Table des annexes

Annexe I : Utilisation de la première version de withState

```
const Connection = connect(
  state => ({
    user: getUser(state)
  }),
  authActions
)(withState(
  {
    credentials: {
      onLoginChange: (_1, _2, { target }) => ({loginField: target.value}),
      onPasswordChange: (_1, _2, { target }) => ({password: target.value}),
      onSubmit: ({loginField, password}, {login}, e) => {
        e.preventDefault()
        login(loginField, password)
      }}
  }, {
    credential: {loginField: ({ initLogin }) => initlogin, password: ''}
  }

)(class extends PureComponent {
  render () {
    if (this.props.user) {
      return <a>You are already logged in</a>
    }
    return (
      <div>
        <Helmet title='Connection' />
        <form onSubmit={this.props.onSubmit}>
          <label>
            Login
            <Input onChange={this.props.onLoginChange} value={this.props.loginField} type='text' />
          </label>
          <br />
          <label>
            Password
            <Input onChange={this.props.onPasswordChange} value={this.props.password} type='password' />
          </label>
          <Button primary>Connection</Button>
        </form>
      </div>
    )
  }
}))
```

Annexe II : Première version du module withState

```
const withState = (actions, initValue) => {
  return Component => class extends PureComponent {
    constructor (props) {
      super(props)

      const initState = {}
      forIn(initValue, (reducer) => {
        forIn(reducer, (init, name) => {
          initState[name] = typeof init === 'function' ? init(props) : init
        })
      })
      this.state = initState

      this._actions = {}
      forIn(actions, (groupActions, key) => {
        forIn(groupActions, (action, actionName) => {
          this._actions[actionName] = (...args) => {
            const res = action(this.state, this.props, ...args)
            this.setState(res !== undefined ? res : this.state)
          }
        })
      })
    }

    render () {
      return <Component
        {...this.props}
        {...this._actions}
        {...this.state}
      />
    }
  }
}
```

Annexe III : Composant TodoList utilisant la seconde version de withState

```
const TodoList = withState({
  todos: {
    list: {
      init: Map([
        [ cuid(), {title: 'test', completed: true} ],
        [ cuid(), {title: 'test1', completed: false} ],
        [ cuid(), {title: 'test2', completed: false} ]
      ]),
      create: list => title => list.set(cuid(), {title, completed: false}),
      remove: list => id => list.delete(id),
      toggle: list => id => list.update(id, value => ({ ...value, completed: !value.completed })),
      toggleAll: list => event => list.map(value => ({...value, completed: event.target.checked})),
      removeCompleted: list => () => list.filter(value => !value.completed)
    },
    filteringPredicate: {
      init: () => list => list,
      setFilterActive: () => () => value => !value.completed,
      setFilterComplete: () => () => value => value.completed,
      setFilterAll: () => () => null // do not filter
    }
  },
  creationForm: {
    title: 'updateTitle'
    },

    onSubmit: ({title}, _, {create}, init) => event => {
      event.preventDefault()
      create(title)
      return init()
    }
  }
}, {
  getFilteredList: createSelector(
    state => state.todos.filteringPredicate,
    state => state.todos.list,
    (filteringPredicate, list) => filteringPredicate === null
      ? list
      : list.filter(filteringPredicate)
  ),
  getNumberOfIncompleteItems: createSelector(
    state => state.todos.list,
    list => list.filter(value => !value.completed).size
  )
})({
  creationForm,
  getFilteredList,
  getNumberOfIncompleteItems,
  listSelector,
  onSubmit,
  remove,
  removeCompleted,
  setFilterActive,
```

```
  setFilterAll,
  setFilterComplete,
  todos,
  toggle,
  toggleAll,
  updateTitle
}) => {
  const filteredList = getFilteredList()
  const leftItems = getNumberOfIncompleteItems()

  return (
    <div>
      <form onSubmit={onSubmit}>
        <h1>todos</h1>
        <Input
          checked={leftItems === 0 && filteredList.size > 0}
          onChange={toggleAll}
          type='checkbox'
        />
        <Input
          onChange={updateTitle}
          placeholder='What needs to be done'
          value={creationForm.title}
        />
      </form>
      <ul>
        {filteredList.map((todo, key) =>
          <li key={key}>
            <Input type='checkbox'
              checked={todos.list.get(key).completed}
              onChange={() => toggle(key)}
            />
            {todo.title}
            <Button onClick={() => remove(key)} type='button'>Remove</Button>
          </li>

        )}
      </ul>
      {todos.list.filter(todo => todo.completed).size > 0 &&
        <Button onClick={removeCompleted} type='button'>Remove all toggled</Button>
      }

      <div>
        {`${leftItems} item${leftItems < 2 ? '' : 's'} left`}
        <Button onClick={setFilterComplete} type='button'>Completed</Button>
        <Button onClick={setFilterActive} type='button'>Active</Button>
        <Button onClick={setFilterAll} type='button'>All</Button>
      </div>
    </div>
  )
})
```

Annexe IV : Implémentation de la deuxième version du module withState

```
const withState = (states, selectors) => Component =>
  class extends PureComponent {
    constructor (props) {
      super(props)
      this.actions = {}
      this.selectors = {}

      const setAction = (index, substate, path, func) => {
        if (index in this.actions) {
          throw new Error(`there is already an action ${index}`)
        }
        const substateCopy = typeof substate === 'object' ? cloneDeep(substate) : substate
        this.actions[index] = (...args) => {
          // if the first arg is a React event, persist it
          const [ first ] = args
          if (first != null && typeof first.persist === 'function') {
            first.persist()
          }
          const res = func(path.length === 0 ? this.state : get(this.state, path), props, this.actions, () => substateCopy)(...args)
          this.setState((prevState) => {
            const stateCopy = cloneDeep(prevState)
            return res !== undefined
              ? path.length === 0 ? {...prevState, ...res} : set(stateCopy, path, res)
              : prevState
          })
        }
      }
```

```
    const constructState = (obj, path) => {
      let substate = {}
      let pathCopy
      if (obj.init !== undefined) {
        substate = typeof obj.init === 'function' ? obj.init(props) : obj.init
      }
      forEach(pickBy(obj, val => typeof val !== 'function'), (value, key) => {
        pathCopy = path.slice()
        pathCopy.push(key)
        if (typeof value === 'object') {
          if (typeof substate !== 'undefined' && typeof substate !== 'object') {
            throw new Error('cannot provide an init value and substates')
          }
          substate[key] = constructState(value, pathCopy)
        } else if (typeof value === 'string' && key !== 'init') {
          const func = stateField => event => event.target.value
          substate[key] = ''
          setAction(value, substate, pathCopy, func)
        }
      })
      forEach(pickBy(obj, val => typeof val === 'function' && !path.includes('init')), (value, key) => {
        setAction(key, substate, path, value)
      })
      return substate
    }

    const setSelectors = selectors => mapValues(selectors, value => () => value(this.state, this.props))

    this.state = constructState(states, [])
    this.selectors = setSelectors(selectors)
  }

  render () {
    return <Component
      {...this.props}
      {...this.actions}
      {...this.state}
      {...this.selectors}
    />
  }
}
```

Annexe V : Fichier de tests permettant de valider le module withState

Annexe V.I : Décoration d'un composant stateful

```
it('supports wrapping a stateful component', () => {
  const increment = jest.fn((count, {step}) => () => {
    return count + step
  })

  const init = 3
  const step = 2

  let props
  const Counter = withState(
    {
      counter: {
        count: {
          init,
          increment
        }
      }
    }
  )(class extends PureComponent {
    render () {
      props = this.props
      return null
    }
  })

  render(<Counter init={init} step={step} />, document.createElement('div'))

  expect(props.counter.count).toBe(init)
  expect(typeof props.increment).toBe('function')

  props.increment('foo', 'bar')

  expect(increment.mock.calls[0][0]).toEqual(init)
  expect(increment.mock.calls[0][1]).toEqual({ init, step })
  expect(increment.mock.calls[0][2]).toEqual({ increment: props.increment })

  expect(props.counter.count).toBe(init + step)
})
```

Annexe V.II : Décoration d'un composant stateless

```
it('supports wrapping a stateless component', () => {
  const initCount = 5
  const initText = 'coucou'

  const step = 2
  const num = 3

  const increment = jest.fn((count, { step }) => () => count + step)

  const repeat = jest.fn(text => num => text.repeat(num))

  let props
  const Foo = withState(
    {
      counter: {
        count: {
          init: initCount,
          increment
        },
        resetCounter: (_1, _2, _3, init) => () => init()
      },
      text: {
        text: {
          init: 'coucou',
          repeat
        }
      },
      resetState: (_1, _2, _3, init) => () => init()
    }
  )(props_ => {
    props = props_
    return null
  })
  render(
    <Foo initCount={initCount} step={step} />,
    document.createElement('div')
  )

  props.increment()
  expect(props.counter.count).toBe(initCount + step)

  props.repeat(num)
  expect(props.text.text).toBe(initText + initText + initText)

  // console.log(props.resetCounter());
  props.resetCounter()
  expect(props.counter.count).toBe(initCount)
  expect(props.text.text).toBe(initText + initText + initText)
```

```
  props.resetCounter()
  expect(props.counter.count).toBe(initCount)
  expect(props.text.text).toBe(initText + initText + initText)

  props.increment()

  props.resetState()

  expect(props.counter.count).toBe(initCount)
  expect(props.text.text).toBe(initText)
})
```

Annexe V.III : Utilisation d'une fonction pour initialiser la valeur d'un sous-state

```
it('supports a function to compute initial value from props', () => {
  const init = 0

  let props
  const Counter = withState(
    {
      counter: {
        count: {
          init: props => props.init
        }
      }
    }
  )(props_ => {
    props = props_
    return null
  })

  render(<Counter init={init} />, document.createElement('div'))
  expect(props.counter.count).toBe(init)
})
```

Annexe V.IV : Remise de la valeur d'un sous-state à sa valeur initiale

```
describe('withState()', () => {
  it('provides a function to retrieve the initial value.', () => {
    const initCount1 = 5
    const initCount2 = 4
    const initText = 'test'

    const step = 2
    const num = 3

    const increment2 = jest.fn((count, { step }) => () => count + step)
    const increment1 = jest.fn((count, { step }) => () => count + step + 1)

    const repeat = jest.fn(text => num => text.repeat(num))

    let props
    const Foo = withState(
      {

        counter: {
          counter1: {
            init: initCount1,
            increment1
          },
          resetCounter: (_1, _2, _3, init) => () => init(),
          counter2: {
            init: initCount2,
            increment2
          }
        },
        stringTest: 'onStringTestChange',
        text: {
          init: initText,
          repeat,
          resetText: (_1, _2, _3, init) => () => init()
        },
        resetState: (_1, _2, _3, init) => () => init()
      }
    )(props_ => {
      props = props_
      return null
    })
    render(
      <Foo initCount={initCount1} step={step} />,
      document.createElement('div')
    )
```

```
  const e = {target: {value: 'test'}}

  props.increment2()
  expect(props.counter.counter2).toBe(initCount2 + step)

  props.increment1()
  expect(props.counter.counter1).toBe(initCount1 + step + 1)

  props.repeat(num)
  expect(props.text).toBe(initText + initText + initText)

  props.resetText()
  expect(props.text).toBe(initText)

  props.onStringTestChange(e)
  expect(props.stringTest).toBe(e.target.value)

  props.resetCounter()
  expect(props.counter.counter2).toBe(initCount2)
  expect(props.counter.counter1).toBe(initCount1)
  expect(props.stringTest).toBe(e.target.value)
  expect(props.text).toBe(initText)

  props.resetState()
  expect(props.counter.counter2).toBe(initCount2)
  expect(props.counter.counter1).toBe(initCount1)
  expect(props.stringTest).toBe('')
  expect(props.text).toBe(initText)
})
```

Annexe V.V : Test d'une action qui ne retourne rien

```
it('supports an action without return', () => {
  const increment = jest.fn(count => step => {
    return count + step
  })

  const nothing = jest.fn(() => () => {})

  const init = 3
  const step = 2

  let props
  const Counter = withState(
    {
      counter: {
        count: {
          init,
          increment
        },
        count1: {

        },
        nothing
      },
      plop: {
        text: 'coucou'
      }
    }
  )(props_ => {
    props = props_
    return null
  })

  render(<Counter />, document.createElement('div'))

  props.increment(step)
  expect(props.counter.count).toBe(init + step)

  props.nothing()

  props.increment(step)
  expect(props.counter.count).toBe(init + 2 * step)
})
```

Annexe V.VI : Initialisation d'une chaine de caractères vide et création d'une fonction de mise à jour de celle-ci

```
it('supports initialisation of value and onChange function with a single string', () => {
  let props
  const Text = withState(
    {
      textReducer1: {
        text1: 'onText1Change',
        text2: {
          init: '',
          ontext2Change: () => () => {}
        },
        text3: 'onText3Change',
        text4: 'onText4Change'
      },
      textReducer2: 'onTextReducer2Change'
    }
  )(props_ => {
    props = props_
    return null
  })

  render(<Text />, document.createElement('div'))

  expect(props.textReducer1 === {text1: '', text2: '', text3: '', text4: ''})

  const event = {target: { value: 'test' }}
  props.onText1Change(event)
  expect(props.textReducer1.text1).toBe('test')
})
```

Annexe V.VII : Déclenchement d'une erreur lorsque plus d'une valeur est déclarée dans un sous-state

```
it('throws if both init and substates are provided', () => {
  const Component = withState({
    foo: {
      init: 'bar',
      baz: {}
    }
  })(() => null)

  expect(() => {
    render(<Component />, document.createElement('div'))
  }).toThrow('cannot provide an init value and substates')
})
```

Annexe VI : Commandes de gestion d'un VM_Appliance

Commands to manage a VM_Appliance

Create a VM_Appliance :

```
xe appliance-create name-label=<VM_Appliance label> name-description=<VM_Appliance description>
```

display a VM_Appliance :

```
xe appliance-param-list uuid=<VM_Appliance uuid>
```

List all the VM_Appliances :

```
xe appliance-list
```

Set the label and/or the description :

```
xe appliance-param-set uuid=<VM_Appliance uuid> name-description=<VM_Appliance description > name-label=<VM_Appliance label >
```

Delete a VM_Appliance :

```
xe appliance-destroy uuid=<VM_Appliance uuid>
```

Get a parameter :

```
xe appliance-param-get uuid=<VM_Appliance uuid> param-name=<name label>
```

Assign a VM to a VM_Appliance

```
xe vm-param-set appliance=<VM_Appliance uuid> uuid=<VM uuid>
```

Shutdown the VMs assigned to a VM_Appliance

```
xe appliance-shutdown uuid=<VM_Appliance uuid>
```

Start the VMs assigned to a VM_Appliance

```
xe appliance-start uuid=<VM_Appliance uuid>
```

Set an order

```
xe vm-param-set order=<order value> uuid=<VM uuid>
```

With a low order parameter, the VM will start first and shutdown last

Set a delay on machine

- on starting :

```
xe vm-param-set start-delay=<time in seconde> uuid=<VM uuid>
```

-on shutdowning :

```
xe vm-param-set shutdown-delay=<time in seconde> uuid=<VM uuid>
```

Annexe VIII : Vue générale d'un hôte