Dokument som samler Dokumentationen af FarmTracking

Dokumentet indeholder:

Dokumentation af Plante-app

Dokumentation af Hotspot Funktionalitet

Dokumentation af Gylleflydelags-app

Dokumentation af IT platformen

FarmTracking, 2015
Hovedforfatter: Jesper Riber Nielsen
Biddrager: Piter Poddubnyak, Mads Nielsen, Mikkel Mortensen

Indholdsfortegnelse

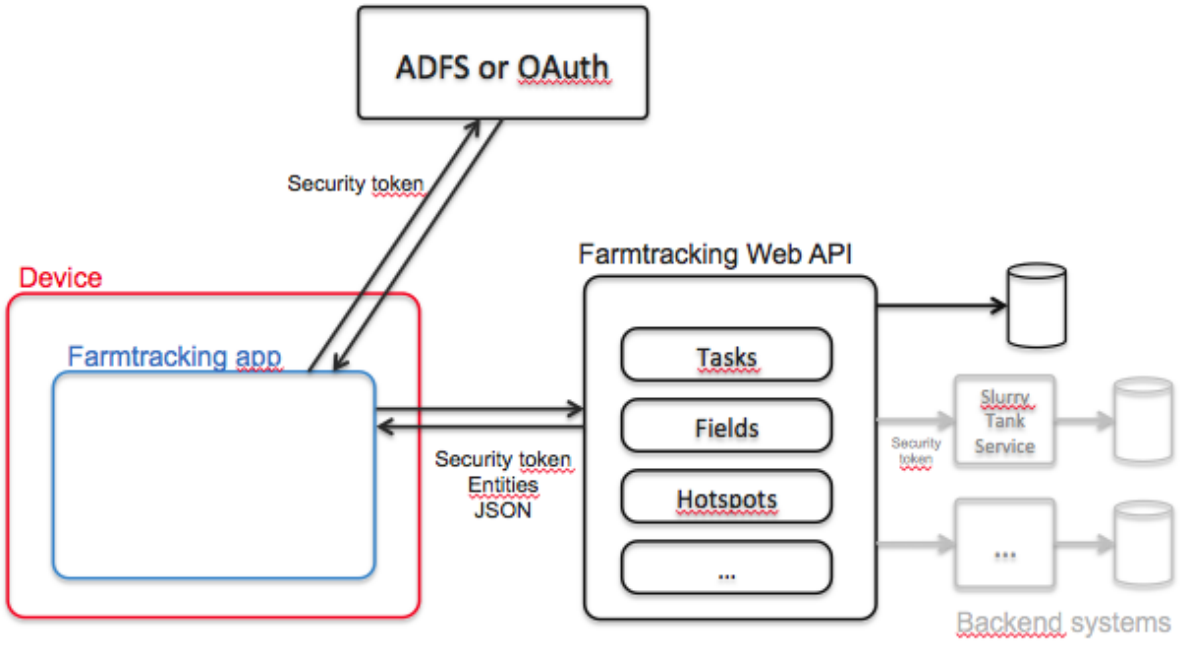
1 Introduktion

I dette dokument vil de enkelte dele FarmTracking blive dokumenteret. Eftersom de enkelte funktionaliteter er blevet samlet under en applikation vil der være dele af dokumentationen som er dækkende for flere af leverancerne. Nedenfor vil en oversigt informere om hvilke afsnit i dokumentet som tilhøre de enkelte leverancer.

- Dokumentation af Plante-app
    - Afsnit 2 Generel IT Arkitektur i FarmTracking
    - Afsnit 4 FarmTracking Android app documentation
    - Afsnit 5 Farmtracking API description
- Dokumentation af Hotspot Funktionalitet
    - Afsnit 2 Generel IT Arkitektur i FarmTracking
    - Afsnit 4 FarmTracking Android app documentation
    - Afsnit 5 Farmtracking API description
- Dokumentation af Gylleflydelags-app
    - Afsnit 2 Generel IT Arkitektur i FarmTracking
    - Afsnit 4 FarmTracking Android app documentation
    - Afsnit 5 Farmtracking API description
- Dokumentation af IT platformen
    - Afsnit 3 IT platform til oprettelse af påmindelser

2 Generel IT Arkitektur i FarmTracking

Figur 1 nedenfor illustrere den generelle arkitektur i FarmTracking og hvordan de enkelte element er forbundet. Startende fra højre side er backend systemerne illustreret. Disse systemer indeholder forskellige databaser og services som laver snitfladerne til kommunikation med blandt andet FarmTracking Web Api. FarmTracking Web API er den kode blok som kommunikere med de enkelte Android enheder. ADFS eller OAuth er den teknologi som bruges til at sikre brugernes data. Lavpraktisk betyder det at FarmTracking præsentere brugeren for en log-in skærm og inputtet herfra kontrolleres så via ADFS eller OAuth servicen. Den røde Device symbolisere en Android telefon og FarmTracking applikationen heri er det endelige FarmTracking program som lever i sin egen "sandkasse".

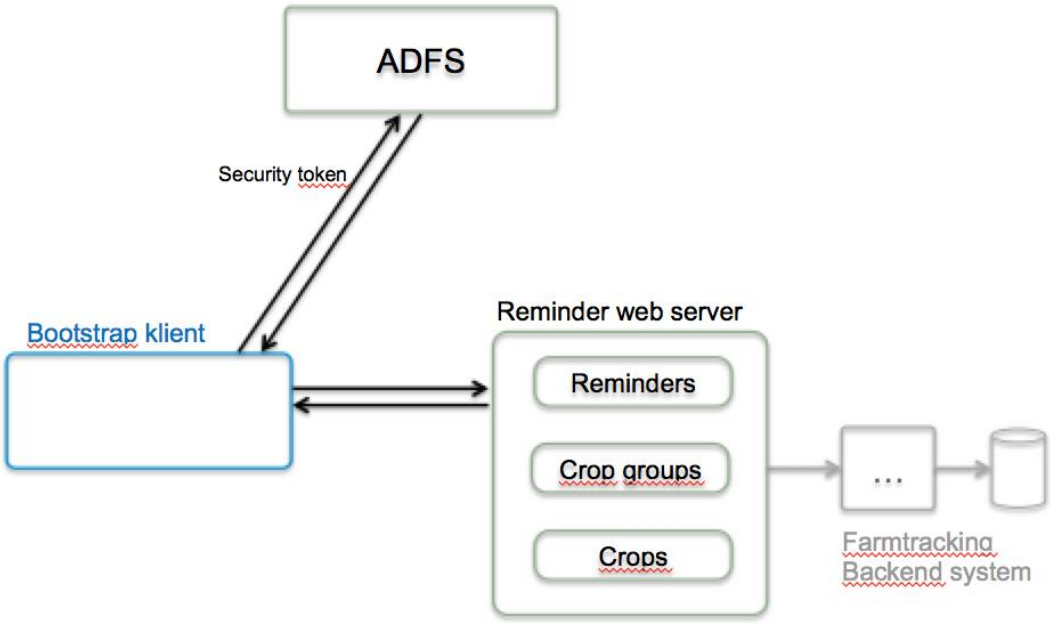


Figur 1 - Arkitekturen på IT platformen

3 IT platform til oprettelse af påmindelser

Figur 2 Illustrere arkitekturen som websiden til oprettelse af påmindelser er opbygget omkring.

I højre side har vi FarmTracking backend system som er den database hvorigennem hotspots opbevares. Oven på databasen er opbygget en snitflade som gør den skalerbar i forhold til andre applikationer. Reminder web serveren er den snitflade hvorigennem data omkring de enkelte påmindelser, afgrøde og afgrødegrupper distribueres til en brugerflade. Bootstrap klienten er hvorigennem brugerfladen generes. Brugerfladen er opbygget omkring et simpelt Bootstrap bibliotek for at minimere vedligeholdes aktiviteter efterfølgende. Når brugeren logges ind mødes de af samme sikkerheds anordning som før beskrevet i afsnit 2.



Figur 2 - Arkitektur af IT platform til oprettelse af Påmindelser

4 FarmTracking Android app documentation

I dette afsnit vil de tekniske aspekter af FarmTracking android applikationen blive beskrevet.

4.1 Overall app structure.

Minimal android version: 4.0.3

Lines of code: 11 136

Rules Compliance according to Sonar: 98.8%

Used libraries:

Volley – Google net framework

Ksoap2 – SOAP lib

Espresso – Android test kit

LikeOrmLib – library for working with SQLite DB

GSON – used for work with JSON data.

Google Play Services – Google framework for Maps, Accounts etc.

Butterknife – view injection library for Android

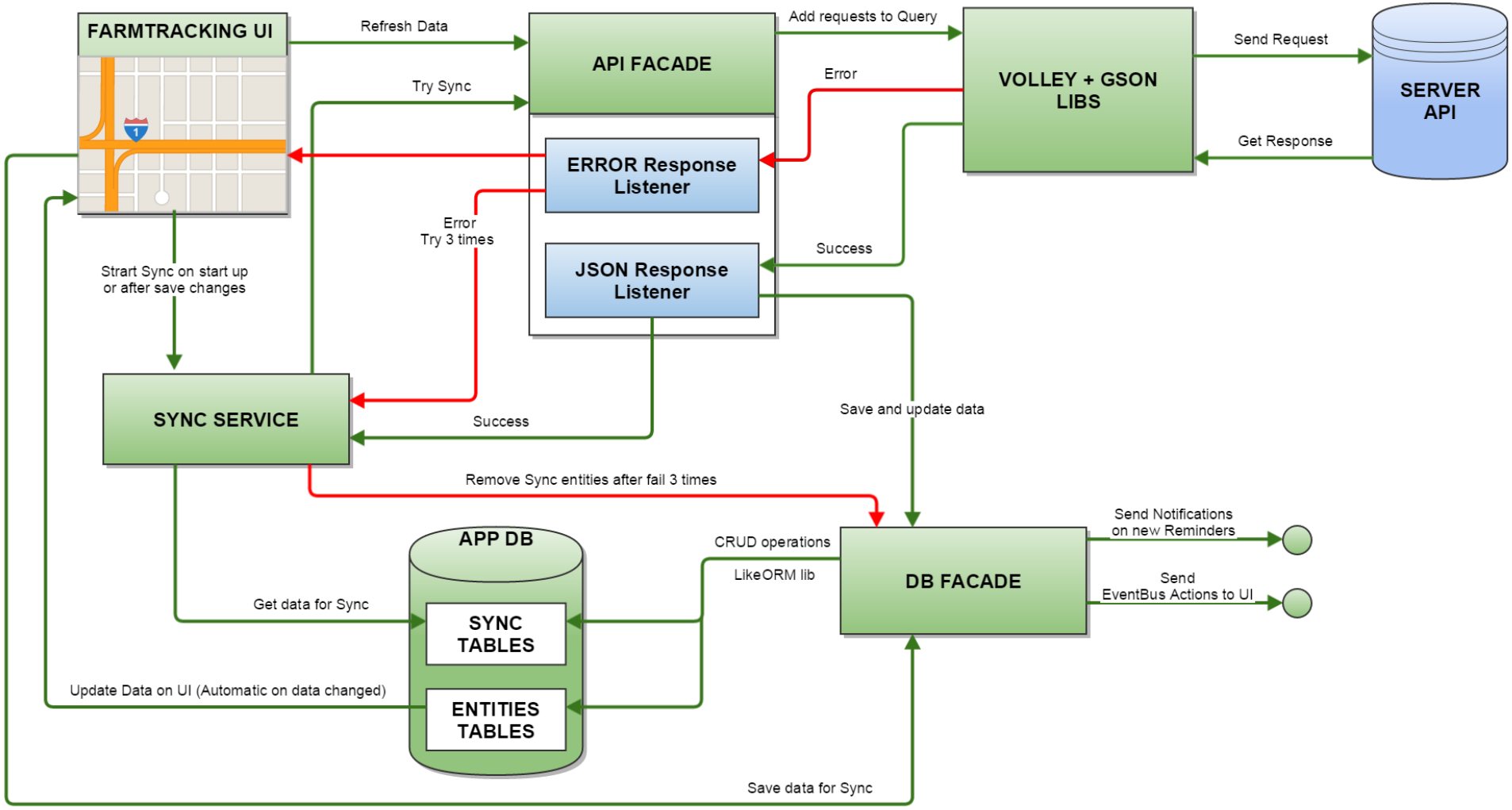
EventBus - publish/subscribe event bus optimized for Android

Calligraphy – custom fonts

Android maps utils – library for clustering markers on map

Picasso – work with images (download, show, cache)

Crashlytics – analytic library for crashes



Figur 3 - Generel applikation struktur

4.2 Main parts of application

- **UI** - FarmTracking is fragment oriented application.

There are only two Activities: Login activity and Main activity (used for navigation, update data, includes sliding menu etc.).

All UI based on fragments and custom view.

Supports landscape/portrait modes (handle orientation changes, save data on change etc.).

For all screen was created fragments with needed functionality (sometimes universal, like map screen)

UI can be easily updated, expanded and modified in future.

- **Network –** APIFacade main class for network communications (used for preparing requests). Lower network layer includes Volley and GSON libraries which responsible for handling and parsing of request and responses.

JSON response listener – used for handling success responses, parsing and storing data through DB layer and sends callbacks to UI.

Error response listener – used for handling errors.

- **DB layer –** FarmTracking used standard android DB – SQLite through LikeORM library for model mapping and CRUD operations (This library provide good performance and flexibility).

DBFacade – main class for store/update/delete all data within applications. Within UI access to data is asynchronous and supports update UI automatically on data change.

- **Services –** FarmTracking includes 2 background services.

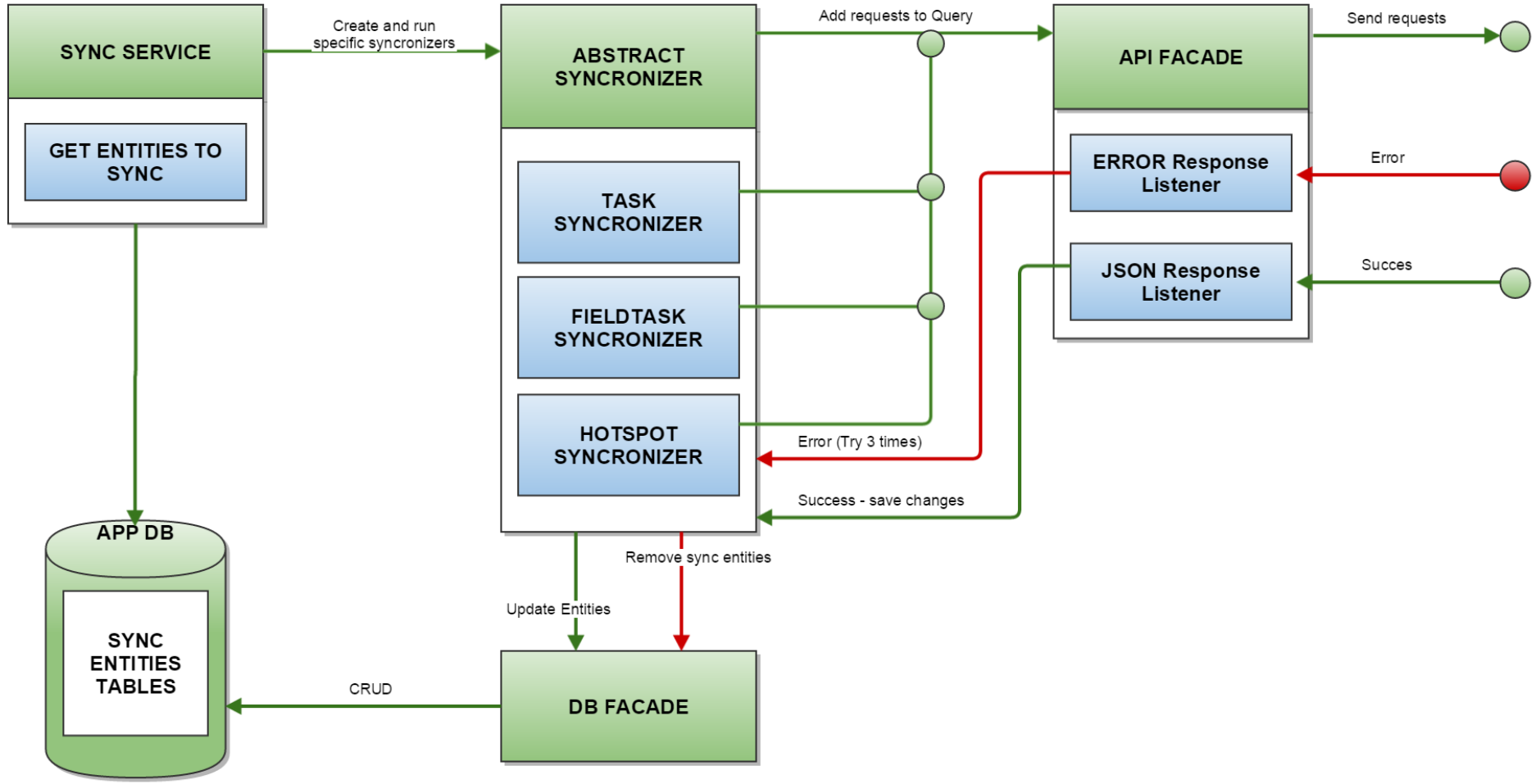
UpdateService – used for update Tasks and Hotspots during application lifetime (update data every 5 minutes).

SyncService – used for background synchronization in-app data with server. (Runs on start-up, connection to internet or after changes within app – new hotspot, read/unread reminder etc.)

SOAP service – used for logging network problems.

Application used background operations for all heavy operations (network, request data from DB etc.)

4.3 Synchronization.



Figur 4 - Synkroniserings skema

SyncService – main class for synchronization process. It's Intent service, working in background.

Service runs on application start-up, internet connection available event or manually after changes within app (add/update/delete hotspot etc.)

In DB we have separate tables for sync entities which contains all needed information for synchronization – ids of entities, path for images, operation types (CRATE, UPDATE, DELETE) etc.

Synchronization flow:

- Start service
- Check for any data in Sync tables
- Get data for sync from DB (if exist or shut down if sync DB is empty)
- Depends on type of data Service creates different types of synchronizers (extended from BaseSynchronizer) and run sync process.
- Synchronizers creates request to server through APIFacade and wait for response.
- If response is success – data updates in DB and sync entity removes. UI updates automatically.
- If any errors in response (except token expiration and server not respond) – synchronizer try 2 more times, if no success – removes sync entity from DB.

4.4 Error handling

Each request to server includes ServerErrorListener. This listener responsive for handling errors and exceptions during network operations.

After error appears listener check for error type.

If error type is not “No Internet Connection” or “Token expired” - app will send logs to server through SOAP service.

Also listener send callback with information about error to part of application where this request was called (UI, service etc.) for additional handling – save data locally, try one more time etc.

If error type is “Token expired” - listener sends notification using EventBus and application will log out (immediately or after save changes).

4.5 Update strategy

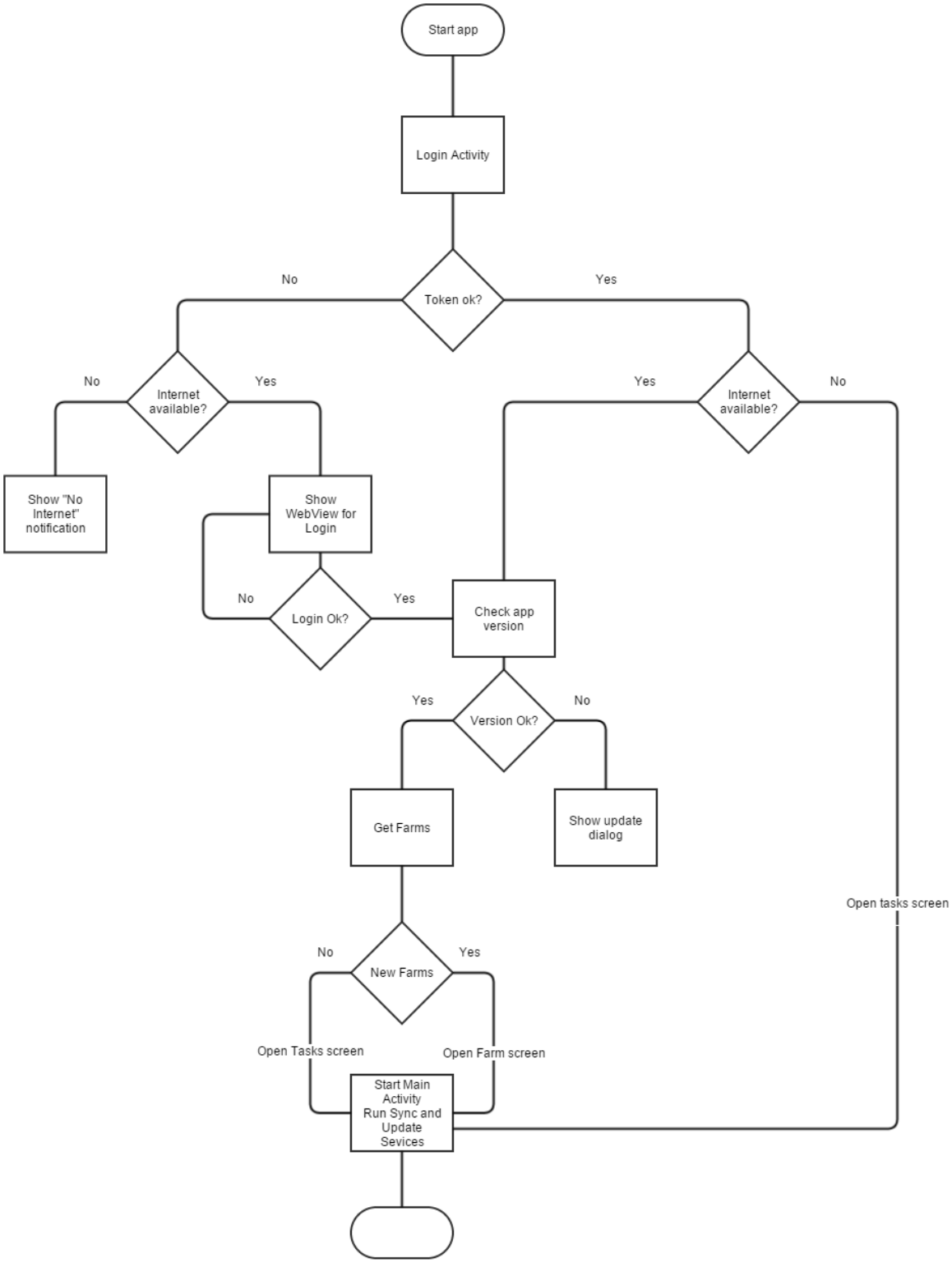
Farm list - updates on start-up of application.

Images - loads only if they need (thumbnails on map and full size on Hotspot details screen) and cached on device.

Hotspots, tasks, field tasks – updates on start-up and every 5 minutes in background.

Fields – updates every time, when user opens fields list screen

4.6. Application flow



Figur 5 - Opstart af applikation

Figur 6 - Flow i applikation

4.7 Unit tests

Application includes 23 unit tests.

Parts of application covered by unit tests:

- DBFacade – all write/udate/delete operations with FarmTracking DB.
- FormatUtils – utilities for format data.
- GeoUtils – utilities for working with geometry(calculates polygon geometry, centroid, line intersection etc.)
- HarvestYearUtils – utilities for calculating and storing harvest year.

5 FarmTracking API description

This document describes the API for the FarmTracking service which is part of the FarmTracking application.

5.1 Entities

The following entities will be sent via the API

5.1.1 Task

A task (also called notification in the *FarmTracking developer introduction* document) is the entity returned when the device is polling the backend with a new GPS position. It describes the message the user will see in the phones notifications system.

Name	Data type	Description	Editable
Id	String	Unique id	No
ServiceId	String	Origin subsystem	No
UserId	String	User the Task addressed to	No
Title	String	Title of the task	No
Text	String	Message in the task	No
Url	String	Related url the user can visit	No
IsRead	Boolean	User has read the task	Yes
Postponed	Nullable<DateTime>	User has postponed task to this date (UTC)	Yes
FieldTasks	Collection<FieldTasks>	List of fields the task is attached to	No

5.1.2 FieldTask

A FieldTask is only used as a child entity to a Task.

Name	Data type	Description	Editable
Id	String	Unique id	No
TaskId	String	Parent Task Id	No
Title	String	Title of the FieldTask (Field number and name)	No
FieldId	Int32	Id of the Field this FieldTask represent	No
IsDone	Boolean	The Task has been carried out on this field	Yes

5.1.3 Farm

Represents a farm owned or in another way available for the user.

Name	Data type	Description	Editable
Id	Int32	Unique id of the farm	No
Name	String	Farm name	No

5.1.4 Field

Represents a field owned or in another way available for the user.

The field entity includes the field geometry in Well-known Text (WKT) format.

Name	Data type	Description	Editable
Id	Int32	Unique id	No
FarmId	Int32	Id of the farm	No
FarmName	String	Farm name	No
HarvestYear	Int32	The harvest year (season) this field is available in	No
FieldNumber	String	Displayed field number e.g. ”15-1”	No
FieldName	String	Field name	No
PreCropName	String	Previous year crop name	No
Geometry	String	Geometry of the field. Well-known Text format.	No

5.1.5 Crop

A field can have one or more crops.

Name	Data type	Description	Editable
Id	Int32	Unique id	No
FieldId	Int32	Id of parent Field	No
FarmId	Int32	Id of the farm	No
HarvestYear	Int32	The harvest year (season) this crop is available in	No
CropName	String	Crop name	No
SuccessionNo	String	The crop with the highest succession number on a field is the main crop	No
VarietyName	String	Variety name	No

5.1.6 Hotspot

A hotspot is a registered place of interest with a geo position and type

Name	Data type	Description	Editable
Id	Int32	Unique id	No
LastUpdatedDate	DateTime	Date for last change (UTC). Is automatic updated on POST and PATCH requests	No
HotspotTypeId	Int32	Hotspot type	Yes
HotspotSubTypeId	Collection<Int32>	List of hotspot subtypes e.g. weed type	Yes

Geometry	String	Geometry of hotspot. Well-known Text format.	Yes
Description	String	User written description	Yes
ImageUrls	Collection<String>	List of url to images the user has taken of the hotspot	No
FarmId	Int32	Farm this hotspot is belonging to	Yes

5.1.7 HotspotType

Hotspot type (e.g. weed, stone) to be used when registering a hotspot

Name	Data type		
Id	Int32	Unique id	No
Name	String	Hotspot type name	No
GeometryType	Int32	Specifies valid geometries for the hotspot: 1: Point only 2: Point and line 3: Point, line and polygon	No

5.1.8 HotspotSubType

A subtype to the hotspot type (e.g. weedtypes)

Name	Data type		
Id	Int32	Unique id	No
HotspotTypeId	Int32	Id of the parent hotspot type	No
Name	String	Hotspot type name	No

5.1.9 MinAppVersion

Holds version of minimum required app version

Name	Data type		
Major	Int32	Major version number	No
Minor	Int32	Minor version number	No

5.2 API usage

The API is implemented as a Web API and has the following OData endpoints:

Url	Commands
https://farmtracking.dlbr.dk/api/Tasks	GET, PATCH
https://farmtracking.dlbr.dk/api/FieldTasks	PATCH
https://farmtracking.dlbr.dk/api/Farms	GET

https://farmtracking.dlbr.dk/api/Fields	GET
https://farmtracking.dlbr.dk/api/Crops	GET
https://farmtracking.dlbr.dk/api/Hotspots	GET, POST, PATCH, DELETE
https://farmtracking.dlbr.dk/api/Images	GET, POST, DELETE
https://farmtracking.dlbr.dk/api/HotspotTypes	GET
https://farmtracking.dlbr.dk/api/HotspotSubTypes	GET
https://farmtracking.dlbr.dk/api/MinAppVersion	GET

Notes:

- The entity and property names are case sensitive
- Currently the API has no authorization. For now every request will automatically be authenticated as the user fttest1@PROD.DLI
- The odata requests can be filtered, ordered etc. with use of the odata syntax. See for example http://www.odata.org/getting-started/basic-tutorial/
-

5.2.1 Task

To get all Tasks for the user use the following request

```
GET https://devtest-farmtracking.vfltest.dk/api/Tasks
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

Used on reminders main page

The following examples shows the use of the odata syntax for ordering and filtering

```
GET https://devtest-farmtracking.vfltest.dk/api/Tasks?$orderby=Title desc
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

```
GET https://devtest-farmtracking.vfltest.dk/api/Tasks?$filter=startswith(Title,'Test')
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

The following request will return the list of Tasks and include all child FieldTasks

```
GET https://devtest-farmtracking.vfltest.dk/api/Tasks?$expand=FieldTasks
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

To get a single Task by its id, use this request

```
GET https://devtest-farmtracking.vfltest.dk/api/Tasks('Plant_Task22')
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

This can be combined with the expand command to get a single Task with expanded child FieldTasks

```
GET https://devtest-
farmtracking.vfltest.dk/api/Tasks('Plant_Task22')?$expand=FieldTasks
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

Used on reminder details page

To update a Task use the PATCH method. This will only update the properties for the entity sent in the request body.

To update Task postponed date:

```
PATCH https://devtest-farmtracking.vfltest.dk/api/Tasks('Plant_Task22') HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Content-Type: application/json
Authorization: Bearer <encodedToken>

{
   "Postponed":"2014-09-23T00:00:00Z"
}
```

Used on reminder details page

Clear the postponed date:

```
PATCH https://devtest-farmtracking.vfltest.dk/api/Tasks('Plant_Task22') HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Content-Type: application/json
Authorization: Bearer <encodedToken>

{
   "Postponed":null
}
```

Used on reminder details page

Update IsRead property:

```
PATCH https://devtest-farmtracking.vfltest.dk/api/Tasks('Plant_Task22') HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Content-Type: application/json
Authorization: Bearer <encodedToken>

{
   "IsRead":true
}
```

Used on reminder details page

Update multiple properties:

```
PATCH https://devtest-farmtracking.vfltest.dk/api/Tasks('Plant_Task22') HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Content-Type: application/json
Authorization: Bearer <encodedToken>

{
   "Postponed":"2014-09-23T00:00:00Z",
   "IsRead":true
}
```

Used on reminder details page

Delete a Task with this request:

```
DELETE https://devtest-
farmtracking.vfltest.dk/api/Tasks('SlurryTank15803_Date20141222') HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

The following request the will poll for new Taks in relation to the current position. It will return a list of new taskIds – if any. In case of any the tasks, the client should download a fresh Task list

```
POST https://devtest-
farmtracking.vfltest.dk/api/Tasks/FT.TaskChangesAvailableForLocation HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Content-Type: application/json
Authorization: Bearer <encodedToken>

{
   "latitude": 1,
   "longitude": 1
}
```

Used in background polling thread

5.2.2 FieldTask

FieldTasks is only used on the reminder details screen. As described above the only way to get a list of FieldTask is by expanding the Task entity.

This request will return a single FieldTask:

```
GET https://devtest-
farmtracking.vfltest.dk/api/FieldTasks('Plant_Task22_FieldId12345')
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

This request will update the IsDone property on a FieldTask entity:

```
PATCH https://devtest-
farmtracking.vfltest.dk/api/FieldTasks('Plant_Task22_FieldId12345') HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Content-Type: application/json
Authorization: Bearer <encodedToken>

{
   "IsDone":true
}
```

5.2.3 Farm

The available farms for the current user can be received with the following request:

```
GET https://devtest-farmtracking.vfltest.dk/api/Farms
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

5.2.4 Field

When requesting a Field list you always have to specify FarmId(s) and HarvestYear parameters in the odata filter string.

Get all Fields for a specific farm in a specific harvest year :

```
GET https://devtest-farmtracking.vfltest.dk/api/Fields?$filter=FarmId eq 71965 and
HarvestYear eq 2014
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

Get all Fields for two farms in a specific harvest year :

```
GET https://devtest-farmtracking.vfltest.dk/api/Fields?$filter=(FarmId eq 71965 or
FarmId eq 29275) and HarvestYear eq 2014
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

Get all Fields for a specific farm in a specific harvest year – and get child crops for each field:

```
GET https://devtest-farmtracking.vfltest.dk/api/Fields?$filter=FarmId eq 71965 and
HarvestYear eq 2014&$expand=Crops
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

5.2.5 Crop

When requesting a Crop list you always have to specify FarmId(s) and HarvestYear parameters in the odata filter string.

Get all Crop for a specific farm in a specific harvest year :

```
GET https://devtest-farmtracking.vfltest.dk/api/Crops?$filter=FarmId eq 71965 and
HarvestYear eq 2014
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

Note: It's also possible to get the Crops by using the Expand parameter on the Fields endpoint

5.2.6 Hotspot

Get Hotspots for specific farms

```
GET https://devtest-farmtracking.vfltest.dk/api/Hotspots?$filter=(FarmId eq 71965 or
FarmId eq 29275)
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

Get single Hotspot by Id:

```
GET https://devtest-farmtracking.vfltest.dk/api/Hotspots(5)
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

To insert a new Hotpost use POST. The Geometry is in WKT format

```
POST https://devtest-farmtracking.vfltest.dk/api/Hotspots HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Content-Type: application/json
Authorization: Bearer <encodedToken>

{
   "HotspotTypeId":1,
   "HotspotSubTypeIds":[12,25,48],
   "FarmId":29275,
   "RegisteredDate":"2014-09-23T00:00:00Z",
   "Description":"Hotspot 33",
   "Geometry":"POLYGON ((10.142741203099936 56.202911526834306, 10.142987966329269
56.202962254387657, 10.142977237493215 56.202857815234104, 10.142741203099936
56.202911526834306))"
}
```

Use PATCH to update a Hotspot:

```
PATCH https://devtest-farmtracking.vfltest.dk/api/Hotspots(6) HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Content-Type: application/json
Authorization: Bearer <encodedToken>

{
   "Description":"Hotspot 33 - Updated from API",
   "Geometry":"POLYGON ((10.142741203099936 56.202911526834306, 10.142987966329269
56.202962254387657, 10.142977237493215 56.202857815234104, 10.142741203099936
56.202911526834306))"
}
```

A Hotspot can be deleted with the following request:

```
DELETE https://devtest-farmtracking.vfltest.dk/api/Hotspots(6)
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

5.2.7 Images

Hotspot images are handled through the images endpoint (not odata…)

All requests has to include a valid Authorization header

Get an image by id

```
GET https://devtest-farmtracking.vfltest.dk/api/images/123 HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

Insert a new image by using multipart/form-data post

```
POST https://devtest-farmtracking.vfltest.dk/api/images HTTP/1.1
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
Content-Type: multipart/form-data; boundary=--------------------------abcdefg123456789

----------------------------abcdefg123456789
Content-Disposition: form-data; name="HotspotId"

75
----------------------------abcdefg123456789
Content-Disposition: form-data;  filename="filenamehere.jpg"
Content-Type: image/jpeg

binary-data-here
----------------------------abcdefg123456789--
```

Delete an image by id:

```
DELETE https://devtest-farmtracking.vfltest.dk/api/images/123
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

5.2.8 HotspotType

Get all HotspotTypes with this request:

```
GET https://devtest-farmtracking.vfltest.dk/api/HotspotTypes
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

5.2.9 HotspotSubType

Get all HotspotSubTypes with this request:

```
GET https://devtest-farmtracking.vfltest.dk/api/HotspotSubTypes
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

To filter the HotspotSubTypes on a specific HotspotType use odata filtering:

```
GET https://devtest-farmtracking.vfltest.dk/api/HotspotSubTypes?$filter=HotspotTypeId
eq 1
Host: devtest-farmtracking.vfltest.dk
Authorization: Bearer <encodedToken>
```

5.2.10 MinAppVersion

This endpoint will return a single object with minimum required version for the app to function with the current API.

This endpoint does not require authorization

```
GET https://devtest-farmtracking.vfltest.dk/api/MinAppVersion
```

5.3 Environments

For development the Web API will be available in a test environment

5.3.1 Test

https://devtest-farmtracking.vfltest.dk/

5.3.2 Production

https://farmtracking.dlbr.dk/

5.4 Authentication / authorization

The authentication model for the FarmTracking application is currently not decided. It can be either ADFS or OAuth.

5.4.1 Authorization

Each request must include an Authorization header with the value “Bearer <encodedToken>”

<encodedToken> is produced by base64 encode the SAML token obtain through the login flow

5.4.2 Authorization Error Codes

When authorization fails the API will respond with http code “401 Unauthorized”. If possible a JSON error entity is also sent with detailed information.

Error entity

Name	Data type	Description
Code	String	Error code
Message	String	Detailed error description

Error codes

Code	Description
401.50	Error decoding token
401.51	Error reading token
401.52	Token expired
401.99	Unknown error

5.5 DLBR Log

5.5.1 Description

VFL has a company wide logging system called DLBR Log. It exposes a SOAP service with one method, PersistLogEntries.

An LogEntry entity looks like:

Name	Data type	Description
Domain	String	Typical the name of the executable
LevelType	log4net leveltyp – see format i example	110000 FATAL 70000 ERROR 40000 INFO 30000 DEBUG
Logger	String	Normally full namespace and name of class where log was written
Message	String	Log message
Exception	String	Exception – if any
Thread	String	Thread name/id – can be empty
Date	DateTime	Log date
UserName	String	User name
IpAddress	String	Device ip address
OperatingSystem	String	Device operation system
ProcessorFamily	String	Device processor info
TotalPhysicalMemory	String	Memory info
ApplicationName	String	Name of application
ActivityId	Guid	Used in special cases – can be 00000000-0000-0000-0000-000000000000

5.5.2 Example

```
POST https://devtest-www1-dlbrlog.vfltest.dk/loggingservice.svc HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction:
"http://dlbrlog.dlbr.dk/LoggingServices/Contracts/2012/04/LoggingService/PersistLogEnt
ries"
Host: devtest-www1-dlbrlog.vfltest.dk

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
      <s:Body>
            <PersistLogEntries
xmlns="http://dlbrlog.dlbr.dk/LoggingServices/Contracts/2012/04">
                  <request xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
                        <LogEntries>
                              <PersistLogEntryRequest>
                                    <Domain>com.farmtracking.vfl</Domain>
                                    <LevelType
xmlns:a="http://schemas.datacontract.org/2004/07/log4net.Core">

      <a:m_levelDisplayName>INFO</a:m_levelDisplayName>
                                          <a:m_levelName>INFO</a:m_levelName>
                                          <a:m_levelValue>40000</a:m_levelValue>
                                    </LevelType>

      <Logger>namespace.where.log.was.written</Logger>
                                    <Message>write message here</Message>
                                    <Exception>exception details here - if
any</Exception>
                                    <Thread>1</Thread>
                                    <Date>2015-01-21T19:53:04.9618532+01:00</Date>

                                    <UserName>fttest1</UserName>
                                    <IpAddress>10.1.4.73</IpAddress>
                                    <OperatingSystem>Microsoft Windows NT 6.1.7601
Service Pack 1</OperatingSystem>
                                    <ProcessorFamily>Intel(R) Core(TM) i7-2600 CPU
@ 3.40GHz</ProcessorFamily>

      <TotalPhysicalMemory>17135849472</TotalPhysicalMemory>
                                    <ApplicationName>FarmTracking</ApplicationName>
                                    <ActivityId>00000000-0000-0000-0000-
000000000000</ActivityId>
                              </PersistLogEntryRequest>
                        </LogEntries>
                  </request>
            </PersistLogEntries>
      </s:Body>
</s:Envelope>
```

Please make sure the xml-elements are in same order as shown above (don't ask me why)

One request can contains multiple `PersistLogEntryRequest` elements

5.5.3 Environments

5.5.3.1 Test

https://devtest-www1-dlbrlog.vfltest.dk/loggingservice.svc

5.5.3.2 Production

https://dlbrlog-service.dlbr.dk/loggingservice.svc