Data Binding

The `Property` Interface

- The `Property` interface provides a standardized API for a single data object that allows us to read and write its data.
- When the property value changes, in most cases a `ValueChangeEvent` object will be created. We can handle these events using the `ValueChangeListener` interface.

The `Property` Interface

```
final TextField tf = new TextField("username:");
tf.setValue("...");
tf.addListener(new Property.ValueChangeListener()
{
    public void valueChange(ValueChangeEvent event)
    {
        ...
    }
});
```

The `Viewer` Interface

- The `Property.Viewer` interface defines two methods that allow us to connect every `Property` with a `Viewer`.

```
void setPropertyDataSource(Property newDataSource)
Property getPropertyDataSource()
```

Simple Data Binding

- The `Field` components implement the `Property` interface.

  We can bind any component that implement `Viewer` with any `Field` component.

Simple Data Binding

```
...
TextField editor = new TextField("username");
Label viewer = new Label();
viewer.setPropertyDataSource(editor);
editor.setImmediate(true);
...
```

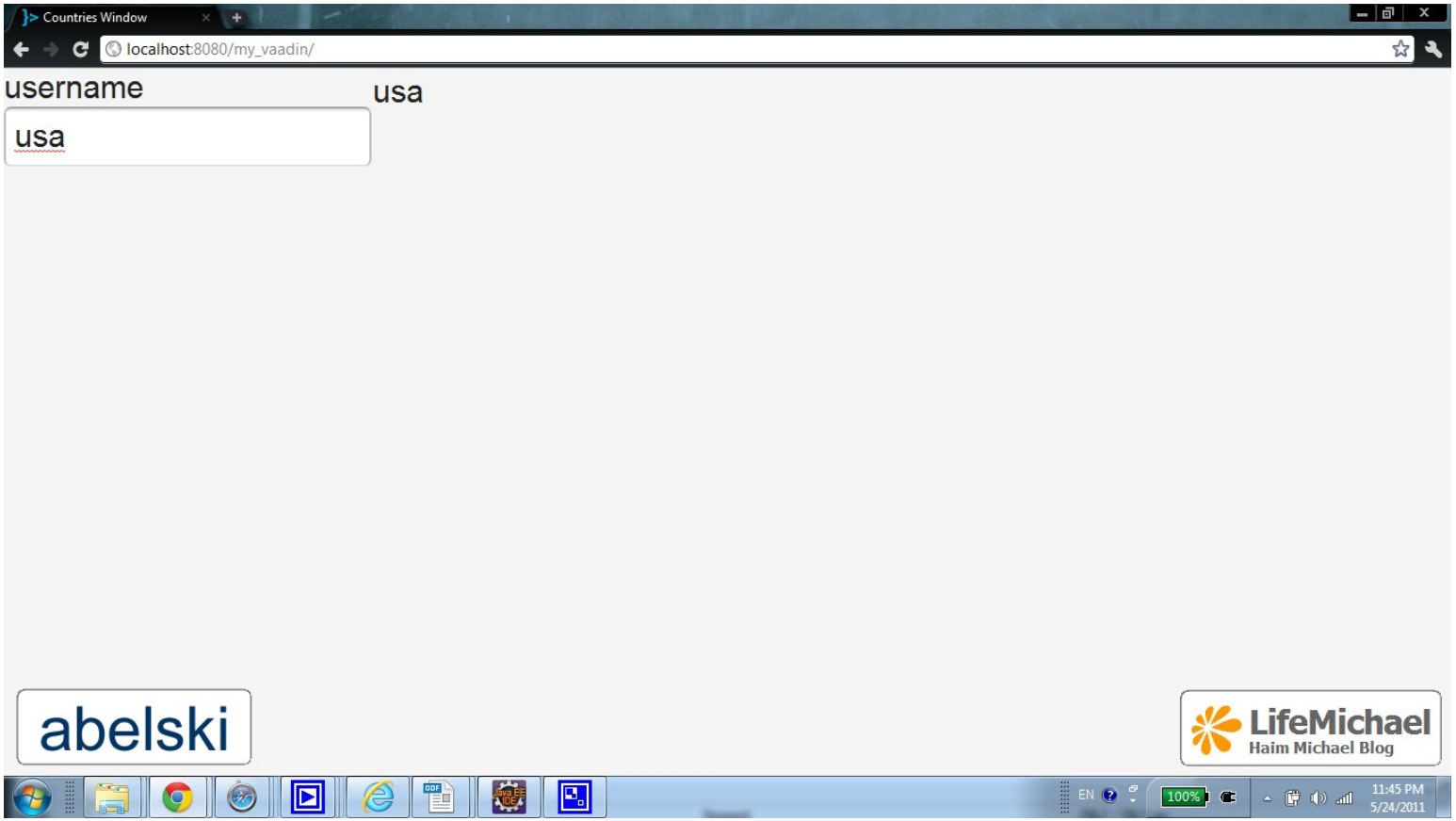
any change in the text field will immediate update the label

Simple Data Binding

```
public class My_vaadinApplication extends Application
{
    private static final long serialVersionUID = 17L;

    @Override
    public void init()
    {
        Window main = new Window("Countries Window");
        HorizontalLayout layout = new HorizontalLayout();
        main.setContent(layout);
        setMainWindow(main);
        TextField editor = new TextField("username");
        Label viewer = new Label();
        viewer.setPropertyDataSource(editor);
        editor.setImmediate(true);
        layout.addComponent(editor);
        layout.addComponent(viewer);
    }
}
```

Simple Data Binding



The `Item` Interface

- The `Item` interface provides access to a set of named properties.
- The `Item` interface defines inner interfaces for maintaining the item properties set and listening to changes made to it.

The `PropertyItem` Class

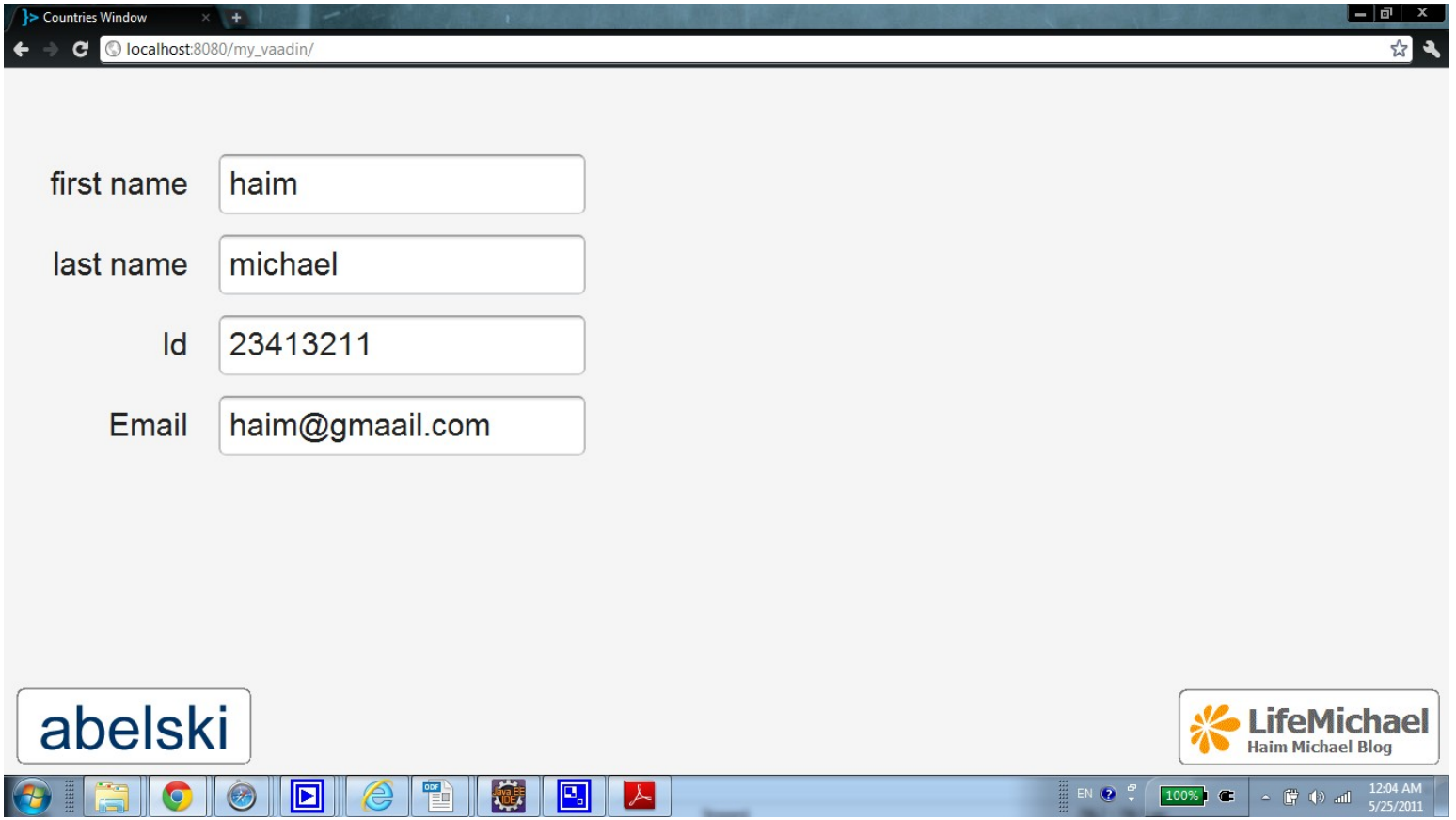
- The `PropertyItem` class is a generic implementation of the `Item` interface.
- We can add properties by calling the `addItemProperty()` method.

The `PropertyItem` Class

```
public class My_vaadinApplication extends Application
{
    private static final long serialVersionUID = 21L;

    @Override
    public void init()
    {
        Window main = new Window("Countries Window");
        PropertysetItem setOfProperties = new PropertysetItem();
        setOfProperties.addItemProperty("first name",
            new ObjectProperty("haim"));
        setOfProperties.addItemProperty("last name",
            new ObjectProperty("michael"));
        setOfProperties.addItemProperty("id",
            new ObjectProperty(23413211));
        setOfProperties.addItemProperty("email",
            new ObjectProperty("haim@gmaail.com"));
        Form form = new Form();
        form.setItemDataSource(setOfProperties);
        setMainWindow(main);
        main.addComponent(form);
    }
}
```

The `PropertyItem` Class



The `BeanItem` Class

- The `BeanItem` class implements the `Item` interface. Instantiating `BeanItem` we get a wrapper for a bean object.
- The getters and the setters are the only methods we should include in our definition for the bean we use. The other requirements are taken care by the `BeanItem` class.

The `BeanItem` Class

```
public class My_vaadinApplication extends Application
{
    private static final long serialVersionUID = 23L;

    @Override
    public void init()
    {
        Window main = new Window("Countries Window");
        Person person = new Person("haim","michael",123123);
        BeanItem<Person> data = new BeanItem<Person>(person);
        Form form = new Form();
        form.setItemDataSource(data);
        setMainWindow(main);
        main.addComponent(form);
    }
}
```

The `BeanItem` Class

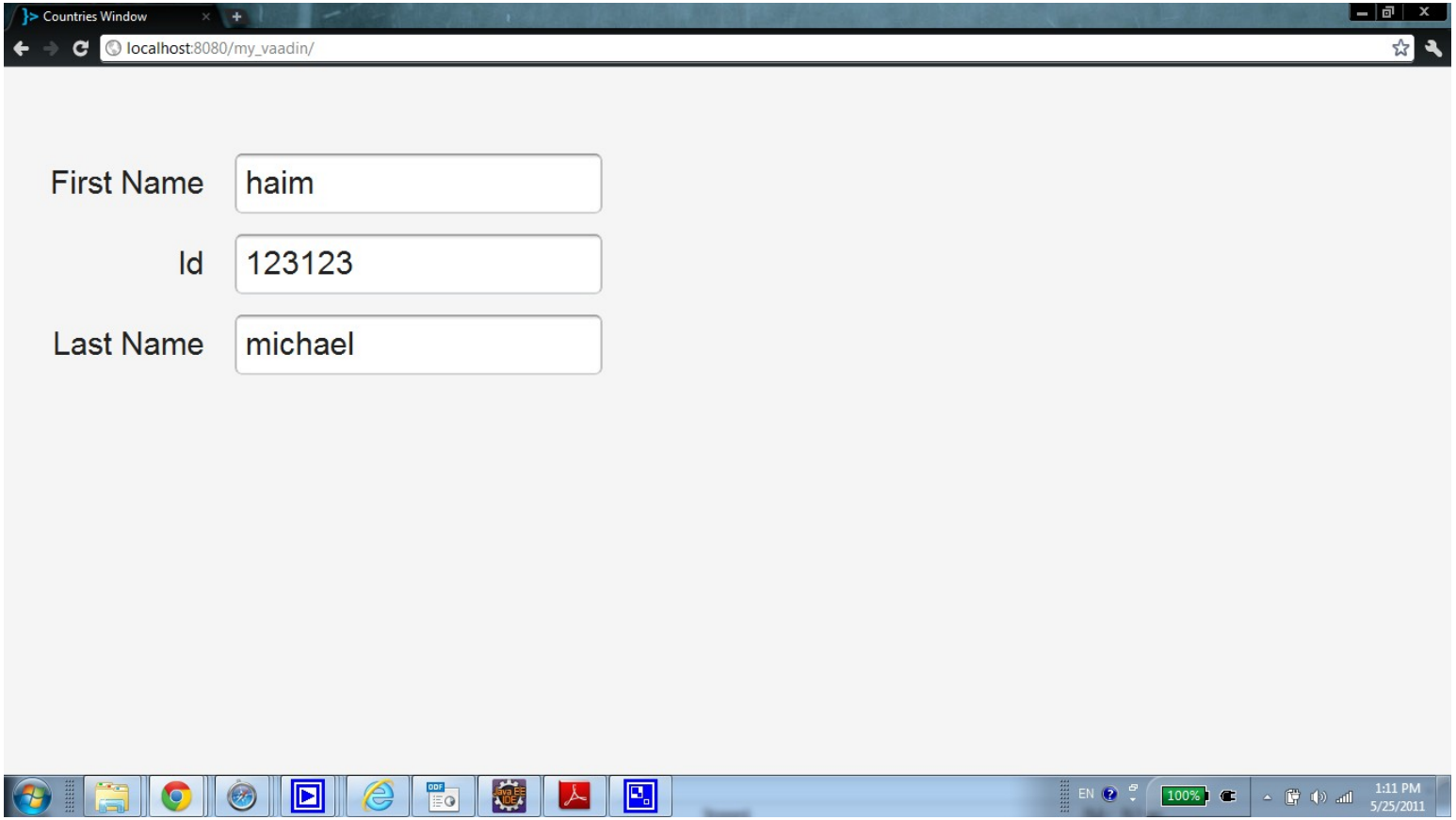
```
package com.example.my_vaadin;

public class Person
{
    private String firstName;
    private String lastName;
    private int id;
    public Person(String firstName, String lastName, int id)
    {
        super();
        this.firstName = firstName;
        this.lastName = lastName;
        this.id = id;
    }
    public String getFirstName()
    {
        return firstName;
    }
    public void setFirstName(String firstName)
    {
        this.firstName = firstName;
    }
```

The `BeanItem` Class

```
    public String getLastName()
    {
        return lastName;
    }
    public void setLastName(String lastName)
    {
        this.lastName = lastName;
    }
    public int getId()
    {
        return id;
    }
    public void setId(int id)
    {
        this.id = id;
    }
}
```

The `BeanItem` Class



The `Container` Interface

- The `Container` interface provides us with a flexible way for managing a set of items that share common properties.
- Each item is identified by its IID (Item IDentifier). We can add items to the container by calling the `addItem()` method.
- The `Container` interface includes inner interfaces that containers can optionally implement for managing the items sequentially.
- The `Table`, `Tree` and `Select` components can be binned with `Container` objects.

The `Container` Interface

- The `Container` supports events for notifying about changes in their contents.
- The Vaadin framework already includes the definition of few useful generic classes that implement `Container`, such as the `IndexedContainer` and the `FileSystemContainer`.

The `Container` Interface

- Many of Vaadin user interface components already implement the `Container` interface. We can bind those components to `View` objects.

The `Container` Interface

```
public class My_vaadinApplication extends Application
{
    private static final long serialVersionUID = 28L;

    @Override
    public void init()
    {
        Window main = new Window("Countries Window");
        setMainWindow(main);
        BeanContainer<Integer, Person> persons =
            new BeanContainer<Integer, Person>(Person.class);
        persons.setBeanIdProperty("id");
        persons.addBean(new Person("haim","michael",123123));
        persons.addBean(new Person("roger","carlma",643454));
        persons.addBean(new Person("dan","yogotasd",743523));
        persons.addBean(new Person("jane","handerson",764563));
        persons.addBean(new Person("rico","michael",345445));
        Table table = new Table("All Persons Table", persons);
        main.addComponent(table);
    }
}
```

The `Container` Interface