Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering

Software Engineering 2 Mandatory Project

Safe Streets.
Design Document

Authors:
Rosetti Nicola
Sartoni Simone
Torri Vittorio

Academic Year 2019/2020

Milano, 09/12/2019
Version 1.0

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions,Acronyms,Abbreviations

1.4 Revision History

1.5 Reference Documents

1.6 Document Structure

2 Architectural Design

2.1 Overview: High-level components and their interaction

The application will be developed using the client-server paradigm on a three-tiered architecture. The three layers of the application (Presentation, Application and Data) are divided into clusters of machines (i.e. tiers) that actually cooperate to provide a specific functionality. In this case we have three tiers and each tier is responsible for one of the three layers. The client side is responsible (only) for the presentation layer; therefore, in this architecture, the thin-client has been adopted considering the fact that the required functionalities client-side are limited. In the App case, the client contains all the presentation layer while in the WebApp case the layer is splitted between the client and WebServer; the WebServer is responsible for contacting the application server and forward the client requests to it. The Application tier takes care of the application layer encapsulating all is needed concerning the application logic. It receives the requests from the clients and handles them. It's also responsible for sending asynchronous notifications to the presentation layer when certain conditions are met. Here we have multiple Servers cooperating together to improve performance, scalability, fault tolerance and availability. An elastic component (i.e. load balancer) is used to rule the accesses to different Application Servers, dinamically balancing the load among all the Servers. The Application tier communicates with the Data tier, responsible for the Data Access layer. This tier is composed by several DataServers: each one is associated with a single replica of the data and exploits the DBMS technology to access the DataBase. The Database is fully replicated in different nodes. Techniques and protocols are used to ensure consistency among replicas. Again a load balancer is used to dinamically share the load among different machines. To provide the required functionalities the system exploits datawarehousing. The datawarehouse is a component in the Data tier able to deal with historical data and aggregate data taken from the Databases exploiting data mining technologies to answer complex queries: used techniques are clustering, associative rules and classification. This component periodically queries one dataServer to retrieve new information and updates on the data since its last update.

2.2 Component view

2.3 Deployment view

2.4 Runtime view

2.5 Component interfaces

2.6 Selected architectural styles and patterns

2.7 Other design decisions

2.7.1 Consistency and update strategies among replicas

In this application, we need a client-centric consistency among replicas because end users (and therefore application servers that act as clients towards the DataServers) don't always connect to the same DataServer, due to the presence of the load balancer. Every DataServer can respond to a request to read or write data so we use an active replication protocol. In particular we exploit leaderless replication in which the decision on the value to read and the write to perform is decided by all the replicas or at least a quorum of them. The type of chosen consistency model is the "read your writes": the effect of a write operation by a process on a data item x will always be seen by a successive read operation on x by the same process. Concerning the update propagation we opted for propagating a notification of the operation, assuming that there will be more writes than reads. The propagation strategy chosen is the Gossiping strategy: when a replica is updated then it just propagates that update to all the nodes that it knows; if a replica receives an update that it has already received then the probability of propagating that information is decreased on that replica.

3 User interface design

4 Requirements Traceability

5 Implementation, integration and test plan

6 Effort spent

Nicola Rosetti

Date	*Hour*	*Section*
17-10-2019	1.5 h*	Goals

Table 6.1

Simone Sartoni

Date	*Hour*	*Section*
17-10-2019	1.5 h*	Goals

Table 6.2

Vittorio Torri

Date	*Hour*	*Section*
18-11-2019	1 h*	Goals

Table 6.3

** Group work*

7 References