

**Escola Politècnica Superior**

**Departament d’Informàtica**

**Projecte de fi de carrera**

CREACIÓ D’APLICACIÓ ANDROID PER A LA CERCA D’EVENTS I INTERACCIÓ EN ELLS

**Autor: Guillem Arbiol López de Zamora**

**Tutora: Montserrat Sendín Veloso**



**Novembre de 2022**

**UDL**

AGRAïMENTS

M’agradaria agrair en primer lloc als meus pares, que sempre m’han motivat a estudiar i m’han donat totes les comoditats necessàries per a fer-ho.

També als meus amics amb els que sempre he pogut contar en els bons i els mals moments i, en gran part, a la UDL i a tots els seus professors per ensenyar-me a ser un bon informàtic.

Voldria agrair en especial a la Montserrat Sendín per compartir els seus coneixements sobre aplicacions mòbils durant les seves assignatures que m’han permès realitzar aquest projecte.

**ABSTRACT/RESUM**

Revisar: Abstact 1 comence amb una mica de fum però semble ser lo que es busque, abstract 2 comence amb descripció de la tecnologia, que es el que diuen que volen però em sembla que no

ABSTRACT 1:

La proliferació d’aplicacions per a dispositius mòbils és imparable, i cada any la població inverteix més temps dels seus dies en elles. El problema és que les aplicacions més utilitzades a nivell mundial estan dissenyades per a ser addictives i mantenir a l’usuari utilitzant-les el major temps possible.

L’objectiu d’aquest projecte és crear una aplicació per a dispositius mòbils que en lloc de generar addicció, fomenti altres activitats diferents, així com la interacció entre usuaris dins i fora de l’aplicació.

Seguint aquestes directrius, en el projecte s’ha desenvolupat una aplicació per cercar events de tot tipus arreu del mon i poder comunicar-se amb altres usuaris interessats en aquests.

El desenvolupament de l’aplicació s’ha realitzat utilitzant l’IDE Android Studio i programant amb els llenguatges Java i Kotlin. També s’ha fet ús dels serveis proporcionats per diferents APIs com ara la que ofereix Ticketmaster per la cerca d’events, l’API de Firebase per implementar un sistema de xat i l’API de Google Maps per mostrar un mapa.

Els events es busquen escrivint el nom i ordenant els resultats per popularitat, proximitat a l’usuari o proximitat a la data d’inici de l’event. Cada event conta amb una sala de xat aïllada per a que els interessats en ell puguin conversar abans i durant l’event.

ABSTRACT 2:

Les aplicacions mòbils són programes que s’instal·len en dispositius mòbils (ja sigui un telèfon o una tauleta) i que es poden integrar les característiques del gadget (artefacte), com ara la càmera o el sistema GPS. A més a més es poden actualitzar per afegir noves característiques amb el pas del temps.

En el projecte es mostrarà el procés que s’ha seguit per a crear una aplicació mòbil des de 0, mostrant la fase prèvia d’estudi i anàlisi i la fase d’implementació, en la que s’explica detalladament el codi utilitzat.

S’ha desenvolupat una aplicació que permet cercar events de tot tipus arreu del mon i poder comunicar-se amb altres usuaris interessats en aquests mitjançant sales de xat. També inclou altres funcionalitats com poder cercar events propers a una ubicació escollida mitjançant un mapa interactiu i veure la seva ubicació exacta en el mapa.

La implementació s'ha realitzat amb l'IDE Android Studio per les nombroses opcions avançades que es poden trobar en aquest entorn de desenvolupament, i les facilitats que ofereix. Per tant l’aplicació resultant està destinada a dispositius Android. Els llenguatges de programació escollits han set Java i Kotlin ja que són els que més s’adapten a la programació en aquest sistema operatiu. També s’ha fet ús dels serveis proporcionats per diferents APIs com ara la que ofereix Ticketmaster per la cerca d’events, l’API de Firebase per implementar un sistema de xat i l’API de Google Maps mostrar un mapa.

Índex

[1 Introducció 11](#_Toc118853268)

[1.1 Motivació 11](#_Toc118853269)

[1.2 Objectius 12](#_Toc118853270)

[1.3 Estructura del document 13](#_Toc118853271)

[2 Planificació del projecte 14](#_Toc118853272)

[2.1 Restriccions 14](#_Toc118853273)

[2.2 Planificació 15](#_Toc118853274)

[2.3 Anàlisi estratègica DAFO 16](#_Toc118853275)

[2.4 Anàlisi de la competència 17](#_Toc118853276)

[2.5 Estudi de viabilitat 18](#_Toc118853277)

[2.6 Model de negoci 18](#_Toc118853278)

[2.7 Estudis realitzats 19](#_Toc118853279)

[3 Descripció de la funcionalitat 20](#_Toc118853280)

[3.1 Requeriments 20](#_Toc118853281)

[3.1.1 Requeriments funcionals 20](#_Toc118853282)

[3.1.2 Requeriments no funcionals 21](#_Toc118853283)

[3.2 Diagrama de casos d’ús 22](#_Toc118853284)

[3.2.1 Descripció dels casos d’ús 22](#_Toc118853285)

[4 Descripció de la tecnologia utilitzada 23](#_Toc118853286)

[4.1 Plataforma seleccionada 23](#_Toc118853287)

[4.2 Llenguatge utilitzat: java i kotlin 24](#_Toc118853288)

[4.2.1 Java 24](#_Toc118853289)

[4.2.2 Kotlin 25](#_Toc118853290)

[4.3 Entorn d’execució 25](#_Toc118853291)

[4.4 SDK 26](#_Toc118853292)

[5 Disseny de l’aplicació 26](#_Toc118853293)

[5.1 Disseny de la interfície 27](#_Toc118853294)

[5.2 Mapa de navegació 29](#_Toc118853295)

[5.3 Disseny de pantalles 29](#_Toc118853296)

[5.4 Disseny de menús 43](#_Toc118853297)

[6 Implementació 44](#_Toc118853298)

[6.1 Decisions d’implementació 44](#_Toc118853299)

[6.2 CONCEPTES 45](#_Toc118853300)

[6.3 CODI 46](#_Toc118853301)

[6.3.1 SignInActivity 46](#_Toc118853302)

[6.3.2 RegisterActivity 54](#_Toc118853303)

[6.3.3 MainActivity 57](#_Toc118853304)

[6.3.4 INICIO 64](#_Toc118853305)

[6.3.5 EVENTOS 71](#_Toc118853306)

[6.3.6 EventAdapter 81](#_Toc118853307)

[6.3.7 MAPA 93](#_Toc118853308)

[6.3.8 LocationActivity 100](#_Toc118853309)

[6.3.9 CHAT 102](#_Toc118853310)

[7 Conclusions i treball futur 114](#_Toc118853311)

[7.1 Conclusions 114](#_Toc118853312)

[7.2 Treball futur 114](#_Toc118853313)

[8 [ANNEXOS] 117](#_Toc118853314)

Taula de figures

[Figura: 1. Funcions que aporta Discovery Api de Ticketmaster. 18](#_Toc117650620)

[Figura: 2. Diagrama de casos d'ús, realitzat amb l’eina https://online.visual-paradigm.com/ 20](#_Toc117650621)

[Figura: 3. Taula dels llenguatges de programació més utilitzats, de la pàgina https://www.tiobe.com/tiobe-index/ 22](#_Toc117650622)

[Figura: 4. Disseny de distribució del SDK, de la pàgina https://developer.android.com/design-for-safety/privacy-sandbox/sdk-runtime 24](#_Toc117650623)

[Figura: 5. Mapa de navegació, creat amb l’eina draw.io 27](#_Toc117650624)

[Figura: 6. Pantalla d’inici de sessió 28](#_Toc117650625)

[Figura: 7. Pantalla de registre 29](#_Toc117650626)

[Figura: 8. Pantalla Inicio 30](#_Toc117650627)

[Figura: 9. Pantalla Eventos 32](#_Toc117650628)

[Figura: 10. Pantalla Mapa 34](#_Toc117650629)

[Figura: 11. Pantalla Chat 35](#_Toc117650630)

[Figura: 12. Pantalla Eventos Guardados 37](#_Toc117650631)

[Figura: 13. Pantalla Detalles de evento 39](#_Toc117650632)

[Figura: 14. Menú inferior, on està seleccionada la primera opció 40](#_Toc117650633)

[Figura: 15. Menú superior en la pantalla Detalles de Evento 40](#_Toc117650634)

[Figura: 16. Captura que mostra els fitxers bottom\_nav\_menu i mobile\_navigation 41](#_Toc117650635)

[Figura: 17. Part de l'arxiu activity\_sign\_in.xml on es mostra el codi del botó "SignInButton" de Google 43](#_Toc117650636)

[Figura: 18. Part de la funció onCreate(), de la classe SignInActivity 43](#_Toc117650637)

[Figura: 19. Part de la funció onCreate(), de la classe SignInActivity 44](#_Toc117650638)

[Figura: 20. Part de la funció onCreate(), de la classe SignInActivity 45](#_Toc117650639)

[Figura: 21. Funció authenticateUser(), de la classe SignInActivity 45](#_Toc117650640)

[Figura: 22. Funció signIn(), de la classe SignInActivity 46](#_Toc117650641)

[Figura: 23. ActivityResultLauncher anomenat someActivityResultLauncher, de la classe SignInActivity 46](#_Toc117650642)

[Figura: 24. Funció firebaseAuthWithGoogle (), de la classe SignInActivity 47](#_Toc117650643)

[Figura: 25. Captura del quadre de diàleg enviat per recuperar la contrasenya 48](#_Toc117650644)

[Figura: 26. Funció showRecoverPasswordDialog (), de la classe SignInActivity 48](#_Toc117650645)

[Figura: 27. Part de la funció showRecoverPasswordDialog (), de la classe SignInActivity 49](#_Toc117650646)

[Figura: 28. Funció beginRecovery (), de la classe SignInActivity 50](#_Toc117650647)

[Figura: 29. Part de la funció registerUser (), de la classe RegisterActivity 51](#_Toc117650648)

[Figura: 30. Funció verifyEmail (), de la classe RegisterActivity 52](#_Toc117650649)

[Figura: 31. Funció switchToLogin (), de la classe RegisterActivity 53](#_Toc117650650)

[Figura: 32. Part de l’arxiu AndroidManifest.xml on s’especifica l’activitat principal 53](#_Toc117650651)

[Figura: 33. Part de l’arxiu MainActivity.xml on es crea el menú inferior tipus “BottomNavigationView”. 53](#_Toc117650652)

[Figura: 34. Part de l’arxiu MainActivity.xml on es crea el “Fragment”. 54](#_Toc117650653)

[Figura: 35. Part de la funció onCreate() de l'activitat MainActivity 54](#_Toc117650654)

[Figura: 36. Part de la funció onCreate() de l'activitat MainActivity 54](#_Toc117650655)

[Figura: 37. Gràfic de navegació entre fragments 55](#_Toc117650656)

[Figura: 38. Part de l'arxiu mobile\_navigation.xml on es mostra el codi del fragment "Inicio" 56](#_Toc117650657)

[Figura: 39. Part de la funció onCreate() de l'activitat MainActivity 57](#_Toc117650658)

[Figura: 40. Funció onStart() de l'activitat MainActivity 58](#_Toc117650659)

[Figura: 41. Funció checkPermissions() de l'activitat MainActivity 58](#_Toc117650660)

[Figura: 42. Funció requestPermissions() de l'activitat MainActivity 58](#_Toc117650661)

[Figura: 43. Quadre que mostra les opcions que té l'usuari per atorgar o no el permís d'ubicació 59](#_Toc117650662)

[Figura: 44. Funció startLocationPermissionRequest() de l'activitat MainActivity 59](#_Toc117650663)

[Figura: 45. Funció onRequestPermissionsResult() de l'activitat MainActivity 60](#_Toc117650664)

[Figura: 46. Exemple d'una de les LinearLayouts mencionades 60](#_Toc117650665)

[Figura: 47. Part de l'arxiu fragment\_home.xml que mostra el codi d'un dels components LinearLayout 61](#_Toc117650666)

[Figura: 48. Part de la funció onCreateView() del Fragment HomeFragment 62](#_Toc117650667)

[Figura: 49. Part de la funció onCreateView() del Fragment HomeFragment 62](#_Toc117650668)

[Figura: 50. Part de la funció onCreateView() del Fragment HomeFragment 62](#_Toc117650669)

[Figura: 51. Part de la funció onCreateView() del Fragment HomeFragment 63](#_Toc117650670)

[Figura: 52. Funció openLayout() del Fragment HomeFragment 63](#_Toc117650671)

[Figura: 53. Part de la funció onCreateView() del Fragment HomeFragment 63](#_Toc117650672)

[Figura: 54. Funció logoutUser() del Fragment HomeFragment 64](#_Toc117650673)

[Figura: 55. Funció onResume() de l'activitat HomeFragment 64](file:///C:\Users\yonki\OneDrive\Documents\Deures\4rt%20gei\TFG\MEMÒRIA_TFG%20(Autoguardado).docx#_Toc117650674)

[Figura: 56. Part de la funció onCreate() de l'activitat Configuration 64](#_Toc117650675)

[Figura: 57. Funció revokeAcces() de l'activitat Configuration 65](#_Toc117650676)

[Figura: 58. Funció signOut() de l'activitat Configuration 65](#_Toc117650677)

[Figura: 59. Part de la funció onCreate() de l'activitat SavedEvents 66](#_Toc117650678)

[Figura: 60. Part de la funció onCreate() de l'activitat SavedEvents 67](#_Toc117650679)

[Figura: 61. Dades que es poden aportar per la cerca d'events 68](#_Toc117650680)

[Figura: 62. Part de l’arxiu build.gradle referent a l’emmagatzematge de claus d’API 69](#_Toc117650681)

[Figura: 63. Estructura inicial de la URL utilitzada per obtenir events 69](#_Toc117650682)

[Figura: 64. Codi per transformar una Location a un GeoHash, en la classe EventsRequester 70](#_Toc117650683)

[Figura: 65. Opcions d’ordenació dels resultats obtinguts en la consulta https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/ 70](#_Toc117650684)

[Figura: 66. Mètodes de la llibreria Volley 71](#_Toc117650685)

[Figura: 67. Estructura de la resposta rebuda per part de la API de Ticketmaster 72](#_Toc117650686)

[Figura: 68. Variables que guarden la Informació de la classe Event 73](#_Toc117650687)

[Figura: 69. Variables que guarden la Informació de la classe EventBuilder 74](#_Toc117650688)

[Figura: 70. Mètodes de la classe EventBuilder 74](#_Toc117650689)

[Figura: 71. Mètode constructor per crear l’objecte Event, en la classe Event 75](#_Toc117650690)

[Figura: 72. Mètodes utilitzats per transformar una String a una JSONArray, en la classe EventsRequester 75](#_Toc117650691)

[Figura: 73. Exemples de mètodes utilitzats de la llibreria JSON 75](#_Toc117650692)

[Figura: 74. Part de la funció processEvent(), de la classe EventsRequester 76](#_Toc117650693)

[Figura: 75. Codi que defineix l'element RecyclerView, en l'arxiu fragment\_events.xml 76](#_Toc117650694)

[Figura: 76. Codi en que s’inicialitzen els components de la RecyclerView, en la classe EventsFragment 77](#_Toc117650695)

[Figura: 77. Funció onCreateViewHolder() de la classe EventAdapter 77](#_Toc117650696)

[Figura: 78. Funció bind(), en la classe ViewHolder que es troba niuada en la classe EventAdapter 78](#_Toc117650697)

[Figura: 79. Representació d'exemple d'un event, amb les icones encerclades 79](#_Toc117650698)

[Figura: 80. Funció goToChat(), de la classe ViewHolder que es troba niuada en la classe EventAdapter 79](#_Toc117650699)

[Figura: 81. Funció goToMap(), de la classe ViewHolder que es troba niuada en la classe EventAdapter 80](#_Toc117650700)

[Figura: 82. Funció showBigEvent(), de la classe ViewHolder que es troba niuada en la classe EventAdapter 80](#_Toc117650701)

[Figura: 83. Part de la funció onCreate(), de la classe BigEvent 82](#_Toc117650702)

[Figura: 84. Part de la funció onCreate(), de la classe BigEvent 82](#_Toc117650703)

[Figura: 85. Part de la funció loadImages(), de la classe BigEvent 83](#_Toc117650704)

[Figura: 86. Part de la funció onCreate(), de la classe BigEvent 83](#_Toc117650705)

[Figura: 87. Funció openCalendar(), de la classe BigEvent 84](#_Toc117650706)

[Figura: 88. Part de la funció onCreate(), de la classe BigEvent 85](#_Toc117650707)

[Figura: 89. Menú superior present en l’activitat BigEvent 85](#_Toc117650708)

[Figura: 90. Funció onCreateOptionsMenu(), de la classe BigEvent 86](#_Toc117650709)

[Figura: 91. Funció setFavouriteIcon(), de la classe BigEvent 86](#_Toc117650710)

[Figura: 92. Part de la funció onOptionsItemSelected(), de la classe BigEvent 87](#_Toc117650711)

[Figura: 93. Part de la funció onOptionsItemSelected(), de la classe BigEvent 87](#_Toc117650712)

[Figura: 94. Codi de l’element FragmentContainerView, en l’arxiu fragment\_map.xml 88](#_Toc117650713)

[Figura: 95. Pantalla del mapa marcant la ubicació d’un event 89](#_Toc117650714)

[Figura: 96. Pantalla del mapa mostrant els events trobats en un radi de 500km 90](#_Toc117650715)

[Figura: 97. Interfície OnMapReadyCallback 90](file:///C:\Users\yonki\OneDrive\Documents\Deures\4rt%20gei\TFG\MEMÒRIA_TFG%20(Autoguardado).docx#_Toc117650716)

[Figura: 98. Part de la funció onCreateView(), de la classe MapFragment 91](#_Toc117650717)

[Figura: 99. Part de la funció onCreateView(), de la classe MapFragment 91](#_Toc117650718)

[Figura: 100. Funció findNearEvents(), de la classe MapFragment 92](#_Toc117650719)

[Figura: 101. Funció getNearEvents(), de la classe MapFragment 92](#_Toc117650720)

[Figura: 102. Funció onCreate(), de la classe MapFragment 93](#_Toc117650721)

[Figura: 103. Funció onMapReady(), de la classe MapFragment 94](#_Toc117650722)

[Figura: 104. Constructor de la classe LocationActivity 95](#_Toc117650723)

[Figura: 105. Part de la funció lastLocation(), de la classe LocationActivity 95](#_Toc117650724)

[Figura: 106. Part de la funció lastLocation(), de la classe LocationActivity 96](#_Toc117650725)

[Figura: 107. Funció focusCamera(), de la classe LocationActivity 97](#_Toc117650726)

[Figura: 108. Barra inferior de la pantalla Chat 98](#_Toc117650727)

[Figura: 109. Llibreries utilitzades en el sistema de xat, implementades des de l’arxiu build.gradle 99](#_Toc117650728)

[Figura: 110. Classe ChatMessage 99](#_Toc117650729)

[Figura: 111. Variables de la classe ChatFragment 100](#_Toc117650730)

[Figura: 112. Part de la funció onCreateView(), de la classe ChatFragment 100](#_Toc117650731)

[Figura: 113. Part de la funció onCreateView(), de la classe ChatFragment 101](#_Toc117650732)

[Figura: 114. Classe MyScrollToBottomObserver 102](#_Toc117650733)

[Figura: 115. Part de la funció onCreateView(), de la classe ChatFragment 102](#_Toc117650734)

[Figura: 116. Classe MyButtonObserver 102](#_Toc117650735)

[Figura: 117. Botó d'enviar quan hi ha text Figura: 118. Botó d'enviar quan no hi ha text 103](#_Toc117650736)

[Figura: 119. Funció getUserName(), de la classe ChatFragment 103](#_Toc117650737)

[Figura: 120. Funció getPhotoUrl(), de la classe ChatFragment 103](#_Toc117650738)

[Figura: 121. Part de la funció onCreateView(), de la classe ChatFragment 103](#_Toc117650739)

[Figura: 122. Part de la funció onCreateView(), de la classe ChatFragment 104](#_Toc117650740)

[Figura: 123. Variable de la classe ChatFragment 104](#_Toc117650741)

[Figura: 124. Classe MyOpenDocument 104](#_Toc117650742)

[Figura: 125. Part de la funció onCreateView(), de la classe ChatFragment 104](#_Toc117650743)

[Figura: 126. Funció onImageSelected(), de la classe ChatFragment 105](#_Toc117650744)

[Figura: 127. Funció putImageInStorage(), de la classe ChatFragment 106](#_Toc117650745)

[Figura: 128. Funció onCreateViewHolder(), de la classe MessageAdapter 107](#_Toc117650746)

[Figura: 129. Funció onBindViewHolder(), de la classe MessageAdapter 107](#_Toc117650747)

[Figura: 130. Funció bind(), de la classe interna MessageViewHolder, que pertany a la classe MessageAdapter 107](#_Toc117650748)

[Figura: 131. Funció setTextColor(), de la classe interna MessageViewHolder, que pertany a la classe MessageAdapter 108](#_Toc117650749)

[Figura: 132. Funció loadImageIntoView (), de la classe MessageAdapter 108](#_Toc117650750)

# Introducció

## Motivació

A dia d’avui, en l’any 2022, la població mundial és de 7.400 milions de persones, de les quals es calcula que més de 5.300 milions tenen una subscripció a un telèfon mòbil.

D’entre tota aquesta gent, el 84% d’ells tenen un dispositiu Android, fent un total de 4.536 milions d’usuaris.

A més a més, es calcula que cada any la població dedica més temps a navegar per aplicacions mòbils, fent que al 2018 un terç dels espanyols dediqués 20 hores setmanals a aquestes aplicacions.

Actualment, la tendència va en augment i tot i que les dades varien i no són exactes, es calcula que més o menys el 38% dels joves entre 18 i 24 anys dediquen més de 6 hores diàries al mòbil, el que genera addicció i problemes de salut.

Dit això, s’ha decidit fer el projecte sobre el desenvolupament d’una aplicació per a dispositius Android degut a la importància que tenen actualment, però en lloc d’una aplicació que generi addicció i faci perdre el temps a l’usuari es vol fer una aplicació que pugui resultar realment útil i, en part, combatre aquests hàbits.

Així s’ha arribat a la idea d’una aplicació que ajunti la cerca d’events online amb la possibilitat de comunicar-se amb la gent interessada en assistir, de forma que ajuda a trobar els events desitjats o events propers així com trobar a gent que també està interessada en ells i compartir informació o trobar noves amistats.

A més a més, Espanya es un país pioner en la celebració d’events, sent al 2019 el 3r país del món que més augmentava el numero d’events celebrats i el líder a Europa.

Per una altra banda, el que ha motivat la idea final de l’aplicació han set també altres factors:

La idea de fer una aplicació dirigida a dispositius Android es deguda en part als coneixements obtinguts en les assignatures “aplicacions per a dispositius mòbils” i “plataformes en xarxa”, les dos impartides per la professora Montserrat Sendín, qui és també la tutora d’aquest projecte.

La idea d’implantar un sistema d’events, mapa i xat combinats ve per la motivació de crear una aplicació innovadora així com per les ganes d’utilitzar diverses APIs i aprendre com funcionen.

Finalment en el resultat final també s’ha buscat una aplicació amb una funcionalitat que tingués en compte les tendències actuals i que jo mateix pogués arribar a utilitzar, ja que és la manera més simple de considerar-la realment útil.

## Objectius

Els objectius que es busquen al realitzar aquest projecte són els següents:

* **Familiaritzar-me amb l’ús d’APIs:**

Aprendre els conceptes necessaris sobre la utilització d’APIs com la seva implementació i comunicació.

Revisar:

* **Familiaritzar-me amb l’entorn Android:**

Aprendre els conceptes necessaris sobre la programació per a dispositius Android, sobretot les diferent eines i serveis que ofereix l’IDE “Android Studio”.

* **Realitzar un estudi complert sobre els factors a tenir en compte en el desplegament d’una aplicació funcional en el mercat:**

Estudiar tots els factors implicats, com ara costos de desplegament, aspectes legals com drets d’imatge o copyright o l’existència d’altres aplicacions similars entre altres.

* **Integrar peticions dels usuaris als servidors de les APIs:**

Fer consultes personalitzades segons les peticions dels usuaris a una API per obtenir resultats concrets.

* **Mostrar diversos events socials que es realitzin a prop de l’usuari**

Integrar una API que permet rebre informació sobre events propers a una ubicació determinada així com aplicar filtres a la cerca.

* **Mostrar un mapa amb la ubicació de l’usuari**

Integrar un mapa funcional amb el que l’usuari pugui interactuar per buscar punts d’interès o events propers a la seva ubicació.

* **Aplicar un sistema de xat per sales i vincular-les als events**

Integrar un sistema de comunicació per missatges de text a temps real entre els diferents usuaris de l’aplicació, i que cada event en pugui tenir un separat.

* **Planificar el sistema de monetització de l’aplicació**

Dissenyar un sistema viable per a treure un profit econòmic a l’hora de publicar l’aplicació, de forma que sigui realista i que els guanys superin les despeses que la publicació de l’aplicació suposa.

* **Integrar totes les funcionalitats en una aplicació Android**

Unificar tots els aspectes mencionats en una única aplicació Android totalment funcional.

## Estructura del document

Aquest document es divideix en un total de 8 capítols. Els 5 primers pertanyen a la fase d’anàlisi, el sisè és el més important i extens, pertanyent a la fase d’implementació, i els dos últims pertanyen a la fase final que inclou les conclusions i extres. Revisar: annexos? Extres?

Es comença amb el capítol d’introducció que serveix per presentar el projecte. Inclou la motivació d’aquest, els objectius principals que es busquen, l’índex i aquest mateix apartat que descriu l’estructura del document.

El segon capítol explica el procés de planificació del projecte. En ell s’expliquen tots els processos previs a la realització del projecte que inclouen valorar les restriccions existents, un anàlisi DAFO, estudis de la competència i de viabilitat, crear un model de negoci i altres estudis previs que s’han realitzat.

El tercer capítol descriu la funcionalitat del projecte. Primerament s’exposen els requeriments funcionals i no funcionals i després es mostra un diagrama de casos d’ús amb l’explicació pertinent de cada cas d’ús.

El quart capítol es centra en les diferents tecnologies utilitzades durant el desenvolupament del projecte com ara l’IDE, els llenguatges de programació i l’entorn d’execució.

El cinquè capítol tracta el disseny de l’aplicació. Aquest inclou el disseny de la interfície, de les pantalles i del menú així com el mapa de navegació.

El sisè capítol és el més extens de tots, ocupant un total de Revisar: 67 pàgines. Aquest capítol s’encarrega d’explicar tot el codi utilitzat en l’aplicació pas per pas ja que es considera que és la part a la que s’ha dedicat més esforç i treball i és, per tant, la part més important del projecte.

El setè capítol explica les conclusions professionals i personals que s’han tret del projecte, així com les futures millores que es poden realitzar.

El vuitè capítol recull tots els annexos necessaris així com la bibliografia.

# Planificació del projecte

## Restriccions

Degut a la naturalesa de l’aplicació, aquesta requereix del servei de tercers per a realitzar moltes de les severs tasques, fet que ha dificultat la implementació de funcions que depenen estrictament dels serveis o dades aportats per tercers.

La primera restricció trobada sorgeix en la API per buscar events. N’existeixen unes quantes però no gaires ofereixen els seus serveis gratuïtament, i les que ho fan limiten les funcionalitats i informació que ofereixen, així com el límit de peticions diàries i peticions per segon al servidor.

Després d’una cerca en profunditat, s’ha escollit l’API de Ticketmaster, tot i que cal tenir en compte que no ofereix un gran nombre d’events i menys a Espanya, ja que es centra en Estats Units i Canadà. Actualment es disposa de la versió gratuïta anomenada “Discovery API”, tenint després les versions de pagament “Partner API”, “Publish API” i “International Discovery API” que ofereixen més serveis.

La següent restricció ve donada per les APIs de Google. Inicialment es volia fer ús de la anomenada Places API per obtenir dades sobre la importància o afluència dels establiments que gestiona, però aquestes són dades molt valuoses i no es comparteixen de cap manera, fet que passa també en altres APIs que ofereixen serveis similars. Una altra restricció dels serveis de Google és l’ús limitat de la API de Google Maps, ja que aquesta comença a cobrar a partir de les 28500 càrregues de mapa, el que equival exactament a 200$.

L’última restricció important ve donada per els serveis que requereixen un servidor propi actiu 24h, ja que no es disposa d’un, per tant no s’han pogut implementar.

## Planificació

El projecte té la finalitat de crear una aplicació mòbil per promoure i facilitar la interacció social en grans events.

Per a realitzar-ho, primerament es trobaran els events amb l’ajuda de l’API “Discovery Api” de Ticketmaster. Aquesta aportarà dades sobre els events com ara el nom, data, ubicació, fotografies... I fent ús d’aquestes dades es mostrarà una vista prèvia de l’event a l’usuari.

Un cop es tingui, caldrà integrar el sistema de xat utilitzant els serveis que Firebase ofereix. Per distingir usuaris s’utilitzarà l’API d’autenticació de Firebase amb un sistema de registre que asseguri un compte únic per cada usuari i per guardar missatges s’utilitzarà la base de dades a temps real de Firebase, la “Firebase Realtime Database”.

Després caldrà crear un sistema que permeti disposar d’una sala de xat independent per cada event disponible a les que qualsevol usuari pugui entrar i que permetin una comunicació instantània entre múltiples usuaris a la vegada.

Finalment s’afegirà una pantalla destinada a allotjar un mapa global, obtingut mitjançant l’API de Google Maps. En aquest mapa s’hauria de poder veure la ubicació actual de l’usuari així com els events propers a ell.

Cal remarcar que inicialment es planejava una altra funció que no s’han pogut dur a terme per diferents raons que s’exposaran en futurs apartats (Apartat 2.7). Aquesta funció és la següent:

Buscar ubicacions amb grans multituds mitjançant dades extretes de la API “places API”. Aquesta API obtindria informació sobre diferents llocs entre la que s’inclou l’afluència de gent, permetent classificar diferents punts geogràfics segons la gent present en ells i per tant localitzar grans punts d’afluència de gent on l’aplicació serà útil.

Després, s’integraria un sistema de xat per ubicació. Aprofitant la informació obtinguda anteriorment, es crearien diferents sales de xat a les que els usuaris només podrien accedir si es troben físicament en l’event o localització de la sala. En llocs com centres comercials, universitats, espais lúdics o events socials regulars hi podria haver sales fixes creades a mà o automàticament a partir de les dades extretes de la “Places API”, de manera que sempre que l’usuari estigués en un lloc amb una mínima afluència de gent tindria una sala on comunicar-se amb la gent de l’àrea.

Per acabar, s’ha estudiat també quin seria el perfil de l’usuari en potència de l’aplicació. Seria una persona més aviat jove, d’entre 15 i 35 anys. El fet que sigui una aplicació amb un format social implica que sempre és més fàcil obtenir usuaris joves ja que tendeixen a socialitzar més, així com a assistir a més events o a llocs d’oci i a utilitzar més les noves tecnologies. Dins del perfil dels joves la podria utilitzar qualsevol persona ja que no hi ha un perfil més específic d’usuari. L’única condició és que l’usuari ha de buscar i anar a events amb una mínima regularitat per treure-li profit.

Tot i ser aquest el perfil de l’usuari més comú, també es té en compte que hi poden haver molts altres tipus d’usuaris, sobretot gent una mica més gran, d’uns 35-60 anys, que busquen nous hobbies o events com anar a exposicions d’art, cotxes, literatura, esports... En els que conèixer nova gent i gaudir.

Un cas d’ús típic seria un jove estudiant que ha anat a una nova ciutat a cursar la carrera o realitzar un Erasmus, i aprofita l’aplicació tant per buscar events en la zona com per socialitzar un cop en ells.

## Anàlisi estratègica DAFO

Debilitats

* L’aplicació necessita un nombre inicial d’usuaris per a que el xat funcioni fluidament, ja que si no hi ha suficient usuaris els xats privats dels events no seran útils.
* Es conta amb un pressupost bastant limitat i desplegar a gran escala moltes API i parts del projecte requereixen d’una inversió inicial considerable.
* No es conta amb una cartera de clients inicial.
* L’API utilitzada per obtenir els events no té massa suport a Espanya.

Amenaces

* Les demés aplicacions amb funcionalitats similars són la principal amenaça, ja que tenen anys de desenvolupament darrera i equips professionals, així com una base d’usuaris establerta. També conten amb molt pressupost (milions d’euros) i per tant, amb funcionalitats que aquest projecte no es pot permetre.
* També és una amenaça el COVID, ja que tot i que actualment ja no hi ha restriccions, molta gent segueix tenint por o respecte als events multitudinaris.
* Finalment, també cal considerar l’encariment de la vida que s’està donant, que impedeix que molta gent pugui sortir i gastar diners en events que majoritàriament no son gratuïts.

Fortaleses

* Les principals fortaleses de l’aplicació són les novetats incorporades respecte a altres aplicacions: El sistema de xat per events pot donar lloc a una aplicació molt útil, ja que dona una nova oportunitat a socialitzar, fer preguntes sobre l’event, trobar gent un cop allí...
* També cal mencionar el sistema de buscar events fent ús del mapa, que resulta molt útil per trobar directament events en un lloc determinat sense haver d’estar consultant la ubicació d’aquests.
* Ja es sap que les aplicacions de cerca d’events funcionen bastant bé en el mercat, per tant unir els aspectes útils d’aquestes i oferir noves funcionalitats és una gran fortalesa.

Oportunitats

* La principal oportunitat ens la dona el propi COVID 19: com es porten uns anys realitzant molt pocs events presencials, actualment s’estan reprenent i la gent en té moltes ganes de participar. A això es suma que els joves no han pogut socialitzar massa durant aquest temps per les restriccions imposades i es té una bona oportunitat: joves que sortiran a gaudir d’events socials i multitudinaris amb energies renovades i moltes ganes, i que per tant inspeccionaran de nou la “Play Store” en busca d’aplicacions que els semblin interessants o útils. Aquest fet permet obtenir clients nous que mai havien utilitzat aplicacions similars però també clients que antigament utilitzaven altres aplicacions similars i que se les van desinstal·lar per la falta d’events socials.

## Anàlisi de la competència

És molt fàcil trobar aplicacions que es dediquen a recopilar els diferents events que es realitzen en tot el món, sent “Eventbrite” i “Meetup” les més populars amb més de 10 milions de descàrregues. Després d’aquestes n’hi ha unes quantes desenes més amb objectius molt similars, moltes de les quals tenen molts milers de descàrregues també.

L’aplicació Eventbrite ofereix un repertori d’events pròxims segons la ubicació indicada. Té altres funcionalitats com seguir a usuaris, compartir events, guardar entrades... però no conta amb interacció entre usuaris ni cerca per mapa. El seu punt fort es la gran oferta d’events amb la que conta i totes les funcionalitats que ofereix.

L’aplicació Meetup és l’altra aplicació d’events amb +10 milions d’usuaris, però aquesta sí conta amb interacció entre usuaris. Aquesta aplicació aprofita els events pròxims per organitzar-los segons el tipus d’activitat que es realitza i recomanar-los als usuaris, però a la vegada també organitza als usuaris segons els seus gustos i ajuda a crear grups de persones amb gustos en comú, que poden anar junts a events que els agradin. L’aplicació conta també amb un sistema de missatgeria entre usuaris que es segueixin, en format de xat privat individual.

És el principal competidor ja que conta amb +10 milions d’usuaris, moltes funcionalitats i un objectiu similar, per tant molts usuaris hauran d’escollir entre una i l’altra.

A poc temps d’acabar aquest projecte, Meetup ha tret una actualització en que es poden buscar events pròxims mitjançant el mapa, exactament com a l’aplicació creada. Almenys això demostra que és una bona idea.

Les demés aplicacions que s’han observat segueixen el patró d’una d’aquestes dos, mostrant petits canvis en la funcionalitat com “Fever”, que inclou un sistema de compra més desenvolupat.

Per una altra banda, no existeix cap aplicació gaire popular que apliqui el sistema de xat per events ni n’hi havia cap que apliques el sistema de cerca per mapa fins fa poc, tot i que n’hi ha algunes amb conceptes similars per sota de les 5 mil descàrregues.

## Estudi de viabilitat

L’aplicació pot ser viable, ja que fa ús un mercat existent i amb eficàcia provada però afegint noves funcionalitats bastant innovadores que poden resultar interessants. El moment de sortida de l’aplicació també serà ideal per a que la provin els joves que portaven temps sense poder usar aplicacions similars pel COVID 19, fet que pot ser decisiu a l’hora de aconseguir els usuaris inicials requerits per fer funcionar correctament el sistema de xat. Una altra opció és també realitzar una petita campanya publicitària en el moment del llançament per assegurar més descàrregues inicials.

En definitiva, el llançament no serà fàcil, però si s’aconsegueix una base de usuaris estable la aplicació té moltes opcions de prosperar satisfactòriament.

## Model de negoci

A l’hora d’obtenir guanys amb l’aplicació es tenen diferents opcions, tot i que unes no exclouen a les altres. Inicialment, la manera més senzilla és amb anuncis dins de l’aplicació. Aquests poden venir de Google AdMob o altres plataformes, tot i què actualment generen pocs beneficis i cal mostrar vídeos de pantalla complerta als usuaris per tal de generar una quantitat d’ingressos acceptable. Es podrien utilitzar pancartes conegudes com “banners” en barres superiors o inferiors de la pantalla per mostrar anuncis menys intrusius, que ajudarien a complementar els ingressos generats per altres fonts.

Per una altra banda, una font d’ingressos que pot resultar més productiva seria un sistema de promoció d’events, en que els organitzadors d’un event poguessin pagar una quantitat per ressaltar el seu event al buscar-n’hi en les seves proximitats (com fan moltes xarxes socials o inclús el buscador de Google).

Un altre model de negoci seria afegir una versió de pagament o Premium que permetés als usuaris tenir funcionalitats exclusives. Aquestes inclourien treure els anuncis, poder crear sales de xat públiques i privades (inclús es pot posar un sistema de preus per nº de sales creades), poder penjar events propis, obtenir emoticones exclusives per al xat, així com un distintiu al nom etc. Aquesta funció estaria més destinada a “influencers”, organitzadors d’events o gent coneguda i no tant a l’usuari mitjà, per tant no és la millor idea per començar a guanyar diners, però sí que pot ser útil en una etapa més avançada de l’aplicació ja que si aquesta gent més coneguda crea una relació entre l’aplicació i els seguidors, l’aplicació obté nous usuaris de manera senzilla.

Finalment, quan la aplicació ja estigués assentada, es podria afegir un sistema de compra d’entrades com altres aplicacions similars tenen, permetent obtenir una comissió com a “despeses de gestió” per la venda i emmagatzematge de l’entrada.

## Estudis realitzats

Per realitzar aquest projecte s’han realitzat molts estudis en diferents àrees, així que mencionaré els més rellevants.

Per torbar una API de la que extreure els events se’n van consultar bastantes, inclús hi ha llistes a internet sobre APIs que realitzen aquesta funció. Tot i això, la gran majoria no tenen cap versió o proba gratuïta o només donen cobertura als Estats Units.

Entre les més rellevants es troben les API de Eventbrite, SeatGeek, Eventful i Ticketmaster.

Després d’estudiar-les en detall s’ha escollit la de Ticketmaster degut a la seva extensa versió gratuïta, cobertura internacional, varietat d’informació, una extensa i detallada documentació i facilitat d’implementació.

La versió gratuïta de la API, anomenada Discovery API permet realitzar 5000 consultes a la API per dia i un límit de 5 consultes per segon, tot i que si l’aplicació que l’utilitza conta amb tots els permisos i documentació en regla, aquests límits es poden expandir gratuïtament.

També es valora la possibilitat d’obtenir més serveis posteriorment amb versions de pagament més complexes.

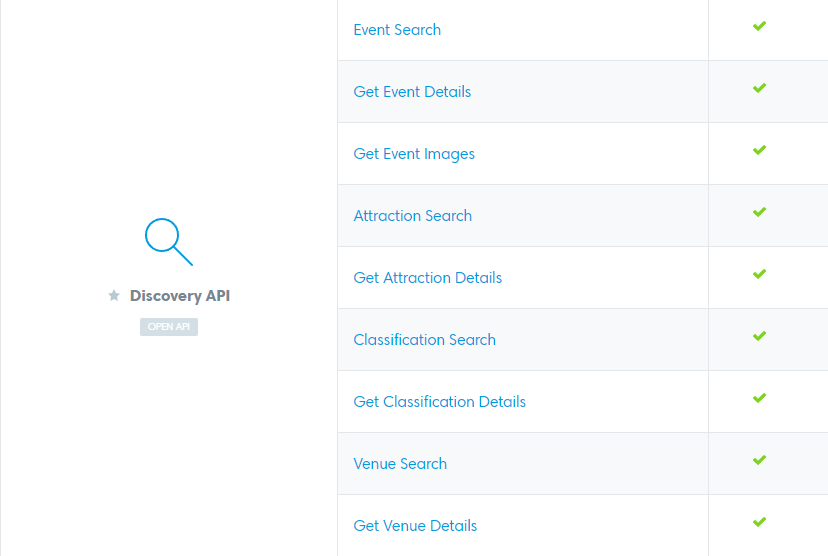


Figura: 1. Funcions que aporta Discovery Api de Ticketmaster.

<https://developer.ticketmaster.com/products-and-docs/apis/getting-started/>

A l’hora d’implementar el mapa també s’ha realitzat un estudi, valorant sobretot les API de Google Maps i d’OpenStreetMap.

L’avantatja d’OpenSteretMap és que es gratuït i de codi obert, mentre que Google Maps va canviar la seva política de preus i ha passat a cobrar als seus usuaris un cop excedeixin els 200$ al mes (unes 28.500 cargues de mapa).

Tot i així, per la comoditat de l’usuari (que només estar més acostumat a utilitzar el mapa de Google) i per la llarga llista de serveis que ofereix la llibreria de Google Maps, de moment s’ha seleccionat aquesta API.

També s’ha realitzat un estudi sobre com obtenir punts d’interès geogràfics en moltes APIs, llistes de coordenades i pàgines web, però cap s’adapta a les necessitats del projecte. Google conta amb la llibreria “Places API”, però és de pagament i només dona informació sobre locals i negocis, sense indicar els més rellevants. OpenStreetMap també té una funció similar però aquesta no està dissenyada per ser utilitzada des d’aplicacions Android i tampoc aporta el que es busca.

Per als sistemes d’autenticació i base de dades, incloent el xat, no hi ha hagut tants problemes: Firebase aporta totes les eines i funcionalitats necessàries, de manera gratuïta i amb una gran documentació de qualitat.

Durant el procés de creació de la aplicació s’han realitzat molts més estudis, des d’estudis de llibreries o arxius fins estudis per decidir els colors de l’aplicació, però els mencionats són els que han tingut més impacte en el resultat final de l’aplicació.

# Descripció de la funcionalitat

## Requeriments

Un requeriment és el conjunt de necessitats que ha de tenir en forma de funcionalitat una aplicació, és a dir, els requeriments ens indiquen el que ha de fer l’aplicació, però no la manera de com fer-ho.

Els requeriments funcionals són els que indiquen què ha de fer una aplicació, o sigui, les necessitats reals dels usuaris. En canvi, els no funcionals, són el que especifiquen aspectes que poden afectar a l’aplicació com ara la facilitat d’ús d’aquesta, el temps de resposta, manteniment etc.

### Requeriments funcionals

Al moment d’iniciar la creació de l’aplicació, els principals requeriments del projecte eren la cerca d’events online, la visualització i cerca d’events en un mapa interactiu i la disponibilitat de múltiples sales de xat, almenys una per event.

Es va pensar també en un sistema de sales de xat dinàmiques accessibles només des d’ubicacions determinades, que estarien situades en punts d’interès o zones amb gran afluència de gent, però finalment, degut a les escasses dades que es poden obtenir en aquesta àrea i al seu elevat preu, a més a més que la funcionalitat no acabava de seguir el perfil de l’aplicació, es va considerar una funcionalitat innecessària i no rentable.

Finalment es va substituir per els requisits de poder registrar usuaris i guardar events en el dispositiu, així com algunes petites funcionalitats més.

### Requeriments no funcionals

Usabilitat: Les diferents pantalles que composen la interfície segueixen un mateix estil, que facilita a l’usuari la navegabilitat i funcionament del programa.

Temps de resposta: Després de proves en diferents dispositius s’ha determinat que el temps de resposta no supera els 3 segons establerts com a màxim en cap situació.

Interfície del sistema:El sistema presenta una interfície d’usuari senzilla perquè sigui de fàcil maneig per als usuaris del sistema. També té una estructura clara, ordenant el contingut i les funcions de l’aplicació en pestanyes o apartats que abasten totes les funcionalitats disponibles.

Escalabilitat: L’arquitectura del sistema està dissenyada per a fer la aplicació fàcilment escalable, de forma que tots els components independents estan aïllats per evitar problemes en cascada i facilitar la seva edició en un futur, però gran part del codi es reutilitza en diverses parts per fer-ho més clar i permetre afegir noves funcionalitats fàcilment.

Eficiència: L’aplicació maximitza l’eficiència fent ús de bones pràctiques, llibreries especialitzades i components que permeten estalviar recursos com ara les RecyclerView.

Solidesa:Quan es produeix un error, l’aplicació es capaç de detectar-lo en temps d’execució i tractar-lo, seguint amb el seu funcionament sempre que sigui possible.

## Diagrama de casos d’ús

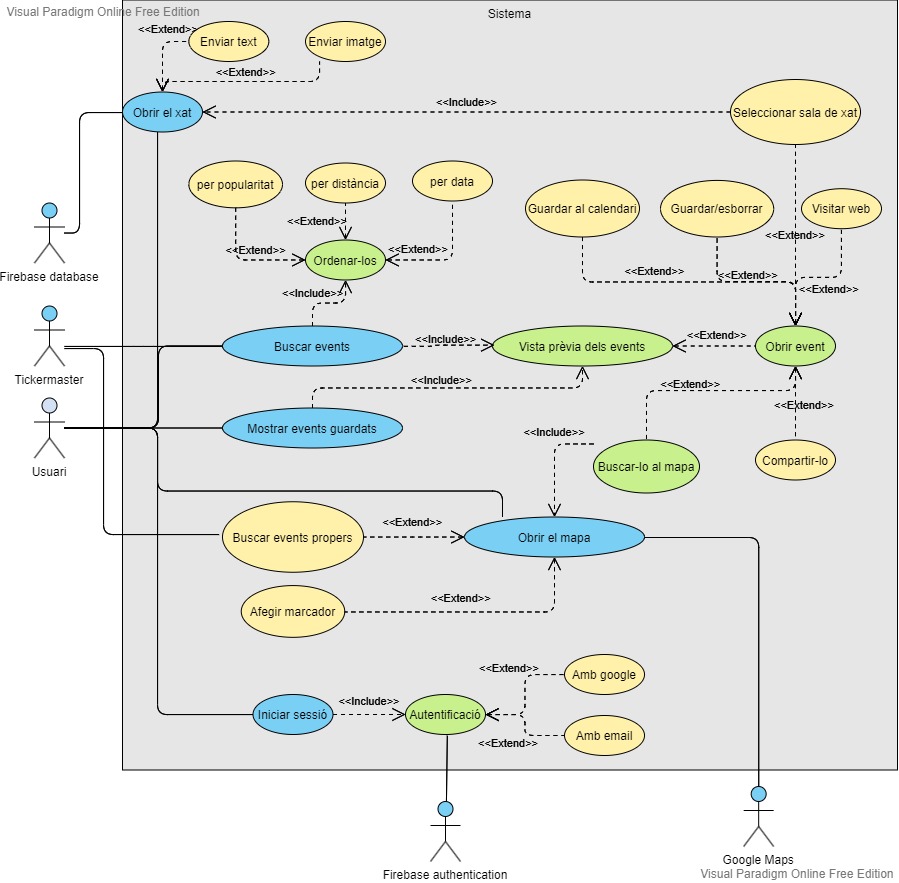


Figura: 2. Diagrama de casos d'ús, realitzat amb l’eina <https://online.visual-paradigm.com/>

### Descripció dels casos d’ús

Primerament, l’usuari ha d’iniciar sessió per tal de poder utilitzar l’aplicació, donant pas al cas d’us “*iniciar sessió”*. Aquest procediment requereix ser autenticat mitjançant el servei d’autenticació de Firebase, el que dona lloc al següent cas d’ús: “*autenticació”*. L’autenticació es pot fer aportant un correu i una contrasenya o aportant un compte de Google, opcions representades respectivament pels casos d’us “*amb e-mail*” i “*amb Google*”.

Per una altra banda hi ha cas d’us “*buscar events*”, que es dona quan l’usuari, amb l’ajuda dels serveis de l’API de Ticketmaster, busca events des de la pantalla anomenada “Eventos”. A més a més, abans de fer-ho haurà d’haver seleccionat un mètode d’ordenació per a determinar l’ordre amb que es mostraran aquests. Els casos d’us “*per popularitat*”, “*per data*” i “*per distància*” fan referència a les 3 maneres possibles d’ordenar els events.

El cas d’us “*mostrar events guardats*” determina quins events s’han guardat prèviament i el cas d’ús “*vista prèvia dels events*” mostra una vista amb informació reduïda d’aquests.

Si es fa clic en la vista prèvia s’arriba al cas d’us “*obrir event*”, on es mostra l’event a pantalla completa amb més funcionalitats i detalls. Aquestes són guardar l’event al calendari del dispositiu (*Guardar al calendari*), guardar una referència a l’event en l’aplicació o esborrar-la (*Guardar/esborrar*), visitar la pàgina web de Ticketmaster on comprar l’entrada de l’event (*visitar web*), compartir l’event mitjançant una aplicació del dispositiu (*compartir-lo*), buscar les coordenades de l’event en un mapa real (*buscar-lo al mapa*) i seleccionar la sala de xat d’aquest event concret (*seleccionar sala de xat*).

Al seleccionar la sala de xat es dona pas al cas d’ús “*Obrir el xat*”, que també el pot realitzar directament l’usuari i sempre conta amb els serveis de la base de dades de Firebase. Un cop al xat es poden enviar missatges de text (*Enviar text*) o imatges (*Enviar imatge*).

Al buscar un event al mapa es dona pas al cas d’ús “*Obrir el mapa*”, que també el pot realitzar directament l’usuari i sempre conta amb els serveis de la base de l’API de Google Maps. Un cop en el mapa, es poden afegir marcadors (*afegir marcador*) o buscar els events propers a la zona que s’està enfocant (*buscar events propers*), amb l’ajuda també de l’API de Ticketmaster.

# Descripció de la tecnologia utilitzada

Revisar:

Explicar APIS per sobre? Es pot fer una explicació de la seva configuració (aconseguir les keys, contes, sha1...). o posar el link a la explicacio oficial

• També es pot mencionar llibreries per sobre, sent que algunes com geoHash, Volley, Glide, gson... les explicarem més endavant, a l’apartat 6.

• Conceptes important de programació o Android Studio? Adapters, L’editor de disseny visual (i explicar que es una View, ViewGroup, widget...), sharedPrefs, lifeCycle... Actualment està en part fet a l’apartat 6, no se si serie millor aquí, la cosa es que hi ha infinites coses a explicar sobre això.

## Plataforma seleccionada

La plataforma seleccionada per al desenvolupament d’aquesta aplicació ha set “Android Studio”.

Android Studio és un IDE o entorn de desenvolupament integrat. Un IDE és una aplicació que combina en un únic lloc totes les eines necessàries per realitzar un projecte de desenvolupament de software i que consta com a mínim d'un editor de text, eines d'automatització de compilació i un depurador. A més, alguns IDE venen amb els beneficis d’instal·lar complements per estendre les seves funcionalitats a un altre nivell.

Revisar: <https://geekflare.com/es/best-ide-for-mobile-app-development/> per lo de abaix

La decisió d’utilitzar Android Studio ha estat presa avaluant les diferents eines per al desenvolupament d’aplicacions mòbils de les que es pot disposar actualment ( A.S. , QT IDE, Xcode, Eclipse IDE, JetBrains Rider, Visual Studio Code...).

Android Studio és la més complerta entre elles, ja que inclou editor de disseny visual: permet crear dissenys ràpidament arrossegant elements a la pantalla; analitzador de APK: verifica la composició de l’aplicació, redueix el temps de depuració i també la mida de la APK; emulador ràpid: conta amb l’emulador d’Android que ajuda a provar l’aplicació en diferents dispositius sense haver-los de tenir físicament; Un sistema de construcció flexible: permet generar múltiples versions de compilació per diferents dispositius utilitzant un únic projecte i moltes altres funcions.

Revisar:

SEGUIR DESDE AQUÍ. No se si fer la explicació detallada de android Studio. Si es fa s’haurie d’explicar la estructura principal i suposo que (projecte, manifest, gradle, editor visual, organització, emulador, SDK manager... però no s’acabarie mai, es massa extens.

INSERTAR PETITA EXPLICACIÓ AMB CAPTURES SOBRE EL FUNCIONAMENT GENERAL D’ANDROID STUDIO (projecte, manifest, gradle, editor visual, organització, emulador, SDK manager...)

## Llenguatge utilitzat: java i kotlin

### Java

Java és un llenguatge de programació i una plataforma informàtica que va ser comercialitzada per primera vegada el 1995 per Sun Microsystems. Actualment conta amb més de 10 milions d’usuaris i és una de les millors opcions al desenvolupar aplicacions per a dispositius Android, ja que aquest sistema operatiu també ha estat desenvolupat en Java.

És un llenguatge senzill orientat a objectes, que permet el desenvolupament d’aplicacions en diverses àrees com ara seguretat, animació, accés a bases de dades, client-servidor, aplicacions mòbils etc.

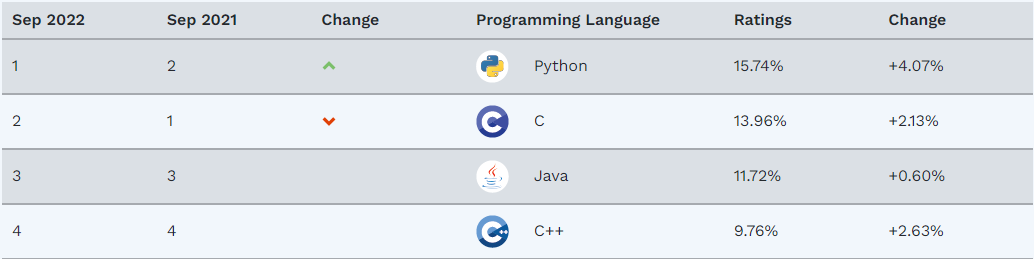


Figura: 3. Taula dels llenguatges de programació més utilitzats, de la pàgina <https://www.tiobe.com/tiobe-index/>

Java ha estat el llenguatge més utilitzat en aquest projecte ja que segueix sent el llenguatge més utilitzat per a desenvolupar aplicacions en Android, així com el 3r més utilitzat entre tots els llenguatges. Això implica que hi ha multitud de tutorials o comentaris d’ajuda en la xarxa que resulten molt útils a l’hora de realitzar un projecte sobre un tema que no s’ha tractat en profunditat.

A més a més, tant a l’assignatura de plataformes en xarxa com a la de aplicacions per a dispositius mòbils, on he treballat amb Android Studio, personalment també he utilitzat més Java.

### Kotlin

És un llenguatge de programació de codi obert creat per JetBrains que s'ha popularitzat gràcies al fet que es pot utilitzar per programar aplicacions Android.

Kotlin corre sobre una màquina virtual de Java, però s’ha especialitzat en aplicacions Android i permet reduir fins a un 40% les línies de codi d’una aplicació.

Gracies a això, Android va declarar al 2019 que prioritzaria aquest llenguatge sobre la resta i recomanava als desenvolupadors el seu ús, ja que el considera la millor opció i passaria a donar-li un suport prioritari. Alguns dels motius d’aquesta decisió són les valoracions de Kotlin:

Expressiu i concís: Redueix la quantitat de codi estàndard i augmenta la productivitat.

Codi més segur: No existeix l’excepció “NullPointerException” i ajuda a evitar errors. Es calculen un 20% menys d’errors.

Interoperabilitat: Kotlin és 100% interoperable amb Java, per tant es pot cridar codi basat en Java des de Kotlin i viceversa

Simultaneïtat estructurada: Kotlin facilita treballar amb codi asíncron o de bloqueig mitjançant corrutines

Revisar: <https://developer.android.com/kotlin/first#why>

Actualment, es calcula que el 60% dels desenvolupadors professionals d’Android utilitzen Kotlin.

En l’aplicació s’ha utilitzat Kotlin aprofitant la seva interoperabilitat amb Java per cobrir algunes àrees en les que Kotlin resultava més òptim per algun dels motius mencionats anteriorment, com a l’hora de realitzar el xat.

## Entorn d’execució

Un entorn d'execució o RE (runtime environment) és un estat de màquina virtual que subministra serveis per als processos d'un programa d'ordinador que s'està executant, o dit d’una altra manera aquest actua com un petit sistema operatiu que proveeix totes les funcionalitats necessàries perquè un programa s'executi.

Com Android Studio és una aplicació Java, es requereix de l’entorn d’execució de Java, anomenat JRE (Java Runtime Environment).

En la següent versió d’Android: Android 13, es planeja agregar una nova capacitat de plataforma que permetrà que els SDK de tercers s'executin en un nou entorn d'execució dedicat a l'entorn d'execució de SDK.

## SDK

Un SDK o Software Development Kit (Kit de desenvolupament de software) reuneix un conjunt d’eines que permeten la programació d’aplicacions mòbils.

Aquest conjunt d’eines es pot dividir en 3 categories:

SDK per a entorns de programació o sistemes operatius (iOS, Android, etc.)

SDK per al manteniment d'aplicacions

SDK de màrqueting i publicitat

A més a més, les aplicacions solen contenir codi de tercers en forma de SDK, com SDK d’anuncis o estadístiques. El RE permet que els diferents SDK s’executin de manera aïllada i asíncrona, facilitant també la comunicació bidireccional entre el procés de l’aplicació i el seu entorn d’execució del SDK.

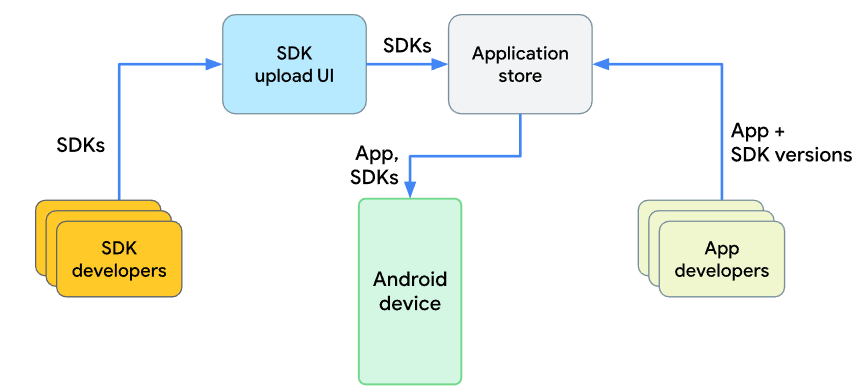


Figura: 4. Disseny de distribució del SDK, de la pàgina <https://developer.android.com/design-for-safety/privacy-sandbox/sdk-runtime>

# Disseny de l’aplicació

## Disseny de la interfície

La interfície de l’aplicació segueix un model simplista per no sobrecarregar la vista de l’usuari i totes les pantalles importants (inicio, eventos, mapa i chat) són accessibles amb un únic clic des de qualsevol d’aquestes mateixes pantalles utilitzant el menú inferior.

El color principal de l’aplicació és el blau, ja que en aplicacions mòbils representa seguretat, confiança i informació. Aquests aspectes són els adequats, ja que l’aplicació ha d’aportar informació sobre els events, i l’usuari ha de confiar en l’aplicació i sentir-se segur tant per anar als events oferts com per parlar amb els demés usuaris de l’aplicació. És per això que moltes xarxes socials o xats són blaus (com Facebook, Skype, Telegram...).

Principalment s’utilitzaran els tres colors següents:

El color principal és el següent:



Blau actual utilitzat com a color principal, amb codi HEX: #0336ff

Com a color secundari s’utilitza una variant d’aquest blau més saturada per donar-li menys importància.



Blau actual utilitzat com a color secundari, amb codi HEX: #819bff

Finalment també s’utilitza una tonalitat de taronja anàleg al blau principal per ressaltar certs elements que s’han de distingir més o que estan sobre fons blau.



Taronja actual utilitzat com a color extra, amb codi HEX: #ff4e03

El fons de l’aplicació es manté de color blanc per no saturar la vista amb colors innecessaris i fer-ho més clar i simple, amb alguns tocs grisos per distingir elements quan és necessari.

Per una altra banda, totes les icones que es mostren en l’aplicació s’han extret de l’ “Asset Studio” que inclou Android Studio. Totes elles són vectors, per tant tenen un gràfic vectorial com un arxiu XML que descriu la imatge que mostraran. En total s’utilitzen unes 40 icones.

En quant al text, aquest és majoritàriament de color negre per a que ressalti amb el fons blanc, amb l’excepció de texts en color gris si aquests s’eliminen automàticament quan l’usuari escriu en el lloc que guarden. També hi ha un text en blau en la pantalla de registre per ressaltar que és possible fer-li clic.

Una altra manera utilitzada per ressaltar texts amb els que l’usuari pot interactuar és subratllant-los, com amb el text per revocar l’accés al compte o les opcions d’ordenació dels events.

La mida dels texts varia segons l’activitat o fragment i l’espai del que aquest disposi, tot i que hi ha texts amb mida fixa (sempre ocupen el mateix número de píxels), i texts que s’adapten a les dimensions de la pantalla del dispositiu que els mostri.

En quant als botons, no se n’han utilitzat gaires ja que la majoria d’interaccions es realitzen per mitjà d’icones o Layouts. En les activitats d’inici de sessió i registre es conta amb dos botons amb el color blau principal i el text en gran, ja que són els botons que completen les accions de registre i inici de sessió i han de ressaltar.



Botó per a completar el registre.

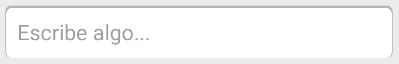
També està el botó d’inici de sessió amb Google, tot i que aquest utilitza el seu propi format predefinit ja que ressalta suficient i la gent només estar acostumada a ell.

En quant a les icones interactuables se’n poden trobar a gairebé totes les pantalles. La majoria tenen una mida de 24dp, tot i que algunes són més grans i arriben fins a 36dp i unes poques són més petites. Aquestes adopten diferents colors, ja que al ser vectors no en tenen cap inicialment: se’n poden trobar de grises, blaves, taronges, blanques i inclús que canvien de color, com la icona d’enviar missatge en el xat. El color depèn de la importància de la icona, del color general de la pantalla, del color de fons sobre el que està la icona i de si ha de ressaltar o no sobre altres.

Icones de perfil, tancar sessió i guardar event, respectivament.

En quant als camps d’entrada de text, tant al de la pantalla de xat com al de la pantalla de buscar events se’ls ha afegit un efecte d’ombra per a que quedin millor:



Camp d’entrada de text de la pantalla de xat.

Per últim, mencionar que el control també s’ha fet el més senzill possible: En les 4 pantalles principals es navega mitjançant el menú inferior, que permet a l’usuari desplaçar-se entre elles sense problemes. Un cop en una, es poden obrir noves activitats que eliminaran el menú inferior, però l’usuari sempre contarà amb una fletxa enrere present en el menú superior per tornar on estava anteriorment.

A més a més, el menú superior sempre indica el nom de la pantalla actual per a que l’usuari no es perdi.

## Mapa de navegació

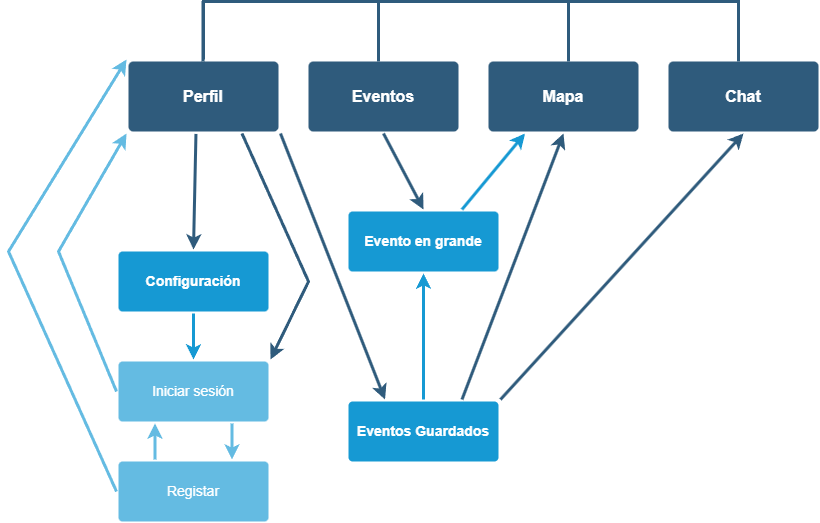


Figura: 5. Mapa de navegació, creat amb l’eina draw.io

Revisar: tmb es poden afegir els bocetos inicials de les pantalles a paper. Estan fets. O als annexos igual

## Disseny de pantalles

En primer lloc cal mencionar que els noms donats a les pantalles estan en castellà ja que aquest és el llenguatge base de l’aplicació.

L’aplicació conta amb un total de 8 pantalles diferents: Primerament estan les pantalles per a l’inici de sessió i per al registre. Un cop s’accedeix a l’aplicació amb un usuari registrat, s’obre la pantalla “Inicio”, des de la que es té fàcil accés a 3 pantalles més mitjançant el menú inferior. Aquestes són “Eventos”, “Mapa” i “Chat”. Finalment esta la pantalla per veure els events que s’han guardat al dispositiu, la qual s’anomena “Eventos Guardados” i la pantalla que mostra informació més detallada sobre un event concret, que s’anomena “Detalles de Evento”.

Pantalla d’inici de sessió

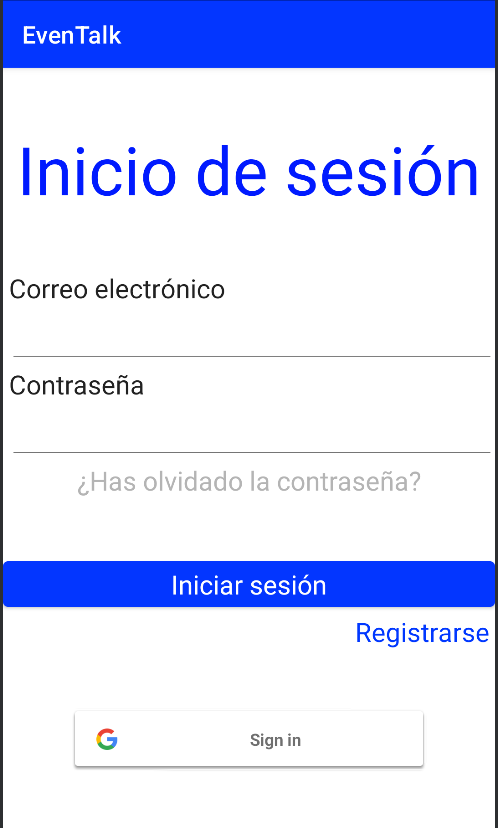


Figura: 6. Pantalla d’inici de sessió

Quan un nou usuari obra per primer cop l’aplicació, aquesta és la pantalla que es mostra.

El disseny és molt simple i amb els mínims elements possibles, utilitzant el color blau principal per a ressaltar alguns elements important i el negre i gris per la resta. El botó per iniciar sessió amb Google té el disseny predeterminat que implementa la seva llibreria.

Pantalla de registre

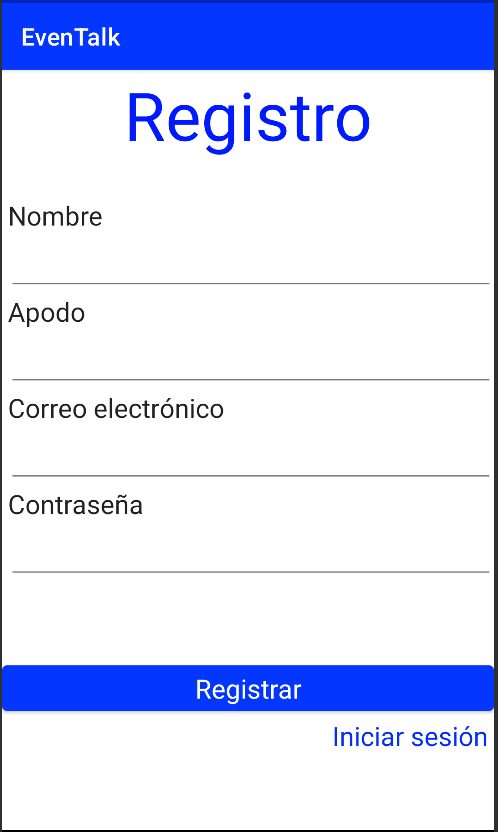


Figura: 7. Pantalla de registre

Aquesta pantalla conta amb un disseny molt similar a la d’inici de sessió ja que tenen funcions també similars i no convé que desentonin. De nou el títol i botons són de color blau mentre que el formulari és de color negre.

Pantalla Inicio

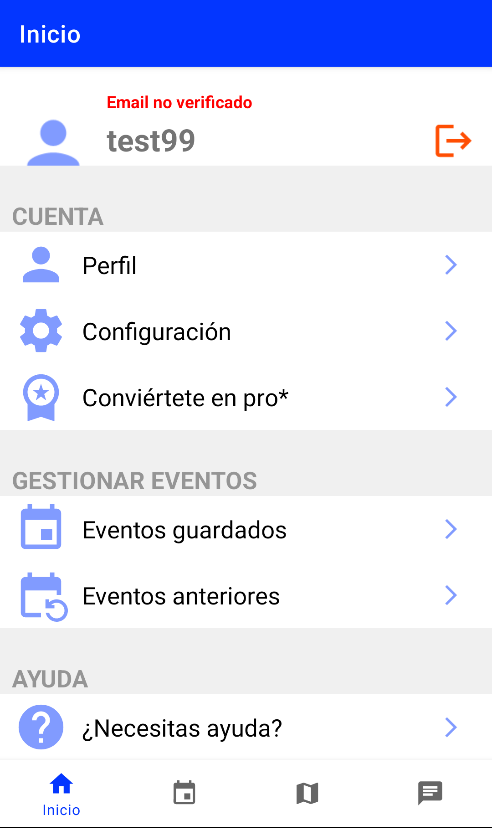


Figura: 8. Pantalla Inicio

Un cop l’usuari inicia sessió, l’aplicació mostra aquesta pantalla, anomenada “Inicio”.

Els colors principals també són el blau i el blanc, tot i que el blau que s’utilitza per les icones és el color secundari. És així ja que hi ha molts elements amb aquest color i el blau principal destacaria massa ja que és més fort, fent la vista incòmoda. A part és un color que es guarda per fer destacar elements concrets, per tant no pot estar distribuït en tots els elements de la pantalla. Per distingir entre apartats s’utilitza el gris per donar la impressió de relleu, ja que es sol utilitzar com a fons fent que el blanc cobri més importància.

Finalment, la icona per tancar la sessió és del color taronja anàleg al blau per fer-la destacar i el text “Email no verificado” és roig per a que l’usuari el percebi com un perill i verifiqui el correu.

En quan a l’estructura de la pantalla, aquesta segueix un patró concret que es repeteix 7 cops. Esta dividida en files horitzontals i cada una conté 3 elements. A l’esquerra estan les icones que indiquen visualment la funcionalitat de la fila, al mig hi ha un text que indica la funcionalitat per escrit i a la dreta hi ha una altra icona que mostra a l’usuari que pot interactuar amb la fila. Els apartats en gris s’utilitzen per dividir les funcionalitats en grups i donar-los-hi nom, fent la pantalla més entenedora i agradable visualment.

Pantalla Eventos

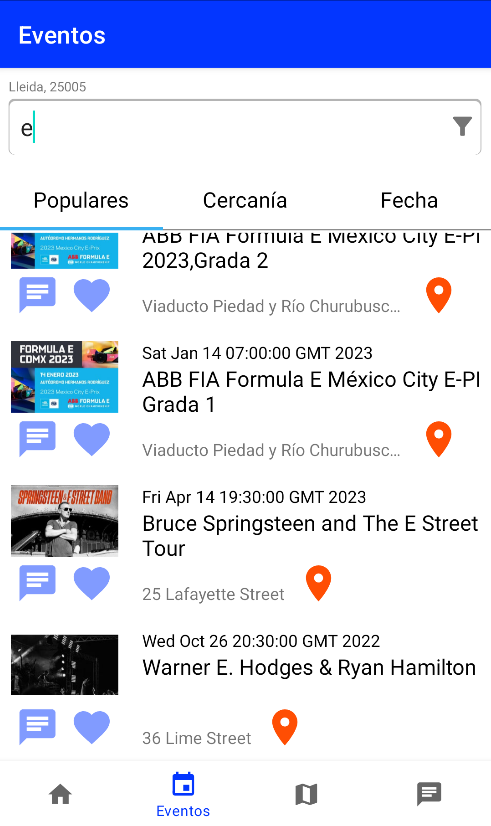


Figura: 9. Pantalla Eventos

La següent pantalla en el menú és diu “Eventos”. De nou el color de fons és el blanc i les icones són del blau secundari o taronja. Els texts més importants són de color negre mentre que els més petits o menys rellevants són de color gris.

L’estructura de la pantalla es divideix en la zona superior i inferior. En la zona superior es troba el buscador d’events seguit de les opcions d’ordenació d’aquests. L’opció d’ordenació que l’usuari escull es marca amb una línia de color blau fluix més ampla que les altres.

En la part inferior es mostren els events que s’han trobat. Cada event es mostra amb el mateix disseny, que inclou una imatge, la data i hora de l’event, el nom d’aquest, la seva localització i tres icones: una d’un xat, una d’un cor i una d’ubicació. La última és de color taronja i no blau com les demés perquè canvia de posició segons la llargada del text que indica la direcció de l’event, per tant cal ressaltar-la més. A més a més manté a l’usuari atent, ajuda a l’usuari a distingir els events i dona millor impressió que una icona d’ubicació blava a la que la gent no sol estar acostumada.

Pantalla Mapa



Figura: 10. Pantalla Mapa

La tercera pantalla del menú és el mapa. Aquesta pantalla manté els menús superior i inferior però únicament mostra el mapa que aporta la API de Google Maps. Els marcadors dels events són del clàssic color roig i el text que mostren és negre sobre un fons blanc per a distingir-los.

Pantalla Chat



Figura: 11. Pantalla Chat

L’última pantalla del menú és el xat. Aquesta té el fons de color blanc ja que mostra els missatges de dos possibles colors: si ha estat enviat per una altra persona, el missatge es veu de color gris, mentre que si l’ha enviat el propi usuari del compte es mostra de color blau. A més a més, si el fons del missatge és blau el text es mostra de color blanc, mentre que si el fons del missatge és gris, el text d’aquest es mostra de color negre. Com el fons de la pantalla és blanc, la imatge de perfil d’usuari predeterminada té el fons negre, mentre que si l’usuari té una imatge de perfil pròpia, aquesta és manté intacta i es mostra en una forma circular.

En la part inferior de la pantalla hi ha una zona horitzontal grisa que serveix per distingir l’espai destinat a rebre i llegir missatges de l’espai destinat a escriure’ls i enviar-los. A l’esquerra d’aquesta zona, de color taronja, es troba la icona per penjar imatges, al mig es troba el camp de text per escriure el missatge, amb un text temporal de color gris clar per mostrar que no és gaire important i a la dreta el botó per enviar el missatge. Aquest és manté de color gris mentre no hi ha text al camp anterior però passa a color taronja quan es comença a escriure un missatge. Així s’evita que l’usuari el premi sense escriure res i s’evita tenir més icones taronges (quantes menys millor, ja que ressalten molt), però es aconsegueix que l’usuari es fixi en el botó quan l’ha d’utilitzar.

Pantalla Eventos Guardados

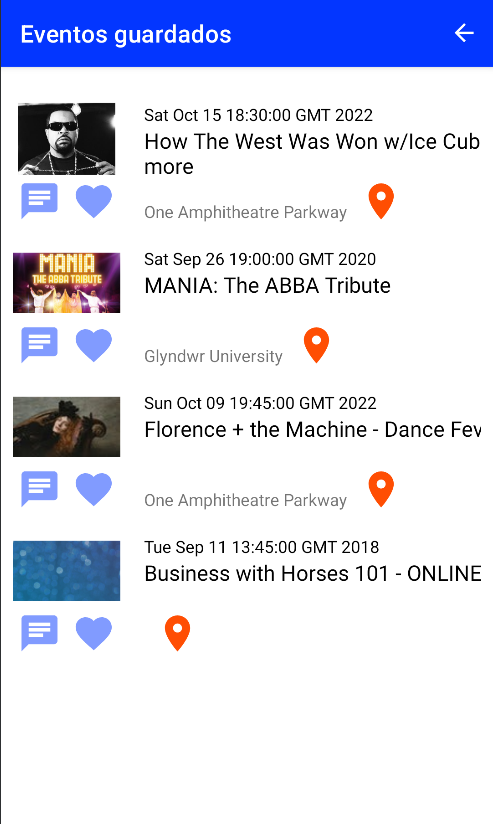


Figura: 12. Pantalla Eventos Guardados

A la pantalla que mostra els events guardats s’accedeix des de la pantalla “Inicio”. Aquesta pantalla s’ha mantingut el més simple possible ja que només té la funció de mostrar una llista d’events prèviament guardats, per tant afegir més elements sense funcionalitat no té cas i confondria a l’usuari. L’estructura dels events mostrats és la mateixa que en la pantalla “Eventos”.

En el menú superior s’ha afegit la icona d’una fletxa enrere de color blanc per donar contrast amb el fons blau.

Pantalla Detalles de evento

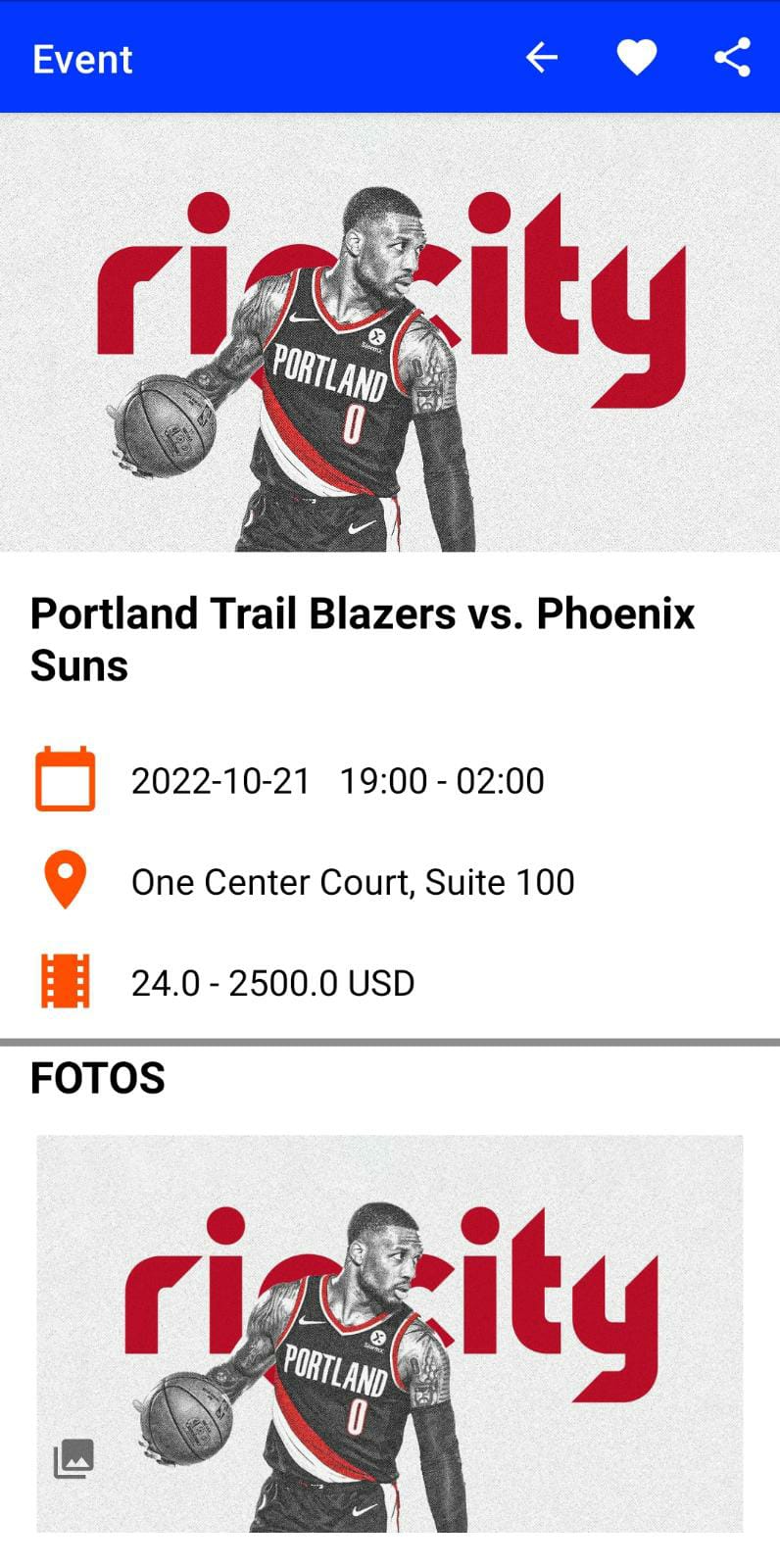


Figura: 13. Pantalla Detalles de evento

Finalment, l’última pantalla mostra informació sobre un event concret. El color de fons és blanc com de costum, tot i que en aquesta pantalla les 3 icones més importants són de color taronja. És així per a que adquireixin més rellevància comparades amb els títols i imatges en gran.

Tots els texts de la pantalla són de color negre, però es distingeix la seva importància mitjançant l’ús de negreta, diferència de mida i majúscules.

En la part superior hi ha una imatge d’alta qualitat que ocupa tot l’ample de la pantalla.

En la part del mig hi ha la informació de l’event, que inclou nom, data i hora, direcció i preu. Com totes les icones de color taronja de l’aplicació, les que es mostren en aquesta zona també són interactives i tenen certes funcionalitats.

En la part inferior, separada per una línia grisa, hi ha una galeria d’imatges de l’event.

## Disseny de menús

L’aplicació conta amb dos senzills menús per navegar.

El primer i més important és un “Tab bar”. Aquest tipus de menús es caracteritza per utilitzar ítems i navegar entre pantalles mútuament exclusives en una mateixa vista.

Aquest menú està situat en la part inferior de la pantalla i conté les 4 pantalles més importants: Inicio, Eventos, Mapa i Chat. Es mostra solament en aquestes mateixes pantalles ressaltant la que està en ús.



Figura: 14. Menú inferior, on està seleccionada la primera opció

El segon menú és el superior, però aquest varia depenent de la pantalla en la que ens trobem per adaptar-se a les necessitats de cada una:

En les 4 pantalles mencionades anteriorment, simplement mostra el nom de la pantalla.

En pantalles derivades d’aquestes, mostra també una icona d’una fletxa apuntant cap a l’esquerra, que permet tornar a la pantalla anterior al prémer-la.

En la pantalla que mostra l’event en gran (Detalles de evento) apareixen dues icones més: una és un cor, que permet a l’usuari guardar l’event en qüestió al seu perfil i l’altra és la icona de compartir, que permet compartir l’event per diverses vies que estiguin disponibles al dispositiu.

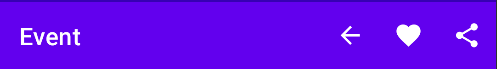


Figura: 15. Menú superior en la pantalla Detalles de Evento

# Implementació

## CONCEPTES

Abans de començar a explicar la fase implementació s’explicaran uns quants termes i conceptes que es veuran repetidament al llarg d’aquest apartat per facilitar la futura comprensió:

* **Layout:** Una Layout defineix l'estructura d'una interfície d'usuari en l’aplicació. Tots els elements en la Layout es construeixen mitjançant una jerarquia d'objectes View i ViewGroup. Hi ha diferents tipus de Layouts que es veuran al llarg d’aquest apartat.
* **View:** És una classe que representa el bloc bàsic per als components de la interfície d'usuari. Una View ocupa una àrea rectangular a la pantalla i s'encarrega del dibuix i de la gestió d'esdeveniments. View és la classe base dels anomenats “Widgets”, que s'utilitzen per crear components interactius d'IU (botons, camps de text, etc.).

En la resta de l’apartat els objectes View seran referenciats amb el nom de vistes.

* **Binding:** La “Data Binding Library” és una biblioteca de suport que permet enllaçar components de la interfície d'usuari de les Layouts a fonts de dades de l'aplicació mitjançant un format declaratiu en lloc de programar.

És una funció que permet escriure més fàcilment el codi que interactua amb les vistes.

* **Activity:** Una activitat proporciona la finestra en què l'aplicació dibuixa la seva interfície d'usuari i aquesta finestra normalment omple la pantalla.
* **Fragment:** És una part reutilitzable de la interfície d'usuari de les aplicacions que està vinculada a una activitat.
* **onCreate() i onCreateView():** onCreate() és una funció que es crida quan es crea una activitat per primera vegada o per fer la creació inicial d’un Fragment, mentre que onCreateView() es crida una vegada que el Fragment ha d'inflar una vista.

És a dir que fan funcions similars però s’utilitzarà onCreate() en les activitats i onCreateView() en els Fragments.

* **Intent:** Serveix per realitzar una acció en la pantalla. S'utilitza principalment per iniciar una activitat, iniciar serveis i enviar missatges entre dues activitats.
* **Listener:** És una interfície de la classe View que conté un únic mètode “callback” o de devolució de trucada. Aquest mètode serà cridat quan la vista a la qual s'ha registrat el Listener s'activi per la interacció de l'usuari amb l'element a la interfície d'usuari.
* **JSON:** Es tracta d’un format que serveix per guardar i intercanviar informació. Els arxius JSON contenen solament text i utilitzen l’extensió .json.

Funciona guardant la informació d’una manera estructurada seguint un sistema de Clau/Valor que en facilita també la lectura.

## Decisions d’implementació

Primerament es voldria mencionar que moltes decisions que s’han pres al llarg del procés no estan presents en el resultat final ja sigui perquè no eren òptimes, perquè n’han sorgit de millors, per requeriments tècnics o per el propi camí que ha seguit el desenvolupament de l’aplicació.

Per resumir aquest projecte i no saturar aquest capítol no es mencionaran les decisions que finalment s’han canviat o eliminat, per tant només es parlarà de les decisions que s’han implementat en el producte final.

Inicialment es tenia en ment la idea de les 4 pantalles principals amb unes funcions bastant similars a les que realitzen actualment, per aquest motiu la primera decisió en la implementació de la aplicació ha set el Tab Bar inferior des de el que es gestionen les pantalles principals.

Aquesta Tab Bar s’ha realitzat utilitzant una “BottomNavigationView”, que representa una barra de navegació inferior estàndard. Això implica que l’activitat activa en les pantalles principals és la MainActivity i les diferents pantalles que es mostren són fragments, o sigui porcions reutilitzables de la interfície d’usuari.

Aquest menú inferior es defineix al fitxer bottom\_nav\_menu, i es vinculen els fragments al fitxer mobile\_navigation.

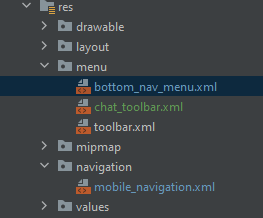


Figura: 16. Captura que mostra els fitxers bottom\_nav\_menu i mobile\_navigation

Per facilitar treballar amb els diferents fragments, tots implementen el sistema de vinculació de vistes “Binding” que facilita la cerca d’elements de la Layout.

Revisar: no m’agrade com quede aquest paràgraf. El següent que s’ha implementat és una base no funcional de les pantalles principals. Aquesta serveix per donar un esquelet sobre el que treballar i poder aplicar les funcionalitats de l’aplicació sense acabar de definir el disseny final, que pot variar una mica en funció del desenvolupament.

Seguidament es procedirà a explicar el funcionament i estructura del codi que s’ha creat per l’aplicació, seguint un ordre que permeti fer-ho el més entenedor possible. S’ha de tenir en compte que només en els fitxers Java i Kotlin hi ha un total de més de 3000 línies de codi, per tant no es senzill explicar tots els conceptes d’una manera lineal, ja que molts estan relacionats entre ells i es reutilitzen.

Tot i així, el contingut s’ha ordenat per classes de manera que es respecta la navegació natural que seguiria un usuari al entrar a l’aplicació.

Revisar: això aquí o a l’apartat 4? O al principi del tot? algun concepte més a incloure?

## CODI

Seguint la navegació natural que seguiria un usuari al entrar a l’aplicació, la primera pantalla amb la que es trobaria és la d’inici de sessió, controlada per l’activitat SignInActivity:

SignInActivity

És la primera activitat que veurà un usuari quan obri l’aplicació per primer cop. Aquesta s’ocupa de mostrar un formulari d’inici de sessió, així com opcions alternatives com ara recuperar la contrasenya, registrar-se per primer cop o accedir amb un compte de Google.

El disseny de l’activitat és bastant simple ja que la majoria d’elements UI que el componen són “TextView”, amb l’excepció de dos “EditText” per al correu i contrasenya del formulari i dos botons: un “Button” per a accedir al compte amb l’usuari i contrasenya i un “com.google.android.gms.common.SignInButton” per accedir amb el compte de Google escollit.

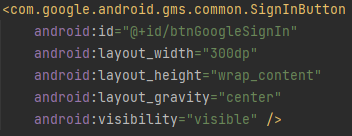


Figura: 17. Part de l'arxiu activity\_sign\_in.xml on es mostra el codi del botó "SignInButton" de Google

La primera funció que la classe SignInActivity executa és onCreate(): Aquesta s’ocupa d’inicialitzar les variables requerides de Firebase i Google:

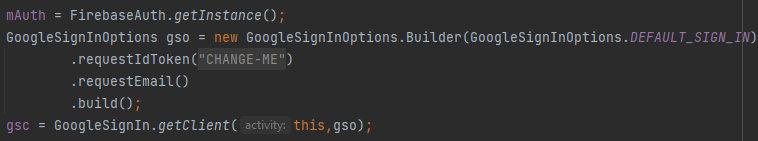


Figura: 18. Part de la funció onCreate(), de la classe SignInActivity

Primerament s’obté una instància de la classe abstracta FirebaseAuth, que és el punt inicial de la Firebase Authentication SDK i ’utilitzarà per obtenir l’usuari. Es realitza amb el mètode:

*mAuth = FirebaseAuth.getInstance();*

Després s’obté també una instància de la classe GoogleSignInClient, que és un client per interactuar amb la API d’inici de sessió de Google. Per fer-ho s’utilitza el mètode:

*gsc = GoogleSignIn.getClient(this,gso);*

On “gso” és una instància de la classe GoogleSignInOptions, que conté opcions utilitzades per configurar el GoogleSignInClient. Per crear-lo s’utilitza el mètode:

*GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT\_SIGN\_IN)*

*.requestIdToken(getString(com.firebase.ui.auth.R.string.default\_web\_client\_id))*

*.requestEmail()*

*.build();*

La funció onCreate () també s’ocupa de redirigir a l’usuari a la pantalla principal en cas que ja hagi iniciat sessió:

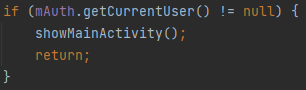


Figura: 19. Part de la funció onCreate(), de la classe SignInActivity

I de registrar Listeners per cada element de la UI que ha de ser interactuable: els dos botons de SignIn, el text de “register” i el text per recuperar la contrasenya.



Figura: 20. Part de la funció onCreate(), de la classe SignInActivity

Quan es fa clic al botó per iniciar sessió amb usuari i contrasenya es crida a la funció “authenticateUser()”.



Figura: 21. Funció authenticateUser(), de la classe SignInActivity

Aquesta obté el correu i contrasenya de les vistes pertinents i inicia la sessió amb el mètode:

*mAuth.signInWithEmailAndPassword(email, password)*

Si funciona, redirigeix l’usuari a la pantalla principal de l’aplicació: MainActivity i si no, mostra un quadre de text indicant si falta per omplir algun camp o si ha fallat l’autenticació per altres motius.

Quan es fa clic al botó per iniciar sessió amb Google es crida a la funció SignIn(), que s’ocupa d’iniciar el procés.

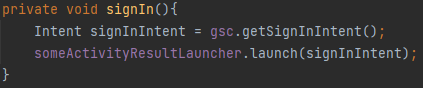


Figura: 22. Funció signIn(), de la classe SignInActivity

Primerament obté un Intent per iniciar el flux d’inici de sessió de Google amb el mètode

*gsc.getSignInIntent()*

I després l’inicia amb el mètode:

*someActivityResultLauncher.launch(signInIntent);*

On someActivityResultLauncher és el nom de l’objecte ActivityResultLauncher que s’utilitza per rebre la resposta de l’intent un cop finalitzi.



Figura: 23. ActivityResultLauncher anomenat someActivityResultLauncher, de la classe SignInActivity

Si l’Intent finalitza amb èxit, només queda extreure les dades del resultat amb el mètode

*Intent data = result.getData();*

Crear una tasca de tipus “GoogleSignInAccount” amb elles utilitzant el mètode:

*GoogleSignIn.getSignedInAccountFromIntent(data);*

I extreure l’objecte GoogleSignInAccount d’aquesta tasca, que és el que conté la informació bàsica del compte de l'usuari de Google que ha iniciat la sessió, mitjançant el mètode

*task.getResult(ApiException.class);*

Un cop s’ha obtingut, es crida a la funció firebaseAuthWithGoogle(account); i es passa l’objecte per paràmetre.

La funció firebaseAuthWithGoogle() s’ocupa d’iniciar la sessió.

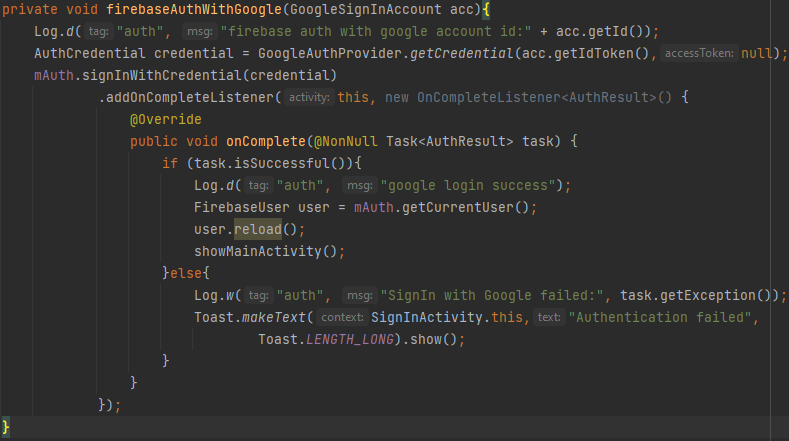


Figura: 24. Funció firebaseAuthWithGoogle (), de la classe SignInActivity

Per fer-ho, utilitza l’objecte GoogleSignInAccount per obtenir la credencial necessària per a que el servidor d’autenticació de Firebase autentiqui a l’usuari:

*AuthCredential credential = GoogleAuthProvider.getCredential(acc.getIdToken(),null);*

I inicia la sessió amb aquesta credencial:

*mAuth.signInWithCredential(credential)*

Un cop s’ha iniciat la sessió amb èxit, es redirigeix a l’usuari a la pantalla principal.

Per una altra banda, si l’usuari fa clic al TextView amb text “¿Has olvidado la contraseña?”, es crida a la funció showRecoverPasswordDialog().

La funció showRecoverPasswordDialog() s’ocupa de mostrar un missatge per pantalla que permet escriure una direcció de correu electrònic per recuperar la contrasenya.



Figura: 25. Captura del quadre de diàleg enviat per recuperar la contrasenya

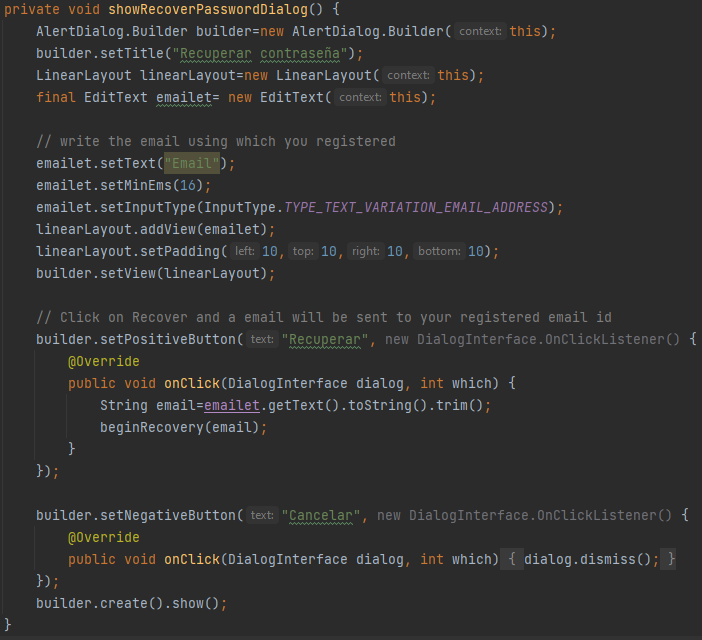


Figura: 26. Funció showRecoverPasswordDialog (), de la classe SignInActivity

Per crear i mostrar el missatge s’utilitza un AlertDialog.Builder:

*AlertDialog.Builder builder=new AlertDialog.Builder(this);*

Que s’edita per donar-li un títol amb el mètode

*builder.setTitle("Recuperar contraseña");*

I Un camp de text on introduir el correu electrònic dins d’una LinearLayout:

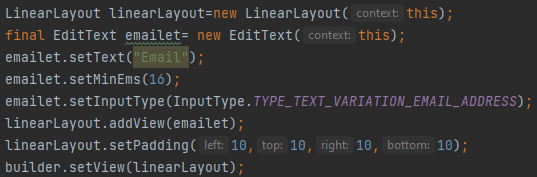


Figura: 27. Part de la funció showRecoverPasswordDialog (), de la classe SignInActivity

Després es creen i s’especifica la funcionalitat dels botons positiu i negatiu, anomenats “recuperar” i “cancelar” respectivament. El botó negatiu tanca el missatge amb el mètode

*dialog.dismiss();*

I el botó positiu crida a la funció beginRecovery (email), on “email” és el correu escrit que s’obté del camp de text anteriorment creat amb el mètode:

“*String email=emailet.getText().toString().trim();*”.

La funció beginRecovery() s’ocupa d’enviar un correu per a canviar la contrasenya del compte a la direcció especificada.

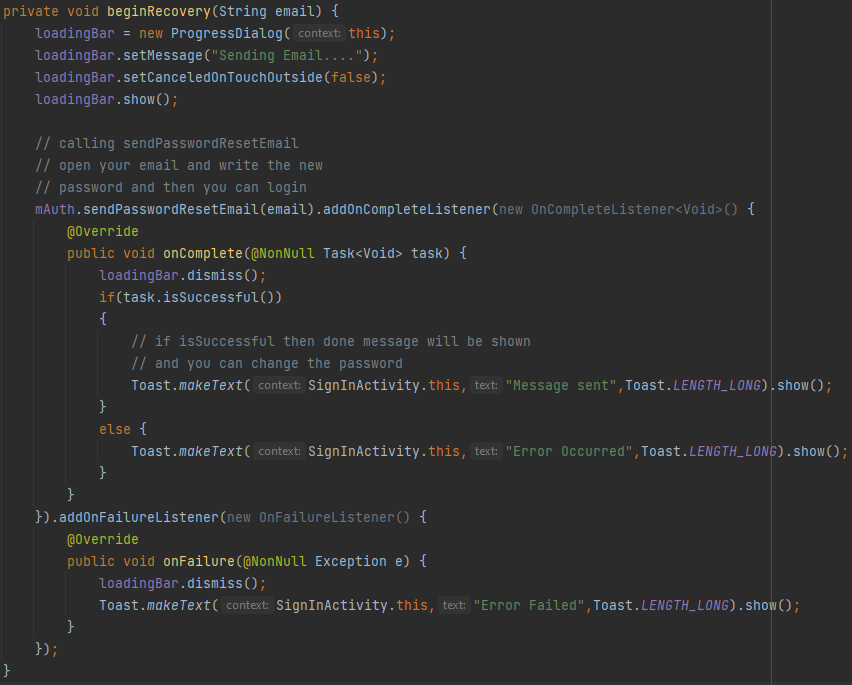


Figura: 28. Funció beginRecovery (), de la classe SignInActivity

Primerament mostra un indicador de progrés amb un missatge de text per indicar que el correu s’està enviant, mitjançant un ProgressDialog.

Seguidament envia el missatge per canviar la contrasenya al correu especificat amb el mètode:

*mAuth.sendPasswordResetEmail(email)*

Quan la operació acaba, es tanca l’indicador de progrés i es notifica a l’usuari si el correu s’ha enviat amb èxit o no mostrant un missatge per pantalla.

Finalment, si l’usuari fa clic al TextView amb text “Registrarse”, es crida a la funció switchToRegister().

Aquesta funció simplement obra l’activitat RegisterActivity.

RegisterActivity

Aquesta activitat s’ocupa de mostrar un formulari de registre, així com l’opció de tornar a la pantalla d’inici de sessió.

El disseny de l’activitat és molt similar a SignInActivity ja que la majoria d’elements UI que el componen són “TextView”, amb l’excepció de dos botons, un per a completar el registre i un altre per enviar un correu electrònic per a validar el correu indicat.

El codi d’aquesta també es bastant similar: la primera funció és onCreate(), i també s’ocupa de redirigir a l’usuari a la pantalla principal en cas d’estar registrat i registrar els Listeners necessaris.

El primer Listener detecta quan es fa clic en el botó de registre i crida a la funció registerUser()

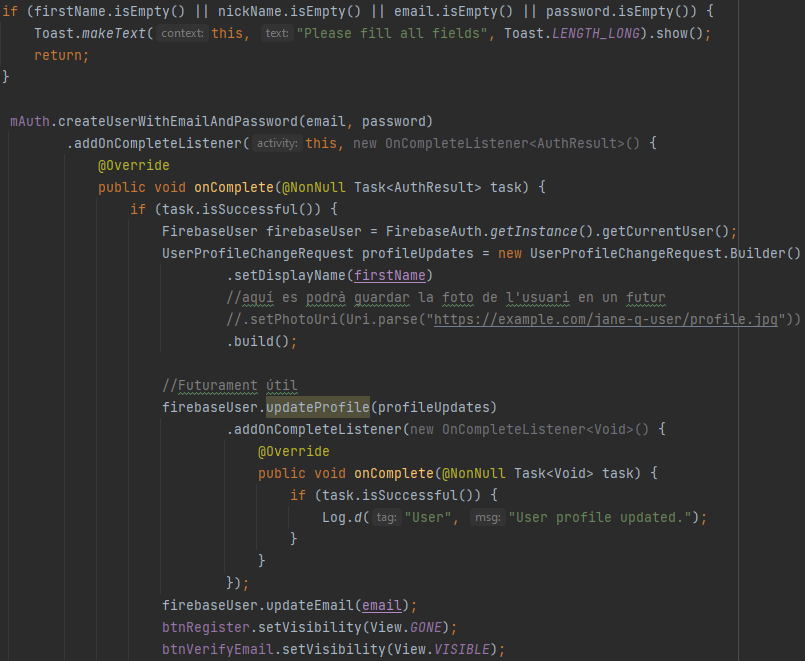


Figura: 29. Part de la funció registerUser (), de la classe RegisterActivity

Aquesta obté les dades de tots els camps omplerts per l’usuari i crea l’usuari amb el mètode:

*mAuth.createUserWithEmailAndPassword(email, password)*

Després s’actualitza la informació de l’usuari per afegir el nom utilitzant la classe UserProfileChangeRequest i el mètode setDisplayName():

*UserProfileChangeRequest profileUpdates = new UserProfileChangeRequest.Builder()*

*.setDisplayName(firstName)*

*.build();*

Per últim, es canvia el botó de “Registrar” per el de “Verificar Email” fent-n’hi un invisible i l’altre visible amb el mètode:

*btnRegister.setVisibility(View.GONE);*

*btnVerifyEmail.setVisibility(View.VISIBLE);*

El segon Listener registrat detecta quan l’usuari clica en el recentment mencionat botó “Verificar Email” i crida a la funció verifyEmail().

Aquesta s’ocupa d’enviar un correu electrònic a la direcció de correu registrada a Firebase. El correu enviat li permet a l’usuari verificar la seva direcció de correu electrònic en el compte de l’aplicació.

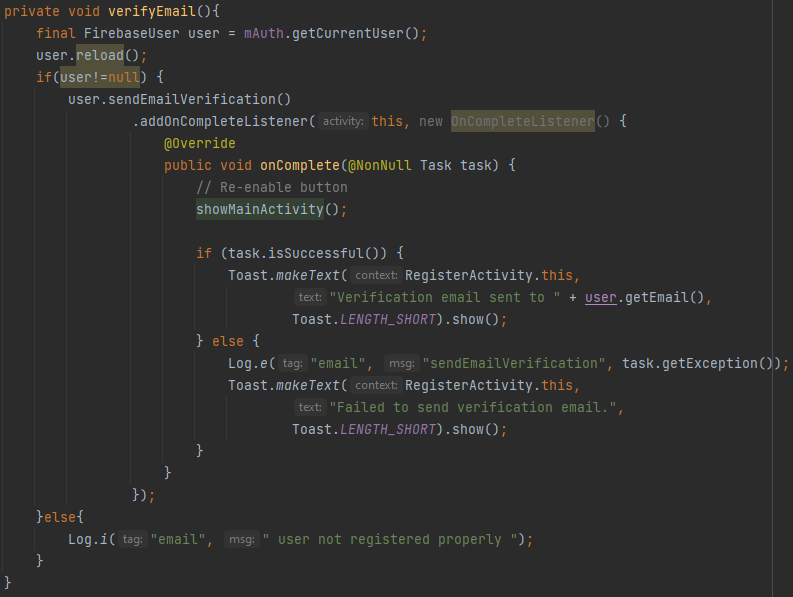


Figura: 30. Funció verifyEmail (), de la classe RegisterActivity

Primerament s’obté l’usuari registrat a Firebase amb el mètode

*mAuth.getCurrentUser();*

I seguidament s’envia el correu amb el mètode

*FirebaseUser.sendEmailVerification()*

Fianlment es notifica a l’usuari si el correu s’ha enviat o no amb èxit i de ser així s’obra la pantalla principal MainActivity.

Per últim, el tercer Listerner detecta quan l’usuari clica en el text “Iniciar sesión” i crida a la classe switchToLogin(), que simplement s’ocupa de retornar a la pantalla d’inici de sessió:

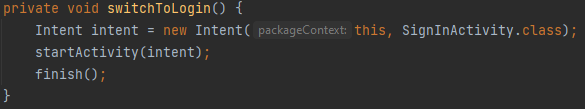


Figura: 31. Funció switchToLogin (), de la classe RegisterActivity

MainActivity

És l’activitat principal, el que significa que és la primera activitat en carregar. Cada aplicació només pot tenir una “main activity” i s’especifica a l’arxiu AndroidManifest.xml:

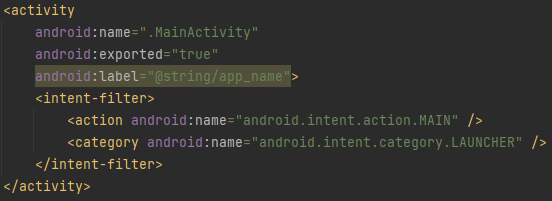


Figura: 32. Part de l’arxiu AndroidManifest.xml on s’especifica l’activitat principal

L’arxiu de disseny d’aquesta activitat només conta amb dos elements. El primer serveix per allotjar el menú inferior i és un “BottomNavigationView”

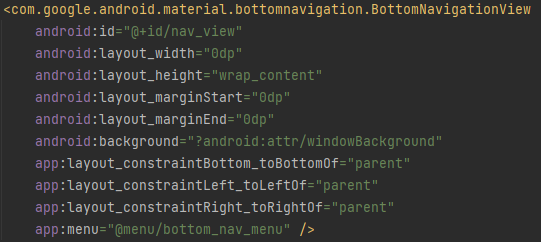


Figura: 33. Part de l’arxiu MainActivity.xml on es crea el menú inferior tipus “BottomNavigationView”.

Aquest representa una barra de navegació inferior estàndard per a l'aplicació.

El segon element és un Fragment.



Figura: 34. Part de l’arxiu MainActivity.xml on es crea el “Fragment”.

Els Fragments que es vinculen a l’activitat principal són 4 en total, i es navega entre ells mitjançant el menú inferior de la MainActivity. Aquests són els fragments “Inicio”, “Eventos”, “Mapa” i “Chat”.

Tot i no tenir més components de disseny pròpies, MainActivity sí realitza certes tasques.

La funció principal de la classe MainActivity és onCreate().

El primer que es fa en aquesta funció és comprovar si s’ha de redirigir a l’usuari a una altra pantalla. Si l’usuari no ha iniciat sessió és redirigit a l’activitat SignInActivity():

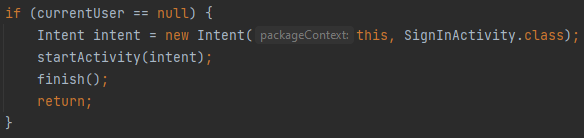


Figura: 35. Part de la funció onCreate() de l'activitat MainActivity

Després, s’inicialitza el component de navegació

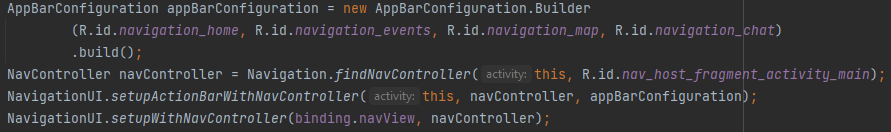


Figura: 36. Part de la funció onCreate() de l'activitat MainActivity

Primerament s’obté un Objecte “AppBarConfiguration”. Aquest conté les opcions de configuració per als mètodes “NavigationUI” que interactuen amb implementacions del patró de la barra d'aplicacions (el menú inferior). Per obtenir-lo s’utilitza el mètode:

*AppBarConfiguration appBarConfiguration = new AppBarConfiguration.Builder(R.id.navigation\_home, R.id.navigation\_events, R.id.navigation\_map, R.id.navigation\_chat)*

*.build();*

Els 4 paràmetres “R.id.navigation\_NOM” fan referència als 4 fragments en l’arxiu “Mobile\_navigation.xml” i serviran per poder desplaçar-se entre ells utilitzant el menú inferior.

Aquest arxiu permet identificar les pantalles entre les quals es navegarà i crear un gràfic de navegació entre elles:

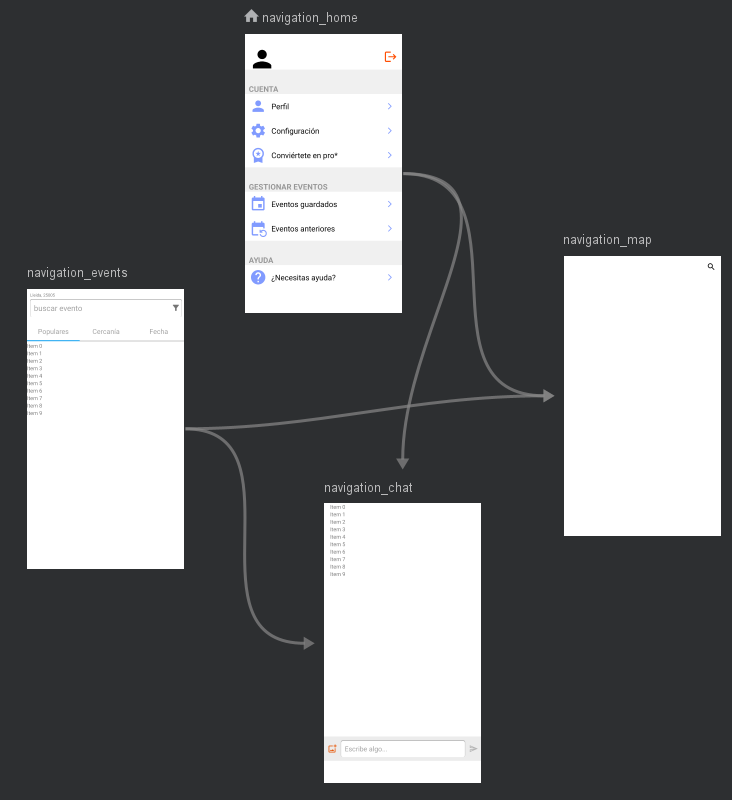


Figura: 37. Gràfic de navegació entre fragments

Les fletxes mostren entre quins fragments existeix una acció per desplaçar-se (a part del menú inferior), i aquestes accions venen identificades per un id propi, per exemple el codi del fragment “Inicio” en aquest arxiu és:

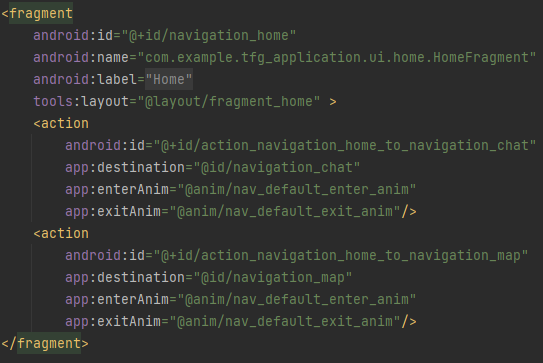


Figura: 38. Part de l'arxiu mobile\_navigation.xml on es mostra el codi del fragment "Inicio"

On cada “action” representa una de les fletxes que surten de la seva pantalla i van al fragment “mapa” o “chat”.

L’usuari també és redirigit si prové d’una altra pantalla i ha passat dades concretes a MainActivity per a que el redirigeixi. És dona el cas quan no es pot accedir directament a un fragment de MainActivity i s’ha de cridar a aquesta activitat primer.



Figura: 39. Part de la funció onCreate() de l'activitat MainActivity

Si l’activitat rep informació de la pantalla anterior, es busca un valor tipus enter que indica quin fragment s’ha d’obrir, i s’obra mitjançant el mètode:

*Navigation.findNavController(findViewById(R.id.profile\_layout)).navigate*

*(R.id.action\_navigation\_home\_to\_navigation\_map, bundle, navOptions);*

Aquest mètode utilitza el component de navegació, que permet navegar entre les diferents peces de contingut de l’aplicació, i utilitza també les accions recentment mencionades per anar del fragment “inicio” als fragments “mapa” (R.id.action\_navigation\_home\_to\_navigation\_map) o “chat” (R.id.action\_navigation\_home\_to\_navigation\_chat), ja que el fragment “inicio” és el primer en obrir-se automàticament.

La següent funció important és onStart(), ja que s’ocupa d’iniciar el procés per a demanar permisos.

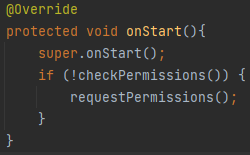


Figura: 40. Funció onStart() de l'activitat MainActivity

Primer comprova si s’han de demanar els permisos o no amb la funció checkPermissions()

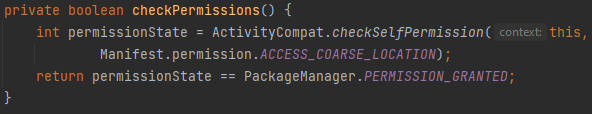


Figura: 41. Funció checkPermissions() de l'activitat MainActivity

Aquests mètodes comproven si l’usuari ha donat permís per utilitzar la seva localització aproximada, la anomenada Coarse location, i de ser així es retorna un valor vertader.

De no ser així, es comença el procés per obtenir els permisos amb la funció requestPermissions()

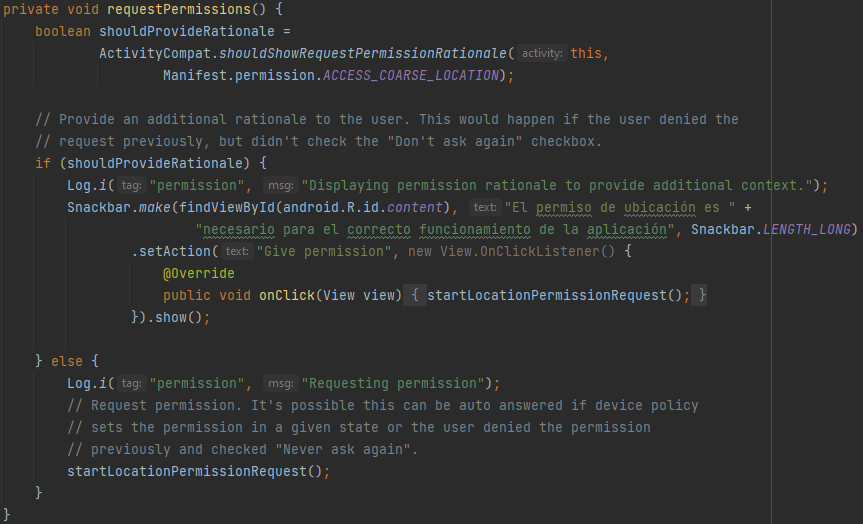


Figura: 42. Funció requestPermissions() de l'activitat MainActivity

Aquesta funció comprova si cal mostrar un quadre de diàleg addicional abans de demanar el permís a l’usuari. Això passa quan l’usuari ha denegat el permís prèviament però no ha marcat la casella “no volver a preguntar”. Per comprovar-ho s’utilitza el mètode:

*ActivityCompat.shouldShowRequestPermissionRationale(this,*

*Manifest.permission.ACCESS\_COARSE\_LOCATION);*

Que retorna un valor vertader en cas que s’hagi de mostrar el quadre de diàleg addicional.

Finalment, si l’usuari clica el botó “Dar permiso” del quadre de diàleg o el quadre no es necessari, es crida a la funció startLocationPermissionRequest()

Aquesta funció és la que demana el permís especificat, fent que es mostrin a l’usuari les possibles respostes que pot escollir:

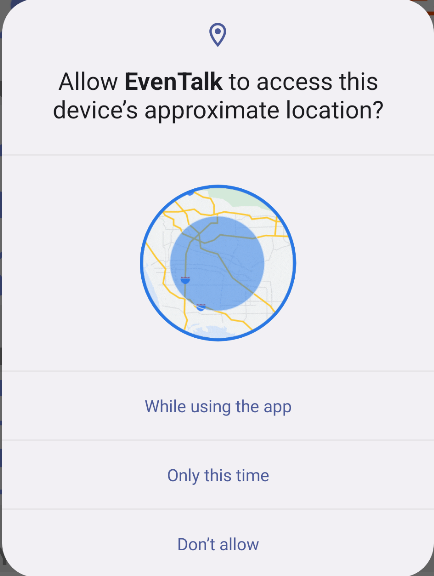


Figura: 43. Quadre que mostra les opcions que té l'usuari per atorgar o no el permís d'ubicació

Opcions que l’usuari rep sobre el permís per accedir a la ubicació aproximada del dispositiu.



Figura: 44. Funció startLocationPermissionRequest() de l'activitat MainActivity

Un cop s’escull una de les opcions, es crida automàticament a la funció onRequestPermissionsResult(), que rep el resultat de la operació així com l’opció escollida.

Si l’usuari accepta el permís i no hi ha cap error, no es fa res.

Si hi ha algun error durant el procés, es notifica a l’usuari i aquest podrà tornar-ho a intentar.

Si l’usuari rebutja el permís, es mostra un quadre de diàleg per pantalla que indica que el permís no s’ha atorgat correctament, amb el botó “Dar permiso”. Si l’usuari el clica, se’l redirigeix a la configuració del telèfon, concretament als detalls de l’aplicació per a que pugui atorgar el permís des d’allí, ja que un cop l’ha rebutjat a l’aplicació no es pot mostrar el quadre de diàleg per a que l’accepti de nou.

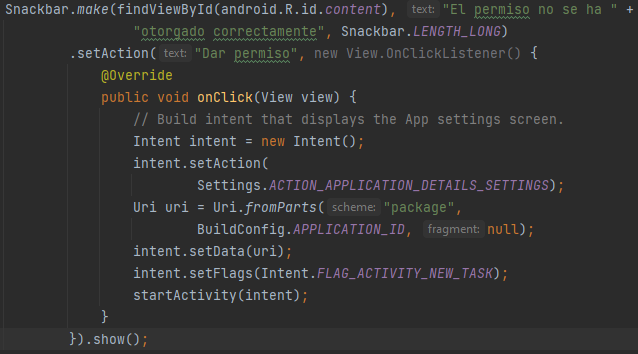


Figura: 45. Funció onRequestPermissionsResult() de l'activitat MainActivity

Per obrir la configuració del dispositiu amb la ruta desitjada s’utilitza un Intent amb l’acció:

*Settings.ACTION\_APPLICATION\_DETAILS\_SETTINGS)*

I les dades que indiquen la URI desitjada:

*Uri uri = Uri.fromParts("package",BuildConfig.APPLICATION\_ID, null);*

*intent.setData(uri);*

INICIO

Pantalla inicial amb múltiples accions que s’utilitza per gestionar el funcionament de l’aplicació així com accedir al perfil i als events guardats.

El disseny és extens però no té molta complexitat: Es basa en un seguit de LinearLayouts amb el mateix pes, de forma que independentment del dispositiu sempre tindran la mateixa proporció respecte a la pantalla i seran iguals entre elles.



Figura: 46. Exemple d'una de les LinearLayouts mencionades

Cada una conta amb una ImageView per la icona esquerra, un TextView per al nom i una altra ImageView per a la fletxa a la dreta. A continuació es mostra el codi d’una de les LinearLayouts:



Figura: 47. Part de l'arxiu fragment\_home.xml que mostra el codi d'un dels components LinearLayout

Aquestes Layouts estàn dividides en 3 grups: Conta, gestionar events i ajuda, sent 6 Layouts en total. A part hi ha una setena LinearLayout a la part superior de la pantalla, una mica més gran que les demés, que mostra la foto de perfil de l’usuari, el seu nom, un missatge recordatori si l’usuari no ha confirmat el seu correu i una icona que permet tancar la conta.

Tot i això, de les 6 accions de les LinearLayouts mencionades actualment només se n’han realitzat 2: “configuración” i “eventos guardados”.

La classe principal d’aquesta pantalla és diu HomeFragment.

La primera funció a explicar és “onCreateView()”: aquesta s’ocupa de donar valors a diferents variables, registrar Listeners per detectar les interaccions de l’usuari amb els elements de la UI i posar el nom i foto de l’usuari en pantalla.

El primer que realitza es inicialitzar variables: es fa el binding (s’explica més tard, copiar-ho i posar-ho aquí o al primer), s’obtenen referències a les LinearLayouts amb les que s’interactuarà i l’usuari que ha iniciat sessió registrat a Firebase.

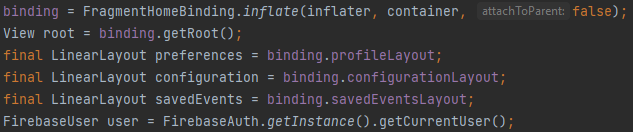


Figura: 48. Part de la funció onCreateView() del Fragment HomeFragment

S’utilitza l’usuari obtingut per comprovar si ha verificat el correu electrònic, i si no és així es mostra un text a la part superior de la pantalla indicant-ho. Per comprovar si ha verificat el correu electrònic s’utilitza el mètode

*user.isEmailVerified()*

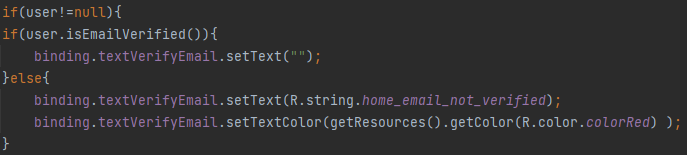


Figura: 49. Part de la funció onCreateView() del Fragment HomeFragment

L’usuari obtingut també s’utilitza per obtenir i mostrar el seu nom i la seva foto de perfil:

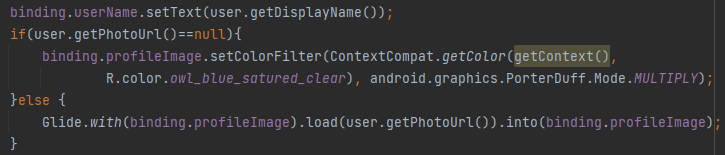


Figura: 50. Part de la funció onCreateView() del Fragment HomeFragment

Per obtenir i mostar el nom en un TextView anomenat “userName” s’utilitza el mètode

*binding.userName.setText(user.getDisplayName());*

Per mostrar la foto de perfil, primer es comprova si l’usuari disposa d’una amb el mètode.

*if(user.getPhotoUrl()==null) { ... }*

De no ser així es mostra una icona per defecte i se li canvia el color amb el mètode:

*binding.profileImage.setColorFilter(ContextCompat.getColor(getContext(),*

*R.color.owl\_blue\_satured\_clear), android.graphics.PorterDuff.Mode.MULTIPLY);*

Si l’usuari sí que conta amb una foto de perfil, aquesta es carrega amb la llibreria Glide:

*Glide.with(binding.profileImage).load(user.getPhotoUrl()).into(binding.profileImage);*

Finalment, aquesta funció conta amb 3 Listeners per a les dos accions de les LinearLayouts actualment funcionals (configuració i events guardats) i per al botó de tancar sessió.

Els Listeners de les LinearLayouts detecten quan l’usuari fa clic en elles i criden a la funció “openLayout()”, passant-li com a paràmetre la classe que es vol obrir:

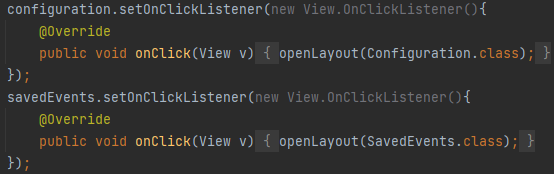


Figura: 51. Part de la funció onCreateView() del Fragment HomeFragment

La classe openLayout() simplement llença un Intent amb la classe indicada per paràmetre:

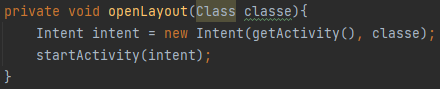


Figura: 52. Funció openLayout() del Fragment HomeFragment

L’últim Listener detecta quan es clica el botó de tancar sessió i crida a la funció logoutUser()

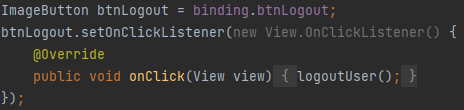


Figura: 53. Part de la funció onCreateView() del Fragment HomeFragment

Aquesta tanca la sessió actual i redirigeix a l’usuari a la pantalla d’inici de sessió:

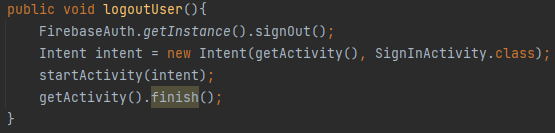


Figura: 54. Funció logoutUser() del Fragment HomeFragment

Per tancar la sessió s’utilitza el mètode:

*FirebaseAuth.getInstance().signOut();*

L’última funció d’aquesta classe és onResume(), que simplement comprova si l’usuari ja ha verificat el correu electrònic i elimina el missatge recordatori de ser el cas:

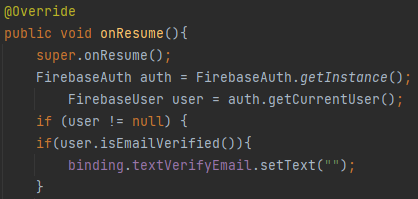


Figura: 55. Funció onResume() de l'activitat HomeFragment

La següent classe a explicar es diu “Configuration”. Actualment només conta amb un text interactuable que permet revocar l’accés al compte de l’usuari.

En la funció onCreate() s’inicialitzen les variables necessàries i es subratlla el text:

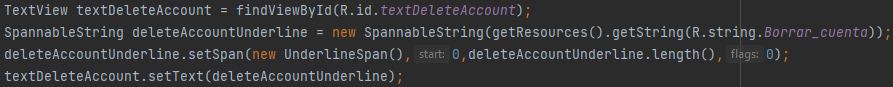


Figura: 56. Part de la funció onCreate() de l'activitat Configuration

Creant un nou objecte “SpannableString” i utilitzant el mètode:

*deleteAccountUnderline.setSpan(new UnderlineSpan(),0,deleteAccountUnderline.length(),0);*

Aquesta funció també registra el Listener per detectar quan es fa clic al text, i aquest crida a la funció “revokeAcces ()”.

La funció “revokeAcces ()” revoca l’accés de l’usuari al compte i tanca la sessió:



Figura: 57. Funció revokeAcces() de l'activitat Configuration

Per fer-ho, primer tanca la sessió actual per evitar problemes amb el mètode

*FirebaseAuth.signOut();*

I després necessita un objecte “GoogleSignInClient” que a la seva vegada requereix un objecte “GoogleSignInOptions”. Els dos s’obtenen amb els mateixos mètodes que a la pantalla SignInActivity.

Finalment es revoca l’accés mitjançant el mètode

*GoogleSignInClient.revokeAccess()*

I un cop complert es crida a la funció signOut(), que s’ocupa de redirigir a l’usuari a la pantalla d’inici de sessió:

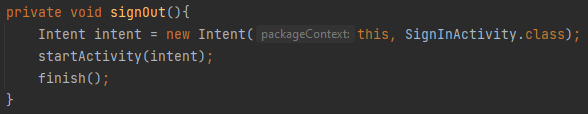


Figura: 58. Funció signOut() de l'activitat Configuration

Per acabar amb la pantalla “Inicio” queda per explicar la classe “SavedEvents”. Aquesta es crida quan es clica la LinearLayout amb el text “eventos guardados” i s’ocupa de mostrar tots els events que s’han guardat fent clic a la icona del cor (“favourites”).

El disseny de la pantalla només conta amb una RecyclerView que s’encarregarà de mostrar de manera ordenada tots els events que l’usuari hagi guardat. Una RecyclerView facilita que es mostrin de manera eficient grans conjunts de dades, però el seu funcionament s’explicarà en detall més endavant.

La funció principal de la classe SavedEvents és onCreate(): Aquesta s’ocupa de donar els valors inicials a les variables, de recuperar els events guardats i iniciar el procés per mostrar-los per pantalla.

Primerament es fa el binding i es crea la instància de RecyclerView i del seu adaptador, que en aquest cas és una classe anomenada “EventAdapter” que també s’explicarà més endavant. Seguidament es vinculen els components necessaris a la RecyclerView (l’adaptador i un LayoutManager).

Per guardar i recuperar valors s’utilitza la interfície “SharedPreferences”. Aquesta ens permet accedir i modificar les dades de preferència (les més importants), retornades per “Context.getSharedPreferences(String, int)”. Per a qualsevol conjunt concret de preferències, hi ha una única instància d'aquesta classe que comparteixen tots els clients. Les modificacions a les preferències han de passar per un objecte Editor per garantir que els valors de les preferències es mantinguin en un estat coherent i controlar quan es comprometen en l'emmagatzematge.

Revisar: <https://developer.android.com/reference/android/content/SharedPreferences>

Obtenim les mencionades instàncies a SharedPreferences i Editor amb els mètodes:

*sharedPref = getSharedPreferences("com.example.tfg\_application.saved\_events", Context.MODE\_PRIVATE);*

*editor = sharedPref.edit();*

Com es pot veure a continuació:

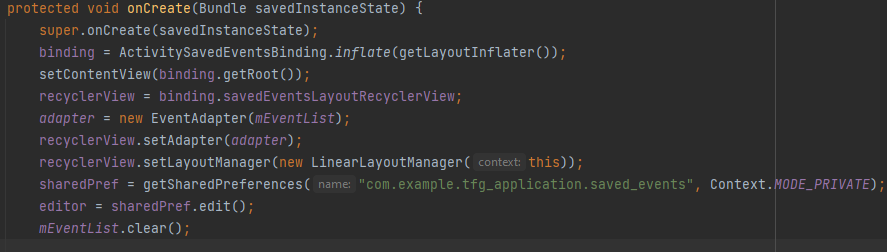


Figura: 59. Part de la funció onCreate() de l'activitat SavedEvents

El següent pas és obtenir els events guardats de la instància de SharedPreferences obtinguda. Per fer-ho s’obtenen primer tots els elements guardats en format Map amb el mètode:

*Map<String, ?> eventos = sharedPref.getAll()*

Però la interfície SharedPreferences només permet guardar dades de certs tipus. Els events es guarden en una instància de la classe “Event” que s’explicarà posteriorment, per tant prèviament s’ha transformat aquest objecte en una String. Per fer-ho s’ha utilitzat la llibreria Gson, que permet transformar qualsevol objecte en un text en format JSON i recuperar l’objecte després amb aquest text. Per tant ara cal transformar cada String obtinguda de les SharedPreferences en un objecte de tipus “Event”.

Per fer-ho, s’itera entre totes les Strings obtingudes amb un bucle “for” i s’utilitza la llibreria Gson per a recuperar els objectes Events:

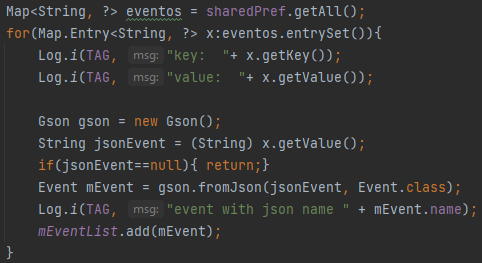


Figura: 60. Part de la funció onCreate() de l'activitat SavedEvents

Cada element recuperat de les SharedPreferences té una clau i un valor. La clau és l’id de l’event, i el valor és tota la informació d’aquest. En el bucle s’anomena als elements “x” i el seu valor s’obté amb el mètode:

*x.getValue();*

El mètode utilitzat per transformar les “String” a “Event” és:

*Event mEvent = gson.fromJson(jsonEvent, Event.class);*

I finalment s’afegeixen tots els elements en el format desitjat a una llista.

Aquesta llista està vinculada a l’editor de la RecyclerView, encarregada de mostrar tots els events per pantalla, i l’editor es crida a continuació amb el mètode:

*adapter.notifyDataSetChanged();*

Però tot el relacionat amb la RecyclerView s’explica més endavant per seguir amb el progrés i fer-ho més entenedor.

### EVENTOS

La pantalla anomenada “Eventos” té un disseny simple: Conta amb un buscador i 3 selectors.

Com ja s’ha mencionat anteriorment, la funció d’aquesta pantalla es permetre a l’usuari buscar nous events.

Primerament s’explicarà com funciona el sistema de cerca d’events per a que el codi resulti més entenedor.

Els events els cerca una classe a part anomenada EventsRequester mitjançant les dades aportades anteriorment per la classe principal EventsFragment.

Les dades que es poden aportar són:

Location: localització actual de l’usuari. Es passa quan es volen ordenar els resultats per proximitat. S’explicarà com s’obté la localització de l’usuari més endavant.

Radius: radi en km dins del qual es volen buscar events.

Sorting: mètode seleccionat per ordenar els resultats. Depenent de quin s’esculli els ordenarà per popularitat, per proximitat o per la data més pròxima a l’actual.

Procedence: indica quina classe ha fet la petició per buscar events. Pot ser “EventsFragment” o “MapFragment”

TextFilter: el text que l’usuari ha escrit en el buscador d’events. Els resultats obtinguts hauran de tenir en una part del seu nom tot el text indicat.

City: ciutat en la que es realitzarà la cerca d’events.



Figura: 61. Dades que es poden aportar per la cerca d'events

Un cop aportades les dades es crida al mètode getEvent() de la classe anteriorment mencionada EventsRequester i es crea la consulta a la API de Ticketmaster per obtenir els events.

Per realitzar la consulta es necessita generar una URL específica on es detallen les preferències escollides amb les dades que s’han aportat. Gran part de la informació necessària per a fer-ho està a la pàgina oficial de la API de Ticketmaster anomenada “Discovery API”:

Revisar:

<https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/>

Totes les consultes comencen amb l’estructura següent:

* https://app.ticketmaster.com/discovery/v2/

I posteriorment es van encadenant nous paràmetres que permeten especificar la cerca. El primer que s’ha afegit és: “/events.json?” . Aquest paràmetre indica que es busquen events (ja que també es poden obtenir altres resultats com artistes, suggeriments, vendes, classificacions...) i indica també que es vol el resultat en format JSON

Revisar: explicar tema format JSON

Per tant l’estructura de la consulta passa a tenir aquest format:

* https://app.ticketmaster.com/discovery/v2/events.json?

A partir d’aquí, cada paràmetre que s’afegeix va precedit d’un signe “&” (menys el primer) i té l’estructura nom\_paràmetre=valor\_paràmetre.

El següent paràmetre que s’ha afegit és: “apikey”. Aquest paràmetre és necessari ja que indica la nostra clau privada de la API, sense la qual no funcionaria la consulta. Per protegir aquesta clau el seu valor està guardat al fitxer local.properties, ja que aquest fitxer no serà compartit al sistema de control de versions i, per tant, no serà tant senzill sostreure-la.

Per obtenir la clau de la API s’ha escrit el següent codi a l’arxiu build.gradle:



Figura: 62. Part de l’arxiu build.gradle referent a l’emmagatzematge de claus d’API

Aquest codi llegeix la clau de l’arxiu local.properties i l’escriu en el fitxer BuildConfig, que és un fitxer que es genera automàticament quan es construeix l’aplicació a partir del codi.

L’estructura de la URL després d’aquests passos és la següent:

* https://app.ticketmaster.com/discovery/v2/events.json?apikey="+BuildConfig.ticketmaster\_api



Figura: 63. Estructura inicial de la URL utilitzada per obtenir events

Posteriorment, depenent de les dades que la classe EventRequester hagi obtingut, afegirà o no els següents paràmetres amb els valors indicats:

&keyword= 🡪 text per filtrar els events, passat com “TextFilter”

&geoPoint= 🡪 localització espacial en format GeoHash. Únicament s’afegeix al ordenar per proximitat

GeoHash: Aquest format està constituït per una curta seqüencia de números i lletres (màxim 12), la qual és més precisa quan més llarga és, i a la que se li poden eliminar caràcters del final per reduir el seu pes però perdre precisió gradualment.

Com la localització obtinguda a l’aplicació només conta amb els valors de latitud i longitud, cal transformar-la al format geoHash. Per fer-ho s’implementa la llibreria ch.hsr:geohash:

*implementation 'ch.hsr:geohash:1.4.0'*

Després s’importa el component necessary amb el codi:

*import ch.hsr.geohash.GeoHash;*

I finalment es fa la transformació amb el codi:

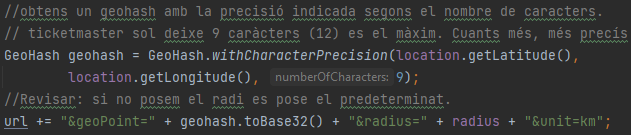


Figura: 64. Codi per transformar una Location a un GeoHash, en la classe EventsRequester

En el codi es pot apreciar que també s’indiquen els paràmetres “radius” i “unit”, que indiquen el radi al voltant del “geoPoint” sobre el que es realitzarà la cerca i la unitat de mesura per aquest radi (en aquest projecte sempre és el kilòmetre)

&city= 🡪 ciutat en es realitzarà la cerca

&countryCode= 🡪 codi del país on es realitzarà la cerca (ex: Espanya = ES)

&sort= 🡪 ordre en el que es mostraran els events trobats. Les opcions possibles són:



Figura: 65. Opcions d’ordenació dels resultats obtinguts en la consulta <https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/>

De les quals s’ha utilitzat: “relevance,desc” per popularitat, “distance,asc” per proximitat i “date,asc” per la data més pròxima.

Un cop s’ha generat tota la URL desitjada per a la consulta, cal realitzar-la. Per fer-ho es fa ús de la llibreria “Volley”:

*implementation 'com.android.volley:volley:1.2.1'*

Volley és una biblioteca HTTP que facilita i agilitza l’ús de xarxes en aplicacions Android. Té moltes utilitats de les quals s’utilitzarà el seu sistema de sol·licituds amb els següents mètodes:



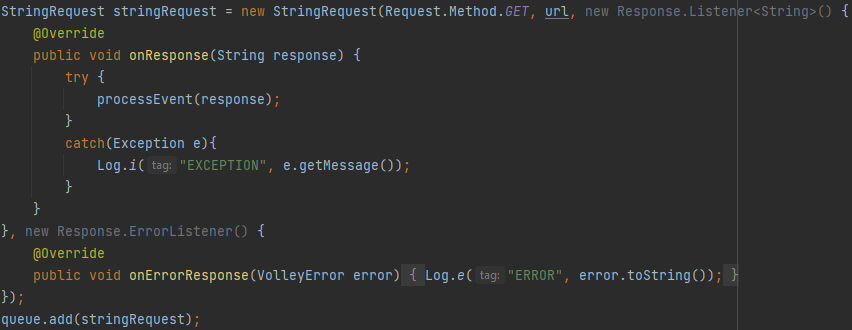


Figura: 66. Mètodes de la llibreria Volley

Un cop realitzada la consulta es rep la resposta com una String que s’ha anomenat “response”, tot i que com s’ha mencionat, arriba en format JSON. El format de la resposta és el següent:



Figura: 67. Estructura de la resposta rebuda per part de la API de Ticketmaster

Tot i que depenent de l’event pot tenir més paràmetres o menys.

El següent pas es processar aquesta String per a extreure tots els events que té i totes les dades que interessen d’aquests. Això es realitza al mètode “processEvent()”.

ProcessEvent(): Aquest mètode extreu els diferents events i els guarda utilitzant la classe “Event”, la qual s’explicarà primer.

Classe Event: Classe on s’emmagatzema tota la informació relacionada a un event.

La informació en qüestió emmagatzemada és la següent:

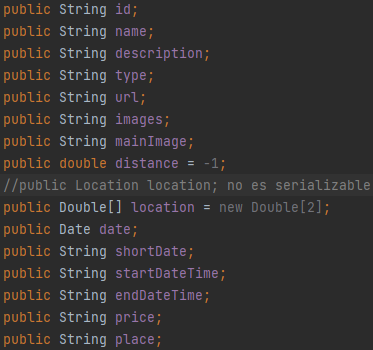


Figura: 68. Variables que guarden la Informació de la classe Event

Aquesta classe conta també amb diversos mètodes i una classe niada per assistir a la creació de l’event.

Com ja s’ha mencionat, cada event retornat per la consulta pot tenir molts possibles paràmetres (pot ser que un en tingui 7 i un altre 20), per tant a l’hora de crear l’Event cal fer-ho pas per pas, tenint en compta que cada un pot contenir uns paràmetres totalment diferents a l’anterior. Tenint en compte tot això s’ha optat per utilitzar el “Builder patern” o patró de disseny de constructor.

El patró de disseny de constructor té com a objectiu separar la construcció d'un objecte complex de la seva representació de manera que el mateix procés de construcció pugui crear diferents representacions. S'utilitza per construir un objecte complex pas a pas i el pas final retornarà l'objecte. El procés de construcció d'un objecte ha de ser genèric perquè es pugui utilitzar per crear diferents representacions d'un mateix objecte.

Revisar: link

<https://www.geeksforgeeks.org/builder-design-pattern/>

Aquesta classe niuada a Event s’anomena EventBuilder i guarda la mateixa informació que la classe Event:

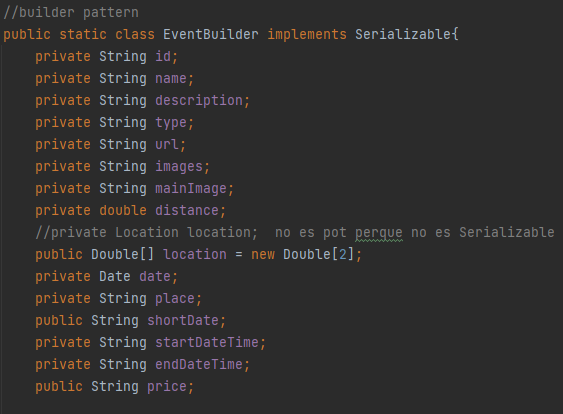


Figura: 69. Variables que guarden la Informació de la classe EventBuilder

Però té múltiples mètodes constructors que permeten la creació de l’objecte EventBuilder pas a pas:

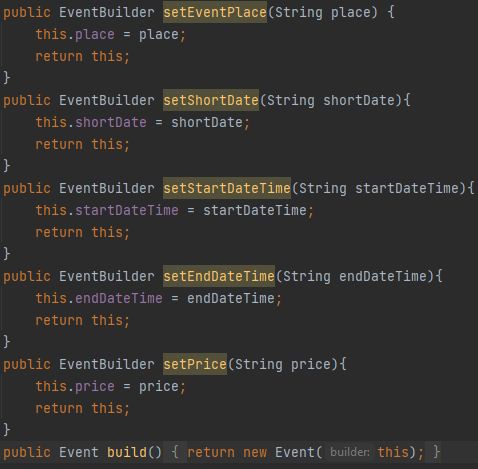


Figura: 70. Mètodes de la classe EventBuilder

Finalment es crea l’event passant-li per paràmetre l’objecte “EventBuilder”:

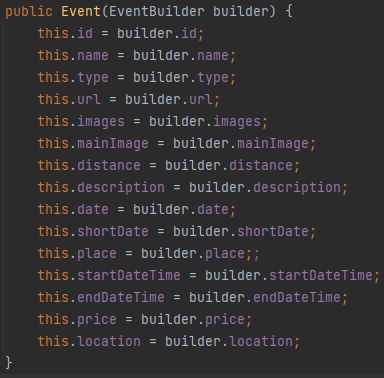


Figura: 71. Mètode constructor per crear l’objecte Event, en la classe Event

Un cop explicada la classe “Event”, cal retornar a l’explicació del mètode processEvent(String response), de la classe EventsRequester.

En aquest mètode es transforma la String obtinguda com a resposta de la consulta en un objecte de tipus JSON (una JSONArray) aprofitant que la resposta obtinguda té aquesta sintaxis:

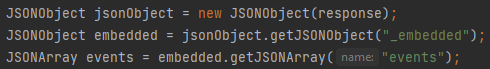


Figura: 72. Mètodes utilitzats per transformar una String a una JSONArray, en la classe EventsRequester

Un cop es té, s’utilitzen funcions proporcionades per la llibreria JSON, com ara les següents:



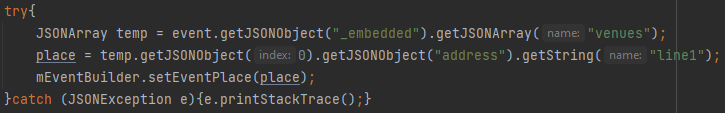


Figura: 73. Exemples de mètodes utilitzats de la llibreria JSON

Per navegar a través de l’extens document de text que és cada resposta.

Cal mencionar també que és necessari rodejar cada intent d’obtenir un paràmetre amb un try-catch, ja que aquest paràmetre pot no estar present en un event i aquest sistema ens evita que el codi falli si això passa.

Una altra funció d’aquest mètode també és donar el format correcte a cada paràmetre obtingut, ja que molts no venen amb el disseny desitjat:



Figura: 74. Part de la funció processEvent(), de la classe EventsRequester

I, Finalment, va passant un per un els valors de tots els paràmetres obtinguts al EventBuilder com es veu a l’última línia de la imatge anterior, crea els objectes Event, els guarda en una llista i els envia com a paràmetre a la funció “callEventAdapter(Event[] mAllEvents)”.

Després d’això es fan unes quantes comprovacions de seguretat i s’envien els objectes Event de nou al fragment principal EventsFragment, qui s’ocupa de fer que s’iniciï el procés per mostrar aquests events per pantalla, mitjançant el mètode “adapter.notifyDataSetChanged();”

Abans de seguir amb el procés, cal explicar uns quants conceptes necessaris per entendre el que ve a continuació. Quan es fa la consulta no es saben quants events retornarà, però és bastant segur que més dels que caben en la pantalla (que son uns 5). En aquests casos es disposa d’un widget anomenat RecyclerView.

RecyclerView facilita que es mostrin de manera eficient grans conjunts de dades. Com el seu propi nom indica, RecyclerView recicla les vistes dels elements en pantalla, de forma que quan un element es desplaça fora d’aquesta no destrueix la seva vista, sinó que la reutilitza per als nous elements que han entrat a la pantalla. Això millora en gran mesura el rendiment i la capacitat de resposta de l’aplicació, i redueix el seu consum d’energia.

Com RecyclerView és el ViewGroup que conté les vistes corresponents a les nostres dades, és una vista en si mateixa, i per tant s’agrega directament a layout com qualsevol altre element de la IU.

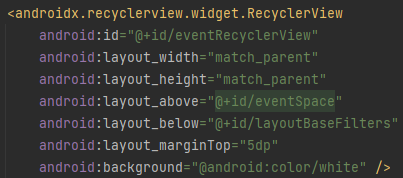


Figura: 75. Codi que defineix l'element RecyclerView, en l'arxiu fragment\_events.xml

Ara es té definida la vista, però falta definir l’objecte contenidor de vistes en el que anirà cada event. Aquest s’anomena RecyclerView.ViewHolder. Quan se’l crea, aquest contenidor no té dades associades i després de crear-lo, la RecyclerView el vincula a les seves dades.

La RecyclerView sol·licita les vistes d’aquest objecte i les vincula a les seves dades mitjançant crides als mètodes en l'adaptador. Per definir l'adaptador, s’estén el RecyclerView.Adapter.

Finalment, l’administrador de disseny organitza els elements individuals de la llista. En aquest cas se n’utilitzarà un proporcionat per la llibreria RecyclerView. Tots els administradors de disseny es basen en la classe abstracta LayoutManager de la llibreria RecyclerView.

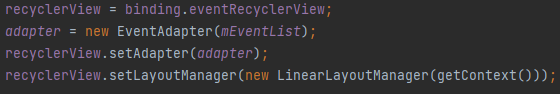


Figura: 76. Codi en que s’inicialitzen els components de la RecyclerView, en la classe EventsFragment

Revisar: referencia a link.

<https://developer.android.com/guide/topics/ui/layout/recyclerview?hl=es-419>

En resum, RecyclerView.ViewHolder són cada un dels contenidors que contindran els events, als que se’ls han de definir els valors, i RecyclerView.Adapter és qui crea els objectes ViewHolder i associa les dades a aquests.

Un cop explicat, es pot seguir amb el procés per mostrar els events a l’usuari:

La classe EventsFragment utilitza el mètode adapter.notifyDataSetChanged() per fer la crida a l’adaptador i començar el procés explicat anteriorment. En aquest cas el l’ adaptador utilitzat és una classe anomenada EventAdapter, escrita amb llenguatge kotlin.

EventAdapter

Per dur a terme la seva tasca com adaptador, fa ús de 3 funcions:

La primera funció és *onCreateViewHolder(parent: ViewGroup, viewType: Int)*.

Es crida quan s’ha de crear un nou ViewHolder i s’ocupa d’afegir les vistes (Views) al grup de vistes (ViewGroup) mitjançant el mètode inflate.

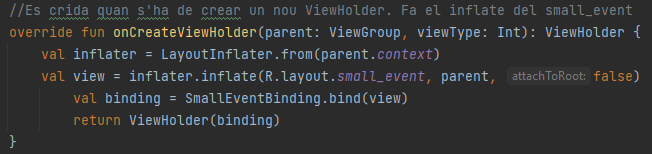
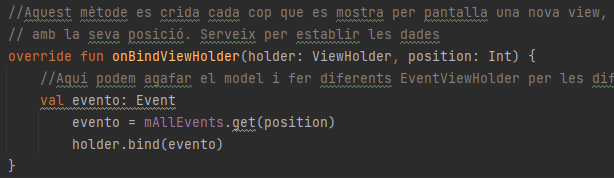


Figura: 77. Funció onCreateViewHolder() de la classe EventAdapter

Com es pot veure en la imatge superior, les vistes que s’afegeixen pertanyen a la layout anomenada “small\_event”.

La segona funció és *onBindViewHolder(holder: ViewHolder, position: Int)*.

Es crida cada cop que es mostra per pantalla una nova vista i ens indica la posició d’aquesta. Serveix per establir les dades que utilitzarem en aquesta vista.



Dins de la classe niuada *ViewHolder(private val binding: SmallEventBinding) : RecyclerView.ViewHolder(binding.root){*

Tenim l’última funció: bind(item: Event).

Es crida des de la funció onBindViewHolder() i s’ocupa de donar els valors corresponents a cada element de la vista utilitzant les dades anteriorment establertes:

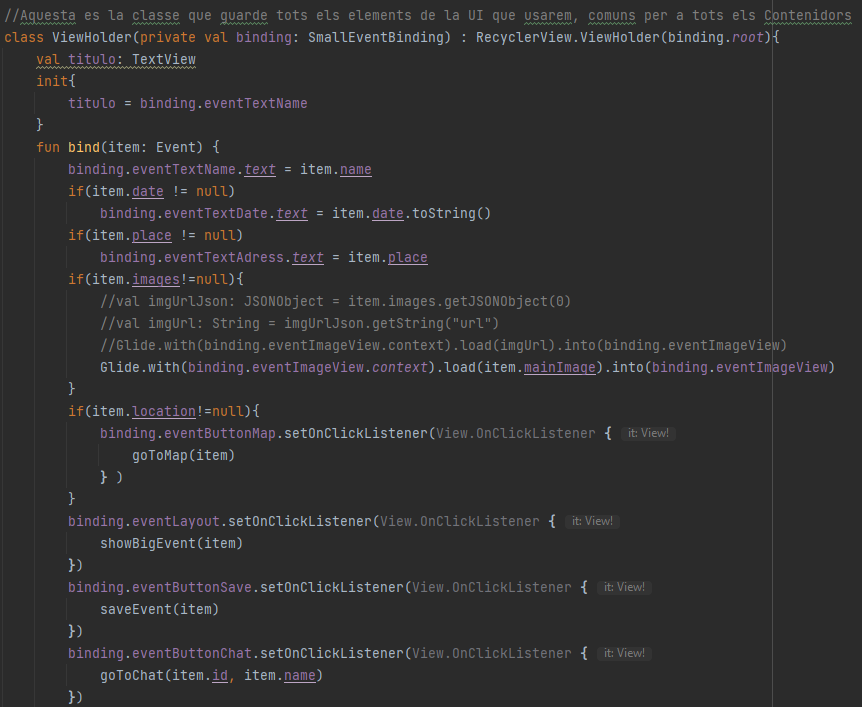


Figura: 78. Funció bind(), en la classe ViewHolder que es troba niuada en la classe EventAdapter

Cal fer menció especial a la llibreria Glide, que permet carregar una imatge mitjançant la seva URL de manera ràpida i eficient, utilitzant una sola línia de codi:

*Glide.with(binding.eventImageView.context).load(item.mainImage).into(binding.eventImageView)*

Dins de la classe ViewHolder s’han creat 4 funcions més. 3 d’elles s’encarreguen de donar funcionalitat al clicar les icones presents en la representació de cada event:

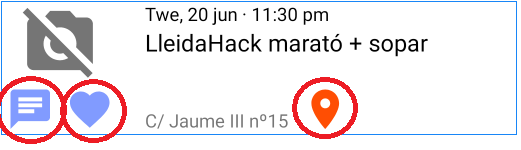


Figura: 79. Representació d'exemple d'un event, amb les icones encerclades

1a icona per l’esquerra: crida a la funció “goToChat()”. Aquesta obra la pantalla del xat, passant com a paràmetres l’id de l’event i el seu nom:

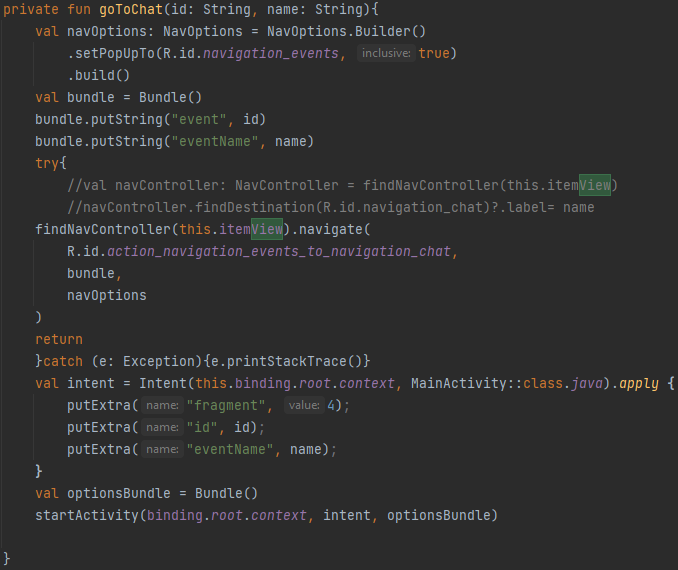


Figura: 80. Funció goToChat(), de la classe ViewHolder que es troba niuada en la classe EventAdapter

2a icona: guarda l’event. S’explicarà el seu funcionament amb més detall més endavant, ja que requereix de l’explicació d’altres conceptes previs.

3a icona: crida a la funció “goToMap()”. Aquesta obra la pantalla del mapa, passant l’objecte “Event” com a paràmetre.

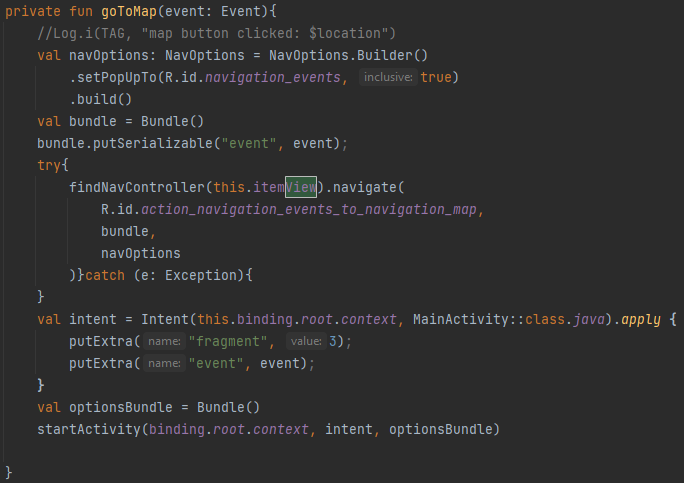


Figura: 81. Funció goToMap(), de la classe ViewHolder que es troba niuada en la classe EventAdapter

I finalment està l’última funció, que es crida quan es prem a qualsevol altre lloc de la vista de l’event. Aquesta obra una nova activitat encarregada de mostrar un únic event utilitzant tota la pantalla, mostrant més detalls d’aquest:

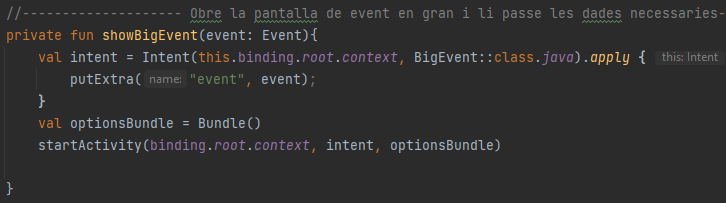


Figura: 82. Funció showBigEvent(), de la classe ViewHolder que es troba niuada en la classe EventAdapter

L’activitat “BigEvent” és la que s’encarrega de destinar tota la pantalla a un únic event per a mostrar més informació sobre aquest. En aquesta pantalla es pot veure la data i hora amb més detall, el rang de preus, més imatges i es tenen les funcionalitats de compartir, guardar l’event a l’aplicació, guardar-lo al calendari i visitar la pàgina per comprar entrades.

El disseny de la nova activitat “BigEvent” consta de la imatge principal de l’event en la part superior, ocupant tot l’ample de la pantalla, amb el títol de l’event a sota.

Després hi ha 3 files amb icones: La primera és d’un calendari:  i va seguida de la data i hora d’inici i final de l’event. La segona és la icona de localització:  i va seguida de la direcció en que es realitza l’event. La tercera és una icona d’un tiquet:  i va seguida del rang de preus que pot valdre l’event.

Finalment hi ha una línia divisòria i una galeria de fotos que mostra inicialment la imatge secundària de l’event. (cal recalcar que Ticketmaster va realitzar certs canvis en la seva API i actualment la gran majoria d’imatges que es poden obtenir amb la versió gratuïta són la mateixa en diferents formats, per això en molts events es mostra la mateixa imatge repetida).

El fitxer de disseny està basat en una RelativerLayout i només conta amb vistes ImageView i TextView.



Revisar: (veure com quede posar això, sinó s’elimine)

En quan al codi, es començarà per explicar la funció onCreate(), ja que és la que assigna les dades de l’event a les vistes i registra els Listeners per quan es vulgui interactuar amb l’aplicació.

El primer que es fa en la funció és obtenir els valors de les variables que necessitarem, sent aquestes el ViewBinding, un objecte SharedPreferences com en la pantalla “saved\_events” explicada anteriorment, juntament amb el seu editor i un Bundle que contindrà les dades que s’han enviat des de l’activitat anterior (l’objecte Event).

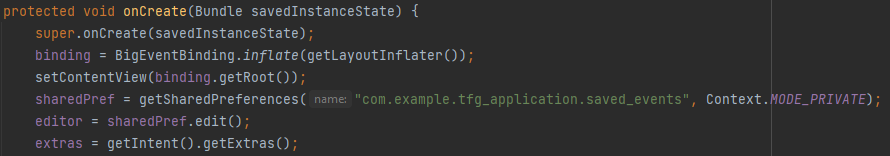


Figura: 83. Part de la funció onCreate(), de la classe BigEvent

El següent és començar a vincular les dades amb les vistes:

Primerament s’obté l’objecte Event que s’hauria d’haver passat des de la pantalla anterior amb el mètode

*event =(Event) extras.getSerializable("event");* (on “extras” és el Bundle)

i després s’obtenen d’ell totes les dades necessàries i es vinculen.

Les dades s’obtenen simplement fent referència a la variable de la classe Event que les guarda, per exemple el preu s’obté amb el mètode Event.price, i es vinculen fàcilment gràcies al binding, per exemple el títol es posa amb el mètode *binding.titleTextView.setText(event.name);*

La informació sobre la data consta de la concatenació de la data, l’hora d’inici de l’event i l’hora d’acabament:

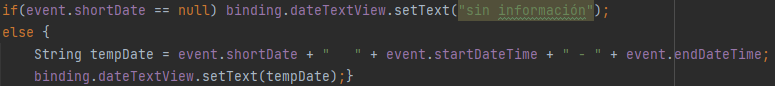


Figura: 84. Part de la funció onCreate(), de la classe BigEvent

Per obtenir les imatges es crida a la funció “loadImages()”, que s’encarrega d’obtenir-les, buscar les apropiades i carregar-les en pantalla:

Primerament s’obté l’objecte JSONArray que guarda totes les imatges de l’event amb el mètode

*“JSONArray jsonArrayGallery = new JSONArray(event.images);”.*

Després es filtra cada imatge que contingui per trobar-ni una amb la qualitat adient, ja que poden (i solen) estar desordenades i algunes són per a miniatures i tenen molt poca resolució.

Per fer-ho es transforma cada imatge individual en un JSONObject i es busquen els paràmetres “ratio” i “width” amb el mètode “JSONObject.getString()”. Es busca un ratio de 16\_9 o de 4\_3 i un width superior a 600 per a que la imatge tingui la forma i qualitat adequades.



Figura: 85. Part de la funció loadImages(), de la classe BigEvent

Després s’utilitza la llibreria Glide per carregar les imatges en les vistes corresponents:

*Glide.with(binding.topImageView).load(jsonArrayGallery.getJSONObject(galleryImageIndex1).getString("url")).into(binding.topImageView);*

Un cop ja s’han carregat totes les vistes s’estableixen els Listeners:

Els dos primers executen la mateixa funció i detecten quan l’usuari clica en la icona del calendari o en el text que l’acompanya. La funció en qüestió es diu “openCalendar()”

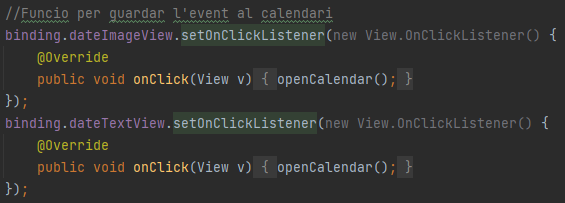


Figura: 86. Part de la funció onCreate(), de la classe BigEvent

“openCalendar()” s’ocupa d’obrir l’aplicació del calendari de google del dispositiu i crea un nou event en ell; establint el dia, hora, nom, ubicació i URL de l’event seleccionat per a que l’usuari pugui guardar-lo còmodament en el seu calendari en un únic clic:

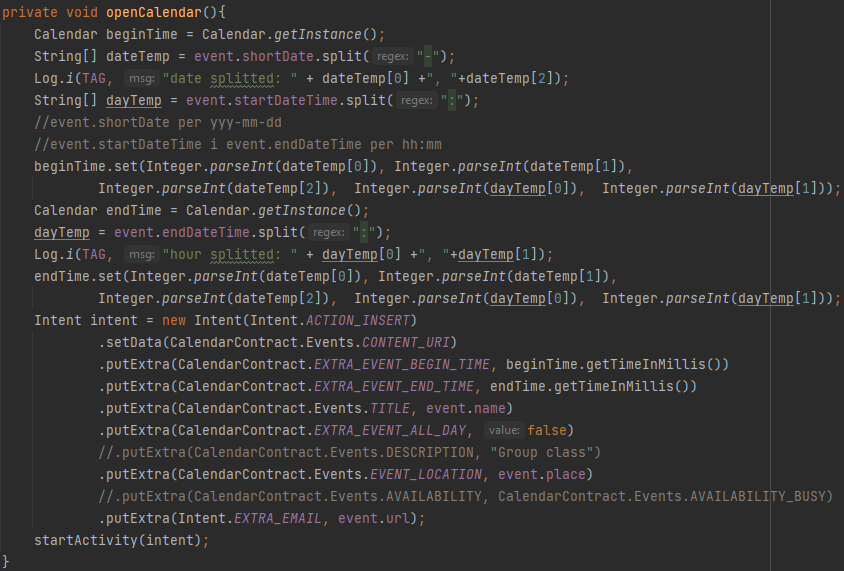


Figura: 87. Funció openCalendar(), de la classe BigEvent

Primerament s’obté un objecte tipus Calendar amb la funció

*Calendar.getInstance();*

Aquest objecte conté paràmetres per guardar un any, mes, dia, hora i minut; que s’estableixen amb la funció Calendar.set(year, month, date, hour, min). Per fer-ho s’extreuen aquests paràmetres individualment de les dates que té guardades cada event amb les funcions:

*String[] dateTemp = event.shortDate.split("-");* (format de la data yyyy-mm-dd)

*String[] dayTemp = event.startDateTime.split(":");* (format de la data hh:mm)

Que divideixen una String en varies separant-les segons el caràcter indicat “-” i “:”.

Es realitza el procés 2 cops, per obtenir el moment d’inici de l’event i el del final, i després s’obra l’aplicació del calendari mitjançant un Intent. Per a que l’Intent obri el calendari cal establir l’acció “insert”, que inserta un ítem buit en un contenidor determinat i la direcció del calendari amb el mètode setData():

*.setData(CalendarContract.Events.CONTENT\_URI)*

Després es donen valors a l’ítem que s’afegirà al calendari amb el mètode .putExtra(): s’afegeixen el moment d’inici i final de l’event calculats anteriorment:

*.putExtra(CalendarContract.EXTRA\_EVENT\_BEGIN\_TIME, beginTime.getTimeInMillis())*

Així com el títol i ubicació amb els paràmetres Events.TITLE i Events.EVENT\_LOCATION.

Finalment s’obra el calendari i l’usuari té l’opció de guardar o no l’event, així com modificar qualsevol dels paràmetres.

Els dos següents Listeners executen també un mateix codi i detecten quan l’usuari clica en la icona del tiquet o en el text que l’acompanya. El codi en qüestió és un Intent que obra una URL:

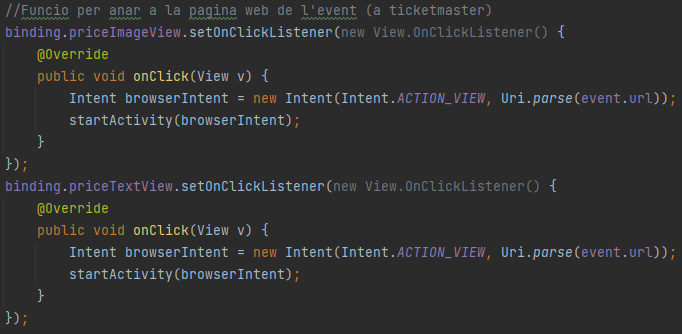


Figura: 88. Part de la funció onCreate(), de la classe BigEvent

L’Intent obra el navegador del dispositiu i dirigeix a l’usuari a la URL de la pàgina de Ticketmaster on es poden comprar les entrades per l’event.

Les següents funcions fan referència al menú superior o “toolbar” i s’utilitzen per interactuar amb ell o modificar-lo. Aquesta pantalla té un menú personalitzat propi anomenat “toolbar” que conté 3 ítems amb icones: una fletxa enrere (back), un cor (favourites) i una icona de compartir (share).



Figura: 89. Menú superior present en l’activitat BigEvent

La primera funció és onCreateOptionsMenu(). Aquesta es crida automàticament quan es crea el menú i s’utilitza per vincular el menú adient i editar la icona “favourites” en cas que es requereixi.

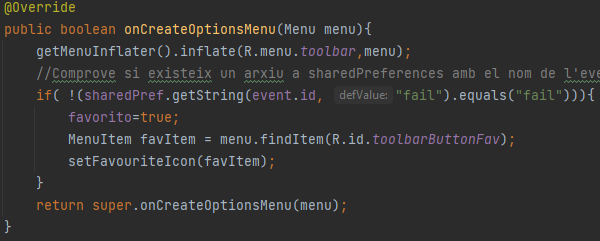


Figura: 90. Funció onCreateOptionsMenu(), de la classe BigEvent

Per vincular el menú s’utilitza el mètode

*getMenuInflater().inflate(R.menu.toolbar,menu);*

Que s’ocupa d’instanciar el fitxer de menú XML en l’objecte Menu.

Per escollir la icona “favourites” es comprova si l’event està guardat a les SharedPreferences i de ser així es canvia la icona de  a . Per comprovar si l’event es troba a SharedPreferences es fa ús del mètode:

*if( !(sharedPref.getString(event.id, "fail").equals("fail"))){...*

Si aquest retorna un valor vertader, vol dir que l’event es troba guardat i es marca en un booleà. Després s’utilitza l’objecte Menu que s’ha obtingut com a paràmetre en la funció per obtenir una referència a la icona “favourites” en un objecte MenuItem, i aquesta es passa a la funció “setFavouriteIcon()”.

La funció “setFavouriteIcon()” simplement revisa si l’event està marcat al booleà mencionat anteriorment com a guardat o no, i li canvia la icona mostrant un cor pintat o sense pintar respectivament.

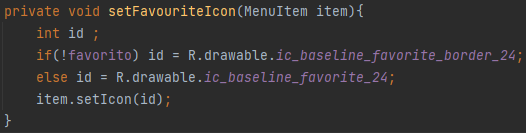


Figura: 91. Funció setFavouriteIcon(), de la classe BigEvent

Per canviar la icona es fa ús de mètode MenuItem.setIcon().

L’última funció d’aquesta classe és “onOptionsItemSelected(MenuItem item)”. Aquesta funció es crida quan l’usuari selecciona un dels ítems del menú, i ens indica quin ítem és en concret.

Si l’ítem és la fletxa enrere, simplement es crida a la funció finish() per tancar l’activitat actual i tornar a l’anterior.

Si l’ítem és la icona de compartir, s’obra un selector per escollir quin servei o aplicació es vol utilitzar per a compartir l’event i s’envia la URL d’aquest al destinatari escollit.

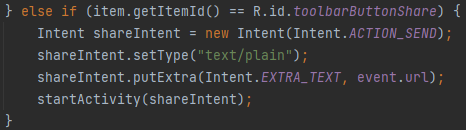


Figura: 92. Part de la funció onOptionsItemSelected(), de la classe BigEvent

Per fer-ho, s’utilitza un Intent amb l’acció “ACTION\_SEND”, que ens permet enviar informació. S’indica que només s’envia text amb el mètode:

*shareIntent.setType("text/plain");*

I s’afegeix com a text la URL de l’event amb el mètode:

*shareIntent.putExtra(Intent.EXTRA\_TEXT, event.url);*

Per una altra banda, si l’ítem és el cor, s’afegeix o elimina l’event de les SharedPreferences i s’actualitza la icona.

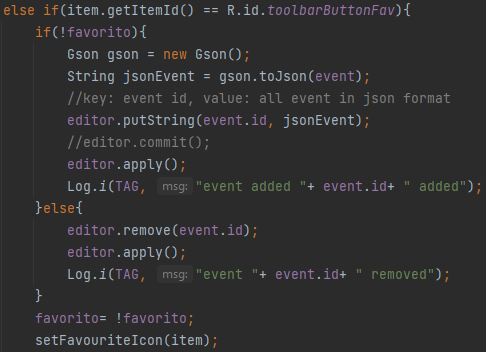


Figura: 93. Part de la funció onOptionsItemSelected(), de la classe BigEvent

Per guardar un objecte de tipus “Event” a les SharedPreferences es fa ús de la llibreria Gson.

*Gson gson = new Gson();*

Després es transforma l’objecte “Event” com a una String amb format intern JSON:

*String jsonEvent = gson.toJson(event);*

I finalment s’utilitza el SharedPreferences.Editor per penjar i guardar l’event:

*editor.putString(event.id, jsonEvent)” i “editor.apply();*

Per eliminar un objecte de les SharedPreferences és més senzill, només cal utilitzar els mètodes

*editor.remove(event.id);* i *editor.apply();*

Finalment s’inverteix el valor del booleà que registra si l’event està guardat i es crida a la ja mencionada funció setFavouriteIcon() per a que actualitzi la icona.

MAPA

Té el disseny més simple de totes: simplement els menús superior i inferior i el mapa.

En l’arxiu de disseny, el mapa es troba en un contenidor tipus androidx.fragment.app.FragmentContainerView :

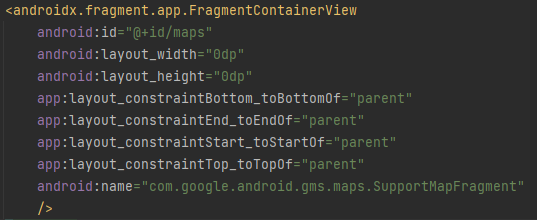


Figura: 94. Codi de l’element FragmentContainerView, en l’arxiu fragment\_map.xml

Aquest tipus de contenidor s’utilitza per a mostrar “fragments” dins d’un altre contenidor, en aquest cas, el mapa en sí.

Com ja s’ha mencionat anteriorment, aquesta pantalla s’ocupa de mostrar un mapa on l’usuari podrà veure la ubicació dels events, tot i que aquesta pantalla té dos funcions:

La primera és veure la ubicació d’un event seleccionat al mapa. Per fer-ho només cal prémer la icona d’ubicació:  d’un event en concret i l’aplicació es dirigirà automàticament al mapa, centrant-lo en la ubicació exacta de l’event i posant-li un marcador per a que es pugui reconèixer.

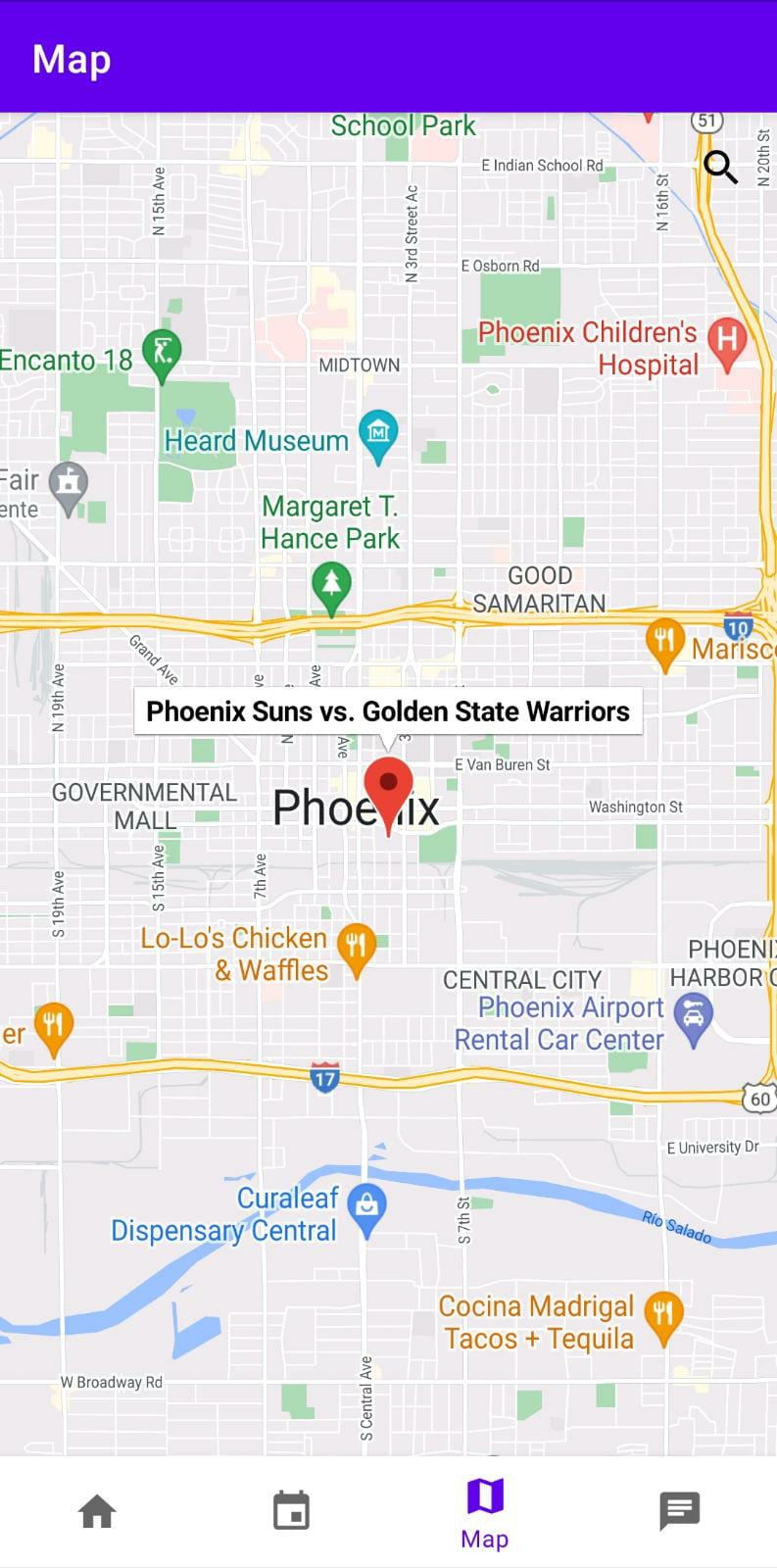


Figura: 95. Pantalla del mapa marcant la ubicació d’un event

La segona funció és trobar events propers a la posició desitjada mitjançant el mapa. Per fer-ho només cal prémer la icona superior dreta de cerca i automàticament es marcaran al mapa tots els events que estiguin a menys de 500km de la posició central del mapa.



Figura: 96. Pantalla del mapa mostrant els events trobats en un radi de 500km

Tot i això, ja majoria del codi de l’activitat que s’utilitza en les dos funcionalitats és compartit.

El primer a mencionar és que la classe MapFragment implementa la interfície OnMapReadyCallback, la qual només conté una funció:

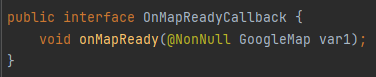


Figura: 97. Interfície OnMapReadyCallback

És una interfície “callback” o de devolució de trucada per quan el mapa estigui llest per ser utilitzat. Un cop s’ha establert una instància d’aquesta interfície en un objecte MapFragment o MapView, el mètode onMapReady(GoogleMap) s'activa quan el mapa està llest per utilitzar-se i proporciona una instància no nul·la de GoogleMap.

En la classe MapFragment, la primera funció important és onCreateView(), ja que a part d’inicialitzar algunes variables també estableix la instància de la interfície recentment mencionada:



Figura: 98. Part de la funció onCreateView(), de la classe MapFragment

Com es pot veure, primer s’obté l’objecte SupportMapFragment a partir del fragment actual i després s’utilitza el mètode getMapAsync(), que estableix l’objecte “callback” o de devolució de trucada que s'activarà quan la instància de GoogleMap estigui a punt per utilitzar-se.

Aquesta funció també conté el codi que inicia el procés de buscar events pròxims, ja que aquests es busquen quan l’usuari prem un botó de la interfície, per tant el “Listener” d’aquest botó ha d’estar en aquesta funció:

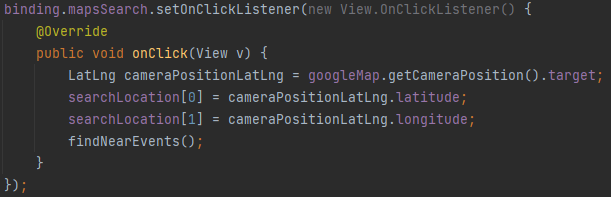


Figura: 99. Part de la funció onCreateView(), de la classe MapFragment

Primerament s’utilitza la funció GoogleMap.getCameraPosition().target per obtenir un objecte de tipus LatLng que conté les coordenades en les que està centrada la càmera del mapa, o sigui, el punt central del mapa que es mostra per pantalla.

Seguidament es guarden les coordenades i es crida a la funció “findNearEvents()”:

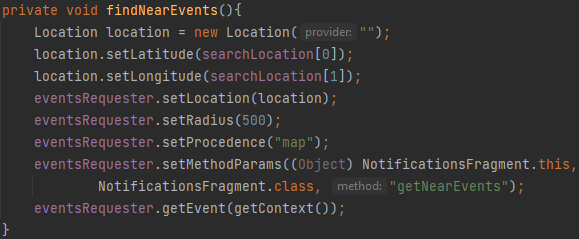


Figura: 100. Funció findNearEvents(), de la classe MapFragment

Aquesta s’ocupa de passar la informació necessària a la classe EventsRequester (anteriorment explicada) per a que busqui els events desitjats: li passa les coordenades prèviament guardades, el radi de cerca, una funció pròpia i la procedència de la petició. En la classe EventsRequester tot funciona exactament igual fins a l’últim punt, on s’utilitza la dada de procedència i la funció pròpia per retornar els events trobats.

Com la cerca d’events és asíncrona, el mètode utilitzat per retornar els events és la crida d’una funció prèviament especificada de la classe MapFragment des de la classe EventsRequester.

- Cal remarcar que un altre mètode més comú seria utilitzar el sistema WorkManager d’Android, però actualment no funciona correctament en algunes versions de Xiaomi (la marca del telèfon utilitzat durant el desenvolupament) i no es pot assegurar el seu funcionament.-

La funció mencionada que rep la resposta s’anomena “getNearEvents(List<Event> events)”, i s’ocupa de posar marcadors al mapa en les localitzacions exactes dels events trobats.

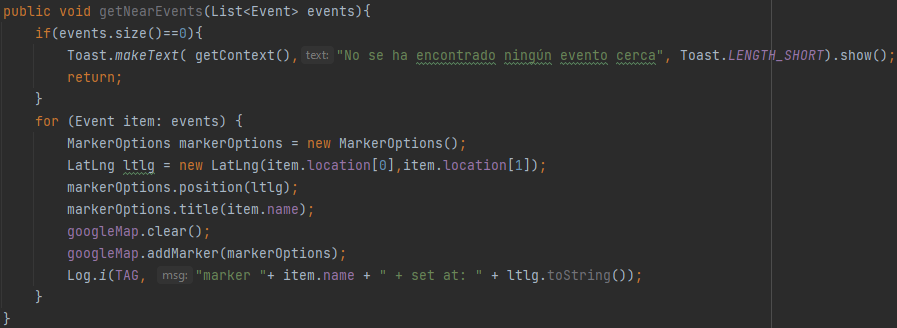


Figura: 101. Funció getNearEvents(), de la classe MapFragment

Per fer-ho, només cal tenir una instància de l’objecte GoogleMaps, que més tard s’explicarà com s’obté, i un objecte MarkerOptions amb la posició del marcador establerta. Es crea el MarkerOptions amb la funció:

*MarkerOptions markerOptions = new MarkerOptions();*

Se li estableix la posició guardada a l’event trobat amb el mètode:

*markerOptions.position(LatLng latlng);*

així com el títol d’aquest amb el mètode

*markerOptions.title(String name);*

S’esborren també els marcadors anterior per a que no s’acumulin massa

*googleMap.clear();*

I finalment s’afegeix el marcador al mapa:

*GoogleMap.addMarker(markerOptions);*

De manera que s’obté un marcador en la posició indicada que mostrarà el nom de l’event quan es cliqui en ell.

La següent funció a comentar és onCreate(), que simplement busca si la pantalla anterior li ha passat un objecte Event com a paràmetre, i si és així el guarda juntament amb la seva localització.

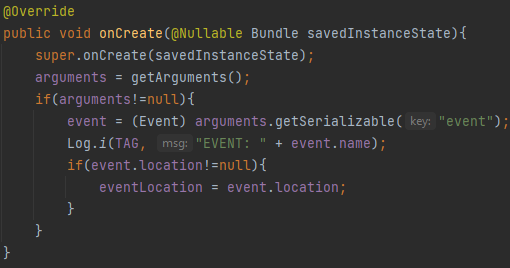


Figura: 102. Funció onCreate(), de la classe MapFragment

Les dades que es reben com a paràmetre des d’una altra activitat, s’obtenen amb la funció “getArguments()”.

Aquesta funció s’utilitza quan es crea un nou “Fragment” i retorna un objecte de tipus Bundle especificat anteriorment.

Per tal de passar entre classes un objecte de tipus “Event”, s’ha utilitzat la interfície “Serializable”, implementada en la classe “Event”. Com la classe “Event” la implementa, aquesta també es de tipus “Serializable” i per tant es té un mètode per obtindré l’event a partir del Bundle:

*event =(Event) arguments.getSerializable("event");*

On “arguments” és el Bundle obtingut i “(Event)” transforma el tipus d’objecte retornat de “Serializable” a “Event”.

Després d’aquesta s’arriba a la ja mencionada funció:

“onMapReady(@NonNull GoogleMap googleMap)”, que es crida automàticament quan el mapa s’ha carregat completament. Pertany a la interfície “OnMapReadyCallback”.



Figura: 103. Funció onMapReady(), de la classe MapFragment

Primerament, es guarda la instància de la classe GoogleMap que la pròpia funció proporciona, ja que posteriorment se li donen més usos (com en la funció anteriorment mencionada):

*this.googleMap = googleMap;*

Abans de seguir amb l’ordre establert en la funció onMapReady(), s’explicarà el mètode googleMap.setOnMapClickListener() per facilitar la progressió del codi.

Aquest ens permet establir un Listerner que s’activarà quan l’usuari premi el mapa, indicant les coordenades exactes del lloc on ha clicat mitjançant un objecte LatLng. Un cop s’activa, s’executa la funció “onMapClick()”, en la que es posa un marcador en el punt indicat com en la funció “getNearEvents()” explicada anteriorment, i també es desplaça la càmera per centrar aquest punt en el mapa i fer-li zoom, tota aquesta última acció amb el mètode:

*googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(latLng, 15));*

On “latLng” és l’objecte “LatLng” obtingut i 15 és al nivell de zoom, que va de 2 a 21.

Després es crida a la funció “focusCamera”. Si l’usuari ha arribat al mapa mitjançant la ubicació d’un event, es passa aquesta ubicació a la funció:

*if(arguments!=null || eventLocation[0] != null || eventLocation[1] != null){*

*focusCamera(eventLocation[0], eventLocation[1]); }*

Sinó, s’obté la ubicació actual de l’usuari utilitzant una funció de la classe LocationActivity:

*else { findEvents =true;*

*locationActivity.lastLocation(o, getClass(), "focusCamera"); }*

Per fer-ho, s’utilitza el mateix mètode que s’ha utilitzat anteriorment per cridar un mètode de la classe EventsRequester desde MapFragment i obtenir la resposta d’aquest: es passa com a paràmetre la classe actual i el nom de la funció en la que es vol obtenir la resposta (en aquest cas, focusCamera). També s’activa un booleà per marcar-ho.

LocationActivity

Classe utilitzada per obtenir la ubicació actual de l’usuari.

Per fer-ho, s’utilitza la classe “FusedLocationProviderClient”, que serveix per interactuar amb la localització utilitzant el “fused location provider” (proveïdor d'ubicació fusionada).

El “fused location provider” és una API d'ubicació dels “Google Play services” que combina de manera intel·ligent diferents senyals per proporcionar la informació d'ubicació que una aplicació pot necessitar.

La classe LocationActivity només conta amb una funció i un constructor.

El constructor serveix per obtenir un parell de valors necessaris: Un de tipus Context i un de tipus Activity, i per donar un valor a la instància del FusedLocationProviderClient:

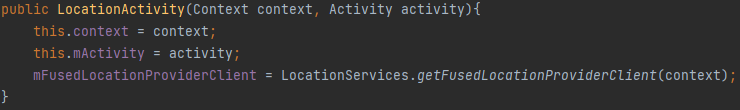


Figura: 104. Constructor de la classe LocationActivity

La funció que diferents classes criden per obtenir la localització actual del dispositiu és la “lastLocation()”.

Aquesta funció comença revisant que l’usuari conti amb tots els permisos necessaris per a accedir a la ubicació, utilitzant el mètode ActivityCompat.checkSelfPermission i comprovant els permisos de “fine location” i “coarse location”, explicats anteriorment.

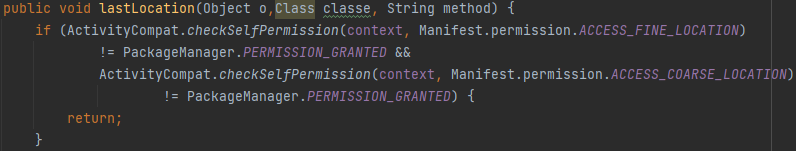


Figura: 105. Part de la funció lastLocation(), de la classe LocationActivity

El funcionament d’aquest procés és el mateix que a la classe MainActivity.

Si tot està en ordre es procedeix a obtenir la ubicació desitjada amb el mètode

“FusedLocationProviderClient.getLastLocation()”.

Aquest retorna la ubicació històrica més recent disponible actualment. Retornarà “null” si no hi ha cap ubicació històrica disponible. La ubicació històrica pot ser d'una edat arbitrària, però com no es necessita estar actualitzant la localització constantment, ja que la majoria d’events estan separats per bastant kilòmetres, serveix perfectament.

Si es volgués saber la ubicació amb més precisió s’hauria de comprovar l’antiguitat d’aquesta o utilitzar el mètode FusedLocationProviderClient.getCurrentLocation(), però aquest consumeix més recursos ja que sempre farà una nova petició per buscar la ubicació.

Un cop el sistema obté la ubicació, es detecta amb un Listener i aquesta s’envia a la funció especificada:

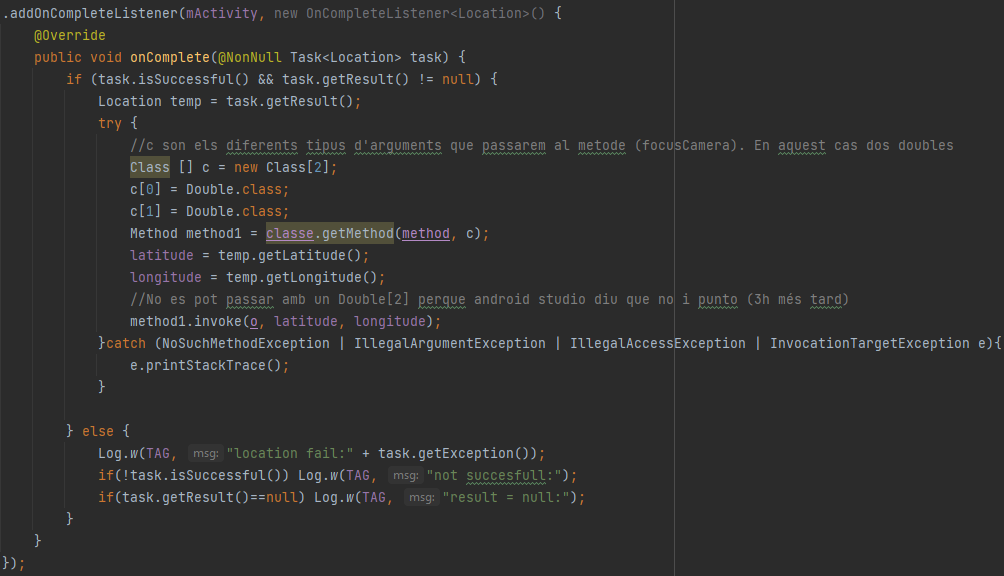


Figura: 106. Part de la funció lastLocation(), de la classe LocationActivity

Primer es comprova si la operació ha tingut èxit, ja que l’usuari pot tenir la ubicació desactivada en el seu dispositiu o pot fallar per altres motius:

*if (task.isSuccessful() && task.getResult() != null) {...*

Després es guarda la ubicació obtinguda en un objecte Location, ja que aquest és el format en que la retorna, utilitzant el mètode:

*Task.getResult();*

Finalment es retorna la ubicació obtinguda mitjançant la crida a una altra funció pertanyent a la classe que la requereix. S’utilitza el mètode:

*Method.invoke(Object receiver, Object... args);*

Aquest mètode requereix que primer s’especifiquin el tipus de paràmetres que es volen enviar, així com la classe a la que pertany el mètode i el seu nom. Finalment es crida i es passen per paràmetre dos objectes de tipus Double: la latitud i la longitud de la ubicació obtinguda.

Si s’ha cridat la funció lastLocation des de el mapa, es rebrà la resposta en la funció “focusCamera(Double latitude, Double longitude)”.

Retornant a la classe MapFragment, independentment de quin camí s’hagi recorregut (venint des d’un event o anant directament al mapa) s’ha obtingut una localització i s’executa la funció “focusCamera()”:

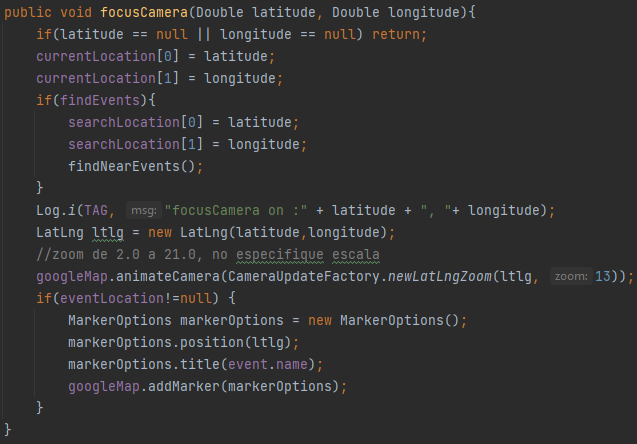


Figura: 107. Funció focusCamera(), de la classe LocationActivity

Primerament es comprova que s’ha obtingut una localització correcta i es guarda.

Si el booleà “findEvent” és cert, indica que s’ha vingut directament al mapa i es volen trobar els events propers a l’usuari, per tant es guarda de nou la ubicació obtinguda i es crida al ja mencionat mètode “findNearEvents()” per buscar-los.

Independentment del booleà, el següent pas és centrar la càmera del mapa en les coordenades indicades amb el ja mencionat mètode GoogleMap.animateCamera().

I finalment, si s’ha arribat al mapa des d’un event, es posa un marcador en la posició d’aquest indicant el seu nom, com ja s’ha mostrat en altres ocasions.

CHAT

Té un disseny bastant simple també: conta amb una part superior on es veuen els missatges que ocupa gairebé tota la pantalla i una part inferior amb dos botons i un camp per inserir text. El botó esquerra serveix per carregar i enviar imatges i el botó dret per enviar el missatge escrit al camp de text.

La barra inferior està creada amb dos ImageView per als botons i un EditText, tot dins d’una LinearLayout.



Figura: 108. Barra inferior de la pantalla Chat

La part superior és una RecyclerView per les mateixes raons que en la pantalla EventFragment: s’aprofiten les vistes dels missatges que ja no es mostren per pantalla per mostrar els nous.

A la pantalla de xat també es pot arribar de dos maneres: clicant la seva icona en el menú inferior o clicant la icona de xat:  d’un event concret.

La diferència és que si s’entra des d’un event, la sala de xat serà privada d’aquest event i si s’entra directament, la sala de xat serà global entre tots els usuaris de l’aplicació.

Cal mencionar que gestionar sales de xat només es gratuït fins a un cert punt, gràcies a Firebase. Per això de moment només hi ha una sala de xat global a part de les sales d’events.

Revisar: cal explicar la configuració de Firebase?

El primer pas en la implementació del xat és configurar Firebase correctament per tal de poder utilitzar el seu sistema d’autenticació i la seva “Realtime Database”.

També caldrà obtenir les següents llibreries:

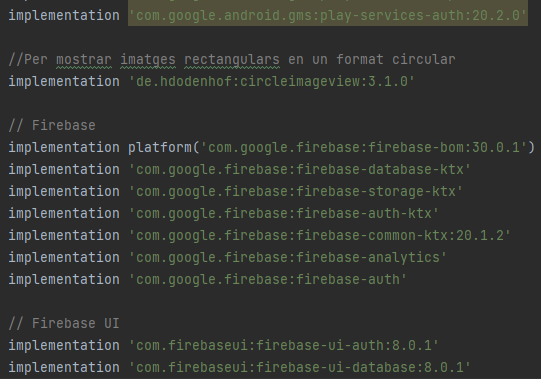


Figura: 109. Llibreries utilitzades en el sistema de xat, implementades des de l’arxiu build.gradle

El següent és escriure el codi, el qual s’ha realitzat tot en Kotlin.

Abans de començar, cal mencionar que gran part del sistema de xat ha estat inspirat en el repositori “friendlychat” de Firebase, ja que aquest mostra detalladament com crear un sistema de xat utilitzant les eines que Firebase proporciona.

Es començarà explicant la classe ChatMessage, que guarda tota la informació d’un missatge. Aquesta és el text del missatge, el nom de l’usuari que l’ha enviat, la foto de l’usuari i la URL d’una imatge en cas que n’hagués enviat una.

La classe només té aquestes variables i un constructor per donar-los-hi valor:

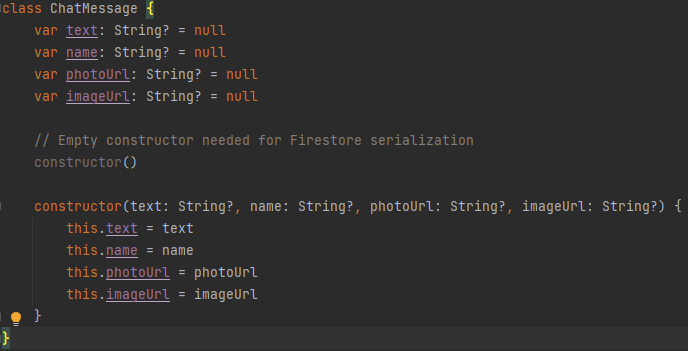


Figura: 110. Classe ChatMessage

Seguidament s’explicarà la classe principal ChatFragment:

Conté unes quantes variables que s’aniran explicant al llarg del procés:

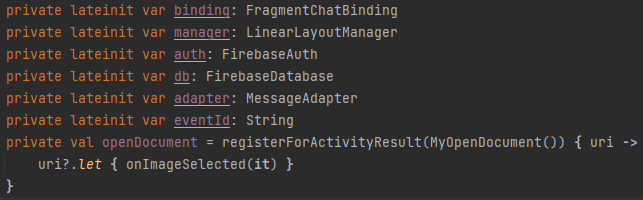


Figura: 111. Variables de la classe ChatFragment

La primera funció a explicar és “onCreateView()”:

És la funció més extensa de la classe i realitza unes quantes accions:

La primera és donar valors a totes les variables que s’han mostrat mitjançant els següent mètodes:

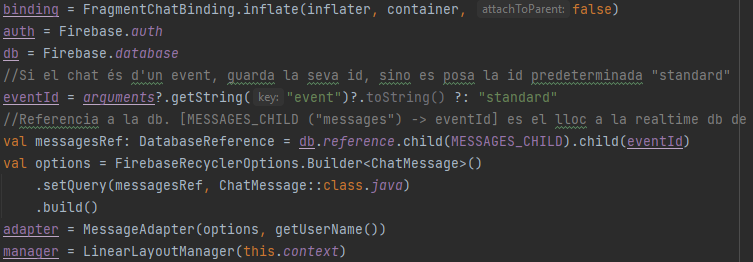


Figura: 112. Part de la funció onCreateView(), de la classe ChatFragment

*binding = FragmentChatBinding.inflate(inflater, container, false)*

Per al binding, explicat anteriorment

*auth = Firebase.auth*

Obté una instància de la classe abstracta FirebaseAuth, que és el punt inicial de la Firebase Authentication SDK. S’utilitzarà per obtenir l’usuari.

*db = Firebase.database*

Obté una instància de la classe FirebaseDatabase, necessària per accedir a la base de dades de Firebase. S’utilitzarà per guardar i extreure dades d’aquesta.

*eventId = arguments?.getString("event")?.toString() ?: "standard"*

Si s’ha arribat des d’un event, obté el ID d’aquest i el guarda, sinó simplement guarda la paraula “standard”. S’utilitzarà per ubicar la sala de xat.

*manager = LinearLayoutManager(this.context)*

Obté una instància de la classe LinearLayoutManager, utilitzant-n’hi una de la llibreria RecyclerView com s’ha fet també en el sistema d’events.

*adapter = MessageAdapter(options, getUserName())*

Obté una instància de la classe “MessageAdapter”, que s’explicarà més endavant. El constructor requereix dos paràmetres: un FirebaseRecyclerOptions i un String.

El FirebaseRecyclerOptions s’ocupa de controlar com FirebaseUI omple la RecyclerView amb les dades de la “Realtime Database”. S’obté amb la següent funció:

*val options = FirebaseRecyclerOptions.Builder<ChatMessage>()*

*.setQuery(messagesRef, ChatMessage::class.java)*

*.build()*

De nou, aquesta funció requereix un paràmetre que no es té, referenciat com “messagesRef”. Aquest paràmetre és un DatabaseReference i representa una localització concreta en la base de dades. Serveix per llegir o escriure dades en ella i la funció que s’utilitza per obtenir-lo és:

*val messagesRef: DatabaseReference = db.reference.child(MESSAGES\_CHILD).child(eventId)*

Finalment, l’adaptador requereix una String que obté amb la funció getUserName(), que s’explicarà més tard.

L’últim pas mostrat és vincular l’adaptador i el manager de la RecyclerView amb el MessageAdapter i el LinearLayoutManager obtinguts respectivament.

Un cop fet, la classe registra tres Listeners i un Observer:

L’Observer utilitza el mètode adapter.registerAdapterDataObserver(...). Aquest registra un Observer que “observa” els canvis en l’adaptador definit.

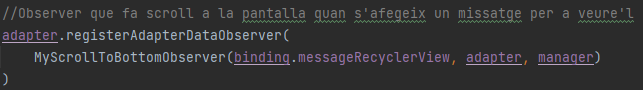


Figura: 113. Part de la funció onCreateView(), de la classe ChatFragment

Un cop detecta canvis, significa que s’ha afegit o eliminat un missatge, i avisa a la classe “MyScrollToBottomObserver”

Aquesta classe simplement s’ocupa de desplaçar-se en la RecyclerView per tal de mostrar el nou missatge o no perdre espai amb un missatge eliminat.

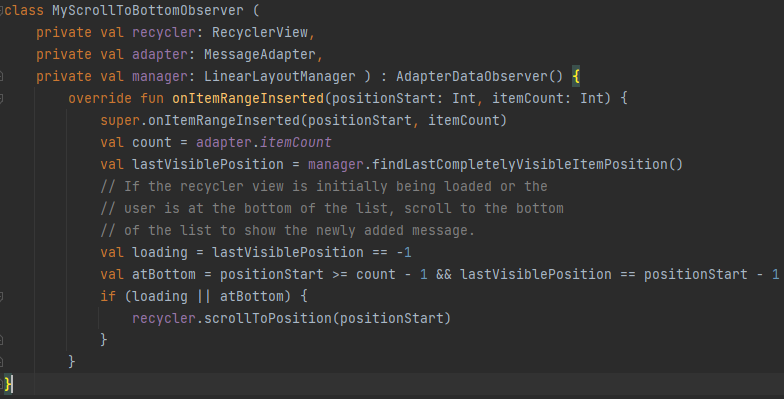


Figura: 114. Classe MyScrollToBottomObserver

Per fer-ho, primer calcula si realment s’ha de desplaçar amb una operació utilitzant el nombre de missatges existents en la sala amb el mètode “adapter.itemCount” i la seva posició actual amb el mètode “manager.findLastCompletelyVisibleItemPosition()”.

Si s’ha de desplaçar utilitza el mètode:

*recycler.scrollToPosition(positionStart)*.

El primer Listener registrat utilitza el mètode “EditText.addTextChangedListener”. Aquest escolta els canvis en el editor de text indicat (quan l’usuari escriu un missatge) i crida a la classe “MyButtonObserver”.



Figura: 115. Part de la funció onCreateView(), de la classe ChatFragment

Aquesta classe s’ocupa de canviar el color del botó per enviar els missatges quan hi ha text, de forma que quan l’usuari comença a escriure un missatge, el botó canvia de color i quan l’envia o borra tot el text el botó torna al seu color original.

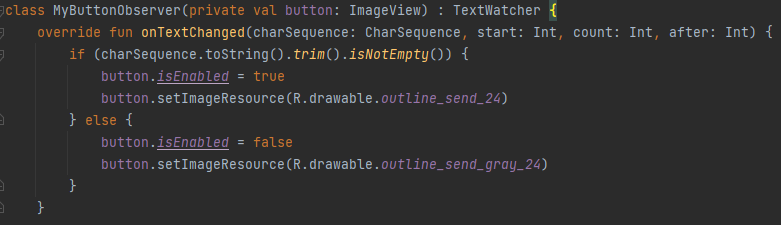


Figura: 116. Classe MyButtonObserver

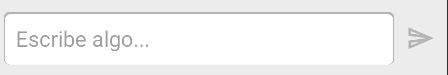
 

Figura: 117. Botó d'enviar quan hi ha text Figura: 118. Botó d'enviar quan no hi ha text

El segon Listener registrat detecta quan es fa clic sobre el botó d’enviar i crea una instància de la classe ChatMessage amb el missatge, el nom de l’usuari i la seva fotografia. Per obtenir el nom i fotografia de l’usuari utilitza dos funcions:

“getUserName()”: Utilitza el sistema d’usuaris de Firebase en el que prèviament es va guardar la informació d’aquest (al registrar-se) per extreure el nom amb el mètode “user.displayName”

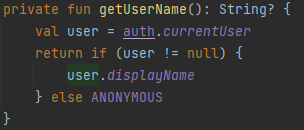


Figura: 119. Funció getUserName(), de la classe ChatFragment

“getPhotoUrl()”: Utilitza el mateix sistema amb el mètode “user.photoUrl”.

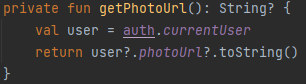


Figura: 120. Funció getPhotoUrl(), de la classe ChatFragment

Posteriorment, aquest Listener penja el missatge a la base de dades i borra el camp de text en el que s’ha escrit el missatge: . Part de la funció onCreateView(), de la classe ChatFragment

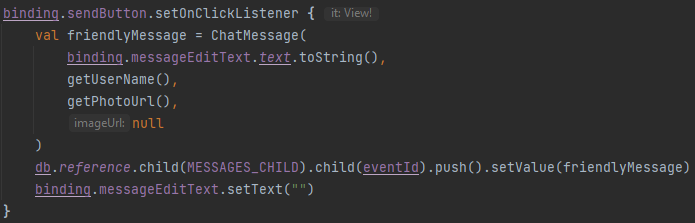


Figura: 121. Part de la funció onCreateView(), de la classe ChatFragment

L’últim Listener registrat detecta quan es fa clic sobre la icona de penjar imatges.



Figura: 122. Part de la funció onCreateView(), de la classe ChatFragment

OpenDocument és una variable que s’inicialitza directament amb el següent codi:

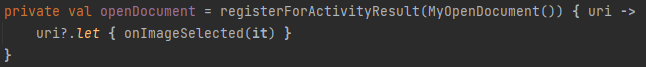


Figura: 123. Variable de la classe ChatFragment

Aquest utilitza un “registerForActivityResult”, que serveix per obtenir el resultat d’una activitat. En aquest cas l’activitat és “MyOpenDocument()”:

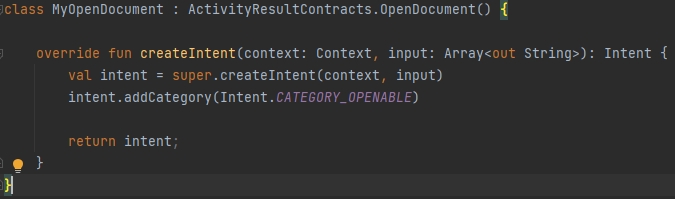


Figura: 124. Classe MyOpenDocument

Aquesta activitat retorna un Intent amb la categoria CATEGORY\_OPENABLE. Aquesta categoria denota que el tipus de dada que es vol rebre de l’Intent pot ser oberta com un objecte “File”; en aquest cas es voldrà una imatge.

El codi d’inicialització de la variable openDocument també indica que es passarà el resultat de l’Intent a la funció onImageSelected().

En resum, quan el listener s’activa, el mètode “openDocument.launch(arrayOf("image/\*"))” busca en el dispositiu de l’usuari arxius de tipus imatge i un cop se’n selecciona un, es passa la URI d’aquest a la funció onImageSelected().

Per últim, la funció onCreateView() també s’ocupa de canviar el nom del xat en el menú superior en cas que sigui la sala de xat d’un event concret, posant el nom d’aquest:

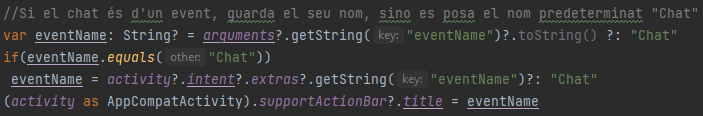


Figura: 125. Part de la funció onCreateView(), de la classe ChatFragment

El que es fa és buscar si s’han passat paràmetres en els Bundles “arguments” i “intent.extras”, ja que depenent de la pantalla de procedència poden estar en qualsevol dels dos. Si es troba el nom d’un event en un d’ells es canvia el títol del menú superior, sinó es deixa com “Chat”.

El mètode per canviar el títol és:

*(activity as AppCompatActivity).supportActionBar?.title = eventName*

S’utilitza la part “(activity as AppCompatActivity)” ja que es requereix una Activity o AppCompatActivity i no una FragmentActivity (que és la que es rep) per utilitzar el mètode supportActionBar, que permet accedir al menú superior per modificar-lo.

La següent classe a explicar és “onImageSelected(uri: Uri)”, que com ja s’ha explicat es crida quan l’usuari selecciona una imatge per penjar-la al xat i conté la URI d’aquesta:

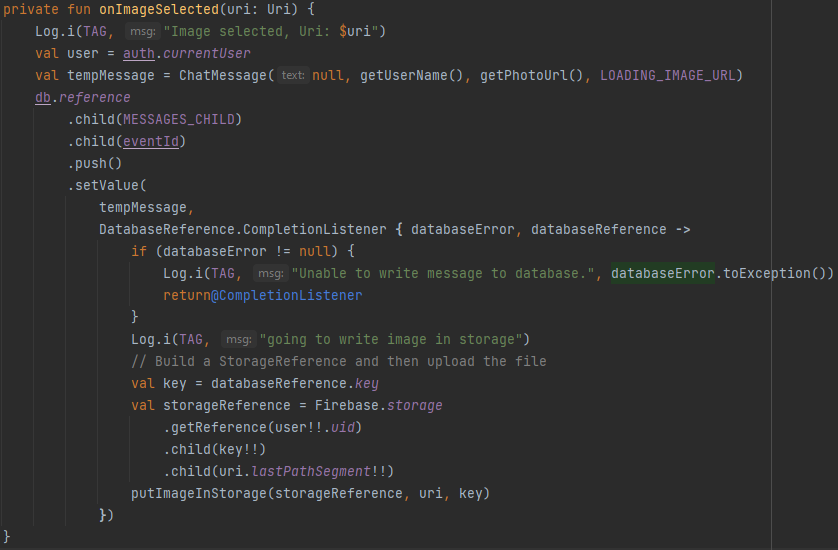


Figura: 126. Funció onImageSelected(), de la classe ChatFragment

Aquesta classe crea un ChatMessage temporal que enlloc de tenir la ruta de la imatge té la URL d’un gif:  que mostra un efecte de càrrega i el penja a la base de dades.

Al mateix temps obté una referencia a la base de dades (un objecte DatabaseReference), de la que s’extreu la clau, i al emmagatzematge (un objecte FirebaseStorage), que utilitza com a paràmetres al cridar a la funció putImageInStorage():

Finalment, la classe putImageInStorage() s’ocupa de penjar la imatge utilitzant la referència a l’emmagatzematge de Firebase i actualitzar el missatge guardat a la base de dades amb la URI que s’ha obtingut al penjar la imatge.

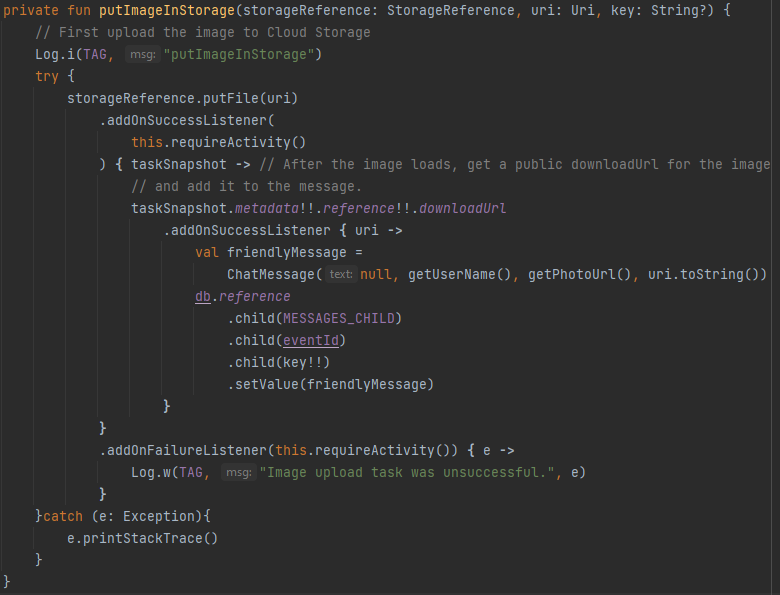


Figura: 127. Funció putImageInStorage(), de la classe ChatFragment

La imatge es penja amb el mètode

*storageReference.putFile(uri)*.

Si es penja amb èxit, s’obté la seva URL amb el mètode *taskSnapshot.metadata!!.reference!!.downloadUrl*

i finalment s’actualitza el missatge com si se’n penges un de nou, però amb la ruta d’un missatge ja existent, per tant s’actualitza.

Cal mencionar que aquest procés actualment falla en alguns dispositius per un bug en la llibreria RecyclerView que encara no s’ha solucionat.

Per acabar amb el xat queda per explicar l’adaptador de la RecyclerView: la classe MessageAdapter. Com l’EventAdapter, aquest s’ocupa de vincular les diferents dades a les vistes.

Comença amb la funció onCreateViewHolder(), que distingeix si el missatge és un text o una imatge i vincula una un disseny o Layout diferent per a cada un.

Els dos dissenys conten amb un camp de text per al nom de l’usuari i una imatge redona en la que es mostra la foto de l’usuari, utilitzant la llibreria 'de.hdodenhof:circleimageview:3.1.0'.

El disseny per a missatges conta amb un segon camp de text, i el disseny per a imatges amb una vista d’imatges normal o “ImageView”.

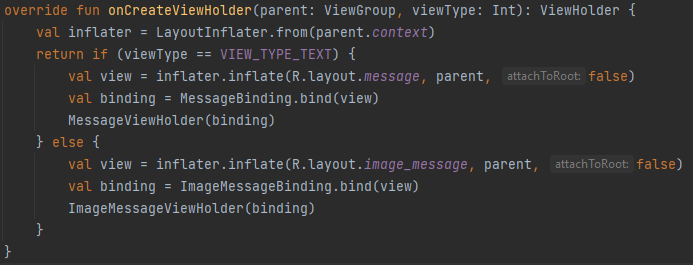


Figura: 128. Funció onCreateViewHolder(), de la classe MessageAdapter

També fa el “binding” o unió del disseny amb la vista, explicat anteriorment i el passa a les classes internes MessageViewHolder o ImageMessageViewHolder.

La següent funció, que s’executa quan detecta el binding, és onBindViewHolder(), i simplement s’ocupa de cridar a la funció bind() de la classe interna corresponent, depenent de nou si és un missatge de text o una imatge:

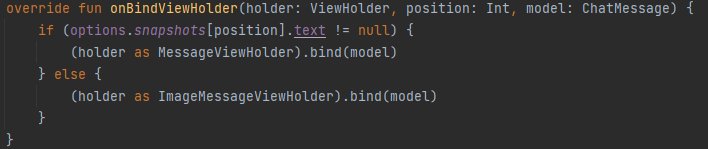


Figura: 129. Funció onBindViewHolder(), de la classe MessageAdapter

La funció bind() s’ocupa de vincular les dades del missatge a les vistes del disseny.

La classe MessageViewHolder posa el text al missatge, li dona color, posa el nom de l’usuari i carrega la imatge d’aquest si és que en té:

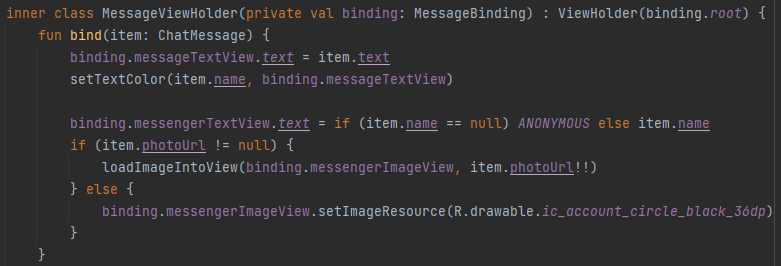


Figura: 130. Funció bind(), de la classe interna MessageViewHolder, que pertany a la classe MessageAdapter

Per donar el color al text s’utilitza la funció setTextColor(), que detecta si el missatge en qüestió és del propi usuari o d’una altra persona i el posa de color blanc amb fons blau o color negre amb fons gris respectivament.

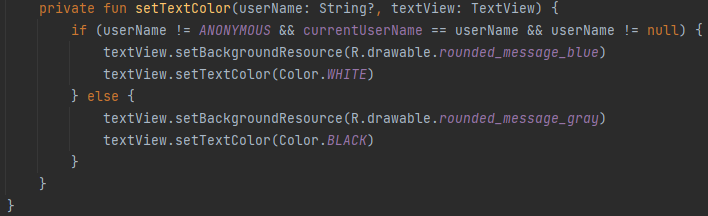


Figura: 131. Funció setTextColor(), de la classe interna MessageViewHolder, que pertany a la classe MessageAdapter

Per carregar la imatge s’utilitza la funció loadImageIntoView(). Aquesta revisa la URL de la imatge i si comença per gs:// vol dir que pertany a l’emmagatzemament de google i es pot accedir a ella mitjançant la SDK de Firebase Storage, o sigui amb una referència a l’emmagatzematge o “StorageReference”, i obtenir la seva URL de descàrrega. Si la URL inicial no comença amb “gs://” s’interpreta que és una URL normal i s’utilitzarà aquesta. Un cop es té la URL de descàrrega o la URL inicial respectivament, s’utilitza de nou la llibreria Glide per carregar les imatges a la vista pertinent.

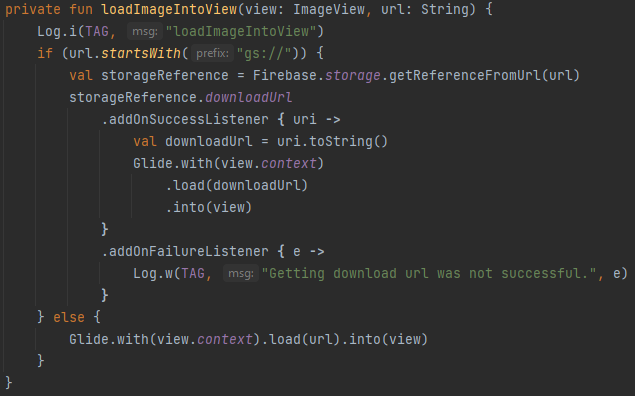


Figura: 132. Funció loadImageIntoView (), de la classe MessageAdapter

· Map: explicar procés de implementació del mapa mitjançant google developer console

<https://console.cloud.google.com/apis/dashboard?project=eventalk-bc6d4>

# Conclusions i treball futur

## Conclusions

El primer a remarcar és que s’han assolit la gran majoria dels objectius establerts a l’inici d’aquest projecte. S’ha pogut crear una aplicació funcional per a dispositius Android que mitjançant l’ús de diverses APIs és capaç de buscar i mostrar events, ubicar-los en un mapa juntament amb l’usuari i amb una sala de xat pròpia per cada un d’ells.

L’objectiu de familiaritzar-se amb les API s’ha assolit al haver-n’hi d’utilitzar 3 de diferents: l’API de Ticketmaster, l’API de Google Maps i l’API de Firebase. Després d’hores consultant detalls i escrivint codi, considero que he aprés els conceptes més importants per a la implementació d’APIs.

Revisar: si s’ha canviat lo de familiaritzar-me amb android canviar aquí tmb

L’objectiu de familiaritzar-se amb el desenvolupament d’aplicacions en entorns Android també s’ha assolit ja que es contava amb una base prèvia i aquesta s’ha assentat al desenvolupar el projecte.

Els objectius d’implementar el sistema de cerca d’events, un mapa funcional i el sistema de xat també s’han assolit, l’únic problema ha estat a l’hora d’afegir zones amb molta afluència de gent o punts d’interès al mapa, tasca que ha resultat impossible degut a la falta de dades aportades per les API de les que es pot disposar al mercat amb un pressupost limitat.

Finalment, els objectius d’implementar totes aquestes característiques en una sola aplicació totalment funcional i planejar la seva monetització també s’han complert, ja que l’aplicació es podria treure al mercat en el seu estat actual (tot i que per fer-ho bé encara li queden bastantes millores i retocs per afegir) i afegir els sistemes de monetització mencionats en el seu apartat per a intentar treure un profit econòmic d’ella.

Personalment, considero que ha set una experiència molt enriquidora que m’ha aportat coneixements en molts àmbits que desconeixia totalment i m’ha permès entendre d’una manera molt més profunda tot el procés de desenvolupar una aplicació Android des de zero. Estic segur que utilitzaré tots aquests coneixements per futurs projectes, ja que degut al sistema de documentació del TFG m’he hagut d’informar de molts aspectes tècnics que a més a més, han quedat guardats en múltiples documents així com en aquest document de text.

## Treball futur

En quant al futur del projecte respecta, és tracta una aplicació molt complexa, per tant no s’han pogut implementar totes les funcionalitats que podria arribar a tenir. Per arribar a igualar a altres aplicacions de la competència en quant a funcionalitats que s’ofereixen i qualitat d’aquestes encara està molt lluny, inclús a anys de desenvolupament ja que aquestes conten amb grans equips de professionals que porten anys treballant en elles amb pressuposts milionaris.

Per una altra banda, no es això al que s’aspira amb aquesta aplicació, però si que hi ha un seguit de característiques que es volen afegir en un futur:

Primerament, s’ha de millorar l’aspecte visual de l’aplicació: és un aspecte en el que no ha rebut el focus principal d’atenció però és molt important. Cal millorar les fonts, el disseny, posar efectes a les transicions de pantalla i als botons així com animar el fons i donar un format específic i propi als elements del disseny.

Per una altra banda també caldria afegir moltes funcionalitats que es voldria que estiguessin en la versió comercial de l’aplicació, com un sistema de gestió de perfil, notificacions, un sistema de contes prèmium amb múltiples avantatges, un espai de contacte, opcions de configuració personalitzades...

Amb tot això ja estaria més llesta per al llançament, però abans m’agradaria fer alguns canvis més com ara afegir una altra API d’events, ja que Ticketmaster no n’ofereix massa, afegir més filtres a la cerca d’aquests, afegir més funcionalitats als events, com ara canviar la galeria (que per canvis en la API de Ticketmaster ja no és funcional) per una llista d’assistents a l’event entre altres, i millorar també el xat afegint la possibilitat de tenir múltiples sales obertes i compartir events entre usuaris.

Finalment, un cop publicada s’haurien de seguir fent millores per apropar-se més a la competència directa com ara sales de xat privades, afegir punts d’interès amb sales de xat al mapa, guardar les consultes a les API d’events en memòria RAM per no repetir-les, poder crear events propis, un sistema de compra d’entrades... Però quasi totes les funcionalitats recentment mencionades (menys la de guardar consultes) costen diners, per tant no s’espera aplicar-n’hi cap a curt termini.

Com es pot veure, les opcions són il·limitades i es podrien afegir tantes funcionalitats que no es poden redactar totes, sense contar amb els constants avenços en el camp de la informàtica que permeten seguir millorant i optimitzant qualsevol aplicació ja existent.

BIBLIOGRAFIA

Com posar pàgines web:

Sitio / Página Web

Apellido, Inicial del nombre o Autor Corporativo (fecha del copyright o última actualización, si se desconoce, poner n.d.): Título de la página web. Disponible en: URL del recurso [Consulta: día-mes-año]

Centre for Academic Writing (2006):The List of References Illustrated. Disponible en: http://home.ched.coventry.ac.uk/caw/harvard/index.htm [Consulta: 20 de julio de 2006]

Ticketmaster, (2022). *Discovery API 2.0*

<https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/>

Consultat el: dia/09/2022

Visual Paradigm, *Use case diagram*

Cerc que igual haurie de ser un annex ja no es informacio

<https://online.visual-paradigm.com/es/diagrams/solutions/free-use-case-diagram-tool/>

Sameeksha Medewar (2/02/2022). *Los 7 mejores IDE para el desarrollo de aplicaciones móviles*

<https://geekflare.com/es/best-ide-for-mobile-app-development/>

Consultat el: 12/02/2022 al 29/05/2022

Es d’on s’ha tret una imatge tot i que indica els llenguatges més usats cada més, igual val com a info a l’annex

<https://www.tiobe.com/tiobe-index/> al 4.2, per si volem els llenguatges mes usats

Android, (2022-09-01 última actualització). *¿Por qué Kotlin tiene prioridad en el desarrollo de Android?*

<https://developer.android.com/kotlin/first#why> Consultat el: 13/09/2022

Android, (2022-10-10 última actualització). *Entorno de ejecución de SDK*

<https://developer.android.com/design-for-safety/privacy-sandbox/sdk-runtime> Consultat el 12/09/2022

m2.material, (X). *Material 2 Buttons*

<https://m2.material.io/components/buttons#theming> explicació sobre botons, annex si eso

Consultat el: 25 /10/ 2022

M2.material, (X). *Material 2 Bottom Navigation*

<https://m2.material.io/components/bottom-navigation/android#using-bottom-navigation>

Consultat el: Inici del projecte: 07/05/2022

Android, (2022-02-10 última actualització). *SharedPreferences*

<https://developer.android.com/reference/android/content/SharedPreferences>

Consultat el: 26 /09/ 2022

Saket Kumar, (18-10-2021 última actualització). *Builder Design Pattern*

<https://www.geeksforgeeks.org/builder-design-pattern/>

Colsultat el: 15-09-2022

Android, (2021-08-02 última actualització). *Cómo crear listas dinámicas con RecyclerView*

<https://developer.android.com/guide/topics/ui/layout/recyclerview?hl=es-419>

Consultat el: 15-09-2022

# [ANNEXOS]

**• Manual d’usuari • Relació de figures / taules • Especificació casos d’ús, Diagrames de seqüència, col·laboració, etc.....**

Bibliografia, preparació de l’entorn, configuracions, manuals d’instalació dels programes, guia d’us de la app? També llista d’imatges i taules (num, nom, pag), i poco mas

Afegiria guies per tot lo que no he explicat, en plan links a les guies d’instal·lació de les API, d’iniciació a Android Studio, a firebase etc.

Coses a afegir si s’escau: patrons utilitzats – carpetes de vàlues – conceptes d’android Studio que utilitzarem: manifest, gradle, algunes llibreries, ...

<https://app.logo.com/>

<https://es.statista.com/estadisticas/894093/uso-semanal-de-las-apps-moviles-en-espana/>

Paraules a la intró, objectius i conclusions

Vol que posi frases als agraïments