ITIS “E. Mattei”

ANNO SCOLASTICO 2015/2016

CLASSE 5^D

INDIRIZZO INFORMATICA

ROBERTO CABASSI

SARplay

Un’applicazione nativa per iPad che mostra la deformazione del terreno sulla base di dati satellitari

Introduzione

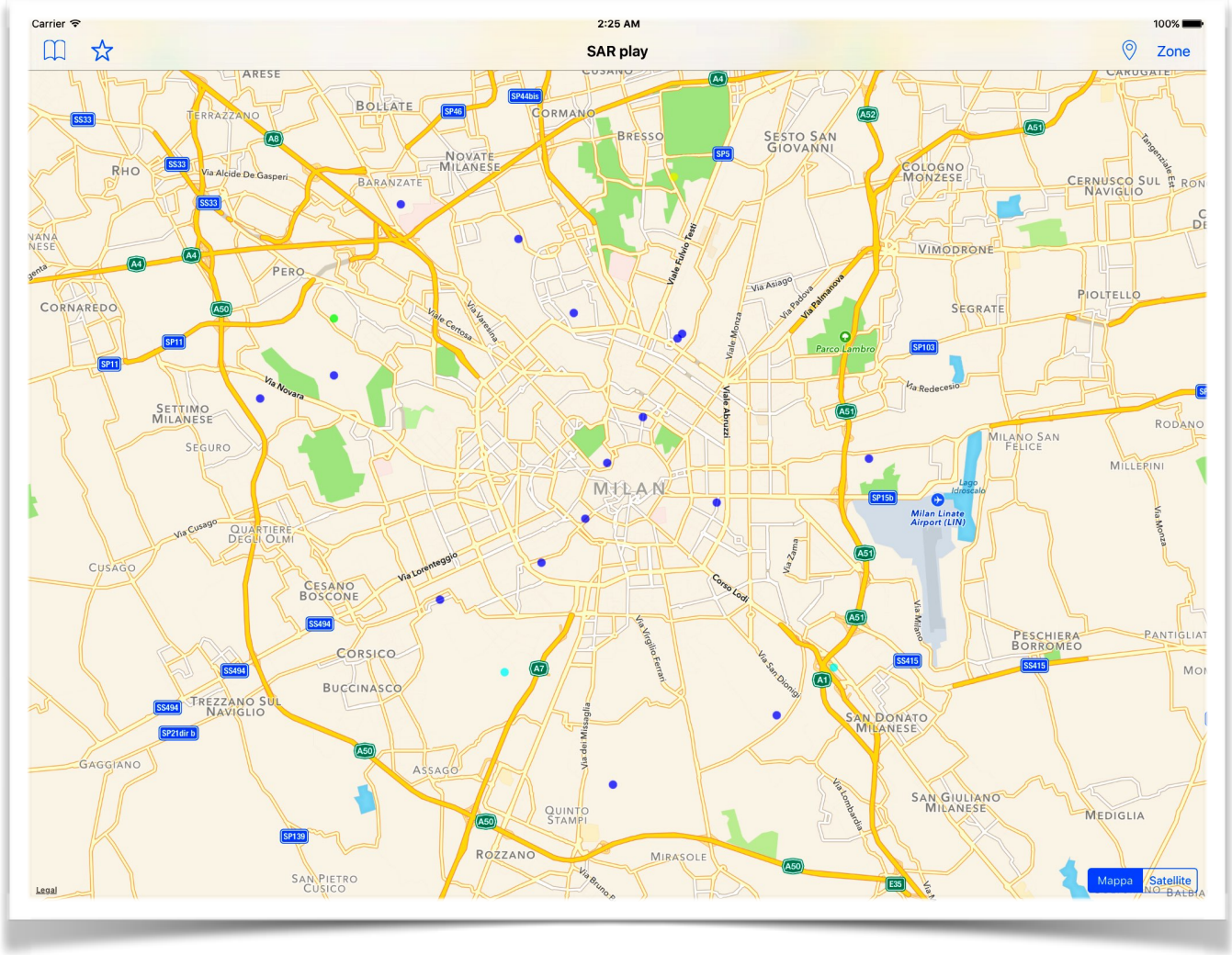
SARplay nasce grazie a due eventi che si sono svolti presso il nostro istituto, durante l'anno scolastico 2015/2016:

1. la collaborazione della scuola con la software house "Visuality srl", la quale, in cerca di una sede a Morbegno e di nuovo personale, si è offerta di istruire e trasmettere la sua esperienza nel mondo Apple verso gli studenti interessati;

2. il coinvolgimento dell'ex professore di Informatica Sabatino Buonanno, il quale, dopo aver insegnato per alcuni anni presso l'I.T.I.S. di Sondrio e il Liceo Scientifico "C. Donegani", è tornato a Napoli, dove attualmente lavora per il CNR (Consiglio Nazionale delle Ricerche), e si occupa di elaborazione di immagini satellitari, riguardanti la deformazione della crosta terreste nel tempo.

L'istituto IREA (Istituto per il Rilevamento Elettromagnetico dell'Ambiente, dipartimento del CNR) ha il compito di raccogliere ed elaborare i dati dei satelliti, i quali vengono poi rappresentati su una mappa, in maniera che essa possa contenere tutti i punti analizzati. La mappa viene messa a disposizione chiedendo l'autorizzazione al CNR, ed è consultabile tramite un web server; tuttavia, la fluidità dell'interfaccia web dipende molto dalla connessione di cui si dispone ed inoltre è parecchio complicata da utilizzare; pertanto un utente medio non sarebbe in grado di farne un buon uso. L'unione di queste due problematiche ci ha spinti a pensare ad un'app per iPad semplice ed intuitiva, da realizzare insieme a Visuality Software e con la collaborazione dell'IREA.

Conformazione dell’app



L’app è formata dai seguenti elementi:

- connettore, ovvero una classe che si interpone tra server e applicazione; il suo compito è quello di scaricare i dati delle deformazioni e renderli disponibili in modo leggibile alla app, tramite una serie di richieste http

- database, contenente tutti i punti scaricati dal server tramite il connettore. È formato dalle entità “point”, “zone” e “deformation”

- mappa, contenuta nella view principale, nella quale sono visibili i punti di deformazione in base alla zona selezionata. È possibile scegliere due tipi di mappa diversi (satellitare e carta politica)

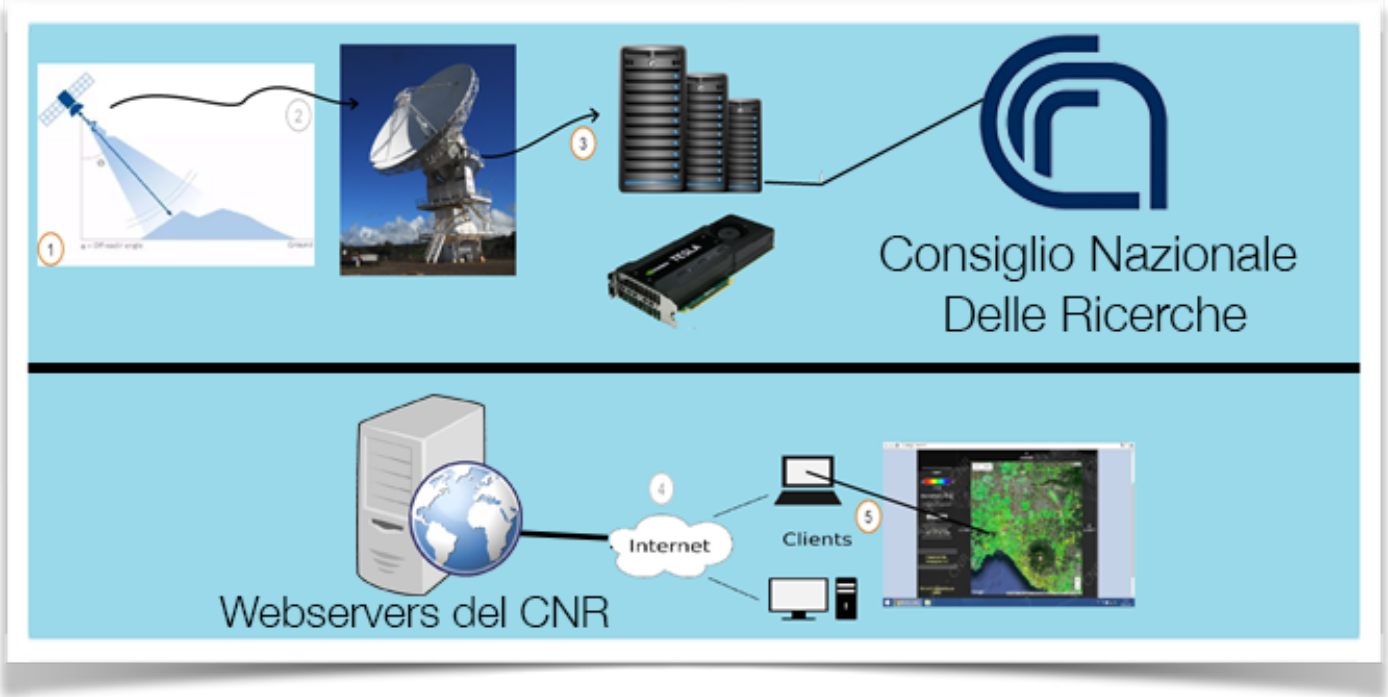
- posizione corrente, in modo che la mappa visualizzi i punti di deformazione della zona in cui si trova l’utente

- menu delle zone disponibili, costituito da una tabella contenente tutte le zone messe a disposizione dal connettore

- menu dei preferiti, costituito anch'esso da una tabella contenente tutte le zone visitate maggiormente dall'utente. Per poter inserire una zona tra i preferiti, si può utilizzare il bottone apposito, rappresentato da una stella

Raccolta, elaborazione e visualizzazione dei dati lato web

Per poter visualizzare la deformazione del terreno nelle zone interessate, i dati devono essere raccolti, elaborati e visualizzati:



Il processo si suddivide in cinque fasi:

1. l'acquisizione dei dati è affidata al satellite, il quale esegue i rilevamenti tramite l'uso di radar SAR.
   I sensori SAR sono associati a specifiche bande dello spettro elettromagnetico. Nelle applicazioni InSAR le bande comunemente utilizzate sono la banda L (frequenza 1-2 GHz, λ ~24 cm), la banda C (frequenza 5-6 GHz, λ ~6 cm) e la banda X (frequenza 8-12 GHz, λ ~3 cm).

Il principio di funzionamento di questi sensori è il seguente: un antenna trasmittente propaga nello spazio un' onda elettromagnetica che, incidendo sulla superficie terrestre, subisce un fenomeno di riflessione. Una parte del campo diffuso torna verso la stazione trasmittente, equipaggiata per la ricezione, dove vengono misurate le sue caratteristiche.
Il dispositivo è in grado di individuare il bersaglio elettromagnetico (funzione di detecting) e, misurando il ritardo temporale tra l'istante di trasmissione e quello di ricezione, valutare la distanza a cui è posizionato, localizzandolo in modo preciso lungo la direzione di puntamento dell'antenna (direzione di range). Il segnale radar relativo ad un bersaglio è caratterizzato da due valori: ampiezza e fase. Questi valori permettono di realizzare due immagini. La fase in particolare racchiude l'informazione più importante ai fini delle applicazioni interferometriche;

2. dopo aver raccolto i dati questi verranno inviati ai centri di trasmissione mediante l'uso di antenne satellitari. Queste faranno da intermediari con i server del CNR ai quali saranno inviate tutte le informazioni in formato immagine in scala di grigi ancora da elaborare;

3. il compito dei server è quello di elaborare le immagini satellitari ed estrapolare dati concreti.
   L'elaborazione consiste nel creare un immagine di interferometria sovrapponendo due o più immagini in scala di grigi fornite dal satellite. L'interferometria è la misurazione delle variazioni della fase del segnale SAR tra due acquisizioni distinte. Questo intero processo, totalmente automatizzato tramite l'uso di programmi in C, è realizzato tramite l'utilizzo di una architettura hardware per elaborazioni in parallelo, chiamata CUDA (Compute Unified Device Architecture), sviluppata da NVIDIA. Essa permette di raggiungere alte prestazioni di computing grazie alla potenza di calcolo delle GPU. Nel caso analizzato vengono utilizzate schede video tesla k20 in un cluster di 23 computer;

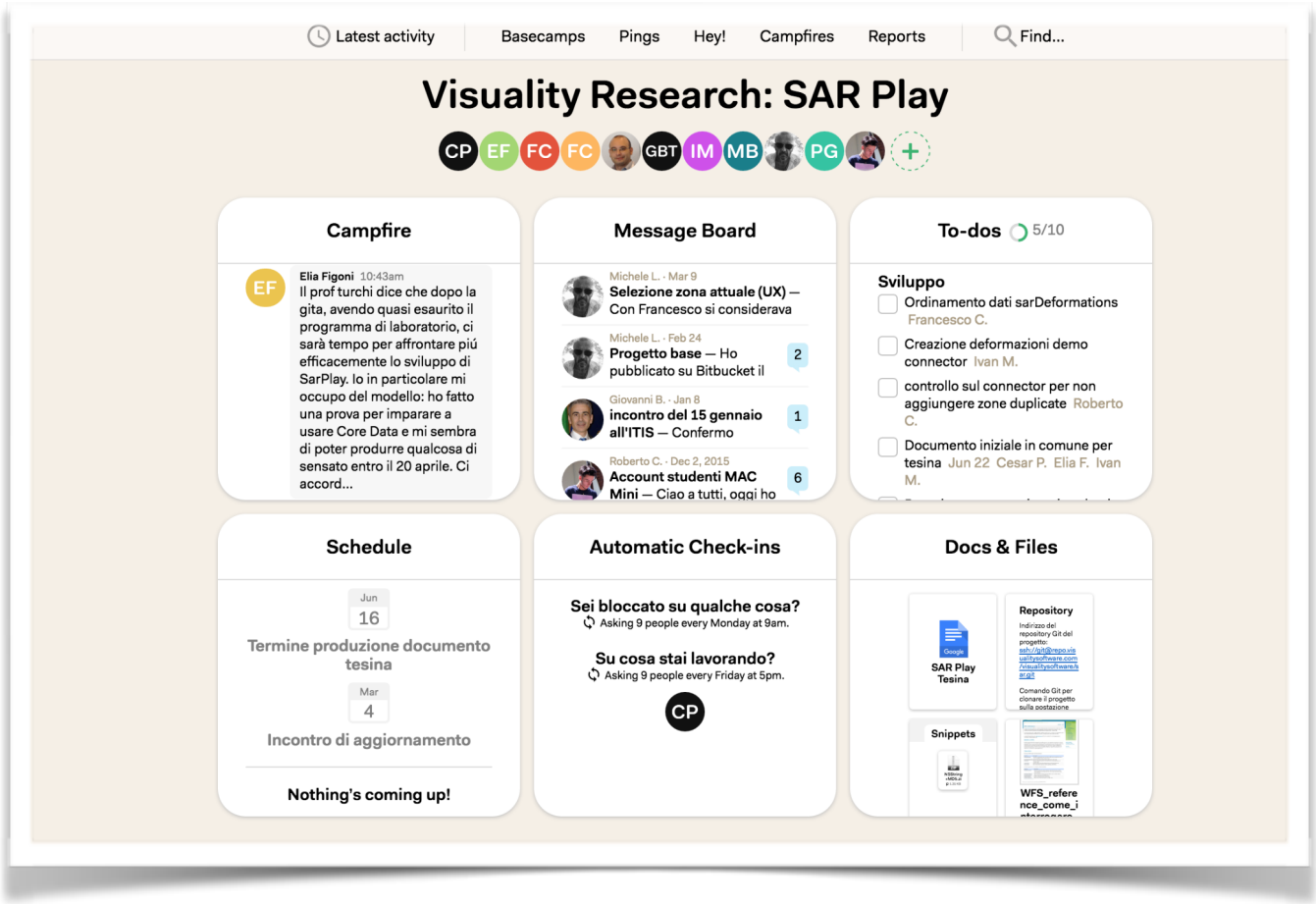
4. i dati elaborati vengono quindi caricati in formato compatibile con gli standard OGC (Open Geospatial Consortium) su di un database collegato a un'applicativo web chiamato GeoServer. Quest'ultimo non fa nient'altro che porsi come interfaccia tra i dati e l'utente;

5. tramite internet e con l'utilizzo di un browser si è in grado di collegarsi al GeoServer gestito dal CNR e visualizzare così su una mappa i dati sotto forma di punti di deformazione. Tramite diverse gradazioni di colore si può capire il movimento del terreno, ed è possibile ispezionare lo storico di deformazione di uno specifico punto.

Metodologia e ambiente di lavoro

Basecamp - Collaborazione e organizzazione del team

Dal momento che SARplay nasce come progetto di gruppo, sin da subito è nata la necessità di trovare un modo per rimanere in contatto tra tutti e di poter condividere tra tutti il progetto di lavoro. Oltre ai numerosi incontri tenuti durante l'anno, sia con i membri di Visuality, sia tra noi ragazzi, rimaneva comunque il problema di rimanere in contatto e della condivisione del materiale, utilizzando mezzi che non fossero le email. Nel nostro caso, Visuality software usa da tempo Basecamp (basecamp.com): un potente strumento per lavorare in team. Pertanto è stato creato uno spazio apposito per il progetto (un basecamp, appunto).



Le principali funzionalità di basecamp sono:

- Ping: sono dei messaggi tra due o più utenti, molto comodi per chiarire, specificare o chiedere concetti in maniera immediata; il funzionamento è lo stesso di ogni programma di chat;
- Hey: è una schermata contenente tutte le notifiche;

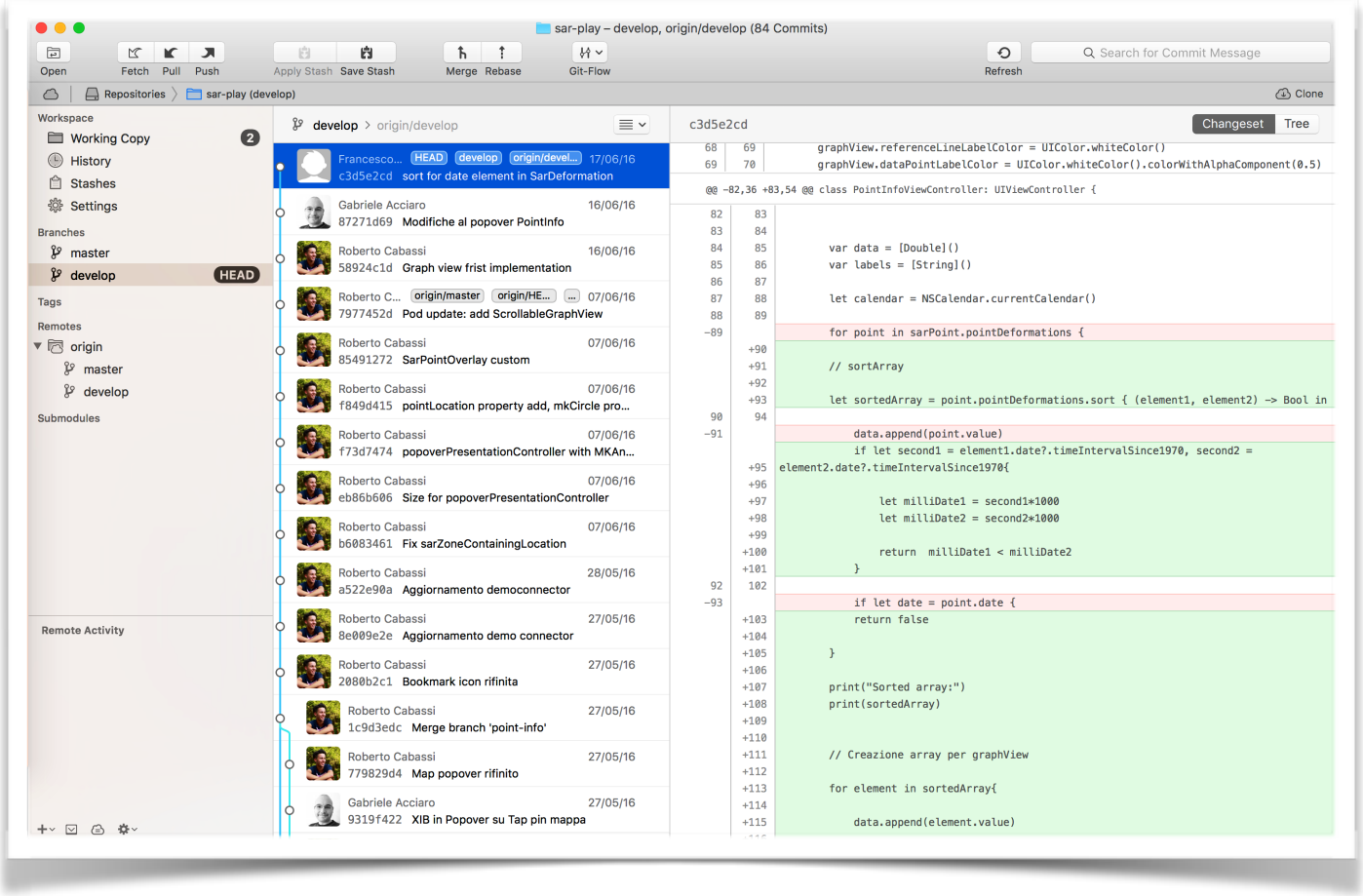
- Campfire: è una chat tra tutti i membri del gruppo;
- To-dos: è una schermata che contiene tutte le cose da fare. Ogni utente può creare dei to-do, e delegare ad un altro utente il compito; una volta terminato il compito, è possibile archiviarlo;
- Schedule: contiene tutte le scadenze dei to-dos. È possibile visualizzare tutti i to-dos di tutti i mesi futuri, oltre a quelli già scaduti;
- Check-ins: sono delle domande programmate che vengono inviate all'utente costantemente. Una domanda può essere "Sei bloccato su qualcosa?", oppure "Su cosa stai lavorando?"; in base alla risposta inviata, si possono ricevere chiarimenti, oppure si può organizzare ancora meglio il lavoro rimanente;
- Docs & files: è uno spazio nel quale vengono caricati file utili per il progetto corrente.

Tuttavia, il codice non viene caricato su basecamp, in quanto è scomodo e quasi impossibile tenere traccia di tutti gli aggiornamenti: per questo Visuality usa BitBucket (http://bitbucket.org) come servizio di source control & versioning. BitBucket, fratello di GitHub, è un repository hosting basato su git.

Git & BitBucket - Source control & versioning

Git è in assoluto il sistema di versioning più utilizzato da parecchio tempo.
Ma cosa vuol dire versioning ?
Per versioning si intende un sistema in grado di tenere traccia di tutti i cambiamenti avvenuti ad uno o più files nel tempo, così da poterne recuperare una versione precedente in qualunque momento, capire come è mutato un progetto durante il suo corso, sapere chi ha modificato qualcosa e quando. Lavorare appoggiandosi a git significa che qualsiasi errore o malfunzionamento introdotto dal programmatore può essere ripristinato in pochi secondi.
Per usare questo sistema si può scegliere di usare il terminale e dare i vari comandi operativi tramite CLI oppure di appoggiarsi a vari programmi che mettono a disposizione una GUI di facile utilizzo.

L’esempio riportato raffigura Tower. Sulla sinistra i branch e lo spazio di lavoro, al centro la lista di tutti i commit e sulla destra il dettaglio delle modiche relative al commit selezionato.



Git considera i propri dati come una serie di istantanee (snapshot) di un mini filesystem. Ogni volta che l’utente effettua un commit, git salva lo stato del proprio progetto, fondamentalmente fa un’immagine di tutti i file presenti in quel momento, salvando un riferimento allo snapshot. Se alcuni file non sono stati modificati, git non li clona ma crea un collegamento agli stessi file della versione precedente.

<u>Un progetto git è composto dai seguenti elementi</u>:

- Working dir o directory di lavoro che contiene i file appartenenti alla versione corrente del progetto sulla quale l’utente sta lavorando;
- Index o Stage che contiene i file in transito, cioè quelli candidati ad essere committati;
- Head che contiene gli ultimi file committati.

É possibile inizializzare un progetto git in due modi:
- definire un nostro progetto preesistente come git Repository;
- clonare un repository git esistente da un server.

Storico dei commit, il comando log
Mediante il comando log è possibile visualizzare l'elenco degli ultimi commit effettuati. Ciascun commit è contrassegnato da un codice SHA-1 univoco, la data in cui è stato effettuato e tutti i riferimenti dell'autore. Il comando, lanciato senza argomenti, mostra i risultati in ordine cronologico inverso, quello più recente è mostrato all'inizio.
Naturalmente sono disponibili numerosi argomenti opzionali utilizzabili con il comando log che permettono di filtrare l'output.

```
>> git log
```

Stato dei file, il comando status
Mediante il comando status, è possibile analizzare lo stato dei file. GIT ci indicherà quali sono i file modificati rispetto allo snapshot precedente e quali quelli già aggiunti all'area STAGE.

```
>> git status
```

Il modus operandi più diffuso è il seguente:
- fare pull (git pull) per ottenere le ultime modifiche dal server
- aggiornare il progetto effettuando le modifiche desiderate
- fare il commit delle modifiche (git commit -m "commento")
- fare push per pubblicare le proprie modifiche e renderle disponibili per chiunque (git push)

Oltre alle funzionalità base qui elencate, git offre moltissime altre features; per maggiori informazioni si rimanda al sito di supporto di git (www.git-scm.org).

Xcode - Integrated Development Environment

Apple, per tutti i suoi dispositivi ha un solo IDE, Xcode per l'appunto.
Esso contiene - come ci si aspetta da un IDE - tutti gli strumenti per lo sviluppo, il testing e la distribuzione del software.
Xcode permette la creazione di applicazioni per iPhone, iPad, Mac, appleWatch e appleTV.

In particolare vi sono due grandi famiglie di framework su cui Xcode permette di lavorare:

- Cocoa
- Cocoa touch

Cocoa è costruito per macOS e Cocoa touch per iOS, tvOS e watchOS. Siccome un approfondimento di Xcode sarebbe lungo oltremodo, per le specifiche si rimanda a Apple Developer (developer.apple.com)

Swift - Il linguaggio di programmazione

Come per ogni progetto software, anche noi per la realizzazione di SARplay abbiamo dovuto scegliere un linguaggio di programmazione.
Avendo scelto Apple come ambiente di sviluppo a priori le scelte ricadono su Objective-C e Swift.
Obj-C è nato negli anni ’80 ed è sempre stato utilizzato da Apple come unico linguaggio di programmazione proveniente dal C. Nel 2014 è nato Swift, questa volta completamente e interamente progettato da Apple per essere veloce, moderno, interattivo e sicuro.

Le novità introdotte da Swift sono così tante che è quasi impossibile elencarle tutte; è da notare come Swift erediti caratteristiche da tantissimi linguaggi di programmazione prendendo il meglio da ognuno di essi.
Un esempio eclatante è che in Swift un errore del tipo NullPointerException che tutti i programmatori Java conoscono (e odiano) non esiste. Questo perché effettuare una chiamata su di un oggetto nullo non è possibile. Non perché gli oggetti non saranno mai nulli, ma semplicemente perché è impossibile scrivere del codice che possa effettuare una chiamata di questo tipo.
Altro caso è quello delle Extension: questo tipo di funzionalità unica nel suo genere permette di aggiungere funzionalità ad una classe, una struttura, una enumerazione o un protocollo. Questo include la possibilità di estendere tipi per il quale non si ha accesso al codice sorgente originale (questa pratica viene chiamata retroactive modeling).
Per le specifiche si rimanda a swift.org e a Apple Developer.

A seguito della suddivisione del lavoro, mi è stata assegnata la realizzazione del connettore, parte fondamentale in quanto è responsabile di mettere a disposizione i dati da consultare.

SARplayConnector

Il connettore, come accennato in precedenza si occupa di scaricare i dati dai server del CNR e di passarli al modello che in seguito li salverà nel database interno alla applicazione. Per arrivare alla realizzazione dello stesso ho dovuto affrontare diversi passi:

- capire come accedere al GeoServer
- studiare lo standard WFS (Web Feature Service)
- trovare un formato dati fornito dalle API di GeoServer compatibile e il più semplice da “parsare”
- studiare le tecnologie messe a disposizione da Apple per lo scaricamento di dati in background
- codificare il connettore

GeoServer - Accesso ai dati

GeoServer (applicativo web utilizzato dal CNR per la visualizzazione dei dati) utilizza due protocolli per funzionare:

- WMS: Web Map Service
- WFS: Web Feature Service

WMS è il protocollo il cui compito è quello di scaricare e rappresentare la mappa vera e propria. Sia essa una mappa satellitare, politica, fisica, geologica ecc. In sostanza scarica le tiles che compongono la mappa e le dispone nello spazio di visualizzazione in maniera georeferenziata. E immaginabile come una sorta di agente che si occupa di comporre un puzzle per poi mostrare il risultato finale incorniciato in un quadretto.

WFS invece è il protocollo chi si occupa di scaricare e posizionare sulla mappa punti, immagini, polilinee, geofence, cerchi, forme di vario genere e tutto ciò che di visibile si possa sovrapporre a una mappa.

WFS - Web Feature Service

Le deformazioni del terreno su GeoServer vengono rappresentate tramite un dodecagono con contorno colorato per capire il tipo di spostamento. Questi dodecagoni sono subset di feature. Questo vuol dire, informaticamente parlando, che un dodecagono eredita da feature e implementa nuove caratteristiche e proprietà.

Questo processo avviene per tutti gli elementi grafici rappresentabili con WFS.
WFS si interfaccia con un database Postgres da cui ottiene i dati necessari per costruire graficamente la feature e posizionarla sulla mappa. Questo avviene tramite una richiesta HTTP asincrona verso il GeoServer che restituirà un documento XML mediante la tecnologia AJAX.
La richiesta è del tipo:

```
http://server.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:<Nome della zona>
```

Questa query può essere usata per ottenere tutte le features contenute in una determinata zona specificando l'identificatore della zona stessa. Si può inoltre specificare il formato dei dati se non lo si desidera in XML.
Altri formati supportati sono:

1. JPEG image
2. PDF document
3. PNG image
4. Zipped Shapefile
5. GML 2.0 Format
6. GML 3.1.1 Format
7. CSV Format
8. Excel Format
9. GeoJSON Format
10. KML Format
11. Google Earth Format
12. Tiles Format

A priori è stato scartato tutto a parte GML 2.0, GML 3.1.1 e GeoJSON. GML è un derivato dell' XML e GeoJSON è un derivato del JSON. Questi ultimi sono i due formati più semplici da elaborare e interpretare.
Di questi ho scelto il file in formato GeoJSON (www.json.org per le specifiche) e scaricato l'area di Roma per effettuare dei test. Il file pesa 417 MB.

Ecco un esempio del file (ovviamente tagliato):

```
// Begin JSON Document
{
"type":"FeatureCollection",
"totalFeatures":2,
"features":
	[
		{
			"type":"Feature",
			"id":"csk_roma.fid-68b99caf_1522fa8eedd_-faf",
			"geometry":
			{
				"type":"Polygon",
				"coordinates":
				[
					[
						[12.2461841605744,41.7362017017566],
						[12.2460957779932,41.7361792358623],
						[12.2460042395264,41.736192879839],
						[12.2459340727034,41.7362389778141],
						[12.2459040786496,41.7363051779241],
						[12.2459222943581,41.7363737419347],
						[12.2459838390735,41.7364262981546],
						[12.2460722219571,41.7364487641356],
						[12.246163760776,41.7364351201044],
						[12.2462339276487,41.736389021988],
						[12.2462639214001,41.7363228217914],
						[12.2462457053395,41.7362542578353],
						[12.2461841605744,41.7362017017566]
					]
				]
			},
			"geometry_name":"the_geom",
			"properties":
			{
				"id":1,
				"plot":"<a href=\"#\" onClick=\”window.open('http://
IP_ADDRESS/static/plot.php?id=1&table=csk_roma_0','PLOT','width=700, height=400,
location=no','_blank')\">Plot data<\/a>",
				"east":270977.84,
				"north":4624165.5,
				"lat":41.736313,
				"lon":12.246084,
				"azimuth":354,
				"range":0,
				"coherence":0.844166,
				"velocity":0.158807,
				"elevation":5.09221,
				"z2010_5115":0,
				"z2010_5334":-0.0446284,
				"z2010_5581":-0.349941,
				"z2010_5773":-0.00378048,
				"z2010_5976":0.175714,
				"z2010_6195":0.0349877,
				"z2010_6864":0.233866,
				"z2010_7275":-0.113367,
				"z2010_7752":-0.221647,
				"z2010_8191":-0.265989,
				"z2011_2248":0.312297,
				"z2011_2670":0.0925604,
				"z2011_3108":-0.280589,
				"z2011_3558":-0.20474,
				"z2011_3997":-0.194254,
				"z2011_4419":0.210427,
				"z2011_4857":0.423169,
				"z2011_5307":0.41748,
				"z2011_6606":0.408881,
				"z2011_7028":-0.207416,
				"z2011_7467":0.0135972
			}
```

```
		},

		{
			"type":"Feature",
			"id":"csk_roma.fid-68b99caf_1522fa8eedd_-fae",
			"geometry":
			{
				"type":"Polygon",
				"coordinates":
				[
					[
						[12.3665579379854,41.7292305959677],
						[12.3664695228108,41.7292082235154],
						[12.3663780201262,41.7292219642424],
						[12.3663079478732,41.7292681363521],
						[12.3662780818378,41.7293343681167],
						[12.3662964247108,41.7294029128199],
						[12.366358061663,41.7294554039439],
						[12.3664464771406,41.7294777764826],
						[12.3665379801768,41.7294640357007],
						[12.3666080524784,41.7294178634497],
						[12.3666379182108,41.7293516315988],
						[12.3666195749861,41.7292830869504],
						[12.3665579379854,41.7292305959677]
					]
				]
			},
			"geometry_name":"the_geom",
			"properties":
			{
				"id":1,
				"plot":"<a href=\"#\" onClick=\"window.open('http://
IP_ADDRESS/static/plot.php?id=1&table=csk_roma_1','PLOT','width=700, height=400,
location=no','_blank')\">Plot data<\/a>",
				"east":280965.5,
				"north":4623078,
				"lat":41.729343,
				"lon":12.366458,
				"azimuth":276,
				"range":279,
				"coherence":0.972302,
				"velocity":-0.00610888,
				"elevation":9.30568,
				"z2010_5115":0,
				"z2010_5334":0.00768796,
				"z2010_5581":0.0592088,
				"z2010_5773":0.0775075,
				"z2010_5976":0.0660014,
				"z2010_6195":-0.274582,
				"z2010_6864":0.235149,
				"z2010_7275":0.00518118,
				"z2010_7752":-0.54853,
				"z2010_8191":-0.954134,
				"z2011_2248":-0.222902,
				"z2011_2670":-0.00346928,
				"z2011_3108":0.0973639,
				"z2011_3558":0.128939,
				"z2011_3997":-0.195534,
				"z2011_4419":0.0562914,
				"z2011_4857":0.156633,
				"z2011_5307":-0.173955,
				"z2011_6606":-0.330861,
				"z2011_7028":-0.308144,
				"z2011_7467":0.0955062
			}
		}
	]
}

// End JSON Document
```

Si ha quindi una collezione di features: ogni features come già accennato corrisponde a un pallino sulla mappa.

Ogni pallino ha dei dati che lo caratterizzano:

- type: indica il tipo di oggetto, che è una feature
- id: identifica la feature in maniera univoca
- geometry: è un oggetto che contiene:
  - type: è sempre Polygon a quanto ho potuto vedere ed è il luogo geometrico della feature (infatti si può osservare che in realtà non sono circonferenze quelle visualizzate sulla mappa, bensì dodecagoni come precedentemente accennato)
  - coordinates: è l'insieme dei punti che definisce il poligono
- geometry_name: è il nome della geometria che nello standard GeoJSON è sempre "the_geom"
- properties: è un oggetto che contiene:
  - id: è ciò che identifica l'oggetto properties, ma visto che ogni feature ha solo un oggetto properties è sempre 1
  - plot: è l'indirizzo HTTP del grafico di deformazione del terreno che mostra l'andamento nel tempo
  - east: riferimento in gradi rispetto all'origine X della zona
  - north: riferimento in gradi rispetto all'origine Y della zona
  - lat: indica la latitudine della feature
  - lon: indica la longitudine della feature
  - azimut

- range
- coherence: è un indice di affidabilità dei dati
- velocity: è la velocità di spostamento misurata
- elevation: è l'elevazione AMSL della feature
- una serie di campi zAAAA_xxxx. AAAA indica l'anno in 4 cifre, xxxx sono le ore passate dall'inizio dell'anno indicato. Sono i dati utilizzati per costruire il plot

Un esempio di plot rappresentante i dati della prima feature nel documento JSON qui sopra:

JSON parsing

Dopo aver interpretato la struttura del file GeoJSON, la prosecuzione naturale dello sviluppo è stata quella di scrivere del codice che fosse in grado di analizzare il file, che nient'altro è se non una stringa, attraverso le API JSON di Apple e successivamente creare delle entità per il database.

Il file dopo essere stato scaricato viene salvato in maniera temporanea nella Document directory della applicazione per poi essere serializzato. Di seguito uno snippet di codice che mostra la funzione che carica il file JSON in maniera uncached:

```
private func loadJSON() -> [String: AnyObject]? {

    do {

        guard let url = NSBundle.mainBundle().URLForResource("loadedZone", withExtension: "json")

        else {
                print("Error loading json file from resource")
                return nil
        }

        let data = try NSData(contentsOfURL: url, options: .DataReadingUncached)

        let object = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)

        if let dict = object as? [String: AnyObject] {

            return dict
        }
    }
    catch {
        print("Error parsing json: \(error)")
    }

    return nil
}
```

Questo tipo di lettura permette una velocità maggiore in quando il file non verrà caricato totalmente in memoria per poi essere serializzato, bensì verrà letto un carattere alla volta e trasformato in un NSDictionary del tipo [String: AnyObject].

Ottenuto il dizionario, questo viene analizzato e confrontato con un pattern. Per pattern si intende una struttura in cui viene indicato il tipo di dati che un'implementazione della stessa deve contenere. E' possibile paragonarla alla dichiarazione di una struttura in C mediante typedef struct {}.
Se il confronto con il pattern corrisponde vuol dire che è possibile creare un oggetto di tipo feature dal record contenuto nel dizionario e pertanto esso viene creato e inserito in un array che successivamente verrà passato al modello.

Scaricamento dati in background - Background fetch & NSURLSession

L'ultimo problema della lista e forse il più importante, è stato scaricare effettivamente i files delle zone sul dispositivo. Data la dimensione media di una zona (~500MB) è chiaro come la app debba necessariamente effettuare questo scaricamento in maniera asincrona sfruttando un thread separato e soprattuto che il download possa proseguire anche dopo che la app sia passata dallo stato di foreground a quello di background.

IOS a differenza di Android, quando una app passa in background la congela istantaneamente. Questa operazione viene effettuata per ovvi motivi di risparmio energetico. E' difatti questo comportamento uno dei motivi per il quale un iPhone ha la stessa autonomia di un terminale Android di fascia alta che però monta una batteria con capacità doppia. Apparentemente non sembrerebbe possibile effettuare operazioni in background in quando la app è congelata. Apple ha deciso di introdurre delle eccezioni chiamate Background Modes. Queste funzionalità permettono di eseguire le più svariate operazioni in una coda a bassa priorità del sistema operativo senza interferire con altre operazioni che l'utente sta effettuando.
In SARplay ci serviamo della Background Mode chiamata Background fetch. Quest'ultima ci permette di scaricare il file da internet ed eseguire le operazioni di parsing precedentemente descritte.

Viene dunque configurata una NSBackgroundSessionConfiguration che permettere di eseguire una NSURLSession asincrona e in background:

```
let sessionIdentifier =  "url_session_background_download"

var configuration = NSURLSessionConfiguration.backgroundSessionConfigurationWithIdentifier
    (sessionIdentifier)
```

Una volta configurata la sessione, il procedimento continua analogamente ai download con app in foreground, cioè facendo partire la sessione e utilizzando un Delegate per intercettare e gestire l'evento di fine download. Alla fine del download, la NSURLSession chiamerà dunque il Delegate che a sua volta si preoccuperà di prendere il file JSON scaricato sotto forma di NSData e salvarlo nella Document directory.

Librerie di terze parti

Per la realizzazione di SARplay è stato necessario appoggiarsi a software scritto da altre persone in quanto realizzare una soluzione di persistenza (database) e la creazione di grafici sarebbe stato inutile data la moltitudine di librerie già presenti e la loro elevata qualità.

CocoaPods - Dependency manager

CocoaPods è un dependency manager per progetti in Swift e Objective-C. Ha più di 18.000 librerie ed è utile in quanto è facilmente scalabile e sopratutto facile da mantenere. Se infatti una libreria viene aggiornata sarà sufficiente eseguire un comando sulla CLI per aggiornare la libreria in locale.
CocoaPods è realizzato in Ruby ed è facilmente installabile sul proprio Mac aprendo il terminale e dando il seguente comando:

```
>> sudo gem install cocoapods
```

```
platform :ios, '9.0'

use_frameworks!

target 'SARplay' do

pod 'RealmSwift'
pod 'ScrollableGraphView'

end
```

Per abilitare CocoaPods nel nostro progetto, sempre tramite CLI sarà sufficiente recarsi nella cartella del progetto e dare il comando:

```
>> pod init
```

Pod creerà dunque un file chiamato “Podfile”. E’ riportato a fianco quello del nostro progetto.

Una volta creato e salvato il Podfile, per completare l’installazione delle librerie si esegue il comando:

```
>> pod install
```

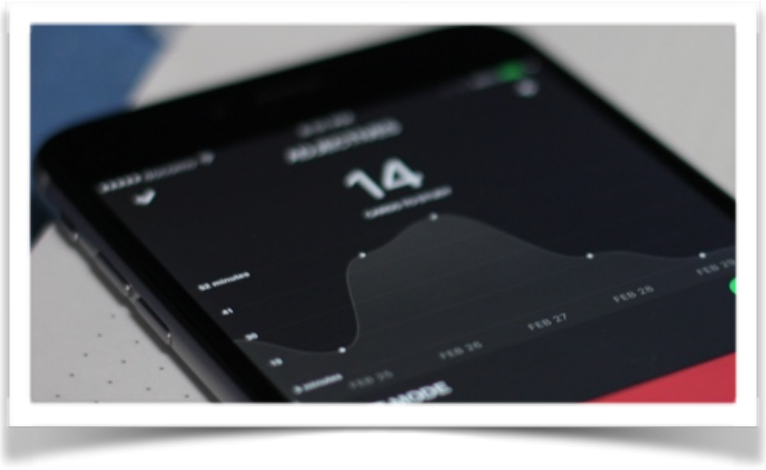
Una volta terminata la procedura, basterà aprire il progetto e tutte le librerie saranno disponibili e funzionanti.

RealmSwift

Realm è una soluzione di persistenza nata come progetto interno a Zynga (una softwarehouse famosa per la moltitudine di giochi pubblicati su Facebook) e in seguito resa open source e disponibile a tutti gratuitamente.
Inizialmente SARplay utilizzava CoreData che è un database ORM basato su SQLite sviluppato da Apple. Purtroppo una volta testato il codice ci siamo accorti che CoreData è troppo lento per gestire una moltitudine di dati e relazioni come quelli presenti nella app. Siamo dunque giunti alla decisione di utilizzare Realm come database, che oltre a essere più veloce di circa l'80% è molto più semplice e richiede molte meno righe di codice per ottenere lo stesso risultato.

ScrollableGraphView

ScrollableGraphView è una libreria per grafici adattivi nata, come Realm, per supportare un progetto privato e in seguito resa open source tramite GitHub.



Il suo uso è davvero semplice e richiede pochissime righe di codice.

Si crea la vista contenente il grafico e la si aggiunge al ViewController

```
let graphView = ScrollableGraphView(frame: someFrame)
let data: [Double] = [4, 8, 15, 16, 23, 42]
let labels = ["one", "two", "three", "four", "five", "six"]
graphView.setData(data, withLabels: labels)
```

```
someViewController.view.addSubview(graphView)
```