<https://fullstackopen.com/en/part2/forms>

<https://github.com/fullstack-hy2020/part2-notes/tree/part2-2/>

<https://github.com/hiadore/fullstackopen/commits>

<https://github.com/anancarv/fullstackopen>

**b**

**Forms**

Let's continue expanding our application by allowing users to add new notes. You can find the code for our current application [here](https://github.com/fullstack-hy2020/part2-notes/tree/part2-1).

In order to get our page to update when new notes are added it's best to store the notes in the *App* component's state. Let's import the [useState](https://reactjs.org/docs/hooks-state.html" \t "_blank) function and use it to define a piece of state that gets initialized with the initial notes array passed in the props.

import { useState } from 'react'

import Note from './components/Note'

const App = (props) => { const [notes, setNotes] = useState(props.notes)

return (

<div>

<h1>Notes</h1>

<ul>

{notes.map(note =>

<Note key={note.id} note={note} />

)}

</ul>

</div>

)

}

export default App

The component uses the useState function to initialize the piece of state stored in notes with the array of notes passed in the props:

const App = (props) => {

const [notes, setNotes] = useState(props.notes)

// ...

}

If we wanted to start with an empty list of notes, we would set the initial value as an empty array, and since the props would not be used, we could omit the props parameter from the function definition:

const App = () => {

const [notes, setNotes] = useState([])

// ...

}

Let's stick with the initial value passed in the props for the time being.

Next, let's add an HTML [form](https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms) to the component that will be used for adding new notes.

const App = (props) => {

const [notes, setNotes] = useState(props.notes)

const addNote = (event) => { event.preventDefault() console.log('button clicked', event.target) }

return (

<div>

<h1>Notes</h1>

<ul>

{notes.map(note =>

<Note key={note.id} note={note} />

)}

</ul>

<form onSubmit={addNote}> <input /> <button type="submit">save</button> </form> </div>

)

}

We have added the addNote function as an event handler to the form element that will be called when the form is submitted, by clicking the submit button.

We use the method discussed in [part 1](https://fullstackopen.com/en/part1/component_state_event_handlers#event-handling) for defining our event handler:

const addNote = (event) => {

event.preventDefault()

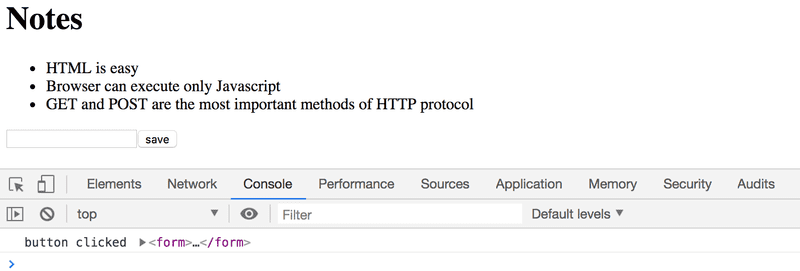
console.log('button clicked', event.target)

}

The event parameter is the [event](https://reactjs.org/docs/handling-events.html) that triggers the call to the event handler function:

The event handler immediately calls the event.preventDefault() method, which prevents the default action of submitting a form. The default action would, [among other things](https://developer.mozilla.org/en-US/docs/Web/API/HTMLFormElement/submit_event), cause the page to reload.

The target of the event stored in event.target is logged to the console:



The target in this case is the form that we have defined in our component.

How do we access the data contained in the form's *input* element?

**Controlled component**

There are many ways to accomplish this; the first method we will take a look at is through the use of so-called [controlled components](https://reactjs.org/docs/forms.html#controlled-components).

Let's add a new piece of state called newNote for storing the user-submitted input **and** let's set it as the *input* element's *value* attribute:

const App = (props) => {

const [notes, setNotes] = useState(props.notes)

const [newNote, setNewNote] = useState( 'a new note...' )

const addNote = (event) => {

event.preventDefault()

console.log('button clicked', event.target)

}

return (

<div>

<h1>Notes</h1>

<ul>

{notes.map(note =>

<Note key={note.id} note={note} />

)}

</ul>

<form onSubmit={addNote}>

<input value={newNote} /> <button type="submit">save</button>

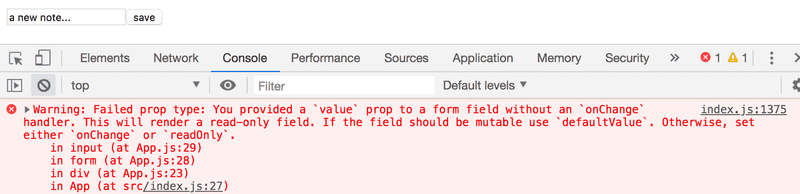
</form>

</div>

)

}

The placeholder text stored as the initial value of the newNote state appears in the *input* element, but the input text can't be edited. The console displays a warning that gives us a clue as to what might be wrong:



Since we assigned a piece of the *App* component's state as the *value* attribute of the input element, the *App* component now [controls](https://reactjs.org/docs/forms.html#controlled-components) the behavior of the input element.

In order to enable editing of the input element, we have to register an *event handler* that synchronizes the changes made to the input with the component's state:

const App = (props) => {

const [notes, setNotes] = useState(props.notes)

const [newNote, setNewNote] = useState(

'a new note...'

)

// ...

const handleNoteChange = (event) => { console.log(event.target.value) setNewNote(event.target.value) }

return (

<div>

<h1>Notes</h1>

<ul>

{notes.map(note =>

<Note key={note.id} note={note} />

)}

</ul>

<form onSubmit={addNote}>

<input

value={newNote}

onChange={handleNoteChange} />

<button type="submit">save</button>

</form>

</div>

)

}

We have now registered an event handler to the *onChange* attribute of the form's *input* element:

<input

value={newNote}

onChange={handleNoteChange}

/>

The event handler is called every time *a change occurs in the input element*. The event handler function receives the event object as its event parameter:

const handleNoteChange = (event) => {

console.log(event.target.value)

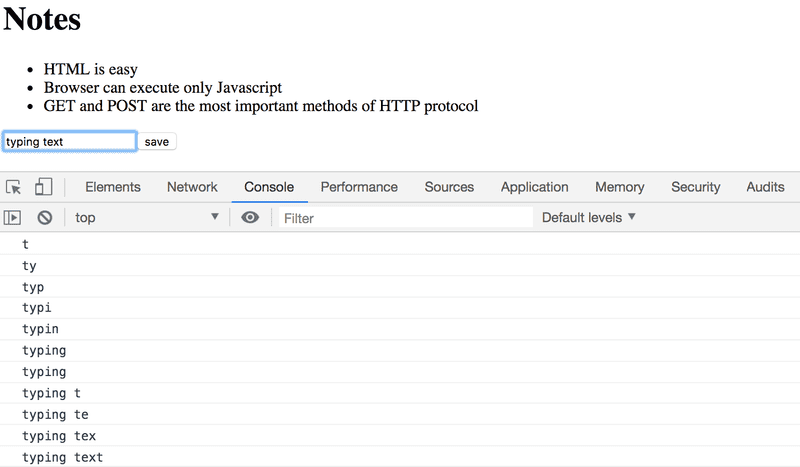
setNewNote(event.target.value)

}

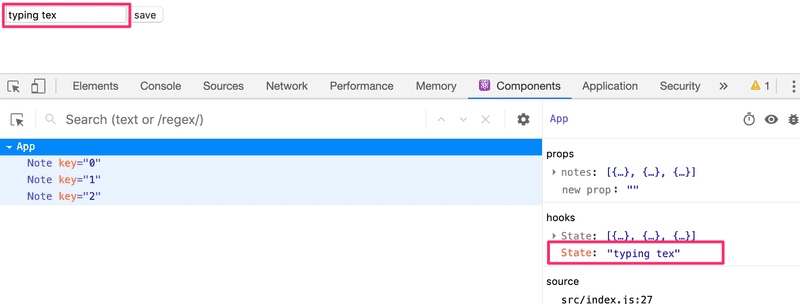
The target property of the event object now corresponds to the controlled *input* element, and event.target.value refers to the input value of that element.

Note that we did not need to call the event.preventDefault() method like we did in the *onSubmit* event handler. This is because there is no default action that occurs on an input change, unlike on a form submission.

You can follow along in the console to see how the event handler is called:



You did remember to install [React devtools](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi), right? Good. You can directly view how the state changes from the React Devtools tab:



Now the *App* component's newNote state reflects the current value of the input, which means that we can complete the addNote function for creating new notes:

const addNote = (event) => {

event.preventDefault()

const noteObject = {

content: newNote,

date: new Date().toISOString(),

important: Math.random() < 0.5,

id: notes.length + 1,

}

setNotes(notes.concat(noteObject))

setNewNote('')

}

First, we create a new object for the note called noteObject that will receive its content from the component's newNote state. The unique identifier *id* is generated based on the total number of notes. This method works for our application since notes are never deleted. With the help of the Math.random() function, our note has a 50% chance of being marked as important.

The new note is added to the list of notes using the [concat](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/concat" \t "_blank) array method, introduced in [part 1](https://fullstackopen.com/en/part1/java_script#arrays):

setNotes(notes.concat(noteObject))

The method does not mutate the original notes array, but rather creates *a new copy of the array with the new item added to the end*. This is important since we must [never mutate state directly](https://reactjs.org/docs/state-and-lifecycle.html#using-state-correctly) in React!

The event handler also resets the value of the controlled input element by calling the setNewNote function of the newNote state:

setNewNote('')

You can find the code for our current application in its entirety in the *part2-2* branch of [this GitHub repository](https://github.com/fullstack-hy2020/part2-notes/tree/part2-2).

  // we have to register an event handler that synchronizes the changes

  // mow we can also add columns

import { useState } from 'react'

import Note from './components/Note'

const App = (props) => {

    const [notes, setNotes] = useState(props.notes)

    const [newNote, setNewNote] = useState(

      'a new note...'

    )

    const addNote = (event) => {

        event.preventDefault()

        const noteObject = {

          content: newNote,

          date: new Date().toISOString(),

          important: Math.random() < 0.5,

          id: notes.length + 1,

        }

        setNotes(notes.concat(noteObject))

        setNewNote('')

      }

    const handleNoteChange = (event) => {

        console.log(event.target.value)

        setNewNote(event.target.value)

    }

    return (

        <div>

          <h1>Notes</h1>

          <ul>

            {notes.map(note =>

              <Note key={note.id} note={note} />

            )}

          </ul>

          <form onSubmit={addNote}>

            <input

              value={newNote}

              onChange={handleNoteChange}

            />

            <button type="submit">save</button>

          </form>

        </div>

      )

    }

  export default App

Note.js

import React from 'react'

const Note = ({ note }) => {

  return (

    <li>{note.content}</li>

  )

}

export default Note

index.js

import React from 'react'

import ReactDOM from 'react-dom/client'

// import App from './P2\_a\_rendering\_a\_collection\_modules/App'

import App from './P2\_b\_forms/App'

const notes = [

  {

    id: 1,

    content: 'HTML is easy',

    date: '2019-05-30T17:30:31.098Z',

    important: true

  },

  {

    id: 2,

    content: 'Browser can execute only JavaScript',

    date: '2019-05-30T18:39:34.091Z',

    important: false

  },

  {

    id: 3,

    content: 'GET and POST are the most important methods of HTTP protocol',

    date: '2019-05-30T19:20:14.298Z',

    important: true

  }

]

ReactDOM.createRoot(document.getElementById('root')).render(

  <App notes={notes} />

)

**Filtering Displayed Elements**

Let's add some new functionality to our application that allows us to only view the important notes.

Let's add a piece of state to the *App* component that keeps track of which notes should be displayed:

const App = (props) => {

const [notes, setNotes] = useState(props.notes)

const [newNote, setNewNote] = useState('')

const [showAll, setShowAll] = useState(true)

// ...

}

Let's change the component so that it stores a list of all the notes to be displayed in the notesToShow variable. The items of the list depend on the state of the component:

import { useState } from 'react'

import Note from './components/Note'

const App = (props) => {

const [notes, setNotes] = useState(props.notes)

const [newNote, setNewNote] = useState('')

const [showAll, setShowAll] = useState(true)

// ...

const notesToShow = showAll ? notes : notes.filter(note => note.important === true)

return (

<div>

<h1>Notes</h1>

<ul>

{notesToShow.map(note => <Note key={note.id} note={note} />

)}

</ul>

// ...

</div>

)

}

The definition of the notesToShow variable is rather compact:

const notesToShow = showAll

? notes

: notes.filter(note => note.important === true)

The definition uses the [conditional](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator) operator also found in many other programming languages.

The operator functions as follows. If we have:

const result = condition ? val1 : val2

the result variable will be set to the value of val1 if condition is true. If condition is false, the result variable will be set to the value ofval2.

If the value of showAll is false, the notesToShow variable will be assigned to a list that only contains notes that have the important property set to true. Filtering is done with the help of the array [filter](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter) method:

notes.filter(note => note.important === true)

The comparison operator is in fact redundant, since the value of note.important is either *true* or *false*, which means that we can simply write:

notes.filter(note => note.important)

The reason we showed the comparison operator first was to emphasize an important detail: in JavaScript val1 == val2 does not work as expected in all situations and it's safer to use val1 === val2 exclusively in comparisons. You can read more about the topic [here](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness).

You can test out the filtering functionality by changing the initial value of the showAll state.

Next, let's add functionality that enables users to toggle the showAll state of the application from the user interface.

The relevant changes are shown below:

import { useState } from 'react'

import Note from './components/Note'

const App = (props) => {

const [notes, setNotes] = useState(props.notes)

const [newNote, setNewNote] = useState('')

const [showAll, setShowAll] = useState(true)

// ...

return (

<div>

<h1>Notes</h1>

<div> <button onClick={() => setShowAll(!showAll)}> show {showAll ? 'important' : 'all' } </button> </div> <ul>

{notesToShow.map(note =>

<Note key={note.id} note={note} />

)}

</ul>

// ...

</div>

)

}

The displayed notes (all versus important) are controlled with a button. The event handler for the button is so simple that it has been defined directly in the attribute of the button element. The event handler switches the value of showAll from true to false and vice versa:

() => setShowAll(!showAll)

The text of the button depends on the value of the showAll state:

show {showAll ? 'important' : 'all'}

You can find the code for our current application in its entirety in the *part2-3* branch of [this GitHub repository](https://github.com/fullstack-hy2020/part2-notes/tree/part2-3).

**Exercises 2.6.-2.10.**

In the first exercise, we will start working on an application that will be further developed in the later exercises. In related sets of exercises it is sufficient to return the final version of your application. You may also make a separate commit after you have finished each part of the exercise set, but doing so is not required.

**WARNING** create-react-app will automatically turn your project into a git-repository unless you create your application inside of an existing git repository. It's likely that you **do not want** your project to be a repository, so simply run the rm -rf .git command at the root of your application.

**2.6: The Phonebook Step1**

Let's create a simple phonebook. ***In this part we will only be adding names to the phonebook.***

Let us start by implementing the addition of a person to phonebook.

You can use the code below as a starting point for the *App* component of your application:

import { useState } from 'react'

const App = () => {

const [persons, setPersons] = useState([

{ name: 'Arto Hellas' }

])

const [newName, setNewName] = useState('')

return (

<div>

<h2>Phonebook</h2>

<form>

<div>

name: <input />

</div>

<div>

<button type="submit">add</button>

</div>

</form>

<h2>Numbers</h2>

...

</div>

)

}

export default App

## Phonebook

Top of Form

name: 

add

Bottom of Form

## Numbers

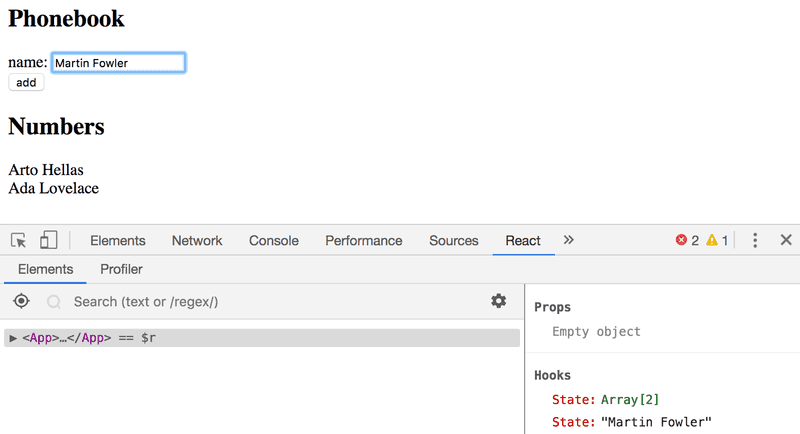
The newName state is meant for controlling the form input element.

Sometimes it can be useful to render state and other variables as text for debugging purposes. You can temporarily add the following element to the rendered component:

<div>debug: {newName}</div>

It's also important to put what we learned in the [debugging React applications](https://fullstackopen.com/en/part1/a_more_complex_state_debugging_react_apps) chapter of part one into good use. The [React developer tools](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi) extension especially, is incredibly useful for tracking changes that occur in the application's state.

After finishing this exercise your application should look something like this:



Note the use of the React developer tools extension in the picture above!

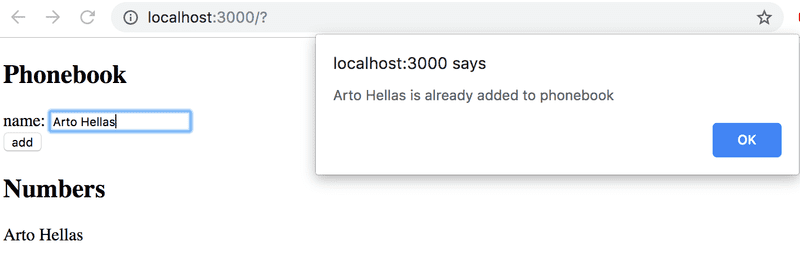
**NB:**

* you can use the person's name as value of the *key* property
* remember to prevent the default action of submitting HTML forms!
* // 2.6: The Phonebook Step1
* import React, { useState } from 'react';
* function App() {
* const [ persons, setPersons] = useState([
* { name: 'Arto Hellas' }
* ])
* const [ newName, setNewName ] = useState('')
* const addPerson = (event) => {
* event.preventDefault()
* const personObject = {
* name: newName
* }
* console.log(personObject)
* setPersons(persons.concat(personObject))
* }
* const handleNameChange = (event) => {
* console.log(event.target.value)
* setNewName(event.target.value)
* }
* const row\_names = () => persons.map(person =>
* <p key={person.name}>{person.name}</p>
* )
* return (
* <div>
* <h2>Phonebook</h2>
* <form onSubmit={addPerson}>
* <div>
* name: <input
* value={newName}
* onChange={handleNameChange}
* />
* </div>
* <div>
* <button type="submit">add</button>
* </div>
* </form>
* <h2>Numbers</h2>
* <>{row\_names()}</>
* </div>
* )
* }
* export default App;

**2.7: The Phonebook Step2**

Prevent the user from being able to add names that already exist in the phonebook. JavaScript arrays have numerous suitable [methods](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) for accomplishing this task. Keep in mind [how object equality works](https://www.joshbritz.co/posts/why-its-so-hard-to-check-object-equality/) in Javascript.

Issue a warning with the [alert](https://developer.mozilla.org/en-US/docs/Web/API/Window/alert) command when such an action is attempted:



**Hint:** when you are forming strings that contain values from variables, it is recommended to use a [template string](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals):

`${newName} is already added to phonebook`

If the newName variable holds the value *Arto Hellas*, the template string expression returns the string

`Arto Hellas is already added to phonebook`

The same could be done in a more Java-like fashion by using the plus operator:

newName + ' is already added to phonebook'

Using template strings is the more idiomatic option and the sign of a true JavaScript professional.

import React, { useState } from 'react';

function App() {

  const [ persons, setPersons] = useState([

    { name: 'Arto Hellas' }

  ])

  const [ newName, setNewName ] = useState('')

  const addPerson = (event) => {

    event.preventDefault()

    const personObject = {

      name: newName

    }

    console.log(personObject)

    const all\_names = persons.map(person => person.name)

    if (all\_names.includes(newName)) {

      alert(`${newName} is already added to phonebook`)

      return

    }

    setPersons(persons.concat(personObject))

    setNewName('')

  }

  const handleNameChange = (event) => {

    console.log(event.target.value)

    setNewName(event.target.value)

  }

  const row\_names = () => persons.map(person =>

    <p key={person.name}>{person.name}</p>

  )

  return (

    <div>

      <h2>Phonebook</h2>

      <form onSubmit={addPerson}>

        <div>

          name: <input

            value={newName}

            onChange={handleNameChange}

          />

        </div>

        <div>

          <button type="submit">add</button>

        </div>

      </form>

      <h2>Numbers</h2>

      <>{row\_names()}</>

    </div>

  )

}

export default App;

**2.8: The Phonebook Step3**

Expand your application by allowing users to add phone numbers to the phone book. You will need to add a second *input* element to the form (along with its own event handler):

<form>

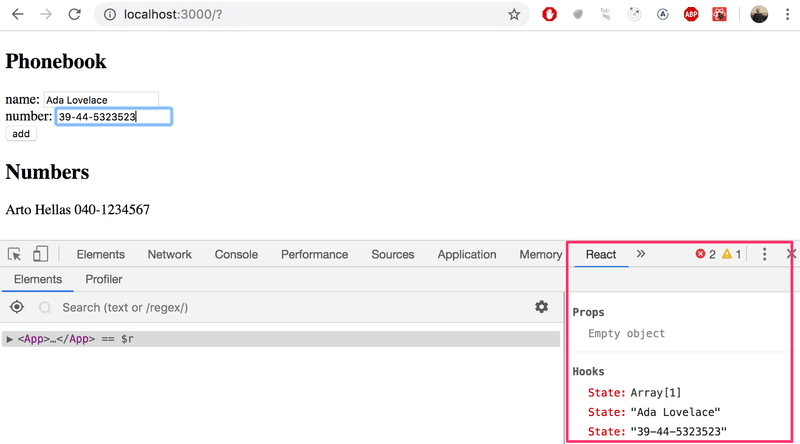
<div>name: <input /></div>

<div>number: <input /></div>

<div><button type="submit">add</button></div>

</form>

At this point the application could look something like this. The image also displays the application's state with the help of [React developer tools](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi):



// 2.8: The Phonebook Step3

// Expand your application by allowing users to add phone numbers to the phone book

import React, { useState } from 'react';

function App() {

  const [ persons, setPersons] = useState([

    { name: 'Arto Hellas', number: '040-1234567' }

  ])

  const [ newName, setNewName ] = useState('')

  const [ newNumber, setNewNumber ] = useState('')

  const addPerson = (event) => {

    event.preventDefault()

    const personObject = {

      name: newName,

      number: newNumber

    }

    console.log(personObject)

    const all\_names = persons.map(person => person.name)

    if (all\_names.includes(newName)) {

      alert(`${newName} is already added to phonebook`)

      return

    }

    setPersons(persons.concat(personObject))

    setNewName('')

    setNewNumber('')

  }

  const handleNameChange = (event) => {

    console.log(event.target.value)

    setNewName(event.target.value)

  }

  const handleNumberChange = (event) => {

    setNewNumber(event.target.value)

  }

  const row\_names = () => persons.map(person =>

    //<p key={person.name}>{person.name}</p>

    <p key={person.name}>{person.name} {person.number}</p>

  )

  return (

    <div>

      <h2>Phonebook</h2>

      <form onSubmit={addPerson}>

        <div>

          name: <input

            value={newName}

            onChange={handleNameChange}

          />

        </div>

        <div>

          number: <input

            value={newNumber}

            onChange={handleNumberChange}

          />

        </div>

        <div>

          <button type="submit">add</button>

        </div>

      </form>

      <h2>Numbers</h2>

      <>{row\_names()}</>

    </div>

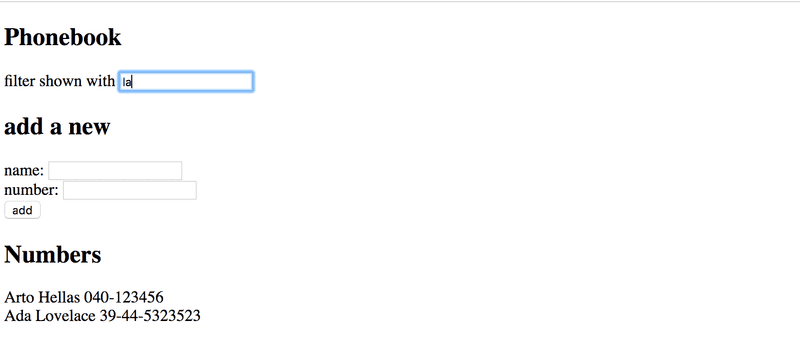
  )

}

export default App;

**2.9\*: The Phonebook Step4**

Implement a search field that can be used to filter the list of people by name:



You can implement the search field as an *input* element that is placed outside the HTML form. The filtering logic shown in the image is *case insensitive*, meaning that the search term *arto* also returns results that contain Arto with an uppercase A.

**NB:** When you are working on new functionality, it's often useful to "hardcode" some dummy data into your application, e.g.

const App = () => {

const [persons, setPersons] = useState([

{ name: 'Arto Hellas', number: '040-123456', id: 1 },

{ name: 'Ada Lovelace', number: '39-44-5323523', id: 2 },

{ name: 'Dan Abramov', number: '12-43-234345', id: 3 },

{ name: 'Mary Poppendieck', number: '39-23-6423122', id: 4 }

])

// ...

}

This saves you from having to manually input data into your application for testing out your new functionality.

// 2.9\*: The Phonebook Step4

// filter

import React, { useState } from 'react';

function App() {

  const [ persons, setPersons] = useState([

    { name: 'Arto Hellas', number: '040-123456', id: 1 },

    { name: 'Ada Lovelace', number: '39-44-5323523', id: 2 },

    { name: 'Dan Abramov', number: '12-43-234345', id: 3 },

    { name: 'Mary Poppendieck', number: '39-23-6423122', id: 4 }

  ])

  const [ newName, setNewName ] = useState('')

  const [ newNumber, setNewNumber ] = useState('')

  const [ filter, setFilter ] = useState('')

  const addPerson = (event) => {

    event.preventDefault()

    const personObject = {

      name: newName,

      number: newNumber

    }

    console.log(personObject)

    const all\_names = persons.map(person => person.name)

    if (all\_names.includes(newName)) {

      alert(`${newName} is already added to phonebook`)

      return

    }

    setPersons(persons.concat(personObject))

    setNewName('')

    setNewNumber('')

  }

  const handleNameChange = (event) => {

    console.log(event.target.value)

    setNewName(event.target.value)

  }

  const handleNumberChange = (event) => {

    setNewNumber(event.target.value)

  }

  const handleFilterChange = (event) => {

    setFilter(event.target.value)

  }

  const personsToShow = filter === ''

    ? persons

    : persons.filter(person =>

    person.name.toLowerCase().includes(filter.toLowerCase()))

  const row\_names = () => personsToShow.map(person =>

    <p key={person.name}>{person.name} {person.number}</p>

  )

  return (

    <div>

      <h2>Phonebook</h2>

      <div>

        filter shown with <input

          value={filter}

          onChange={handleFilterChange}

        />

      </div>

      <h2>add a new</h2>

      <form onSubmit={addPerson}>

        <div>

          name: <input

            value={newName}

            onChange={handleNameChange}

          />

        </div>

        <div>

          number: <input

            value={newNumber}

            onChange={handleNumberChange}

          />

        </div>

        <div>

          <button type="submit">add</button>

        </div>

      </form>

      <h2>Numbers</h2>

      <>{row\_names()}</>

    </div>

  )

}

export default App;

**2.10: The Phonebook Step5**

If you have implemented your application in a single component, refactor it by extracting suitable parts into new components. Maintain the application's state and all event handlers in the *App* root component.

It is sufficient to extract ***three*** components from the application. Good candidates for separate components are, for example, the search filter, the form for adding new people into the phonebook, a component that renders all people from the phonebook, and a component that renders a single person's details.

The application's root component could look similar to this after the refactoring. The refactored root component below only renders titles and lets the extracted components take care of the rest.

const App = () => {

// ...

return (

<div>

<h2>Phonebook</h2>

<Filter ... />

<h3>Add a new</h3>

<PersonForm

...

/>

<h3>Numbers</h3>

<Persons ... />

</div>

)

}

**NB**: You might run into problems in this exercise if you define your components "in the wrong place". Now would be a good time to rehearse the chapter [do not define a component in another component](https://fullstackopen.com/en/part1/a_more_complex_state_debugging_react_apps#do-not-define-components-within-components) from last part.

// 2.10: The Phonebook Step5

import React, { useState } from 'react';

import Filter from './P2\_b\_forms/components/filter';

import PersonForm from './P2\_b\_forms/components/personform';

import Persons from './P2\_b\_forms/components/persons'

function App() {

  const [ persons, setPersons] = useState([

    { name: 'Arto Hellas', number: '040-123456', id: 1 },

    { name: 'Ada Lovelace', number: '39-44-5323523', id: 2 },

    { name: 'Dan Abramov', number: '12-43-234345', id: 3 },

    { name: 'Mary Poppendieck', number: '39-23-6423122', id: 4 }

  ])

  const [ newName, setNewName ] = useState('')

  const [ newNumber, setNewNumber ] = useState('')

  const [ filter, setFilter ] = useState('')

  const addPerson = (event) => {

    event.preventDefault()

    const personObject = {

      name: newName,

      number: newNumber

    }

    console.log(personObject)

    const all\_names = persons.map(person => person.name)

    if (all\_names.includes(newName)) {

      alert(`${newName} is already added to phonebook`)

      return

    }

    setPersons(persons.concat(personObject))

    setNewName('')

    setNewNumber('')

  }

  const handleNameChange = (event) => {

    console.log(event.target.value)

    setNewName(event.target.value)

  }

  const handleNumberChange = (event) => {

    setNewNumber(event.target.value)

  }

  const handleFilterChange = (event) => {

    setFilter(event.target.value)

  }

  const personsToShow = filter === ''

    ? persons

    : persons.filter(person =>

    person.name.toLowerCase().includes(filter.toLowerCase()))

  const row\_names = () => personsToShow.map(person =>

    <p key={person.name}>{person.name} {person.number}</p>

  )

  return (

    <div>

      <h2>Phonebook</h2>

      <Filter value={filter} onChange={handleFilterChange} />

      <h2>add a new</h2>

      <PersonForm

        onSubmit={addPerson}

        name={{value: newName, onChange: handleNameChange}}

        number={{value: newNumber, onChange: handleNumberChange}}

      />

      <h2>Numbers</h2>

      <Persons persons={row\_names()} />

    </div>

  )

}

export default App;

import React from 'react';

const PersonForm = ({ onSubmit, name, number }) => {

  return (

    <form onSubmit={onSubmit}>

      <div>

        name: <input

          value={name.value}

          onChange={name.onChange}

        />

      </div>

      <div>

        number: <input

          value={number.value}

          onChange={number.onChange}

        />

      </div>

      <div>

        <button type="submit">add</button>

      </div>

    </form>

  )

}

export default PersonForm;

person.js

import React from 'react';

const Persons = ({ persons }) => <>{persons}</>

export default Persons;

filter.js

import React from 'react'

const Filter = ({ filter, onChange }) => {

  return (

    <div>

        filter shown with <input

          value={filter}

          onChange={onChange}

        />

      </div>

  )

}

export default Filter;