



POLITECNICO DI MILANO

SOFTWARE ENGINEERING 2 PROJECT

---

# MeteoCal

## Design Document

---

*Authors:*  
Andrea CELLI  
Stefano CEREDA

November 28, 2014

## Part I

# Introduction

## 1 Purpose of this document

This document describes the general and specific architecture of MeteoCal, the project of the course of Software Engineering 2 at Politecnico di Milano. The document will explain the architectural decisions and trade offs chosen in the design process and its justifications.

## 2 Scope

The architectural descriptions provided concern the functional view, module view, deployment view, data layer, business logic and the user interface of the RASD. Hence the architecture will consider the following functionalities offered by MeteoCal:

- *Users*: MeteoCal will manage personal data of the users. MeteoCal will manage registering, logging in/out and the modification of personal data.
- *Calendars*: MeteoCal will manage a calendar for each user. User will be able to create, update and delete an event and to see other people's events. MeteoCal will also manage event invitation and notifications for the event's update.
- *Weather*: MeteoCal will manage weather forecasts and send notifications to event's participants one day in advance in case of bad weather. It will also have to propose an alternative schedule to the event creator with three day of advance.

## 3 Definitions and acronyms

### 3.1 Definitions

- *Calendar*: a calendar is the agenda of an user
- *Event*: a task that a user has into his calendar
- *Registered user*: a user that has created an account on MeteoCal
- *Logged user*: a registered user that has performed the login process
- *Unlogged user*: either a non registered user or a registered user that is logged out of the system
- *Participant*: a participant to an event is either its creator or an invited user who accepted the invite
- *Bad weather alert*: the notification send to the user with one day of advance if weather forecasts for outdoor events on the next day are bad

- *Date changed notification*: the notification send to every participant if the event creator change the event date
- *System*: the MeteoCal system

### 3.2 Acronyms and abbreviations

- *MeteoCal*: Meteorological Calendar
- *G*: Goal
- *JVM*: Java Virtual Machine
- *JEE*: Java Enterprise edition
- *DBMS*: Database management system
- *AS*: Application server
- *FR*: Functional requirement
- *NFR*: Non-functional requirement
- *BWA*: Bad weather alert
- *DCN*: Date changed notification

## 4 References

- Analysis document: `./RASD.pdf`

## 5 Overview

This document specifies the architecture of MeteoCal spreading from the general into the specific. It also describes and justifies the architectural decisions and trade offs. The design was guided by a top-down process approach and the document structure reflects this tactic.

The document is organized as follows:

- *Part I, Introduction*: provides a synopsis of the architectural descriptions.
- *Part II, Design Overview*: provides a general description of MeteoCal including its functionality and matters related to the overall system and its design.
- *Part III, Design Considerations*: describes the design assumptions and constrains of MeteoCal.
- *Part IV, Software Architecture*: specifies the general architecture, describes the basic structure and interactions of the main subsystems.
- *Part V, Detailed System Design*: specifies in detail the components of the system through different architectural views.
- *Part VI, Appendixes*: provides supporting information and additional material.

## Part II

# Design overview

This section provides a general description of the software system including its functionality and concerns related to the overall system and its design.

## 6 Design context

The design context sets the limits for the system design, considering the functional and technological context.

### 6.1 Functionalities

The following functional requirements were identified in the RASD. These functionalities are grouped by the following functional areas:

#### 6.1.1 Managing users

Functional requirements:

- FR 1: Register to system
- FR 2: Login
- FR 3: Logout
- FR 4: Modify password
- FR 5: Recover password
- FR 6: Update personal data

#### 6.1.2 Managing calendars

Functional Requirements:

- FR 7: Add a new event
- FR 8: Modify an existing event
- FR 9: Delete an existing event
- FR 10: View your own schedule
- FR 11: View the details of your own event
- FR 12: Send an invitation to other users
- FR 13: Reply to an invitation
- FR 14: See the schedule of other users if their calendar is public
- FR 15: See the details of other user's public events
- FR 16: Receive a notification when the event details changes

### 6.1.3 Managing weather forecasts

Functional requirements:

- FR 17: Send a notification the day before an event in case of bad weather to all the event's participants
- FR 18: Propose an alternative schedule three days before an event in case of bad weather to the event creator
- FR 19: Show the weather forecasts for the scheduled events

## 6.2 System technologies

MeteoCal will be designed considering the client-server 3-tier distributed architectural style. Each tier requires specific technologies as depicted below:

### 6.2.1 Web tier

- Dynamic web pages containing XHTML, which are generated by web components.
- Web components developed with Java Server Faces technology, which is a user interface component framework for web applications.

### 6.3 Business logic tier

- Java Enterprise Edition 7(JEE7) platform supports applications that provide enterprise services in the Java language. It is the common foundation for the various kinds of components in Java.
- Enterprise Java Beans (EJB) 3.1, business components that capture the logic that solves or meets the needs of a particular business domain and persistence entities.
- GlassFish 4.1, a server that provides services such as security, data services, transaction support, load balancing, and management of distributed applications and supports the JEE7 platform.

### 6.4 Persistence tier

- MySQL Server 5.6.21, a RDBMS

## 7 General design description

This section presents the road map followed to model the architecture of MeteoCal, including its functionality and matters related to the overall system and its design.

## 7.1 Design approach

The design approach is based on a client-server 3-tier distributed system, where each tier is described as follows:

- *Client tier*: This tier is responsible of translating user actions and presenting the output of tasks and results into something the user can understand.
- *Business Logic tier*: This tier coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the client and the persistence tiers.
- *Persistence tier*: This tier holds the information of the system data model, and is in charge of storing and retrieving information from a database.

The design process followed a top-down process approach, so the outermost tiers were first identified and then broken into components that encapsulate the functionality. Hence each component is responsible for certain functionalities and interacts with others.

## 7.2 Overall design

This subsection presents the design model of MeteoCal, specifying the basic relations between packages, use cases and users.

### 7.2.1 General package design

Since each tier is broken into components and each component is responsible for a set of functionalities that fulfill the requirements, there is a correlation between use cases (functionality) and package design. In the diagram we can identify three packages:

- *User UI*: This package contains the user interfaces. It is responsible for the interaction with the user such as getting UI requests, referring them to the Business Logic package and retrieving the data back for displaying.
- *Business Logic*: This package contains the business logic components. This package is responsible for handling the User UI package requests, processing them and accessing the Persistence package if required to provide a response.
- *Persistence*: This package is responsible for managing the data requests from the Business Logic package.

Logged and unlogged users access directly the User UI package and submit requests to accomplish their tasks.

### 7.2.2 Detailed package design

Given the functional requirements identified we can encapsulate them within specific components in the package diagram as follows:

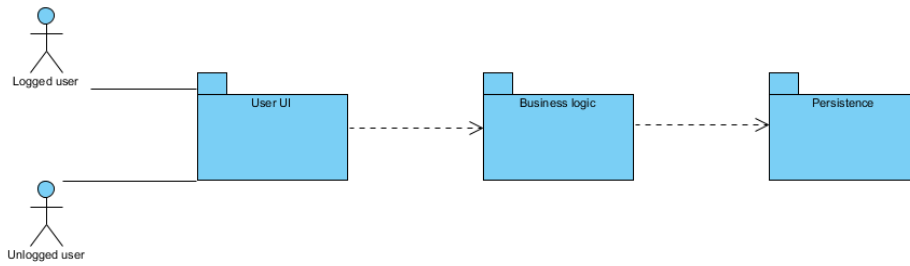


Figure 1: Basic package diagram

**User UI** These set of sub packages are responsible for encapsulating the user actions and forwarding information requests to the Business logic sub packages.

- *Login page*: this package implements FR2, FR5
- *Sign up page*: this package implements FR1
- *Calendar page*: this package implements FR3, FR7-FR12, FR14, FR15, FR19
- *Notification viewer*: this package implements FR13, FR16-FR18
- *User profile page*: this package implements FR4-FR6

**Business logic** These set of sub packages are responsible for handling requests from the User UI package, processing them and send back a response. These packages may access the Persistence package.

- *Login manager*: this package implements FR2, FR3
- *User profile manager*: this package implements FR1, FR4-6
- *Calendar manager*: this package implements FR7-FR11, FR14, FR15
- *Search manager*: this package implements FR12, FR14, FR15
- *Notification manager*: this package implements FR13, FR16-FR19
- *Forecast manager*: this package implements FR17-FR19

**Persistence** This sub package contains the data model for the system. It accepts requests from the Business Logic package.

- *Entity manager*: This package implements FR1-FR19

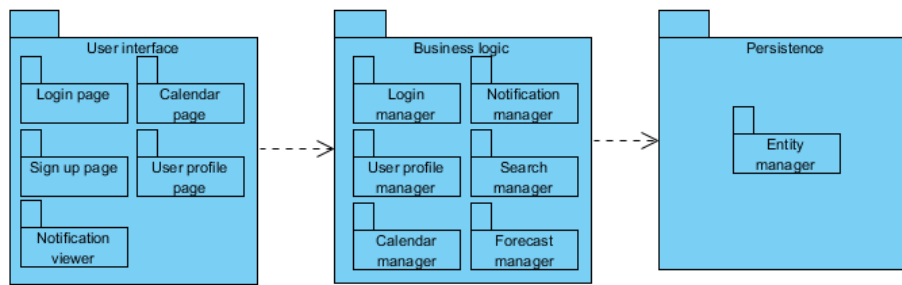


Figure 2: Detailed package diagram



## Part III

# Design considerations

This section encompasses the design considerations taken into account in the MeteoCal system design. Assumptions, dependencies, general constraints and performance requirements are clearly stated.

## 8 Assumptions

WHERE ARE THE ASSUMPTIONS??

## 9 Dependencies

<i>Dependency</i>	<i>Impact</i>
Java virtual machine that supports JEE7 is already installed on the OS.	MeteoCal only runs on operating systems that support the JEE7 platform.
The supported browsers will be Firefox and Chrome	MeteoCal outputs XHTML code that requires browsers that support most of the web standards, elsewhere the UI experience will be affected.
A JEE7 AS is required on the server side	MeteoCal cannot operate if there is no AS that supports the JEE7 standard

## 10 General constraints

This section describes the NFRs and the QoS details related to the design of the software product.

<i>Element</i>	<i>Requirement</i>
Memory	2 GB+
Database server	MySQL
Network	Internet access, HTTP protocol
Security	User data will be encrypted. SSL is not supported
Hard disk space	40 GB+

## 11 Performance requirements

### 11.1 Standard compliance

The software product does not have to meet any standard compliance.

### 11.2 Reliability

For assuring the reliability of the software product, it is mandatory to back up the database periodically.

### **11.3 Availability**

An AS is used to guarantee the availability of the software product. In a real scenario some redundancy in the AS instances is recommended, however here we will assume that all the tiers run on the same physical server.

### **11.4 Security**

The software product does not support SSL in AS. It supports the hashing of the user password according to sha5 algorithm in the database. It supports authorization according JAAS.

### **11.5 Maintainability**

The architectural style and the component definition described contribute to low coupling and high cohesion of the software product.

### **11.6 Portability**

The software product will be developed using the Java language and related dependent technologies. Java is specifically designed to have as few implementation dependencies as possible. Meaning that code that runs on one platform does not need to be recompiled to run on another.

## Part IV

# Software architecture

MeteoCal will be developed using a 3-tier JEE architecture. We have identified the components of each tier. Section 12 gives a brief description of the architecture we will adopt to develop MeteoCal and a small description for each component. A detailed description will be provided in part V.

## 12 Conceptual design

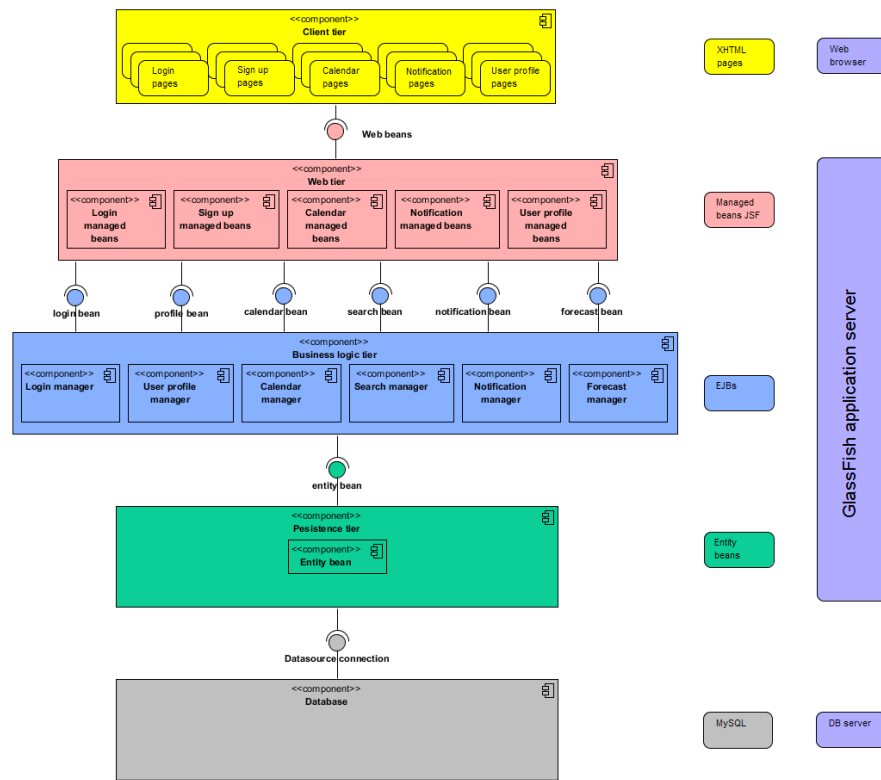


Figure 3: General architecture design

The diagram represented in figure 3 represents our conceptual design of MeteoCal. This diagram depicts all the components of the designed software, by clarifying the logical separation between the tiers (Although we will build our application based on the 3-tier architecture). It is clear that the user will interact with the XHTML pages, which in their side are implemented with beans to manage user interaction. This web interaction is then supported by the business tier, which holds on the information provided by the persistence layer. The persistence layer is the one in charge of the connection to the database and managing all the queries needed from the above layers.

## Part V

# Detailed software design

## 13 Database model

### 13.1 Conceptual Design

We developed the entity-relationship diagram following what we specified in the class diagram presented in the RASD.

#### Notes

- In the future the possibility of changing the user name could be implemented. Therefore we used an integer ID as the User primary key instead of the user name. In this way it will be simpler to manage future changes in the way the system manages user data. Emails can already be changed and thus weren't a suitable choice for the primary key.
- Places have an ID as primary key. It's the id that identifies the specified city in Open Weather Map (the external service used to get forecasts). In this way it will be simpler to manage places and forecasts according to the external service.

**ER diagram description** The User entity contains all the user's personal information. A user has exactly one calendar. A user can create events and he may receive notifications. Calendar is a weak entity with respect to User because if a user is deleted from the database's records then there's no need to keep track of its calendar. A calendar contains 0 or more events. An event is created by exactly one user and it's contained in at least one calendar (the one belonging to its creator). Events can generate notifications. An event is held in one Place (that may be unspecified) and has a related forecast, if available. A Place is the location of at least one Event. It can also be the object of some weather forecasts. Forecast is a weak entity with respect to Place because forecasts referring to places that are not in the DataBase make no sense. A forecast has to be used by at least one event, otherwise it would be useless. A notification has to have at least one receiver and it concerns exactly one event. ECN, BWA, SDP and Invite are heir classes of Notification because they share the basic structure but have to be managed in different ways.

**Logical design** The logical design of the database represent its final representation. The diagram shows the different tables required to store MeteoCal data. Every attribute is a column of the table in which is declared. Every table may involve foreign keys. These are attributes that refer to another entity in the database, which is specified through its primary key. The LD diagram is derived from the ER diagram. Thus it presents the same structure for information. The relations specified in the ER are commonly specified in the LD diagram using foreign keys. We decided to remove the entity calendar. Conceptually it was useful but in practice, given that the relation with the user is 1:1 it can be omitted without any loss. A calendar table wouldn't be reasonable because it

would not contain enough information. We included the privacy setting of the calendar in the User table and connected directly users and the events their events. The User table contains all the personal details of a user. Its primary key is an integer generated automatically by the DBMS. Users are connected to their events with the Participates table. This table has a primary key composed by the couple user - event. The first one is a foreign key referring to the ID of the participant, the second is a foreign key with respect to the ID of the event. The Event table contains all the tuples representing events. Each tuple has a unique ID generated by the database. The attributes Place and Weather are optional because if the event is indoor users are not required to specify them. Place is a foreign key referring to the place ID. It specifies that each an event is held in 0 or 1 place. Weather is a foreign key referring to the forecast ID. It specifies the forecast which is shown to the user for the event. The Forecast table has an integer primary key generated by the dbms. It has a foreign key that links the forecast to a specific place ID. The Place table has an integer primary key which is the city code used in the Open Weather Map. Therefore it has to be set by MeteoCal system and not automatically by the database. The Notification table contains all the different type of notification. The type is specified in the NotificationType attribute. We decided to store all the types in a single table because it simplifies the process of searching pending notifications for a given user. A notification has a primary key composed by the notification ID and a user ID. The notification ID is an integer generated by the database. The notification ID alone is not enough to identify a single notification because the same notification can be sent to multiple user and each of them may reply in a different way to it.

## **Part VI**

# **Appendixes**

## **14 RASD modifications**

### **14.1 Alloy model**

In the alloy model we stated that in our database we only keep track of one forecast per place (the most recent one). This fact is not true according to the database design and has to be removed. In fact we can have different events in the same place but on different days, so we need to have different forecasts.

# Contents

<b>I</b>	<b>Introduction</b>	<b>1</b>
<b>1</b>	<b>Purpose of this document</b>	<b>1</b>
<b>2</b>	<b>Scope</b>	<b>1</b>
<b>3</b>	<b>Definitions and acronyms</b>	<b>1</b>
3.1	Definitions . . . . .	1
3.2	Acronyms and abbreviations . . . . .	2
<b>4</b>	<b>References</b>	<b>2</b>
<b>5</b>	<b>Overview</b>	<b>2</b>
<b>II</b>	<b>Design overview</b>	<b>3</b>
<b>6</b>	<b>Design context</b>	<b>3</b>
6.1	Functionalities . . . . .	3
6.1.1	Managing users . . . . .	3
6.1.2	Managing calendars . . . . .	3
6.1.3	Managing weather forecasts . . . . .	4
6.2	System technologies . . . . .	4
6.2.1	Web tier . . . . .	4
6.3	Business logic tier . . . . .	4
6.4	Persistence tier . . . . .	4
<b>7</b>	<b>General design description</b>	<b>4</b>
7.1	Design approach . . . . .	5
7.2	Overall design . . . . .	5
7.2.1	General package design . . . . .	5
7.2.2	Detailed package design . . . . .	5
<b>III</b>	<b>Design considerations</b>	<b>8</b>
<b>8</b>	<b>Assumptions</b>	<b>8</b>
<b>9</b>	<b>Dependencies</b>	<b>8</b>
<b>10</b>	<b>General constraints</b>	<b>8</b>
<b>11</b>	<b>Performance requirements</b>	<b>8</b>
11.1	Standard compliance . . . . .	8
11.2	Reliability . . . . .	8
11.3	Availability . . . . .	9
11.4	Security . . . . .	9
11.5	Maintainability . . . . .	9
11.6	Portability . . . . .	9

<b>IV</b>	<b>Software architecture</b>	<b>10</b>
<b>12</b>	<b>Conceptual design</b>	<b>10</b>
<b>V</b>	<b>Detailed software design</b>	<b>11</b>
<b>13</b>	<b>Database model</b>	<b>11</b>
13.1	Conceptual Design . . . . .	11
<b>VI</b>	<b>Appendixes</b>	<b>13</b>
<b>14</b>	<b>RASD modifications</b>	<b>13</b>
14.1	Alloy model . . . . .	13