Progetto di Ingegneria del Software

Università degli studi di Milano Bicocca

TRILO APP

TRAVEL MONITOR - 1

Relazione finale del progetto

GRUPPO 1

Di Meo Luca, matricola 833325
Fermini Simone, matricola 830748
Poveromo Marco, matricola 830626
Zorat Lorenzo, matricola 830641

Sommario

1. Specifiche dei requisiti

1.1 Definizione delle specifiche dei requisiti

Dato il testo del travel monitor, abbiamo rilevato i seguenti requisiti:

REQUISITI FUNZIONALI

1. Il sistema permette di suddividere i viaggi scelti come preferiti in una routine settimanale, divisi giorno per giorno.
2. Il sistema permette di aggiungere uno o più viaggi ai viaggi preferiti, ordinati giornalmente secondo la routine settimanale.
3. Il sistema permette la scelta tra più tipi di notifiche legate al sistema dei trasporti e relativi ai viaggi preferiti che verranno ricevute dall'utente.
4. Il sistema permette la ricerca di stazioni e itinerari di viaggio in Lombardia.
5. Il sistema permette di visualizzare le informazioni riguardanti i viaggi e le stazioni aggiornate in tempo reale.

REQUISITI NON FUNZIONALI

1. Il sistema possiede una sezione in cui è possibile visualizzare informazioni relative alle notizie social provenienti dai canali ufficiali dei gestori dei servizi ed eventuali notizie su scioperi e/o interruzioni del servizio (Fonte: Twitter ATM, TRENORD)
2. Il sistema deve permettere l'aggiunta di nuovi tipi di notifica in fase programmativa.
3. Il sistema deve visualizzare l'insieme delle principali mappe del sistema dei trasporti in Lombardia (STIBM, LINEE REGIONALI, METRO MILANESE).
4. Il sistema deve permettere di salvare le informazioni principali sull'utente.

1.2 Casi d'uso: formato breve

Dato il diagramma dei casi d'uso, esposto in seguito, si analizzano i seguenti casi d'uso:

CERCA VIAGGIO

Nome del caso d'uso: Cerca viaggio.

Scenario principale di successo:
L'utente che utilizza il sistema vuole cercare all'interno di esso un itinerario di viaggio da un punto di partenza a un punto di arrivo. L'utente quindi aprendo il sistema accede all'apposita sezione di ricerca, inserisce la stazione di partenza, inserisce la stazione di arrivo e la data e l'ora in cui vuole effettuare il viaggio. Il sistema darà in risposta tutti i possibili viaggi composti da una o più tratte tra la stazione di partenza e la stazione di arrivo. I mezzi visualizzati sono ordinati per orario dall'orario scelto nella ricerca.

Scenari alternativi:

1. L'utente inserisce solamente il punto di arrivo, in questo caso il sistema deve forzare l'utente a inserire una stazione di partenza.

2. L'utente inserisce solamente la stazione di partenza, in questo caso il sistema deve forzare l'utente a inserire una stazione di arrivo.
3. Se l'utente inserisce una stazione di partenza uguale a una stazione di arrivo, il sistema deve forzare l'utente a modificare una delle due scelte.
4. Se l'utente non inserisce la data, considerare il datetime (il momento stesso) come data di ricerca.
5. Se l'utente non inserisce una stazione (di arrivo o di partenza) tra quelle proposte nel database, la ricerca non viene effettuata e viene forzato l'utente a correggere la scelta.

VISUALIZZA NOTIZIE SOCIAL

Nome del caso d'uso: Visualizza social.

Scenario principale di successo:
L'utente che utilizza il sistema può accedere ad una sezione che visualizza la pagina Twitter dei principali servizi dei trasporti (ATM, TRENORD), consultando quindi le ultime notizie riguardanti linee e problematiche. L'utente, effettuando il login al social network, può anche interagire con i tweet.

Scenari alternativi:
1. L'utente che cerca di accedere alla pagina non riesce a visualizzare i contenuti per un errore di rete.

VISUALIZZA PREFERITI

Nome del caso d'uso: Visualizza Preferiti

Scenario principale di successo:
L'utente che utilizza il sistema al momento dell'apertura dell'app visualizzerà nella schermata principale tutte le soluzioni di viaggio, (con informazioni annesse) che ha inserito nei preferiti in precedenza. L'utente può muoversi tra le pagine dell'interfaccia per scorrere i preferiti, giorno per giorno (da lunedì a domenica).

Scenari alternativi:

L'utente non ha inserito nei preferiti nessuna soluzione di viaggio. La schermata risulterà vuota con un bottone per la possibilità di ricercare treni per poi aggiungerli ai preferiti.

L'utente visualizzerà i preferiti inseriti in precedenza, ma per uno o più di essi non saranno presenti una o più informazioni aggiuntive in quanto:

Non fornite dal gestore dei trasporti.

Non correttamente elaborate dal sistema.

Non correttamente fornite dalle API

PERSONALIZZA NOTIFICHE

Nome del caso d'uso: Personalizza Notifiche

Scenario principale di successo:

L'utente che utilizza il sistema accedendo alla schermata delle impostazioni può scegliere, tramite bottoni switch on/off, quali notifiche ricevere per le soluzioni di viaggio inserite nei preferiti.

Potrà scegliere tra:

Ritardo delle soluzioni di viaggio

Cancellazione dei treni

Promemoria abbonamento

Potrà inoltre scegliere se ricevere notifiche anche quando non si trova nella città (o nel raggio di tot km) inserita come "Città di residenza" nel profilo personale.

Scenari alternativi:

Se l'utente non accede alla pagina di impostazioni per settare le notifiche, di default non le riceverà.

CONSULTA STAZIONI

Nome del caso d'uso: Consulta Stazioni

Scenario principale di successo:

L'utente che utilizza il sistema vuole consultare lo stato del servizio in una stazione in real-time. L'utente quindi aprendo il sistema accede all'apposita sezione di ricerca ed inserisce il nome della stazione. Il sistema darà in risposta la stazione ricercata, e, cliccando su di essa potrà visualizzare le seguenti informazioni:

Tabellone degli orari di partenza (e arrivo) dei treni in real-time: orario partenza (arrivo) programmato, stazione di destinazione (origine) della cosa, informazioni sul binario di arrivo (se dato).

Scenari alternativi:

Se il binario non è fornito, viene visualizzato un messaggio di errore ("binario sconosciuto")

1.3 Casi d'uso: formato dettagliato

Dato il testo del travel monitor, abbiamo rilevato i seguenti casi d'uso espressi in formato dettagliato in quanto complessi:

INVIA NOTIFICHE

Nome del caso d'uso: Invia Notifiche.

Portata: Il metodo di notificazione all'utente del sistema che si sta progettando.

Livello: Obiettivo utente.

Attore primario: Utente del sistema.

Attore finale: Utente del sistema.

Parti interessate e interessi: Utente del sistema che desidera ricevere notifiche relative ai treni quando lo stabilisce.

Pre-condizioni: Un evento di interesse per l'utente genera un cambio di stato che deve essere notificato all'utente.

Garanzia di successo: La notifica è arrivata con successo all'utente ed era pertinente.

Scenario principale di successo: L'utente che decide di utilizzare il servizio, ha accesso a tutti i viaggi che può aggiungere ai preferiti come specificato nel caso d'uso Aggiungi Preferiti. Una volta che l'utente ha dei viaggi preferiti, vuole essere a conoscenza di eventuali notizie e cambiamenti su questi. Questo è gestito dal sistema tramite notifiche che, al verificarsi di determinate condizioni scelte sempre dall'utente, invierà al telefono dell'utente una notifica con le informazioni riguardanti l'evento di interesse.

Estensioni:

L'utente non trova o non ha alcun viaggio di interesse. Non riceverà notifiche.

Esiste un evento che però non è segnalato dal gestore dei trasporti. L'utente non riceve la notifica.

L'utente ha alcuni viaggi preferiti e ha impostato alcuni eventi di interesse, ma il sistema non invia la notifica per un problema.

L'evento non viene notificato all'utente per un crash dell'applicazione.

L'utente imposta alcuni determinati eventi di interesse ma riceve le notifiche su tutti gli eventi. Questo non deve accadere.

Se l'utente ha impostato le notifiche "silenziose" (non le riceve) quando è fuori città, il sistema potrebbe non notificarlo per una mancanza di permessi.

Elenco delle varianti tecnologiche e dei dati: Le notifiche arriveranno all'utente tramite notifica push. Questi sono a scelta dall'utente tramite un'interfaccia "impostazioni". L'utente può essere interessato ad alcuni eventi ma non a tutti quelli disponibili nel sistema.

Il sistema supporta una serie di eventi a cui l'utente può essere interessato:

Ritardo di un mezzo preferito

Cancellazione di un mezzo preferito

Ricezione o meno di notifiche se fuori città

Promemoria abbonamenti

Frequenza di ripetizione: Ogni volta che un evento d'interesse per l'utente viene registrato dal sistema, che provvederà ad inviare la notifica.

Varie: Nel sistema la ricezione di notifiche se l'utente si trova fuori città è di dubbia implementazione in quanto richiede di conoscere la posizione real-time dell'utente, e anche eventualmente richiedere all'utente quando è da considerarsi fuori città (in km dalla base).

PERSONALIZZA NOTIFICHE

Nome del caso d'uso: Personalizza Notifiche

Portata: Il sistema

Livello: Obbiettivo utente

Attore Primario: Utente

Parti interessate e interessi: L'utente decide:
- se ricevere o meno notifiche su soppressioni o ritardi dei suoi viaggi preferiti
- se abilitare le notifiche solo quando si trova in città
- se ricevere o meno promemoria sul rinnovo dell'abbonamento

Pre-condizioni: nessuna.

Garanzie di successo: Le scelte espresse dall'utente in merito alle notifiche vengono rispettate.

Scenario principale di successo:
1. Non ricevere più notifiche su soppressioni:
- L'utente accede alla pagina delle impostazioni,
- Se non già fatto, imposta di non voler più ricevere notifiche sulle soppressioni dei suoi viaggi preferiti,
- L'utente non riceverà più notifiche sulle soppressioni dei suoi viaggi preferiti.

Scenari alternativi di successo:

1. Ricevere di nuovo notifiche su soppressioni:
- L'utente accede alla pagina delle impostazioni,
- Se non già fatto, imposta di voler ricevere notifiche sulle soppressioni dei suoi viaggi preferiti,
- L'utente riceverà notifiche in caso uno dei suoi viaggi preferiti verrà soppresso.

2. Non ricevere più notifiche su ritardi:
- L'utente accede alla pagina delle impostazioni,
- Se non già fatto, imposta di non voler più ricevere notifiche sui ritardi dei suoi viaggi preferiti,
- L'utente non riceverà più notifiche sui ritardi dei suoi viaggi preferiti.

3. Ricevere di nuovo notifiche sui ritardi:
- L'utente accede alla pagina delle impostazioni,
- Se non già fatto, imposta di voler ricevere notifiche sui ritardi dei suoi viaggi preferiti,

- L’utente riceverà notifiche in caso uno dei suoi viaggi preferiti accumulerà ritardo.

4. Non ricevere più notifiche fuori città:
- L’utente accede alla pagina delle impostazioni,
- Se non già fatto, imposta di non voler più ricevere notifiche fuori città,
- L’utente non riceverà più notifiche se si fuori città (più di 10 km di distanza da casa).

5. Ricevere di nuovo notifiche sui ritardi:
- L’utente accede alla pagina delle impostazioni,
- Se non già fatto, imposta di voler ricevere notifiche anche fuori città,
- L’utente riceverà notifiche anche fuori città (più di 10 km di distanza da casa).

6. Non ricevere più promemoria per il rinnovo dell’abbonamento:
- L’utente accede alla pagina delle impostazioni,
- Se non già fatto, imposta di non voler più ricevere promemoria per il rinnovo dell’abbonamento,
- L’utente non riceverà più promemoria per il rinnovo dell’abbonamento.

7. Ricevere di nuovo promemoria per il rinnovo dell’abbonamento:
- L’utente accede alla pagina delle impostazioni,
- Se non già fatto, imposta di voler ricevere notifiche per il rinnovo dell’abbonamento,
- L’utente riceverà promemoria per il rinnovo dell’abbonamento.

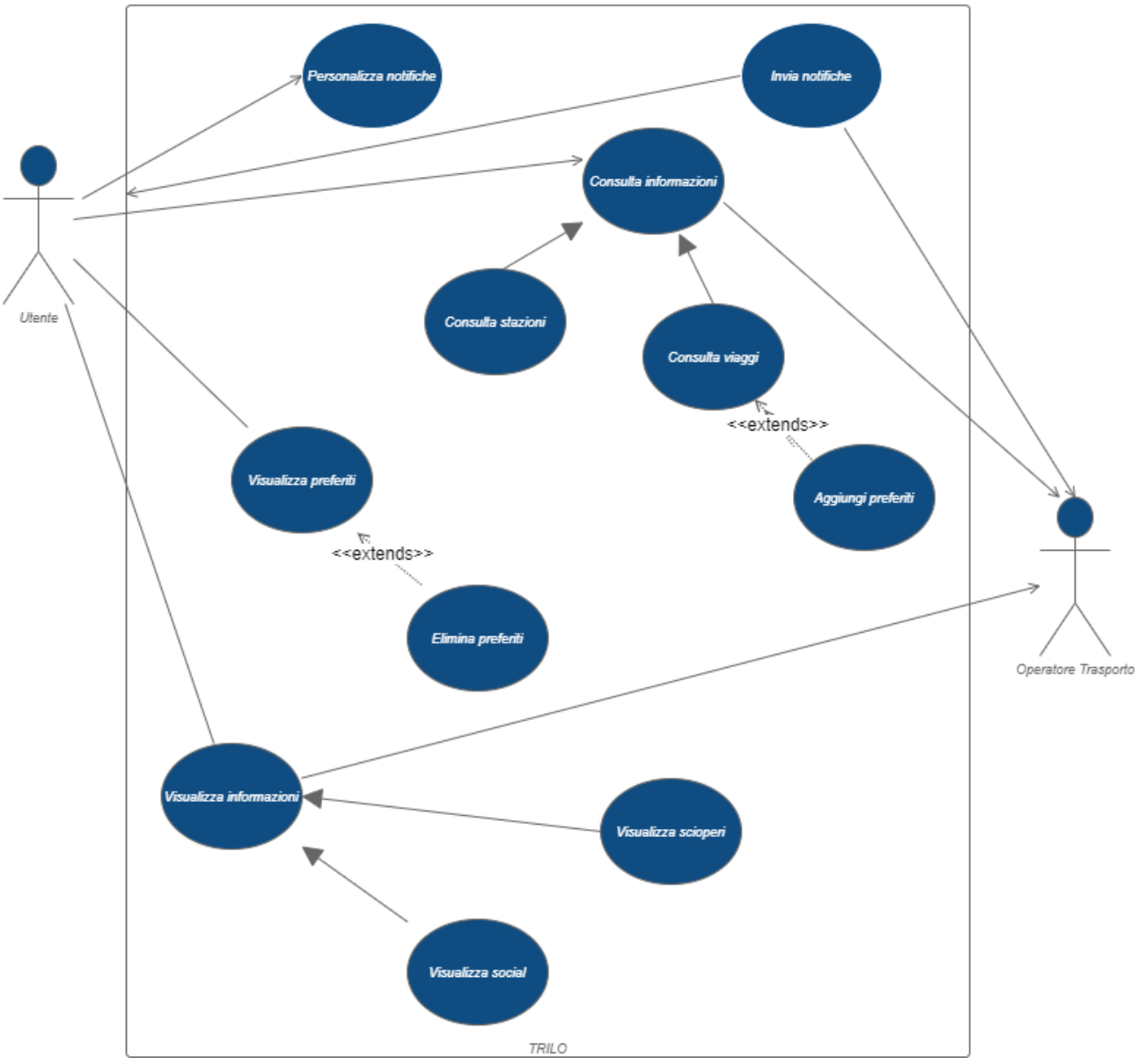
Scenari di fallimento: Se l’utente non modifica nulla non verranno apportate modifiche.

Requisiti speciali:
- L’interfaccia deve essere di facile comprensione.
- I pulsanti devono essere di tipo switch.

Frequenza di ripetizione: Bassa

1.4 Diagramma dei Casi d'uso

Il progetto da sviluppare può essere rappresentato attraverso il seguente diagramma dei casi d'uso:



2. Scelte di programmazione

Qui di seguito sono riportate le principali scelte di programmazione che abbiamo scelto di adottare per quanto riguarda i programmi e il progetto.

2.1 L'Obiettivo

Per il progetto travel monitor, si vuole implementare un sistema per la gestione dei trasporti. Nel nostro caso, abbiamo deciso di dare un taglio all'applicazione più legato alla personalizzazione di una routine settimanale. L'applicazione infatti, ha come scopo principale il salvataggio di soluzioni di viaggio all'interno dei determinati giorni, in modo da ricevere notifiche mirate, anche se non tutti i giorni si effettua la stessa tratta.

2.2 L'ambiente

Una volta accettato il testo del progetto, in fase di riunione si è deciso che la soluzione migliore per creare un "prodotto" quanto più utile possibile per un utente tipo fosse quella di sviluppare un'applicazione (in particolare per dispositivi android). Infatti, l'utente che fruisce dei servizi dei trasporti (in particolare dei treni), si trova in movimento e quindi con il solo telefono a portata di mano. Sarebbe stato altresì irreale sviluppare un qualunque applicativo per Computer.

Android Studio:
Per la programmazione del nostro progetto, come già detto abbiamo scelto di sviluppare un'applicazione Android. Per fare questo, abbiamo utilizzato l'IDE Android Studio.

2.3 Le API Viaggiatreno

L'obiettivo del progetto che ci eravamo prefissati era quello di mantenere un livello di realismo alto, andando a prendere i dati reali servendosi delle API (Application Program Interface). Dopo aver effettuato una fase di ricerca, abbiamo deciso che le uniche API disponibili in tempi brevi e non a pagamento sono quelle fornite basandosi sul sistema ViaggiaTreno.

API ViaggiaTreno:
Le API di ViaggiaTreno forniscono una base di dati utile per avere informazioni relative a Stazioni, treni, orari e tutto quello che riguarda il trasporto ferroviario. Queste sono reperibili al link: https://github.com/bluviolin/TrainMonitor/wiki/API-del-sistema-Viaggiatreno

2.4 Strumenti utilizzati

Per lo sviluppo dell'applicazione finale e per redimere la relazione abbiamo utilizzato diversi programmi tra cui:

- Microsoft Project: per sviluppare il diagramma di Gantt iniziale e finale,
- Draw.io: per disegnare i diagrammi dell'analisi dei requisiti.

Inoltre abbiamo utilizzato gradle per la build del progetto.

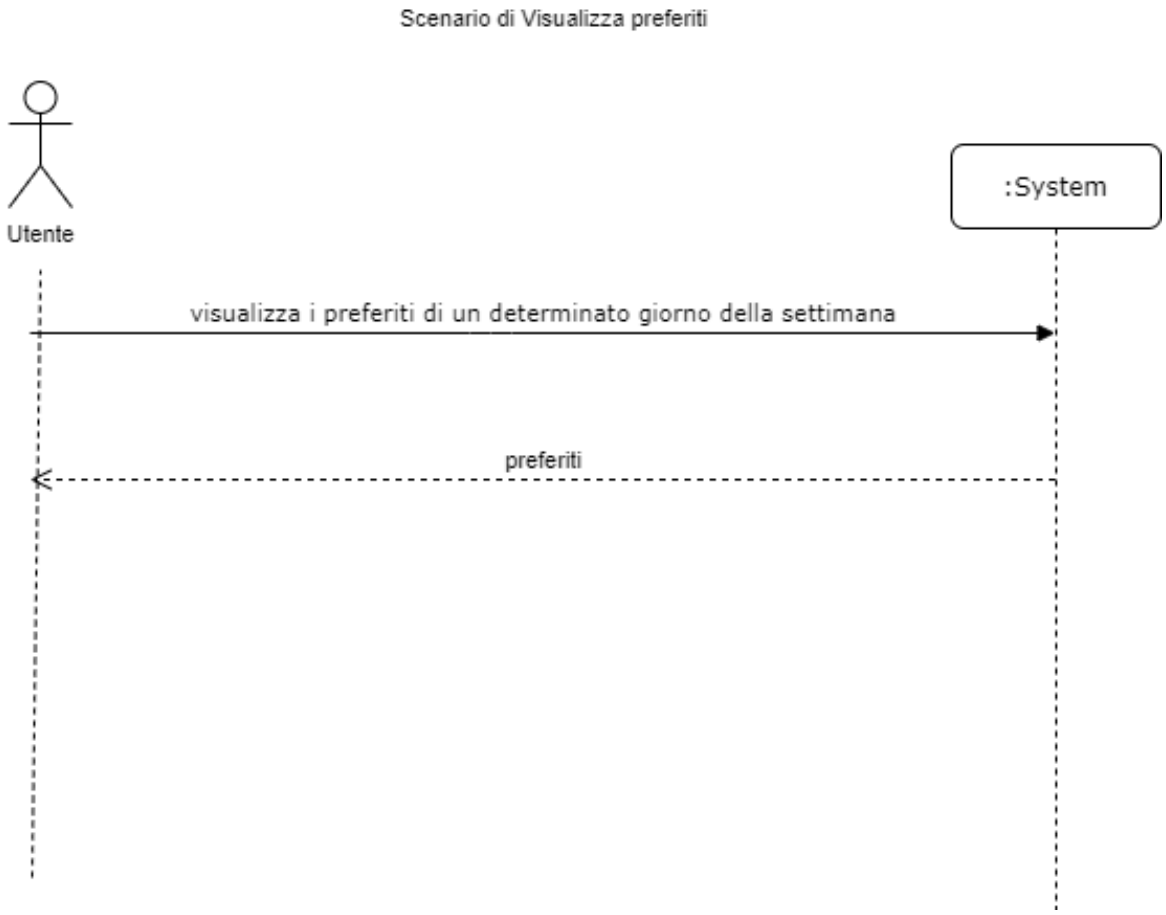
Gradle
Gradle è un sistema open source per l'automazione dello sviluppo fondato sulle idee di Apache Ant e Apache Maven, che introduce un domain-specific language (DSL) basato su Groovy, al posto della modalità XML usata da Apache Maven per dichiarare la configurazione del progetto. Gradle nel nostro caso, è di default implementato in android studio per la build dei progetti.

3. Analisi e progettazione

Per una maggiore comprensione del progetto, abbiamo prodotto diversi diagrammi, utili per uno sviluppo migliore del codice. Per la progettazione si possono fare diversi diagrammi tra cui: diagrammi di attività, diagrammi di stato, diagrammi di sequenza

3.1 Diagramma di sequenza di sistema

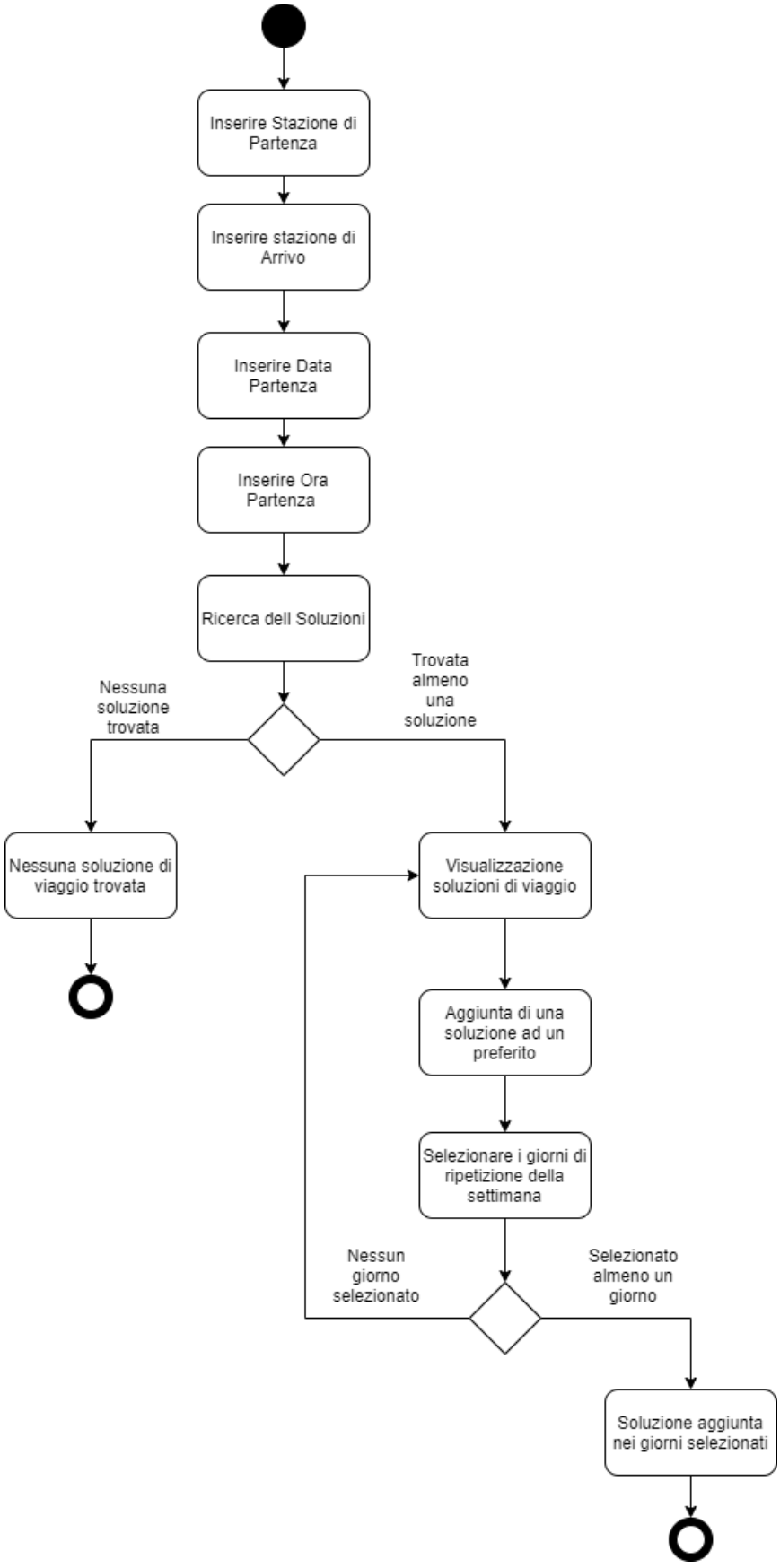
Rappresenta l'interazione dell'utente con il sistema visto come se fosse a scatola chiusa. In questo caso abbiamo analizzato la visualizzazione dei preferiti e la ricerca del viaggio



3.2 Diagramma delle attività

Nel nostro caso abbiamo sviluppato il diagramma delle attività per il caso d'uso cercaviaggio in quanto ne ritenevamo necessario lo sviluppo ai fini di una maggior comprensione del progetto.

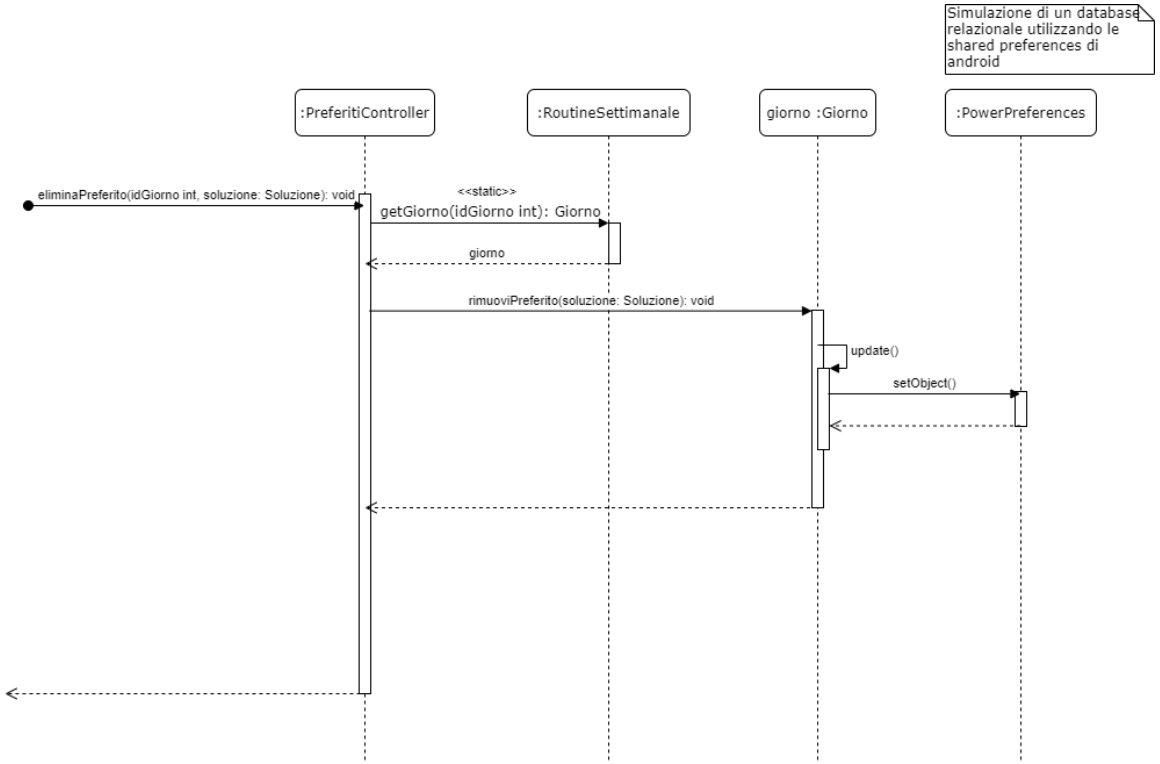
DIAGRAMMA DELLE ATTIVITA' PER IL CASO D'USO CERCA VIAGGIO



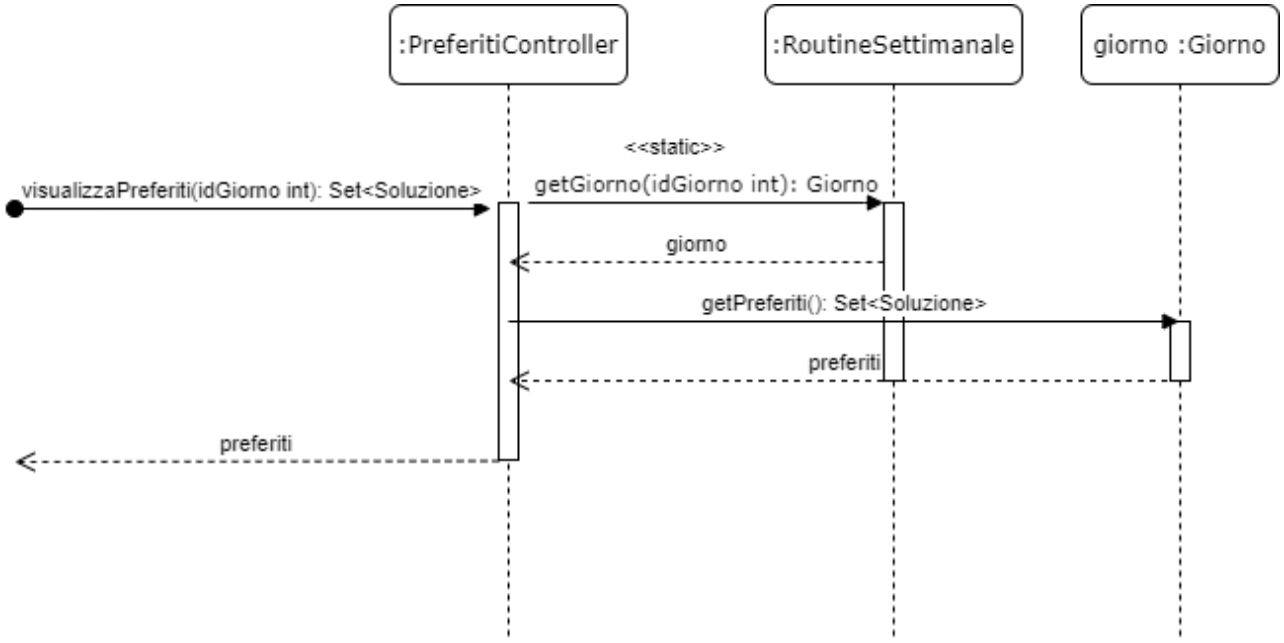
3.3 Diagramma di sequenza

Un diagramma di sequenza descrive la collaborazione di un gruppo di oggetti che devono implementare collettivamente un certo comportamento, che tipicamente rappresenta un singolo scenario di successo. In Trilo abbiamo implementato questo diagramma ad esempio per rendere chiara la gestione della notifica.

ELIMINA PREFERITI

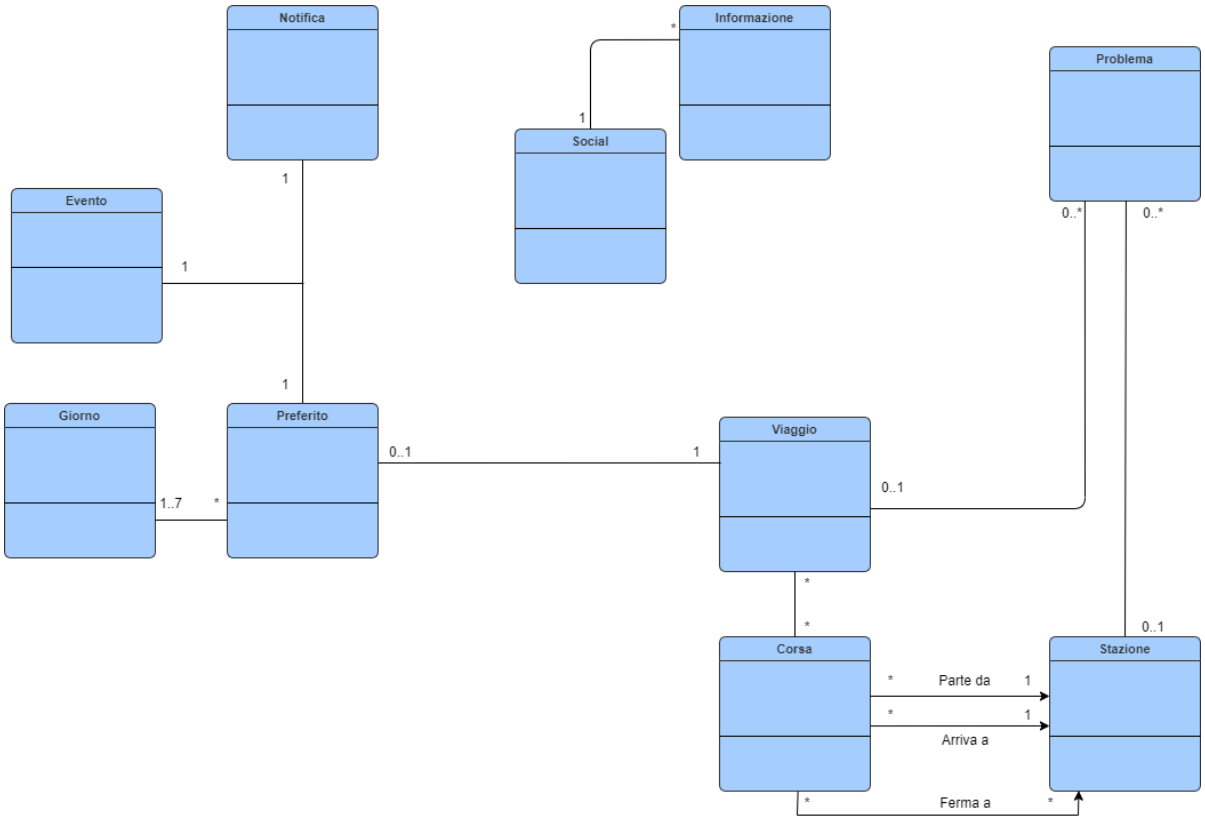


VISUALIZZA PREFERITI

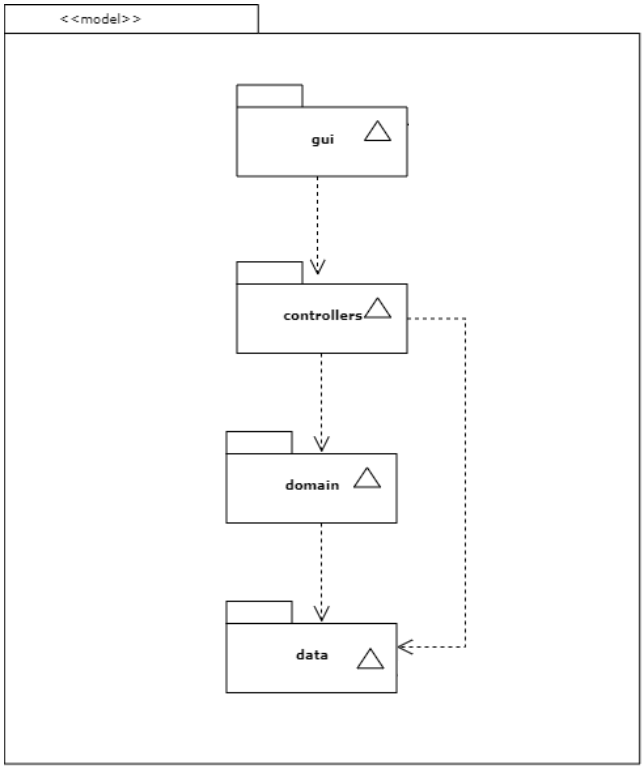


3.4 Modello di dominio

Il seguente modello rappresenta un overview del progetto, ovvero un insieme delle classi principali che formano il cuore del progetto.



3.5 Diagramma dei package

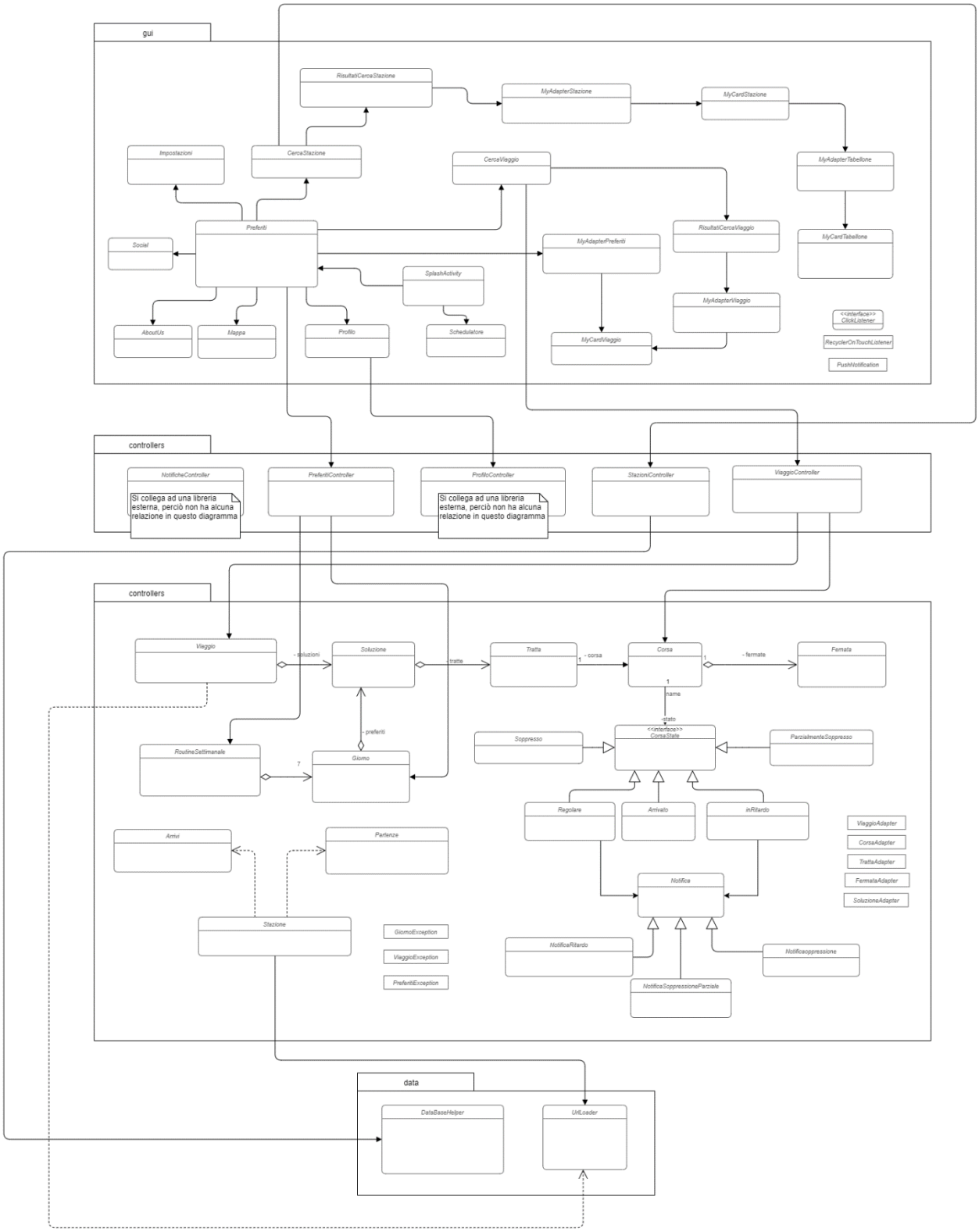


Nel progetto che abbiamo sviluppato sono presenti 4 package. Un package rappresenta un'unità di classi simili (dal punto di vista delle responsabilità) e nel nostro progetto sono i seguenti: gui, domain, data e controller.

3.6 Diagramma delle classi di progetto

Qui dentro si mettono, oltre alle classi, anche i metodi e gli attributi. Inoltre, è fondamentale mantenere una mappatura 1-1 del codice con questo diagramma includendo anche tutti gli eventuali design pattern.

OVERVIEW DEL DIAGRAMMA DELLE CLASSI DI PROGETTO



Nel dettaglio ora, presentiamo il diagramma package per package:

DIAGRAMMA DELLE CLASSI DEL PACKAGE GUI

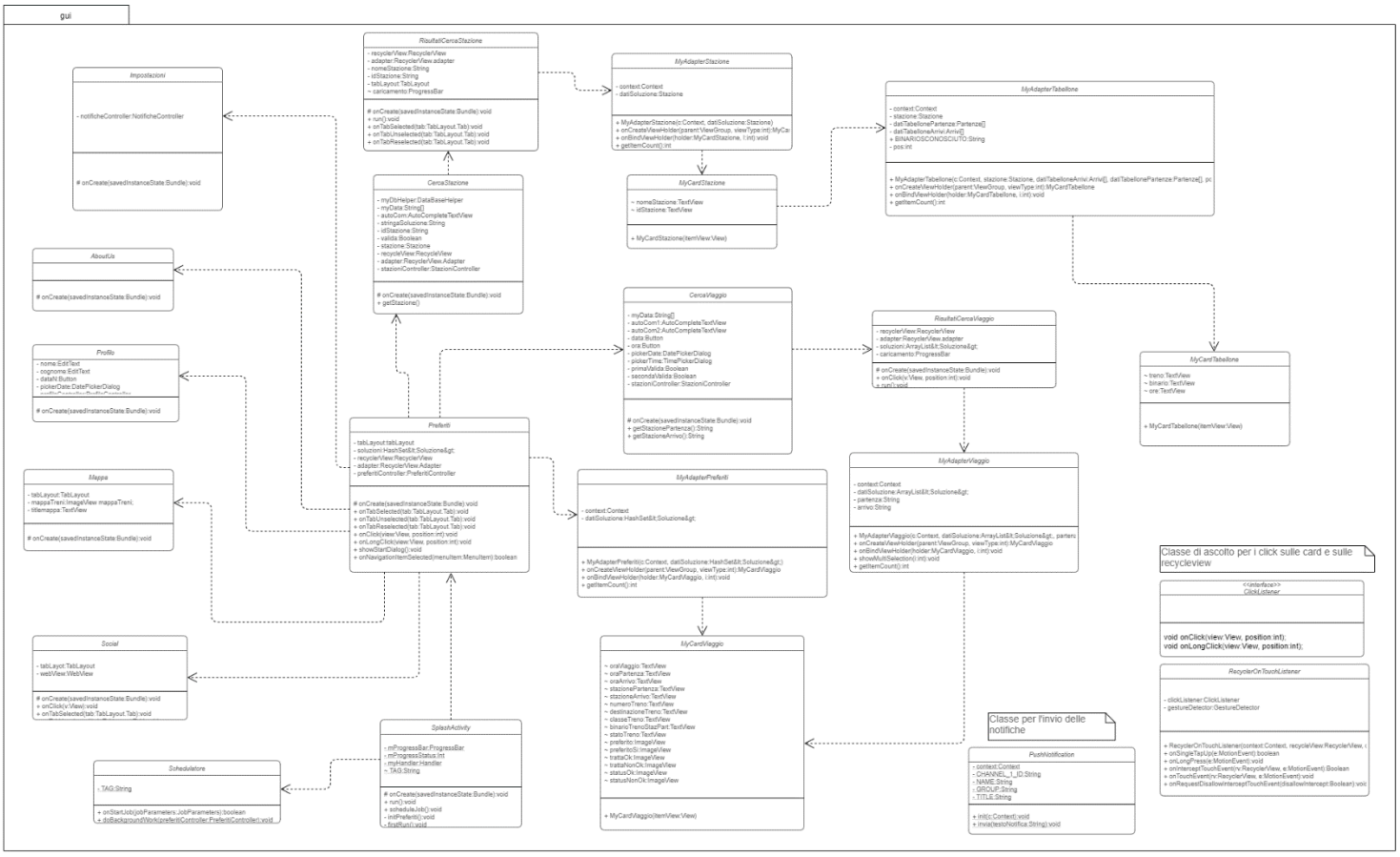

gui
Impostazioni
- notificheController:NotificheController
onCreate(savedInstanceState:Bundle):void
RisultatiCercaStazione
- recyclerView:RecyclerView
- adapter:RecyclerView.adapter
- nomeStazione:String
- idStazione:String
- tabLayout:TabLayout
~ caricamento:ProgressBar
# onCreate(savedInstanceState:Bundle):void
+ run():void
+ onTabSelected(tab:TabLayout.Tab):void
+ onTabUnselected(tab:TabLayout.Tab):void
+ onTabReselected(tab:TabLayout.Tab):void
MyAdapterStazione
- context:Context
- datiSoluzione:Stazione
+ MyAdapterStazione(c:Context, datiSoluzione:Stazione)
+ onCreateViewHolder(parent:ViewGroup, viewType:int):MyCa
+ onBindViewHolder(holder:MyCardStazione, i:int) void
+ getItemCount():int
MyAdapterTabellone
- context:Context
- stazione:Stazione
- datiTabellonePartenze:Partenze[]
- datiTabelloneArrivi:Arrivi[]
+ BINARIOSCONOSCIUTO:String
- pos:int
+ MyAdapterTabellone(c:Context, stazione:Stazione, datiTabelloneArrivi:Arrivi[], datiTabelloneePartenze:Partenze[], pc
+ onCreateViewHolder(parent:ViewGroup, viewType:int):MyCardTabellone
+ onBindViewHolder(holder:MyCardTabellone, i:int):void
+ getItemCount():int
CercaStazione
- myDbHelper:DataBaseHelper
- myData:String[]
- autoCom:AutoCompleteTextView
- stringaSoluzione:String
- idStazione:String
- valida:Boolean
- stazione:Stazione
- recyclerView:RecyclerView
- adapter:RecyclerView.Adapter
- stazioniController:StazioniController
onCreate(savedInstanceState:Bundle):void
+ getStazione()
MyCardStazione
~ nomeStazione:TextView
~ idStazione:TextView
+ MyCardStazione(itemView:View)
AboutUs
onCreate(savedInstanceState:Bundle):void
Profilo
- nome:EditText
- cognome:EditText
- dataN:Button
- pickerDate:DatePickerDialog
onCreate(savedInstanceState:Bundle):void
CercaViaggio
- myData:String[]
- autoCom1:AutoCompleteTextView
- autoCom2:AutoCompleteTextView
- data:Button
- ora:Button
- pickerDate:DatePickerDialog
- pickerTime:TimePickerDialog
- primaValida:Boolean
- secondaValida:Boolean
- stazioniController:StazioniController
onCreate(savedInstanceState:Bundle):void
+ getStazionePartenza():String
+ getStazioneArrivo():String
RisultatiCercaViaggio
- recyclerView:RecyclerView
- adapter:RecyclerView.adapter
- soluzioni:ArrayList<Soluzione>
- caricamento:ProgressBar
onCreate(savedInstanceState:Bundle):void
+ onClick(v:View, position:int):void
+ run():void
MyCardTabellone
~ treno:TextView
~ binario:TextView
~ ore:TextView
+ MyCardTabellone(itemView:View)
Preferiti
- tabLayout:tabLayout
- soluzioni:HashSet<Soluzione>
- recyclerView:RecyclerView
- adapter:RecyclerView.Adapter
- preferitiController:PreferitiController
onCreate(savedInstanceState:Bundle):void
+ onTabSelected(tab:TabLayout.Tab):void
+ onTabUnselected(tab:TabLayout.Tab):void
+ onTabReselected(tab:TabLayout.Tab):void
+ onClick(view:View, position:int):void
+ onLongClick(view:View, position:int):void
+ showStartDialog():void
+ onNavigationItemSelected(menuItem:MenuItem):boolean
MyAdapterPreferiti
- context:Context
- datiSoluzione:HashSet<Soluzione>
+ MyAdapterPreferiti(c:Context, datiSoluzione:HashSet<Soluzione>)
+ onCreateViewHolder(parent:ViewGroup, viewType:int):MyCardViaggio
+ onBindViewHolder(holder:MyCardViaggio, i:int):void
+ getItemCount():int
MyAdapterViaggio
- context:Context
- datiSoluzione:ArrayList<Soluzione>
- partenza:String
- arrivo:String
+ MyAdapterViaggio(c:Context, datiSoluzione:ArrayList<Soluzione>, partenz
+ onCreateViewHolder(parent:ViewGroup, viewType:int):MyCardViaggio
+ onBindViewHolder(holder:MyCardViaggio, i:int):void
+ showMultiSelection():int):void
+ getItemCount():int
Classe di ascolto per i click sulle card e sulle recycleview
<<interface>>
ClickListener
void onClick(view:View, position:int);
void onLongClick(view:View, position:int);
RecyclerOnTouchListener
- clickListener:ClickListener
- gestureDetector:GestureDetector
+ RecyclerOnTouchListener(context:Context, recycleView:RecyclerView, c
+ onSingleTapUp(e:MotionEvent):boolean
+ onLongPress(e:MotionEvent):void
+ onInterceptTouchEvent(rv:RecyclerView, e:MotionEvent):Boolean
+ onTouchEvent(rv:RecyclerView, e:MotionEvent):void
+ onRequestDisallowInterceptTouchEvent(disallowIntercept:Boolean):voi
Mappa
- tabLayout:TabLayout
- mappaTreni:ImageView mappaTreni;
- titlemappa:TextView
onCreate(savedInstanceState:Bundle):void
Social
- tabLayout:TabLayout
- webView:WebView
onCreate(savedInstanceState:Bundle):void
+ onClick(v:View):void
+ onTabSelected(tab:TabLayout.Tab):void
MyCardViaggio
~ oraViaggio:TextView
~ oraPartenza:TextView
~ oraArrivo:TextView
~ stazionePartenza:TextView
~ stazioneArrivo:TextView
~ numeroTreno:TextView
~ destinazioneTreno:TextView
~ classeTreno:TextView
~ binarioTrenoStazPart:TextView
~ statoTreno:TextView
~ preferito:ImageView
~ preferitoSi:ImageView
~ trattaOk:ImageView
~ trattaNonOk:ImageView
~ statusOk:ImageView
~ statusNonOk:ImageView
+ MyCardViaggio(itemView:View)
Classe per l'invio delle notifiche
PushNotification
- context:Context
- CHANNEL_1_ID:String
- NAME:String
- GROUP:String
- TITLE:String
+ init(c:Context):void
+ invia(testoNotifica:String):void
SplashActivity
- mProgressBar:ProgressBar
- mProgressStatus:Int
- myHandler:Handler
~ TAG:String
# onCreate(savedInstanceState:Bundle):void
+ run():void
+ scheduleJob():void
- initPreferiti():void
- firstRun():void
Scheduletore
- TAG:String
+ onStartJob(jobParameters:JobParameters):boolean
+ doBackground(Work(preferitiController:PreferitiController):void

DIAGRAMMA DELLE CLASSI DEL PACKAGE DOMAIN

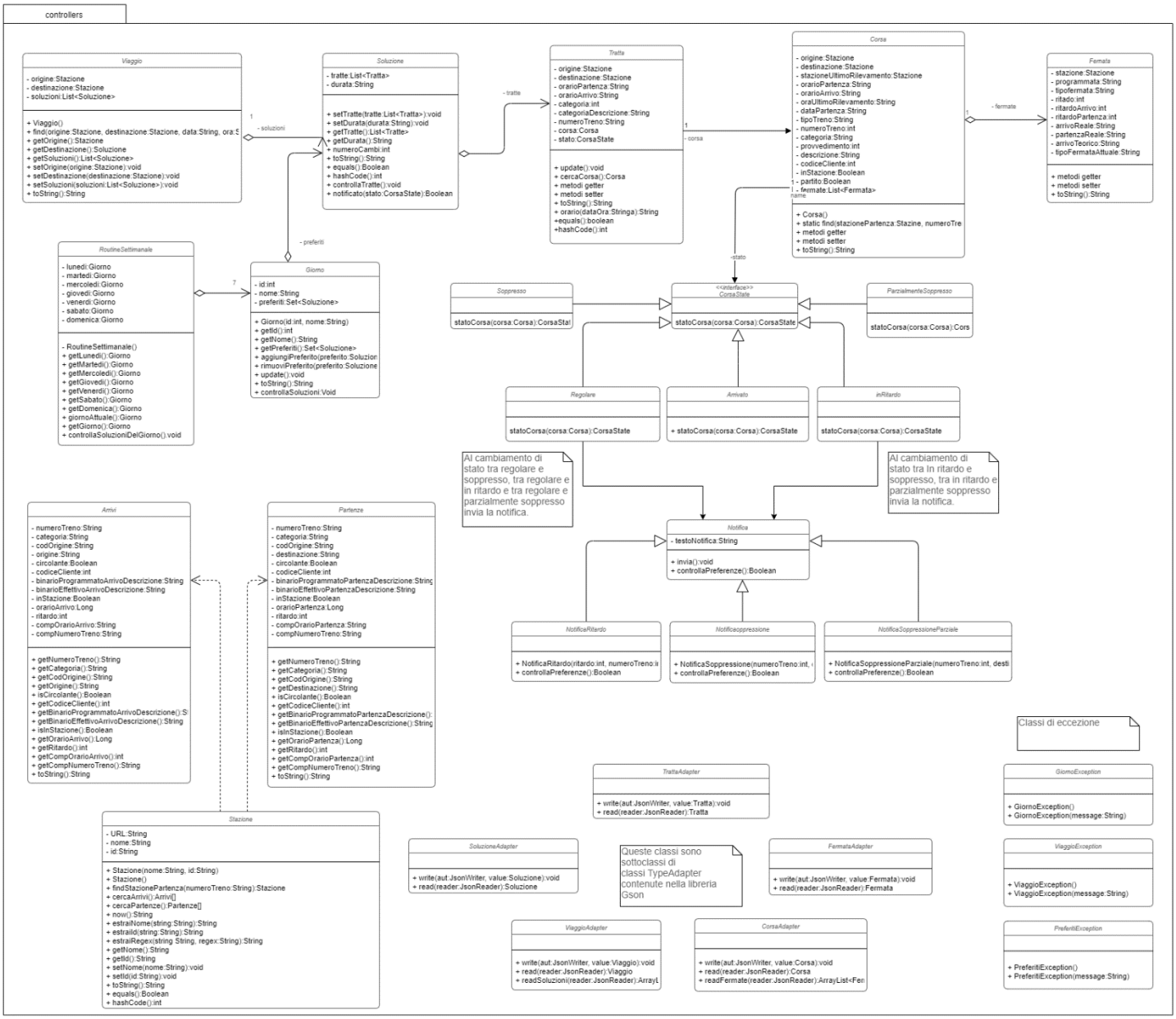


DIAGRAMMA DELLE CLASSI DEL PACKAGE DATA

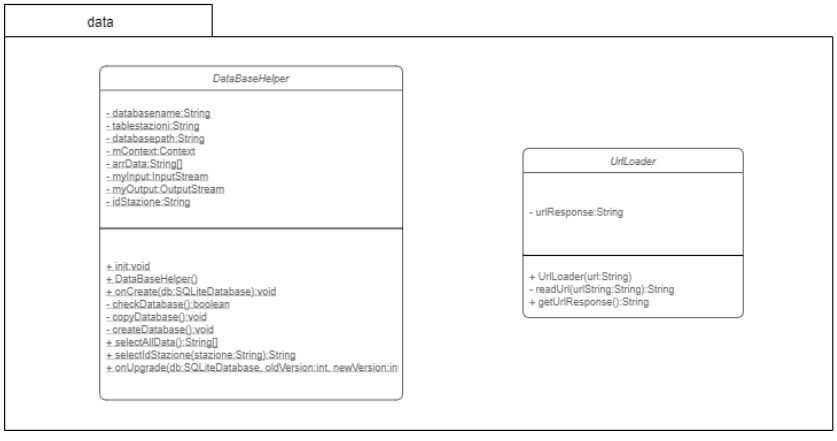
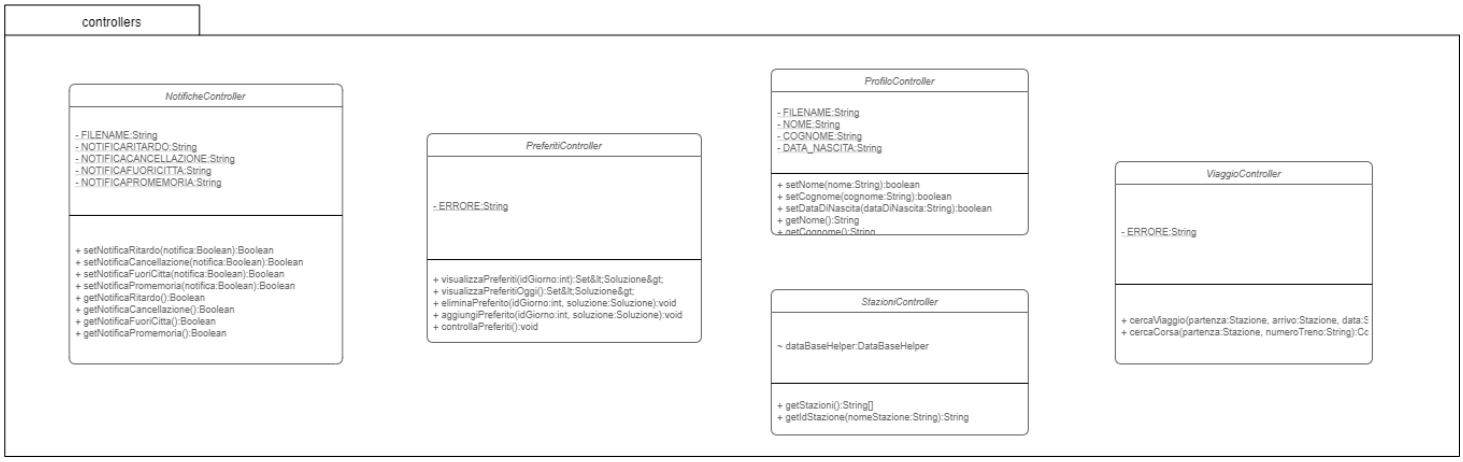
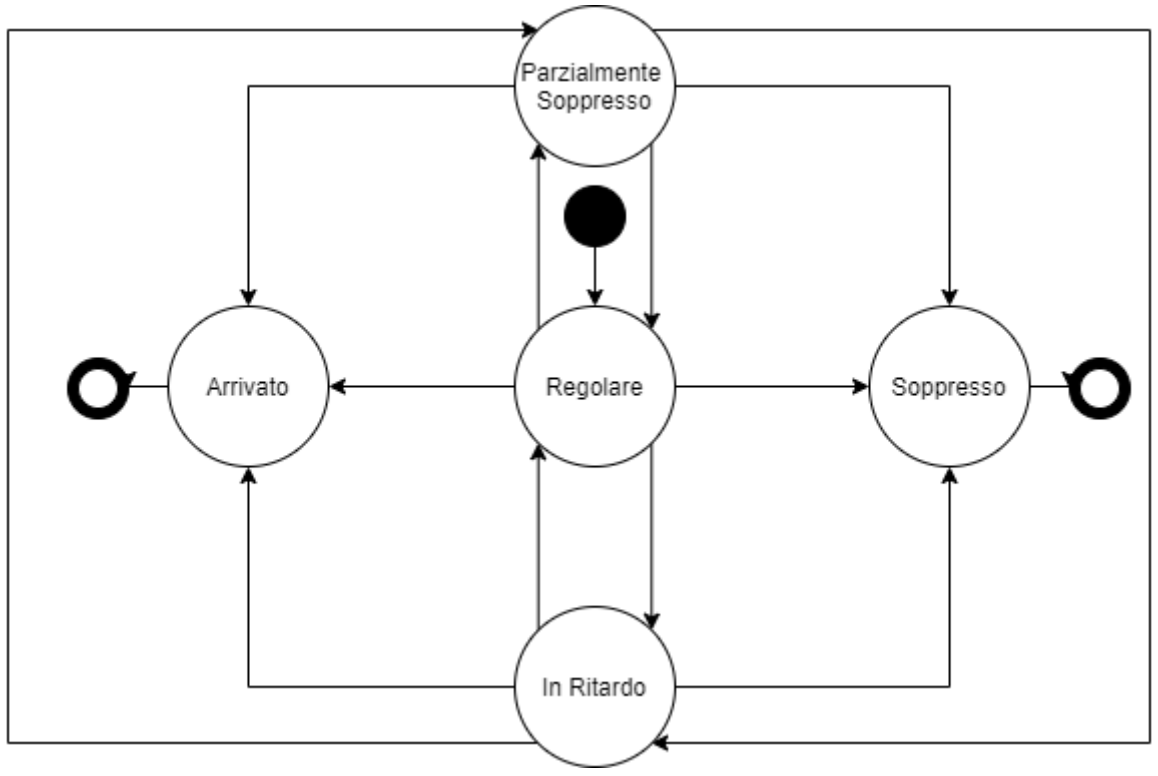


DIAGRAMMA DELLE CLASSI DEL PACKAGE CONTROLLER



3.7 Diagramma degli stati

E' un tipo di diagramma che rappresenta i possibili stati che può assumere un oggetto. Tra di essi vi possono essere dei collegamenti che vengono seguiti quando avviene un cambiamento di stato. Nel nostro caso questo diagramma torna molto utile per mostrare per esempio il funzionamento del cambiamento dello stato di una corsa.



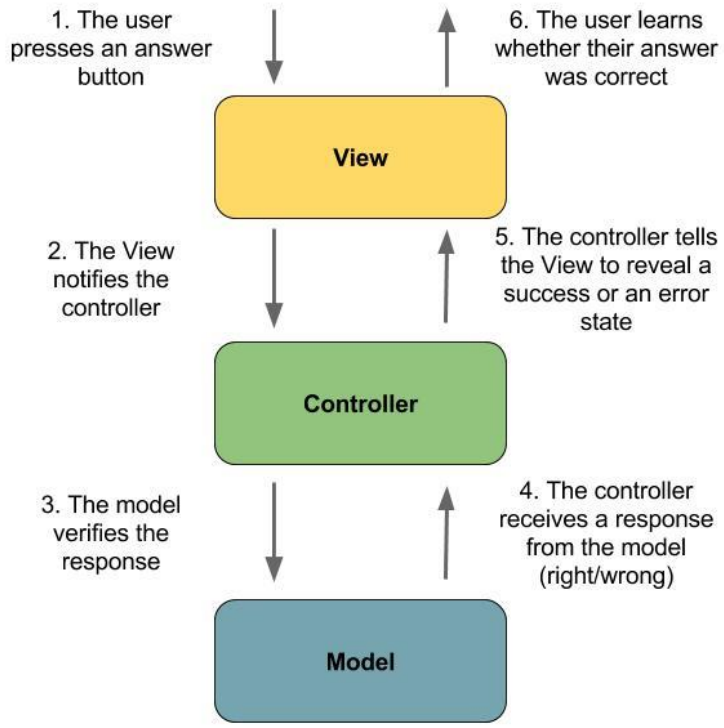
4. Design Pattern

4.1 Pattern architetturali

Per affrontare l'analisi dell'architettura logica, abbiamo tenuto conto di :
- Ogni layer/package è coeso rispetto alle responsabilità.
- Modularizzazione e il separation of concerns.
- Ci ha permesso una buona e corretta organizzazione e progettazione compreso sviluppo in team (con un processo di sviluppo iterativo e incrementale).
- Problemi legati all'architettura:
    - Alto accoppiamento/ poco modulare
    - Poco modulare
    - logica nella GUI
    - servizi tecnici legati alla GUI
    - Abbiamo usato i seguenti pattern GRASP: **CONTROLLER, INFORMATION EXPERT, CREATOR, HIGH COESION, LOW COUPLING.**

Inoltre abbiamo tenuto conto della gestione dei failover. Nonostante questo il nostro codice aveva alta dipendenza con le API viaggiatreno e quindi non vengono gestiti i failover.
Abbiamo altresì gestito le eccezioni tramite la strategia "error dialog".

4.1.1 Model View Controller



Descrizione

Model-view-controller (MVC, talvolta tradotto in italiano con la dicitura modello-vista-controllo), in informatica, è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business.

4.2 Java Best Practices

Durante tutta la fase di programmazione abbiamo tenuto conto delle Java best practices che ci hanno permesso di produrre un codice quanto più pulito e funzionale possibile. Queste pratiche le abbiamo applicate partendo dal libro Effective Java.

4.3 Design Principles

Nel corso della progettazione, abbiamo cercato di utilizzare il più possibile i design principles, analizzando i sintomi nel design tra cui: rigidità, fragilità, immobilità, viscosità
Nonostante siamo a conoscenza dei possibili problemi che presenta la nostra progettazione, soprattutto riguardo a possibili modifiche evolutive, per esempio l'aggiunta di altre API che cerchino altre soluzioni di viaggio per altri operatori di trasporto, abbiamo preferito tenere alta questa dipendenza per fornire all'utente un servizio utile nella vita reale piuttosto che creare dati inconsistenti costruiti appositamente per lo scopo del progetto.
I design principles che abbiamo ritenuto maggiormente utili sono i seguenti:
- **OPC, open closed principles**
- **Dependency inversion principles**

- **LSP liskov sobstitution principles**
- **Tension Between cohesion principles e package coupling principles**
- **Acyclic dependencies principles**
- **Stable dependencies principles**
- **DRY, don't repeat yourself**

4.4 Design Patterns (GoF)

Nel nostro progetto abbiamo utilizzato lo **state** per identificare i cambiamenti che può subire un treno nel corso di un servizio come ad esempio quando accumula ritardo, o quando un treno viene cancellato. Questo pattern permette inoltre l'invio delle notifiche al cambiamento di alcuni stati per esempio da regolare a soppresso.

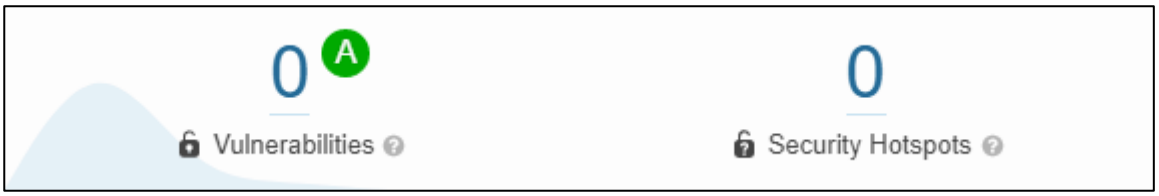
5. Strumenti utilizzati

5.1 Sonarqube

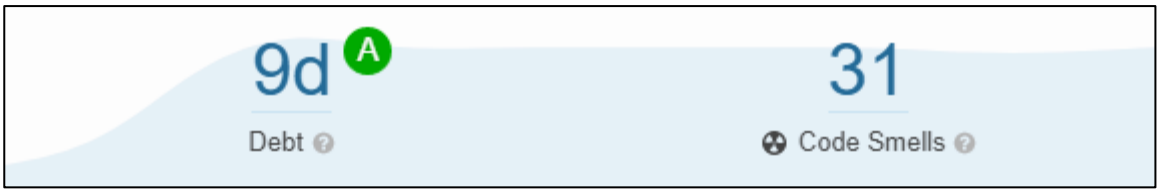
Durante lo sviluppo del progetto sono state condotte delle analisi con sonarqube, e, dopo alcune correzioni di bug, vulnerabilità e code smell, abbiamo ottenuto i seguenti risultati:

RELIABILITY

SECURITY



MAINTAINABILITY

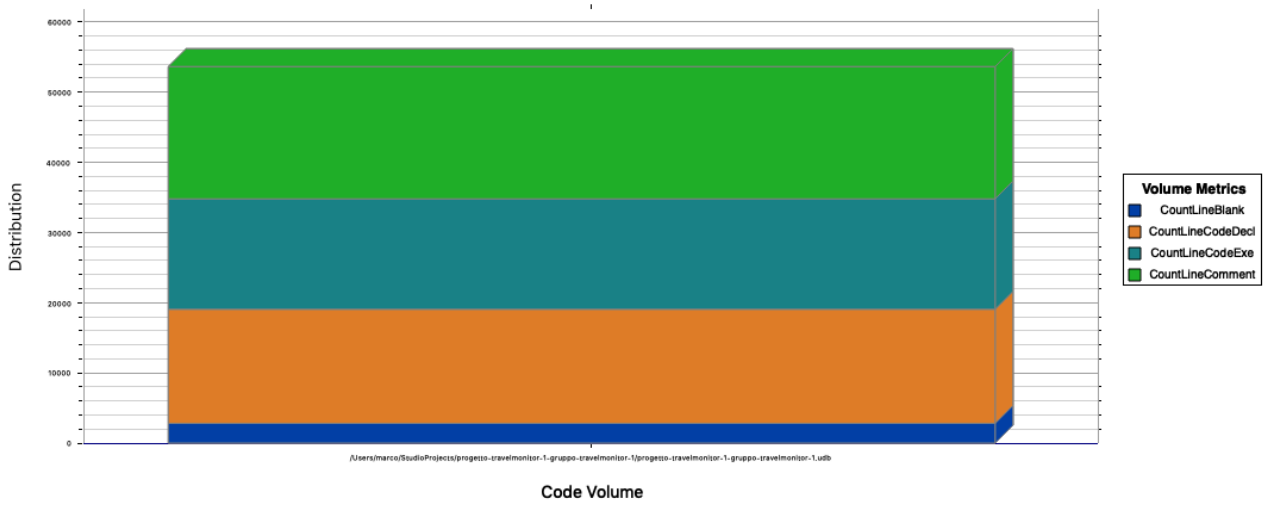


5.2 Understand

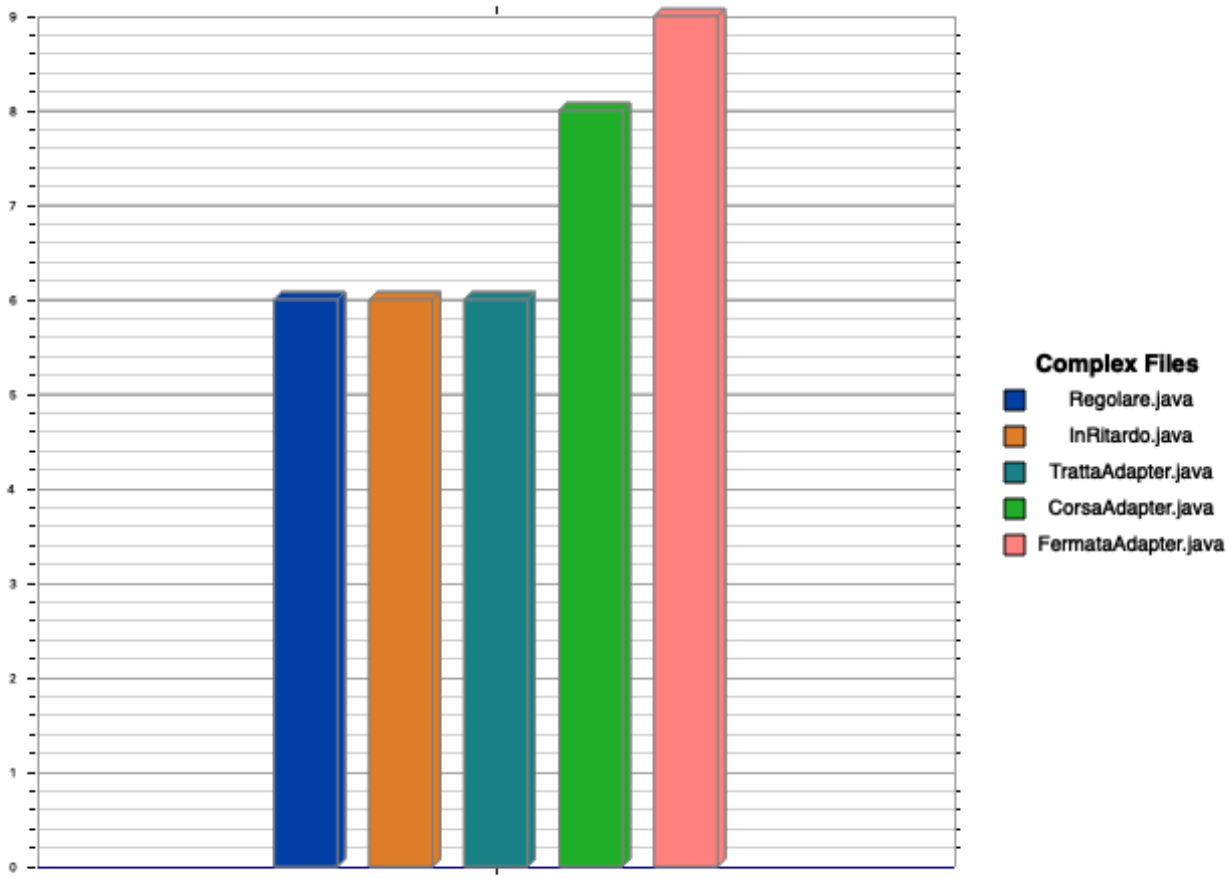
Abbiamo effettuato inoltre un'analisi con Understand per analizzare il codice prodotto ed individuare eventuali anti pattern.

Questi sono alcuni dei risultati che abbiamo ottenuto:

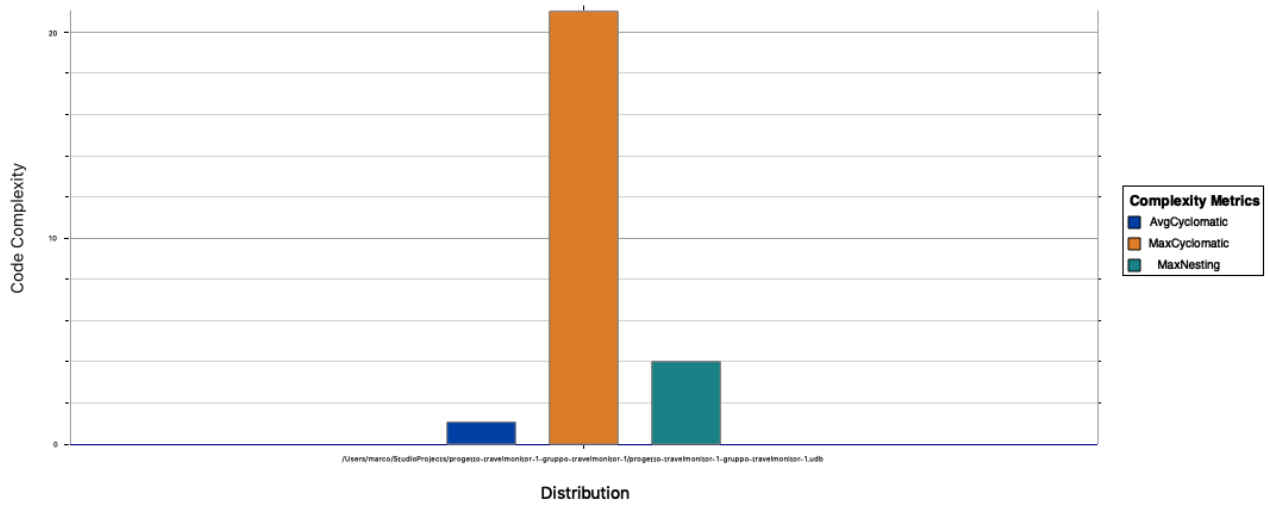
CODE VOLUME DISTRIBUTION



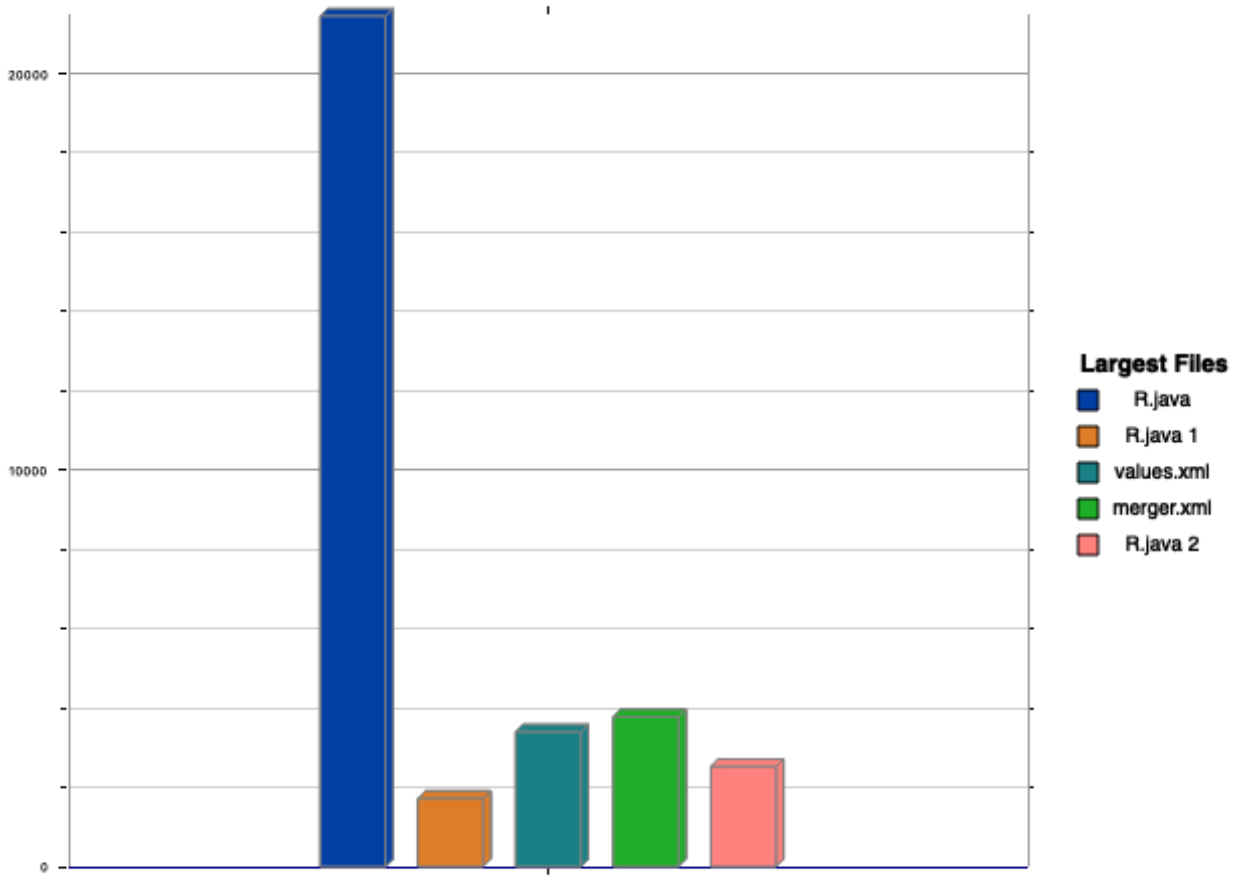
COMPLEX FILE



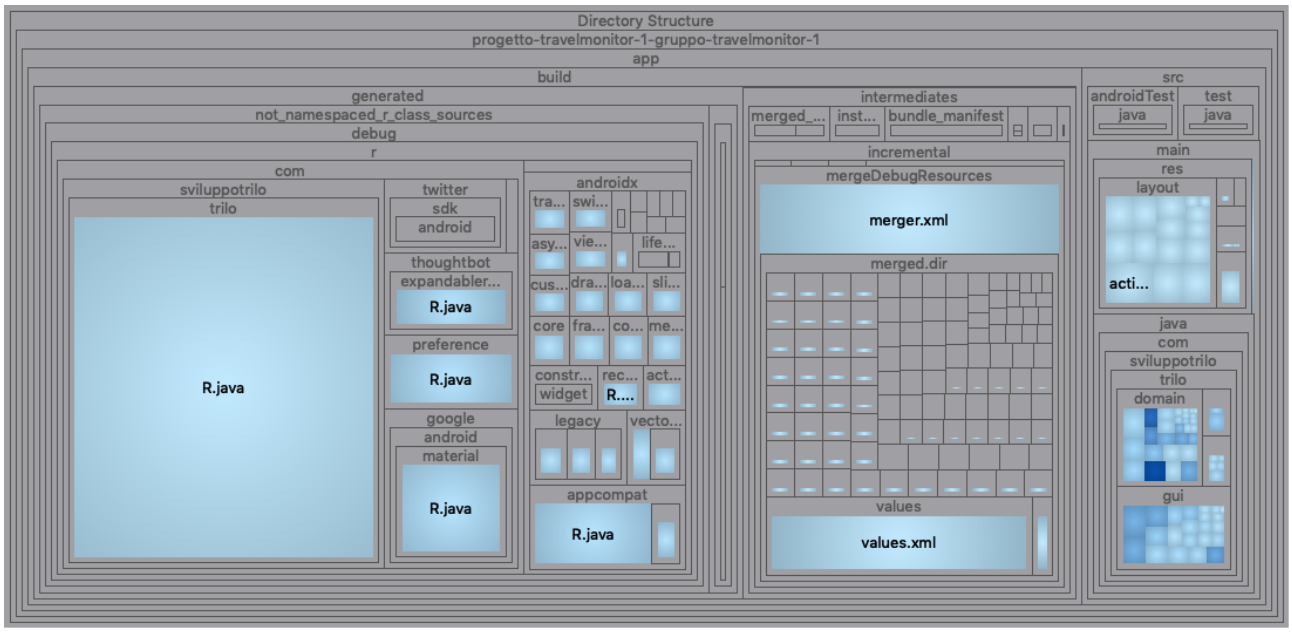
CYCLOMATIC COMPLEXITY



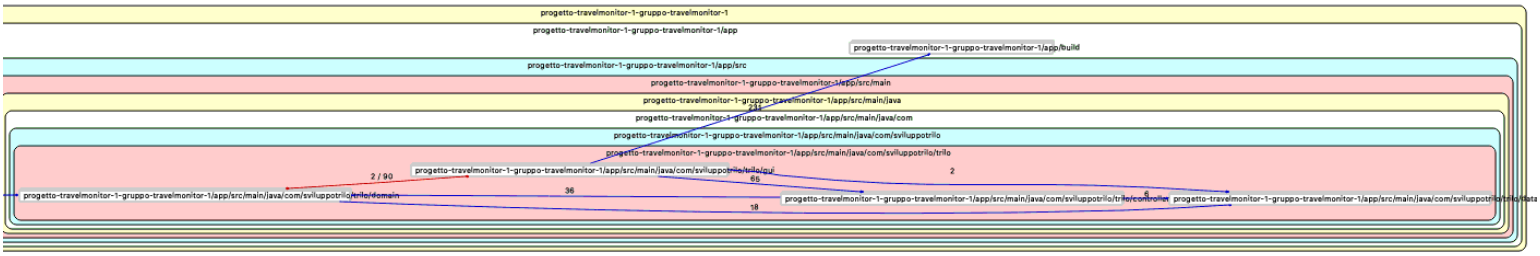
LARGEST FILES



STRUCTURE



ARCHITECTURAL BUTTERFLY DEPENDENCY



6. Problematiche non risolte

6.1 Obiettivi iniziali

All'inizio del progetto, si volevano implementare anche ulteriori funzioni come la notificazione in base alla posizione dell'utente, ma purtroppo, in corso di sviluppo questo non è stato possibile in quanto i tempi erano per noi ristretti. Molte funzionalità avrebbero richiesto un maggiore approfondimento da parte nostra, ma abbiamo preferito raggiungere quanto prima almeno gli obiettivi principali del progetto.

6.2 Problemi di inconsistenza delle API

All'inizio del progetto, si è deciso di intraprendere una strada di sviluppo quanto più realistica possibile, cioè volevamo che i dati generati e presenti all'interno del progetto fossero reali e non creati ad hoc da noi. Questo richiedeva quindi uno studio preliminare di API adatte al caso nostro. Inoltre, l'idea di base prevedeva la serializzazione dei dati in input dalle API, questo però fino a questa versione dell'app non siamo riusciti a formattare i dati come avremmo voluto. ES: Attributi String che avremmo voluto in formato di tipo Date. Con il passare dei giorni ci siamo poi resi conto dell'inconsistenza di alcuni dati forniti dalle API da noi scelte come attributi che non vengono mai modificati o stazioni con più chiavi (codici).

6.3 Problemi riscontrati

Uno dei principali scogli nello sviluppo di questa applicazione è stato l'utilizzo delle API, in quanto, nonostante fossero le migliori da noi utilizzabili, non sempre forniscono dati certi con ridondanze nei campi e spesso con dati non aggiornati. Questo non ci ha permesso di sviluppare tutte le funzionalità che avevamo pensato in fase di analisi, portandoci poi alla modifica dei diagrammi.

Utilizzo di GitHub
Abbiamo riscontrato anche alcune difficoltà nell'utilizzo di GitHub tramite android studio, che ci ha indotto ad un errore soprattutto quando, nel tentativo di fare una push sul branch master di una versione completa di progetto, è stata effettuata una copia del branch sul quale stavamo lavorando.

Utilizzo di SonarQube
Sonarqube, ci ha creato problemi che non siamo riusciti a risolvere nella sezione di Coverage Test. Infatti, non siamo riusciti ad implementare (In Android Studio) questi test di progetto per migliorare l'indicatore di sonarqube. In ogni caso sono stati

utilizzati diversi metodi di test dell'applicazione tra cui, quelli a linea di comando e quelli tramite emulatore o su dispositivo.
Inoltre, nonostante fossero presenti dei metodi di test nelle apposite classi, non veniva modificato il campo coverage code su sonarqube. Per risolvere questi problemi abbiamo provato anche ad utilizzare la libreria JaCoCo, consigliata dalla documentazione sonarqube.

7. Conclusioni

Come aprire ed utilizzare Trilo (README)

Trilo
Trilo è un'app per Android che ti permette di avere sotto controllo i tuoi viaggi in treno. Cerca un viaggio, clicca il cuore e salvalo nei giorni che vuoi!

Instruzioni
Per installare correttamente l'app segui le istruzioni sotto riportate. Le istruzioni ti permetteranno di importare ed effettuare la build di Trilo e successivamente avviare l'app su un dispositivo android.

Prerequisiti
Cosa installare per importare correttamente il progetto:

1) Scaricare git
2) Scaricare Android Studio
Installazione
Per installare Trilo, dopo aver scaricato ed installato git ed Android Studio:

All'apertura di android studio, effettuare la "clone" del progetto
Checkout out project from Version Control --> Git --> <Credenziali>
--> <link github> --> Clone
Sincronizzare il progetto con Gradle
File --> Sync project with Gradle Files
Effettuare la build del progetto
Build --> Make Project
4a) Se si vuole eseguire dall'emulatore integrato, premere il pulsante verde play e seguire la procedura per installare l'emulatore di android

Run --> Run 'app'
4b) Se si vuole esportare il file apk per eseguirlo su un dispositivo Android

Build --> Build Bundle(s) / APK(s) --> Build APK(s)
Ora il file apk è pronto, basterà copiarlo sullo smartphone android e avviare la procedura d'installazione
Built With
Gradle - Dependency Management