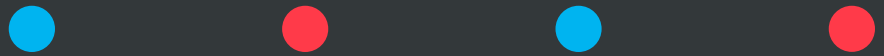
QUICK RECAP

A reminder of what we have done during Lecture 05



vaadin}>

LAST TIME...

- Gory details of Vaadin
  - How to handle errors?
  - What are resources?
  - What is UI?
- Extending server-side components
  - What should be extended and when?

EXTENDING THE VAADIN FRAMEWORK: WORKING WITH CLIENT-SIDE CODE

Development of Modern Web Applications (with Vaadin)

Lecture 06

vaadin

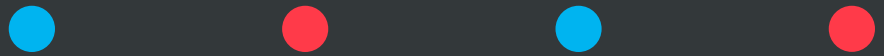
OVERVIEW

- Developing an add-on

- Shoutbox app continues indirectly
  - Add-on is a separate project

CASE EXAMPLE

Behind the scenes of a Vaadin add-on



SHOUTBOX – REQUIREMENTS

- Support room navigation
  - An area in which recently visited rooms are shown
  - No scrolling, all rooms shown
    - Multipe rows if needed
  - Clicking a room name navigates to it
- Reusable in other settings
  - Selection clearly visible
  - Vaadin add-on

GOALS

- A bunch of labels to select one from
  - Something like a menu bar, but a select
- Possible to wrap into a number of lines
  - By specifying maximum row length
- Works with any container
  - Reacts to any and all changes
- Let's call it `FlatSelect`

ASSUMPTIONS AND LIMITATIONS

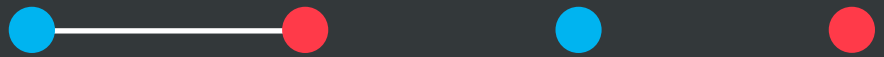
- Ignore efficiency
  - This is an example, not production code
- Derive from `AbstractSelect`
  - On the server-side, that is
  - Code is a bit more complex
  - More functionality for free
- Ignore styling

DESIGN DECISIONS

- Decouple client from server
  - Server does not render
  - Client does not handle component logic
    - A set of panels with buttons should do
- Flexibility
  - Choose property that holds a caption
  - Specify maximum number of items per row
- Meaningful default behaviour
  - Satisfying results out of the box
  - Must work with a simple collection

PROJECT STRUCTURE

Getting started



MAVEN

```
mvn -B archetype:generate
    -DarchetypeGroupId=com.vaadin
    -DarchetypeArtifactId=
       vaadin-archetype-widget
    -DarchetypeVersion=7.7.3
    -DgroupId=org.vaadin.miki
    -DartifactId=flatselect
    -Dversion=0.1-SNAPSHOT

cd flatselect-addon
mvn install
cd ../flatselect-demo
mvn package jetty:run
           # WARNING: compiles widgetset
```

CREATED DIRECTORIES

- `{project}`
  - `/{project}-addon`
    - Contains source code for the add-on itself
  - `/{project}-demo`
    - Contains a sample demo app
    - Includes the dependency to add-on

CREATED ADD-ON FILES

- `{package}.{foo}.java`
  - Server-side component
  - Similar to pure server-side component
- `{package}.{foo}Widgetset.gwt.xml`
  - In resources
  - Client side widgetset
- `{package}.public.{foo}.style.css`
  - In resources
  - Style for the add-on

CREATED ADD-ON FILES

- `{package}.client.{name}`
  - `{foo}ClientRpc.java`
    - Allows the server to call the client almost on-demand
  - `{foo}Connector.java`
    - Synchronises state with widget
    - Handles communication from server side
  - `{foo}ServerRpc.java`
    - Allows the client to call the server side on-demand
  - `{foo}State.java`
    - Holds the information about the state
  - `{foo}Widget.java`
    - Client-side code, by default GWT

SHARED STATE

Communication channel

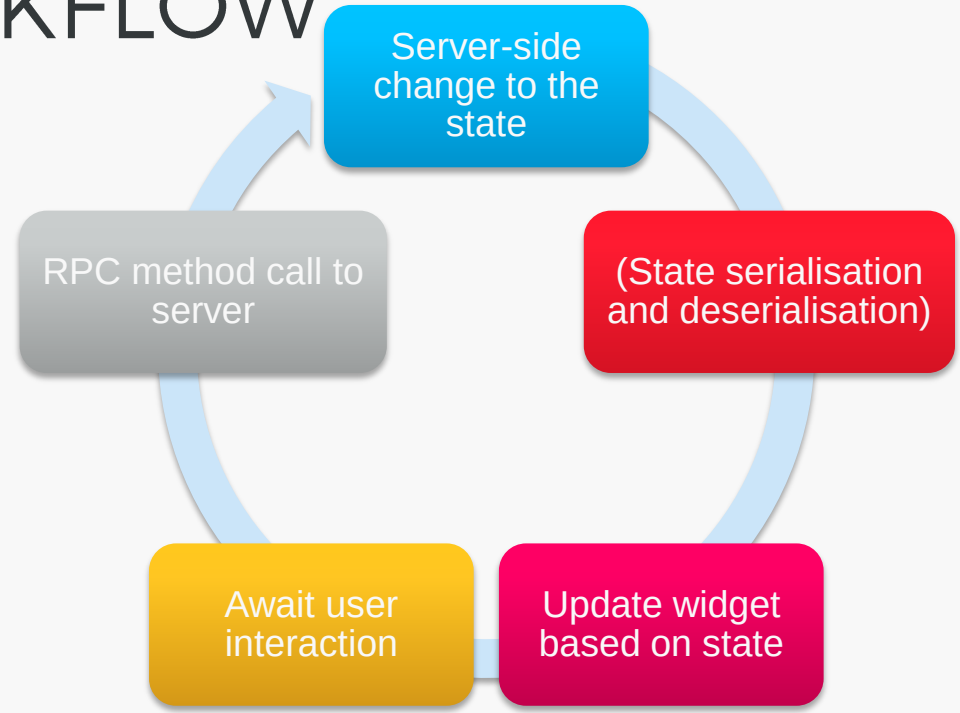
SHARED STATE

- Communication from server to client
  - Not the other way
    - Use server RPC for that
- Properties common to client and server
  - Determined by the functionality
- Serialised by the framework
  - All primitive types
    - Except Object
  - Beans
  - Maps and arrays of the above
    - Maps must be string-indexed
- Handled behind the scenes
  - Only changes to the state are sent
  - Dirty components

REFERENCING COMPONENTS

- Cannot be done directly
  - Client side has widgets
  - Server side has components
- Use `com.vaadin.shared.Connector`
- Components must exist in run-time hierarchy
  - Otherwise `null` is passed around
  - Straightforward
    - `setParent(foo);`
    - `addComponent(foo);`
  - Forces the server side to remember its components

WORKFLOW


Server-side change to the state
(State serialisation and deserialisation)
Update widget based on state
Await user interaction
RPC method call to server

COM.VAADIN.SHARED.COMMUNICATION.SHAREDSTATE

- Base class for all states
  - Not an interface
- Automatic (de)serialisation
  - Bean with getters and setters
  - Bean with public fields
  - Default zero-argument constructor
- Meaningful default values

COM.VAADIN.SHARED. ABSTRACTCOMPONENTSTATE

- Base class for component states
  - Subclass component states from this one
- Stores component properties
  - Caption
  - Description
  - Style names
  - Height and width
- Subclasses for other components
  - Layout, Embedded, SplitPanel, Label, MenuBar…
  - Use when appropriate

```
package org.vaadin.miki.
               client.flatselect;

import com.vaadin.shared.
              AbstractFieldState;

public class FlatSelectState
     extends AbstractFieldState {

 public int value = -1;
 public String[] options =
                  new String[0];
 public int optionsPerRow = 0;
}
```

FLATSELECTSTATE

Index of current value
Available options
Max options per row

SERVER SIDE

Managing state changes

SERVER-SIDE COMPONENT

- Listens to client-side RPC calls
- Modifies shared state
  - According to the logic of the component
- Dirty components
  - `markAsDirty()`
  - Client-side widget must be synchronised
  - Sends state changes on next request
    - `public void beforeClientResponse(boolean initial);`
- No rendering
  - Or anything related to it

KEY METHODS

- Setters and getters for properties
  - Or other methods related to behaviour
  - Call `markAsDirty()` to force synchronisation
    - `markAsDirtyRecursive()` synchronises all child components
- `public FlatSelectState getState()`
  - `return (FlatSelectState) super.getState();`
  - Just to get rid of constant typecasting ☺
- `public void beforeClientResponse(boolean initial)`
  - Update shared state to reflect changes
  - Or access shared state directly in the component

FLATSELECT

Methods (setters/getters)

- Selection index
- Caption property
- Options per row

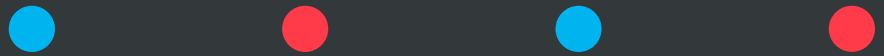
Marked as dirty when…

- Setting index
- Changing options per row
- Any container events
  - To update available options

- Getting the state marks the component as dirty

CLIENT SIDE

Reacting to state changes



COM.VAADIN.SHARED.CONNECTOR

- Interface for classes that can communicate
- `com.vaadin.client.ServerConnector`
  - Client-side connector
  - Receives state changes from server-side components
    - Framework takes care of that
  - Implementations must handle changes
- `com.vaadin.server.ClientConnector`
  - Server-side connector
  - Able to send state changes to client-side connectors
    - Framework takes care of that
  - Implementations must prepare state changes

COM.VAADIN.CLIENT.COMPONENTCONNECTOR

- Used by client-side widgets (connectors)
- Reference components in shared state
- Useful subclass
  - Base class for custom connectors
  - com.vaadin.client.AbstractComponentConnector
    - `public void onStateChanged(...);`

```
@Connect(FlatSelect.class)
public class
        FlatSelectConnector
     extends
AbstractComponentConnector
```

FLATSELECTCONNECTOR

Annotation specifies server-side class – it is the only reference to it.
Framework magic!

```java
@Override
protected Widget createWidget() {
  return GWT.create(FlatSelectWidget.class);
}

@Override
public FlatSelectWidget getWidget() {
  return (FlatSelectWidget)super.getWidget();
}

@Override
public FlatSelectState getState() {
  return (FlatSelectState)super.getState();
}
```

FLATSELECTCONNECTOR

Constructs a widget, no need to store reference – handled by GWT.

Helper code to reduce the amount of typecasts ☺ Similar in many connectors.

```
// Whenever the state
// changes in the server-
// side, this method is called
@Override
public void onStateChanged(
                StateChangeEvent event)
{
  super.onStateChanged(event);
  getWidget().setOptions(
    callback,
    getState().options,
    getState().optionsPerRow,
    getState().value
  );
}
```

FLATSELECTCONNECTOR

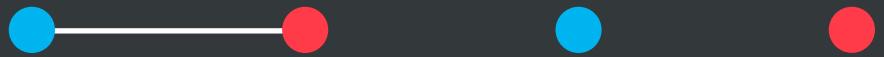
Handles events from widget and calls server's RPC.
Updates widget based on shared state or client's RPC.

FLATSELECTWIDGET

- Extends GWT widget
  - Possible to integrate pure JS components
  - Or write from scratch
- GWT library is huge
  - Still using GWT 2.7, so no Java 8 ☹
- Just the rendering
  - No worries about state

REMOTE PROCEDURE CALLS

On-demand communication



RPC

- Stateless communication
  - A thing happened
    - Button clicked
    - Cursor hovered
- Works both ways
  - Client calls server by making a request
  - Server calls client in the response
- Maven creates working stubs
  - Interfaces
  - Simple implementations
  - Registration and proxies

FROM CLIENT TO SERVER

- `interface FlatSelectServerRpc extends ServerRpc`
  - Defines methods that the client can call
  - `public void selected(int index);`
- Server side has the implementation
  - Must be registered before it is used
  - `registerRpc(implementation);`
- Client side creates a proxy
  - Framework serialises the call into a request
  - ```
    private FlatSelectServerRpc rpc =
        RpcProxy.create(FlatSelectServerRpc.class, this);
    ```
  - `this.rpc.selected(number);`
    ```

FROM SERVER TO CLIENT

- `interface FlatSelectClientRpc extends ClientRpc {...}`
  - Defines methods that the server can call
  - Note that we are **not using it** in the component
    - Everything goes through state
    - Matter of design or preferences what goes where
  - `public void setSelected(int index);`
- Client side has the implementation
  - Must be registered before it is used
  - `registerRpc(Class, implementation);`
- Server side creates a proxy
  - Framework serialises the call into a response
  - `getRpcProxy(FlatSelectClientRpc.class).setSelected(-1);`

DEMO!

FlatSelect
http://github.com/vaadin-miki/flatselect

THE PLAN

- Run the demo
- Explain the code

DEMO!

Shoutbox step 9
http://github.com/vaadin-miki/shoutbox

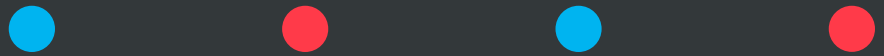
end branch: `step-09`

THE PLAN

- Use `FlatSelect` for room navigation
  - Add a dependency
- When entering a room, add its name to the select
  - Well, to a container assigned to that select
- Navigate to a room when selection changed
- Use cdn for widgetset
  - Nope; despite my best efforts, did not figure how to cdn

SUMMARY

What did we do today



LESSONS OF TODAY (HOPEFULLY)

- Client side development
  - What are connectors?
  - How to use shared state?
- RPC
  - When should RPC be used?
  - How to call methods using RPC?
- Add-ons
  - How to use custom add-ons?

COMING UP NEXT

- Best practices / Mobile First
- Declarative UI / Vaadin Designer
- Web Components and Vaadin Elements
- Quality, debugging and testing
- Progressive Web Applications

THE END

SUGGESTIONS?
QUESTIONS?

miki@vaadin.com
t: @mikiolsz

vaadin}>