Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

CLup
Design Document

Andrea Riva
10560217
Alessandro Sanvito
10578314
Luca Vecchio
10565156

December 27, 2020

POLITECNICO
MILANO 1863

Contents

1. Introduction

1.1 Purpose

This document provides the functional description of CLup, a system that allows handling access to supermarkets when the flux of people is restricted. Starting from CLup's Requirements And Specification Document, the system architecture is described, together with the design of the user interfaces. Each system component is mapped to one or more requirements specified in the RASD document. Finally, adequate plans to adopt when implementing, integrating and testing the system are provided.

1.2 Scope

CLup is a system that allows supermarkets' customers to line-up remotely (i.e., without being physically in a line outside the supermarket) and to manage their access to the supermarkets to prevent overcrowding. The reasons behind these needs are described in details in the CLups' Requirements And Specification Document.

To be effective, CLup must be adopted by the greatest number of supermarkets possible, and be usable by the greatest number of customers possible. To reach such an objective, the scalability and the usability of the system are taken into account when defining:

- the system architecture, which should scale well with the rate of adoption of the system;
- the user experience, which should be effective for people of all ages and conditions;
- the testing of all the system's components and of the system as a whole, to guarantee an adequate level of quality.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

Backend The part of a computer system or application that is not directly accessed by the user, typically responsible for storing and manipulating data.

CI Pipeline A series of steps that introduces automation to improve the process of application development, particularly at the integration and testing phases.

Elasticity The degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner.

Layer A logical structuring mechanism for the elements that make up a software solution.

Load balancer A device that acts as a reverse proxy and distributes network or application traffic across a number of servers.

Pilot project An initial small-scale implementation that is used to prove the viability of a project idea.

Tier A physical structuring mechanism for a system infrastructure.

Web application An application software that runs on a web server and is accessed by the user through a web browser with an active internet connection.

1.3.2 Acronyms

CI Continuous Integration.

DD Design Document.

CLup Customers Line-up.

RASD Requirements Analysis and Specification Document.

UML Unified Modeling Language.

1.3.3 Abbreviations

1.4 Revision history

Version	Date	Notes
V1.0	December 27, 2020	Initial release.

1.5 Reference documents

- **R&DD Assignment AY 2020-2021**
- **CLup - Requirements Analysis and Specification Document**
- **UML documentation**
- **Cloud Computing Patterns** by Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.

1.6 Document structure

This document is structured in the following way:

1. The first chapter is an introduction and an overview of the project, setting the context leading to the design choices made in this document.

2. The second chapter describes the system architecture, providing at first a high level description of the components and their interactions, then describing in detail each component both through static and dynamic views. Moreover, this chapter describes how the components are to be deployed using well known architectural styles and patterns and detailing the motivations for each choice.

3. The third chapter extends the "User interfaces" section described in the RASD by adding further details on how all the interactions with the users will be handled.

4. The fourth chapter provides traceability information that allows to map the components described in this document with the requirements enumerated in the RASD.

5. The fifth and last chapter provides the plans that should be adopted when implementing, integrating and testing the system.

2. Architectural design

2.1 Overview

As stated in the RASD, from an initial pilot project, it is forecasted that CLup will quickly scale up to a large user base, comprising a large portion of the national population of the countries where it will be deployed. To support such a vast and heterogeneous user base, CLup will offer a unified interface to all IT devices in the form of a web application, optimized for the different environments. In order to be flexible enough in handling the growth of the user base, a cloud architecture seems most suited to the task. In particular, CLup's backend, in a client-server fashion, is centred around a three-tier structure, where the use of a load balancer and queues between the tiers ensures a high decoupling. Thus, each tier can evolve independently and can be resized according to the actual load, exploiting the elasticity offered by a cloud environment.

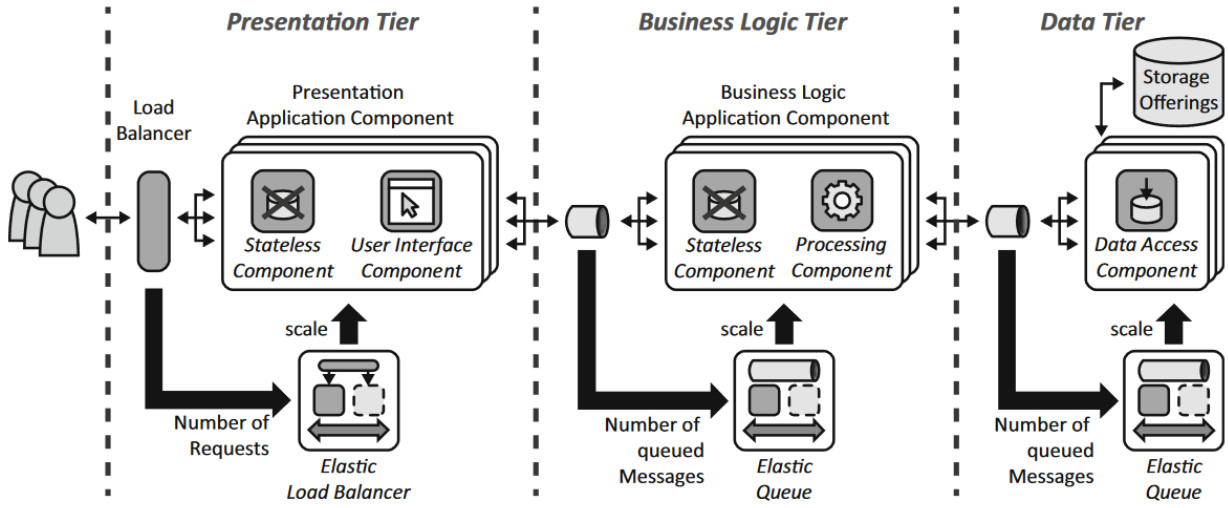


Figure 2.1: The inspiration for CLup's architecture: a generic three-tier cloud application
Source: Cloud Computing Patterns

An additional tier is then placed between the user and the application to act as a CDN, to relief some load from the system and improve performances in serving static contents. Overall, the resulting application will have five tiers. The presentation layer will be distributed between the user's device, the CDN and the first server tier of the backend, which will act as a web server. The application layer will map on the second server tier of the backend, while the data layer will map on the third.

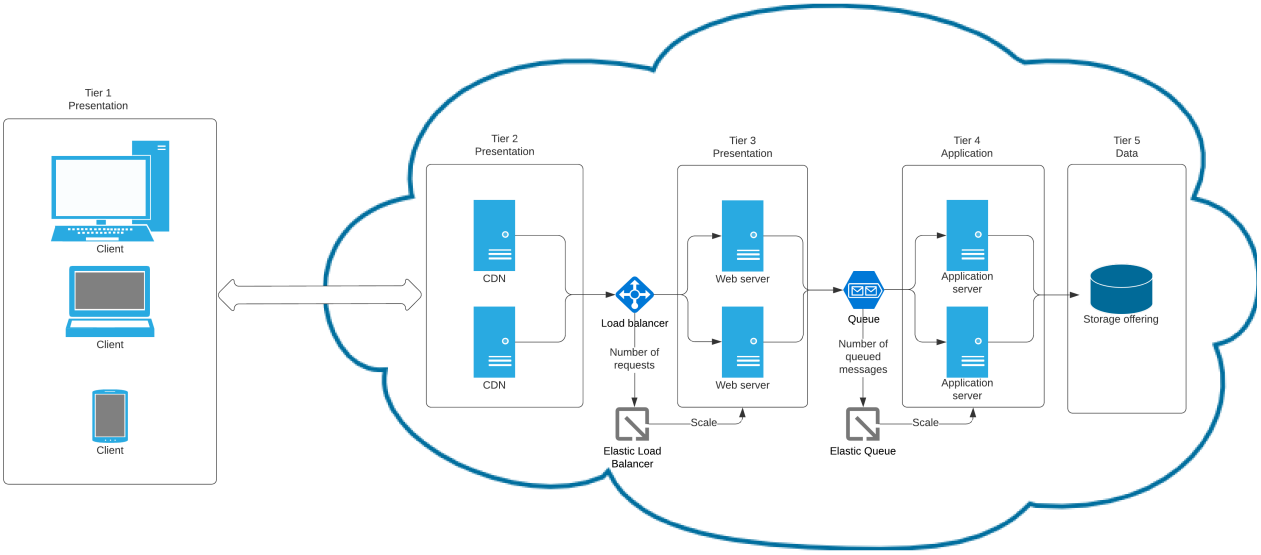


Figure 2.2: CLup high level architecture

2.2 Component view

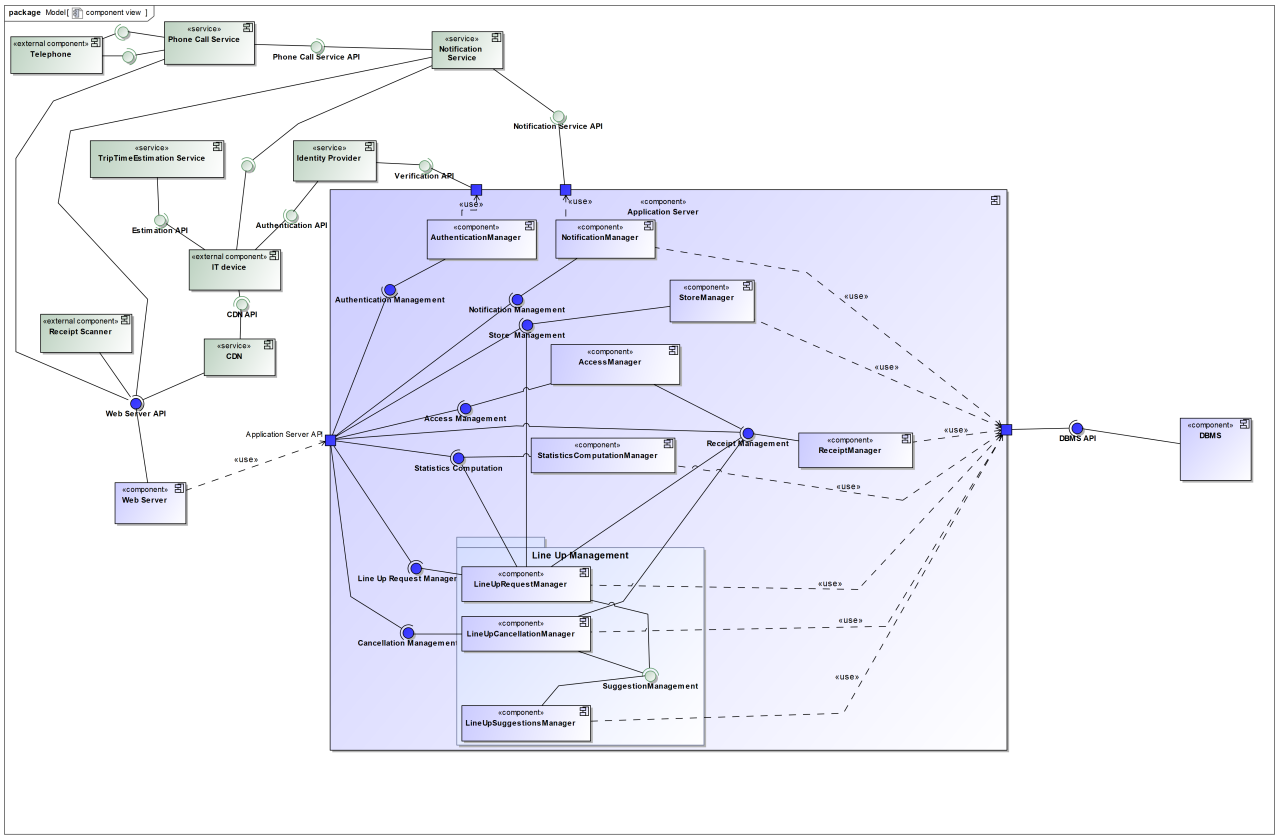


Figure 2.3: Component diagram

The component diagram above presents a general view of the application. The image describes the application server and its components with the highest level of detail, as developers will have to implement the business logic themselves. The other nodes in the system, as well as the services, are depicted more abstractly. The services include a notification service, an authentication service, an identity provider, and a trip time estimation service. The trip time estimation service is in charge of estimating the time taken to reach a destination, in this case the store, from the IT device's position. Developers of the project might choose to use commercial off the shelf solutions to provide those APIs, thus losing control on the internal components, but accelerating the development.

The main components in the application server, providing all the business logic in CLup, are:

AccessManager The AccessManager encapsulates the business logic necessary to manage both entries in and exits from the store. This component interacts with the ReceiptManager both to validate the entrance and to record the departure from the store. The AccessManager also unlocks the entrance if the receipt is valid.

AuthenticationManager The AuthenticationManager handles authentication for the functionalities of the system reserved to store managers and store assistants. The authentication manager delegates the verification of the identity of the user to an external provider.

LineUpCancellationManager The LineUpCancellationManager is in charge of the cancellation process for a reserved time slot. The change is performed in the persistent storage of the database and the receipt is marked as invalid through the interface provided by the ReceiptManager. The LineUpCancellationManager also interacts with the LineUpSuggestionsManager to find if other customers would like to take the freed time slot.

LineUpRequestManager The LineUpRequestManager implements the business logic behind the lining up to the store. The LineUpRequestManager is in charge of both immediate queueing and reservations, checking the availability of the time slot in the given store from StoreManager and then interfacing with the ReceiptManager to generate the receipt to enter the store. During the process, the component also retrieves statistics from the StatisticsComputationManager. If the time slot is not available, the LineUpRequestManager interfaces with the LineUpSuggestionsManager to find alternatives.

LineUpSuggestionsManager The LineUpSuggestionsManager is in charge of finding alternatives to the time slot and store requested by the customer if they are not available. To do so, it interacts with the database retrieving data regarding the queues to other stores. The LineUpSuggestionsManager is also in charge of finding customers interested in a time slot.

NotificationManager The NotificationManager is in charge of sending notifications to customers when their visit's time is near. This component is the only one not implementing a typical client-server interaction in the application server. The push behaviour in the notification management is achieved by using the interfaces provided by notification services. The NotificationManager interacts with the database periodically to check the customers to be notified and provides an interface to the Web Server to receive confirmation that the notification has been shown to the customer.

ReceiptManager The ReceiptManager is in charge of the operations performed on the receipt by updating its state in the database. CRUD operations on the receipt are performed through this component. This component subsequently handles the status of the visit in the database.

StatisticsComputationManager The StatisticsComputationManager is responsible for generating the statistics for a given store at request by consulting the database and providing an interface to retrieve them.

StoreManager The StoreManager implements the interface to set and retrieve all parameters regarding the store. In particular, the StoreManager allows store managers to set the maximum number of people allowed inside. The StoreManager then mirrors the change in the database.

2.3 Deployment view

The following diagram describes the deployment view of CLup. CLup's backend is deployed in a cloud platform, where elasticity is achieved by using autoscalers, which scale the instances count assigned to each tier based on the load measured in the load balancer or in the queue. The CDN is here represented as an external service, which might be provided by the same cloud platform on which CLup's backend is deployed.

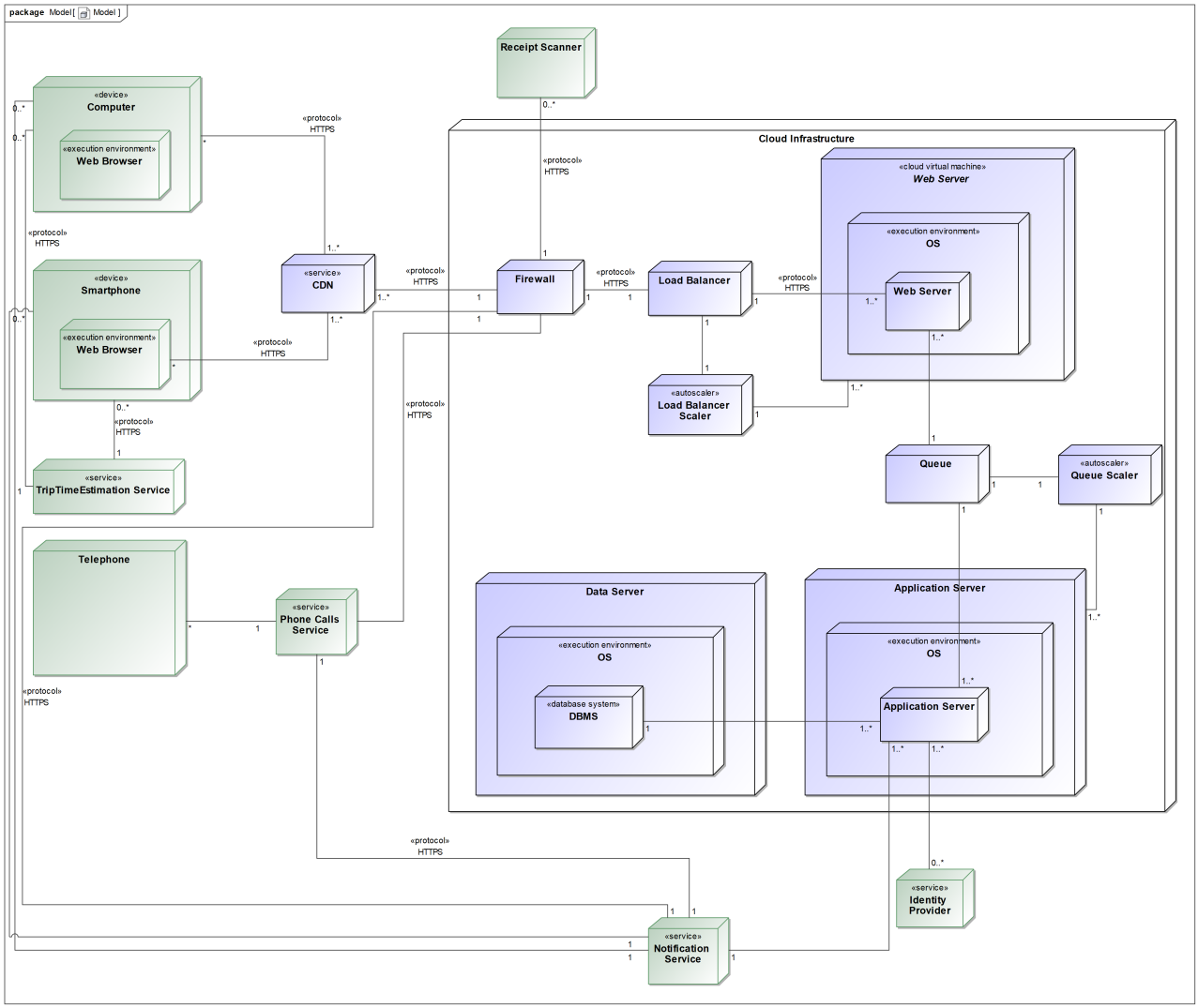


Figure 2.4: Deployment diagram

2.4 Runtime view

This view describes the behavior and interaction of the system's building blocks as runtime elements.

2.4.1 Runtime Scenario 1 - Store employees login

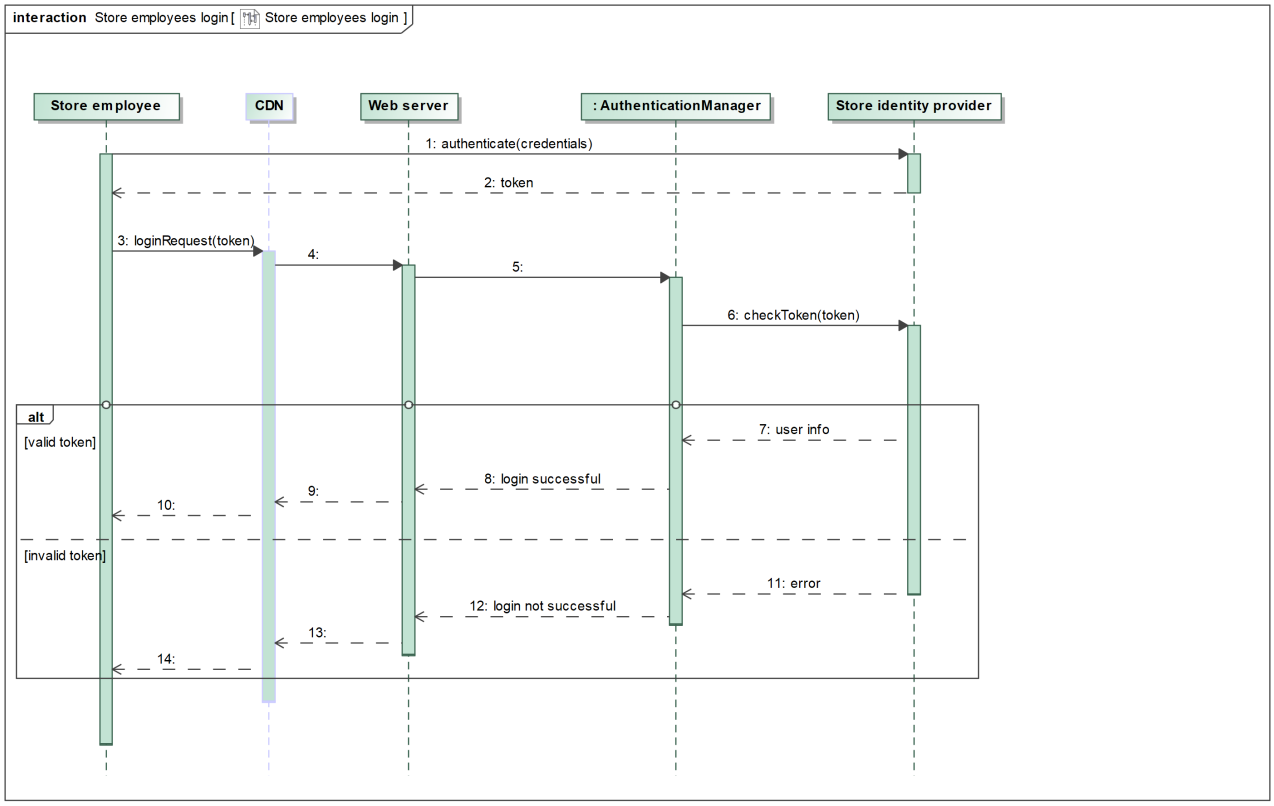


Figure 2.5: Store employees login sequence diagram.

2.4.2 Runtime Scenario 2 - Definition of safety standard parameters

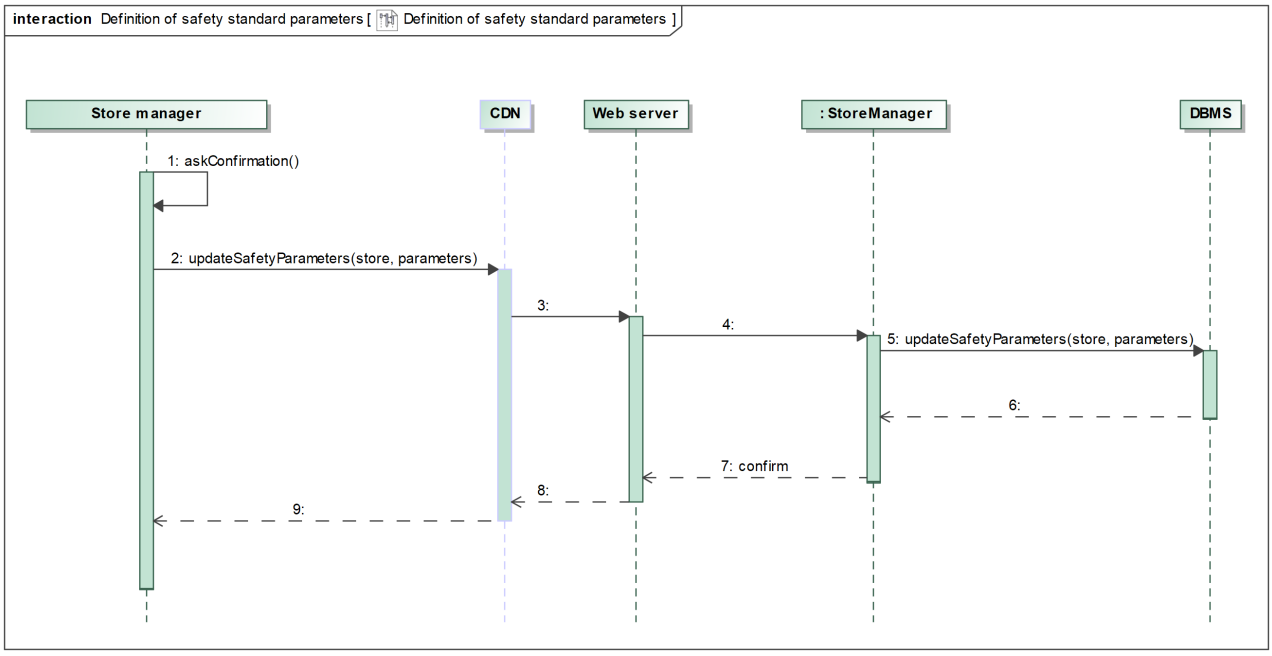


Figure 2.6: Definition of safety standard parameters sequence diagram.

2.4.3 Runtime Scenario 3 - Consulting statistics

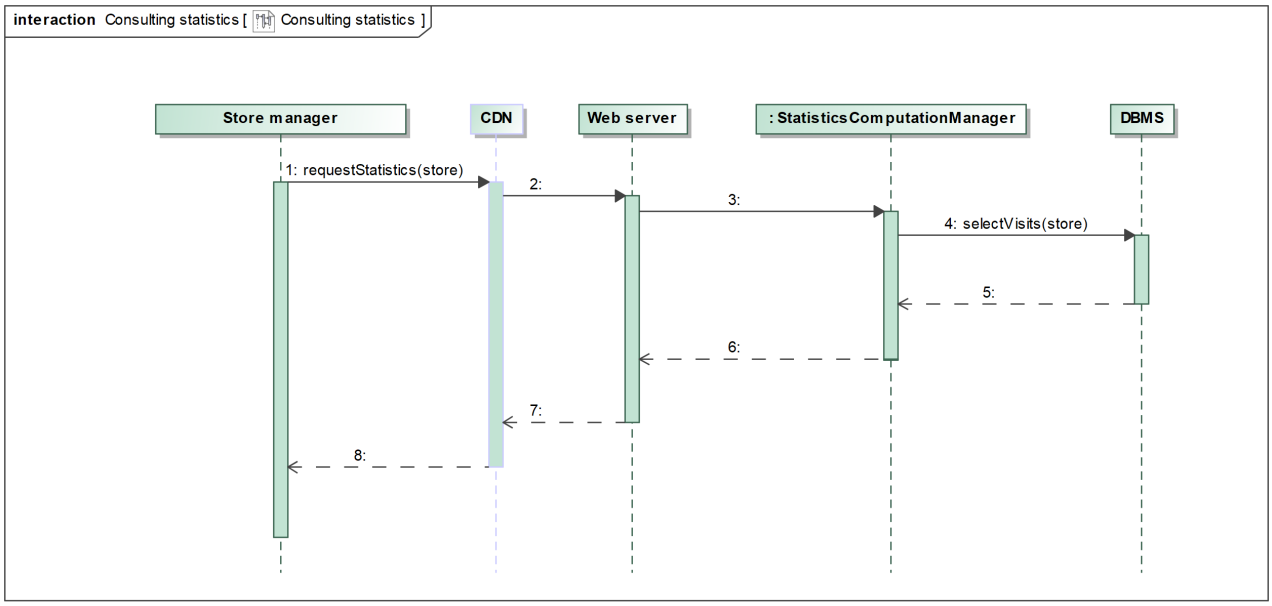


Figure 2.7: Consulting statistics sequence diagram.

2.4.4 Runtime Scenario 4 - In presence ticket generation

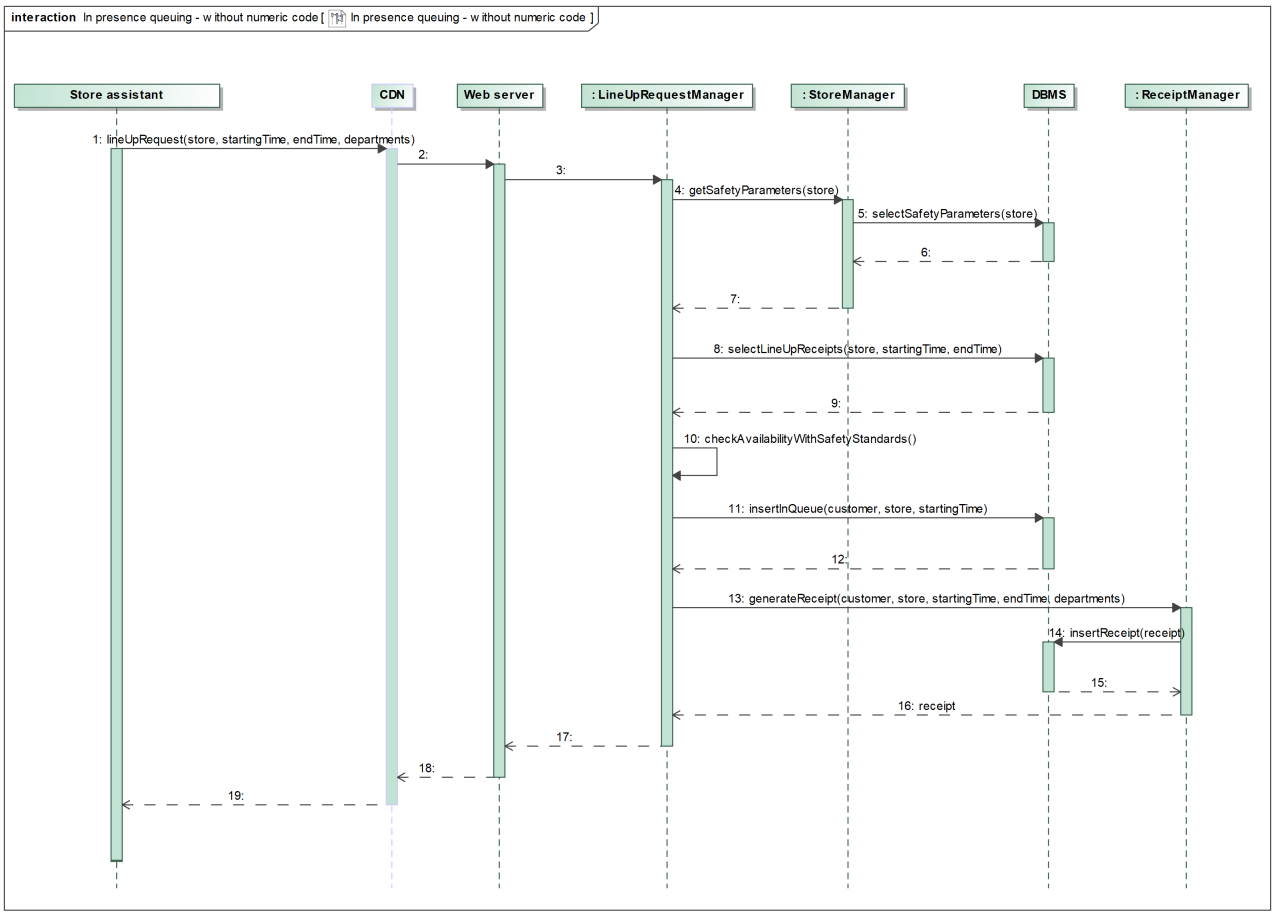


Figure 2.8: In presence ticket generation - without numeric code sequence diagram.

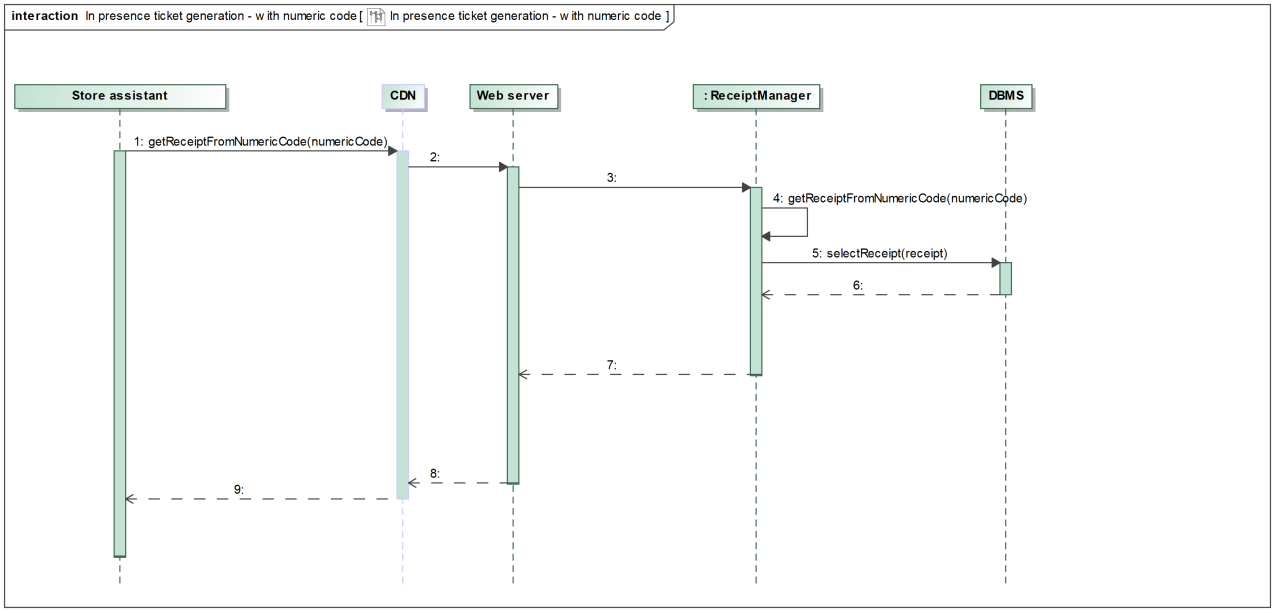


Figure 2.9: In presence ticket generation - with numeric code sequence diagram.

2.4.5 Runtime Scenario 5 - Queueing with telephone

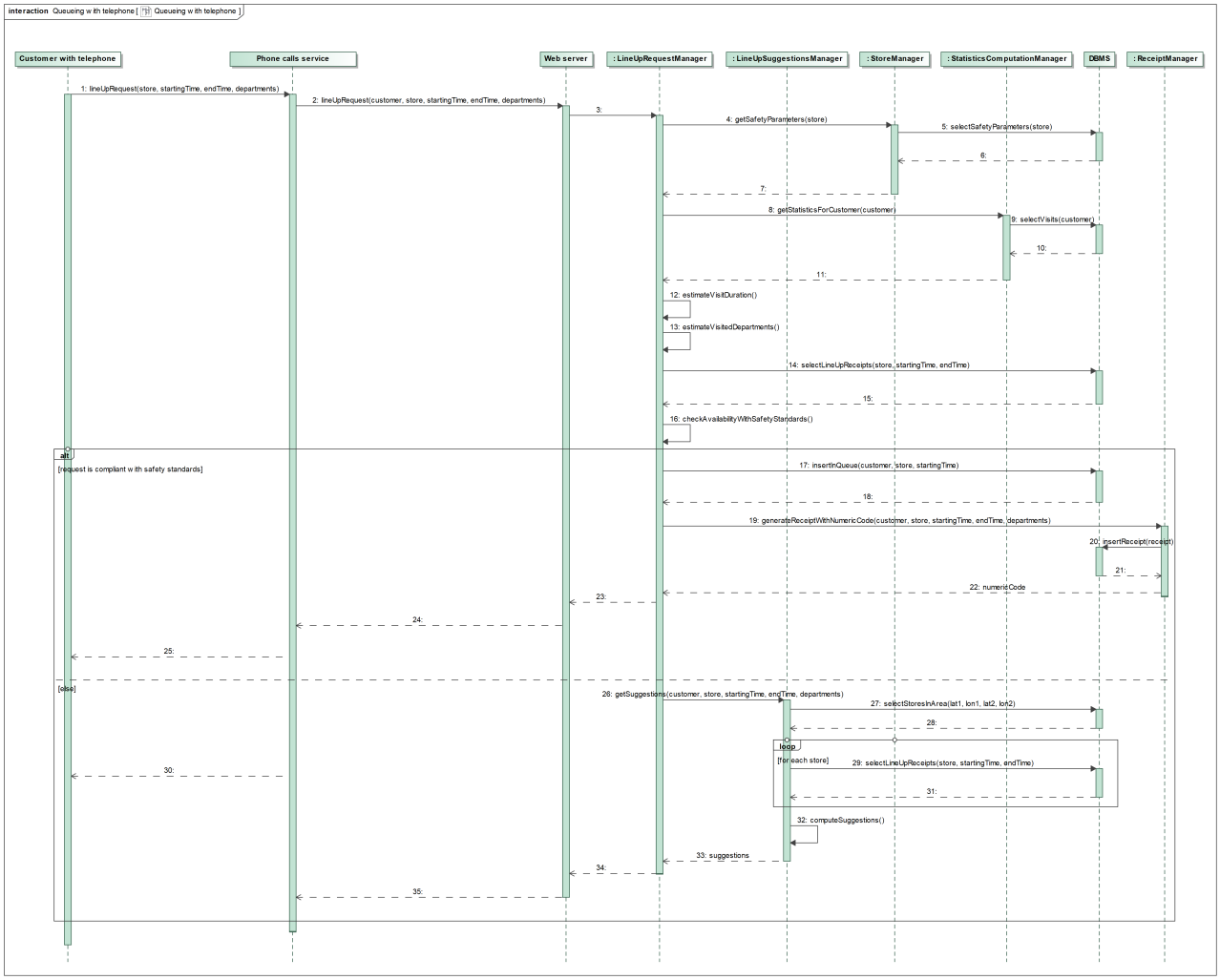


Figure 2.10: Queueing with telephone sequence diagram.

2.4.6 Runtime Scenario 6 - Queueing with IT device

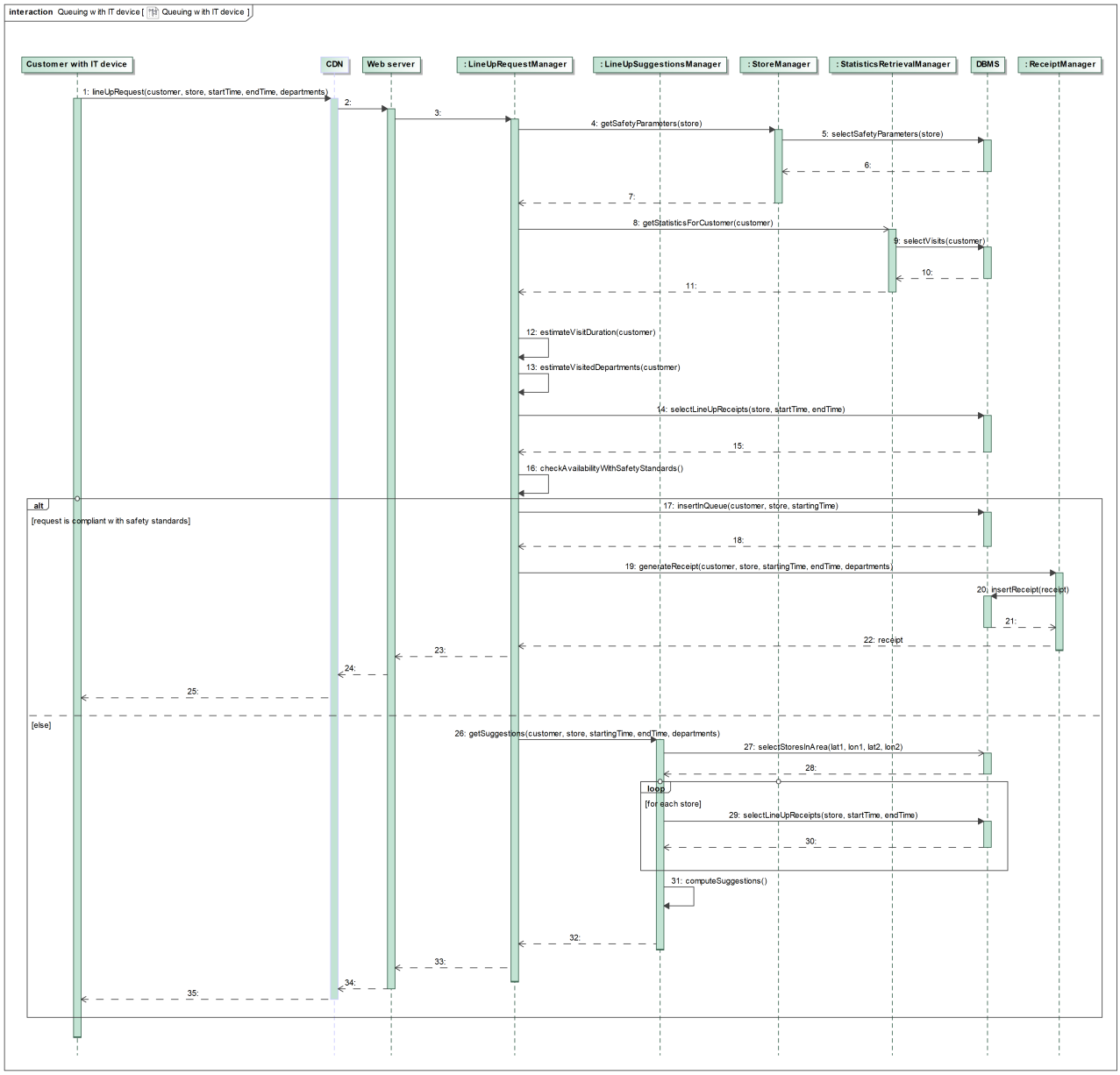


Figure 2.11: Queueing with IT device sequence diagram.

2.4.7 Runtime Scenario 7 - Cancellation

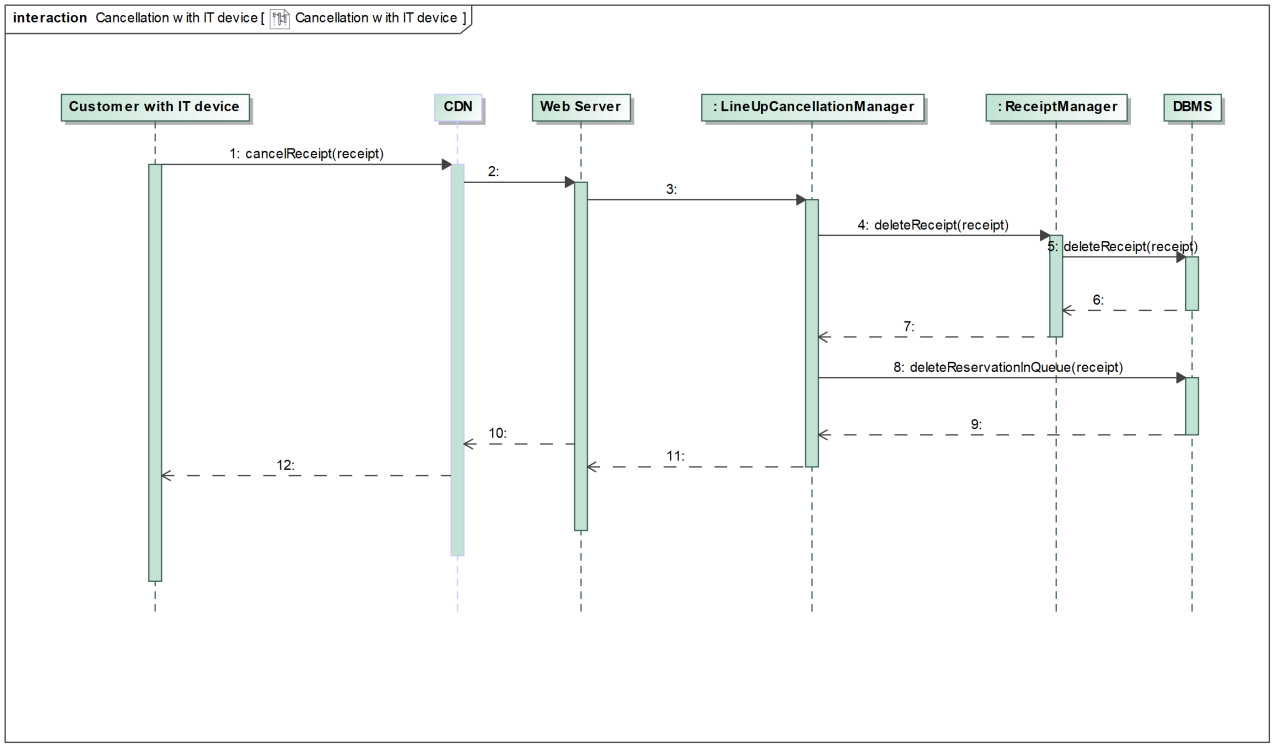


Figure 2.12: Cancellation with IT device sequence diagram.

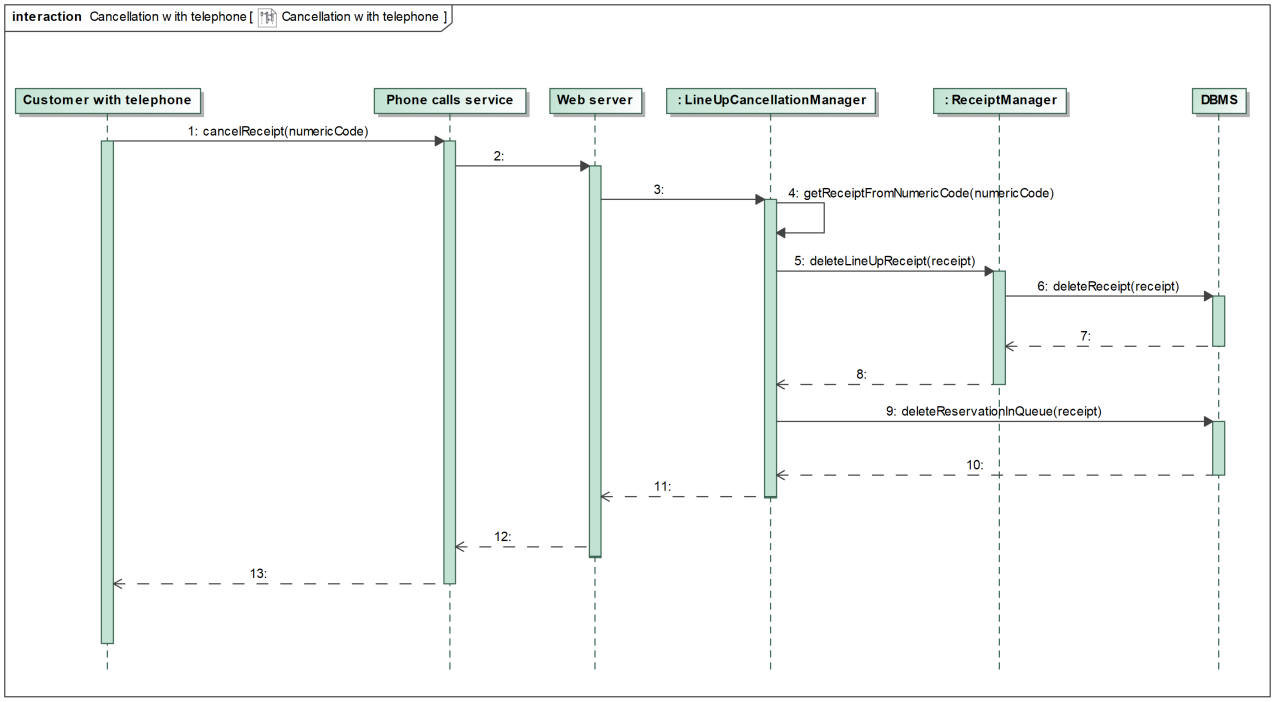


Figure 2.13: Cancellation with telephone sequence diagram.

2.4.8 Runtime Scenario 8 - CLup notification

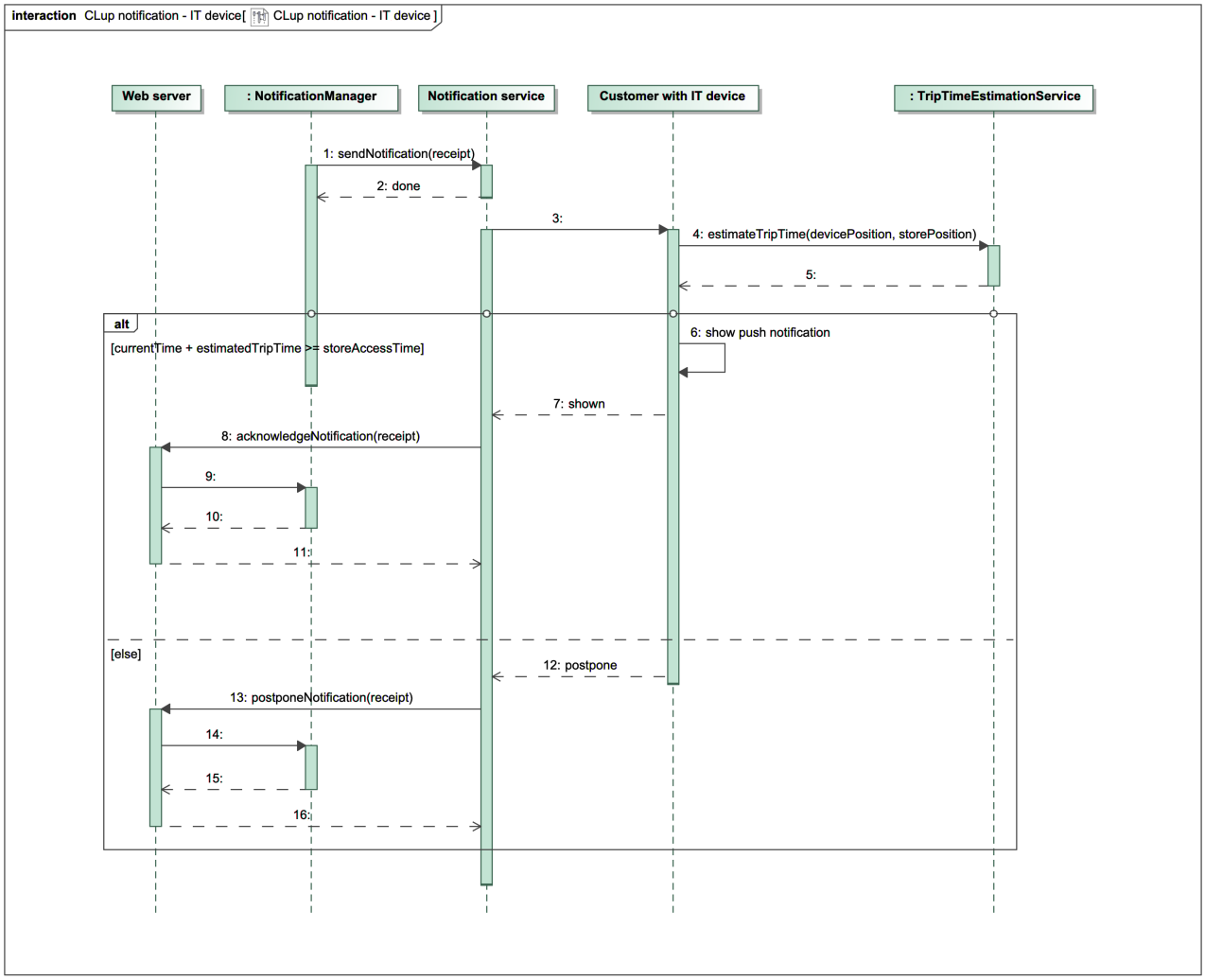


Figure 2.14: CLup notification - IT device sequence diagram.

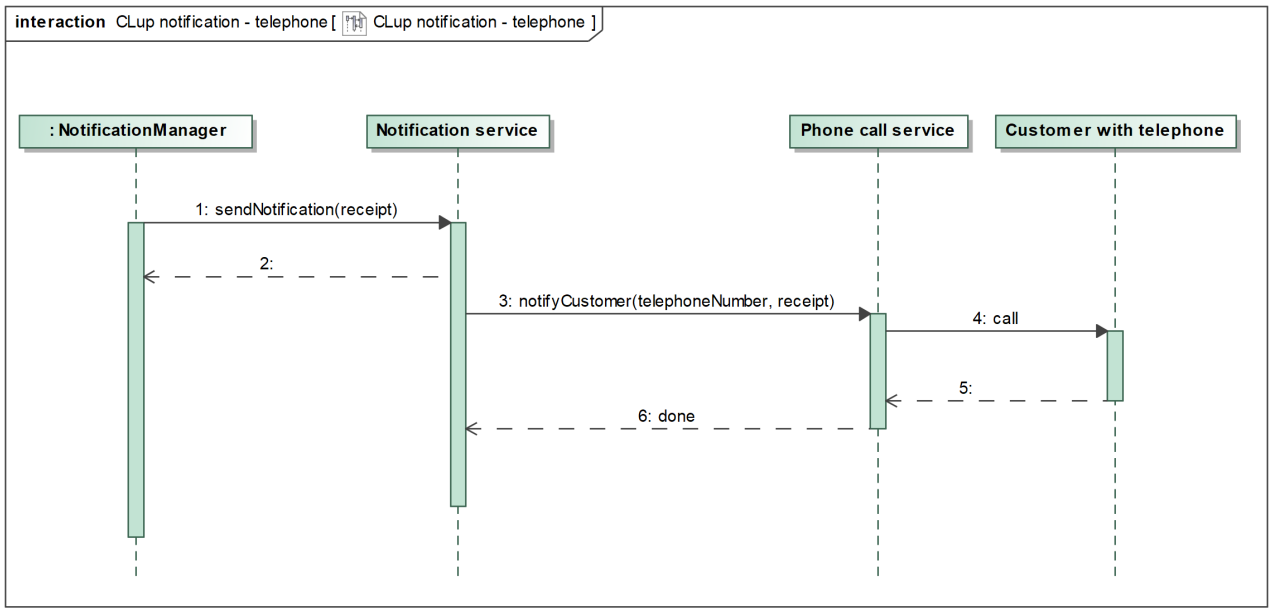


Figure 2.15: CLup notification - telephone sequence diagram.

2.4.9 Runtime Scenario 9 - Store access

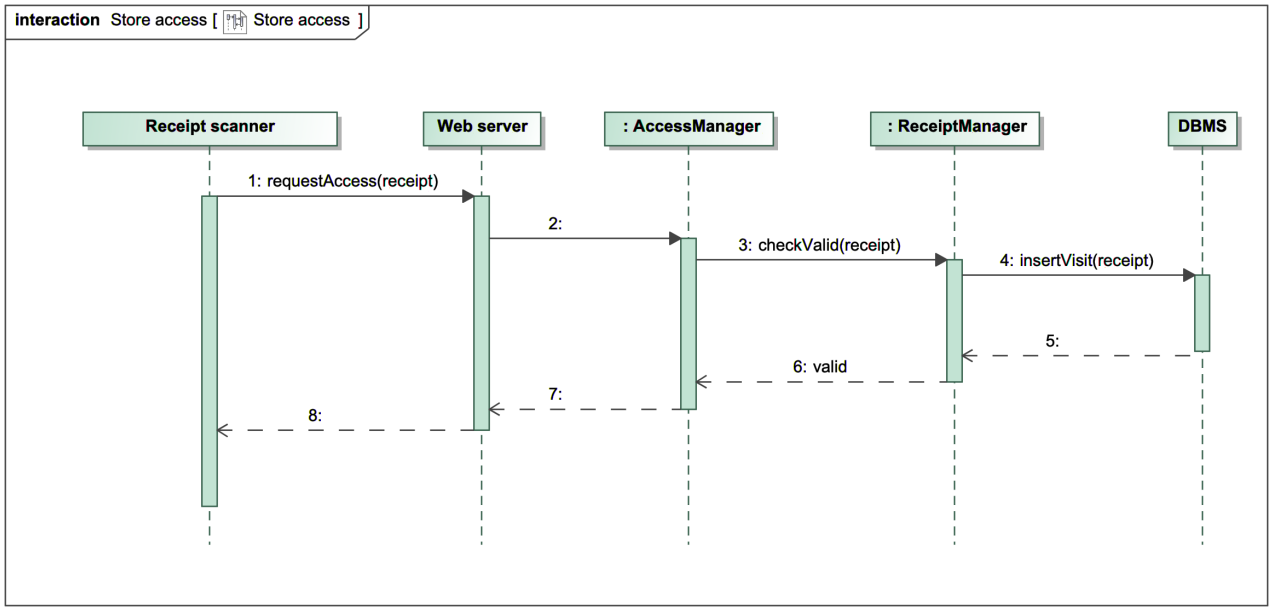


Figure 2.16: Store access sequence diagram.

2.4.10 Runtime Scenario 10 - Store exit

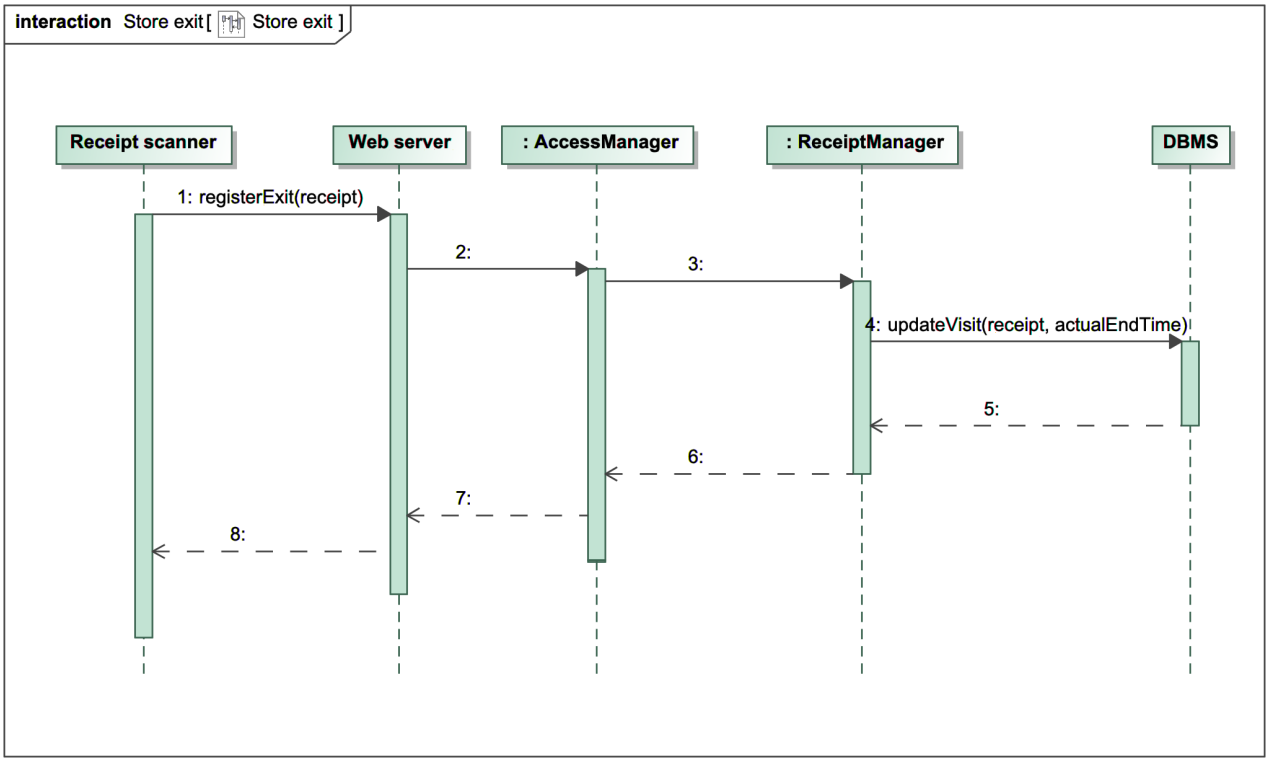


Figure 2.17: Store exit sequence diagram.

2.5 Component interfaces

```
// Components

public interface AccessManagement {
  void registerExit(LineUpReceipt receipt);
  boolean requestAccess(LineUpReceipt receipt);
}

public interface AuthenticationManagement {
  boolean loginRequest(String token);
}

public interface CancellationManagement {
  void cancelReceipt(LineUpReceipt receipt);
  void cancelReceipt(String numericCode);
}

public interface LineUpRequestManagement {
```

```
  LineUpReceipt lineUpRequest(Store store, LocalDateTime startingTime, LocalDateTime endTime,
      Collection<StoreDepartment> departments);
  LineUpReceipt lineUpRequest(Customer customer, Store store, LocalDateTime startingTime,
      LocalDateTime endTime, Collection<StoreDepartment> departments);
}

public interface NotificationManagement {
  void acknowledgeNotification(LineUpReceipt receipt);
  void postponeNotification(LineUpReceipt receipt);
}

public interface ReceiptManagement {
  boolean checkValidReceipt(LineUpReceipt receipt);
  void deleteReceipt(LineUpReceipt receipt);
  LineUpReceipt generateReceipt(Customer customer, Store store, LocalDateTime startingTime,
      LocalDateTime endTime, Collection<StoreDepartment> departments)
  String generateReceiptWithNumericCode(Customer customer, Store store, LocalDateTime startingTime,
      LocalDateTime endTime, Collection<StoreDepartment> departments)
  LineUpReceipt getReceiptFromNumericCode(String numericCode);
  }

public interface StatisticsComputationManagement {
  Collection<Visit> requestStatistics(Store store);
  Collection<Visit> getStatisticsForCustomer(Customer customer);
}

public interface StoreManagement {
  Map<StoreDepartment, int> getSafetyParameters(Store store);
  void updateSafetyParameters(Store store, Map<StoreDepartment, int> parameters);
}

public interface SuggestionManagement {
  Collection<Map<Store, LocalDateTime>> getSuggestions(Customer customer, Store store, LocalDateTime
      startingTime, LocalDateTime endTime, Collection<StoreDepartment> departments);
}

// DBMS

public interface DBMSAPI {
  void deleteReceipt(int receiptId);
  void deleteReservationInQueue(int receiptId);
  void insertInQueue(int customerId, int storeId, String startingTime);
  void insertReceipt(Object receipt);
  void insertVisit(int receiptId);
  Collection<Object> selectLineUpReceipts(int storeId, String startingTime, String endTime);
  Map<Object, id> selectSafetyParameters(int storeId);
  Collection<Object> selectVisits(int customerId);
  Collection<Object> selectVisits(int storeId);
  Object selectReceipt(int receiptId);
  void updateSafetyParameters(int storeId, Map<int, int> parameters);
  void updateVisit(int receiptId, String actualEndTime);
}

// External services
```

```
public interface AuthenticationAPI {
  String authenticate(Object credentials);
}

public interface EstimationAPI {
  Duration estimateTripTime(Location devicePosition, Location storePosition);
}

public interface NotificationServiceAPI {
  void sendNotification(LineUpReceipt receipt);
}

public interface PhoneCallServiceAPI {
  void notifyCustomer(String telephoneNumber, LineUpReceipt receipt);
}

public interface VerificationAPI {
  boolean checkToken(String token);
}
```

2.6 Selected architectural styles and patterns

As already stated in the overview, the general architecture draws inspiration from the classical client-server architectural style, adapted in a multi-tiered cloud configuration. This pattern is field-proven and offers enough space for future adjustments due to the separation of concerns fostered by the mapping of tiers into different layers. The presence of only web applications, thus implementing only the presentation layer, is typical of a thin client. The main advantage of such a configuration is the simplified management both for the customer, not forced to download any software, and for the developers, who can adopt a write once, run everywhere approach and push updates to the clients very easily.

The architecture implements a publish/subscribe pattern to handle notifications to the clients, exploiting existing notification services. The publish/subscribe pattern also collapses into a queue-based communication protocol to ensure the highest decoupling between the web server and the application server.

Finally, the architecture makes use of an external CDN to distribute static content with the highest efficiency and performance. The CDN caches the static contents that implement the client of the web app, while it acts as a reverse proxy for the API requests to the backend.

2.7 Other design decisions

Other design decisions involve the use of HTTPS for communication between IT devices and the web server, as appropriate for a web application. Moreover, the usage of HTTPS fosters security as requested by the RASD.

No other specific technology is mandatory, even though it is advisable to implement the data tier as a relational database, as CLup will be dealing with structured data. Moreover, relational databases are a mature technology, and commercial solutions offer distributed configurations, which are well suited to the large predicted user base.

Communication via telephone relies external services that can both receive and place outgoing telephone calls. Such services, which are assumed to communicate with the backend through a RESTful interface, should provide the following functionalities:

- place telephone calls;
- report to the application server when a new call is received
- receive from the application server arbitrary text and perform voice synthesis in an active call;
- report to the application server the input that the user provided in an active call, both using DTMF tone recognition and voice recognition

The notification service is assumed to offer the possibility to report to the application server, through a REST call, the outcome of the notification request according to the response provided by the end device.

The trip time estimation service is assumed to offer a REST API that can be used by the IT devices to determine when it's time to show the reminder for the user to head to the store. Customers who placed their reservation using a telephone cannot be localized with enough precision, so they will be notified with a phone call that will be placed a fixed amount of time before the entrance time.

Finally, to identify customers, each device should be assigned a random and unique identifier stored securely in the database. Store managers and store assistants are identified through an identity provider with either SAML or OAuth.

3. User interface design

3.1 Overview

The mockups of the graphical user interfaces to interact with CLup using an IT device were presented in section 3.1.1 of the Requirements And Specification Document. The graphical user interfaces will be translated into English and Italian; the device settings will be used to determine the users' preferred language. If the user preferred language is not available, English will be chosen.

While store managers and store assistants only interact with the system through IT devices, customers may interact with the system also using a standard telephone line or in presence (through store assistants). The interactions with the store assistants are human-to-human, so they are not described in this document. The interactions through a telephone line, instead, are described in the next paragraph, as a reference for the programmatic implementation that is required.

3.1.1 Customers interactions through a standard telephone line

Interactions with the customers will be either initiated by the customers themselves (e.g., to reserve their place in a queue) by calling CLup's telephone number, or initiated by the system (e.g., to notify that it is time to leave to reach the store in time for the expected entrance time) by calling the telephone number the customers originally used to interact with the system. The customers will be allowed to interact with the system using the language of their choice between English and Italian. Different telephone numbers will be provided to the users to let them choose their preferred language.

Voice recognition or DTMF tone recognition will be used to receive inputs from the customers, while the system will use voice synthesis to reply.

Each store will be assigned a numeric identifier that can be used by the customers to reference it.

As a receipt for their line up requests, customers will receive from the system a numeric code that they will communicate before entering the store to the store assistant, who will insert it in the store assistant application to print the QR code that the customer will use to access the store.

Line up immediately

```
System: Welcome to CLup. If you want to line up, press 1 or say line up.
        If you want to cancel a previous reservation, press 2 or say cancel.
Customer: Line up.
S: If you know the numeric identifier of the shop for which you want to line up,
   press 1 or say yes. If not, press 2 or say no.
C: No.
S: In what city is the shop for which you want to line up?
C: <City>.
S: What is the name of the shop for which you want to line up?
```

C: <Name of the shop>.
S: If you want to line up immediately, press 1 or say immediately.
   If you want to line up in the future, press 2 or say in the future.
C: Immediately.
S: Say or enter the estimated duration of your visit in minutes.
C: 30.
S: You asked to queue for <Name of the shop> in <City> for 30 minutes.
   Press 1 or say confirm to confirm, press 2 or say change to change something.
C: Confirm.
S: The first availability is at 3.15 PM.
   Say yes or press 1 to accept. Say no or press 2 to find an alternative.
C: Yes.
S: You have lined up to access <Name of the shop> in <City> at 3.15 PM for 30 minutes.
   This is the numeric code you will need to provide to a store assistant at the entrance
   of the shop to enter: 123456789. You can now hang up the call. If you want the code
   to be repeated, press 1 or say repeat.
C: *hangs up*

Line up in the future

System: Welcome to CLup. If you want to line up, press 1 or say line up.
        If you want to cancel a previous reservation, press 2 or say cancel.
Customer: *presses 1*
S: If you know the numeric identifier of the shop for which you want to line up,
    press 1 or say yes. If not, press 2 or say no.
C: *presses 1*
S: Enter the numeric identifier of the shop for which you want to line up, or say it
   one digit at a time.
C: *enters numeric identifier of the shop*
S: You have chosen <Name of the shop> in <City>. If you want to line up
   immediately, press 1 or say immediately. If you want to line up in the
   future, press 2 or say in the future.
C: *presses 2*
S: Say or enter the number of the month for the reservation of your visit.
C: *presses 1* *presses 1*
S: Say or enter the number of the day for the reservation of your visit.
C: *presses 2* *presses 2*
S: Say or enter the hour (in 24 hours format) for the reservation of your visit.
C: *presses 1* *presses 5*
S: Say or enter the estimated duration of your visit in minutes.
C: *presses 4* *presses 5*
S: You asked to queue for <Name of the shop> in <City> for 45 minutes
   on 22/11 starting from 3 PM.
   Press 1 or say confirm to confirm, press 2 or say change to change something.
C: *presses 1*
S: The first availability is at 3.15 PM.
    Say yes or press 1 to accept. Say no or press 2 to find an alternative.
C: *presses 2*
S: The following availability is at 3.45 PM.

```
    Say yes or press 1 to accept. Say no or press 2 to find an alternative.
C: *presses 1*
S: You have lined up to access <Name of the shop> in <City> at 3.45 PM for 45 minutes.
    This is the numeric code you will need to provide to a store assistant at the entrance
    of the shop to enter: 123456789. You can now hang up the call. If you want the code
   to be repeated, press 1 or say repeat.
C: *hangs up*
```

Cancel a reservation

```
System: Welcome to CLup. If you want to line up, press 1 or say line up.
        If you want to cancel a previous reservation, press 2 or say cancel.
Customer: Cancel.
S: Enter the numeric code of the reservation you want to cancel, or say it
one digit at a time.
C: 1 2 3 4 5 6 7 8 9
S: If you are sure you want to cancel your reservation at <Name of the shop> in <City>
   at <Time>, press 1 or say yes. Otherwise, press 2 or say no.
C: Yes.
S: You have cancelled your reservation at <Name of the shop> in <City> at <Time>.
   You can now hang up the call.
C: *hangs up*
```

Noification from CLup

```
System: This is a reminder for CLup that your visit at <Name of the shop> in <City>
        is planned at <Time>. Remember to be at the shop entrance on time.
        If you realize you will not be able to get to the shop, press 1 or say cancel.
        Otherwise, you can hang up the call.
Customer: *hangs up*
```

4. Requirements traceability

4.1 Overview

The following table shows the mappings between the Requirements presented in the Requirements And Specification Document and the components described in this document that implement their functionalities.

Component	Requirement IDs																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
AccessManager	x									x		x					
AuthenticationManager		x	x		x												
LineUpCancellationManager																x	x
LineUpRequestManager	x			x	x	x	x	x	x				x	x	x	x	
LineUpSuggestionsManager													x	x	x		
NotificationManager											x					x	
ReceiptManager						x				x		x					x
StatisticsComputationManager			x					x	x								
StoreManager		x	x	x			x						x	x			

5. Implementation, integration and test plan

5.1 Overview

This section concerns the implementation, integration and testing plan of the components described in the component view section (2.2). For each part it is defined a strategy that should be followed by the various teams developing CLup in order to deliver the product in the most efficient way, with the lowest time to market possible and, at the same time, conforming to the software system attributes which are defined in RASD section 3.5.

5.2 Implementation Plan

The entire system, along with its relative sub-systems, has to be implemented, tested and integrated using a bottom-up approach. With this strategy, the system can be assembled in an incremental way so that the testing process can begin in parallel with the implementation. This approach allows for better robustness since the components are implemented, tested and validated in a hierarchical order, from bottom to top level.

The order of implementation follows an hierarchical order, from the bottom-most components, with a closer interaction with the database, to the upper-most components, which are farther from the back end components and may depend on other ones. This way it is guaranteed that the system can be built in an incremental way, and the testing and integration process can begin in parallel with the implementation, therefore allowing for a better bug tracking, which leads to better quality. Consequently, the components should be implemented following this order:

1. StoreManager
2. ReceiptManager
3. StatisticsComputationManager
4. LineUpSuggestionsManager
5. AccessManager
6. LineUpRequestManager
7. LineUpCancellationManager
8. NotificationManager
9. AuthenticationManager
10. Web server (if it is chosen to implement it rather than using a pre-existing one)

Developers might implement the client-side components of the application in parallel with the rest of the system. In particular, the user interface is independent of the web server APIs, and designers and developers might produce prototypes and mocks and implement them as soon as possible, validating and verifying them. Then, when all the backend is complete, it will be possible to attach its API calls to the clients.

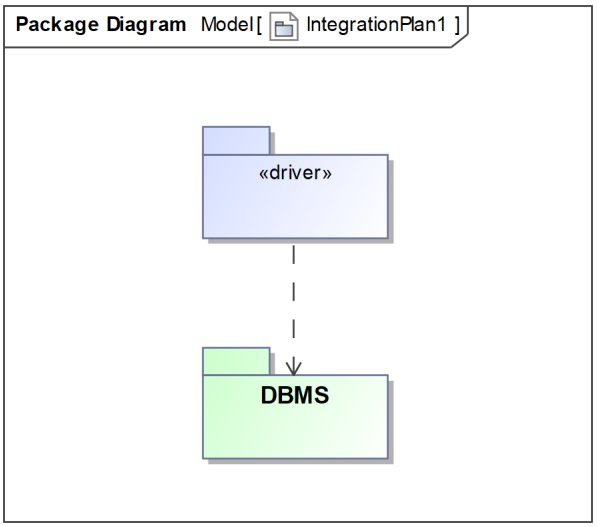
5.3 Integration and Testing Plan

5.3.1 Overview

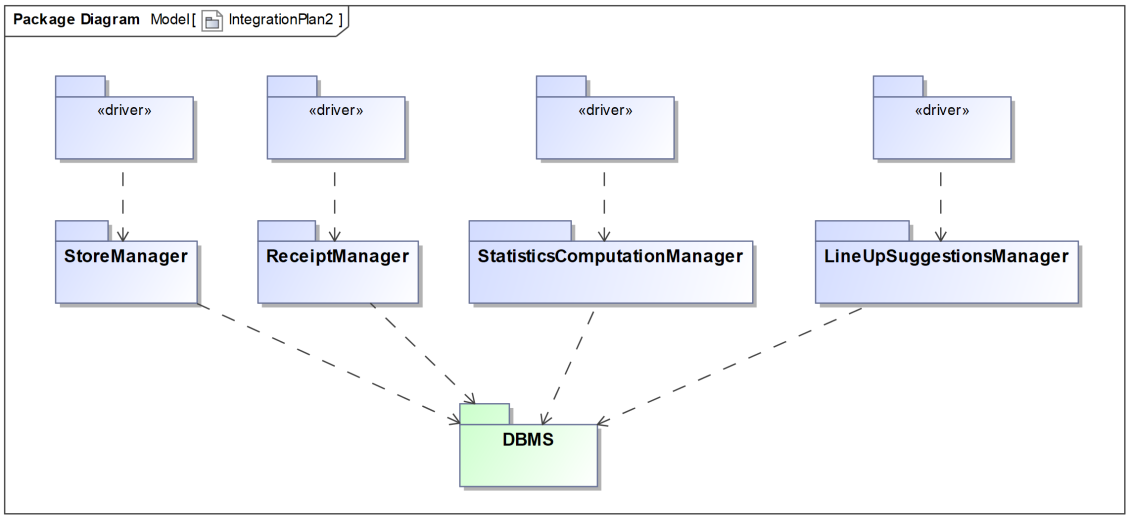
Immediately after the implementation of a component, it should be unit tested; if needed, a driver can be used in place of the components that are not yet fully implemented. As a consequence of choosing a bottom-up approach in the implementation process, we should choose a bottom-up strategy also for integration and testing as well. After the unit tests of a component succeeds, the component is integrated in the system and integration testing is performed: every interface that the component uses is tested. Since a bottom-up approach is used, this process ends when the components at the top of the hierarchy are integrated in the system. At that point, the system as a whole can be tested with system tests.

5.3.2 Plan

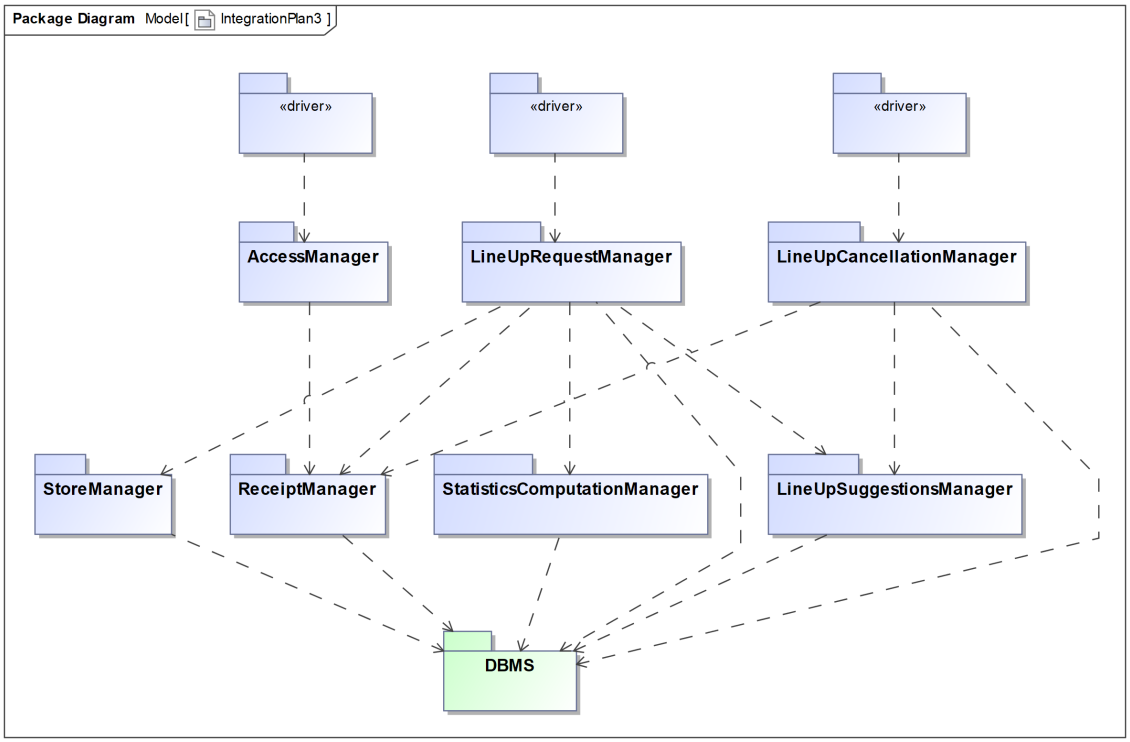
1. At first, the DBMS API has to be tested, in order to check for misconfiguration and correctness of the interface.



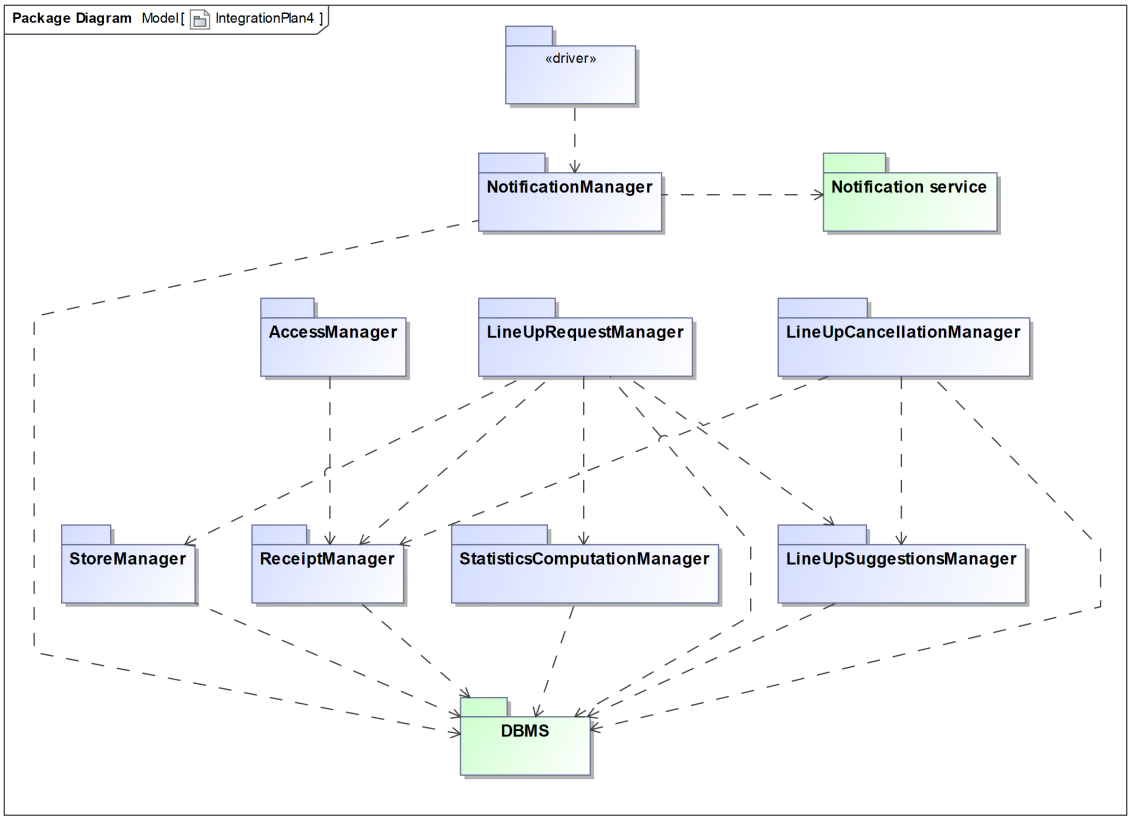
2. Then, the implementation goes on for StoreManager, ReceiptManager, StatisticComputationManager, and LineUpSuggestionsManager.



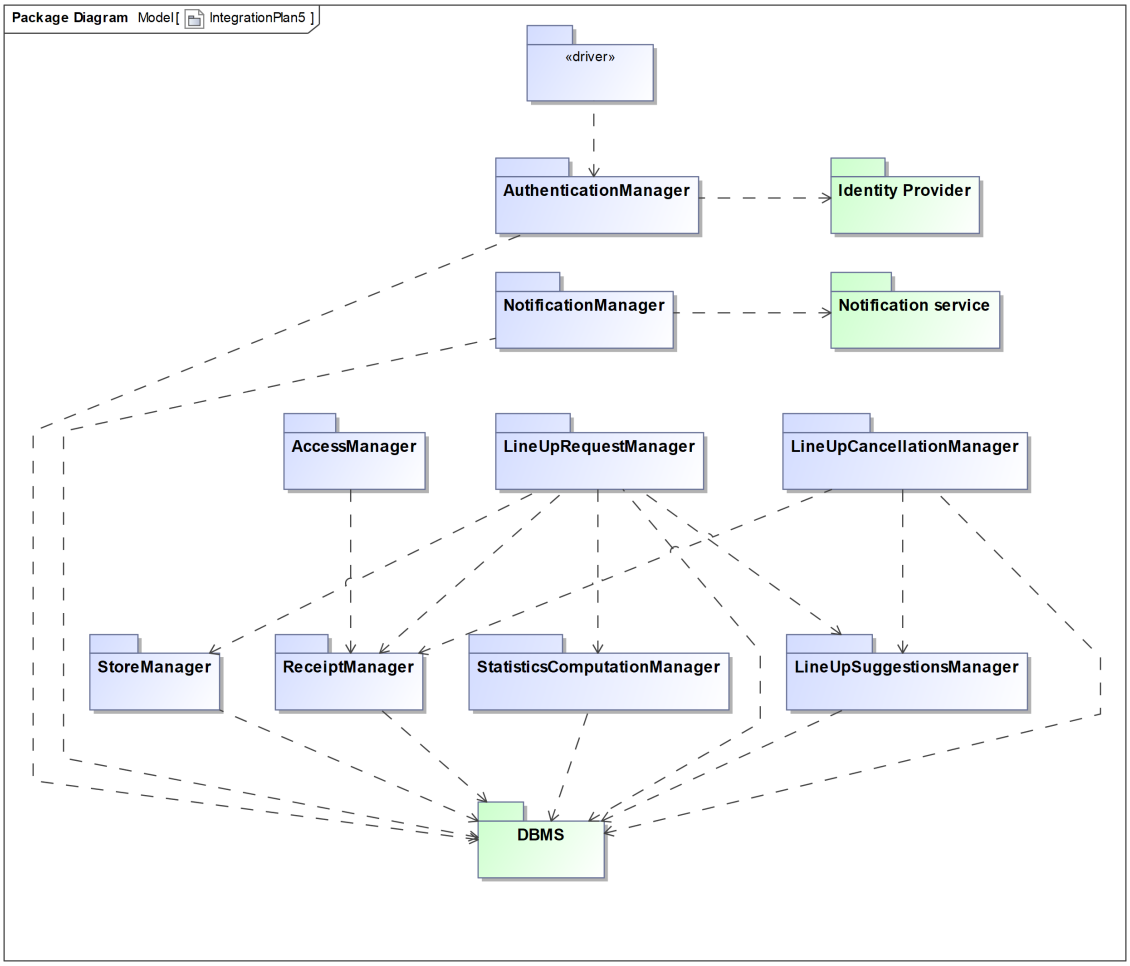
3. Then, AccessManager, LineUpCancellationManager, and LineUpRequestManager are integrated and tested.



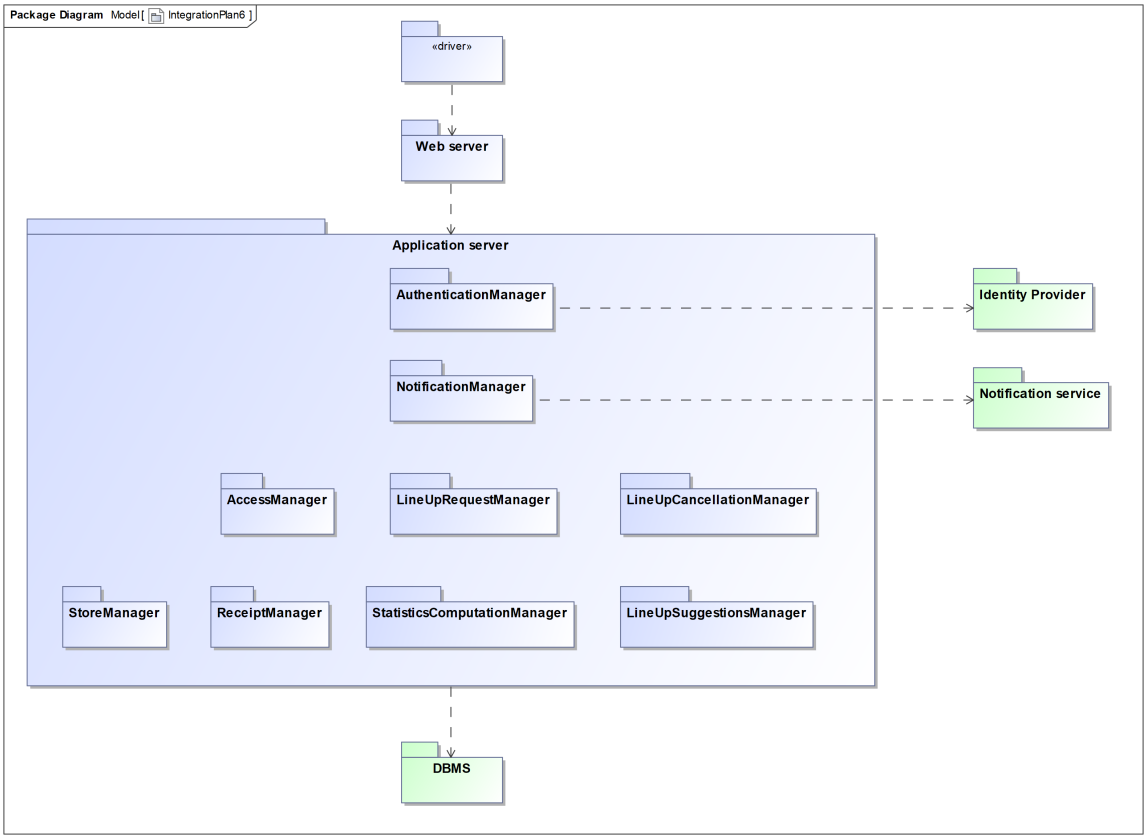
4. Later, the implementation of NotificationManager and its integration with Notification Service can begin.



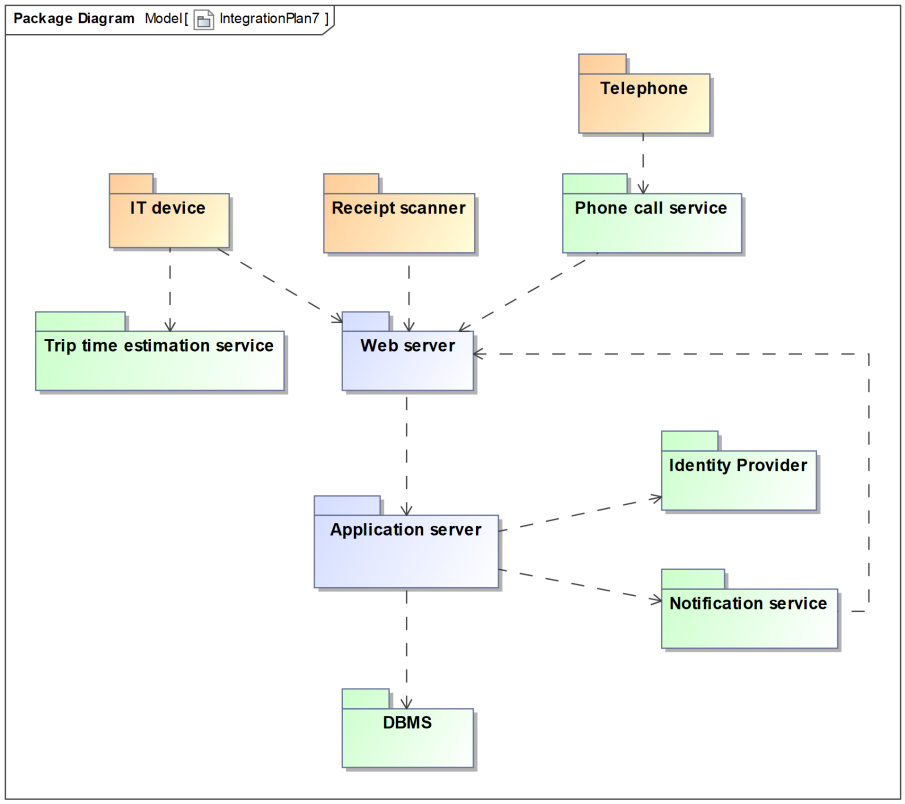
5. Later, the implementation of AuthenticationManager and its integration with the Identity Provider can begin.



6. Then, after the deployment of the Web Server component, its integration in the whole system is tested.



7. Finally, the Phone Call Serice and the TripTimeEstimation Service are integrated into the system. IT devices, telephones, and receipt scanners are used to interact with the completed system.



5.4 System Testing Plan

Once the system is completed, it must be tested as a whole to verify that functional and non-functional requirements, as well as software system attributes and performance requirements, are satisfied.
This phase of CLup's system testing can be divided in:

- **Load testing**: this test allows for understanding how a system behaves under an expected load. In this context, we expect that the system works without any performance issue until the system reaches its maximum capacity. In particular, the goal of this test is to check whether CLup is able to maintain responsiveness even under the stated conditions of maximum capacity [see RASD 3.3].
- **Stress testing**: these tests help to understand the upper limits of a system's capacity simulating a load beyond the expected maximum. The goal of this test is to check whether the upper limits of CLup's system are over the maximum capacity stated in RASD 3.3.
- **Performance testing**: this test is performed to determine how a system performs in terms of responsiveness and stability under a particular workload. The goal of performance testing in our context is to validate and verify the quality attributes of the system, such as scalability, reliability and resource usage. This test reveals itself to be important also for estimating cloud platform costs, because they are directly related to resource usage. Therefore, we can estimate and even predict the expenses related to cloud platforms, based on the data gathered from this testing phase.

At runtime, it is useful to have monitoring implemented, to constantly check if the system is behaving as expected. Monitoring should take into account both infrastructure-scoped statistics and application-scoped statistics. Infrastructure-scoped statistics may leverage the data reported by the cloud platforms in use, since CLup relies on them for hosting its services.

Before releasing any feature or change to the production environment, all updates are deployed in a staging environment which is configured as similar as possible to the production environment. After the core functionalities of CLup and the ones that have been affected by the update are verified to work properly, they are propagated to the production environment.

5.5 Additional Specifications on Testing

During the whole implementation process we can rely on version control systems such as Git and on continuous integration (CI) pipelines. In this way, all developers' working copies can be merged to a shared mainline several times a day. The CI runs the tests defined by the developers and checks that they all pass before allowing the version control system to accept changes to the code base. The developers have the possibility to run the unit tests also in their local environment before committing to the main repository. This helps avoid one developer's work-in-progress breaking another developer's copy.

This continuous application of quality control aims to improve the quality of software, and to reduce the time taken to deliver it, by replacing the traditional practice of applying quality control after completing all development.

6. Effort spent

6.1 Andrea Riva

Date	Effort spent (h)	Notes
7/12/2020	1.5	Introduction briefing
7/12/2020	1.0	Document setup
7/12/2020	1.5	Introduction
18/12/2020	2.0	User interface design
19/12/2020	1.0	Touchbase meeting
19/12/2020	1.5	Mappings between components and requirements
21/12/2020	2.0	Runtime view
23/12/2020	1.0	Alignment meeting
24/12/2020	1.0	Improvements to runtime view
26/12/2020	1.5	Component interfaces
27/12/2020	3.0	Alignment meeting
27/12/2020	0.5	Refactoring and general improvements

6.2 Alessandro Sanvito

Date	Effort spent (h)	Notes
6/12/2020	1.5	Introduction briefing
7/12/2020	0.5	Harmonization
10/12/2020	2.0	Overview
15/12/2020	1.0	Overview completion
18/12/2020	3.0	Deployment view
18/12/2020	2.0	Design decisions
19/12/2020	1.0	Briefing
19/12/2020	1.0	Harmonization
21/12/2020	4.0	Component view
21/12/2020	1.0	Component view
23/12/2020	1.0	Alignment meeting
24/12/2020	1.0	Improvements to deployment and component view
27/12/2020	3.0	DD finalization

6.3 Luca Vecchio

Date	Effort spent (h)	Notes
6/12/2020	1.5	Introduction briefing
13/12/2020	1.5	PR review
19/12/2020	1.0	Harmonisation Meeting
19/12/2020	4.0	Runtime View
20/12/2020	2.0	Runtime View
22/12/2020	1.0	PR review
23/12/2020	1.5	Organisation Meeting
26/12/2020	1.5	PR review
26/12/2020	3.0	Implementation Integration Testing
27/12/2020	2.0	Implementation Integration Testing
27/12/2020	3.0	DD finalisation