POLITECNICO
MILANO 1863

Software Engineering II

CLup - Customers Line-up

Design Document

Authors:
Cosimo Sguanci
Roberto Spatafora
Andrea Vergani

PAGINE DI COSIMO

PAGINE DI ANDREA

2.F SELECTED ARCHITECTURAL STYLES AND PATTERNS

Client-server architecture, HTTPS and REST

The choice for a client-server architecture is quite natural for *CLup*: in fact, the application is distributed. Since the model is quite straightforward, no further details are needed to specify.
The communication protocol for all messages between client and server is HTTPS: it guarantees a high level of security, confidentiality and data integrity.
HTTPS is enriched with the usage of REST architectural style, which provides a stateless protocol ensuring reliability, reusability and scalability.

Three-tier architecture

The client-server architecture follows a three-tier pattern: in particular, we can distinguish a Presentation tier, a Logic/Business tier and a Data tier.
The division into three layers allows to separate tasks and increases the level of reusability and decoupling; in addition, the whole architecture results to be more flexible and maintainable (a single tier can be internally modified, fixed, … without consequences on other tiers).
More specifically, the tier division is the following:

- Presentation tier is client application's business (mobile application, call center web application); it allows the interaction between user and *CLup*
- Logic/Business tier is present both on the client and server sides: servers' logic layers control application functionalities; client logic is related to tracking mobile app user position, in order to send specific notifications *
- Data tier includes the database and all mechanisms for storing data

** client logic is better described in section named "Thick client" (which immediately follows)*

Thick client

With reference to the mobile application, a thick client configuration is adopted: this implies that, in addition to the presentation layer, the mobile app client also incorporates a part of the business layer.
More in detail, *CLup* specification highlights the need to calculate the user's distance to the grocery shop, in order to send proper notifications for arriving on time and not missing the reservation; furthermore, the application's RASD describes how the customer can insert, in optional fields, his starting position and transportation means, so that the system can more easily deal with its computations (server-side, in general). However, the user may not add this additional information: to solve the situation, the decision is that of making the client itself responsible for sending proper notifications.

The main cause behind this design decision is that server, not having an estimate about when to start checking user's position (because the optional fields were not filled), would have to continuously track him (or something similar, at least for a time period): however, this is costly and would require special permissions for privacy reasons. For this reason, the logic regarding position monitoring for notifications (in case no additional information is provided by the customer) is left to the client, thus leading to a thick client design.

Model-View-Controller

Integrated with the three-tier architecture, Model-View-Controller design pattern guarantees a clear separation (but also interaction) between the different layers of the application.
In detail:

- Model lies on the server, in particular on the business logic component and the database
- View lies on the client and corresponds to the Presentation layer of the three-tier architecture
- Controller is both client-side and server-side: on client some simple controls about inputs are performed, together with the more consistent part regarding the check of user's position (as described in "Thick client" section, which immediately precedes the current one); on server all simple controls are replicated (for robustness), with the addition of all the necessary for the correct system functioning

Model-View-Controller pattern particularly fits the development of a mobile application and Web application.
Together with MVC, the usage of an Observer design pattern guarantees a flexible interaction between layers.

Relational DBMS and SQL

The DBMS managing data in the database is built upon the relational model. The choice of a relational DBMS is widely used and there is no particular reason for deviating from it in *CLup* application; all commercial DMBSs guarantee ACID properties, which in turn ensure good features (efficiency, concurrency, …).
SQL is used for querying and managing data; the choice for SQL follows the standard for relational DBMSs (so there is no need to further comment on it).

2.G OTHER DESIGN DECISIONS

Replication

CLup must guarantee at least 99% of availability, as discussed in the *Requirements Analysis and Specification Document*. In order for this feature to be fulfilled, server business logic and database are replicated twice: in case of problems to one of the units, the others can still serve clients and provide all the application services.

The choice for “two nines” (99%) availability means that the average downtime period must not be greater than 3.65 days in a year. Supposing that a grocery store is open for 12 hours a day (every day of the year), which is a quite reasonable assumption, the downtime that really influences the possibility to manage entrances affects 1.825 days.

This line of reasoning follows the fact that downtimes outside supermarket’s opening hours are not a big issue: customers do not have the possibility to book or manage reservations, but in reality this is not a very big problem. The critical part, instead, is about server unavailability during opening hours: entrance management becomes impossible and every grocery shop should put some “manual” solutions into practice, thus causing crowds to form outside the shops themselves. Since this kind of situation happens less than 2 days in a year (on average) and the nature of the application is not extremely critical (it does not concern emergency situations, since keeping distance in a line for one day can be done, hopefully without serious consequences), 99% of availability has been chosen.

Proxy *descriverne l’utilizzo e motivare la scelta*

Firewall *descriverne l’utilizzo e motivare la scelta (sicurezza)*

PAGINE DI ROBERTO