RE. CAP
Reinforcing CAP

PROJECT ACRONYM: RECAP
PROJECT NUMBER: 693171
PROJECT FULL TITLE: Personalised public services in support of the implementation of the CAP

WORK PACKAGE NUMBER & NAME:
WP.3  Service Integration and Customization

DELIVERABLE NUMBER & NAME:
D3.1  RECAP System Architecture

The project has received funding from the European Union’s Horizon 2020 Research and Innovation Programme under Grant Agreement No. 693171.
Grant Agreement Number: 693171
Acronym: RECAP
Project Full Title: Personalised public services in support of the implementation of the CAP
Start Date: 01/05/2016
Duration: 30 months
Project URL: www.recap-h2020.eu

Deliverable Number & Name: D3.1 RECAP System Architecture

Work Package Number & Name: WP3 Service Integration and Customization

Date of Delivery: 28/02/2017
Contractual: 28/02/2017
Actual: 28/02/2017
Nature: Report
Dissemination Level: Public
Lead Beneficiary: DRAXIS ENVIRONMENTAL SA
Responsible Author: Ioulia Anastasiadou (DRAXIS), Stelios Kotsopoulos (DRAXIS), Ifigeneia-Maria Tsioutsia (DRAXIS)
Contributions from: 3-OPEKEPE

DOCUMENT HISTORY:

<table>
<thead>
<tr>
<th>Versions</th>
<th>Issue Date</th>
<th>Stage</th>
<th>Changes</th>
<th>Contributor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>15/02/2017</td>
<td>Internal review</td>
<td>First version to be internally reviewed</td>
<td>DRAXIS</td>
</tr>
<tr>
<td>2.0</td>
<td>24/02/2017</td>
<td>Review</td>
<td>Review Feedback</td>
<td>OPEKEPE</td>
</tr>
<tr>
<td>3.0</td>
<td>28/02/2017</td>
<td>Final</td>
<td>Final version</td>
<td>DRAXIS</td>
</tr>
</tbody>
</table>

© RECAP Consortium, 2016
This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.

Disclaimer
Any dissemination of results reflects only the author’s view and the European Commission is not responsible for any use that may be made of the information it contains.
Table of Contents

Table of figures ...........................................................................................................................................5
Table of tables .............................................................................................................................................5
Executive summary ...........................................................................................................................................6
1 Introduction ..................................................................................................................................................6
2 Functional requirements ............................................................................................................................7
  2.1 RECAP System roles .............................................................................................................................7
  2.2 User-stories .............................................................................................................................................7
  2.3 RECAP system components ...................................................................................................................12
  2.4 Wireframes ...........................................................................................................................................14
    2.4.1 Mockups .........................................................................................................................................21
3 Non-functional requirements .....................................................................................................................29
  3.1 Hardware ..............................................................................................................................................29
  3.2 Performance ..........................................................................................................................................29
  3.3 Availability ..........................................................................................................................................30
  3.4 User friendliness .................................................................................................................................31
  3.5 Security .................................................................................................................................................31
4 RECAP architecture and layers description ..........................................................................................31
  4.1 Data layer services ...............................................................................................................................33
    4.1.1 Remote sensing Services ..................................................................................................................33
      4.1.1.1 Tier 1: Open data collection and pre-processing .......................................................................34
      4.1.1.2 Tier 2: Core processing ...............................................................................................................34
      4.1.1.3 Tier 3: Value ................................................................................................................................35
    4.1.2 Data flow between RS component and the rest of system ...............................................................35
  4.1.3 Documentation warehouse ................................................................................................................37
  4.1.4 Warehouse of spatial data for visualization .......................................................................................38
  4.1.5 PostGIS / PostgreSQL as a central DB ............................................................................................38
  4.2 Middleware layer services ..................................................................................................................38
    4.2.1 Workflow services ..........................................................................................................................39
    4.2.2 Geoserver for spatial data visualization .........................................................................................39
4.2.3 Spatial services .................................................................................................................. 40
4.2.4 Extractor services .............................................................................................................. 40
4.2.5 SDK services ..................................................................................................................... 40
4.3 Frontend layer services ....................................................................................................... 40
4.3.1 Web application ................................................................................................................ 41
4.3.2 Web Map client .................................................................................................................. 41
4.3.3 Mobile / Tablet application .............................................................................................. 42
5 RESTful API and Software Development Kit (SDK) ............................................................... 42
5.1 Why RESTful API? ............................................................................................................... 42
5.1.1 RESTful API endpoints description .................................................................................. 42
5.1.1.1 Verbs ............................................................................................................................. 42
GET............................................................................................................................................. 43
POST........................................................................................................................................... 43
PUT............................................................................................................................................... 43
PATCH......................................................................................................................................... 43
DELETE ....................................................................................................................................... 43
5.1.1.2 Exception handling ....................................................................................................... 44
5.1.1.3 Data validation .............................................................................................................. 44
5.1.1.4 Pagination ...................................................................................................................... 44
5.1.1.5 Endpoints ....................................................................................................................... 45
5.1.1.6 JSON Web Token (JWT) ............................................................................................... 47
5.1.1.7 Rest API documentation ............................................................................................... 47
5.2 SDK description ................................................................................................................... 48
6 Translation .............................................................................................................................. 50
7 Deployment methodology ......................................................................................................... 51
7.1 7.1 Software development methodology .............................................................................. 51
7.2 7.2 The Git workflow and versioning .................................................................................... 52
7.3 7.3 Continuous integration .................................................................................................... 52
7.4 7.4 Staging environment ....................................................................................................... 52
7.5 7.5 Testing methodology ....................................................................................................... 53
7.5.1 7.5.1 API & Integration testing .......................................................................................... 53
7.5.2 7.5.2 User Interface Testing .............................................................................................. 53
Table of figures

Figure 1: RECAP system architecture diagram ................................................................. 32
Figure 2: RECAP Data layer ................................................................................................ 33
Figure 3: Workflow and architecture of the Remote Sensing Services component .......... 34
Figure 4: RS component architecture diagram ................................................................. 37
Figure 5: RECAP Middleware layer ................................................................................ 39
Figure 6: RECAP Frontend layer ...................................................................................... 41
Figure 7: JWT authentication mechanism ......................................................................... 47
Figure 8: RECAP Swagger Web page ............................................................................... 48
Figure 9: SDK used to build custom mobile applications, communicating with the RECAP platform via the REST API .................................................................................................................................................. 49
Figure 10: Scrum in a picture (source: canvasinfotech.com) ........................................... 51

Table of tables

Table 1: Data exchange between the RS component and the rest of the system ............... 36
Table 2: Indicative Rest Api end points ............................................................................. 47
Executive summary

This deliverable aims at analyzing the user stories and stakeholders' requirements and mapping them into application requirements. It describes the functional and non-functional software requirements. In this document, we define the set of techniques and technologies that we consider to be the most appropriate in order to produce the system architecture and design, comprising of the overall architecture, the Application Programming Interface (API) description and Software Development Kit (SDK) characteristics to support any technical integration with external systems. Finally, in the last chapter we describe the development methodology and testing methodology as well as any other housekeeping stuff that will help us organize the development process.

1 Introduction

The overall objective of RECAP is to develop and pilot test a platform for the delivery of public services that will enable the improved implementation of the Common Agricultural Policy (CAP). RECAP will integrate open and user-generated data into added value services, co-designed, and co-created by public authorities, farmers, and agricultural consultants.

The RECAP project is focused in particular on supporting the administration of the Basic Payments Scheme (BPS). Following the 2013 CAP reform EU farmers are able to apply for direct payments through the Basic Payments Scheme, to act as a safety net in the form of a basic income support. Cross compliance and Greening are two mechanisms that are linked to this payment to ensure more environmentally-friendly farming approaches and deliver continued food security and safety in Europe. There are three main ways that the scheme is monitored; administrative checks of paperwork, visual on-the-spot checks (OTSC) and Control with Remote Sensing (CwRS). The RECAP project is focused on developing a cloud-based Software as a Service (SaaS) platform to assist the stakeholders, namely farmers, agricultural consultants, Paying Agencies and inspectors during the process of the implementation and monitoring of the scheme.

This document details the software architecture of the RECAP platform. As the first step, the user requirements described in the D2.2 report on user requirements in relation to the RECAP platform were analyzed in order to define functional and non-functional requirements of the system. Functional requirements are articulated through User Stories. Regarding the non-functional requirements, requirements in terms of hardware, interfaces, user friendliness, performance and security are identified. The various platform components are described from a technological point of view with the support of diagrams that highlight the different modules, software interfaces as well as communication protocols used. Finally, the deployment methodology that will be used is described.

As we describe in section 7, we intend to use Scrum as our software development methodology. Thereafter, some implementation details will be defined as we go along and we will have a clearer understanding of what needs to be done. At this stage, we have defined the architecture and the technologies that we intend to use but as the development of the platform proceeds and new findings come to the surface, we will readjust our decisions and more details will be added to the description of the APIs.
2 Functional requirements

2.1 RECAP System roles

The RECAP platform will provide services to four different types of users: farmers, agricultural consultants, Paying Agencies and inspectors.

- **Paying Agencies** will use RECAP to retrieve valuable information in order to optimize and improve the selection of farms to be inspected on Cross Compliance and Greening rules. They will also use RECAP for communication with the farmers in order to prevent breaches of the rules and respond to farmers’ requests for support.

- **Inspectors** will be able to access the data and material related to the farm to be inspected in the case of remote inspection or to download any data needed for OTSC in the RECAP mobile/tablet application environment. They will be provided with a checklist of rules specific to the farm that will guide them through the inspection process.

- **Farmers** will be provided with the checklist of rules they are obliged to be compliant with according to their BPS applications. They will be guided through the rules and measures that they should take. RECAP platform will allow them to store the data, records and documents that they need to obtain, keep and present to inspectors. Using the mobile application, they will also be able to collect and store timestamped and geotagged photos from fields. Finally, they will be able to communicate with the PA in case of exceptional circumstances.

- **Consultants** working for farmers will use RECAP to improve the advice to farmers in terms of the Cross Compliance procedure. They will mainly use the spatial component to access and analyse spatial data from various sources and to access the data collected by farmers.

2.2 User-stories

In this section of the deliverable, the functional requirements of the RECAP system are presented in the form of User Stories. A User Story is a simplified description of a requirement that defines who the user is, what does the user want delivered and why. This form fits well into Scrum that we will use as the development methodology.

The User Stories are derived from the users’ requirements that have been gathered for the RECAP platform, defined through involving end users during the co-creation process, and take into account the technical feasibility with regards to the different RECAP components. They will form our backlog throughout the development phase. In accordance to Scrum, we expect that that some of the User Stories will be amended, some will become obsolete and some new ones will be created during the co-creation and pilot phases.

The User Stories are organized below based on the different user roles.

**Sign in / Registration**

**US**: The user will be able to register to the RECAP platform.

During registration, it will be mandatory for the user to complete the following information:
First name
Last name
Are you the farmer-claimant? Yes-No (If answer is positive, the user can administer the farm’s account)
Email address
Password
Confirm password

**US**: The user will receive the confirmation e-mail in order to activate his account.

**US**: The user will be able to log in to the RECAP platform.

**US**: The user will be able to log out from the RECAP platform.

**US**: The user will be able to delete his account.

**US**: The user will be asked if he wants the system to remember him (along with a reminder that it is not recommended on shared computers).

**US**: The user will be able to reset his password in case he forgets it.

**FARMER**

**Profile settings/ Farm account admin**

**US**: The farmer will be able to use his BPS application to populate his farm’s profile.

**US [LT]**: The farmer will be able to fill in the farm’s profile (BPS application) because they will not be able to automatically populate the profile from the existing BPS application.

**US**: The farmer will be able to view his BPS application history and previous inspection results.

**US [GR, LT?, ES?, UK?]**: The farmer is able to keep the records on input/output/work diary.

**US**: The user who is the claimant – person responsible that the cross compliance rules are met can grant access to the farm’s account with different privileges to the other registered users (farms staff, agronomist/consultants, veterinary). The privileges will be: 1) can edit (add and delete material), 2) can view only, 3) can add material (images, documents, data, comments, etc.).

**US**: The farmer will be able to access his Document Manager (DM) where he can view/upload/delete/edit all the documents/images uploaded to RECAP (from both web and mobile application) and apply filters based on type/tags/CC rule/etc. The farmer will be able to assign each document/image to a cross compliance rule. Images with the tag “internal” will be visible only to the farmer.

**US [all except LT]**: Upon request from the inspector, the farmer can open his farm data to the inspector.
**US [LT]:** The farmer before signing up should be informed that he is obliged to share his electronic folder with the PA/inspectors in the event of an inspection.

**US:** The farmer will be able to send a request for LPIS inspection to the PA.

**Checklist**

**US:** The farmer will be able to see the list of cross compliance rules (SMR, GAEC, Greening) along with a short description of each rule, that are applicable to his farm.

**US:** When the farmer selects a cross compliance rule he is presented with: the more detailed description of the rule, the parcels that the rule is applicable to, the list of tasks that farmer must/ must not do through the year to meet the compliance rule and records/documents that should be kept/obtained in order to be shown in case of an inspection. This list will be related to the crops and agricultural activities that the farmer registered in his farm profile. The key dates will be presented for each of the tasks when applicable.

**US:** The farmer will be able to set reminders for the tasks he should perform.

**US:** The farmer will receive reminder alerts through the system for the key dates during the year.

**US:** The farmer will be able to insert/delete/edit/save the relevant data, images, comments for each of the cross compliance rules. When documents/images are already assigned to a CC rule, they are automatically visible.

**US:** The farmer will be able to connect a document (e.g. permits) to the referent LPIS parcel.

**US:** The farmer will be able to open his farm's data related to cross compliance rules (images, comments, numerical data, documents etc.) and make it visible to the PA and inspectors upon request by the inspector.

**US:** The farmer will be able to perform mock self-inspection based on the criteria used by the relevant paying agency and the data that he entered in RECAP. The results will indicate for each of the cross compliance rule, and for each parcel that the rule is applicable to, its status: 1) No problems identified, 2) Breach detected, 2) Not enough data provided. If breach is detected, farmer will get the details of the problem (including the problems detected by mapping calculations). For each task, when applicable, questions are provided to check whether the task is done (in case that the compliance can’t be checked automatically). In case of not enough data provided, the system will indicate what data is missing.

**US:** The farmer will be able to apply Greening Calculator (Part 1 and Part 2) based on the data stored in the farm profile (area of arable land, number of crops, area of each crop, etc.).

**US:** The farmer will be able to view the list of edits in the farm account with time, date and who made the edits (log history).

**US:** The farmer will be able to receive alerts if the RS component identified a problem (e.g. the green cover does not follow the guidelines) or the age of a crop reached maximum.

**US:** The farmer will be able to report a problem/exceptional event to the paying agency that includes: description of the problem, additional photos, data etc.
**Spatial component**

**US:** The farmer will be able to search his parcels by LPIS number.

**US:** The farmer will be able to visualize his parcels (LPIS) in the web map interface and map sub-divisions-subparcels (including EFA - Ecological Focus Areas) by visual interpretation of RS imagery and to add land use to each new polygon (e.g. one subparcel is wheat, the other is path etc.). When the edits are saved, the system calculates area of each of the subparcels. The results are stored in the database. The system compare vectorized areas with the BPS declared data to check for inconsistencies.

**US [GR-ES]:** The farmer will be able to insert the dose of pesticide used per hectare and the system calculates the total amount of pesticide used for the selected area. There is an option to select the units. The system will inform them how they are doing regarding the maximum amount of pesticide that the farmer can use per season and keeps a record of it at the end of the season which is added to the farmer’s electronic record.

**US:** The farmer will be able to choose recent RS/ ortophoto image after a flood occurred, and to digitize flooded area and the size of it is calculated by the system.

**US:** The farmer will be able to select a LPIS parcel/field and to get data related to the parcel/field (including subparcels and geotaged photos and age of the crop related to the field)

**US:** The consultant working for the farmer will use the GIS functionalities and all available data (including LPIS, open data, RS data, data generated by the farmer, etc.) in the Spatial component to retrieve useful information.

**US:** The farmer can view the locations of the problems alerted by the RS component (e.g. the green cover does not follow the guidelines).

**Mobile application**

**US:** The farmer will be able to take photos from within the app. The photos will be automatically timestamped/geotaged. The farmer will be able to assign the photo to a parcel and add tags (from predefined list of tags, including the cross compliance rule the photo is related to) and comments. Images with the tag “internal” will be visible just for the farm staff.

**US:** The farmer will be able to upload the photos to the RECAP database when connected to internet.
PAYING AGENCY (PA)

Admin

**US:** The PA has administration rights on RECAP platform: creating, editing and deleting official user accounts, granting privileges to official users (e.g. inspector’s privileges).

**US:** The PA uploads the information and help materials for farmers (including tutorials, examples of good and bad practices etc.).

Selection of farms

**US:** The PA will be able to use Extractor component to analyze the available large datasets and combine data in order to uncover meaningful (and previously unknown) pattern.

**US:** The PA will be able to use GIS functionalities and spatial data within the Spatial component to retrieve information useful for selection of farms to be inspected (e.g. show all farms that lay within Nitrate Vulnerable Zones (NVZs)).

**US:** The PA will be able to enter farm identification number (in UK case it is Signal Business Identifier (BSI) which is a unique farm identifier) to retrieve farm profile data including the data on cross compliance if the farmer opened it to the PA.

Communication with farmers

**US:** The PA receives the information that the farmer submitted when reporting accidents/exceptional circumstances and replies providing advice on how to manage/rectify the unintentional problem.

**US:** The PA receives farmer's request for LPIS inspection.

INSPECTOR

Checklist

**US:** The inspector sends a request to the farmer selected for inspection to open the farm’s data. In case of Lithuania, the farm’s data will be automatically visible to the inspector.

**US:** The inspector will be able to enter the farm identification number (in UK case it is Signal Business Identifier (BSI) which is a unique farm identifier) in order to get the data on the farm which is to be inspected.

**US:** The inspector chose type of the inspection for the selected farm: 1) Boundary Inspections – LPIS updates, 2) Eligibility inspections, 3) Cross Compliance Inspections: a) Responsive inspections, b) Full Inspections, c) Animal ID and movement inspections. The inspector chose if the inspection will be remote or OTSC.

**US:** The inspector will be able to see previous inspection results for the farm (in the form of checklist with all requirements that were checked).
US: The inspector will be provided with the pre-filled inspection form (checklist) for the selected farm and for the selected inspection type. If the farmer opened his data related to cross compliance it will be visible to the inspector for inspection via RECAP.

US: The inspector will be able to add/delete/edit/save the relevant data, images, comments for each of the cross compliance rule for the inspected farm.

US: The inspector will be able to make decision on each of the cross compliance rules applicable to the inspected farm by selecting: 1) No problems identified, 2) Breach detected. Or, in Greek case, each answer gets a score, all of which gets added up and a final score is produced by an algorithm.

US: The inspector will be able to generate a pdf report, sign it electronically and add it to the farm profile. The signed pdf can optionally be sent to the farmer’s e-mail.

US: Optionally, the inspector sends notification (48hours/14days/3hours... in advance) to the farmer that his farm will be inspected.

US: In the case of remote inspection, the inspector can send e-mail to the farmer through RECAP to notify him about the problem identified and to suggest rectification or to ask for additional documents.

Spatial component

US: The inspector will be able to use RECAP spatial component with GIS functionalities to view and retrieve information from different vector and raster layers (e.g. LPIS, subparcels data, Natura2000, Land cover data, RS data etc.).

US: The inspector will be provided with the route for selected inspection sites (parcels within the farm to be inspected).

Mobile application

US: The inspector will be able to select spatial data layers (images, LPIS boundaries, designated areas, etc.) and download it for the selected farm parcels in order to prepare for the OTSC inspection.

US: The inspector will be able to take photos from within the app. The photos will be automatically timestamped/geotaged. The farmer will be able to add tags (from predefined list of tags, including the cross compliance rule the photo is related to) and comments.

US: The inspector will be able to upload the photos to the RECAP database when connected to internet.

US: The inspector will be able to edit the LPIS data during inspections.

US: The inspector will be able to capture new boundaries of the LPIS parcels (using GPS or drawing on the map) and upload it into RECAP database.

2.3 RECAP system components
Based on the user requirements and the user stories as these are described above, we concluded that the system should be comprised of the following components:

**The Workflow Component**

This component will provide to farmers, consultants and inspectors check lists of Cross Compliance rules applicable to the farm based on information from the BPS application submitted by the farmer. It will guide farmers and inspectors with personalized information on the procedures that they have to follow. The component will allow the farmers to store the required information and material for each CC rule and make it available to inspectors if needed. The component will also generate notifications to farmers based on calendar of key dates.

**Spatial Component**

The role of Spatial Component is to provide PA officers, inspectors and consultants access to the spatial information generated by the Remote Sensing component and allow performing spatial analysis and queries upon data stored in the RECAP database. The Spatial Component will also provide a web cartographic environment for visual representation and integration of spatial data stored in the RECAP database and the external spatial data.

**Extractor Component**

The Extractor Component will be a business intelligence analysis tool that enables public officers to analyze the available large datasets and be able to combine data in order to uncover meaningful (and previously unknown) patterns. It will be based on techniques such as online analytical processing, reporting and text mining, comprising a component able to analyze large datasets available and to uncover meaningful (and previously unknown) patterns. Data used in the extractor tool can be combined with spatial and temporal dimensions allowing to compare with older data or data from other areas which will help to determine the proper CAP implementation.

**Application Component**

The RECAP web and mobile applications aim at bringing regulations and compliance at the fingertips of the farmers and the inspectors. Their key features will be: simplicity, mobility, flexibility. Two mobile applications will be developed: A smartphone optimized app dedicated to the farmer and a tablet-optimized app focusing on the inspector’s needs. The mobile apps will be mainly used by the farmers for collecting data on the field, and by the inspectors for OTSC. The RECAP web application will act as the main user environment for the Paying Agencies, farmers completing their self-assessment, Independent Software Vendors (to use the SDK component) and agricultural consultants. We have foreseen a system of user accounts with varying privileges/access permissions.

**Remote Sensing Component**

The Remote Sensing Component will extract useful features, such as indices relevant for crop monitoring, from a time series of primarily open Earth Observation data. These features will be fed to an automatic crop classification model, in order to identify the type of crop(s) cultivated in a specific parcel, and other characteristics. The classification output will be combined with geo-information data available to public organizations, and model this information for enabling the identification of potential breaches of cross-
compliance by public authorities and inspectors. User-generated data such as in-situ photos will be also used for offline validation purposes.

SDK Component

The **SDK** will be used by anyone who wants to develop their own services on top of RECAP using design tools, libraries and communication with the database under an open approach. The SDK Tool will provide the specific components so that other developers can create add-ons that extend the functionality of RECAP, e.g. Report Generator Tool, Calendar Events and Follow Up Reminders, RECAP Telemetry Interface (IoT), E-learning Tool.

### 2.4 Wireframes

A set of wireframes were created to help the consortium members better understand the main functionalities of the RECAP platform but also to be used in the co-creation process so that we get feedback from the end users. It is very important to note that the wireframes are not intended to depict the look and feel of the final application, but are very simple drawings that aim at visualizing the main features of the application. Hence, the wireframes below have nothing to do with the final UI of the RECAP platform.

**Login/ Register page**

![Login/ Register page wireframe](image-url)
The register/login page serves as the gate for the users to access the RECAP services.

**Administration page**

![Administration page screenshot]

The owners of the farm can add other users to the farm’s account and manage their privileges. For example, farm staff will be able to add documents and field data related to compliance measures but will not be able to edit farm’s profile. In a similar way, the Paying Agency can manage the inspectors’ accounts.
Farmer is able to populate his farm’s profile from his BPS application and to add/edit the data.
Farmer can go through the checklist of cross-compliance rules that apply to his farm and get the detailed info of tasks he should accomplish with the related deadlines (if exist). Farmer will also be able to upload data/documents related to the rule that is required as a proof of compliance. Farmer can check compliance if all required data is provided.
Reminders

Farmers are able to set and receive reminders for key dates of tasks related to cross compliance rules.
Map viewer and GIS

Farmer can inspect spatial data of his farm. Paying Agency can perform advanced queries and spatial analysis (as support for decision which farms are to be inspected) upon data stored in RECAP database using available GIS tools. Inspectors will use the map viewer to perform remote inspections and also during OTSC inspection (on their tablets).
Inspector goes through the checklist of cross-compliance rules that apply to the farm under inspection. The results of the inspection are visible to the farmer. For each rule, detailed findings are provided.
Using the mobile application, both farmers and inspectors can collect timestamped/geotaged field data (e.g. photos) as the evidence of compliance (or non-compliance).

### 2.4.1 Mockups

At the moment, a first version of the mockups is created for farmer’s part of the RECAP application. Mockups are created for both, desktop and mobile applications. Some of them are presented below. During the project, mockups will be corrected according to the feedback from the users (farmers, inspectors and PAs).

#### Login/register
Dashboard/reminders

Edit account
Farm profile
## CCRule/Tasks

### Cross-compliance rule: GAEC6

**Rule description**

Enforcing a popular belief, some given in and supply random text, it has roots in a piece of classical Canon literature from Dr. D.C., making it over 2000 years old. Written by a unknown author, it is usually not recognized in modern teaching today. First, every rule's text and its explanation should be reviewed and written down in a rule's file. Then the rule is uploaded to the RECAp system's database. The final step is to review the rule with the entire team. Then, any rules that do not match the rules within a database are removed.

**Must do**

<table>
<thead>
<tr>
<th>Task</th>
<th>Comments</th>
<th>Documents</th>
<th>Image</th>
<th>Type</th>
<th>Date</th>
<th>Time</th>
<th>Document ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>No comments</td>
<td>No documents</td>
<td></td>
<td>Task 1</td>
<td>2019/02/13</td>
<td>11:30 PM</td>
<td></td>
</tr>
<tr>
<td>Task 2</td>
<td>No comments</td>
<td>No documents</td>
<td></td>
<td>Task 2</td>
<td>2019/02/27</td>
<td>08:19 PM</td>
<td></td>
</tr>
<tr>
<td>Task 3</td>
<td>1 comment</td>
<td>1 comment</td>
<td></td>
<td>Task 3</td>
<td>2019/03/04</td>
<td>02:44 AM</td>
<td></td>
</tr>
<tr>
<td>Task 4</td>
<td></td>
<td></td>
<td></td>
<td>Task 4</td>
<td>2019/03/25</td>
<td>07:51 AM</td>
<td></td>
</tr>
</tbody>
</table>

Report problem [here](https://www.example.com).

## Document manager

### My Documents Manager

**Search My Documents**

Enter your search query and hit enter (space separated keywords):

- Type
- Tag
- Document Type
- Label

Search your query and hit enter or search:

<table>
<thead>
<tr>
<th>Title</th>
<th>Type</th>
<th>Tag</th>
<th>Document Type</th>
<th>Label</th>
<th>Date</th>
<th>Time</th>
<th>Document ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document 1</td>
<td>Office Document</td>
<td>Tag1</td>
<td>Task 1</td>
<td>Task 1</td>
<td>2019/02/13</td>
<td>11:30 PM</td>
<td></td>
</tr>
<tr>
<td>Document 2</td>
<td>Office Document</td>
<td>Tag2</td>
<td>Task 2</td>
<td>Task 2</td>
<td>2019/02/27</td>
<td>08:19 PM</td>
<td></td>
</tr>
<tr>
<td>Document 3</td>
<td>Office Document</td>
<td>Tag3</td>
<td>Task 3</td>
<td>Task 3</td>
<td>2019/03/04</td>
<td>02:44 AM</td>
<td></td>
</tr>
<tr>
<td>Document 4</td>
<td>Office Document</td>
<td>Tag4</td>
<td>Task 4</td>
<td>Task 4</td>
<td>2019/03/25</td>
<td>07:51 AM</td>
<td></td>
</tr>
<tr>
<td>Screen Shot 367-20-26</td>
<td>JPG Image</td>
<td>Tag5</td>
<td>Task 5</td>
<td>Task 5</td>
<td>2019/01/15</td>
<td>07:51 AM</td>
<td></td>
</tr>
</tbody>
</table>

Advanced Search

Copyright © RECAp 2017
Service Integration and Customization

RECaP system architecture

Log history

Log History for Sir Oliver

<table>
<thead>
<tr>
<th>Username</th>
<th>Action</th>
<th>Description</th>
<th>Object</th>
<th>Date</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sir Oliver Farm Owner</td>
<td>Profile updated</td>
<td>Profile's farm is set to default</td>
<td><a href="https://dashboard.recap.com/owners/sir-oliver">https://dashboard.recap.com/owners/sir-oliver</a></td>
<td>2019/07/01</td>
<td>75.22 KB</td>
</tr>
<tr>
<td>Sir Oliver Farm Staff</td>
<td>Activation revoked</td>
<td>User's farm is revoked</td>
<td>/profile/sir-oliver</td>
<td>2018/07/17</td>
<td>627 KB</td>
</tr>
<tr>
<td>Mrs. Food Consultant</td>
<td>Status revoked</td>
<td>Mrs. Food's status is revoked</td>
<td><a href="https://dashboard.recap.com/owners/mrs-food">https://dashboard.recap.com/owners/mrs-food</a></td>
<td>2018/03/04</td>
<td>52.44 KB</td>
</tr>
<tr>
<td>Bob Haystack Inspector</td>
<td>Profile updated</td>
<td>Profile's farm is set to default</td>
<td><a href="https://dashboard.recap.com/owners/bob-haystack">https://dashboard.recap.com/owners/bob-haystack</a></td>
<td>2018/07/01</td>
<td>75.24 KB</td>
</tr>
<tr>
<td>Alan Good Postal Agency</td>
<td>Profile updated</td>
<td>Profile's farm is set to default</td>
<td><a href="https://dashboard.recap.com/owners/alan-good">https://dashboard.recap.com/owners/alan-good</a></td>
<td>2018/07/01</td>
<td>75.24 KB</td>
</tr>
<tr>
<td>Bob Haystack Inspector</td>
<td>User blocked</td>
<td>User is blocked for 24 hours</td>
<td><a href="https://dashboard.recap.com/owners/bob-haystack">https://dashboard.recap.com/owners/bob-haystack</a></td>
<td>2018/07/01</td>
<td>75.24 KB</td>
</tr>
</tbody>
</table>

Log history
3 Non-functional requirements

3.1 Hardware

The RECAP platform should be reliable, resilient and needs to cope with peak usage. We reckon that the inspection season and the months that precede this season, will be the most demanding period for the platform. Inspectors will download and use a massive amount of data from the RECAP platform and farmers will often use the platform in order to make sure they comply with all rules. The RECAP platform will need to be at optimal functionality during that time of the year – maintenance, server requirements and help requests are likely to be at peak demand. The website must be efficient, quick and work without restrictions. All the hardware that will be used will be configured to support the above-mentioned requirements. We have concluded that the production server should have the following specifications in order to efficiently support all the RECAP functionalities.

a. CPU: six cores,
b. Intel® Xeon® E5-2620v3 or E5-2630v4 processor,
c. 32 GB RAM (guaranteed),
d. 600 GB disk space (100% SSD),
e. 100% SSD disk space,
f. UNLIMITED traffic !*,
g. 100 Mbit/s port,
h. Live support every day,
i. 365 days a year via e-mail and telephone, Root access,
j. Reboot web interface (preview),
k. OS reinstall console (preview),
l. VNC access, 1 IP address included,
m. /64 IPv6 network included,
n. 2 snapshots included.

3.2 Performance

As mentioned above we plan to build a fast and reliable system and in this chapter we mention the strategy that we plan to be following. The nature of the RECAP system demands CPU intensive Web Mapping technologies especially for tile rendering processes. For that reason, the system will use a tile caching mechanism named MapProxy. MapProxy is a very effective system to make the map and image services run up to 300x faster and is responsible to protect the system’s CPU from heavy load conditions. When we create a tile cache, the server draws the entire map in different scales and stores copies of the map images. The server can then distribute these images whenever someone asks for a map. It’s much quicker for the server to return a cached image than to draw the map each time someone requests it.

Another performance drawback of systems that handle spatial data files (GEOTIFF/vectors) is the huge amount of hard disk reads/writes actions. By attaching an SSD storage to the production server we will significantly increase the number and speed of concurrent I/O.

At database level, PostgreSQL stores the spatial and business layer’s data and also drives a major part of the API and SDK subsystems. Due to the relational nature of PostgreSQL, there are several steps that take place during a query execution and before it is ready for consumption by a client:
1. Transmission of the query string to the database backend
2. Parsing of the query string
3. Planning of query to optimize retrieval of data
4. Retrieval of data from hardware
5. Transmission of results to client.

In order to avoid executing the above steps every single time, we plan to fine tune the default configurations of PostgreSQL database by upgrading the buffer sizes in order to preload tables to RAM or by using internal caching. This way we can eliminate the database response delay and at the same time increase the concurrency of connections.

The RS System will be able to serve offline data to RECAP system using a pre computational philosophy. This way we can serve data to the end users directly instead of waiting for on the fly heavy calculation of RS.

By using an Apache module named “pagespeed” we minifie (i.e. remove whitespace, comments, etc.) and compress external CSS and JavaScript in order to speed up the performance of the Frontend layer.

Caching will be used to improve server performance by allowing commonly requested content to be temporarily stored in a way that allows for faster access. This speeds up processing and delivery by cutting out some resource intensive operations. The apache module mod_file_cache will accelerate the process of serving static files. The apache modules mod_cache and mod_cache_disk will probably be used for standard HTTP caching, to store responses and validate them when they expire.

Finally, we plan to use a web-browser based storage in order to store already seen tiles and avoiding the round trip of tiles over http.

3.3 Availability

The RECAP web platform will work well across different devices – phone, tablet, desktop/laptop and the mobile application will be accessible from mobile phones and tablets.

Users should be able to set certain tasks to happen only when Wi-Fi access exists to avoid high mobile data usage. The application should automatically sync with the online system when a Wi-Fi network is available. Furthermore, the mobile and tablet applications need to be accessible offline so that farmers with poor or no broadband can access the RECAP data and make changes (e.g. take a photo). Inspectors should be able to download data to his/her tablet for the purpose of using RECAP application in offline mode. The RECAP application will automatically upload changes when the connection is adequate.

Mobile, tablet and desktop applications will be accessible in areas of poor internet access (or other interruptions). They need to be responsive to internet connection speed. RECAP will detect the internet connection speed and provide appropriate image resolution or type, e.g. in case of poor internet access user will get low resolution image.

Finally, farmers close to the border between two countries may have high costs if they access internet from the neighbouring country. The RECAP application will offer the option to only use certain networks when data usage is high.
3.4 **User friendliness**

The end users of the RECAP platform might be people with poor technical background. Similar existing platforms follow bad practices, presenting too many options and making it difficult for the users to navigate to the option they need. Taking that into account, the platform needs to be simple to use and intuitive, with clear and self-explanatory screens, easy to use maps and colours that would allow it to be adopted by the users. It needs to be easy to access and get started with. We plan to follow all the latest guidelines around usability and efficiency.

Education material, help, tutorials, examples of good and bad practices etc. should be available to users in order to offer a better understanding of the entire CC system and the platform itself.

3.5 **Security**

A high level of data security and reliability is required for the RECAP platform in order for farmers, consultants and Paying agencies to trust it. Permissions will be used in order to prevent unauthorized use of data. Below we list a set of rules that we will be following to make sure we build a secure system.

- Installed software on both the RS component and the rest of the system will ban IPs that show malicious signs like too many SSH or SMTP password failures, seeking for exploits, etc. (i.e. fail2ban).
- Both systems will be accessible through SSH protocol by not using the common protocol ports (i.e. port 22). In addition, developers and sysadmin will use SSH keys.
- Both systems will provide HTTP over TLS connections for encrypted communication.
- PostgreSQL will be only be accessible from internal systems.
- A UFW firewall will manage the IP tables of both systems.
- The Mod_security Apache module will secure the RECAP platform from common attacks like SQL injections, XSS attacks etc.
- HTTP access to the RS component will be possible only from RECAP system's IP in order to avoid unknown connections.

4 **RECAP architecture and layers description**

In this section, we describe all the decisions we have made around the architecture of the system. The system is described in three layers corresponding to the data, the middleware and the front-end parts of the system. The data layer is responsible for gathering and storing all the data, the middleware layer is where all the business logic lives and the connection between the Data and the Frontend layers and finally, the Frontend layer is the medium through which the users interact with the system. An overall architecture diagram is presented in Figure 1 and in the subsequent sections we focus on each one layer.
Figure 1: RECAP system architecture diagram
4.1 Data layer services

The data layer is responsible for storing the RECAP data. It consists of two parts:

- **PostGIS / PostgreSQL DB** – all of the non-spatial data (authentication, user-generated data etc.) and all of the spatial vector data (parcel and subparcel borders, position of geotagged photos, Land Cover, Natura2000 etc.) will be stored here;
- **File System** – this is the main storage unit for documents, pdfs, and photos. Access to these data is possible through links which are kept in PostGIS / PostgreSQL DB. Also, raster data, that will be used for visualization through the Geoserver, is stored here;

The data from this layer is accessible to the end users through Middleware layer (REST API) and Frontend layer.

![RECAP Data layer](image)

4.1.1 Remote sensing Services

The remote sensing component of the RECAP architecture (bottom left) will serve as a stand-alone processing unit that ingests geo-information from the RECAP platform, and publishes results that are in turn ingested by RECAP. At the core of the component is a PostGIS geodatabase, responsible for managing and to some extent processing of time-series of satellite data and classification products. Geoserver will be used for disseminating data and products.

The workflow of the component is presented in Figure 3, while the corresponding architecture comprises of three tiers, following the route of a satellite image along the value chain:

1. The satellite data collection and pre-processing tier.
2. The core-processing tier, where the basic remote sensing products are generated.
3. The value tier, where the products are wrapped to be moved to the RECAP platform and statistical information is generated.
Figure 3: Workflow and architecture of the Remote Sensing Services component

4.1.1.1 Tier 1: Open data collection and pre-processing

The data tier will build scripts to ingest data that are necessary to perform the cross-compliance analysis. Data to be consumed include:

- Open satellite data, e.g. primarily Sentinel-2, Landsat and Sentinel-1. A synchronizer shall be developed to automatically download and pre-process a new satellite image, when available. These data will be pre-processed on the fly to generate spectral, spatial and temporal features (e.g. multi-temporal NDVI).
- Commercial satellite data, acquired in the framework of RECAP and/or provided by the Paying Agencies. These will be Very High Resolution, at least 4-band (infrared will be included) multi-spectral data.
- Open geospatial data, including LPIS, land use / land cover, and other geo-information.
- RECAP geospatial and imagery data, i.e. the inspectors’ in-field observations or user generated photos that can be used for offline validation purposes.

4.1.1.2 Tier 2: Core processing

This level is responsible for forming an input feature space based on the data and pre-processing products of the previous tier, that is fed to the classifier to identify the different types of crops. Different approaches will be used, starting from decision tree analysis and proceeding to machine learning techniques (e.g. neural networks). The products that will populate the PostGIS will include:

- Crop identities, classified based on fully automatic processors, written in Python (exploiting SciPy and NumPy fundamental package for scientific computing with Python).
- Products of the historical evolution of the crop at its different phonological phases.
4.1.1.3 **Tier 3: Value**

Upon classifying the different crops through a time-series approach, vector products will be bundled and enriched with additional attributes. A statistical analysis will be conducted to detect periodic and nonperiodic changes in the parcels, and set appropriate flags.

The information about the classified crops statistical information will be transferred in the RECAP platform and cross checked with the farmers’ declarations to what concerns the pre-selected cross-compliance rules. Upon the detection of a breach, an alert will be issued, in a format to be agreed at a later stage.

It may happen, as in the case of the Greek Paying Agency, that a system to detect cross-compliance breaches based on manual photo-interpretation of remotely sensed imagery is already in place. In this context, we will explore the technical feasibility to either migrate the cross-compliance analysis directly to the Paying Agency, or even integrate this analysis module to the RECAP platform.

Overall, the data flow in the Remote Sensing component will be in vector format, while raster data (raw imagery, spectral indices) will be kept only if there is some clear value for the project.

4.1.2 **Data flow between RS component and the rest of system**

The following table describes the data flow between the RS component and the RECAP platform, using both the RS API and OGC services.

The RS component will ingest data from the RECAP platform in online and offline mode. Results of in-field inspected crops, user generated photos and Very High Resolution satellite imagery will be requested on demand, in order to make improvements to the in-house developed algorithms. In addition, the crop types as declared and/or inspected in the previous year will be requested from RECAP. The RS component will publish in an offline mode updated RGB imagery (GEOTIFF) based on Sentinel-2 data, to be used as background in the declaration centers of the paying agencies.

In an online mode, the RS component will publish classification products (GeoJSON file) for the requested geometries (e.g. a certain parcel or parcels).

<table>
<thead>
<tr>
<th>RECAP platform Data Needs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functionality</strong></td>
</tr>
<tr>
<td>Operational</td>
</tr>
</tbody>
</table>
Remote Sensing Component Data Needs

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Service Description</th>
<th>Input from NOA</th>
<th>Output to NOA</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Operational/Offline</td>
<td>Get inspected crops by PA/inspectors</td>
<td>all related records</td>
<td>RECAP API: /v1/noa/inspected</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(geometry and</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>attributes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Get farmer’s uploaded photo</td>
<td>photos</td>
<td>RECAP API: /v1/noa/photos</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Get Very High Resolution data</td>
<td>Parameters:</td>
<td>Satellite data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• date</td>
<td>RECAP API: /v1/noa/satellite/?date=</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• bbox</td>
<td>dd-mm-yy&amp;bbox=xxx.xx,xxx.xx,xxx.xx,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>xxx.xx</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Data exchange between the RS component and the rest of the system

As it is shown in the figure below, the communication will occur through the REST API of DRF and OGC services via UMN Mapserver and MapProxy. Specifically, DRF will provide the appropriate service (e.g. /v1/rs?filter={geometry}) in order to give RS results for each incoming geometry in GeoJSON format. UMN Mapserver is an open source geographic data rendering engine written in C and is indicated for serving large volume raster data according to OGC services (Web Map Service, Web Map Tile Service, Web Coverage Service, Web Feature Service). Additionally, MapProxy is an open source tile server, reading WMS sources and caching data on file system.

The Celery package will be used for the scheduled tasks, which is a pure python library. PostgreSQL/PostGIS will be used for storing spatial data and attributes.
4.1.3 Documentation warehouse

Documentation warehouse is a unique File System of all of the documentation uploaded by users like photos, pdfs, docs etc. Every document gets its own system generated name and an access link for the document is saved in PostGIS / PostgreSQL DB. Storing and accessing the documentation is performed by the Restful API.
of the Mayan EDMS (Electronic Document Management System) in the Workflow services (Middleware Layer).

### 4.1.4 Warehouse of spatial data for visualization

Unlike the Documentation warehouse, this warehouse has the role to store spatial data to be used for map layers rendering. Raster spatial data are stored in the File System and vector spatial data are stored in PostGIS / PostgreSQL DB. As such, they are ready to be used by the Geoserver. Storing and accessing these data is performed by the Geoserver’s Restful API. The separation of vector and raster data has been chosen for rendering performance reasons.

### 4.1.5 PostGIS / PostgreSQL as a central DB

PostGIS / PostgreSQL DB is a central part of the system due to the fact that Django, (the framework that will be used for the implementation of the middleware layer) will use this DB for implementing its models. Django models map to a database table, and provide a place to encapsulate business logic, storing the information from the user. It contains the essential fields and behaviors of the data you’re storing.

In order to populate the entity models and validate the inputs, Django uses the ORM system which is able to store the information in the corresponding database tables of the PostgreSQL / PostgreSQL relational DB. These tables are directly connected to the corresponding models of the ORM, keeping the development process fast and reliable. PostGIS and GeoDjango give additional possibilities for managing the spatial data.

Most of the data, i.e. information about PAs, inspectors, farmers and cross-compliance data will be stored here. Also, all of the logs (i.e. every user activity) will be stored in PostGIS / PostgreSQL.

### 4.2 Middleware layer services

The core of the RECAP system is the Middleware layer which represents a collection of Rest API endpoints. This layer is responsible for handling the retrieval, processing, transformation and management of the data coming from the Data layer. It is the connection between the Data layer and the Frontend layer. It is able to securely respond both to the web and mobile / tablet client applications, following the business information workflow. The Middleware layer is implemented using the Django Rest Framework (DRF), which is a powerful and flexible Django based toolkit for building Web APIs. It has authentication policies including packages for OAuth1 and OAuth2, and serialization that supports both ORM and non-ORM data sources.

An Apache web server serves the Middleware layer API. Using the RESTful patterns, the Middleware layer offers a set of endpoints where the Apache Web server is able to respond to the /GET, /POST, /PUT, /PATCH, /DELETE verbs from clients. This layer is also responsible for the security and integrity of the data stored, as it authenticates any request originated from the Frontend Layer services despite the type of client (mobile or web). Any CRUD operation to any Business entity (Users, Farms etc.) is filtered for not authorized access using a role based authentication (Farmer, Administrator etc.) through a mechanism called JWT.

All of the API endpoints can be classified to 3 sets, related to the component it refers:

- Workflow component API
- Spatial component API
- Extractor component API
Some of the Spatial component API endpoints relate to the Geoserver. Geoserver provides API endpoints for serving spatial data for visualization to the Frontend layer.

On top of the aforementioned REST API, the RECAP Software Development Kit (SDK) will be built. More details can be found in sections 4.2.5 and 5.2. In Figure 5, the architecture of the middleware layer can be seen.

### 4.2.1 Workflow services

Workflow services cover all of the API endpoints for managing data related to Farmers, PAs and Inspectors and their cross-compliance related activities. Mayan EDMS, a free open source Electronic Document Management System will be used to handle files uploaded via the platform, i.e. photos and documents. The RESTful part of the Mayan EDMS (built using Django REST framework) supports data versioning, full text searching, automatic OCR processing, tagging, etc. and will be adapted and upgraded to serve the RECAP application.

### 4.2.2 Geoserver for spatial data visualization

Geoserver is an open source server for sharing geospatial data. It has its own Rest API, but permissions and access to the data is regulated through the DRF. It is used for serving static spatial data to the web map client. Geoserver covers APIs for storing, getting and also for changing styles of spatial data for visualization.
The Geoserver in RECAP will be used to provide a standard interface for requesting a geospatial map image mostly in form of OGC WMS services, or if necessary in order to provide access to raw geospatial data via OGC WFS/WCS.

### 4.2.3 Spatial services

Because cross-compliance rules contain a lot of spatial rules, spatial services will be able to deal with it. Spatial services are implemented using GeoDjango, a Django spatial module that handles:

- Django model fields for OGC geometries and raster data.
- Extensions to Django’s ORM for querying and manipulating spatial data.
- Loosely-coupled, high-level Python interfaces for GIS geometry, raster operations and data manipulation in different formats.
- Editing geometry fields.

Spatial services will provide GIS tools like measuring, red lines tools or even more advanced tools for spatial analysis like buffering or proximity which are important for some of the cross-compliance rules, e.g. greening.

### 4.2.4 Extractor services

Extractor services will provide to the public officers the ability to extract the most important information from vast amounts of data, in order to uncover previously unknown patterns that may be relevant to current agricultural problems, thereby helping them to transform data into business decisions.

### 4.2.5 SDK services

On top of the aforementioned REST API, the RECAP Software Development Kit (SDK) exists, providing a wrapper over the REST API endpoints. An SDK represents a collection of one or more APIs, programming tools and documentation that enables a programmer to develop applications for a specific software package, software framework, operating system or hardware platform. In order to create applications, developers need to download an SDK and start coding. The SDK will be written in commonly used languages like Python, PHP etc. For example, having an SDK for Python, means that Python applications can be written to programmatically interact with the RECAP engine like they would do with any other Python library. In section 5.2 we describe in more details how the SDK will be build.

### 4.3 Frontend layer services

Through the Frontend layer users interact with the RECAP system. The User interface (UI) will be customized based on the User Role and/or device. The UI is implemented using the React library, whereas the mobile and tablet apps are developed using React Native. OpenLayers3 brings together spatial data from Geoserver and PostGIS DB in the form of a map which is embedded to the UI. Figure 6 represents the FrontEnd layer architecture.
4.3.1 Web application

The front-end part of the application will be implemented using React library, well known for great performance, modularity and scalability. Being developed at Facebook, and due to its current popularity within the web-developer community, React provides an excellent choice for developing component-based web-applications. With JSX (JavaScript XML) along with standard React library, code complexity will be greatly reduced. Apart from JSX, various other helper libraries will encompass React:

- Yeoman for project scaffolding
- Webpack for module handling and task automation
- Babel for ES2016 support
- React Router for routing
- Sass as CSS preprocessor
- Bootstrap for front-end components
- and many more

This will provide a stable toolset for the implementation of the RECAP web application, allowing the use of best practices established in current web-development trends.

4.3.2 Web Map client

OpenLayers JavaScript library will be used for implementing the web-mapping functionality. It has great support for both rasterized layers (layers could be served in different ways e.g. OSM, Bing, MapBox or OGC WMS) and row vector data (GeoJSON, TopoJSON, KML, GML, OGC WFS), mobile devices and modern web standards (WebGL), styling with CSS, and other customization options. Numerous features and high customizability make OpenLayers a proven solution for any web-mapping task.
4.3.3  Mobile / Tablet application

Along with React and the already mentioned helper libraries, React Native framework will be used for the purpose of developing the mobile and tablet applications. With only slight modifications of the web application’s codebase, the mobile apps will have native performance, while retaining all the web application’s functionality. Apart from the native app performance, cross-platform support is also a great advantage as we can use it to build both Android and iOS apps.

5  RESTful API and Software Development Kit (SDK)

5.1  Why RESTful API?

Recently, REST has become one of the most important technologies in Web applications and it will continue to play a central role as all of the new trends are API oriented. Every major development language now includes frameworks for building RESTful Web services. As such, there is a clear benefit to use RESTful API for the communication between the different layers and for assisting the SDK component.

As already mentioned, the Python Django REST framework will be used for the RESTful API implementation. It is a powerful and flexible toolkit for building Web APIs. Python Django framework is also chosen because Python together with R will be used in the Extractor component and GeoDjango will be used to build the Spatial component, i.e. Web GIS application.

REST provides a set of architectural constraints that, when applied as a whole, emphasize scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems (Roy Thomas Fielding, 2000). The advantage of using REST is that both frontend and backend could be changed but functionality would still remain the same.

5.1.1  RESTful API endpoints description

In a RESTful API, endpoints (URLs) define the structure of the API and how end users access data from our application using the HTTP verbs. Endpoints should be logically organized around collections and elements, both of which are resources. A resource is defined as an object with a type, associated data, relationships to other resources, and a set of methods that operate on it.

A description of the verbs that will be used follows.

5.1.1.1  Verbs

The HTTP verbs that we plan to use for the API are the following: GET, POST, PUT, PATCH and DELETE. GET and DELETE require no request body while POST, PUT and PATCH require the provided data to be encoded
as application/json and to be provided in the request body. Successful requests will return the 200 OK HTTP status code and a json encoded response body if available.

Server side errors that cannot be categorized will respond with a 500 and will try to inform the API user why this has happened.

GET
Retrieve one or more items (list of items) encoded in JSON (application/json).
eg. GET /api/document will return an array of all documents related to current BPS of the authenticated user.
eg. GET /api/document/1 will return a single document with the ID = 1.
eg. GET /api/document/?user=1 will return an array of documents that belong to the user with ID = 1.

POST
Creates a new item using the provided data.
eg. POST /api/document with request body:
   {
     "task": "1",
     "tag": "tag1",
     "document": document,
     "description": "Description of the document"
   }
will try to create a new document with tag and description related to tasks with ID = 1.

PUT
Complete replacement of an item using the provided data.
Requires that the complete item's data is provided, not only the changes.
eg. PUT /api/document/1 with request body:
   {
     "task": "1",
     "tag": "tag1",
     "document": document,
     "description": "Description of the document"
   }
will edit the document with ID = 1.

PATCH
Edits an item using the provided data.
Requires that the item's ID is provided with the data.
It defers from PUT in that it doesn’t need to list the complete items data but one can only supply the requested changes.
eg. PATCH /api/document/1 with request body:
   {
     "tag": "tag2"
   }
will change only tag of the document with ID = 1.

DELETE
Deletes a single item and optionally all the other items related to it.
eg. DELETE /api/document/1 will delete the document with ID = 1.
5.1.1.2 Exception handling

The Django REST framework handles various exceptions and returns appropriate error responses. In each case, DRF will return a response with an appropriate status code and content-type (e.g. JSON). The body of the response will include any additional details regarding the nature of the error.

The most commonly used exceptions are:

- `AuthenticationFailed` - "401 Unauthenticated" or "403 Forbidden"
- `NotAuthenticated` - "401 Unauthenticated" or "403 Forbidden"
- `PermissionDenied` - "403 Forbidden"
- `NotFound` - "404 Not Found"
- `MethodNotAllowed` - "405 Method Not Allowed"
- `NotAcceptable` - "406 Not Acceptable"
- `ValidationError` - "400 Bad Request"

Full list of all handled exceptions can be found in [Django REST framework documentation](https://www.django-rest-framework.org/api-guide/exceptions/).

On the client-side, server exceptions will be handled via pop-up and modals alongside a brief notice stating the HTTP response code and/or a descriptive message.

5.1.1.3 Data validation

Data validation needs to ensure that the RECAP application is robust against all client’s input data, i.e. to ensure that the data is strongly typed, correct syntax, within length boundaries, contains only permitted characters, or if numeric is correctly signed and within range boundaries, etc.

Django REST framework already includes database validators like validators for uniqueness, nullability, integer, character fields, etc. Besides that, custom data validators will be implemented in two ways:

- field-level validators - validation for only one field
- object-level validators - validation that requires access to multiple fields

If there is any validation error, it will raise an exception and return a 400 Bad Request response with an error explanation.

Each input field will have a set of validation rules assigned to it which, if not satisfied, will present error messages adjacent to the field. Once the user provides the correct data, error messages will be hidden.

5.1.1.4 Pagination

Pagination allows you to modify how large result sets are split into individual pages of data, providing faster initial page load.

The Django REST framework includes support for customizable pagination styles. The pagination API can support either:

- Pagination links that are provided as part of the content of the response.
- Pagination links that are included in response headers, such as Content-Range or Link.
The default pagination style will be set globally, using the Django REST framework setting keys:

- **DEFAULT_PAGINATION_CLASS**:
  - PageNumberPagination
    ```python
    "count": 1023
    "next": "http://recap.com/api/documents/?page=5",
    "previous": "http://recap.com/api/documents/?page=3",
    "results": [
      ...
    ]
    }
  
  - LimitOffsetPagination
    ```python
    "count": 1023
    "next": "http://recap.com/api/documents/?limit=100&offset=500",
    "previous": "http://recap.com/api/documents/?limit=100&offset=300",
    "results": [
      ...
    ]
    }
    ```

- **PAGE_SIZE** - number of elements per page in response

If there is a need to parameterise the pagination settings, it is possible to override the global pagination settings for a particular Rest API endpoint.

### 5.1.1.5 Endpoints

The table below presents examples of Rest API endpoints. The list is indicative aiming to give an idea to the reader of the extent the the API is going to cover. The full list of endpoints will be available through Swagger documentation (5.2.4 Rest API documentation).

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Indicative endpoints</th>
<th>Possible verbs</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>/v1/authentication/registration</td>
<td>POST</td>
<td>Successful registration with token / Fail with reason</td>
</tr>
<tr>
<td>Login</td>
<td>/v1/authentication/login</td>
<td>POST</td>
<td>Successful login with token / Fail with reason</td>
</tr>
<tr>
<td>Logout</td>
<td>/v1/authentication/logout</td>
<td>POST</td>
<td>Successful logout / Fail with reason</td>
</tr>
<tr>
<td>User details / edit</td>
<td>/v1/user/{user_id}</td>
<td>GET PATCH DELETE</td>
<td>User details / Fail with reason</td>
</tr>
<tr>
<td>Insert BPS</td>
<td>/v1/bps/create</td>
<td>POST</td>
<td>Successful creation of new BPS / Fail with reason</td>
</tr>
<tr>
<td>BPS details / edit</td>
<td>/v1/bps/{bps_id}</td>
<td>GET PUT PATCH DELETE</td>
<td>BPS details / Fail with reason</td>
</tr>
<tr>
<td>Grant access</td>
<td>/v1/grant</td>
<td>GET POST</td>
<td>Get list of grants related to authenticated user, Successfully added grant / Fail with reason</td>
</tr>
<tr>
<td>Grant access details</td>
<td>/v1/grant/{grant_id}</td>
<td>GET PUT PATCH DELETE</td>
<td>Grant access details / Fail with reason</td>
</tr>
<tr>
<td>Grant access confirmation</td>
<td>/v1/grant/{grant_id}/confirm</td>
<td>POST</td>
<td>Confirmation of grant access / Fail with reason</td>
</tr>
<tr>
<td>History (BPS applications and previous inspections)</td>
<td>/v1/bps/{user_id}/history</td>
<td>GET</td>
<td>History by User / Fail with reason</td>
</tr>
<tr>
<td>Get BPS's log history (list of edits)</td>
<td>/v1/bps/logs</td>
<td>GET</td>
<td>Log history of authenticated User / Fail with reason</td>
</tr>
<tr>
<td>Checklist (CC rule list) according to BPS</td>
<td>/v1/bps/checklist</td>
<td>GET</td>
<td>Checklist of current BPS for authenticated User / Fail with reason</td>
</tr>
<tr>
<td>Tasks related to specific CC rule</td>
<td>/v1/bps/checklist/{cc_id}</td>
<td>GET</td>
<td>CC rule with related tasks / Fail with reason</td>
</tr>
<tr>
<td>BPS parcels</td>
<td>/v1/bps/parcels</td>
<td>GET POST</td>
<td>Get current BPS parcels, Successfully added parcel to BPS / Fail with reason</td>
</tr>
<tr>
<td>BPS parcel details</td>
<td>/v1/bps/parcels/{parcel_id}</td>
<td>GET PUT PATCH DELETE</td>
<td>BPS parcel detail / Fail with reason</td>
</tr>
<tr>
<td>Documents</td>
<td>/v1/documents</td>
<td>GET POST</td>
<td>Get all documents and photos related to current BPS, Successfully added document or photo / Fail with reason</td>
</tr>
</tbody>
</table>
5.1.1.6 JSON Web Token (JWT)

JSON Web Tokens are an open, industry standard RFC 7519 method for securely representing claims between two parties. The RECAP’s Rest API is based on the JWT authentication mechanism. When users login or register to the system, the API provides them with a unique encoded token with a specific expiration date that the user’s devices have to send back each time they need authenticated information from the platform.

![JWT authentication mechanism](image)

5.1.1.7 Rest API documentation

Swagger is a powerful open source framework backed by a large ecosystem of tools that helps with designing, building, documenting, and consuming RESTful APIs. All of the RECAP Rest API endpoints will be documented in one Web page which also provides basic testing options with different parameters and permissions.

Below is an example of RECAP’s Swagger Web page:
5.2 SDK description

With the RECAP SDK, developers will be able to write their own custom applications that will interact with the RECAP engine. The SDK will be built on top of the REST API, providing a wrapper over the REST API endpoints, so that developers can write custom applications with fewer lines of code that:

- Search RECAP data,
- Manage RECAP configurations and objects.
- Integrate search results into their applications.
- Log directly to RECAP.
- Present a custom UI.
The RECAP SDK initially will be developed and offered to developers using the PYTHON programming language but in the future it could also be expanded to other programming languages too.

RECAP SDK, as any other SDK, will be designed for developers to use. That requires an easily readable coding style, a modular object-oriented architecture that localizes the impact of changes and clean documentation with code samples serving as examples to show how the components can be used.

The RECAP SDK will consist of three main modules:

Authentication module

When using the SDK, users must authenticate against the central RECAP authentication JWT (JSON Web Token) mechanism before they can use the rest of the service. The SDK keeps the returned JWT token in...
memory for the duration of the active SDK process run, offering a ready for use valid token to subsequent core services. Any core SDK function seeks for a “in memory” token, before being able to perform its business target. If a token is not found, the authentication module is responsible to handle expired the tokens triggering the authentication cycle again when it is needed.

In order for the SDK to receive a token, the developer needs to authenticate via a valid username and password. Important part of the Authentication component is the configuration file where developers will need to enter their credentials once, during the installation process of the RECAP SDK into their development environment.

Core Services Module

The main SDK functions will be called in accordance to the ones in the RECAP REST API described above. A bundle of methods of the Core Services Class will be responsible to implement the appropriate calls from the SDK to the REST API, offering a suitable way to the developers to integrate their systems directly with RECAP.

As mentioned, the RECAP SDK calls need to use an active JWT token in order to be verified against the RECAP platform and be allowed to perform the essential actions. This requires a close connection between the Core Services Module and the Authentication Module, by maintaining an Authentication instance inside Core Services, offering authentication control to all SDK services.

Finally the Core Services will implement the Singleton pattern in order to avoid multiple connections (and multiple tokens) with the RECAP server during the same process. Due to this pattern the module will be responsible to handle error exceptions or connectivity and timeout issues which occur from the SDK or the API centrally.

Converter Module

Converter is the module which will be responsible to convert the API responses to the appropriate SDK objects based on the programming language version of the RECAP SDK. It will be also responsible to convert SDK objects to POST parameters according to the API Endpoint needs.

6 Translation

The translation of the RECAP application will be done in three ways:

1. Adding additional attributes in the RECAP Database
   Character or text fields in the RECAP Database will have additional fields with prefix tr_* if translation is needed (e.g. CC rules and Tasks description).
2. In Python code - message files
   Django provides utilities to extract the translation strings into a message file. It uses Python’s standard library gettext. The message file is a convenient way for translators to provide the equivalent of the translation strings in the target language. This approach will be used e.g. for translation of exceptions.
3. Frontend - template react-intl library from Yahoo! provides seamless integration of JavaScript Internationalisation API standard in React. It provides support for string interpolation, internationalization of dates, numbers, pluralization, and supports over 150 languages.

Every pilot will be able to use RECAP platform in English and native language.

### 7 Deployment methodology

#### 7.1 Software development methodology

Agile approach has been chosen for the development of the RECAP service platform, because it provides flexibility and fast response to change, allowing adaptive planning and evolutionary development while keeping code simple, testing often, and delivering functional bits of the application as soon as they're ready.

Scrum is the specific framework that will be used, dividing work to groups of cross-functional, self-organizing teams. The product is built in a series of fixed-length iterations, called sprints, that give teams a framework for shipping software on regular intervals. Every sprint begins with the sprint planning meeting, in which the Product Owner and the technical team discuss and decide which stories will be moved from the product backlog into the sprint backlog. The process is iterative and incremental, maximizing the team's ability to deliver quickly and respond to changing and new user requirements, while attempting to build a potentially shippable (properly tested) product increment on every iteration.

![Figure 10: Scrum in a picture (source: canvasinfotech.com)](canvasinfotech.com)
Starting point for the development are the user requirements, described through User Stories (US). Each US describes (on a very high level) the needs of a specific user regarding the platform. From these stories functional requirements are derived, so each story is expected to have the right amount of information for the team to be able to estimate the effort needed to implement it. Once the development is completed, testing is passed, bugs are fixed and the product owner approved the story - the story is marked as done.

In the case of RECAP the pilot partners will represent the product owner role in Scrum. The process will be harmonized with the activities of Co-production of services that will take place within the scope of the WP2: Users’ needs analysis & co-production of services. The outcomes of the co-production workshops will feedback to the RECAP platform development.

7.2 The Git workflow and versioning

Git-flow branching strategy will be used on top of Git source-control management, allowing developers to separate codebase for features, releases and hotfixes. While Git-flow remains nothing but convention for regular Git branching features, no extra requirements are needed for its usage. For further information on git-flow conventions, refer to official introductory blog post.

The Git-flow philosophy is that the develop branch is intended to be the default branch where most of the work will happen, and the master branch keeps track of production-ready code. So, the master branch will be used in production, and the develop branch will be used in staging environment.

Version prefixes will be added to the Rest API endpoints during the development phase (e.g. /v1/api/uk/bps), which will be manually changed for every new version.

7.3 Continuous integration

In order to reduce integration problems between several parts of RECAP and to allow a rapid development process, a CI approach has been adopted based on Jenkins. Jenkins is a tool which enables a team to build and test a project continuously making it easier for developers to integrate changes to the project, and easier for users to obtain a fresh build.

Apart from deploy automation, Jenkins has excellent support for automated testing. Once properly set up, Jenkins will provide enough info for both end-users and developers about the project regarding build status, test success rates, code coverage, and many more.

7.4 Staging environment

After unit and integration testing, code will be put in a production-like staging environment. This environment will be used throughout the development of the platform for testing purposes. Each component of the platform will reside in a separate Docker container.

The staging environment will be on completely separate server from the production environment. It will be secure by the Apache authentication and only pilots will be provided with credentials.
7.5 Testing methodology

In this section we describe the testing approach that we will be following during the development of the RECAP platform. By having a thorough testing methodology we want to verify and validate the platform from the very low level technical details all the way to the User Interface. Our approach will be updated accordingly for the duration of the project and any specific testing requirements and considerations will be taken into consideration as they come along.

In order to get clear insight regarding the application feature and structure, principles of test-driven development (TDD) will be pursued, mostly during the development of user interface. That way the developer will have a clear overview about code status during changes made by specification change or code refactoring, which will greatly improve development stability over time.

Apart from following TDD principles, manual testing from the perspective of user stories (US) will be applied as well. Towards the end of the sprint, we will allocate a specific amount of time to test the stories that have finished development. The stories will be tested based on the acceptance criteria that are defined within the US and the list of requirements. Bugs found as part of this process will be logged in our issue tracking system and will be fixed before the end of the sprint if possible. In some cases and after a common decision among the developers, the tester and the product owner some bugs might slip to another sprint based on the severity and the impact of the bug. Once all bugs are fixed, a final overview of the delivered functionality will take place to verify we are ready to go.

At this stage all the changes will be moved from the local testing server to the staging server so that it can be tested in conjunction with the latest updates of the other components and so that partners can have access to all the new features developed as part of the last 2 sprints. Partners will have the opportunity to give us feedback in the form of issues within Pivotal Tracker which will be taken into consideration and be prioritized accordingly.

Automated tests will also be written in order to ensure that new changes don’t break existing functionality.

7.5.1 API & Integration testing

The REST API handles all the communication between the various modules of the RECAP platform and needs to be tested in order to ensure its successful operation. We will test the API’s endpoints by sending requests with specific parameters and making sure that proper data and status codes are returned in the response.

Whenever a new module becomes available, we will integrate it into the RECAP platform and perform integration testing by passing the necessary inputs to the module and validating that it produces the expected outputs. The communication between the “new module” and the rest of the system will be tested and we will make sure that if the specific module is not working, the system will not fail as a whole.

7.5.2 User Interface Testing

The RECAP application will be tested in the latest versions of all common browsers (Chrome, Firefox, Safari, IE, Mobile browsers) in order to ensure that the look and feel experience will be the same for all users no matter what browser they are using.

Jest library will be used for testing the user interface, as well as for unit testing. Being developed by Facebook provides great reliability when it comes to testing React applications, but it is also suitable for JavaScript code in general.
7.5.3 Security Testing

We will ensure that the RECAP platform is always secure and protected from malicious attacks. In this section we describe the security policies and rules that we will be following:

**Authentication / Authorisation**

Only authorised users can have access to specific information

**Cross-site scripting**

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. Our platform will be using Laravel’s XSS protection filters and middleware to filter html output. In all users’ input that will be displayed on screen, we will be entering malicious scripts to check that the malicious act is prohibited and the script is only displayed as text.

**SQL injection**

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. Our web and mobile application will only talk to the database through the RECAP API. The API will filter user’s input for malicious scripts and ensure the request comes from an authorized user. All suspicious actions will be logged and an email will be sent to the technical team. In order to test it, we will be entering some malicious sql scripts and make sure these scripts are not executed.

**Insecure direct object reference**

Insecure Direct Object References allow attackers to bypass authorization and access resources directly by modifying the value of a parameter used to directly point to an object. We will be preventing insecure direct object reference by:

- Enforcing access control to ensure the user is authorized for the requested object.
- Enforce validation on both client and server side in order to avoid the use of proxy tools that can bypass client side validation.
- Using per user or session indirect object references: This approach can be used to prevent attackers from directly targeting unauthorized resources. For example, use a drop down list of resources authorized instead of database key for the current user to limit the user input. The application has to map the per-user indirect reference back to the actual database key on the server

Whenever a URL contains an id/string that corresponds to a user’s specific resources we will be manually testing that changing the id cannot lead to accessing another user’s resources.

7.5.4 User Acceptance Testing (UAT)

User Acceptance testing will be performed by the pilot partners who will verify that the platform was developed according to the requirements. As described above, functionality will be delivered in iterations and partners will be able to give us feedback through our issue tracking tool.
8 Conclusion

In this deliverable, we have described the architecture of the RECAP platform, its components and the implementation plan that we plan to follow. In addition, it describes all the tools that will be used for the development of RECAP and the principles around security and testing that will be followed. Software and hardware requirements as well as an extensive description of the API points that will be developed are also listed in this document.

This document’s target audience is mainly the technical partners of the consortium so that we all have a common understanding of RECAP’s architecture. It will act a reference and a source of information for all the partners.