OSU Student Meetup App

Deliverable #4 - *Dave Holmes, Zach Knickerbocker*

For this deliverable, we implemented two use-cases defined in previous deliverables to enable users to actually interact with the application was conceptualized. The use cases in particular that were implemented are:

1. Users can **create events** with criteria including event name, description, start time, end time, location, category, and participation threshold (e.g. 10 people must attend this event, otherwise it should be canceled).

2. Users can **view existing events in a calendar-like interface** which organizes events by their start time, and provides basic information about each such that the user can make an informed selection of interesting events.

The technologies used to implement these ideas are those specified by the deliverable guidelines: EJB for logic and JSF for server-side templating running on top of a JBOSS server with an H2 database backing the application. Since this application implements just a select few use cases, no user sessions are managed. That is, there are no user logins, no tying of events to specific users, etc. We implemented strictly event-specific use cases.

Code Walkthrough

Our application is presently built from two packages: MeetupBeans (EJBs) and MeetupUI (Logic & JSF templates).

MeetupBeans

Event.java

Created upon retrieving an event from the db. Destroyed by garbage collector.

The Entity Bean that represents an Event in the database.

EventDetails.java

Created manually (i.e. via "new") to convey event information to a client application. Destroyed by garbage collector.

Representation of an event which contains getters and setters for managing the name, description, etc. of a given event. We decided to include such a class to enable passing a data structure containing event information to a client-space program (i.e. MeetupUI). This method is preferable to passing on Event objects to the client-space because it holds up the isolation of client-side logic and the database. We don't want to expose the ORM layer to the client. This is also better than simply passing Strings or primitives in a list because it enables additional functionality in the client-space like converting values for years, months, and days to an equivalent UNIX timestamp. Such functionality is implemented in the overloaded method setStartTime() and setEndTime().

EventService.java & EventFinder.java

Created via an @EJB injection, and persists as long as it is being used (e.g.

created by another EJB, and continues to exist for as long as that client servlet is in use).
Controls for managing events and finding them. We decided to separate the logic of managing events (e.g. creating, deleting) and the logic for finding events. While not necessarily something that needed to be done for this particular project, in the future, as the application needs to support more and more functionality for a given event, we thought it best that methods designed to find events (rather than manipulate them) are kept separate for the sake of shorter and more cohesive classes.

CalendarService.java
Created via an @EJB injection, and persists as long as it is being used (e.g. created by a servlet, and continues to exist for as long as that servlet exists).
A controller related to the calendar. Functions provided within generally provide functionality for retrieving events relevant to a given calendar display. For the purposes of our application, all this controller winds up doing is routing calls to other controllers (e.g. EventFinder). Even though this means more code, it keeps a client down the road from using event-related objects, when in reality it should be community with calendar-related objects. This approach could, in the future, enable added calendar-specific functionality on top of a simple call like retrieving events from a specific date. Such functionality might include pruning events which the user is already registered for (i.e. use EventFinder to get events from a given date, then remove events from the list which the user is registered for).

MeetupUI

Calendar.java (logic)

Created upon an HTTP request (e.g. page load, AJAX request), and destroyed once the request is answered.

Logic to accompany a JSF page for displaying events in a calendar-like interface. Specifically, functionality is provided to retrieve a list of events by year, month, and day. Lists are also created and initialized to populate select boxes containing months and days, which in turn enable a user to navigate the calendar.

CreateEvent.java (logic)

Created upon an HTTP request (e.g. page load, AJAX request), and destroyed once the request is answered.

Logic to accompany a JSF page for creating events. Provides getters and setters for the properties of an event, and a create() method for actually creating the event. The fields of this class are referenced in the JSF page on form elements by the getter methods. That is, the value of a given form element on the event creation page is tied to a field in this class. The create() method is in turn called upon form submission, and an event is ultimately created using these now-set fields. Note that there is some simple server-side validation done on the incoming event attributes. In the future, this could certainly be made more robust.

Miscellaneous .xhtml Pages (templates)

Our JSF templates are included as a number of different .xhtml pages, each of which is named thoughtfully. Our header and footer are abstracted as separate templates under the WEB-INF/templates directory. Since we are making use of a third-party framework to style the application (FlatUI and Twitter Bootstrap), we have to use numerous CSS and JS includes in the header.xhtml template. The relevant included files are all stored in a relevant folder nested within the WebContent root folder.

AJAX

We have implemented AJAX functionality on the calendar.xhtml page. Whenever a the dropdown menus for month and day are changed, the list of events is updated using AJAX. The AJAX functionality is implemented using JSF 2.0’s built in support for it with the <f:ajax> tag.

Example Transaction - Create Event Form Submission

When the create event form (create-event.xhtml) is submitted, the following events happen in the order specified:

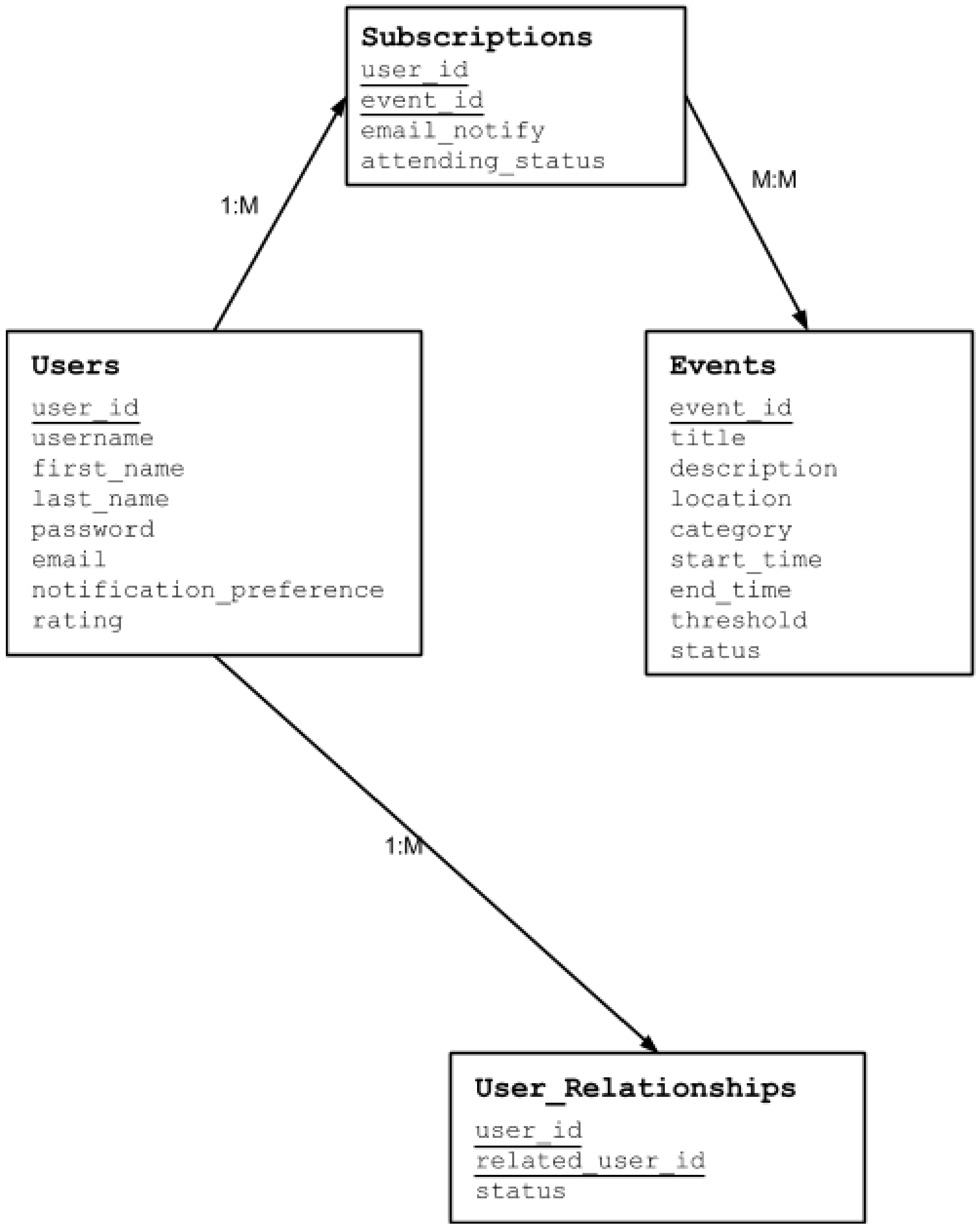
1. User -> Servlet: An instance of *MeetupUI/Event/CreateEvent.java* is created and its fields are filled with data corresponding to the form inputs on the create event page.
2. Servlet -> Session Bean: The create() method is called as a result of that action being bound to the “submit” button on the create event page. This method’s execution results in a call to createEvent() in the session bean EventService. An instance of EventDetails is passed to this method as an argument.
3. Session Bean -> Entity Bean: The createEvent() function in the session bean EventService receives the EventDetails paramater, and uses it to construct an Event entity bean.
4. Entity Bean -> Database Query: The Event() entity bean has its persist() function called to persist it in the database.

The transaction is complete (assuming nothing went wrong) - an event has been created and persisted in the database.

Database Schema

The database schema for this deliverable is constructed via the entity beans included in the application. Since our application only implements event-related use cases for the time being, this results in an extremely simple database schema consisting only of an event.

For the sake of furthering the completeness of our half-implementation, we've also included an unused entity bean representing a user. In the future, we would want to define relationships between these two entities. Said relationships would represent the tables in our database which tie an event to a user (Subscriptions), and a user to other users (User_Relationships). The start of defining such a relationship has been included in the Event and User entity beans. However, as it is not used in either of our use cases, it is purely included for the sake of including it - it's not actually used anywhere. A copy of our database schema, as we designed it in an earlier deliverable, is included on the next page. It has been replicated as entity beans in as truthful and complete a manner as reasonably possible.


Subscriptions
user_id
event_id
email_notify
attending_status
1:M
M:M
Users
user_id
username
first_name
last_name
password
email
notification_preference
rating
Events
event_id
title
description
location
category
start_time
end_time
threshold
status
1:M
User_Relationships
user_id
related_user_id
status