CaptainCasa

Enterprise Client

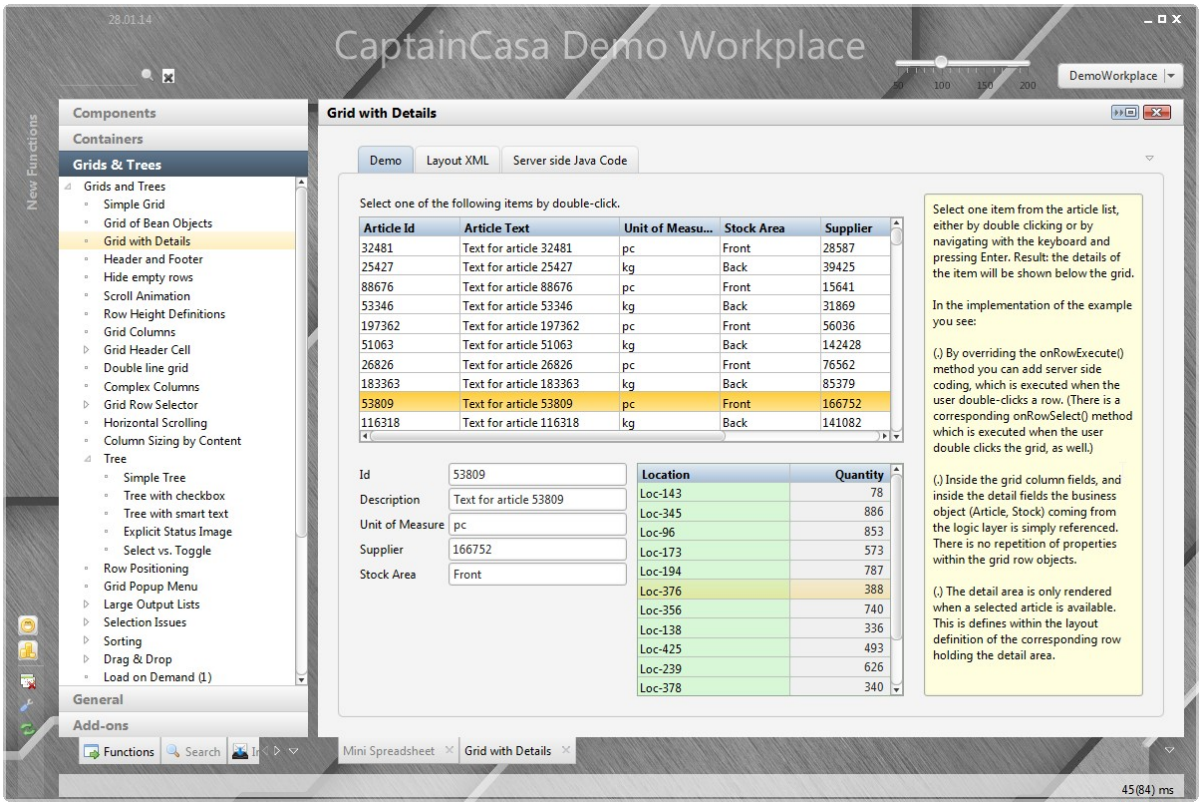
In a Nutshell

CaptainCasa Enteprise Client - In a Nutshell

CaptainCasa Enterprise Client is a rich client solution that meets the requirements of typical business applications:

- Many dialogs
- Complex, server side business logic
- Heavily used by employees to constantly support them doing their job
- Long term life cycle – developed, sold and used for many years

CaptainCasa Enterprise is based on Java standards (JavaFX, Java Server Faces in the backend) following a server-centric frontend architecture.

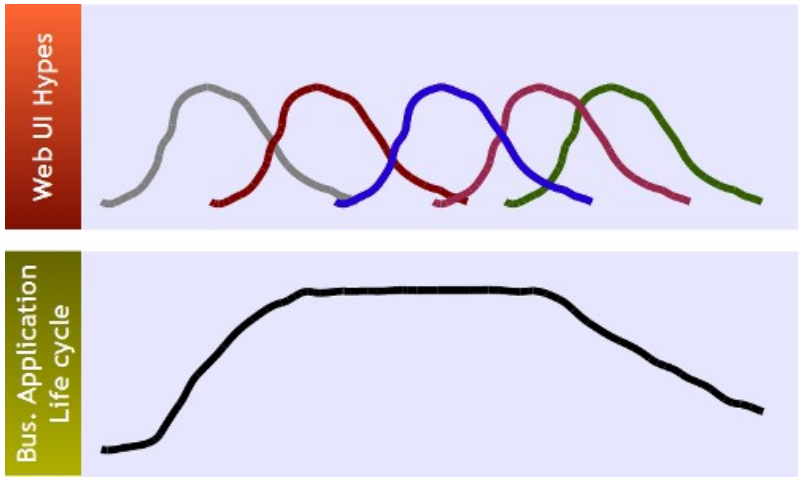


CaptainCasa Demo Workplace

Frontend Technology

Today's hypes in the area of user interface development are dominated by HTML5 based frameworks. While these frameworks in general significantly simplify the development of web scenarios, they still fail to be an adequate technology solution meeting the requirements of business applications:

- Browser incompatibilities and performance continue to be a constant problem.
- Development efficiency in the web is still way behind expectations.



- The long term stability of frameworks still IS a problem. There's a constant coming and going of frameworks and hypes.

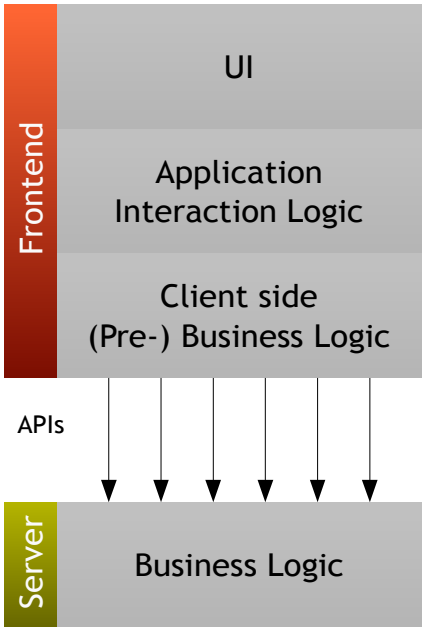
All this means risk and cost for your long term oriented application development. And all this means that many applications are still stuck with UI technology coming from the “late 90s” (C++, VB, Java Swing, Delphi) that more and more fail to meet the usability requirements of modern frontends.

JavaFX – the Java standard in the are of UI processing – is an up to date UI technology that meets these requirements and that provides the long term stability expected by you and your customers.

The client side of CaptainCasa Enterprise Client is based on JavaFX as consequence.

Frontend Architecture

The core architectural question when defining a frontend architecture is, if you should follow a “client-centric” or a “server-centric” approach.



Client-centric

In the client centric architecture your development is split between a frontend part (developed by frontend developers) and a backend part (developed by backend developers). A couple of APIs are defined by the backend side that are accessed from the UI processing of the frontend.

While being an adequate architecture for smaller projects, the client centric programming comes with significant disadvantages for bigger applications:

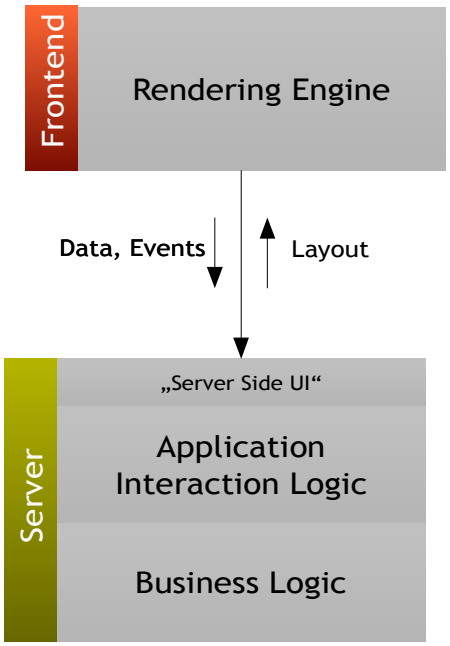
- The efficiency of development lacks because of having a split between frontend and backend development.
- The number of APIs is growing and growing. The client UI logic tends to require very specific, fine granular APIs in order to provide a responsive user interface.
- Many APIs automatically mean potencial security and potential roundtrip problems.

- There is the “real” logic on server side and some “pre”-logic on client side, which always have to be in sync.
- The size of the client is growing with a growing size of the application.

Many application developments start with the client centric approach because it's the easiest to start with – and because of not having thought about the frontend architecture too much. The result are client programs that were called “fat clients” in the 90s.

Server-centric

The server centric approach requires some more framework background.



Here a generic client serves as some kind of “form processor”: it receives some form definition from the server side form processing and outputs the form onto the user's screen. The user processes the screen (e.g. performs some data input). On certain events (e.g. button pressed) the updated form data is sent back to the server processing. On server side the data changes and the event are processed, the form is updated and sent back to the client side.

In the server centric approach all logical form processing is on server side – while the client is a “stupid” rendering engine. The client does not know what the business content of a form really is, but just know the arrangement of components and the data contained in the components.

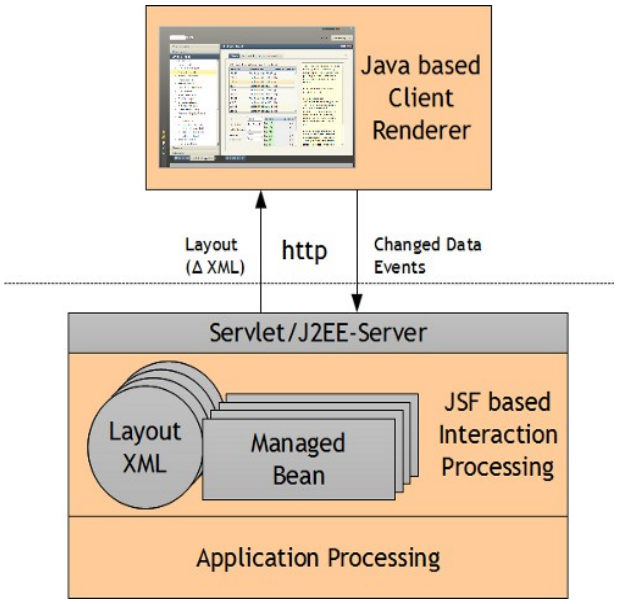
The advantages of the server centric architecture are:

- The application development purely resides on server side. There is no split up between development groups.
- There is only one interface between the frontend client and the backend: the “form channel”.
- There is no exposing of fine granular business APIs by the server.
- There is a guaranteed one-roundtrip-behavior between client and server.
- The size of the client is stable – and independent from the number of forms to be processed. There is no need to constantly roll out the client (e.g. due to application bug fixes), bug fixes are on server side.

CaptainCasa Enterprsie Client

Server-centric Architecture!

CaptainCasa Enterprise Client follows the server centric frontend architecture. The client part is a generic frontend based on JavaFX, the server side is built on top of the Java Server Faces (J2EE-JSF) framework. The client talks to the server via http(s), only changes are transferred between client and server (“lightweight roundtrips”).



Java Server Faces is optimally suited to serve as Java standard for the backend processing. It allows the definition of descriptive layout definitions on the one hand and the direct manipulation of forms at runtime on the other hand. It provides strong concepts in the area of binding the form data and functions to server side bean processing.

Java Server Faces does not imply any further server side frameworks, but is open to adapt to any type of server side business logic processing. Frameworks like Spring, Hibernate, Enterprise Java Beans can be used.

CaptainCasa on the one hand uses Java Server Faces, on the other hand completely hides it behind its tools. There is no need at all to learn JSF before starting to develop with CaptainCasa – developers implicitly work on JSF without knowing.

Component Library

CaptainCasa Enterprise Client comes with a huge set of standard components, that are directly and flexibly usable in application forms:

- Basic components: field, combo box, button, icon, ...
- Grid components: simple and complex grids, trees, dynamic rows, optimized loading of items
- Container components: pane, tab pane, titled pane, ...
- Graphics components: chart, svg, ...
- Drag&Drop, right mouse button menu, tooltips, online help, internationalization consistently supported through all components.

The component library is extensible by adding own components – both composite components and completely new components.

Programming Model

Programming Enterprise Client is completely done on server side. Forms are descriptively defined in XML layouts. Forms are bound to Java beans, which themselves may connect any business logic below.

Example: the following dialog is built using CaptainCasa Enterprise Client:

The server side layout definition is an XML definition:

```
<t:rowtitlebar text="JavaFX in front of JSF" />
<t:rowbodypane>
  <t:row id="g_3">
  <t:tabbedpane width="100%">
    <t:tabbedpanetab padding="20"
       rowdistance="5"
       text="First Tab">
      <t:row>
        <t:label text="Your Name"
           width="100" />
        <t:field
           text="#{DemoHelloWorld.name}"
           width="200" />
      </t:row>
      <t:row>
        <t:coldistance
           width="100" />
        <t:button
           actionListener="#{DemoHelloWorld.onHello}"
           text="Hello" />
      </t:row>
      <t:rowdistance  height="20" />
      <t:row>
        <t:label text="Result"
           width="100" />
      <t:field enabled="false"
         text="#{DemoHelloWorld.output}"
         width="100%" />
      </t:row>
      </t:tabbedpanetab>
      <t:tabbedpanetab
         text="Second Tab" />
    </t:tabbedpane>
  </t:row>
</t:rowbodypane>
```

The attributes of components either are directly defined or are bound by using an expression. The expression points into a server side bean processing:

```
package demo;

import javax.faces.event.ActionEvent;

public class DemoHelloWorld
{
    String m_name;
    String m_output;

    public void setName(String value)
    {
        m_name = value;
    }
    public String getName()
    {
        return m_name;
```

```
    }
    public String getOutput()
    {
        return m_output;
    }

    public void onHello(ActionEvent ae)
    {
        if (m_name == null)
            m_output = "No name set.";
        else
            m_output = "Hello World,
                       "+m_name+"!";
    }

}
```

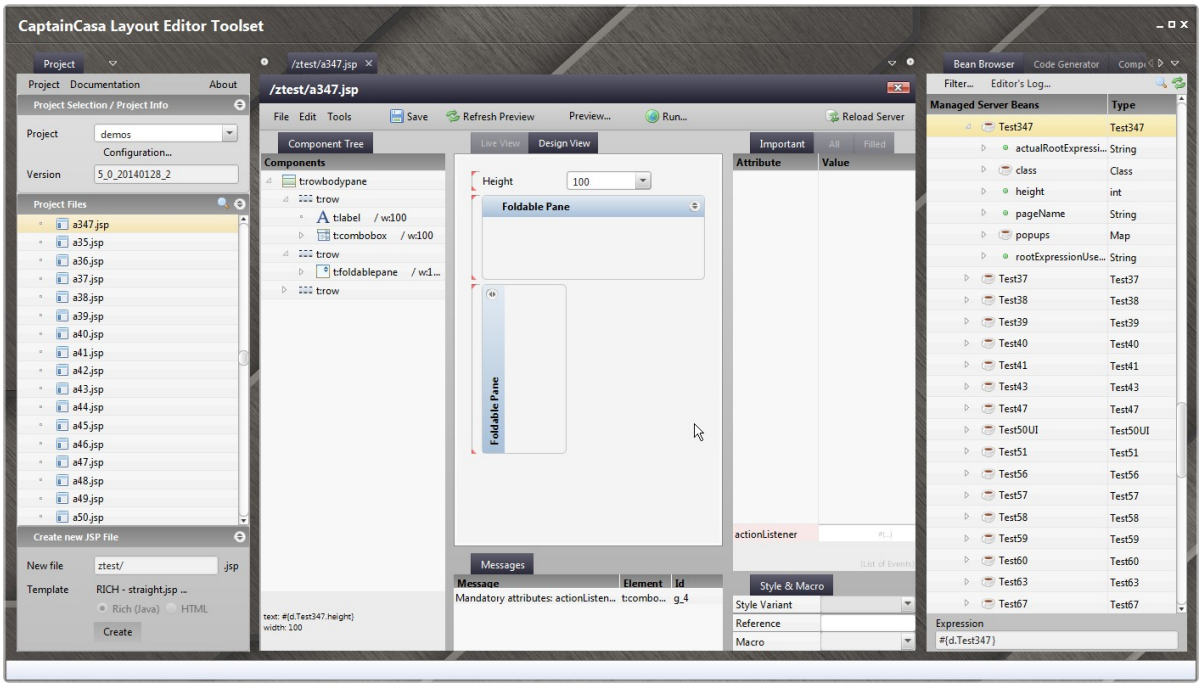
At runtime the dialog is loaded on server side, the components pick their data via expressions from the bean processing. A corresponding XML definition is sent to the client.

On client side the user keys in his/her name – when pressing the “Hello” button all data changes are sent to the server side.

On Server side the data changes are passed into the bean processing and the event associated with the button is called. As part of the event processing properties are changed.

After event execution the form data is recollected – finding out that some properties (e.g. “output” in the example) have changed. A corresponding delta-XML-definition is sent to client side and their updates corresponding components.

Tools



CaptainCasa Layout Editor

CaptainCasa comes with a set of tools, including:

- WYSIWYG (what you see is what you get) Layout Editor
- Bean code generator
- Internationalization/Translation tools

- Profiling tools

The actual Java development is done within an IDE of your choice (e.g. Eclipse, Netbeans, etc.).

CaptainCasa Community Concept

User interfaces are a constant area of changes and improvements:

- New components
- New designs
- New ways of interaction

On the one hand this needs to be technically reflected by a solid and open architecture.

On the other hand this needs to be reflected by an organization that quickly reacts when it comes to improving the frontend framework accordingly.

CaptainCasa is a corporate community of independent software vendors, that are commonly using and driving the CaptainCasa Enterprise Client. The framework is released and updated regularly.

Advantages of using CaptainCasa Enterprise Client

Using CaptainCasa Enterprise Client includes the following advantages:

- Your application's frontend is built using long term oriented technology standards – to be supported over the next decade(s).
- The development of frontends is extremely efficient: there is only server side development, there is no expertise on JavaFX and/or on Java Server Faces required.
- The resulting frontends look modern, the look&feel can be easily adapted. Modern aspects of frontends (animations, multi touch, ...) are covered.
- The performance and stability of the client is high. There is no dependency to any browser installation.
- You are part of a community of professional software developers, joining their forces in the area of frontend architecture.

Licensing CaptainCasa starts with no-cost licenses (without restrictions but of course without warranty and without service) and reaches up to priced licenses (with warranty, with defined services, with code ownership).

Please set up contact to CaptainCasa by writing a mail to info@CaptainCasa.com.

CaptainCasa GmbH

Hindemithweg 13
D – 69245 Bammental

06223 484147

http://www.CaptainCasa.com
info@CaptainCasa.com