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*
* Interactive mobile app mockup: *<https://customerlineup.bubbleapps.io/>*
* Interactive web app mockup: *<https://customerlineupmanager.bubbleapps.io/> (login with user: [admin@mail.it](mailto:admin@mail.it) password: admin)*
  1. *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. As represented in figure 2.1 the user, through a smartphone or a tablet, and the physical dispenser can directly communicate with the application server, while the store manager can access the functionalities devoted to him through a web application communicating with the web server. The use of a web server for the store manager’s functionalities is based on the fact that, in general, the web apps are quicker and easier to build, maintain and update and less expensive than the mobile apps, even if slower. The application server communicates with the data server to store the needed information. A more detailed description of the architectural design is given in the next section.



Figure 2.1 – high level architecture

* 1. *Component view*

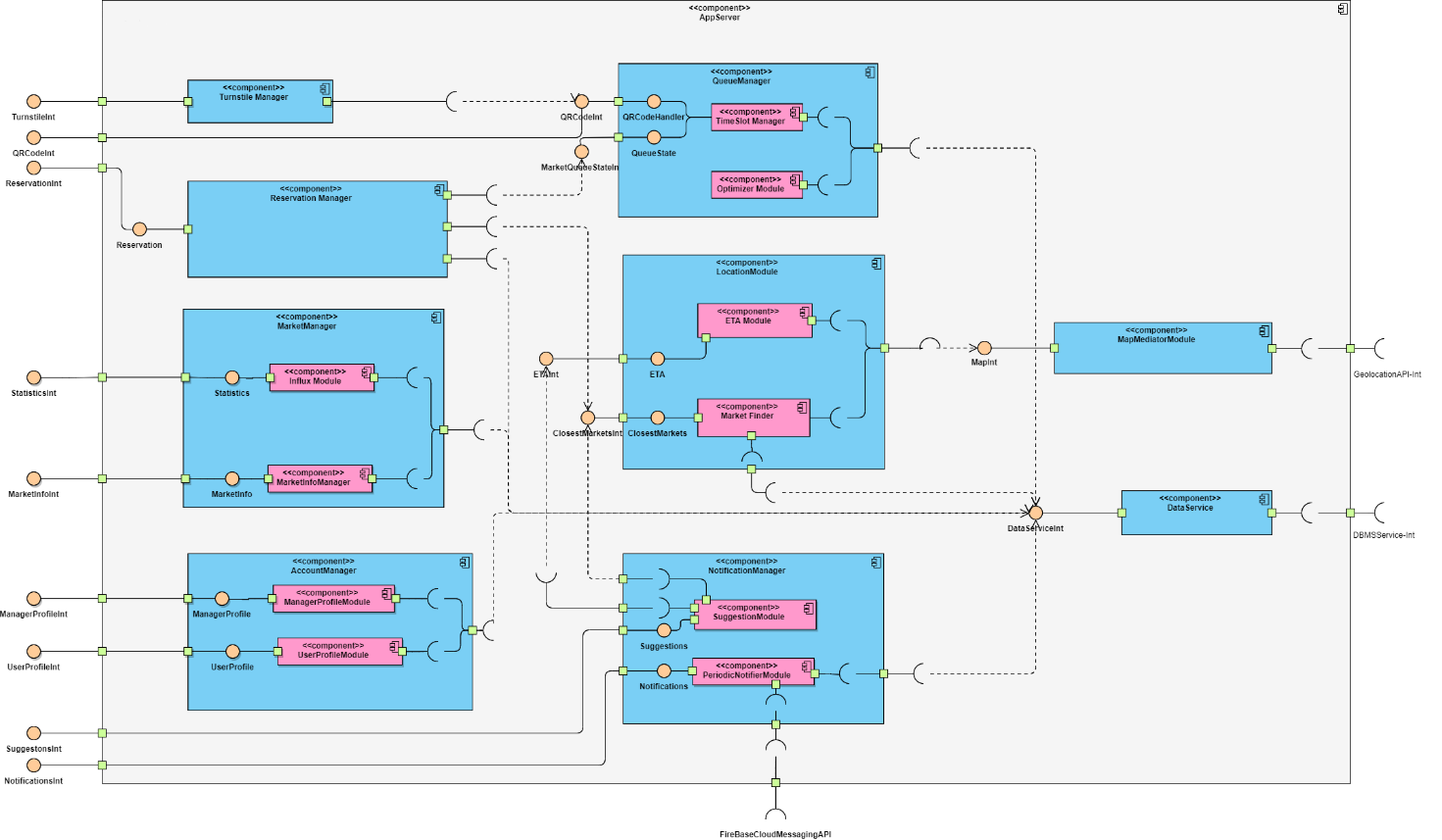
1. *General component view*

*Immagine che contiene testo, screenshot

Descrizione generata automaticamente*

In the above image represents the general component representation of the system. Here are shown the interfaces with the application server needed by the other components of the system: turnstiles have an interface to handle QR codes checking; the physical dispenser interacts with the application server as it was a normal device getting a ticket, but with “restricted” degrees of freedom (e.g. it cannot choose store, but it is linked to the one it is in front of, it cannot specify shopping list…); Web Server will interface with application server to get all information on statistics it needs to allow the correct usage of the web app; Mobile App, having a thin client, will interface with the system to do basically everything it needs: account and ticket managing, subscribing to notifications, getting new tickets and visits. It is also shown that the application server communicates with external DBMS to maintain permanent data and with a geolocation service, that must provide services allowing retrieving distances between locations (both in terms of space and time) and retrieving info on nearby places given a type of place (in our case, shops). Google Maps is a good candidate, as it provides APIs as “Places API” that allow lookups for specific types of places, with useful parameters as the “radius”. Firebase Cloud Messaging is used to provide push notification services (see details in section H-additional specifications).

1. *Application Server 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.

* ***Turnstile Manager***

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 Store Managers and of any user that wants to log in or register, allowing also account modifications and general account managing. Furthermore, it provides to all users the possibility to fetch all of theirs tickets.

Once a user is logged in the system, the application server must provide functionalities to the user that allow him 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***
* ***Notification Manager***

This component has the job of handling notifications of two types:

- reminding users when it is time to leave their current location so to reach in time their destination; to do this, the component interfaces with the map provider, getting info on ETA. Push notifications are handled through Firebase Cloud Messaging.

- provide suggestions of relevant available stores and timeslots when the requested one is full or the user is inactive.

Instead, functionalities related to the store managers are provided by the following component:

* ***Market Manager***

Issues, through its interfaces, the functionalities of controlling how many customers can enter in the store and how many people are inside of said store. It also provides the statistics as average entrances per time range, average duration of shopping, most selected items in shopping lists (for visit only).

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

* ***Queue Manager***

This is the most critical and important component of the system, as it defines and handles timeslots and the queue. Through the timeslot manager subcomponent, timeslots are searched to see or update their availability on new tickets requests. It also generates QR codes for new tickets and visits, assigning them to available timeslots, and checks QR codes on scan, so to correctly update the state of the queue in that timeslot.  
The other component is a timeslot optimizer, that is a periodic activated component, which handles aspects of the optimization of timeslots, as the maximum number of people that can enter considering info about user visits that declared their shopping list or departments they will visit; another possible optimization is the duration of timeslots, performed once in a greater period of time with respect to the previous one, that could optimize the duration of timeslots based on the average shopping duration. To have a clearer understanding of how the timeslot system works, see the referenced document.

* ***Location Module***
* ***Data Service***

It provides access to the external interface of the database, doing queries and generally interacting with the database for needed operations.

* 1. *Deployment view*  
       
     

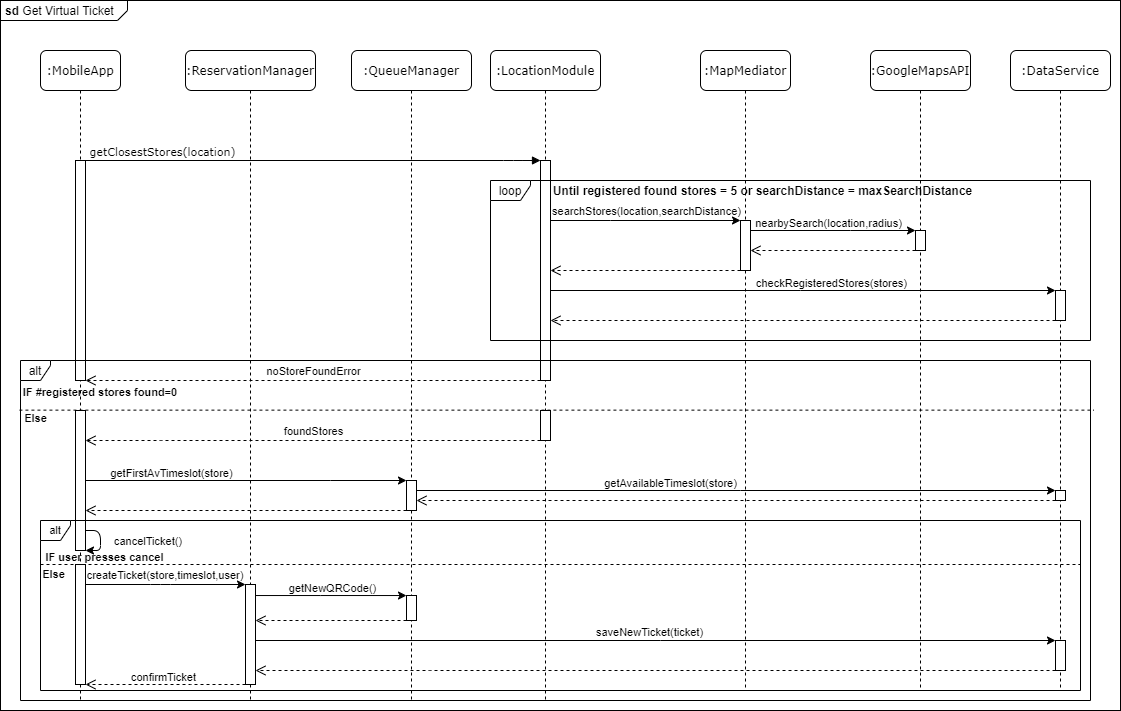
Some important aspects to highlight are the following:

* With Portable SmartDevice we mean any mobile device such as tablets and smartphones. With OS we intend either IOS or Android.   
  In computer, any OS and Web browser can communicate with the web server as long as the HTTP protocol is being used.
* The CLup mobile app will store information in the client side such as upcoming tickets data, for faster data retrieval and user information for identification. It furthermore uses a specific communication protocol that the programmers might see fit.
* The installation of the CLup mobile app is done through the mobile phone’s application store and is not been shown in the deployment view.
* The application server is connected to the database through an internal LAN with no firewalls, for increased connection speed.
* Turnstiles and cash registers need to forward information regarding QR codes through the internet.
  1. *Runtime view:*

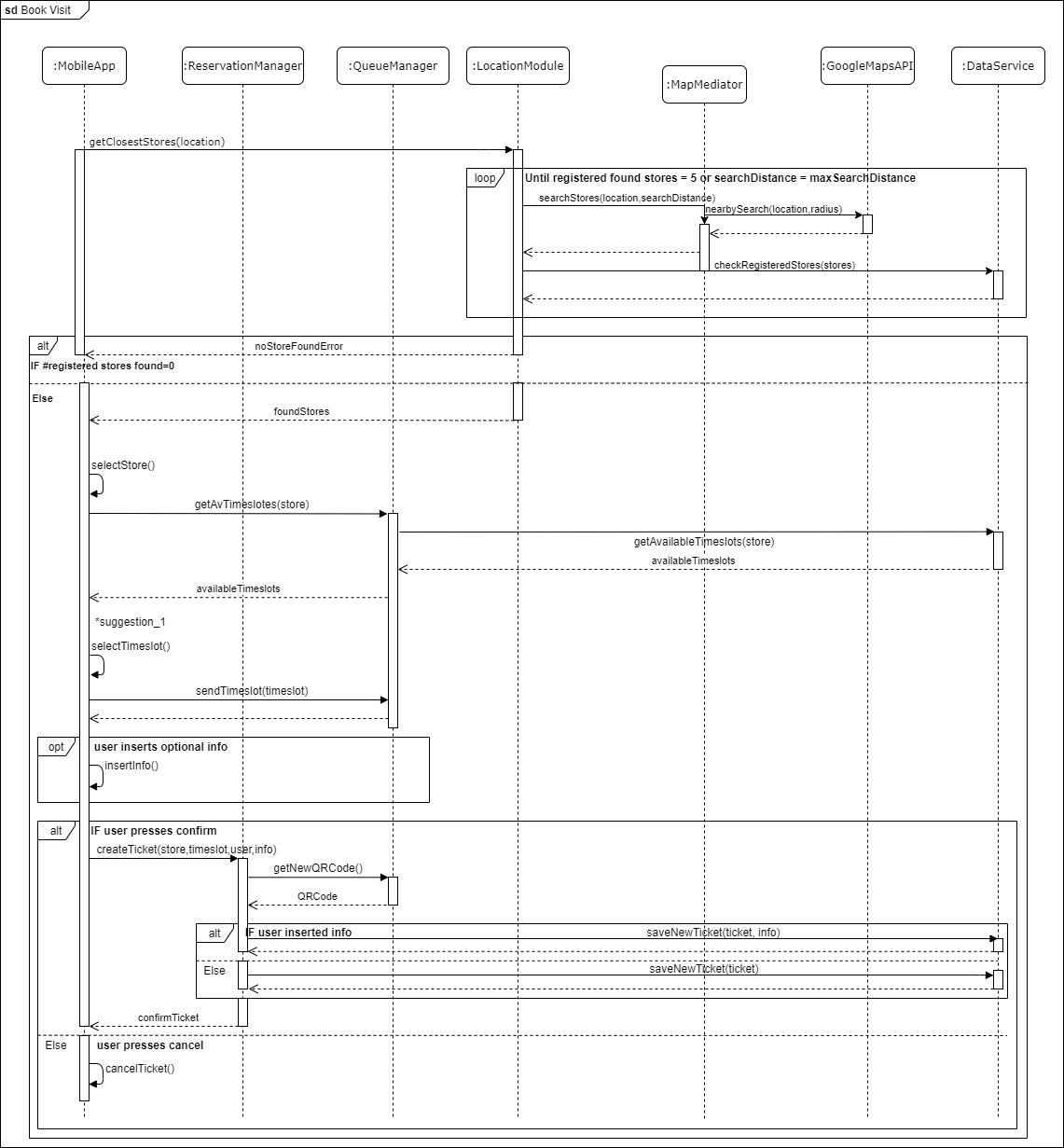
1. *Immagine che contiene tavolo

   Descrizione generata automaticamente User Registration*
2. *User Login*

The user, whose representation has been considered superfluous, attempts to login in the application through the Mobile App and his access is granted only after a credential check through the Data Service component. If the credential check is not passed, then the application notifies the user, who can retry the procedure, that an error occurred due to either a username or password error.

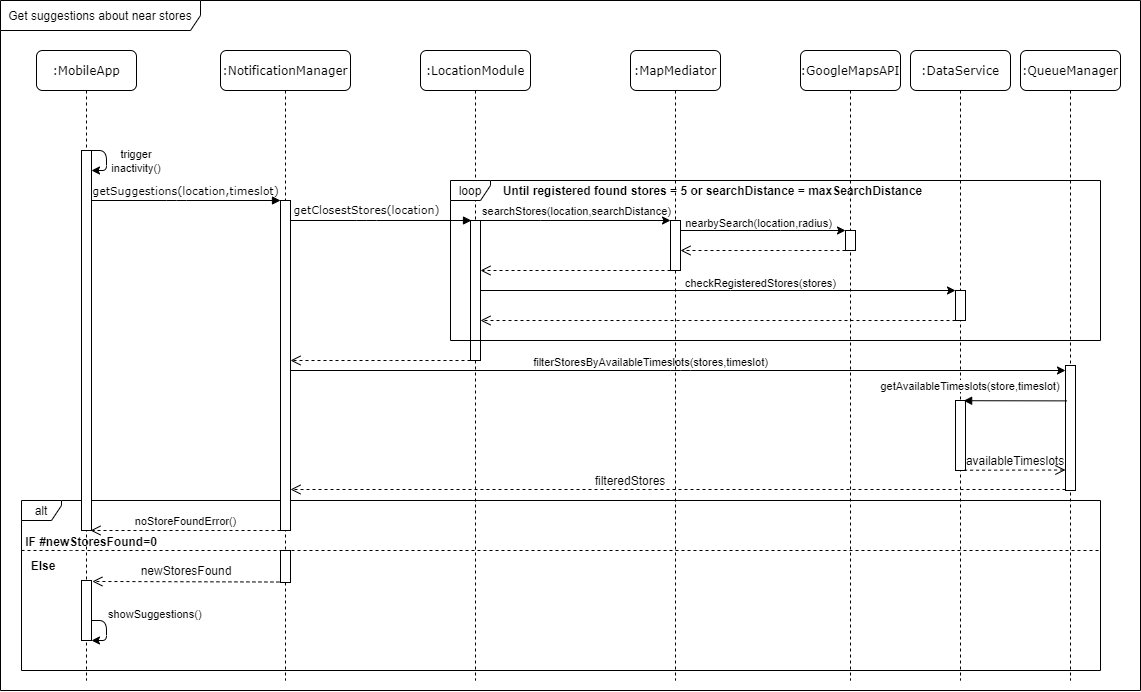
1. * Get Virtual Ticket*

This sequence diagram represents the interaction between components to search for an available shop to get a virtual ticket. As we can see, the system handles cases where the user has no close markets by trying to increase the radius if no close markets or too few have been found, eventually stopping when a max distance of search has been reached. This allows to give some degree of choice to all users.

1. * Book visit*

This sequence diagram represents the procedure to book a visit. The differences respect to the “Get Virtual Ticket” sequence diagram are:

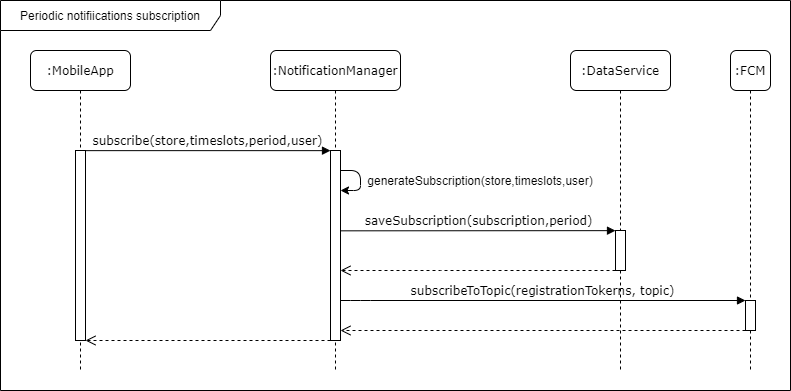
* The user can either selects the desired timeslot or store as first option, while in getting a virtual ticket he can only selects a store, accepting or rejecting the first available timeslot of the chosen store. Note that in this case the user selects a store as first option and then a timeslot between those ones provided by the application. The case in which the user selects a timeslot as first option is symmetrical to this sequence diagram respect to the getStores/selectStore and getAvailableTimeslots/selectTimeslot parts. Moreover, in the second case, \*suggestion\_1 would be substituted \*by suggestion\_2,
* The user can, optionally, fill a form where he inserts the estimated duration of the shop trip, and either the specific items or the categories of goods he intends to purchase. Other than the new ticket, the additional information, if provided by the user, will be stored in the database through the Data Service Component.
* The \*suggestion\_1 and \*suggestion\_2 represent the cases in which the user is inactive for 15 seconds (INSERT THE SPECIFIC TIMING???) while selecting, respectively, a store and a timeslot. These two sequence diagrams are represented below.

1. *Get suggestions about other available near stores (suggestion\_1)*

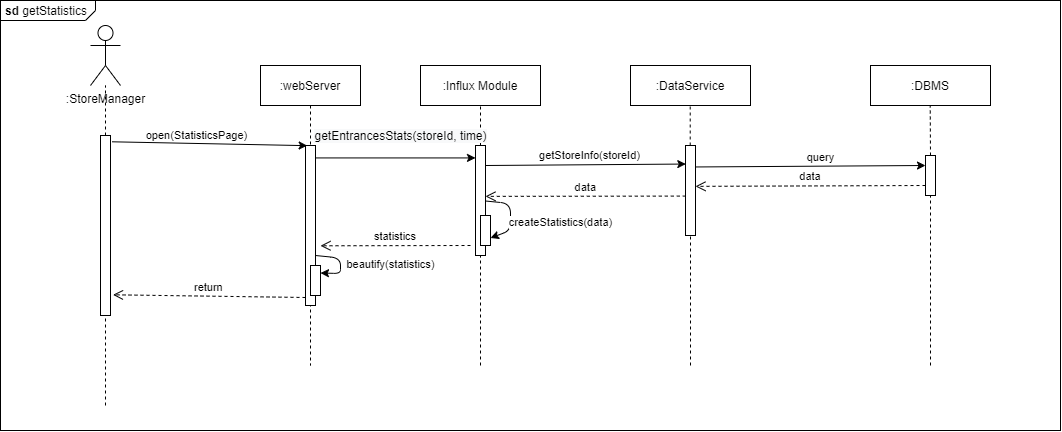
This diagram shows the interactions providing the first type of suggestion given to inactive users during a ticket/visit booking. As shown, the system searches for further available stores near the selected one. After the suggestion is shown to user, if he selects on it, he will start browsing through the suggested stores, and the diagram is the same of the ticket standard booking from the point of selecting a store.

The “Get suggestions about available timeslots (suggestion\_2)” sequence diagram is not represented to avoid redundancy. Indeed, the difference between these two are:

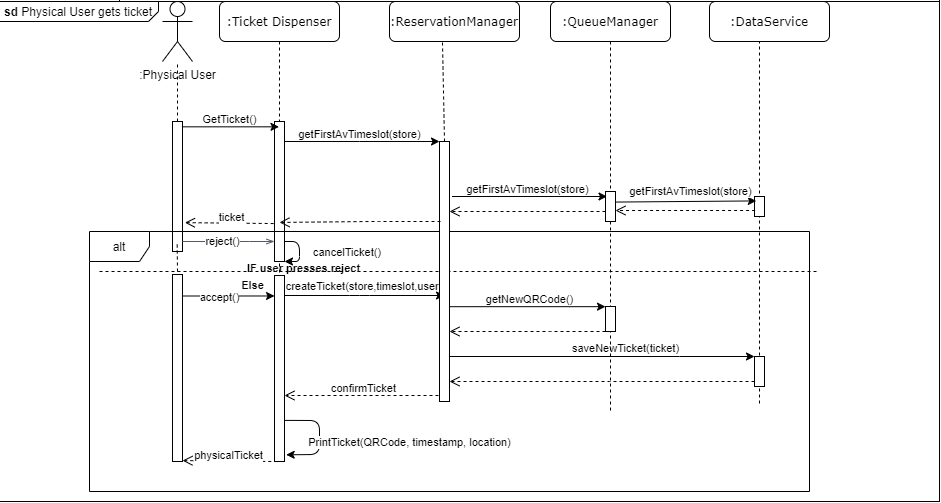
* After the user being inactive for 15 seconds, Suggestion\_2 is triggered while the user is selecting a timeslot (Suggestion\_1 is triggered while the user is selecting a store).
* Suggestion\_2, unlike Suggestion\_1, provides further stores near the user’s location filtering them through their first available timeslot. Since the user has already selected a timeslot, it is likely that the user would want to select a store with an available timeslot close to the selected one.

1. *Periodic notifications subscription*

The user subscription request may have different granularity, as to be notified about available timeslot for near stores, or for a specific store, or even for a specific timeslot available in a specific store. Once received the request is processed, information about the subscriptions are saved, then the user is registered to the topic through Firebase Cloud Messaging.

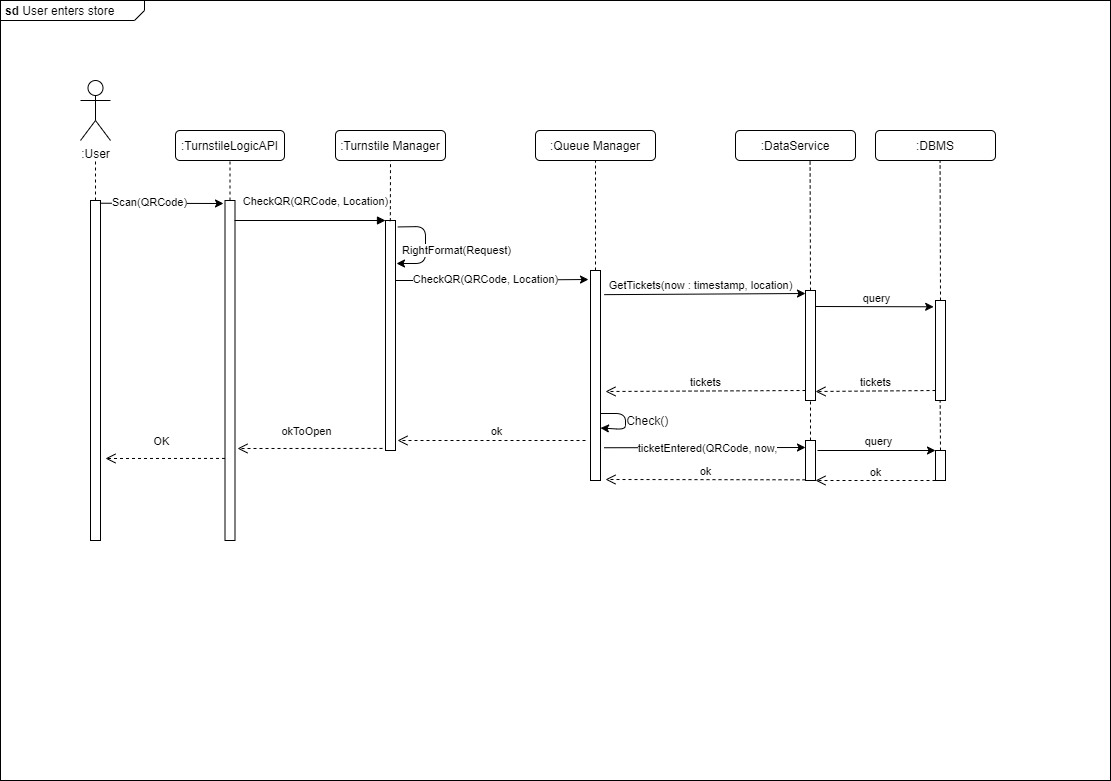
1. * Store Manager statistics*

Once the store manager, through his browser opens the statistics page, the web server requests all the data regarding the store’s entrances. Instead of requesting all statistics to be created on the DBMS, the Data Service component retrieves only the data regarding that store and the influx module creates all the statistics internally. This so we can ease the computational workload of the DBMS and increase it for the Influx Module, a component which is going to be used way less in comparison.

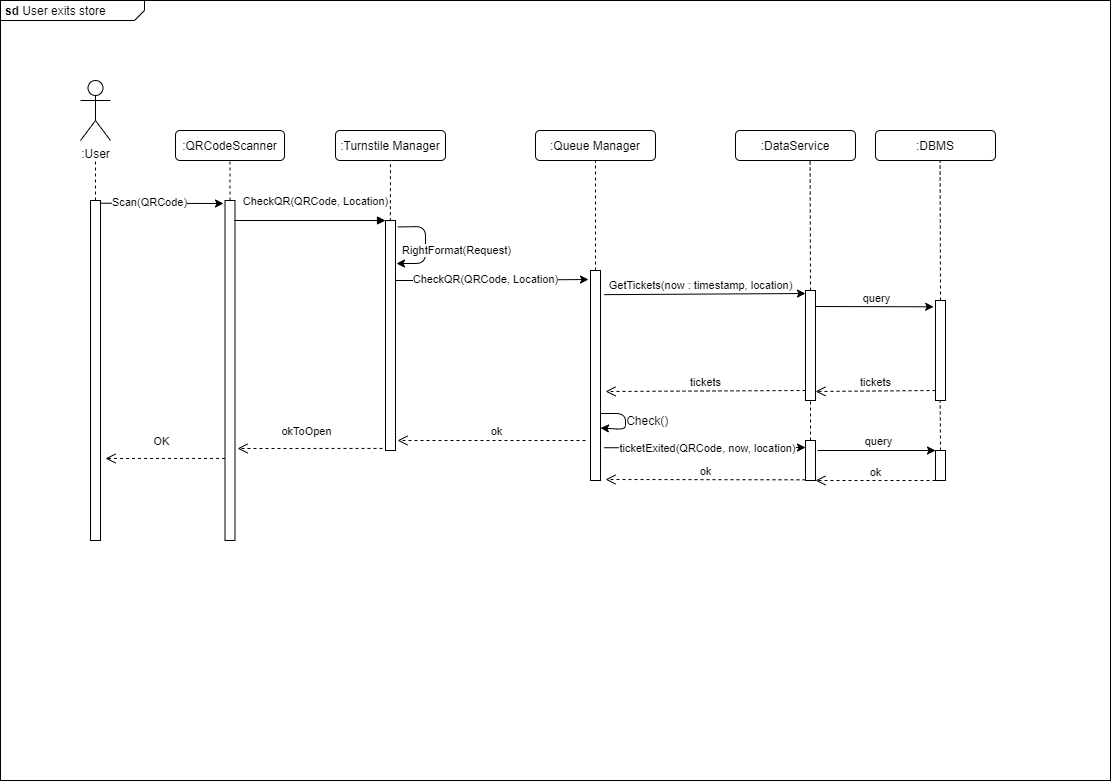
1. * Physical ticket acquisition*

The user who goes physically to the store, requests a ticket and the ticket’s dispenser internal logic communicates with the system, asking the first available time slot of that one store. Just like the virtual ticket, the execution of the methods is the same.

1. *User enters store*

**

A user that initially scans the code activates the turnstile’s internal logic which will invoke the method that checks if the scanned QR code is correct with respect to the location of the turnstile. The mediator turnstile manager will translate the request so that it is compatible with the internal server protocol. After the check has been completed, the queue manager will save the data regarding the ticket that has just been scanned as a valid entrance, without waiting for a confirmation from the turnstile logic API that the user has actually pushed the turning mechanism. Instead in the case the scanned QR is early, late or in the wrong store, there is no data to be saved and our system will send a NOK response that will not allow to unlock the turnstile.

1. *User exits store*

The only difference from the “User enters store” sequence diagram is that in this case, there might be a QR code scanner inside the cash register that communicates with the system’s mediator. The scan can be done by the cashier or by the system’s user through a default turnstile where all users who have not bought any products go to.

* 1. *Component interfaces*
  2. *Selected architectural styles and patterns*:

**Notification Push Pattern**

To provide the required periodic notifications functionalities, a push pattern has been chosen, so to have more efficiency and scalability. This decision is mainly influenced by the expected high number of users using the system: with a pull approach, the server would risk getting flooded by periodic notifications requests. The general approach with this pattern has 3 layers:

- **Mobile notification services** (MNS) are components that allow third-party app developers to send information from their servers to the user’s devices  
- **Application Server** (push Server) is the server creating push requests handled by MNS, targeting directly specific users, or group of users, based e.g., on a topic they have subscribed to.

- **Client Application** registers to the notification service, getting uniquely identified. The client will then keep a background connection, nor data or power expensive, that will be used to receive notifications by MNS.

Upon server push requests, the MNS uses the connection with the user to send the notifications.

1. device registers to MNS
2. Upon successful registration, MNS server issues registration token to client device.
3. Client device sends registration token to app server
4. App server stores client registration token for future notification uses.
5. Server sends push notification request to MNS identifying the device(s).
6. MNS Server sends notification to device(s).

The decision is to use Google’s Firebase Cloud Messaging(FCM), that provides a cross platform (Android, IOS) full handling of push notifications(sending, routing, queuing). It is also implementing subscriptions to specific topics, with no limits on the number of topics nor subscriptions to a topic (important given our system expected clients and variety of topics, that can be each timeslot or store or a combination of both). To use FCM, it must be set up both server side and client side.

TO DO: Mediator with QR code scanner of cashier.

Mediator with Physical dispenser (outside shops). Or instead we build them for shops

* 1. *Other design decisions*
  2. *Additional Specifications*

1. *USER INTERFACE DESIGN*:

Here we include some draft mockups about how the mobile applications and the web application should look like. As they are drafts, these are not to be intended as strictly constraining designs: backgrounds, styles, text formatting and page structure are simplified and rough, as we want just to give an idea of the general schema of the pages.

* 1. *Mobile App Mockups*

Map

Description automatically generatedGraphical user interface, application

Description automatically generatedAs already mentioned in RASD, the mobile app must have simple and intuitive interfaces, as to be easily usable by users that may not be practical with technology (as may be elderly people).

Shop selection

Home page of the app

The app will allow user to define first either the shop or the date and time, in the case of “Book a visit”, it is not shown in this demo because it is similar and irrelevant.

\*Mockups are taken from an interactive graphic demo with graphic-driving purposes only, linked in the “reference documents” paragraph.

Graphical user interface, text, application, chat or text message

Description automatically generatedQr code

Description automatically generatedDiagram

Description automatically generatedIn the case of “book a visit”, after the shop selection, the app will allow registered users to select their shopping list:

book a visit as registered user

book a visit as Guest

Ticket Confirmation

Ticket proposal

* 1. *Shop Manager Web App mockups*

The web app for managers will allow only to login with an authorized account. For security purposes, such account cannot be registered directly through the Web App, but it has to be set up by CLup System staff through certified communication with the market/market chain.  
The web app must allow managers to trace entrances, to see the queue state of any timeslot, and to see statistics as average time of shopping and the average influx in a given period.

In the real app other statistics may be implemented, as the most declared shopping items or departments (given by user that specified these elements in their visits). Graphics in the real app may be interactive, allowing more interactivity, usability and utility.

Graphical user interface, chart, application, line chart

Description automatically generated

Web App homepage

The “Manage Shop” page is not shown here, it simply allows managers to add or update logistics info for the shop, as product positions or departments. It just lists departments and their items.

Chart, line chart

Description automatically generated

Web App Statistics page

A picture containing chart

Description automatically generatedThe page shown upon clicking the “See” button next to the declared shopping list is not shown here, since it is unnecessary as it simply shows the list of declared items.   
As shown in the image, physical users have no Id, but they still are tracked and considered correctly in the queue state.

Web App Queue State

1. *REQUIREMENTS TRACEABILITY*

|  |  |
| --- | --- |
| **R1** | The system shall allow users to get a ticket with a date and time that shows when to go to a certain store, virtually |
| **R3** | The system shall allow users to book a visit virtually with their desired store. |
|  | Components:   * Reservation Manager * Queue Manager * Location Module * Map Mediator Module |
| **R2** | The system shall allow users to get a ticket with a date and time that shows when to go to a certain store, physically |
|  | Components:   * Reservation Manager * Queue Manager |
| **R4** | The system shall allow users to look up on a map available registered stores where to go to |
|  | Components:   * Map Mediator * Location Module |
| **R5** | The system shall ask users how much he or she thinks the trip to the store will last |
|  | Components:   * Reservation Manager |
| **R6** | The system shall allow users to be identified by their device unique ID |
| **R7** | The system shall allow users to be identified by a username of their choosing |
|  | Components:   * Account Manager |
| **R8.1** | The system shall notify the user who is inactive while on the confirmation page of booking a visit on other available stores he could go to and other timeslots of the same store |
| **R8.2** | The system shall notify the user who is inactive while on the confirmation page of getting a ticket on other available stores he could go to |
|  | Components:   * Notification Manager: Suggestion Module * Location Module: Market Finder * Map Mediator * Queue Manager: Timeslot Manager |
| **R9** | The system shall allow its users to insert information about which categories or items they want to buy |
|  | Components:   * Reservation Manager |
| **R10** | The system shall infer how long it will take for “Book a Visit” customers to buy the expressed shopping list |
|  | Components:   * Queue Manager: Optimizer Module |
| **R11** | The system shall store data about registered virtual users' visits durations and expressed shopping lists |
|  | Components:   * Reservation Manager * Turnstile Manager * Queue Manager: Timeslot Manager |
| **R12.1** | The system shall inform users periodically of the available time slots in the store for which he subscribed to the service |
| **R12.2** | The system shall allow its users to select which store(s) to get informed about |
| **R12.3** | The system shall allow its users to select which time slots he is interested to get informed about |
| **R12.4** | The system shall allow its users to select how often to get notified |
|  | Components:   * Notification Manager: Periodic Notifier Module |
| **R13** | The system shall allow the user to scan its QR code in entrance through the turnstiles |
| **R14** | The system shall allow the user to scan its QR code in exit through the turnstiles or cash register |
| **R15.1** | The system shall unlock turnstiles after a unique QR code scan in entrance |
| **R15.2** | The system shall unlock turnstiles after a unique QR code scan in exit |
| **R16** | The system shall map the QR code of a scan to the virtual user who owns the QR code |
|  | Components:   * Turnstile Manager * Queue Manager: Timeslot Manager |
| **R15.3** | The system shall lock the turnstiles after a push has occurred |
|  | Components:   * Turnstile Manager |
| **R17** | The system shall send a reminder to the user when it is time for him to leave so that he can arrive at the store in time |
|  | Components:   * Notification Manager: Suggestion Module |
| **R18** | The system shall calculate the time it takes the user to go to a shop in which he has an appointment |
|  | Components:   * Notification Manager. Suggestion Module * Location Module: ETA Module * Map Mediator Module |
| **R19** | The system shall manage how many people can enter inside the store for each timeslot, as to not exceed the maximum number of people allowed inside the store |
| **R20** | The system shall calculate the maximum number of people allowed inside each store, as to allow for social distancing to take place |
|  | Components:   * Queue Manager: Optimizer Module |
| **R21** | The system shall count the number of entrances and exits each day for each market |
| **R22** | The system shall store the number of daily entrances and exits for each market |
|  | Components:   * Queue Manager: Timeslot Manager |
| **R23** | The system shall manage how many people can enter inside the store for each timeslot, as to not exceed the maximum number of people allowed inside the store hourly, without exceeding the maximum calculated by the system |
| **R24** | The system shall allow store managers to see how many customers have entered the store in any day. |
|  | Components:   * Market Manager: Influx Module * Web Server |
| **R25** | The system shall allow store managers to register their stores |
| **R26** | The system shall allow store managers to input the store’s location |
| **R27** | The system shall allow store managers to input what categories of goods and what products are contained in the store |
| **R28** | The system shall allow store managers to input the dimensions of the store |
|  | Components:   * Market Manager: Market Info Manager * Web Server |

Note that, as shown in the component and in the sequence diagrams, each component interacts with the database through Data Service component, that was omitted here to avoid repeating it in every requirement.

*Traceability table:*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | MM | WS | *MMM* | *TM* | *AM* | *RM* | *NM* | *QM* | *LM* |
| R1 |  |  | **×** |  |  | **×** |  | **×** | **×** |
| R2 |  |  |  |  |  | **×** |  | **×** |  |
| R3 |  |  | **×** |  |  | **×** |  | **×** | **×** |
| R4 |  |  | **×** |  |  |  |  |  | **×** |
| R5 |  |  |  |  |  | **×** |  |  |  |
| R6 |  |  |  |  | **×** |  |  |  |  |
| R7 |  |  |  |  | **×** |  |  |  |  |
| R8.1-2 |  |  | **×** |  |  |  | **×** | **×** | **×** |
| R9 |  |  |  |  |  | **×** |  |  |  |
| R10 |  |  |  |  |  |  |  | **×** |  |
| R11 |  |  |  | **×** |  | **×** |  | **×** |  |
| R12.1-4 |  |  |  |  |  |  | **×** |  |  |
| R13 |  |  |  | **×** |  |  |  | **×** |  |
| R14 |  |  |  | **×** |  |  |  | **×** |  |
| R15.1-2 |  |  |  | **×** |  |  |  | **×** |  |
| R15.3 |  |  |  | **×** |  |  |  |  |  |
| R16 |  |  |  | **×** |  |  |  | **×** |  |
| R17 |  |  |  |  |  |  | **×** |  |  |
| R18 |  |  | **×** |  |  |  | **×** |  | **×** |
| R19 |  |  |  |  |  |  |  | **×** |  |
| R20 |  |  |  |  |  |  |  | **×** |  |
| R21 |  |  |  |  |  |  |  | **×** |  |
| R22 |  |  |  |  |  |  |  | **×** |  |
| R23 | **×** | **×** |  |  |  |  |  |  |  |
| R24 | **×** | **×** |  |  |  |  |  |  |  |
| R25 | **×** | **×** |  |  |  |  |  |  |  |
| R26 | **×** | **×** |  |  |  |  |  |  |  |
| R27 | **×** | **×** |  |  |  |  |  |  |  |
| R28 | **×** | **×** |  |  |  |  |  |  |  |

*Legend of components:*

* *Map Mediator Module: MMM*
* *Turnstile Manager: TM*
* *Account Manager: AM*
* *Reservation Manager: RM*
* *Notification Manager: NM*
* *Market Manager: MM*
* *Queue   
  Manager: QM*
* *Location   
  Module: LM*
* *Data Service: DS*
* *Web Server: WS*

1. *IMPLEMENTATION, INTEGRATION AND TEST PLAN*
2. *Implementation Plan*

The implementation of the CLup system will be done component by component. The order in which the implementation will be carried out depends on a number of factors such as the complexity of the component (e.g., number of provided functionalities) and the dependence on the other components being implemented. In parallel, unit tests should be carried out in order to discover flaws early, make easier the changes and guide the design. The implementation’s order of the components is the following:

1. Data Service, Map Mediator Module
2. Location Module, Queue Manager, Account Manager, Market Manager
3. Reservation Manager, Notification Manager, Turnstile Manager

Note that components belonging to the same group are independent, so they can be implemented in parallel. Moreover, the DBMS service, the Geolocalization API and the FirebaseCloudMessaging API are assumed to work properly since are external services.

The **Data Service** and the **Map Mediator Module** components should be implemented as first because all the remaining components of the application server depend directly or indirectly on the former since it handles the communication with the database to store and retrieve data, while the latter is independent from all the others because it just communicates with the external geolocalization 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.

**Queue Manager**, **Account Manager** and **Market Manager** are grouped together becausethey directly exploit the Data Service module to retrieve information, store data or carry out consistency and security checks. Moreover, **Location Module** belongs also to the second group because it depends on Data Service, to retrieve information about the registered stores, and Location Module to find the closest stores in a range respect a location and compute the ETA.

Eventually, in the third group belongs the **Turnstile Manager**, communicating with the Queue Manager to check the validity of a QR code; the **Notification Manager**, depending on Queue Manager and Location Module to retrieve information about time slots and stores; the **Reservation Manager**, that communicates with the Queue manager, Location Module and Data Server to handle the creation of a ticket.

1. *Integration Strategy*

Considering both the overall system’s architecture and the implementation plan, the chosen integration strategy is the **bottom-up approach.** System integration begins with the integration of the lowest level modules and uses test drivers to drive and pass appropriate data to the lower-level modules. As and when the code for the other module gets ready, these drivers are replaced with the actual module.

This approach allows to start the integration and testing without necessarily waiting for the completion of the development and the unit testing of each system’s component. Being the low-level modules and their combined functions often invoked by other modules, it is more useful to test them first so that meaningful effective integration of other modules can be done. Moreover, starting at the bottom of the hierarchy means that the critical modules are built and tested first and therefore any errors in these modules are identified early in the process.

Each integration in the same level (defined by the groups of the previous section) is independent and there is no specific order in which to complete them. In this way, the integration process and its testing are more flexible.

1. *Integration and testing*

In this section it is defined the order of the integration between components. Test drivers will be used to simulate higher components not yet implemented. As already stated, since using the bottom-up approach we should begin to integrate the lowest module, the first components chosen for integration are the Data Service (figure 5.1) and the Map Mediator Module (figure 5.2).



Figure 5.1

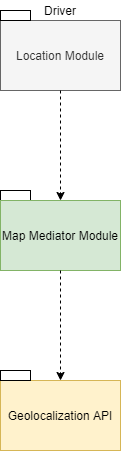


Figure 5.2

After that Data Service and Map Mediator Module have been implemented and tested, the integration of the components depending on them can start, namely, the Account Manager (figure 5.3), the Market Manager (figure 5.4)and the Queue Manager (figure 5.5), relying only on the Data Service, and the Location Module (figure 5.6), relying on the Data Service and the Map Mediator Module. Note that Queue Manager and Location Module should be preferably implemented, integrated and tested before Account Manager and Market Manager because they are more critical and so more likely subject to flaws.



Figure 5.3



Figure 5.4



Figure 5.5



Figure 5.6

Eventually, the integration plan involves the Reservation Manager (figure 5.7), depending on the Queue Manager, Location Module and Data Service; the Notification Manager (figure 5.8), depending on the Location Module and Data Service; the Turnstile Manager (figure 5.9), depending on the Queue Manager.



Figure 5.7



Figure 5.8



Figure 5.9

1. *EFFORT SPENT*

* Etion Pinari

|  |  |
| --- | --- |
| Topic | #Hours |
| Introduction | 8 |
| Overview: high-level components and their interaction | 1 |
| Components diagram | 1.5 |
| Deployment view | 1.5 |
| Runtime view | 0.5 |
| Components interface | 1.5 |
| Architectural styles and patterns | 6 |
| User interface design | 1 |
| Requirements traceability | 1 |
| Implementation, integration and test plan |  |
| Research on various topics | 3 |
| Writing on Word and formatting | 5 |
| Total | 30 |

* Giorgio Romeo

|  |  |
| --- | --- |
| Topic | #Hours |
| Introduction | 8 |
| Overview: high-level components and their interaction | 1 |
| Components diagram | 1.5 |
| Deployment view | 1.5 |
| Runtime view | 0.5 |
| Components interface | 1.5 |
| Architectural styles and patterns | 6 |
| User interface design | 1 |
| Requirements traceability | 1 |
| Implementation, integration and test plan |  |
| Research on various topics | 3 |
| Writing on Word and formatting | 5 |
| Total | 30 |

* Cristian Sbrolli

|  |  |
| --- | --- |
| Topic | #Hours |
| Introduction | 8 |
| Overview: high-level components and their interaction | 1 |
| Components diagram | 1.5 |
| Deployment view | 1.5 |
| Runtime view | 0.5 |
| Components interface | 1.5 |
| Architectural styles and patterns | 6 |
| User interface design | 1 |
| Requirements traceability | 1 |
| Implementation, integration and test plan |  |
| Research on various topics | 3 |
| Writing on Word and formatting | 5 |
| Total | 30 |

1. *REFERENCES*

* Web App to generate mockups: <https://bubble.io/home>
* Web app for components diagram: <https://online.visual-paradigm.com/diagrams/>
* Web app for sequence diagrams: <https://app.diagrams.net/>