# UI components – Comparing Vaadin & GWT Framework

## GWT (v_2.4)

[http://www.gwtproject.org/doc/latest/DevGuideUi.html]:

1. **Cross-Browser Support** - Use widgets and composites for cross-browser compatibility
2. **Layout Using Panels** – Explore the various panels available for layout
3. **Widgets** – Create user controls with widgets
4. **Creating Custom Widgets** – Create new widgets, composite widgets, or native JavaScript widgets
5. **Cell Widgets** – Work with widgets, panels, the DOM, events, CSS, declarative UI and images
6. **Editors** – Allows data stored in an object graph to be mapped onto a graph of Editors
7. **Working with the DOM** – When necessary, manipulate the browser's DOM directly
8. **Events and Handlers** – Handle events published by widgets
9. **Working with CSS** – Style widgets with cascading style sheets
10. **Declarative UI with UiBinder** – Build widget and DOM structures from XML markup
11. **Bundling Image Resources** – Optimize image loading by reducing the number of HTTP requests for images
12. Not supported: validation provider (-resolver)
13. Not supported: ConstraintValidator (workaround using GWT.create())
14. Not supported: XML configuration

### *UI-Library - Widgets*

*[http://www.gwtproject.org/doc/latest/RefWidgetGallery.html]*

Button PushButton RadioButton CheckBox DatePicker ToggleButton TextBox
PasswordTextBox TextArea Hyperlink ListBox CellList MenuBar
TreeCellTree SuggestBox RichTextArea FlexTable Grid CellTable CellBrowser
TabBar DialogBox

### *UI-Library - Panels*

*[http://www.gwtproject.org/doc/latest/RefWidgetGallery.html]*

PopupPanel StackPanel StackLayoutPanel HorizontalPanel VerticalPanel

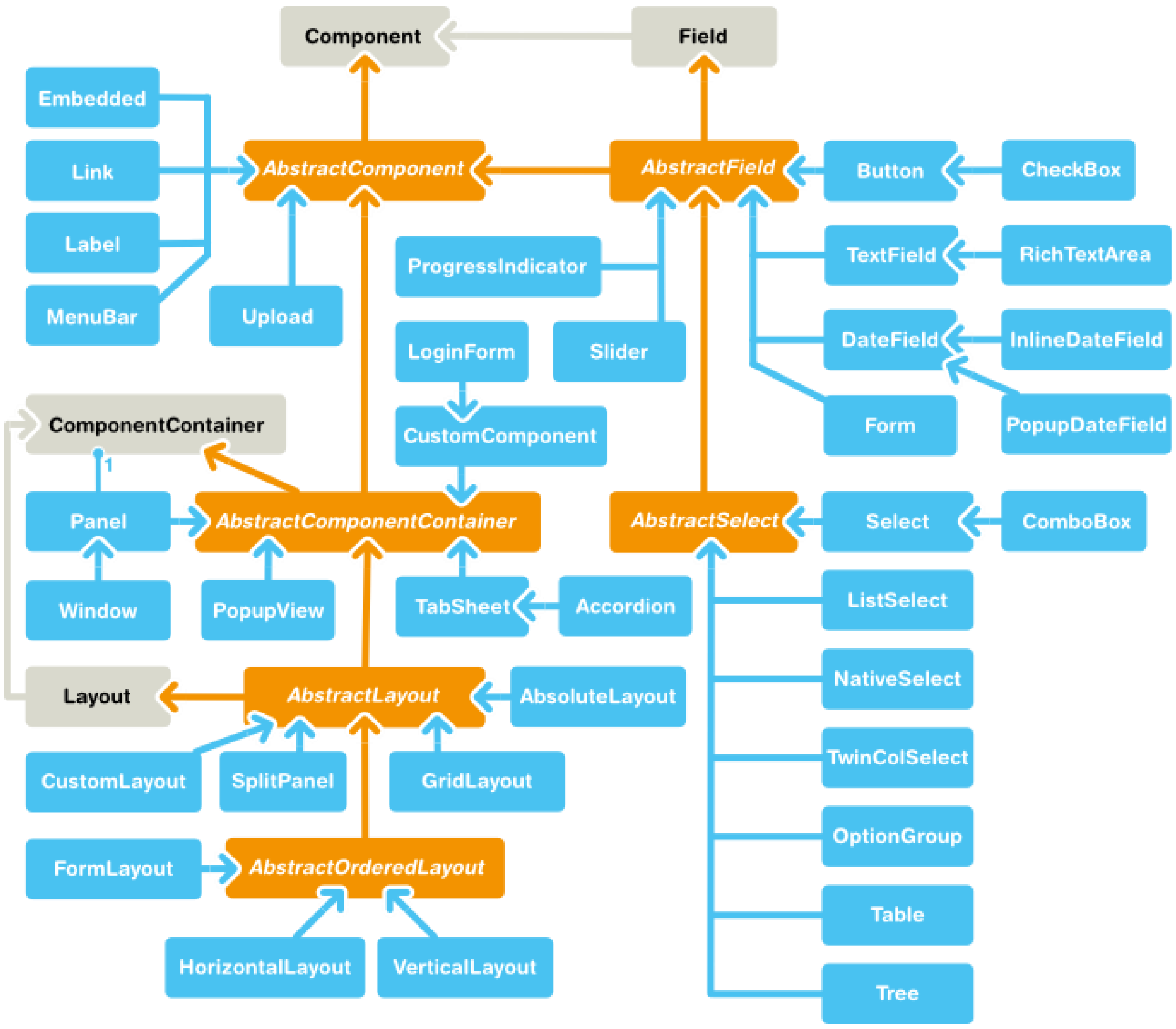
FlowPanel VerticalSplitPanel HorizontalSplitPanel SplitLayoutPanel DockPanel

DockLayoutPanel TabPanel TabLayoutPanel DisclosurePanel

# Vaadin (v_7)

## UI components

[https://vaadin.com/book/-/page/components.html]



*graphic 1: UI Components (already in Vaadin v_6)*

## Conclusion

### GWT

- In GWT application logic is normally run on client side. It only calls server when it needs to read/save some data
- App logic (replies to user interaction) is faster as it is run locally in the browser. It's also relatively insensitive to bad network conditions. Network is used only when needed (to read/save new data), which saves net traffic (important for high traffic sites)
- You have to separate your application into Client and Server code, this will affect the architecture of your application
- In GWT client code, you must code in Java, and have a limited subset of language features available (that the GWT compiler can translate into Javascript)
- GWT compilation is very slow, although in development mode you have the emulator. This makes production environment updates painful
- It's very simple to extend GWT with 3rd party widgets, or roll your own
- No way to hide your code
- Data binding to forms/tables quite complex

### Vaadin

- In Vaadin application logic is on server side. Client side must normally call server after every user interaction
- App logic slower and introduces a lag in UI interaction which is annoying to user. If network is bad this will show in UI responsiveness
- App logic is run on the server so it cannot be inspected by the user. Arguably (Vaadin claims) that makes it more secure
- In Vaadin, you can code in any JVM language, since everything runs in the server (I'm using Vaadin with Scala)
- Client-server communication is built-in
- Creating new Vaadin widgets is more complex
- use built-in GWT to do something on client-side (simplicity of server-side programming model with full control of browser events)
- simple automated data binding to your forms and tables
- adding server-side data validation

➢ **Up from v7 Vaadin Framework includes GWT as core component (and commercial support for GWT projects)**