joobto

Time Tracking Enhanced

di

Majid Abdul - 5^AI
Andrea Antonioni - 5^AI
Nipuna Perera - 5^CI

Esame di Stato - anno scolastico 2015/16

in collaborazione con

MELACOM
>> Value created and delivered!

Premessa

La decisione di sviluppare questo argomento come Tesina d'Esame nasce dopo uno stage svolto durante l'anno scolastico 2014/15 presso l'azienda MELACOM di Gussago.
Il signor Massimiliano Birbes, fondatore e proprietario di MELACOM, ci ha commissionato un progetto chiamato "Joobto" con l'obiettivo iniziale di sviluppare un'applicazione Android.

Il team di lavoro, composto inizialmente da Majid e Nipuna, ha realizzato una prima versione dell'applicazione Android dopo circa tre mesi di lavoro. Successivamente si è avviato lo sviluppo della web app che è stato completato due mesi dopo.

A febbraio 2016 Andrea si è unito al team con lo scopo di sviluppare l'applicazione per iOS.

Il progetto "Joobto" è stato interamente realizzato da noi ragazzi durante il periodo scolastico. I tempi sono stati sì lunghi, ma ci hanno consentito di apprendere ed applicare nuovi linguaggi di programmazione e tecniche di sviluppo che altrimenti non saremmo mai stati in grado di conoscere.

Indice

INTRODUZIONE **6**
PUNTI DI FORZA 6
FASI DI PROGETTAZIONE *(a cura di Majid Abdul)* **7**
ANALISI FUNZIONALE 7
QR CODE 7
Struttura *7*
Versione *8*
Error Correction *8*
DATI 9
Azienda *9*
Dipendente *9*
Location *10*
Timbro *10*
BACK-END *(a cura di Majid Abdul)* **11**
RESTFUL API STRUCTURE 11
PHP 14
DISTANZA TRA DUE COORDINATE 14
SLIM FRAMEWORK 16
URL Structure *16*
FRONT-END *(a cura di Majid Abdul)* **18**
ANGULAR JS 18
Single Page Application *18*
Model View Controller *18*
Direttive *19*
Two-way Data Binding *20*
ANDROID *(a cura di Majid Abdul e Nipuna Perera)* **21**
APPLICATION STRUCTURE 21
Attività (activity) *21*
Frammento (fragment) *21*
SERVIZI 21
Location *21*
QR Code Scanner *22*
ACCESSO API 23
IOS *(a cura di Andrea Antonioni)* **24**
SWIFT 24
Gestione del tempo (NSDate) *25*
DESIGN PATTERN 26

Factory Method *27*
Observer *27*
Delegate *28*
SERVIZI 29
Location *30*
Notifiche *31*
Framework *32*
SALVATAGGIO DATI 33
UX 34
UI *35*
Information Architecture *38*
Design Research *39*
CONCLUSIONI **40**
SITOGRAFIA **41**

Introduzione

Joobto nasce per rispondere alle esigenze di tutte le aziende che hanno la necessità di un sistema di rilevazioni delle presenze per i propri dipendenti. Si tratta di un servizio in grado di registrare e gestire l'entrata e l'uscita dei dipendenti dalle varie sedi di lavoro.

Joobto è un'applicazione cloud based che restituisce informazioni organizzate ovunque ci si trovi ed in ogni momento. Elimina ogni componente fisica ed usa le tecnologie che si hanno già in tasca: lo smartphone.
Riduce i costi di gestione di oltre il 50% rispetto all'uso delle attuali tecnologie alternative.
L'applicazione risolve il problema del time-tracking dei dipendenti per ogni tipologia di impresa, in modo immediato e user-friendly.

Ogni dipendente registra la propria presenza usando uno smartphone connesso ad Internet, in modo che l'ufficio del personale verifichi l'ingresso dei dipendenti tramite un browser.

Punti di forza

- Joobto, al costo di € 3.99 al mese a dipendente, offre alle aziende un servizio sicuro, senza costi di manutenzione, e pratico da usare, con l'eliminazione di badge e tessere cartacee, che utilizza semplicemente uno smartphone;
- Joobto geo-localizza i dipendenti in modo da verificarne l'effettiva presenza sul luogo di lavoro, senza invaderne la privacy;
- Joobto è completamente cloud based e eliminando la parte cartacea della contabilità permette a chi deve eseguirla di diminuire considerevolmente i tempi di redazione (e di conseguenza i costi per l'azienda).

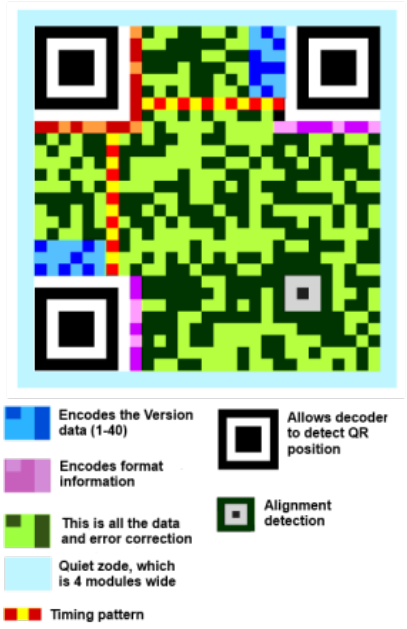
Fasi di progettazione

Analisi Funzionale

L'obiettivo principale di questo progetto era quello di modernizzare gli attuali sistemi di conteggio-ore già presenti nelle aziende, in modo da poterli sostituire, oppure rappresentare per le nuove aziende una soluzione alternativa e più efficiente sia dal punto di vista dell'utilizzo che dal punto di vista economico.
Gli attuali sistemi prevedono la presenza di macchinette elettroniche e dei badge magnetici forniti ai dipendenti, ovviamente questi sistemi hanno costi più elevati sia in termini di spesa dei componenti che in termini di manutenzione. Inoltre questi sistemi non sono abbastanza flessibili, per permettere l'utilizzo in ambienti di lavoro dove il dipendente si ritrova in posizioni differenti durante l'arco della settimana (es: Impresa pulizie).
Dunque per garantire la presenza di un lavoratore in un determinato punto geografico si è scelto di utilizzare dei **QR Code** stampati su dei fogli di carta ed allo stesso tempo georeferenziati. Per ottimizzare i costi si è scelto di fornire al lavoratore un'applicazione che gli permetta di poter "timbrare" tramite l'utilizzo del suo smartphone personale, visto il clamoroso successo di questi dispositivi che ormai sono diffusi tra tutti gli strati della popolazione. E si può dire con quasi assoluta certezza che arriverà un giorno in cui ogni persona ne possiederà uno.

QR Code

Il QR Code è un codice a barre; sostanzialmente è stato creato per superare i limiti che erano posti da un codice a barre standard (come quelli che troviamo sul retro dei prodotti in un supermercato). Un QR Code è formato da una matrice di moduli di colore nero e bianco all'interno di una forma quadrata che rappresentano l'informazione codificata.
Il nome è l'abbreviazione di *quick response code* proprio perchè è stato creato per una rapida decodifica del suo contenuto.



Struttura

Il QR Code è formato da 3 position detection patterns visibili nella foto seguente grazie ai quali il codice può essere letto da qualsiasi direzione in 360°; inoltre anche la velocità di lettura è significativamente più veloce, riducendo le interferenze create dal background al momento della scannerizzazione.

Se per esempio un codice contiene l'url di un sito web, la presenza del formato dell'informazione codificata permette allo scanner di interpretare immediatamente l'informazione

letta; in questo caso uno smartphone potrebbe aprire un browser mostrando all'utente il sito web.

Versione

Ogni QR Code può essere codificato utilizzando versioni diverse. Le versioni vanno nel range da 1 a 40, all'aumentare della versione aumenta oltre alla capacità di dati che si possono codificare anche il numero dei moduli.
La seguente tabella descrive le specifiche tecniche della versione 1 e 40.

Versione	**Moduli**	**Numeri**	**Caratteri alfanumerici**
1	21 x 21	41	25
40	171 x 171	7089	4296

In base alle esigenze e alla quantità di informazione da codificare si sceglie la versione da utilizzare. Per Joobto è stata utilizzata la versione 2 per la quale si possono codificare al massimo 47 caratteri alfanumerici.

Error Correction

Il QR Code essendo un'immagine può subire dei danneggiamenti come lo scolorimento oppure, essendo una stampa, può sporcarsi. Per risolvere questo problema esiste la possibilità di applicare al momento della codifica un livello di correzione.
Esistono 4 diversi livelli di correzione che possono essere applicati ad ogni versione di QR Code:

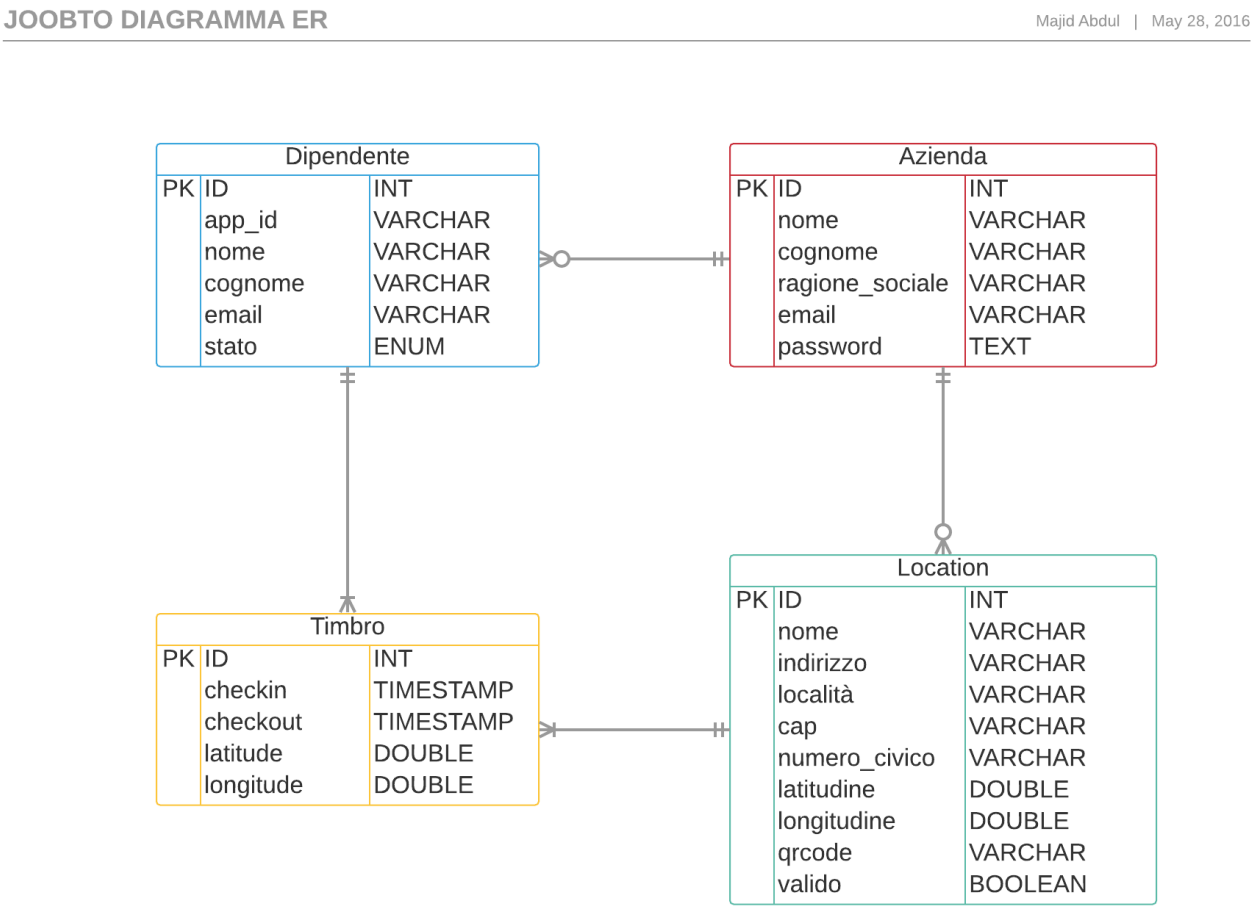
- **Livello L: 7%** dell'informazione codificata può essere ripristinata
- **Livello M: 15%** dell'informazione codificata può essere ripristinata
- **Livello Q: 25%** dell'informazione codificata può essere ripristinata
- **Livello H: 30%** dell'informazione codificata può essere ripristinata

L'applicazione del livello di correzione riduce la capacità di dati che una versione di QR Code può codificare, più si alza il livello di correzione più si abbassa la quantità di informazione memorizzata.

Per Joobto, avendo ambienti di utilizzo di ogni genere, si è scelto di utilizzare il livello di correzione Q.

Infine l'utilizzo della versione 2 e il livello di correzione Q permettono la codifica di 29 caratteri. Quindi vengono generate per ogni location stringhe di lunghezza fissa (20 caratteri) da codificare all'interno di un QR Code che successivamente potrà essere stampato ed appeso all'interno del posto di lavoro dove i dipendenti potranno scansionarlo tramite l'applicazione per registrare la loro presenza.

Dati



Azienda

L'entità azienda contiene le credenziali di accesso al sistema da parte dell'azienda che acquista Joobto. Le password sono salvate solo in formato hash, generato dall'algoritmo di hashing **sha256**.

Dipendente

Ogni azienda può creare N dipendenti: al momento della creazione ad essi viene associato un app_id, che viene inviato tramite mail al dipendente, da utilizzare per accedere all'applicazione mobile. Si è ritenuto non necessario l'utilizzo anche di una password di accesso poiché è stato inserito lo *stato* che può assumere solo i seguenti valori:

- ***logged:*** Il dipendente ha effettuato l'accesso sull'applicazione
- ***not logged:*** Il dipendente può effettuare l'accesso
- ***blocked:*** Il dipendente ha effettuato un logout

Grazie a questa approccio si può garantire che il dipendente non possa "imbrogliare" chiedendo al collega di effettuare un timbro per lui, poiché al primo tentativo di successo del login lo stato del suo app_id viene impostato a *logged*, dopodiché nessun'altra persona può

effettuare l'accesso all'applicazione utilizzando quel codice. In caso di logout da parte dell'utente il sistema blocca l'app_id che può essere nuovamente utilizzato solo tramite l'intervento del manager, che dovrà modificare lo stato del dipendente in questione loggandosi sulla webapp.

Bisogna comunque precisare che in realtà si può imbrogliare, basta portarsi dietro lo smartphone di un collega ed effettuare il timbro per lui. Ma lo scenario è meno probabile rispetto al sistema tradizionale, dove uno scambio di badge diventa una cosa più fattibile rispetto allo scambio del proprio smartphone personale. Inoltre una volta effettuato il timbro il dipendente compare immediatamente nella dashboard della webapp, e di conseguenza il datore di lavoro ha una disponibilità in tempo reale dei dati. Ciò scoraggia ancora di più i dipendenti a barare.

Location

Ogni azienda può inoltre registrare N posizioni georeferenziate attraverso i QR Code, i quali vengono generati automaticamente.

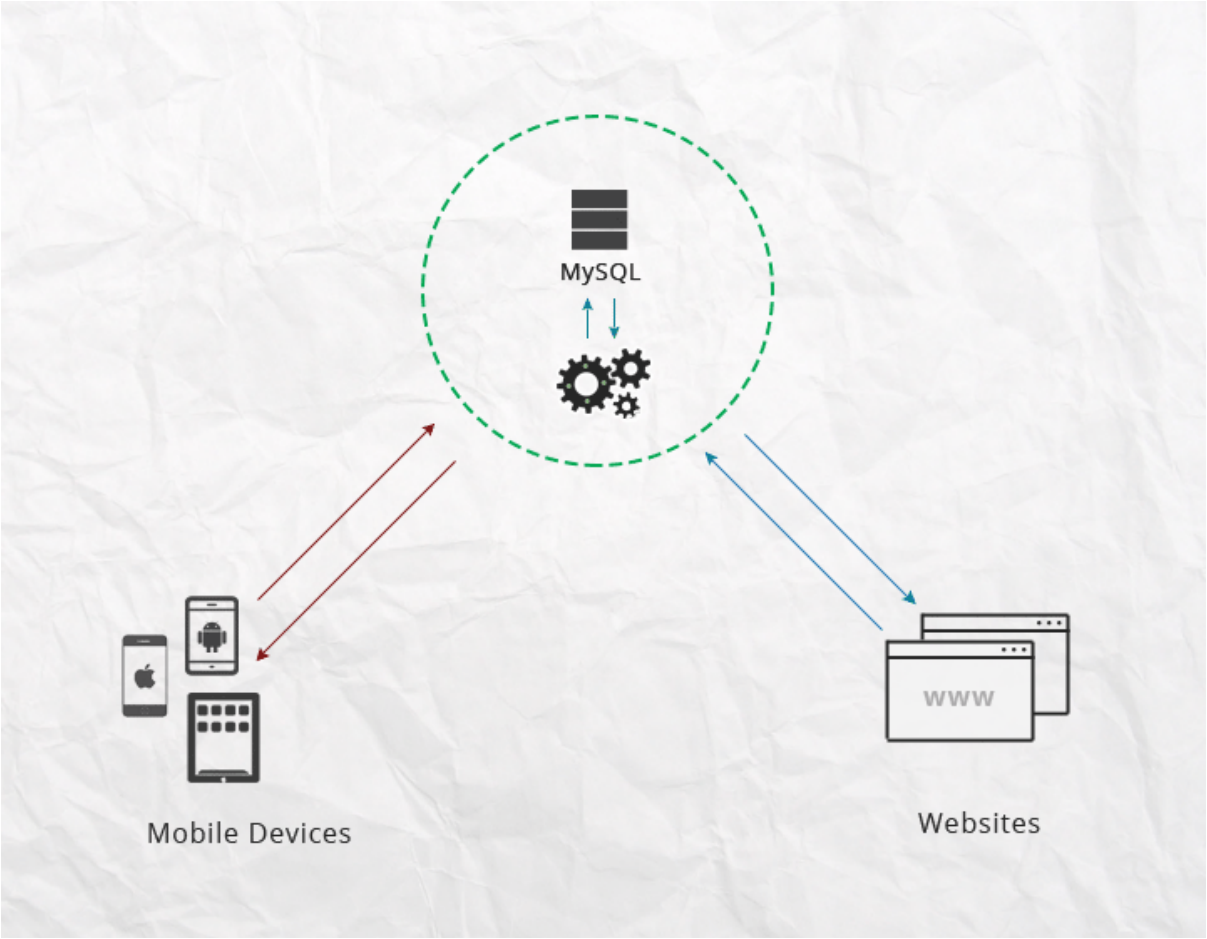
Timbro

Al momento del timbro da parte del dipendente vengono salvate le coordinate geografiche trasmesse dall'applicazione e viene registrato l'orario di entrata. L'orario di uscita viene messo a *null* finché il dipendente non effettua l'uscita.

Back-end

RESTful API Structure

Per Joobto era necessario non solo sviluppare un'applicazione per smartphone, ma anche fornire all'azienda una piattaforma web in grado di gestire i propri dipendenti e le location.
Di conseguenza si è scelto di strutturare l'applicazione con l'architettura REST (Representational State Transfer), avendo un database centrale che gestisce tutta la base di dati e li espone per l'utilizzo tramite delle API.
Il seguente schema illustra il funzionamento dell'architettura:



L'implementazione di REST è molto semplice: si basa principalmente sul protocollo HTTP e usa i suoi metodi in base all'operazione che si intende eseguire. La seguente tabella illustra i verbi più utilizzati:

GET	Richiesta di una risorsa
POST	Creazione nuova risorsa
PUT	Modifica di una risorsa esistente

DELETE	Cancellazione di una risorsa

Seguendo questa struttura sono state create le API di Joobto. Di seguito è riportato l'endpoint check-in che permette di effettuare il check-in di un dipendente in un posto di lavoro:

POST: /checkin (Auth)

```
{
    "qrcode":"YagT1GIhFaQCDJcXlrls",
    "latitude":"45.555794050807",
    "longitude":"10.216270245508"
}
```

I parametri richiesti sono:

- qrcode: codice scannerizzato dallo smartphone
- latitude: latitudine ottenuta tramite gps
- longitude: longitudine ottenuta tramite gps

Nel caso in cui la richiesta non vada a buon fine, ecco i vari tipi di errori:

PARAMETRI RICHIESTI MANCANTI

statuscode: **400** *Lista di parametri che mancano, o non formattati correttamente*

```
{
  "error": true,
  "message": "Required field(s) app_id is missing or empty"
}
```

NON AUTENTICATO

statuscode: **401** *Utente non loggato che ha richiesto una risorsa protetta*

```
{
  "error": true,
  "message": "Access denied."
}
```

ERRORE DI CONNESSIONE

statuscode: 200 *Nel caso si siano verificati problemi durante l'elaborazione della richiesta*

```
{
  "error": true,
  "status": 14,
  "message": "An error occurred. Try later."
}
```

Queste invece sono le risposte in caso di successo della richiesta:

status code: 200

Il dipendente ha già un timbro attivo sul sistema, di conseguenza non ne può effettuare un altro finchè non è stato chiuso il precedente.

```
{
  "error": true,
  "status": 24,
  "message": "Employee already checked in."
}
```

Il QR CODE scannerizzato non è stato riconosciuto dal sistema.

```
{
  "error": true,
  "status": 20,
  "message": "Qrcode not recognized."
}
```

Il dipendente sta cercando di effettuare il checkin in una location dove non è autorizzato a farlo.

```
{
  "error": true,
  "status": 19,
  "message": "This location does not belong to you!"
}
```

```
Il dipendente non si trova sul posto di lavoro, viene inviata la distanza in metri
dell'utente dal posto in cui si sta cercando di effettuare il check-in.

{
  "distance": 50852.390065978,
  "error": true,
  "status": 18,
  "message": "Employee not on location"
}


Il dipendente ha effettuato con successo il checkin, vengono inviati data ed ora in
formato timestamp di mysql con timezone UTC/GMT.

{
  "distance": 0,
  "error": false,
  "status": 8,
  "message": "Checkin successfull",
  "id_timbro": "43",
  "checkin_time": "2016-02-22 09:51:10"
}
```

PHP

PHP è un linguaggio di scripting *server side* che, in fase di esecuzione, interpreta le informazioni ricevute da un client (request) grazie al Web server, le elabora e restituisce un risultato al client che ha formulato la richiesta (response). Generalmente questo linguaggio viene molto usato per la generazione di pagine web dinamiche, ma questo tipo di approccio non rispetta l'architettura di tipo REST che è stata scelta per lo sviluppo del back-end di Joobto. Essendo comunque un linguaggio facile da imparare e con una vasta comunità di developers, lo si è scelto per sviluppare le API di Joobto. Sostanzialmente PHP collega insieme i dati che risiedono su un server MySql e in base alle richieste dei client li restituisce in un formato predefinito, in modo che il client possa "farne qualcosa".

Distanza tra due coordinate

In Joobto abbiamo avuto la necessità di calcolare la distanza tra due coordinate geografiche per poter determinare la distanza del dipendente dal luogo in cui egli sta cercando di effettuare il check in. Tra le possibili scelte questo calcolo poteva essere effettuato direttamente lato client oppure lato server, con i relativi vantaggi e svantaggi. Si è scelto di farlo lato back-end poichè in questo modo si poteva offrire la possibilità all'azienda di impostare un raggio di distanza a scelta, entro il quale il timbro del dipendente veniva accettato.
Esistono due famosi algoritmi per il calcolo della distanza tra due coordinate geografiche:

- Formula di Haversine

- Formula di Vincenty

Si è scelto di utilizzare la formula di Vincenty poiché risulta più precisa. Una possibile implementazione della formula per il calcolo della distanza in metri tra due coordinate geografiche nel linguaggio PHP è la seguente:

```php
/**
 * Calculates the great-circle distance between two points, with
 * the Vincenty formula.
 * @param float $latitudeFrom Latitude of start point in [deg decimal]
 * @param float $longitudeFrom Longitude of start point in [deg decimal]
 * @param float $latitudeTo Latitude of target point in [deg decimal]
 * @param float $longitudeTo Longitude of target point in [deg decimal]
 * @param float $earthRadius Mean earth radius in [m]
 * @return float Distance between points in [m] (same as earthRadius)
 */
public static function vincentyGreatCircleDistance(
  $latitudeFrom, $longitudeFrom, $latitudeTo, $longitudeTo, $earthRadius = 6371000)
{
  // convert from degrees to radians
  $latFrom = deg2rad($latitudeFrom);
  $lonFrom = deg2rad($longitudeFrom);
  $latTo = deg2rad($latitudeTo);
  $lonTo = deg2rad($longitudeTo);

  $lonDelta = $lonTo - $lonFrom;
  $a = pow(cos($latTo) * sin($lonDelta), 2) +
    pow(cos($latFrom) * sin($latTo) - sin($latFrom) * cos($latTo) * cos($lonDelta), 2);
  $b = sin($latFrom) * sin($latTo) + cos($latFrom) * cos($latTo) * cos($lonDelta);
```

```
  $angle = atan2(sqrt($a), $b);

  return $angle * $earthRadius;

}
```

Slim Framework

Invece di sviluppare un framework REST da zero, è sempre meglio appoggiarsi su framework e librerie esistenti e utilizzati da altri developers. Slim è un micro framework per PHP che rende più veloce e facile la scrittura di applicazioni web o API. Inoltre ha una serie di caratteristiche che si adattano molto bene alle esigenze di Joobto, tra le quali:

- E' molto leggero, pulito e facile da imparare
- Supporta tutti i metodi HTTP come GET, POST, PUT e DELETE necessari per lo sviluppo di una REST API
- Fornisce una *middleware architecture layer* per filtrare le richieste, nel nostro caso è stato utilizzato per autenticare l'accesso alle risorse protette

Per far capire la semplicità d'utilizzo del framework riportiamo l'esempio di uno snippet di codice che rappresenta una ROUTE sul server.

```
<?php
require 'vendor/autoload.php';

$app = new \Slim\App;
$app->get('/hello/:message, function ($variable) {
    echo $variable . 'world';
});
$app->run();
```

Ora basta chiamare con un qualsiasi client HTTP (es: browser) l'url dell'endpoint, per esempio: `http://indirizzoserver/api/hello/ciao` e la risposta che riceveremo sarà *'ciao world'.*

URL Structure

Nello sviluppo di una API che poi deve essere usata da altri sviluppatori è molto importante strutturare bene gli URL che rappresentano gli endpoint. Essi devono essere semplici e formattati bene ed anche facili da leggere. Ogni URL deve identificare univocamente una risorsa: le API di Joobto sono state costruite seguendo tutti questi criteri.

Per quanto riguarda le versioni delle API esse possono essere mantenute nell'url, oppure direttamente nell'Header delle richieste HTTP. Per Joobto è stato scelto di mantenere la versione direttamente nell'url, poichè questo approccio diventa molto conveniente quando il client deve migrare da una versione all'altra. Di seguito è riportata la URL structure di alcuni endpoint delle API di Joobto:

BASE URL: https://joobto.me/joobtorest/v1/index.php

URL	Metodo	Parametri	Descrizione
/register	POST	email, password, ecc..	Registrazione di una nuova azienda
/login	POST	email, password	Login azienda
/loginapp	POST	appid	Login dipendente
/records/:datefrom/:dateto	GET		Lista checkin nel range della data specificata
/record/:id	GET		Prende il checkin con l'id specificato
/record/:id	PUT		Aggiorna checkin
/employee	GET		Lista di tutti gli employees dell'azienda
/employee/:id	GET		Prende l'employee con l'id specificato
/employee	POST	nome, cognome, email, appid	Crea un nuovo employe
/employee/:id	PUT		Aggiorna dati di un employee
/employee/:id	DELETE		Cancella l'employee con l'id specificato

Seguono la stessa struttura di employee gli endpoints per la risorsa location, mentre gli endpoints /checkin e /checkout sono specifici per effettuare il checkin e checkout di un dipendente. Invece non esiste la possibilità di cancellare dal server un timbro: il motivo di questa scelta verrà spiegato successivamente.

Si è scelto, sia per il body delle richieste che per le risposte, il formato **JSON** (Javascript object notation) poichè è molto semplice da usare e per quasi tutti i linguaggi di programmazione esistono librerie in grado di gestire array ed oggetti json.

Front-end

Generalmente con front-end si intende la parte dell'applicazione web con cui l'utente interagisce. Joobto doveva fornire alle aziende un'interfaccia comoda e innovativa per gestire i propri dipendenti, le proprie locations, visualizzare i timbri, effettuare ricerche con filtri e altro. Abbiamo deciso dunque di sviluppare il front-end come una web application.
HTML è il linguaggio in cui sono scritte tutte le pagine web che noi vediamo su internet, solo che è stato pensato per creare pagine web statiche, quindi non si adatta bene alle esigenze di una webapp. Proprio per questo motivo è quasi d'obbligo appoggiarsi su framework Javascript, che permettono di aggiungere tutte quella funzionalità dinamiche di cui si ha bisogno all'interno di un sito web. Le webapp sono comunque novità recenti e fino a qualche anno fa non esistevano proprio; inoltre le tecnologie web negli ultimi anni sono in continua evoluzione e continuano a cambiare in tempi brevi. Di conseguenza scegliere che cosa usare è stato abbastanza difficile.

Angular JS

Angular JS è un framework Javascript per lo sviluppo di applicazioni web che ha riscosso un notevole successo, (pur essendo un progetto relativamente giovane: la versione 1.0 è stata rilasciata nel 2012), dovuto al suo approccio e all'infrastruttura fornita, che incoraggia l'organizzazione del codice e la separazione dei compiti nei vari componenti (logica separata dalla grafica).
Ricapitolando, Angular trasforma l'HTML da linguaggio per descrivere documenti in un linguaggio per descrivere interfacce grafiche per le applicazioni Web.
Oltre al fatto che sia un framework moderno, esso possiede molte caratteristiche che ci hanno spinto ad usarlo, poiché semplifica molto la scrittura di codice; eccone alcune:

- Single Page Application
- *Two-way* Data Binding
- Dependency Injection
- Supporto MVC
- Riusabilità dei componenti

Single Page Application

Nella consultazione di un sito web, tutti noi avremo notato che quando si passa da una sezione all'altra dello stesso sito, utilizzando per esempio il menù, la pagina viene completamente ricaricata e nel frattempo l'utente deve aspettare qualche secondo. Con Angular è stato introdotto il concetto di Single Page Application, che elimina il concetto di reload totale della pagina ma carica i contenuti dinamicamente, man mano che l'utente li richiede. In questo modo non solo il sito risulta più interattivo e veloce, ma viene ottimizzato anche l'uso della quantità di dati.

Model View Controller

Il Model-View-Controller (MVC) è un pattern architetturale utilizzato nello sviluppo di software ed applicazioni in grado di separare la logica di presentazione dei dati dalla logica di business.

MVC è composto da tre parti:

- Model: si occupa dell'accesso ai dati e della loro gestione;
- View: rappresenta graficamente i dati in modo da renderli fruibili all'utente;
- Controller: gestisce gli input dell'utente in modo da modificare lo stato degli altri due componenti.

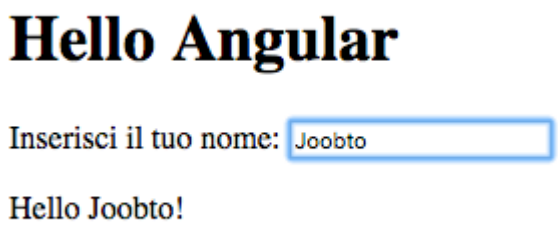
In Angular il **Model** è il dato che viene visualizzato all'utente tramite una **View,** mentre quest'ultima rappresenta l'HTML vero e proprio, ovvero come l'utente vede il dato che vogliamo mostrargli ed eventualmente interagire con esso. Alle View si possono applicare dei **Filtri,** che sono delle funzioni in grado di formattare i dati prima di visualizzarli su una view. Angular fornisce una serie di Filtri predefiniti, come la formattazione degli orari oppure quella della valuta, ma il programmatore può sempre estendere le funzionalità creando dei filtri personalizzati.
Mentre ciò che controlla l'interazione tra il Model e la View è chiamato **Controller** e sostanzialmente esegue tutta la logica di business dell'applicazione, dal punto di vista pratico un Controller è rappresentato da un oggetto Javascript.
Come possiamo notare, questo tipo di approccio ci permette di ottenere una netta separazione tra la logica applicativa e la presentazione. Inoltre esiste la possibilità di creare dei **Service** che hanno l'obbiettivo di fornire funzionalità indipendenti dalla grafica, come per esempio l'accesso ad un server via HTTP.

Direttive

Le direttive sono un elemento molto importante in Angular, poichè estendono il normale comportamento degli elementi HTML standard. Tutte le direttive iniziano con il prefisso "ng-". Guardiamo un esempio per capire come funzionano:

```
<h1>Hello Angular</h1>

<div ng-app>

    <p>Inserisci il tuo nome: <input type="text" ng-model="name"></p>

    <p>Hello {{name}}!</p>

</div>
```

E questo è ciò che appare all'utente nel browser:



Nel nostro caso abbiamo definito come contesto della nostra applicazione un div, lasciando quindi fuori l'intestazione h1. Questo indica ad Angular di concentrarsi soltanto sul contenuto del div. Se avessimo voluto prendere in considerazione l'intera pagina avremmo potuto impostare l'attributo **`ng-app`** sull'elemento body.
L'altra direttiva, **ng-model**, viene utilizzata sull'elemento input. Questa direttiva definisce un modello di dati associato alla casella di testo il cui nome è `name`. Immaginiamo una situazione come se `ng-model` definisse una sorta di variabile `name` a cui viene automaticamente assegnato il valore presente nella casella di testo. Questo meccanismo, che tratteremo tra poco, è chiamato **data binding**.

Two-way Data Binding

Nell'esempio precedente abbiamo visto come funziona il concetto di data binding; quest'ultimo diventa "two-way", ovvero ci dà la possibilità di manipolare le variabili dichiarate all'interno della direttiva ng-model direttamente dal nostro controller. Questo avviene grazie all'esistenza di un oggetto chiamato $scope, che viene passato dal framework al controller e sostanzialmente è un oggetto condiviso con la view.
Il compito dello **scope** è di consentire la definizione del modello dei dati e la sua esposizione alla view. Tutte le proprietà di questo oggetto saranno direttamente accessibili dalla view.

Android

Android è un sistema operativo per dispositivi mobili sviluppato da Google Inc. e basato sul kernel Linux.
È un sistema embedded progettato principalmente per smartphone e tablet, con interfacce utente specializzate. Fin dalla sua nascita ha riscosso particolare successo essendo di natura open source, ed oggi è presente sulla maggior parte dei telefoni usati nel mondo.

Application Structure

Android fornisce una serie di componenti base con i quali si possono costruire applicazioni native di qualsiasi genere. Per Joobto sono stati utilizzati i seguenti componenti:

Attività (activity)

Un'activity è essenzialmente una finestra che contiene l'interfaccia utente di un'applicazione ed il suo scopo è quello di permettere un'interazione con gli utenti. Un'applicazione può avere zero o più activity, anche se solitamente almeno una è presente. Ad ogni activity è associata una classe Java all'interno della quale può essere implementata la logica di business.

Frammento (fragment)

Un Fragment è una porzione di Activity. Non si tratta solo di un gruppo di controlli o di una sezione del layout. Può essere definito più come una specie di sub-activity con un suo ruolo funzionale ed un suo ciclo di vita. Ogni fragment ha una sua classe jJava e un suo file layout separato. Mentre l'activity è un componente fine a se stesso, un fragment non può esistere se non all'interno di un activity parent.

Servizi

Location

Per Joobto era un punto fondamentale ottenere la posizione del dipendente per poi verificare la sua presenza sul luogo di lavoro. In Android la posizione può essere ottenuta in 2 modi:

- Network: utilizzare la rete cellulare o il WI-Fi per ottenere la posizione
- GPS: utilizzare il ricevitore integrato nel telefono per ottenere la miglior posizione attraverso il satellite

Scegliere quale Provider usare tra i due dipende dall'utilizzo. Con il GPS si ottengono risultati più precisi ma può essere lento poichè il telefono deve effettuare prima la ricerca di un satellite, inoltre è inefficace quando il dispositivo si trova all'interno di luoghi chiusi. La determinazione della posizione tramite Network comporta invece una precisione più bassa ed è di solito più veloce rispetto al GPS, ma funziona anche all'interno di luoghi chiusi, basta che l'utente sia connesso alla rete internet. in Joobto è stato implementato l'utilizzo di entrambi i Providers per ottenere la miglior posizione possibile.

QR Code Scanner

Come abbiamo visto prima i QR Code sono nati per essere letti dagli smartphone ma non esiste un componente standard in Android che permetta di farlo, di conseguenza bisogna appoggiarsi su librerie esterne. Per la scansione del QR code è stata utilizzata una libreria Java molto famosa chiamata **ZXing**. L'implementazione di questa libreria direttamente all'interno di un'applicazione richiede lunghi tempi di lavoro, ma esistono sulla rete numerose librerie che la integrano all'interno di un'applicazione Android già pronta da utilizzare attraverso gli **Intent.** L'Intent è un oggetto attraverso il quale la nostra applicazione può invocare activity di altre applicazioni per svolgere un compito che noi non siamo in grado di svolgere (es: Scannerizzazione di un QR Code).
La libreria che è stata scelta per l'utilizzo dispone delle seguenti caratteristiche:

- può essere utilizzata tramite Intent.
- Poche righe di codice necessarie per l'utilizzo
- la scansione può essere effettuate in modalità orizzontale o verticale.
- la fotocamera è gestita da un thread in background.

Di seguito è riportato il codice che invoca una scansione utilizzando la libreria prima descritta:

```
IntentIntegrator.forSupportFragment(this)
        .setCaptureActivity(AnyOrientationCapture.class)
        .setOrientationLocked(false)
        .setBeepEnabled(true)
        .setDesiredBarcodeFormats(IntentIntegrator.QR_CODE_TYPES)
        .setPrompt(message)
        .initiateScan();
```

Il risultato della scansione viene inoltrato alla nostra activity mediante un meccanismo di callback. Di seguito è riportato il codice che interpreta il risultato della scansione:

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    IntentResult result = IntentIntegrator.parseActivityResult(requestCode, resultCode,
data);
    if(result != null) {
        if(result.getContents() == null) {
            System.out.println("L'utente ha chiuso lo scanner");
        } else {
            System.out.println("Risultato della scansione: " + result.getContents());
        }

    }
}
```

Visto che per la scansione del QR code è necessario l'utilizzo della fotocamera, nel progetto devono essere forniti i permessi per l'accesso alla fotocamera del cellulare. La gestione dei

permessi in Android è svolta mediante la presenza di un file chiamato AndroidManifest.xml all'interno del quale vengono dichiarate tutte le activity e i permessi utilizzati dalla nostra applicazione.

Accesso API

Come già detto in precedenza il back-end di Joobto espone delle API per l'interazione con i dati. Per utilizzarle è necessaria la presenza di un client HTTP. In Android è presente il client default di Java e di conseguenza il suo utilizzo non permette la scrittura di un codice pulito con logica separata dalla grafica. Per fare ciò è stata utilizzata **Android Asynchronous Http Client**, una libreria callback-based per eseguire le chiamate HTTP. Dispone di molte caratteristiche che incoraggiano il suo utilizzo, alcune fra le quali:

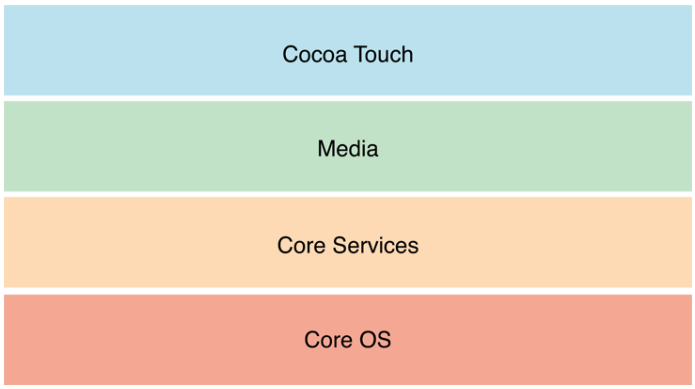
- Compatibilità con quasi tutte le versioni Android, comprese quelle recenti (Android Marshmallow)
- Richieste HTTP asincrone con gestione dei risultati tramite callbacks
- Le richieste vengono effettuate fuori dall'UI thread
- Dimensioni ridotte (solo 9KB)
- Parsing automatico delle risposte in JSON
- Persistenza automatica dei Cookie ricevuti
- Supporto ai principali metodi HTTP (GET, PUT, POST, DELETE)

Per vedere come funziona di seguito è riportato uno snippet di codice che usa la libreria in questione per ottenere dal server, dato l'id, le informazioni sul checkin:

```
public static final String url = "https://joobto.me/joobtorest/v1/index.php/";

public void getCheckin(int id) throws JSONException {
    JoobtoRestClient.get("checkin/ + id", null, new JsonHttpResponseHandler() {

        // Success connection
        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONObject response) {
            // response contiene l'oggetto JSON ricevuto con la risposta
        }

        // Fail connection
        @Override
        public void onFailure(int statusCode, Header[] headers, JSONObject error) {
           // richiesta fallita (es: statusCode = 404)
        }
    });
}
```

iOS

iOS è il sistema operativo sviluppato da Apple per iPhone, iPad ed iPod Touch. Il sistema operativo gestisce l'hardware del dispositivo e fornisce le tecnologie richieste per implementare le applicazioni native.
L'implementazione delle tecnologie di iOS può essere vista come una serie di strati. Gli strati più bassi contengono i servizi e le tecnologie fondamentali. Gli strati più alti invece sono costruiti sopra gli strati bassi e forniscono servizi e tecnologie più sofisticate.



iOS ha quattro gradi di astrazione:

- **Core OS**: è lo strato che contiene il sistema operativo ed i servizi sopra i quali si basano gli altri servizi e le altre applicazioni presenti su iOS. In particolare il Core OS si occupa dell'accesso al Bluetooth, elaborazione delle immagini, servizi di sicurezza, gestione delle reti e operazioni matematiche;
- **Core Services**: è la parte di iOS dedicata alla gestione dei servizi che vengono offerti alle app di terze parti. In particolare si occupa della fruizione dei servizi di iCloud, servizi di localizzazione, social media e connessione alla rete internet;
- **Media**: si occupa di tutto ciò che ha a che fare con la grafica video ed audio. Permette di gestire le animazioni, grafica 2D e 3D attraverso l'uso di OpenGL ed anche l'accesso alla libreria foto e video dell'utente;
- **Cocoa Touch**: si tratta dello strato più importante e più utilizzato dagli sviluppatori. All'interno di questo strato sono contenuti tutti gli elementi grafici che permettono la creazione dell'interfaccia utente.

Swift

Il 2 giugno 2014, il WWDC di Apple si concluse con un annuncio scioccante:"Abbiamo un nuovo linguaggio di programmazione". Ciò si trasformò in una enorme sorpresa per gli sviluppatori presenti che fino ad allora erano abituati a programmare utilizzando Objective-C, un linguaggio che nasce per essere un'estensione ad oggetti di C.

Gli ingegneri Apple sono partiti da zero nella costruzione di Swift, basandosi su queste quattro caratteristiche principali:

- **Linguaggio ad oggetti**. Swift è un linguaggio ad oggetti moderno. Tutto in Swift è un oggetto;

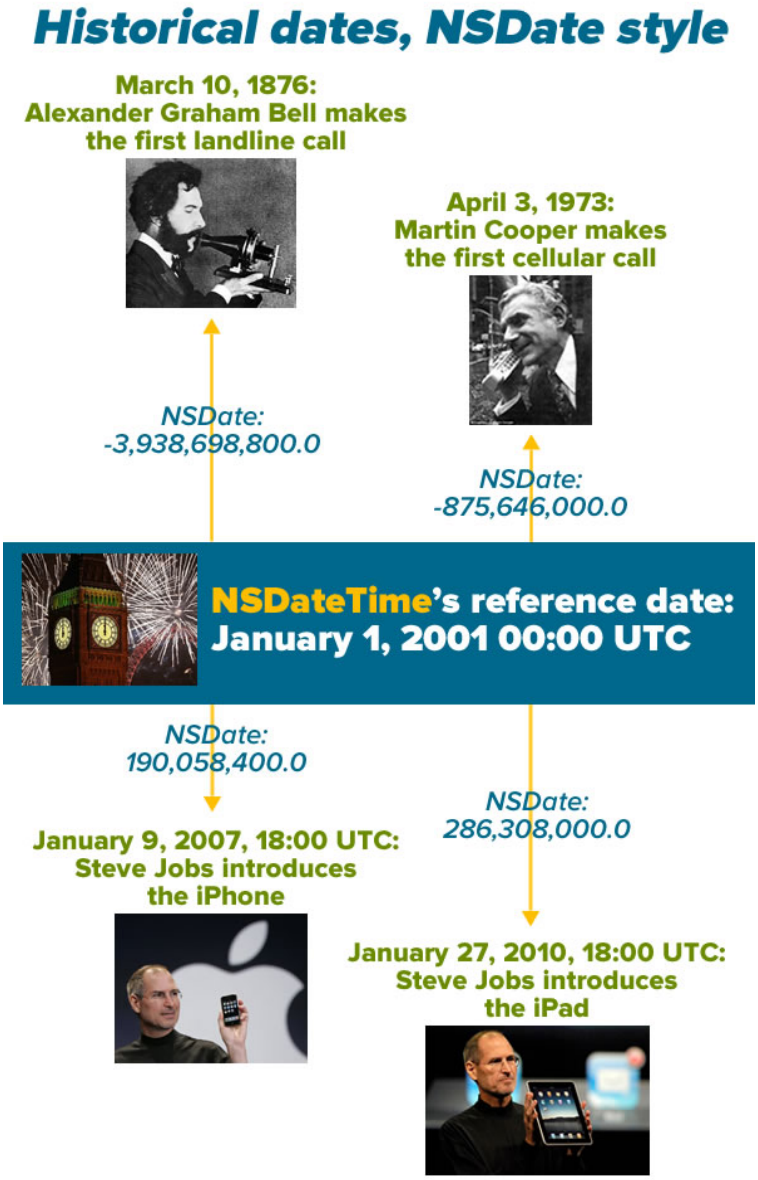
- **Chiarezza**. Il codice è facile da leggere e da scrivere, come se fosse il testo di un libro;
- **Sicurezza**. Le variabili vengono sempre inizializzate prima del loro utilizzo e gli interi e gli array sono controllati per evitare overflow;
- **Gestione della memoria**. Swift gestisce automaticamente la memoria, liberando il programmatore da ogni responsabilità.

L'alternativa, Objective-C, esiste ancora ed è utilizzabile dai programmatori che lo desiderano. E' infatti una scelta comune quella di scrivere un'applicazione utilizzando sia codice Swift che codice Objective-C.
Come detto precedentemente, Objective-C aggiunge gli oggetti a C, il che significa che è solo parzialmente ad oggetti. La sua sintassi è a volte difficoltosa e contorta; il codice composto da chiamate di metodi annidate una dentro l'altra rende difficile la comprensione immediata di cosa sta succedendo in quella determinata istruzione. Inoltre la gestione della memoria è manuale; successivamente è stato introdotto ARC (Automatic Reference Counting) che ha permesso ai programmatori di smettere di preoccuparsi della gestione della memoria, diminuendo così i possibili errori.

Gestione del tempo (NSDate)

Joobto nasce dalla necessità di tenere traccia del tempo in modo automatico, all'avanguardia. Per raggiungere questo scopo una delle cose da fare era preoccuparsi di come Swift stesso gestisse il tempo. Prima ancora di questo bisogna domandarsi: come si misura il tempo? Qual è il punto di riferimento per la misura del tempo?

L'UTC (Coordinated Universal Time) è l'ora di riferimento internazionale sulla quale si basano tutti i fusi orari. L'UTC è il tempo su cui si basano la maggior parte dei sistemi informatici (Unix e Windows), i quali memorizzano la data come il numero di secondi passati dal 1° gennaio 1970.
In Swift si utilizza la classe `NSDate` per rappresentare una data o un orario. `NSDate`. A differenza delle normali convenzioni, usa come punto di riferimento l'inizio del terzo millennio secondo il calendario gregoriano: 1° gennaio 2001 alle 00:00 UTC. Esso memorizza il tempo in un float a 64-bit che rappresenta il numero di secondi. I valori negativi rappresentano il numero di secondi prima del 1° gennaio 2001 00:00 UTC, mentre i valori positivi rappresentano il numero di secondi passati dopo quella data.
Secondo la documentazione Apple, questa rappresentazione del tempo produrrebbe una precisione al millisecondo in grado di coprire un range di 10.000 anni, il che significa che un'istanza della classe `NSDate` è in grado di rappresentare un punto del tempo dal 3000 a.C. al 7000 d.C. .



Design Pattern

Un design pattern è una soluzione comune ad un problema ricorrente. Si tratta di un approccio da seguire durante la fase di sviluppo in grado di risolvere un problema.
I pattern sono classificati in:

- **Creazionali**: nascondono i costruttori delle classi e mettono dei metodi al loro posto, creando un'interfaccia. In questo modo si possono utilizzare oggetti senza sapere come sono implementati;
- **Strutturali**: I pattern strutturali consentono di riutilizzare degli oggetti esistenti, fornendo agli utilizzatori un'interfaccia più adatta alle loro esigenze;
- **Comportamentali**: sono pattern che forniscono soluzione alle più comuni tipologie di interazione tra gli oggetti.

Factory Method

Il pattern Factory Method fornisce un'interfaccia per creare un oggetto, ma lascia che le sottoclassi decidano quale oggetto istanziare.

L'esempio qui sotto mostra come è stata utilizzato il pattern Factory Method durante l'analisi della risposta HTTP ricevuta dal server. Ogni risposta contiene un codice che identifica univocamente un tipo di risposta. In base a ciò, attraverso l'uso di uno `switch-case` viene identificata la risposta e creato un oggetto in grado di rappresentarla.

```
static func analyze(response: HTTPResult) -> Response {

        let json = JSON(response.json!)

        switch json["status"].intValue {

        case    genericError,    employAlreadyLogged,    employBlocked,    incorrectAppID,
qrNotRecognized, locationNotBelongs, recordNotFound:
            return Response(json: json)
        case employLogged:
            let cookie = response.cookies[ServerRequest.cookieName]!

            return LoggedResponse(cookie: cookie, json: json)

        case correctCheckIn:
            return CheckInResponse(json: json)

        case correctCheckOut:
            return CheckOutResponse(json: json)

        case employNotOnLocation:
            return DistanceErrorResponse(json: json)

        default:
            return Response(error: true, message: json["message"].stringValue)

        }
    }
```

Observer

Il pattern Observer definisce una dipendenza uno a molti fra oggetti diversi, in maniera tale che se un oggetto cambia il suo stato, tutti gli oggetti dipendenti vengono notificati del cambiamento avvenuto e possono aggiornarsi.

Nell'esempio sottostante si vede l'applicazione del pattern scritta in Swift tratta dal codice della classe `NotificationModel` di Joobto. Ogni volta che il valore dell'attributo `time` cambia, viene eseguito il codice all'interno di `didSet`. Una volta impostata l'ora della notifica, si controlla se l'orario è uguale o minore dell'orario attuale, in questo caso si aggiungono a `time` 24 ore (l'equivalente in millisecondi nel codice) in modo da far suonare la notifica il giorno dopo.

```
dynamic var time: NSDate = NSDate() {
       didSet {
            if time.equalToDate(NSDate()) || time.isLessThanDate(NSDate()) {
                time = time.dateByAddingTimeInterval(24*60*60)
            }
        }
}
```

Delegate

Il pattern Delegate è uno dei più utilizzati nella programmazione iOS. E' un design pattern che sposta le responsabilità da una classe ad un'altra, creando così divisione dei compiti.
All'interno del pattern ci sono 3 figure fondamentali:

- Un *contratto* (rappresentato nel codice da un'interfaccia, in Swift chiamate **protocolli**), cioè una "cosa" in mezzo a due parti che stanno cercando di negoziare per un affare. Ad una delle due parti il contratto garantisce che verranno rispettati certi termini. All'altra parte il contratto sarà come un insieme di operazioni obbligatorie da soddisfare;
- Un *delegante*, che delega delle operazioni ad un oggetto che ha implementato il protocollo (il *contratto*);
- Un *delegato*, che implementa il protocollo e implementa quando richiesto per soddisfare le esigenze del *delegante*.

Nella pratica, il pattern Delegate è spesso utilizzato come un modo per far comunicare tra loro due classi.

Questo è il protocollo (il contratto) che il delegato andrà ad implementare per soddisfare le esigenze del delegante. All'interno del protocollo è presente solo il metodo `didChangeSwitchState`, utilizzato per segnalare quando l'utente decide di attivare/disattivare una notifica.

```
protocol NotificationsCellDelegate: class {
    func didChangeSwitchState(sender: NotificationsTableViewCell, enabled: Bool)
}
```

Questo è il codice del *delegante* nel quale è presente l'attributo `delegate` che punta al *delegato*. Quando lo switch viene cliccato dall'utente, il sistema notifica l'evento chiamando `switchChanged()`, che al suo interno delegherà il compito di gestire quanto accaduto al *delegato* attraverso il metodo `didChangeSwitchState`.

```
class NotificationsTableViewCell: UITableViewCell {
       ...
       weak var delegate: NotificationsCellDelegate?
       ...

       @IBAction func switchChanged(sender: UISwitch) {
            self.delegate?.didChangeSwitchState(self, enabled: notificationSwitch.on)
       }
}
```

Ed ecco il codice del delegato che implementa il protocollo `NotificationsCellDelegate` e gestisce l'evento. Una volta chiamato il metodo `didChangeSwitchState`, questo si occupa di aggiornare il valore nel database e di creare/eliminare le `UILocalNotification` collegate a quella notifica.

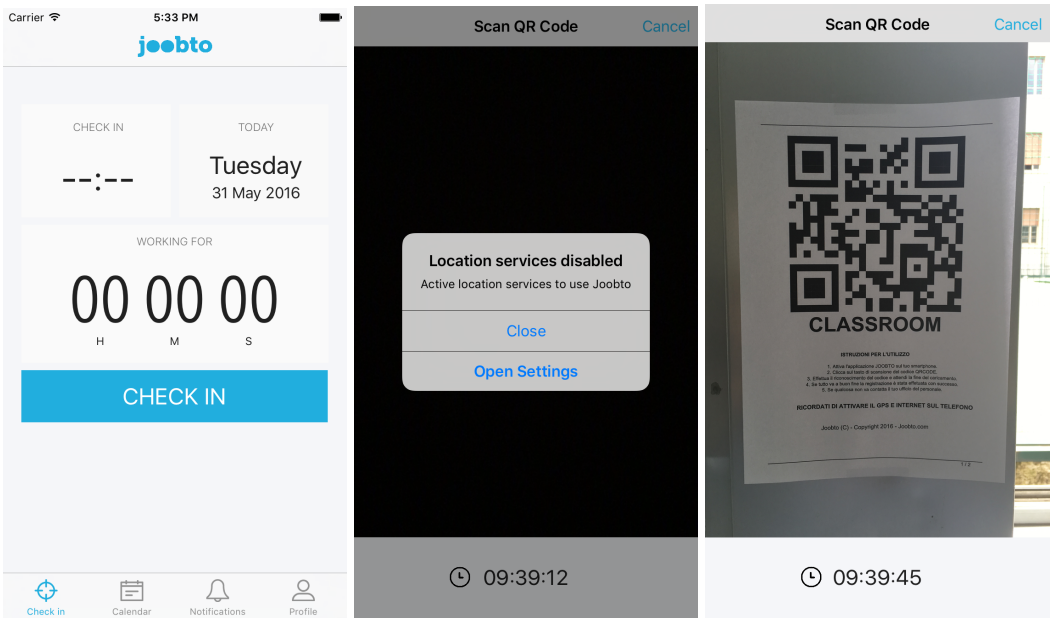
```
class NotificationsTableViewController: NotificationsCellDelegate {
       func didChangeSwitchState(sender: NotificationsTableViewCell, enabled: Bool) {
               let indexPath = self.tableView.indexPathForCell(sender)
               try! DatabaseManager.realm.write {
                   notifications[indexPath!.row].active = enabled

               }
               sender.enabled = enabled

               if enabled {

       NotificationModelManager.createLocalNotification(notifications[indexPath!.row])
               }
               else {

       NotificationModelManager.deleteLocalNotification(notifications[indexPath!.row])
               }
           }

}
```

Servizi

Quando si inizia a sviluppare un'applicazione bisogna avere ben chiari quali saranno gli utenti finali che la utilizzeranno. Parlando di Joobto, si tratta di un servizio che verrà utilizzato dal personale di un'azienda, la cui età possiamo ipotizzare sia compresa tra 20 e 60 anni.
Essendo il range di età molto ampio, è importante che l'applicazione sia resa disponibile non solo per i dispositivi di ultima generazione, ma anche per dispositivi non molto recenti in grado di soddisfare ancora le esigenze dei propri possessori.

Nell'autunno 2015 Apple ha rilasciato iOS 9, l'ultimo sistema operativo compatibile a partire dall'iPhone 4s (dispositivo lanciato nel 2011). Stando ai dati riportati sul sito dell'azienda stessa, ad oggi l'84% degli utenti ha installato iOS 9 su propri dispositivi.
Il vantaggio di utilizzare l'ultimo sistema operativo sta nel fatto di poter utilizzare le nuove API messe a disposizione da Apple.
E' per questi due motivi principali (la larga diffusione e l'aggiunta di nuove API) che Joobto è compatibile a partire da iOS 9 e successivi.

Location



Come descritto nella fase di progettazione, per verificare che l'utente si trovi fisicamente sul luogo di lavoro al momento della timbratura Joobto utilizza i servizi di localizzazione. Sul server è stato precedentemente memorizzato il punto in cui è stato posizionato il QR Code. Al momento della timbratura vengono inviate al server latitudine e longitudine del dispositivo in modo tale da verificare che esso si trovi all'interno di un'area ristretta al cui centro è presente il QR Code ed evitare quindi che il dipendente fotografi il QR Code e timbri da casa.

Con iOS 9 sono stati aggiungi dei nuovi metodi alla classe che si occupa di gestire i servizi di localizzazione chiamata `CLLLocationManager`; uno di questi metodi, chiamato `requestLocation()` permette di ottenere la posizione attuale del dispositivo in modo molto più facile ed intuitivo rispetto alle precedenti API.

Il codice sottostante è un esempio di come abbiamo impiegato i nuovi metodi di localizzazione. In particolare è possibile notare l'applicazione del pattern Delegate:

```
override func viewDidLoad() {
// Create a location manager object
       self.locationManager = CLLocationManager()

       // Set the delegate
       self.locationManager.delegate = self

           locationManager.desiredAccuracy = kCLLocationAccuracyBest
   }
```

```
func getQuickLocationUpdate() {
   // Request location authorization
   self.locationManager.requestWhenInUseAuthorization()

   // Request a location update
   self.locationManager.requestLocation()
   // Note: requestLocation may timeout and produce an error if authorization
    has not yet been granted by the user
}

func locationManager(manager: CLLocationManager, didUpdateLocations locations:
    [CLLocation]) {
   // Process the received location update
}
```

Notifiche

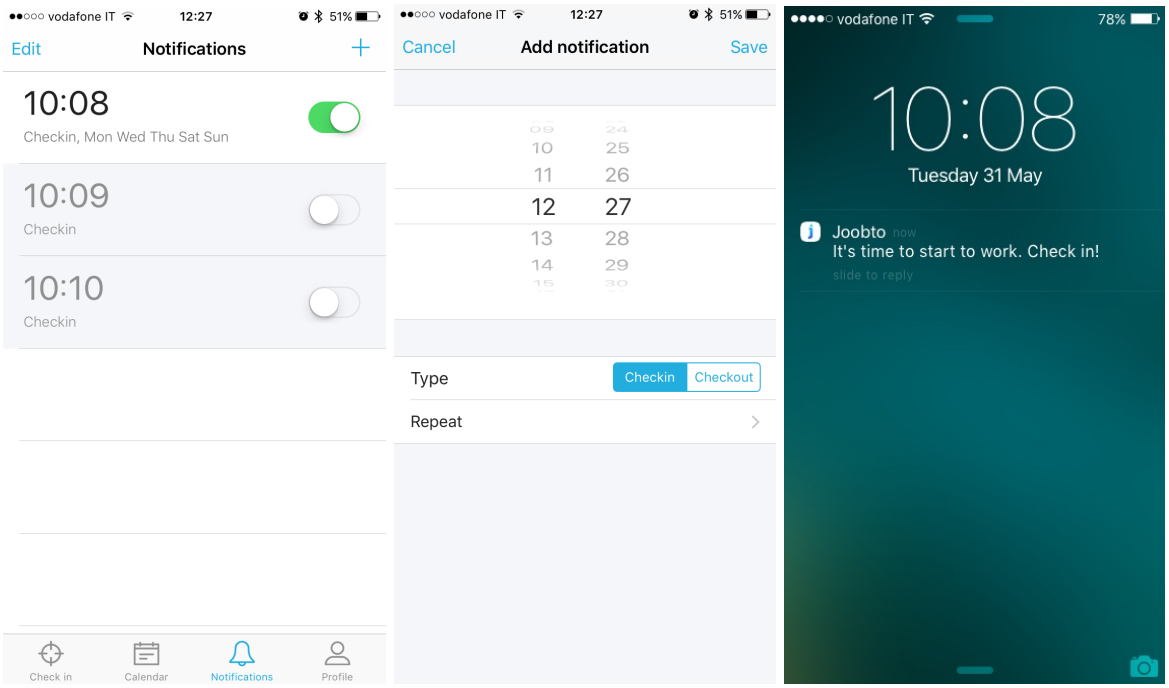
Una delle ultime novità a cui abbiamo pensato per Joobto è la possibilità di poter ricordare all'utente a che ora timbrare l'entrata e l'uscita. Per questo motivo abbiamo creato la sezione Notifiche, nella quale l'utente può creare una notifica selezionando l'orario, il tipo di timbratura (entrata o uscita) e gli eventuali giorni in cui ripetere la notifica.

In iOS sono presenti diverse tipologie di notifiche; quelle che più si addicono alle nostre esigenze sono le UILocalNotification.
Una UILocalNotification è una notifica che viene programmata dall'applicazione stessa senza bisogno di una connessione internet o di un server di appoggio.

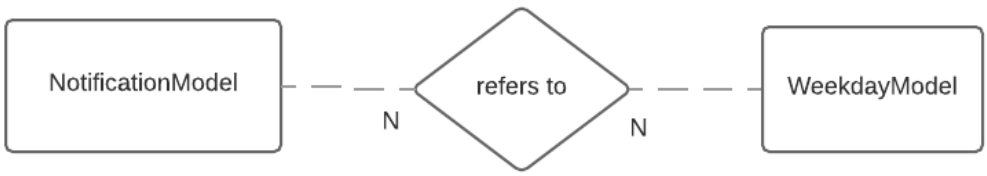
Ecco un'esempio del metodo `scheduleLocal()`, che programma una UILocalNotification all'orario passato come parametro:

```
func scheduleLocal(fireDate: NSDate) {
    let notification = UILocalNotification()
    notification.fireDate = fireDate
    notification.alertBody = "Hey you! Yeah you! This is a notification!"
    notification.soundName = UILocalNotificationDefaultSoundName
    UIApplication.sharedApplication().scheduleLocalNotification(notification)
}
```

Attraverso l'interfaccia utente, l'utente può selezionare un orario, il tipo di timbratura (entrata o uscita) ed eventualmente anche i giorni in cui si vorrebbe ripetere la notifica.



All'interno di Joobto è presente una classe chiamata `NotificationModel`, che rappresenta la singola notifica creata dall'utente. Come abbiamo spiegato prima, però, l'utente può scegliere di ripetere la notifica per ogni giorno della settimana. Questo significa che per ogni oggetto NotificationModel possono esistere più oggetti UILocalNotification, che differiscono solo per il valore di fireDate.
Per gestire tutto questo all'interno del database Realm abbiamo creato l'entità WeekdayModel già riempita con i giorni della settimana; inoltre tra NotificationModel e WeekdayModel è presente una relazione N-N, in modo tale da una NotificationModel può riferirsi a uno o più WeekdayModel e viceversa.



Framework

Due fantastiche cose che caratterizzano lo sviluppo di applicazioni iOS sono la community e la presenza di una vasta scelta di librerie di terze parti scritte da altri sviluppatori.

In Joobto abbiamo usato 5 framework che hanno permesso di rendere molto più semplice e veloce lo sviluppo dell'app.
Per gestire tutti questi framework abbiamo utilizzato Carthage, un dependecy manager compatibile con Swift molto utilizzato nel settore. Basta creare un file di testo chiamato Cartfile, nel quale si specifica il nome della libreria che vogliamo aggiungere al progetto XCode ricordandoci di specificare il nome dell'host sul quale è stata pubblicata la libreria.

```
github "JustHTTP/Just"
github "SwiftyJSON/SwiftyJSON"
github "ashleymills/Reachability.swift"
github "Mozharovsky/CVCalendar"
github "realm/realm-cocoa"

Esempio di Carfile da utilizzare in Carthage
```

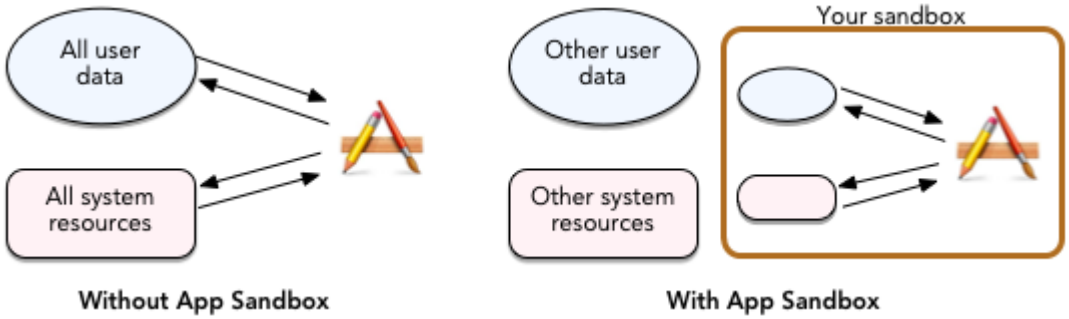
I framework utilizzati in Joobto sono i seguenti:

- **Just**: attraverso questa libreria è possibile inviare delle richieste HTTP in modo veloce;
- **SwiftyJSON**: converte i file JSON in array utilizzabili in Swift;
- **Reachability**: permette in modo veloce di verificare la presenza di una connessione internet, sia WiFi che tramite la rete cellulare;
- **Realm**: si tratta di un database cross-platform largamente diffuso che va a sostituire SQLite;
- **CVCalendar**: gestisce il calendario presente all'interno della sezione Calendario dell'applicazione.

Salvataggio Dati

Quella che in inglese viene chiamata Persisting Data è oggi una delle caratteristiche fondamentali di un'applicazione per smartphone. Significa che una volta chiusa l'applicazione i dati vengono salvati nella memoria di massa, in modo da poter essere ritrovati all'avvio successivo.

In iOS il salvataggio file, che possono contenere le impostazioni dell'utente o anche solo semplici file temporanei, è gestito diversamente rispetto ad altri sistemi operativi come Android.
La sicurezza è stata da sempre una delle principali priorità per Apple. Proprio per questo motivo ogni applicazione iOS è posizionata all'interno di una sandbox.
Con il termine sandbox ci si riferisce a un metodo di programmazione che in un certo senso "incapsula" l'applicazione, isolandola e limitandone la possibilità di accedere a vari elementi del sistema operativo.
Per Apple il sandboxing è "un ottimo metodo per proteggere il sistema e gli utenti, che limita le risorse a cui le applicazioni possono accedere e rende più difficile la compromissione del sistema da parte del malware".



Without App Sandbox With App Sandbox

Nella prima versione di Joobto le informazioni dell'utente si limitavano ad essere nome, cognome, qualche altro dato sul dipendente e l'azienda, ed eventualmente l'orario in cui si era effettuato il check-in. Successivamente il singolo orario è diventato invece un vettore di check in effettuati durante l'arco della giornata.
Oggi Joobto implementa una sezione Calendario, in cui l'utente può visionare tutti i check in effettuati. Ed è stata anche aggiunta la sezione Notifiche.
I dati da tenere in memoria sono aumentati ed abbiamo dovuto decidere quale fosse lo strumento più adatto alle nostre esigenze.

iOS mette a disposizione una libreria chiamata Core Data in grado di memorizzare grandi quantità di dati all'interno di un database. Dopo un consulto in rete abbiamo però visto che la sintassi della libreria era molto ricca ed avrebbe richiesto del tempo per essere utilizzata al meglio.
Dopo alcune ricerche abbiamo deciso di utilizzare Realm, un framework molto diffuso in grado di gestire un database in maniera facile e veloce.

In Realm qualsiasi classe che erediti da Object può diventare un'entità. Ecco un esempio della classe NotificationModel:

```
let notification = NotificationModel()
notification.fireDate = NSDate()
notification.active = true
notification.type = 0

let realm = try! Realm()

try! realm.write {
    realm.add(notification)
}
```

Anche le query sono facili da eseguire:

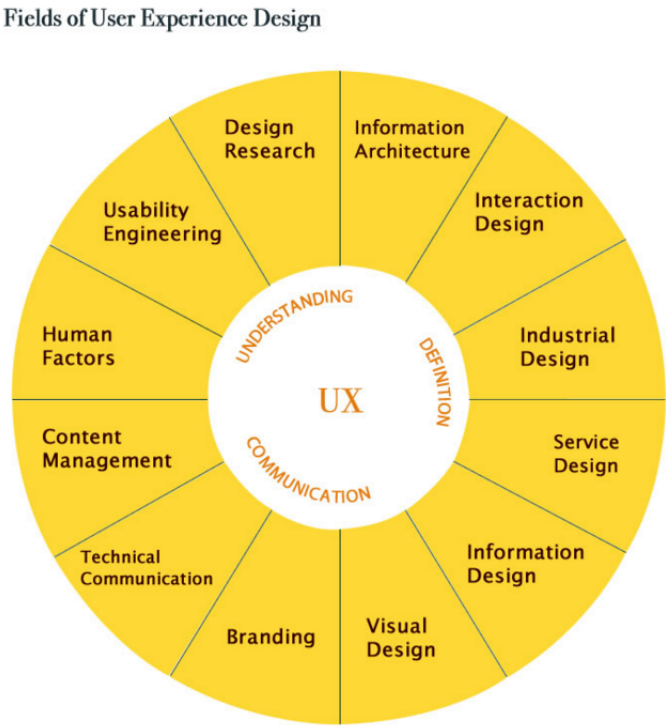
```
let realm = try! Realm()
let results1 = realm.objects(NotificationModel).filter("active == true")

// Queries are chainable
let results2 = results1.filter("type == 0")
```

UX

Per User Experience (UX) si intende ciò che una persona prova quando utilizza un prodotto, un sistema o un servizio. L'esperienza d'uso concerne gli aspetti esperienziali, affettivi, l'attribuzione di senso e di valore collegati al possesso di un prodotto e all'interazione con esso, ma include anche le percezioni personali su aspetti quali l'utilità, la semplicità d'utilizzo e l'efficienza del sistema.

In Joobto ci siamo dovuti improvvisare anche User Experience Designer ed imparare le basi di una buona UX.

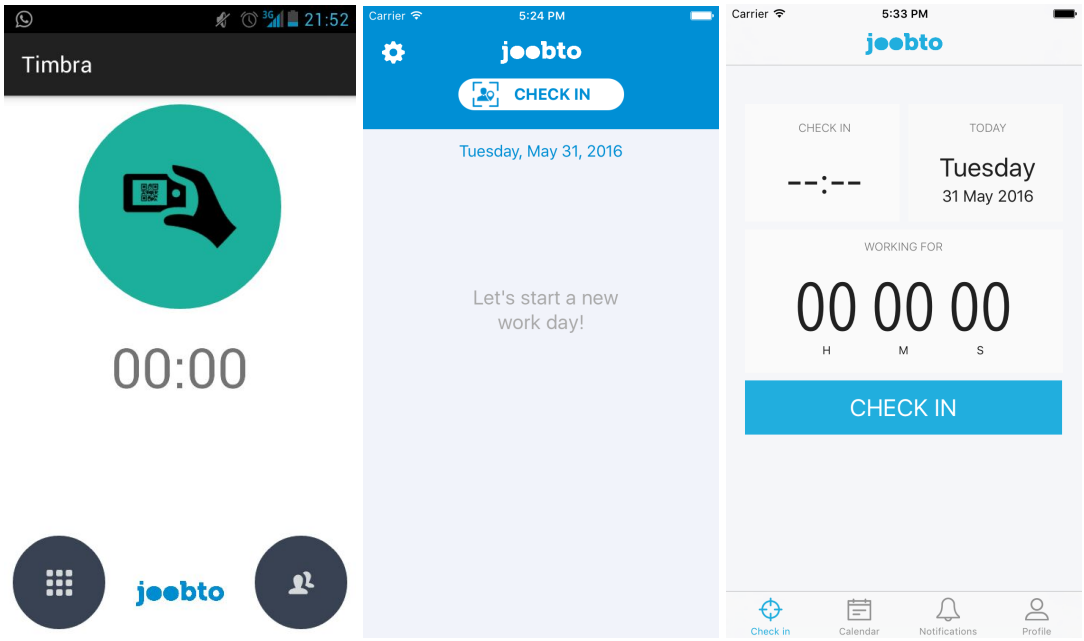
La User Experience Design è una materia molto ricca, che racchiude dentro di sé molti altri campi. Con Joobto ci siamo limitati a trattarne solo alcuni, in modo da permettere all'utente l'esperienza migliore durante l'uso dell'app.



UI

L'interfaccia utente (User Interface) è ciò che permette all'utente di interagire con il sistema. L'obiettivo di questa interazione è quello di consentire un funzionamento efficace del sistema e nello stesso tempo lasciare che sia l'utente a comandarlo in maniera semplice.

La prima versione di Joobto è stata sviluppata in Android (figura a sinistra) con lo scopo di implementare l'aspetto funzionale del progetto, tralasciando l'aspetto grafico. Successivamente è stata pensata una nuova grafica per le piattaforme iOS e Android (figura centrale) fino ad arrivare all'attuale interfaccia (figura a destra).



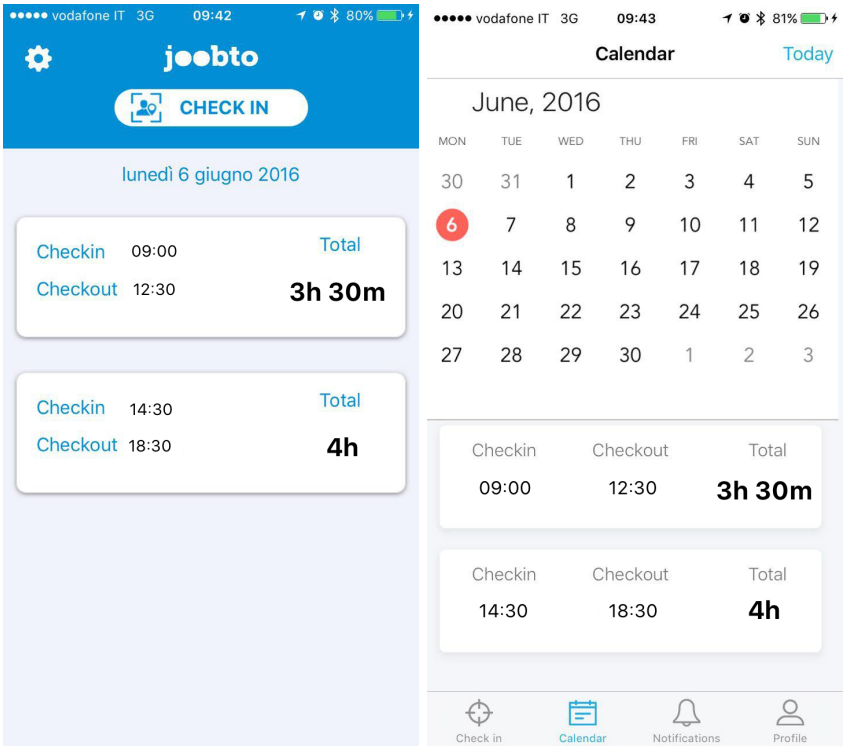
Contemporaneamente all'interfaccia utente anche l'icona dell'applicazione si è trasformata nel tempo, fino ad arrivare a quella attuale (immagine a destra).

Sono 3 i principi che abbiamo seguito durante la progettazione della nuova interfaccia grafica:

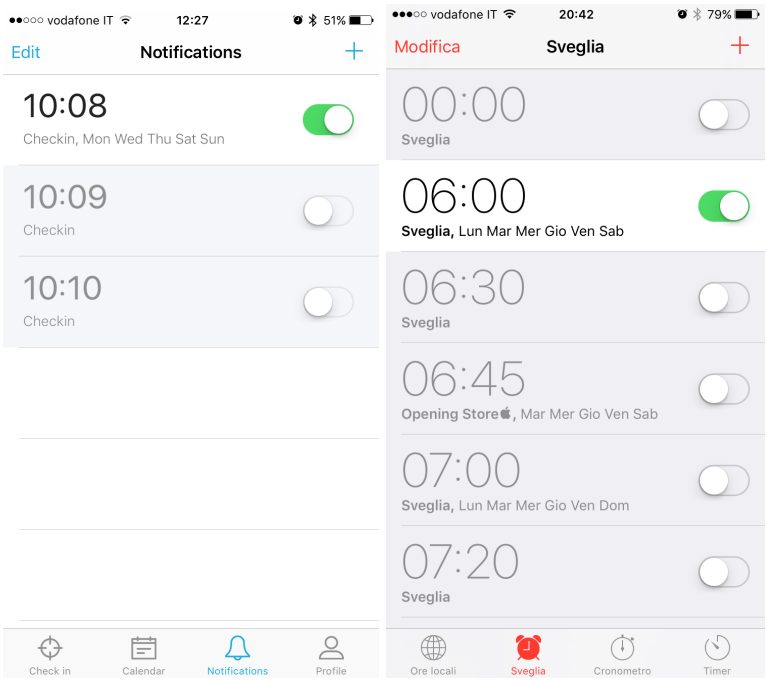
- **Chiarezza**: per poter utilizzare al meglio l'applicazione, l'utente deve essere sempre informato con chiarezza su cosa è appena successo, dove si trova, cosa può fare e cosa accadrà se farà questo. La chiarezza è sinonimo di semplicità. Più un sistema è semplice da utilizzare, senza essere troppo banale, più è chiaro per l'utente;
- **Flessibilità**: l'applicazione deve soddisfare le esigenze dell'utente in qualsiasi situazione. L'interfaccia dev'essere accomodante e funzionale con qualsiasi tipo di dato possa generarsi;
- **Familiarità**: una grafica rivoluzionaria, fuori dalle righe potrebbe risultare spiacevole all'utente. E' dimostrato come le persone si trovino a proprio agio utilizzando ciò che è per loro familiare. E' importante quindi seguire le linee guida fornite da iOS per il design.

La schermata del check-in è tornata alle origini, solo un po' più organizzata e moderna: l'elemento principale è lo scorrere del tempo che ricorda all'utente da quanto ha timbrato l'entrata. Sono in bella vista anche l'orario di check in e la data del giorno, in modo da dare all'utente tutte le informazioni di cui abbia bisogno.

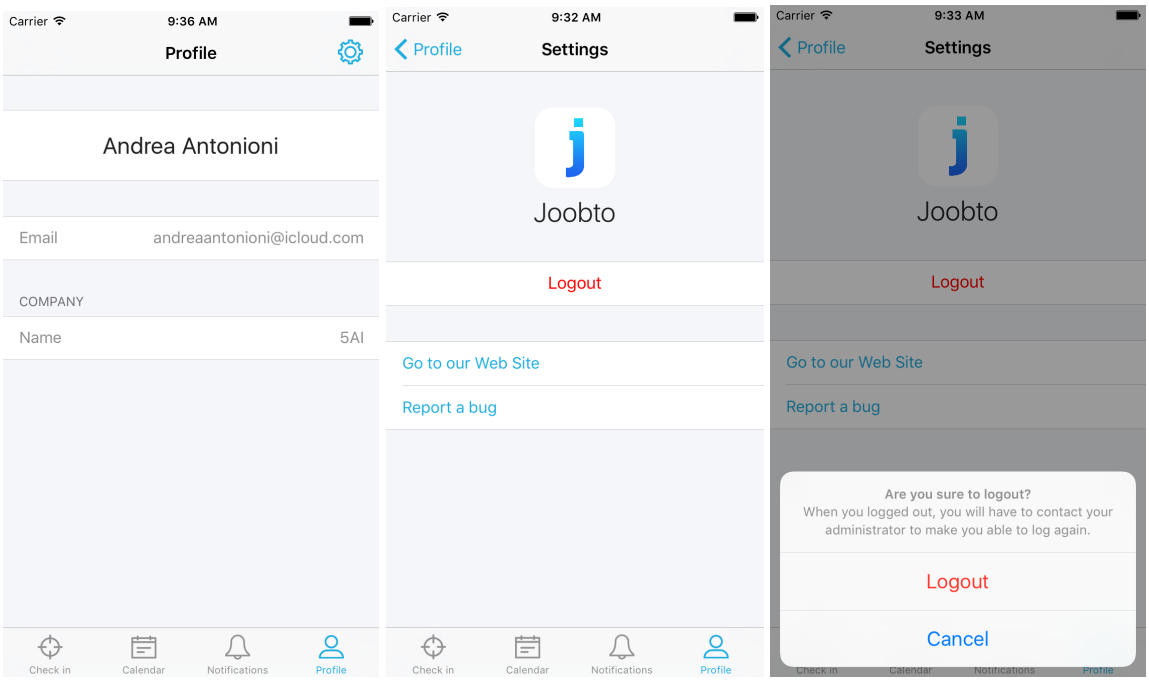
È stata aggiunta la nuova schermata Calendario, che permette all'utente di consultare non più le timbrature effettuale nel giorno corrente (immagine a sinistra), ma tutte quelle effettuale dal momento dell'installazione dell'app (immagine a destra).



La nuova sezione Notifiche è stata pensata per ricordare all'utente di timbrare l'entrata e l'uscita con Joobto. La grafica della schermata è stata ripresa dall'applicazione Orologio (immagine a destra) presente di default su tutti i dispositivi iOS, in modo da risultare familiare all'utente.



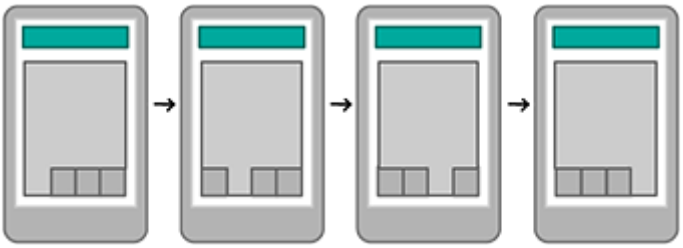
Infine sono presenti anche il Profilo dell'utente e le Impostazioni, dove l'utente è in grado di effettuare i logout, di visitare il sito web di Joobto oppure di segnalare un bug dell'applicazione attraverso l'invio di un email.



Information Architecture

Uno degli aspetti di cui ci siamo preoccupati durante l'evoluzione di Joobto era la Information Architecture, ovvero come presentare le varie funzioni dell'app in modo che l'utente possa navigare tra le varie schermate in maniera veloce ed intuitiva.

Jooboto nasce con una semplice schermata dedicata al check in. Con il tempo si sono aggiunte le notifiche, il calendario, le impostazioni e il profilo dell'utente. Per presentare tutte queste funzioni abbiamo pensato di applicare il Tabbed View pattern.



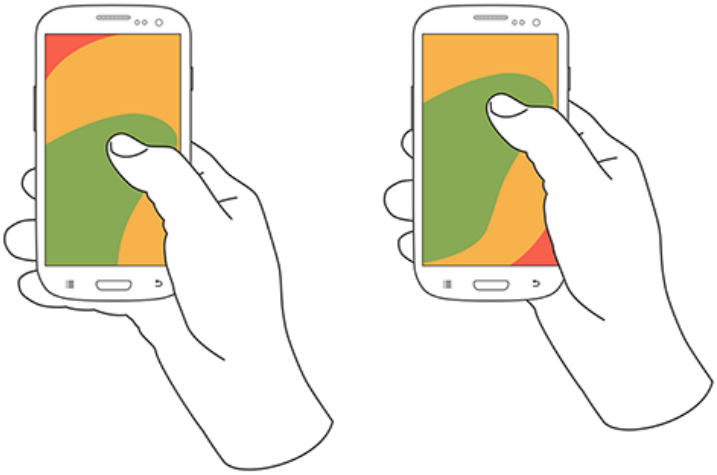
Ogni funzione principale viene organizzata in tab attraverso le quali l'utente potrà navigare utilizzando la Tab bar presente nella parte bassa dello schermo. In questo modo non solo

l'utente ha la percezione di come sono organizzate le informazioni, ma può navigare tra esse con la sicurezza di poter tornare indietro attraverso la Tab bar sempre presente.

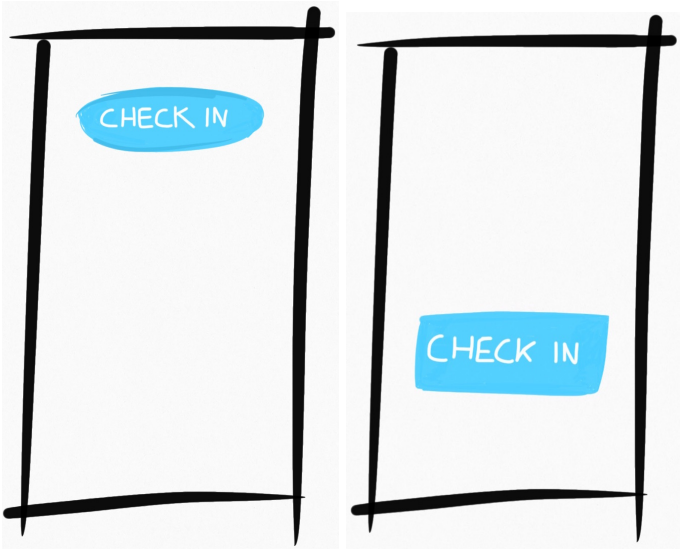
Design Research

Una parte importante della UX è composta dal Design Research, ovvero la possibilità di effettuare ricerche e sondaggi utili allo sviluppo dell'applicazione.
Durante lo sviluppo di quella che è l'attuale interfaccia utente abbiamo condotto delle ricerche per capire come l'utente tenga in mano il proprio smartphone.
Joobto è un'applicazione pensata per essere utilizzata dai lavoratori i quali devono inquadrare un QR Code. Si tratta della funzione principale di Joobto, la più importante.

Secondo uno studio pubblicato in rete, il 49% delle persone utilizzano il proprio smartphone con una sola mano. Guardando l'immagine sottostante è possibile capire il raggio di azione che l'utente, attraverso l'uso di una mano, ha sullo schermo.



A seguito di questa ricerca, in fase di sketching della nuova grafica abbiamo abbassato il bottone che permette di inquadrare il QR Code poco sotto il centro dello schermo; in questo modo il bottone è più visibile e più accessibile agli utenti.



Conclusioni

Con Joobto siamo riusciti ad applicare tutte le conoscenze apprese durante i tre anni di specializzazione, al fine di creare un sistema che non si limita ad essere un esercizio in classe, ma un vero prodotto di mercato (con l'obbiettivo di diventare una Startup).
Ognuno di noi ha approfondito il proprio aspetto di competenza, imparando in modo professionale quali sono gli strumenti e le metodologie di sviluppo più utilizzate nel mondo del lavoro.
Uno degli aspetti fondamentali durante lo sviluppo di Joobto è stato il team working: utilizzando strumenti come Trello e MeisterTask siamo rimasti sempre in contatto tra di noi, in modo da condividere sia opinioni che problematiche riscontrate durante lo svolgimento del progetto.

Oggi Joobto è pronto per essere commercializzato e venduto come servizio per le aziende. Ovviamente non abbiamo una preparazione commerciale tale da poter gestire anche quest'aspetto del progetto: per questo motivo ci appoggiamo a MELACOM ed alla sua esperienza in questo campo.

Sitografia

- https://angularjs.org/
- www.php.net
- www.slimframework.com
- http://developer.apple.com
- https://it.wikipedia.org/wiki/Swift_(linguaggio_di_programmazione)
- “How to work with dates and times in Swift” di Joey Devilla, pubblicato su www.globalnerdy.com
- https://it.wikipedia.org/wiki/Design_pattern
- “How Delegation Works - A Swift Developer’s Guide” di Andrew Bancroft, pubblicato su www.andrewcbancroft.com
- http://realm.io
- “Apple con sandbox obbligatoria, Mac come l'iPhone” di Valerio Porcu, pubblicato su www.tomshw.it
- “Designing for Mobile, Part 1: Information Architecture” di Elaine McVicar, pubblicato su www.uxbooth.com
- “How Do Users Really Hold Mobile Devices?” di Steven Hoober, pubblicato su www.uxmatters.com
- “The core principles of UI Design” di InVision, pubblicato su www.medium.com
- “Understanding User Experience” di Nick Babich, pubblicato su www.medium.com