Design Document

Abbo Giulio  
Accordi Gianmarco  
Bonetti Massiliamo

**1.Introduction**

**2.Architectural Design**

**A. Overview: High-­‐level components and their interaction**The highlevel architecture of the SafeStreets’ system is highlighted in the below Figure 1:

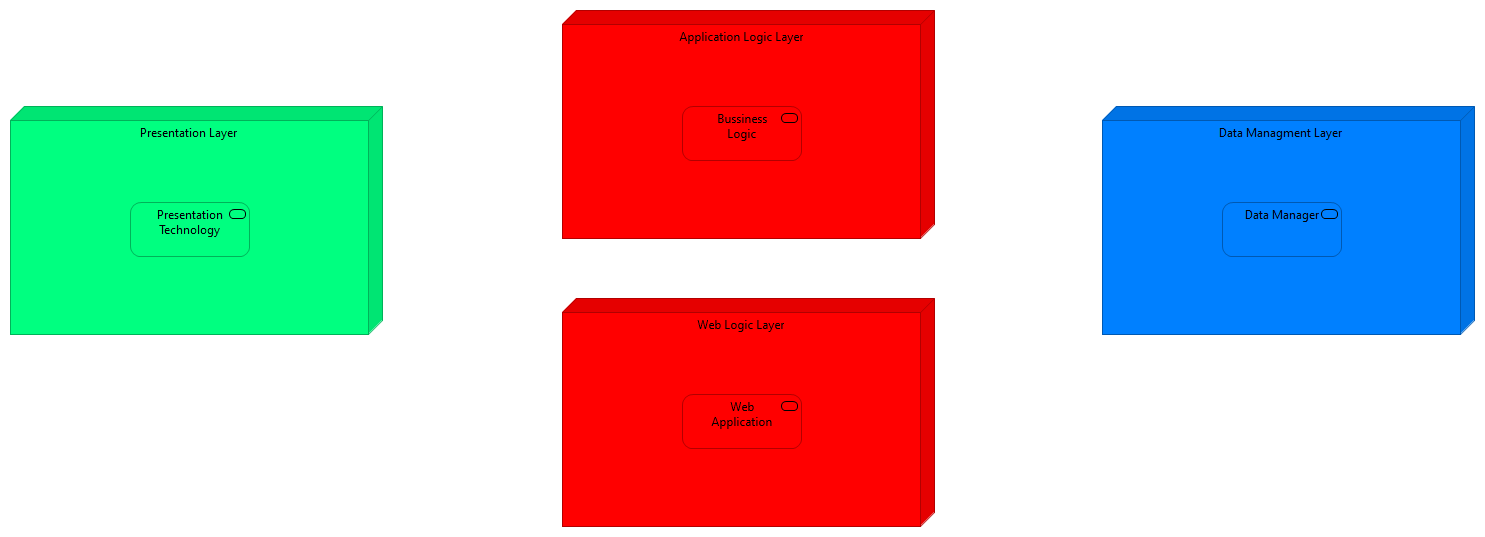


Figure 1

The defined architecture is a Four Tier architecture in which each components is briefly described below:

* *Presentation Layer* is the part that manage the visualization of the data and the possible interaction with the system in a human readability way, and also human friendly as much as possible, so this part knows how to show to the final user the results that comes from the Application or from the Web layer, also the user interact with this layer in order to interact with the system by using the application or the web access point through a browser;
* *Application Logic Layer* implements all the business logic of the Safestreets’ system, it receive all the request from the application on the users’ devices, and also the request coming from the web access, then it elaborates them, by retrieving all the information contained in the data manager, also it takes care of managing the data, by integrate them with the data provided by the municipality;
* *Data Management Layer* has the task of managing the physical allocation of the data, and to responds to: the queries that come from the application layer, and to store the data that the application layer wants to memorize.
* *Web Logic Layer* is used to respond to the web application request made by the user, but it doesn’t implement any type of business logic, so it’s only used to decouple the logic from the web visualization of the data, then all the requests made by the user through a web page are redirected to the application layer

The different levels of abstraction allow to manage different functionalities offered by the system on different machines, that need only to implement the required interface, allowing the replicate of the different machines, in order to scale in case of necessity and to be fault tolerant. The user is an aware of the different distribution of the levels: their distribution must be as possible transparent; he only needs to communicate throw a graphic interface. The data used by the system are obtained directly through the users’ registration, but it also need to retrieve this data from the municipality data manager, so periodically the system will integrate the new data obtain by the municipality with the data present inside the data manager of the Safestreets’ System.

To allow the communication to system different from the SafeStreets’ system the architecture makes use of adapter, this allows all the other component inside the system to use the same sets of operation, but the implementation of the adapter will change during time based on the implementation of the third party it refers to.  
The different level of abstractions are decoupled as much as possible, this means that they communicate through well defined interfaces, that allows us to extend some layer if necessary or to change them, for example it we want to change the data manager is sufficient to change the component on the Application Level.

**B. Component View**  
The Safestreets’ System is composed by a component structure defined in the below Figure 2:

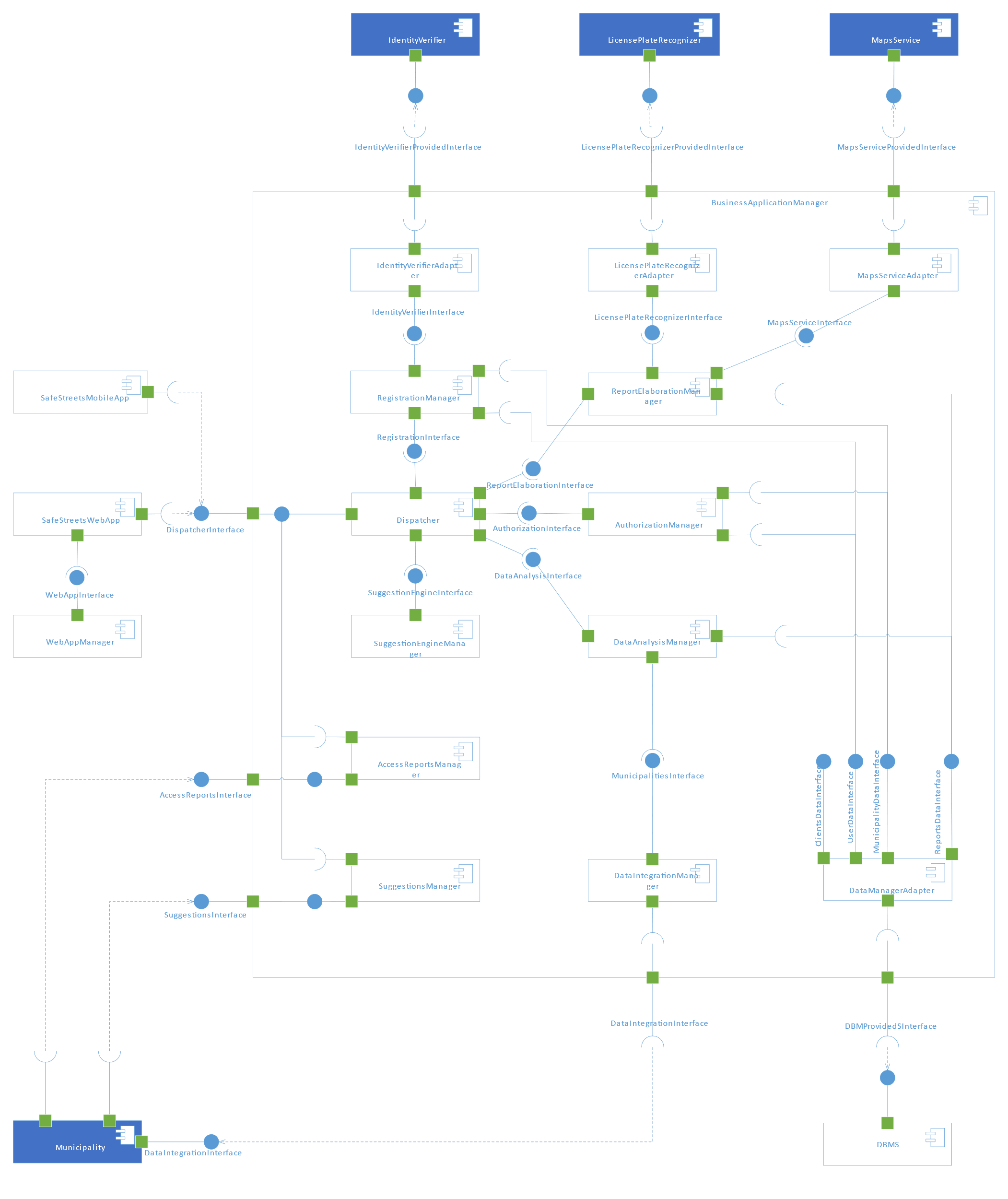


Figure 2 – Component View of the SafetStreets’ System

The figure illustrated how the components are linked between them: the figure highlight ,as already state in the high level architecture, that the module communicates among them through well-defined interface, so the figure defined which components expose an interface and which other components are interested in the same interface.  
In this chapter we will defined into details all the components that characterize our system.  
The blue coloured components indicates which component the system relies not, that are not develop with the system, in fact the system will use the services expose by some third parties, so the other components will be implemented by the software to be, if not contrary explicated.

The following list define the sets of tasks carried out by each single component that is external to the implementation to the *Business Application Manager*:

* ***SafeStreets’ Mobile App*** is the component that works on the client’s device, in fact the user interact with the system also trough a graphical interface that is similar to the one developed for the web page, this component allows the user to see how to interact with the Safestreets’ System, this means that this component sends directly to the *Business Application Manager* component all the requests made by the user, so it needs to work with the *Dispatcher Interface* of our system, in fact all the request will be dispatched by the *Dispatcher* component(in a transparent way to the user), the component also display to the user all the data that come from the *Business Application Manager*;
* ***Maps Service*** this component provides to the system a geographical meaning to geographical space the system works with, the *Maps Service* implemented by a third party and not during the development of the system: the system relies on a third parties that offers this service(like for example any GIS), so an adapter will be used to connect the *Maps Service* to the system, this service needs to provide an interface that allows our system to retrieve the address of the violations, and also to get an overlay map that we can modify in order to provide a better graphical representation to the user, and in general it helps the system to have a better understanding of the geographical space;
* ***SafeStreets’ Web App*** is another method that the user can use in order to interact with the SafeStreets’ system, it is a similar style to the Mobile App, but in this case it can be used to any kind of internet browser, this in fact will contact the *Web App Manager*, through the *Web App Interface*, and it will provide the required page, then the component can make the business logic requests directly to the *Business Application Manager*;
* ***Web App Manager*** takes care of all the requests coming from the Web App of the system, this means that it has to provide all the page requested and due to the fact that it has no understanding of the business rules, he simply forward the request to the *Business Application Manager*, and when it responds the *Web App Manager* will sends back to the User’s Web App a web page with the results;
* ***Municipality Manager*** is the component managed and deployed by the Municipality, this components needs to communicate to different interfaces defined by our system in order to: obtain the suggested intervention through the *Suggestions Interface*, allow the municipality to access the submitted violations reports sent by the users through the *Access Reports Interface*, and if the municipality allow us to access their repository with violations registered by them, they need to expose the *Data Integration Interface*, so as to improve the analysis of the data provided by the SafeStreets’ system, that periodically ,or when needed, will try to retrieve the latest violations from the municipality, and then it memorize them;
* ***License Plate Recognizer*** as highlighted by Figure 2 this component will not be implemented by the system, but the system will rely on a third party that offers this service, that carries out the task of recognize the plate from the report’s photos provided by the user, as already state in the RASD the response that we get from this component will be trusted by the system, so it will not be subject to other verification, and in order to communicate with the component the system implements an adapter;
* ***Identity Verifier*** the system relies on this component developed by a third party, that offers the service of verify the identity of the user, so when the user provides its credentials during the registration, the system use the *Identity Verifier Provided Interface*  in order to trust the user’s provided identity, similar to the *License Plate Recognizer* we trust the results obtained through this service;
* ***DBMS*** this component is responsible of the physical allocation and management of all the data used by the system, it exposes the *DBMS Provided Interface*, that will be used by the system to query the data and also to memorize the required data, the exposed interface will be wrapped by an adapter internally to system, in order to easily change the implementation of the query or of the technology.

Now we take a closer look to the subcomponents that implement the ***Business Application Manager***, it provides the business logic of our system:

* ***Dispatcher*** the figure makes clear that the dispatcher is used as an ingress point to the request coming from the different access method to the system, so the component will implement the *Dispatcher Interface*, used by the external component to communicate with the system, the dispatcher than is able to redirect the request based on his type, to the different interface to the other component, as better clarified in Figure 3:  
  A close up of a map

  Description automatically generated

Figure 3 – Interface used by the Dispatcher

the Dispatcher need to use the interface provided: by the *Suggestions Manager* component, to complete the request made by the Municipality, by the *Registration Manager* in order to complete the registration requested by the user or by the municipality, by the *Report Elaboration Manager* that will take care of elaborate the report sent by the user, by the *Authorization Manager* that will very the credential of the user and his access rights, by the *Data Analysis Manager* to retrieve the analysis of the data requested by the user or by the municipality;

* ***Registration Manager*** this component will get from the *Dispatcher* the registration request of the user, so in this case it needs to verify the user’s identity through the service of the *Identity Verifier*, accessible with the *Identity Verifier Provided Interface*, and then if everything is fine it memorize the use in the system by using the *User Data Interface*, or it gets the registration request from the municipality, that contains the contract code of the registration, that will be verified, and then it memorize the registration by using the *Municipality Data Interface*;
* ***Identity Verifier Adapter*** as already mentioned in the description of the *Identity Verifier* component, the access to the *Identity Verifier Provided Interface* is not made directly through the system, but the access needs to pass from this adapter, that expose the *Identify Verifier Interface*, of which the implementation is used as a wrapper of the real interface, thus to allow to system to be decoupled from the implementation of the Identity Verifier: in fact if the Identity Verifier change, is sufficient to change the implementation of this component;
* ***License Plate Recognizer Adapter*** similar to the previous component also this one is an adapter, that expose the *License Plate Recognizer Interface*, that can be used by all the internal component, to retrieve the license plate from the photo provided by the user’s report, this interface is used as a wrapper for the *License Plate Recognizer Provided Interface* exposed by the *License Plate Recognizer,* this allows us to easily change it;
* ***Maps Service Adapter***  also this component is used to expose an interface, the *Maps Service Interface,* that is used as a wrapper to real interface exposed *Maps Service Provided Interface* exposed by the Maps Service, this component is used to the internal component the possibility of understand the geographical coordinates, and to manipulate the data;
* ***Report Elaboration Manager*** it receives the report sent by the user through the Dispatcher, so it implements the *Report Elaboration Interface*, when a report is received it verifies the identity of the user, and then if necessary it validate also the position of the violation or to recognize the license plate present in the report’s photo, and ,as stated in the RASD, the result retrieved from the *Maps Service*  and the *License Plate Recognizer* is trusted, also the component have the task to notify the user if the something goes wrong during the elaboration, then if something had gone right, the component use the *Report Data Interface* to memorize the data;
* ***Authorization Manager*** this component is responsible to verify the credentials provided by the user or the municipality during the login and to set the correct access rights for the municipality or the user, that will be used during its navigation in the system, so it gets the login request from the *Dispatcher* through the *Authorization Interface* that this component implements, and then he access to user or municipality data by using respectively the *User Data Interface* and the *Municipality Data Interface;*

2.F Component interfacesImmagine che contiene screenshot

Descrizione generata automaticamente

Here the main interfaces used in the component diagram are presented. The diagram also contains some other interfaces – the ones with the tag “Provided” – that are provided by external parties, also missing is the interface through which the web app fetches the code from the web app manager. These interfaces depend heavily on the implementation and are omitted here.

In the image, the first row contains the interfaces that interact somewhat directly with the outside; on the second line the interfaces that allow the internal workings are shown and in the third row there are the interfaces that allow the communication with the data layer. Among these it is worth pointing out that UserDataInterface and MunicipalityDataInterface extend ClientsDataInterface, this was done to clarify that they have some methods in common and that these methods do not depend on the access type.