CLup

# Customer Line-up

1. **INTRODUCTION**
   1. *PURPOSE*  
      The purpose of this document is to build a more concrete foundation of what the system-to-be will be. It will also define the general behaviour and specific limitations of the system. This document is primarily addressed to the programmers and mostly includes technical language.
   2. *SCOPE*

The scope of the design document is to define the system’s behaviour in general cases and some critical scenarios, and to design the architecture of the system-to-be so as to provide a time-efficient, logical allocation of the components and the interaction between these components.

The document is not only limited to the architecture and behaviour of the components, but it also extends in part to the implementation and testing plan, where one possible course of action is explained, user interface design of user applications and requirements traceability relating to the Requirements and Specifications Document (RASD).

* 1. *DEFINITIONS, ACRONYMS, ABBREVIATIONS*
  2. *REVISION HISTORY*
  3. *REFERENCE DOCUMENTS*
  4. *DOCUMENT STRUCTURE*

1. **ARCHITECTURAL DESIGN**
   1. *Overview: high-level components and their interaction*

The architecture of the application is structured according to three logic layers:

* *Presentation Layer (P)* manages the presentation logic, namely the interaction with the user. It comprises a GUI (Graphic User Interface) that makes the application’s functionalities more understandable to the user.
* *Business Logic* or *Application Layer* (*A*) handles all the functions to provide to the user and manages the exchange of information between the user interface and the data source.
* *Data Access Layer* (*D*) provides access to the stored data. The implementation of the access logic should be both easy and structurally robust to guarantee a correct abstraction from the specific database and provide a model easy to use.

In order to guarantee as much flexibility and scalability as possible, the system is based on a 4-tier architecture (Client, Web Server, Application Server, Database Server) with a thin client. Since the application should be easy to use and executable in several different devices, the use of a thin client prevents a heavy computation load client side, carrying out all the heavy operation at server side. The user can directly communicate with the application server through the installed app in his device, while the store manager can access the functionalities devoted to him through a web application communicating with the web server.

* 1. *Component view*

The image above shows the internal architecture of the Application Server and all the interfaces that are provided to the outside world, and all interfaces required.

Starting from what are conceptually two mediators, the system will contain two components called:

* *Map Mediator Module*   
  This component’s job is to communicate with the external API, by modifying the API’s information so that it can be comprehensible by the Application server, and adapting the requests to the API’s protocol, required by other modules of the server. It provides information regarding the local map with all the relevant stores in the map as well as an estimation of the time it takes to reach one store from a certain location. [SHOULDN’T ETA BE IN LOCATION MODULE]
* *Turnstile manager*  
  The turnstile manager, as opposed to the previous component, provides an interface for the turnstiles so that they can be opened or closed through the scans of the right QR codes.

For the authentication of an account, be it user or store manager, the following component will be of help:

* *Account manager*  
  This is the component which is responsible for the authentication of any user that wants to log in or ***register*** (???), so it saves all the relevant information about that account and furthermore, provides to all users the possibility to fetch all upcoming tickets that they have an appointment for.

Once a user is logged in the system, it has to provide functionalities to the user that allow them to book visits, get tickets, be reminded of upcoming visits and be suggested different stores in case the requested ones are full. The following components allow users to:

* *Reservation manager*  
  It provides the interfaces that allow booking of tickets and ticket requests to be carried out. It communicates with different components [ADD INTERFACE TO LOCATION MODULE] to check all the relevant, free timeslots, and show a map of all the available shops around the user’s current location. It also provides internal interfaces that will aid the functionalities of other components. [Why is ticket fetcher in user mng and not in reservation mng?]
* *Notification manager*  
  This component has the job of reminding users of the time when it is their time to leave their current location to reach in time their destination, provide periodical information of available stores when the requested one is full and suggestions of available stores in specific requested times.

Instead, to provide functionalities related to the store managers the following component provides:

* *Market Manager*  
  Issues, through its interfaces, the functionalities of controlling how many customers can enter in the store, how many people are inside of said store and the statistics regarding all entrances within a certain period.

The components which connect everything together and provide the logic of the dispensing of tickets are the following:

* *Queue Manager*  
  Which generates all the information regarding all booked and available tickets, such as the free time slots, the validation of tickets in entrances and exits inside stores. It also forwards this information to the database to be stored for a longer interval of time.
* *Location Module*  
  It provides all the interfaces that allow the search of the closest stores around a location. [SHOULDN’T THIS HAVE NO SUBCOMPONENTS? See comment above too]
  1. *Deployment view*
  2. *Runtime view*: You can use sequence diagrams to describe the way components interact to accomplish specific tasks typically related to your use cases
  3. *Component interfaces*
  4. *Selected architectural styles and patterns*: Please explain which styles/patterns you used, why, and how
  5. *Other design decisions*

1. USER INTERFACE DESIGN: Provide an overview on how the user interface(s) of your system will look like; if you have included this part in the RASD, you can simply refer to what you have already done, possibly, providing here some extensions if applicable
2. REQUIREMENTS TRACEABILITY: Explain how the requirements you have defined in the RASD map to the design elements that you have defined in this document.
3. IMPLEMENTATION, INTEGRATION AND TEST PLAN: Identify here the order in which you plan to implement the subcomponents of your system and the order in which you plan to integrate such subcomponents and test the integration.
4. EFFORT SPENT: In this section you will include information about the number of hours each group member has worked for this document.
5. REFERENCES