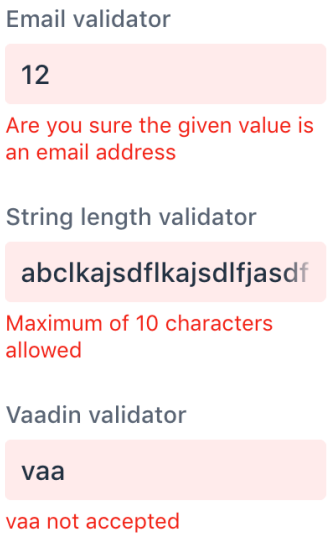


Exercises

Exercise 1: Validation

An essential part of manipulating user input is validating the input values. In this exercise we practice applying validators on fields.

You will start with a view that has three `TextField` components in it. Each `TextField` should have its own validator connected to a binder which is bound to the field. The validators you need to use are: `EmailValidator`, `StringLengthValidator` and a custom validator.



- EmailValidator should only accept valid email addresses
- String length validator should accept strings with a maximum length of ten characters
- The custom validator should implement the Validator Interface and check if the user entered “Vaadin”

Hints:

- To get started, you need to implement a simple java bean to use with the `Binder` instance. Your bean should have three properties, one for each `TextField`.
- Bind the fields using `binder.forField(textField)`
- Each of the fields have ready-made validator classes, except for the last one.
- If you encounter null value exceptions with the binder, take a look at the method `Binder.BindingBuilder.withNullRepresentation(String)`

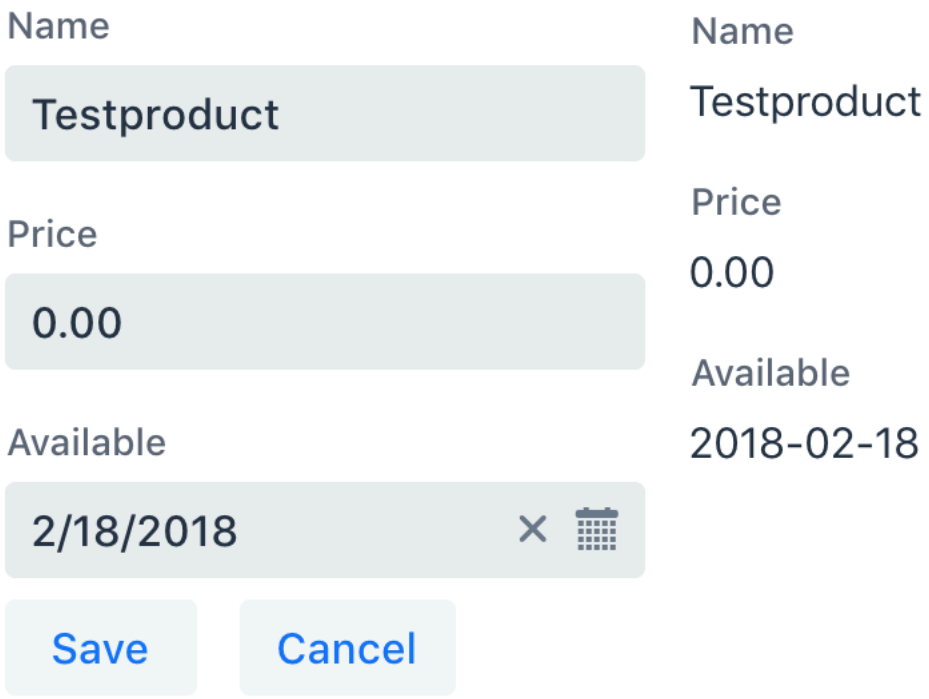
Bonus Task:

Try to use status label to show the error messages. You can just define one Label added to the top, all the fields could share the same Label as their status label.

Exercise 2: Binding Data to Forms

In this exercise, we want to create a form for editing a `Product` bean and display the current values of its properties in a read-only view on the right hand side. Your task is to create the form for editing the product properties and use the `Binder` helper class to bind the product object to the fields of the form.

The required components are `TextField` and `DatePicker`. When you click on save, the values from the Binder should be committed into the product bean and the read-only view updated. If you click on cancel, any changes in the form should be reverted (essentially reading back the values from the product bean).



Steps for this exercise:
1. Implement the ProductEditor class, which is the form for product.
2. Implement the ProductViewer class, which is the read-only view.

Hints:
- If you need to use a converter for a field, you need to do it like this: `binder.forField(field).withConverter(…).bind(…)`
- The rest of the fields can be annotated with @PropertyId(“propName”) and bound with `binder.bindInstanceFields(editorForm)`

Bonus task: If you are quick with this exercise, try implementing a Converter that allows you to enter currencies to the Price field. In the application below, I can enter euro values with either the postfix “€” or “EUR” and it will be correctly interpreted as a double value.