Automated a11y testing

Tim Damen
22 January 2024
DevCon

Tim Damen

- Team Warp, Grid DCC
- Frontend Chapter Lead
- Accessibility Guild Lead

Accessibility (a11y)

Why

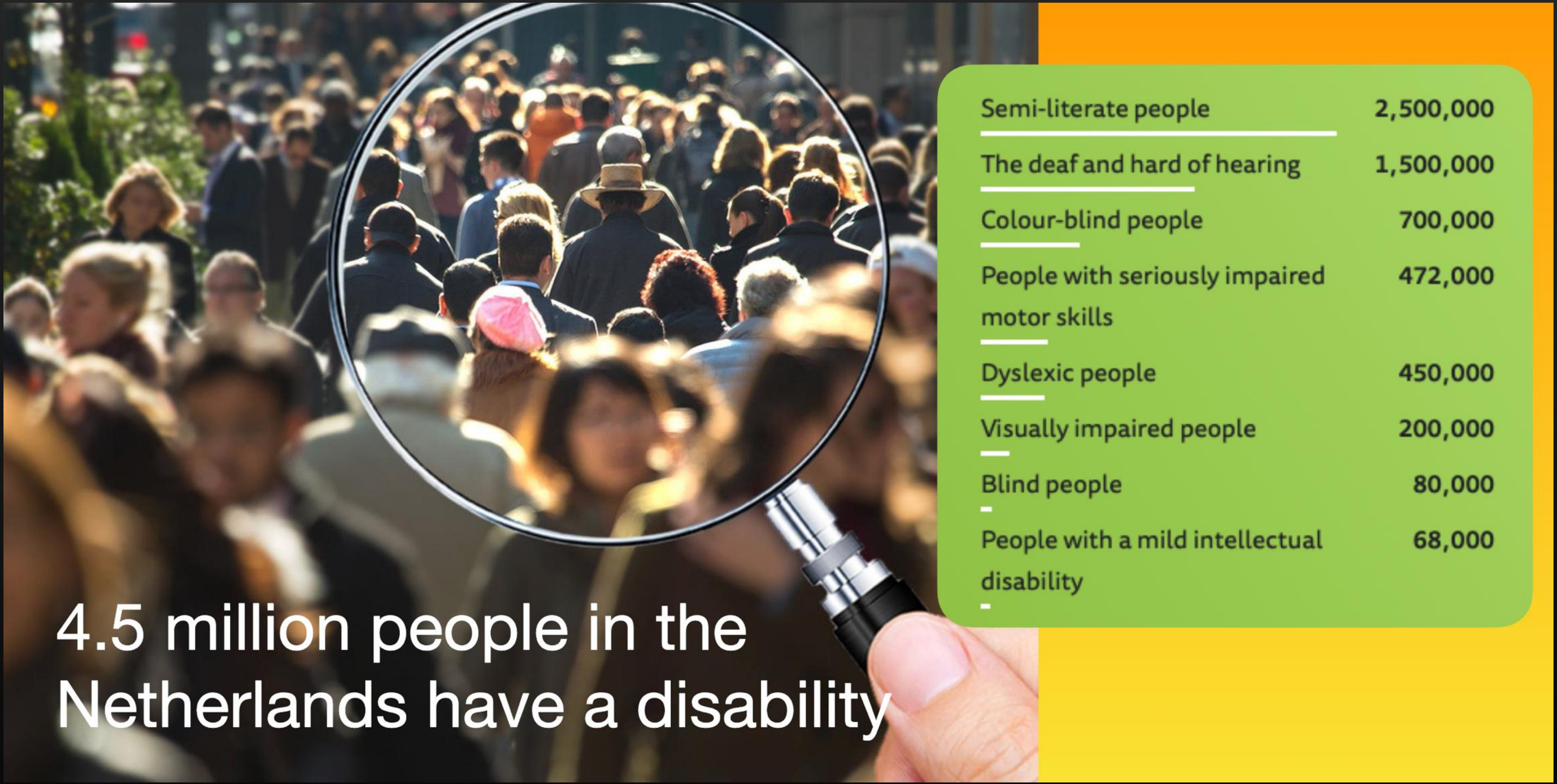
How

How +

Our applications are not accessible

The most recent two audits show that we comply with only 45% of the successciteria

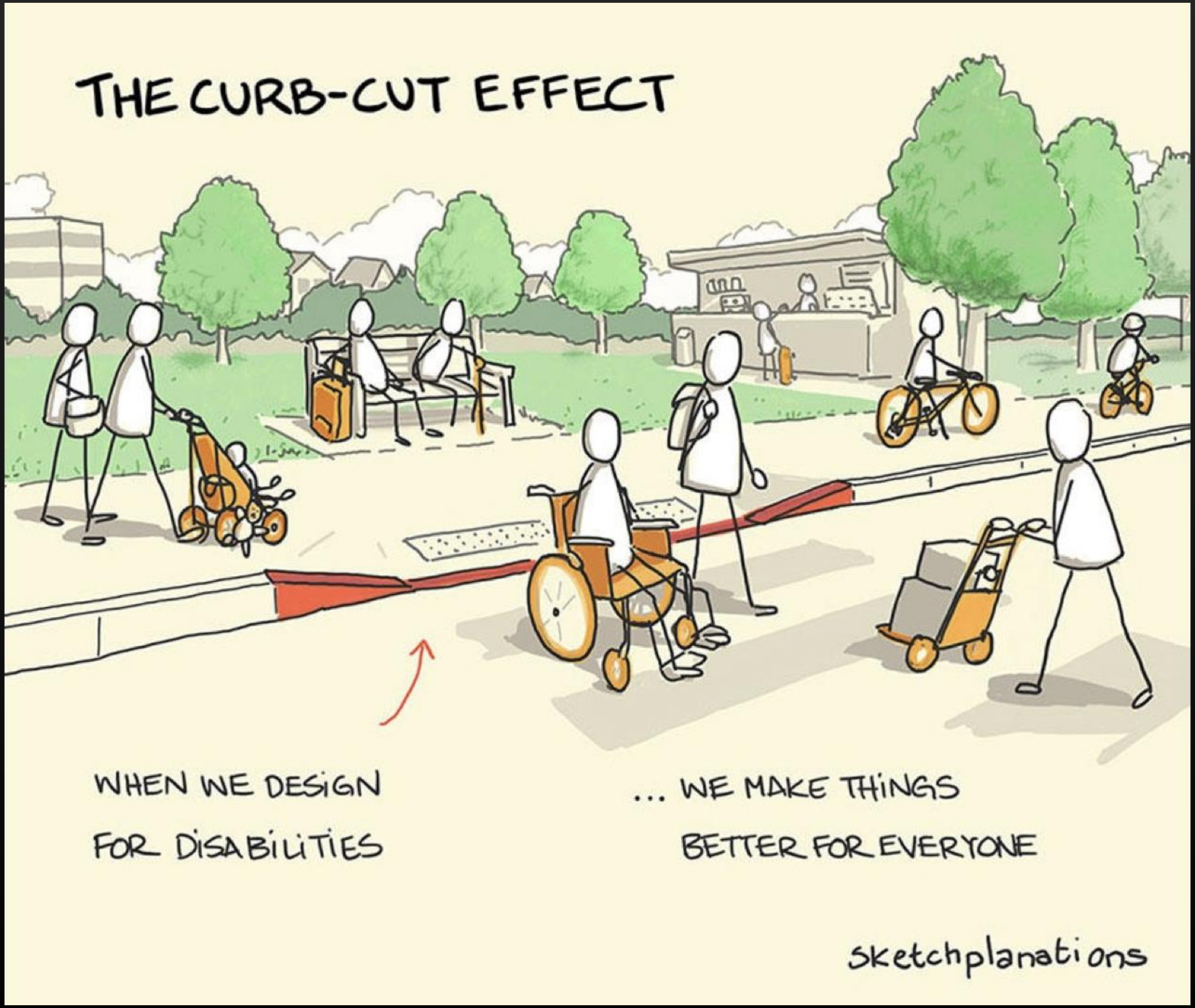
Why care about Accessibility (a11y)



4.5 million people in the Netherlands have a disability

Semi-literate people	2,500,000
The deaf and hard of hearing	1,500,000
Colour-blind people	700,000
People with seriously impaired motor skills	472,000
Dyslexic people	450,000
Visually impaired people	200,000
Blind people	80,000
People with a mild intellectual disability	68,000

Of the 7.000.000 customers of the ABN AMRO, 1.600.000 have a disability


THE CURB-CUT EFFECT
WHEN WE DESIGN FOR DISABILITIES
... WE MAKE THINGS BETTER FOR EVERYONE
sketchplanations

European Accessibility Act (2025)

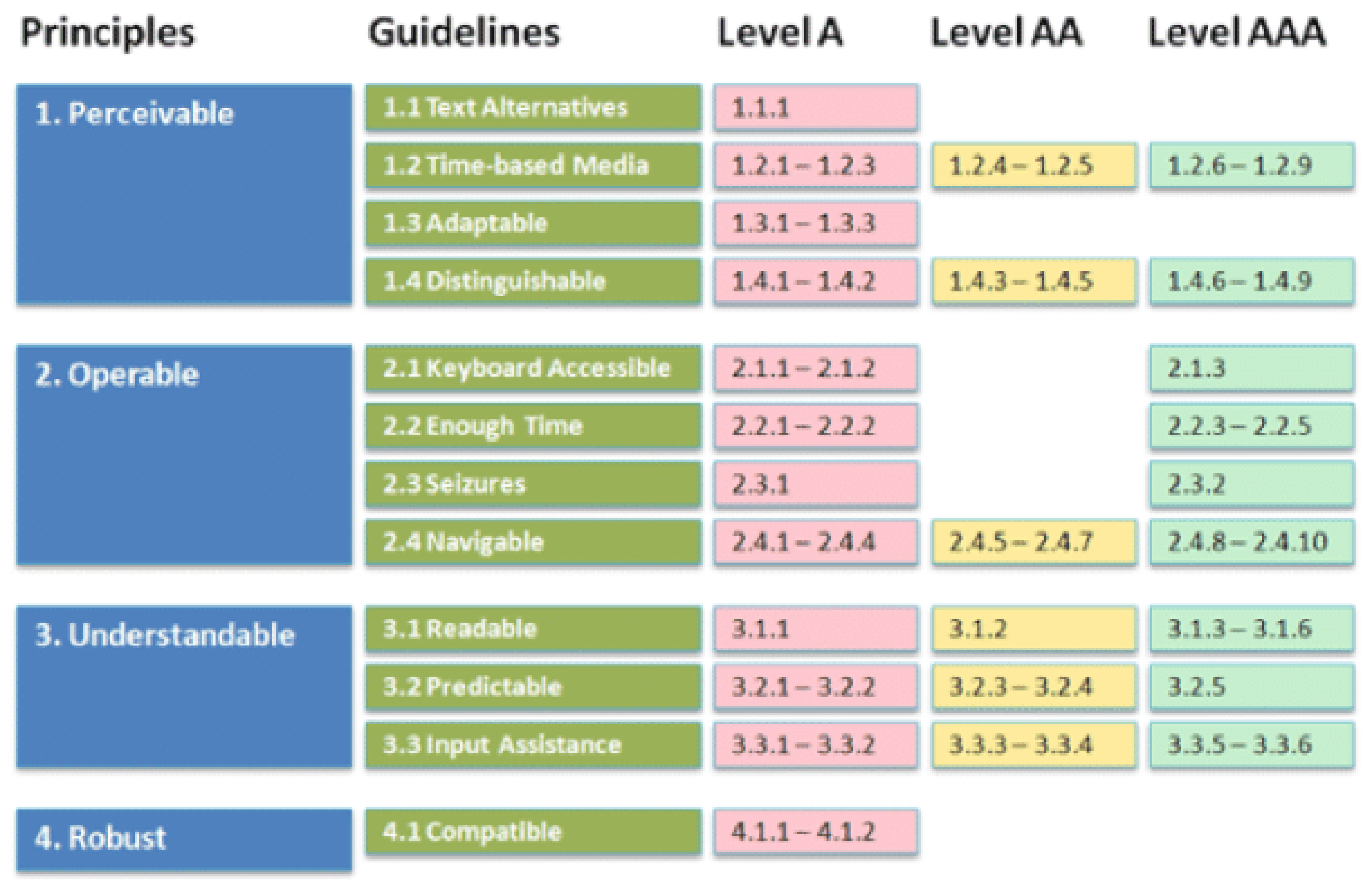
European
Commission

"(...) should ensure that people with (functional) disabilities have the same (online) opportunities throughout the European Union as people without (functional) disabilities.'

The starting point (here) is that the user experience for everyone should be as similar as possible."

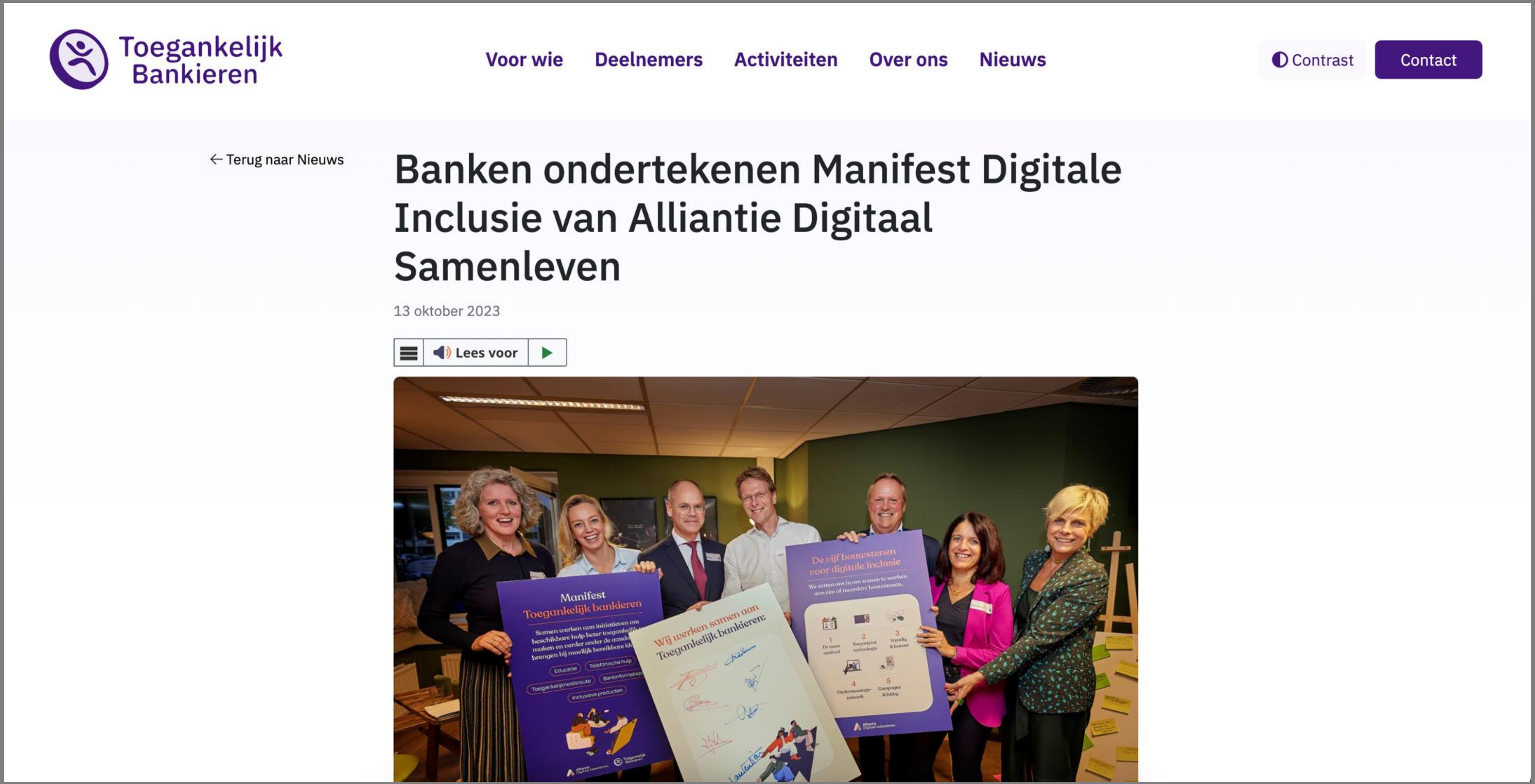
- **All our apps need to be WCAG (Web Content Accessibility Guidelines) 2.1 AA complaint by 2025**
- **AND We must be able to show proactive that we are accessible by submitting an accessibility statement and submitting audits and test results.**

WCAG 2.1(Web Content Accessibility Guidelines)



Principles	Guidelines	Level A	Level AA	Level AAA
1. Perceivable	1.1 Text Alternatives	1.1.1		
	1.2 Time-based Media	1.2.1 – 1.2.3	1.2.4 – 1.2.5	1.2.6 – 1.2.9
	1.3 Adaptable	1.3.1 – 1.3.3		
	1.4 Distinguishable	1.4.1 – 1.4.2	1.4.3 – 1.4.5	1.4.6 – 1.4.9
2. Operable	2.1 Keyboard Accessible	2.1.1 – 2.1.2		2.1.3
	2.2 Enough Time	2.2.1 – 2.2.2		2.2.3 – 2.2.5
	2.3 Seizures	2.3.1		2.3.2
	2.4 Navigable	2.4.1 – 2.4.4	2.4.5 – 2.4.7	2.4.8 – 2.4.10
3. Understandable	3.1 Readable	3.1.1	3.1.2	3.1.3 – 3.1.6
	3.2 Predictable	3.2.1 – 3.2.2	3.2.3 – 3.2.4	3.2.5
	3.3 Input Assistance	3.3.1 – 3.3.2	3.3.3 – 3.3.4	3.3.5 – 3.3.6
4. Robust	4.1 Compatible	4.1.1 – 4.1.2		

4 Principles, 12 Guidelines, 50 Succescirteria


Toegankelijk Bankieren
Voor wie
Deelnemers
Activiteiten
Over ons
Nieuws
Contrast
Contact
← Terug naar Nieuws
Lees voor
Manifest Toegankelijk bankieren
Wij werken samen aan Toegankelijk bankieren:
De vijf bouwstenen voor digitale inclusie

Banken ondertekenen Manifest Digitale Inclusie van Alliantie Digitaal Samenleven

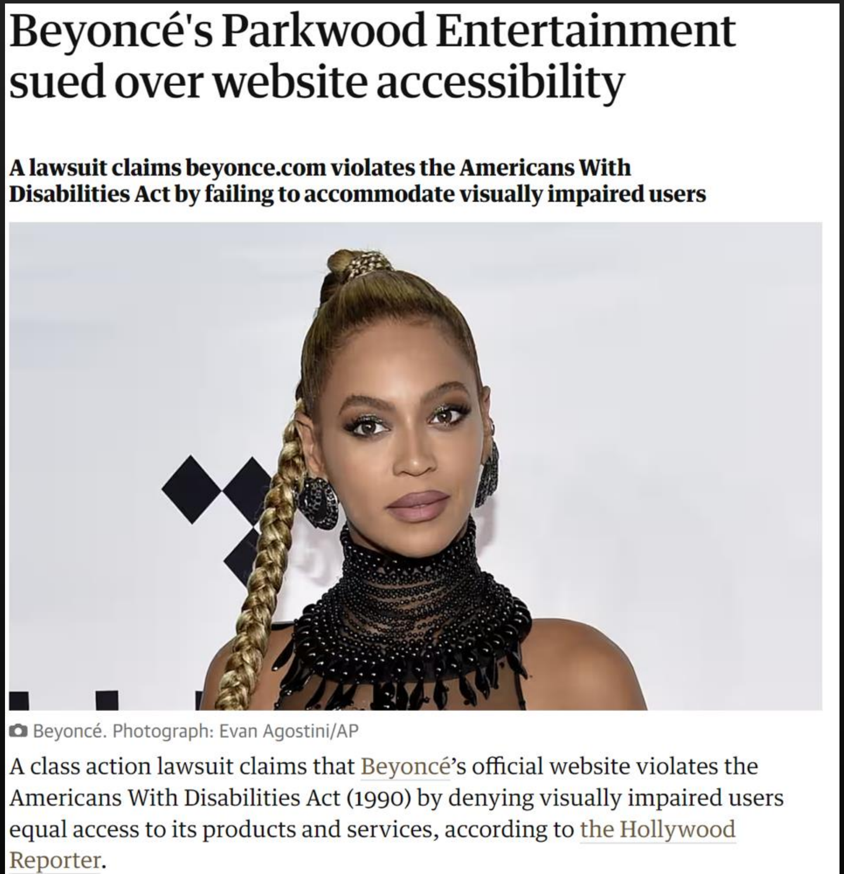
13 oktober 2023

Rabobank

ING

ABN·AMRO

Complaint leads to penalty?



Beyoncé's Parkwood Entertainment sued over website accessibility

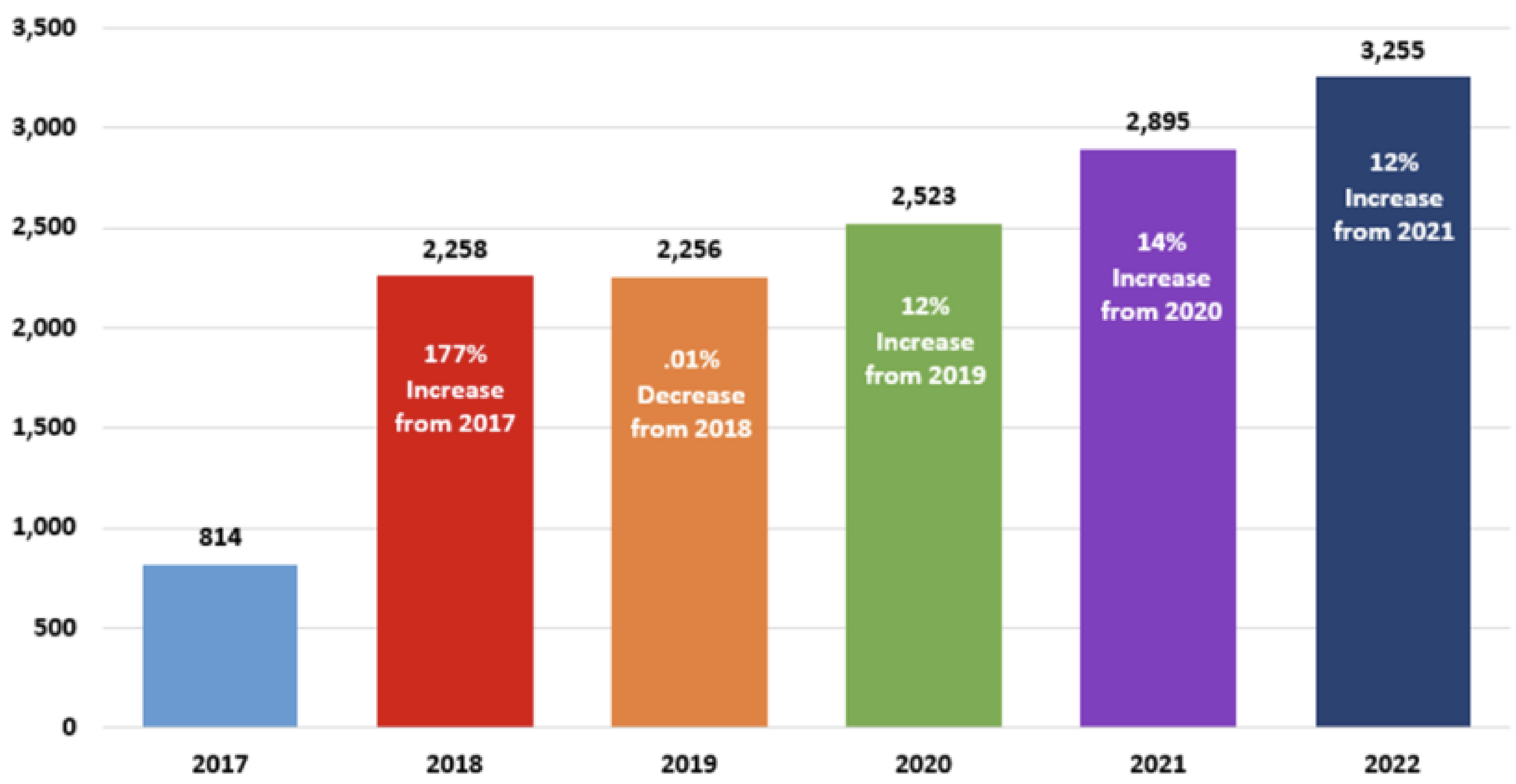
A lawsuit claims beyonce.com violates the Americans With Disabilities Act by failing to accommodate visually impaired users

Beyoncé. Photograph: Evan Agostini/AP

A class action lawsuit claims that Beyoncé's official website violates the Americans With Disabilities Act (1990) by denying visually impaired users equal access to its products and services, according to the Hollywood Reporter.

Amazon Class Action Lawsuit Says Website Not Accessible to the Blind

In 2022 more than 3.255 lawsuits were filed against allegedly inaccessible websites



ADA Titile III Website Accessiblity Lawsuits in Federal Court
2017-2022

A11y improvement cases

"Improvement in code quality was observed when we focused on testing and resolving accessibility issues, lower costs, more consistent code, improved customer experience, a decrease in complaints and calls to customer service, higher customer score and a lot of good press."

- Matthew Luke, Head of accessibility US BANK (USA)

POSITIVE IMAGE

HIGHER SALES, BY APPEALING TO LARGER AUDIENCES AND EASY-TO-USE APPS

REDUCED HELP DESK LOAD

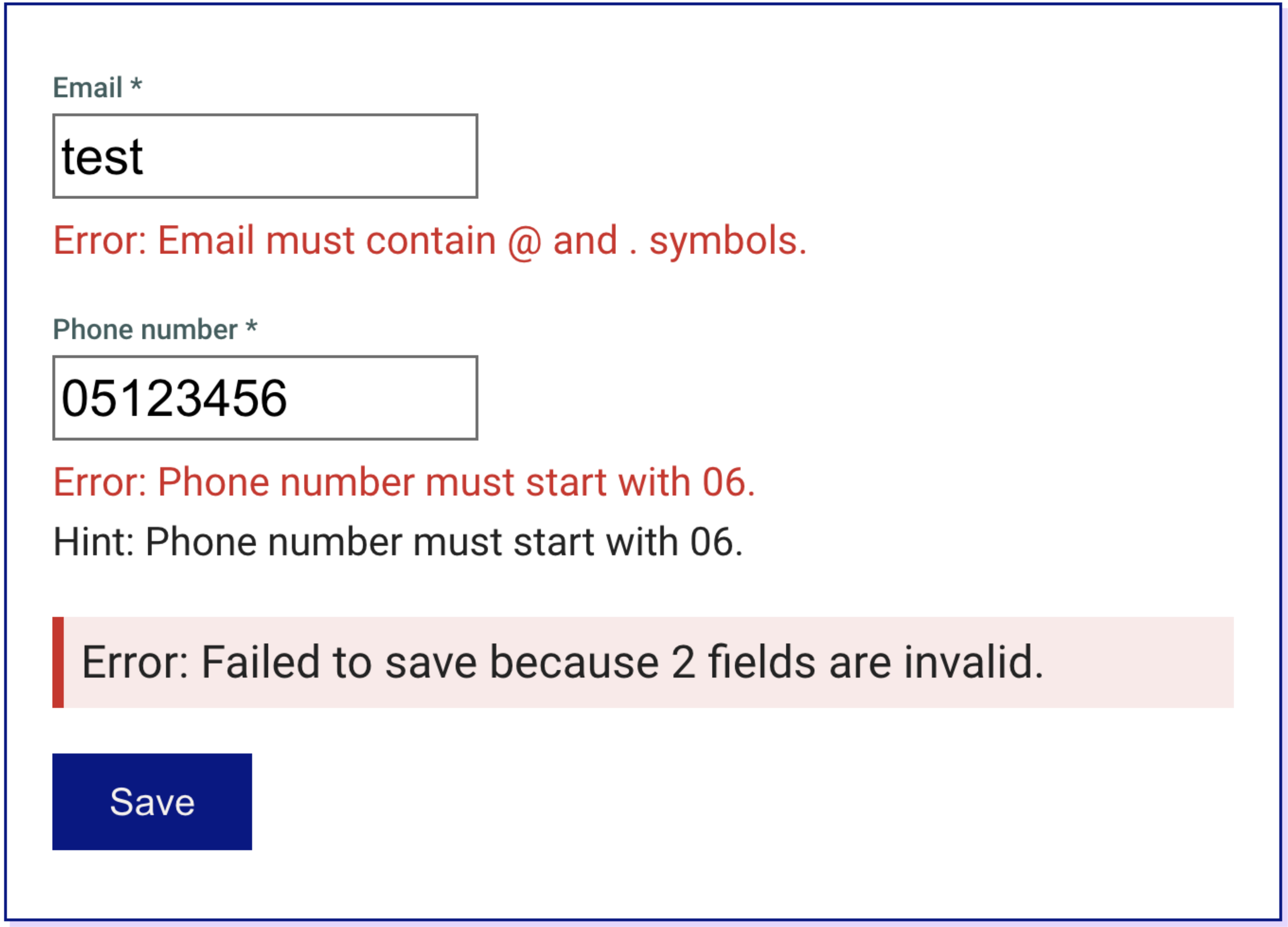
HIGHER LEVELS OF CUSTOMER SATISFACTION

BETTER CODE QUALITY AND CONSISTENCY YIELDS LOWER COSTS IN THE LONG RUN

How?

Contact page

* Fill the required fields



Email *

test

Error: Email must contain @ and . symbols.

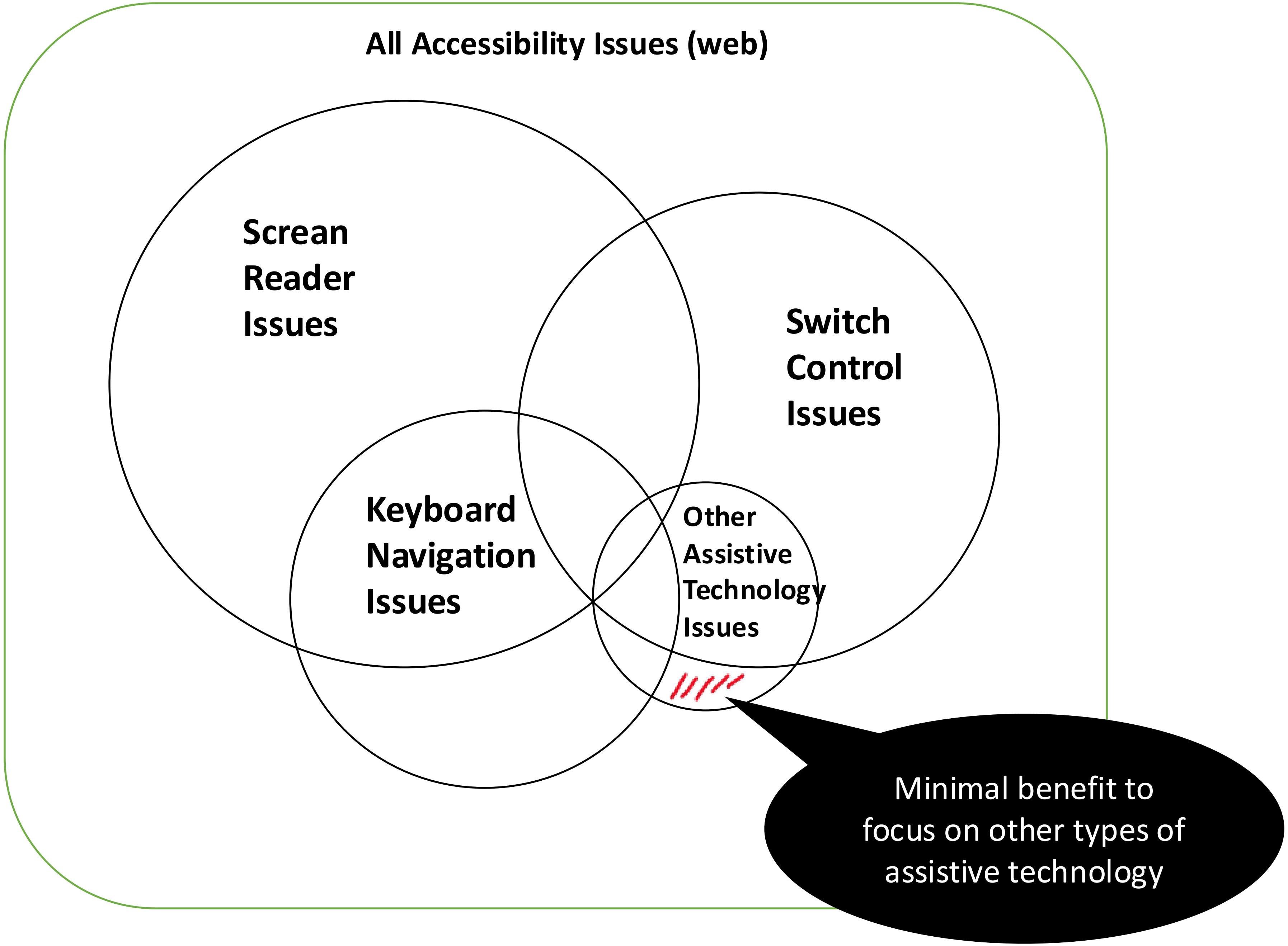
Phone number *

05123456

Error: Phone number must start with 06.

Hint: Phone number must start with 06.

Error: Failed to save because 2 fields are invalid.



All Accessibility Issues (web)

Screan Reader Issues

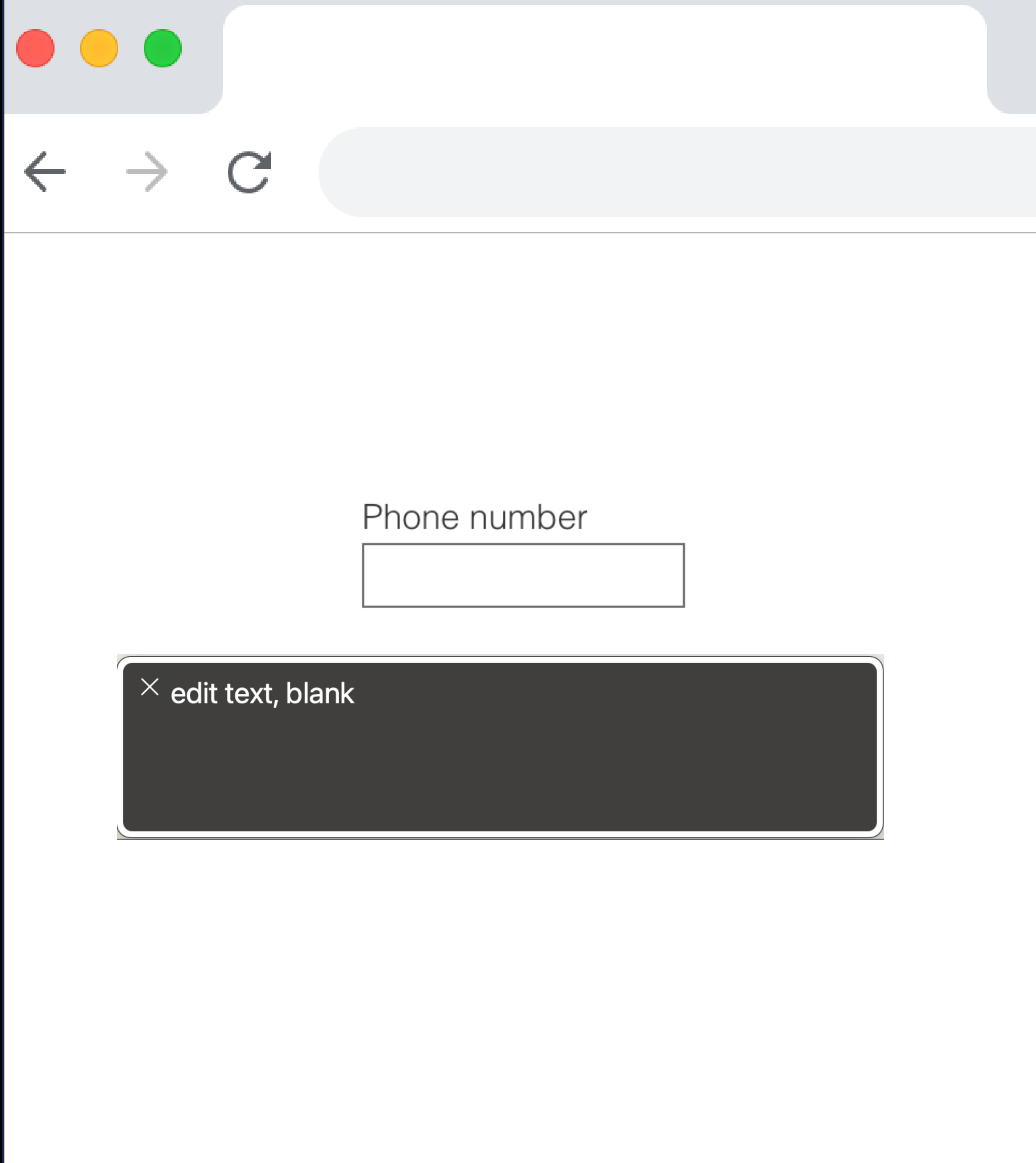
Switch Control Issues

Keyboard Navigation Issues

Other Assistive Technology Issues

Minimal benefit to focus on other types of assistive technology

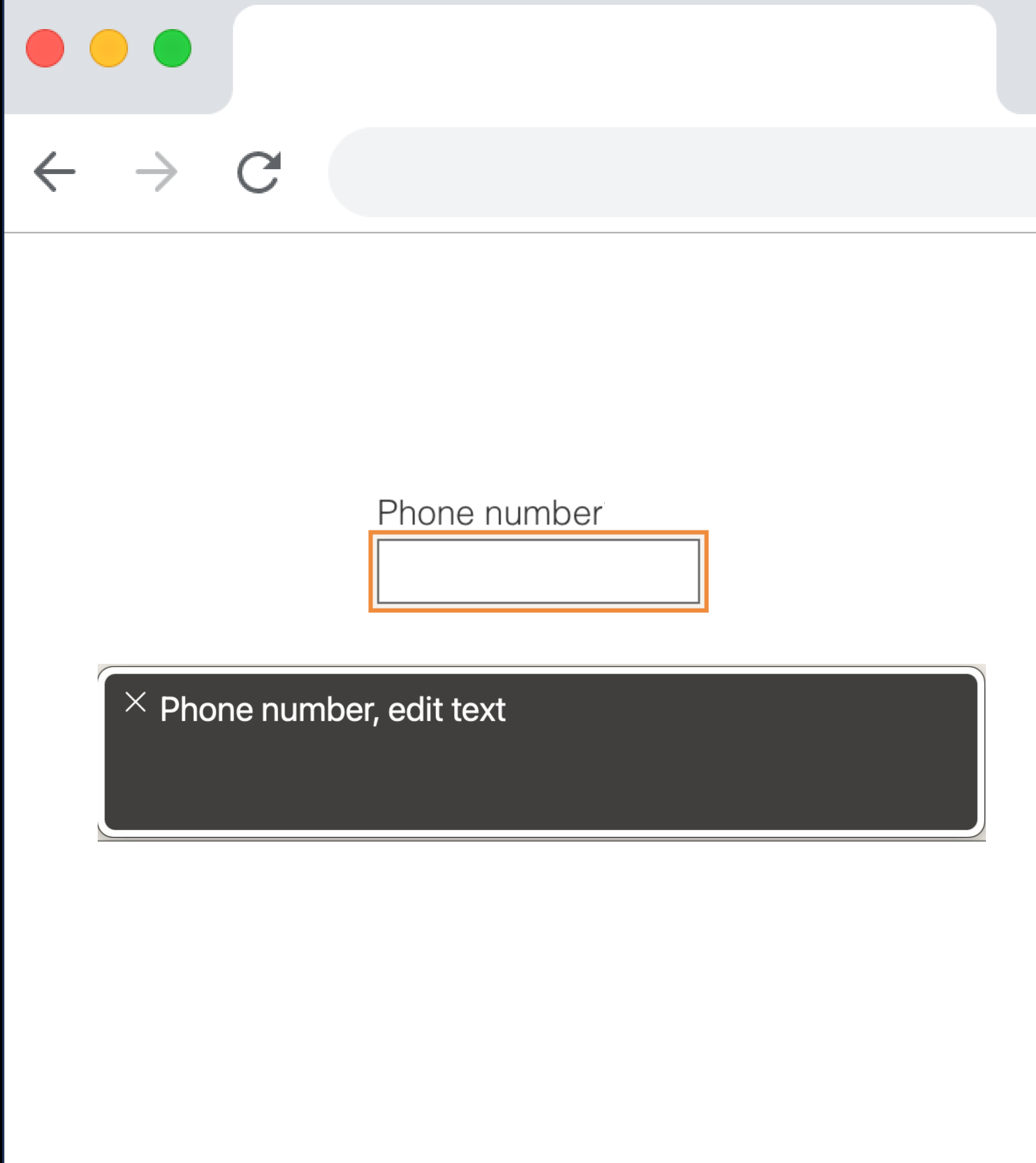
```html
<label>Phone number</label>
<input type="phone"/>
```



Phone number

edit text, blank

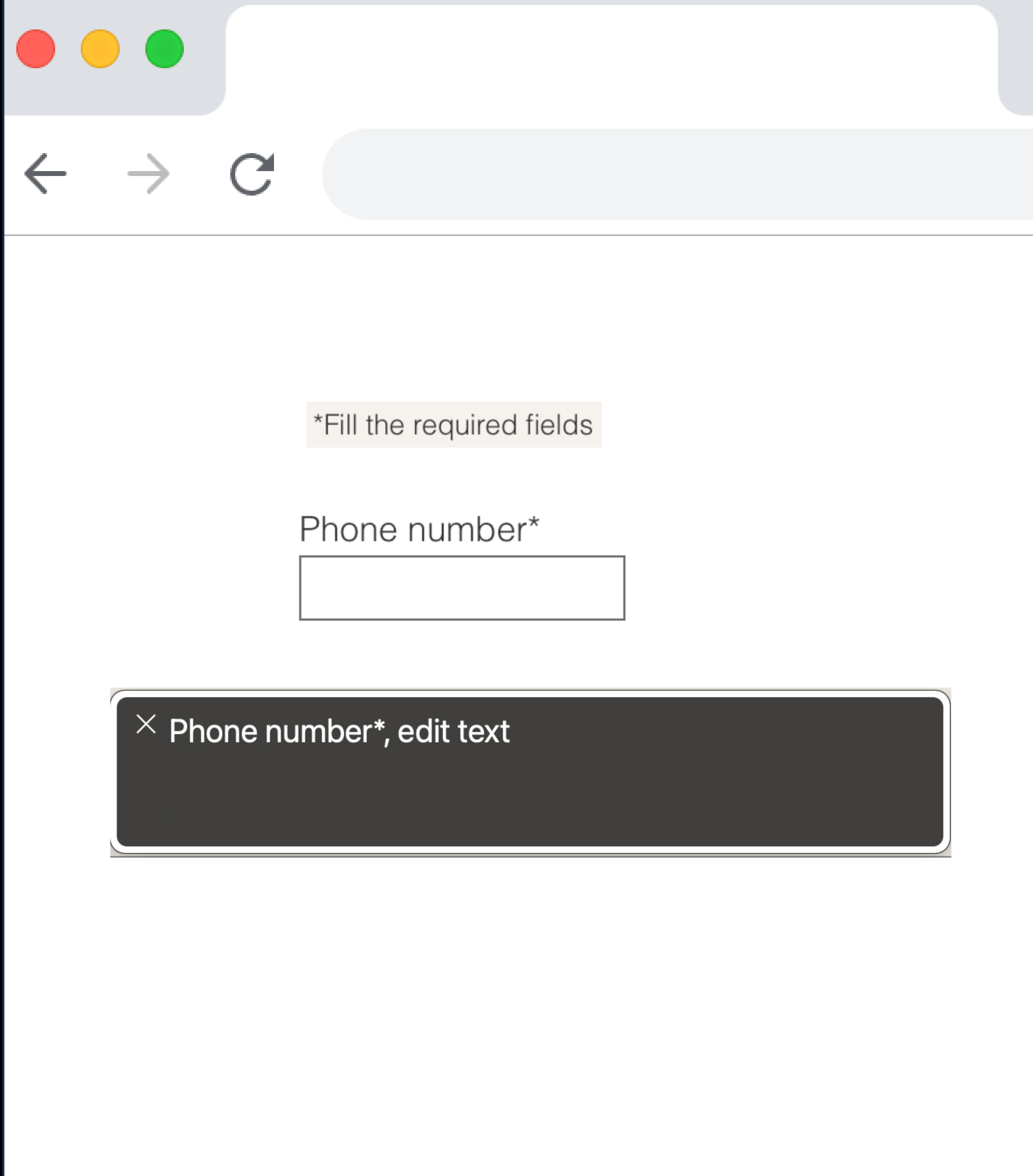
```html
<label for="phone">Phone number</label>
<input id="phone" type="phone"/>
```



Phone number

Phone number, edit text

```html
<label for="phone">Phone number*</label>
<input id="phone" type="phone"/>
```

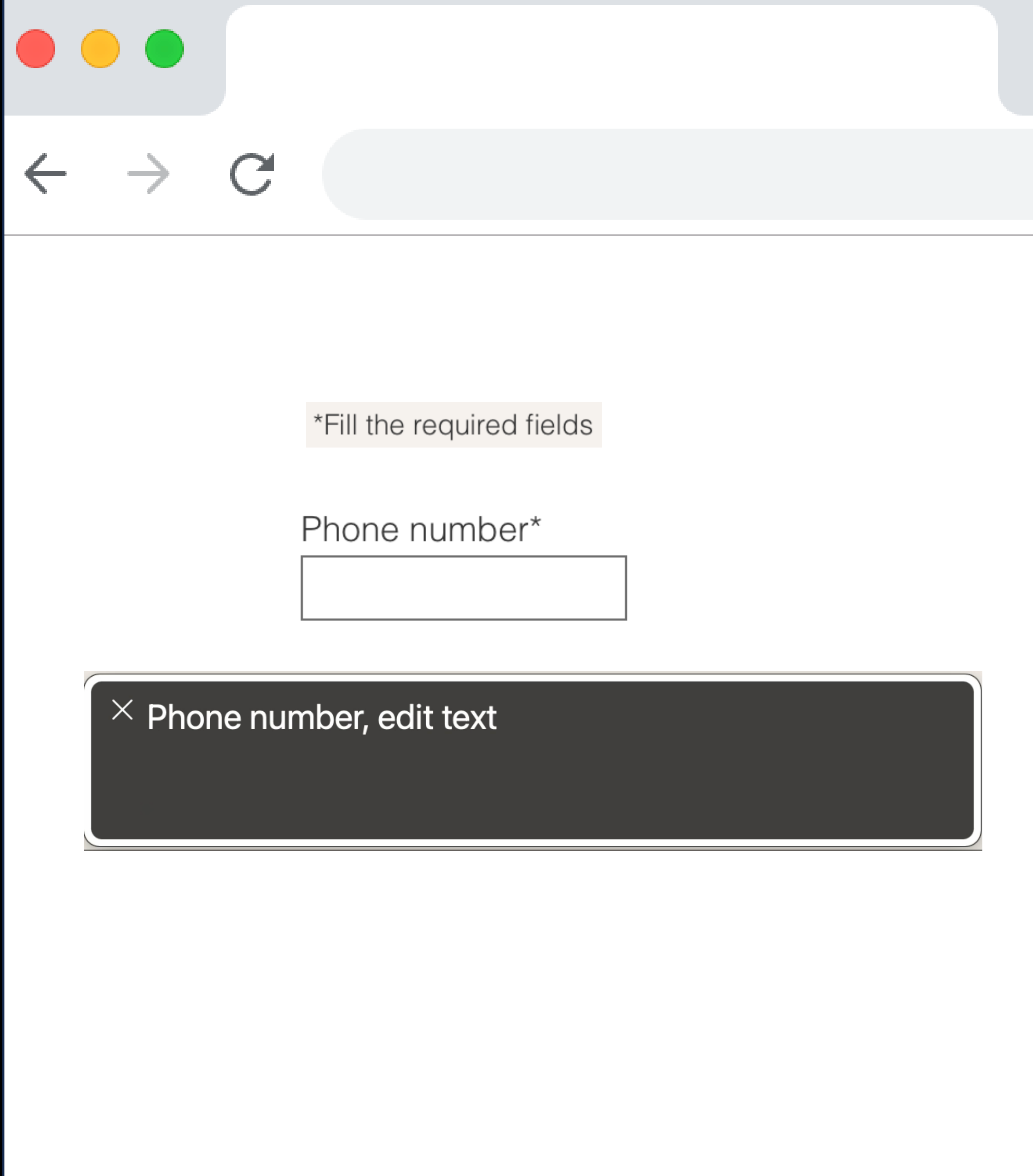


*Fill the required fields

Phone number*

Phone number*, edit text

```html
<label for="phone">
  Phone number<span aria-hidden="true">*</span>
</label>
<input type="phone" id="phone"/>
```

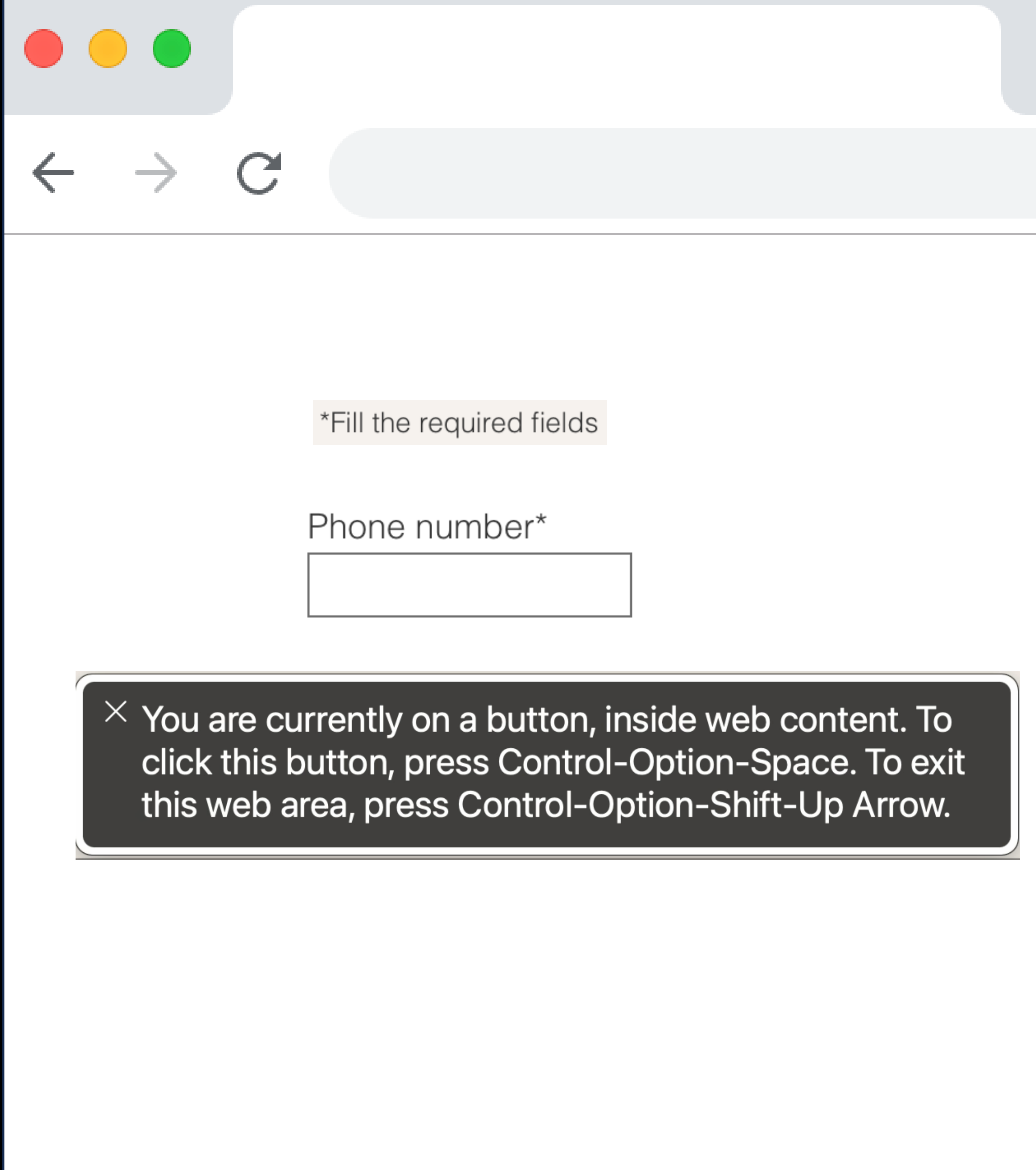


*Fill the required fields

Phone number*

Phone number, edit text

```
<label for="phone">
  Phone number<span aria-hidden="true">*</span>
</label>
<input type="phone" id="phone" aria-required="true"/>
```

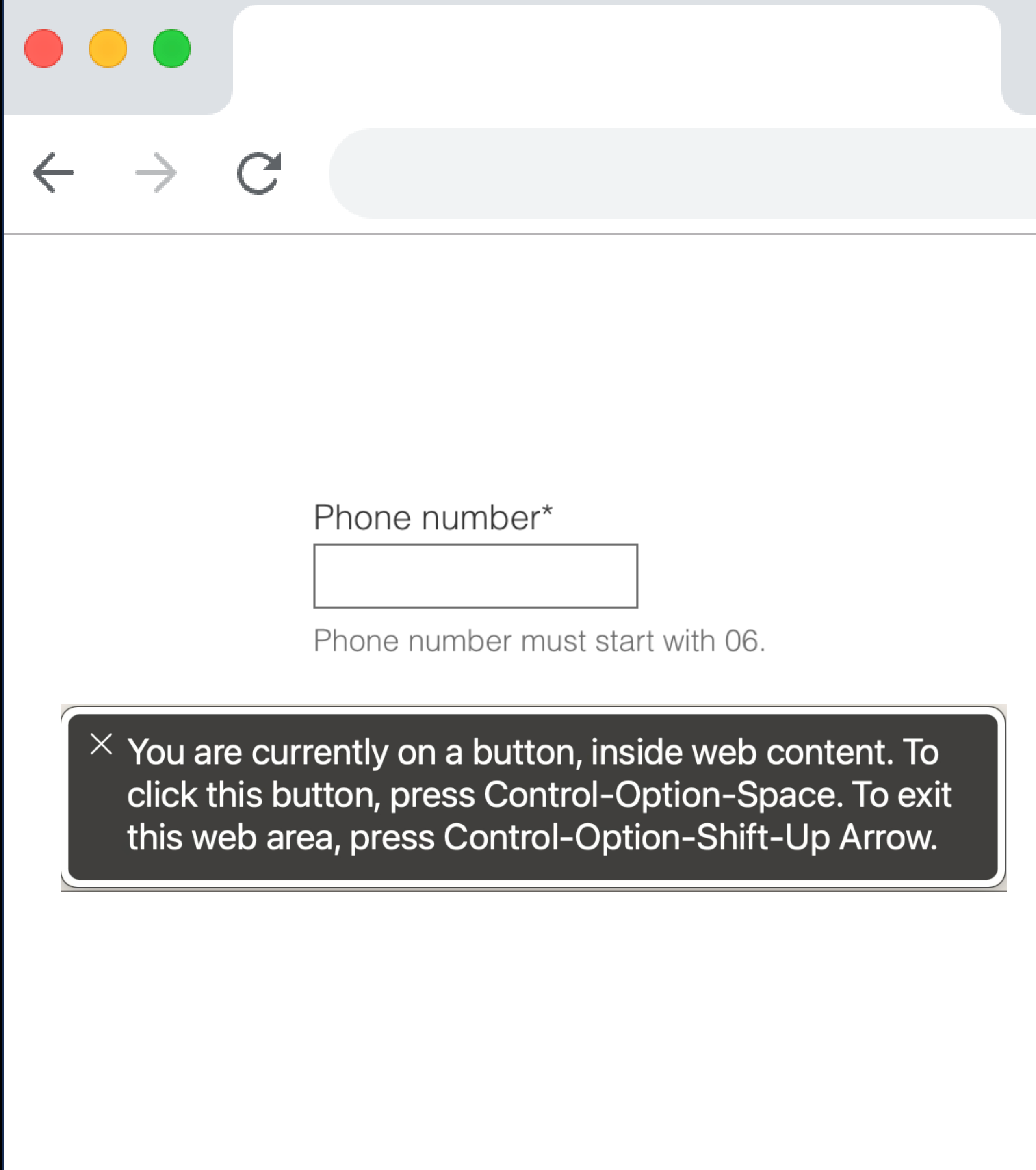


*Fill the required fields

Phone number*

You are currently on a button, inside web content. To click this button, press Control-Option-Space. To exit this web area, press Control-Option-Shift-Up Arrow.

```html
<label for="phone">
 Phone number<span aria-hidden="true">*</span>
</label>
<input
 type="phone"
 id="phone"
 aria-required="true"
/>

<p>Phone number must start with 06.</p>
```

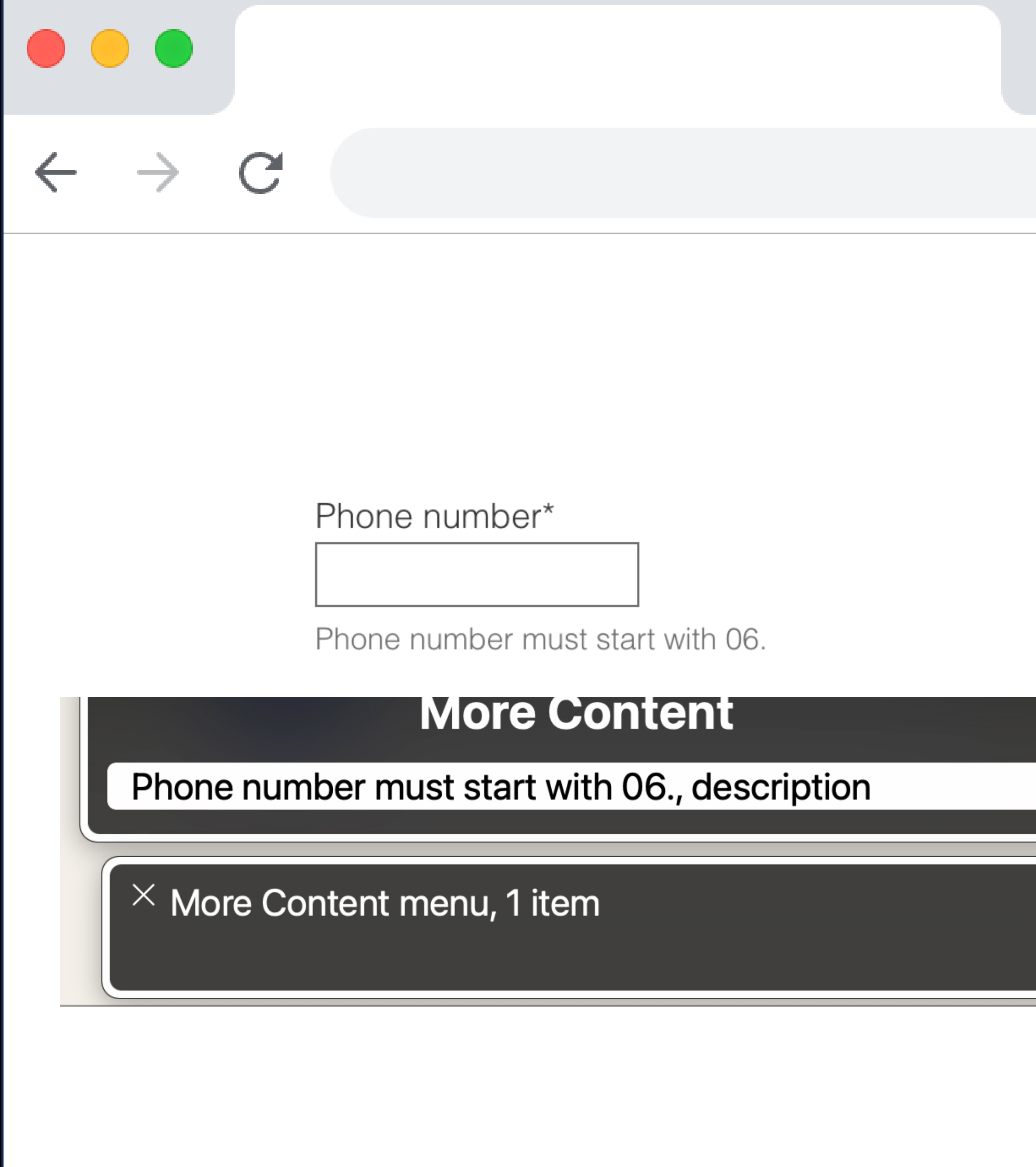


Phone number*

Phone number must start with 06.

You are currently on a button, inside web content. To click this button, press Control-Option-Space. To exit this web area, press Control-Option-Shift-Up Arrow.

```html
<label for="phone">
 Phone number<span aria-hidden="true">*</span>
</label>
<input
 type="phone"
 id="phone"
 aria-required="true"
 aria-describedby="phoneHint"
/>

<p id="phoneHint">Phone number must start with 06.</p>
```



Phone number*

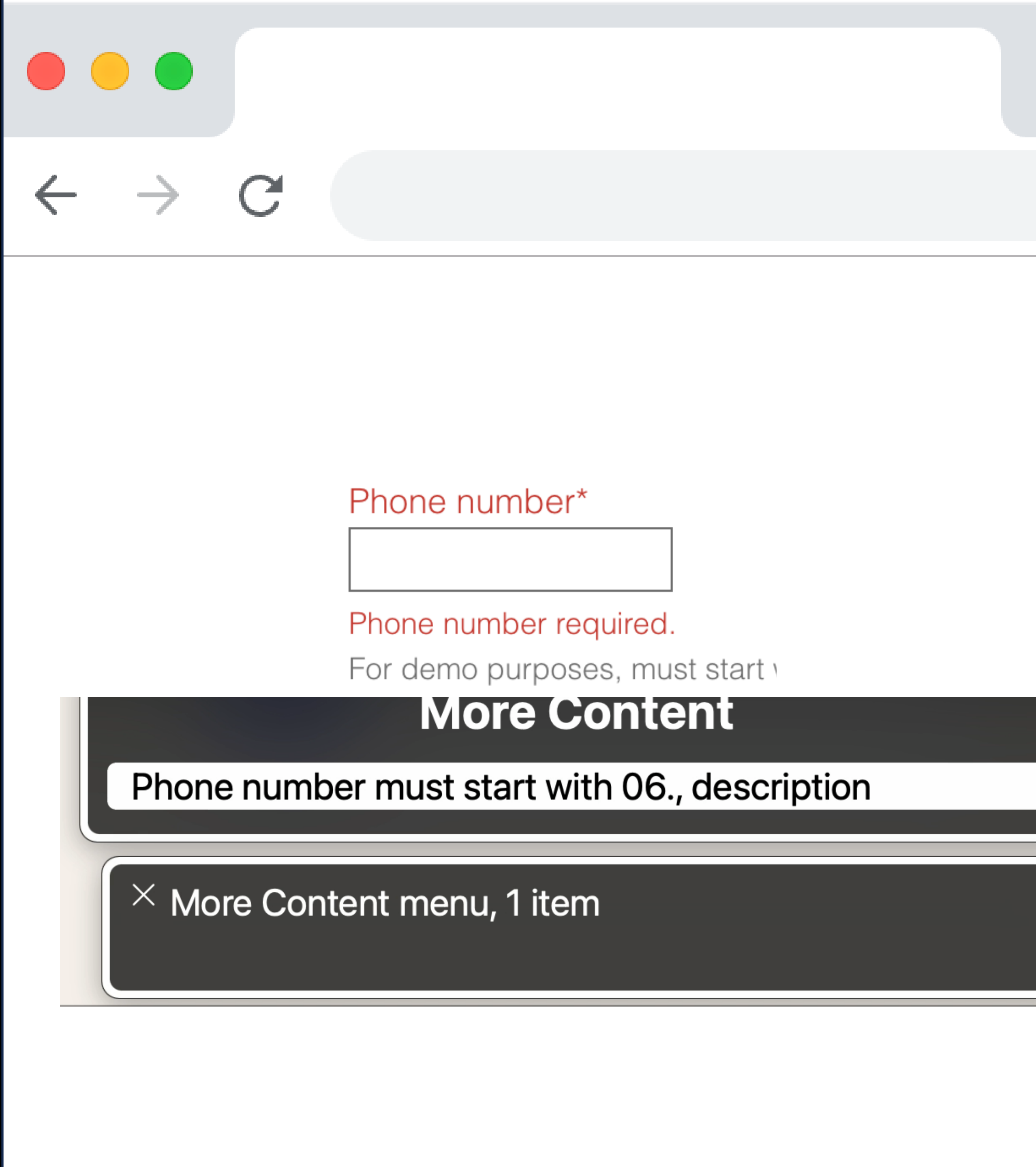
Phone number must start with 06.

More Content

Phone number must start with 06., description

More Content menu, 1 item

```html
<label for="phone">
 Phone number<span aria-hidden="true">*</span>
</label>
<input
 type="phone"
 id="phone"
 aria-required="true"
 aria-describedby="phoneHint"
/>

<p>Phone number required.</p>

<p id="phoneHint">Phone number must start with 06.</p>
```



Phone number*

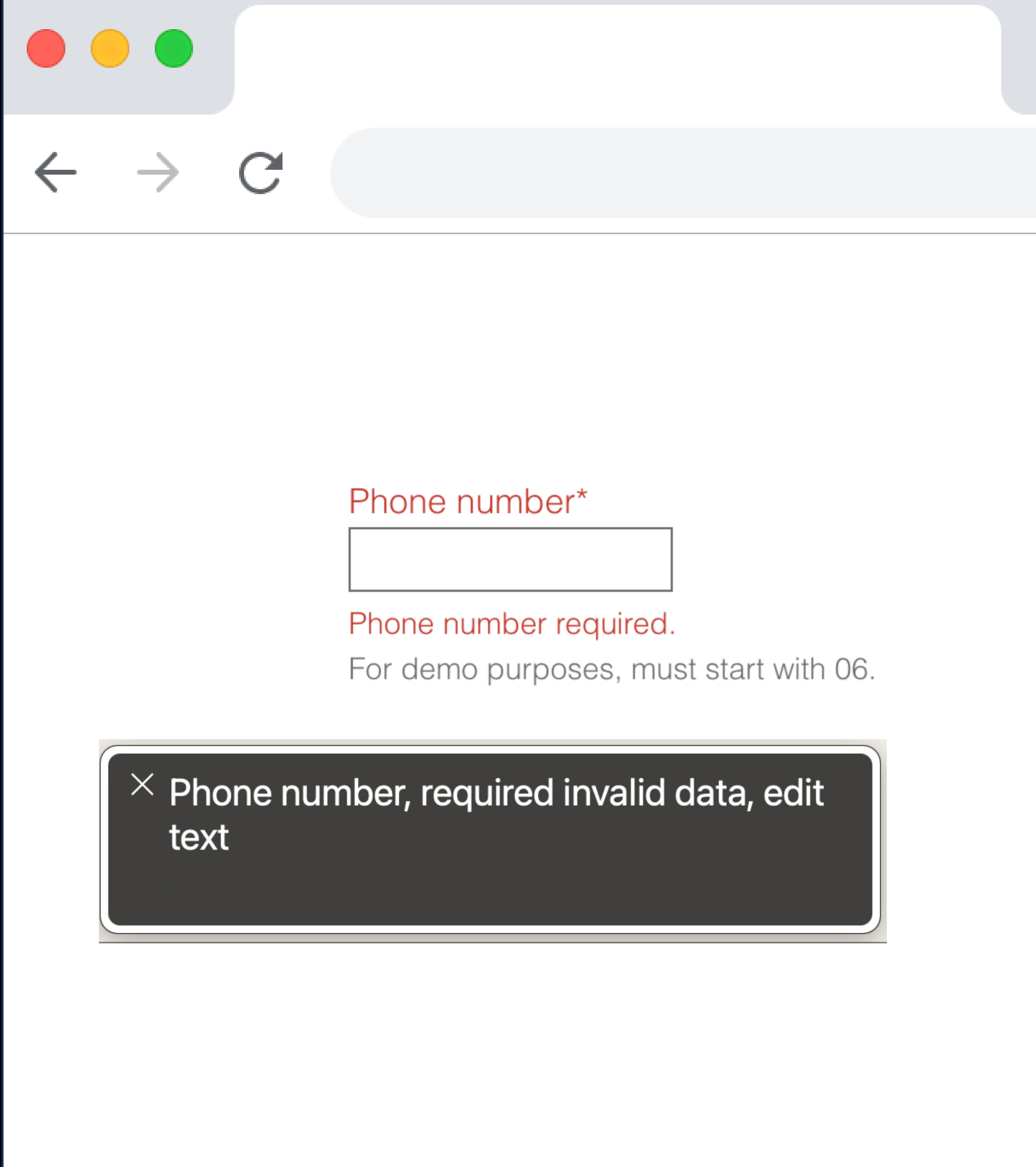
Phone number required.

For demo purposes, must start

More Content

Phone number must start with 06., description

```
<label for="phone">
 Phone number<span aria-hidden="true">*</span>
</label>
<input
 type="phone"
 id="phone"
 aria-required="true"
 aria-invalid="true"
 aria-describedby="phoneHint"
/>

<p>Phone number required.</p>

<p id="phoneHint">Phone number must start with 06.</p>
```



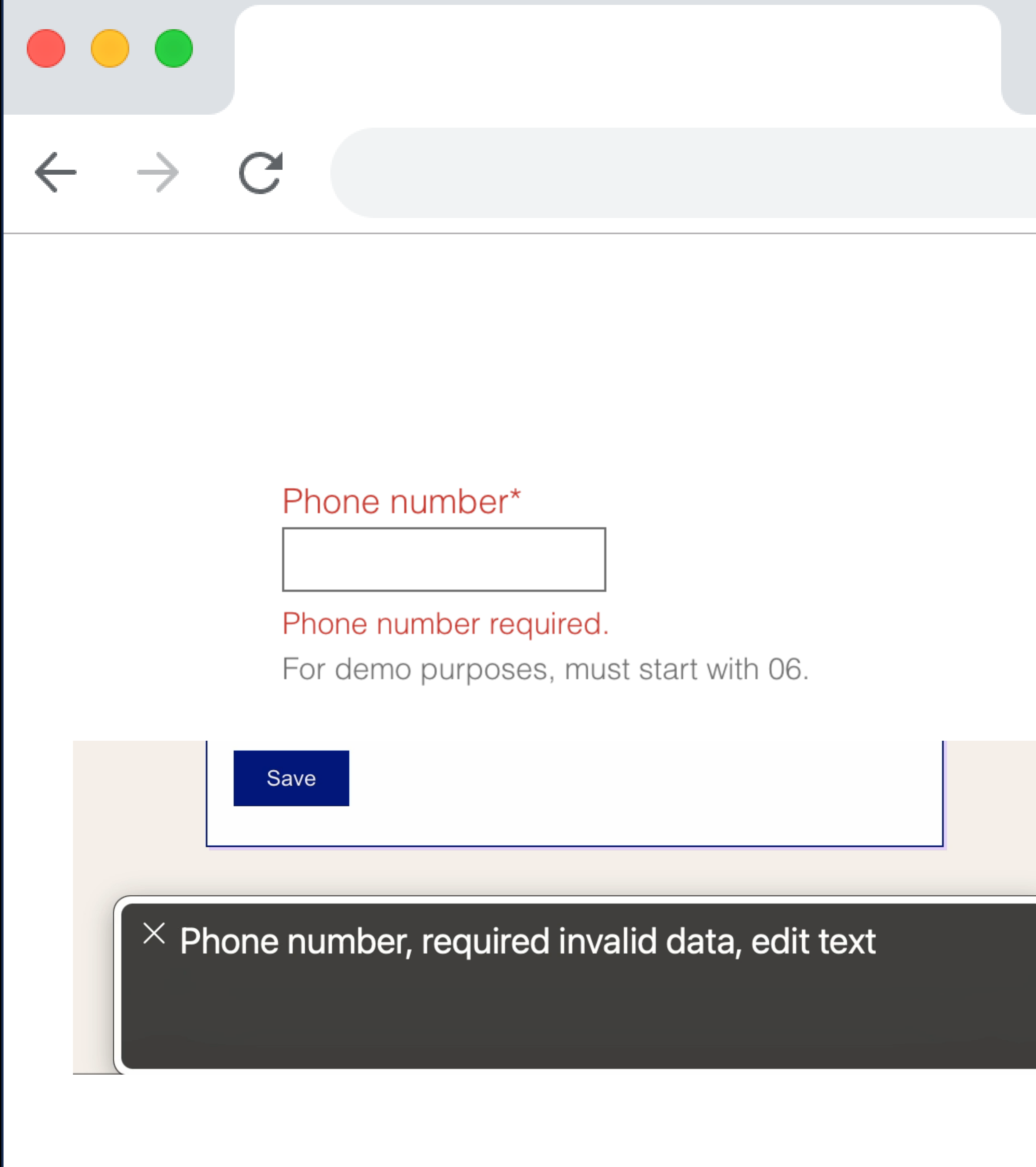
Phone number*

Phone number required.

For demo purposes, must start with 06.

Phone number, required invalid data, edit text

```
<label for="phone">
 Phone number<span aria-hidden="true">*</span>
</label>
<input
 type="phone"
 id="phone"
 aria-required="true”
 aria-invalid="true"
 aria-describedby="phoneHint phoneError"
/>

<p id="phoneError">Phone number required.</p>

<p id="phoneHint">Phone number must start with 06.</p>
```



Phone number*

Phone number required.

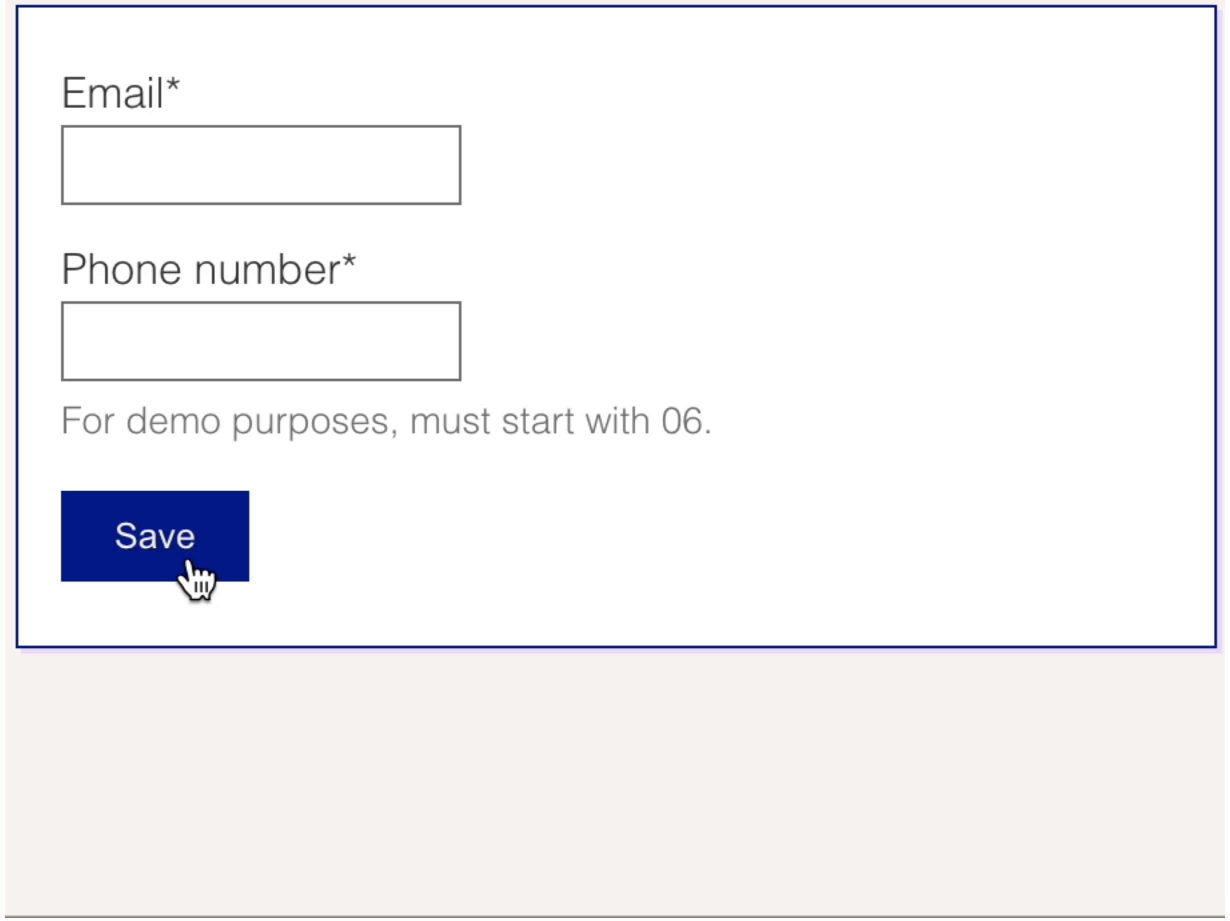
For demo purposes, must start with 06.

Phone number, required invalid data, edit text

Moment of validation

Please don’t disable the save button!

OnSubmit



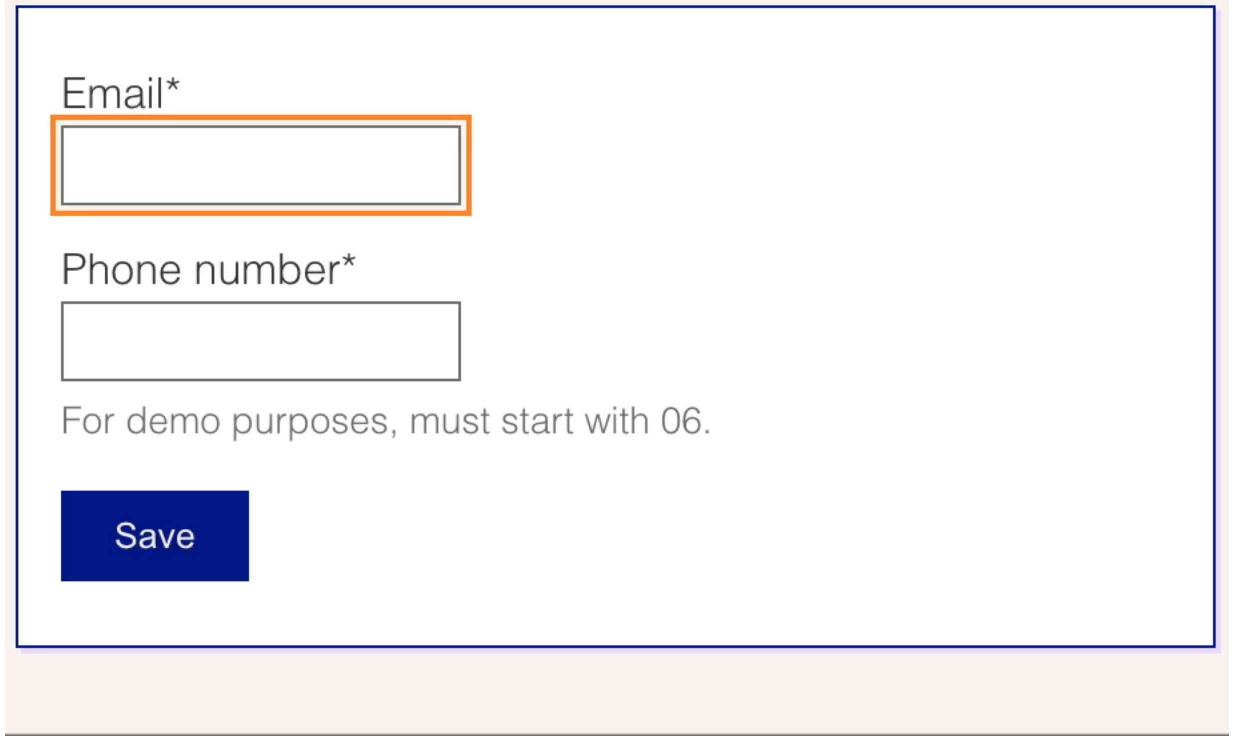
Email*

Phone number*

For demo purposes, must start with 06.

Moment of validation

OnBlur



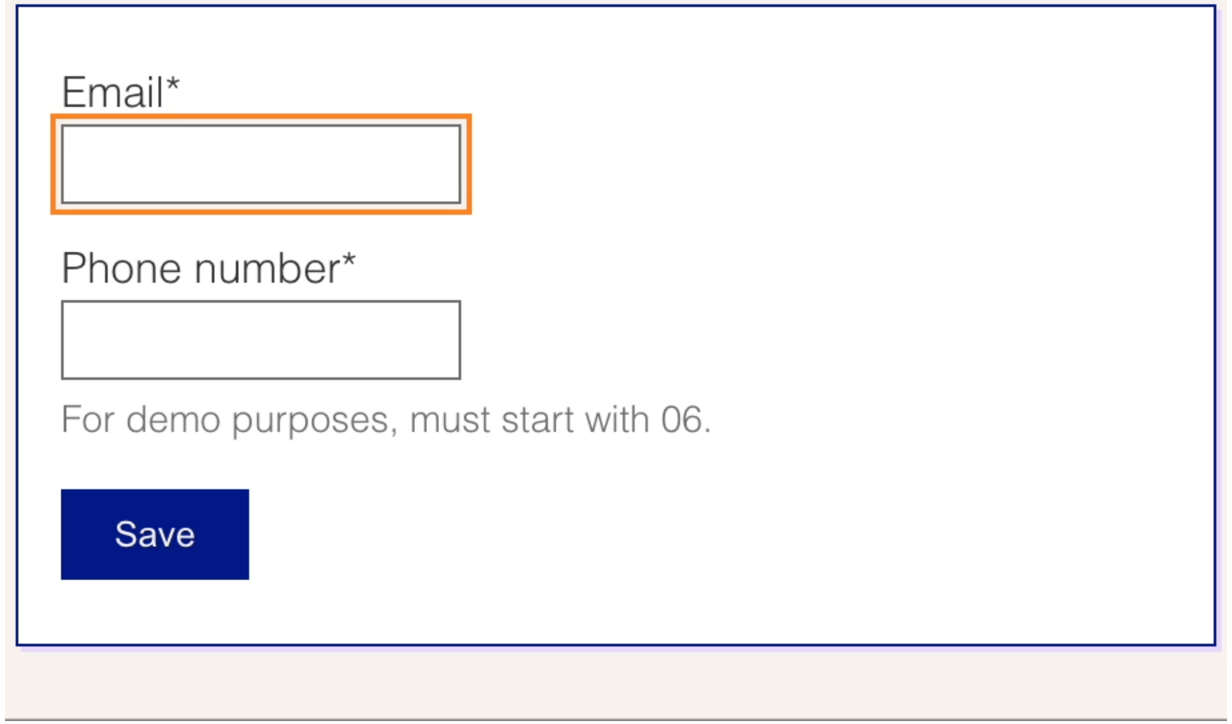
Email*

Phone number*

For demo purposes, must start with 06.

Moment of validation

OnChange



Email*

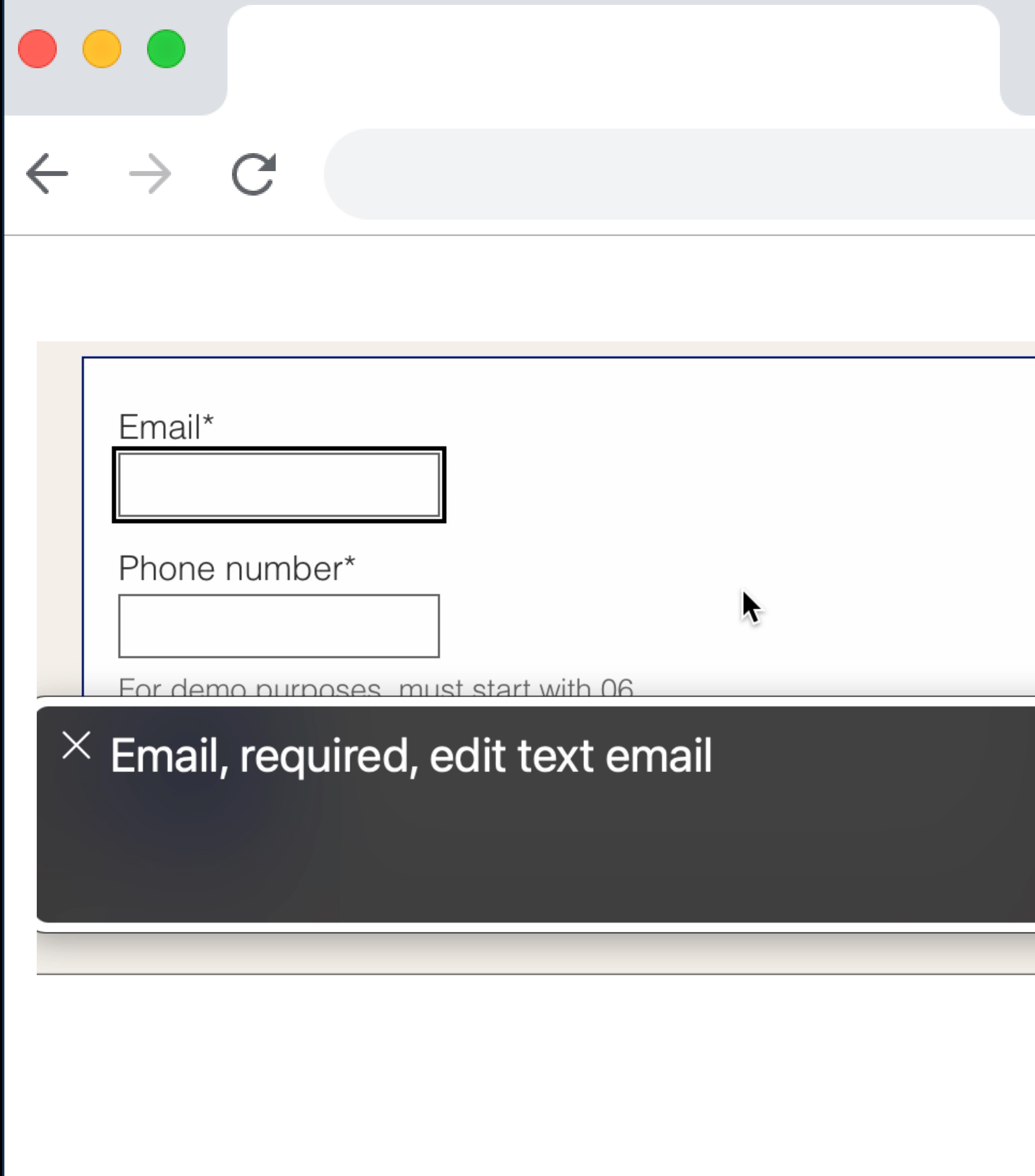
Phone number*

For demo purposes, must start with 06.

OnBlur + OnSubmit

- User gets notified while filling in the form
- User gets summary of error after submit
- Best of both worlds
- Best for medium/large forms

```
<label for="phone">
 Phone number<span aria-hidden="true">*</span>
</label>
<input
 type="phone"
 id="phone"
 aria-required="true”
 aria-invalid="true"
 aria-describedby="phoneHint phoneError"
/>

<p id="phoneError" aria-live="assertive">
 Phone number required.
</p>

<p id="phoneHint">Phone number must start with 06.</p>
```



Email*

Phone number*

For demo purposes, must start with 06

Email, required, edit text email

Let’s automate!

Vitest

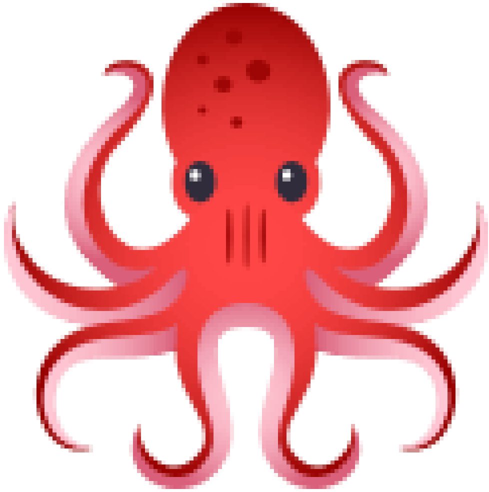
Testing-library

Guidepup

VoiceOver automation

The testing-library principles

“your test should resemble how users interact with your code as much as possible.”



Query elements

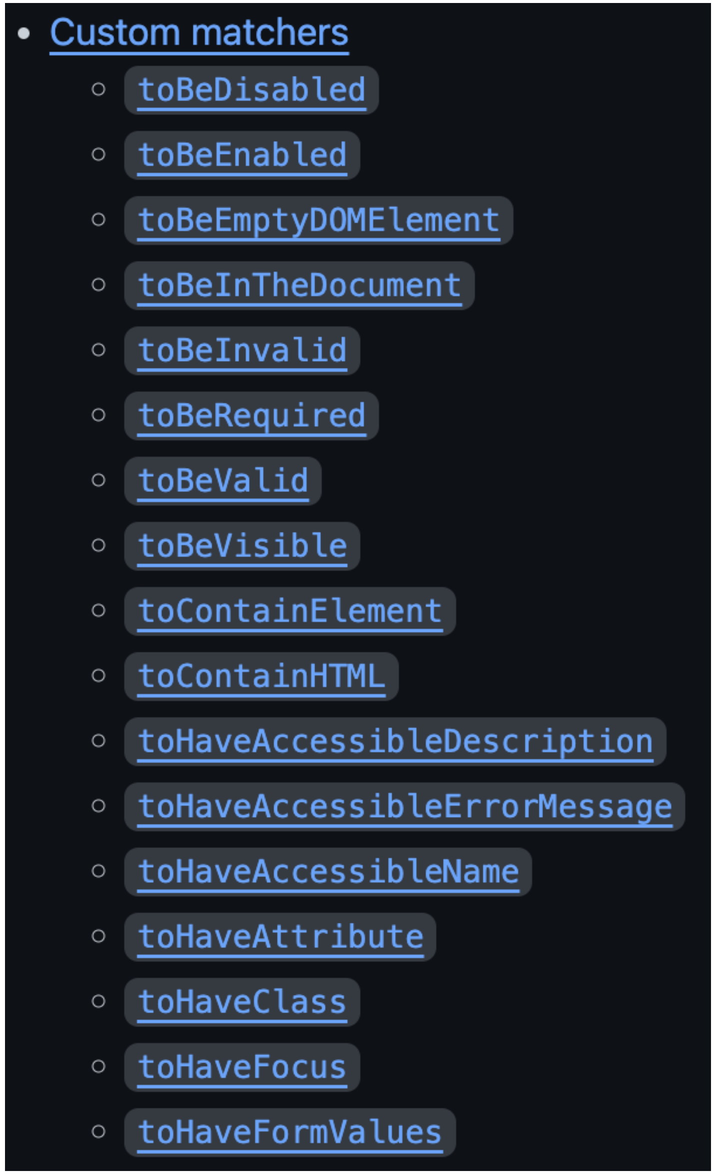
Queries Accessible to Everyone Queries that reflect the experience of visual/mouse users as well as those that use assistive technology.

1. **getByRole**: This can be used to query every element that is exposed in the accessibility tree. With the name option you can filter the returned elements by their accessible name.
2. **getByLabelText**: method is really good for form fields.
3. **getByPlaceholderText**
4. **getByText**
5. **getByDisplayValue**

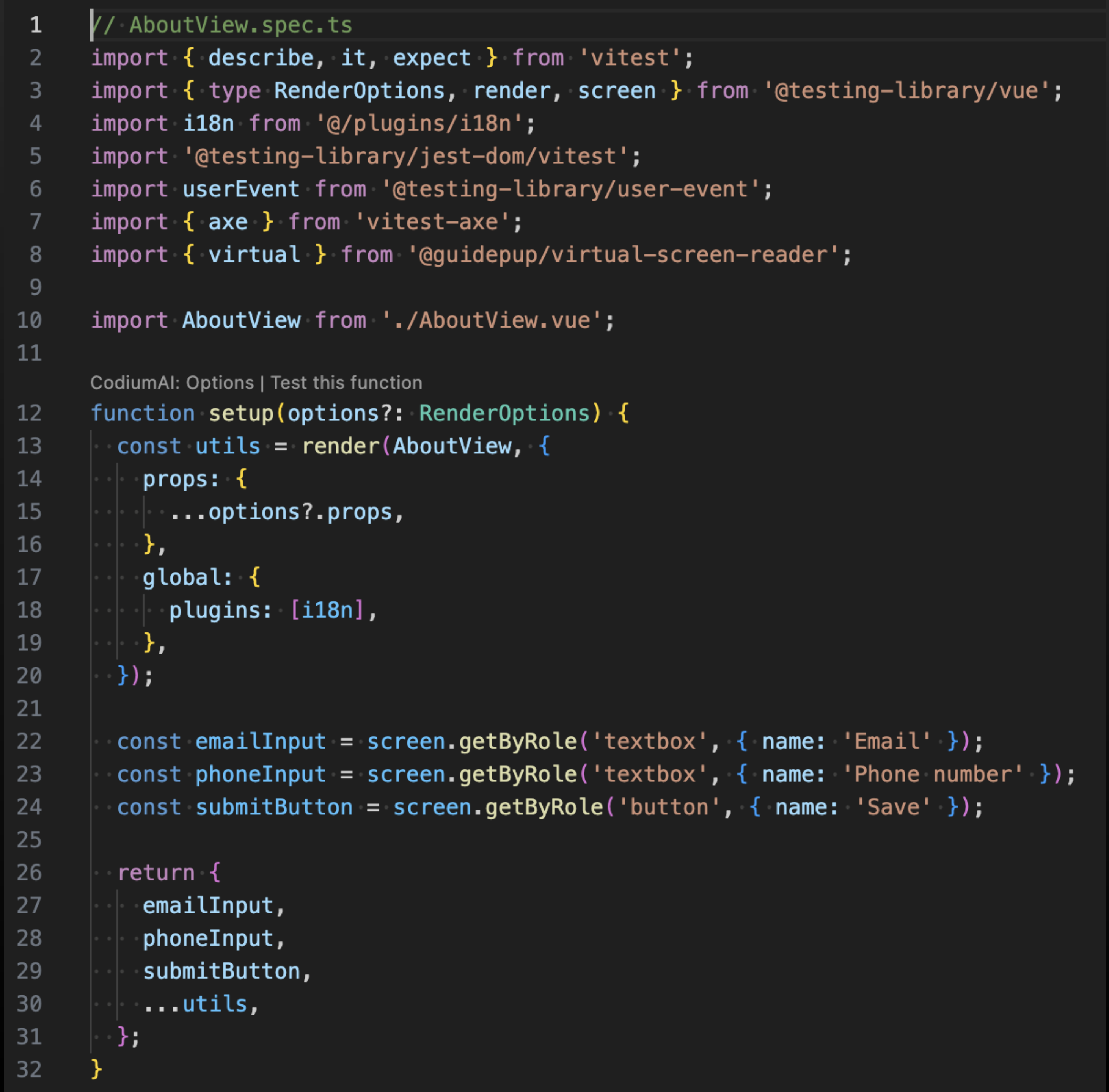
User events

```
userEvent.click(submitButton);

userEvent.type(phoneInput, '1234567890');

userEvent.tab();
```

Custom matchers

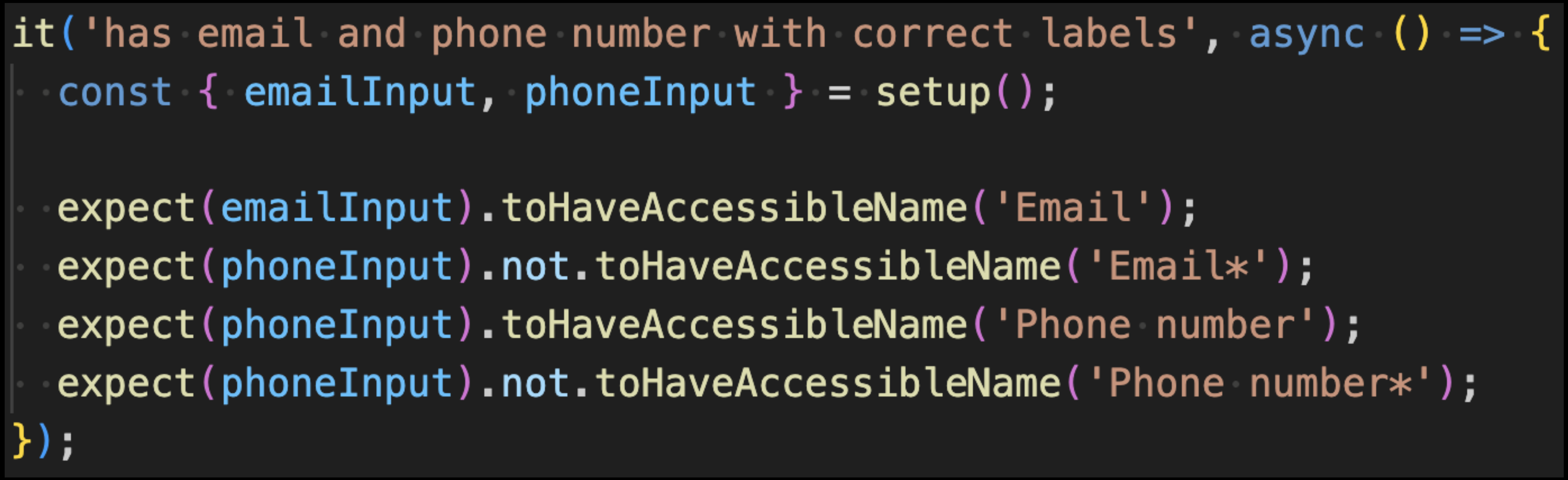


- Custom matchers
  - `toBeDisabled`
  - `toBeEnabled`
  - `toBeEmptyDOMElement`
  - `toBeInTheDocument`
  - `toBeInvalid`
  - `toBeRequired`
  - `toBeValid`
  - `toBeVisible`
  - `toContainElement`
  - `toContainHTML`
  - `toHaveAccessibleDescription`
  - `toHaveAccessibleErrorMessage`
  - `toHaveAccessibleName`
  - `toHaveAttribute`
  - `toHaveClass`
  - `toHaveFocus`
  - `toHaveFormValues`



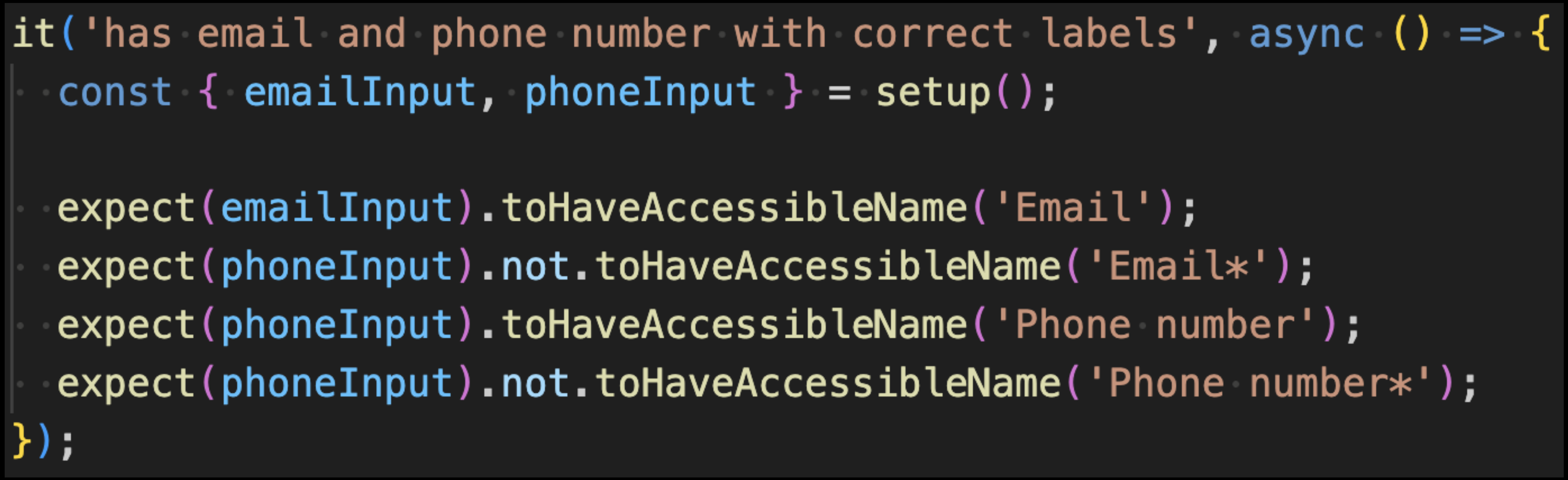
```ts
// AboutView.spec.ts
import { describe, it, expect } from 'vitest';
import { type RenderOptions, render, screen } from '@testing-library/vue';
import i18n from '@/plugins/i18n';
import '@testing-library/jest-dom/vitest';
import userEvent from '@testing-library/user-event';
import { axe } from 'vitest-axe';
import { virtual } from '@guidepup/virtual-screen-reader';

import AboutView from './AboutView.vue';

function setup(options?: RenderOptions) {
  const utils = render(AboutView, {
    props: {
      ...options?.props,
    },
    global: {
      plugins: [i18n],
    },
  });

  const emailInput = screen.getByRole('textbox', { name: 'Email' });
  const phoneInput = screen.getByRole('textbox', { name: 'Phone number' });
  const submitButton = screen.getByRole('button', { name: 'Save' });

  return {
    emailInput,
    phoneInput,
    submitButton,
    ...utils,
  };
}
```

```html
<label for="phone">Phone number</label>
<input id="phone" type="phone"/>
```



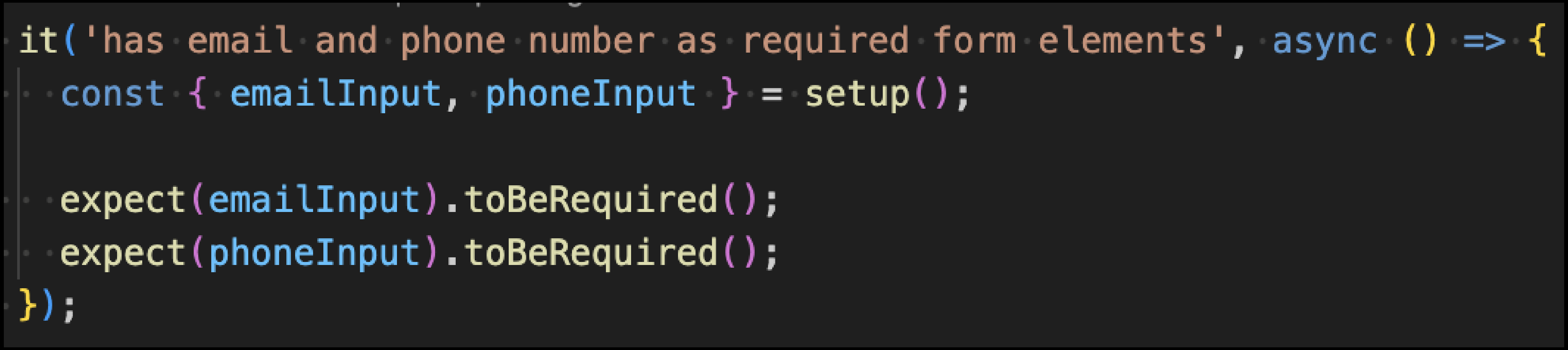
```javascript
it('has email and phone number with correct labels', async () => {
  const { emailInput, phoneInput } = setup();

  expect(emailInput).toHaveAccessibleName('Email');
  expect(phoneInput).not.toHaveAccessibleName('Email*');
  expect(phoneInput).toHaveAccessibleName('Phone number');
  expect(phoneInput).not.toHaveAccessibleName('Phone number*');
});
```

```html
<label for="phone">
  Phone number<span aria-hidden="true">*</span>
</label>
<input type="phone" id="phone"/>
```



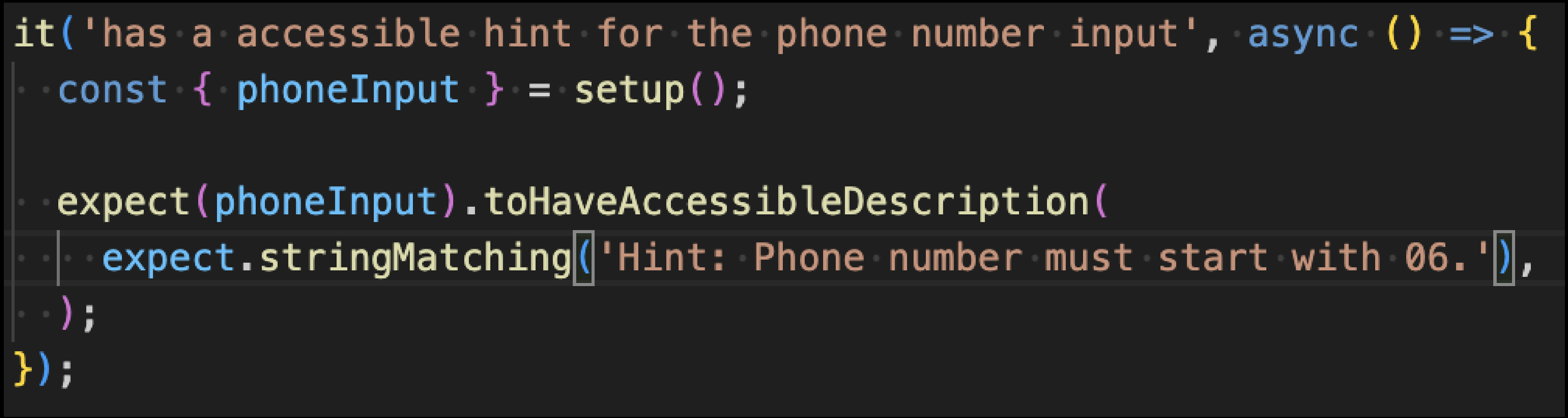
```javascript
it('has email and phone number with correct labels', async () => {
  const { emailInput, phoneInput } = setup();

  expect(emailInput).toHaveAccessibleName('Email');
  expect(phoneInput).not.toHaveAccessibleName('Email*');
  expect(phoneInput).toHaveAccessibleName('Phone number');
  expect(phoneInput).not.toHaveAccessibleName('Phone number*');
});
```

```html
<label for="phone">
  Phone number<span aria-hidden="true">*</span>
</label>
<input type="phone" id="phone" aria-required="true"/>
```



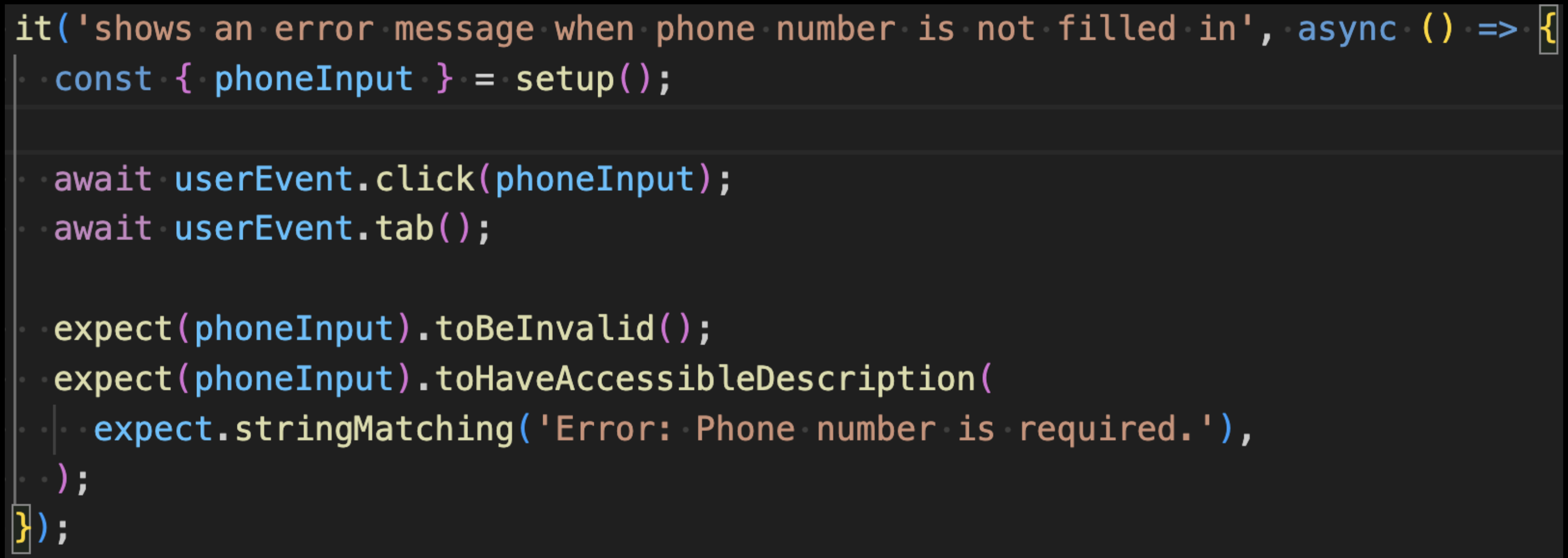
```javascript
it('has email and phone number as required form elements', async () => {
  const { emailInput, phoneInput } = setup();

  expect(emailInput).toBeRequired();
  expect(phoneInput).toBeRequired();
});
```

```html
<label for="phone">
  Phone number<span aria-hidden="true">*</span>
</label>
<input
  type="phone"
  id="phone"
  aria-required="true"
  aria-describedby="phoneHint"
/>

<p id="phoneHint">Phone number must start with 06.</p>
```



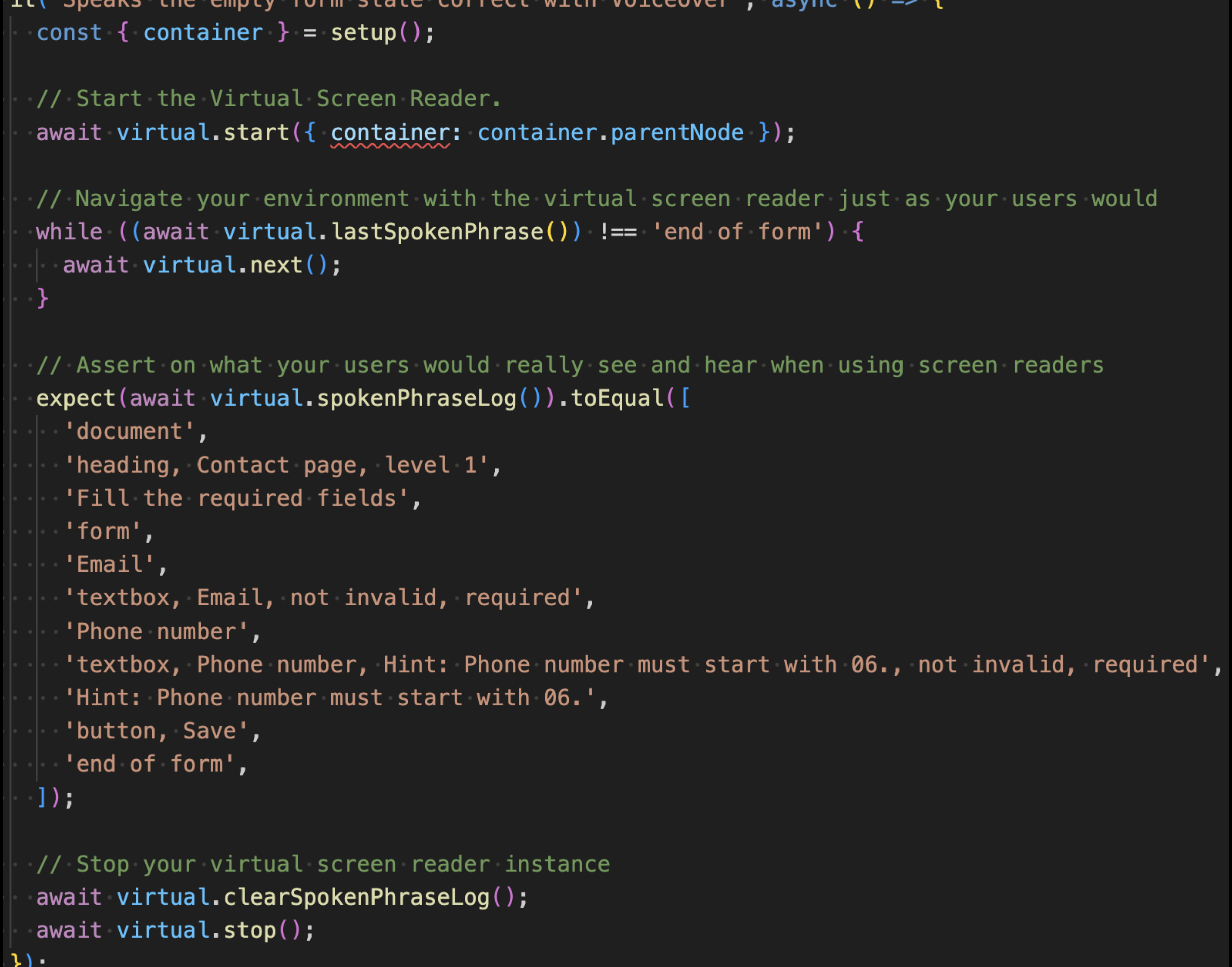
```javascript
it('has a accessible hint for the phone number input', async () => {
  const { phoneInput } = setup();

  expect(phoneInput).toHaveAccessibleDescription(
    expect.stringMatching('Hint: Phone number must start with 06.'),
  );
});
```

```html
<label for="phone">
  Phone number<span aria-hidden="true">*</span>
</label>
<input
  type="phone"
  id="phone"
  aria-required="true”
  aria-invalid="true"
  aria-describedby="phoneHint phoneError"
/>

<p id="phoneError">Phone number required.</p>

<p id="phoneHint">Phone number must start with 06.</p>
```

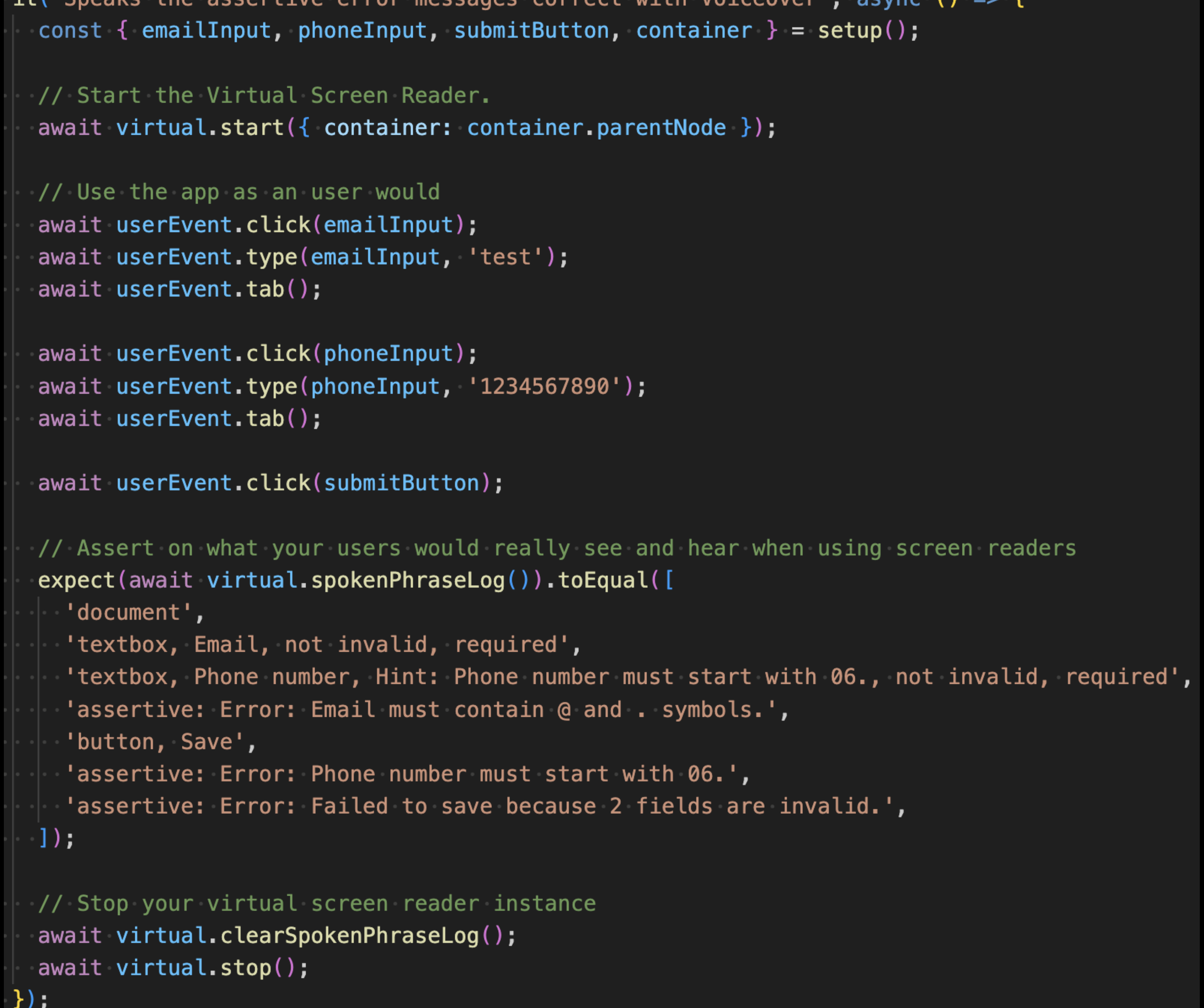


```javascript
it('shows an error message when phone number is not filled in', async () => {
  const { phoneInput } = setup();

  await userEvent.click(phoneInput);
  await userEvent.tab();

  expect(phoneInput).toBeInvalid();
  expect(phoneInput).toHaveAccessibleDescription(
    expect.stringMatching('Error: Phone number is required.'),
  );
});
```

```html
<label for="phone">
  Phone number<span aria-hidden="true">*</span>
</label>
<input
  type="phone"
  id="phone"
  aria-required="true”
  aria-invalid="true"
  aria-describedby="phoneHint phoneError"
/>

<p id="phoneError" aria-live="assertive">
  Phone number required.
</p>

<p id="phoneHint">Phone number must start with 06.</p>
```



```
it('Speaks the empty form state correct with voiceover', async () => {
  const { container } = setup();

  // Start the Virtual Screen Reader.
  await virtual.start({ container: container.parentNode });

  // Navigate your environment with the virtual screen reader just as your users would
  while ((await virtual.lastSpokenPhrase()) !== 'end of form') {
    await virtual.next();
  }

  // Assert on what your users would really see and hear when using screen readers
  expect(await virtual.spokenPhraseLog()).toEqual([
    'document',
    'heading, Contact page, level 1',
    'Fill the required fields',
    'form',
    'Email',
    'textbox, Email, not invalid, required',
    'Phone number',
    'textbox, Phone number, Hint: Phone number must start with 06., not invalid, required',
    'Hint: Phone number must start with 06.',
    'button, Save',
    'end of form',
  ]);

  // Stop your virtual screen reader instance
  await virtual.clearSpokenPhraseLog();
  await virtual.stop();
});
```



```javascript
it('Speaks the assertive error messages correct with voiceover', async () => {
  const { emailInput, phoneInput, submitButton, container } = setup();

  // Start the Virtual Screen Reader.
  await virtual.start({ container: container.parentNode });

  // Use the app as an user would
  await userEvent.click(emailInput);
  await userEvent.type(emailInput, 'test');
  await userEvent.tab();

  await userEvent.click(phoneInput);
  await userEvent.type(phoneInput, '1234567890');
  await userEvent.tab();

  await userEvent.click(submitButton);

  // Assert on what your users would really see and hear when using screen readers
  expect(await virtual.spokenPhraseLog()).toEqual([
    'document',
    'textbox, Email, not invalid, required',
    'textbox, Phone number, Hint: Phone number must start with 06., not invalid, required',
    'assertive: Error: Email must contain @ and . symbols.',
    'button, Save',
    'assertive: Error: Phone number must start with 06.',
    'assertive: Error: Failed to save because 2 fields are invalid.',
  ]);

  // Stop your virtual screen reader instance
  await virtual.clearSpokenPhraseLog();
  await virtual.stop();
});
```

Wrapping up

- **why:** working on accessibility has a lot of benefits.
- **for**: To connect labels to inputs.
- **aria-hidden**: To hide elements to the screen reader.
- **aria-required**: To mark a field as required.
- **aria-invalid**: To mark the field as invalid.
- **aria-describedby**: To connect the description and error message to the input.
- **aria-live**: To announce dynamic error messages.
- We automaticly tested all these accessibility improvements with Testing-library and Guidepup

Questions?

Let’s connect!

- www.timdamen.io
- Linkedin: