment; viene eseguito l’override del metodo onCreateView(...);
- **funzionalità:** la classe DialogHardcoreMode racchiude le seguenti funzionalità:
 - esegue l'inflate del file .xml di layout “dialog_hardcore_mode”;
 - mappa le view espresse nel file .xml “dialog_hardcore_mode” in oggetti mediante dataBinding;
 - cambia l’aspetto del bottone “okay”, utile a chiudere il dialog, mediante TouchListener ogni qualvolta l’utente vi interagisce;
 - avvia l’activity MainActivity, mediante intent esplicito, quando viene rilevato il click del bottone “okay” attraverso un clickListener, oppure quando l’utente chiude il dialog cliccando una qualsiasi parte dello schermo fuori dal dialog stesso; l’intent esplicito non prevede passaggio di informazioni tra dialog ed activity;

3. DialogActivityStoryStarted:

- **tipologia classe:** dialog;
- **classe estesa:** DialogFragment; viene eseguito l’override del metodo onCreateView(...);
- **funzionalità:** la classe DialogActivityStoryStarted racchiude le seguenti funzionalità:
 - esegue l'inflate del file .xml di layout “dialog_activity_story_started”;
 - mappa le view espresse nel file .xml “dialog_activity_story_started” in oggetti mediante dataBinding;
 - cambia l’aspetto del bottone “okay”, utile a chiudere il dialog, mediante TouchListener ogni qualvolta l’utente vi interagisce;

- chiude il dialog quando viene rilevato il click del bottone “okay” attraverso un clickListener, oppure quando l’utente clicca una qualsiasi parte dello schermo diversa dal dialog;

4. DialogCoins:

- **tipologia classe:** dialog;
- **classe estesa:** DialogFragment; viene eseguito l’override del metodo onCreateView(...);
- **funzionalità:** la classe DialogCoins racchiude le seguenti funzionalità:
 - esegue l’inflate del file .xml di layout “dialog_coins”;
 - mappa le view espresse nel file .xml “dialog_coins” in oggetti mediante dataBinding;
 - mostra le monete guadagnate dall’user al termine di una sessione di produttività;
 - cambia l’aspetto del bottone “okay”, utile a chiudere il dialog, mediante TouchListener ogni qualvolta l’utente vi interagisce;
 - avvia l’activity MainActivity, mediante intent esplicito, quando viene rilevato il click del bottone “okay” attraverso un clickListener, oppure quando l’utente chiude il dialog cliccando una qualsiasi parte dello schermo fuori dal dialog stesso; l’intent esplicito non prevede passaggio di informazioni tra dialog ed activity;

5. DialogFragmentOrderDetail:

- **tipologia classe:** dialog;
- **classe estesa:** DialogFragment; viene eseguito l’override del metodo onCreateView(...);
- **funzionalità:** la classe DialogFragmentOrderDetail racchiude le seguenti funzionalità:
 - genera un’istanza del viewModel “DialogOrderDetailViewModel” a cui delega la logica dell’activity e la comunicazione con il database;
 - esegue l’inflate del file .xml di layout “dialog_fragment_order_detail”;
 - mappa le view espresse nel file .xml “dialog_fragment_order_detail” in oggetti mediante dataBinding;
 - lega l’istanza del viewModel al file .xml “dialog_fragment_order_detail”;
 - mostra la frase di checkout “sei sicuro di voler acquistare quest’opera per x monete?”; il numero x di monete è impostato dinamicamente a seconda dell’opera d’arte considerata;

- cambia l'aspetto del bottone “no”, utile a chiudere il dialog di checkout, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- chiude il dialog quando viene rilevato il click del bottone “no” attraverso un clickListener, oppure quando l'utente clicca una qualsiasi parte dello schermo diversa dal dialog;
- cambia l'aspetto del bottone “si”, utile a concludere la procedura di checkout, mediante TouchListener ogni qualvolta l'utente vi interagisce;
- istanzia un oggetto della classe DialogOutcome, che mostra il dialog che descrive l'esito positivo o negativo della transazione, quando viene rilevato, mediante listener, il click del bottone “si”;
- chiude il dialog quando quando viene rilevato, mediante listener, il click del bottone “si”;
- esegue la funzione buyShopElement che serve a controllare se l'utente ha monete sufficienti per completare la transazione;
 - in caso di esito positivo, viene sottratto il costo dell'opera alle monete dell'utente e l'artwork viene aggiunto a quelli posseduti dall'utente;
 - in caso di esito negativo, monete e opere d'arte nel database non vengono aggiornate;

6. DialogOutcome:

- **tipologia classe:** dialog;
- **classe estesa:** DialogFragment; viene eseguito l'override del metodo onCreateView(...);
- **funzionalità:** la classe DialogOutcome racchiude le seguenti funzionalità:
 - esegue linflate del file .xml di layout “dialog_outcome”;
 - mappa le view espresse nel file .xml “dialog_outcome” in oggetti mediante dataBinding;
 - configura le textView nel dialog a seconda dell'esito della transazione:
 - se l'acquisto è avvenuto correttamente, vengono mostrati testi con congratulazioni;
 - se l'acquisto non può avvenire a causa dell'insufficienza delle monete dell'user, vengono mostrati testi di invito a provare di nuovo in futuro;
 - cambia l'aspetto del bottone “okay”, utile a chiudere il dialog di checkout, mediante TouchListener ogni qualvolta l'utente vi interagisce;

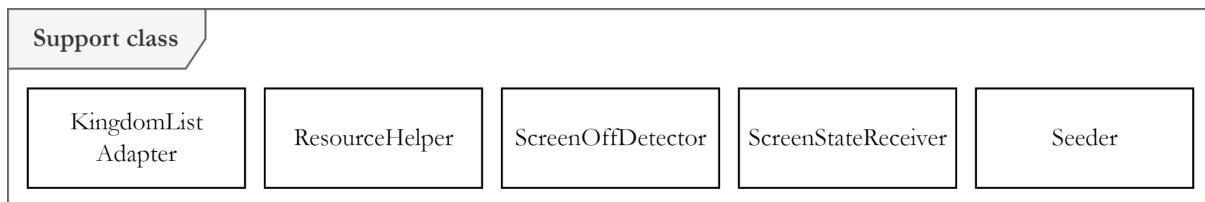
- esegue una transaction, per mostrare a schermo il fragment ShopFragment che, a sua volta, esegue linflate del file “fragment_shop”, e chiude il dialog quando viene rilevato il click del bottone “okay” attraverso un clickListener;

7. DialogSettings:

- **tipologia classe:** dialog;
- **classe estesa:** DialogFragment; viene eseguito loverride del metodo onCreateView(...);
- **funzionalità:** la classe DialogSettings racchiude le seguenti funzionalità:
 - esegue linflate del file .xml di layout “dialog_settings”;
 - mappa le view espresse nel file .xml “dialog_settings” in oggetti mediante dataBinding;
 - cambia l’aspetto del bottone “okay”, utile a chiudere il dialog di checkout, mediante TouchListener ogni qualvolta l’utente vi interagisce;
 - chiude il dialog quando viene rilevato il click del bottone “okay” attraverso un clickListener, oppure quando l’utente clicca una qualsiasi parte dello schermo diversa dal dialog;

Utils: Support class

Continuiamo, ora, l’analisi dei moduli presentando classi che svolgono funzionalità di supporto:



1. KingdomListAdapter:

- **tipologia classe:** recyclerView adapter;
- **classe estesa:** RecyclerView.Adapter<RecyclerView.ViewHolder>; viene eseguito loverride dei metodi onCreateViewHolder(...), getItemCount(), onBindViewHolder(...), getItemViewType(...);
- **funzionalità:** la classe KingdomListAdapter è utilizzata per presentare la lista delle storie dell’utente in una RecyclerView;

2. ResourceHelper:

- **tipologia classe:** non è una classe, è un file contenente metodi;
- **classe estesa:** /

- **funzionalità:** ResourceHelper mette a disposizione i seguenti metodi:
 - **fromShopElementNameToLocalizedName:** mappa una stringa di testo con il percorso di una risorsa nel file strings; in particolare, i percorsi conducono a tag con il nome delle opere d'arte;
 - **fromShopElementNameToResource:** mappa una stringa di testo con il percorso di una risorsa drawable; in particolare, i percorsi conducono a tag con l'immagine completa delle opere d'arte;
 - **fromShopElementNameToDescription:** mappa una stringa di testo con il percorso di una risorsa nel file strings; in particolare, i percorsi conducono a tag con la descrizione della storia delle opere d'arte;
 - **encouragementQuotes:** assegna ad una variabile l'id di una risorsa nel file strings; le variabili sono poi raccolte in una collection; in particolare, i percorsi conducono a tag con frasi di incoraggiamento;
 - **pauseQuotes:** assegna ad una variabile l'id di una risorsa nel file strings; le variabili sono poi raccolte in una collection; in particolare, i percorsi conducono a tag con frasi di invito al riposo e di congratulazioni;

3. ScreenOffDetector:

- **tipologia classe:** broadcast receiver;
- **classe estesa:** BroadcastReceiver; viene eseguito l'override del metodo onReceive(...);
- **funzionalità:** la classe ScreenOffDetector racchiude le seguenti funzionalità:
 - rileva se l'utente blocca il cellulare; in altre parole, rileva se lo schermo diviene non interattivo, ovvero se l'utente preme il pulsante di blocco del device oppure se il device va in standby causa inattività;
 - rileva il task appena eseguito dall'utente come, ad esempio:
 - apertura di un'app diversa da WIP mediante notifica o mediante icona;
 - apertura del task manager, ovvero del gestore che ci permette di chiudere le app e di vedere quelle in background;
 - visualizzazione della schermata home dopo aver premuto il tasto "home" del device, ecc...

- controlla se l'activity, relativa al task appena eseguito dall'utente, è StoryStartedActivity; in caso di esito positivo, viene settato a true il flag che, in modalità hardcore, viene utilizzato per comunicare al sistema che l'user non ha eseguito azioni illegittime;

4. ScreenStateReceiver:

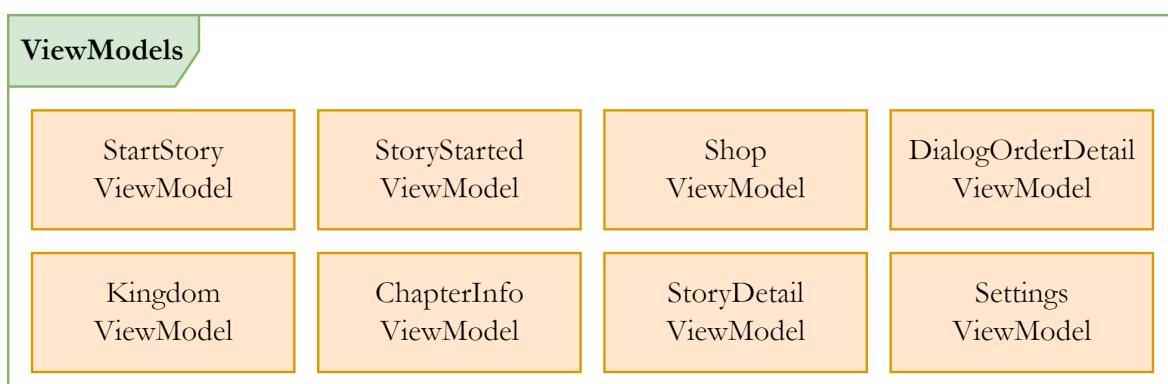
- **tipologia classe:** broadcast receiver;
- **classe estesa:** BroadcastReceiver; viene eseguito l'override del metodo onReceive(...);
- **funzionalità:** la classe ScreenStateReceiver racchiude le seguenti funzionalità:
 - rileva se l'utente sblocca il device, ovvero se lo schermo diviene interattivo;
 - seleziona randomicamente, e mostra a schermo in ActivityStoryStarted, una frase motivazionale nel caso lo sblocco del device avvenga durante il tempo di studio;
 - seleziona randomicamente, e mostra a schermo in ActivityStoryStarted, una frase di invito alla pausa o un incoraggiamento nel caso lo sblocco del device avvenga durante il tempo di pausa;

5. Seeder:

- **tipologia classe:** non è una classe, è un file contenente un metodo;
- **classe estesa:** /
- **funzionalità:** il file Seeder contiene la funzione seed utile a popolare la base di dati;

Controller indipendenti dall'activity lifecycle: ViewModels

Concludiamo la trattazione dei moduli dell'architettura dell'app WIP presentando i viewModel:



1. StartStoryViewModel:

- **tipologia classe:** view model;
- **classe estesa:** AndroidViewModel;
- **funzionalità:** la classe StartStoryViewModel racchiude le seguenti funzionalità:
 - mantiene oggetti LiveData come variabili d'istanza; tali oggetti popoleranno l'interfaccia utente con dati che sopravvivono all'activity lifecycle;
 - gestisce la comunicazione con il database;
 - carica la lista delle storie precedentemente inserite come titolo; tali stringhe sono, poi, suggerite all'user mentre inserisce il titolo della nuova storia nell'apposita editText;
 - carica la lista degli avatar selezionabili dall'utente (ovvero, gli avatar acquistati);
 - memorizza tutti i parametri configurabili dall'utente in ActivityStartStory, ovvero:
 - titolo della storia;
 - partizione studio-pausa;
 - modalità opzionale;
 - avatar;
 - in questo modo, se esco e rientro nell'app senza chiuderla, i valori inseriti non subiscono refresh;
 - gestisce la logica della modalità silenziosa, attivando e disattivando la vibrazione;
 - gestisce la logica della modalità hardcore e l'interazione con la modalità silenziosa;

2. StoryStartedViewModel:

- **tipologia classe:** view model;
- **classe estesa:** AndroidViewModel;
- **funzionalità:** la classe StoryStartedViewModel racchiude le seguenti funzionalità:
 - gestisce la comunicazione con il database;
 - genera un'istanza della classe ScreenOffDetector per il rilevamento delle azioni consentite dall'user in hardcore mode;
 - carica la lista dei quadri/background posseduti dall'utente (ovvero, i quadri acquistati);
 - calcola le monete guadagnate dall'utente al termine della sessione di produttività; l'algoritmo di calcolo è stato esplicato nei task relativi all' User story 5: Conclusione storia;

- aggiorna le monete dell'utente mediante scrittura su database;
- memorizza la storia appena conclusa nel Kingdom;
- memorizza il capitolo relativo alla storia appena conclusa;

3. ShopViewModel:

- **tipologia classe:** view model;
- **classe estesa:** AndroidViewModel;
- **funzionalità:** la classe ShopViewModel racchiude le seguenti funzionalità:
 - mantiene oggetti LiveData come variabili d'istanza; tali oggetti popoleranno l'interfaccia utente con dati che sopravvivono all'activity lifecycle;
 - gestisce la comunicazione con il database;
 - carica le monete possedute dall'utente;
 - carica la lista degli avatar contenuti nella base di dati;
 - carica la lista dei quadri contenuti nella base di dati;
 - carica la lista delle opere d'arte possedute, ovvero già acquistate, dall'utente;

4. DialogOrderDetailViewModel:

- **tipologia classe:** view model;
- **classe estesa:** AndroidViewModel;
- **funzionalità:** la classe DialogOrderDetailViewModel racchiude le seguenti funzionalità:
 - mantiene oggetti LiveData come variabili d'istanza; tali oggetti popoleranno l'interfaccia utente con dati che sopravvivono all'activity lifecycle;
 - gestisce la comunicazione con il database;
 - gestisce la procedura di acquisto dell'opera d'arte selezionata dall'utente mediante il metodo “buyShopElement”; in tale metodo, se il numero delle monete dell'user è sufficiente ad acquistare l'opera d'arte, viene eseguita una coroutine che invoca i seguenti metodi:
 - **updateUserCoins:** aggiorna, nella base di dati, il numero di monete possedute dall'user;
 - **insertNewShopped:** aggiorna la base di dati inserendo, tra gli elementi acquistati, l'opera d'arte appena comprata;

5. KingdomViewModel:

- **tipologia classe:** view model;
- **classe estesa:** AndroidViewModel;

- **funzionalità:** la classe KingdomViewModel racchiude le seguenti funzionalità:
 - gestisce la comunicazione con il database;
 - genera la lista delle storie eseguite dall'utente;
 - ordina le storie dalla più recente alla meno recente;
 - genera la lista dei capitoli relativa a ciascuna storia dell'utente;

6. ChapterInfoViewModel:

- **tipologia classe:** view model;
- **classe estesa:** AndroidViewModel;
- **funzionalità:** la classe ChapterInfoViewModel racchiude le seguenti funzionalità:
 - gestisce la comunicazione con il database;
 - raccoglie tutte le informazioni relative al capitolo selezionato dall'utente;

7. StoryDetailViewModel:

- **tipologia classe:** view model;
- **classe estesa:** AndroidViewModel;
- **funzionalità:** la classe StoryDetailViewModel racchiude le seguenti funzionalità:
 - gestisce la comunicazione con il database;
 - raccoglie tutte le informazioni relative ai capitoli della storia selezionata dall'utente;

8. SettingsViewModel:

- **tipologia classe:** view model;
- **classe estesa:** AndroidViewModel;
- **funzionalità:** la classe SettingsViewModel racchiude le seguenti funzionalità:
 - mantiene oggetti LiveData come variabili d'istanza; tali oggetti popoleranno l'interfaccia utente con dati che sopravvivono all'activity lifecycle;
 - gestisce la comunicazione con il database;
 - memorizza tutti i parametri configurabili dall'utente in SettingsActivity, ovvero:
 - massimo tempo limite della sessione studio-pausa;
 - partizione studio-pausa;
 - modalità mancino;

- memorizza, mediante coroutines, i parametri configurabili dall'utente nel database;

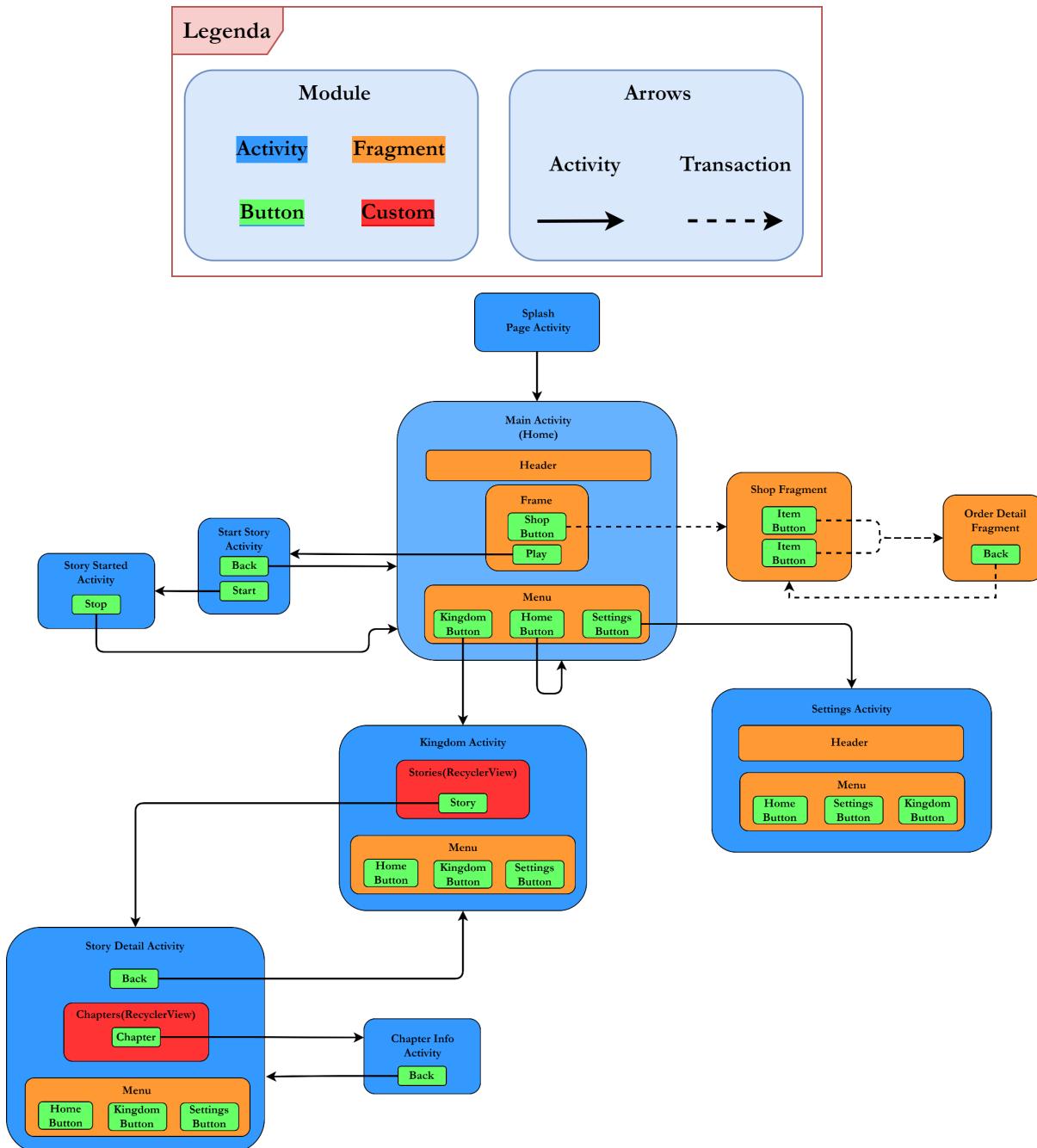
Nota: nell'analisi dell'architettura dell'applicazione non si considerano:

- le classi DAO, poiché mappano query SQL di immediata comprensione;
- le classi Entity, poiché mappano le stesse identiche entità della base di dati; a tali entità è stato dedicato un'intera sezione del documento progettuale;

Intent map

mappa degli intent e delle transaction

Per illustrare al meglio la gestione degli intent e delle transaction tra activity e fragment, riportiamo la seguente mappa.



Test

unit tests, UI tests e database tests

Per effettuare i Test che certificano il corretto funzionamento di alcuni aspetti della nostra app, ci siamo serviti del modulo JUnit.
in particolare, i test eseguiti sono:

Unit tests: suite di test: ShopElementNameTest:

1. **findStringResource:** il seguente test serve per far vedere che il mapping delle risorse stringa con stringhe, proposto dal metodo “fromShopElementNameToLocalizedName”, funziona correttamente; infatti, passando il parametro attuale "girl_with_pearl_earring", il metodo ritorna “R.string.girl_with_a_pearl_earring”; il metodo assertEquals dimostra quanto appena espresso;

```
class ShopElementNameTest { // Testing methods inside "it.wip.utils.ResourceHelper" file
    @Test
    fun findStringResource() {
        val nameResource = fromShopElementNameToLocalizedName( shopElementName: "girl_with_pearl_earring")
        assertEquals(R.string.girl_with_a_pearl_earring, nameResource)
    }
}
```

2. **findDrawableResource:** il seguente test serve per far vedere che il mapping delle risorse drawable con stringhe, proposto dal metodo “fromShopElementNameToResource”, funziona correttamente; infatti, passando il parametro attuale “magritte_apple”, il metodo ritorna “R.drawable.magritte”; il metodo assertEquals dimostra quanto appena espresso;

```
@Test
fun findDrawableResource() {
    val drawableResource = fromShopElementNameToResource( shopElementName: "magritte_apple")
    assertEquals(R.drawable.magritte, drawableResource)
}
```

3. **findDescriptionResource:** il seguente test serve per far vedere che il mapping delle risorse string con stringhe, proposto dal metodo “fromShopElementNameToDescription”, funziona correttamente; infatti, passando il parametro attuale “the_scream”, il metodo ritorna

“R.string.urlo_di_munch_description”;
il metodo assertEquals dimostra quanto appena espresso;

```
@Test  
fun findDescriptionResourceWrong(){  
    val descriptionResource = fromShopElementNameToDescription( shopElementName: "the_scream")  
    assertEquals("The Scream", descriptionResource) // FAILURE  
}
```

Database tests:
suite di test: DatabaseTest:

1. **getFirstChapter:** si occupa di verificare che il capitolo “chapter1”, istanziato dentro la classe getChaptersById, e relativo alla seconda storia nel DB, sia uguale alla stringa di testo “Capitolo 1”;
2. **getSecondChapter:** si occupa di verificare che il capitolo “chapter2”, istanziato dentro la classe getChaptersById, e relativo alla seconda storia nel DB, sia uguale alla stringa di testo “Capitolo 2”;
3. **insertAndRetrive:** richiama il metodo insertStoryWithChapter, che si occupa di inserire sequenzialmente una nuova storia con un capitolo nel DB. Per questo utilizza i due metodi insertWithoutCoroutines(<storyId>) & insertWithoutCoroutines(<chapterId>). Quindi verifica che il nome della storia e del capitolo inseriti corrispondano a quelli estratti tramite un getAll by story ID per la storia e un getAll by chapter ID (IDs: 8 per la storia, 10 per il capitolo).

UI tests:
suite di test: StartStoryActivityTest:

1. **newStoryName:** questo test verifica come prima cosa che si mostri a schermo un edit text con autocompletamento situato all'interno dell'activity Start Story; di seguito cancella in testo al suo interno e sovrascrive la parola “Prova” per poi verificarne il valore mediante un metodo di controllo:
.check(matches(withText("Prova"))).
2. **isSilentModeSelected:** questo test, all'aprire l'activity menzionata, effettua il click sullo switch di “Silent Mode” e successivamente verifica che questo risulti cliccato.

3. **isHardcoreModeSelected:** questo test, all'aprire l'activity menzionata, effettua il click sullo switch di “Hardcore Mode” e successivamente verifica che questo risulti cliccato.

suite di test: MainActivityTest:

1. **activityShown:** questo test verifica semplicemente se il layout principale viene mostrato a schermo.

suite di test: KingdomActivityTest:

1. **scrollBottomRecyclerView:** questo test effettua lo scroll fino in fondo della recycler view verticale contenente le varie storie dell'utente in Kingdom.
2. **clickItemOnRecyclerView:** questo test effettua il click sul 5 elemento nel recycler view verticale contenente le varie storie.

Contributi

lavorare in team

L'app Work Is Progress è stata progettata in modo tale da ripartire, in egual misura, la responsabilità dei membri del team rispetto a ciascuna delle attività costitutive del processo software.

Di seguito, si illustrano i contributi di ciascuno dei partecipanti allo sviluppo dell'app:

Colleuori Federico	<ul style="list-style-type: none">● User story● Glossario dei termini● Requisiti funzionali utente● Requisiti funzionali sistema● Use case● Documentazione degli use case● Dizionario dei dati● Mappa dell'architettura● Analisi dei moduli dell'architettura● Documentazione Android● Test● Codice: in generale, tutto il codice ma approfondimento verso la sezione Kingdom● generazione apk
Frisi Emanuele	<ul style="list-style-type: none">● User story● Requisiti funzionali utente● Requisiti funzionali sistema● Use case● Documentazione degli use case● Implementazione del database● Mappa dell'architettura● Analisi dei moduli dell'architettura● Documentazione Android● Documentazione Flutter● Intent map● Codice: in generale, tutto il codice ma approfondimento verso la sezione Shop● generazione apk
	<ul style="list-style-type: none">● User story● Divisione delle user story in tasks● UI● Requisiti funzionali utente● Requisiti funzionali sistema● Use case

Giusti Kevin	<ul style="list-style-type: none"> ● Documentazione degli use case ● Vincoli non esprimibili ● Mappa dell'architettura ● Analisi dei moduli dell'architettura ● Documentazione Android ● Intent map ● Codice: in generale, tutto il codice ma approfondimento verso la sezione startStory
---------------------	--

Strumenti utilizzati

IDE, librerie, risorse online, ecc...

Per sviluppare l'applicazione WIP ci si è avvalsi dei seguenti strumenti (per i quali è possibile effettuare, in caso di software, il download, oppure essere reindirizzati al sito, tramite CTRL + Clic sul nome stesso dello strumento):

1. [Android Studio](#): IDE;
2. [Kotlin](#): linguaggio di programmazione;
3. [Git](#): software per il versioning del software;
4. [Github](#): risorsa online per la gestione del repository;
5. [Google Docs](#): editor online per la produzione della documentazione software;
6. [Stack Overflow](#): risorsa online per soluzione dubbi relativi alla programmazione;
7. [Discord](#): software per le comunicazioni in team;
8. [Anydesk](#): software per condivisione del desktop tra pc remoti;
9. [Pixelart](#): risorsa online per la produzione degli elementi in pixel art;
10. [Draw.io](#): risorsa online per la produzione di diagrammi UML e schemi;
11. [TalkBack](#): tool per l'accessibilità;
12. [Room](#): libreria che fornisce un livello di astrazione su SQLite;
13. [XML](#): metalinguaggio per la definizione dei layout;
14. [JUnit](#): framework per lo unit testing;
15. [Espresso](#): framework per il testing UI in android.

BUG

**abbiamo iniziato con i corollari della Legge di Murphy, finiamo con i corollari
della Legge di Murphy**

1. Quando si attiva la modalità hardcore, e si esegue il blocco e lo sblocco dello schermo in rapidissima successione per più volte, la sessione di produttività termina come se si fosse eseguita un'azione illegale;
il motivo è dovuto al broadcast receiver che necessita di alcuni istanti per rilevare, correttamente, il lock e l'unlock dello schermo;
2. in alcuni casi, l'unico permesso dichiarato nel Manifest non viene correttamente letto e, di conseguenza, l'app crasha;

Fine

Gli studenti:
Colleluori Federico, Frisi Emanuele, Giusti Kevin