Overview

For this assignment, you will design and build an online interactive calendar. The user will be able to view a month at a time, or a list of events that are scheduled. By clicking on a scheduled event, the user can edit it or delete it. By clicking on part of a calendar day that is not scheduled, the user can enter a new event to be added to the calendar.

Calendar view

The main view of the calendar will show a single month at a time. When your page is first loaded, it should show the current month in a standard calendar view, i.e., one week per row and one day per column. The current month and year should be labeled at the top, and each day should be labeled with the day.

The user should be able to move forward or backward a month at a time by clicking on buttons or arrows that you provide. When the user moves forward a month, the entire display should be updated to show the next month instead of the current month.

Adding events

Initially, all of the days displayed will be empty. Since there is no server interaction required in the assignment, the calendar can be empty every time it is reloaded.

When the user clicks on a day, you should prompt for an event to be added to that day. This can be in a popup box, in a fixed location on the page, or (ideally) inside the day itself. A simple text input field should suffice. You should detect a mouse click in the day and present the input field. When the user enters something and either presses `<enter>` or leaves the field, its contents should be processed and added as an event for that day.

Events that have been added should be displayed in the appropriate day. In addition, if the user clicks on an event that has been added, you should convert it back into an input box and allow the user to edit it. Setting it to empty should delete the event, and other changes should be reflected in the display for that day.

Event view

In addition to the standard month view, the user should be allowed to select an event-list view. Instead of displaying a calendar, you should instead list the events that have been entered in chronological order.

In this list view, you should create a header for each day that has at least one event (skip the days with nothing), and under each header list the events that are scheduled for that day.

Only display a maximum number of events, or a maximum number of days that have events scheduled (it is your choice). You should also limit the display to one year in the future. If the user has a busy calendar, you should not display a huge list going years into the future.

Start the with current day, hiding events in the past. A nice addition would be to allow the event view to go back in time and forward in time, but this is optional.

Editing events

As in the calendar view, you should allow the user to click on an event and edit it. You do not need to support the creation of new events (though you are encouraged to—keep in mind that you will need to be able to specify the date as well), but you should allow the user to delete an event by clearing its description.

Hints

Store all of your events in Javascript objects, completely independent of the DOM. When updating the display, you can query you objects to see which days have events associated with them, and when you switch months, you will not need to scrape information from the HTML display.

One possible scheme is this:

- Use a single object that maps months to data. The key for each entry (the Javascript object property) could
be the year followed by the month, for example “2009-10” or “2010-02”.

- Each entry in the top-level object is itself another Javascript object that stores the events of a single month, indexed by day. Example keys would be “01”, “02”, “31”, etc.

- Each entry in a given month is an array that represents all of the events stored for that day. Each entry in the array is a string that stores whatever the user typed. The order of the events in the array is the same as the order they are displayed in the calendar.

- Each object is sparse. If there are no events stored for a given month, that entry does not exist in the top-level object. If no events are stored for a given day in a month, that day does not have a key in the month object.

Using this representation, it is straightforward to find the events when rendering a given month. Start by looking up the object for that month (use an empty object `{}` if the month does not exist in the data structure). As you build the UI for each month of the day, retrieve the list of events scheduled for that day from the month object. When the user schedules a new event, update the object for that day as well as updating the display.

When rendering the event view, start by looking up the current month, and then loop through starting with the current day. As you consider each day, check if it has any events scheduled. You may need to move on to future months as well.

When you find a month with no entries, you should skip the entire month (rather than checking every single day in the month for events). This optimization (combined with a maximum number of days/events to display) should be sufficient to give good performance even with a sparsely populated calendar.

To make it possible to edit an entry by clicking on it, store displayed event twice in the HTML—one inside a `span` and once inside an `input` box. The `span` should have a click handler that hides itself and displays the `input` box, and the `input` box should have handlers that hide it and re-display the `span` when a user finishes entering data (or deletes it when they have emptied the event out). It is usually easier to hide and show elements as needed than to create and destroy them on demand.

**Extras**

You are only required to implement the two main views with event creation and editing as described above. The suggestions here are not for extra credit, so do not attempt them until you have everything else working.

Additional navigation buttons would be helpful. A “today” button could take you back to the current month from wherever you are, and having the current day highlighted would be a nice touch as well. “Next year” and “previous year” buttons could also be helpful.

It would be nice to be able to specify a time for each event (other than entering one in the description). Google’s calendar lets you do this by typing the time first in the description, and then it parses it. For example, if you enter “1pm class in 119”, it will notice that “1pm” means 1:00 pm. It will then sort events by time, and label the times in a standard way. To make this easier, specify a specific format that you can parse, and then use a regular expression to detect it.

Recurring events are another common feature for calendars. Allow some way of indicating that an event repeats every week, or every two days, or on Mondays and Fridays, etc. Displaying them in the calendar view is straightforward, but you may need to think more about how to display them in the event view.

The biggest flaw in this calendar is that everything is reset when the page is refreshed. Using AJAX to integrate with the server would allow a persistent calendar, though that would require server-side programming (which is beyond the scope of this course). Modern browsers also permit persistent client-side storage, though the process is not well standardized across browsers.

**What to turn in**

Submit all of the files in your solution as a single `.zip` file. Include comments throughout your code documenting what you have done. In addition, please post your complete viewer on your student website ([citlogin>.php.cs.dixie.edu]). After the deadline, everyone will demonstrate his/her solution in class.