```
{"total_rows":10,"offset":0,"rows":[
{"id":"10027a",  "key":"10027a",   "value":{"rev":"30-acad43f766c745f3a190d9fb10fd6829"}},
{"id":"10028b",  "key":"10028b",   "value":{"rev":"20-982999a575a228e06c841bf8b7569eb5"}},
{"id":"10029c",  "key":"10029c",   "value":{"rev":"10-6c479df81f6d5ad51c672c68b9c6a5d1"}},
{"id":"10030d",  "key":"10030d",   "value":{"rev":"18-850aaeca9a0d8be9b3cf80b24a3b65f2"}},
{"id":"10031e",  "key":"10031e",   "value":{"rev":"25-7e6646be51203aeedd3cb949598b0f3f"}},
{"id":"10032f",  "key":"10032f",   "value":{"rev":"18-6957a7a9a717c0092e029dbdba61ff62"}},
{"id":"10033g",  "key":"10033g",   "value":{"rev":"18-b5a967907fffd0510ac015f923565107"}},
{"id":"10034h",  "key":"10034h",   "value":{"rev":"21-a3446aed3bdf83c41db8db0b22840539"}},
{"id":"10035i",  "key":"10035i",   "value":{"rev":"15-d49bd8cff1a397ce9c5bd71b705bac68"}},
{"id":"10036j",  "key":"10036j",   "value":{"rev":"19-a286a32e7bc8a55ac2745595fe447394"}}
]}
```



```
{
 "_id":                "10029c",
 "_rev":               "10-6c479df81f6d5ad51c672c68b9c6a5d1",
 "category":           "Pop, Rock",
 "date":               "12/05/2015",

 "description":        "Perfetto è stato pubblicato a più di due anni di distanza dal precedente album
                       in studio dell'artista, Noi. Lo stesso Ramazzotti afferma che vi ha lavorato per due
                       anni e che questo è un album più curato rispetto ai precedenti, infatti presenta
                       collaborazioni con importanti parolieri, tra cui Mogol, Francesco Bianconi,
                       Federico Zampaglione e Pacifico.",

 "insertDate":         "29/06/2017",
 "musicalInstruments": "Piano, Chitarra, Batteria, Basso",
 "performer":          "Eros Ramazzotti",
 "performers":         "Claudio Guidetti, Gary Kemp, Pacifico",
 "price":              19.99,
 "title":              "Perfetto",

 "tracks":             "1 Alla fine del mondo - 3.55
                       2 Il tempo non sente ragione - 4.04
                       3 Perfetto - 3.51
                       4 Sbandando - 3.56
                       5 Sogno N.3 - 3.53
                       6 Rosa nata ieri - 3.50
                       7 Vivi e vai - 3.52
                       8 Un'altra estate - 4.07
                       9 L'amore è un modo di vivere - 3.55
                       10 Il viaggio - 3.44
                       11 Tu gelosia - 3.50
                       12 Sei un pensiero speciale - 3.47
                       13 Buon Natale (se vuoi) - 4.04
                       14 Tra vent'anni - 3.39
                       15 Alla fine del mondo - 2.28",

 "type":               "CD",
 "_attachments": {
                       "10029c.jpg": {
                                "content_type": "image/jpeg",
                                "revpos": 9,
                                "digest": "md5-Wg5bPOAf19fohShhlKduAQ==",
                                "length": 436453,
                                "stub": true
                       },
                       "10029c_small.jpg": {
                                "content_type": "image/jpeg",
                                "revpos": 8,
                                "digest": "md5-eHCGxlUQ6hF1vTCITTJxww==",
                                "length": 20511,
                                "stub": true
                       }
        }
}
```

NoSQL DB: UTENTI


```
{"total_rows":2,"offset":0,"rows":[

{"id":"admin",                          "key":"admin",
            "value":{"rev":"4-4afbdc8869c749f67425ede2cad5b804"}},

{"id":"vlad.bragoi@gmail.com",    "key":"vlad.bragoi@gmail.com",
            "value":{"rev":"4-6755c67c45cbd93a877cb380399bb288"}}
]}
```



```
{
  "_id":              "vlad.bragoi@gmail.com",
  "_rev":             "5-08e998369f0bc555a5e2245a0e3ac310",
  "codice_fiscale":   "BRGVDS95B04Z140N",
  "password":         "prodottimusicali",
  "nome":             "Vladislav",
  "cognome":          "Bragoi",
  "indirizzo":        "via Torino, 9 - Bovolone",
  "telefono":         3925093287,
  "carta":            "email@paypal.com",
  "iban":             null,
  "paypal":           null,
  "bonus":            false
}
```


NoSQL DB: VENDITE


```
{"total_rows":3,"offset":0,"rows":[

{"id":"Thu Jul 13 15:47:12 GMT+02:00 2017","key":"Thu Jul 13 15:47:12 GMT+02:00 2017",
            "value":{"rev":"2-726b57dff9030b61850bbf4caddd4ec5"}},

{"id":"Thu Jul 13 15:48:56 GMT+02:00 2017","key":"Thu Jul 13 15:48:56 GMT+02:00 2017",
            "value":{"rev":"1-698b53e3c74363249cabc69c1daff750"}},

{"id":"Thu Jul 13 15:50:20 GMT+02:00 2017","key":"Thu Jul 13 15:50:20 GMT+02:00 2017",
            "value":{"rev":"1-7999cffd72e306b9feff8358d4de8d35"}}
]}
```



```
{
  "_id":              "Thu Jul 13 15:50:20 GMT+02:00 2017",
  "_rev":             "1-7999cffd72e306b9feff8358d4de8d35",
  "corriere":         "Bartolini",
  "articoli":         [
                        "10029c : PERFETTO - Eros Ramazzotti",
                        "10032f : G I R L - Pharrell Williams",
                        "10034h : ÷ - Ed Sheeran"
                      ],
  "cliente":          "Vladislav Bragoi",
  "prezzo":           79.89,
  "indirizzo":        "via Torino, 9 - Bovolone"
}
```

Vladislav Bragoi VR397344
Andrea Faggion VR401550

2Emme

Università di Verona
Dipartimento di Informatica

A.A. 2016 - 17

Specifica

Si vuole progettare un sistema informativo per gestire le informazioni relative alla gestione di un negozio virtuale di CD e DVD musicali (vende solo via web). Il negozio mette in vendita CD di diversi generi: jazz, rock, classica, latin, folk, world-music, e così via.

Per ogni CD o DVD il sistema memorizza: un codice univoco, il titolo, i titoli di tutti i pezzi contenuti, eventuali fotografie della copertina, il prezzo, la data dalla quale è presente sul sito web del negozio, il musicista/band titolare, una descrizione, il genere del CD o DVD, i musicisti che vi suonano, con il dettaglio degli strumenti musicali usati. Per ogni musicista il sistema registra il nome d'arte, il genere principale, l'anno di nascita, se noto, gli strumenti che suona.

Sul sito web del negozio è illustrato il catalogo dei prodotti in vendita. Cliccando sul nome del prodotto, appare una finestra con i dettagli del prodotto stesso.

I clienti possono acquistare on-line selezionando gli oggetti da mettere in un "carrello della spesa" virtuale. Deve essere possibile visualizzare il contenuto del carrello, modificare il contenuto del carrello, togliendo alcuni articoli. Al termine dell'acquisto va gestito il pagamento, che può avvenire con diverse modalità. Il sistema supporta differenti ricerche: per genere, per titolare del CD o DVD, per musicista partecipante, per prezzo. Coerentemente, differenti modalità di visualizzazione, sono altresì supportate.

Ogni vendita viene registrata indicando il cliente che ha acquistato, i prodotti acquistati, il prezzo complessivo, la data di acquisto, l'ora, l'indirizzo IP del PC da cui è stato effettuato l'acquisto, la modalità di pagamento (bonifico, carta di credito, paypal) e la modalità di consegna (corriere, posta, …).
Per ogni cliente il sistema registra: il suo codice fiscale, il nome utente (univoco) con cui si è registrato, la sua password, il nome, il cognome, la città di residenza, il numero di telefono ed eventualmente il numero di cellulare. Per i clienti autenticati, il sistema propone pagine specializzate che mostrano suggerimenti basati sul genere dei precedenti prodotti acquistati. Se il cliente ha fatto già 3 acquisti superiori ai 250 euro l'uno entro l'anno, il sistema gli propone sconti e consegna senza spese di spedizione.

Il personale autorizzato del negozio può inserire tutti i dati dei CD e DVD in vendita. Il personale inserisce anche il numero di pezzi a magazzino. Il sistema tiene aggiornato il numero dei pezzi a magazzino durante la vendita e avvisa il personale del negozio quando un articolo (CD o DVD) scende sotto i 2 pezzi presenti in magazzino.

INTRODUZIONE

Il prototipo è stato progettato e sviluppato in ambiente Android, pertanto l'utilizzatore finale si potrà eseguirlo esclusivamente su un dispositivo mobile (tablet inclusi).

Il prototipo si limita alla sola implementazione della parte "cliente", in particolare:

- autenticazione
- registrazione
- visualizzazione dei prodotti e relativi dettagli
- ricerca dei prodotti e filtri di visualizzazione
- aggiunta (e rimozione) dei prodotti al carrello
- acquisto e relative fasi di scelta della spedizione, metodo di pagamento ecc.

L'applicazione inoltre prevede visualizzazioni diverse a seconda della classe di appartenenza del cliente. Se questo infatti è abilitato dal fornitore ad avere diritto di ricevere un bonus (quali riduzioni di prezzo su ciascun prodotto e consegna gratuita), gli verrà presentata un'interfaccia adeguata per la visualizzazione del suo premio.

Assumiamo inoltre che la figura del fornitore (non implementata ma simulata in fase di testing utilizzando un'interfaccia grafica da web browser) abbia diretto accesso alla base di dati contenente i prodotti e gli acquisti effettuati dai clienti, in modo tale da poterne monitorare i cambiamenti. Tale interfaccia è accessibile da qualsiasi dispositivo collegato alla stessa rete del computer su cui abbiamo installato il database con le seguenti credenziali:

login: **ingegneria**
password: **prodottimusicali**

e permette di aggiungere nuovi prodotti, modificare, eliminare i prodotti già presenti nel database, visualizzare gli acquisti effettuati dai clienti, e gestire gli utenti e i rispettivi account.

Ricordiamo inoltre che l'applicazione, per le semplici operazioni di consultazione del catalogo, è utilizzabile anche senza essere autenticati. Inoltre la sessione "carrello" è volatile, perciò se un cliente vorrà effettuare un acquisto dovrà concluderlo durante la sessione attiva, visto che i prodotti aggiunti al carrello resteranno memorizzati soltanto fino alla chiusura dell'applicazione.

PATTERN - **MVC**

Abbiamo utilizzato questo design pattern per gestire la separazione dei compiti fra i componenti software implementati.

Le classi che si occupano dei dati sono stati racchiusi in un package denominato “Model”. Al loro interno sono inclusi tutti metodi che si interfacciano con il database, i metodi che permettono di sincronizzarsi con esso e aggiornarlo, e i metodi che permettono di scaricare/caricare i singoli documenti. Tali metodi sono accessibili e controllabili dalle classi “Controller”, situate nel rispettivo package, che consentono di gestire il flusso dati da e verso la base di dati e la loro visualizzazione all’utente.
Inoltre nel package “View” sono incluse tutte le classi che permettono l’interazione con l’utente (specifichiamo che oltre alle classi, la separazione della parte View dalla parte Controller la garantiscono maggiormente tutti i file di layout situati nella cartella /res del progetto. La suddivisione del layout, scritto in linguaggio XML, dal resto del codice Java, è una delle caratteristiche peculiari del framework Android, che permette all’interfaccia di adattarsi a qualsiasi tipo e risoluzione di schermo su cui l’applicazione viene installata).

Abbiamo inoltre voluto tenere separate alcune parti perché hanno un funzionamento ibrido rispetto a tale pattern.

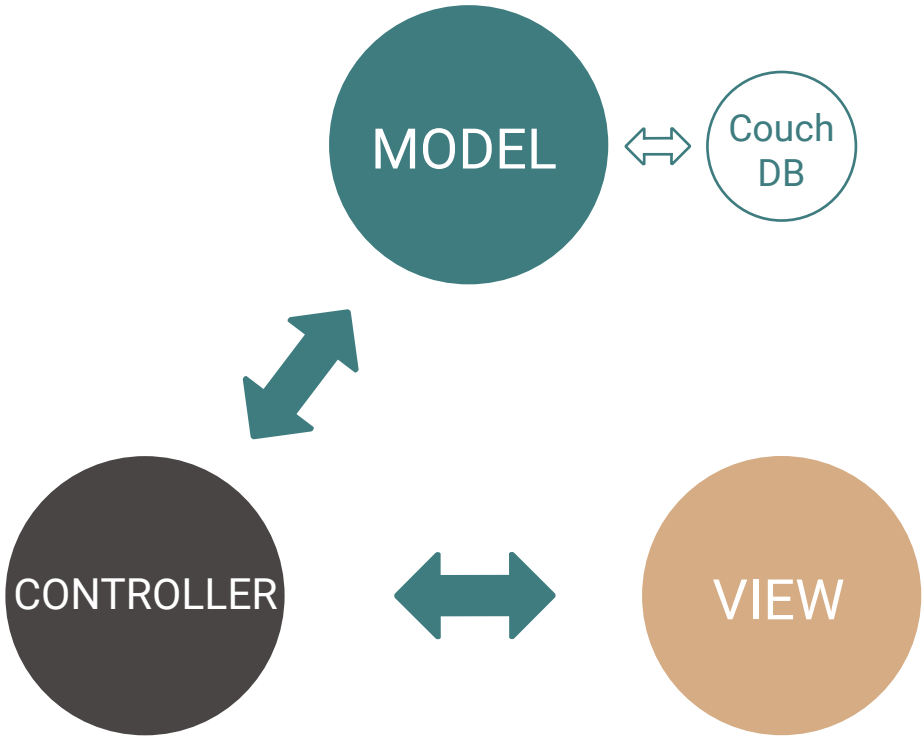
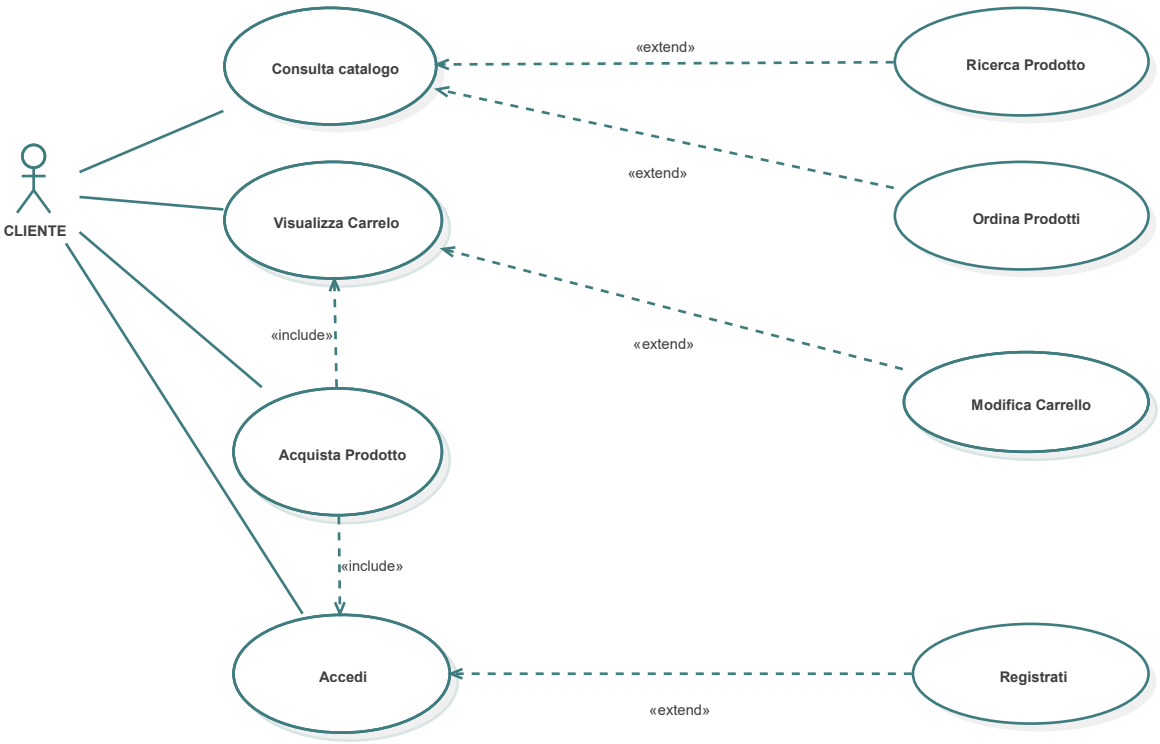


DIAGRAMMA **CASI D'USO**



CASO D'USO - **CONSULTA CATALOGO**

ID: UC001
Attori: Utente / Sistema
Pre-condizioni: Aprire l'applicazione
Sequenza degli eventi: 1 - Visualizzazione di un insieme di prodotti 1.1 - Seleziona il prodotto 1.2 - Seleziona un filtro di visualizzazione 1.3 - Ricerca un prodotto 2 - Visualizza dettagli prodotto
Post-condizioni: Nessuna

CASO D'USO - **LOGIN**

ID: UC002/01
Attori: Utente / Sistema
Pre-condizioni: Selezionare la voce Login | Registrati o effettuare un acquisto
Sequenza degli eventi: 1 - Inserimento username e password 2 - Il sistema verifica le credenziali 2.1 - Credenziali corrette 2.1.1 - Accesso avvenuto 2.2 - Credenziali errate 2.2.1 - Reinserimento credenziali
Post-condizioni: Accesso a funzionalità utente

CASO D'USO - **REGISTRATI**

ID: UC002/02
Attori: **Utente / Sistema**
Pre-condizioni: **Selezionare la voce Login | Registrati o effettuare un acquisto**
Sequenza degli eventi: 1 - **Compilazione di tutti i campi richiesti** 2 - **Il sistema verifica i parametri** 2.1 - **Dati validi** 2.1.1 - **Registrazione avvenuta con successo. Login automatico** 2.2 - **Dati mancanti o errati** 2.2.1 - **Riprova**
Post-condizioni: **Login effettuato**

CASO D'USO - **VISUALIZZA CARRELLO**

ID: UC003
Attori: **Utente / Sistema**
Pre-condizioni: **Selezionare la voce Carrello**
Sequenza degli eventi: 1 - **L'utente visiona il carello** 2.1 - **Può eventualmente cancellare un prodotto dal carrello**
Post-condizioni: **Carrello aggiornato**

CASO D'USO - **ACQUISTA PRODOTTO**

ID: UC004
Attori: Utente / Sistema
Pre-condizioni: Premere il pulsante "Acquista" all'interno del carrello
Sequenza degli eventi: 1 - Se non autenticato, effettuare l'accesso 2 - Selezionare la modalità di spedizione 3 - Selezionare / confermare modalità di pagamento 3.1 - Modificare modalità di pagamento 4 - Visualizzare riepilogo 4.1 - Accettare e generare ordine 4.2 - Annullare ordine
Post-condizioni: Il sistema registra l'ordine

DIAGRAMMI DELLE ATTIVITA’

Rappresentiamo di seguito i diagrammi che definiscono le attività che il cliente può svolgere per le funzionalità principali implementate.

DIAGRAMMA DELLE ATTIVITA’:
ACCEDI + REGISTRATI

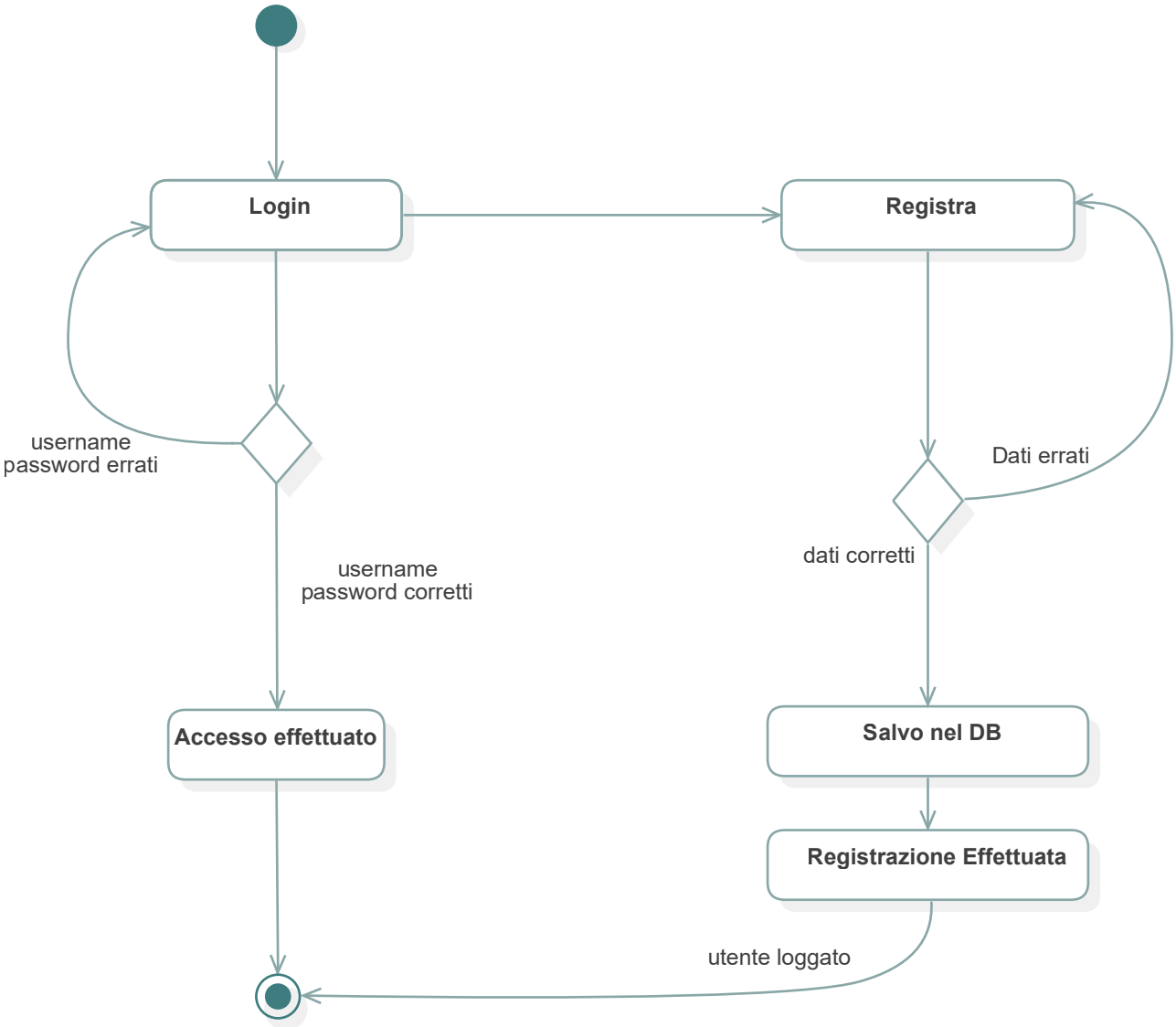


DIAGRAMMA DELLE ATTIVITA':

ACQUISTA PRODOTTO

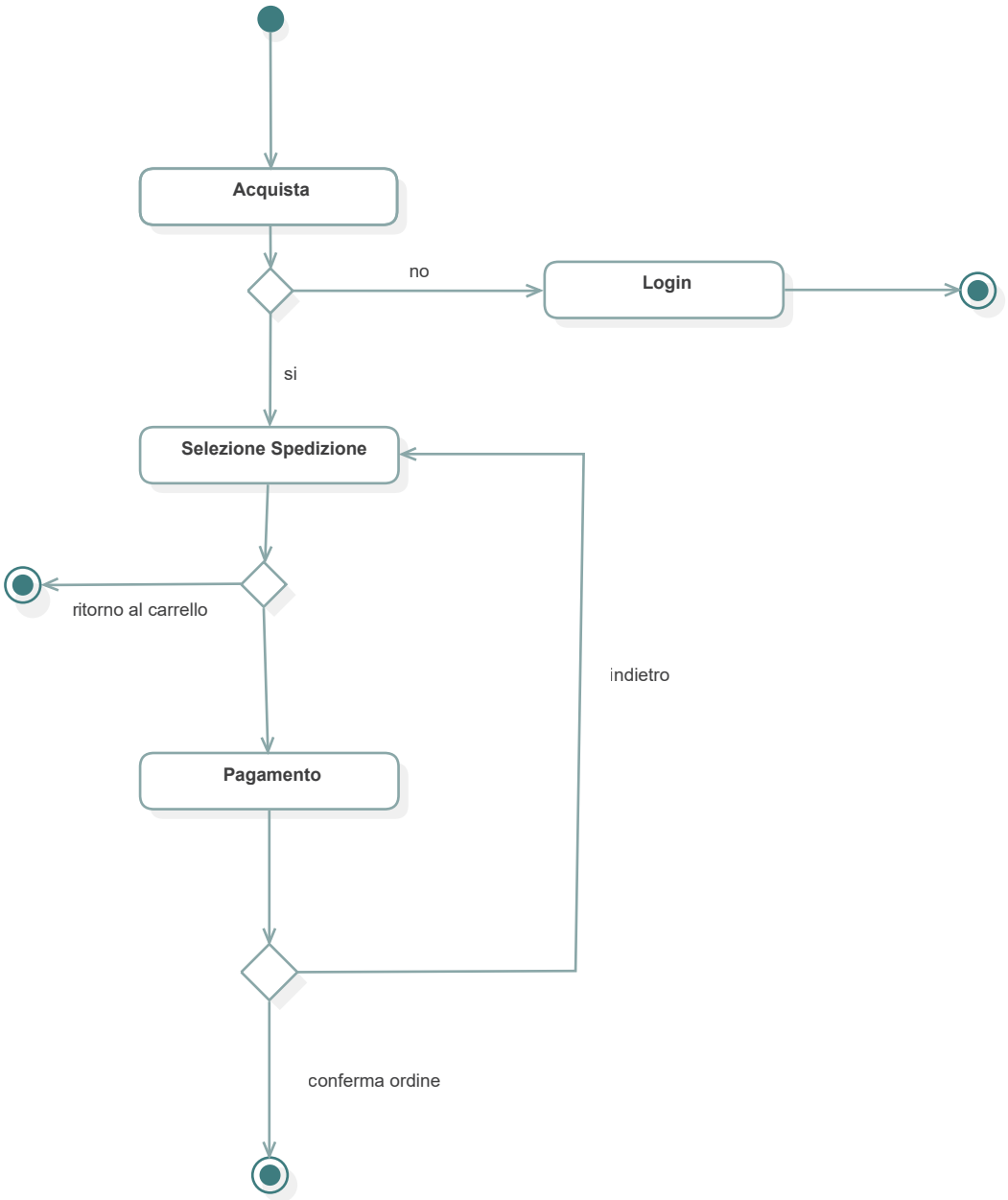
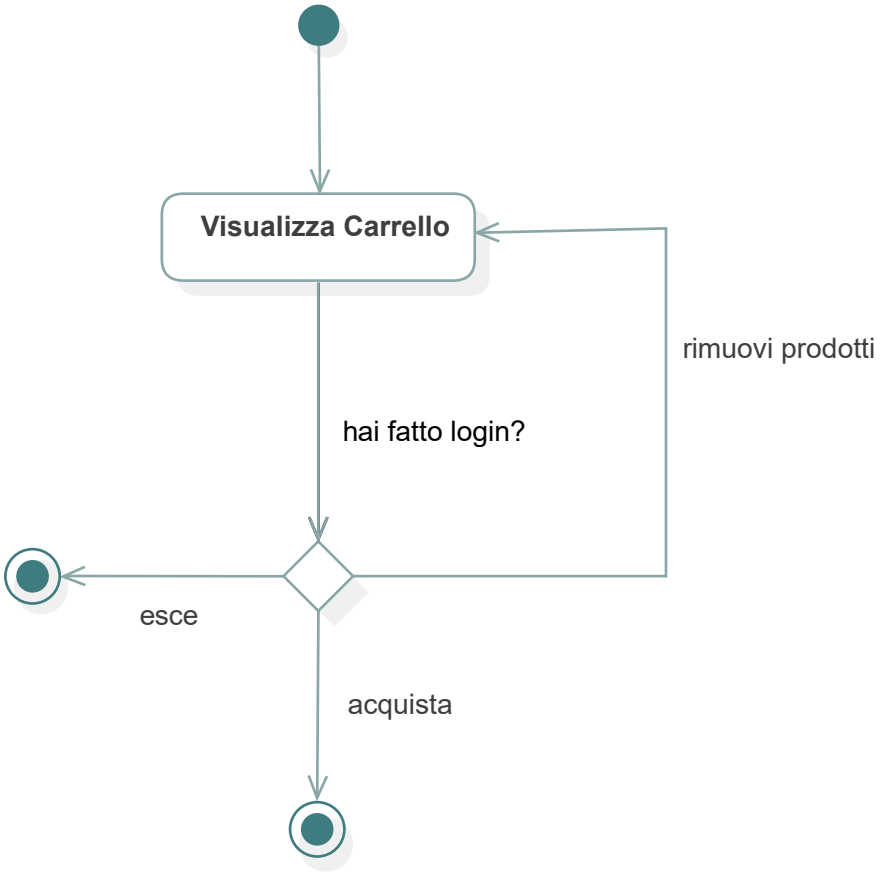


DIAGRAMMA DELLE ATTIVITA':

VISUALIZZA / MODIFICA CARRELLO



DATABASE NoSQL - **COUCHDB**

Per il salvataggio e la gestione dei dati abbiamo utilizzato un database di tipo non-relazionale: CouchDB. Tale sistema, congiuntamente ad una libreria dedicata per Android (***Couchbase***), che ci ha permesso di interfacciarci con il database server installato su pc in locale, ci ha fornito un supporto utile e semplice per gestire ogni tipo di informazione che fosse necessaria al funzionamento dell'applicazione.

CouchDB utilizza un sistema di salvataggio basato su file JSON. Ogni informazione nel database è memorizzata sotto forma di Documento JSON, e ogni documento è unico al suo interno, identificato da un "_id" univoco.
Abbiamo considerato l'opzione di utilizzare 3 database differenti, uno per tipologia di informazione da memorizzare (nonostante i JSON siano dinamici e permettano di memorizzare qualsiasi tipo di oggetto, o collezione di oggetti) e la scelta si è rivelata vincente. Le tre tipologia di informazioni memorizzate sono:

- prodotti (CD/DVD) e relative informazioni;
- utenti registrati;
- acquisti effettuati.

Un esempio di struttura del database e dei file JSON memorizzati al suo interno vengono riportati di seguito.

_id	_rev	corriere	articoli	cliente	prezzo	indirizzo
Thu Jul 13 15:50:20 GMT+02:00 2017	1-7999cf-fd72e306b9fef-f8358d4de8d35	Bartolini	10029c : PERFE... 10032f : G I R L... 10034h : ÷ - Ed...	Vladislav Bragoi	79.89	via Torino, 9 - Bovolone
Fri Jul 14 15:21:52 GMT+02:00 2017	1-241a66540fc-be81af2582080c39d90f3	Gls	10032f : G I R L... 10028b: NEWS...	Andrea Faggion	46.90	via Chiavica 38 - Roveredo di Guà


```
{
  "_id": "Thu Jul 13 15:50:20 GMT+02:00 2017",
  "_rev": "1-7999cffd72e306b9feff8358d4de8d35",
  "corriere": "Bartolini",
  "articoli": [ "10029c : PERFETTO - Eros Ramazzotti",
                "10032f : G I R L - Pharrell Williams",
                "10034h : ÷ - Ed Sheeran"],
  "cliente": "Vladislav Bragoi",
  "prezzo": 79.89,
  "indirizzo": "via Torino, 9 - Bovolone"
}
```

```
{
  "_id": "Fri Jul 14 15:21:52 GMT+02:00 2017",
  "_rev": "1-241a66540fcbe81af2582080c39d90f3",
  "corriere": "Gls",
  "articoli": [ "10032f : G I R L - Pharrell Williams",
                "10028b : NEWS OF THE WORLD - Freddy Mercury"],

  "cliente": "Andrea Faggion",
  "prezzo": 46.90,
  "indirizzo": "via Chiavica 38 - Roveredo di Guà"
}
```

DIAGRAMMI DI SEQUENZA

Rappresentiamo con dei diagrammi di sequenza il flusso temporale di esecuzione dei principali scenari che l'applicazione presenta.

DIAGRAMMA DI SEQUENZA

ACCEDI + REGISTRATI

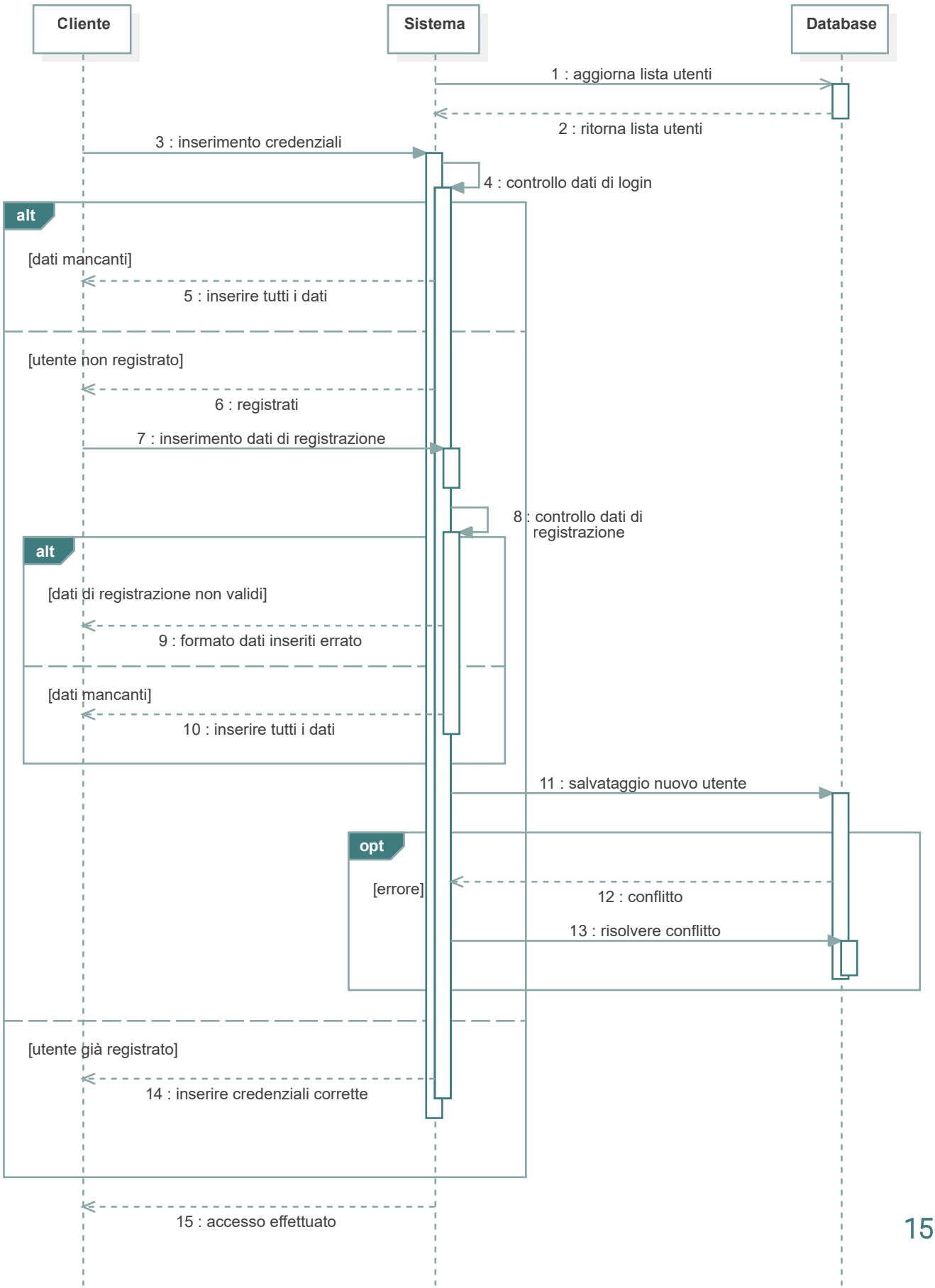


DIAGRAMMA DI SEQUENZA

RICERCA PRODOTTO

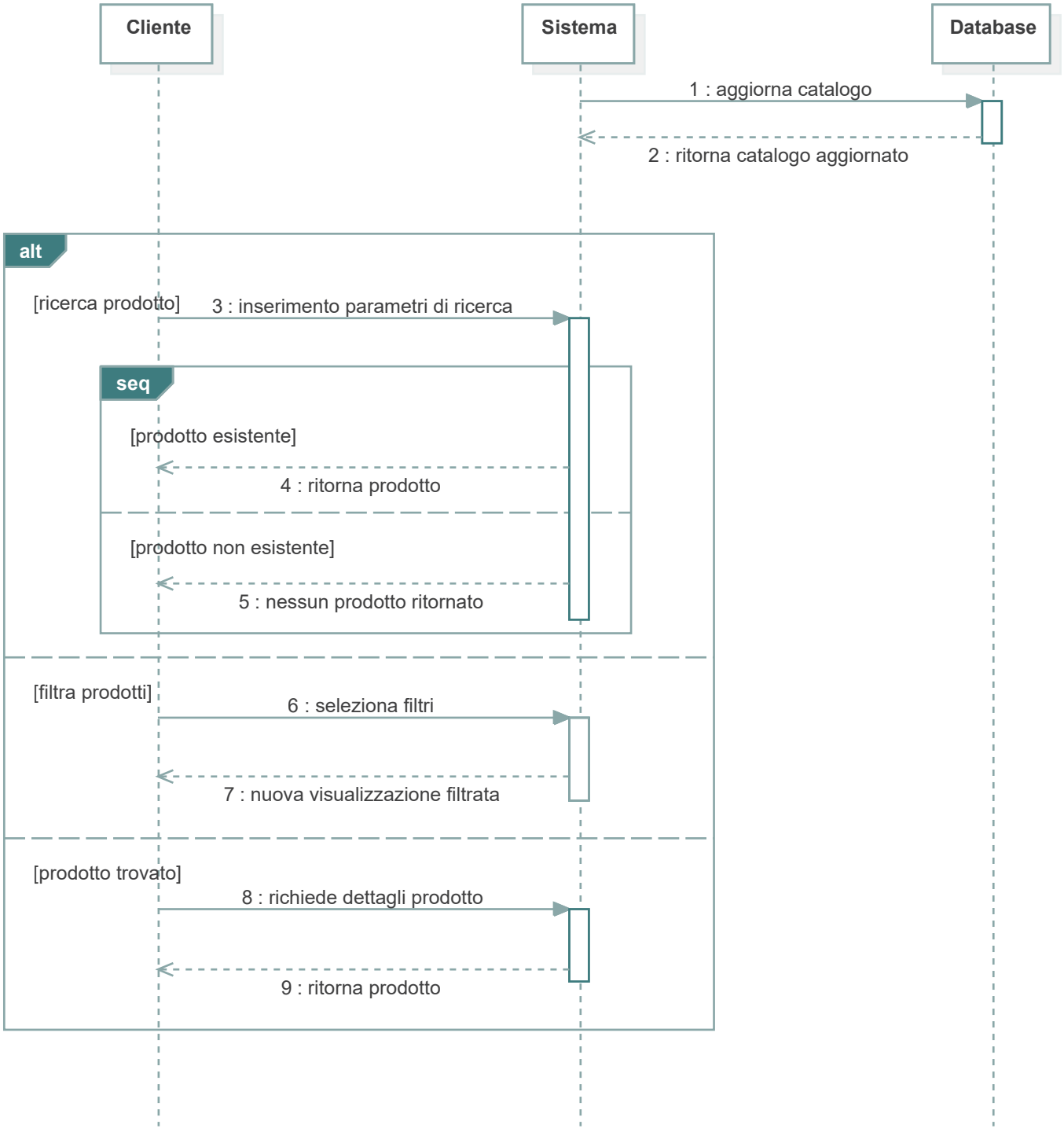


DIAGRAMMA DI SEQUENZA

ACQUISTA PRODOTTO/I

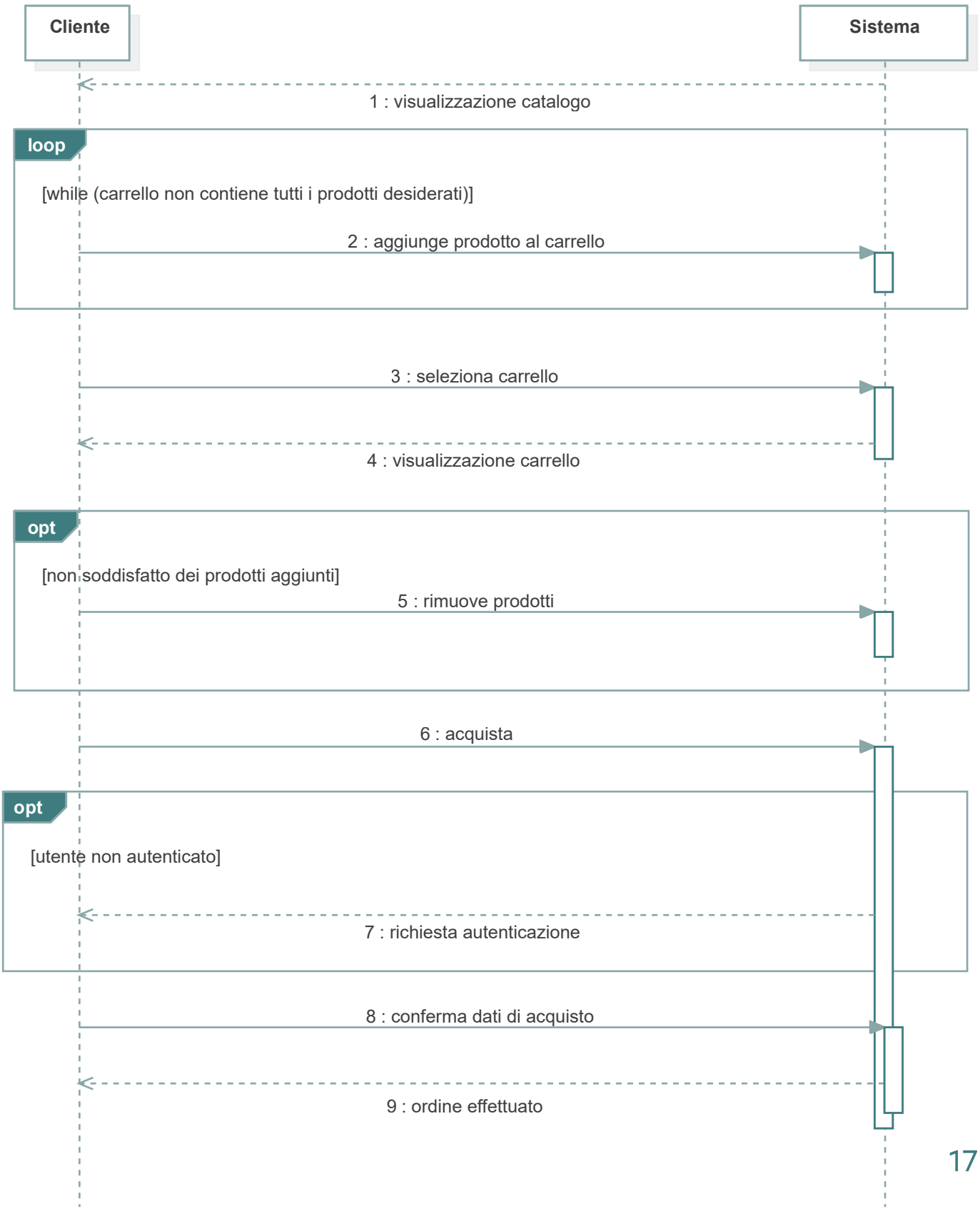


DIAGRAMMA DI SEQUENZA DEL PROTOTIPO

VISUALIZZA/MODIFICA CARRELLO

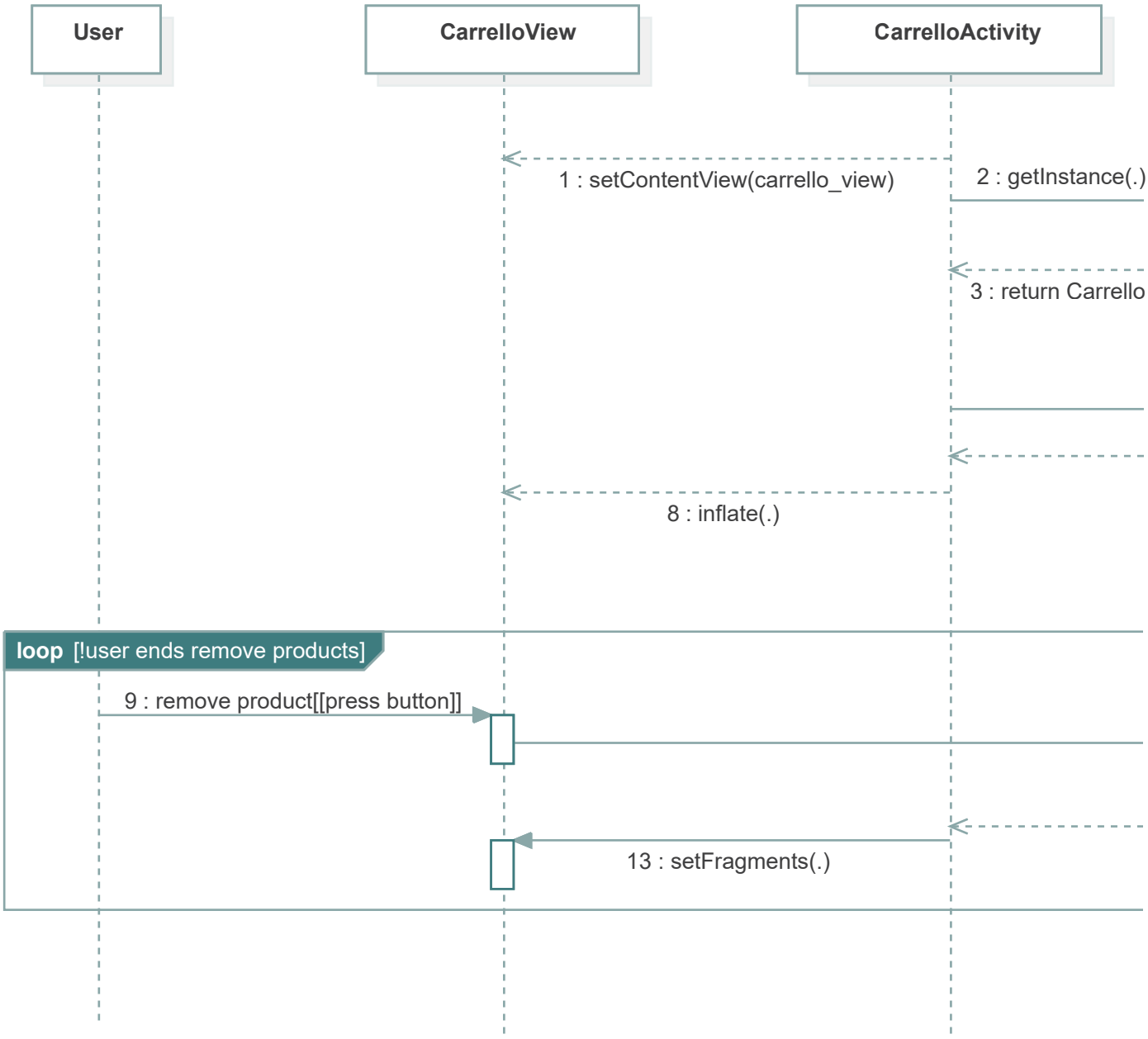
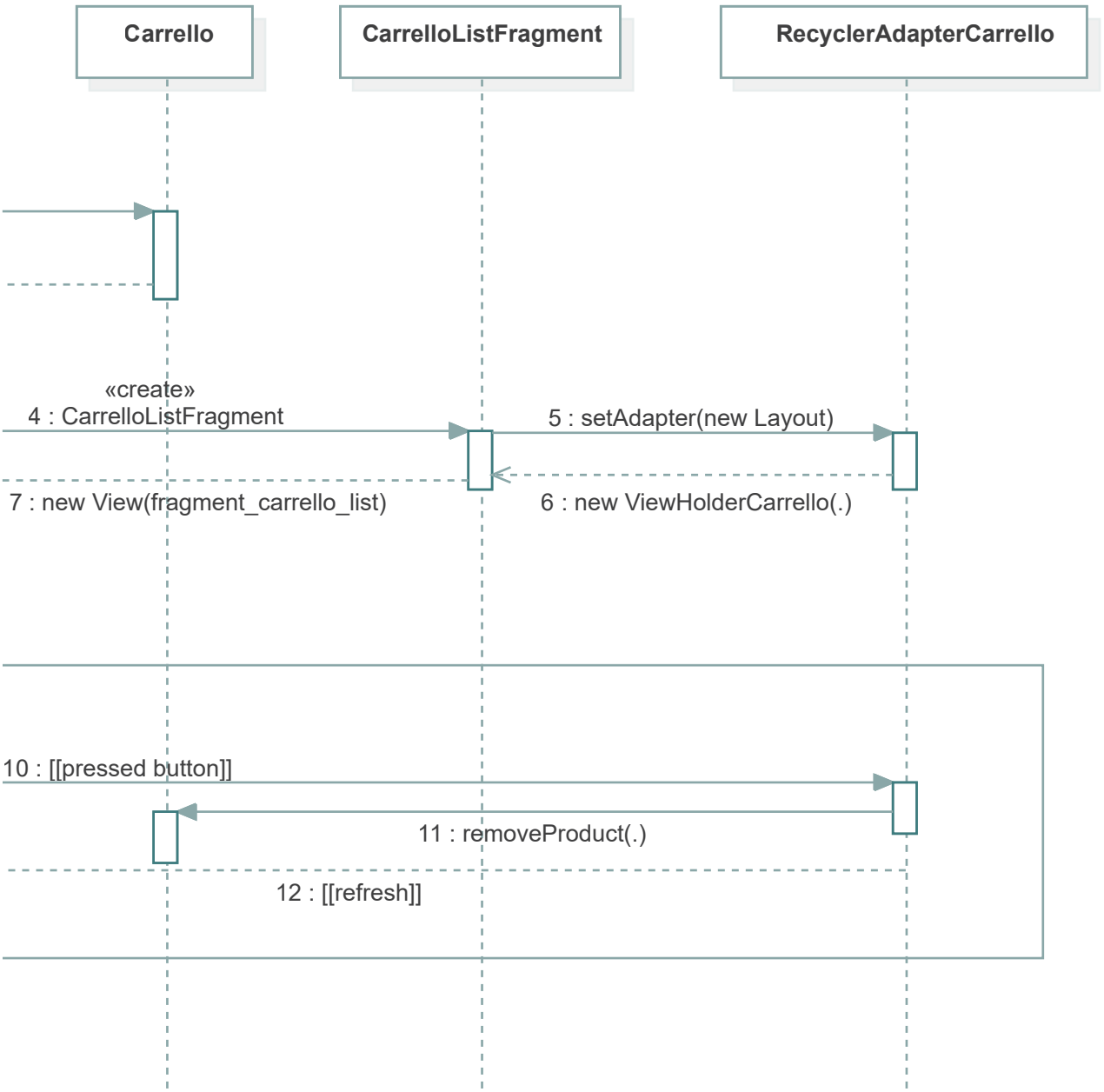


DIAGRAMMA DI SEQUENZA DEL PROTOTIPO

VISUALIZZA/MODIFICA CARRELLO

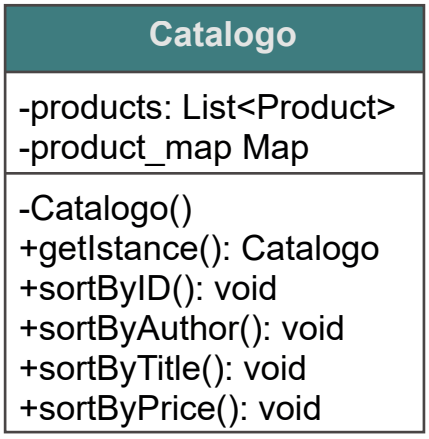


PATTERN - **SINGLETON**

Abbiamo utilizzato questo design pattern creazionale per vincolare l'istanziazione di un solo oggetto relativamente alle classi Catalogo, Carrello e MainValues.

Al loro interno vengono memorizzati i dati di sessione, ovvero i dati necessari all'applicazione durante la sessione attiva. Questi dati vengono richiesti dalle classi Controller al database, oppure gestiti direttamente, e subito caricati in memoria per permettere un'esecuzione più rapida dell'applicazione.
Dato che questi dati devono essere accessibili da qualsiasi classe, utilizzare il pattern Singleton si è rivelato indispensabile.

Di seguito riportiamo il diagramma della classe Catalogo come esempio:



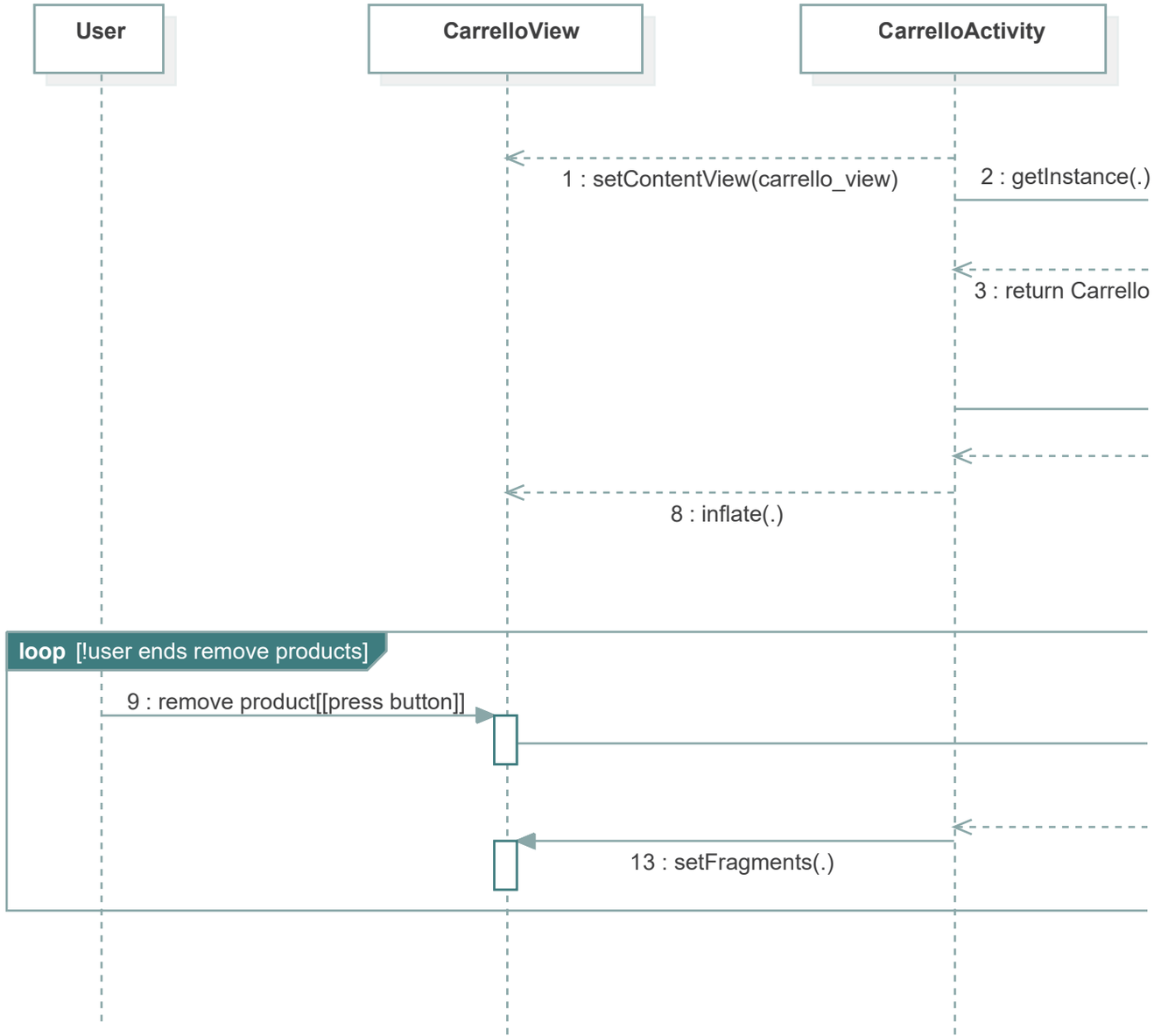
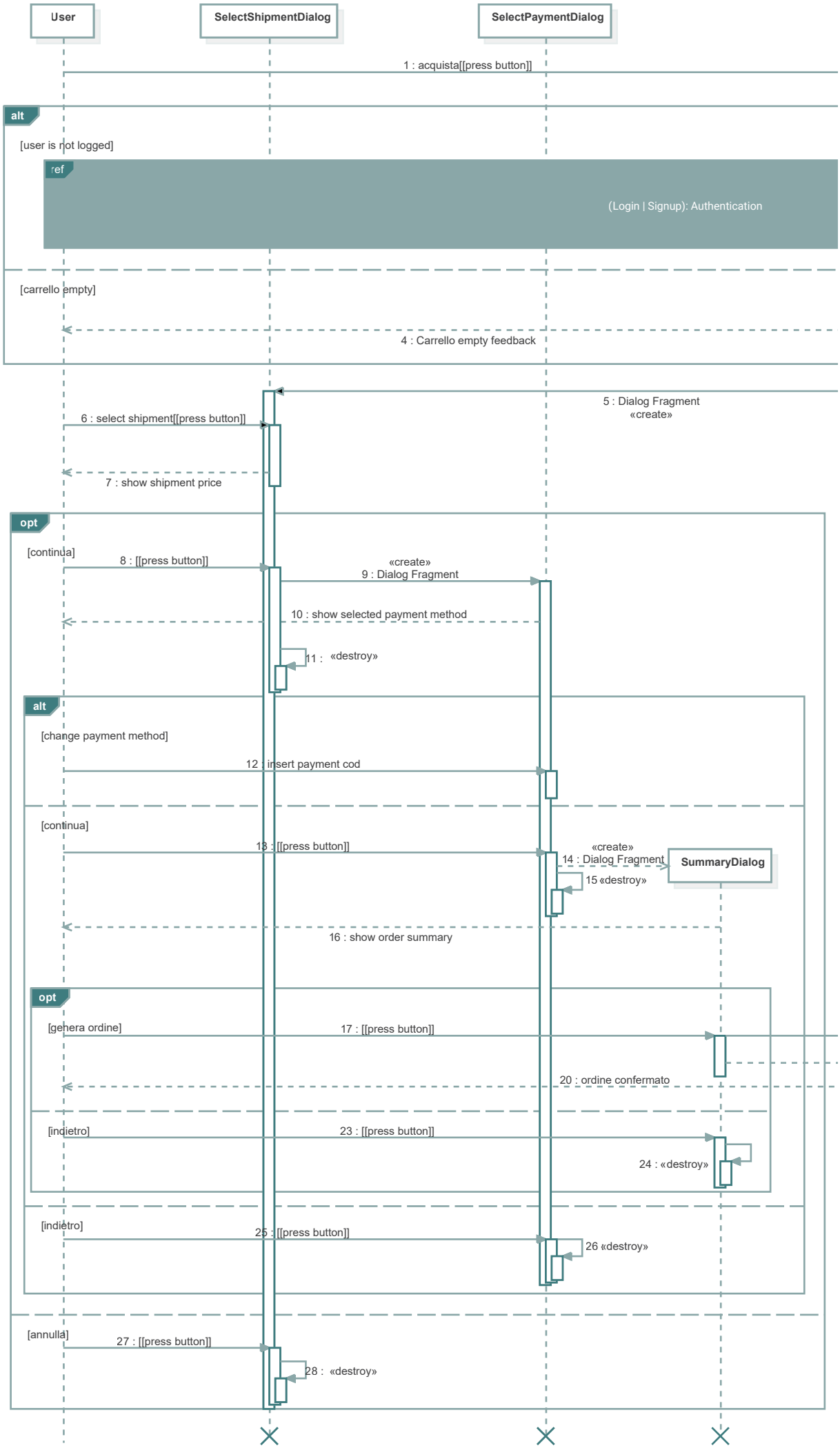
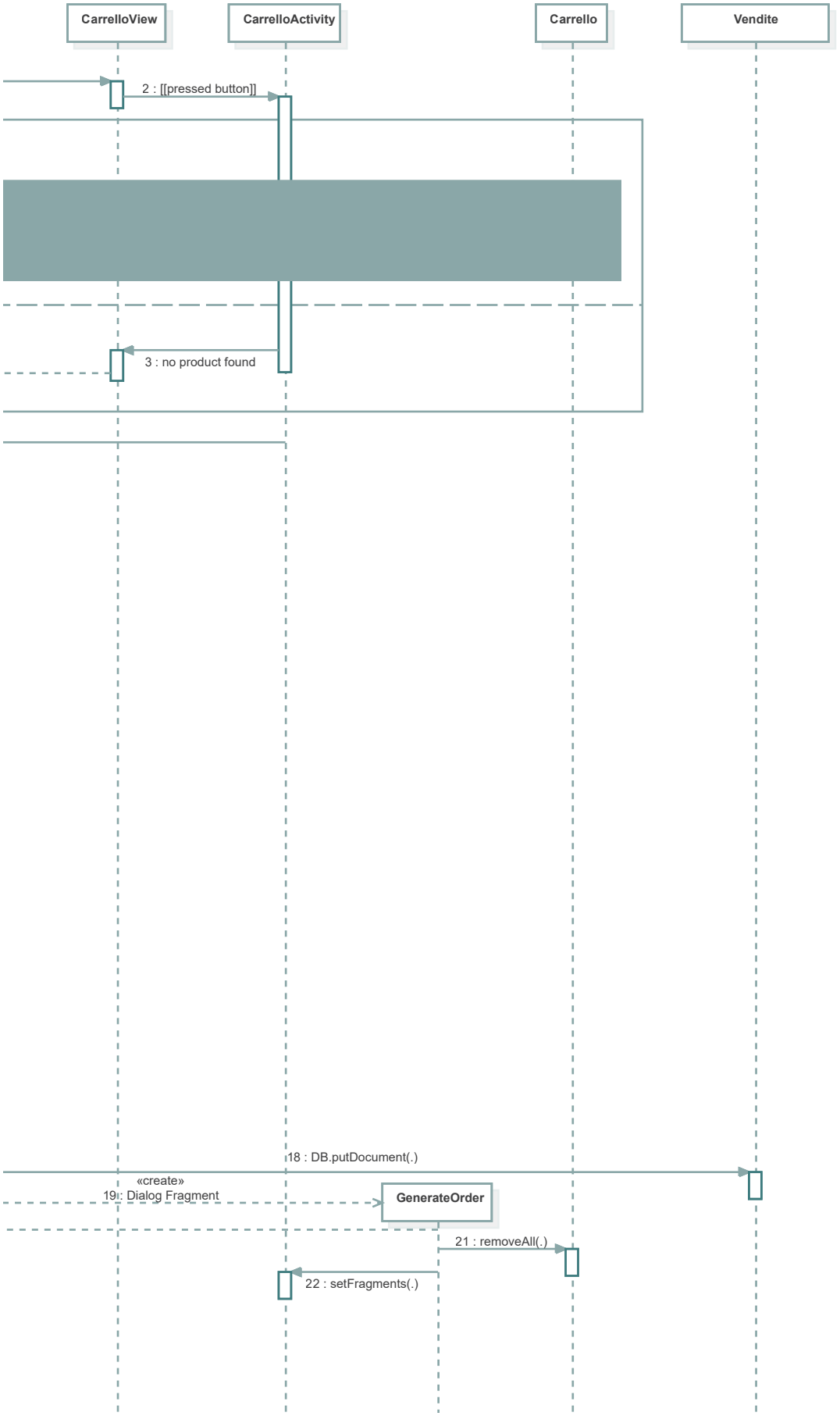

User
CarrelloView
CarrelloActivity
1 : setContentView(carrello_view)
2 : getInstance(.)
3 : return Carrello
8 : inflate(.)
loop [!user ends remove products]
9 : remove product[[press button]]
13 : setFragments(.)

DIAGRAMMA DI SEQUENZA DEL PROTOTIPO

ACQUISTA PRODOTTO/I


User
SelectShipmentDialog
SelectPaymentDialog
1 : acquista[[press button]]
alt
[user is not logged]
ref
(Login | Signup): Authentication
[carrello empty]
4 : Carrello empty feedback
5 : Dialog Fragment
«create»
6 : select shipment[[press button]]
7 : show shipment price
opt
[continua]
8 : [[press button]]
«create»
9 : Dialog Fragment
10 : show selected payment method
11 : «destroy»
alt
[change payment method]
12 : insert payment cod
[continua]
13 : [[press button]]
«create»
14 : Dialog Fragment
SummaryDialog
15 «destroy»
16 : show order summary
opt
[genera ordine]
17 : [[press button]]
20 : ordine confermato
[indietro]
23 : [[press button]]
24 : «destroy»
[indietro]
25 : [[press button]]
26 «destroy»
[annulla]
27 : [[press button]]
28 : «destroy»


CarrelloView
CarrelloActivity
Carrello
Vendite
2 : [[pressed button]]
3 : no product found
18 : DB.putDocument(.)
«create»
19 : Dialog Fragment
GenerateOrder
21 : removeAll(.)
22 : setFragments(.)

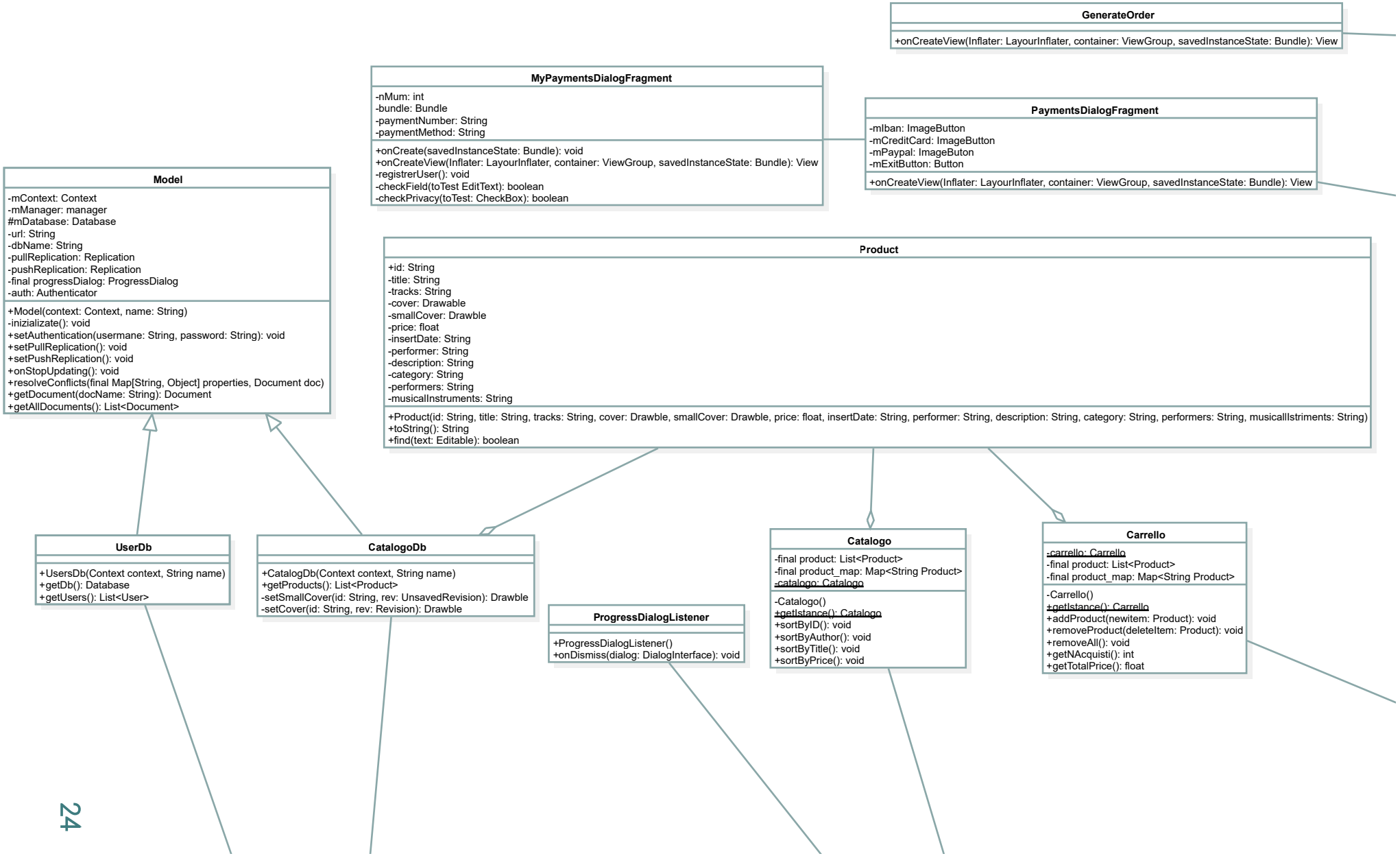
TEST EFFETTUATI

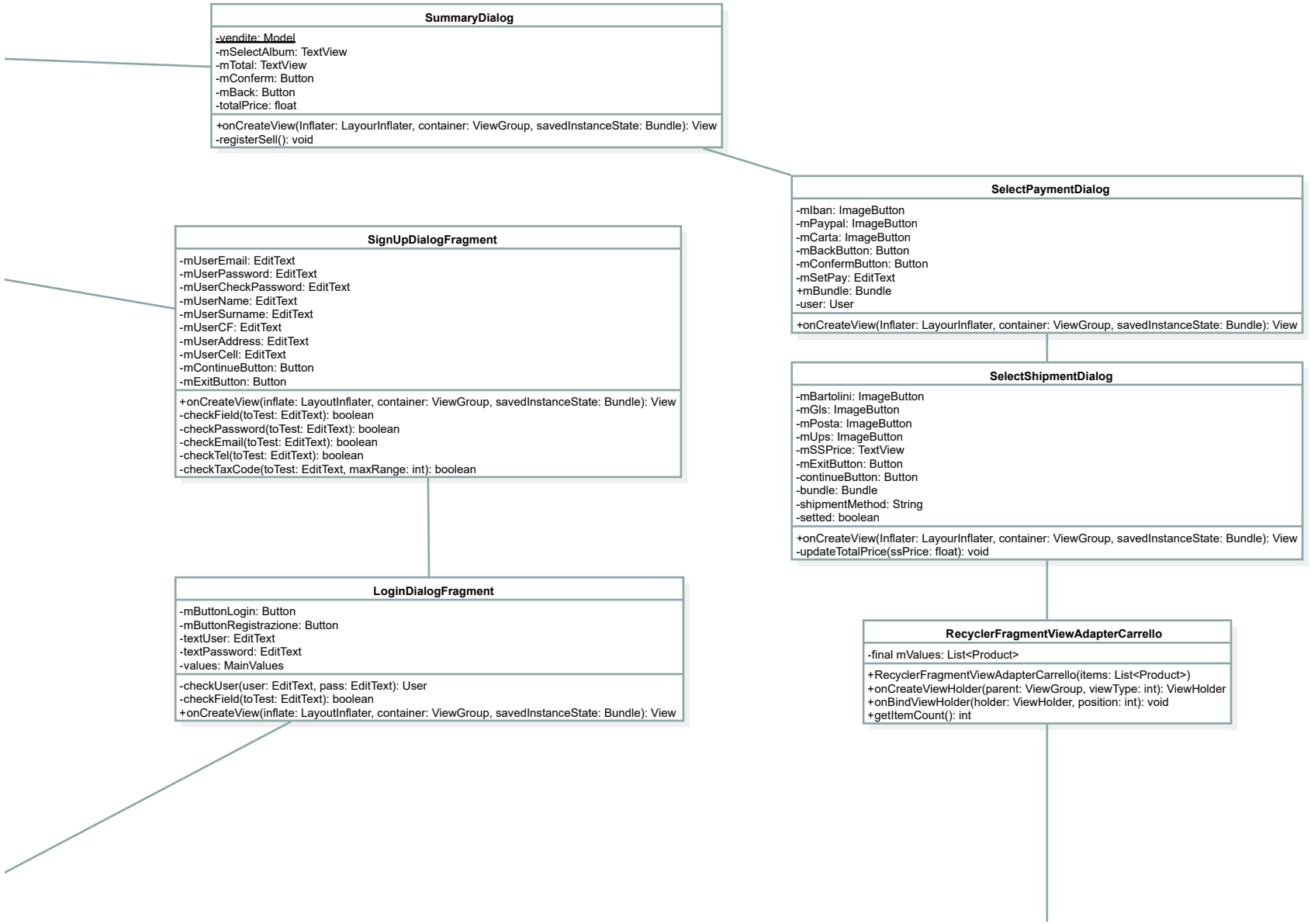
La fase di testing è stata essenziale alla corretta implementazione e funzionamento dell'applicazione.

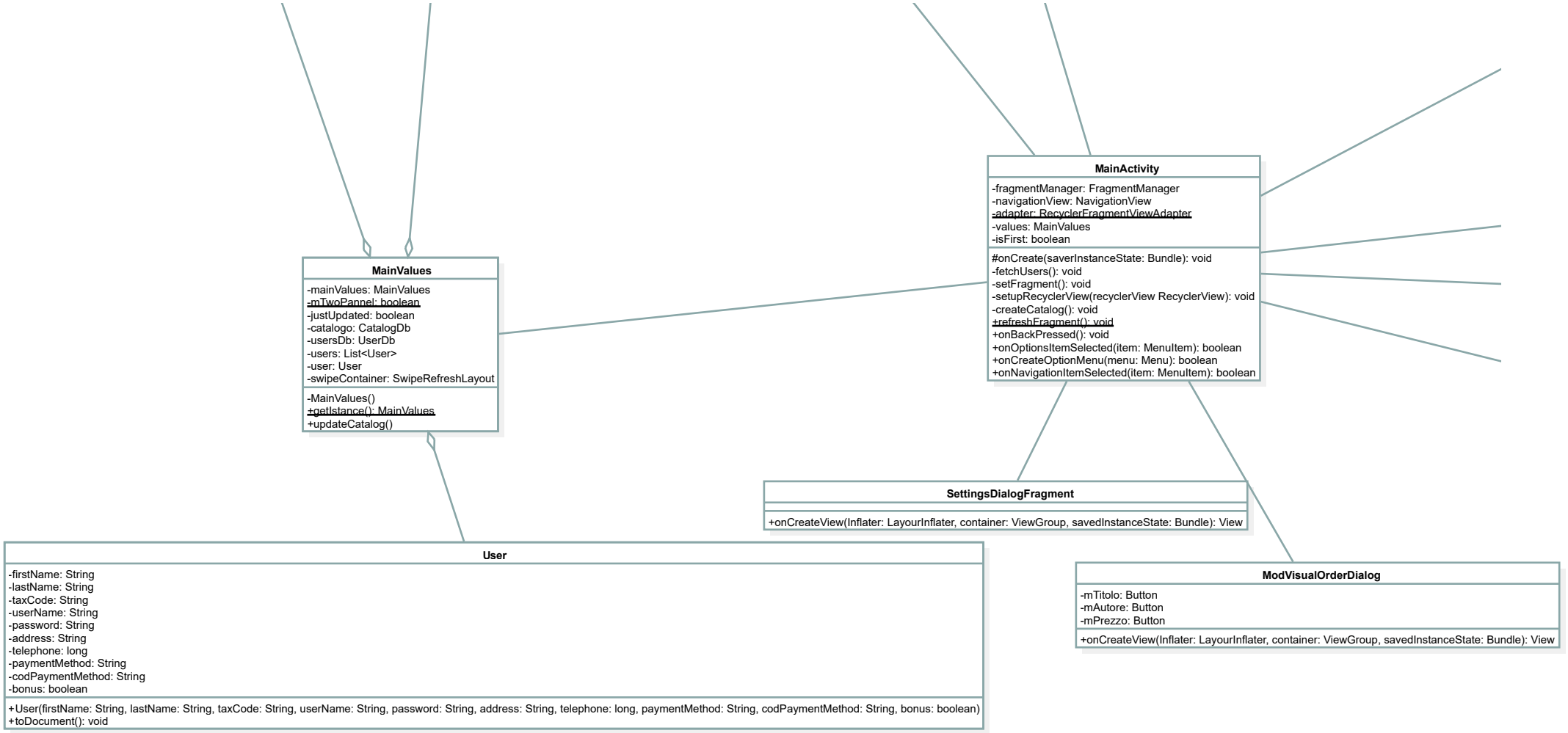
In primo luogo il software è stato testato in seguito a ciascuna modifica del codice, soprattutto sulle parti più delicate e importanti. In secondo luogo abbiamo effettuato dei test più specifici per verificare che le parti implementate funzionino in qualsiasi situazione. Abbiamo applicato diverse tecniche per stressare l'applicazione e verificare il suo utilizzo anche in condizioni non ottimali.
L'applicazione si comporta bene anche offline, registrando su un database locale qualsiasi modifica e, non appena è disponibile una connessione, aggiorna il database remoto (ovviamente in questa fase prototipale, il tutto funziona solamente se si è collegati alla stessa rete su cui gira in ascolto il database).

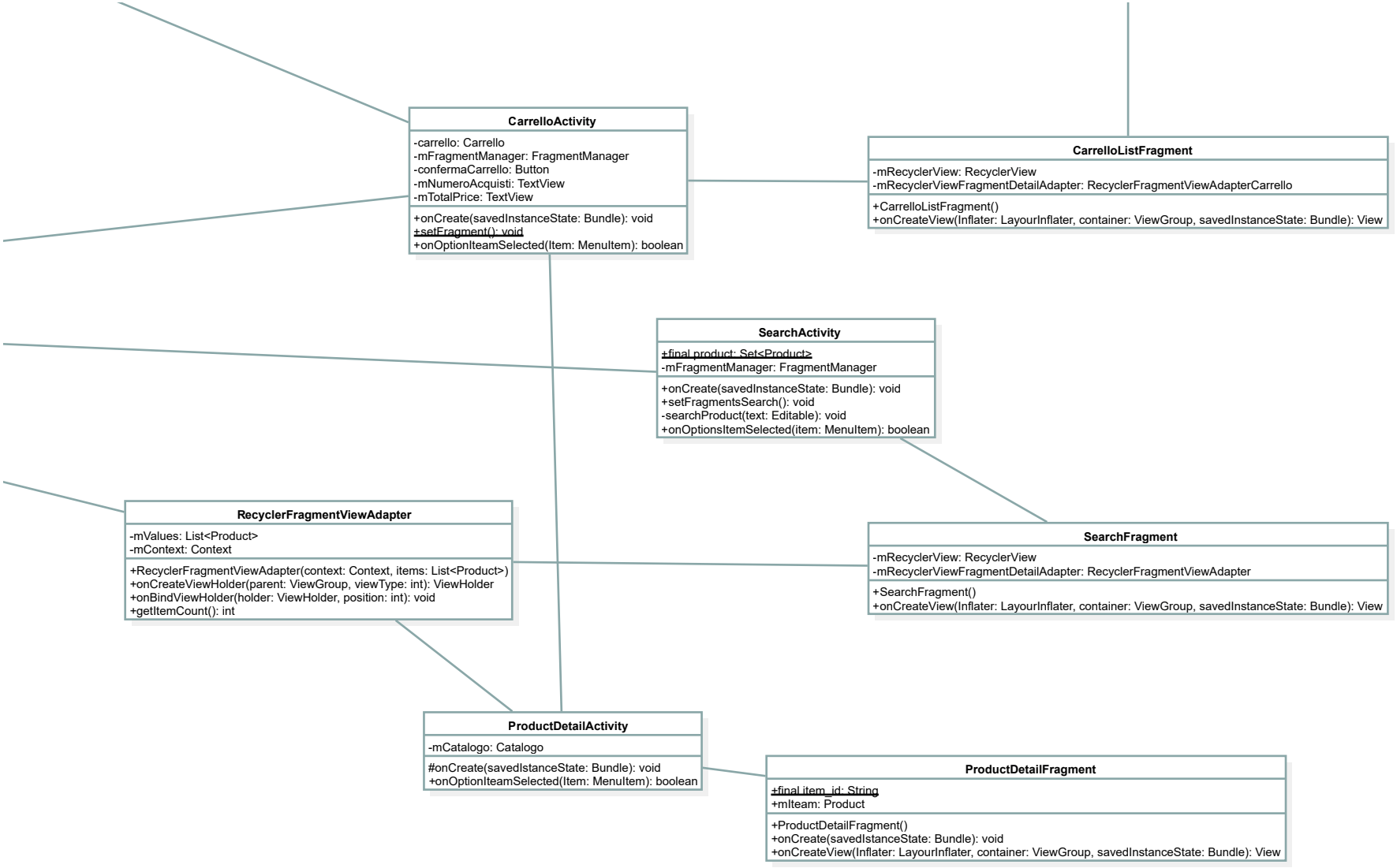
Oltre ai nostri test di stress sull'applicazione, abbiamo chiesto a persone non coinvolte nell'implementazione, tra queste anche persone di scarse abilità informatiche, di testare tutte le funzionalità implementate. Diverse anomalie rilevate, sono state corrette immediatamente.

DIAGRAMMA DELLE CLASSI


GenerateOrder
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View
MyPaymentsDialogFragment
-nMum: int
-bundle: Bundle
-paymentNumber: String
-paymentMethod: String
+onCreate(savedInstanceState: Bundle): void
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View
-registrerUser(): void
-checkField(toTest EditText): boolean
-checkPrivacy(toTest: CheckBox): boolean
PaymentsDialogFragment
-mIban: ImageButton
-mCreditCard: ImageButton
-mPaypal: ImageButton
-mExitButton: Button
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View
Model
-mContext: Context
-mManager: manager
#mDatabase: Database
-url: String
-dbName: String
-pullReplication: Replication
-pushReplication: Replication
-final progressDialog: ProgressDialog
-auth: Authenticator
+Model(context: Context, name: String)
-inizializate(): void
+setAuthentication(usermane: String, password: String): void
+setPullReplication(): void
+setPushReplication(): void
+onStopUpdating(): void
+resolveConflicts(final Map[String, Object] properties, Document doc)
+getDocument(docName: String): Document
+getAllDocuments(): List<Document>
Product
+id: String
-title: String
-tracks: String
-cover: Drawable
-smallCover: Drawble
-price: float
-insertDate: String
-performer: String
-description: String
-category: String
-performers: String
-musicalInstruments: String
+Product(id: String, title: String, tracks: String, cover: Drawble, smallCover: Drawble, price: float, insertDate: String, performer: String, description: String, category: String, performers: String, musicalIlstriments: String)
+toString(): String
+find(text: Editable): boolean
UserDb
+UsersDb(Context context, String name)
+getDb(): Database
+getUsers(): List<User>
CatalogoDb
+CatalogDb(Context context, String name)
+getProducts(): List<Product>
-setSmallCover(id: String, rev: UnsavedRevision): Drawble
-setCover(id: String, rev: Revision): Drawble
ProgressDialogListener
+ProgressDialogListener()
+onDismiss(dialog: DialogInterface): void
Catalogo
-final product: List<Product>
-final product_map: Map<String Product>
-catalogo: Catalogo
-Catalogo()
+getIstance(): Catalogo
+sortByID(): void
+sortByAuthor(): void
+sortByTitle(): void
+sortByPrice(): void
Carrello
-carrello: Carrello
-final product: List<Product>
-final product_map: Map<String Product>
-Carrello()
+getIstance(): Carrello
+addProduct(newitem: Product): void
+removeProduct(deleteItem: Product): void
+removeAll(): void
+getNAcquisti(): int
+getTotalPrice(): float


SummaryDialog
-vendite: Model
-mSelectAlbum: TextView
-mTotal: TextView
-mConferm: Button
-mBack: Button
-totalPrice: float
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View
-registerSell(): void
SelectPaymentDialog
-mIban: ImageButton
-mPaypal: ImageButton
-mCarta: ImageButton
-mBackButton: Button
-mConfermButton: Button
-mSetPay: EditText
+mBundle: Bundle
-user: User
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View
SignUpDialogFragment
-mUserEmail: EditText
-mUserPassword: EditText
-mUserCheckPassword: EditText
-mUserName: EditText
-mUserSurname: EditText
-mUserCF: EditText
-mUserAddress: EditText
-mUserCell: EditText
-mContinueButton: Button
-mExitButton: Button
+onCreateView(inflate: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle): View
-checkField(toTest: EditText): boolean
-checkPassword(toTest: EditText): boolean
-checkEmail(toTest: EditText): boolean
-checkTel(toTest: EditText): boolean
-checkTaxCode(toTest: EditText, maxRange: int): boolean
SelectShipmentDialog
-mBartolini: ImageButton
-mGls: ImageButton
-mPosta: ImageButton
-mUps: ImageButton
-mSSPrice: TextView
-mExitButton: Button
-continueButton: Button
-bundle: Bundle
-shipmentMethod: String
-setted: boolean
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View
-updateTotalPrice(ssPrice: float): void
LoginDialogFragment
-mButtonLogin: Button
-mButtonRegistrazione: Button
-textUser: EditText
-textPassword: EditText
-values: MainValues
-checkUser(user: EditText, pass: EditText): User
-checkField(toTest: EditText): boolean
+onCreateView(inflate: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle): View
RecyclerFragmentViewAdapterCarrello
-final mValues: List<Product>
+RecyclerFragmentViewAdapterCarrello(items: List<Product>)
+onCreateViewHolder(parent: ViewGroup, viewType: int): ViewHolder
+onBindViewHolder(holder: ViewHolder, position: int): void
+getItemCount(): int


MainActivity
-fragmentManager: FragmentManager
-navigationView: NavigationView
-adapter: RecyclerFragmentViewAdapter
-values: MainValues
-isFirst: boolean
#onCreate(saverInstanceState: Bundle): void
-fetchUsers(): void
-setFragment(): void
-setupRecyclerView(recyclerView RecyclerView): void
-createCatalog(): void
+refreshFragment(): void
+onBackPressed(): void
+onOptionsItemSelected(item: MenuItem): boolean
+onCreateOptionMenu(menu: Menu): boolean
+onNavigationItemSelected(item: MenuItem): boolean
MainValues
-mainValues: MainValues
-mTwoPannel: boolean
-justUpdated: boolean
-catalogo: CatalogDb
-usersDb: UserDb
-users: List<User>
-user: User
-swipeContainer: SwipeRefreshLayout
-MainValues()
+getIstance(): MainValues
+updateCatalog()
SettingsDialogFragment
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View
User
-firstName: String
-lastName: String
-taxCode: String
-userName: String
-password: String
-address: String
-telephone: long
-paymentMethod: String
-codPaymentMethod: String
-bonus: boolean
+User(firstName: String, lastName: String, taxCode: String, userName: String, password: String, address: String, telephone: long, paymentMethod: String, codPaymentMethod: String, bonus: boolean)
+toDocument(): void
ModVisualOrderDialog
-mTitolo: Button
-mAutore: Button
-mPrezzo: Button
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View


CarrelloActivity
-carrello: Carrello
-mFragmentManager: FragmentManager
-confermaCarrello: Button
-mNumeroAcquisti: TextView
-mTotalPrice: TextView
+onCreate(savedInstanceState: Bundle): void
+setFragment(): void
+onOptionIteamSelected(Item: MenuItem): boolean
CarrelloListFragment
-mRecyclerView: RecyclerView
-mRecyclerViewFragmentDetailAdapter: RecyclerFragmentViewAdapterCarrello
+CarrelloListFragment()
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View
SearchActivity
+final product: Set<Product>
-mFragmentManager: FragmentManager
+onCreate(savedInstanceState: Bundle): void
+setFragmentsSearch(): void
-searchProduct(text: Editable): void
+onOptionsItemSelected(item: MenuItem): boolean
RecyclerFragmentViewAdapter
-mValues: List<Product>
-mContext: Context
+RecyclerFragmentViewAdapter(context: Context, items: List<Product>)
+onCreateViewHolder(parent: ViewGroup, viewType: int): ViewHolder
+onBindViewHolder(holder: ViewHolder, position: int): void
+getItemCount(): int
SearchFragment
-mRecyclerView: RecyclerView
-mRecyclerViewFragmentDetailAdapter: RecyclerFragmentViewAdapter
+SearchFragment()
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View
ProductDetailActivity
-mCatalogo: Catalogo
#onCreate(savedIstanceState: Bundle): void
+onOptionIteamSelected(Item: MenuItem): boolean
ProductDetailFragment
+final item_id: String
+mIteam: Product
+ProductDetailFragment()
+onCreate(savedIstanceState: Bundle): void
+onCreateView(Inflater: LayourInflater, container: ViewGroup, savedInstanceState: Bundle): View