

Getting Started with BootsFaces: a Tutorial



Getting Started with BootsFaces: a Tutorial

Riccardo Massera, Stephan Rauh

March 29, 2017 at 09:00:00



132



[Suggest JSFC features.](#)

In this series of articles, Riccardo Massera and Stephan Rauh introduce you to the BootsFaces and AngularFaces JSF frameworks. In the [first article](#), they explained how these frameworks can be useful in your JSF Projects and highlighted their main features. This second article will get you started developing a simple project. The last article will cover more advanced features like the new AJAX engine of BootsFaces and the advanced search expressions.

Many tutorials on UI frameworks start with things like to-do lists. It's obvious we could do something like this, too. BootsFaces is a JSF framework that can be used for professional applications. But that's not what we'll do in this article. Let's do something unexpected and program a game. Granted, it's not going to be a fast-paced 3D game—we're talking about chess, but the nice thing about writing a chess game is that we can show many concepts of BootsFaces while developing a decent UI.

By the way, we've also started to experiment with 3D models controlled by the gyro sensor of your cell phone. Who's to say JSF is boring?

The basic page layout

The nice thing about chess is that so many people know it, or have at least an idea what chess is. Today, you don't have to know the rules. This article focuses on the UI. We've got a chess board populated by a couple of figures, which can be moved either by pointing and clicking, or by dragging a figure to another field and dropping it. Drag and drop is fairly uncommon for a JSF application, but we realized that the AJAX engine of BootsFaces is fast and reliable enough to support it.

Let's have a look at the final layout of our game.

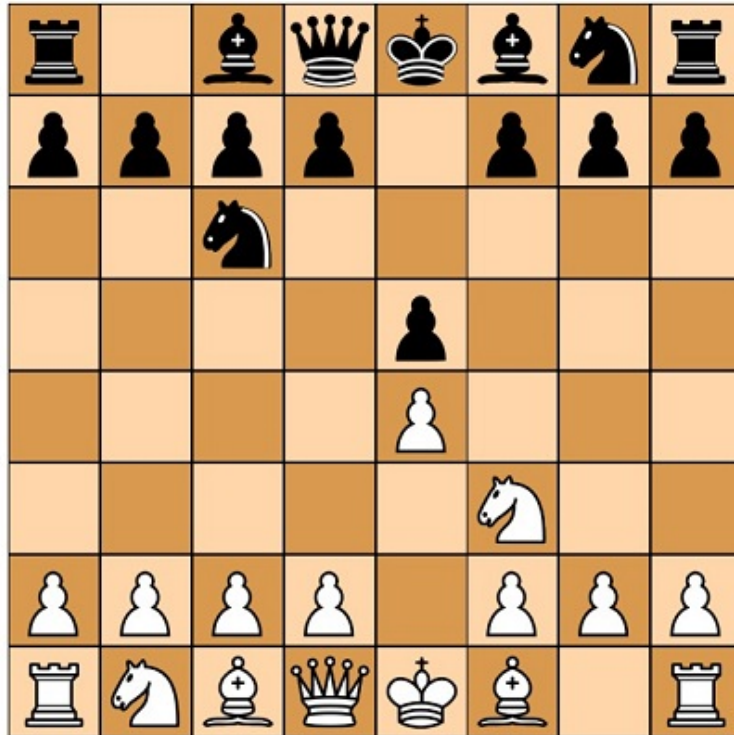
Sign up for newsletter
Email address

[Sign me up](#)

[The Editor's Des Podcasts](#)
[Inside Facelets](#)
[In the Trenches](#)

Google Custom Search

Waiting for your move



History

1. PE2- E4 E7- E5
2. NG1- F3 PB8- C6

Settings

Look-ahead: 5

Calculation breadth: 6

multithreading ☒

statistics

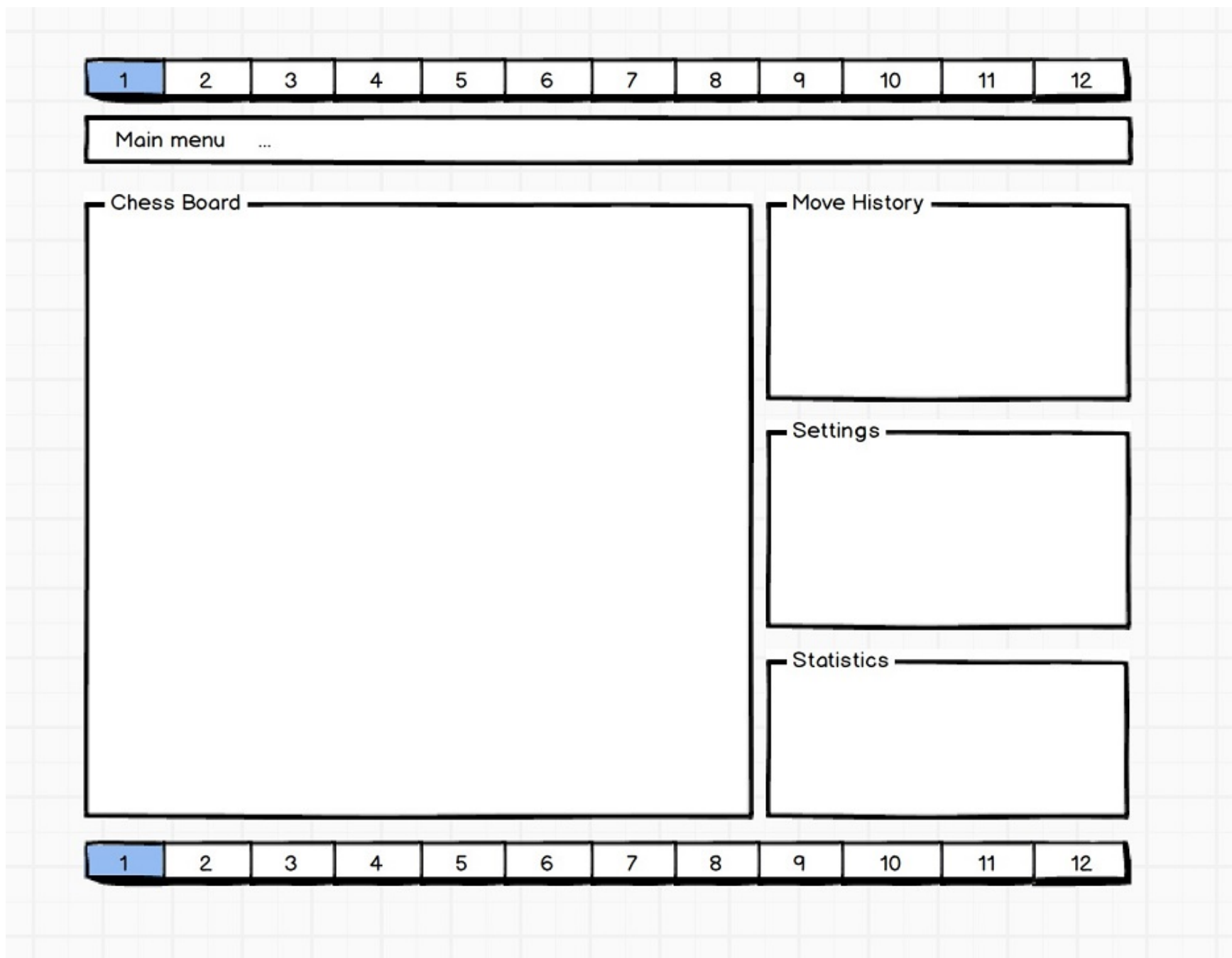
Elapsed time: 918.02ms
CPU time: 3673 ms
evaluated moves: 570.386
Evaluation: W:621 B: 560

There are many things here that you might also see in a business application. There is a main menu and a content area. The content area, in turn, is divided into several different panels, most of them showing output data. One of the panels contains a form consisting of a couple of input fields. You can also adjust the strength of the chess engine in a little "settings" form.

Containers and the Bootstrap grid system

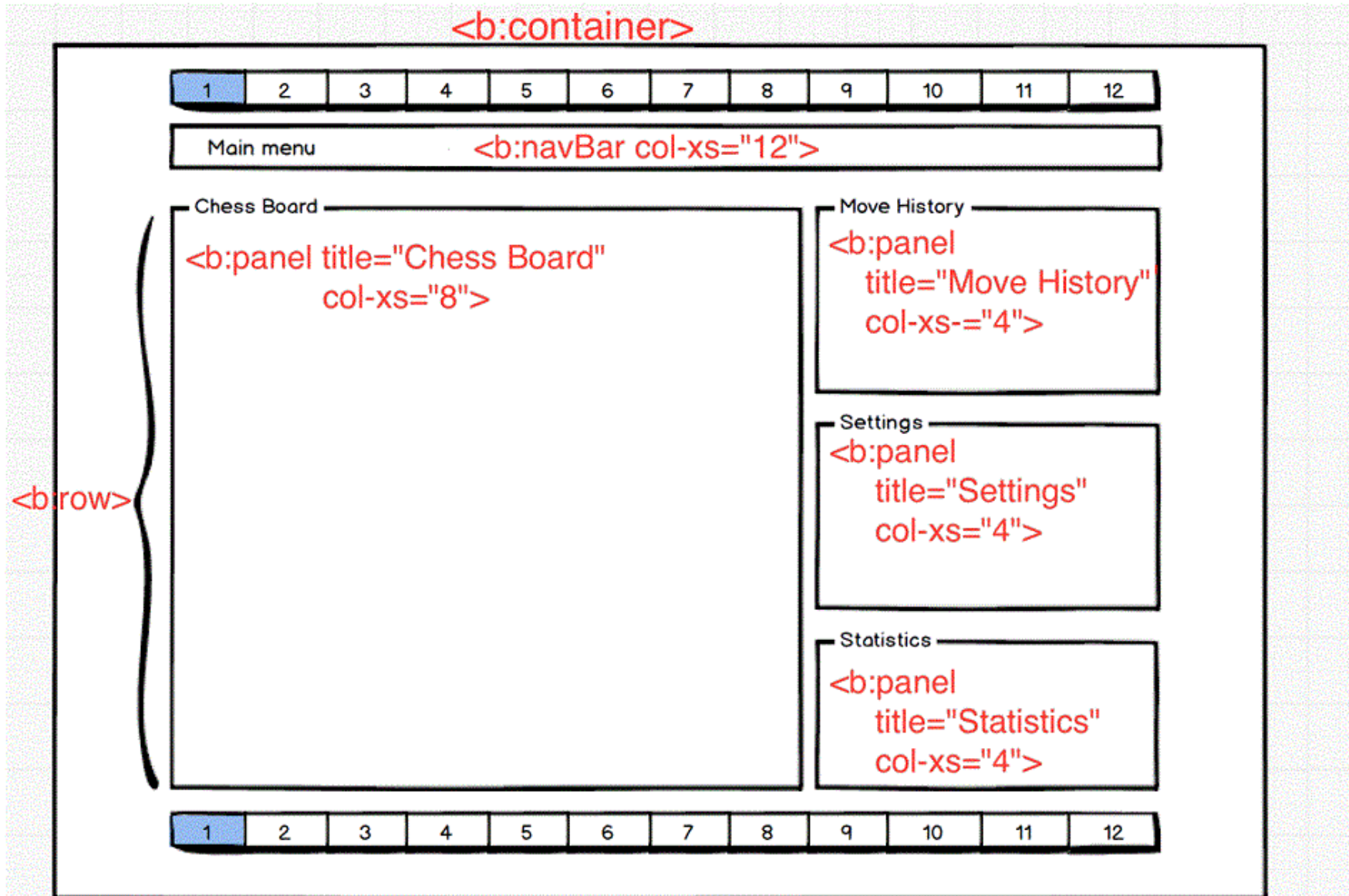
Let's look at an abstract sketch of the layout elements of the game, so we can illustrate the grid system of Bootstrap. The rulers at the top and the bottom of the image are not part of the application, but symbolize a core idea of Bootstrap: every page is divided into twelve equidistant columns.

The basic layout of our application looks like this:



If you're already familiar with JSF (or any other UI framework), you probably already know that these five items are containers. In other words, you can put other stuff into them. However, with BootsFaces there's a twist. The containers don't accept their content at arbitrary positions on the screen. There are only twelve legal positions on the x-axis. The layout aligns to a grid consisting of twelve equidistant columns. What sounds like a disadvantage at first glance, soon proves to be a gift from heaven. You can stop moving widgets pixel-by-pixel. Widgets automatically snap in at one of these twelve position, and the nice thing is that this twelve-column grid almost always makes for an attractive design.

To illustrate the point, here's the same diagram with some code added. For instance, there's a BootsFaces widget to display a container with a caption, the `<b:panel>`. Similarly, the main menu is rendered by another BootsFaces widget, the `<b:navBar>`. We've also added the attributes indicating how many Bootstrap columns the panels and the menu cover:



As you can see, the top menu spans all twelve columns, while the chessboard below takes eight columns. The other panels take the remaining four columns and are stacked over each other.

Traditionally, the Bootstrap community uses rather cryptic names to define this design. BootsFaces optionally allows you to follow this tradition, which leads to compact lines such as `<b:panel col-sm="8">`. This line contains a lot of information. It means that the panel takes two thirds of the available screen width on small screens. If your screen is very small, the panel takes the entire width of the screen. The other panels which may fit side-by-side on larger screens are stacked on top of each other on tiny screens.

However, the BootsFaces team thought it might be a good idea to provide a more expressive version of this syntax as an optional alternative:

```
<b:panel small-screen ="two-thirds">
...
</b:panel>
<b:column small-screen="one-third">
  <b:panel small-screen="full-width" title="History">
    ...
  </b:panel>
  <b:panel small-screen="full-width" title="Settings">
    ...
  </b:panel>
  <b:panel small-screen="full-width" title="Statistics">
    ...
  </b:panel>
</b:column>
```

Every container of BootsFaces is itself divided into twelve equal-size columns. The columns of the containers don't align with the columns of the entire page. Keep this in mind when you design your web app, because it might cause an unruly design. However, that hardly ever happens. The twelve-column design works surprisingly well, even if it's used recursively.

The overarching <b:container>

There's another container we've painted in the wireframe drawing but we haven't mentioned it yet. Every BootsFaces page should be embedded in a <b:container>. This widget helps to make your page look attractive. Among other things, it adds 15% white space to the left and to the right of your application. If you don't want this extra white space, you can turn it off by setting the attribute "fluid" to true. However, think twice before you do. Granted, sooner or later your business department will force you to get rid of the white space in order to display more data, but this comes at a cost. The additional white space makes the page much more readable, thus improving the productivity of the occasional user. Of course, if your application is targeted at power users, that's a different story altogether. In this case, it often pays to sacrifice attractive design to being able to see everything at a glance.

If every BootsFaces page has to be embedded into a <b:container>, why isn't the container generated automatically?

Well, from a technical point of view, that would be possible. But this approach has disadvantages. For one, you can't add attributes to configure the container if there's no container tag. Just think of the "fluid" attribute. Second, the obvious way to have the container generated automatically by BootsFaces is to redefine the class rendering the <h:body> tag. However, that's the sort of thing that's done by frameworks like PrimeFaces and LiferayFaces. We went to great lengths to avoid using the same approaches in order to maintain compatibility with those libraries.

Another reason why we don't generate the <b:container> tag automatically is that this would modify the look and feel of applications which have been developed without BootsFaces if you decide to add BootsFaces later.

Rows

Usually, columns are put into <b:row> containers. Every widget within the row is aligned to the top of the row. Sometimes, this causes ugly white spaces if the widgets have different heights. In the chess demo, we solved the problem by putting the small panels on the right-hand side into a common column, and by explicitly setting the height of the panels.

Of course, that's precisely the sort of thing you want to avoid by using Bootstrap. There are advanced grid frameworks like Masonry that can align widgets both along the x-axis and the y-axis. Currently, BootsFaces doesn't support such a feature.

Talking of rows, you may notice that the panels in the right-hand side column aren't in a row. In fact, they are: Bootstrap implicitly adds the row if you omit it. However, explicitly adding the `<b:row>` widget adds a few more layout changes, such as a margin to the left and the right.

The structure of the layout becomes clearer if you look at the version of the code that doesn't use any of the shortcuts:

```
<b:container>
  <b:row>
    <b:column small-screen="two-thirds">
      <b:panel>
        ...
      </b:panel>
    </b:column>
    <b:column small-screen="one-third">
      <b:row>
        <b:column small-screen="full-width">
          <b:panel title="History">
            ...
          </b:panel>
        </b:column>
      </b:row>
      <b:row>
        <b:column small-screen="full-width">
          <b:panel title="Settings">
            ...
          </b:panel>
        </b:column>
      </b:row>
      <b:row>
        <b:column small-screen="full-width">
          <b:panel title="Statistics">
            ...
          </b:panel>
        </b:column>
      </b:row>
    </b:column>
  </b:row>
</b:container>
```

Working with forms: the settings panel

The settings panel stands out from the other panels because it contains a form. However, defining a form in BootsFaces is not a big deal:

```
<b:panelGrid columns="2">
  <h:outputText value="Look-ahead:" />
  <b:inputText value="#{settings.lookAhead}" />
  <h:outputText value="Calculation breadth:" />
  <b:inputText value="#{settings.movesToConsider}" />
  <h:outputText value="multithreading" />
  <b:selectBooleanCheckbox
```



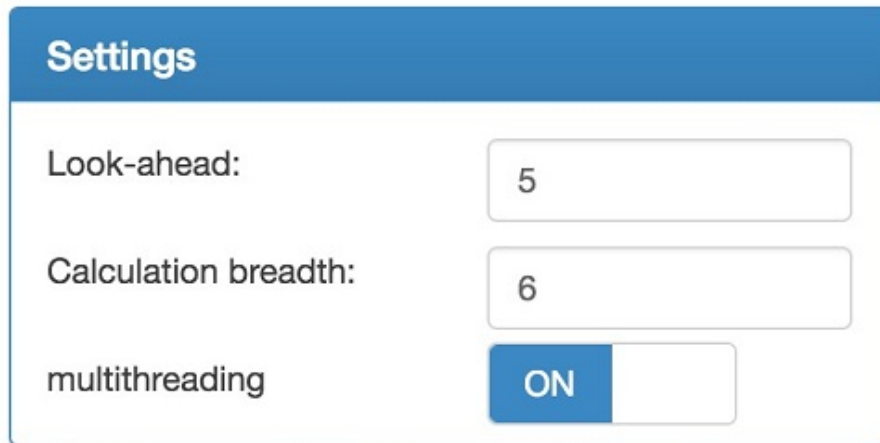
```
value="#{settings.multithreading}" />
</b:panelGrid>
```

The average form consists of two columns, one for the label of the input field and one for the input field itself. This use case is so common we've written the `<b:panelGrid>` component. In this case, it divides the available space into two equidistant columns.

There's not much to say about the form itself. Most JSF input field widgets have one or two BootsFaces counterparts. In most cases, it suffices to replace the "h:" prefix with "b:" to make the widget match the Bootstrap design.

Bootstrap switch as an alternative to the checkbox

Arguably, the checkbox responsible for activating multithreading doesn't look that good in a BootsFaces application. As an alternative, you can use the BootsFaces `<b:switch />`. That's the switch you probably already know from your smartphone.



Rendering the chess board

Drawing the chess board turns out to be pretty easy. Basically, it's just eight rows of eight pictures. The biggest challenge is to position the images close to each other. Bootstrap encourages you to use a lot of white space, so we had to use a lot of the CSS style to remove the white space again. The disadvantage is that the chess board has a fixed size. It's not responsive. From that point of view, it's not the ideal example to show how to use BootsFaces. In general, the point of using BootsFaces is to make applications responsive and to target the mobile platforms with little effort. So playing with CSS styles to modify the size or position of widget usually backfires on you sooner or later.

This is the JSF source code without the CSS magic:

```
<b:panel title="#{board.title}" look="primary">
  <ui:repeat var="row" value="#{board.rows}"
    varStatus="rs">
    <div>
      <ui:repeat var="image" value="#{row.images}"
        varStatus="cs">
```



```

<span>
  

  <b:image value="#{image}"
          style="#{board.getOpacity(rs.index,cs.index)}"
          onclick="ajax:board.onclick(rs.index,cs.index)"
          update="@form :messages"
          process="@form" />
</span>
</ui:repeat>
</div>
</ui:repeat>
</b:panel>

```

The background of the field is a traditional HTML `` tag. The chess piece is a BootsFaces `<b:image>` widget. Both widgets are put into a common `` tag to make sure both images are drawn over each other. That's done by the CSS magic we omitted for the sake of simplicity in the listing.

AJAX

The difference between `` and `<b:image>` is that the BootsFaces version supports AJAX. In the next installment of this series, we'll talk about the BootsFaces approach to AJAX in more detail, but for now, we'll just describe the example briefly. Let's have a look at the code snippet:

```

<b:image onclick="ajax:board.onclick(rs.index,cs.index)"
        update="@form :messages"
        process="@form" />

```

and the simplified version of Java bean:

```

@ManagedBean
@ViewScoped
public class Board implements Serializable {
    public void onclick(int row, int row) {
        selectChessPiece(row, column);
    }
}

```

When the `<b:image>` is clicked, the "onclick" event handler is called. In BootsFaces, this event handler may contain both JavaScript and an AJAX call. To distinguish between JavaScript and AJAX, simply add the prefix "ajax:". When the AJAX call is executed, it sends the input fields defined by the "process" attribute to the server, and the AJAX response updates the region of the screen defined by the "update" attribute. In this case, the entire form is sent to the server, and both the error messages and the form are updated by the response.

Using the JavaScript event handlers means that almost every DOM event can be used to trigger an AJAX event. Some components even support jQuery events. The chess demo uses this feature to support drag and drop via AJAX. That's probably nothing you'll want to do in production unless you've got a limited number of users and a fast network connection with low latency. However, it works surprisingly well in our demo, even over the Internet.

Wrapping it up

You've seen the basic ideas of both BootsFaces and the underlying CSS framework, Bootstrap. In general, every page is wrapped in a `<b:container>` to ensure a nice layout with a lot of white space, unless you decide to use the "fluid" layout to get rid of the wide margins around your page. Most widgets are embedded in the grid system consisting of `<b:row>` and `<b:column>` widgets. Sometimes you can omit the rows and the columns by using a shortcut, but the grid system is always there, helping you to avoid wasting time aligning widgets pixel-by-pixel. Leave that to BootsFaces. It's not perfect, but usually it does a good job.

You've also seen how to create a form with BootsFaces. There's nothing fancy with that: every HTML input field has a BootsFaces counterpart, and there are even a couple of BootsFaces widgets that go beyond the HTML standard. For instance, there are sliders, switches, and datepickers. Starting with HTML5, there's actually a standard HTML datepicker, but it doesn't fit in with the Bootstrap look and feel.

Sneak preview

The next part of this series describes AJAX and the search expressions of BootsFaces in depth. The goal of the BootsFaces team is to make JSF programming easier, and both AJAX and the advanced search expressions are important steps to reach this goal.

Resources

- [Showcase and manual of the BootsFaces project](#)
- [GitHub repository of the BootsFaces project](#)
- [Source code of the project on GitHub](#)

Riccardo Massera is the founder of BootsFaces. He has over 15 years of professional experience in Web development and was fortunate to witness the dawn of the Internet, authoring one of the first personal web pages in Italy, and later working for one of the first Italian Internet Service Providers. He is now a full-stack developer, working with both server side and client side, with a focus on the latter. He likes working with Bootstrap, jQuery and AngularJS, building projects with Gradle, developing Java Enterprise applications and loves Virtualization Technologies. When not coding, Riccardo likes traveling around the world and learning new languages.

Stephan Rauh is a senior consultant at OPITZ CONSULTING GmbH (<http://www.opitz-consulting.com/en/home.php>). Chances are you already know Stephan's blog, <http://www.BeyondJava.net>. During the last couple of years, he has spent a lot of time with JSF and AngularJS. He is the author of AngularFaces and a regular committer to the BootsFaces project. Every once in a while, you may meet him at a conference.

401 / UNAUTHORIZED

POWERED BY ADVERTPRO

Site version 2.00 [Report web site problems](#)

Copyright (C) 2003-2011 [Virtua, Inc.](#) All Rights Reserved. Java, JavaServer Faces, and all Java-based marks are trademarks or registered trademarks of Oracle Corporation. in the United States and other countries. Virtua, Inc. is independent of Sun Microsystems, Inc. All other trademarks are the sole property of their respective owners.