Reach worldwide audiences by implementing Internationalization

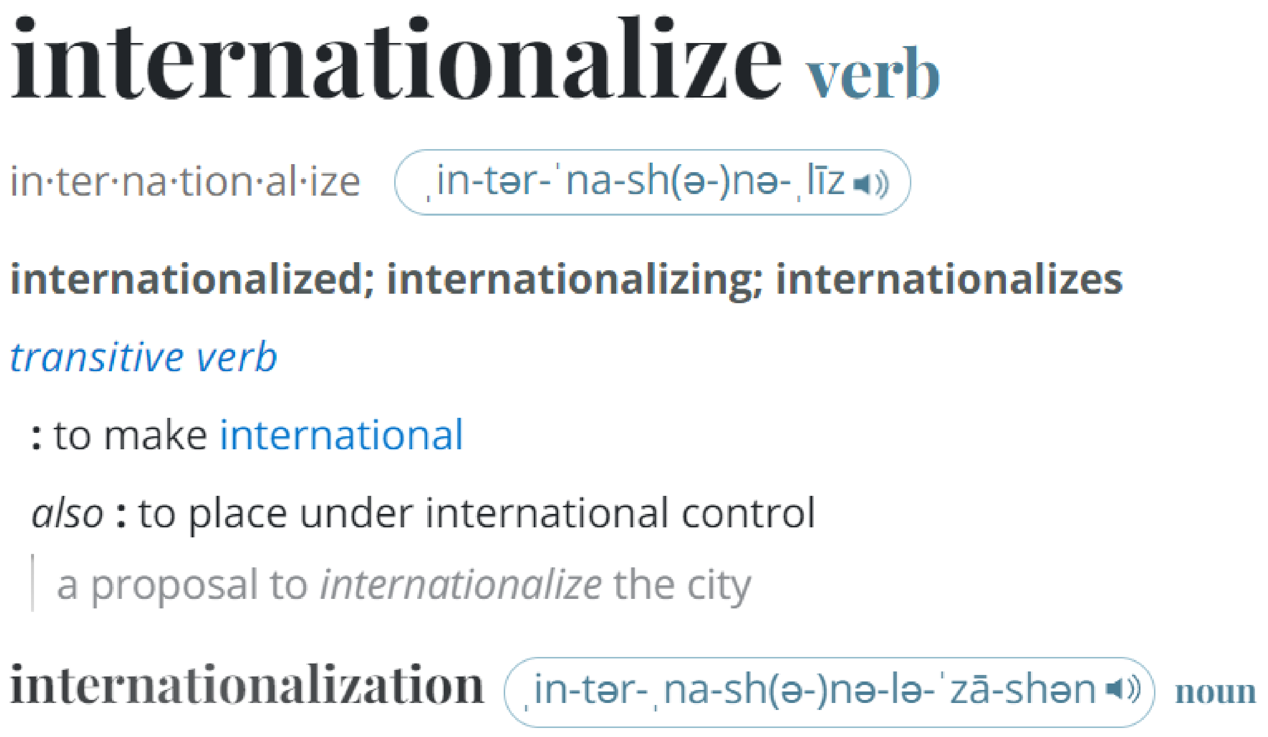
Rodney Wormsbecher

Github:
https://github.com/rwormsbecher/workshop-internationalization

Raise your hand if …

- You have ever supported more than 1 language on a website/app via translated text.
- You have formatted dates for different cultures and/or time zones.
- You have dealt with pluralization.
- You have added relative time formats. E.g. “2 hours ago”.
- You have gone through displaying numbers correctly for sanctioned units. E.g. Celsius, Fahrenheit, currencies, mile/kilometers, etc.

What is internationalization



internationalize verb

in·ter·na·tion·al·ize ˌin-tər-ˈna-sh(ə-)nə-ˌlīz

internationalized; internationalizing; internationalizes

transitive verb

: to make international

also **:** to place under international control

a proposal to *internationalize* the city

internationalization ˌin-tər-ˌna-sh(ə-)nə-lə-ˈzā-shən noun

What is internationalization

In order words, we want to make a website/web app international.

Why internationalization

In order to allow a bigger percentage of the world population to interact with our websites, web apps, e-commerce platforms, or other digital informational products, We must make sure that people around the world can interact with our product in their own language and tailored to their cultural-specific standards.

Reminder: 09/02/2023 Free pizza for everyone!

Tech giant’s support

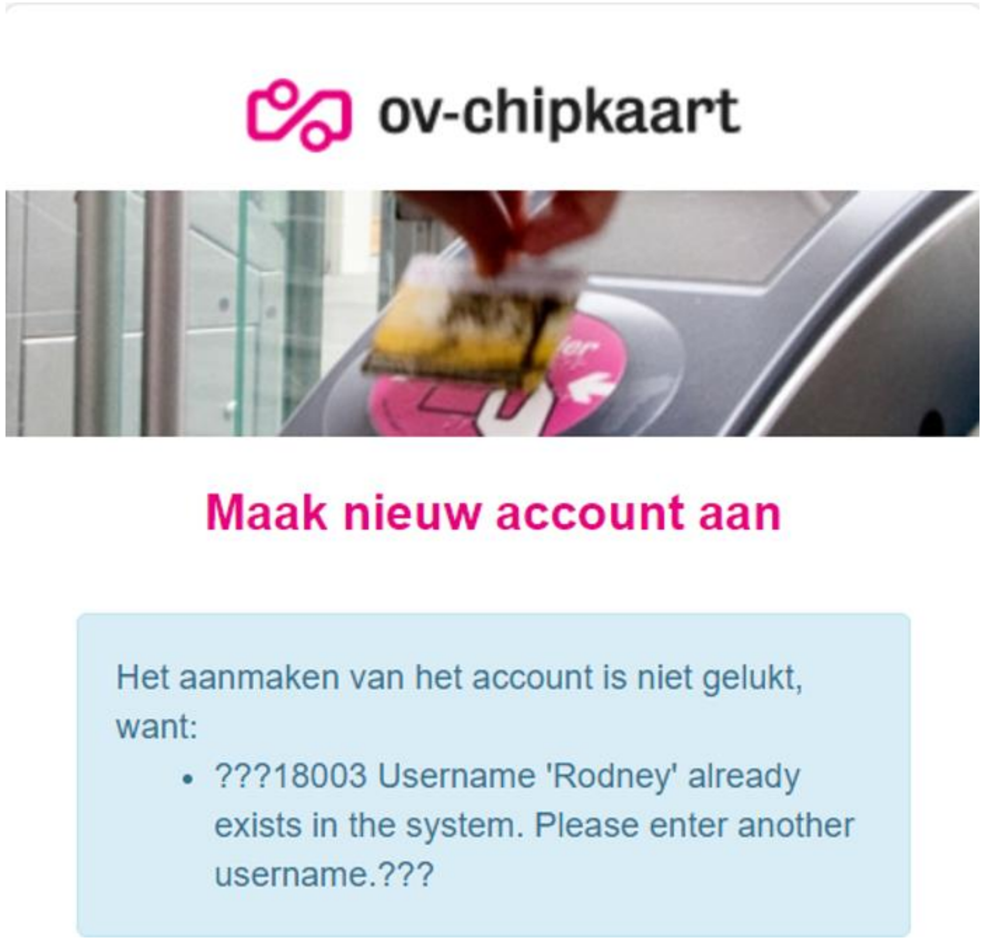
Current tech Giants have made a tremendous effort to support as many languages as possible.

Software	Number of languages supported
Windows	110
Mac OSX	41
Google search	348

Common challenges we face

- Translating strings and pluralization.
- Dealing with date and time.
- Display ingnumbers with the correct radix characters.
- Displaying sanctioned units.
- Displaying currencies.
- Adding relative time.

How not to do it



Maak nieuw account aan

Het aanmaken van het account is niet gelukt, want:

- ???18003 Username 'Rodney' already exists in the system. Please enter another username.???

Common Internationalization Solutions

Existing packages

Many software engineers before us have already invented and reinvented the reinvented wheel. Tested and used strategies oftentimes work better than writing it from scratch and thus we will take a look at the most popular packages.

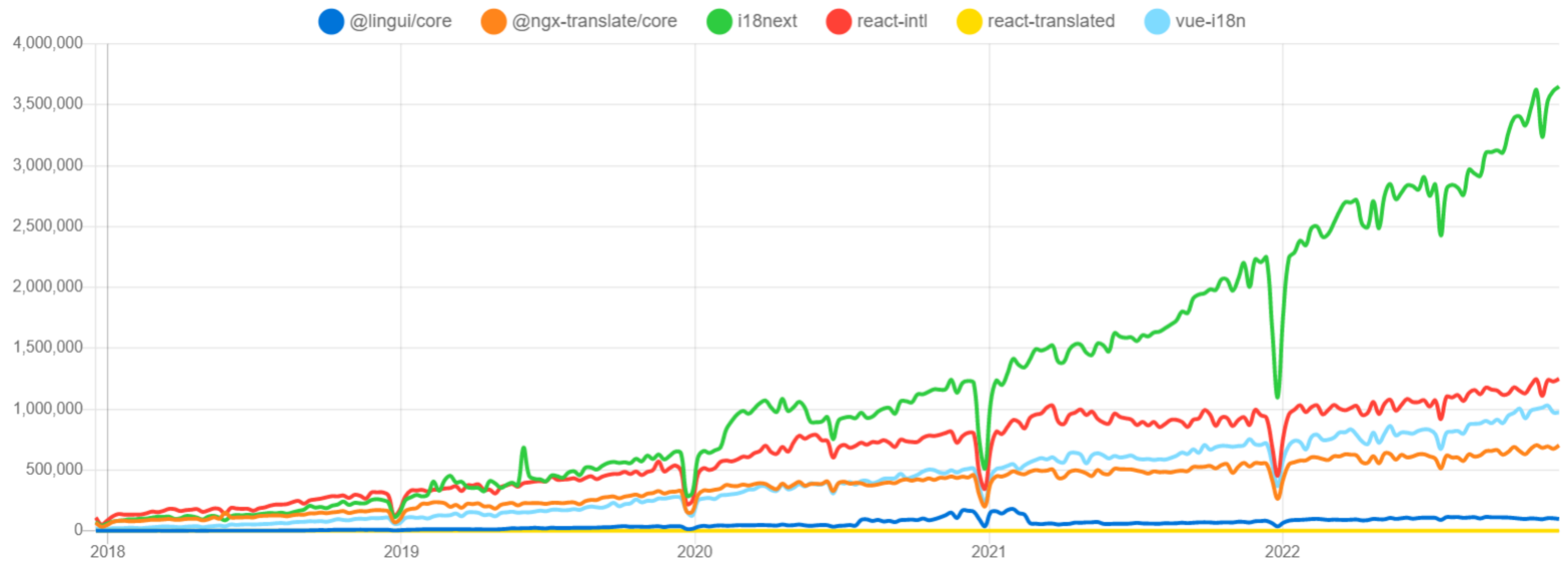
Existing packages

In the JavaScript world, these are the most notable contemporary internationalization packages.

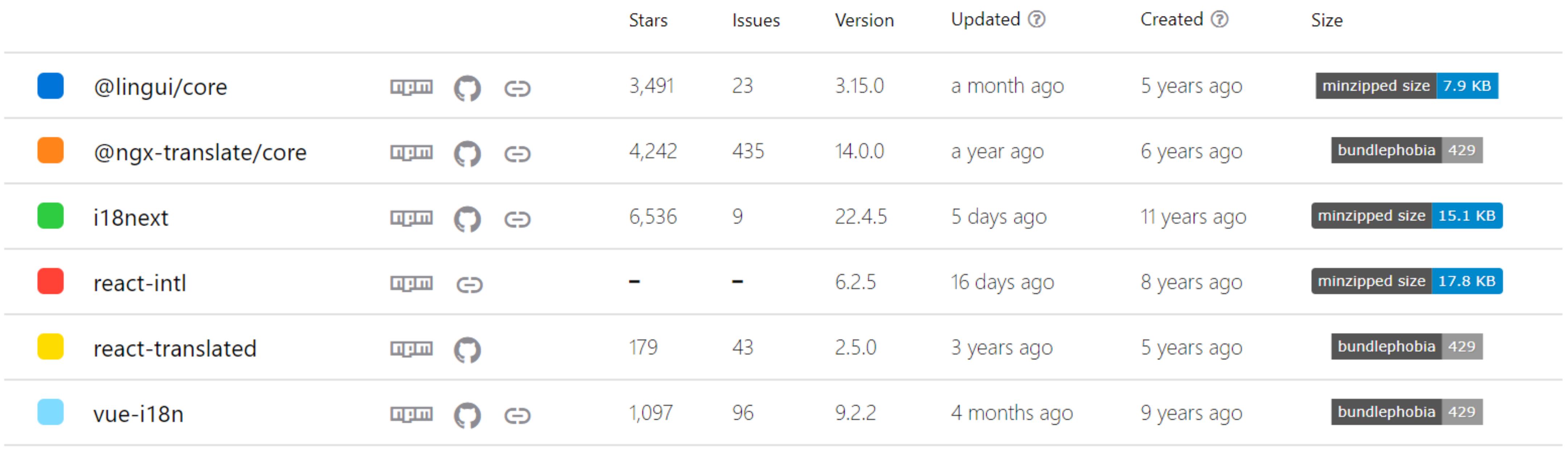
Package name	Supported frameworks
I18next	Any framework
React-intl (format-js)	Previously only React, now any framework
@ngx-translations	Angular
Vue-i18n	Vue.js
React-translated	React.js
@lingui/core	Any framework

Existing packages

Downloads in past 5 Years



Existing packages



Stats

		Stars	Issues	Version	Updated	Created	Size
🟦	@lingui/core	3,491	23	3.15.0	a month ago	5 years ago	minzipped size 7.9 KB
🟧	@ngx-translate/core	4,242	435	14.0.0	a year ago	6 years ago	bundlephobia 429
🟩	i18next	6,536	9	22.4.5	5 days ago	11 years ago	minzipped size 15.1 KB
🟥	react-intl	-	-	6.2.5	16 days ago	8 years ago	minzipped size 17.8 KB
🟨	react-translated	179	43	2.5.0	3 years ago	5 years ago	bundlephobia 429
🟦	vue-i18n	1,097	96	9.2.2	4 months ago	9 years ago	bundlephobia 429

Setting up React-intl

History

To support its international audience, Yahoo wrote React-intl in 2014. It tried to make internationalization as simple as possible, supporting the ICU internationalization format.

The future

React-intl has recently been rebranded to Format.js to fit in an ecosystem of core libraries, that build on the JavaScript Intl built-ins and industry-wide i18n standards, plus a set of integrations for common template and component libraries. Supporting any framework.

The scenario

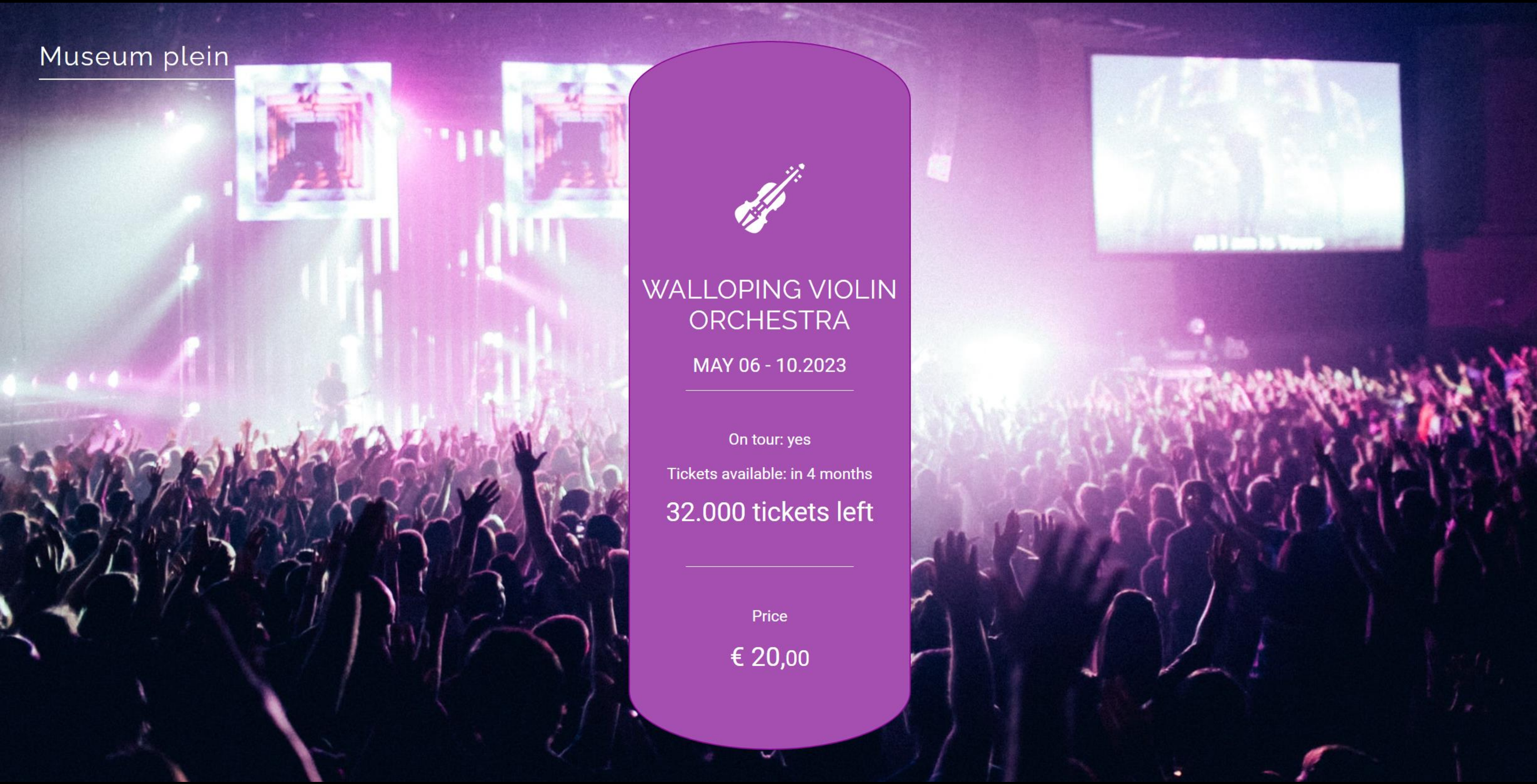
In this presentation, we will take a look at a website that does not adhere to any internationalization standards, and gradually I will show you how to make it internationalized.

The scenario

In order to run the project we must do 4 things.

- Download the source code here
- Run “npm install”
- Run “npm install –S react-intl”
- Run “npm start”

Museum plein



WALLOPING VIOLIN ORCHESTRA

MAY 06 - 10.2023

On tour: yes

Tickets available: in 4 months

32.000 tickets left

Price

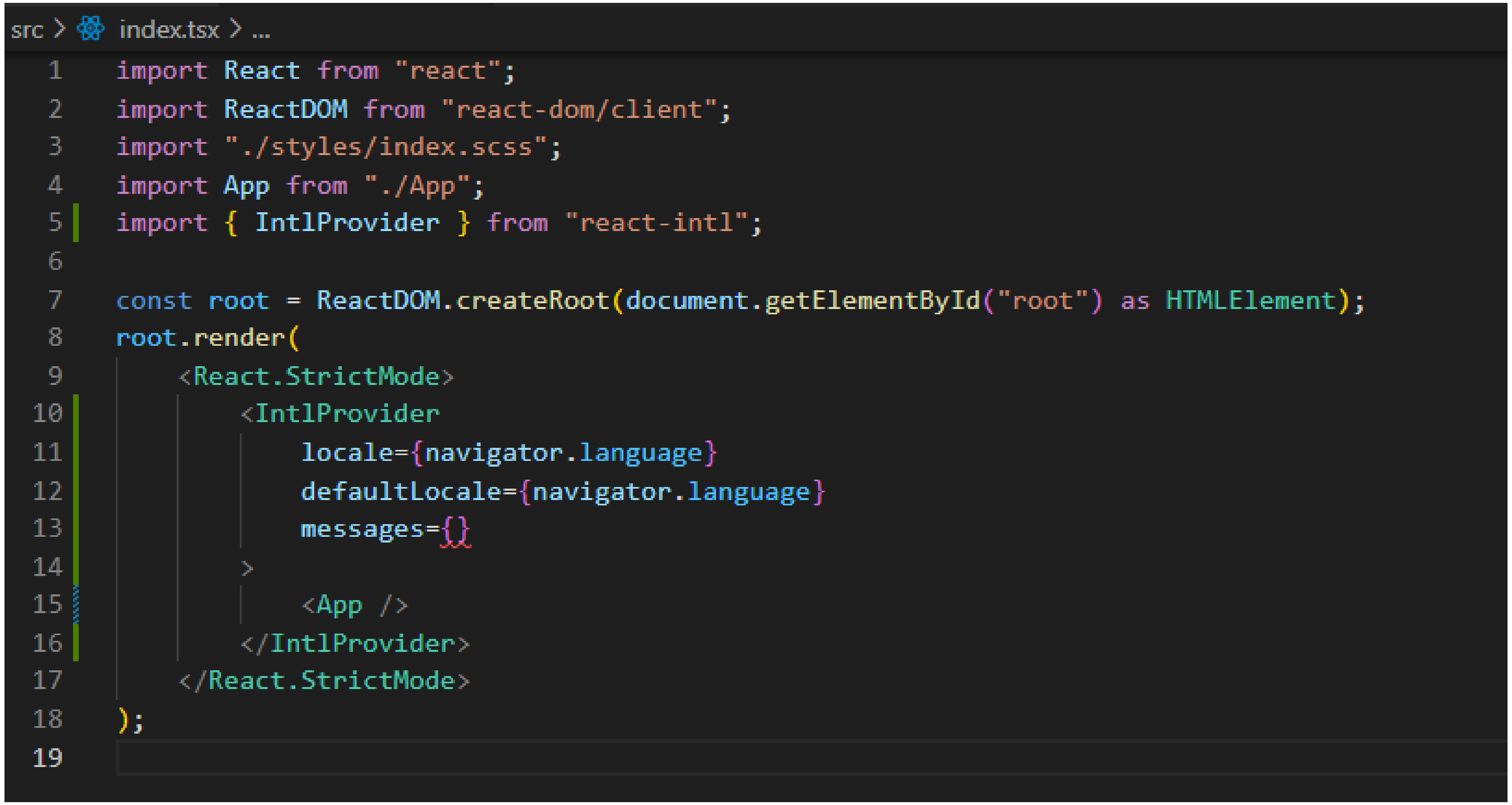
€ 20,00

Providing the Provider

In order to use React-intl it must be provided in the code (preferably at bootstrap time). The Intl provider takes required parameters:

1. Messages: A Record<string, string> type, containing the ids and values to be translated.
2. Locale: The locale we want to make use of. React-intl supports ISO 639-1 and ISO 639-3 which enable locale code like “en-us” or “nl”.

Providing the Provider

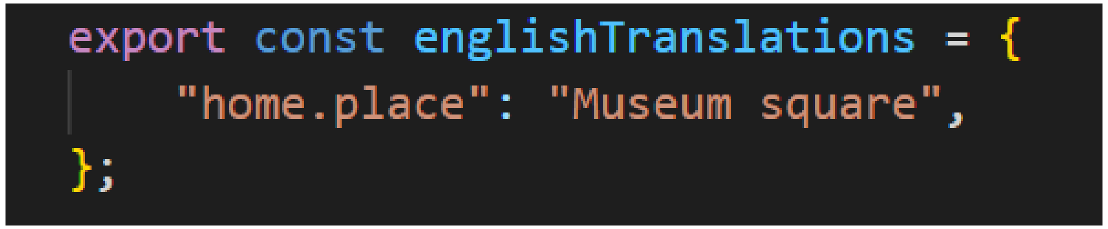

src > index.tsx > ...

```tsx
import React from "react";
import ReactDOM from "react-dom/client";
import "./styles/index.scss";
import App from "./App";
import { IntlProvider } from "react-intl";

const root = ReactDOM.createRoot(document.getElementById("root") as HTMLElement);
root.render(
    <React.StrictMode>
        <IntlProvider
            locale={navigator.language}
            defaultLocale={navigator.language}
            messages={}
        >
            <App />
        </IntlProvider>
    </React.StrictMode>
);

```

Creating the language files

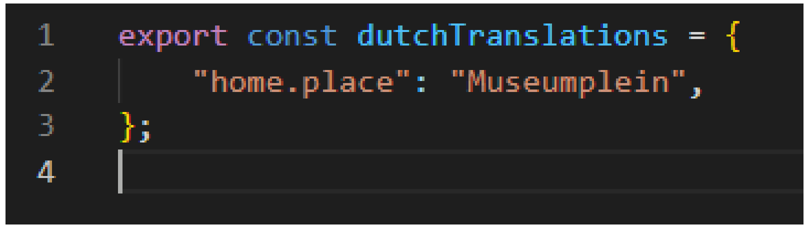
Next we create the language files we need:

src/i18n/lang/en.ts



```
export const englishTranslations = {
    "home.place": "Museum square",
};
```

src/i18n/lang/nl.ts

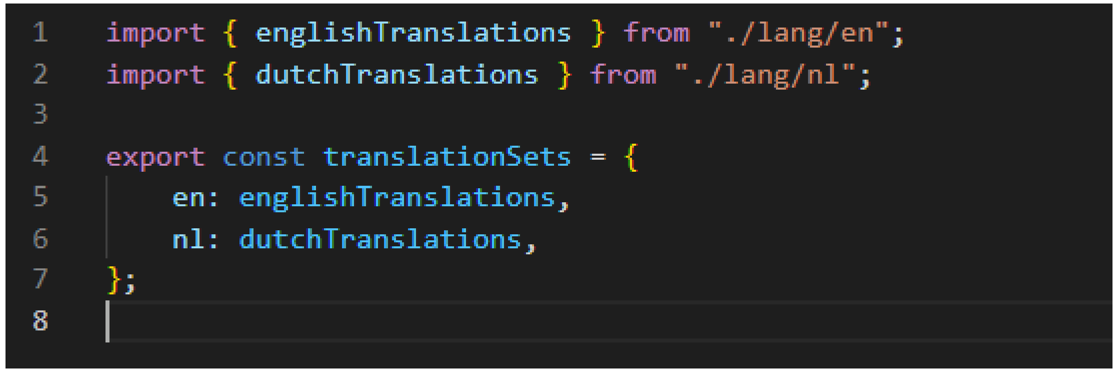


```
export const dutchTranslations = {
    "home.place": "Museumplein",
};

```

Exporting the languages

Create a file that will export all languages within a single object, this way we can keep our index.ts file neat.

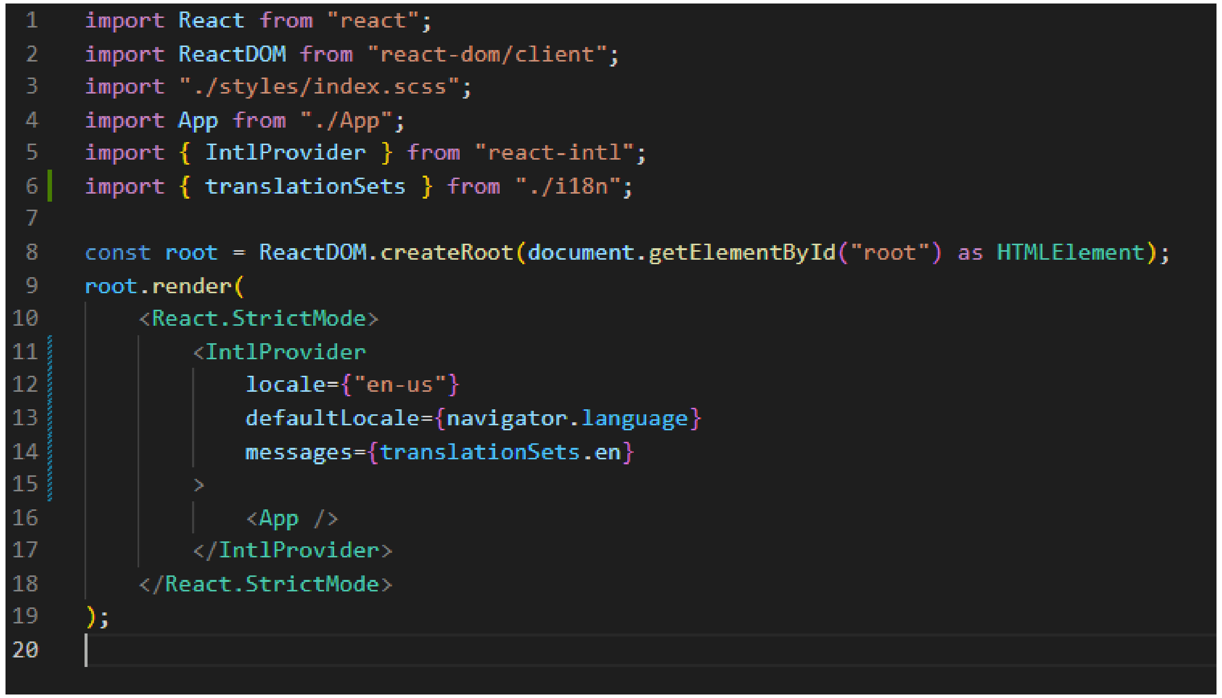
src/i18n/index.ts



```
import { englishTranslations } from "./lang/en";
import { dutchTranslations } from "./lang/nl";

export const translationSets = {
    en: englishTranslations,
    nl: dutchTranslations,
};

```

Provide the language file

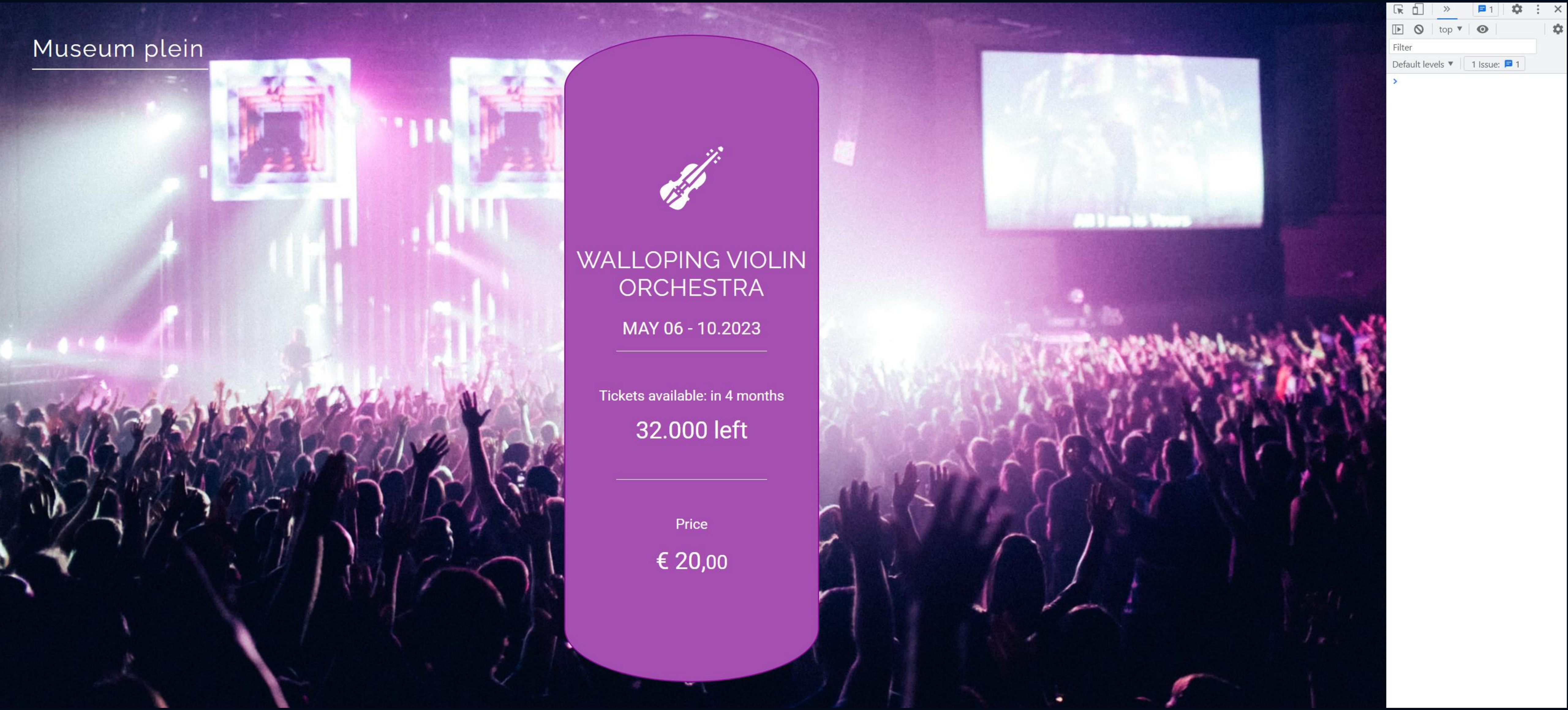
Import the “translationSets” Object to the messages attribute of the <IntlProvider>. During this presentation I will make use of “nl”, “en-us” and “zh-cn”.



```tsx
import React from "react";
import ReactDOM from "react-dom/client";
import "./styles/index.scss";
import App from "./App";
import { IntlProvider } from "react-intl";
import { translationSets } from "./i18n";

const root = ReactDOM.createRoot(document.getElementById("root") as HTMLElement);
root.render(
    <React.StrictMode>
        <IntlProvider
            locale={"en-us"}
            defaultLocale={navigator.language}
            messages={translationSets.en}
        >
            <App />
        </IntlProvider>
    </React.StrictMode>
);
```

The big moment

Are you ready?



Museum plein

WALLOPING VIOLIN ORCHESTRA

MAY 06 - 10.2023

Tickets available: in 4 months

32.000 left

Price

€ 20,00

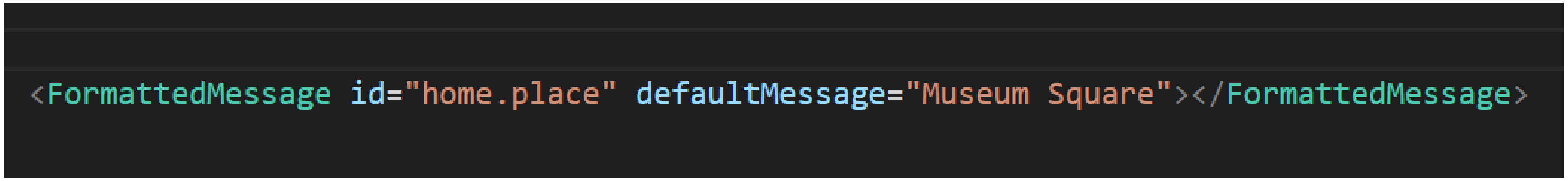
The big moment

Nothing has really happened. But make sure the console is empty, it verifies we set it up correctly!

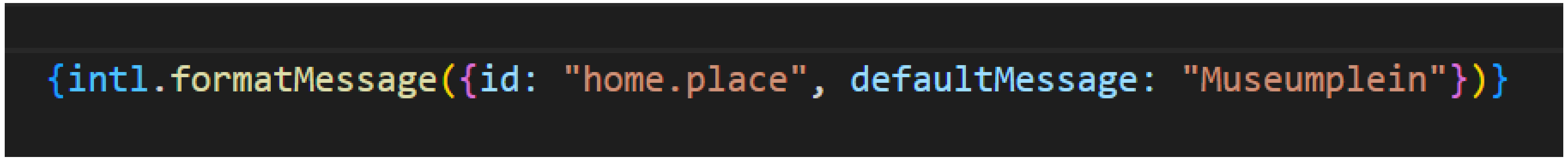
Translating text

Imperative v.s. Component API

React-intl offers two ways to engage with its API, the imperative versus the component API. Let’s look at translating the “home.place” placeholder.



```
<FormattedMessage id="home.place" defaultMessage="Museum Square"></FormattedMessage>
```



```
{intl.formatMessage({id: "home.place", defaultMessage: "Museumplein"})}
```

Imperative v.s. Component API

	Component API	**Imperative API**
Advantages	• Uses React components • Rendered directly in HTML • Easy to style	• Can be used anywhere • Technically speaking render faster
Disadvantages	• Cannot always be used in attributes of HTML elements	• Are separated from the HTML

Museum plein

WALLOPING VIOLIN ORCHESTRA

MAY 06 - 10.2023

On tour: yes

Tickets available: in 4 months

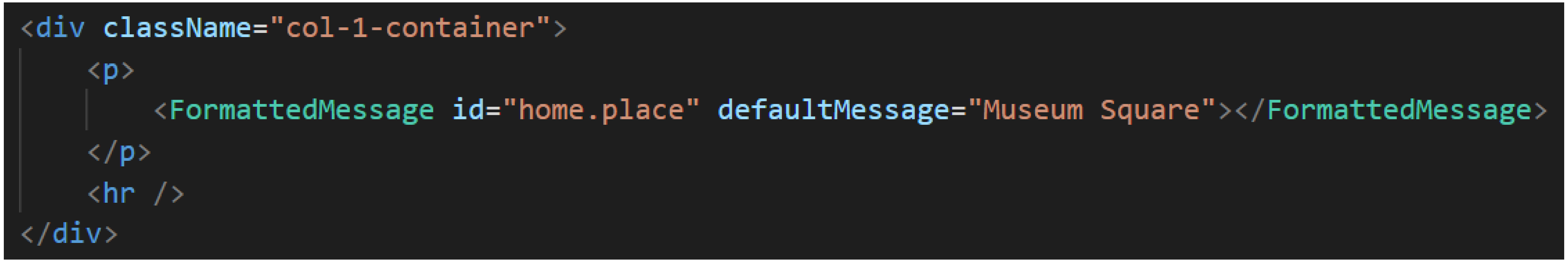
32.000 tickets left

Price

€ 20,00

Simple text translation

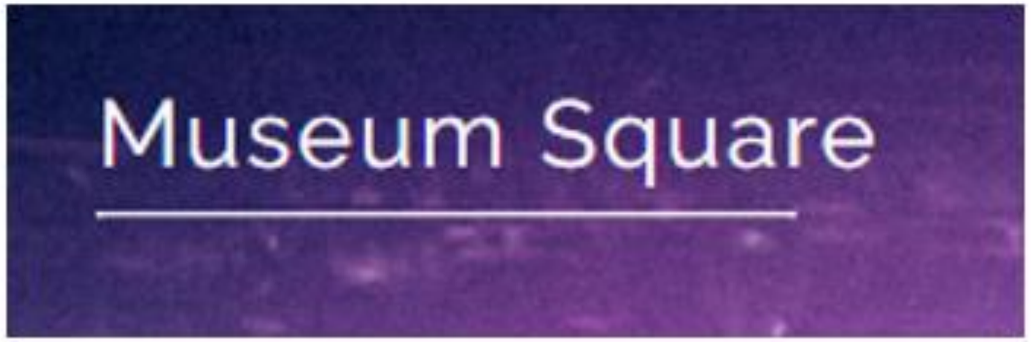
1. Open “App.tsx”
2. Import { FormattedMessase} from “react-intl”
3. Update the code as below

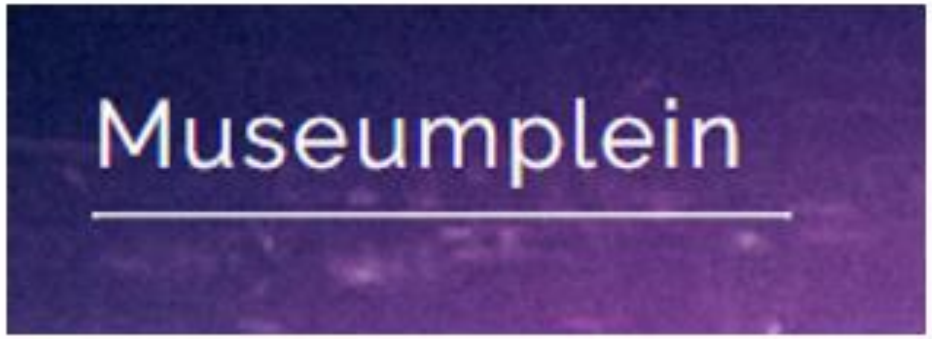


```
<div className="col-1-container">
    <p>
        <FormattedMessage id="home.place" defaultMessage="Museum Square"></FormattedMessage>
    </p>
    <hr />
</div>
```

Simple text translation

If everything went correctly, the webpage should now render:



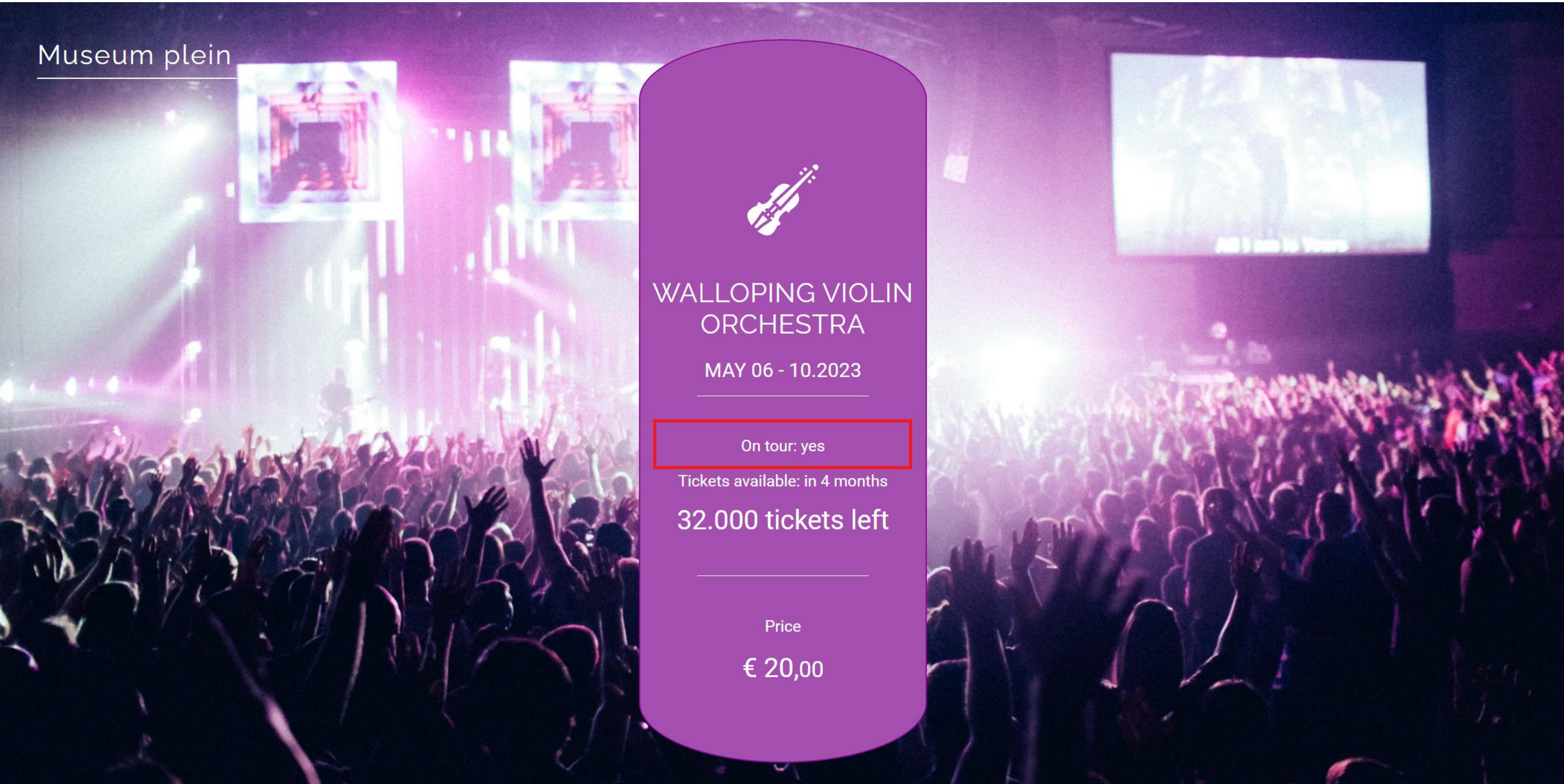


DefaultMessage

If react-intl cannot find the placeholder within the translation files, it will use the fallback message. Therefore, it is important to make sure all defaultMessages are written in the fallback language you want to provide.

Dynamic formattedMessage

Sometimes you want to be able to construct a message that has a sentence with a dynamic value. Luckily for us, React-intl makes use of the ICU Message syntax, which empowers us to do a whole range of formatting directly within the <FormattedMessage> component syntax



Museum plein

WALLOPING VIOLIN ORCHESTRA

MAY 06 - 10.2023

On tour: yes

Tickets available: in 4 months

32.000 tickets left

Price

€ 20,00

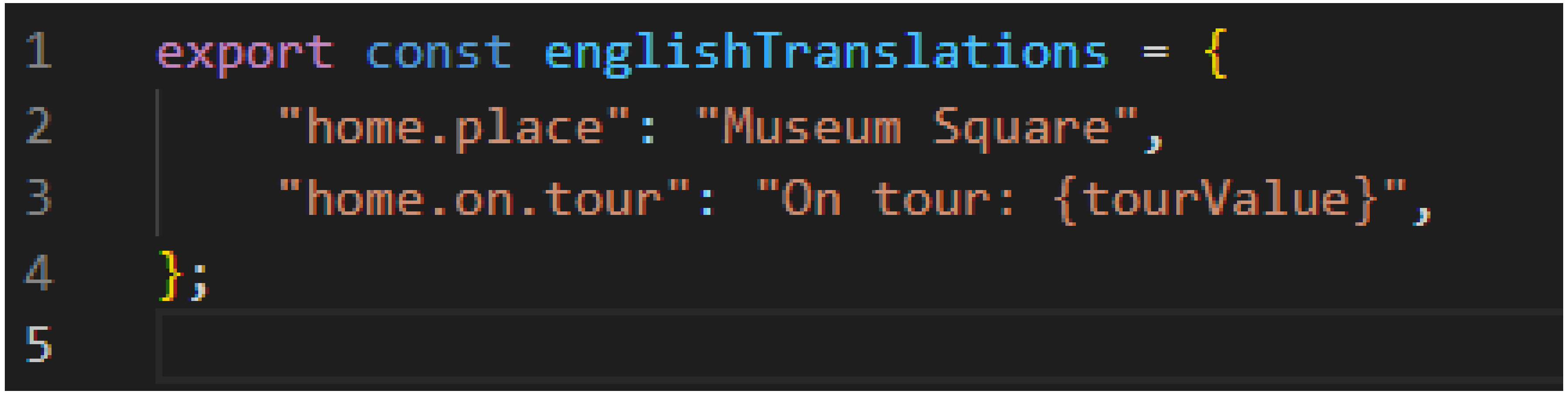
Dynamic formattedMessage

I will do the following:

1. The “On tour” message will be replaced with “On Tour: yes”
2. I will add a placeholder for the dynamic value in the translation file.
3. I will use a state hook to record the dynamic value and pass it into the formattedMessage

Dynamic formattedMessage

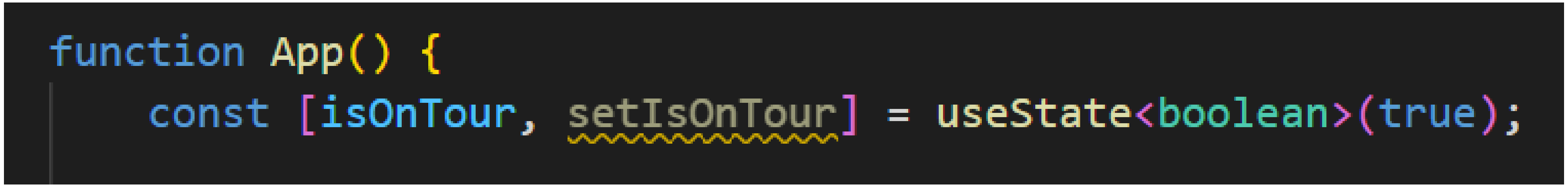
Open your translation files and add the “home.on.tour” placeholder as added below:



```
export const englishTranslations = {
    "home.place": "Museum Square",
    "home.on.tour": "On tour: {tourValue}",
};
```

Dynamic formattedMessage

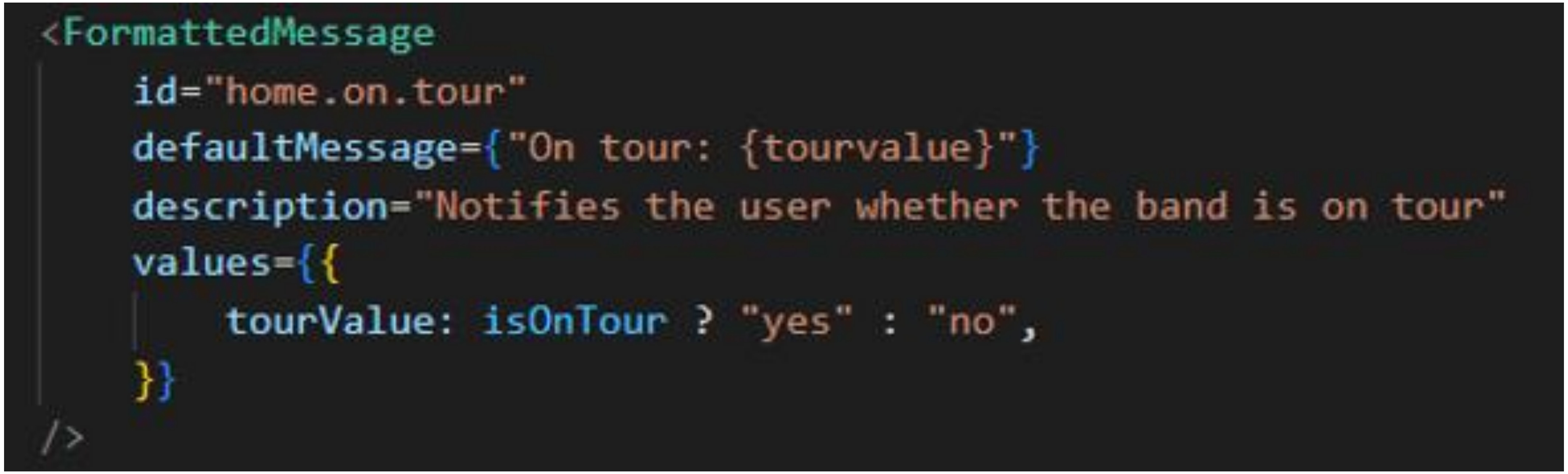
Open “App.tsx” add a state hook that will remember the value for the status of the band. Remember to import “useState” from “React”



```
function App() {
    const [isOnTour, setIsOnTour] = useState<boolean>(true);
```

Dynamic formattedMessage

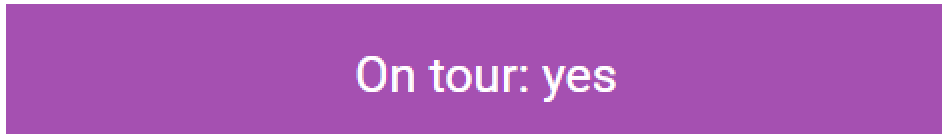
Within the FormattedMessage we will use the Values attribute which is an object where we can set one or multiple values for our placeholders.

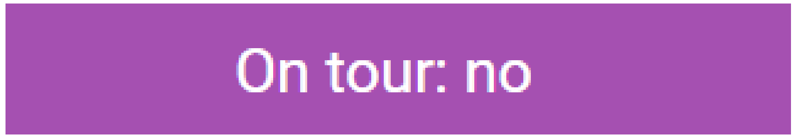


```
<FormattedMessage
    id="home.on.tour"
    defaultMessage={"On tour: {tourvalue}"}
    description="Notifies the user whether the band is on tour"
    values={{
        tourValue: isOnTour ? "yes" : "no",
    }}
/>
```

Dynamic formattedMessage

If everything went correctly you should now be able to see:


On tour: yes


On tour: no

Did anyone spot a problem with my approach?

Dynamic XML formattedMessage

The method above works well in many cases, but we still have untranslated text.

We will solve this by using an XML tag in our message; that way, we can use HTML inside of our value of the <FormattedMessage />.

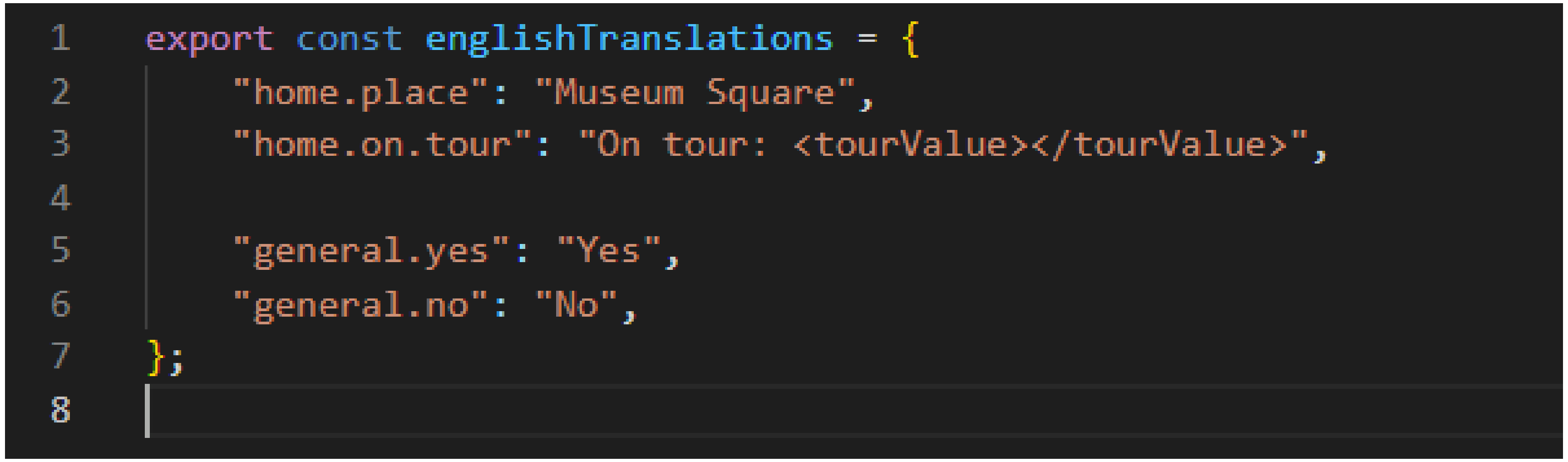
Dynamic XML formattedMessage

I will do the following:

1. Replace the placeholder to an XML placeholder.
2. Add the “yes” and “no” texts to the translation file.
3. Add 2 CSS styles, red and green text.
4. Replace the value of <FormattedMessage /> to JSX.

Dynamic XML formattedMessage

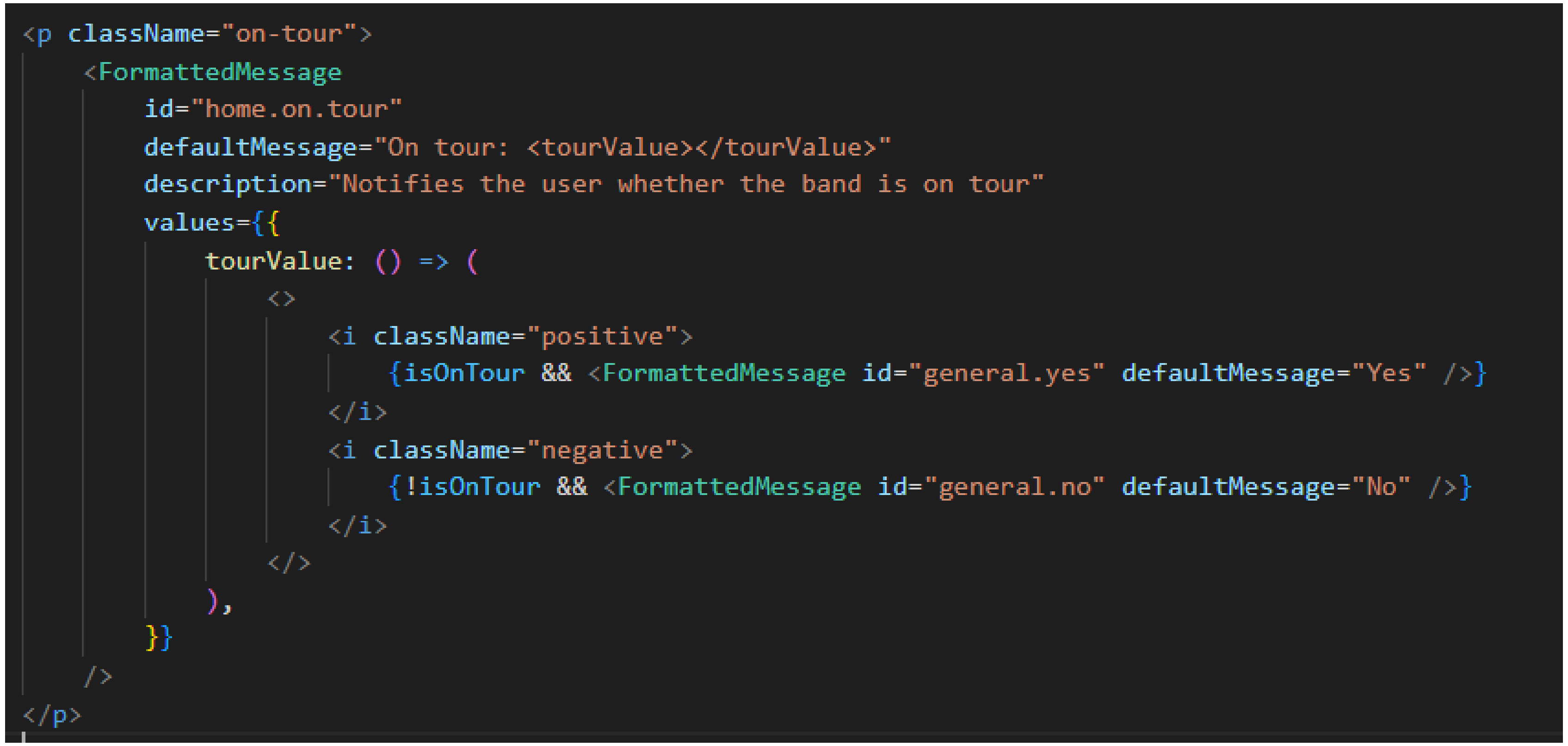
Let’s open The translation files and add two rules for “Yes” and “No”. Also, make sure to add the home.on.tour placeholder to use XML instead of a regular placeholder.



```
export const englishTranslations = {
    "home.place": "Museum Square",
    "home.on.tour": "On tour: <tourValue></tourValue>",

    "general.yes": "Yes",
    "general.no": "No",
};

```

Dynamic XML formattedMessage

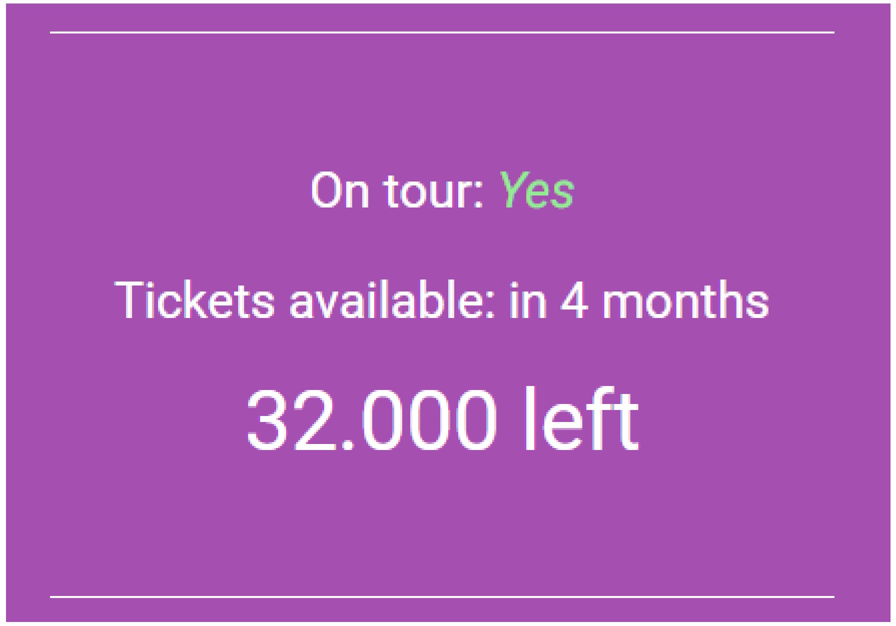
Next, change <FormattedMessage /> value within “App.tsx” like below:



```
<p className="on-tour">
    <FormattedMessage
        id="home.on.tour"
        defaultMessage="On tour: <tourValue></tourValue>"
        description="Notifies the user whether the band is on tour"
        values={{
            tourValue: () => (
                <>
                    <i className="positive">
                        {isOnTour && <FormattedMessage id="general.yes" defaultMessage="Yes" />}
                    </i>
                    <i className="negative">
                        {!isOnTour && <FormattedMessage id="general.no" defaultMessage="No" />}
                    </i>
                </>
            ),
        }}
    />
</p>
```

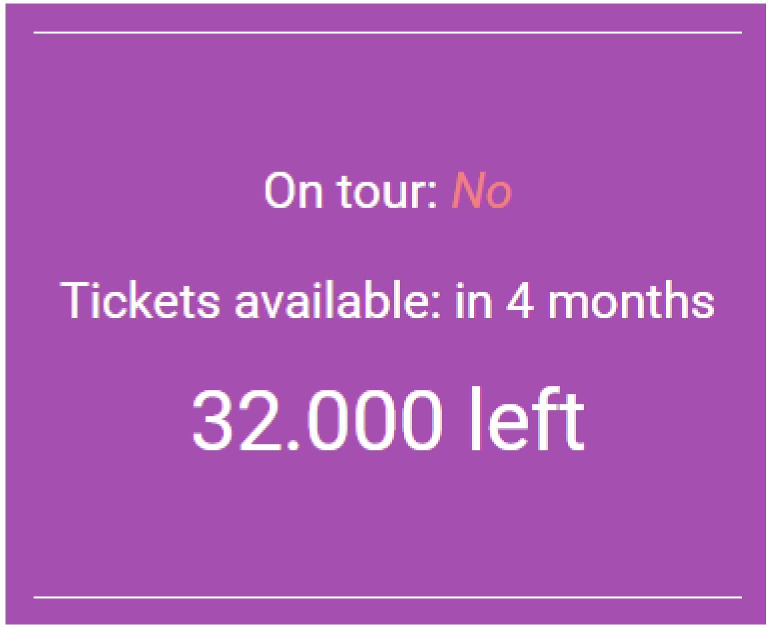
Dynamic XML formattedMessage

When your browser now renders your application, you should now see the following:


On tour: *Yes*

Tickets available: in 4 months

32.000 left


On tour: *No*

Tickets available: in 4 months

32.000 left

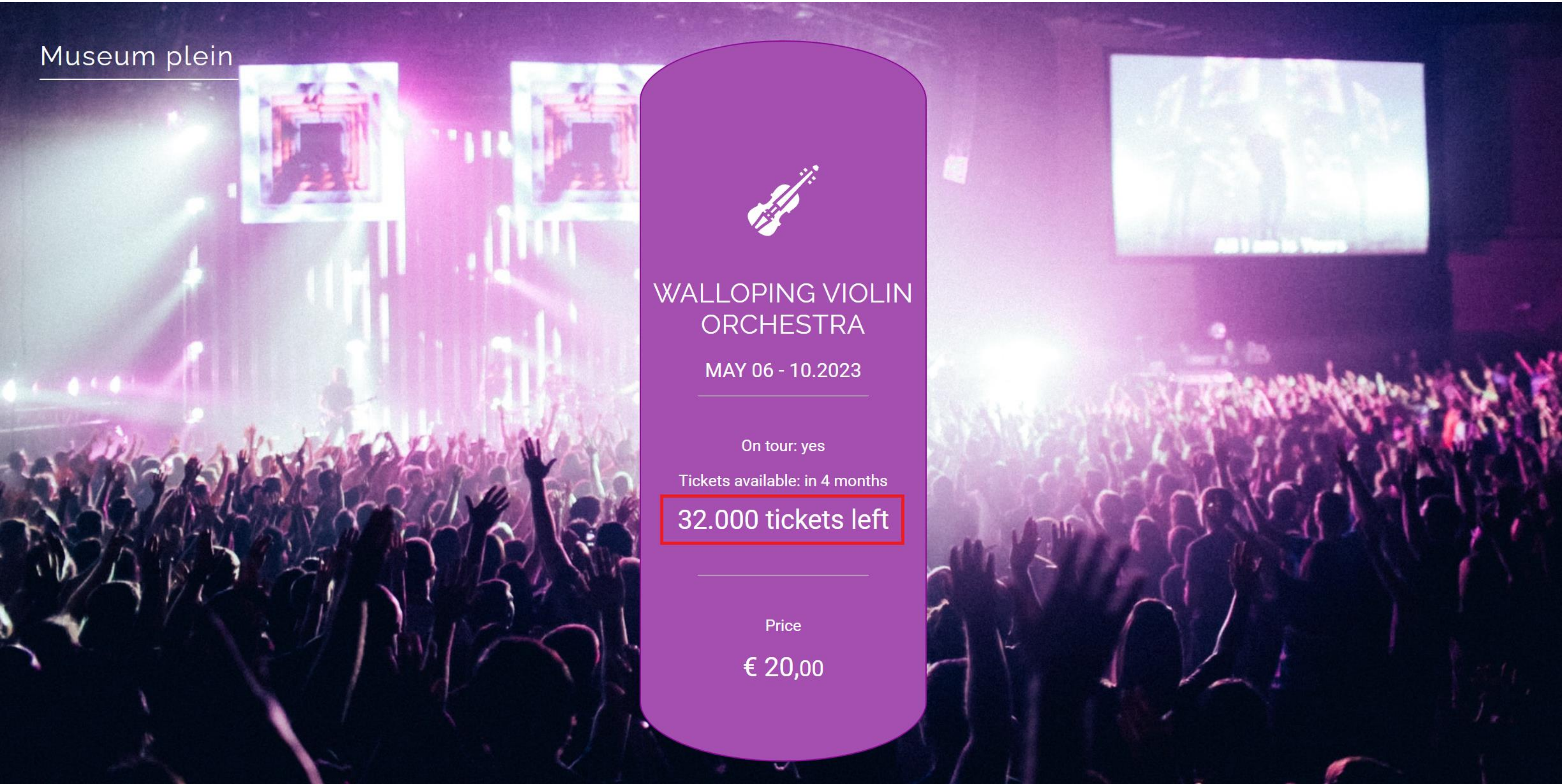
The benefits

As you can see above the value of “On tour” is now being translated and styled at the same time without having to do difficult compositions involving multiple <FormattedMessage> elements in the HTML. It’s one JSX element that helps keep us developers sane

Pluralization

Sometimes there is a need to change text depending on the quantifier. For example, an e-mail client might want to notify the user like this:

- There is 1 new email.
- There are 12 new emails.



Museum plein

WALLOPING VIOLIN ORCHESTRA

MAY 06 - 10.2023

On tour: yes

Tickets available: in 4 months

32.000 tickets left

Price

€ 20,00

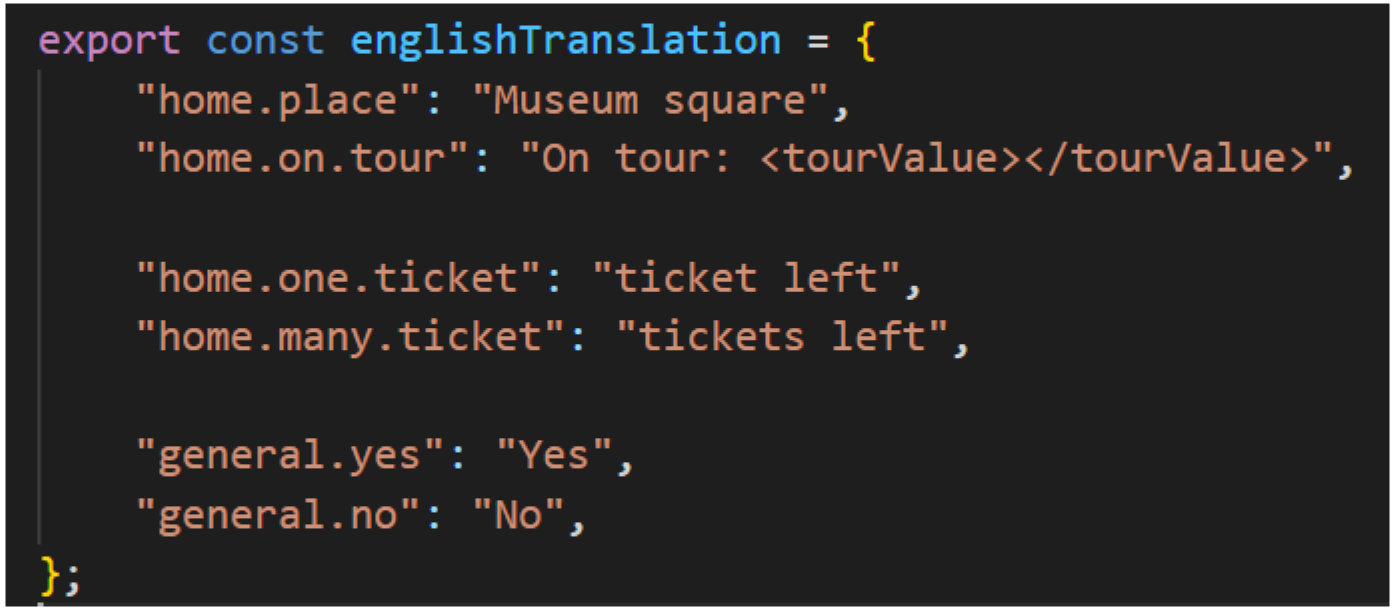
Pluralization

Based on the number of tickets left I will render whether tickets be singular or plural. I will need to do the following steps:

1. Add new translations for “ticket” and “tickets”
2. Import { FormattedPlural} from “react-intl”
3. Change the code in the “tickets left” code block

Pluralization

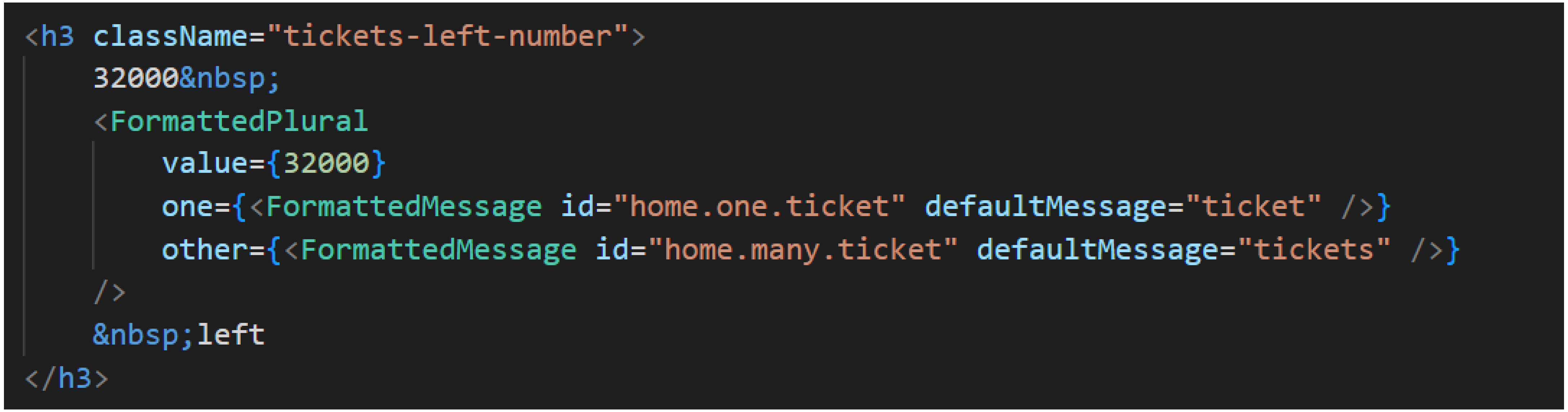
Firstly add the translation messages to all your language resource files.



```
export const englishTranslation = {
    "home.place": "Museum square",
    "home.on.tour": "On tour: <tourValue></tourValue>",

    "home.one.ticket": "ticket left",
    "home.many.ticket": "tickets left",

    "general.yes": "Yes",
    "general.no": "No",
};
```

Pluralization

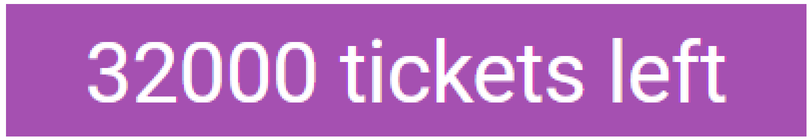
Next add the import to the top and change the code as below:

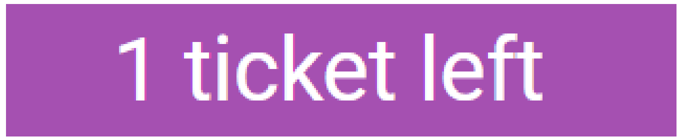


```
<h3 className="tickets-left-number">
    32000&nbsp;
    <FormattedPlural
        value={32000}
        one={<FormattedMessage id="home.one.ticket" defaultMessage="ticket" />}
        other={<FormattedMessage id="home.many.ticket" defaultMessage="tickets" />}
    />
    &nbsp;left
</h3>
```

Pluralization

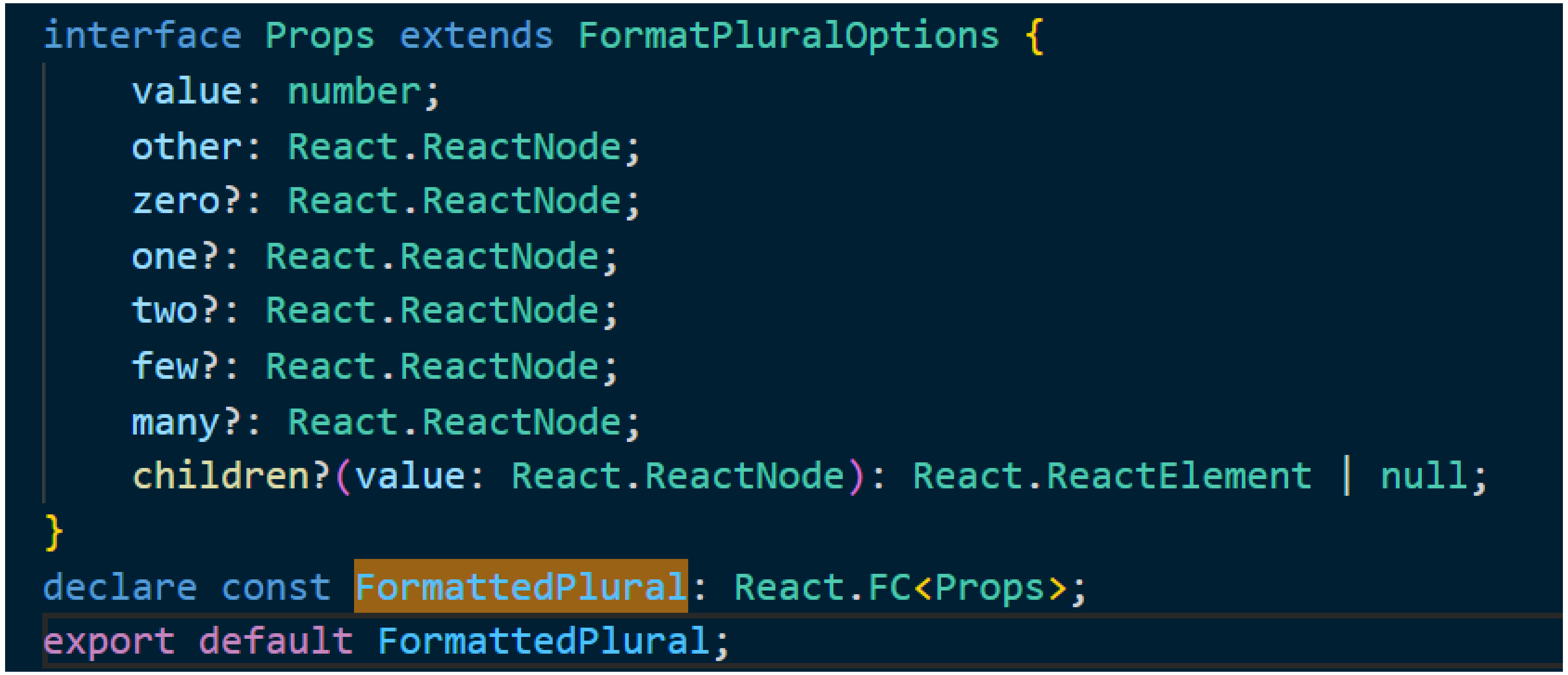
Depending on the variable, the application now shows:


32000 tickets left


1 ticket left

Pluralization

The following attributes are available on <FormattedPlural />.



```
interface Props extends FormatPluralOptions {
    value: number;
    other: React.ReactNode;
    zero?: React.ReactNode;
    one?: React.ReactNode;
    two?: React.ReactNode;
    few?: React.ReactNode;
    many?: React.ReactNode;
    children?(value: React.ReactNode): React.ReactElement | null;
}
declare const FormattedPlural: React.FC<Props>;
export default FormattedPlural;
```

Translating date and time

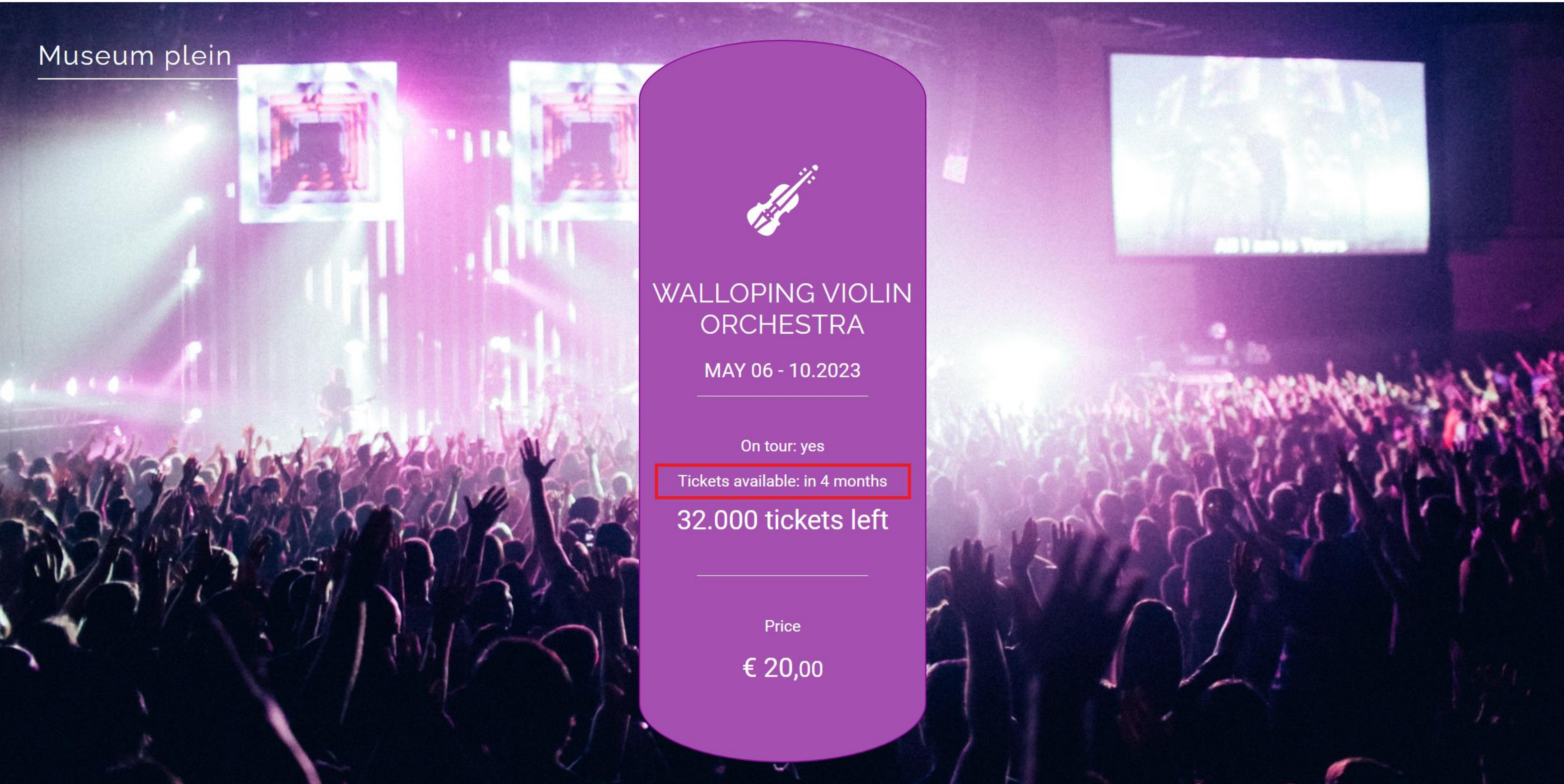
Dates and time

Whoever has done internationalization in the past knows that displaying dates and time is one of the most challenging things to do. Depending on where you live you might write the day, month, or year first. This makes it cumbersome for (software) developers to cater to different cultures

Dates and time

3 common ways to write dates:

Country	Format
DD-MM-YYYY	Most countries in the world
MM-DD-YYYY	USA, Canada, Belize
YYYY-MM-DD	East Asia and Iran



Museum plein

WALLOPING VIOLIN ORCHESTRA

MAY 06 - 10.2023

On tour: yes

Tickets available: in 4 months

32.000 tickets left

Price

€ 20,00

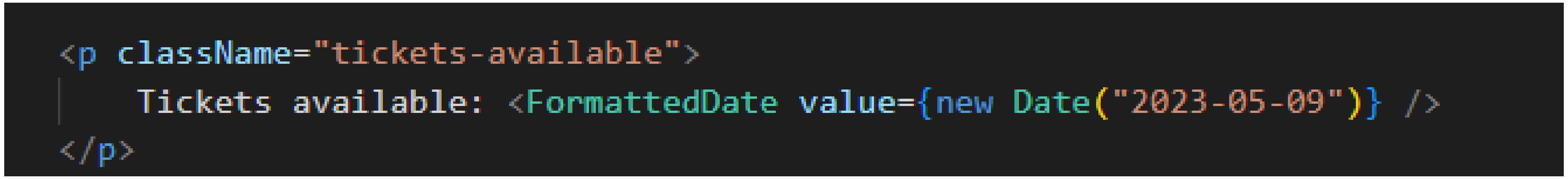
Simple date

Let’s look at the simplest form of a date in action. I will do the following:

1. Make sure the IntlProvider has a locale with country code set up.
2. Import the <FormattedDate /> from “react-intl”
3. Create a new date to format

Simple date

If you want to see different outcomes from the locale you can make use of the following country codes: “en-us”, “nl”, “zh-cn”



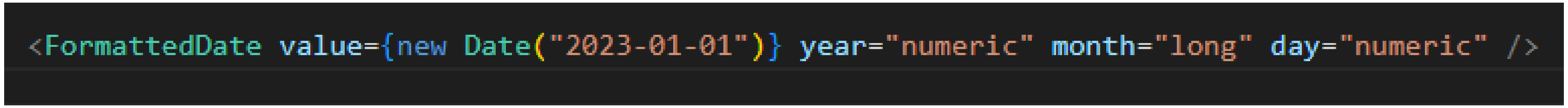
```
<p className="tickets-available">
    Tickets available: <FormattedDate value={new Date("2023-05-09")} />
</p>
```

Simple date

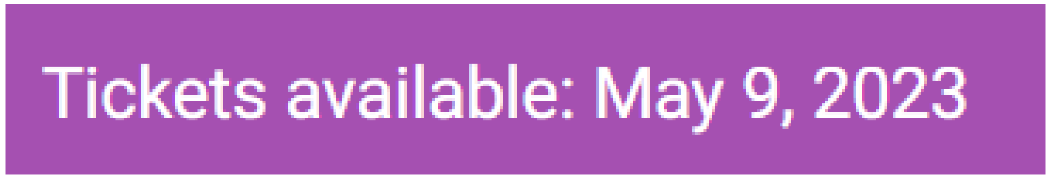
Of course, this way of date representation is not always what we would like, sometimes you would rather write the date out to avoid any confusion among your readers. For the next example let’s write a date, in which we can control the separate parts. E.g: “1 January 2023”

Simple date

Let’s add a few props to the <FormattedDate /> component.

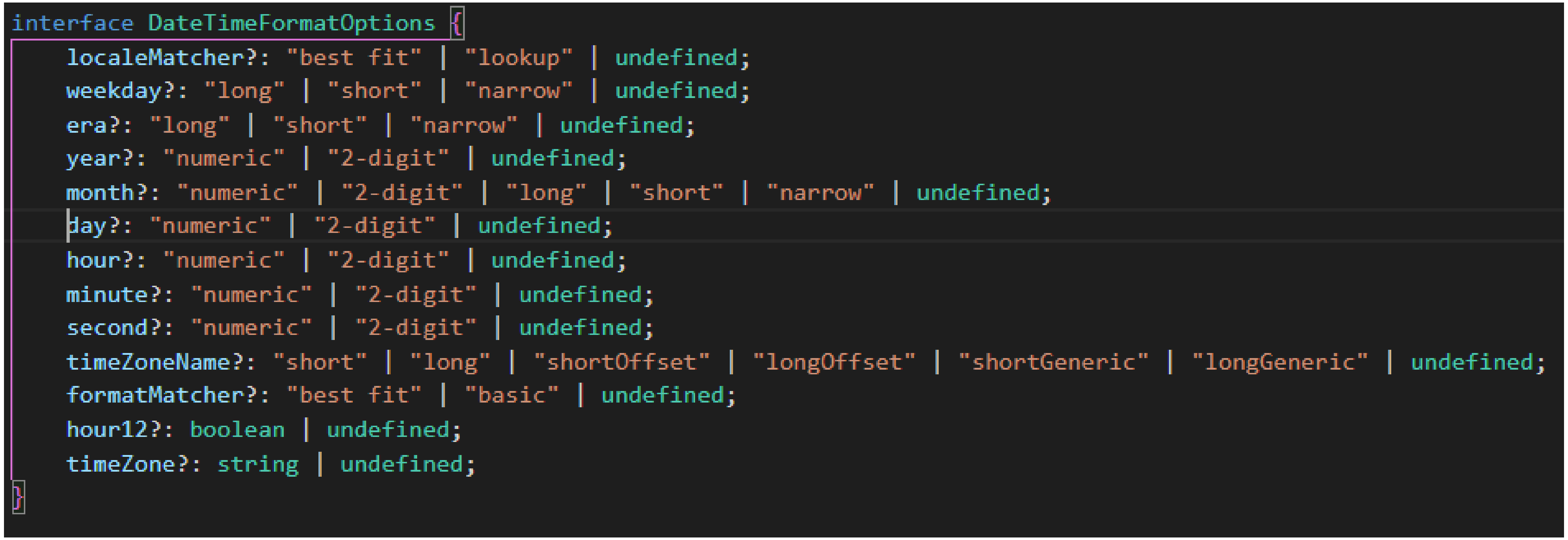


```
<FormattedDate value={new Date("2023-01-01")} year="numeric" month="long" day="numeric" />
```



Tickets available: May 9, 2023

FormattedDate interface



```
interface DateTimeFormatOptions {
    localeMatcher?: "best fit" | "lookup" | undefined;
    weekday?: "long" | "short" | "narrow" | undefined;
    era?: "long" | "short" | "narrow" | undefined;
    year?: "numeric" | "2-digit" | undefined;
    month?: "numeric" | "2-digit" | "long" | "short" | "narrow" | undefined;
    day?: "numeric" | "2-digit" | undefined;
    hour?: "numeric" | "2-digit" | undefined;
    minute?: "numeric" | "2-digit" | undefined;
    second?: "numeric" | "2-digit" | undefined;
    timeZoneName?: "short" | "long" | "shortOffset" | "longOffset" | "shortGeneric" | "longGeneric" | undefined;
    formatMatcher?: "best fit" | "basic" | undefined;
    hour12?: boolean | undefined;
    timeZone?: string | undefined;
}
```

Relative time

Sometimes when reading a blog with replies or a forum where people post a reply, you see a format: “2 hours ago”. React-intl has the capabilities to render relative times just as easily. We can use the <FormatRelativeTime /> component.

Relative time

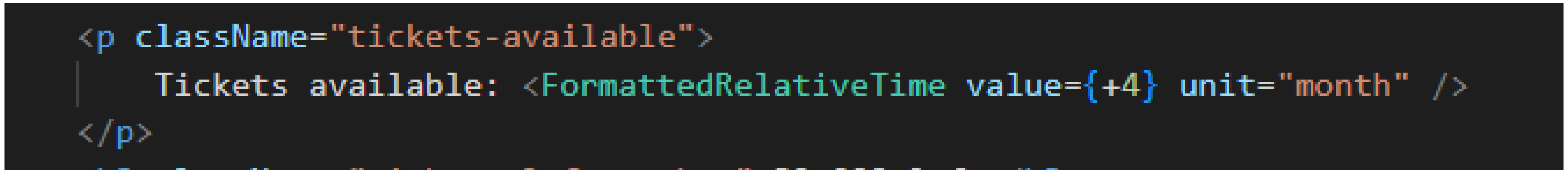
It takes 2 values to enable relative time formatting:

1. A unit of time to subtract the current date. Hours, minutes, seconds numbers

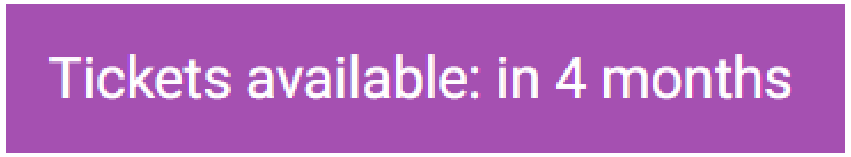
2. A unit/format to represent the date.

Relative time

Open “App.tsx” and let’s change the text after: “Tickets available:”



```
<p className="tickets-available">
    Tickets available: <FormattedRelativeTime value={+4} unit="month" />
</p>
```



Tickets available: in 4 months

Relative time

If you work with a shorter time span, you could apply automatic updating of the value determined by seconds. Add the “updateIntervalInSeconds” attribute to the <FormattedRelativeTime /> element. It will only update till the next unit.

* Please refer to the documentation for extensive knowledge of how it works.

Custom style date format

Finally, let’s look at a way to format a date in a custom style. Just as with <FormattedMessage /> we can also access the styling of the date with the component <FormattedDateParts />. For this example, we would like the first part of the Date to be bold, and the second part to be in italics.

No, Don’t ask me why.

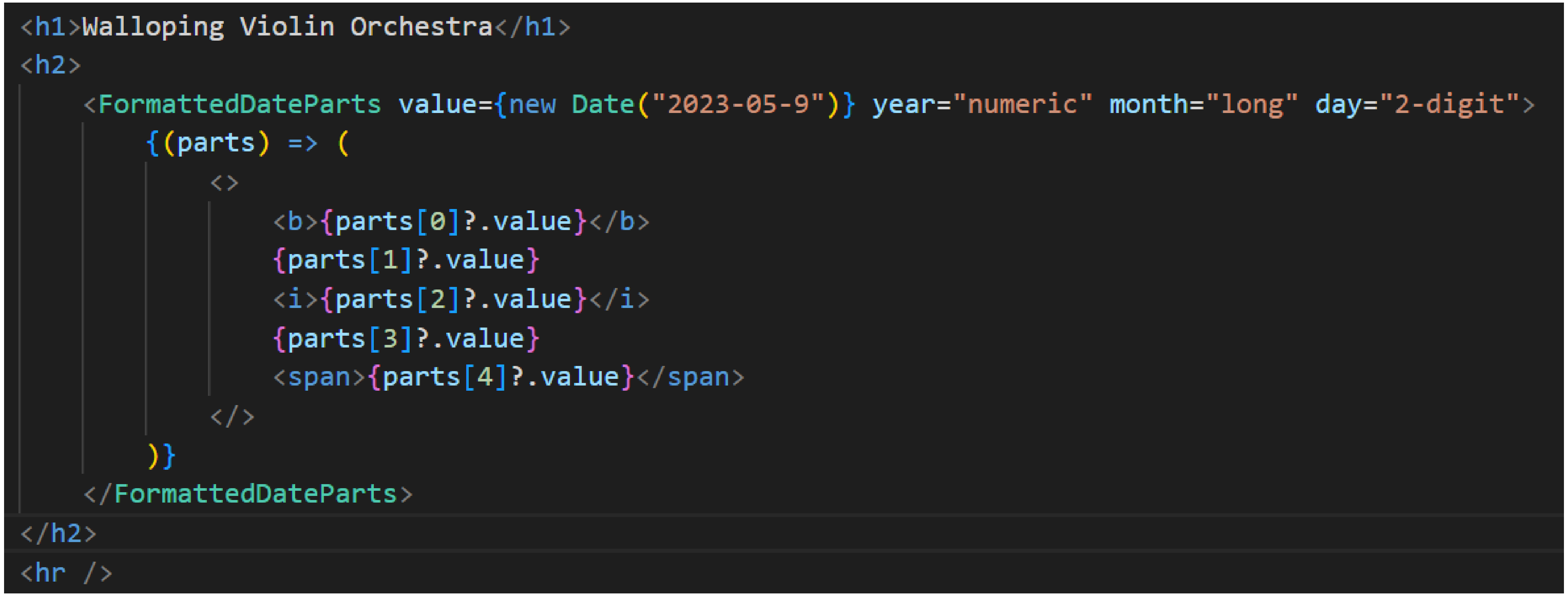
Custom style date format

For the most part, <FormattedDateParts /> works similarly to <FormattedDate />

but it takes a children function which allows us to use any JSX code.

Custom style date format

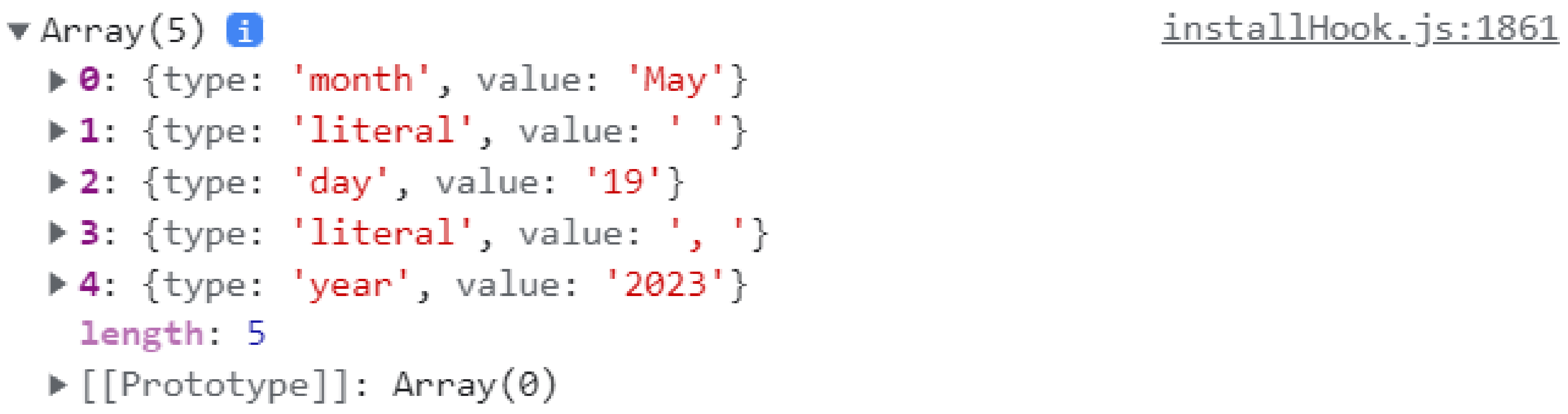
Let’s update our code like below:



```
<h1>Walloping Violin Orchestra</h1>
<h2>
    <FormattedDateParts value={new Date("2023-05-9")} year="numeric" month="long" day="2-digit">
        {(parts) => (
            <>
                <b>{parts[0]?.value}</b>
                {parts[1]?.value}
                <i>{parts[2]?.value}</i>
                {parts[3]?.value}
                <span>{parts[4]?.value}</span>
            </>
        )}
    </FormattedDateParts>
</h2>
<hr />
```

Custom style date format

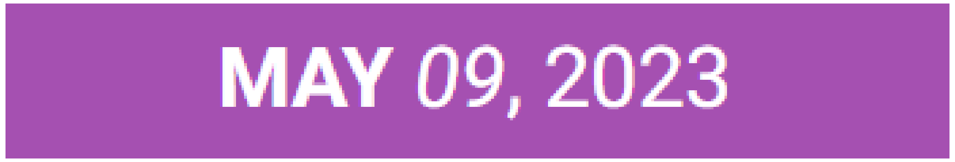
The previous slide's code had something strange to it. I wanted to print 3 parts of a date but rendered 5 to the screen. The catch here: React-intl also adds in the separators as a separate value. If you log the parts array, you will see the following:



```
▼ Array(5) i                                              installHook.js:1861
  ▶ 0: {type: 'month', value: 'May'}
  ▶ 1: {type: 'literal', value: ' '}
  ▶ 2: {type: 'day', value: '19'}
  ▶ 3: {type: 'literal', value: ', '}
  ▶ 4: {type: 'year', value: '2023'}
    length: 5
  ▶ [[Prototype]]: Array(0)
```

Custom style date format

As we can see above for English there is a comma between the day and the year, therefore we must always render all the parts to avoid confusion for our audience.

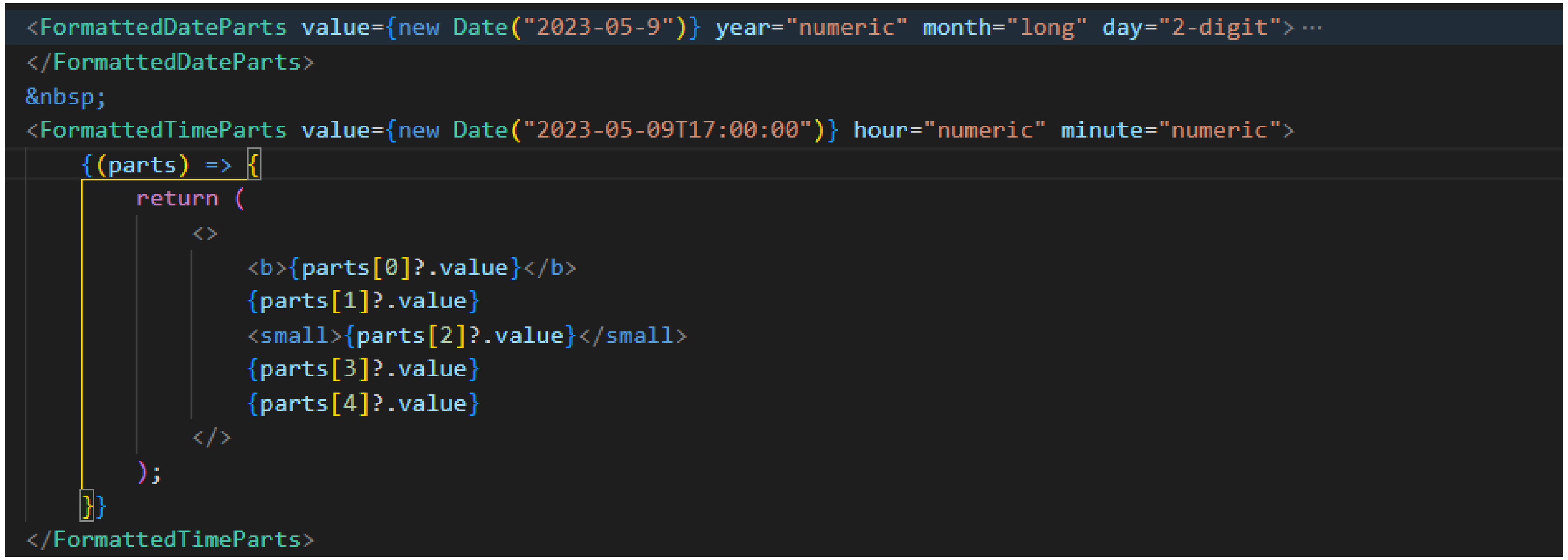


MAY *09*, 2023

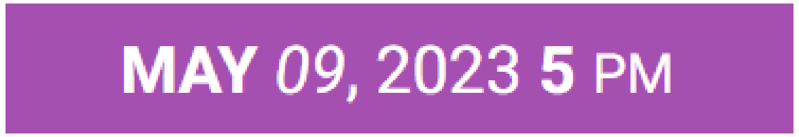
Custom style datetime format

Besides formatting dates, we can also format times with React-intl. It works almost the same as the <formattedDate /> component, therefor I will only show a complex example with custom styling to the time.

Custom style datetime format

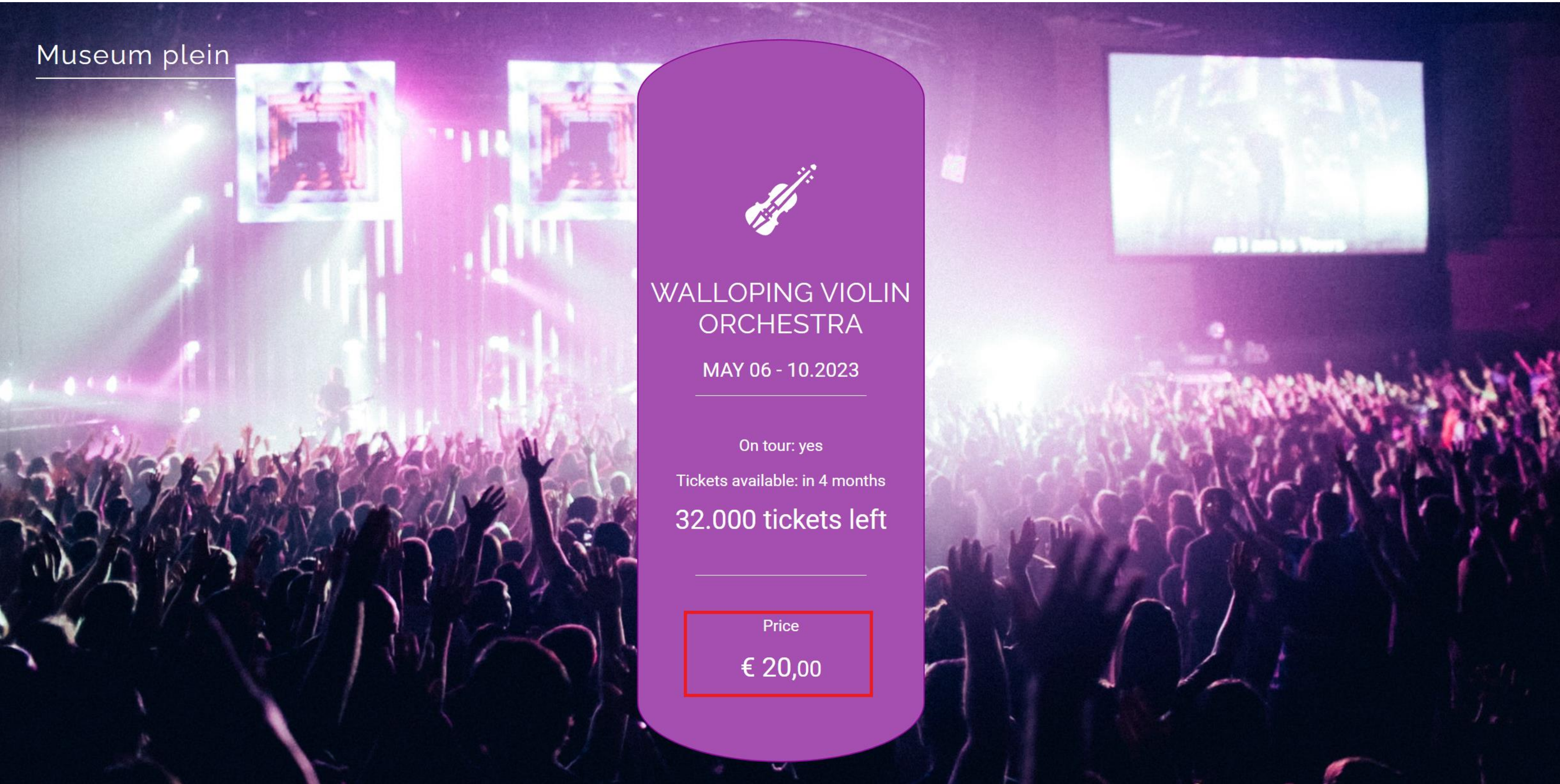


```
<FormattedDateParts value={new Date("2023-05-9")} year="numeric" month="long" day="2-digit">…
</FormattedDateParts>
&nbsp;
<FormattedTimeParts value={new Date("2023-05-09T17:00:00")} hour="numeric" minute="numeric">
    {(parts) => {
        return (
            <>
                <b>{parts[0]?.value}</b>
                {parts[1]?.value}
                <small>{parts[2]?.value}</small>
                {parts[3]?.value}
                {parts[4]?.value}
            </>
        );
    }}
</FormattedTimeParts>
```



MAY *09*, 2023 **5** PM

Translating numbers



Museum plein

WALLOPING VIOLIN ORCHESTRA

MAY 06 - 10.2023

On tour: yes

Tickets available: in 4 months

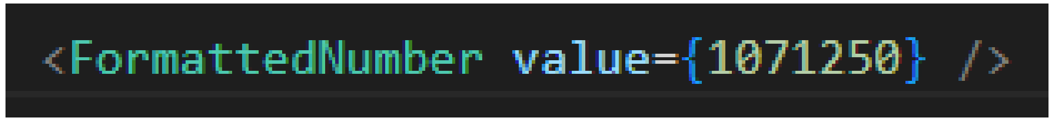
32.000 tickets left

Price

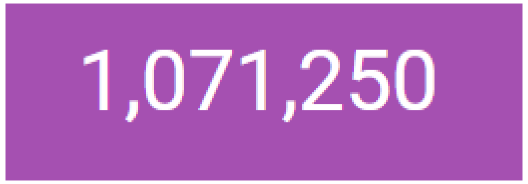
€ 20,00

Simple number formatting

Numbers are everywhere in our applications, be it prices, quantities, or just parts of strings in general. What makes it difficult is the difference in Radix characters and monetary values. Let’s first have a look at the most basic implementation.



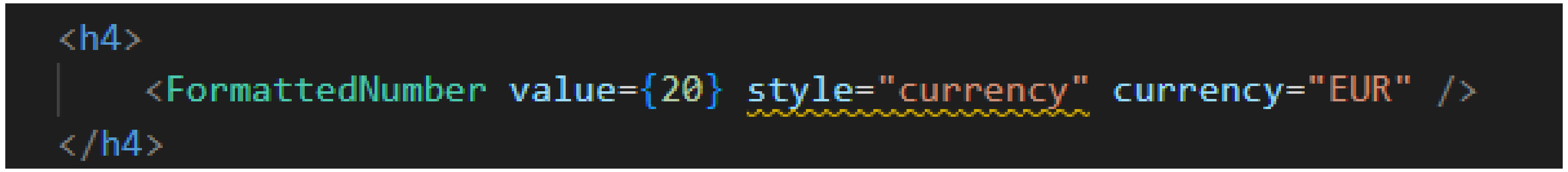
```
<FormattedNumber value={1071250} />
```



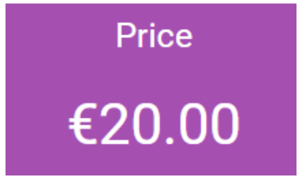
```
1,071,250
```

Currency formatting

React-intl provide currency formatting out of the box. It will correctly place the monetary sign and all radix characters.



```
<h4>
    <FormattedNumber value={20} style="currency" currency="EUR" />
</h4>
```

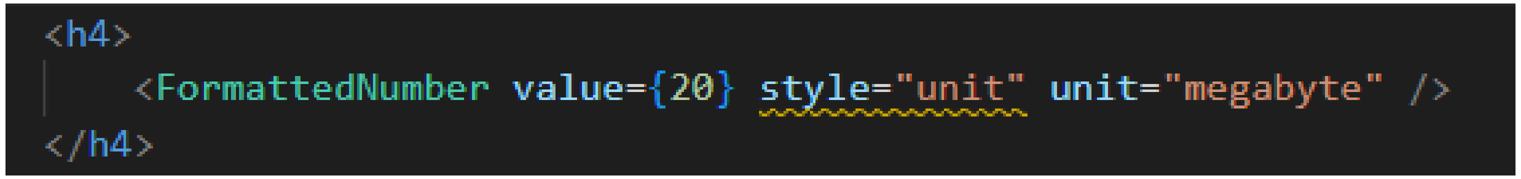


Price

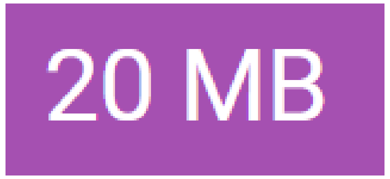
€20.00

Sanctioned unit formatting

In the real world number implementation can be more complex. The <FormattedNumber /> accepts a unit attribute for the most common sanctioned units on earth.



```
<h4>
    <FormattedNumber value={20} style="unit" unit="megabyte" />
</h4>
```



20 MB

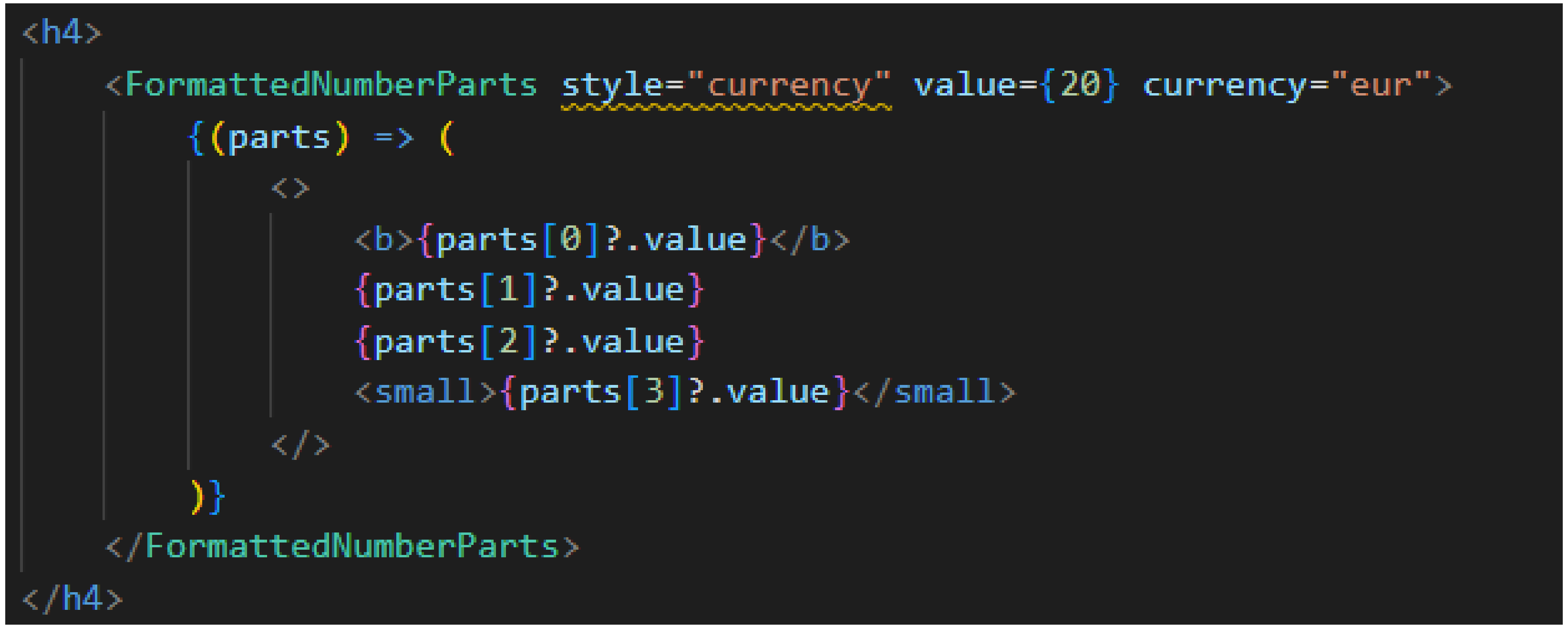
* For a full list refer to the documentation:

https://formatjs.io/docs/polyfills/intl-numberformat/#SupportedUnits

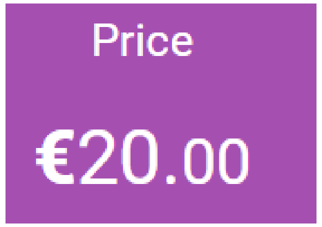
Currency formatting

Of course, <FormattedNumberParts /> is also available for numbers. It again takes a children function which allows us to use any JSX code. I will use it to render the part after the fractional part in a smaller font.

Custom currency formatting



```
<h4>
    <FormattedNumberParts style="currency" value={20} currency="eur">
        {(parts) => (
            <>
                <b>{parts[0]?.value}</b>
                {parts[1]?.value}
                {parts[2]?.value}
                <small>{parts[3]?.value}</small>
            </>
        )}
    </FormattedNumberParts>
</h4>
```



Custom currency formatting

These are the options that React-intl provides us with in terms of numbers. It’s possible to format almost any common number format with the sanctioned units, though from experience I can tell you, oftentimes you will spend some time in the documentation to find out the exact way how to render it.

Unit testing react-intl

Unit tests

To have confidence in our codebase, it’s always a good idea to unit test as much as possible. With React-intl we have the possibility to test our translations hassle-free.

Unit tests approach

There are 3 steps that we need to take.

1. Include the polyfills to use <FormattedDate and <FormattedNumber />

2. Create a wrapper function in which we can set the expected locale

3. Write some solid unit tests.

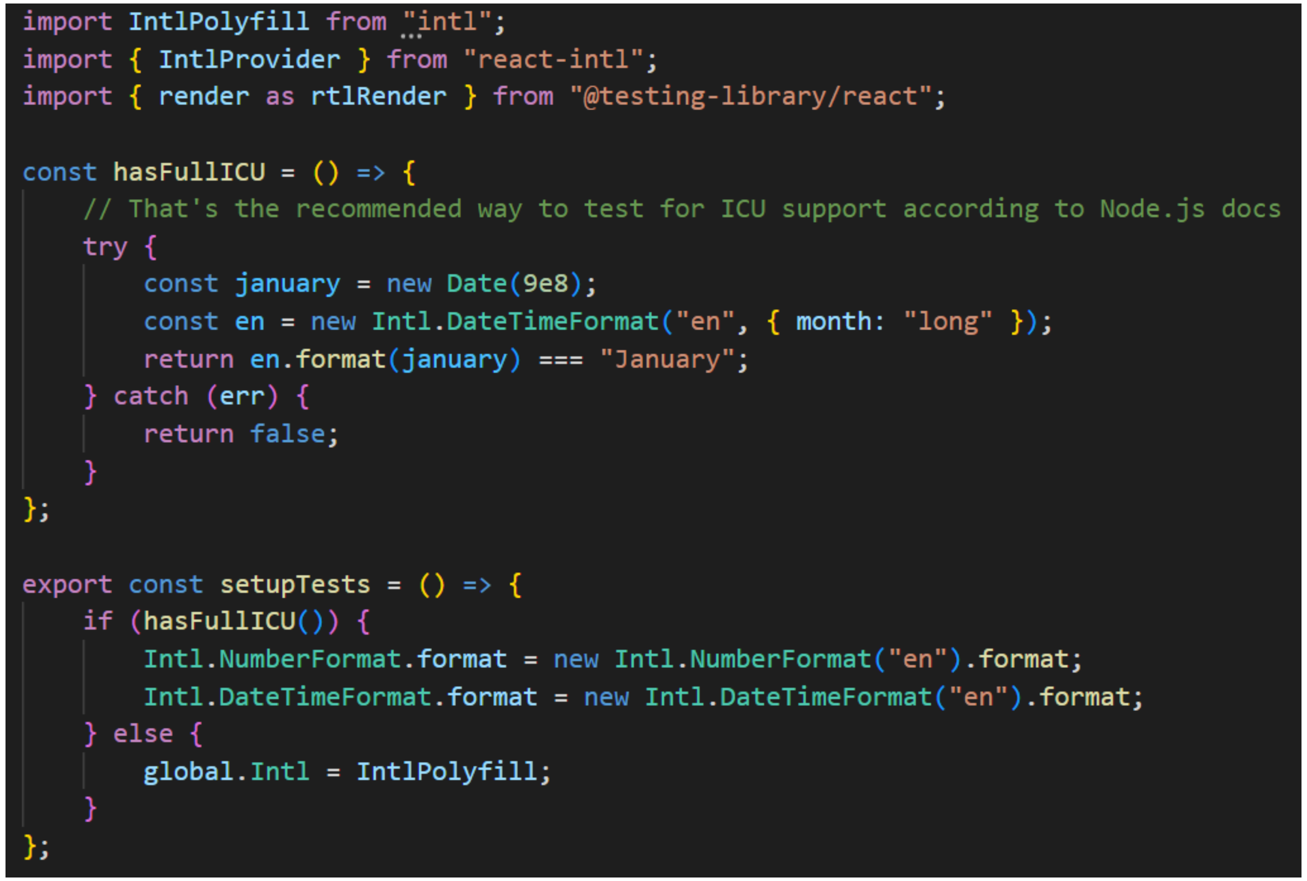
Unit tests polyfills

From Node v13 onwards we now have full support of ICU, the underlying system to describe Unicode translations. Let’s write a helper function to see if it’s available.

1. Run “npm install –D intl”

Unit tests polyfills

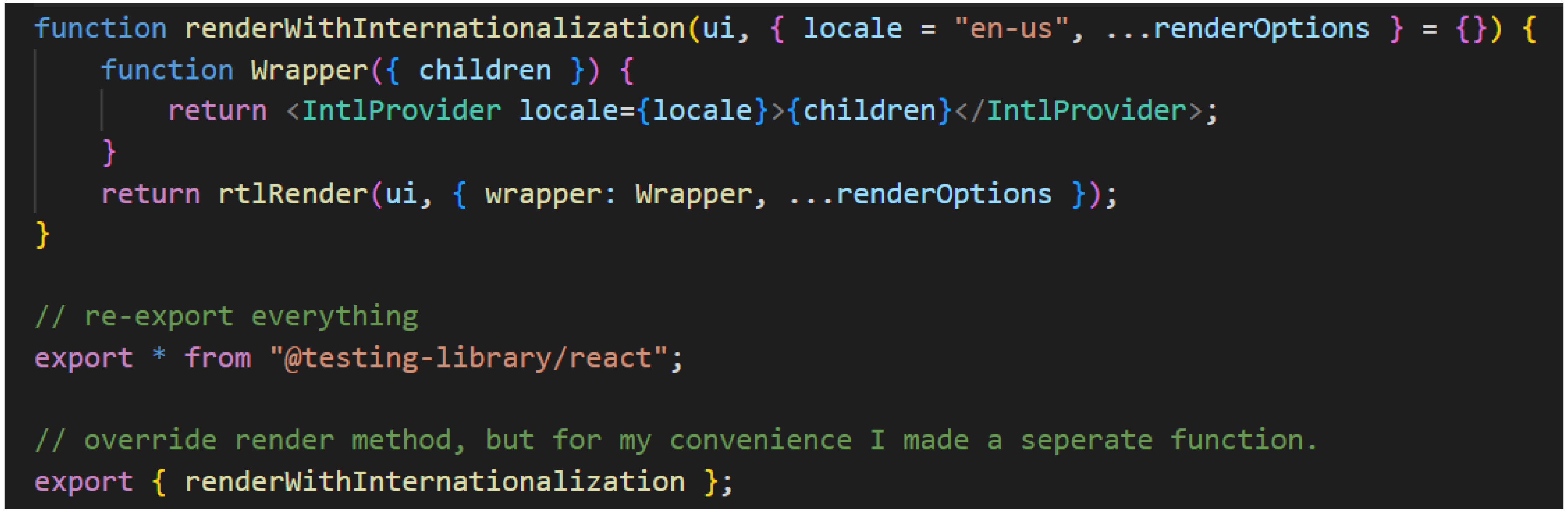
/src/test-utils.js



```
import IntlPolyfill from "intl";
import { IntlProvider } from "react-intl";
import { render as rtlRender } from "@testing-library/react";

const hasFullICU = () => {
    // That's the recommended way to test for ICU support according to Node.js docs
    try {
        const january = new Date(9e8);
        const en = new Intl.DateTimeFormat("en", { month: "long" });
        return en.format(january) === "January";
    } catch (err) {
        return false;
    }
};

export const setupTests = () => {
    if (hasFullICU()) {
        Intl.NumberFormat.format = new Intl.NumberFormat("en").format;
        Intl.DateTimeFormat.format = new Intl.DateTimeFormat("en").format;
    } else {
        global.Intl = IntlPolyfill;
    }
};
```

Unit tests polyfills

Next I will need a custom render function to wrap our components inside an "intlProvider" wrapper in which we can set the locale. (Note: omitted messages).



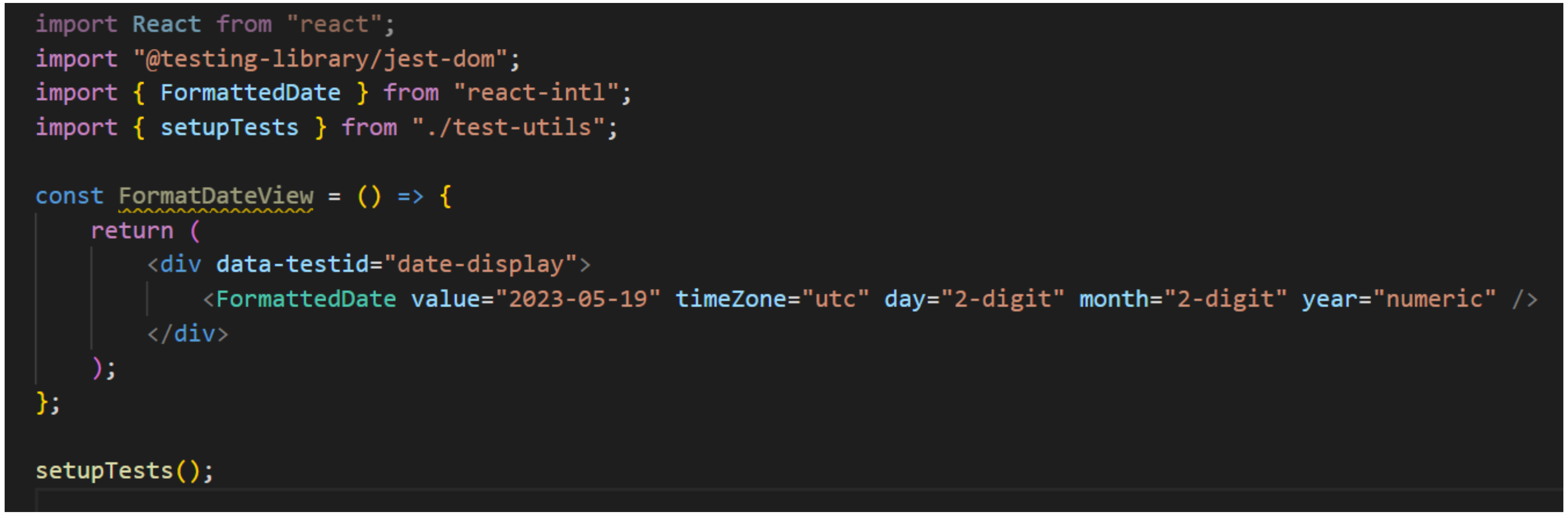
```
function renderWithInternationalization(ui, { locale = "en-us", ...renderOptions } = {}) {
    function Wrapper({ children }) {
        return <IntlProvider locale={locale}>{children}</IntlProvider>;
    }
    return rtlRender(ui, { wrapper: Wrapper, ...renderOptions });
}

// re-export everything
export * from "@testing-library/react";

// override render method, but for my convenience I made a seperate function.
export { renderWithInternationalization };
```

Unit tests polyfills

With all the preliminary work out of the way, let’s create our “App.test.tsx” file and write some tests. We will render a date with which we will translate to both English and Dutch.

Unit tests dates

First we create our reusable JSX DOM element. We add a data-testid for easy querying.

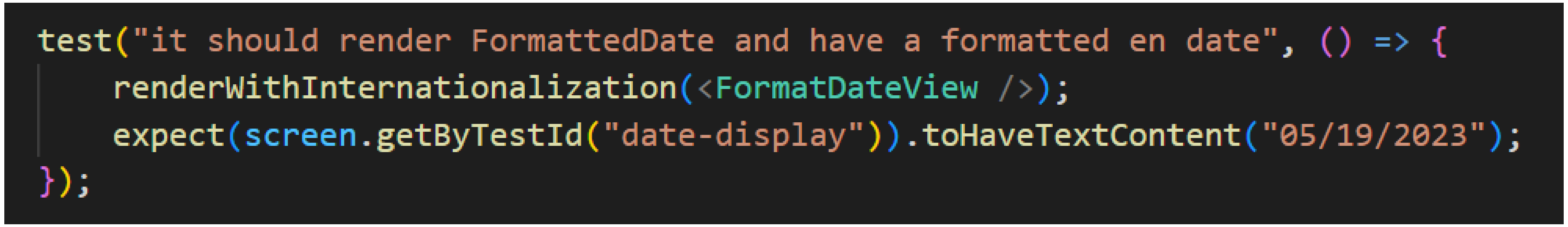


```
import React from "react";
import "@testing-library/jest-dom";
import { FormattedDate } from "react-intl";
import { setupTests } from "./test-utils";

const FormatDateView = () => {
    return (
        <div data-testid="date-display">
            <FormattedDate value="2023-05-19" timeZone="utc" day="2-digit" month="2-digit" year="numeric" />
        </div>
    );
};

setupTests();
```

Unit tests dates

Next we write our test for the English date. We could either use

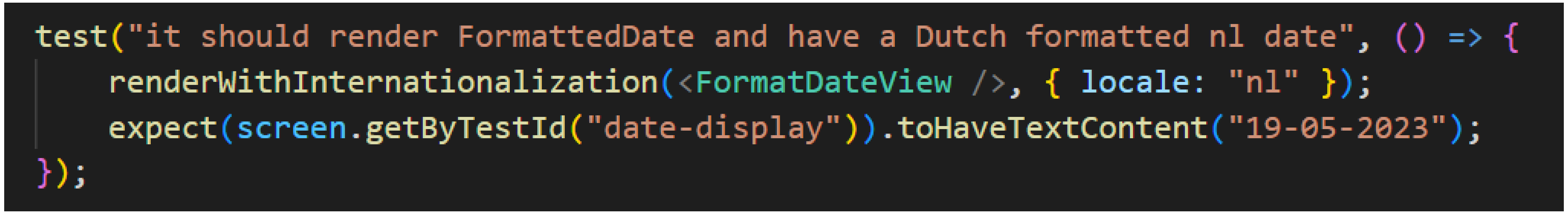
“toHaveTextContent” or “innerHTML”



```
test("it should render FormattedDate and have a formatted en date", () => {
    renderWithInternationalization(<FormatDateView />);
    expect(screen.getByTestId("date-display")).toHaveTextContent("05/19/2023");
});
```

Unit tests dates

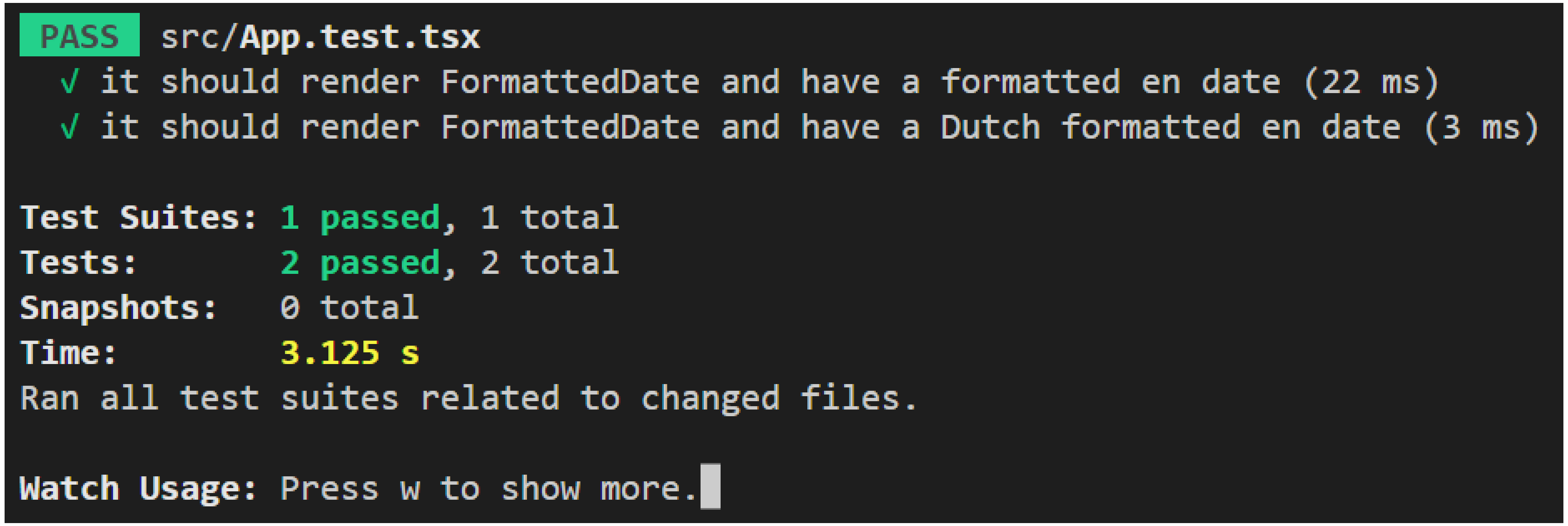
For Dutch we do the same thing, but change the value we are expecting.



```
test("it should render FormattedDate and have a Dutch formatted nl date", () => {
    renderWithInternationalization(<FormatDateView />, { locale: "nl" });
    expect(screen.getByTestId("date-display")).toHaveTextContent("19-05-2023");
});
```

Unit tests dates

If you run your tests with “npm test”, you will see both tests pass. Feel free to change the expected values to verify the tests will break.



```
 PASS  src/App.test.tsx
  √ it should render FormattedDate and have a formatted en date (22 ms)
  √ it should render FormattedDate and have a Dutch formatted en date (3 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        3.125 s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

Conclusion

We have covered the following today:

1. Overview of JavaScript i18n packages and setup of React-intl.
2. An overview of common internationalization problems.
3. Dealing with string, number and date translations.
4. ICU support.
5. Unit testing React-intl.

Thank you

Linkedin:
https://www.linkedin.com/in/rocodemy