



## Culture finder

[Github](#) - [Taiga](#) - [Drive](#) - [Project Record Track](#)

Cognom	Nom	Responsable	UPC e-mail	Taiga	GDrive	Github
Morón	Daniel	Management services of	daniel.moron.roces@estudiantat.upc.edu	danielmr_6	daniel.moron.roces@estudiantat.upc.edu	danielmr6
Risueño	Iván	Inception	ivan.risueno@estudiantat.upc.edu	ivan.risueno	ivan.risueno@estudiantat.upc.edu	ivan-risueno
Rodriguez	Marc	Sprint 2	marc.rodriguez.martin@estudiantat.upc.edu	marcrd11	marc.rodriguez.martin@estudiantat.upc.edu	MarcRd11
Duch	Marc	Sprint 1	marc.duch@estudiantat.upc.edu	marc.duch	marc.duch@estudiantat.upc.edu	Marcarrones
Moreno	Miguel	Final demo and closing documentation	miguel.moreno.alcaraz@estudiantat.upc.edu	MiguelMorenoAlcaraz	miguel.moreno.alcaraz@estudiantat.upc.edu	MiguelMorenoAlcaraz
Delgado	Òscar	Sprint 3	oscar.delgado.gomez@estudiantat.upc.edu	oscard14	oscar.delgado.gomez@estudiantat.upc.edu	oscard147

<b>1. Requisites</b>	<b>4</b>
<b>1.1. Concepció general del projecte</b>	<b>4</b>
<b>1.2. “NOT” list</b>	<b>4</b>
<b>1.3. Model conceptual de les dades</b>	<b>7</b>
<b>1.2. Resum del backlog del producte al document</b>	<b>8</b>
1.2.1. Sprint 1 Backlog	8
1.2.2. Sprint 2 Backlog	16
1.2.3. Sprint 3 Backlog	16
<b>1.4. Requisites no funcionals</b>	<b>17</b>
1.5.1 Requisites de rendiment	17
1.5.2 Requisites de preservació i suport	17
1.5.3 Requisites de capacitat d'ús i humanitat	18
<b>1.5. Tractament dels aspectes transversals</b>	<b>19</b>
<b>1.6. Serveis de tercers</b>	<b>23</b>
1.6.1 Repartiment de funcionalitats entre els equips	23
1.6.2 Comunicació amb la resta dels equips	24
<b>2. Metodologia</b>	<b>24</b>
2.1. Visió General	24
2.2. Gestió Projecte	25
2.3. Gestió versions	26
2.4. Comunicació d'equip CultureFinder	28
2.5. Frameworks i llenguatges	29
2.6. IDEs	30
2.7. Gestió de qualitat	30
2.8. Interacció amb altres grups	31
2.9. Convencions de codi	31
2.9.1. Variables	31
2.9.2. Funcions i mètodes	31
2.9.3. Classes	32
2.9.4. Fitxers i directoris	33
2.10. Gestió de bases de dades	33
<b>3. Descripció tècnica</b>	<b>34</b>
<b>3.1. Concepte de l'arquitectura</b>	<b>34</b>
<b>3.1.1. Arquitectura física</b>	<b>34</b>
<b>3.1.2. Patrons d'arquitectura aplicats</b>	<b>35</b>
<b>3.2. Capa de domini</b>	<b>36</b>
<b>3.2.1. Diagrama de models del domini (optional)</b>	<b>36</b>
<b>3.2.2. Patrons de dissenys aplicats</b>	<b>36</b>
<b>3.3. Diagrama de la base de dades (UML)</b>	<b>37</b>
<b>3.4. Llistat de les tecnologies</b>	<b>38</b>
<b>3.5. APIs</b>	<b>38</b>
<b>3.5.1. API del nostre producte</b>	<b>38</b>
3.5.1.1. /users	38

3.5.1.2. /lists	39
3.5.1.3. /incidents	40
3.5.1.4. /events	41
3.5.1.5. /assistances	44
<b>3.5.2. APIs externes</b>	<b>44</b>
3.5.2.1. Google Maps API	45
3.5.2.2. Firebase Auth API	45
3.5.2.3. Geolocator API	46
3.5.2.4. Permission handler API	46
3.5.2.5. Shared preferences API	46
<b>3.5.3. Consum del servidor</b>	<b>46</b>
<b>3.5.4. Subministrament del servidor</b>	<b>47</b>
<b>3.5.5. Consum de Dades Obertes</b>	<b>47</b>
<b>3.6. Eines de desenvolupament i entorn de treball</b>	<b>48</b>

# 1. Requisites

## 1.1. Concepció general del projecte

El nostre projecte consisteix en una aplicació nativa per a Android que recull tots els esdeveniments de tipus cultural que es fan a Catalunya, i permet a l'usuari buscar-los i organitzar-los de manera personalitzada i apuntar-se als que vulgui.

## 1.2. “NOT” list

A continuació mostrem la NOT-List actualitzada que anirem modificant a mesura que avancem en les fases del projecte. En **verd** surten les funcionalitats que estan actualment implementades, i en **vermell** les que hem decidit deixar-les fora.

IN SCOPE	OUT OF SCOPE
Geolocalització	Compra d'entrades
Localitzar esdeveniments (Vista Mapa)	Integració amb taxis o altres sistemes de transport
Guardar dades dels esdeveniments que ens proporciona la API, pero només els posteriors al 1/1/2023	Afegir nous esdeveniments
Calendari d'esdeveniments (Vista Calendari)	Perfils d'organitzadors
Esdeveniments mostrats en forma de llista (Vista Llista)	Modificar esdeveniments
Poder buscar i filtrar esdeveniments per diferents criteris (geogràficament, categories, data, nom, popularitat...)	Informació dels esdeveniments a temps real
Mostrar informació detallada dels esdeveniments (Vista Detall)	Mètodes d'iniciar sessió amb reconeixement facial o empremta dactilar
Sincronització de dades (Dades de l'API agenda cultural)	Comprovació de la veracitat i confiança dels esdeveniments
Sistema multi idioma (català, castellà, anglès) [Només menú]	Funcionament fora de Catalunya
Recomanació d'activitats segons els interessos dels usuaris	Possibilitat de promocionar activitat cultural

Compartir esdeveniments a través de xarxes socials o URL	
Valoració d'esdeveniments entre 1 i 5 punts	Xat amb els responsables dels esdeveniments
Creació, modificació i alta de perfils	Traçat del camí fins l'esdeveniment / direccions
Notificar assistència als esdeveniments	Sistema multiplataforma
Llista d'esdeveniments favorits	Fòrum de comentaris sobre els esdeveniments
Creació de llistes d'esdeveniments personalitzades	
Sistema per reportar errors d'esdeveniments	
Pàgina per administrador per consultar i respondre als reports dels usuaris	
Sincronització amb Google Calendar	
Apartat de dubtes freqüents i ajuda	
Sistema de notificacions per als usuaris sobre els esdeveniments als que han afegit a alguna llista, assistiran o els poden interessar	
Estadístiques de l'esdeveniment (número assistents, valoració mitjana...)	
Mostrar els esdeveniments més populars a través d'un mapa de calor	

D'aquesta llista, primerament podem dir que no hem implementat cap funcionalitat inclosa en la part 'out of scope', ja que van quedar descartades a l'etapa d'incepció i a mesura que hem avançat el projecte hem confirmat que l'abast del nostre projecte és prou adequat. Aquestes funcionalitats estan marcades en color **vermell**.

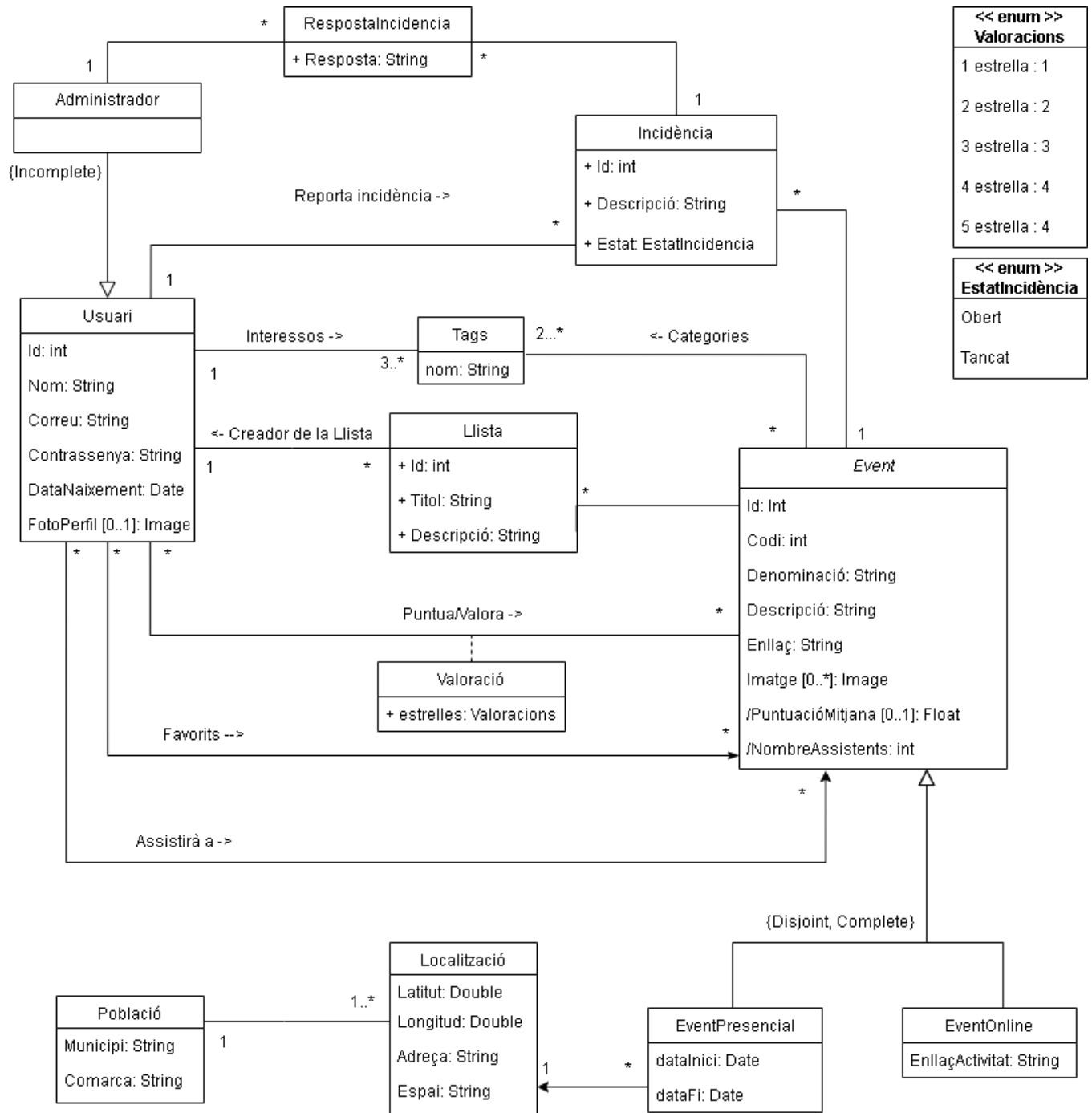
Les funcionalitats que hem implementat a l'aplicació durant aquest esprint han estat les marcades en **blau** a la not list. Aquestes funcionalitats han estat completament finalitzades i funcionen completament. Incloent la part de

Backend i Frontend. Aquestes funcionalitats són les més importants del sistema com les vistes principals (vista detallada, llista, calendari...), mostrar el mapa amb la localització pròpia i la dels esdeveniments i les funcionalitats més importants del Backend com guardar els esdeveniments a partir del 2023 a la base de dades i proporcionar l'API al Frontend.

Les funcionalitats que hem implementat a l'aplicació, però que no han estat finalitzades del tot, per problemes de temps o d'altres, estan marcades de color **groc**. Aquestes funcionalitats inclouen parts crítiques de l'aplicació que han portat una complexitat alta i d'altres que encara que no eren massa complexes, han sorgit altres problemes con canvis de prioritat o conflictes previs. Per exemple, per les estadístiques d'esdeveniments està implementada la valoració mitjana però falta per acabar d'afegir el nombre d'assistents. A més, la sincronització periòdica de dades ha tingut una complexitat més alta de l'esperada al primer sprint, però pel segon ja es va poder tancar. Les altres dues, sistema per reportar errors d'esdeveniments i la pàgina d'administració no s'han pogut acabar pel segon sprint. Pel que fa a aquestes en el segon sprint es van finalitzar la implementació de la lògica d'incidències, a falta de l'API Token. En addició, pel que fa a la pàgina d'administració encara queda terminar-la amb les crides a la nostra API. En el cas dels perfils, s'ha implementat la lògica de creació d'usuari i l'inici de sessió a la part del Backend, i després ja es van implementar les vistes del Frontend. Tampoc s'ha implementat la modificació de perfil.

Les funcionalitats que no hem implementat de la llista de la part de 'in scope' estan marcades de color **taronja**. Aquestes funcionalitats no han estat implementades, ja que no estaven previstes per l'sprint 1 ni pel sprint 2.

### 1.3. Model conceptual de les dades



## RESTRICCIONS TEXTUALS

### 1. Claus Primaries:

**Usuari** (Id); **Tags** (nom); **Event** (Id); **Llista** (Id + Usuari.Id); **Incidència** (Id); **Localització** (Latitud + Longitud + Espai); **Població** (Municipi + Comarca)

2. La dataInici d'un EventPresencial no pot ser posterior a la dataFi

3. La dataNaixement d'un Usuari no pot ser posterior a la data actual

4. Un Usuari no pot Puntuar/Valorar un Event al que no Assistirà

5. L'atribut PuntuacióMitjana d'un Event es calcula segons la mitjana d'estrelles de Valoració

6. L'atribut NombreAssistents d'un Event és el nombre d'Usuaris que *assistiran* al Event

## 1.2. Resum del backlog del producte al document

### 1.2.1. Sprint 1 Backlog

Èpica		Història d'usuari			Tasques	Front/Back
#	Títol	#	Títol	Punts	#num_tasca: Nom tasca frontend	F
					#num_tasca: Nom tasca backend	B
-	-	-	Storyless tasks	-	#125: Infraestructura del domini	B
					#111: Implementar infraestructura per a la traducció dels elements	F
#5	Altres Funcionalitats	#6 6	Informació consistent	22.5	#82: Setup de la base de dades	B
					#83: Cridar l'Api de l'Agenda Cultural	B
					#85: Processar preu	B
					#86: Processar Comarca i Municipi	B
					#96: Processament de dades per data inicial/final	B
					#114: Processar Tags:Àmbits/Categories/Altres_Categories	B
					#113: Processar Descripcions	B
					#115: Processar Localització	B



#1	Cerca i consulta esdeveniments	#10	Consultar esdeveniments en llista	8	#110: Vista llista esdeveniments	F
					#118: Component element de llista	F
					#117: Exposar endpoint API	B
		#13	Consultar informació detallada d'un esdeveniment	8	#104: Mostrar dadades d'event	F
					#119: Exposar endpoint API	B
		#15	Cercar esdeveniments filtrant per categoria	10	#109: Mostrar esdeveniments filtrats	F
					#121: Crear endpoint API amb paràmetres per filtrar	B
					#122: Crear endpoint per llistar possibles tags	B
		#14	Cercar esdeveniments per nom	4.5	#99: Filtratge per nom	B
					#106: Mostrar events del filtratge	F
		#12	Consulta la meva ubicació al mapa	4.5	#123: Integrar API Google Maps	F
					#108: Mostrar Mapa	F
					#126: Cridar API localització dispositiu	F
		#11	Consulta la ubicació dels esdeveniments al mapa	11	#93: Mostrar ubicació al mapa	F
		#17	Cercar esdeveniments filtrant per rang de dates	8	#101: Filtratge per data (API)	B
					#124: Afegir Selector de dates	F

		#21	Consultar esdeveniments mitjançant el calendari	12	#112: Mostrar esdeveniments en vista calendari	F
#3	Gestió Usuari	#27	Crear Usuari	9	#98: Crear vista creació d'usuari	F
					#100 Implementar formulari inicial	F
					#102: Implementar formulari d'interessos	F
					#103: Guardar usuari a la base de dades	B
		#41	Iniciar Sessió	8	#91: Autenticar usuari (API)	B
					#90: Enviar credencials i rebre resposta	F
					#89: Crear vista d'iniciar sessió	F
#6 3	Incidències	#39	Reportar error d'esdeveniment	13	#94: Crear botó/event/vista per tancar sessió	F
					#107: Crear vista d'incidència	F
#5	Altres Funcionalitats	#64	Dades actualitzades	13.5	#95: Crida periòdica a Dades Obertes (sync)	B
-	-	-	Storyless Task	-	#97: Implementar navegabilitat entre pantalles (navbar)	F

En total ens han sortit 136.5 *story points* dels 190 que haurien d'haver sortit si haguéssim distribuït les històries de forma equitativa. Aquests 136.5 punts no inclouen les 3 tasques (#97, #11, #125) que no estan assignades a cap història d'usuari ni la feina “extra” d'estructurar el codi desde zero.

### 1.2.2. Sprint 2 Backlog

Èpica		Història d'usuari			Tasques	Front/ Back
#	Títol	#	Títol	Punts	#num_tasca: Nom tasca frontend	F
					#num_tasca: Nom tasca backend	B
#5	Altres Funcionalitats	#170	Servei de llista esdeveniments	10	#171: Afegir endpoint a la API per a rebre esdeveniments en un radi d'amplitud i un rang de dades	B
		#64	Dades actualitzades	13.5	#95: Crida periòdica a Dades Obertes (sync)	B
#1	Cerca i consulta esdeveniments	#14	Cercar esdeveniments per nom	4.5	#99: Filtratge per nom	B
					#106: Mostrar events del filtratge	F
		#16	Cercar esdeveniments filtrant per distància	9.5	#157: Afegir endpoint a la API per a rebre esdeveniments en un radi d'amplitud	B
		#11	Consultar la ubicació dels esdeveniments al mapa	11	#93: Mostrar ubicació al mapa	F
					#144: Afegir endpoint a la API per a rebre esdeveniments en un radi d'amplitud	B
		#9	Consultar esdeveniments recomanats per proximitat	10	#143: Afegir endpoint a la API per a rebre esdeveniments en un radi d'amplitud	B
		#15	Cercar esdeveniments filtrant per categoria	10	#109: Mostrar esdeveniments filtrats	F

					#121: Crear endpoint API amb paràmetres per filtrar	B
					#122: Crear endpoint per llistar possibles tags	B
		#8	Consultar esdeveniments recomanats per categories	10	#142: Desenvolupar algorisme de recomanació	B
		#17	Cercar esdeveniments filtrant per rang de dates	8	#101: Filtratge per data (API)	B
					#124: Afegir Selector de dates	F
					#132: Crear vista de filtres	F
#3	Gestió Usuari	#27	Crear Usuari	9	#98: Crear vista creació d'usuari	F
					#100 Implementar formulari inicial	F
					#102: Implementar formulari d'interessos	F
					#133: Guardar usuari a la base de dades	B
		#41	Iniciar Sessió	8	#91: Autenticar usuari (API)	B
					#90: Enviar credencials i rebre resposta	F
					#89: Crear vista d'iniciar sessió	F
					#103: Guardar usuari a la base de dades	B
		#51	Iniciar Sessió amb Google	11	#158: Afegir botó per iniciar sessió amb Google	F
					#159: Autenticació amb API de Google	B

		#42	Tancar Sessió	6	#94: Crear botó/event/vista per tancar sessió	F
#63	Incidències	#39	Reportar error d'esdeveniment	13	#84 Creació de la taula d'incidències (BD + API)	B
					#107: Crear vista d'incidència	F
#3	Gestió Usuari	#29	Consultar el meu perfil	8.5	#160: Afegir vista per consultar perfil	F
					#161: Afegir endpoint a la API per consultar les dades del perfil	B
		#42	Tancar Sessió	6	#94: Crear botó/event/vista per tancar sessió	F
		#28	Eliminar Usuari	7	#162: Crear botó per eliminar usuari	F
					#163: Afegir endpoint a la API per eliminar usuari	B
		#30	Editar els meus interessos al meu perfil	9	#141: Refactoritzar els usuaris per a afegir categories d'interés	B
		#56	Modificar foto de perfil	6.5	#164: Afegir vista per modificar foto de perfil	F
					#165: Afegir nova foto de perfil a la BD	B
		#57	Canviar nom perfil	7	#166: Afegir vista per canviar nom perfil	F
					#167: Canviar el nom perfil a la BD	B
		#52	Canviar contrasenya	7	#168: Afegir vista per canviar contrasenya	F
					#169: Canviar la contrasenya de l'usuari a la BD	B
#4	Interacció	#35	Apuntar-me a un	5.5	#154: Crear una taula amb assistències	B

	usuari-esdeveniments		esdeveniment		#155: Afegir endpoint a la API per a apuntar un usuari a un esdeveniment	B
					#156: Afegir endpoint a la API per a desapuntar un usuari d'un esdeveniment	B
		#32	Afegir un esdeveniment a la llista de favorits	6.5	#153: Afegir endpoint a la API per a afegir un esdeveniment a una llista	B
		#37	Puntuar un esdeveniment	6.5	#151: Afegir endpoint a la API per a valorar un esdeveniment	B
		#38	Esborrar la meva puntuació d'un esdeveniment	6.5	#152: Afegir endpoint a la API per a esborrar una valoració	B
		#43	Crear llista	6.5	#149: Afegir llistes a la BD	B
		#45	Afegir esdeveniment a la llista	5.5	#150: Afegir endpoint a la API per a afegir un esdeveniment a una llista	B
#63	Incidències	#54	Iniciar sessió a la pàgina d'administració	10.5	#138: Afegir el formulari d'iniciar sessió	B
					#139: Refactoritzar els usuaris per a afegir administradors	B
					#140: Afegir endpoint a la API per a autenticació a la pàgina d'administradors	B
		#58	Tancar sessió a la pàgina d'administració	6.5	#137: Afegir el botó de tancar sessió	B
		#49	Veure incidències obertes/tancades	6.5	#148: Crear pagina estàtica amb les incidències	B

		#48	Veure totes les incidències	6.5	#81: Crear pagina estatica amb les incidències	B
		#50	Cambiar l'estat d'una incidència	9.5	#147: Afegir el formulari de la incidència	B
		#53	Respondre incidència	9.5	#146: Afegir el formulari de la incidència	B
-	-	-	Storyless Tasks	-	#97: Implementar la navegabilitat entre pantalles	F
					#111: Implementar la infraestructura necessària per a convertir etiquetes/títols de la UI a strings	F
					#127: Infraestructura MVVM + Repository	F
					#135: Implementar API Repository	F
					#145: Desenvolupar el servei a oferir	B
					#174: Afegir esquelet per al tractament d'excepcions	B
					#175: Afegir tractament d'excepcions per al controlador d'usuaris	B
					#177: Afegir tractament d'excepcions per al controlador d'assistències	B
					#183: Afegir filtrat avançat per a les incidències	B

					#200: Securitzar crida d'incidències amb API Token	B
					#145: Desenvolupar servei a oferir	B
					#189: Afegir DTOs per a les assistències	B
					#176: Afegir tractament d'excepcions per al controlador d'incidències	B
					#188: Afegir DTOs per a les incidències	B
					#173: Afegir seguretat a la API	B
					#178: Afegir tractament d'excepcions per al controlador d'esdeveniments	B

### 1.2.3. Sprint 3 Backlog

Encara estem pendents de fer l'Sprint Planning per aquest tercer sprint



## 1.4. Requisits no funcionals

En aquesta secció es descriu el conjunt de requisits no funcionals que conté el nostre sistema, és a dir, tots aquests que mencionen i expliquen les propietats del nostre producte, sense posar èmfasi en les funcions principals que ha de poder fer la nostra aplicació. Aquests requisits estan identificats per un número i segueixen la classificació utilitzada en la plantilla de *Volere*:

### 1.5.1 Requisits de rendiment

<b>Número</b>	<b>1</b>
<b>Tipus de requisit</b>	15a. Requisits de longevitat
<b>Descripció</b>	El producte s'ha de poder mantenir funcionalment en qualsevol moment.
<b>Justificació del requisit</b>	Cal tenir de manera activa el sistema durant un gran període de temps per tal de poder assolir els objectius proposats per <i>CultureFinder</i> .
<b>Condicció de satisfacció</b>	❖ L'aplicació romandrà funcional i activa fins al darrer dia en el qual acabi el projecte de l'assignatura.

### 1.5.2 Requisits de preservació i suport

<b>Número</b>	<b>2</b>
<b>Tipus de requisit</b>	14b. Requisits de suport
<b>Descripció</b>	Especifica el nivell de suport que el sistema necessita.
<b>Justificació del requisit</b>	Cal tenir la seguretat de que el nostre sistema proporciona suport als usuaris quan troben problemes relacionats amb l'aplicació.
<b>Condicció de satisfacció</b>	❖ S'ofereix la possibilitat d'afegir incidències relacionades amb els esdeveniments de l'aplicació.

### 1.5.3 Requisits de capacitat d'ús i humanitat

<b>Número</b>	<b>3</b>
<b>Tipus de requisit</b>	11a. Requisits de facilitat d'ús
<b>Descripció</b>	L'aplicació ha de ser fàcil d'usar i de navegar per a qualsevol usuari, independentment del seu nivell d'experiència o coneixements tècnics. Ha d'estar dissenyada de manera clara i intuïtiva, amb una interfície d'usuari simple i fàcil d'entendre.
<b>Justificació del requisit</b>	La facilitat d'ús és un requisit important per a qualsevol aplicació, ja que pot afectar directament la satisfacció de l'usuari i l'adopció de l'aplicació. Si l'aplicació és difícil d'usar o de navegar, els usuaris poden sentir frustració i no usar-la, la qual cosa pot afectar negativament l'èxit del projecte.
<b>Condicció de satisfacció</b>	❖ El percentatge d'usuaris que consideren l'aplicació fàcil d'usar i navegar ha de ser superior al 80%. A més, qualsevol problema d'usabilitat que es detecti durant les proves ha de ser corregit abans del llançament de l'aplicació.

<b>Número</b>	<b>4</b>
<b>Tipus de requisit</b>	11b. Requisits de personalització i internacionalització
<b>Descripció</b>	Especifica com l'aplicació pot modificar-se per adaptar-se a les necessitats específiques dels usuaris. Això significa que els usuaris han de poder personalitzar l'aplicació d'acord amb les seves preferències personals, i ha de poder adaptar-se a diferents idiomes i cultures.
<b>Justificació del requisit</b>	Cal tenir clara la intenció de ser utilitzada en diferents mercats i per diferents usuaris amb diferents necessitats i

	preferències. Si l'aplicació no es pot personalitzar o adaptar a diferents idiomes i cultures, els usuaris poden sentir-se alienats o rebutjar l'aplicació per complet.
<b>Condició de satisfacció</b>	❖ S'ofereix la possibilitat d'afegir incidències relacionades amb els esdeveniments de l'aplicació.

<b>Número</b>	<b>6</b>
<b>Tipus de requisit</b>	11c. Requisits d'aprenentatge
<b>Descripció</b>	Especifica el nivell de facilitat que ha de tenir l'aplicació per poder ser utilitzada.
<b>Justificació del requisit</b>	És de vital importància que als usuaris de la nostra aplicació no els hi sigui complicat aprendre el seu funcionament, és a dir, ha de ser intuïtiva. Tanmateix, podrien frustrar-se i perdre l'interès en aquesta.
<b>Condició de satisfacció</b>	❖ No es requereix cap curs ni tutorial previ per tal de fer ús de la nostra aplicació.

## 1.5. Tractament dels aspectes transversals

Sprint 1:

**Geolocalització:** Aquest aspecte transversal no ha estat implementat al acabar l'Sprint 1.

**Xarxes socials:** Aquest aspecte transversal no ha estat implementat al acabar l'Sprint 1.

**Xat:** No tenim pensat implementar aquest aspecte transversal en la nostra aplicació.

**Gamificació:** Aquest aspecte transversal no ha estat implementat al acabar l'Sprint 1.

**Stakeholders reals:** Un cop finalitzat el primer sprint vam fer una petita demostració a un familiar, però ens vam adonar que encara quedaven bastants funcionalitats per ser usable de manera adequada.

**Refutació:** Aquest aspecte transversal no ha estat implementat al acabar l'Sprint 1.

**Calendari:** El calendari encara no mostra del tot bé els esdeveniments.

**Web-app admin:** Aquest aspecte transversal no ha estat implementat al acabar l'Sprint 1.

**Multiidioma:** Aquest aspecte transversal no ha estat implementat al acabar l'Sprint 1.

Aspecte transversal	Estatus actual
Geolocalització	×
Xarxes socials	×
Xat	×
Gamificació	×
Stakeholders reals	✓
Refutació	×
Calendari	×
Web-app admin	×
Multiidioma	×

Sprint 2:

**Geolocalització:** En aquest segon sprint hem estat implementat aquest aspecte tenint en compte totes les històries d'usuari relacionades amb l'ubicació del client que utilitza l'aplicació fent ús del mapa.

**Xarxes socials:** Aquest aspecte transversal no ha estat implementat al acabar l'Sprint 2.

**Xat:** No tenim pensat implementar aquest aspecte transversal en la nostra aplicació.

**Gamificació:** Les valoracions estan implementades correctament a data de fi de l'Sprint 2.

**Stakeholders reals:** Un cop finalitzat el segon sprint farem una demostració a un familiar o amic, per tal de rebre feedback en aspectes com usabilitat o rendiment.

**Refutació:** Aquest aspecte transversal no ha estat implementat al acabar l'Sprint 2.

**Calendari:** S'ha implementat un calendari per tal de tenir una de les funcionalitats principals del nostre projecte que és poder cercar per filtratge i calendari els esdeveniments de l'aplicació.

**Web-app admin:** En aquest segon sprint s'ha començat a implementar la pàgina web d'administració fent la interfície principal, tenint en compte les crides a l'API que hem anat implementat relacionades amb incidències.




**Multiidioma:** Aquest aspecte transversal no ha estat implementat al acabar l'Sprint 2.

Aspecte transversal	Estatus actual
Geolocalització	✓
Xarxes socials	✗
Xat	✗
Gamificació	✓
Stakeholders reals	✓
Refutació	✗
Calendari	✓
Web-app admin	✓
Multiidioma	✗

Sprint 3:

Pendent d'actualitzar quan arribem al tercer sprint :

Aspecte transversal	Estatus actual
Geolocalització	✓
Xarxes socials	✓
Xat	✗
Gamificació	✓
Stakeholders reals	✓
Refutació	✗

Calendari	
Web-app admin	
Multiidioma	

## 1.6. Serveis de tercers

### 1.6.1 Repartiment de funcionalitats entre els equips

Durant la segona fase de l'incepció, vam realitzar una primera toma de contacte en l'àmbit de negoci entre els diferents equips per tal de conèixer de primera mà quins projectes tenen pensats i de quina manera tindran impacte en el nostre producte.

Primerament, ens van comentar que havíem de rebre i donar funcionalitats destacables amb la resta de grups, per tant, vam estudiar quines opcions eren les més interessants, entre les que proporcionaven els companys, de cara a afegir-les a la nostra aplicació.

#### **Mobilitat sostenible**

D'aquesta manera, amb l'objectiu de tenir el millor producte possible ens vam centrar en quines propostes ens interessaven enviar i quines rebre, i així ho vam fer. Com a la nostra aplicació el mapa serà una de les funcionalitats més principals, vam estar negociant amb el grup de Mobilitat Sostenible i es va arribar a la conclusió que aquest grup ens oferirà la possibilitat d'integrar els punts de càrrega al nostre mapa interactiu. Gràcies a això la nostra aplicació tindrà més valor, ja que a més de poder cercar esdeveniments, l'usuari podrà tenir coneixement dels punts de càrrega que estan al voltant del seu entorn.

Així doncs, la funcionalitat que rebem per part del grup de Mobilitat Sostenible és la següent:

- Donada una ubicació concreta, en el mapa es podran mostrar les estacions de càrrega per a vehicles elèctrics a Catalunya.

#### **Licitacions i adjudicacions en curs**

En relació amb la negociació amb el grup de licitacions i adjudicacions en curs, tot va començar comentant amb la gestora dels serveis de l'altra grup quines funcionalitats tenien present i els hi podien ser d'utilitat. La primera idea que va sorgir va ser relacionada amb el mapa, però com és una funcionalitat fonamental pels dos productes, es va descartar ràpidament. Addicionalment, nosaltres vam aportar la idea de proporcionar el calendari per tal de poder observar segons alguns filtres de dates les contractacions públiques de Catalunya.

Com a conclusió, la funcionalitat que proporcionem al grup de licitacions és la següent:

- Donat un conjunt de dades relacionades amb les contractacions públiques, oferir la funcionalitat de calendari perquè es pugui adaptar a les crides de la seva API.

## 1.6.2 Comunicació amb la resta dels equips

Pel que fa a la interacció amb els altres equips per als serveis que haurem d'implementar i oferir entre nosaltres l'estem gestionant a través de dos canals. Per una banda, el nostre responsable aprofita les hores dels dimarts i els dijous per anar als grups i preguntar al grup sencer quines funcionalitats estan buscant o en el cas del servei que hem d'integrar, debatre i explicar els nostres requisits amb relació al que ens poden oferir.

Per l'altra banda, també disposem d'un grup de whatsapp amb tots els membres dels dos equips per tal de facilitar la comunicació durant la resta de la setmana. Els missatges d'aquest grup també serveixen com a punt de referència en sobre quines funcionalitats s'han pactat a l'hora d'implementar el servei i per resoldre qualsevol problema o incongruència que hi pogués haver.

## 2. Metodologia

Individual description of all methodological dimensions at an adequate level of detail (provide demos during the course and evidence in this screenshot)

Use significant events (new sprint, close tasks, ...) to articulate the description

### 2.1. Visió General

Per a organitzar-nos hem decidit seguir la metodologia agile *Scrum*. Seguint aquesta metodologia, hem dividit el projecte en dues fases d'*Inception* per definir, entre altres coses, les diferents funcionalitats que tindrà la nostra aplicació i tres *Sprints* de tres setmanes. Cadascun dels Sprints té associat un membre com a '*Sprint master*' que serà l'encarregat de dirigir les retrospectives i reunions setmanals, verificar les tasques del Project Record Track, entre altres responsabilitats. També tenim a un membre dedicat a relacionar-se amb la resta de grups de cara al servei que haurem d'integrar i oferir a un dels grups.

Actualment, tenim dedicats 3 dies per a reunir-nos, les classes de dimarts i dijous, i els diumenges a la tarda per tal de plantejar les tasques de la setmana vinent o per fer les retrospectives. Com a grup ens comuniquem mitjançant un grup de Whatsapp per notificar-nos sobre canvis recents o importants que afectin el grup en general i un servidor de Discord amb canals de veu per fer les reunions, canals de text per



enviar-nos recursos que poden ser d'utilitat (#link), canals dedicats per les reunions o per enviar-nos referències (mockups, aplicacions semblants, etc.) que ens puguin ajudar.

Finalment, per a les històries d'usuari estem fent servir Taiga, allà tenim definits els criteris d'acceptació de cada història, les tasques associades i els *story points* que té assignats, desglossat en Front, Back i UX.

En els següents apartats, entrarem més en detall sobre les tecnologies i metodologies escollides per al projecte.

## 2.2. Gestió Projecte

Com s'ha esmentat anteriorment, la principal eina de gestió que fem servir és Taiga, la qual ens permet definir històries d'usuari i agrupar-les en èpiques. A la descripció de les històries és on tenim definit els diferents criteris d'acceptació de cada història i el seu pes en *story points* desglossat en diferents apartats.

Aquests *story points* no són necessàriament fixes i som conscients que hi ha tasques que a priori haurem puntuat malament, sigui a l'alta o a la baixa. La idea es anar refinant aquesta puntuació després de cada sprint amb les reunions retrospectives, per tal de distribuir-nos la feina de forma equitativa.

Abans d'assignar els *story points* a cada tasca vam concretar que un *story point* equivaldria a una hora de treball, aquesta hora no inclou el temps de formació directament, però, sí que té en compte la complexitat de la tasca i, per tant, la necessitat de formació.

Amb la definició del valor d'un *story point*, i amb les històries d'usuari definides, vam realitzar un *planning poker* amb tot l'equip per tal de decidir el valor final de cada tasca. Per acabar de decidir, els membres amb els valors més alts i més petits de cada tasca van explicar el perquè havien decidit aquests valors, i conjuntament vam decidir la puntuació final, tenint en compte la mitja que ens va donar el *planning poker* per a cada tasca.

A part del Taiga, també estem fent servir el Trello per poder assignar tasques que no tenen una relació directa amb les històries d'usuari. Això ens permet veure quines tasques queden pendents fora de l'àmbit del desenvolupament, en particular en aquestes primeres fases d'inception.

Finalment, hem acordat el *Definition of Done (DoD)* i el cicle de vida de cada tasca. El DoD de cada història l'hem definit en 5 fases, les quals definim més endavant a partir d'un problema que vam tenir al primer sprint.

Per qüestions de temps hem decidit no fer *testing*. Sabem i considerem la importància d'aquesta pràctica a l'hora de produir *software* de qualitat, però no hem trobat la manera d'incorporar-lo al nostre projecte per aquestes raons.

Un dels aspectes discutits a la primera reunió de retrospectiva va ser la ambigua Definition of Done que havíem definit anteriorment tant per les històries d'usuari com les tasques que les formen. Per aquest motiu, vam tornar a redefinir el nostre concepte de DoD:

- **[New]** - Història d'usuari creada, en aquest estat no té encara cap criteri d'acceptació.
- **[Ready]** - La història d'usuari té definits un conjunt de criteris d'acceptació. Aquests criteris defineixen, en part, quan aquesta història passa a estar acabada.
- **[In Progress]** - Història d'usuari ha estat assignada a un sprint.
- **[Closed]** - La història d'usuari (i per tant les seves tasques) està implementada i funciona correctament.

En quant a les *fases de cada tasca*, el canvi principal que hem incorporat al primer sprint és l'eliminació de la fase intermitja de Ready for Test.

- **[New]** - Nova tasca creada al sprint
- **[In Progress]** - La tasca ha estat assignada a un membre de l'equip
- **[Undocumented]** - Un cop en aquesta fase, el codi que pertany a la tasca ha estat implementat i no hauria de crear problemes a la hora de compilar (dintre de la seva branca de feature)
- **[Closed]** - La tasca ha estat implementada i documentada.
  - En el nostre cas, el tema de documentar és flexible. La documentació hauria d'ajudar als altres membres de l'equip a entendre quina funció té el codi/classe/mètode i no cal que sigui exhaustiva, tot i que si que vam definir una estructura per aquells mètodes que potser requereixen més informació, sobretot si formen part d'un component que farà servir la resta de l'equip.

## 2.3. Gestió versions

Per tal de gestionar les versions del codi, hem decidit que farem servir GitHub com a servidor a on guardar els repositoris. Hem creat l'organització PES-Agenda-Cultural, aquesta organització té tres repositoris, un per al codi del frontend, altre per al codi del backend i un últim afegit a l'*Sprint 2* per al codi de la pàgina web d'administració.

Per facilitar el procés de generar releases i la resolució de possibles conflictes a l'hora de fer merge entre branques, hem decidit definir dos nous rols, Lead Backend i Lead Frontend. Les seves responsabilitats respecte a la gestió de versions consistirà principalment a solucionar conflictes a l'hora de fer merge entre les branques i mantenir els convenis estipulats més endavant i de cara al primer sprint, la inicialització i creació.

La gestió dels dos repositoris seguirà la mateixa metodologia de gestió de versió, el model de control de branques anomenat **Gitflow**, però, amb una petita variació. Aquest model consisteix en dues branques principals, una branca main amb el codi de **producció** i una segona branca de **desenvolupament** amb el codi estable més recent.

La branca de **principal (main)** conté el codi preparat per producció i serveix com a branca històrica on cada commit representa una nova versió, principal o intermèdia per arreglar bugs que no haguem captat durant el desenvolupament. En el cas del back end, aquesta és la branca que clonarà el servidor. Qualsevol canvi efectuat en aquesta branca, s'haurà de replicar a la branca de dev (merge main -> dev), principalment en el cas que es trobés un bug un cop fet el merge.

La branca de **desenvolupament (dev)** mantindrà el codi estable més recent. Aquesta branca serà la branca de la qual es bifurquen les altres branques que explicarem a continuació. Un cop finalitzat un sprint, el Lead Backend i Lead Front End seran els encarregats de fer/solucionar el pull request de **dev -> main** dels seus repositoris respectius.

En quant a la variació de Gitflow, en comptes de fer servir branques '**feature**' per a cada història d'usuari, farem servir branques '**task**'. Tot el desenvolupament relacionat a una tasca es farà dins d'aquesta branca.

El nom de les noves branques seguirà el següent conveni:

- <# num\_tasca>:<nom\_tasca>  
Exemple: #99:Filtratge\_per\_nom

On *num\_tasca* i *nom\_tasca* fan referència al número al títol que té la tasca al Taiga respectivament.

Aquestes branques es crearan a partir de la branca *dev* i un cop acabades es farà un pull request a la branca *dev* (**task -> dev**).

Tot i que les històries d'usuari han estat redactades per tal de maximitzar la seva independència, certes funcionalitats requereixen d'una certa base per tal de funcionar correctament, si aquesta funcionalitat és present en una altra branca de *task*, hem decidit que per tal d'agilitzar el desenvolupament, podem crear branques temporals en les que treballar amb les dos *features* inacabades. Idealment, aquesta branca només existeix per agafar codi d'una tasca o història d'usuari del qual la implementació ja està acabada. En qualsevol cas, aquestes branques es tancaran un cop acabada la integració.

Pel que fa a la correcció d'errors, farem servir una metodologia semblant a les branques '*task*'; creant una nova branca '**issue**' per a cada error que trobem. A diferència de les històries d'usuari, farem servir l'apartat d'issues del mateix Github per especificar els detalls de cada bug. A diferència de les branques '*task*' aquestes noves branques es poden crear a partir de **main** si es trobés l'error després de fer el release, en aquest cas, un cop solucionat el problema, la branca tornaria a ajuntar-se amb main. Aquestes branques han de contenir els canvis mínims per solucionar el problema.

El nom d'aquestes branques farà servir la següent convenció:

- issue/<#num\_issue>:<títol>  
Exemple: issue/#123:Error\_Login\_Google

## · Commits

Finalment, els comentaris dels commits a les branques de 'task' tindran un missatge clar i entenedor de l'estat d'aquella mateixa tasca, com per exemple:

- "Tasca finalitzada"
- o
- "Tasca en procés. Queda afegir el codi del repositori."

## · Pull Requests (PR)

Un altre procés que va sorgir de forma espontània al primer sprint, però que volem incorporar a les nostres convencions de manera continuada, és l'ús dels *pull requests* a l'hora d'ajuntar les diferents branques/features amb la branca de producció (**dev**) o **main** en acabar l'sprint corresponent.

Amb aquest procés assignarem un reviewer (o més) per a comprovar quins són els canvis que hi entren i resoldre els conflictes del merge, només en el cas de fer un pull request de **dev** a **main**. Els *pull requests* de branques relacionades amb desenvolupament de tasques a **dev** no caldrà assignar cap reviewer.

Els pull requests també ens permeten assignar-hi accions a executar un cop acceptada la request que executi de forma automàtica un conjunt de test per tal de garantir que el merge no ha trencat alguna altra funcionalitat.

## 2.4. Comunicació d'equip CultureFinder

Amb l'objectiu de tenir la millor comunicació possible, tenim reunions presencials a classe els dimarts i dijous de 8:00 a 10:00. A més, com s'ha esmentat anteriorment, tenim dos canals de comunicació. Un grup de Whatsapp per notificar a tot l'equip sobre novetats, canvis, enviar petits missatges o concretar hores de reunió i un servidor de Discord amb diferents canals de text especialitzats per enviar enllaços d'interès, parlar de Backend, Frontend, fer sessions de pluja d'idees sense haver de notificar a tot l'equip.

Apart de les hores de classe, també quedem cada diumenge per la tarda per fer una pre-reunió per parlar sobre el que s'ha fet durant la setmana, com van les tasques de cadascú i quins objectius tenim de cara a la setmana vinent. Aquestes reunions tendeixen a ser amb tot l'equip, per normalment ens acabem dividint en canals separats per tal de parlar sobre els aspectes més tècnics i ajudar-nos mútuament amb possibles problemes que haguem trobat.

Per aquestes reunions fem servir el servidor de Discord on tenim canals de veu que ens permeten compartir la pantalla. Canals de text per al front, el back, reunions, millores i consells que ens dona el profe entre d'altres.

## 2.5. Frameworks i llenguatges

Com s'ha esmentat anteriorment, el nostre codi està dividit en dues parts, el *frontend* i el *backend*, aquests tenen els seus llenguatges, frameworks i repositoris diferents que s'expliquen a continuació.

Per la part de Backend de la nostra aplicació utilitzarem el framework Spring Boot. Hi ha diversos motius pels quals hem escollit Spring Boot:

- **Flexibilitat:** Spring inclou un set d'extensions i llibreries per ajudar a desenvolupar qualsevol tipus d'aplicació.
- **Rapidesa:** Spring facilita entre altres coses la creació del projecte permetent inicialitzar un nou projecte de Spring en segons amb l'ajuda del 'Spring Initializer'.
- **Support:** Spring té una comunitat global amb tutorials des de principiants fins a professionals. En conseqüència, no serà difícil buscar material d'ajuda.
- Spring està a tot arreu. Spring ha estat emprat per companyies com Amazon, Google o Microsoft i, per tant, és un al·licient a aprendre a fer-lo servir.

Conjuntament amb Spring Boot farem servir **Java** com a llenguatge de programació. Java és un llenguatge de programació dissenyat el 1990 per James Gosling amb altres companys de Sun Microsystems a partir del llenguatge C. Des del seu naixement fou pensat com un llenguatge orientat a objectes. Tot i això, a l'inici vam decidir utilitzar Kotlin, un llenguatge similar a Java amb el qual tot l'equip està familiaritzat, però està més enfocat al desenvolupament d'aplicacions Android.

El principal motiu pel qual hem modificat el llenguatge del Backend a Java és perquè ens vam trobar en la situació en la qual no trobàvem molta documentació relacionada amb Kotlin i tots els membres de l'equip del Backend estàvem molt més familiaritzats amb Java.

Per la part del frontend farem servir el framework desenvolupat per Google, Flutter. L'aplicació mòbil estarà programada per a dispositius Android tot i que Flutter com a framework permet crear aplicacions multiplataforma, he decidit que no donarem suport per a dispositius iOS, ja que per programar-los és indispensable tenir un Mac i cap membre de l'equip en té.

Respecte al perquè de Flutter, es redueix a dos motius principals. Per una banda, vam veure que en l'àmbit del desenvolupament mòbil, Flutter és un dels llenguatges més populars i més demanats per diferents empreses i anuncis de feina, i com a grup vam concretar que posats a aprendre un nou framework i llenguatge, estaria bé aprendre'n un que ens obri més portes en un futur. Tot i ser prou nou com a framework, la seva documentació és robusta i hem trobat molts recursos per ajudar-nos a aprendre'l al llarg del projecte, entre aquests recursos destaca un curs introductori gratuït a YouTube de més de 37h de duració.

Una altra gran raó per la qual vam escollir Flutter és el llenguatge que fa servir, Dart. També va estar desenvolupat per Google, originalment com a alternativa a JavaScript pel

desenvolupament de pàgines web, el llenguatge no va tenir molt d'èxit fins a la creació de Flutter uns anys més tard.

Tot i que cap membre de l'equip d'experiència prèvia amb Dart, un gran avantatge del llenguatge és la seva sintaxi, la qual és molt semblant a Java i JavaScript, i finalment, tot i ser relativament nou com a llenguatge, Google ofereix molta documentació sobre com funciona Dart, les seves similituds a Java i algun dels aspectes únics en relació amb el desenvolupament per a Android.

A més del *frontend* i el *backend* també contem amb una pàgina web on els usuaris que son administrados poden gestionar les incidències que crean els usuaris de la nostra aplicació.

Les tecnologies utilitzades a aquesta pàgina web son HTML, Javascript i CSS. A més fem servir l'arquitectura model-view-controller que separa la lògica de l'aplicació en tres components: el model, la vista i el controlador. El model representa les dades i la lògica de de l'aplicació, la vista és la interfície d'usuari i el controlador maneja les interaccions entre el model i la vista.

HTML (Hypertext Markup Language) és el llenguatge de marcatge estàndard utilitzat per crear l'estructura i el contingut d'una pàgina web. És molt utilitzat en el desenvolupament d'aplicacions web a causa de la seva facilitat d'ús i el seu ampli suport pels navegadors web.

JavaScript, d'altra banda, és un llenguatge de programació dinàmic que s'utilitza per afegir interactivitat a les pàgines web. És molt popular en el desenvolupament web perquè pot executar-se al navegador de l'usuari i és compatible amb la majoria dels navegadors.

CSS (Cascading Style Sheets) és un llenguatge de fulls d'estil que s'utilitza per definir l'aparença visual d'una pàgina web. Permet separar la presentació d'una pàgina web del contingut, el que facilita el seu manteniment i actualització.

Hem escollit aquestes tecnologies per a seva àmplia adopció en el desenvolupament web, la seva facilitat d'ús i la seva capacitat per proporcionar una experiència d'usuari efectiva i eficient.

## 2.6. IDEs

Per facilitar el desenvolupament, hem escollit dos IDEs (entorns de desenvolupament integrat) que ens proporcionen plugins per ressaltar errors en el codi, breakpoints per debugar, snippets per agilitzar el desenvolupament, i la més important, un entorn consistent per als desenvolupadors. D'aquesta manera podem evitar problemes amb versions inconsistentes de llibreries o altres dependències. Un altre avantatge del IDEs escollits és el fet que contenen eines per a compilar i executar el codi amb les seves dependències sense haver de fer tota la configuració de forma manual.

Per al frontend, el framework de Flutter requereix Android Studio per descarregar i instal·lar el SDK de Android, per això hem decidit fer servir Android Studio com a IDE, el qual té un plugin per desenvolupar aplicacions amb Dart.

Per al backend, farem servir l'IDE IntelliJ de JetBrains. Com ja hem estipulat anteriorment, aquest IDE ens permet ancorar una versió específica del JDK al projecte, evitat així possibles problemes amb versions antigues presents en els ordinadors dels desenvolupadors.

## 2.7. Gestió de qualitat

Per a provar la qualitat del nostre *software* utilitzarem les eines que ens ofereixen els nostres IDE's.

Per la part de Frontend utilitzarem Android Studio que inclou un emulador d'Android. Aquest emulador ens permet comprovar la interfície en diferents dispositius amb diferents dimensions de pantalla. També podem falsificar les coordenades del GPS per comprovar les funcions de geolocalització, rotar la pantalla... d'aquesta manera podem comprovar que la nostra aplicació funcioni correctament a tots els dispositius.

Android Studio també ofereix la possibilitat de connectar un dispositiu per executar la nostra aplicació. Això permet comprovar coses que no permet l'emulació com l'experiència d'usuari i la velocitat de resposta. També hem trobat l'eina ['screp'](#) per mostrar i controlar el nostre dispositiu Android a través del nostre ordinador.

També usarem linters per ajudar a detectar errors de programació comuns, estilístics i per estandarditzar el codi, en particular al Frontend, ja que Dart té un linter integrat.

Més enllà de les eines que ens ofereixen els editors, també hem concretat que per a les funcionalitats més crítiques programarem *smoke tests*. Aquests tests no pretenen ser exhaustius, la idea dels *smoke tests* consisteix en crear tests bàsics per als punts o funcionalitats més crítiques de la nostra aplicació, per tal de detectar aquests casos el més ràpid possible.

## 2.8. Interacció amb altres grups

Pel que fa a la interacció amb els altres equips per als serveis que haurem d'implementar i oferir entre nosaltres l'estem gestionant a través de dos canals. Per una banda, el nostre responsable aprofita les hores dels dimarts i els dijous per anar als grups i preguntar al grup sencer quines funcionalitats estan buscant o en el cas del servei que hem d'integrar, debatre i explicar els nostres requisits amb relació al que ens poden oferir.

Per l'altra banda, també disposem d'un grup de whatsapp amb tots els membres dels dos equips per tal de facilitar la comunicació durant la resta de la setmana. Els missatges d'aquest grup també serveixen com a punt de referència en sobre quines funcionalitats s'han pactat a l'hora d'implementar el servei i per resoldre qualsevol problema o incongruència que hi pogués haver.

## 2.9. Convencions de codi

Hem decidit que no tindrem convencions concretes a la hora d'escriure el codi ja que els llenguatges que utilitzem tenen les seves pròpies convencions i bones pràctiques a l'hora d'escriure codi. En canvi, aquestes convencions son més una sugerencia i al final els estils dels dos codebases seran el més consistens possibles de forma independent. És a dir, la consistència ens mantindrà entre el codi del front i per altra banda, el codi del back

A continuació deixem explicades aquestes pràctiques.

### 2.9.1. Variables

Les variables estaran majoritàriament escrites en *camelCase* i s'intentarà que siguin tan descriptives com sigui possible evitant abreviacions i lletres úniques. Les úniques excepcions a aquesta norma seran els iteradors de bucles o índexs per a arrays. Per les assignacions, deixarem un espai entre el nom, el símbol '=' i l'assignació.

El nom de les variables constants s'escriuran en majúscules amb '\_' indicant els espais (en el cas de ser més d'una paraula).

EX: **String** API\_ROOT\_URL = 'www.exemple.com/api/';

### 2.9.2. Funcions i mètodes

Respecte a les funcions, per norma general serà semblant al de les variables, *camelCase* i amb noms en anglès els més descriptius possibles, i en cas de fer una acció, el nom de la funció haurà de començar amb l'acció (verb).

Per exemple: *getAllEvents()*;

Per ajudar-nos a entendre el codi de cada company, procurarem documentar les funcions de la següent manera:

- **Descripció** breu del que fa la funció o perquè existeix/s'utilitza.
- **<nom paràmetre>** : Si no fos prou evident, una petita descripció i/o consideracions a tenir en compte (si el valor es modifica, si pot ser Null, rang de valors min/max, etc)
- **<excepcions>** : Si no fos prou evident, perquè salten i si són crítiques o no.
- **Resultat**: Descripció del quin és el resultat i consideracions a tenir en compte (semblant als paràmetres, si el resultat pot ser Null, si té resultats que denominen un estat d'error (ex: un -1), etc...

Les funcions més trivials com els getters o els setters no caldrà documentar-les. També cal remarcar que no serem molt estrictes a l'hora de documentar el codi, ja que considerem que en aquest context, és una eina per ajudar-nos a entendre el codi dels altres, i en el cas que tothom entengués una funció/mètode/classe no caldrà necessàriament documentar-la. Tot i això, la idea és documentar (lleugerament) el màxim de codi possible.



### 2.9.3. Classes

El nom de les classes s'escriurà fent servir UpperCamelCase on la primera lletra de cada paraula (inclosa la primera) sigui majúscula. El nom de la classe haurà de ser el mateix que el del fitxer que el conté, excepte en el cas que un fitxer contingui més d'una classe on el nom del fitxer serà el de la classe més important que contingui, o si les classes formen part d'una jerarquia, el nom del fitxer serà el de la superclasse. Si la classe existeix com a element d'un patró s'intentarà que el nom reflecteixi el patró que s'està emprant. EX: **class** UserFactory

L'ordre dels elements de les classes serà el següent:

1. Propietats (variables d'una classe)
2. Constructora
3. Destructora (si calgués crear-ne una específica)
4. Getters
5. Setters
6. Altres mètodes públics
7. Altres mètodes privats

En el cas del backend, les classes relacionades amb els endpoints de la API, seguiran el següent ordre:

1. Get
2. Post
3. Put
4. Delete

### 2.9.4. Fitxers i directoris

Sobre els fitxers (que no són classes) i els packages farem servir camelCase

**Ex:** unDirectorio/unAltreDirectorio/apiKeys.json

Per a qualsevol altre element o en cas que no s'hagi estipulat en aquest apartat farem servir camelCase per tal de mantenir la consistència.

## 2.10. Gestió de bases de dades

Per la part de la base de dades farem servir PostgreSQL. La despleguem a **virtech** de la mateixa manera que el backend.

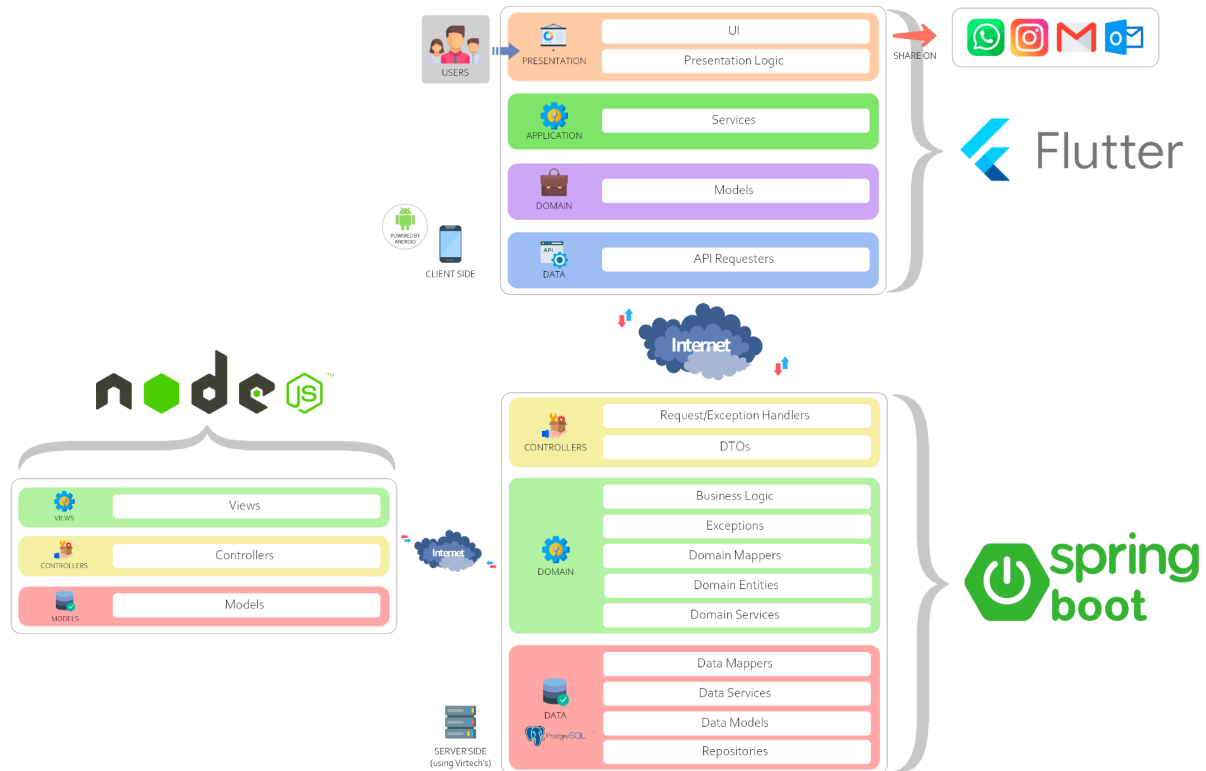
Al principi del nostre projecte teníem pensat utilitzar Amazon Web Services (AWS) per fer el deploy tant de la base de dades com de la nostra aplicació. Tanmateix, quan al primer sprint ens vam posar a configurar la base de dades i creant el compte a AWS, ens vam adonar que havia un límit d'hores d'execució si volíem utilitzar la versió "gratuïta". Per aquest mateix motiu, i gràcies a les importants facilitats que ens proporciona el servei de **virtech**, és un entorn de virtualització de màquines virtuals sota demanda. Aquest servei permet crear, configurar, modificar i esborrar servidors virtuals Linux a un grup d'estudiants. Cada grup pot configurar aquests servidors a mesura de les seves necessitats i adaptar-lo per a desenvolupar les seves pràctiques. D'aquesta manera els alumnes de la UPC disposem d'un (o varis) servidors propis en el "núvol" on es pot

programar qualsevol servei, amb l'avantatge que aquest servei està disponible des de tota internet.

Hem escollit PostgreSQL perquè és un servei amb el qual tots estem familiaritzats. A l'hora de configurar la base de dades a **virtech** tindrem en compte els permissos d'usuari i d'accés. També seleccionarem l'opció d'accés públic per tal de poder accedir des dels nostres ordinadors i no utilitzar cap mena de VPN.

### 3. Descripció tècnica

#### 3.1. Concepte de l'arquitectura



##### 3.1.1. Arquitectura física

Hem decidit fer l'aplicació dividint-la en tres: *frontend*, *backend* i *admin web*.

Pel que fa al *frontend*, fem servir una adaptació de la arquitectura en tres capes fent servir el patró de MVVM:

- **Presentació:** Aquesta capa conté principalment les classes pertanyents als elements de la UI (Widgets) i el seu estat i controladors. Aquesta capa conté les Views i alguns Notifiers (controladors).
- **Aplicació:** En aquesta capa es troben els diferents serveis encarregats de treballar amb els nostres propis models de les dades que agafem del backend. Aquesta capa i la implementació del que serien els *viewmodels* del patró no la tenim molt clara a la hora d'implementar. Però de cara als següents sprints procurarem acabar de definir-la.
- **Domini:** Aquesta capa representa la nostra versió de la estructura de dades que tenim guardades al backend. Aquí és on trobem els diferents Models que utilitzem. Hem decidit que la conversió dels DTOs (principalment en JSON) a instàncies es troba directament al Model, per tal de concentrar les funcionalitats en un punt.
- **Dades:** Aquesta capa encapsula la d'adquisició de dades per construir els components de la UI o la lògica de la capa d'aplicació.

Pel que fa al *backend*, programat utilitzant el framework Spring Boot amb Java, l'hem dividit en tres capes: Controladors, Domini i Dades. La capa dels controladors s'encarrega de proporcionar les dades al *frontend* mitjançant la nostra pròpia API, així com obtenir les dades de la API de l'Agenda Cultural mitjançant uns objectes que anomenem DTOs(Data Transfer Object).

La capa de domini conté les classes de lògica per a cada tipus de model de dades, les quals s'encarreguen de processar les dades que representen aquests models. D'altra banda, conté també les entitats, que són les classes que representen els models de dades d'informació amb les que treballem. A més, també conté els serveis de domini, els quals s'encarreguen d'interactuar amb els de dades. Per últim, conté les excepcions que hem definit nosaltres.

La capa de dades conté les classes de models, on definim les taules que contenen els models de dades amb els que treballem. També les classes de servei, que són aquelles que proveeixen al backend d'interacció senzilla amb la base de dades. Aquestes classes de servei utilitzen els repositoris, interfícies que s'encarreguen de mapejar mètodes directament a sentències SQL sobre les taules generades a partir dels models.

Cal remarcar que les capes de domini i dades tenen un *package* de *mappers*. Dins d'aquests *packages*, guardem les classes que s'encarreguen de fer conversions **DTO <-> Entity**(Mappers de domini), i **Entity <-> Model**(Mappers de dades).

La base de dades, implementada mitjançant un sistema de gestió de models relacionals(PostgreSQL), romandrà dins d'una instància d'una màquina virtual allotjada a Virtech, un servidor de la UPC, així com la *web admin* i l'aplicació *backend*, que estaran en constant execució per tal de poder proveir al *frontend* de la informació i la interacció amb la BD necessàries i per a poder gestionar una part de l'aplicació.

### 3.1.2. Patrons d'arquitectura aplicats

Hem aplicat una arquitectura en tres capes per cadascuna de les dues aplicacions, *frontend* i *backend*, mentre que per a l'aplicació web hem decidit fer-la utilitzant MVC. En quant a l'arquitectura del *frontend* i del *backend*, hem decidit dissenyar-la d'aquesta manera donat que:

- La interacció entre diferents components només ocorre *intra-capa* o entre capes veïnes.
- Aquest disseny permet no fer assumcions sobre com són els tipus de dades que s'utilitzen.
- Cada capa té un grup de funcionalitats perfectament definides. Això fa que sigui senzill determinar on anirà un mètode, classe o atribut.
- La capa de dades és perfectament reutilitzable: conté funcions i mètodes senzills.

- En el nostre cas, la capa de dades també compta amb completa independència respecte a la tecnologia utilitzada(PostgreSQL).

## 3.2. Capa de domini

### 3.2.1. Diagrama de models del domini (optional)

In case you think it helps

### 3.2.2. Patrons de dissenys aplicats

Justification of need

Explanation of application (eventually including some sequence diagram or code for illustration purposes)

#### 3.2.2.1. Repositori

De manera general, aquest patró permet que dos components amb interfícies diferents puguin col·laborar. En el nostre cas hem afegit interfícies a la capa de dades per a que les classes de serveis puguin utilitzar mètodes que interactuen amb la BD, fent ús d'aquestes interfícies (classes de tipus *Repository*) operacions de les quals s'implementen de manera automàtica.

Trobem aquest patró útil i necessari ja que ens permet implementar consultes d'SQL utilitzant la JPA(*Java Persistent API*) sense cap mena d'esforç. Aquesta API funciona de la manera següent: donada una classe *Servei* que vol fer ús de la BD, només cal crear una interfície de tipus *Repositori*, que hereti de *JpaRepository*, i definir mètodes sobre els quals afegirem l'anotació *@Query* amb la consulta SQL en concret.

Això es fa per a obtenir una capa extra d'abstracció en els components corresponents, evitant la repetició del codi (principi DRY), facilitant el testatge i desacoblant-lo de la lògica de negoci.

Gràcies a l'ús d'aquest repositori, concretament a la nostra interfície anomenada *IEventRepository.java*, existeix una major independència entre la capa de dades i la resta de l'arquitectura, ja que si es produeix alguna modificació en concret es podrà rebre a partir de les seves crides utilitzant les consultes SQL i pel cas de voler fer un refactor de la crida a la capa de dades només caldria modificar aquest fitxer en concret. A continuació mostrem una part de la interfície que hem explicat:

```

6 usages  👤 danielmr6 +3
@Repository
public interface IEventRepository extends JpaRepository<Event, Long> {

2 usages  👤 ivan-risueno

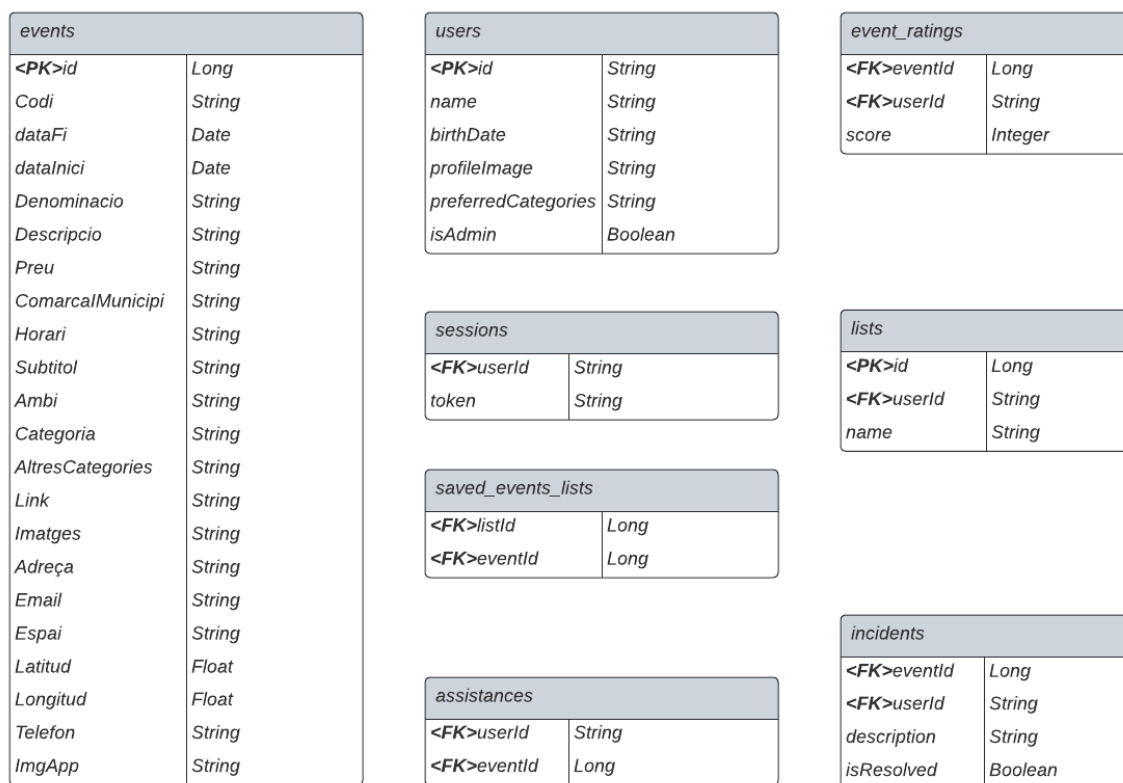
    @Query("SELECT e FROM Event e WHERE e.denominacio LIKE %?1%")
    List<Event> findAllByDenominacio(String denominacio);

2 usages  👤 danielmr6

    @Query("SELECT e FROM Event e WHERE e.preu LIKE %?1%")
    List<Event> findAllByPreu(String preu);

```

### 3.3. Diagrama de la base de dades (UML)



**Nota (I):** No hem afegit les línies que representen claus foranes per simplicitat.

**Nota (II):** Les claus primàries de les taules que referencien a altres taules estan composades per totes les seves claus foranes.

## 3.4. Llistat de les tecnologies

Number of servers

Server configuration

Number of databases

Versioning of databases

Number of languages

Utilitzem només un servidor de Virtech. Aquest servidor conté una màquina virtual amb 4GB de memòria RAM corrent una instància d'Ubuntu 20.04.

La màquina virtual té una única base de dades, anomenada *postgres*, amb la qual accedim amb l'usuari *postgres*(és el superusuari per defecte) i la contrasenya 1234. La base de dades s'ha fet utilitzant Postgresql 14.7, ja que malgrat no ser la més recent, considerem que és la més estable. Principalment tenim una base de dades, on bàsicament guardem tota la informació neta de l'Agenda Cultural, amb totes les dades processades de la millor forma per facilitar la implementació a l'equip del *frontend*.

En quant als llenguatges, fem servir SQL per a la base de dades, Dart per al *frontend*, Java per al *backend*, Kotlin per a les dependències especificades als arxius de *Gradle*, i *JavaScript* per a la *web admin*.

## 3.5. APIs

### 3.5.1. API del nostre producte

La nostra API prové al *frontend* de la informació necessària per a mostrar a l'aplicació nativa, així com la possibilitat d'interacció amb la base de dades. Dividim la nostra API en tres grans grups, un per a cada model de dades.

#### 3.5.1.1. /users

user-controller			^
GET	/users	Gets the profile of all the users registered in the system	✓ [icon]
PUT	/users	Edits the profile of a logged user	▼
POST	/users	Registers a user into the system	▼
DELETE	/users	Deletes a logged user	▼
POST	/users/logout	Logs out an user	▼
POST	/users/authenticate	Logs an user in	▼
GET	/users/profile	Gets the profile info of a logged user	▼

Els endpoints que componen aquest grup, com el seu nom indica, tenen a veure amb els usuaris. En ordre, proveïm una breu descripció de les crides:

- GET /users: Demana el perfil de tots els usuaris guardats a la base de dades i els retorna amb paginació o sense, segons s'hagi especificat.
- PUT /users: Donada una API key que representa la sessió d'un usuari loguejat i un perfil d'usuari en format JSON, modifica l'usuari a qui pertany aquesta API key i li assigna la informació especificada al JSON.
- POST /users: Registra un usuari a la base de dades.
- DELETE /users: Donada una API Key que representa la sessió d'un usuari loguejat, esborra la sessió de l'usuari, les seves assistències, incidències, llistes i finalment el propi usuari.
- POST /users/logout: Donada una API Key que representa la sessió d'un usuari loguejat, esborra aquesta sessió de la BD.
- POST /users/authenticate: Donat l'identificador d'un usuari(obtingut a partir d'una crida mitjançant *Firebase*), crea una sessió per a aquest usuari i retorna una API key.
- GET /users/profile: Donada una API Key que representa la sessió d'un usuari loguejat, obté les dades que conformen el perfil d'aquest usuari.

### 3.5.1.2. /lists

list-controller		^
GET	/lists	Gets all the lists of a logged user
PUT	/lists	Edits the name of a given list
POST	/lists	Creates an empty list for a logged user
DELETE	/lists	Deletes a list of a logged user
PUT	/lists/removeEvent	Removes an event from a list
PUT	/lists/addEvent	Adds an event to a list
GET	/lists/events	Gets all the events of a specified list

Els endpoints que componen aquest grup tenen a veure amb les llistes d'esdeveniments que els usuaris es fan de manera personalitzada. En ordre, proveïm una breu descripció de les crides:

- GET /lists: Donada una API Key que representa la sessió d'un usuari loguejat, obté totes les llistes de l'usuari en qüestió.



- PUT /lists: Donada una API Key que representa la sessió d'un usuari loguejat, l'identificador d'una llista d'aquest usuari i un string, li assigna aquest string al nom de la llista en qüestió.
- POST /lists: Donada una API Key que representa la sessió d'un usuari loguejat i un string, crea una llista buida per a aquell usuari amb el nom indicat.
- DELETE /lists: Donada una API Key que representa la sessió d'un usuari loguejat i un identificador de llista, esborra la llista si aquesta pertany a l'usuari en qüestió.
- PUT /lists/removeEvent: Donada una API Key que representa la sessió d'un usuari loguejat, un identificador de llista i un identificador d'esdeveniment, esborra l'esdeveniment de la llista si aquesta pertany a l'usuari en qüestió.
- PUT /lists/addEvent: Donada una API Key que representa la sessió d'un usuari loguejat, un identificador de llista i un identificador d'esdeveniment, afegeix l'esdeveniment a la llista si aquesta pertany a l'usuari en qüestió.
- GET /lists/events: Donada una API Key que representa la sessió d'un usuari loguejat i un identificador de llista, obté tota la informació dels esdeveniments que formen aquesta llista.

### 3.5.1.3. /incidents

incident-controller	
GET	/incidents Gets all incidents
PUT	/incidents Edits a concrete incident
POST	/incidents Saves a concrete incident in the database
DELETE	/incidents Deletes a concrete incident specified by its id
GET	/incidents/filters/users Gets a concrete incident by an specific user mail and status
GET	/incidents/filters/events Gets a concrete incident by an specific event and status
DELETE	/incidents/event Deletes all the incidents with its event id

Els endpoints que componen aquest grup, com el seu nom indica, tenen a veure amb les incidències sobre els esdeveniments de la nostra aplicació. En ordre, proveïm una breu descripció de les crides:

- GET /incidents: Demana totes les incidències guardades a la base de dades i les retorna amb paginació o sense, segons s'hagi especificat.
- PUT /incidents: Donat una incidència en format JSON, modifica els atributs especificats de la incidència corresponent identificada per l'id, el qual es troba com a primer paràmetre del JSON.
- POST /incidents: Crea una incidència i la guarda a la base de dades.
- DELETE /incidents: Elimina una incidència guardada a la base de dades a partir de l'identificador d'aquesta.
- GET /incidents/filters/users: Obté el conjunt d'incidències reportades per un usuari concret donat el seu email i l'estat d'aquesta, tenint en compte si ha estat resolta o encara no.
- GET /incidents/filters/events: Obté el conjunt d'incidències d'un esdeveniment en concret donat el seu identificador i l'estat d'aquesta incidència, tenint en compte si ha estat resolta o encara no.
- DELETE /incidents/event: Elimina el conjunt d'incidències guardades a la base de dades relacionades amb l'esdeveniment identificat pel seu id corresponent.

#### 3.5.1.4. /events

Els endpoints que componen aquest grup, com el seu nom indica, tenen a veure amb els esdeveniments. En ordre, proveïm una breu descripció de les crides:

event-controller			^
GET	/events	Gets all events from the database	▼
PUT	/events	Edits a concrete event	▼
POST	/events	Saves a concrete event to the database	▼
POST	/events/rate	Rates an event	▼
DELETE	/events/rate	Removes the user rating of an event	▼
GET	/events/{id}/allInfo	Gets an event with the specified id from the database	▼
GET	/events/tags	Gets the possible filter tags from the database	▼
GET	/events/suggestions	Gets the recommended events for a specific user	▼
GET	/events/preu	Gets events with the specified price from the database	▼
GET	/events/distance	Gets events with the specified distance, user position and date range from the database	▼
GET	/events/descripcio	Gets events with the specified description from the database	▼
GET	/events/denominacio	Gets events with the specified denomination from the database	▼
GET	/events/date	Gets events with the specified range of dates from the database	▼
GET	/events/comarcaIMunicipi	Gets events with the specified region from the database	▼
GET	/events/categoria	Gets events with the specified category from the database	▼
GET	/events/ambit	Gets events with the specified ambit from the database	▼
GET	/events/altres	Gets events with the specified id from the database	▼
GET	/events/allFilters	Gets events with the specified filters from the database	▼
DELETE	/events/{id}	Deletes a concrete event from the database	▼

- GET /events: Demana tots els esdeveniments guardats a la base de dades i els retorna amb paginació o sense, segons s'hagi especificat.
- PUT /events: Donat un esdeveniment en format JSON, modifica els atributs especificats de l'esdeveniment identificador del qual es troba com a primer paràmetre del JSON.
- POST /events: Crea un esdeveniment i el guarda a la base de dades.
- POST /events/rate: Donada una API Key que representa la sessió d'un usuari loguejat, crea una valoració per un esdeveniment concret i la guarda a la base de dades.
- DELETE /events/rate: Donada una API Key que representa la sessió d'un usuari loguejat, elimina una valoració d'un esdeveniment guardat a la base de dades donat el seu identificador.
- GET /events/{id}/allInfo: Obté tots els camps d'un esdeveniment guardat a la base de dades identificador del qual s'especifica mitjançant l'URL de la crida.

- GET /events/date: Obté tots els camps dels esdeveniments guardats a la base de dades que es portin a terme entre el rang de dates especificat.
- GET /events/tags: Obté tots els àmbits, categories i altres categories que es troben presents a la nostra base de dades.
- GET /events/suggestions: Donada una API Key que representa la sessió d'un usuari loguejat, obté tots es esdeveniments recomanats per l'usuari que està loguejat a l'aplicació.
- GET /events/preu: Obté tots els camps dels esdeveniments guardats a la base de dades que continguin un string referent al preu proporcionat mitjançant l'URL de la crida.
- GET /events/distance: Obté tots els esdeveniments que es portin a terme durant un rang de dates determinat i estiguin a una distància menor o igual del radi especificat donada una posició en concret, la qual consta d'una latitud i una longitud.
- GET /events/descripcio: Obté tots els camps dels esdeveniments guardats a la base de dades que continguin un string referent a la descripció proporcionada mitjançant l'URL de la crida.
- GET /events/denominacio: Obté tots els camps dels esdeveniments guardats a la base de dades que continguin un string referent a la denominació(nom) proporcionada mitjançant l'URL de la crida.
- GET /events/comarcaIMunicipi: Obté tots els camps dels esdeveniments guardats a la base de dades que continguin un string referent a la comarca i municipi(en format *comarca,municipi*) on es porten a terme dits esdeveniments proporcionada mitjançant l'URL de la crida.
- GET /events/categoria: Obté tots els camps dels esdeveniments guardats a la base de dades que continguin un string referent a la categoria proporcionada mitjançant l'URL de la crida.
- GET /events/ambit: Obté tots els camps dels esdeveniments guardats a la base de dades que continguin un string referent a l'àmbit proporcionat mitjançant l'URL de la crida.
- GET /events/altres: Obté tots els camps dels esdeveniments guardats a la base de dades que continguin un string referent a les altres categories proporcionades mitjançant l'URL de la crida.

- GET /events/allFilters: Obté tots els esdeveniments que concorden amb els filtres especificats com a JSON que es troben a la nostra base de dades.
- DELETE /events/{id}: Elimina un esdeveniment guardat a la base de dades identificador del qual s'especifica mitjançant l'URL de la crida.

### 3.5.1.5. /assistances

assistance-controller	
GET	/assistances Gets all assistances
POST	/assistances Saves a concrete assistance
DELETE	/assistances Deletes a concrete assistance

- GET /assistances: Obté totes les assistències a la BD(afegida per comoditat a l'hora de desenvolupar l'aplicació, aquesta crida s'eliminarà a la versió final).
- POST /assistances: Donada una API Key que representa la sessió d'un usuari loguejat i un identificador d'esdeveniment, afegeix una assistència a la BD d'aquest usuari a aquest esdeveniment.
- DELETE /assistances: Donada una API Key que representa la sessió d'un usuari loguejat i un identificador d'esdeveniment, esborra de la BD l'assistència d'aquest usuari a aquest esdeveniment.

**NOTA:** Les crides de tipus GET que retornen més d'un element implementen totes paginació opcional, que s'ha d'especificar mitjançant paràmetres opcionals de les pròpies crides.

## 3.5.2. APIs externes

Tot i que l'API principal i més utilitzada es la que hem implementat nosaltres, la nostra aplicació utilitza varies APIs que ens faciliten la implementació de funcionalitats com el mapa i el registre d'usuaris. En el cas de Flutter utilitzem plugins, que ens proporcionen els mètodes per comunicarnos amb les següents APIs:

### 3.5.2.1. Google Maps API

Permet mostrar mapes interactius de Google en pantalla completa, personalitzar els marcadors. També permet realitzar diverses interaccions amb els mapes, com fer zoom, moure's i obtenir informació sobre ubicacions específiques.

Per utilitzar el plugin de Google Maps de Flutter, és necessari configurar una API key de Google Maps perquè l'aplicació pugui accedir als serveis de mapes de Google. Aquesta API key és única per a cada aplicació i s'utilitza per a autenticar les sol·licituds i assegurar que l'ús del servei estigui dins dels límits i restriccions establerts per Google.

La API ofereix una sèrie de característiques addicionals, com la detecció 'touch events' en el mapa, la geocodificació (convertir direccions en coordenades geogràfiques) i la inversió de geocodificació (convertir coordenades geogràfiques en direccions). També és possible superposar capes personalitzades en els mapes, com a polígons, polilíneas i cercles, per a representar àrees o elements específics.

De tots els serveis mencionats, el que més utilitzarem seran la detecció de 'touch events' per gestionar les crides al back per obtenir només els esdeveniments que s'haurien de veure per pantalla, i per interaccionar amb els marcadors i veure la info d'un esdeveniment. També utilitzarem la personalització de marcadors, del mapa i la funcionalitat de mostrar la ubicació a temps real del dispositiu,

### 3.5.2.2. Firebase Auth API

Firebase Auth és un servei de Firebase que ofereix una varietat de mètodes d'autenticació per a usuaris, com correu electrònic i contrasenya, Google Sign-In, Facebook Login, Twitter Login, entre d'altres. Permet agregar fàcilment mètodes d'autenticació segurs i confiables, sense haver d'implementar tot el codi des de zero.

Aquestes són les principals funcionalitats que utilitzem per la nostra aplicació:

**Autenticació d'usuaris:** Proporciona mètodes simples per registrar nous usuaris, iniciar sessió amb credencials existents i tancar sessió. Permetem que els usuaris creïn comptes utilitzant la seva adreça de correu electrònic i contrasenya, o iniciïn sessió amb un proveïdor externs com Google.

**Gestió d'usuaris:** Podem accedir a informació i propietats d'usuari, com el seu ID únic, nom i adreça de correu electrònic. A més, podem actualitzar la informació de l'usuari, com canviar l'adreça de correu electrònic o la contrasenya.

**Verificació de correu electrònic:** Firebase Auth permet enviar correus electrònics de verificació als usuaris per verificar les seves adreces de correu electrònic. Podem

verificar si un usuari ha verificat el seu correu electrònic, incrementant la seguretat en el moment de la creació d'un compte.

Restabliment de contrasenya: Si un usuari oblida la seva contrasenya, Firebase Auth ens proporciona mètodes per restablir-la enviant un correu electrònic de restabliment de contrasenya. Això ens facilita la recuperació l'accés als comptes de forma segura.

#### 3.5.2.3. Geolocator API

Ens permet obtenir la ubicació del dispositiu i fer un seguiment en temps real. Ens facilita la obtenció de les coordenades de longitud i latitud, configurar opcions de precisió i freqüència d'actualització, i verificar i demanar permisos necessaris. L'utilitzem principalment per centrar el mapa en la ubicació a temps real del nostre dispositiu i per obtenir la distància en km entre diferents esdeveniments.

#### 3.5.2.4. Permission handler API

Simplifica la gestió de permisos en aplicacions mòbils. Ens permet sol·licitar i verificar fàcilment permisos, com ubicació en el nostre cas, a Android. També s'encarrega de mostrar els quadres de diàleg de sol·licitud de permisos nadius i ofereix opcions de configuració. És útil per garantir una experiència d'usuari adequada i complir amb les polítiques de privacitat i seguretat, i ens serveix per poder accedir a la ubicació del dispositiu per la funcionalitat del mapa.

#### 3.5.2.5. Shared preferences API

Ens permet emmagatzemar dades simples i persistents de l'aplicació. Amb aquesta API podem guardar i accedir a valors clau de forma senzilla. No requereix una base de dades, és fàcil d'usar i molt eficient. Nosaltres l'utilitzem per emmagatzemar preferències les dades de l'usuari loguejat i l'APIToken d'aquest una vegada ha fet login.

### 3.5.3. Consum del servidor

Hem consensuat amb l'equip que desenvolupa l'aplicació *PlugFinder* per a fer ús d'una de les crides a la seva API. La crida ens proporcionarà una llista d'endolls on carregar un cotxe elèctric, informació que li serà útil al *frontend* per a mostrar marcadors amb aquests endolls als mapes on es mostrin esdeveniments.

*Els paràmetres de la crida a la API que consumim encara queden per determinar-se.*

### 3.5.4. Subministrament del servidor

Hem arribat a un acord amb el grup que desenvolupa l'aplicació *Concessiona't* per a oferir-los una crida a la nostra API. La crida en qüestió és una que s'ha descrit a l'apartat 3.5.1.4, concretament, */events/distance*. Aquesta crida, donats un *radi*, *latitud*, *longitud*, i dates d'inici i fi (i paràmetres opcionals de paginació per al resultat), obté una llista d'esdeveniments presencials que tenen lloc a un radi de *radi* kilòmetres a la ubicació (*latitud*, *longitud*) entre les dates *dataIni* i *dataFi*.

Aquesta informació els hi serà útil a l'equip de desenvolupament de *Concessiona't* per a poder mostrar al mapa de la seva aplicació marcadors amb els esdeveniments que retorna la crida.

### 3.5.5. Consum de Dades Obertes

Per a proveir al *frontend* d'informació sobre els esdeveniments que tenen lloc a Catalunya, necessitem consumir la API de l'Agenda Cultural. Concretament, per a obtenir els esdeveniments que formen el dataset (en podem obtenir 1000 com a màxim per crida) hem de fer una petició de tipus GET a l'endpoint "<https://analisi.transparenciacatalunya.cat/resource/rhpy-yr4f>".

Aquesta crida, si la fem sense paràmetres, ens retornarà com a màxim els 1000 primers esdeveniments que es troben a la base de dades de l'aplicació de l'Agenda Cultural. Nosaltres, per conveniència i per fer ús de dades representatives, hem decidit afegir a la crida el paràmetre de tipus *Query* *<data\_inici>*. En concret, la crida resultant retorna els esdeveniments pertinents amb una data d'inici major a la del dia quan es fa la crida, així ens estalviem obtenir esdeveniments que són passats.

Malgrat això, "no tot són flors i violes", i el dataset que ofereix la API de l'Agenda Cultural està ple d'informació inconsistent que, nosaltres, des de l'aplicació *backend*, hem de tractar per netejar o directament eliminar:

- Codi: el codi dels esdeveniments a priori pot semblar que funciona com a clau primària, però no és el cas: és una combinació de l'any quan es fa l'esdeveniment i altres dígitos arbitraris, i no és únic per a cada esdeveniment.
- Dates: les dates d'inici i de fi dels esdeveniments venen en format "dd/mm/AAAA" + "T" + "hh:mm:ss", i n'hi han algunes que valen 99/99/9999.
- Descripció: malgrat haver un camp de descripció en format HTML, el camp de descripció que hauria de ser text pla no ho és en alguns casos.



- Subtítol: mateix cas que amb les descripcions.

Aquests camps que hem esmentat són inconsistents, però també hi han d'altres, com per exemple *àmbits*, *categories*, *altres categories* o *comarca i municipi* que només ens hem d'encarregar de donar-li un format més llegible.

Per últim, remarcar que a banda del *sodi*, denominació i descripció dels esdeveniments, tota la resta de camps poden ser buits.

Davant d'aquestes adversitats ens veiem obligats al *backend* a netejar tota la informació que rebem a través de la API de l'Agenda Cultural per a guardar a la nostra BD dades llegibles i útils per al *frontend* o per a l'aplicació *Concessiona't*.

## 3.6. Eines de desenvolupament i entorn de treball

Use of Frameworks

Continuous Integration

Deployment

Aquest apartat ja ve explicat als punts 2.5, 2.6, 2.7 i 2.10.