## GPS

##### Cos’è e come funziona?

è un sistema di posizionamento e navigazione satellitare civile che, attraverso una rete dedicata di satelliti artificiali in orbita, fornisce ad un terminale mobile o ricevitore GPS informazioni sulle sue coordinate geografiche ed orario, in ogni condizione meteorologica, ovunque sulla Terra o nelle sue immediate vicinanze ove vi sia un contatto privo di ostacoli con almeno quattro satelliti del sistema. La localizzazione avviene tramite la trasmissione di un segnale radio da parte di ciascun satellite e l'elaborazione dei segnali ricevuti da parte del ricevitore.

##### Come è implementato in Android?

Esiste la classe Location Manager, la quale permette all’applicazione di ricevere aggiornamenti periodici sulla posizione indicano precedentemente quale provider per la locazione utilizzare. Esistono tre tipi di provider:

* GPS provider
  + Alta precisione ma alto consumo batteria
* Network provider
  + Bassa precisione(300m) e basso uso batteria
* FusedLocation provider di Google con le google APIs
  + Precisione abbastanza elevata con un utilizzo di energia non troppo elevato

**API** è l'acronimo di Application Programming Interface. Le API permettono di espandere le funzionalità di un programma. Per uno sviluppatore mettere a disposizione un set di API di un suo software significa dare la possibilità ad altri di interagire con la sua piattaforma e, soprattutto, estendere le funzioni e le caratteristiche della struttura base della piattaforma. In altri termini, le API sono un ottimo strumento per promuovere un programma offrendo ad altri un modo per interagirci.

## Firebase

Firebase è una piattaforma che fornisce numerosi servizi per lo sviluppo di applicazioni ad alto livello, tra i quali:

* Database realtime ad oggetti
* Archiviazione online di dati
* Autenticazione sicura
* Hosting di siti web (similmente ad Heroku o Altervista)

Riducendo i problemi legati all’infrastruttura dell’app e alla sua connessione con servizi remoti anche di diverso tipo (ad esempio AAA server, database e archiviazione limitata, limitazioni di costi e performance locando dei propri server a casa), si rende possibile una maggiore attenzione allo sviluppo dell’applicazione, in particolare alla sua interazione con i sistemi per i quali le applicazioni sono create.

Sono disponibili anche altre funzionalità accessorie, quali:

* Firebase Analytics: permette di ottentere un immediato riscontro sull’utilizzo dell’app e dei suoi problemi
* Test Lab: permette di testare le applicazioni Android su un largo numero di dispositivi virtuali contemporaneamente
* Notifiche: si possono mandare delle notifiche ai dispositivi che utilizzano l’applicazione
* AdMob: consente l’inserimento di pubblicità con la quale guadagnare denaro

**Database ad oggetti**

Sebbene i database relazionali siano dagli anni ’70 la scelta migliore per la memorizzazione permanente di grosse quantità di dati, negli ultimi anni, dato l’aumento continuo di necessità di informazioni immediate, si sono sviluppate nuove tecnologie NoSQL.

I database ad oggetti esistono sin dagli anni ’80, ma non sono mai stati presi in considerazione in quanto giudicati poco performanti.

Oggigiorno le motivazioni di una scelta di un OODB sono la semplicità nel design, un utilizzo più flessibile e, in alcuni casi, uguale velocità rispetto ai database relazionali.

La principale differenza con i database relazionali è la scelta di una maggiore disponibilità (percentuale di tempo di utilizzo produttivo del database) a discapito della consistenza; la maggior parte dell’elaborazione dei dati è, inoltre, delegata al software applicativo.

Il paradigma ad oggetti è assai diverso da quello relazionale, derivante dalla teoria degli insiemi, pertanto, molte operazioni, quali le join, non sono disponibili, tuttavia, le letture dei dati sono in media più veloci.

Un particolare sottoinsieme dei database ad oggetti sono i database a documento, dove per documento si intende una particolare codifica ed incapsulamento dei dati (ad esempio XML, JSON, GSON). I documenti possono essere intesi come oggetti in quanto non subiscono la rigida forma della struttura relazionale, in quanto non necessitano tutti le stesse sezioni, chiave, tipi di attributi.

Un programma, infatti ha molti tipi di oggetto, e gli stessi tipi, a loro volta hanno dei campi opzionali introducendo delle caratteristiche proprie. In ugual modo i documenti sono simili perché permettono di avere campi opzionali, sono facilmente ricodificabili e permettono la memorizzazione di molti oggetti in un unico punto di archiviazione.

Da finire

In a relational database, data is first categorized into a number of predefined types, and *tables* are created to hold individual entries, or *records*, of each type. The tables define the data within each record's *fields*, meaning that every record in the table has the same overall form. The administrator also defines the *relationships* between the tables, and selects certain fields that they believe will be most commonly used for searching and defines *indexes* on them. A key concept in the relational design is that any data that may be repeated is normally placed in its own table, and if these instances are related to each other, a column is selected to group them together, the *foreign key*. This design isknownas [*database normalization*](https://en.wikipedia.org/wiki/Database_normalization).[[2]](https://en.wikipedia.org/wiki/Document-oriented_database#cite_note-4)

For example, an address book application will generally need to store the contact name, an optional image, one or more phone numbers, one or more mailing addresses, and one or more email addresses. In a canonical relational database, tables would be created for each of these rows with predefined fields for each bit of data: the CONTACT table might include FIRST\_NAME, LAST\_NAME and IMAGE columns, while the PHONE\_NUMBER table might include COUNTRY\_CODE, AREA\_CODE, PHONE\_NUMBER and TYPE (home, work, etc.). The PHONE\_NUMBER table also contains a foreign key column, "CONTACT\_ID", which holds the unique ID number assigned to the contact when it was created. In order to recreate the original contact, the database engine uses the foreign keys to look for the related items across the group of tables and reconstruct the original data.

In contrast, in a document-oriented database there may be no internal structure that maps directly onto the concept of a table, and the fields and relationships generally don't exist as predefined concepts. Instead, all of the data for an object is placed in a single document, and stored in the database as a single entry. In the address book example, the document would contain the contact's name, image, and any contact info, all in a single record. That entry is accessed through its key, which allows the database to retrieve and return the document to the application. No additional work is needed to retrieve the related data; all of this is returned in a single object.

A key difference between the document-oriented and relational models is that the data formats are not predefined in the document case. In most cases, any sort of document can be stored in any database, and those documents can change in type and form at any time. If one wishes to add a COUNTRY\_FLAG to a CONTACT, this field can be added to new documents as they are inserted, this will have no effect on the database or the existing documents already stored. To aid retrieval of information from the database, document-oriented systems generally allow the administrator to provide *hints* to the database to look for certain types of information. These work in a similar fashion to indexes in the relational case. Most also offer the ability to add additional metadata outside of the content of the document itself, for instance, tagging entries as being part of an address book, which allows the programmer to retrieve related types of information, like "all the address book entries". This provides functionality similar to a table, but separates the concept (categories of data) from its physical implementation (tables).[[3]](https://en.wikipedia.org/wiki/Document-oriented_database#cite_note-5)

In the classic normalized relational model, objects in the database are represented as separate rows of data with no inherent structure beyond that given to them as they are retrieved. This leads to problems when trying to translate programming objects to and from their associated database rows, a problem known as [object-relational impedance mismatch](https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch).[[4]](https://en.wikipedia.org/wiki/Document-oriented_database#cite_note-6) Document stores more closely, or in some cases directly, map programming objects into the store. These are oftenmarketedusing the term [NoSQL](https://en.wikipedia.org/wiki/NoSQL).

**Database Realtime**

## Database relazionali e non relazionali

Con il passare del tempo e il notevole incremento della **diffusione dei dispositivi mobile** (smartphone e tablet in primis), dello sviluppo di **vari Social Network** e delle nuvole di dati (servizi ***cloud***), si è dovuto pensare a un nuovo tipo di database, per accelerare il caricamento dei dati.

Ecco che arrivano i [**database**](http://www.insidevcode.eu/index.php/2015/02/24/parola-del-giorno-database/)**non relazionali**, detti anche **NoSQL**. **Ma cosa sono? Cosa li differenzia da quelli relazionali** (i classici SQL)**? Quali sono i pro e i contro?**

Bene un database può essere visto, in parole povere, come una libreria ikea (quelle quadrate, con tanti box). Dove ogni box è una **tabella**, e in esso sono contenute varie informazioni, detti **record** (righe), **suddivisi da dei campi** (colonne, dettagli del record).  
Questo tipo di struttura, però ha qualche difetto, come quello di avere delle **informazioni spezzettate su più tabelle**, collegate tra di loro. **Questi collegamenti comportano dei rallentamenti**, e delle ripetute richieste di query sql al database. Ma le query non sono solo in uscita, ma anche in entrata, proprio perché anche il semplice inserimento, modifica o eliminazione di un dato, comporterà poi un mix di richieste al **database relazionale**.  
Si chiamano per l’appunto database relazionali, poiché sono dei database, costituiti dalle varie relazioni interne tra le tabelle, come detto prima, collegamenti (le ***join***).

#### ****databasi NoSQL**** (*non relazionali*)

Per prima cosa **la struttura di base è totalmente diversa tra database SQL e NoSQL**. Infatti se nel primo caso il database è costituito da tabelle, **nel secondo caso si tratta di documenti**, dove sono contenute tutte le informazioni richieste, ovvero i dati.  
**Tutti i dati** presenti nel documento (o più documenti) quindi **saranno associati a delle entità, che verranno trattate come oggetti**, e si avranno tutte le informazioni richieste in un colpo solo.

Oggi si può passare da un database relazionale, a uno non relazionale. O per lo meno si può fare in modo che i dati prelevati dalle tabelle SQL, vengano convertiti in un documento tramite codice, in genere **JSON**, e poi richiamati dall’utente.

Riassumendo, quali sono vantaggi e svantaggi dei DB NoSQL quindi?

**I vantaggi sono notevoli, leggerezza di calcolo** (non vengono eseguite operazioni di aggregazione di dati), **non vi sono rischi per l’integrità dei dati**, poiché non essendoci dati spezzettati, i dati si trovano tutti nello stesso posto, e vengono recuperati così come sono. Altro vantaggio è la **scalabilità dei dati in maniera orizzontale**, e non più verticale.

Probabili svantaggi possono essere quelli, dove vi è un progetto già avviato, e quindi bisogna rimettere mano sul codice, per effettuare la conversione.

In genere questo tipo di database viene utilizzato da chi ha delle grossi moli di dati archiviati, su cui lavorare, e deve ripetutamente effettuare richieste al database stesso. Vedesi grosse società gestionali, banche, assicurazioni, o come detto inizialmente i soli **Social Network** come **Facebook**, **Google**, **Twitter**, **Pinterest** e tanti altri.

## Vantaggi Oggetti

* Dato che un elemento contiene tutte le informazioni necessarie **non serve usare** i **dispendiosi** (in termini di performance) **JOIN** come invece avviene per i database relazionali.
* La **semplicità** di questi database è uno degli elementi fondamentali, infatti è possibile **aggiungere nodi a caldo in maniera impercettibile dall’utente finale**.

## Svantaggi oggetti

* La semplicità di questi database, però, porta anche alla **mancanza dei controlli fondamentali sull’integrità dei dati**, il compito ricade quindi totalmente sull’applicativo che dialoga col database che ovviamente dovrebbe essere testato in modo molto approfondito prima di essere messo in produzione.
* La mancanza di uno standard universale (come può essere l’SQL) è un’altra delle pecche di questi database non relazionali, ogni database ha infatti le proprie API e il suo metodo di storing e di accesso ai dati. Detto questo, risulta palese che se lo sviluppo del database sul quale abbiamo basato il nostro applicativo venisse interrotto, il passaggio ad un altro database non sarebbe sicuramente una cosa immediata, ma richiederebbe alcuni cambi più o meno radicali da apportare all’applicativo, è quindi bene tenere in considerazione la cosa al momento del brainstorming iniziale.

### Database Realtime

Un database realtime è un database che utilizza il sistema di realtime processing ,ovvero il fatto di processare i dati in tempo reale, per dei carichi di lavoro il cui stato e in continuo cambiamento. Questo differisce dai database tradizionali che contengono dati permanenti o che cambiano molto di rado.

## Inglese

The Car Finder

My exam project basically consists of an Android app that helps every forgetful person to remember where they parked their car.

My app is based on GPS functionality that nowdays is one of the most used service in the world. It is a [global navigation satellite system](https://en.wikipedia.org/wiki/Satellite_navigation) that provides [geolocation](https://en.wikipedia.org/wiki/Geolocation) and time information to a [GPS receiver](https://en.wikipedia.org/wiki/GPS_receiver) like our android devices. The GPS system provides critical positioning capabilities to military, civil, and commercial users around the world. The United States government created the system, maintains it, and makes it freely accessible to anyone with a GPS receiver.

My app is made by a home where you can find a map,provided by Google, where the users can see the last car parking of their car and with a tap on the car parking marker,the app help the users by showing the route from their actual position to their car on the map.

In the homepage you can also find:

• a spinner with a drop down menu that will list all users' car parkings saved in a database

I have decided to use the google service of Firebase as a database that offers a effective object oriented database that in my opinion will be the future of databases. Firebase also offer an authentication system so the users can log in to my app and save their car parkings in the cloud.

* below the spinner there is a text that show the parking' description
* And below the map there are two buttons:

• a button("ADD") that will make users able to add a car parking giving a name and a description. The parking will be saved in the current user position.

• a button("PATH") that will show to the users the route to the car on the map, and it also show the time and the distance from the selected marker

On the top left of the app there is a button with 3 lines that if clicked it will open a side menu, called drawer, where the users can find their name, email and the account image and other functionality like :

• an option that will expand the map,with car parking markers and route, in fullscreen

• an option that will make users able to edit their car parkings by showing a list of the parkings

* an option called Notes where user can add note about their parking for example in a big super market where there is the number of the parking and the section

• an option that will show the info of the app for example the developers, the version and other legal stuffs

## Inglese

My application has an intuitive and efficient layout because it has only one Activity where inside of it there is a fragment where it’s possible to inflate or put different layout , one layout at time.

The home page is the home fragment and here there are at the top a spinner where the user can select the previous saved parking, bellow there is the description of the parking selected. Then there on the center of the fragment the google map with the preload saved parking retrieved from the database. On the bottom of the fragment there are 2 buttons , one is for showing on the map the path between user’s position and the selected parking, and the other one is for creating new parking by giving a name and a description.

At top of the application there is a blue bar where we can find at the top left corner 3 lines that can be clicked and the app will show the drawer, where the user can find menu links to the other fragment.

There is for example the edit parking where there is a list of the saved parking and the user can tap on the blue pencil to edit the parkings changing for example the title, the description and the coordinates or totally remove it.

There is also the map fragment that is basically the map in fullscreen

A menu for the notes that the user can save to remember something for example when we park in a supermarket’s parking sometimes there are the indications of section and number of parking so here we can save these information when for example there isn’t GPS signal but there is internet connection.

And finally there are the info menu, where there are the application’s info such as version and other stuffs, and log out button that allow user to log out of the application, this makes the application unusable.

## Libro

Il protagonista del romanzo è il giovane **Paul Bäumer** che, sin dai banchi di scuola, viene pesantemente indottrinato dagli insegnanti al **mito della guerra** e ai **valori del nazionalismo**, fondati sull’orgoglio patriottico e sulla superiorità della propria nazione sulle altre. Allo [scoppio del primo conflitto mondiale](http://www.oilproject.org/lezione/attentato-di-sarajevo-francesco-ferdinando-gavrilo-princip-6825.html) nel 1914. Paul e i suoi coetanei (tra cui Kropp, Müller, Westhus, Tjaden) partono **volontari per il fronte**, convinti di vivere una entusiasmante avventura all’insegna degli ideali dell’eroismo e del coraggio. Al fronte conoscono **Stanislaus “Kat” Katczinsky**, un carismatico soldato, più grande di loro, che diventa un punto di riferimento per Paul. Ben presto, il protagonista e i commilitoni si rendono amaramente conto che la realtà della guerra è ben diversa da quella descritta dalla propaganda. La fatica della trincea, la mancanza di cibo, il rischio costante della morte segnano **il passaggio di Paul dall’adolescenza alla vita adulta**: la guerra non è affatto avventurosa ed eroica, ma è solo un tragico ammasso di casualità ed impreparazione: le reclute, male addestrate e lanciate allo sbaraglio in assalti inutili per conquistare risibili metri di campo di battaglia all’avversario, sono quelle che subiscono maggiormente i contraccolpi fisici e psicologici della guerra. L’unico interesse dei soldati, tra le urla di dolore, i topi che infestano le trincee e i continui bombardamenti è **arrivare vivi al giorno successivo** o al nuovo pasto.

La crudele insensatezza della guerra, in cui Paul non vede più né fine né scopi, si manifesta anche nei brevi periodi di congedo e di ritorno alla vita civile, da cui il protagonista si sente ormai del tutto scisso, come se l’orrore della guerra l’avesse **prosciugato di ogni traccia di umanità**. Paul riesce a trovare un’intesa solo con la madre morente, poco prima di tornare al fronte. Qui si susseguono nel frattempo **le morti dei compagni e degli amici**: Westhus è dilaniato da una granata proprio sotto gli occhi di Paul, mentre il soldato Kemmerich muore dopo l’amputazione di una gamba ferita; Müller viene colpito da un razzo e si spegne tra atroci sofferenze. Solo Tjaden sopravviverà alla fine della guerra. Un giorno, durante un’operazione di pattuglia, Paul si rifugia in un fosso, dove si trova di fronte **un soldato nemico**; benché quest’ultimo sia un essere umano come lui, Paul, in un raptus di terrore, lo pugnala, assistendo per ore alla sua agonia. Ferito, Paul è ricoverato in un ospedale con l’amico Albert, che subisce **l’amputazione di una gamba**. Paul, ripresosi, è costretto a salutare l’amico fraterno per tornare al fronte.

Paul vede morire drammaticamente anche il suo modello di riferimento: **“Kat” Katczinsky viene colpito da un colpo d’artiglieria** e, benché il protagonista si prodighi per salvargli la vita trasportandolo in infermeria, Kat vi arriva già morto. Paul perde così ogni desiderio di vivere, tanto che gli risulta del tutto indifferente morire sul campo di battaglia o sopravvivere alle ostilità. Poco prima della [fine della guerra](http://www.oilproject.org/lezione/sintesi-guerra-mondiale-cronologia-triplice-alleanza-triplice-intesa-caporetto-8239.html) nell’autunno del 1918, **Paul è ucciso da una granata** in un giorno in cui, come recita **il bollettino di guerra**, non c’è “niente di nuovo sul fronte occidentale”.