POLITECNICO
DI MILANO

SW Design Description Document

MeteoCal

December 04, 2014

Manuel Grillo
Francesco Fermi
Jerry M. Jude

0. Index

Table of Contents

(continues)

1. Architecture description

The Java EE platform uses a distributed multitiered application model for enterprise applications. Application logic is divided into components according to function, and the application components that make up a Java EE application are installed on various machines depending on the tier in the multitiered Java EE environment to which the application component belongs.

1.1. Java EE architecture overview

Before explaining the architecture of our web application, we emphasize the JEE architecture.

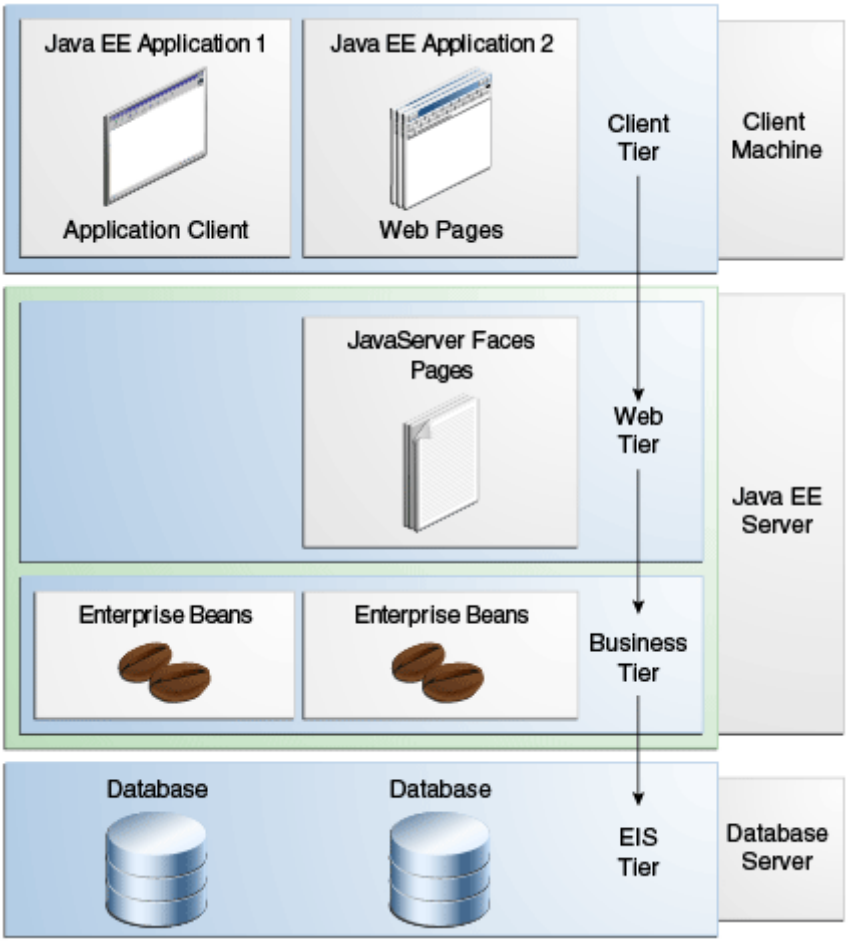

Figure 1 – JEE architecture. Diagram of client-server communication showing detail of entities, session beans, and message-driven beans in the business tier. Thanks to this architecture, we are able to develop, deploy and manage our multi-tier, server-centric application.

Figure 1 – JEE architecture shows two multitiered Java EE applications divided into the tiers described in the following list:

- client-tier components run on the client machine;
- web-tier components run on the Java EE server;
- business-tier components run on the Java EE server;
- EIS (Enterprise Information System) tier runs on the EIS server.

Although a Java EE application can consist of all tiers shown in *Figure 1 – JEE architecture*, Java EE multitiered applications are generally considered three-tiered applications because they are distributed over three locations: client machines, the Java EE server machine, and the database or legacy machines at the back end.

1.1.1. Client tier

The proposed project will be a web application. Accordingly, the users are required to access it using a web browser (i.e. no proprietary application client will be installed). Therefore, the client tier will be composed by a web client. The web client consists of two parts:

- dynamic web pages containing markup languages, which are generated by web components running in the web tier;
- a web browser, which renders the pages received from the server.

A web client is sometimes called a “thin client”.

1.1.2. Web tier

In the proposed project, we are going to use the JSF (Java Server Faces), which are developed integrating the MVC (Model-View-Controller) design pattern, so that applications can be well designed with relatively high maintainability. In this tier there are:

- managed beans;
- facelet views.

1.1.3. Business tier

The business code is the logic of our project. The business tier contains the Java Beans (which contain the business logic of MeteoCal) and the Java Persistence Entities.

1.1.4. EIS tier

The enterprise information system tier handles EIS software and includes the data source. In our application, the database system and our interface to the API to the external information resources represent the data source.

1.2. Identifying sub-systems

The high-level system described in *Section 1.1 – Java EE architecture overview* is subsequently decomposed into other sub-systems:

- Registration.
- Log in.
- Event manager.
- User:
    - search;
    - event manager;
    - notification manager;
    - calendar manager (e.g. for import/export functionalities).
- Weather forecast.
- Email.

Here follows a descriptive diagram:

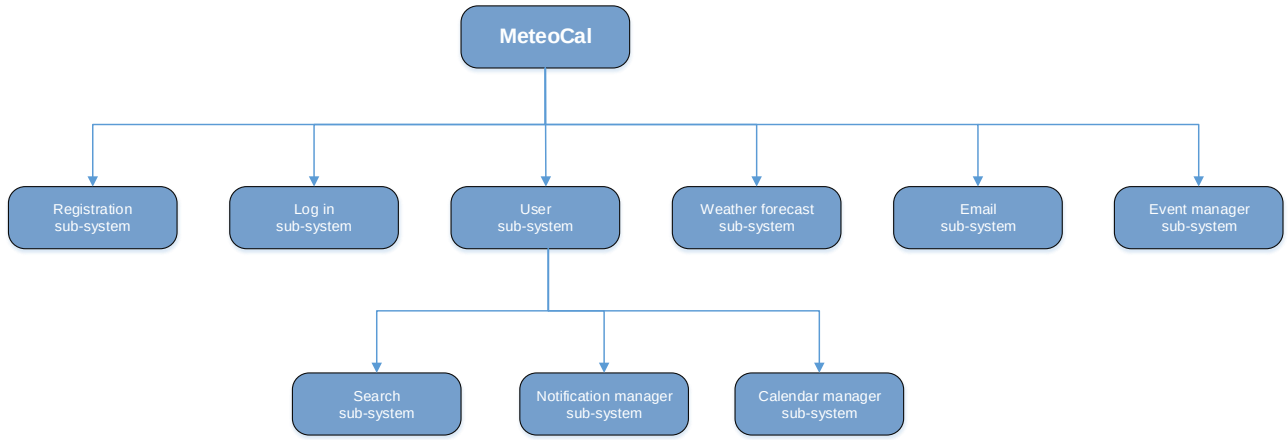

Figure 2 – System context diagram showing the sub-systems. Even if it is in a lower level than the one described in Section 1.1 – Java EE architecture overview, it is still a high-level view of the overall system.

2. Persistent data management

The data required by the tasks handled by the system stored into a MySQL relational database. MySQL is an open source SQL relational database management system (i.e. RDBMS).

2.1. Conceptual design

The conceptual design allows thinking about the data we want to store and the relation between them.

The entities that characterize the system are:

- `Event`: it represents the single instance of an event.
- `Weather condition`: It contains both the weather constraint required by the event. Its instance is created if the user formally gives a valid location.
- `Weather forecast`: It contains both the weather constraint required by the event. Its instance is created if the user formally gives a valid location.
- `User`: It contains all the "static" information of a user. It is created after a successful registration. It also contain the privacy setting of the calendar.
- `Information`: It is a notification. It is a message that may optionally be linked to an event (e.g. an alert about the weather forecast or a stand-alone informative message).
- `Invitation`: It is a notification. The user perceive it in the same way of an information, but it is only a message with an optional link to an event.
- `Answer`: It contains all the answers given by the users after having answered an invitation (i.e. attending or not going).

In addition, there will be a separate database (location with all the supported locations for the weather forecast functionalities.

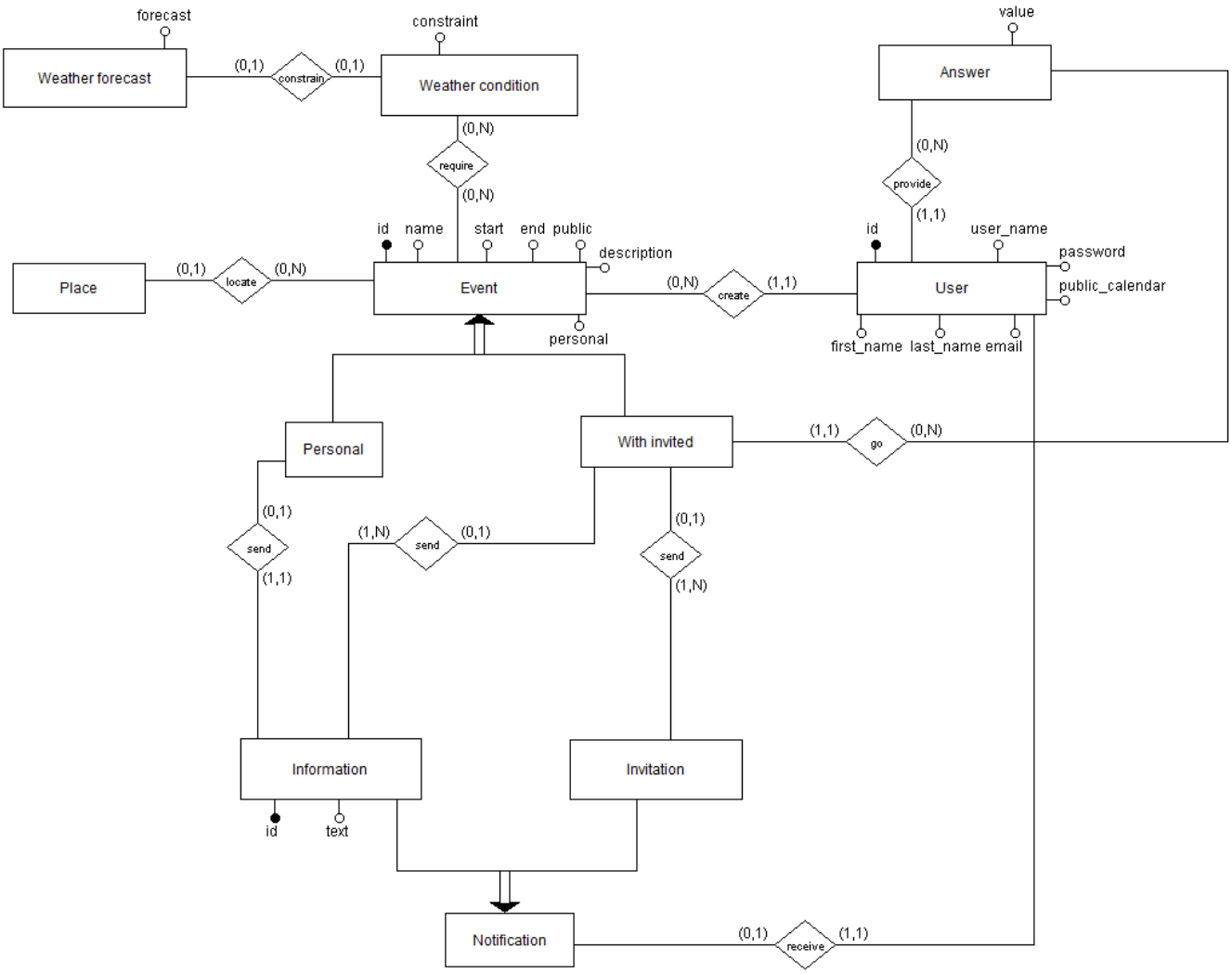

Figure 3 – This diagram is more abstract than the one that we will obtain in Subsection 2.2.1. – ER restructuration, but still created keeping in mind the willingness to build a data source (e.g. it is not necessary to have a separate `id` for the `User`, since `user_name` is a unique value, but in a performance perspective it is a good choice.

2.2. Logical design

The logical design aim to better represent the database structure of the system.

2.2.1. ER restructuration

In order to build the model from the diagram given in *Section 2.1 – Conceptual design*, we have to perform some transformations:

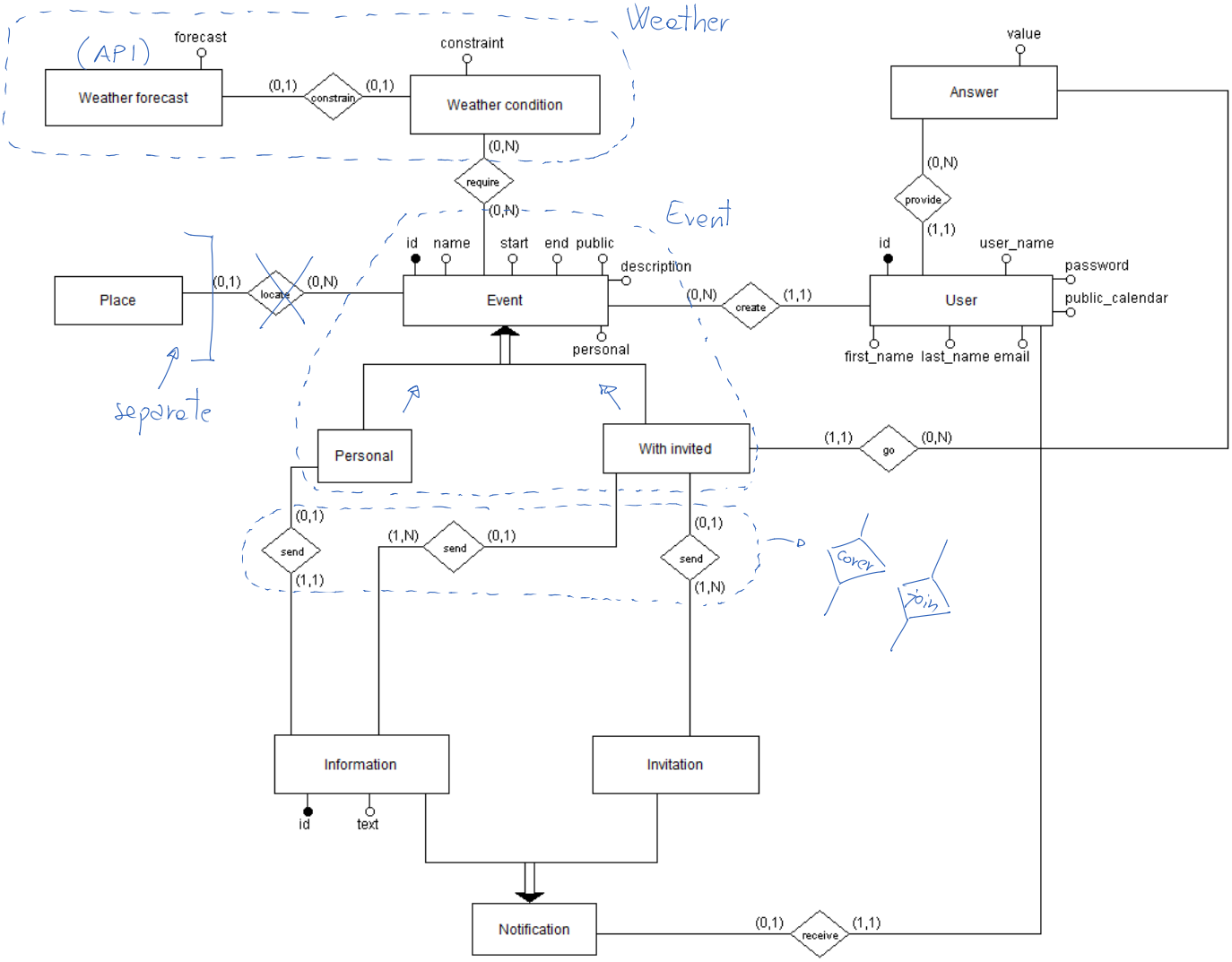

Figure 4 – Intermediate modifications of the restructuration.

- The weather forecast will be periodically updated in the same entity containing the weather condition. Thus, `Weather forecast` and `Weather condition` will be both merged in `Weather`.
- There will be no separate entities to distinguish a personal event (i.e. an event that has only its creator as an attendee) and an event with invited people: both will be in a single entity called `Event`. This information can be retrieved by checking the attribute `personal`.
- We separate the `Place` entity in a distinct database to better handle updates and to avoid expensive queries. It will contain all the supported locations for the weather forecast functionalities. In this way it will be easier to migrate to other weather forecast platforms or to refactor the `Place` database itself
- The `location` attribute is a string that can be either generated by the location given by the user (using the `Place` database) or written by him/her in natural language.

The result is the subsequent diagram:

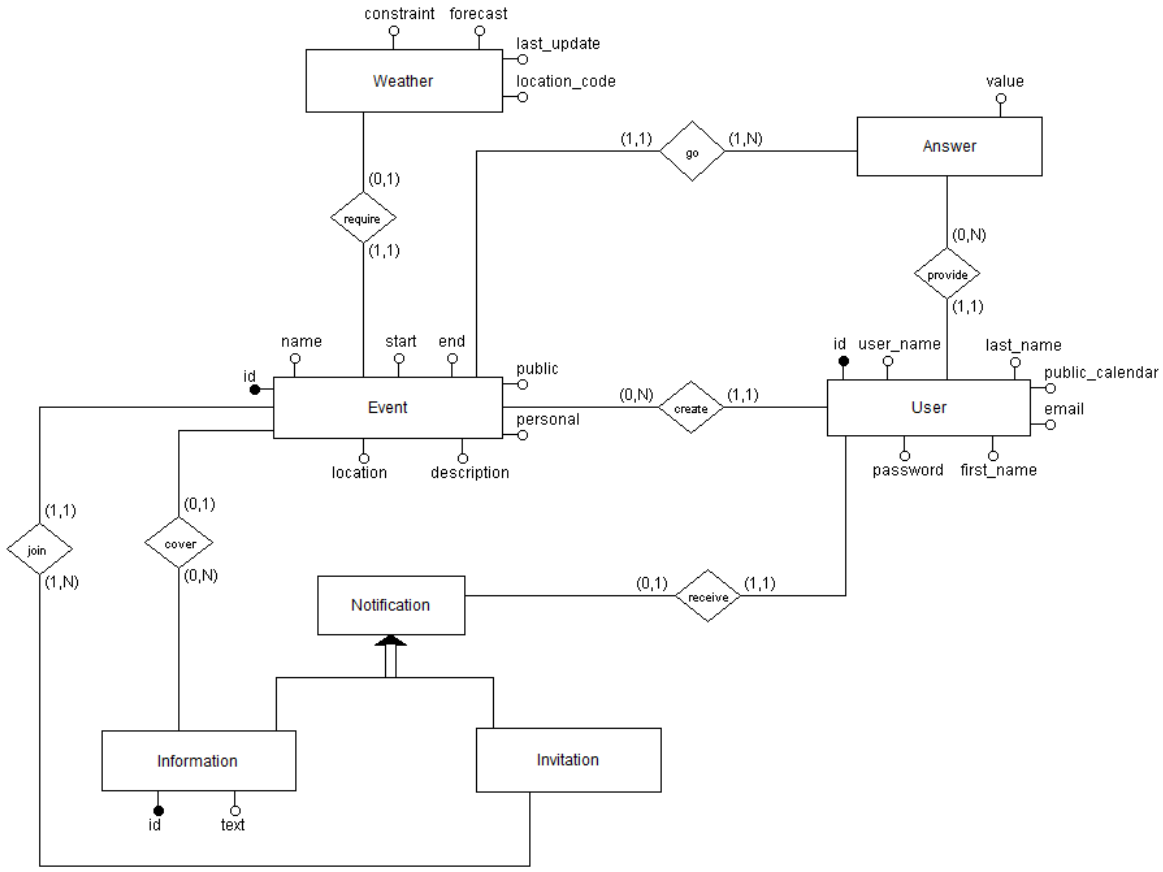

Figure 5 – New diagram. Restructuration of the diagram given in Section 2.1 – Conceptual design.

2.2.2. Translation to the logical model

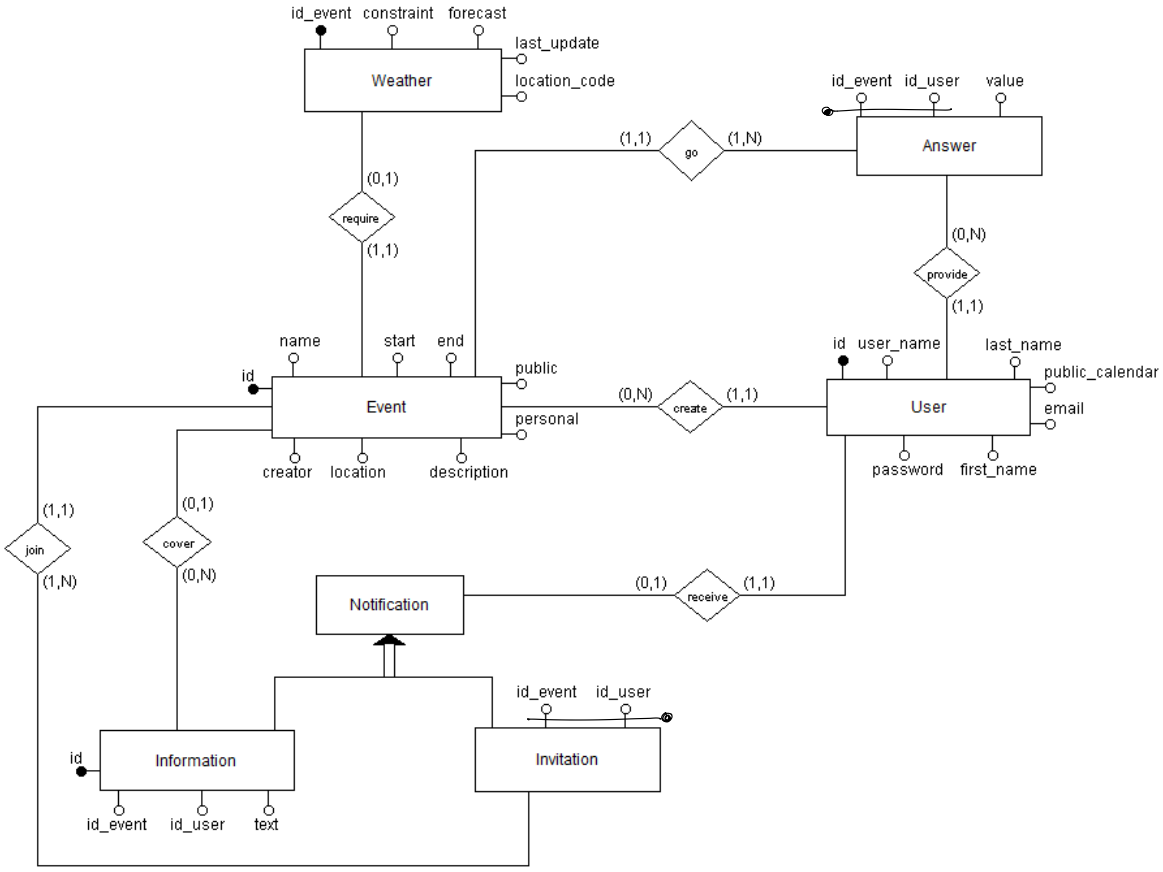

Figure 6 – Final version of the diagram given in Subsection 2.2.1 – ER restructuration. It ultimately matches the final logical design. The difference between this one and the one displayed in Figure 5 – New diagram is that in this one we have written all the primary keys and the foreign keys.

Now we can derive the new attributes to allow the logic of the computing system to join different tables:

- we translate the `create` relation between `User` and `Message` inserting a foreign key (`creator`, which refers to the `id` of `User`) inside into the `Event` table;
- we translate the `provide` relation between `User` and `Answer` by inserting a foreign key (`id_user`, which refers to the `id` of `User`) into the `Answer` table;
- we translate the go relation between `Event` and `Answer` by inserting a foreign key (`id_event`, which refers to the `id` of `Event`) into the `Answer` table;
- we translate the `require` relation between `Event` and `Weather` by inserting a foreign key (`id_event`) into the `Weather` table;
- we translate the `receive` relation (between `User` and the notifications) inserting a foreign key (`id_user`) into the tables of the notifications;
- we translate the `join` relation (previously derived from the `send` relation) between `Invitation` and `Event` by inserting a foreign key (`id_event`) into the `Invitation` table;
- we translate the `cover` relation (previously derived from the `send` relation) between `Information` and `Event` by inserting a foreign key (`id_event`) into the `Information` table.

Here follows the meaning of the BOOLEAN values:

- The `public_calendar` attribute in the `User` table: if it is `true`, the calendar of the user is public, otherwise it is false.
- The `public` attribute in the `Event` table: if it is `true`, the event is public, otherwise it is `false`.
- The `personal` attribute in the `Event` table: if in the creation or the last modification of an event there are no invited it is `true` – `false` otherwise.
- The `value` attribute in the `Answer` table: If it is `true`, the `id_user` user has answered that he/she will attend the `id_event` event; if it is `false` he/she has answered that he/she is not going to attend the event.

2.2.3. Logical model

We draw below the logical model:

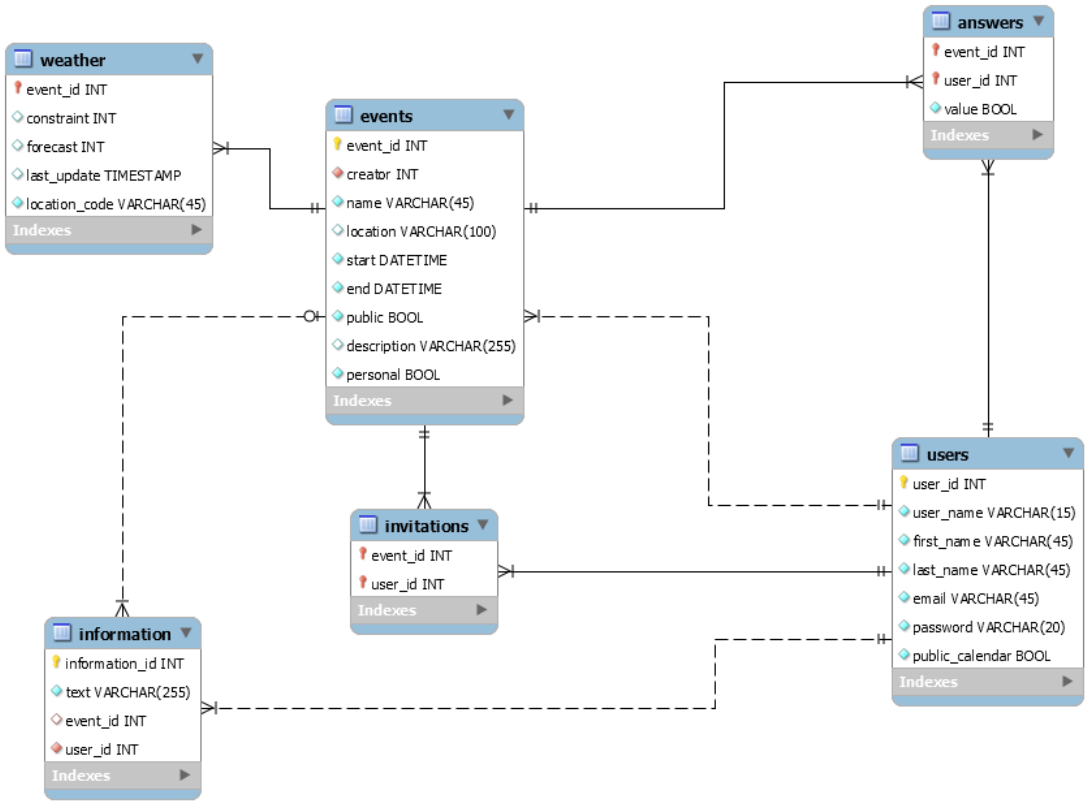

Figure 7 - Final DB.

The final model has the following physical structure:

events (event_id, *creator*, name, location, start, end, public, description, personal);

users (user_id, user_name, first_name, last_name, email, password, public_calendar);

invitations (*event_id*, *user_id*);

information (information_id, text, *event_id*, *user_id*);

answers (*event_id*, *user_id*, value);

weather (*event_id*, constraint, forecast, last_update, location_code).

n.b. we have underlined the primary keys of each table and put in *italic* the *foreign keys* that each table contains.

3. User experience

In this chapter we want to describe the UX (User eXperience) given by our system to its users. We used a class diagram with normal classes and, especially, classes with appropriate stereotypes:

- «`screen`», which represents a single page;
- «`screen compartment`», which represents a part of a page that may be shared with other ones;
- «`input form`», which represents some input fields that can be filled by a user and then submitted to the system by clicking on a button.

To better understand the diagrams, each of the sections below is titled with the name of the functionality represented by the UX diagram drawn.

3.1. Log in, registration and problems

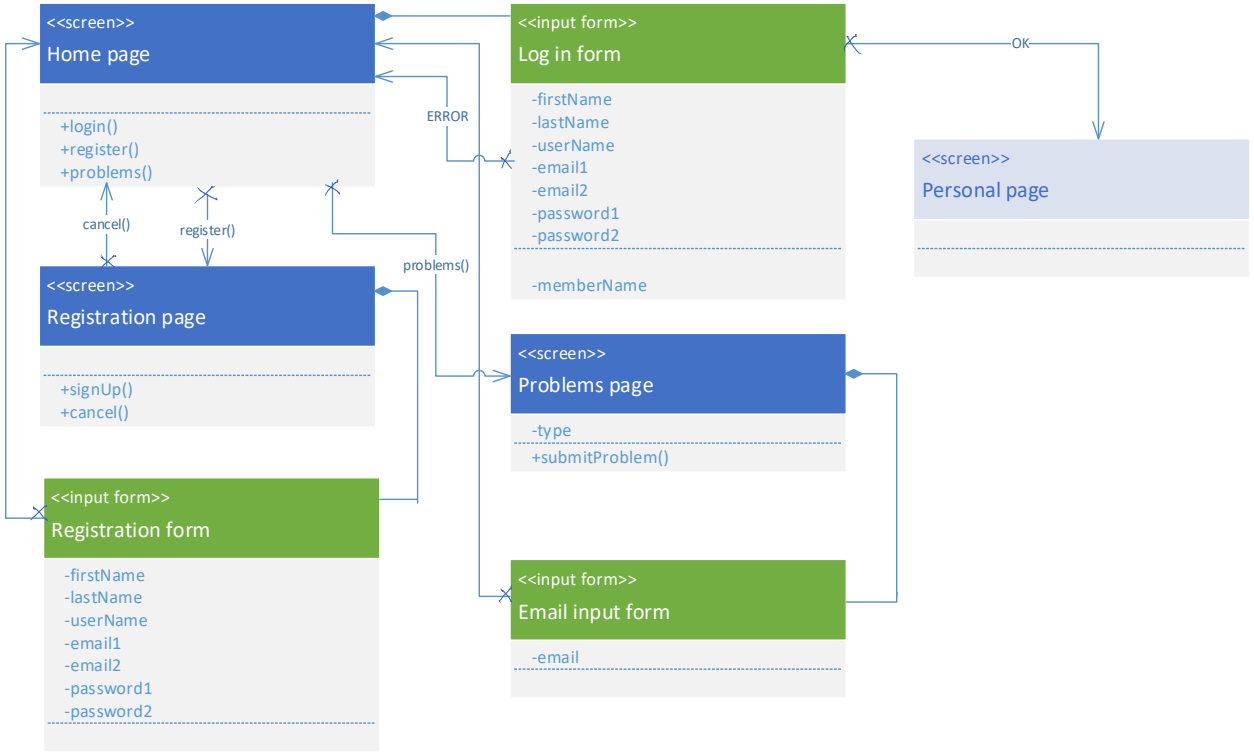

Figure 8 – UX diagram for guests and non-logged users. The `Personal page` class is partial.

From `Home page` a guest can access `Registration page` where he/she can sign up through an input form named `Registration form` providing some information with the purpose of creating a new account.

In addition, it is also possible to log in by providing a user name and a password. If the data are successfully validated, the user's Personal page will be loaded.

When having log in problems, the user can access `Problems page`. There he/she can choose to retrieve his/her password (selected by default, using a radio button) or his/her user name.

3.2. Personal page

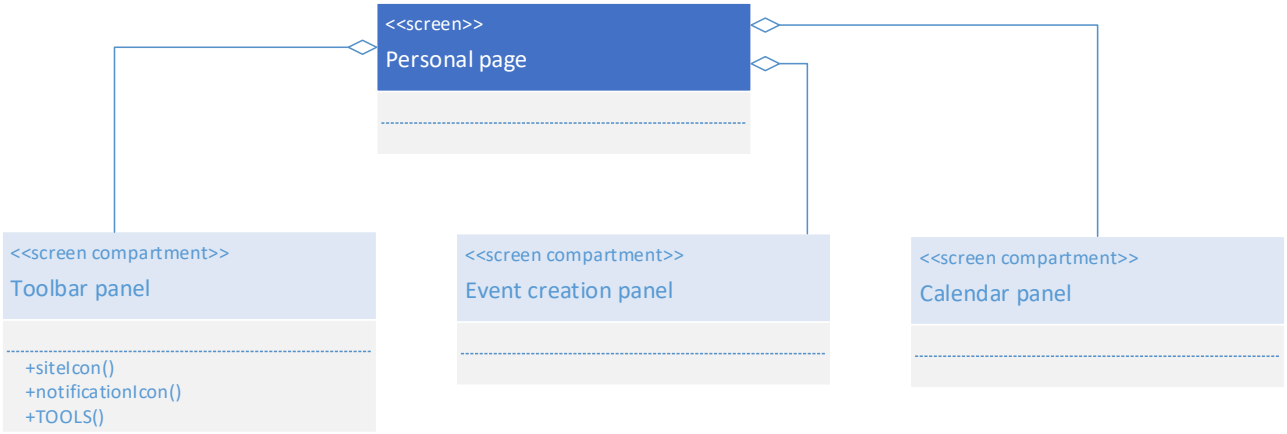

Figure 9 – UX diagram of a high-level view of Personal page. The Toolbar panel, Event creation panel and Calendar panel screen compartments are partial.

The Personal page is composed by three screen compartments which are:

- the toolbar panel;
- the event panel;
- the calendar panel.

3.3. Toolbar panel

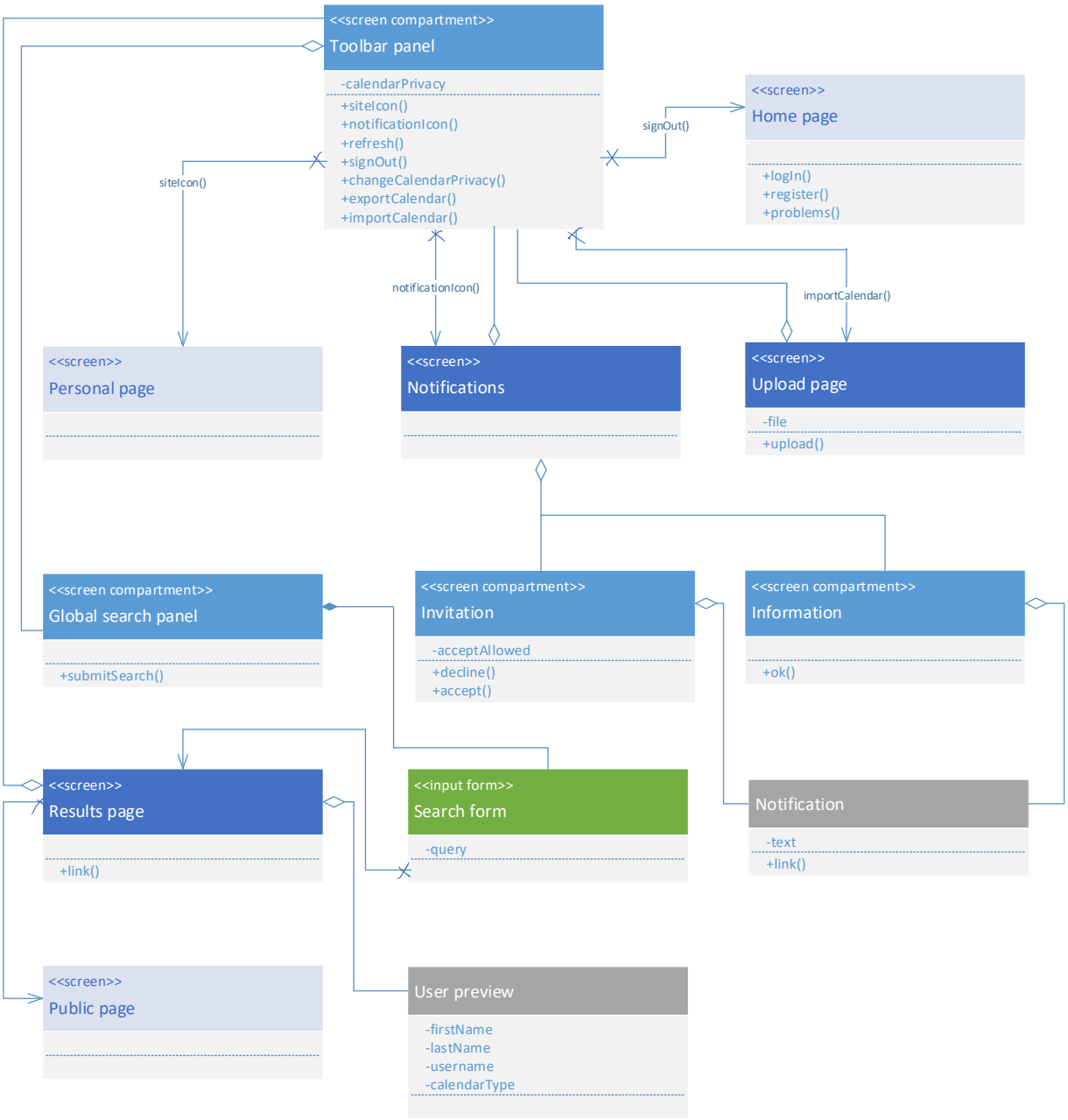

Figure 10 - UX diagram of the toolbar panel. This panel will be present in all the pages loaded to a logged user. In this way, among the other provided functionalities, he/she will always be able to reach his/her personal page. The Home page, Personal page and Public page screens are partial.

From the Toolbar panel, we can access the following functionalities:

- link to the personal page;
- search for people (via Global search panel);
- link to the page of the notifications;
- access to some “tools”.

The tools overlay presents us functionalities like change of the calendar privacy, import or export of the calendar and the refresh of displayed information. From this panel, we can also sign out from the application. This will redirect us to Home page.

If we decide to import a calendar, we are directed to `Upload` page where we can upload a file containing events to our calendar.

We can access notifications (`Invitations` or `Information`) in `Notifications` page. The screen compartment of an `Invitation` will allow us to decline or accept the invitation itself.

We can search for a user using `Global search panel`. The result will be displayed to us in `Result` page, which will display a preview of the user's information and a link to `Public` page of the user himself/herself.

3.4. Public page

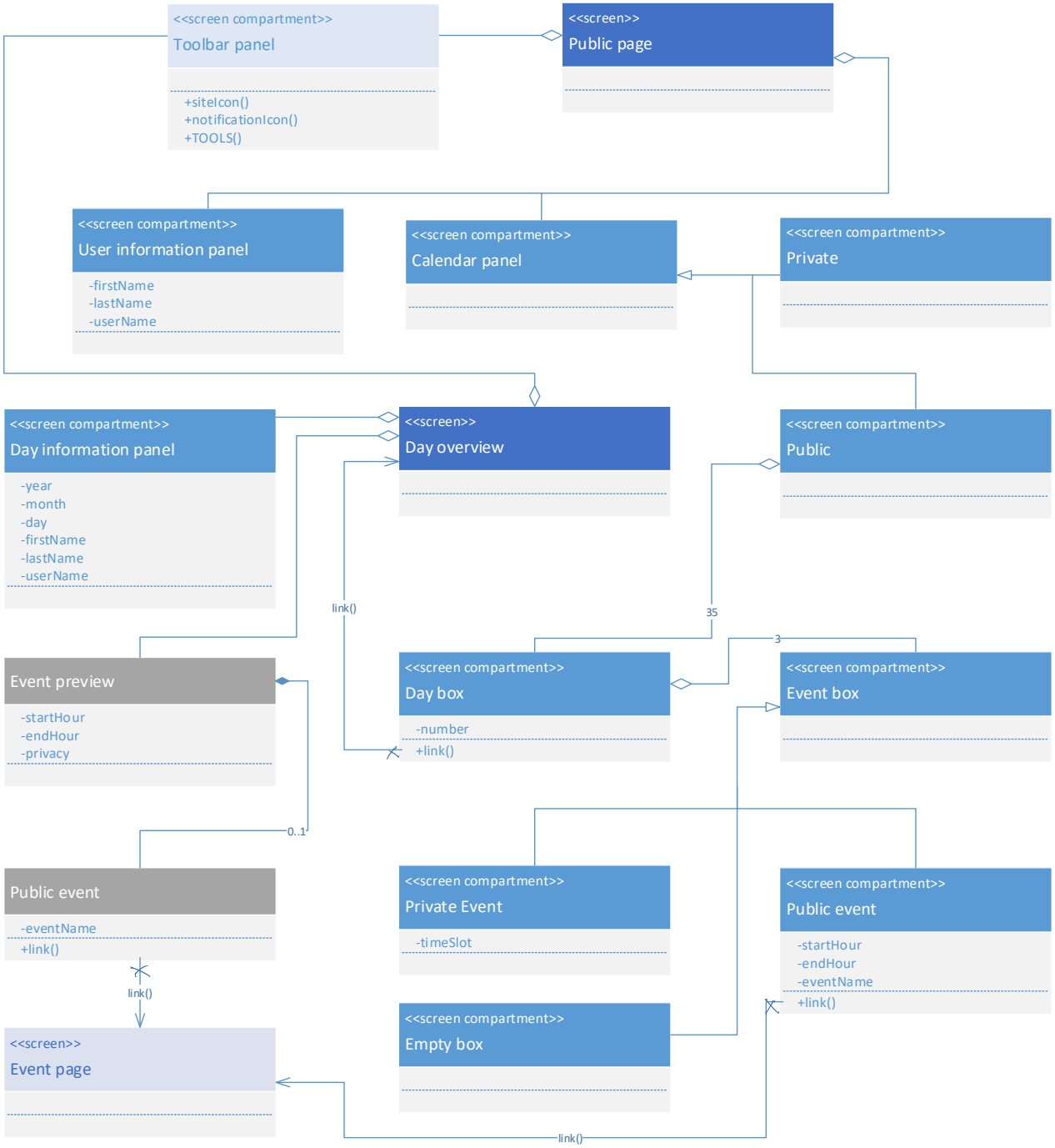

Figure 11 – UX diagram of another user's page. The `Toolbar panel` screen compartment and the `Event page` screen are partial.

User information panel (in the left side) and Calendar panel (in the right side) compose Public page. Depending on the user's privacy settings, the calendar panel can be Private (without an actual calendar displayed) or Public (displaying the public events). The calendar is composed by day boxes (that allow us to open the Day overview screen). The day boxes are themselves composed by a maximum number of three event boxes (which allow us to open the Event page screen).

3.5. Event creation

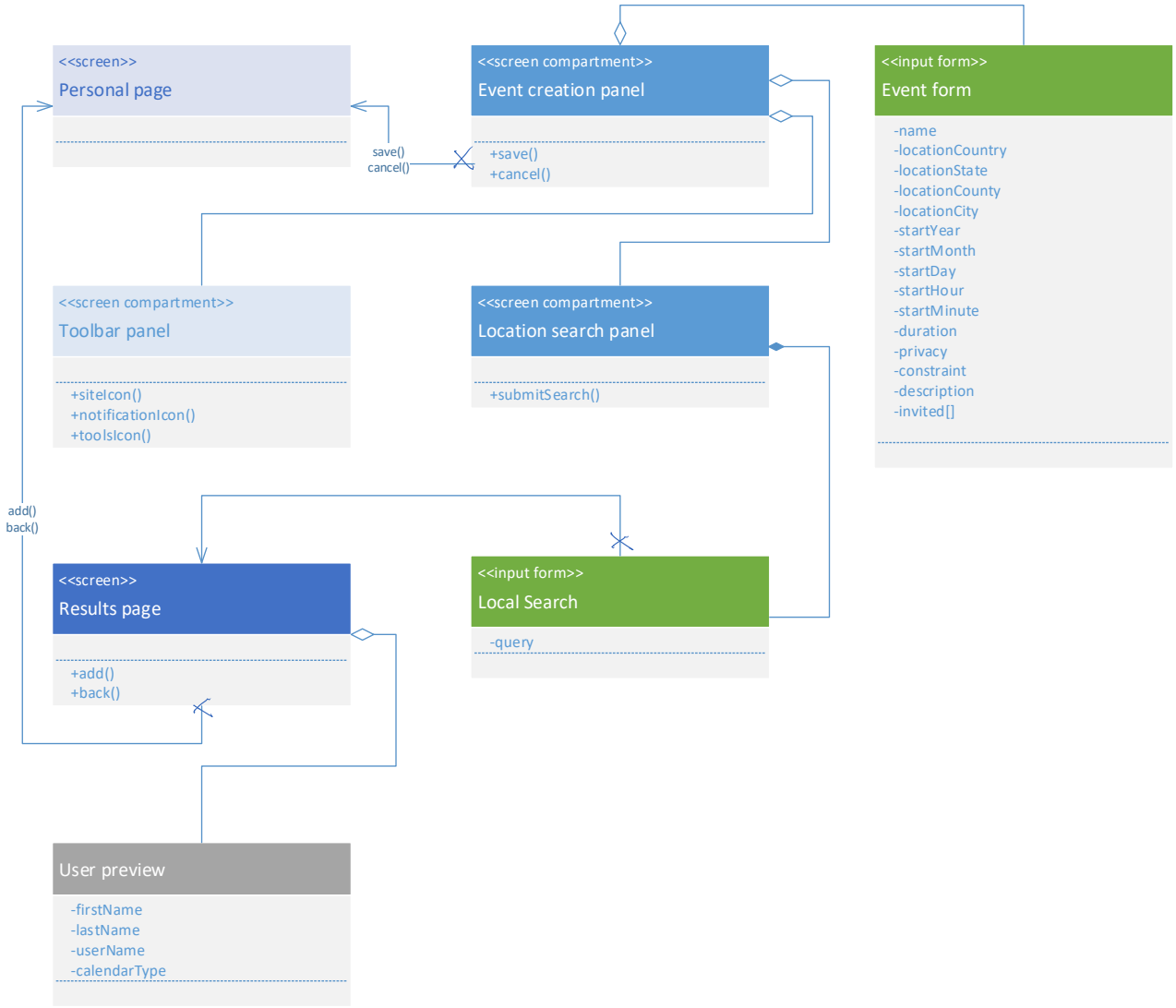

Figure 12 – UX diagram of the left side of the personal page, containing the event creation form. The Personal page screen and the Toolbar panel screen compartment are partial.

We can create an event from this screen compartment completing the required Event form. After having submitted the data (including the eventually searched and invited users), we can try to save them. It is also possible to clear the form using the dedicated button.

3.6. Event details

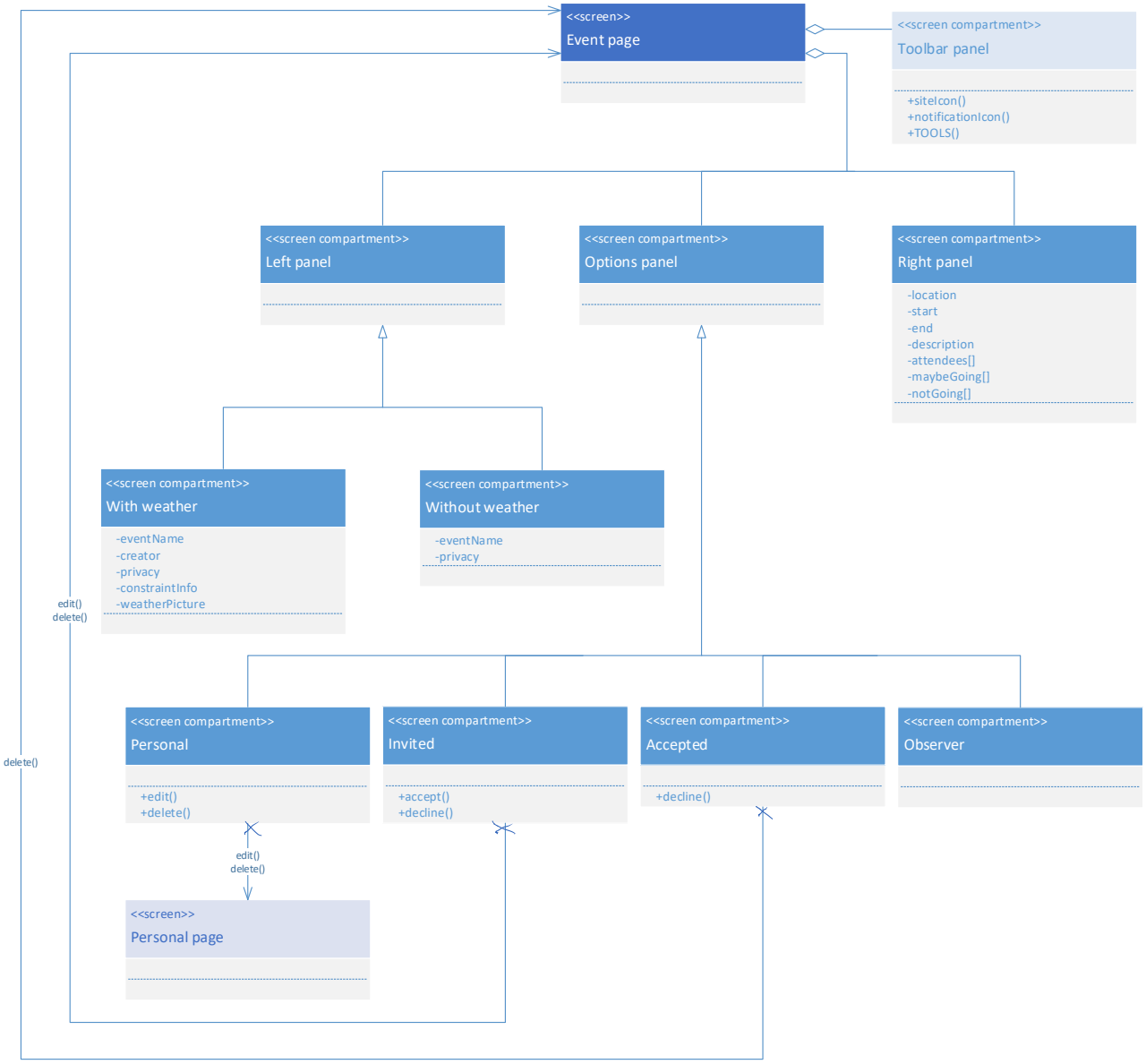

Figure 13 – UX diagram of the page containing the details of an event and the optional actions that the observing user can perform. The `Toolbar panel` screen compartment and the `Personal page` screen are partial.

On `Event page`, we can select an event and view its details, which are spread out on three different screen compartments namely `Left panel`, `Options panel` and `Right panel`.

From `Left panel` we can find the name of the event, its creator and other information (e.g. if it is an event with a particular weather constraint or not). From `Right panel` we can see all the other details (i.e. location, attendees, maybe going users, not going users, date and description).

`Options panel`, (that on the screen is located on the left side, under the `Left panel`), can be of four different types:

- `Personal`, when the displayed event is created by the user that is browsing the page (so he/she will be able to edit or delete it);
- `Invited`, when the user that is browsing the page has a pending invitation for that event (so he/she will be able to accept or decline the invitation);
- `Accepted`, when the user that is browsing the page has previously accepted the invitation (so he/she will be able to decline it);
- `Observer`, when the user that is browsing the page is only looking a public event not related to him/her.

3.7. Day overview

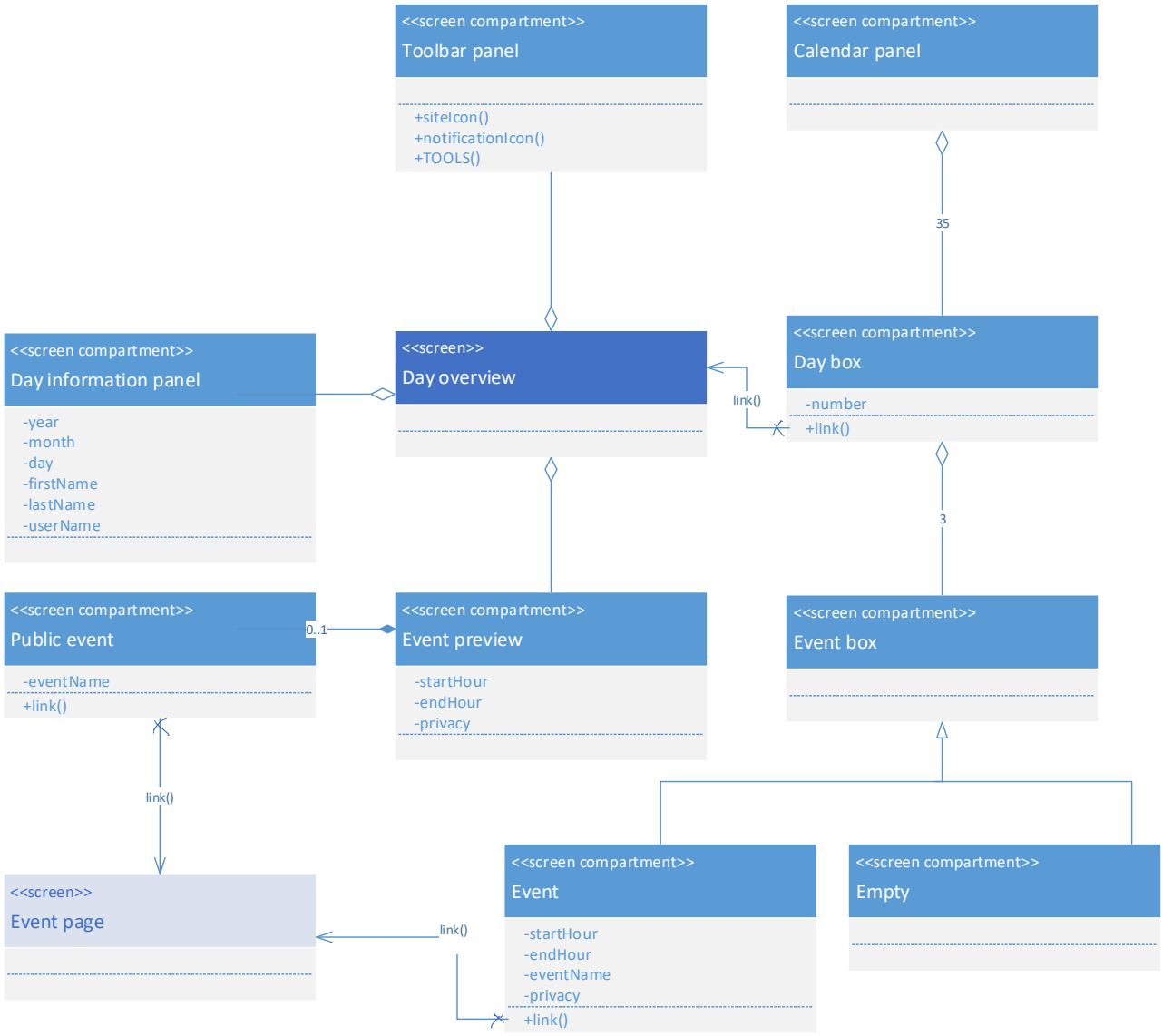

Figure 14 – UX diagram of a Day overview page, accessed from a Calendar panel screen compartment. The Event page screen is partial.

From the Calendar panel screen compartment, we can view information about a day from Day Box that allow us to open the Day overview screen and the Day information panel containing detailed information about the selected day. In Day overview, we will be able to see a preview of all the events scheduled for that day and we can open the respective Event page (which can also be reached from the – non-empty – Event box screen compartment).

4. BCE diagrams

4.1. Entity overview

All the entities of the BCE model will be presented in a very fragmented way, to prevent the diagrams to be too chaotic. For this reason, we provide an entity overview, in such a way that the reader can always refer to this diagram.

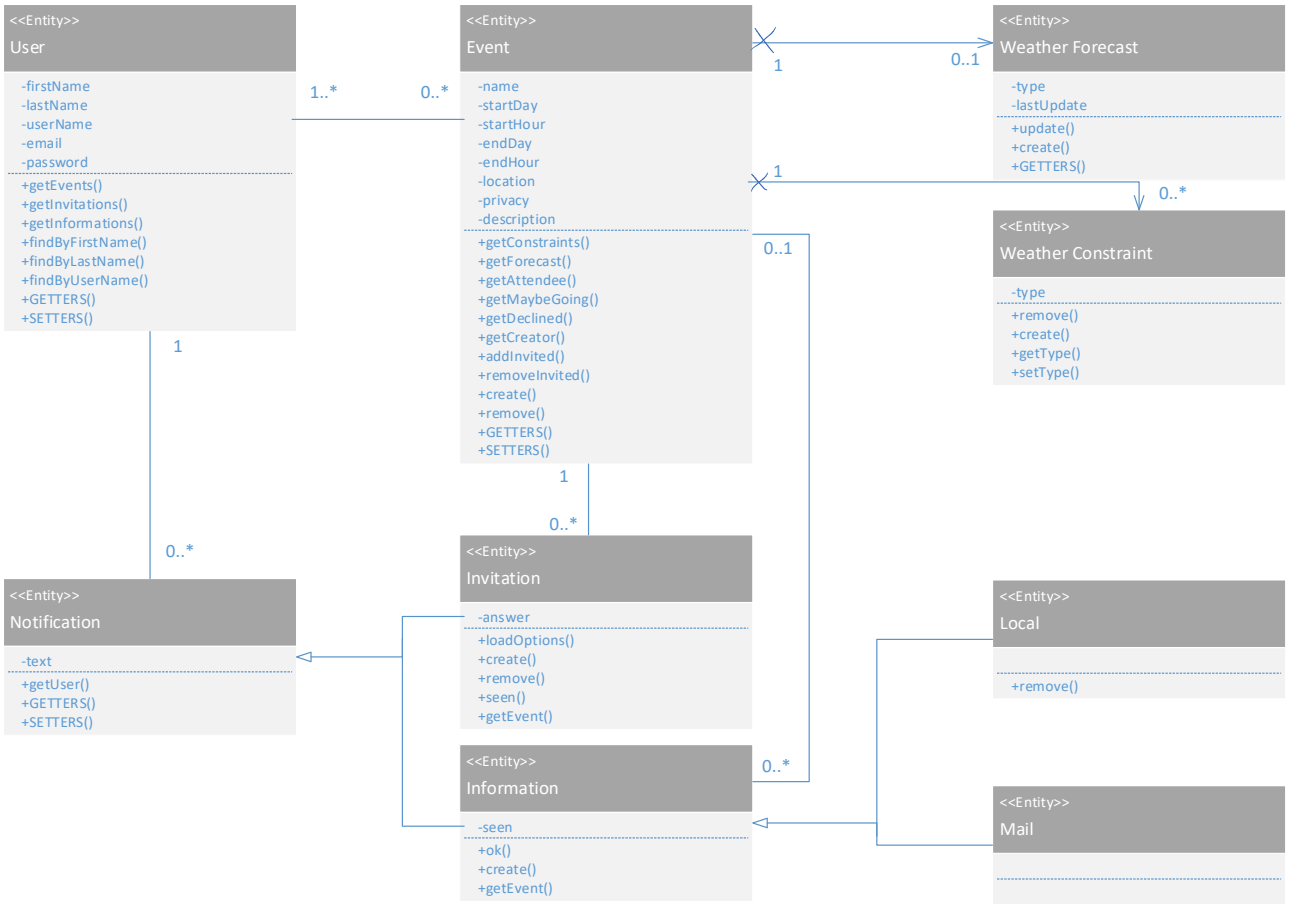

Figure 15 – Entity overview. It represents the conceptual and – therefore – abstracted perspective of the data structures of the system.

n.b. It is important to say that the entities that do not represent the ER diagram of the database, but they are only a conceptual view.

e.g. the `invitation` entity is an high level view of both the `notifications` and `answers` tables of the ER diagram presented in *Subsection 2.2.3 – Logical model*. This is because the implemented logic-to-be will ensure a strict behavior when answering a notification (i.e. at first the user receive a notification, then as soon as his/her answer will be put into the `answer` table the actual invitation will be deleted). Implementing it into the flow of the BCE diagram would reduce the readability of the diagram itself, without a tangible gain in the rightness of the explanation.

n.b. We have simplified the connections between classes by showing only the relevant ones under the given perspective and we have also omitted to specify every entity connection for the subsequent diagrams. The dashed line will mean that there is a non specified connection between the given classes that involves other classes, which are not written because not relevant for the considered perspective. For the actual connections, please refer to *Figure 15 – Entity overview*.

4.2. Log in, registration and problems

The basic functionalities that can be exploited by a guest and a non-logged user are represented by three boundaries:

- `HomePage`;
- `RegistrationPage`;
- `ProblemsPage`.

The description of the controllers used between these boundaries are as follows:

- **`HomePageLoader`**: This control generates `HomePage`, which contains the login form, it generates `RegistrationPage` when a signup is required and it generates the problem page in the case of a login problem.
- **`IssuesDataManager`**: This control handles the verification of the data submitted from `ProblemsPage` before giving the requested support.
- **`ProfileDataManager`**: This control, like the previous one, verifies the data submitted from `RegistrationPage` and uses this data if correct to create a new user.
- **`LogInManager`**: This control checks the login fields to verify if they have been correctly completed then loads the user page for the user information provided; it also logs a user out of the current session.

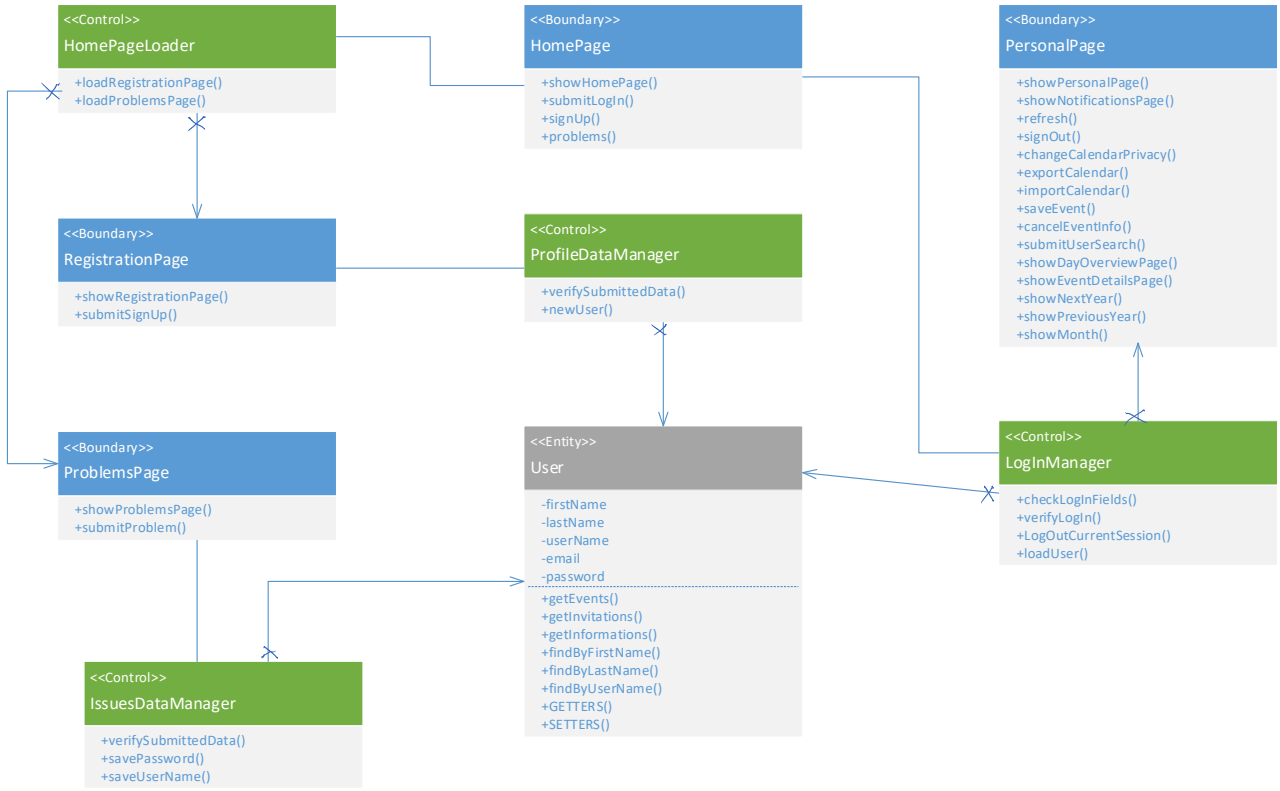

Figure 16 – BCE diagram of Log in, registration and problems.

4.3. Personal page

In this diagram, we have the `PersonalPage` boundary, representing the functionalities available to a logged user.

The controllers used between this boundary and various entities are as follows:

- **`UserDataLoader`**: This control loads the profile and notifications of the specific user.
- **`DataLoader`**: This control is responsible for loading the details of the events and the search results; it also loads the calendar information like the year, a specific month, events, and the overview of a selected day.
- **`EventCreationManager`**: As the name suggests, this control handles the creation of a new event. At first, it validates its consistency by verifying the submitted data before the actual creation. Successively, it sends out invitations to the guests of this event.
- **`EventManager`**: This control is responsible for managing the newly arrived invitations and/or information about an event, it also handles accepting, declining, and revoking of invitations. It also checks for the weather forecast of a scheduled event.

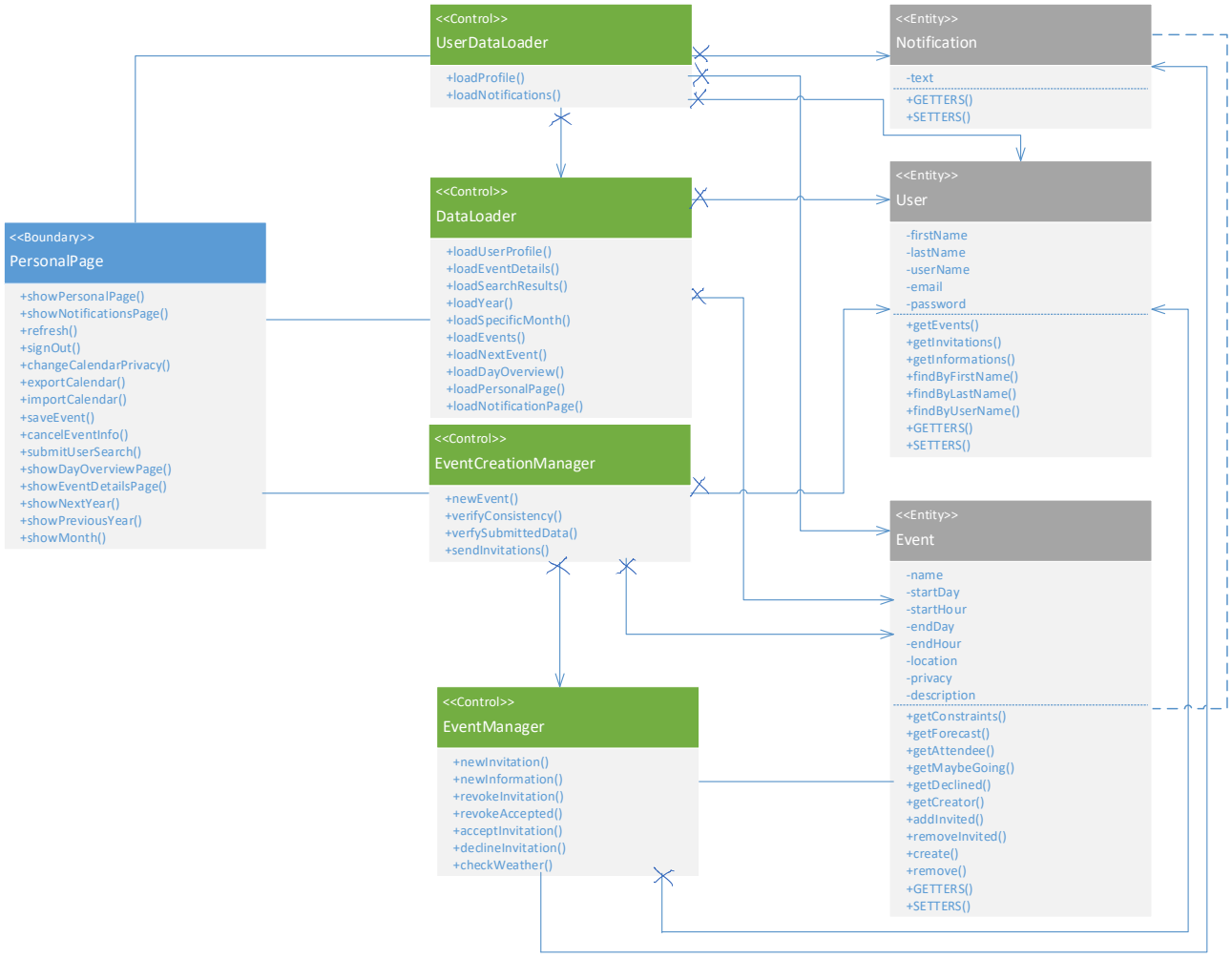

Figure 17 – BCE diagram of the personal page boundary.

4.4. Calendar perspective

The `UploadPage` boundary represents some calendar functionalities that a user can exploit from his/her `PersonalPage` boundary.

There is one control used between these boundaries, it is as follows:

- **`UserCalendarManager`**: This control handles the download and upload of calendars. It verifies the files before uploading them and toggles the privacy of the events in a calendar, when requested.

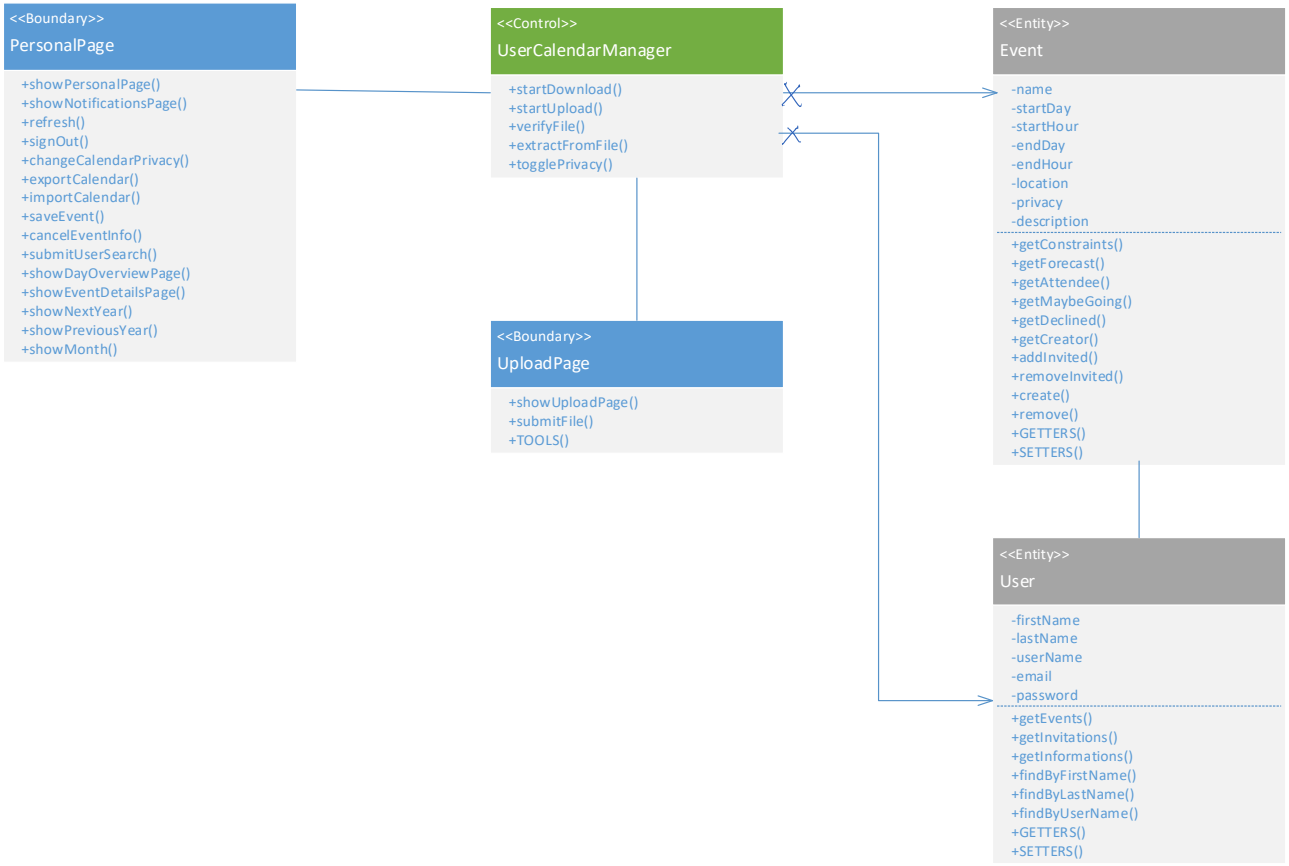

Figure 18 – BCE diagram in a calendar perspective.

4.5. Notifications perspective

The `NotificationsPage` boundary represents the basic notification functionalities a user can exploit from the `PersonalPage` boundary.

The controllers used in between these boundaries are `UserDataLoader`, `DataLoader` and `EventManager`, these controllers have been described in *Section 4.3 – Personal page*.

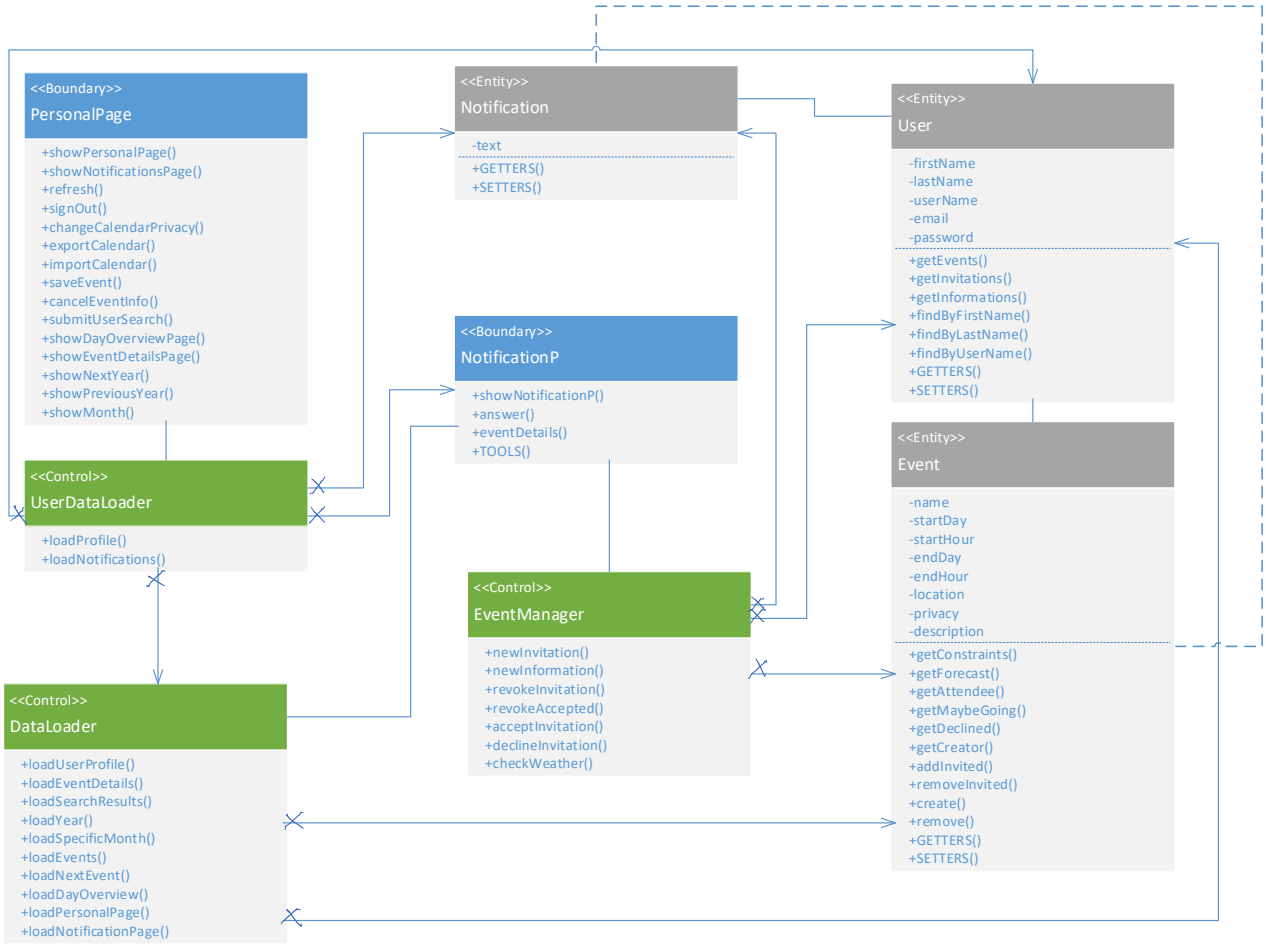

Figure 19 – BCE diagram in a notifications perspective.

4.6. Search procedure and user's profile browsing

The `ResultPage` boundary and `UserProfile` boundary represent the user search functionalities from the `PersonalPage`; the `UserProfile` boundary in this diagram represents the user profile of another user.

The `DataLoader` control used between these boundaries has already been described in *Section 4.3 – Personal page*.

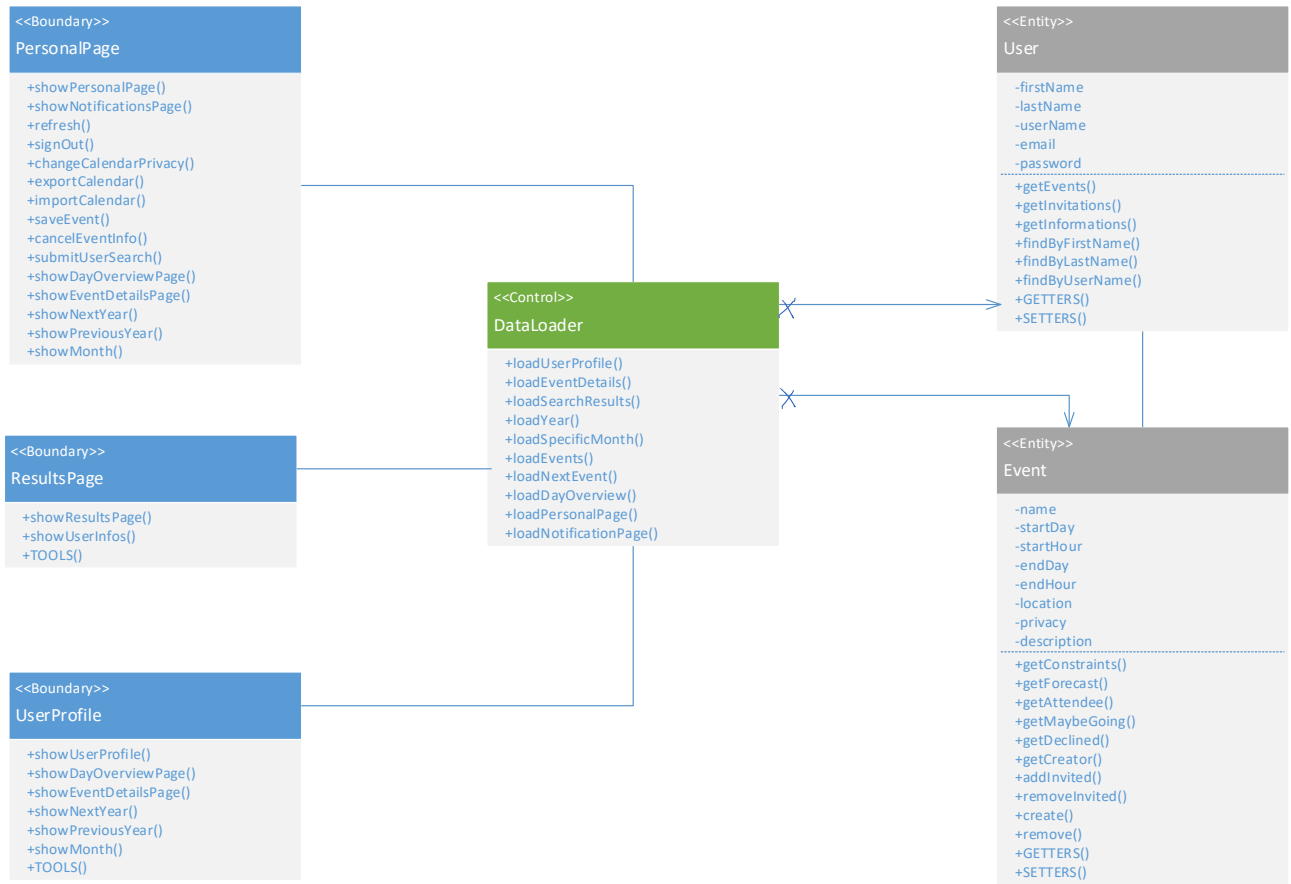

Figure 20 – BCE diagram of the search procedure and – another – user's profile browsing.

4.7. Day overview

The `DayOverview` boundary represent the basic functionalities of the day on a calendar found on the `PersonalPage` boundary. The `UserProfile` boundary in this diagram represents the user profile of another user.

The `DataLoader` control used between these boundaries has already been described in *Section 4.3 – Personal page*.

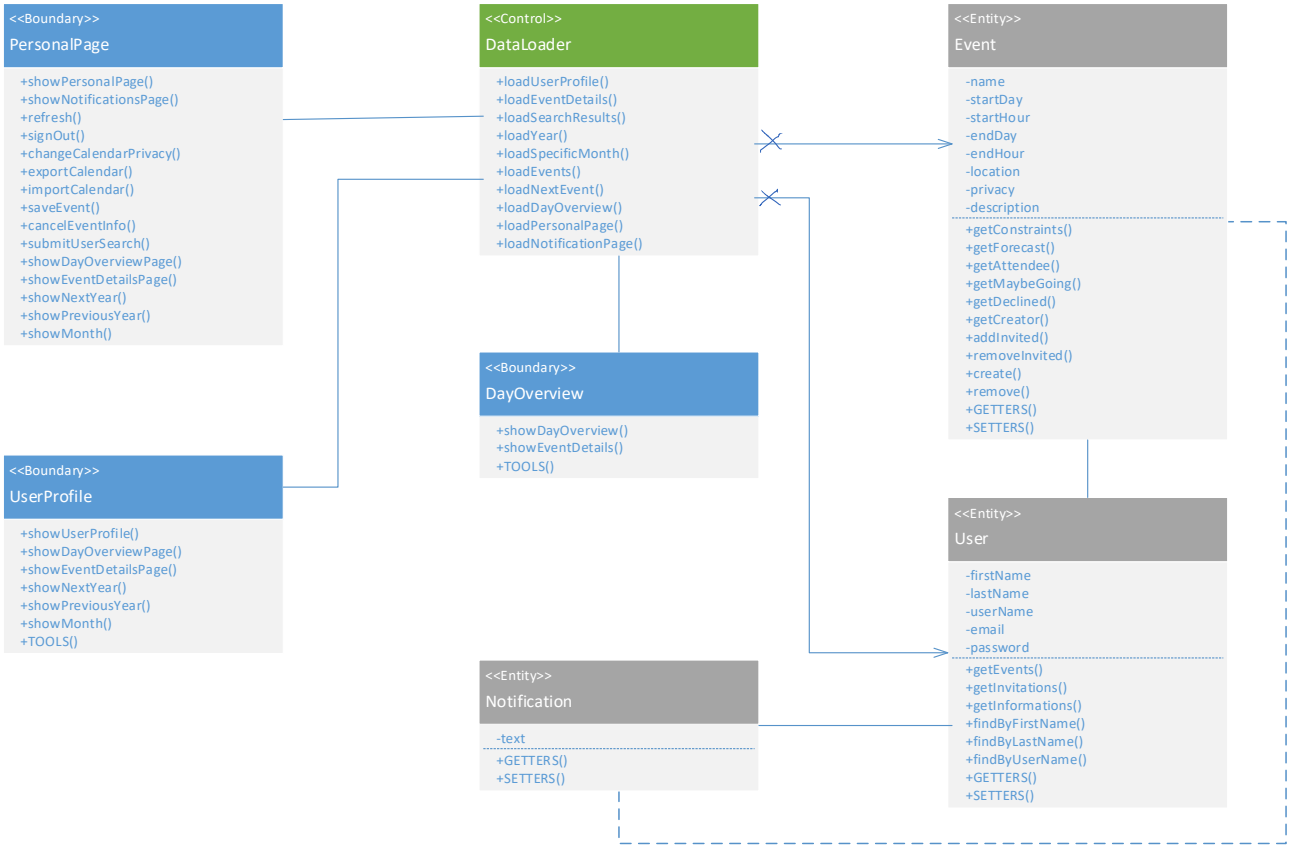

Figure 21 – BCE diagram of the day overview boundary.

4.8. Event details

The `EventDetails` boundary represent the basic functionalities of an event found on the `PersonalPage` boundary.

The `DataLoader` and `EventManager` controllers used between these boundaries have already been described in *Section 4.3 – Personal page*.

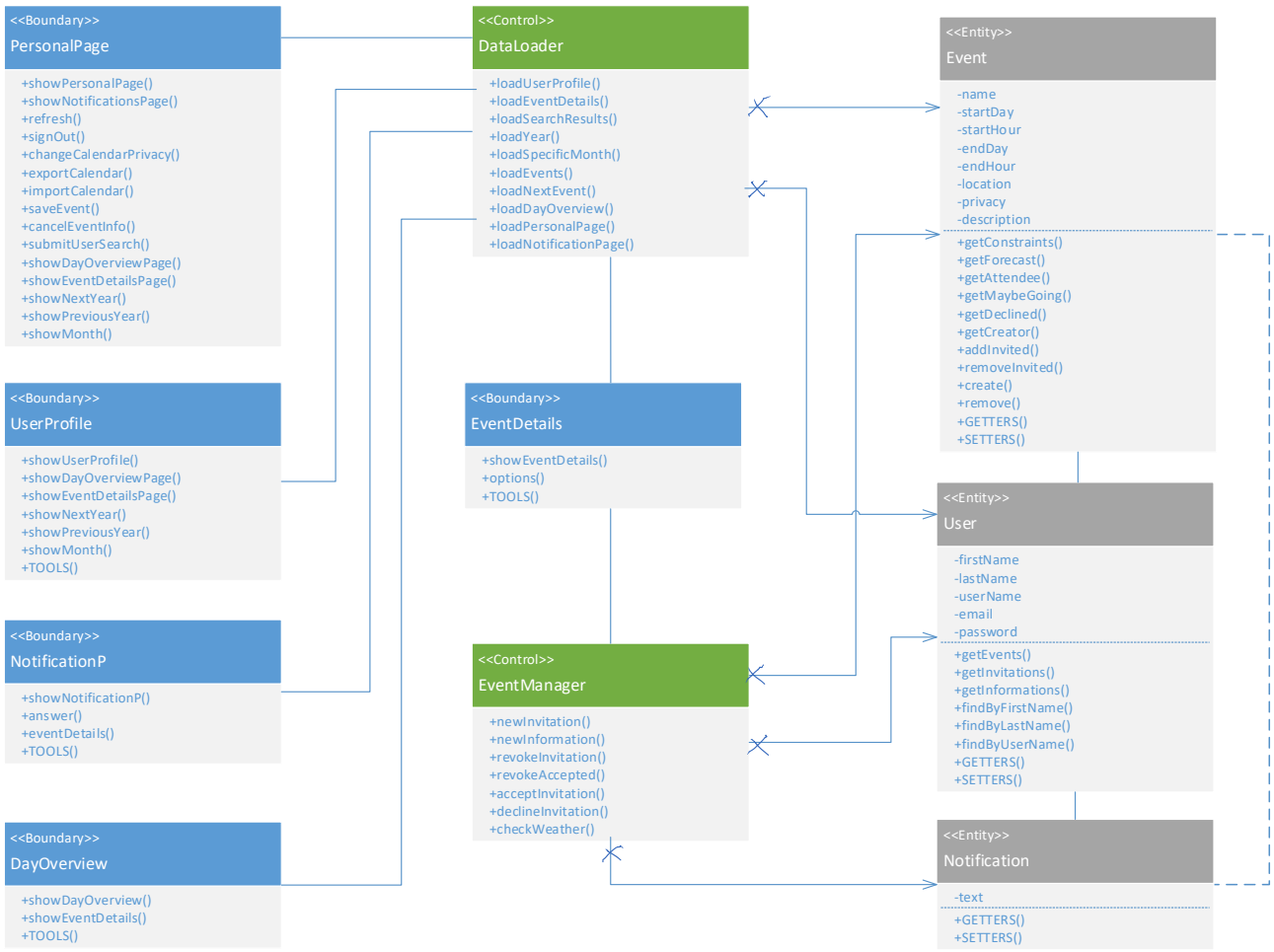

Figure 22 - BCE diagram of the event details boundary.

5. Sequence diagrams

We provide some sequence diagrams to let the reader understand the BCE diagrams (Please refer to *Chapter 4 – BCE diagrams*) in a more practical way. All the methods used are in the BCE boundaries, controls and entities.

5.1. Log in

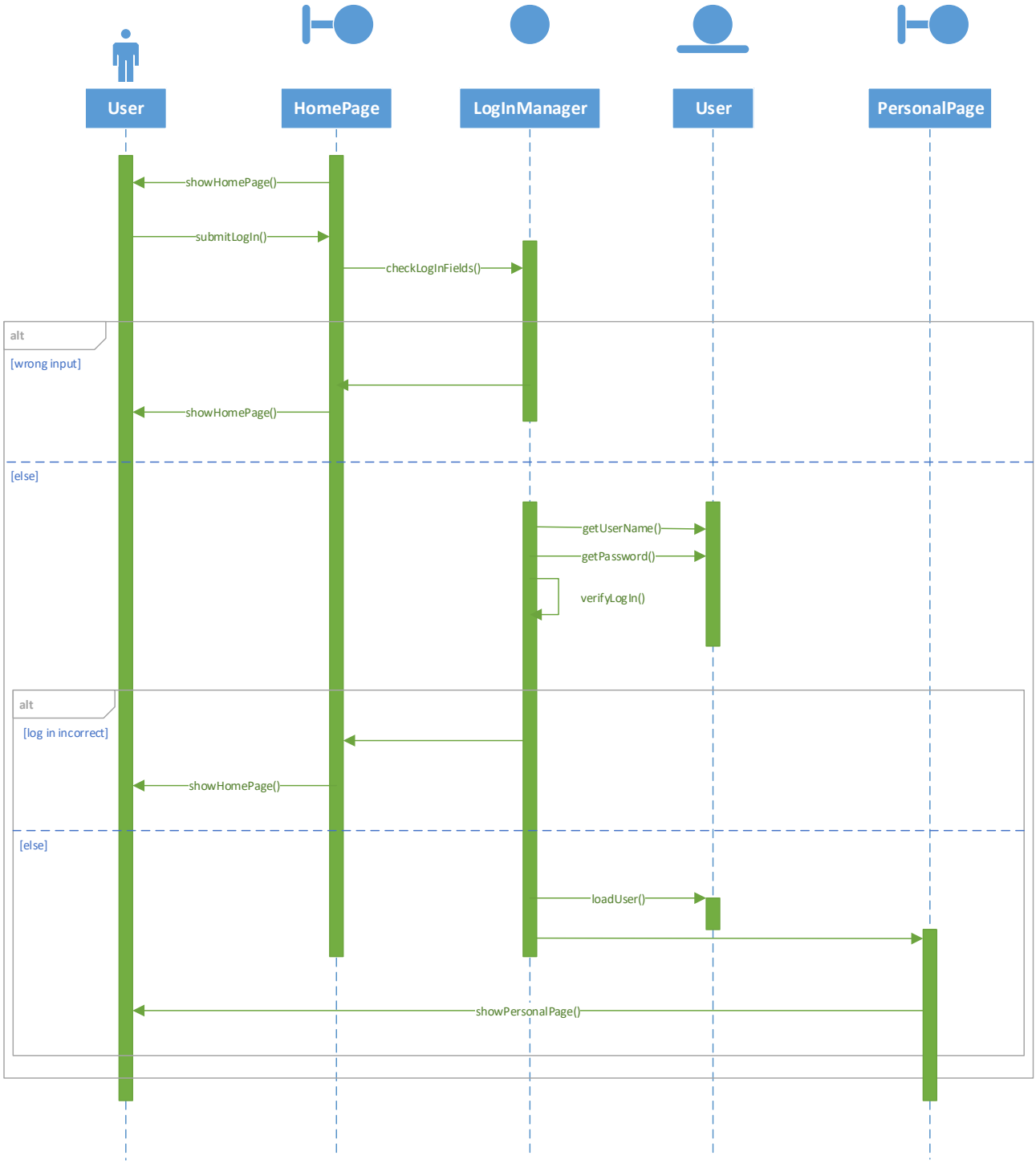

Figure 23 - A user logs into the system by providing the required credentials. If the log in form is properly filled, the system checks the values and will redirect the user to his/her own `PersonalPage` page, otherwise the user will be asked to re-fill the log in form.

5.2. Search for a user

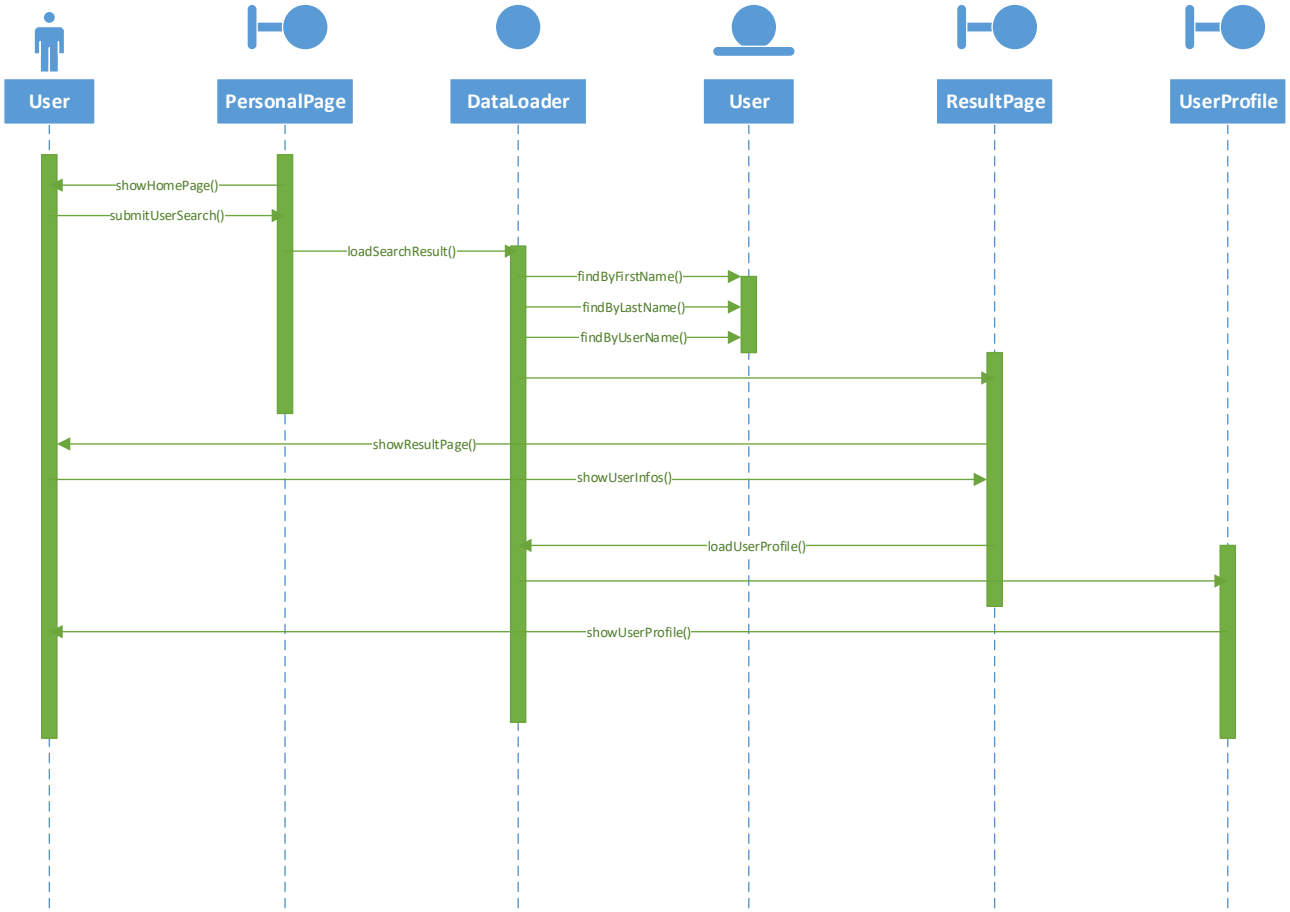

Figure 24 – A user can search for other users by typing their complete first name, last name or user name into the search bar, which is located in the toolbar. In this diagram, the user start his search from his/her `PersonalPage` page. The system will handle the request and show a list of results in the `ResultPage` page. If the user clicks on an element of that list, he/she will be redirected to the `UserProfile` page of that user.

5.3. Answer to an invitation

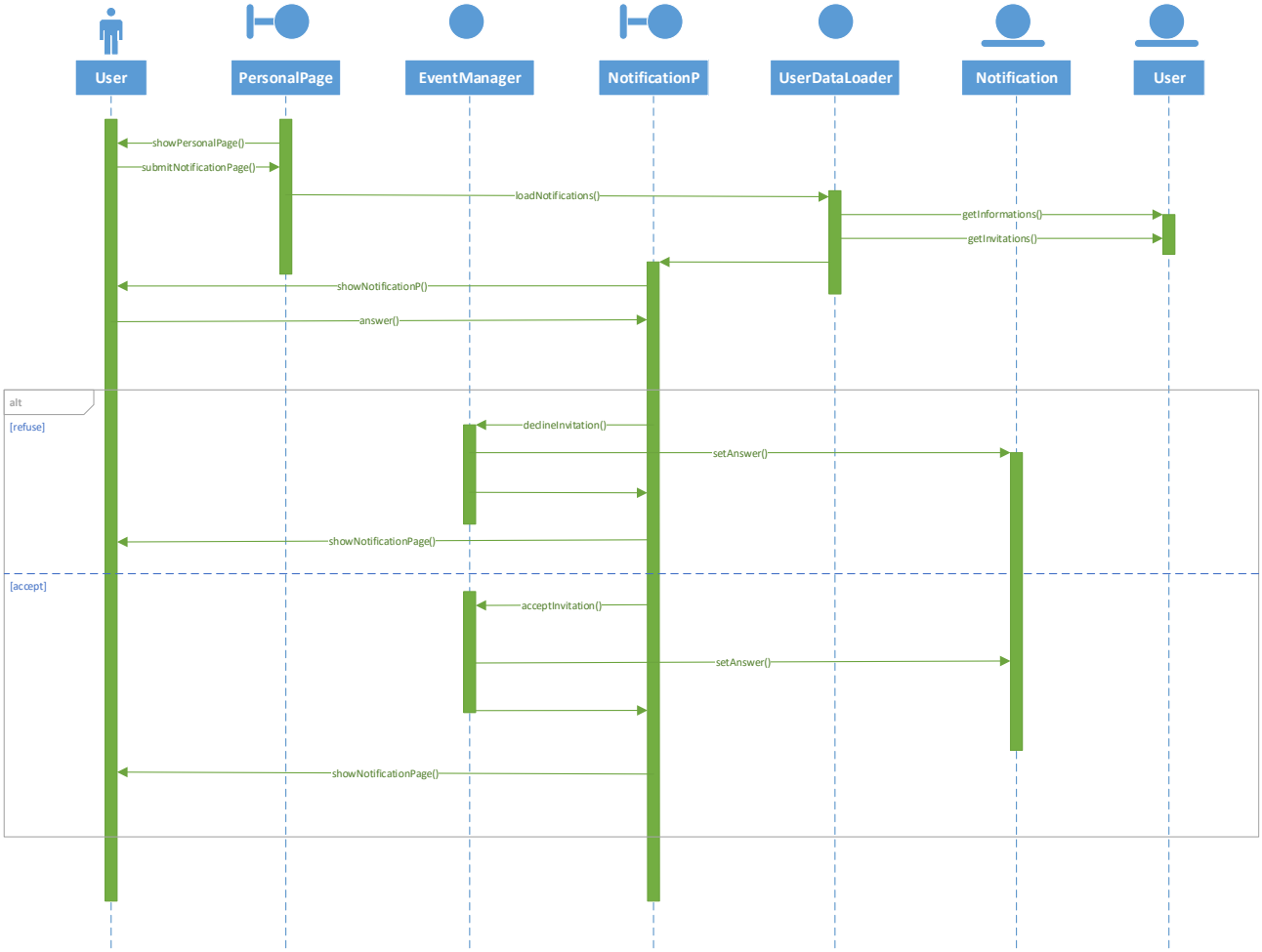

Figure 25 – When a user receives an invitation, he/she will be able to accept it only if he/she has no other events that overlap with such invitation. If the user accepts an invitation, he/she will be added to the attendee list and the event will be added to his/her calendar.

5.4. Exporting a calendar

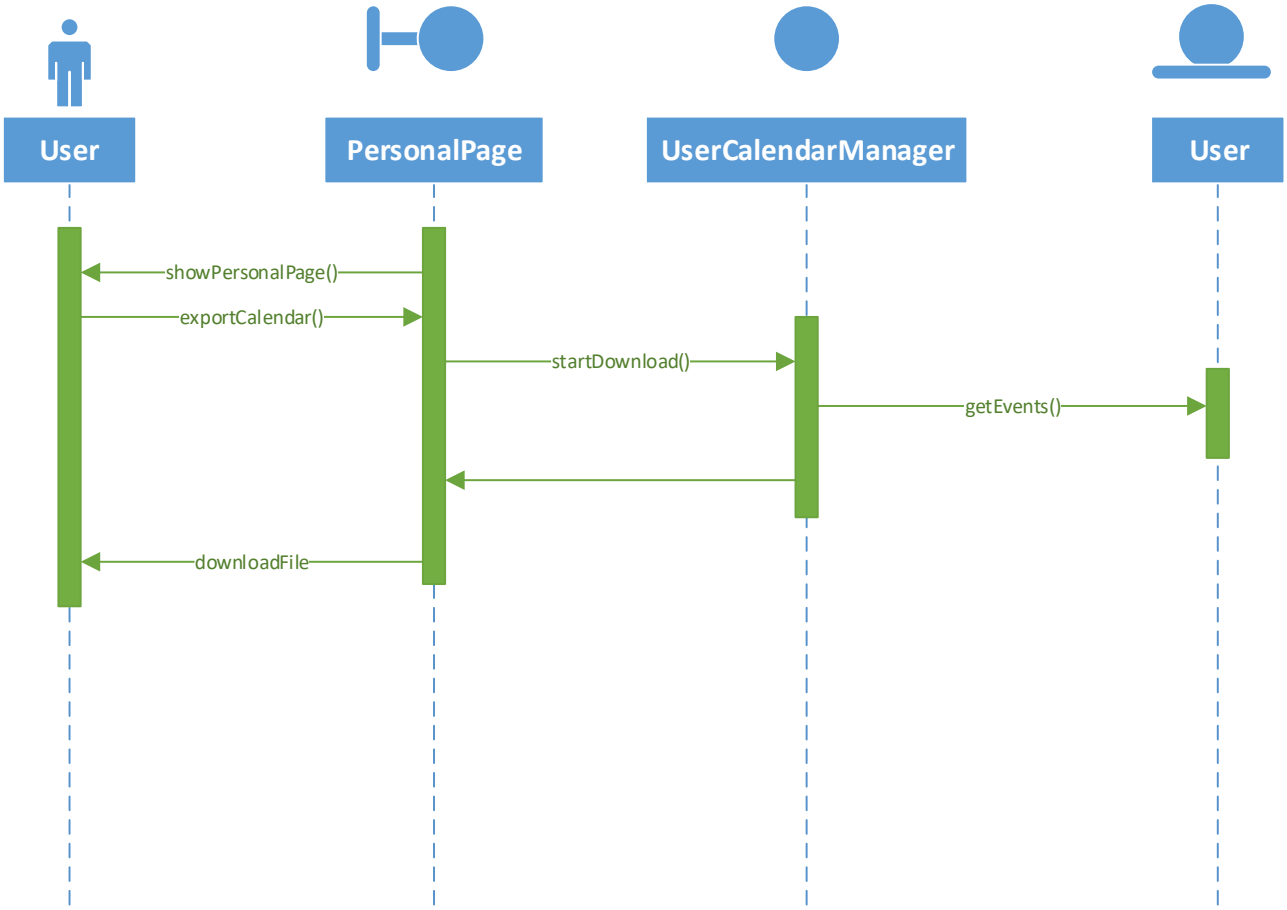

Figure 26 – In this diagram, we can see how the system provides a file containing all the upcoming events of the user. `downloadFile` means that the system has sent the file to be exported to the user's browser.

5.5. Importing a calendar

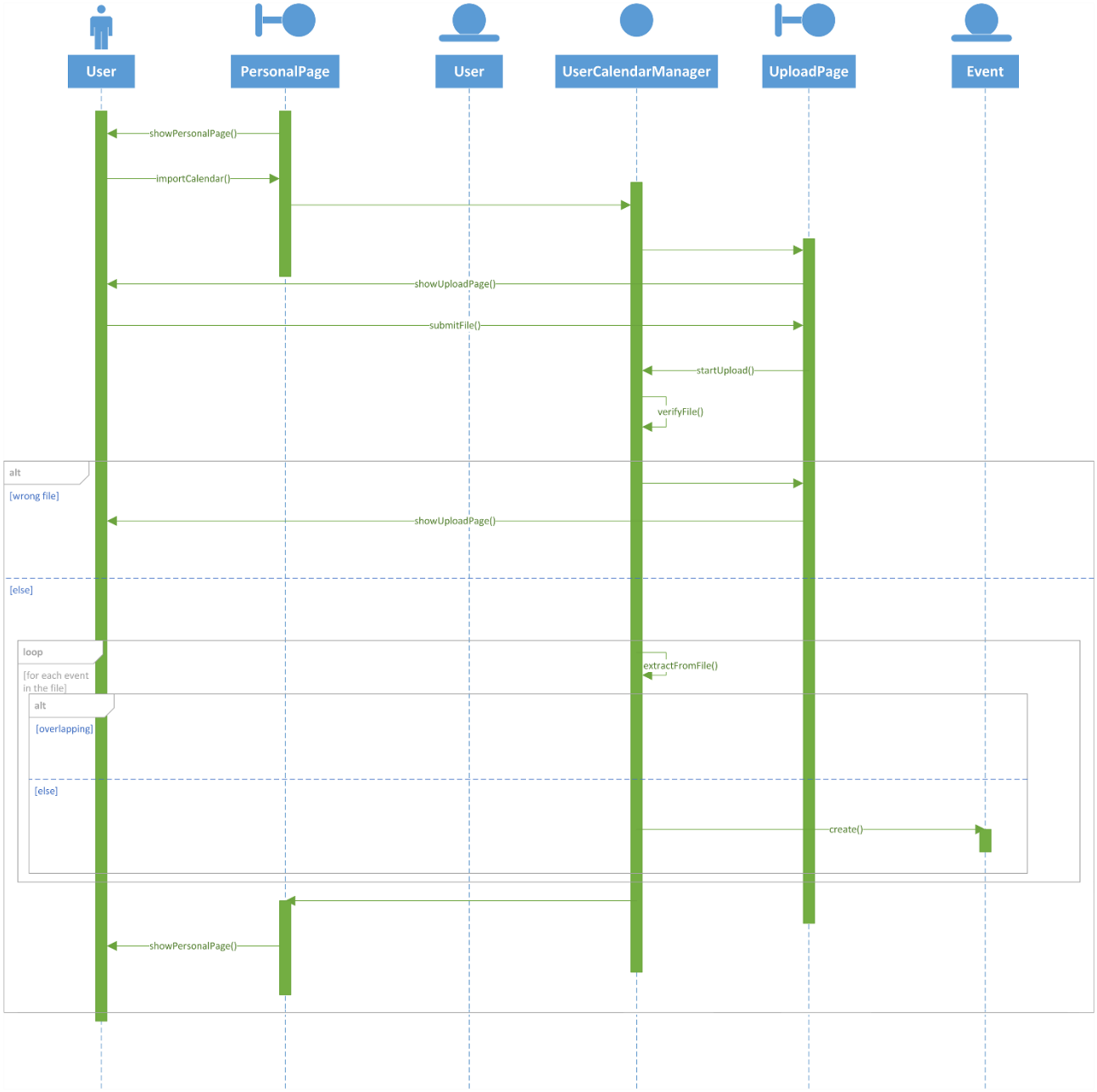

Figure 27 – The user can import a file that was previously exported. It is important to know that only – personal – events that do not overlap with the existing ones will be added to the calendar.

5.6. Event details

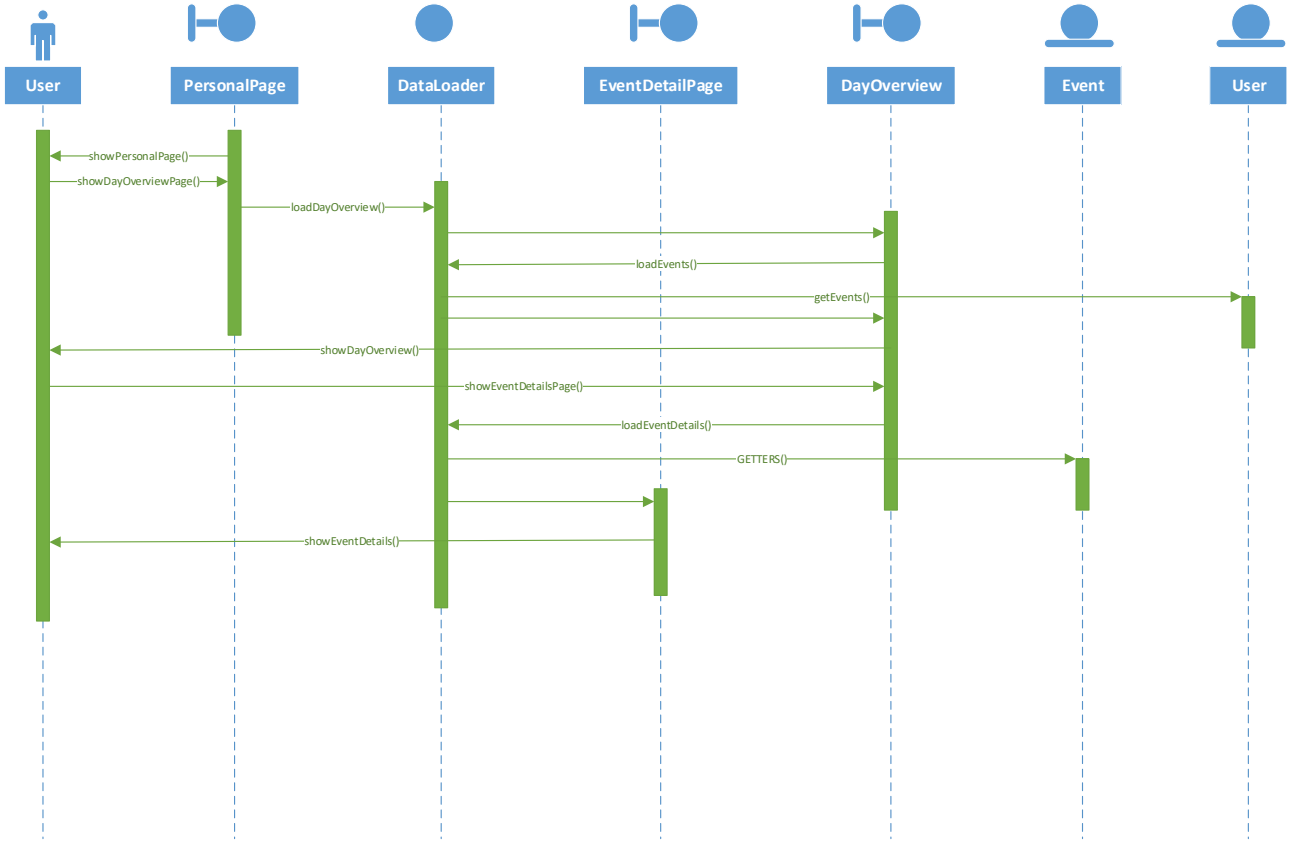

Figure 28 – In this diagram, we can see how a user can access the details of an event. To increase its readability we have collected all the methods to get the information of the event into a single conceptual GETTERS( ) function.

6. Final considerations

From this moment on, we have a very good understanding of how our project will be built.

In this document, where applicable, we decided to draw diagrams in a conceptual way, in order to give a better overall view instead of fully detailed, but – inevitably – messy diagrams (that would not have concretely brought improvements). Accordingly, we chosen to draw User Experience and BCE diagrams instead of more technical ones.

Appendix – Revision history

Initial release:	v1.0
Current release:	v1.01
Date of the last review:	2014/12/09

v1.0

2014/11/07 – Initial document.

v1.01

2014/12/09 – Revised document (removed braces placeholders in subsection 2.2.2. and in chapter 3; changed fonts in part of the text in section 4.1; fixed bad sentence in section 4.3: from "As the name implies, this control handles the creation of a new event by verifying the consistency of the event, verifying the submitted data before creating the event and sending out invitations to the guests of this event." to "As the name suggests, this control handles the creation of a new event. At first, it validates its consistency by verifying the submitted data before the actual creation. Successively, it sends out invitations to the guests of this event"; fixed bad sentence in section 4.4: from "This control handles the download and upload of calendars between users." to "This control handles the download and upload of calendars.").