Low Level Design Document -   
Comm Port 2017

**Chenoa Information & Software Services Pvt. Ltd.**

Document Control

**Change History**

| Version | Change Description | Date | Author |
| --- | --- | --- | --- |
| 1.0 | Draft Version | 08/08/2017 | Rajesh Raghunathan / Jagdish Sahu |
|  |  |  |  |

**Review Details**

| Review Date | Reviewed by | Comments |
| --- | --- | --- |
| <mm/dd/yyyy> | <Name of the Reviewer> |  |
|  |  |  |

**Approval Details**

| Approval Date | Approved by | Comments |
| --- | --- | --- |
| <mm/dd/yyyy> | <Name of the Approver> |  |
|  |  |  |

Contents

[Document Control 2](#_Toc491207555)

[1. **Purpose** 4](#_Toc491207556)

[2. **Functional Description** 4](#_Toc491207557)

[2.1 **Comm-Port System- Introduction** 4](#_Toc491207558)

[2.2 **CommPort System –** 4](#_Toc491207559)

[2.2.1 **Overview** 4](#_Toc491207560)

[2.2.2 **High Level Design** 5](#_Toc491207561)

[2.2.3 **Detailed - Low Level Design** 7](#_Toc491207562)

[2.2.4 **User Authentication Sequence Flow** 8](#_Toc491207563)

[2.2.5 **Sequence Diagram for Queue Mechanism** 9](#_Toc491207564)

[2.2.6 **Process Flow diagram** 10](#_Toc491207565)

[3. **Source code Management** 15](#_Toc491207566)

[4. **Logical Diagrams** 15](#_Toc491207568)

[5. **Core API** 17](#_Toc491207569)

[6. **Web API** 17](#_Toc491207570)

[7. **Infrastructure Service** 18](#_Toc491207571)

[8. **Technology Stack** 18](#_Toc491207572)

[9. **Deployment Process** 19](#_Toc491207573)

[9.1 **System Requirement** 19](#_Toc491207574)

[9.2 **Environment** 19](#_Toc491207575)

[9.3 **Deployment Diagram** 19](#_Toc491207576)

[9.4 **Publishing Comm-Port Exe** 20](#_Toc491207577)

# **Purpose**

The purpose of this document is to provide the details of functionalities required on Comm-Port system.

# **Functional Description**

## **Comm-Port System- Introduction**

The business function of CPAS (Comm-Port Area Scan) is to enable security personnel at the security gate (for incoming traffic) to respond to security threats that moving vehicles (incoming) may pose. By providing comprehensive data, well in advance about:

* A visual trigger as and when a moving vehicle approaches towards the security gate
* Driver’s image
* License plate recognition
* A composite area scan taken from a specialized high frame-rate camera scanning the images from under the vehicle

The security personnel then, can assess the situation based on the data provided by the CPAS system and react accordingly. The CPAS system as such does not provide any functionality to systematically manage the situation, like raising the barricades, alarms, etc

## **CommPort System –**

## **Overview**

The CommPort application will allow the user to view the list of driver details along with the following features such as

* List of Events
* Filtering events based on date / other parameters
* List / Filter Alerts.

### **High Level Design**



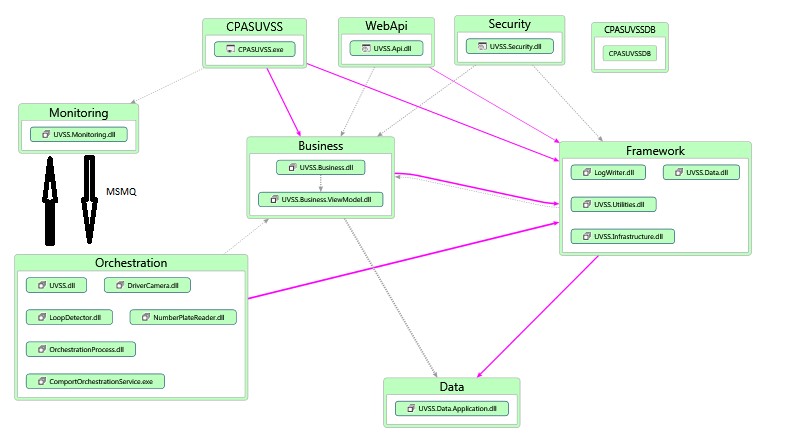
Fig.1 High level Design

Screen Entry Point – The user / guard logs into the system via the login screen and the main screen is displayed

* The UI Layer would be WPF UI which will interact with Rest API layer to display the data to end user.
* The Monitoring Layer would be a Task Parallel library component / background worker process for interacting with Queue (MSMQ) to fetch data. The Monitoring layer would push the data (such as Event ID, LPR data) to UI.
* MSMQ - This would be a used as a storage to get the information from the orchestration Module.
* Orchestration Module – This module will use Task Parallel Library / background worker process which enables the user to write Multithreaded, asynchronous and parallel code for implementation / execution
  1. This Module will do the following activities
     1. Collect and store images in Image Folder
     2. Store event in the database
     3. Push the Event to Queue for further processing.
* Core API – The Core API would be the module which will interact with Physical Hardware of Comm-Port which will connect to our application via IP address
  1. The API consists of services such as DAM, DI, LPR & UVI technology which will capture and store the images of the intruder / existing member along with the vehicle information.
  2. The API will only expose the events, and Orchestration Module will fetch the information for further processing in the Event Bus component.
* Rest API
  1. This API will interact with the Internal Services (which acts as a BAL and DAL Layer) and will provide all the information that are stored in the Database.
  2. The Services will contain the following modules
     1. Diagnostics Module
     2. Events Module
     3. Anamoly Detection
     4. Functionalities for Central Server
  3. BAL / DAL Layer
     1. The BAL layer would expose the Business data to API
     2. The DAL Layer would use the Entity Framework (Code first Approach) to interact with the Database.
* Image Server / Folder
  1. This server / folder would be used to store the images that are captured by the UVI, DI , LPR.
* Database
  1. To store all the user information along with image path in Server.
* Cross Cutting Layer
  1. This layer Consist of
     1. Authentication Service
        1. The Authentication service would be implemented using a token based mechanism so that the security aspect of the system can be maintained
     2. Exception Handling
        1. To Display error of the application, we would store the data in Database via Exception handling services
     3. Logging
        1. To store the sequence of Events that take place we will use Log4net library to be stored in a File in appending mechanism

### **Detailed - Low Level Design**

**Framework of UVSS System**



The following are the details of each layer in Application Framework

**UVSS.Data.Application Layer** – This layer would be a class library which is responsible for interacting with database. It consists of the data context classes and a factory pattern implementation to interact with the database. The layer contains the repository for each entity to map with the database, thus making a complete ORM (Object Relational Mapping) solution. The class library references the EntityFramework DLL to implement the dbcontext classes. The layer is implemented using Code First Entity Framework Model.

**UVSS.Framework –** This layer consists of three different sub projects

* **UVSS.Data –** Thisproject uses repositories to fetch the data from UVSS.Data.Application Layer. The interaction between the Business layer and data layer is done using this project via repository pattern.
* **UVSS.Infrastructure –** This project uses the EntityFrameworkMapperClass and Implements Dependency Injection technique using Registration by convention method by which each service and interface are mapped by name.
* **LogWriter –** This project is used to write Logs into the application and

**UVSS.Business –** This layer consists of two things one the Core Business Logic for the application to interact and another one would be the View Model to interact with the UI of the Application

**WebAPI** – This layer would be used to expose the data of Comm-Port to its partner for data integration.The Web Api layer will used UVSS.Business layer to fetch data from Database.

**CPASUVSS –** This would be the WPF application which would retrieve and store data of Driver, Lane and LPRevents into the Database.

**UVSS.Monitoring** – This layer is used to poll the Queue on regular Intervals to check for any message in the queue and send the same to the UI for further processing.

### **User Authentication Sequence Flow**

The Sequence Diagram will depict the flow of the user authentication in the system  


### **Sequence Diagram for Queue Mechanism**



### **Process Flow diagram**

* Process Flow of Orchestration Service to push data to Queue from LPR, DI & UVSS system.  
   

The Orchestration Module will do the parallel processing of all the three Hardware Components (LPR, DI, UVSS) to save the data in DB and images in a folder specified by the application in Configuration file. The module will use Plug and play functionality which will look up to the modules such as DI, UVSS and LPR for fetching the data.

Before calling the Loop detector the Orchestration Module will call **lock functionality** which checks whether the IP Address which are configured in the application are same as the Mac address binding of the Hardware Devices that are used in the system for capturing the images. If it is found to be tampered then an alert is displayed to user.

* Process Diagram for LPR Check Process



The above diagram depicts the process flow of the LPR Camera functionality into the system.

* Process Flow of reading the Data from MSMQ



Fig-1



Fig-2

Figure 1 depicts the connectivity between the UI and the Monitoring Tool.

The Monitoring tool in Figure 2 above could be an MSMQ Listener / Background worker process to read the data from Queue and in poll the MSMQ based on time intervals. IT will perform two actions

* + Either it sends the Data to UI
  + Or will send Error message to UI
* DI Camera Process flow



Process flow of UVSS Camera



# **Source code Management**

The source code would be stored in the TFS (team foundation server).

The source code is stored in the Chenoa TFS server and the IP is **192.168.133.9** and the URL to access the code base is

<http://192.168.133.9:8080/tfs/comport/Comport>

# **Logical Diagrams**

The following logical diagram will depict the interaction of the Orchestration Module for getting the data from DI, UVIS, LPR to MSMQ Queue.



Monitoring Module Logical Class Diagram



Monitoring Module logical diagram



# **Core API**

# **Web API**

The following are the methods provided in the classes of WebAPI

Class Name – **Search Events**

|  |  |  |
| --- | --- | --- |
| **Attributes** | LPREvent lprEvent | Attribute to get events based on LPREvent |
| **MethodName** | Public List<LPREvent> GetLPRData (LPREvent SearchData) | Method to get the list of LPRData by passing Object of LPREvent.  The output would be List of LPREvents |
|  | Public LPREvent DisplayLPRDetails(string LPRNumber) | Method to display details of LPRdata by passing LPRnumber.  The output would be an object of LPREvent |

ClassName – **UpdateEvent**

|  |  |  |
| --- | --- | --- |
| **MethodName** | Public void UpdateEvent (LPREvent event) | Method to update a flag based on event in the database. |

ClassName – **ManageLists**

|  |  |  |
| --- | --- | --- |
| **MethodName** | Public List<Events> ListVehicles () | Method to get the list of Vehicles.  The output would be List of Type of Vehicles which consist of  a. VIP / Wanted name b. |
|  | Public Event GetEventDetails(string EventId) | Method to display details of Vehicle by passing EventId.  The output would be an object of LPREvent |
|  | Public void AddList() | Method to add details of Name and description in the database. |
|  | Public void EditList() | Method to edit details of Name and description in the database. |
|  | Public bool DeleteList() | Method to delete details of record. |
|  | Public void AddEntry() | Method to add entry of LPR and name in the database. |
|  | Public void EditEntry() | Method to edit the data entered in the table. |
|  | Public void DeleteEntry() | Method to delete an entry of record |
|  | Public void ImportEntry() | Method to import the list of entries in table using .csv file |

ClassName - **Filter Events**

|  |  |  |
| --- | --- | --- |
| **MethodName** | Public List<Events> FilterEvents (string? Licenseplate, string? Drivername, string?vehicledetails, ) | Method to get the list of Vehicles based on the search criteria.  . |

ClassName - **Export Events**

|  |  |  |
| --- | --- | --- |
| **MethodName** | Public Object ExportEvent (list<Event> events) | Method to Export the selected list of data to create CSV file and image associated with event. |

ClassName - **Profile**

|  |  |  |
| --- | --- | --- |
| **MethodName** | Public Object CreateProfile(User user) | Method to Create Profile of user. |
|  | Public Object UpdateProfile(User user) | Method to update profile for a user. |
|  | Public List<User> ListProfile(User user) | Method to list users. |
|  | Public FilterProfile(User user) | Method to filter the profile based on user parameters |

ClassName – **ImageProcessor**

|  |  |  |
| --- | --- | --- |
| **MethodName** | Public static Object UVICompositeImage(string eventId, string ImagePath) | Method to get UVICompositeImage from the Image path based on eventId. |
|  | Public static Object LPRImage(string eventId, string ImagePath) | Method to get LPRImage from the Image path based on eventId. |
|  | Public static Object DIImage(string eventId, string ImagePath) | Method to get DIImage from the Image path based on eventId. |

# **Infrastructure Service**

The **UVSS.Infrastructure** project that is incorporated as part of Solution comprises of the following section

* Security – The security is based on credentials (ID & password) provided by the user. The password is stored in encrypted format in the database.
* Logging Mechanism – The Logging of application data would be provided by Logging of events based on the Hardware components would be logged in a text file using Log4Net tool in an appending manner. The Logs will be created on a daily basis and the Log file format would be <<dd.MM.yyyy.log>> and similarly the logs for other hardware components will be logged accordingly. Moreover, Application logs will also be stored in the File as Error Type based on the configuration set in the System.

# **Technology Stack**

The system consists of the following technologies

* WPF - for UI Application
* MSMQ - for storing and receive data from the Publisher Module of hardware components
* Entity Framework (ORM) - for storing and retrieving data from Database
* SQL Server – Database to store the records for persistence
* Unit Framework – Used to implement Loosely coupled architecture for maintainability, scalability
* Web API – to expose the Comm-Port data for Third Party Integration
* .Net Framework 4.6

# **Deployment Process**

## **System Requirement**

* Minimum 2GHz processor
* Minimum 16GB RAM
* Windows Server 2008 and above
* Microsoft .Net framework 4.5
* MSMQ

## **Environment**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Server IP** | **Application Installed** | **Comments** |
| UAT |  | EXE | Application not exposed to public |

## **Deployment Diagram**



## **Publishing Comm-Port Exe**

The deployment of Comm-Port Exe would be self-installable MSI Build which will help the user to install to the system.