Internationalization

Vaadin 14

vaadin}>

i18N vs l10n

i18n(internationalization) = doing a setup that you can support multiple locales. this is done once for an app.

l10n(localization) = adding a new locale to your app. This can be done multiple times to an app.

so you have to internationalize your application so you can localize it.

Internationalization

Vaadin allows you to develop a multilingual application with a **I18NProvider**

Hello!
Moi!
你好！

I18NProvider

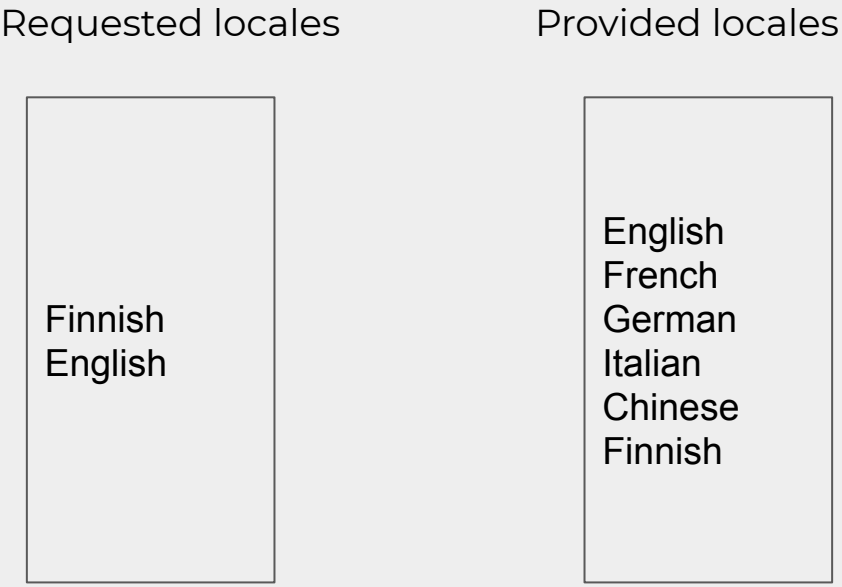
The interface for internationalization. It has two methods to be implemented.

```
public interface I18NProvider extends Serializable {

   List<Locale> getProvidedLocales();

   String getTranslation(String key, Locale locale, Object... params);

}
```

Locale match

The requested locales are read from
`VaadinRequest#getLocales()`

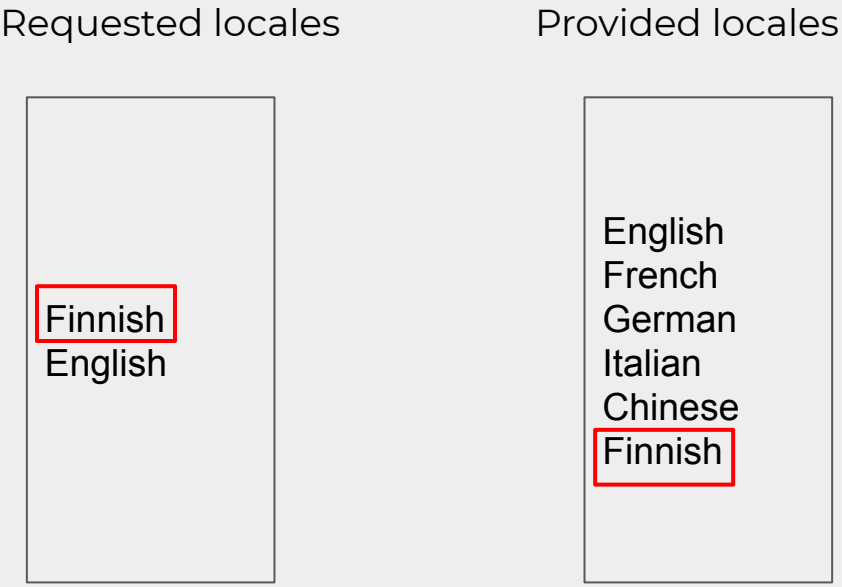
The provided locales are read from
`I18NProvider#getProvidedLocales()`



Locale match

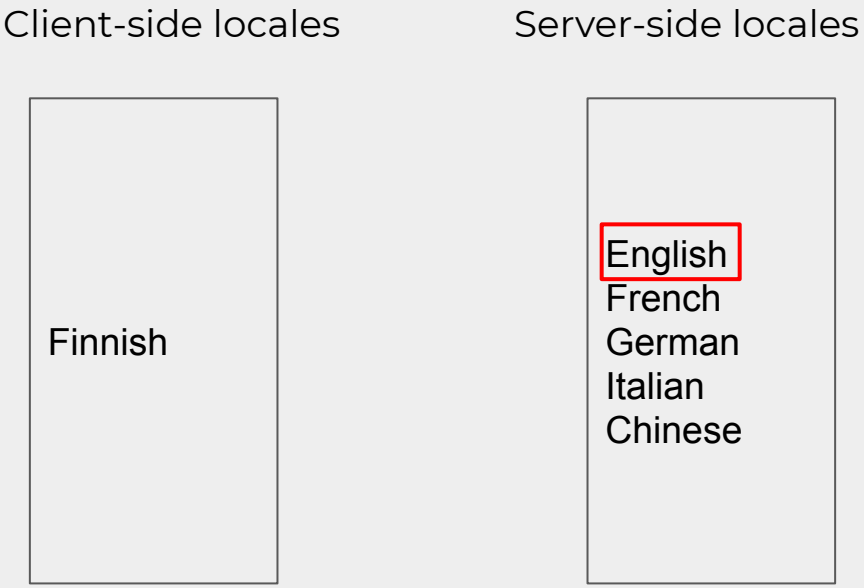
It first tries to find the exact match.

If a match is found, it will be used by setting the value to the current VaadinSession.



Locale match

If no match is found, the **first** one in the provided locales will be used.



Resource Bundle

A resource bundle is a .properties file, which allows the application to load locale-specific data.

The .properties files should have a common base name.

Each .properties file may have suffixes indicating language, country, or platform separated by _.

In a Maven project, the .properties files are stored under the folder src/main/resources.

translation.properties
translation_en.properties
translation_en_US.properties
translation_en_US_UNIX.properties

Resource Bundle

A .properties file has a list of key value pairs.

Use # or ! for comments

```
#translation.properties
btn.key=Click me
```

Resource Bundle in Java

```
//to read a bundle
ResourceBundle bundle = ResourceBundle.getBundle(BUNDLE_BASE_NAME, locale);

//to read the value of a key
String value = bundle.getString(SOME_KEY);
```

An I18NProvider implementation example

```
public class TranslationProvider implements I18NProvider {

   @Override
   public List<Locale> getProvidedLocales() {
       return Collections
               .unmodifiableList(Arrays.asList(new Locale("fi"), new Locale("en")));
   }

   @Override
   public String getTranslation(String key, Locale locale, Object... params) {
       final ResourceBundle bundle = ResourceBundle.getBundle(BUNDLE_BASE_NAME, locale);

       String value= bundle.getString(key);
       if (params.length > 0) {
           value = MessageFormat.format(value, params);
       }
       return value;
   }
}
```

Use localized text for a Component

Component has **getTranslation()** method, which can provide localized text

```
Button button = new Button();
button.setText(getTranslation("btn.key"));
```

Dynamically change the locale

You can change the locale in a UI or the VaadinSession. Change the locale in the VaadinSession will change the locale in all the UIs resides in the session

```
//change the locale in the VaadinSession
VaadinSession.getCurrent().setLocale(newLocale);

//change the locale in a UI
UI.getCurrent().setLocale(newLocale);
```

Listen for locale changes

Use LocalChangeObserver to listen for locale changes

```
public class MainView extends VerticalLayout implements LocaleChangeObserver {

  private final Button button = new Button();

  @Override
  public void localeChange(LocaleChangeEvent event) {
    button.setText(getTranslation("btn.key"));
  }
}
```

Use the I18NProvider

There are 3 different ways of letting your application using the I18NProvider:

- System property
- Custom servlet
- web.xml

Use system property

mvn jetty:run -Dvaadin.i18n.provider=com.vaadin.example.ui.TranslationProvider

Custom Servlet

Could also make a custom servlet, pass the I18NProvider implementation in the @WebInitParam

```
@WebServlet(urlPatterns = "/*",
       initParams = {@WebInitParam(name = Constants.I18N_PROVIDER, value = "com.vaadin.example.ui.TranslationProvider") })
public class ApplicationServlet extends VaadinServlet {

}
```

web.xml

The other equivalent approach is to use the traditional web.xml file

```
<web-app id="WebApp_ID" version="3.0" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

       <servlet>
          <servlet-name>myservlet</servlet-name>
          <servlet-class>com.vaadin.server.VaadinServlet</servlet-class>

          <init-param>
              <param-name>i18n.provider</param-name>
              <param-value>com.vaadin.example.ui.TranslationProvider</param-value>
          </init-param>
       </servlet>

       <servlet-mapping>
          <servlet-name>myservlet</servlet-name>
          <url-pattern>/*</url-pattern>
       </servlet-mapping>
</web-app>
```

Other Runtime configuration

There are many other parameters you can use for runtime configuration purpose, like heartbeat interval, closeIdleSessions etc

For the complete list, visit https://vaadin.com/docs/flow/advanced/tutorial-all-vaadin-properties.html

Exercise

Internationalize a view

Feedback

bit.ly/vaadin-training