MeteoCal

Software Enginnering 2 Project

Design Document

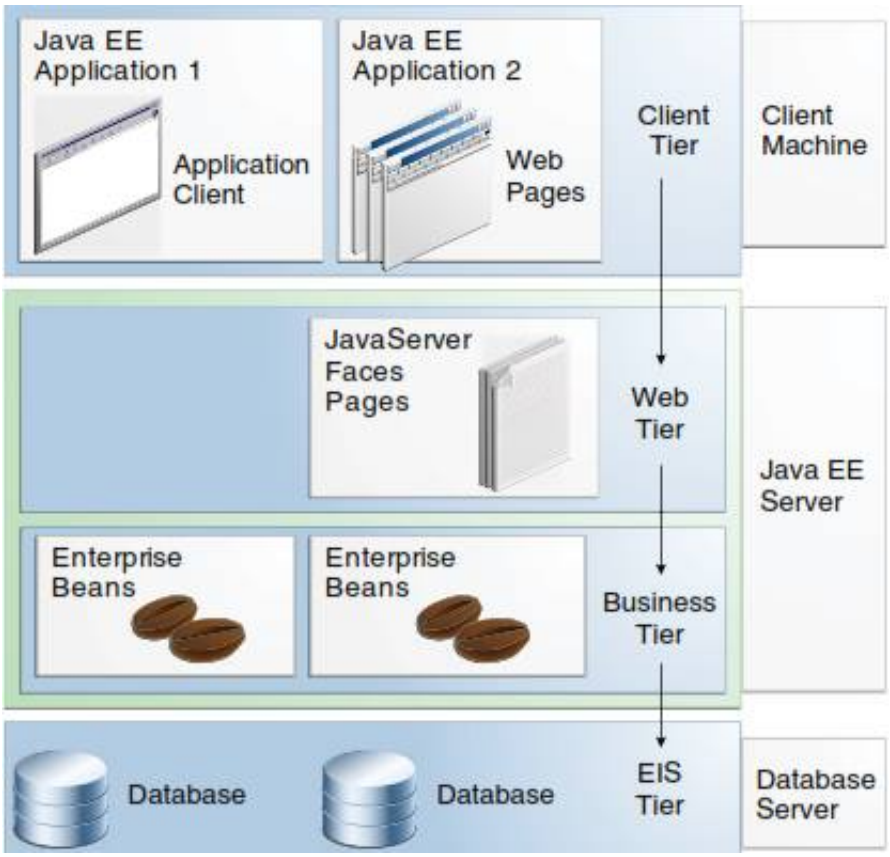
D. Pensa
M. Pini
A. Pintus

Contents

1 Architecture Description

1.1 JEE Architecture Overview

The developing of *MeteoCal* will be based on JEE architecture.



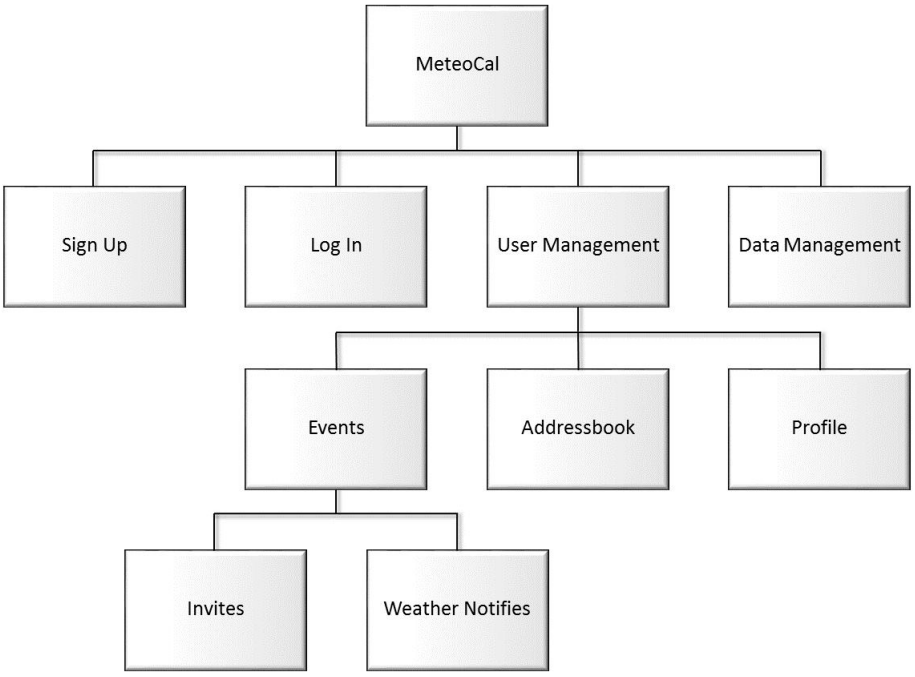
JEE has a four tiered architecture divided as:

- **Client Tier**: it contains Application Clients and Web Browsers and it is the layer that interacts directly with the actors. As our project will be a web application the client will use a web browser to access pages
- **Web Tier**: it contains the Servlets and Dynamic Web Pages that needs to be elaborated. This tier receives the requests from the client tier and forwards the pieces of data collected to the business tier waiting for processed data to be sent to the client tier, eventually formatted
- **Business Tier**: it contains the Java Beans, that contain the business logic of the application, and Java Persistence Entities
- **EIS Tier**: it contains the data source. In our case it is the database allowed to store all the relevant data and to retrieve them

1.2 Identifying Subsystems

Here we have a list of possible subsystems of the application, in order to separate logically components and have a clearer view of functionalities and their interaction:

- Sign Up
- Log In
- User
  - Event Managing
    - Invites
    - Weather Notifies
  - Address Book Managing
  - Profile Managing
- Data

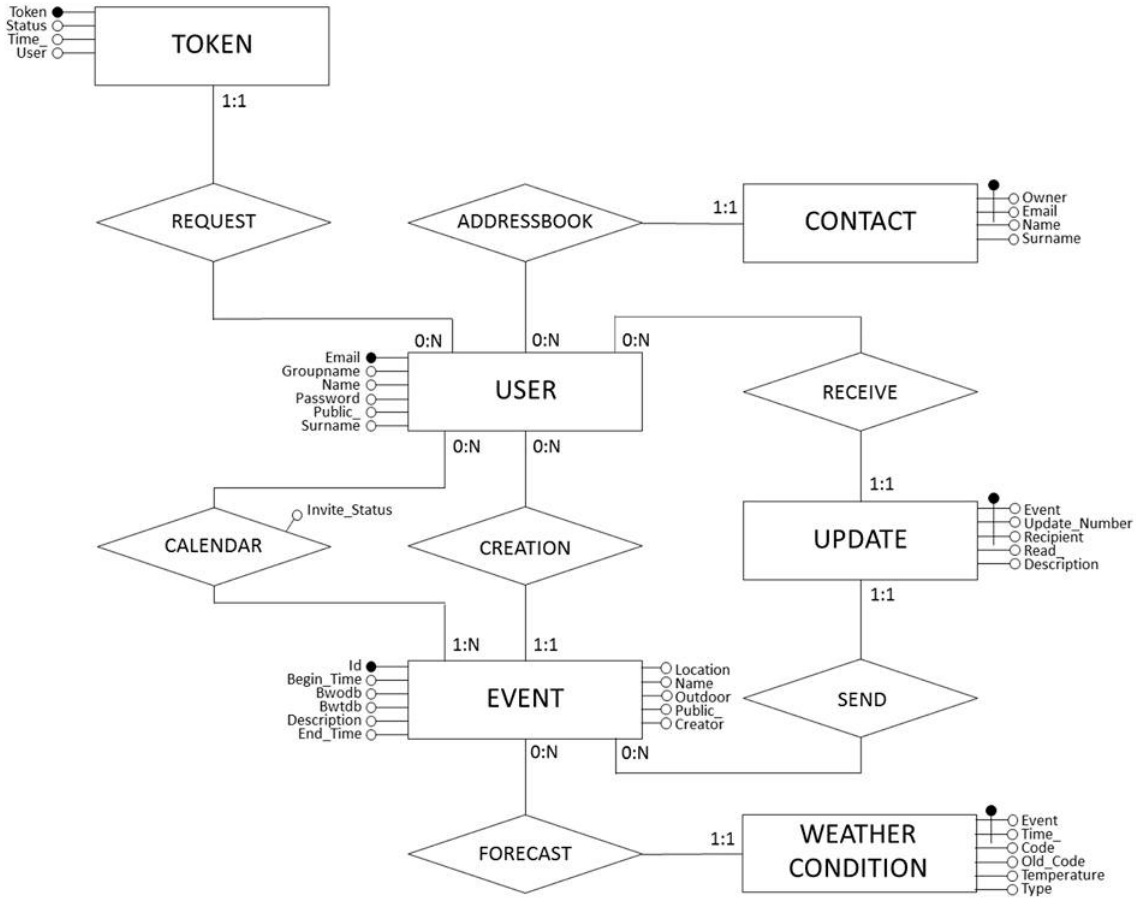


2 Persistent Data Management

The development of *MereoCal* will be based on a MySQL database, which design is described through an Entity-Relation Model (ER)

2.1 Conceptual Design

This is the Entity-Relation Model description



- User : Primary Key = **Email**
  - Email
  - Name
  - Surname
  - Public_ : Flag to disinguish between public and private calendar
  - Groupname: For future extension on the User class (eg: Administrator)
  - Password
- Event : Primary Key = **ID**
  - ID
  - Name
  - Begin_Time
  - End_Time
  - Description
  - Public_ : Flag to distinguish between public and private events
  - Outdoor : Flag to distinguish between outdoor and indoor events
  - *Creator* : Refers to the user who created the event
  - Location
  - Bwodb : Flag to known if the system already sent notification for BadWeatherOneDayBefore
  - Bwtdb : Flag to known if the system already sent notification for BadWeatherThreeDayBefore
- Contact : Primary Key = **Email,Owner**
  - Email
  - *Owner*: Refers to the user who added this contact to his addressbook
  - Name
  - Surname
- Weather Condition : Primary Key = **Time_,Event**
  - Time_: needed because the event related could be on multiple dates
  - *Event*: Refers to the event for which the system has provided a weather forecast
  - Type : Rainy, Sunny, Windy and so on
  - Code : code to identify type of weather
  - Old_Code : Flag to check if weather is changed
  - Temperature

- Update : Primary Key = **Event,Update_Number,Recipient**
  - *Event*: Refers to the event which has been (or needs to be) updated
  - Update_Number: Needed because each event could need to notify multiple changes
  - *Recipient*: Refers to the user which has been (or needs to be) notified
  - Description
  - Read_ : Flag to check if update has been read
- Token : Primary Key = **Token**
  - Token: string represents token value
  - Status: enabled or disabled
  - Time: hour of creation
  - User: who request the token

"One to many" relations are included in the attributes of entities as foreign keys

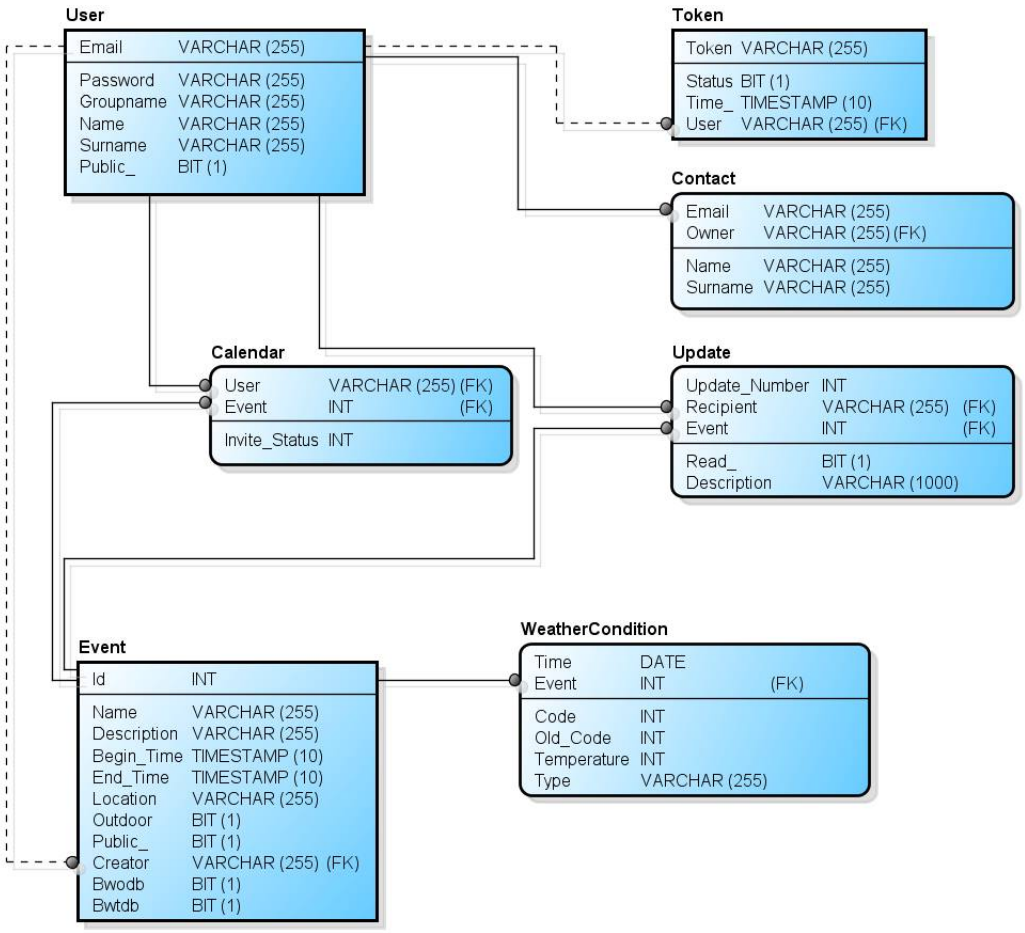
- Addressbook: each User can have from 0 to N Contacts, but each Contact corresponds to one and only one User
- Creation: each User can create from 0 to N Events, but each Event has one and only one creator User
- Forecast: each Event can have from 0 to N Weather Conditions, but each Weather Condition corresponds to one and only one Event
- Send: each Event can send from 0 to N Updates, but each Update corresponds to one and only one Event
- Receive: each User can receive from 0 to N Updates, but each Update corresponds to one and only one Event

"Many to many" relations will be SQL tables identified by the primary keys of the entities in the relation

- Calendar, that includes also a flag "Invite Status" to keep track of the status of invites

2.2 Logical Design

2.2.1 Translation To Logical Model



Here it is the logical description of the SQL tables and their relations through foreign keys

USER(Email,Password,Groupname,Name,Surname,Public_)

CALENDAR(User_,Event,Invite Status)

CONTACT(Email,Owner,Name,Surname)

UPDATE(Update_Number, Event, Recipient,Description,Read_)

EVENT(ID,Name,BeginTime,EndTime,Description,Public_,
Creator,Location_,Outdoor,Bwodb,Bwtdb)

WEATHER_CONDITION(Time,Event,Code,Old_Code,Type,Temperature)

TOKEN(Token,Status,Time,User)

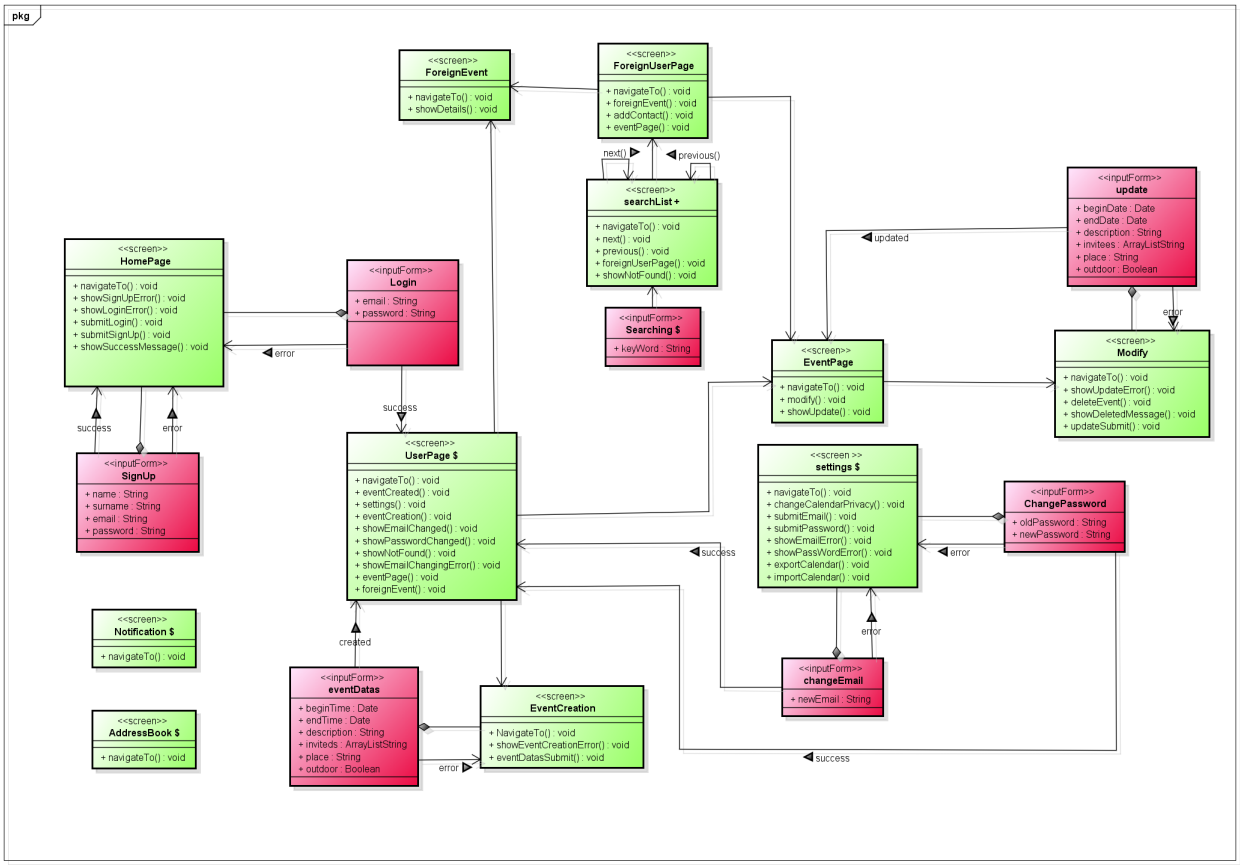
Foreign Keys

- CALENDAR.User —>USER.email
- CALENDAR.Event —>EVENT.ID

- CONTACT.Owner —>USER.email
- UPDATE.Event —>EVENT.ID
- UPDATE.Recipient —>USER.email
- EVENT.Creator —>USER.email
- WEATHER_CONDITION.Event —>EVENT.ID
- TOKEN.User —>USER.email

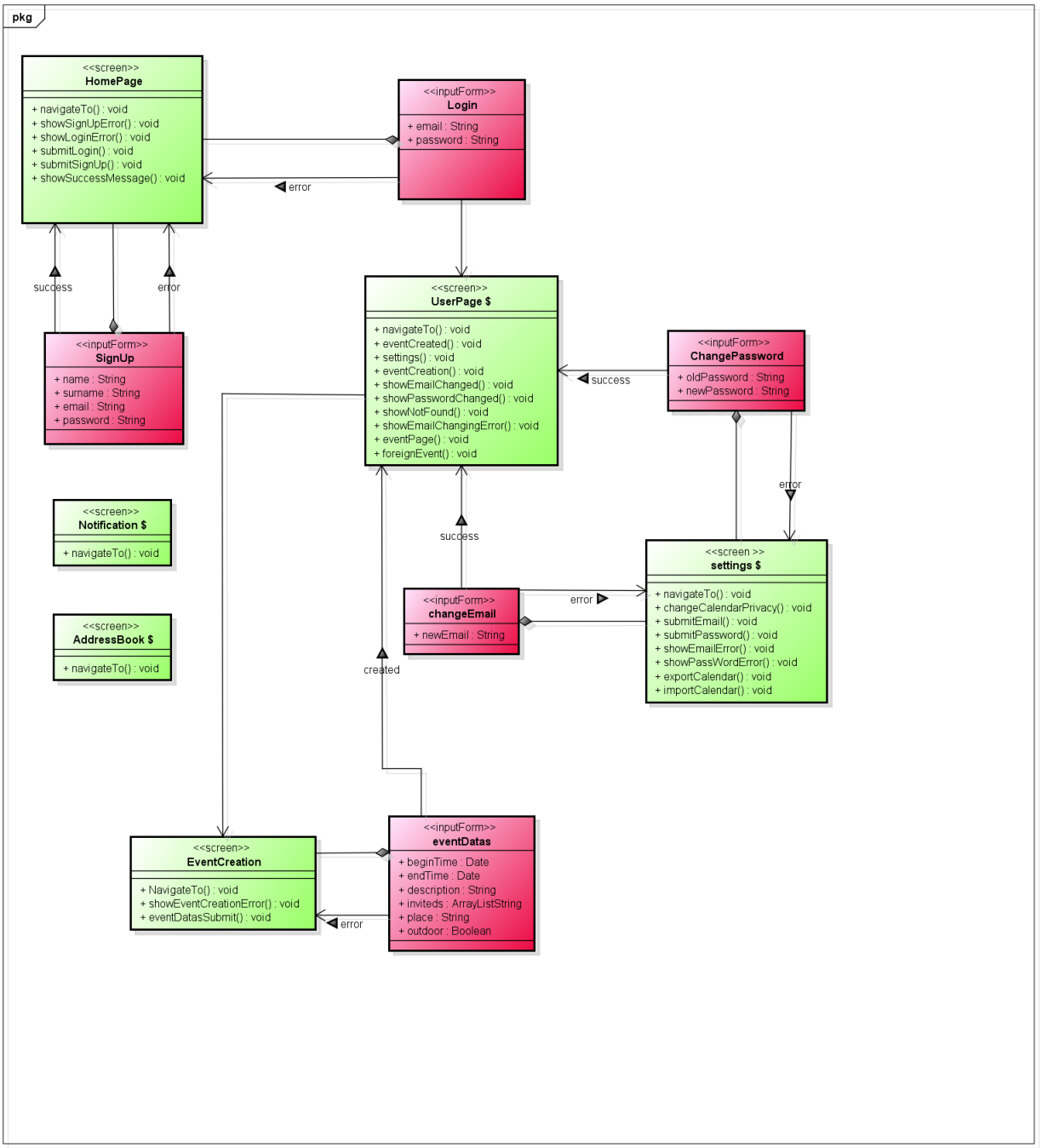
3 User Experience

In this paragraph we describe the UX diagram of our project.
It is a class diagram with two types of stereotypes: «screen» which represents the pages, and «inputForm». The following is the complete UX diagram, after we will show the UX divided by userUX, eventUX and searchingUX.
The dollar symbol indicates that the page (or the form) is reachable from every page by the logged user.



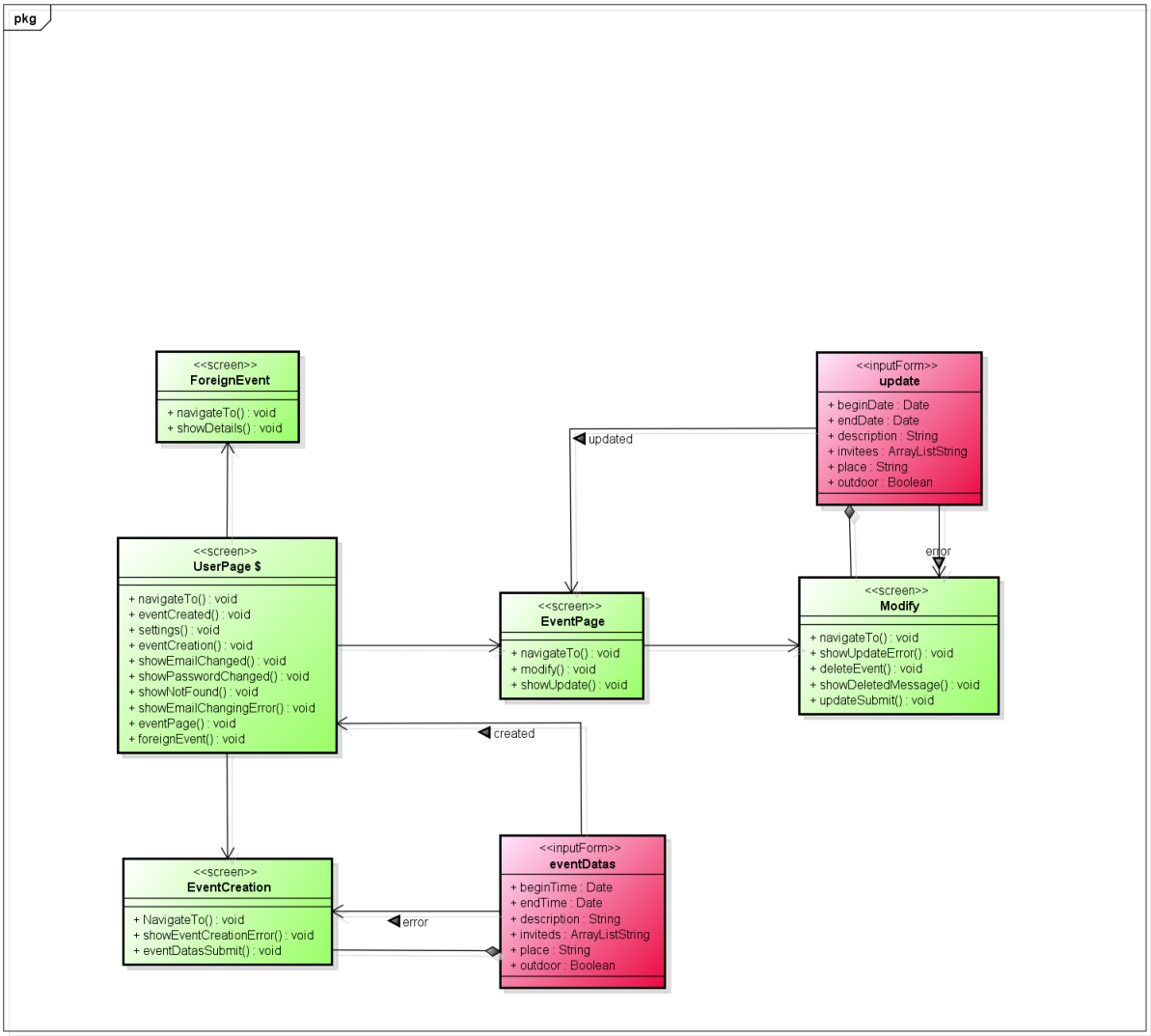
3.1 User UX

This part of the UX diagram is about user functionality.
The users can login or signup in *MeteoCal*, if that fails the system shows in the same page an error message. Users can modify their datas through settings page, if there are problems the system show an error message in the settings page otherwise the system redirects the users on their page after save. The users can also create an event with a click on EventCreation link from userpage and than compiling the EventData form fulfilling request datas, if there'are problems the system show an error message in creationEvent page otherwise the system redirects the users on their page with a confermation message. The Users can read their notifications or visit their address-book through each page, if they are logged in.



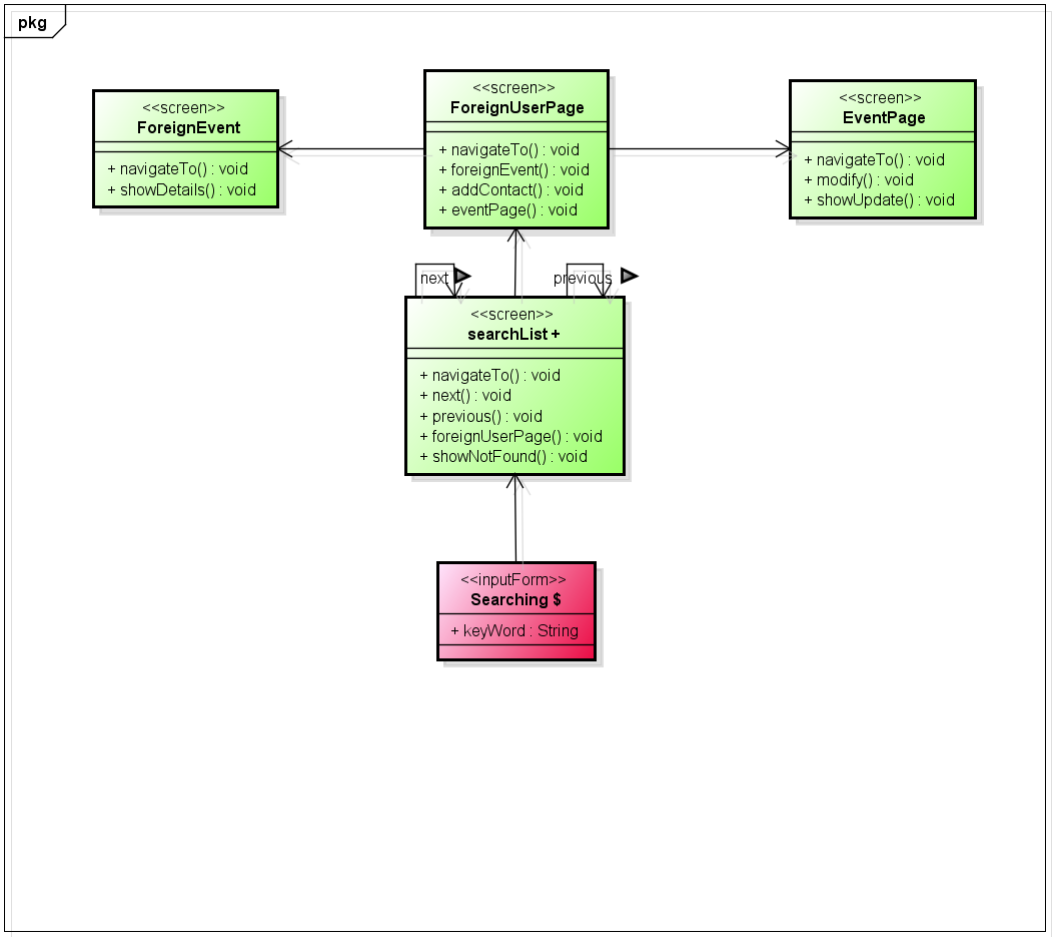
3.2 Event UX

This part of the UX diagram is about event.
When user clicks on EventLink, he will be redirected in the EventPage if he is creator of the event, in the foreignEvent page otherwise. From the EventPage, creator can modify event informations clicking on modify link and compiling the form that there is in modify page, if there'are problems the system show an error message in modifyPage page otherwise the system redirects the users on EventPage with a confermation message. The creation of the event is explained in User UX diagram.



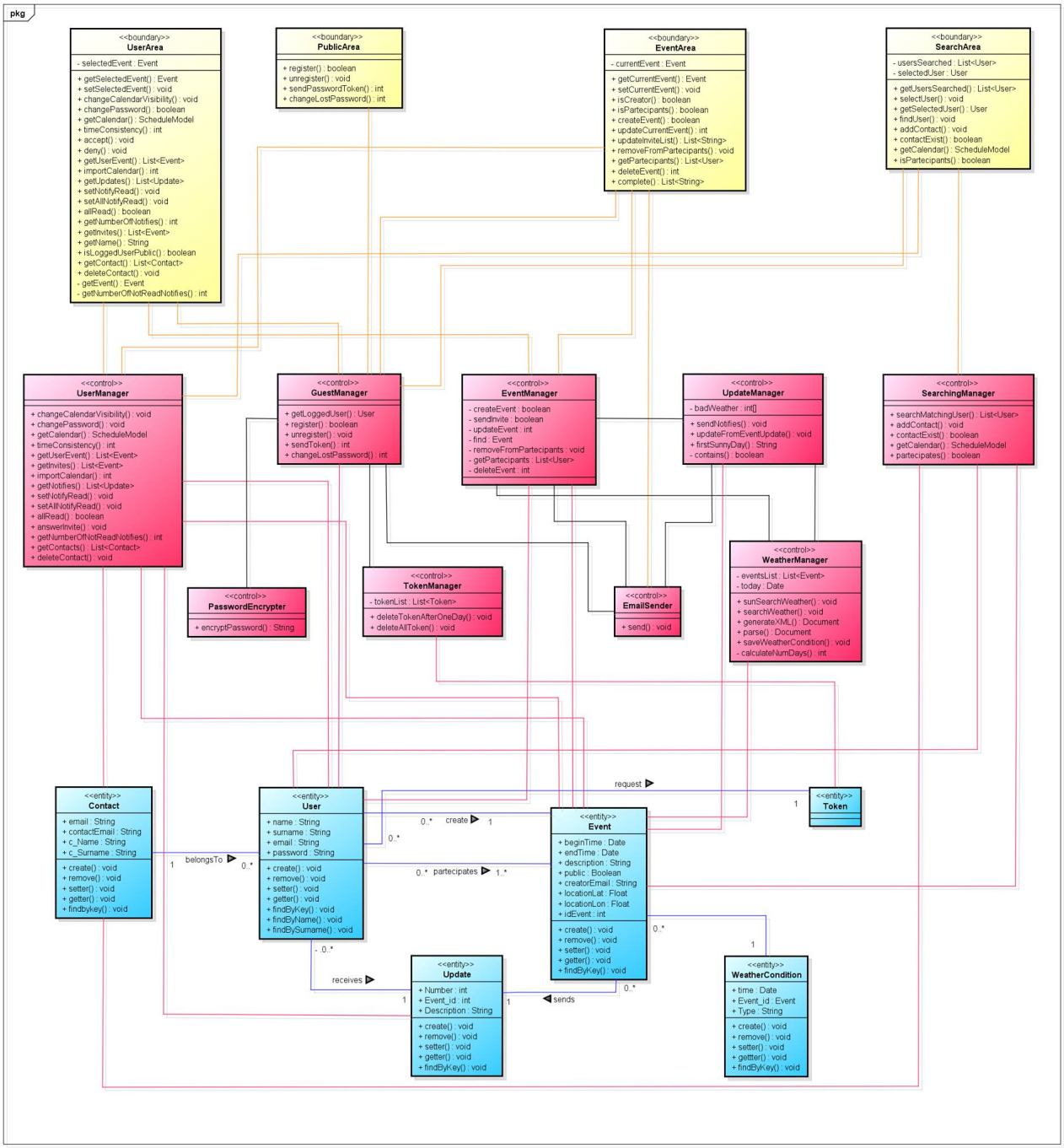
3.3 Searching UX

This part of the UX diagram is about searching function.
Each logged user can search another user compiling the search bar bystander at the top of every page. When users click on searchButton, system shows a results list in searchList page; clicking on a result he will be redirected on page of another user, an then he can click on an event in the other user calendar and the system show him the event page with the above rules.



4 BCE Diagram

We have used a Entity-Control-Boundary model to create the component of our application, this allow us to develop the app with Model-View-Controller pattern.
In this diagram the boundaries are the interface through the users can use *MeteoCal* , the controls are the connection between boundaries and entities and the entities are the data saved in our database.

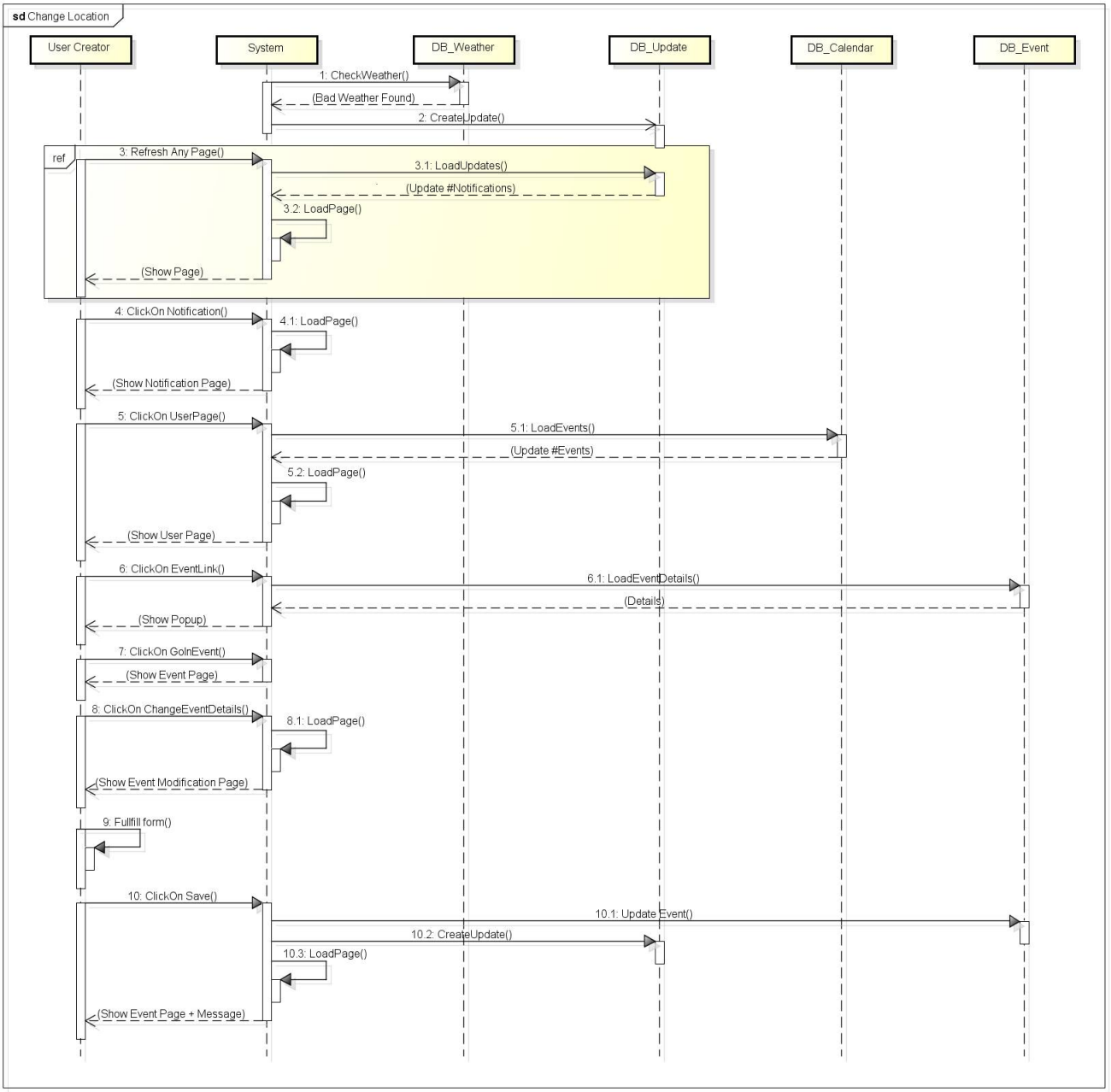


5 Sequence Diagrams

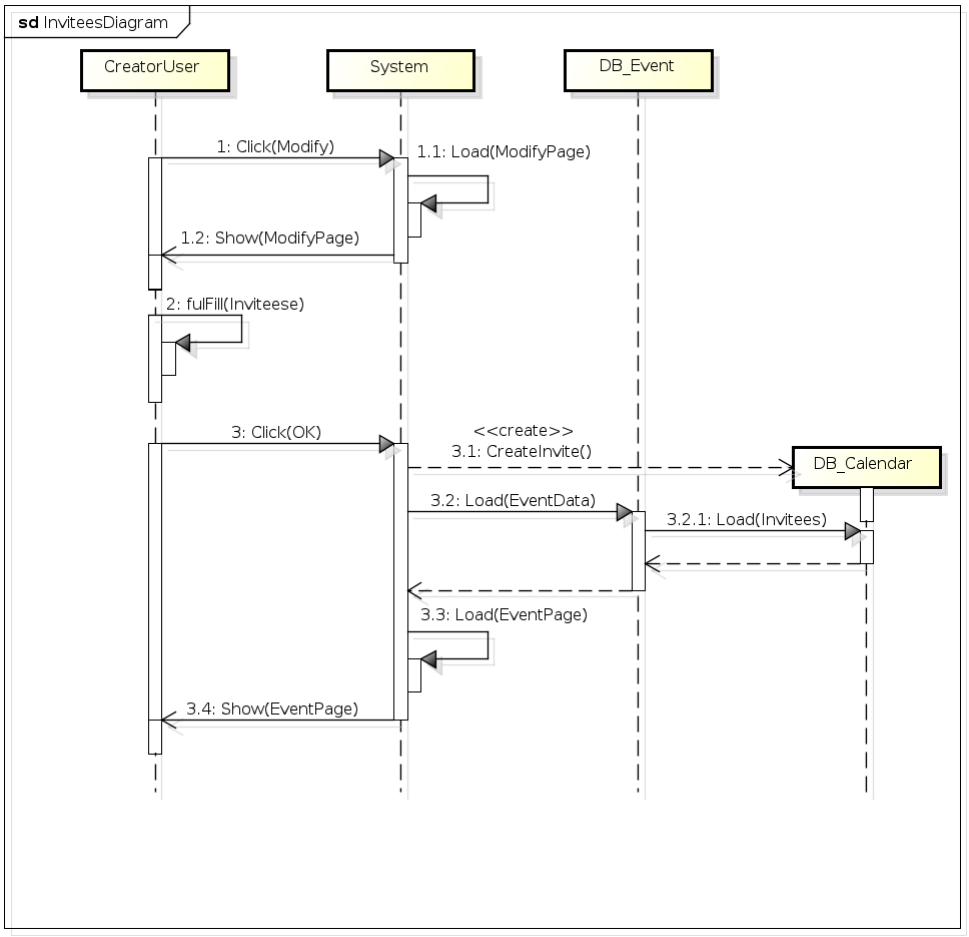
In this section there is a couple of sequence diagram to describe more precisely the interaction between the system and the tables in the database. For each loaded page, the system, checks every table concerning to that page, but in the diagram we wrote only useful table for that schema. When there is a creator arrow mean new line is created in that table.

5.1 Change Location of Event

In this diagram we assume there is another thread updating the weather table.



5.2 Invitees


sd InviteesDiagram
CreatorUser
System
DB_Event
DB_Calendar
1: Click(Modify)
1.1: Load(ModifyPage)
1.2: Show(ModifyPage)
2: fulFill(Inviteese)
3: Click(OK)
<<create>>
3.1: CreateInvite()
3.2: Load(EventData)
3.2.1: Load(Invitees)
3.3: Load(EventPage)
3.4: Show(EventPage)

6 Working Hours Schedule

	Pensa	**Pini**	**Pintus**	**h/sect**
General Description	1	1	1	**3**
Data Planning	2	3	10	**15**
UX Model	9	9	4	**21**
BCE Model	9	5	3	**17**
Sequence Diagrams	3	3	3	**9**
RASD Adjustment	0	1	1	**2**
h/pers	**24**	**22**	**22**	**67**

7 Changing History

2015/01/21:

- ER and Description
- Logic ER and Description
- Method added to BCE diagram
- Sequence diagram updated according to ER model