Forms and Data Binding

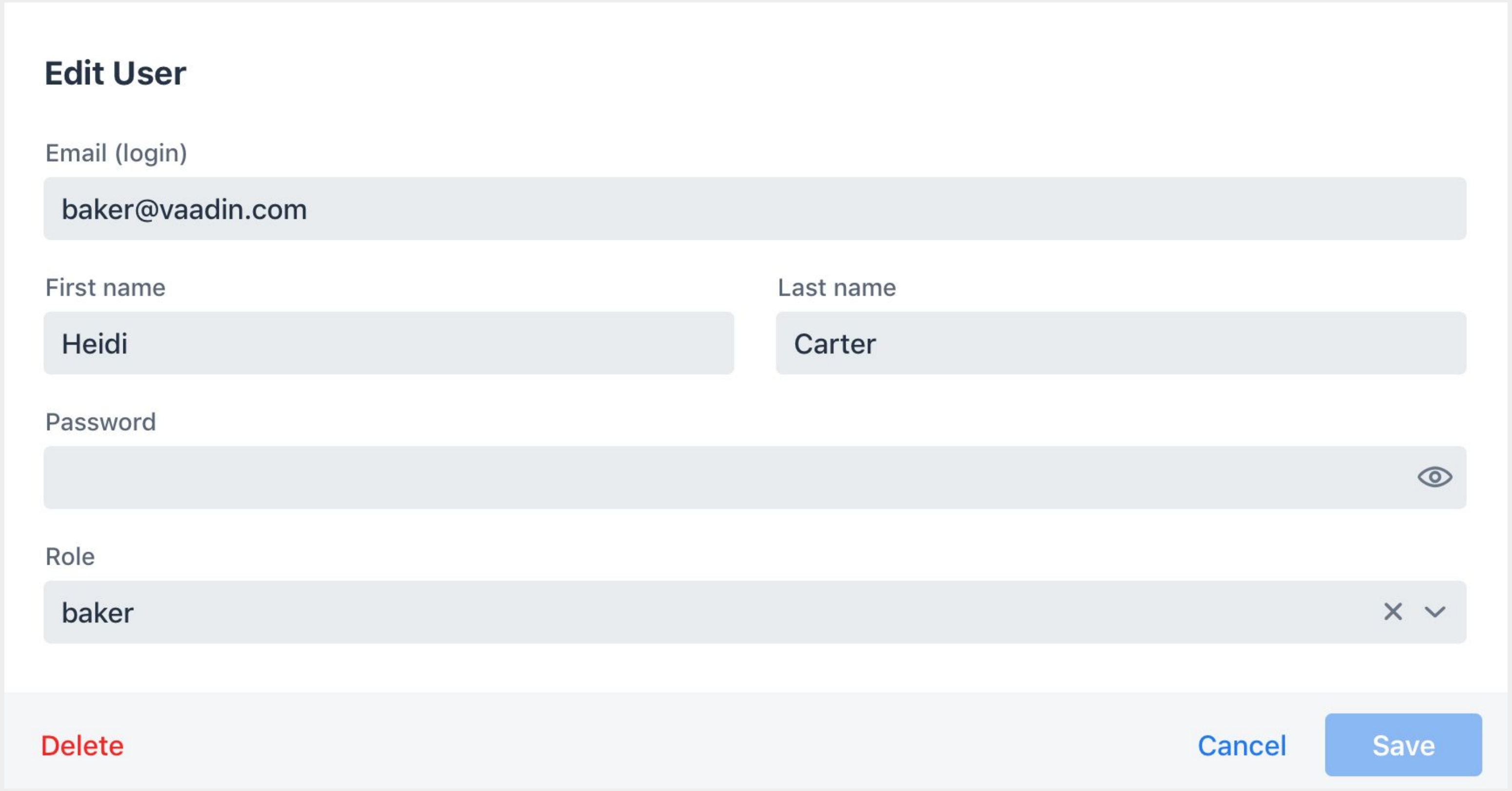
Vaadin 14

Agenda

- Data Binding
- Validation
- Exercise 1
- Conversion
- Custom Field
- Exercise 2

vaadin}>

Form



Edit User

Email (login)

baker@vaadin.com

First name	Last name
Heidi	Carter

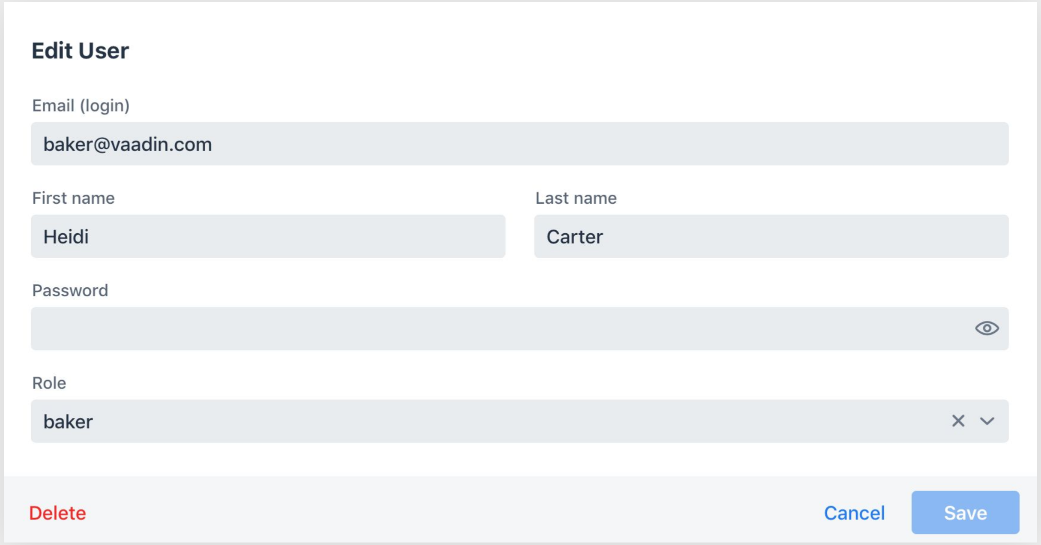
Password

Role

baker

Data binding

```
public class User extends AbstractEntity {

  private String email;

  private String passwordHash;

  private String firstName;

  private String lastName;

  private String role;

  …
  // Properties need getters and setter
}
```



Edit User

Email (login)
baker@vaadin.com

First name
Heidi

Last name
Carter

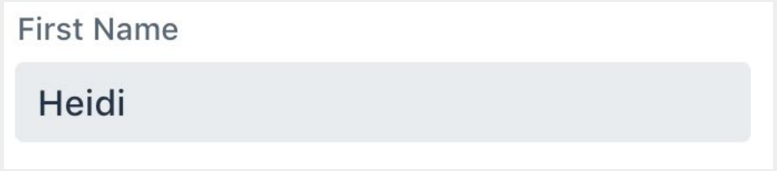
Password

Role
baker

Field

A component that holds value

```
TextField extends Component
          implements HasValue
```



HasValue<E, V>

E specifies the type of the value change event

V specifies the type of the data

HasValue<E, V>

```
V getValue()

void setValue(V value)

Registration addValueChangeListener(ValueChangeListener<? super E> listener)
```

ValueChangeEvent<V>

Value changes originate from two sources:

- Calling `setValue()` in Java code
- User changes the value on the client side

```java
TextField textField = new TextField();

textField.addValueChangeListener(event -> {
   if(event.isFromClient()) {
       //do something
   }
});
```

Data Binding

Data Binding

Vaadin’s data binding system is designed for accessing and modifying application data from UI components

Data can be bound to simple fields, complex forms, and list components

At the core of the data binding is a helper class called **Binder**, which takes care of reading values from the business object(s) and showing it in field Components

Binder

Binds the business data to field components (anything that implements HasValue)

Handles data conversion and validation

Like HasValue, the Binder is always typed to the backing bean, e.g. Binder<Person>

Binder

Instantiation

```
//Option 1: Doesn’t support binding with property names.
Binder<Person> binder = new Binder<>();

//Option 2: Use reflection to scan class for properties so you can bind by property name.
Binder<Person> binder = new Binder<>(Person.class);
```

Creating bindings with Binder

Creating bindings

A binding describes how to move data between a Field and a data object’s Property using Binder.

Bindings are created with a bean’s shared Binder for each Field:

```
Binder<Person> binder = new Binder<>();

TextField titleField = new TextField();              // a Person’s title
IntegerField birthYearField = new IntegerField();    // a Person’s birth year

binder.forField(titleField)...                       // create a binding for titleField
binder.forField(birthYearField)...                   // create a binding for birthYearField
```

There are multiple ways to create bindings, both manual and automatic.

Creating bindings

Binding with method references

```
Binder<Person> binder = new Binder<>();
TextField titleField = new TextField();

binder.forField(titleField)            // Start by defining the Field instance to use
       .bind(                          // Finalize by doing the actual binding to the Person class
              Person::getTitle,        // Callback that loads the title from a person instance
              Person::setTitle));      // Callback that saves the title in a person instance
```

Creating bindings

Binding with property name

```
Binder<Person> binder = new Binder<>(Person.class);
TextField titleField = new TextField();

binder.forField(titleField)
       .bind("title");               // Alternative bind method for Java beans
```

Creating bindings

Binding shorthand

```
Binder<Person> binder = new Binder<>(Person.class);
TextField titleField = new TextField();

//Shorthand for cases without extra configuration
binder.bind(titleField, Person::getTitle, Person::setTitle);

//or with property name
binder.bind(titleField, "title");
```

Creating bindings

Binding with lambda

```
Binder<Person> binder = new Binder<>(Person.class);
TextField titleField = new TextField();

// With lambda expressions
binder.bind(titleField,
       person -> person.getTitle(),
       (person, title) -> person.setTitle(title));
```

Creating bindings

Binding with anonymous class

```java
Binder<Person> binder = new Binder<>(Person.class);
TextField titleField = new TextField();

// With explicit callback interface instances (Java 7 style)
binder.bind(titleField,
       new ValueProvider<Person, String>() {
           @Override
           public String apply(Person person) {
               return person.getTitle();
           }
       },
       new Setter<Person, String>() {
           @Override
           public void accept(Person person, String title) {
               person.setTitle(title);
           }
       });
```

vaadin}>

Creating automatic bindings

Automatic binding with Field name or the @PropertyId annotation

```
public class Person {
   private String name;
   private String email;

   //getter and setter
}
```

```
public class MyForm {
   private TextField name = new TextField();

   @PropertyId("email")
   private TextField emailField = new TextField();

   public MyForm() {
       Binder<Person> binder = new Binder<>(Person.class);
       binder.bindInstanceFields(this);
       // name is now bound to “name”, emailField is bound to “email”
   }
}
```

Creating bindings for nested properties

Bind nested properties with property name

```
public class Person {

    private String name;
    private Address address;

    ...
}

public class Address {

    private String street;
    ...
}
```

```
Binder<Person> binder = new Binder<>(Person.class);

TextField streetField = new TextField();

// this works if you can guarantee “address” is not null
binder.forField(streetField).bind("address.street");
```

Creating bindings for nested properties

Bind nested properties with lambda

```
public class Person {

    private String name;
    private Address address;

    ...
}

public class Address {

    private String street;
    ...
}
```

```
Binder<Person> binder = new Binder<>(Person.class);

TextField streetField = new TextField();

binder.forField(streetField).bind(
        person -> person.getAddress().getStreet(),
        (person, street) -> person.getAddress().setStreet(street));

// Still not safe if getAddress() can return null
```

vaadin}>

Creating bindings for nested properties

Bind nested properties with lambda and a null check

```
public class Person {

    private String name;
    private Address address;

    ...
}

public class Address {

    private String street;
    ...
}
```

```
Binder<Person> binder = new Binder<>(Person.class);

TextField streetField = new TextField();

binder.forField(streetField).bind(
       person -> {
           if(person.getAddress()==null){
               return null;
           }else{
               return person.getAddress().getStreet();
           }
       },
       (person, street) -> {
           if(person.getAddress()!=null){
               person.getAddress().setStreet(street);
           }
           // should we create a new Address object otherwise?
       });
```

Reading and writing values

Buffered or unbuffered?

Reading and writing values

Unbuffered reading and writing

```
Person person = getPerson();
Binder<Person> binder = new Binder<>();

// Unbuffered binding. Fields will immediately write data to the specified bean.
binder.setBean(person);     // Sets the person instance as a data source for the binder
```

Reading and writing values

Buffered reading and writing

```
Person person = getPerson();
Binder<Person> binder = new Binder<>();

// Buffered binding.
binder.readBean(person);    // Reads values from the Person instance to the binder
Button saveButton = new Button("Save", event -> {
   try {
       binder.writeBean(person);   // Writes values from the binder to the person object
   } catch (ValidationException e) {
       // Could not save the values; check exception for each bound field
   }
});
```

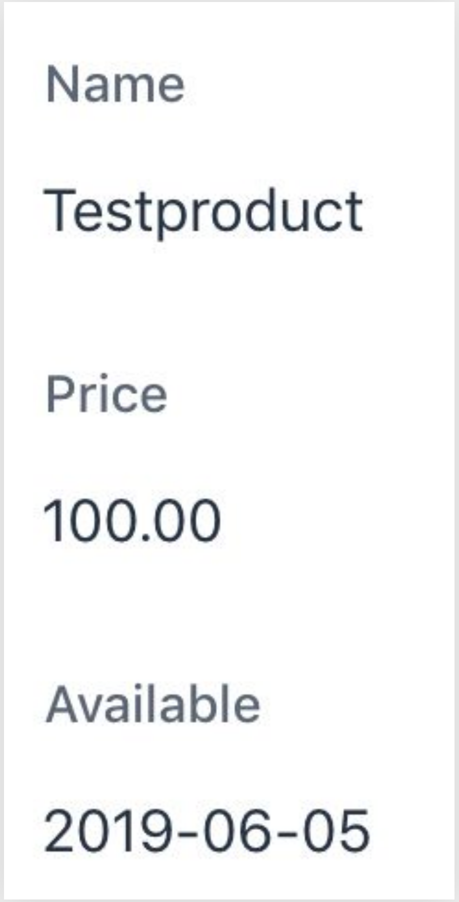
Reading and writing values

Buffered reading and writing / Reset

```java
Person person = getPerson();
Binder<Person> binder = new Binder<>();

// Buffered binding.
binder.readBean(person);    // Reads values from the Person instance to the binder
Button saveButton = new Button("Save", event -> {
   try {
       binder.writeBean(person);   // Writes values from the binder to the person object
   } catch (ValidationException e) {
       // Could not save the values; check exceptions for each bound field
   }
});
Button cancelButton = new Button("Cancel", event ->
       binder.readBean(person)
       // person has not been updated before writeBean -> revert changes in Fields back
);
```

Bind to non-field

Sometimes you might need to bind data to a non-field component, e.g. a Div, Paragraph, Span etc to make the data read only.


Name

Testproduct

Price

100.00

Available

2019-06-05

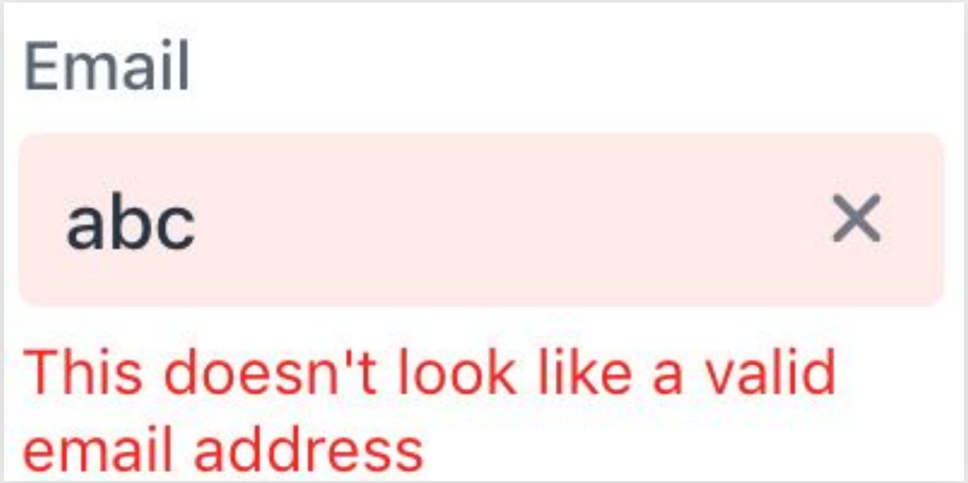
ReadOnlyHasValue

```java
Div nameText = new Div();
ReadOnlyHasValue<String> name = new ReadOnlyHasValue<>(text -> nameText.setText(text));
binder.forField(name).bind(Person::getName, null);
```

Validation

Validation

When validation fails, a field will turn red and display an error message next to it.



This doesn't look like a valid email address

Validation

Validation with an explicit validator

```
binder.forField(emailField)
       // Explicit validator instance
       .withValidator(new EmailValidator("This doesn't look like a valid email address"))
       .bind(Person::getEmail, Person::setEmail);
```

Validation

Validation with lambda

```
binder.forField(nameField)
       // Validator defined based on a lambda and an error message
       .withValidator(
               name -> name.length() >= 3, "Full name must contain at least three characters")
       .bind(Person::getName, Person::setName);
```

Validation

Convenient required validation

```
binder.forField(titleField)
        // Shorthand for requiring the field to be non-empty
        .asRequired("Every employee must have a title")
        .bind(Person::getTitle, Person::setTitle);
```

Validator API

The Validator interface works with two helper interfaces: ValueContext and ValidationResult

```
public interface Validator<T>{

   @Override
   ValidationResult apply(T value, ValueContext context);

}
```

Validator API

The Validator interface works with two helper interfaces: ValueContext and ValidationResult

```
public class MyValidator implements Validator<String> {

   @Override
   public ValidationResult apply(String value, ValueContext context){
       if(value == null || value.length() < 3) {
           return ValidationResult.error(“String is too short”);
       } else {
           return ValidationResult.ok();
       }
   }
}
```

Built-in validators

- BeanValidator
- BigDecimalRangeValidator
- BigIntegerRangeValidator
- ByteRangeValidator
- DateRangeValidator
- DateTimeRangeValidator
- DoubleRangeValidator
- EmailValidator
- FloatRangeValidator
- IntegerRangeValidator
- LongRangeValidator
- RangeValidator
- RegexpValidator
- ShortRangeValidator
- StringLengthValidator

Custom Validation Result Handling

Displays error message(s) in a specified label.

```
Paragraph errMsg = new Paragraph();

binder.forField(nameField).asRequired().withStatusLabel(errMsg).bind("name");

binder.forField(emailField).asRequired().withStatusLabel(errMsg).bind("email");
```

Custom Validation Result Handling

Fully customized validation error handler

```
binder.forField(nameField).asRequired().withValidationStatusHandler(status -> {
    // do sth
}).bind("name");

binder.forField(emailField).asRequired().withValidationStatusHandler(status -> {
    // do sth
}).bind("email");
```

Binder-level validation result handling

Instead of adding status label or status handler to each field, you can also add one to the binder

```
binder.setStatusLabel(errMsg);

binder.setValidationStatusHandler(status -> {
    // do sth
});
```

Binder-level validator

Add validator to binder to do cross-field validation. Binder-level validation will only run if field level validation has passed.

```
/*
* Note that it won’t automatically make any fields invalid or show the error message if it fails.
* Combine it with binder.setStatusLabel() and binder.setValidationStatusHandler() to show the error message and invalidate the fields.
*/
binder.withValidator(flight ->
               flight.getDepartureDate().isBefore(flight.getReturnDate())
                       || flight.getDepartureDate().isEqual(flight.getReturnDate()),
       "Please select a valid date range");
```

Bean Validation (JSR-303)

Use the BeanValidationBinder for Bean validation.

Validation message is read from ValidationMessages.properties resource bundle file

```
public class Product {

  @NotBlank(message = "{bakery.name.required}")
  @Size(max = 255, message = "{bakery.field.max.length}")
  private String name;

}


TextField name = new TextField();
BeanValidationBinder<Product> binder = new BeanValidationBinder<>(Product.class);

binder.bind(nameField, "name");
```

Bean Validation

To use Bean Validation, you can ONLY do the data binding with property name.

Bean Validation

Needs the validation-api and an implementation dependencies

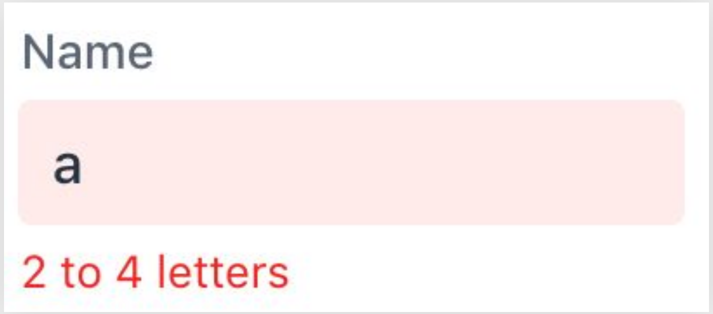
```xml
<!-- API -->
<dependency>
   <groupId>javax.validation</groupId>
   <artifactId>validation-api</artifactId>
   <version>2.0.0.Final</version>
</dependency>

<!-- One possible implementation -->
<dependency>
   <groupId>org.hibernate.validator</groupId>
   <artifactId>hibernate-validator</artifactId>
   <version>6.0.2.Final</version>
</dependency>
```

vaadin}>

Client-side validation

Set validations directly on the field, not through a binder.

```
TextField name = new TextField("Name");
name.setRequired(true);
name.setMinLength(2);
name.setMaxLength(4);
name.setErrorMessage("2 to 4 letters");
```

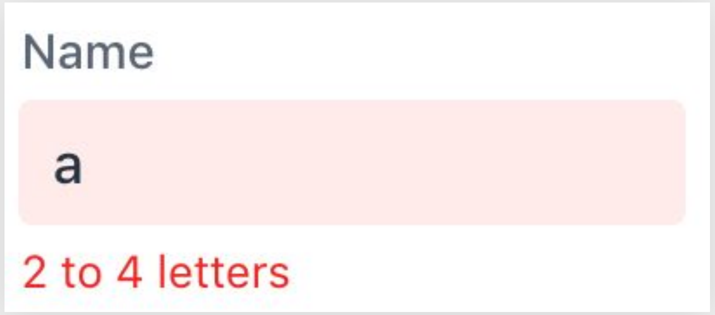


Client-side validation

Looks the same when validation fails.

Validation happens only on the client-side, it doesn't go through the binder.

Avoid client-side validations when using Binder.



Exercise 1

Conversion

Conversion

Both Field and Binder are typed.

When types don’t match, you need a converter.

Conversion

```
TextField age = new TextField("Age");
Binder<Person> binder = new Binder<>(Person.class);
binder.bind(age, "age");
```

```
public class Person {
   private int age;
}
```

```
Caused by: java.lang.ClassCastException:
java.base/java.lang.Integer cannot be cast to
java.base/java.lang.String
	at
com.vaadin.flow.component.textfield.TextField.setVal
ue(TextField.java:32)
	at
com.vaadin.flow.data.binder.Binder$BindingImpl.initF
ieldValue(Binder.java:1130)
	at
com.vaadin.flow.data.binder.Binder$BindingImpl.acces
s$200(Binder.java:971)
	at
com.vaadin.flow.data.binder.Binder.lambda$setBean$1(
Binder.java:1652)
	at
java.base/java.util.ArrayList.forEach(ArrayList.java
:1380)
	at
com.vaadin.flow.data.binder.Binder.setBean(Binder.ja
va:1652)
	at
com.vaadin.starter.skeleton.MainView.<init>(MainView
.java:40)
	... 58 more
```

Converter

Add a converter before binding

```
TextField age = new TextField("Age");
Binder<Person> binder = new Binder<>(Person.class);

binder.forField(age)
       .withConverter(new StringToIntegerConverter("Must enter a number"))
       .bind("age");
```

Converter API

Uses **ValueContext** and **Result**, Convert between **presentation** and **model**

```
public class MyStringToDoubleConverter implements Converter<String, Double> {

   @Override
   public String convertToPresentation(Double value, ValueContext context) {
       return String.format(context.getLocale().get(), "%1$.2f", value);
   }


   @Override
   public Result<Double> convertToModel(String value, ValueContext context) {
       try {
           return Result.ok(Double.parseDouble(value));
       }
       catch (NumberFormatException ex) {
           return Result.error(ex.getMessage());
       }

   }
}
```

Built-in converters

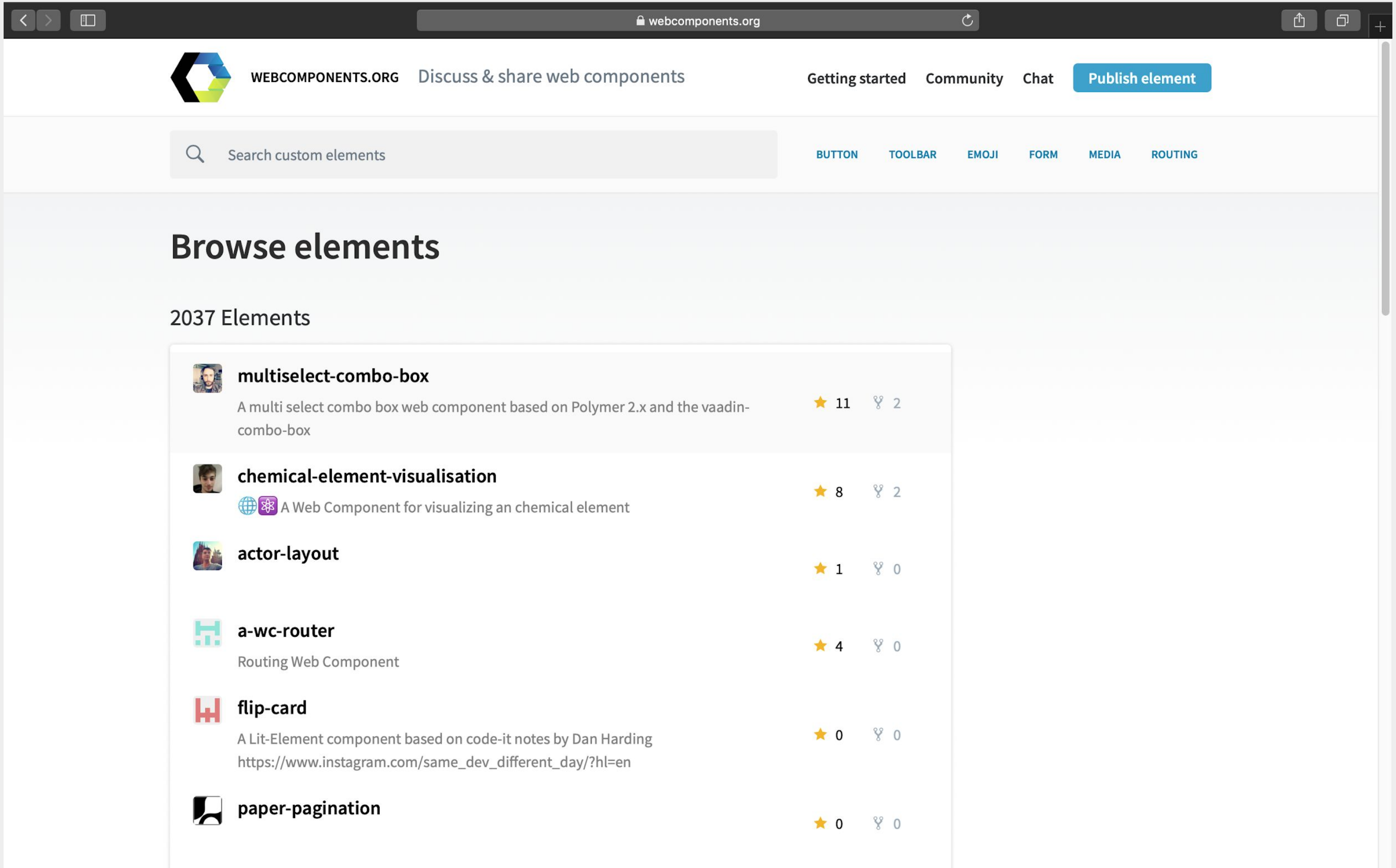
DateToLongConverter
DateToSqlDateConverter
LocalDateTimeToDateConverter
LocalDateToDateConverter
StringToBigDecimalConverter
StringToBigIntegerConverter
StringToBooleanConverter
StringToDateConverter
StringToDoubleConverter
StringToFloatConverter
StringToIntegerConverter
StringToLongConverter

Converters and Validators

You can combine multiple validators and converters, they will be executed in the same order as defined.

```
binder.forField(age)
       .withValidator(value -> validateIsNumber(value), "You can only enter numbers!")
       .withConverter(new StringToIntegerConverter("Not a valid integer!"))
       .withValidator(integer -> isAdult(integer), "Must be 18 or older!")
       .bind("age");
```

Custom Field



Browse elements

2037 Elements

multiselect-combo-box
A multi select combo box web component based on Polymer 2.x and the vaadin-combo-box
★ 11 2

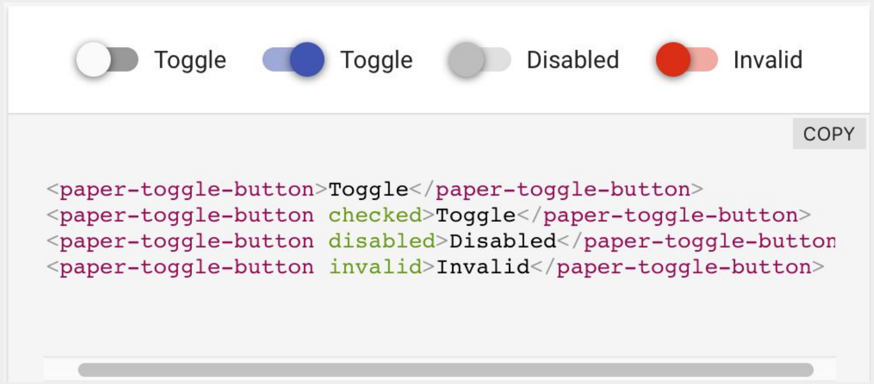
chemical-element-visualisation
🌐⚛️ A Web Component for visualizing an chemical element
★ 8 2

actor-layout
★ 1 0

a-wc-router
Routing Web Component
★ 4 0

flip-card
A Lit-Element component based on code-it notes by Dan Harding https://www.instagram.com/same_dev_different_day/?hl=en
★ 0 0

paper-pagination
★ 0 0

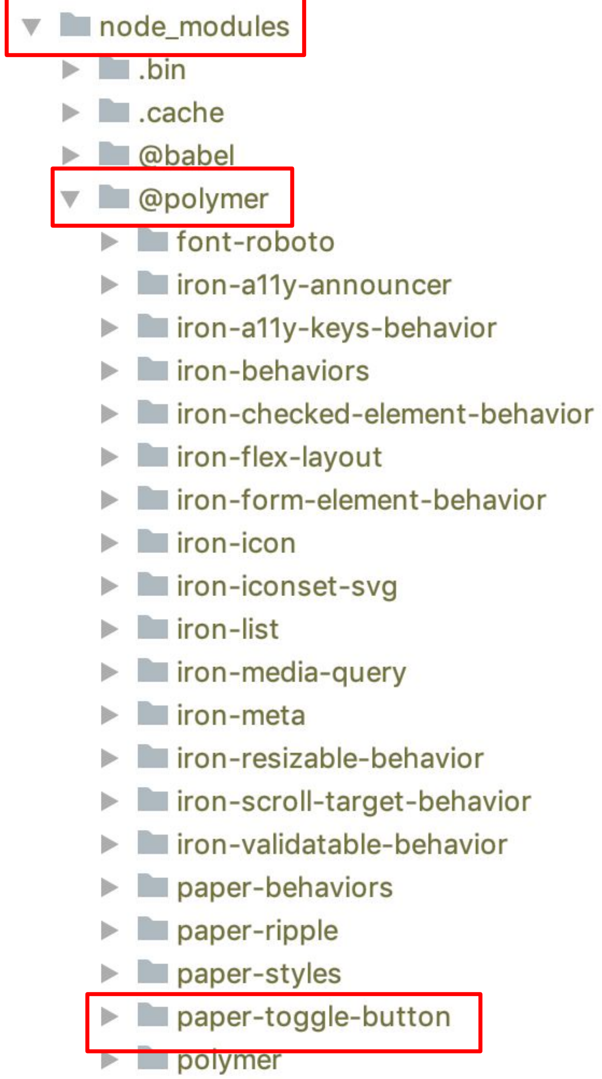


```
<paper-toggle-button>Toggle</paper-toggle-button>
<paper-toggle-button checked>Toggle</paper-toggle-button>
<paper-toggle-button disabled>Disabled</paper-toggle-button
<paper-toggle-button invalid>Invalid</paper-toggle-button>
```

Install

Run npm install on the project's root directory

```
npm install --save @polymer/paper-toggle-button
```



```
▼ node_modules
  ▶ .bin
  ▶ .cache
  ▶ @babel
  ▼ @polymer
    ▶ font-roboto
    ▶ iron-a11y-announcer
    ▶ iron-a11y-keys-behavior
    ▶ iron-behaviors
    ▶ iron-checked-element-behavior
    ▶ iron-flex-layout
    ▶ iron-form-element-behavior
    ▶ iron-icon
    ▶ iron-iconset-svg
    ▶ iron-list
    ▶ iron-media-query
    ▶ iron-meta
    ▶ iron-resizable-behavior
    ▶ iron-scroll-target-behavior
    ▶ iron-validatable-behavior
    ▶ paper-behaviors
    ▶ paper-ripple
    ▶ paper-styles
    ▶ paper-toggle-button
    ▶ polymer
```

Web Component as a Field

Many components usually has a property which holds the value

```
@JsModule("@polymer/paper-toggle-button/paper-toggle-button.js")
@Tag("paper-toggle-button")
public class ToggleButton extends AbstractSinglePropertyField<ToggleButton, Boolean> {
   public ToggleButton() {
       //property name, default value, accept null values
       super("checked", false, false);
   }
}
```

Limitation

AbstractSinglePropertyField holds the value properly, but it doesn’t have validation error message or required indicator.

CustomField

CustomField is a wrapper of component(s) which can be used as a regular field.

CustomField has a validation error message and required indicator.

Make a CustomField

Extend from CustomField which takes a generic parameter for the data type

```java
public class MyCustomField extends CustomField<Boolean> {










}
```

Make a CustomField

Set up the content of the CustomField

```java
public class MyCustomField extends CustomField<Boolean> {










}
```

Make a CustomField

Set up the content of the CustomField

```
public class MyCustomField extends CustomField<Boolean> {

   private ToggleButton toggleButton = new ToggleButton();

   public MyCustomField(){
       add(toggleButton);
   }








}
```

Make a CustomField

Needs to implement the **generateModelValue()** and **setPresentationValue()** methods

```
public class MyCustomField extends CustomField<Boolean> {

   private ToggleButton toggleButton = new ToggleButton();

   public MyCustomField(){
       add(toggleButton);
   }

   @Override
   protected Boolean generateModelValue() {
       return toggleButton.getValue();
   }

   @Override
   protected void setPresentationValue(Boolean newPresentationValue) {
       toggleButton.setValue(newPresentationValue);
   }
}
```

Make a CustomField

Could also combine multiple components/fields

```java
public class MyCustomField extends CustomField<LocalDateTime> {

   private DatePicker datePicker = new DatePicker();
   private TimePicker timePicker = new TimePicker();

   public MyCustomField(){
       add(datePicker, timePicker);
   }

   @Override
   protected LocalDateTime generateModelValue() {
       return LocalDateTime.of(datePicker.getValue(), timePicker.getValue());
   }

   @Override
   protected void setPresentationValue(LocalDateTime newPresentationValue) {
       datePicker.setValue(newPresentationValue.toLocalDate());
       timePicker.setValue(newPresentationValue.toLocalTime());
   }

}
```

Exercise 2

Summary

- Data Binding
- Validation
- Conversion
- Custom Field

Feedback

bit.ly/vaadin-training