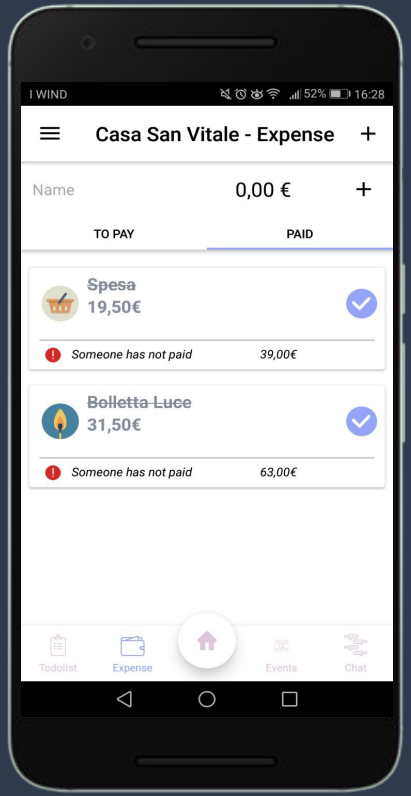
# Utilizzo di Kotlin per sviluppo di applicazioni mobili: un caso di studio

**Presentata da:** *Alessio Koci*
**Relatore:** *Dott. Luca Bedogni*
**Sessione:** *III*
**Anno Accademico:** *2016/2017*

# Applicazione




## Todolist

Gestione delle mansioni da svolgere

## Expense

Gestione delle spese da dividere fra i membri del gruppo

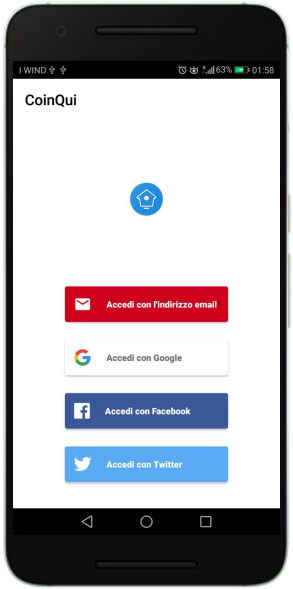
## Events

Gestione degli eventi ricorrenti e singoli

## Chat

Chat di messaggistica istantanea

# Kotlin per lo sviluppo di applicazioni mobili




*Applicazione realizzata in Kotlin utilizzando come MBaaS Firebase.*

*Google durante la conferenza "Google I/O 2017" annunciò il supporto ufficiale a Kotlin come nuovo linguaggio di programmazione.*

**Interoperabilità**

I linguaggi Kotlin e Java possono coesistere all'interno dello stesso progetto

**Programmazione funzionale**

Funzioni di ordine superiore, lambda function

**Smart Casting**

Compilatore riconosce il tipo della variabile

**Lazy e Late Initialization**

Posticipare l'inizializzazione di una variabile

**Struttura di controllo**

Ogni struttura di controllo in Kotlin è un'espressione che restituisce un valore

**String Template**

Fare riferimento a variabili, durante la rappresentazione di stringhe

**Type Safety**

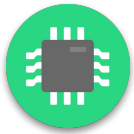
Prevenzione degli errori di tipo

**Data Class**

Creazione e generazione di classi senza codice boilerplate

**Kotlin Android Extension**

Interazione con la View, importanto il layout



**Coroutines**

Semplificazione della programmazione asincrona

# Caratteristiche Kotlin su Android

# Kotlin

```
data class Person(
var name: String,
var surname: String,
var id: String)
```

***JetBrains*** *ha rilasciato un plugin per Android Studio in grado di convertire correttamente più del 70% del codice Java in Kotlin, lasciando allo sviluppatore la possibilità di correggere il rimanente codice dovuto alla conversione.*

# Java

```
public class Person {
    private String name;
    private String surname;
    private String id;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getSurname() {
        return surname;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    @Override public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Person person = (Person) o;
        if (name != null ? !name.equals(person.name) : person.name != null) return false;
        if (surname != null ? !surname.equals(person.surname) : person.surname != null)
            return false;
        return id != null ? id.equals(person.id) : person.id == null;
    }
    @Override public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + (surname != null ? surname.hashCode() : 0);
        result = 31 * result + (id != null ? id.hashCode() : 0);
        return result;
    }
    @Override public String toString() {
        return "Person{" +
                "name='" + name + "'" +
                ", surname='" + surname + "'" +
                ", id='" + id + "'" +
                '}';
    }
}
```

## LAMBDA EXPRESSION, STRING TEMPLATE

```
View view = (View) findViewById(R.id.view); view.setOnClickListener(
new View.OnClickListener() {
  @Override public void onClick(View v) {
  Toast.makeText (this, "Hello" + Person.name, Toast.LENGTH_LONG )
.show();
} });
```

```
view.setOnClickListener {
        toast("Hello ${Person.name}")
}
```

## FUNZIONI , VARIABILI, TYPE INFERENCE

```
String name = null;
void drawText(int x, int y, int size,String text){
   if( x == null) { x = 0; }
   if( y == null) { y = 0; }
   Int sum = x + y;
   name = "Bob";
}
```

```
lateinit var name: String
fun drawText(x: Int = 0, y: Int = 0, String: text){
   val sum = x + y
   name = "Bob"
}
```

## NULL SAFETY

```
String nullableGreeting: = "Hello, World";
nullableGreeting  = null ;
Int len = nullableGreeting.length ; // Runtime Error
```

```
var nullableGreeting: String? = "Hello, World"
nullableGreeting = null
val len = nullableGreeting.length // Compilation Error
```

# FIREBASE

**Firebase** è una piattaforma Mobile backend as a service (MBaaS) che consente di interfacciare applicazioni mobili e web app ad un cloud backend.

---

**Firebase**
Servizio principale di Firebase

**Authentication**
Servizio di autenticazione

**Cloud Functions**
Servizio di scripting backend

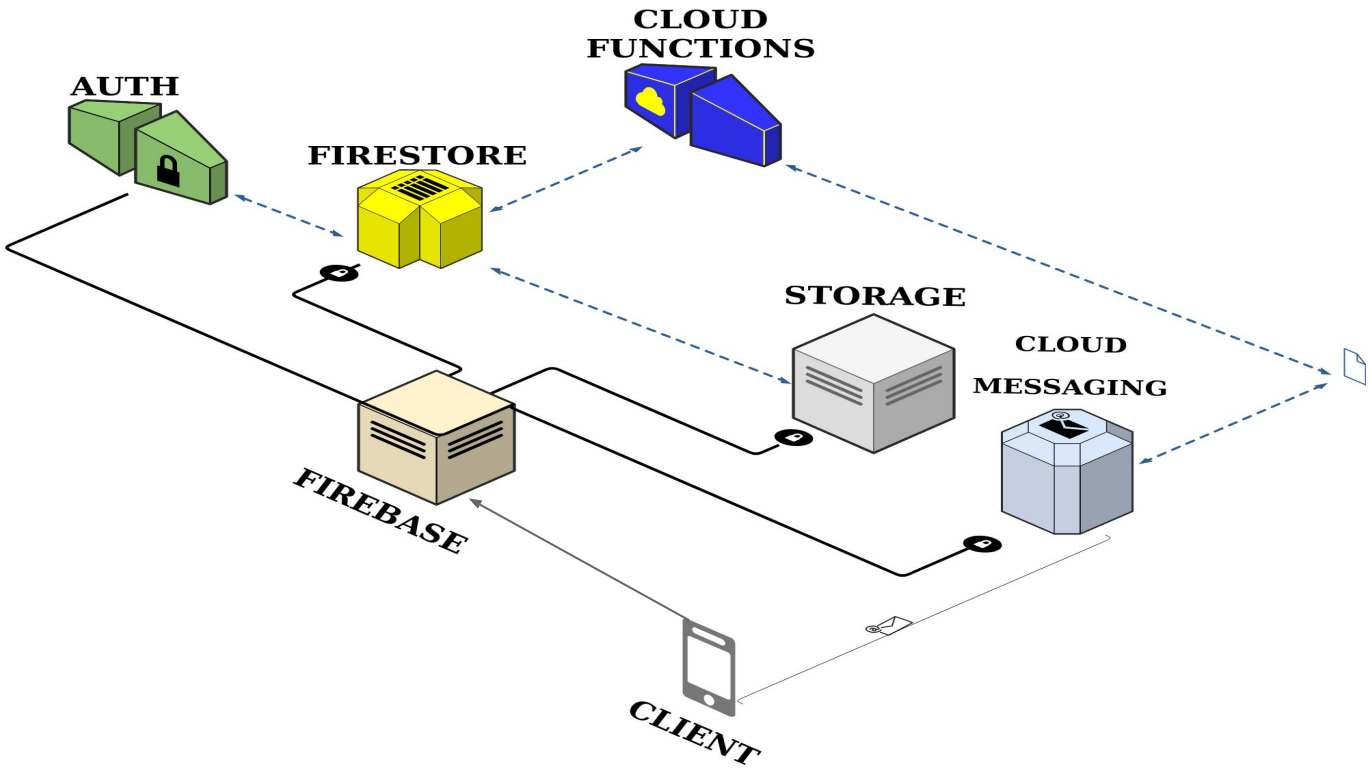
**Storage**
Servizio di archiviazione dei file

**Cloud Messaging**
Servizio di messaggistica istantanea

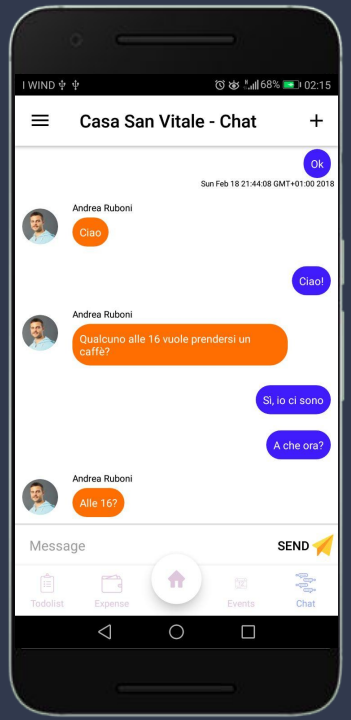
**Firestore**
Servizio di gestione del database

Firebase

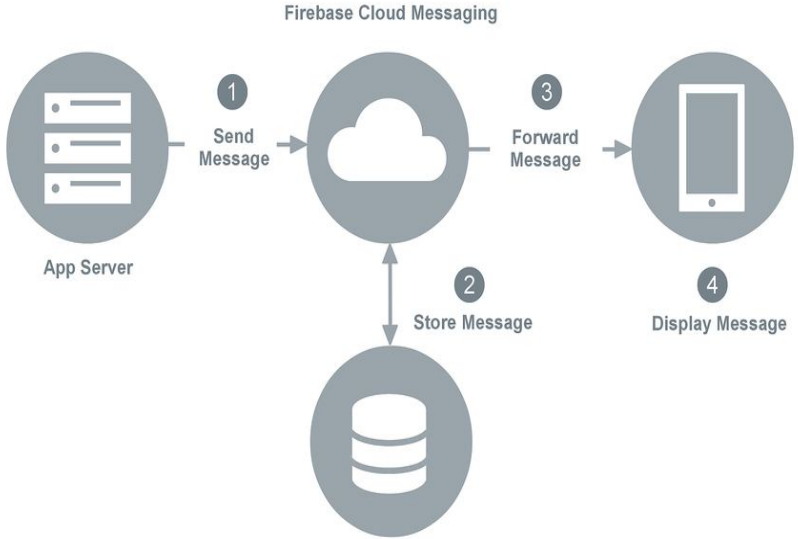
# Architettura server



# Notifiche

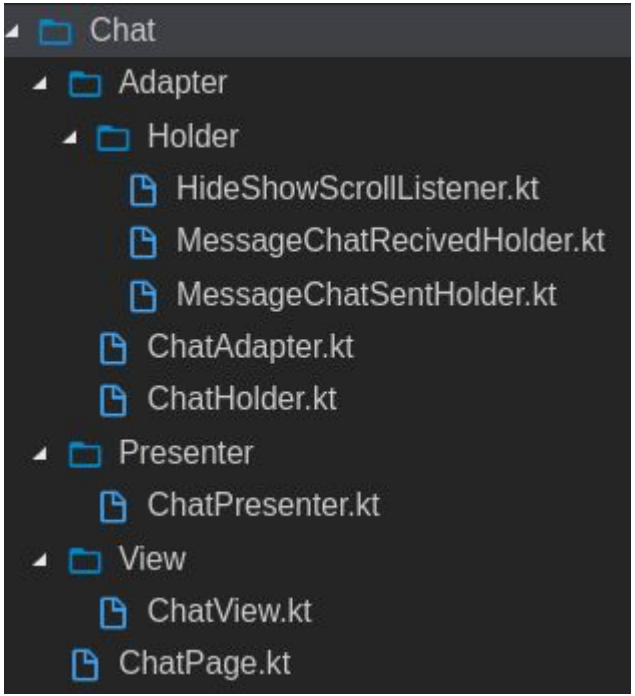



# Implementazione



# Model View Presenter

## Esempio struttura




**View** ⟷ **Presenter**

Interfaccia passiva che visualizza i dati (il modello) e instrada i comandi utente (eventi) al Presenter per agire su tali dati.

Recupera i dati dai repository e li formatta per la visualizzazione.

**Model**

Interfaccia che definisce i dati da visualizzare.

# Reactive Programming

## Cos'è?

La programmazione reattiva o Reactive Programming è un paradigma di programmazione che monitora e gestisce flussi di dati statici o dinamici e la propagazione dei flussi nel tempo.

## Esempi

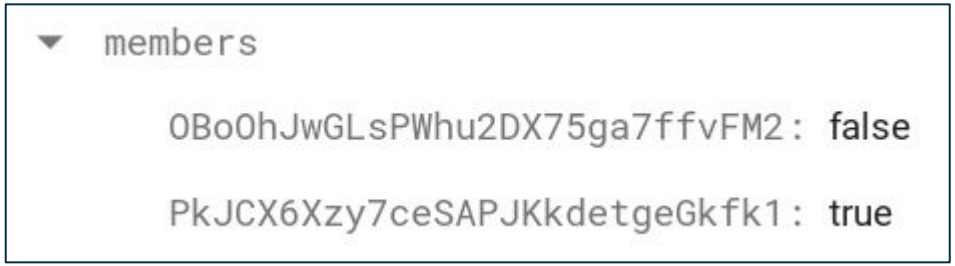
```
fun getTodoItems(): Single<List<TodoItem>> {
    return Single.create<List<TodoItem>> { emitter ->
      queryTodo.getTodoItems().addSnapshotListener({ querySnapshot, exception ->
        if (exception != null)
          emitter.onError(Throwable(exception))
        else {
          emitter.onNext(TodoItem.mapping(querySnapshot))
        }
      })
    }
  }
```

```
groupRepository.createGroup(userUid, groupItem)
.observeOn(AndroidSchedulers.mainThread())
.subscribeOn(Schedulers.newThread())
.doOnSuccess {
    [...]
}
```

| Dato | Tipo |
| --- | --- |
| **ID** | *Int* |
| **Name** | *String* |
| **Date** | *Date* |
| **Description** | *String* |
| **Priority** | *Int* |
| **Members** | *Map* |
| **CompletedUser** | *Int* |
| **CreatedBy** | *Int* |
| **Status** | *Boolean* |

**Todolist**

## Divisione spese fra membri




**Quota parziale** = Ammontare totale / numero di membri

# Esempio di implementazione

# Sicurezza

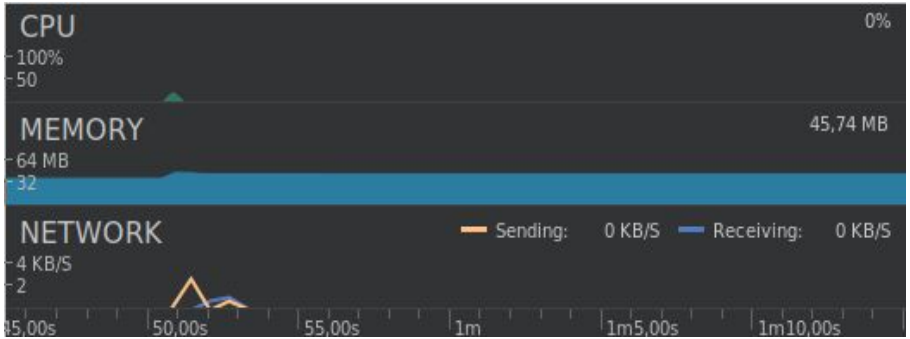
## Firestore Rules

```
match /groups/{groupsId} {
        allow read: if isUserAuthenticated();
        allow write, update, delete: if isUserAuthenticated ()  && userBelongsToGroup()

        match /todolist/{todolistId} {
                allow write, update, delete if canUserReadItem()
                [...]
        }
       [...]
}
```

# Confronto Kotlin e Java

## Java




## Kotlin


- Linee di codice
- Codice coinciso
- Dimensione pacchetto APK
- Tool di sviluppo e analisi

# Domande?

Grazie per l'attenzione.