2020

MobDev Projects

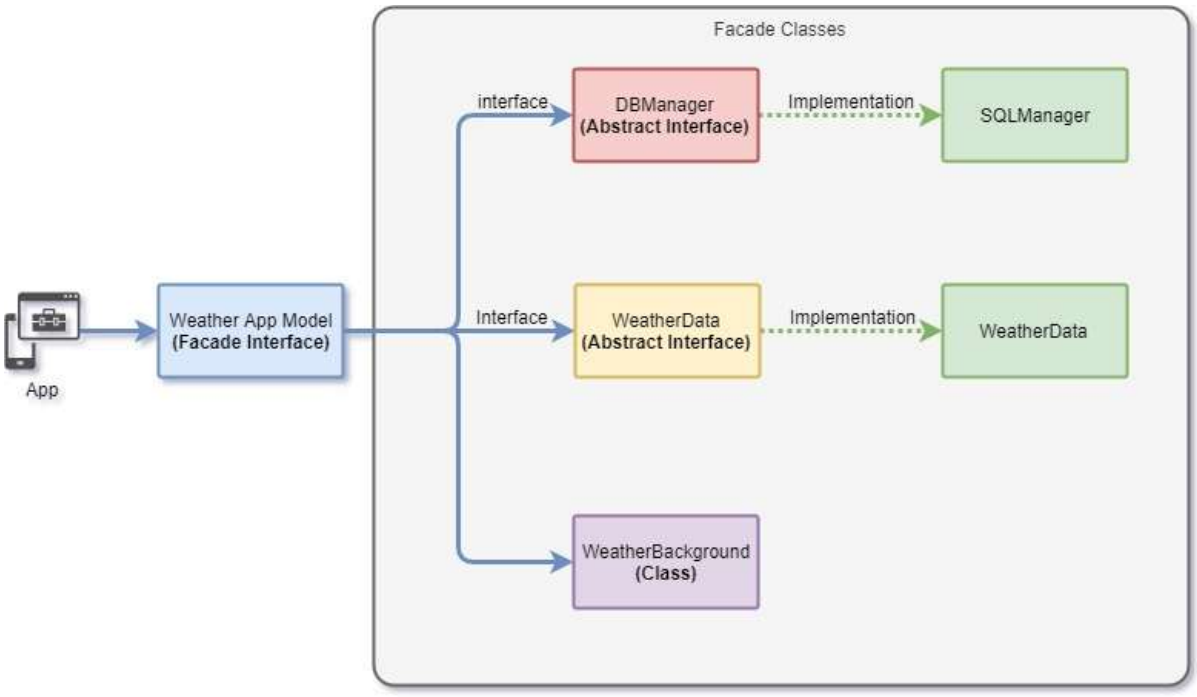
CHRONO TRACKER (ANDROID) WEATHER APP (IOS)

LUCA FAGGION

IOS – WEATHER APP

1.STRUTTURA GENERALE

Come per la piattaforma Android, iOS è basata sul pattern di programmazione **MVC (Model View Controller),** l'app e stata quindi strutturata per adattarsi a questo pattern. Per i dati relativi al meteo sono state utilizzate le **API** di OpenWeatherMap (https://openweathermap.org/api) che provvedono nel fornire i dati sia per il meteo attuale che per previsioni meteo, con previsioni scandite ogni 3 ore per un periodo di 5 giorni. Tutte le query per il ricevimento dati sono fatte utilizzando il networking in modo **asincrono** cosi da garantire una user experience il più fluida possibile anche grazie all'uso di `Skeletal View` che consentono di all'utente di avere transizioni fluide durante il caricamento dei dati. L'app e stata anche strutturata in modo da risultare modulare utilizzando il pattern **Facade** per astrarre le varie implementazioni dei vari componenti chiave dell'app quali il database, la comunicazione con le API (**Networking**), e la gestione delle view in background, la classe WeatherAppModel (**Singleton class**) provvede a questo scopo ed è utilizzata nell'itera app. Per una efficiente gestione della memoria è stato utilizzato il sistema **ARC (Automatic Reference Counting)**



MODELS:

Le classi models sono legate sia allo storage dei dati sia alla rappresentazione in memoria dei dati ricevuti dalle API Meteo tramite networking.

- `CityWeather`**:** questa classe è la rappresentazione in memoria dei dati ricevuti tramite le API meteo si occupa di aggiornare i dati se richiesto e di trasmettere tramite, l'utilizzo del **protocollo** `WeatherModelDelegate,` i dati o eventuali errori durante l'aggiornamento dei dati meteo. La classe utilizza anche il database per recuperare le informazioni statiche, relative alla citta (come nome, posizione, stato, etc..), contenute nel database. **La classe si occupa quindi di aggiornare le informazioni meteo attuali e le previsioni, rappresentate relativamente dalle classi:**
    - **CurrentWeather**: contiene le informazioni meteo attuali

- ForecastWeather: contiene un array di classi **CurrentWeather** che contengono informazioni sul meteo ad intervalli di 3 ore ognuna

```
@interface CityWeather : NSObject

@property (nonatomic,strong)NSNumber * ID;
@property (nonatomic,strong)NSString* name;
@property (nonatomic,strong)NSString* country;
@property (nonatomic,strong)NSNumber* lon;
@property (nonatomic,strong)NSNumber* lat;
@property (nonatomic,strong)NSDate* last_updated;
@property Boolean hasData;
@property (nonatomic,strong)CurrentWeather* current;
@property (nonatomic,strong)ForecastWeather* forecast;

@property NSObject<WeatherModelDelegate>* delegate;

-(instancetype) initWithCityID:(NSNumber*)city_id;
-(instancetype) initWithCityID:(NSNumber*)city_id update:(Boolean)update;
-(void) performUpdate;

@end
```

- **WeatherHistoryEntry:** classe che rappresenta dati storici di uno specifico tempo e condizione meteo associati ad una città è utilizzata per rappresentare questi dati che provengono dal database

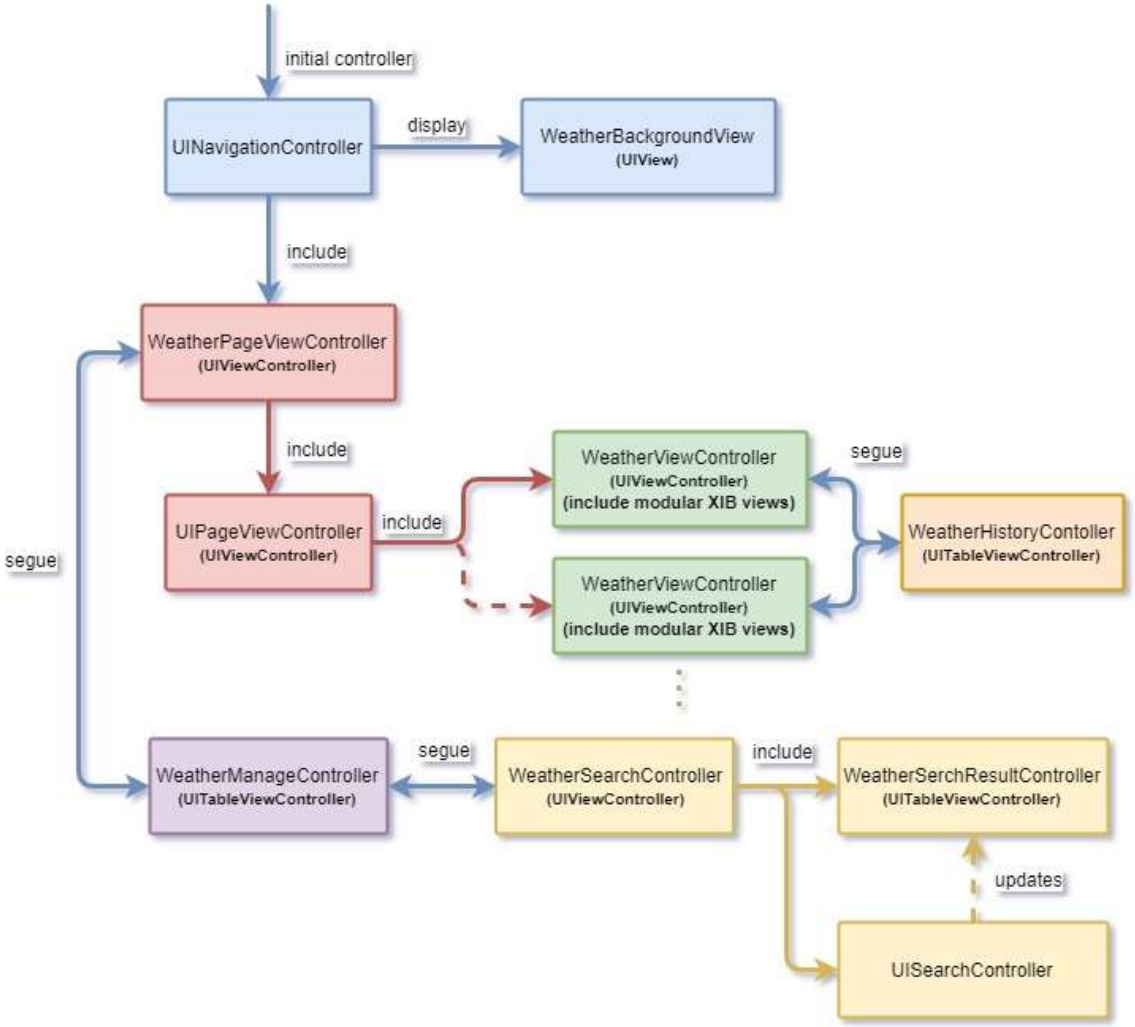
CONTROLLERS:

I controllers in iOS sono la parte centrale dell'applicazione che consente di gestire gli input dell'utente e di processare i dati provenienti dai models e provvedere a inviarli alle view per presentarli.In iOS ogni controller ha associato di base una view vuota che funge da contenitore per tutte le subview che il controller gestisce e o crea, si viene quindi a formare una struttura ad albero che rappresentano astrattamente le relazioni padre figlio di ogni view, e siccome un controller può essere padre di un altro controller (ovvero **Controllers di View Controllers**) le interfacce posso diventare molto complicate ma in questo modo e possibile un alta personalizzazione e flessibilità di esse.

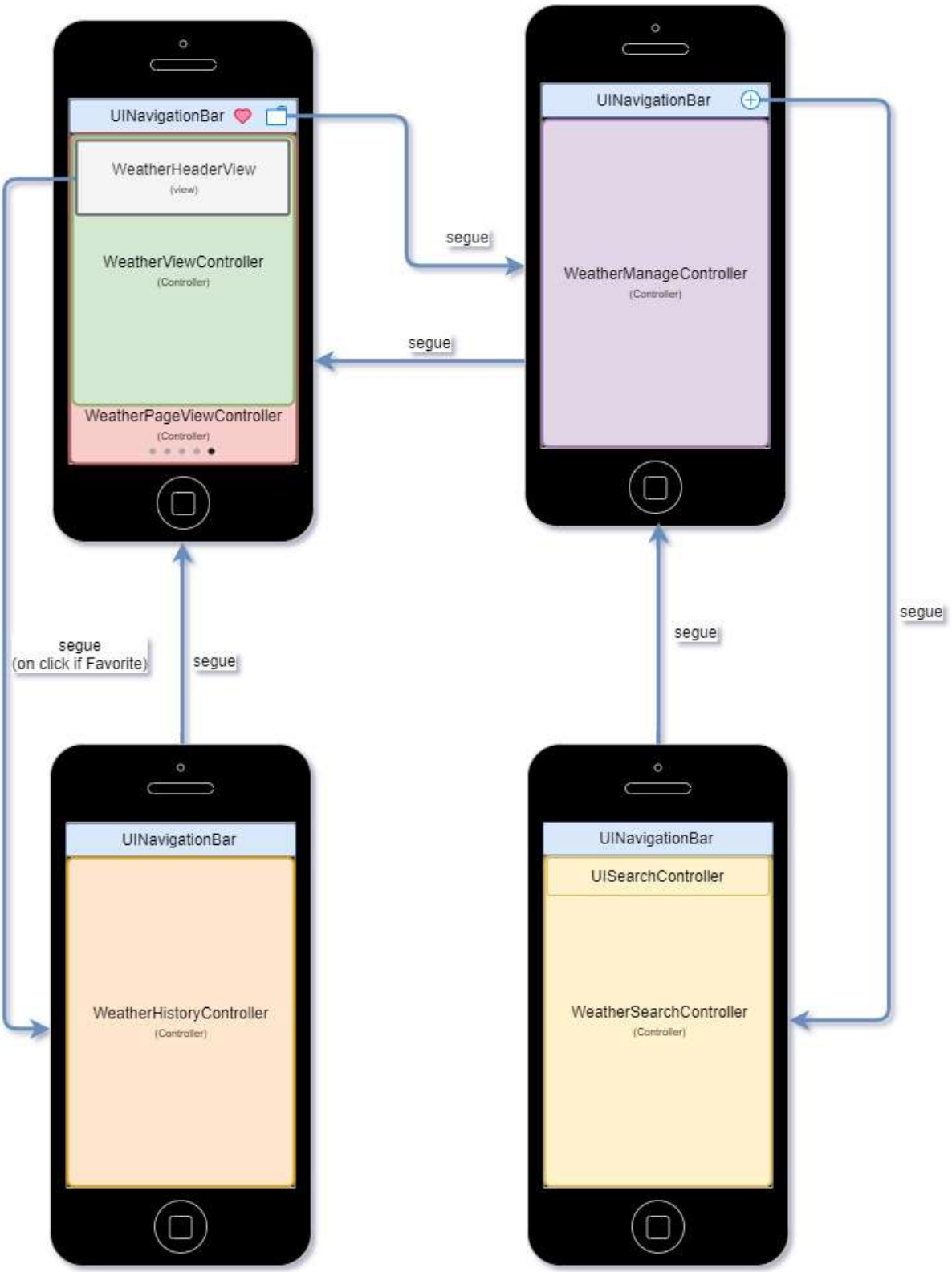
- **WeatherPageViewController:** (UIViewController) gestisce la principale schermata dell'app consente tramite un **UIPageViewController** di gestire con le gestures la navigazione tra i vari WeatherViewControllers che visualizzano ognuno i dati meteo relativi alle città scelte dall'utente.
- **WeatherViewController:** (UIViewController) gestisce molteplici view modulari (XIB) in base o meno alla presenza nei dati di certi eventi atmosferici e il componente principale dell'app. Notifica il WeatherBackgroundView di quale preset mostrare in base all'evento meteorologico attuale della citta visualizzata
- **WeatherManageContoller:** (UITableViewController) gestisce l'aggiunta e rimozione delle città, consente di salvare le città preferite su un file sul dispositivo
- **WeatherSearchController:** (UIViewController) gestisce la ricerca tramite due controller **UISearchController** per gestire l'input dell'utente e creare le query per il database e **WeatherSearchResultController** (UITableViewController) per mostrare i risultati del database, da

notare che le query eseguite sul database completamente asincrone per avere una UI responsiva e fluida.

VIEW HIERARCHY



APP FLOW


UINavigationBar
WeatherHeaderView
(view)
WeatherViewController
(Controller)
WeatherPageViewController
(Controller)
segue
segue
UINavigationBar
WeatherManageController
(Controller)
segue
segue
(on click if Favorite)
segue
segue
UINavigationBar
WeatherHistoryController
(Controller)
UINavigationBar
UISearchController
WeatherSearchController
(Controller)

COMUNICAZIONE TRA CONTROLLERS

La comunicazione tra controllers avviene tramite il passaggio di dati nella funzione

Utilizzata per recuperare un segue tramite il proprio ID e in base a quello e possibile identificare il prossimo controller che verrà visualizzato e quindi passagli i dati necessari. Nell' app è utilizzato questo metodo non solo per passare dati ma per registrare tramite l'utilizzo di protocolli anche delegati utilizzati per ottenere una connessione tra due view controllers come nel caso del passaggio dal **WeatherPageViewController** al **WeatherManageContoller**:

```
#pragma mark - Navigation
-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{
   if([[segue identifier] isEqualToString:@"goToManage"]){
      WeatherManageController* manageController = (WeatherManageController*)[segue
destinationViewController];
      manageController.delegate = self;
   }
}
```

In questo caso il **WeatherPageViewController** viene registrato come delegato nel WeatherManageContoller per essere notificato di quando una citta viene aggiunta o rimossa.

Questo è il protocollo al quale la classe **WeatherPageViewController** si conforma:

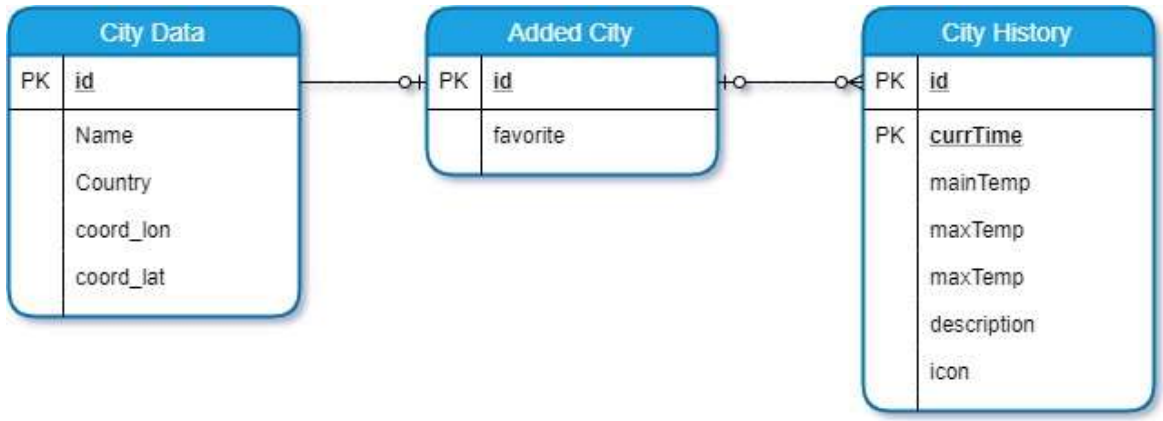
```
@protocol WeatherManageDelegate <NSObject>

-(void) onDeleteCityAtIndex:(NSInteger)index;
-(void) onAddCity:(CityWeather*)data;
-(NSArray*) getCities;
-(NSArray<CityWeather*>*) getCitiesWeather;
@end
```

2.DATABASE

Il database utilizzato e SQLite e la sua implementazione è descritta nella classe **SQLManager.** Il database per questa app e molto semplice e il suo scopo principale e tenere cache della lista della citta per poter eseguire query di ricerca in modo più efficace se si fosse utilizzato il **Networking tramite le API** per gestire le richieste di ricerca oltre che a una più lenta responsività dell'app si sarebbe certamente incorsi in un superamento delle richieste al minuto poste dalle API (60 cpm) siccome sarebbe stato l'utente durante la digitazione del nome della citta a creare di volta in volta richieste.

SCHEMA DATABASE



3.NETWORKING

La tipologia di app la rende pesantemente basata sull'utilizzo di API esterne con richiesta tramite **HTTP** (**HyperText Trasfer Protocol**) per il recupero dei dati metereologici. Nell'app la classe responsabile di gestire queste richieste e restituire i dati è la classe **WeatherData** e protocollo **WeatherModelDelegate** per la gestione asincrona degli eventi per la UI.

```
-(void) getCityCurrentWeatherbyId:(NSNumber*)city_id withSelector:(SEL)selector
ofObject:(id)object;{
   [[[NSURLSession sharedSession] dataTaskWithRequest:[self
setUpRequestAPI:@"weather?id=%@",city_id]
         completionHandler:^(NSData *data,NSURLResponse *response,NSError *error) {
            [self callCallback:selector ofObject:object withData:data];
         }] resume];
}
```

Per questa classe (a solo scopo di sperimentazione) sono stato usati i callback tramite **selectors,**callback tramite l'utilizzo dei **blocks** sono stati utilizzati nella classe **AnimatedBackground** e la categoria che la classe utilizza come input di dati **NSValue+AnimBackgroundData**, qui e mostrata la funzione helper che invoca il selector.

```
-(void) callCallback:(SEL)selector ofObject:(id)object withData:(NSData *)data{
   NSInvocation *inv = [NSInvocation invocationWithMethodSignature:[object
methodSignatureForSelector:selector]];
   [inv setSelector:selector];
   [inv setTarget:object];
   [inv setArgument:&data atIndex:2]; //gli argomenti 0 e 1 sono rispettivamente self e
_cmd, settati automaticamente da NSInvocation
   [inv invoke];
}
```

AGGIORNAMENTO ASINCRONO INTERFACCIA UTENTE

Ogni **WeatherViewController** detiene un riferimento alla classe **CityWeather** che viene utilizzata per ricevere gli aggiornamenti sia iniziati dall'utente sia quelli stabiliti dall'applicazione. Ogni **WeatherViewController** si registra come delegate della propria classe **CityWeather** in questo modo tramite l'implementazione del protocollo `WeatherModelDelegate` e possibile conosce quanto si e iniziato l'aggiornamento dei dati durante inizio dell'aggiornamento i corrente visualizzato **WeatherViewController** informa tutte le proprie views che un aggiornamento e stato iniziato e che quindi i dati che stanno attualmente mostrati sono probabilmente non attuali per creare una transizione non distruttiva e far capire all'utente che un aggiornamento sta venendo eseguito vengono mostrati al posto delle view le **SkeletalView** come mostrato nell'immagine qui accanto, ogni view gestisce le proprie skeletal views, che vengono tolte quando i dati sono nuovamente disponibili. La stessa cosa viene impiegata nella gestione della **WeatherBackgroundView** in questo caso la classe **AnimatedBackground** conensete di eseguire transizioni da un preset all'atro in modo non distruttivo. Alla fine dell'aggiornamento la classe **WeatherViewController** e nuovamente notificata che i dati sono disponibili oppure che e stato rilevato qualche errore e di conseguenza la classe informa tutte le sue views di aggiornarsi oppure di ritornare a una versione in memoria dei dai in caso di errore.