Politecnico di Milano
A.A. 2014-2015
Software Engineering 2: "MeteoCal"
Design **D**ocument

version 2.0

Federico Migliavacca (mat. 836582), Leonardo Orsello (mat. 837176)

25 January 2015

Contents

1 Introduction

1.1 Purpose

This document represents the Design Document (DD). The main goal of this document is to explain the design choice made during the architectural analisys and the functionalities that will be developed.

1.2 Scope

The aim of the DD (Design Document) is to present the main architecture of the MeteoCal application. The strucure will be based on the requirements and assumptions described in the RASD document (Requirements Analysis and Specification Document). This document wants to create a common resource that can be used by all the actors involved in the project.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

User	User registred to MeteoCal application.
Visitor	Visitor who visits MeteoCal application's website but is not registred.
Event	Object direclty created by user.

1.3.2 Acronyms

DBMS	Database Management System.
JEE	Java Enterprise Edition.
API	Application Programming Interface.
ER	Entity-Relational Model.
EJB	Enterprise Java Bean.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
JDBC	Java Database Connectivity.
UML	Unified Modeling Language.
UX	User eXperience.
MVC	Model View Controller.
JPA	Java Persistence API.
XHTML	eXtensible Hypertext Markup Language.

1.3.3 Abbreviations

1.4 Reference Documents

- Specification Document: MeteoCal Project AA 2014-2015.pdf.
- Requirements Analysis and Specification Document (RASD).

- ISO/IEC/ IEEE 42010 Systems and software engineering — Architecture description.
- IEEE Std 1016^{tm}-2009 Standard for Information Tecnology-System Design-Software Design Descriptions.

1.5 Document overview

This document is essentially structured in six parts:

- Section 1: Introduction, it gives a description of document, definitions, acronyms and abbreviations.
- Section 2: General Description, it gives general information about software design focusing on Technological and Performance Requirements.
- Section 3: Software Architecture, this part lists all technological and architectural design choice.
- Section 4: Database Model, this part shows the conceptual and logical design of database, basing the analisys on E-R model.
- Section 5: Client Design, this part gives an idea of navigation models through the sequence diagrams and UX cases.
- Section 6: Appendix, this part contains some information about the used tool.

2 General Description

2.1 Assumption

System interface is reachable web-only to make the service aviable to as many people as possible, and usable in the easiest possible way. To reach this purpose the MeteoCal application offers users intuitive functions to manage their own calendar such as new event creation, modification of existing event, keeping under control weather forecast for event, etc... Anyone who wants to use all MeteoCal functionalities must complete the registration form so that he/she reaches the status of Registered User. User can log in to application through his/her username and password.

2.2 Technological Dependency

- Users must have an updated version of a web browser istalled on his/her machine and have access to internet.
- The user machine must have also installed a Java Virtual Machine that supports JEE7 technologies.
- On the server side an application server that supports JEE7 is required.

2.3 Technological Requirements

RAM	2GB+
Secondary Memory	32GB+
Database Server	MySql
Network	Internet Access, HTTP protocol
Security	Possible HTTPS implementation on the logged in side of application.

2.4 Performance Requirements

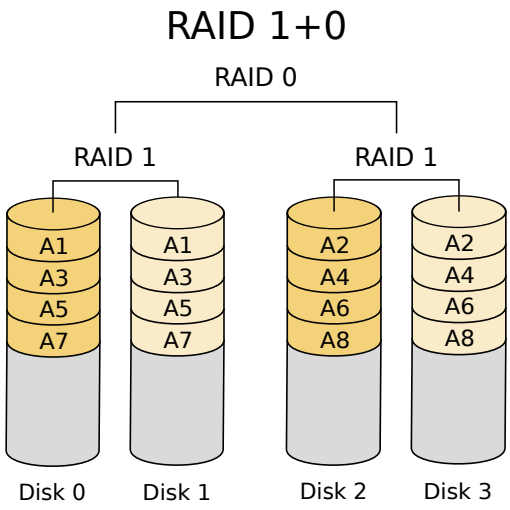
2.4.1 Reliability

To assure the reliability of the software product, it is mandatory to back up the database periodically. Reliability could also be increased using a system composed by a RAID 0 (striping) architecture, specially on database server, but also applied on application server.

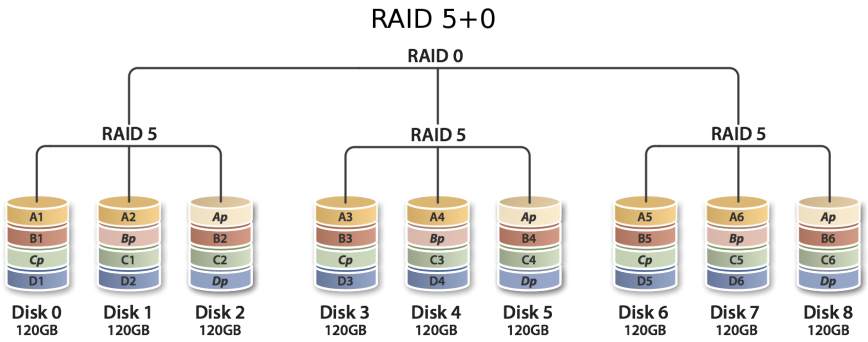
2.4.2 Availability

The application will be accessible online anytime. To achieve this goal, it could be necessary to use a dedicated server but to guarantee more availability, all system could be hosted into cloud platform like Amazon EC2. This solution gives more scalability to performance required by the system and could reduce the cost for dedicated server, mantaining an high level of performance especially

in case of full load with a lot of connected users. In case of a dedicated server choice, it could be necessary to duplicate disks or the entair machine. To keep the system safe against data loss, it could be used RAID 1 (mirroring) technologies. In conclusion the best way to conciliate the requirements of reliability and availability using RAID technologies, is configuration with RAID 1+0 solution (picture1) for database, which creates a striped set from a series of mirrored drives; the array can sustain multiple drive losses so long as no mirror loses all its drives. For the application server, instead, the best RAID solution could be RAID 5+0 (picture2) architecture that combines the straight block-level striping of RAID 0 with the distributed parity of RAID 5; this is a RAID 0 array striped across RAID 5 elements.



[picture1]



[picture2]

2.4.3 Security

Passwords are saved using an hash function. The support of HTTPS protocol could be implemented . Other possible implementations to make the system more safe are well described in 3.6.4 paragraph of RASD document.

2.4.4 Maintainability

The aplication does not provide any specific API, but the whole application code will be documented to well inform future developers about how application works and how it has been developed.

2.4.5 Portability

The software product has been developed using the Java language and related dependent technologies. Java is specifically designed to have as few implementation dependencies as possible. It means that the code which runs on one platform does not need to be recompiled to run on another, of course anyone that has a compatible JVM installed could correctly run MeteoCal application.

3 Software Architecture

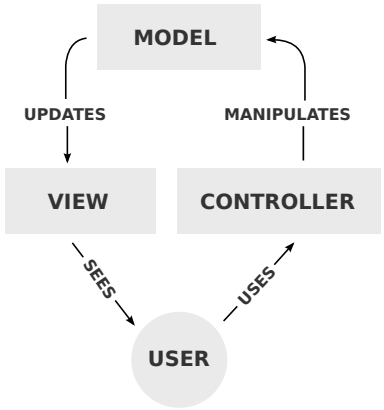
3.1 Technologies Choice

The following are the tecnologies choosen for MeteoCal application. Decisions were made in respect of JEE platform constrains.

- The application is based on JEE 7 platform. In particular EJB 3.2 are used for the application logic development and JSF 2.2.5 for the web application development.
- Glassfish 4.1 is used as application server.
- MySQL 5.6.21 is used as DBMS (any other DBMS compatible with the application server could have been used since the comunications with database are handle by JPA).
- A Goolge code repository has been created to help team cooperation (https://code.google.com/p/meteocal-orsello-migliavacca)

3.2 Architectural Design

We use a multi-tier architecture divided in 4 logic level: client layer, web layer, business layer and persistence layer. This 4 logic level are mapped on 2 tier level creating a tipical client-server architecture. The 4 layer are described in the follow. The MeteoCal architecture is designed thinking to Model-View-Controller (MVC) pattern:



- A *controller* can send commands to the model to update the model's state. It can also send commands to its associated view to change the view's presentation of the model.
- A *model* notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands.

- A *view* requests information from the model that it uses to generate an output representation to the user.

3.2.1 Thin Client Layer

This is the logic level that users can access through any web browser. It exchange messages with the web tier using http protocol, in particular it sends users imput data and receives xhtml file from the server, used for the presentation of web pages. Client tier could also run javascript code, used to implement user imputs inspection system necessary for reducing traffic between client and server.

3.2.2 Web Layer

This logic level receives client requests and depending on the kind of request (get,post etc..) interacts with the business tier through JSF tecnology, then creates a proper web page and sends it to the requesting client.

3.2.3 Business Layer

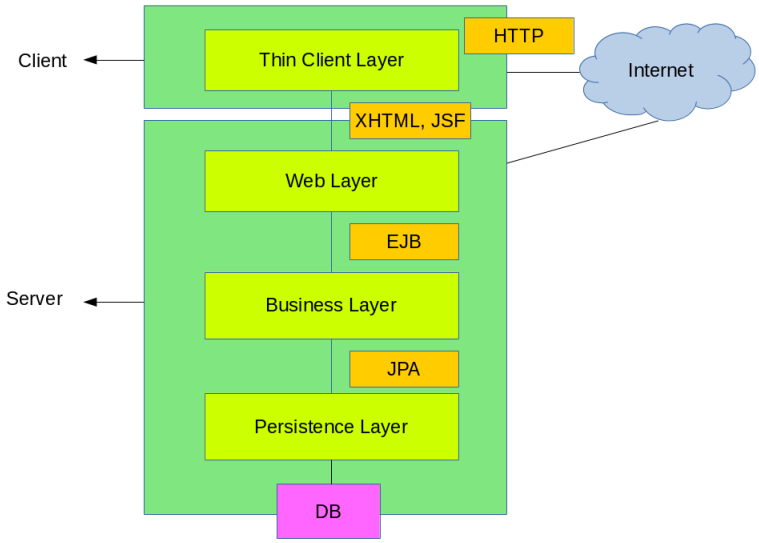
This logic level includes all the application logic of the system and it direclty interacts with database. This layer is realized using SessionBean, used to implement business logic into architecture. Session Beans are business objects having a global state shared within a JVM. Concurrent access to the one and only bean instance can be controlled by the container (Container-managed concurrency, CMC) or by the bean itself (Bean-managed concurrency, BMC). At the end this level replies to the web layer request and uses the persitence level to reach data information.

3.2.4 Persistence Layer

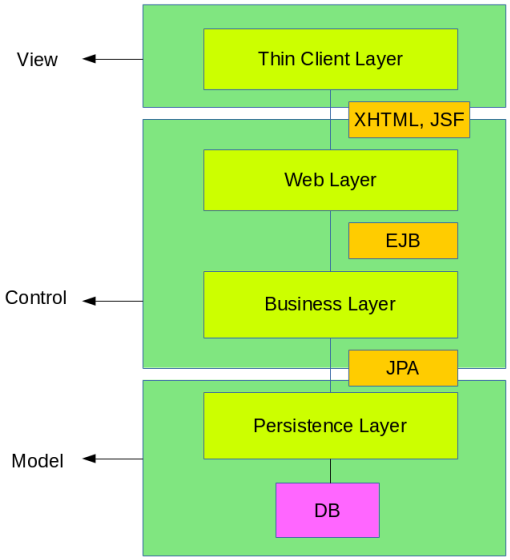
This logic level is the DBMS which implements the data persistence and makes them available to the application server. Comunication is handled with JDBC driver. The purpose of your application's persistence layer is to use a session at run time to associate mapping metadata and a data source in order to create, read, update, and delete persistent objects using queries and expressions, as well as transactions.

3.3 Architectural Description

Here there is a schema about the architecture previously described.



This second schema represents the architecture following the MVC pattern.



4 Database Model

4.1 Conceptual Design

4.1.1 Entity Analisys

The entities are the following:

- **RegistredUser :**
  this entity represents a registred user, namely a visitor that has succesfully completed the registration process and can access to MeteoCal application through the log in page. He/she is represented by an unique ID code and has records (name, surname, birthdate, address) and credentials (Email, Groupname, Password) as attributes.

- **Event :**
  this entity represents an event that has been created by a user, who becomes the owner. All the events are identified by an unique ID code. Public_bool and Indoor_bool are attributs utilized by the application to recognize the event settings and make decision, respectively to know if the event is public or private therefore to apply the proper visibility property, and to know if the event will be indoor and outdoor. Title, EndeventDate, StarteventDate, Location, Descriprion, AllDay, WeatherforecastID and Owner are its other attributes.

- **Calendar :**
  this is a week entity depending on User, from which takes the primary key. Public_bool is an attribute utilized by the application to recognize if the calendar has been set public or private, therefore to apply the right visibility property.

- **WeatherForecast :**
  this entity contains meteo informations represented by description, descriptionId, Icon and Temperature attributes. Any WeatherForecast is associated with an unique ID code.

- **NotifyLink :**
  this entity represents one of the two kind of notice, in particular it represents any message that needs a link (for example the notice that suggests to modify the date of the event due to bad weather will contain the link to the "modify event" form). Its attribute are Message, SuggestedDate, UserEmail, EventId and one unique ID code.

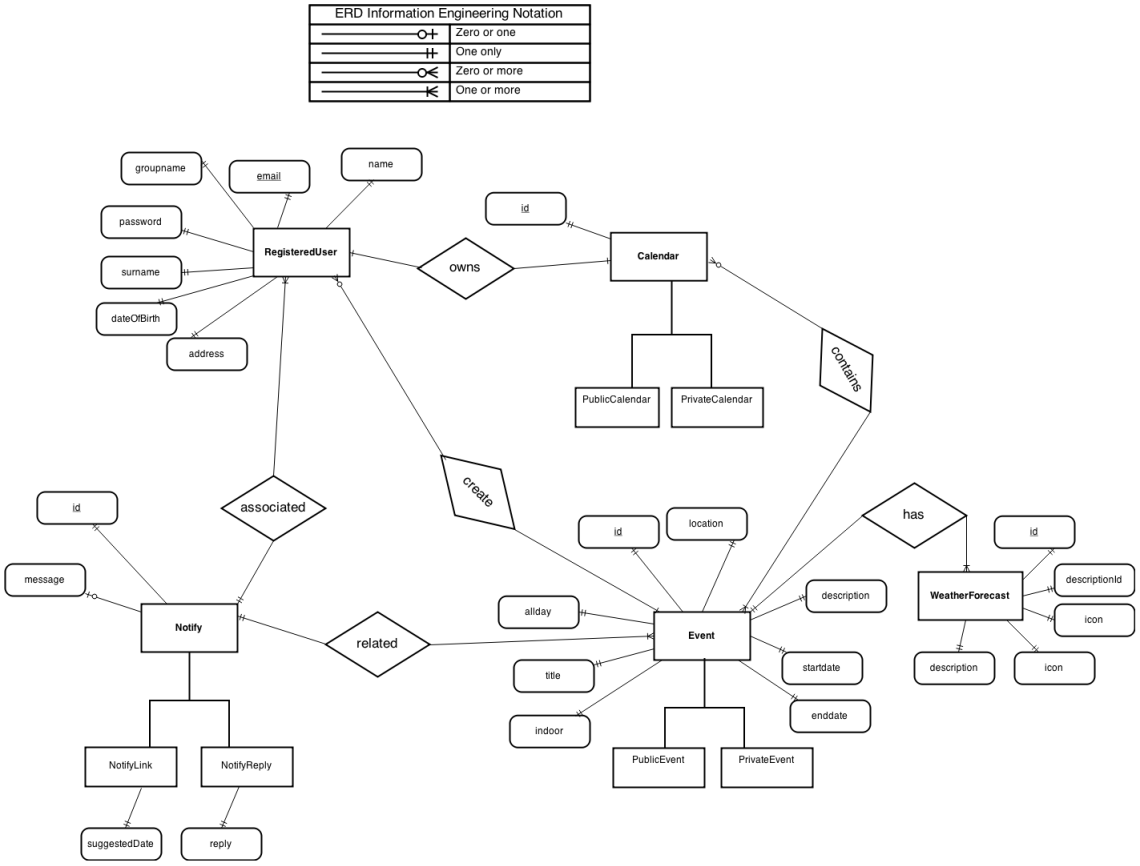
- **NotifyReply :**
  this entity represents one of the two kind of notice, in particular it represents any message that provides a reply choices (for example the notice that asks the user to accept or not an invitation). Its attribute are Message, Reply, EventId, UserEmail and one unique ID code.

4.1.2 Relational Analisys

The entities relations are the following:

- **Owns:**
  this is a relation between the entities RegistredUser and Calendar (one only/one only). All the users own only one Calendar.
- **Contains:**
  this is a relation between the entities Calendar and Event (zero or more/one or more). Through the calendar users can see all the events visible to them.
- **Create:**
  this is a relation between the entities RegistredUser and Event (zero or more/one only). Users can create many events as they want.
- **Associated:**
  this is a relation between the entities RegistredUser and Notify (zero or more/one only). Notice are associated to the user whose they will be displayed.
- **Has:**
  this is a relation between the entities Event and WeatherForecast (one only/one or more). Events have weatherforecast associated to them to check if the weather is bad or not.
- **Related:**
  this is a relation between the entities Event and Notify (one only/one or more). Events could have one or more related notify entities.

4.1.3 Entity-Relational Diagram



4.2 Logical Design

4.2.1 Logical Model

RegistredUser(Email, Groupname, Password, Name, Surname, Birthdate, Address)

Event(EventID, Title, EndeventDate, StarteventDate, Location, Descriprion, WeatherforecastID, Owner, Public_bool, Indoor_bool, AllDay)

Calendar(Owner, Public_bool)

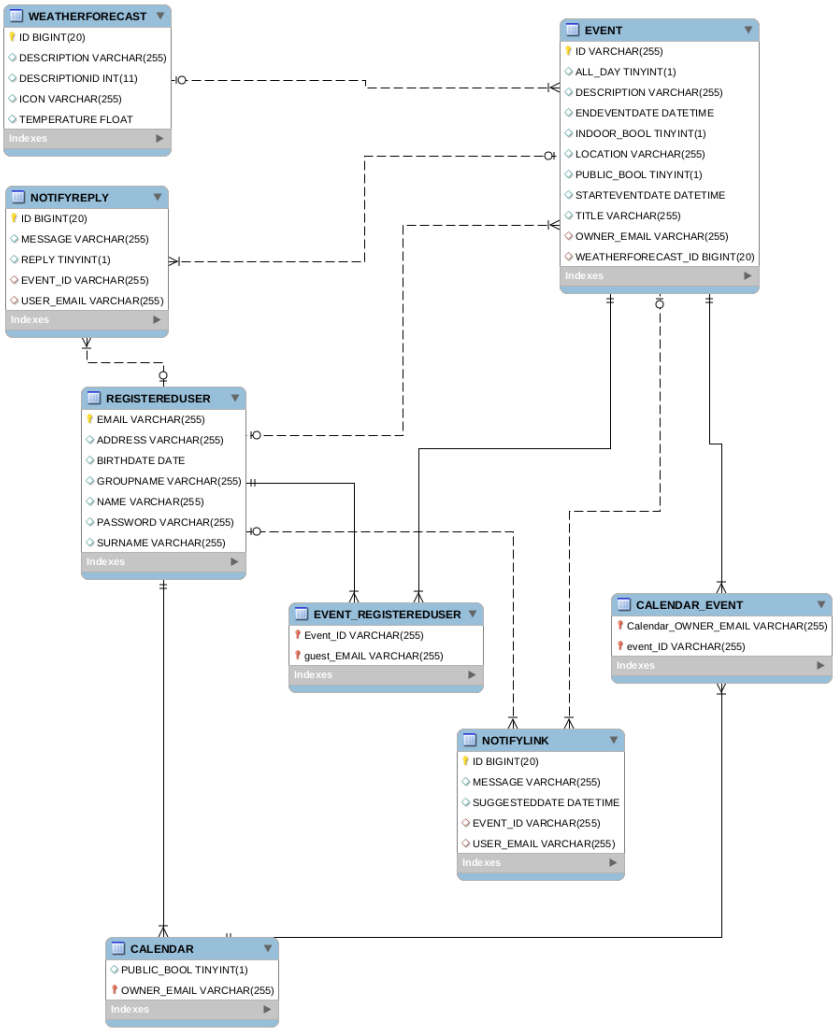
WeatherForecast(WeatherForecastID, description, descriptionId, Icon, Temperature)

NotifyLink(NotifyLnkID, Message, SuggestedDate, EventId, UserEmail)

NotifyReply(NotifyboolID, Message, Reply, EventId, UserEmail)

Event_RegisteredUser(GuestEmail, EventID)

Calendar_Event(EventId, CalendarOwnerEmail)



4.2.2 Foreign Key Constraints

Calendar.Owner → RegistredUser.Email
Event.Owner → RegistredUser.Email
Event.WeatherForecast → WeatherForecast.WeatherForecastID
NotifyReply.Email → RegistredUser.Email
NotifyLink.Email → RegistredUser.Email
NotifyLink.Event_ID → Event.ID
NotifyReply.Event_ID → Event.ID
Event_RegisteredUser.Event_ID → Event.ID
Calendar_Event.CalendarOwnerEmail → RegistredUser.Email
Event_RegisteredUser.GuestEmail → RegistredUser.Email

Calendar_Event.EventID → Event.ID

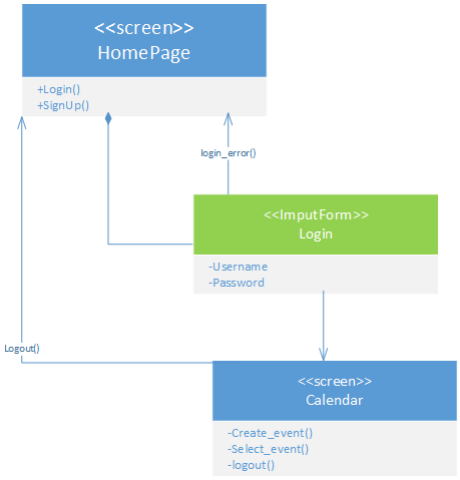
4.2.3 Database Constraints

Email choosen by users must be different for each user. Email must be unique.

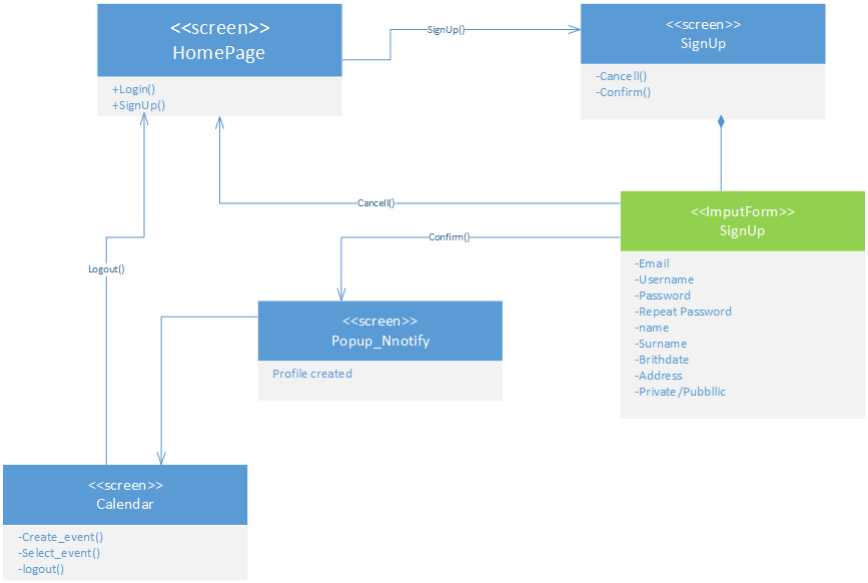
5 Client Design

5.1 Navigation Models

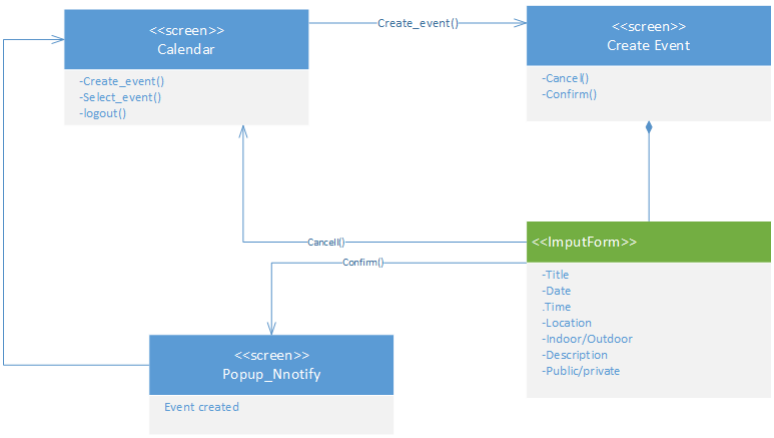
5.1.1 Login UX



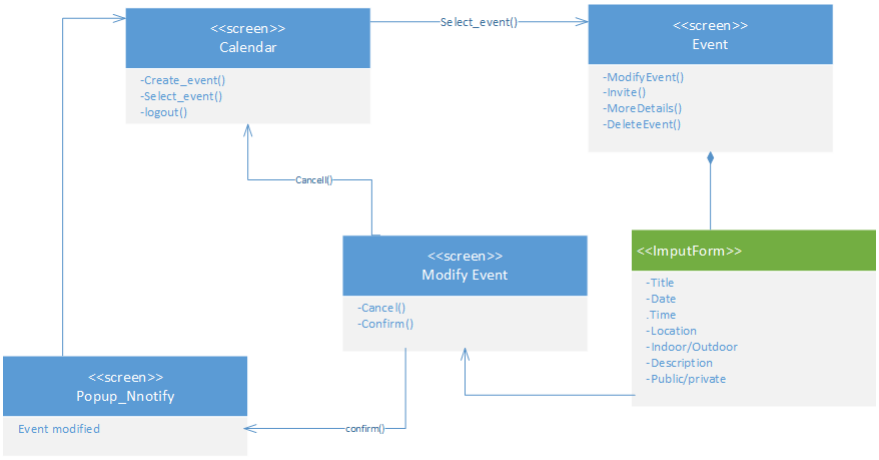
5.1.2 Registration UX



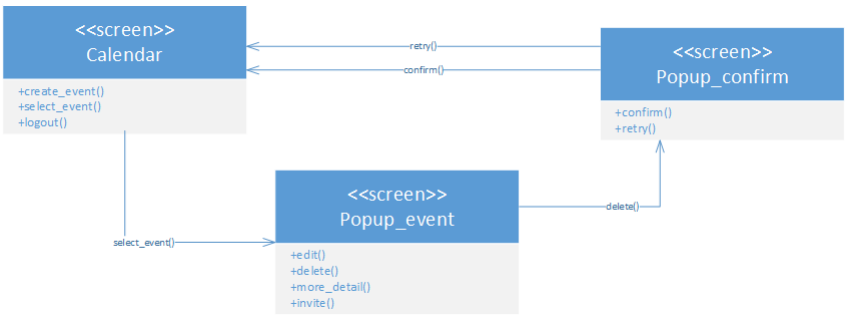
5.1.3 Create Event UX



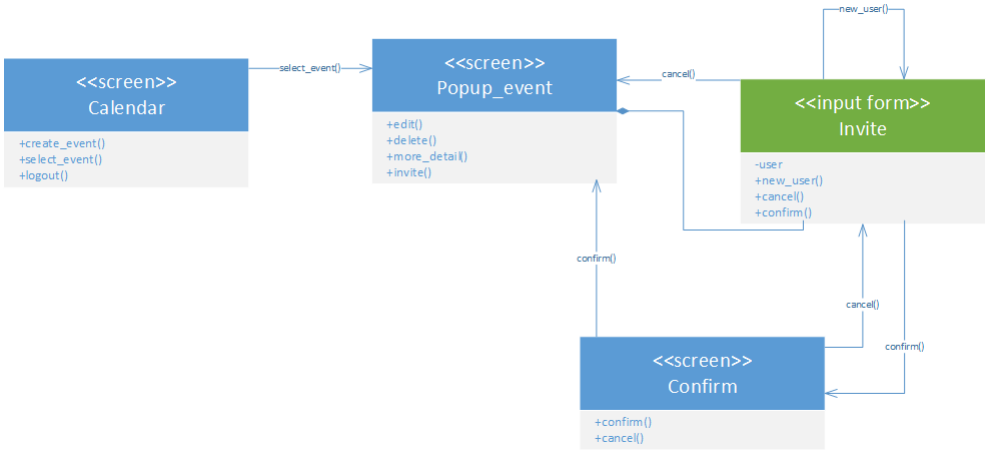
5.1.4 Modify Event UX



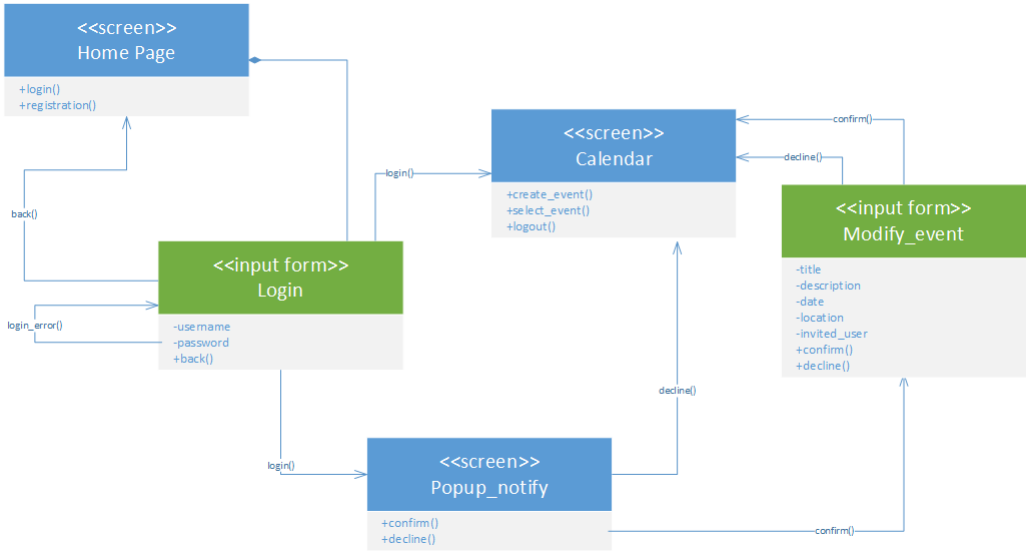
5.1.5 Delete Event UX



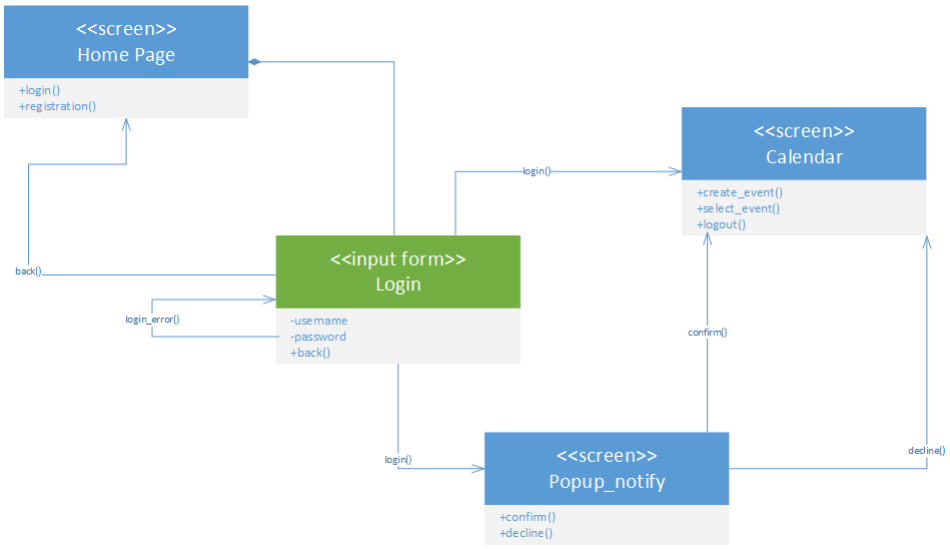
5.1.6 Invite Event UX



5.1.7 Notify Change Date UX



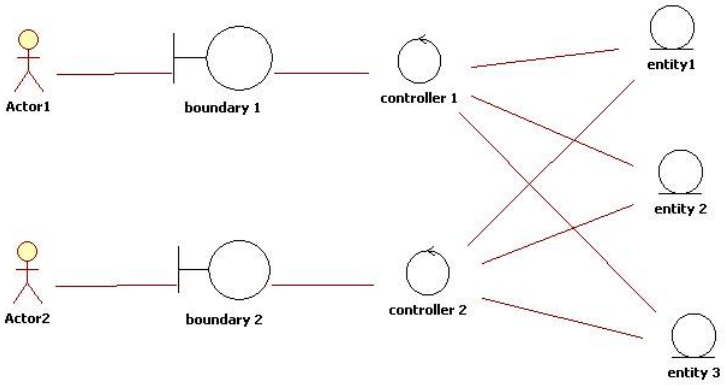
5.1.8 Notify Invite UX



5.2 Design Analysis

5.2.1 Boundary-Control-Entity Model

We say that the MeteoCal architecture is designed thinking of Model-View-Controller (MVC) pattern but it was originally developed for desktop computing. MVC is widely adopted as an architecture for World Wide Web applications with its specific variant: Boundary-Control-Entity pattern, that will be described in the follow also with a simple UML diagram.



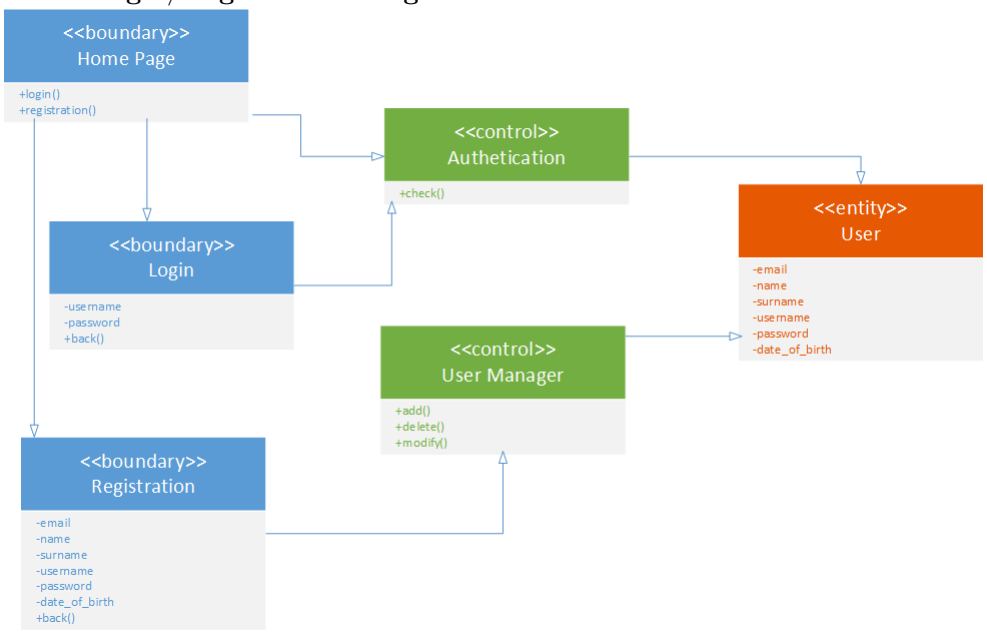
- **Boundary Elements:** A boundary element lies on the periphery of a system or subsystem, but within it. For any scenario being considered either across the whole system or within some subsystems, some boundary elements will be "front end" elements that accept input from outside of the area under design, and other elements will be "back end", managing communication to support elements outside of the system or subsystem. If the interfaces with the system are simple and well-defined, as in our case, a single package may be sufficient to represent the external system.

- **Control Elements:** A control element manages the flow of interaction of the scenario. A control element could manage the end-to-end behavior of a scenario or it could manage the interactions between a subset of the elements. Behavior and business rules relating to the information relevant to the scenario should be assigned to the entities; the control elements are responsible only for the flow of the scenario.

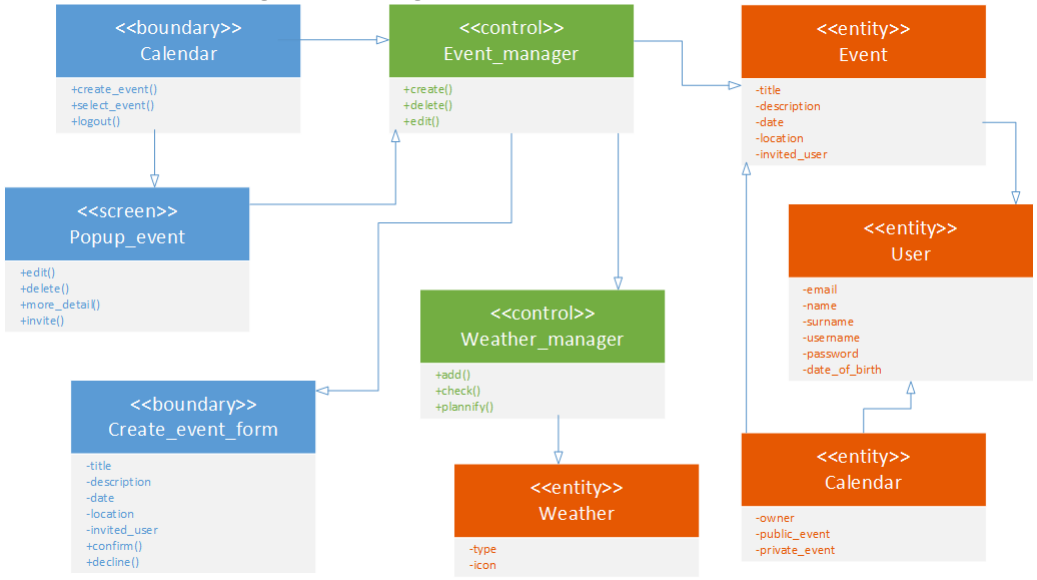
- **Entity Elements:** An entity is a long-lived, passive element that is responsible for some meaningful chunk of information. This is not to say that entities are "data", while other design elements are "function". Entities perform behavior organized around some cohesive amount of data. In this case Entity includes "data" and proper function to use it. It's important to remember that control elements can communicate with boundary and entity, but entities and boundary elements should not communicate directly.

5.2.2 BCE Diagrams

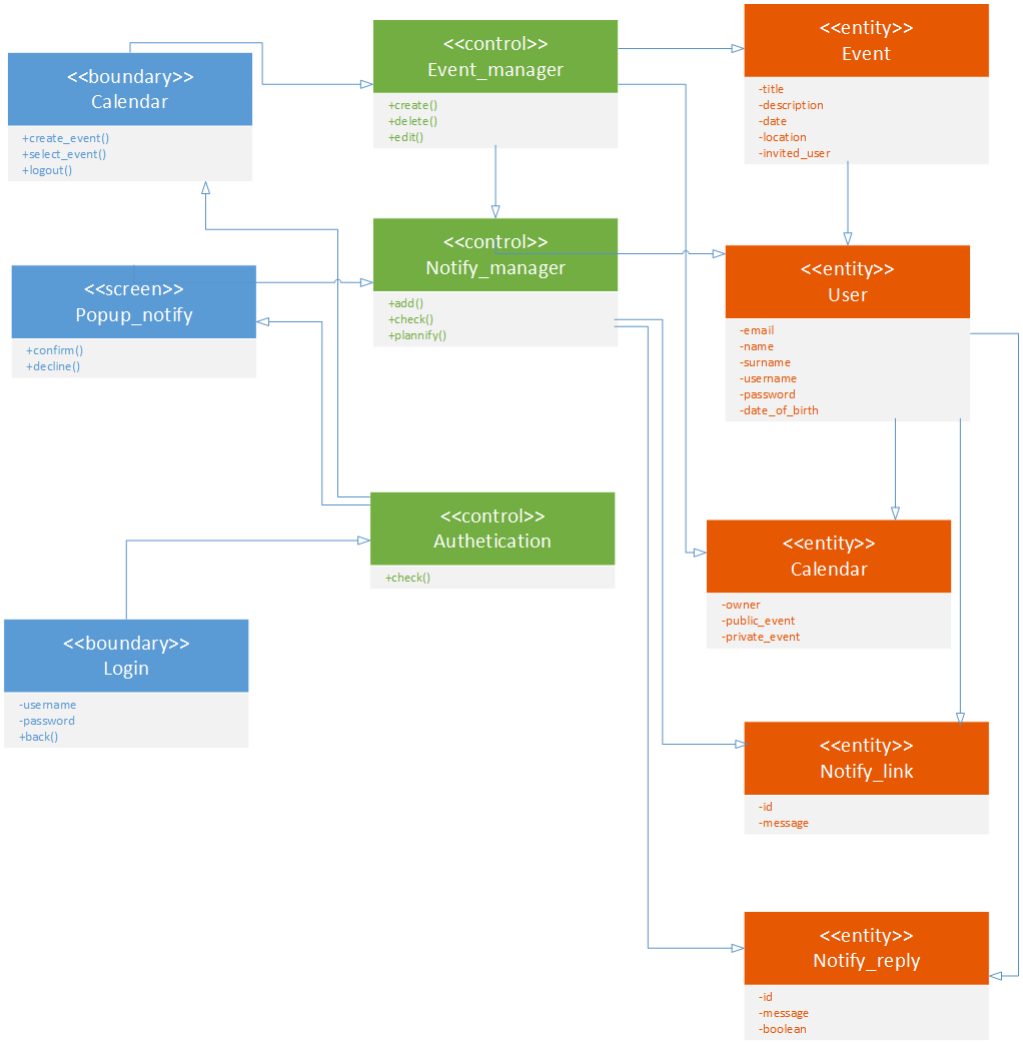
5.2.2.1 Login/Registration Diagram



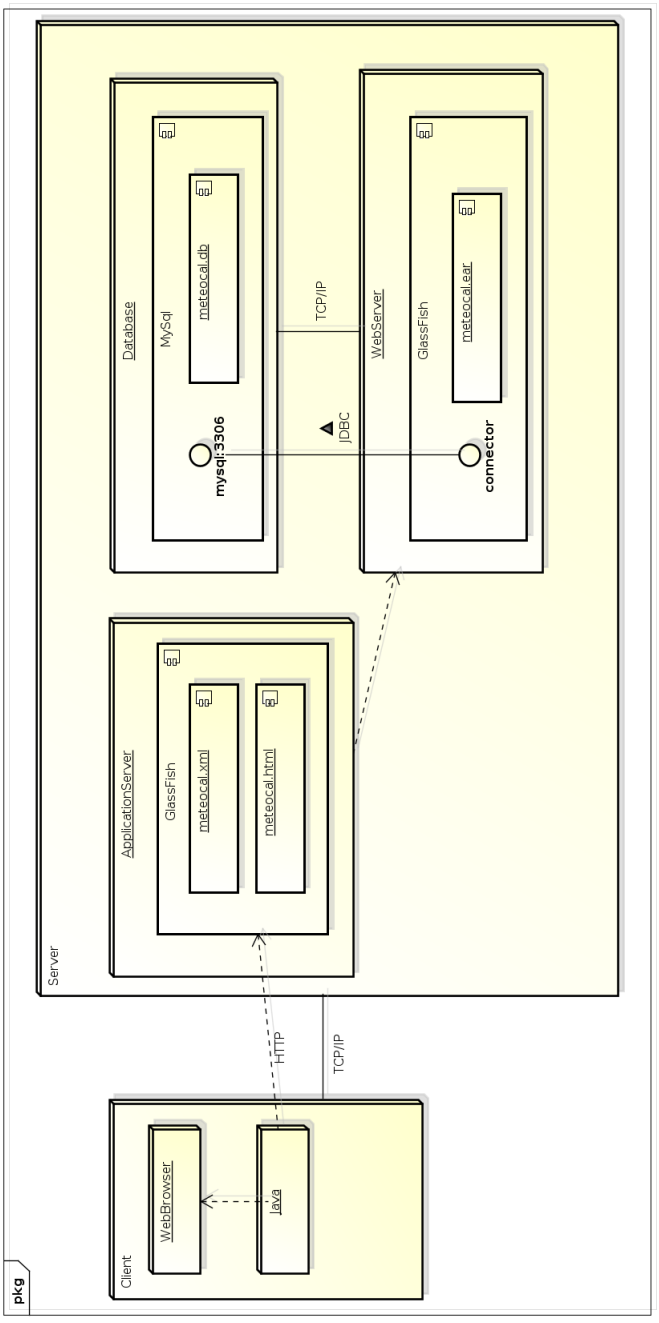
5.2.2.2 Event Management Diagram



5.2.2.3 Notify Diagram



5.3 Deployment Diagrams



6 Appendix

6.1 Software and tool used

- Lyx (http://www.lyx.org/): to redact and to format this document.
- Astah Professional (http://astah.net/editions/professional): to create Diagrams.
- Microsoft Visio Professional 2013(http://office.microsoft.com/it-it/visio/): to create Diagrams.
- https://www.draw.io/: to create Diagrams.
- Libre Office Impress (https://it.libreoffice.org/caratteristiche/impress/): to create some images.
- Gimp(http://www.gimp.org/): to model some images.

6.2 Hours of works

This is the time spent to redact this document:

- Federico Migliavacca: ~27 hours.
- Leonardo Orsello: ~27 hours.

7 Revision

This is 2.0 version of DD that contains update of the document after the entair development of the application. In the follow are listed the difference between the previous version:

7.1 Database Model: Conceptual Design and Logical Design

The entity has been updated with the actually model of the DB. There are only little difference to the attribute. For the "Relational Analisys part" the ViewEv and ViewCal relation has been deleted. The Entity-Relational Diagram after the update reflect the modification previously described. Also the logical design have been updated following the new database model, as result the Logical Schema diagram and the Foreign Key Constraints has been updated.