Resiliència a nivell d’aplicació



1. Resiliència Software: Estat de l’art
2. Principis de Resiliència
3. Pla del projecte
4. Execució del projecte

# Aplicació

Per la implementació dels principis de resiliència hem escollit aprofitar una aplicació que és el resultat de l’assignatura de PES anomenada Hangaround. L’aplicació té coma objectiu facilitar l’accés a vida social de les persones amb discapacitats motrius. El cas emblemàtic d’aquest públic objectiu són les persones que utilitzen cadires de rodes. Amb l’ajuda de la comunitat d’usuaris que la fan servir, l’aplicació proporciona informació sobre el nivell d’adaptació d’espais o locals d’accés públic. A la Figura 1 es pot veure que l’aplicació segueix el paradigma de client-servidor i les seves dependències, tant la part client com de la part servidor.

C:\Users\lucmihai\Documents\TFG-FIB\Photos\Infraestructura.png

Figura 1. Infraestructura i dependències.

## Servidor: MobService

La part servidor és un servei web amb arquitectura API REST, implementada en Java, corre sobre un Tomcat i esta allotjada en Heroku. S’encarrega principalment de la persistència de les dades. Té com a dependència principal l’API de Foursquare que utilitza per a proveir llocs al voltant de la ubicació o direcció cercada. En els casos d’ús només farem servir cerques de ciutats. Foursquare proporciona els llocs: locals, bars, museus, etc. Aquests llocs es guarden en una base de dades relacional en el servidor.

La cerca de llocs es fa en Foursquare però des del la nostra API. Els paràmetres que utilitzarem per fer la cerca són: *near* i *limit*. Amb el primer especifiquem una ciutat i amb el segon limitem en numero de resultats que desitgem mostrar. En els casos d’ús variarem el seu valor entre 1 i 25 segons convingui. Per a cada lloc que no és troba ja en el backend és guarda, amb un thread en background, amb un nivell d’adaptabilitat desconegut: UNKNOWN.

Els usuaris poden fer valoracions en relació amb el nivell d’adaptabilitat dels llocs. Hi ha quatre nivells: UNKNOWN – desconegut, UNADAPTED – sense adaptar, PARTIAL – parcial, i TOTAL. La part servidor determina el nivell d’adaptabilitat d’un lloc en funció de les valoracions que han fet els usuaris sobre aquell lloc. Una valoració requereix tres paràmetres: accés, serveis(wc) i ascensor. Un lloc té com a nivell d’adaptabilitat la que correspon a la ultima valoració que li han fet.

El primer es refereix a l’accés i mobilitat dins del perímetre del lloc, principalment rampa a l’entrada i amplitud dels passadissos. El segon comprova l’existència de serveis adaptats. Tant el primer com el segon tenen una codificació binaria de certo fals. Finalment la presència o absència de l’ascensor és el tercer paràmetre. Aquest només tindria sentit avaluar-lo en el cas en què el local té més d’una planta. La codificació d’aquest paràmetre consisteix en un enumerable amb els següents valors: HAS – el lloc disposa d’ascensor destinat a l’ús públic, NO\_NEED – el lloc només té una planta, finalment HAS\_NOT – el lloc té més d’una planta però no disposa d’ascensor.

## Client: Hangaround

En la part client disposem d’una aplicació mòbil per a dispositius android. Esta implementada en Java, aplicació nativa. Com a principals dependències té l’API de Facebook per facilitar l’accés dels usuaris sense la necessitat de crear un compte nou. Els llocs que proporciona Foursquare es representen en un mapa. Per mostrar els llocs s’utilitza l’API de Google Maps.

Després de fer la cerca de llocs com s’explica a dalt, per a cada lloc recuperat de Foursquare es busca el nivell d’adaptabilitat que té en el backend. Actualitzant un per un els llocs mostrats, amb marcadors de l’API de Google Maps, en el mapa. Aquests marcadors mostren el nivell d’adaptabilitat dels llocs que representen segons la llegenda de colors que es pot veure a la Taula 2.



Taula 2. Codi de colors dels marcadors en el mapa.

Clicant els marcadors es pot fer una valoració sobre aquell lloc mitjançant el formulari que apareix mitjançant un pop-up, com es pot veure a la Figura 3.



Figura 3. Pop-up amb formulari per fer una votació.

# Casos d’ús

Els principis de resiliència que hem implementat són els següents, mode offline, recuperació d’excepcions NullPointerException i vigilància a nivell de mètode mitjançant un mòdul d’intel·ligència artificial.

## Mode offline

Com ja hem mencionat(link als principis), la motivació d’aquest principi és donar el màxim de funcionalitats en cas de desconnexió. Per una banda prova de mitigar els efectes de perdre la connexió al backend. Per l’altre banda la pèrdua de connexió del mòbil a internet. La capa de resiliència s’encarregarà d’abordar els dos problemes.

### Pèrdua de connexió al backend

La xarxa pot fallar o l’aplicació que s’executa en el backend pot fallar. Per un error propi o per culpa d’una dependència. Siguin quines siguin les causes o els causants d’aquests errors, el backend pot arribar a no ser disponible[[1]](#footnote-1). Tal com ha estat dissenyada i implementada l’aplicació no és capaç d’oferir cap funcionalitat. Qualsevol intent de connexió al backend retorna un missatge d’error. En termes de disponibilitat, la indisponiblitat del backend provoca la indisponiblitat de totes les funcionalitats de l’aplicació.

Com ja havíem dit, vam implementar una cache per a cada una de les dues funcionalitats bàsiques: consultar llocs i el seu nivell d’adaptabilitat i valorar llocs. La cache, en els dos casos, és persistent i es troba en el dispositiu; no depenem de la xarxa i per tant és prou ràpida. Les cache són persistents, en una base de dades NoSQL (key-value) proporcionada per l’Android SDK, SharedPreferences.

En primer lloc hem implementat una cache per a les cerques; cerca-resultat. On la cerca és el text cercat en format String i el resultat és el conjunt de llocs que torna el backend per aquella cerca en format JSONArray. La lectura de la cache no es fa indefinidament. L’usuari esta informat que s’ha perdut la connexió amb el servidor de l’aplicació. Però no només això sino que en background s’inicia un thread que cada deu segons va comprovant si s’ha restablert la connexió amb el servidor. En tornar a estar disponible el servidor, l’usuari torna a estar notificat i la funcionalitat torna a estar al cent per cent disponible. El mòdul de resiliència ha provocat que l’aplicació toleri la fallida del backend i que torni a l’estat òptim tan aviat com sigui possible.

En segon lloc hem implementat una cache per a les valoracions. Encara que el backend no estigui disponible, l’usuari pot interaccionar amb l’aplicació valorant llocs. La cache guardarà totes les valoracions efectuades per part de l’usuari mentre està sense connexió amb el servidor. Quan es detecti que el servidor torna a estar disponible, entrada per entrada s’envien les valoracions de la cache per guardar-les en el backend.

### Pèrdua de connexió a internet

El segon problema que havíem mencionant i que te a veure amb la connexió és el fet que el dispositiu es trobi en una zona de cobertura insuficient. Tot i ser semblant al primer hi ha una problemàtica afegida ja que l’aplicació té més dependències no nomes amb el backend. Per una banda el login mitjançant l’API de Facebook i el mapa per ubicar els llocs proporcionada per l’API de Google Maps. Degut als termes d’ús d’aquesta darrera API, Captura 4, hem hagut de canviar la representació dels llocs quan el dispositiu no esta connectat. Només està autoritzat fer ús de cache dins del servei que Google proporciona.

Per no canviar tota la implementació de l’aplicació original s’ha decidit fer una vista en format llista amb els llocs, nova disponible a la Captura 5. Utilitzant l’API d’OpenStreetMap no s’hagués hagut de canviar la representació; menys d’intrusió.



Captura 4. Extret dels termes d’ús de Google Maps API



Captura 5. Nova vista per mostrar els llocs en mode offline

Hi ha una limitació del model escollit deguda a no fer pre-fetch i per tant els llocs disponibles per a consultar sense cap impediment són només els llocs prèviament cercats. L’usuari pot trobar-se utilitzant les dades i encara sent potents l’autonomia dels dispositius mòbils està condicionada per la capacitat de la bateria. Tenint en compte aquests dos factors hem considerat que el millor enfocament seria no consumir ni dades ni espai en el dispositiu per fer pre-fetch.

## Errors interns: NullPointerException

Els errors s’amaguen en el codi de qualsevol projecte de software. Passem, doncs, de les dependències d’una aplicació als errors inesperats que pot produir la mateixa aplicació. Degut a que alguns errors no s’arriben a trobar en la fase de test es solen capturar tots els errors i crear vistes personalitzades d’una varietat d’errors. Hi ha varies raons per actuar així, com ara la seguretat o inclús la confiança de l’usuari. Però aquí volem aplicar un segon principi de resiliència. No esperem que l’aplicació simplement informi de l’error sino que provi d’arreglar-lo. Aquest principi l’aplicarem a la part servidor de l’aplicació.

Suposant que l’ús que li dona un usuari encara i sent correcte provoca una excepció, com per exemple: NullPointerException. El comportament que desitgem que l’aplicació tingui en aquest cas és de provar de recuperar-se d’aquest error i acabar servint l’usuari amb la resposta desitjada. Les limitacions temporals han impedit que la construcció d’un agent Java que faci la substitució de classes en calent quedi fora de l’àmbit del projecte. En conseqüència hem emprat un plugin ja existent, jRebel.

jRebel és capaç fer la substitució en calent, sense que calgui reiniciar el servidor, sense haver d’interrompre[[2]](#endnote-1) la interacció de l’usuari amb l’aplicació. El seu funcionament consisteix en vigilar per una banda l’aplicació desplegada i per l’altre el directori target. En detectar que el timestamp és diferent executa la substitució. Això ens porta al segon ingredient necessari per que aquest principi es pugui aplicar. El mòdul de resiliència se li ha d’especificar la ruta al repositori de classes (l’equivalent al directori target en l’ús convencional del plugin). Aquest repositori de classes compilades podria estar estructurat de manera que proveeixi diverses versions.

En aquest exemple, de caire didàctic, no hem tingut en compte aquesta possibilitat. Per simplicitat també, per l’execució de la demostració el plugin s’ha configurat per un Tomcat que corre en local. L’error s’ha introduït en la primera versió d’una classe FourSquareController i sense altres dependències[[3]](#footnote-2). Però en un cas real s’hauria de buscar la classe i les seves classes dependents, ja que s’haurien de substituir totes en conjunt. Aquestes podrien estar agrupades en *minijars* per facilitar l’aplicació d’aquest principi. Per tant, recomanem aplicar aquest principi en arquitectures de microserveis.

Un aspecte important aquí seria s’hauria de decidir si s’aplica la substitució en temps real, però amb el risc que la solució torni a donar algun altre error, o executar algunes proves abans de fer la substitució.

Considerem com a cas ideal d’aplicar resiliència a una aplicació mitjançant un disseny extensible, es a dir, una arquitectura de plugins. Perseguim un nivell mínim d’intrusisme, de tal manera que els principis de resiliència que es considerin es vagin afegint a l’aplicació sense estar acoblats. En aquest sentit hem implementat els primers dos principis de resiliència amb el paradigma de programació orientada als aspectes(AOP). Degut a limitacions imposades per Android, ha calgut implementar de manera acoblada els mètodes que permeten comprovar en background si s’ha establert la connexió amb el servidor o si el dispositiu torna a tenir connexió a internet.

Per raons de temps i costos s’han pres algunes decisions en quant a la implementació. En projectes reals l’experiència dels desenvolupadors i les necessitats especifiques de cada aplicació donen peu a explorar nous principis de resiliència.

## Vigilància a nivell de mètode

1. El backend perd la disponibilitat si dona una resposta errònia o dona una resposta massa tard. [↑](#footnote-ref-1)
2. En el nostre exemple l’usuari només pateix una demora (2 segons), però no ha de tornar a fer l’acció que tenia en curs quan ha succeït l’error. [↑](#endnote-ref-1)
3. L’error introduït afecta una única classe per tant desapareix en fer una simple substitució de la classe. [↑](#footnote-ref-2)