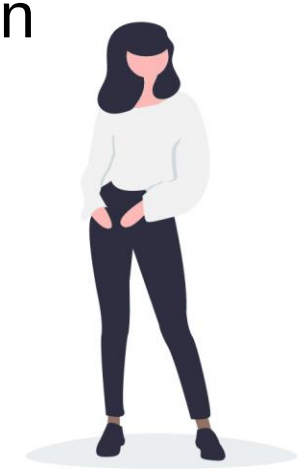
internationalization

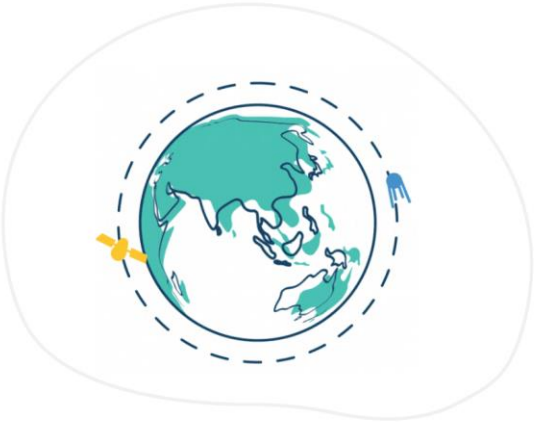
internationalization

What subjects should we consider when building an international webapp?

i18n subjects

- Numbers
- Currencies and exchange rates
- Units
- Date formats
- Timezones
- Direction
- Translation





Numbers

Large numbers look different in other locales, e.g:

Locale	Number
French, Canadian	295,000 967 294 4
German	967.295,000 294 4
Spanish, Italian	4.294.967.295,000
Swedish	295,000 967 294 4
US-English	4,294,967,295.00

Full list here

Numbers

the function `toLocaleString` can be called on a number to get its locale-specific string:

```
const num = 1234567.123
console.log('German locale:', formatNum(num))
// German locale: 1.234.567,123

function formatNumGerman(num) {
    return num.toLocaleString('de')
}
```

Numbers

The `Intl.NumberFormat` object also enables locale-sensitive number formatting:

```
const num = 1234567.123
console.log('Finland locale:', formatNumFi(num))
//  Finland locale: 1 234 567,123


function formatNumFi(num) {
    return new Intl.NumberFormat('fi').format(num)
}
```

Currency

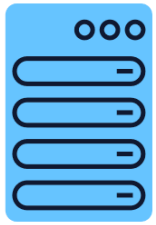
Here, we want to display the number as currency:

```
console.log('With Currency symbol:', formatCurrency(num))
//  With Currency symbol: 1,234,567.12 ₪

function formatCurrency(num) {
    const numFormat = Intl.NumberFormat('he-IL',
                        { style: 'currency', currency: 'ILS' })
    return numFormat.format(num)
}
```

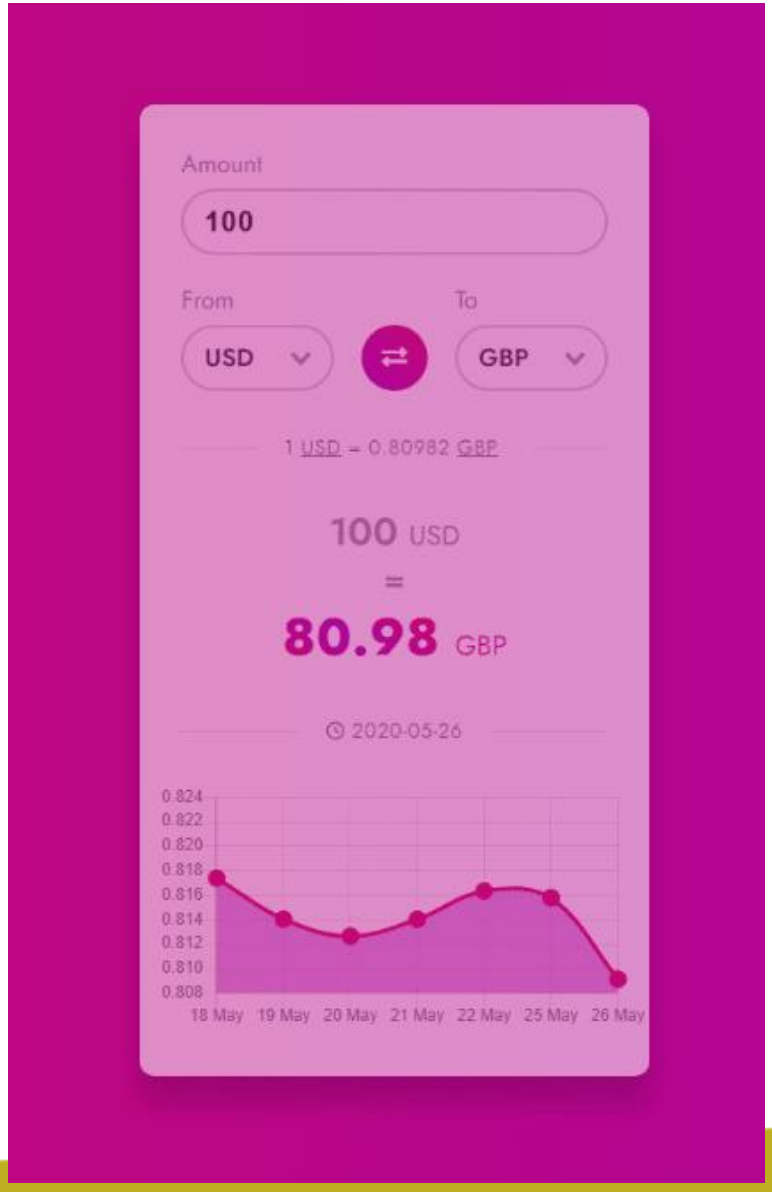
Currency

As we will later see,
showing the correct amount
requires asking some server
for the current rate

Client



Server



Units

Sometimes we may need to convert units, so the user will see the data in his preferred locale:

```
// Kilometers to Miles
function kmToMiles(km) {
    return km / 1.609
}

// Kilograms to Pounds:
function kgToLbs(kg) {
    return kg * 2.20462262185
}
```

Date formats

People describe dates in various formats

Locale	Format
Spanish	dd/MM/yyyy
USA	M/d/yyyy
German	dd.MM.yyyy
French, Dutch	d/MM/yyyy
Bulgarian	yyyy-M-d

More here

Date formats

Example:

```
console.log('Formatted date:', formatDate(new Date()))
// Formatted date: 6 בנוב׳ 2022, 11:12 לפנה״צ

function formatDate(time) {
    var options = {
        year: 'numeric', month: 'short', day: 'numeric',
        hour: 'numeric', minute: 'numeric',
        hour12: true,
    }
    return new Intl.DateTimeFormat('he', options).format(time)
}
```

Date formats

We can check which languages the user has configured in his browser:

```
// The default user-lang
const userLang = window.navigator.language

// All selected languages, such as: ['en-US', 'he']
const userLangs = window.navigator.languages

console.log('Date by user locale:',
    new Intl.DateTimeFormat(userLang,
        {   hour: 'numeric',
            minute: '2-digit',
            month: 'short'
        }).format(Date.now()))

// Date by user locale: Nov, 11:01 AM
```

Relative time

We can present relative times:

```
const date = Date.now()
console.log(new Intl.RelativeTimeFormat('he').format(-1,'day'))
// לפני יום 1
const opt = {numeric:'auto'}
const hebRTF = new Intl.RelativeTimeFormat('he',opt)
console.log(hebRTF.format(-1,'day'))
// אתמול

console.log(hebRTF.format(-2,'day'))
// שלשום

console.log(hebRTF.format(-3,'day'))
// לפני 3 ימים

console.log(hebRTF.format(-1,'year'))
// השנה שעברה

console.log(hebRTF.format(-4,'minute'))
// לפני 4 דקות
```

Relative time

So for an input time, we can create relative time like so:

```
function getPastRelativeFrom(ts) {
    const diff = Date.now() - new Date(ts)
    const seconds = diff / 1000
    const minutes = seconds / 60
    const hours = minutes / 60
    const days = hours / 24

    const formatter = new Intl.RelativeTimeFormat('en-US', {
        numeric: 'auto'
    })
    if (seconds <= 60) return formatter.format(-seconds, 'seconds')
    if (minutes <= 60) return formatter.format(-minutes, 'minutes')
    if (hours <= 24) return formatter.format(-hours, 'hours')
    return formatter.format(-days, 'days')
}
```

Relative time

We can call this function with some timestamp and examine the output:

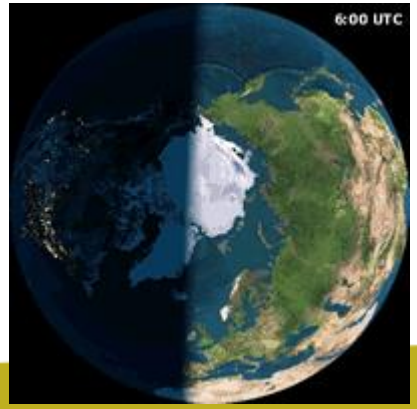
```
const ts1 = Date.now() - 1000 * 20
const ts2 = Date.now() - 1000 * 60 * 14
const ts3 = Date.now() - DAY * 2

console.log('Relative time:', getPastRelativeFrom(ts1))
console.log('Relative time:', getPastRelativeFrom(ts2))
console.log('Relative time:', getPastRelativeFrom(ts3))
```

```
Relative time: 20 seconds ago
Relative time: 14 minutes ago
Relative time: 2 days ago
```

Time Zone

- When we run JS in the browser, dates are according to the user TZ.
- Sometimes, it is needed to calculate stuff based on Timezone.
- Date-Fns is a useful library for handling dates and timezone:

```
const date = new Date('2018-09-01T16:01:36.386Z')
const timeZone = 'Europe/Berlin'
const zonedDate = utcToZonedTime(date, timeZone)
```

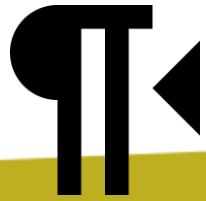


Direction

- The basic is: `direction: rtl;`
- Sometimes needs more fixes, examples:

```
float: right;
margin-right: 1.2em;
border-left: 1.1em;
```

- Usually a separate css file: rtl.css is used
- There are also specific libraries such as – bootstrap-rtl



Translation

When translating an app, we need to hold the translation for each phrase in the UI:

```
var gTrans = {
    appTitle: {
        en: 'My Todos',
        es: 'Mis Cosas',
        he:  'משימות להיום'
    } ,
    add: {
        en: 'Add',
        es: 'Aggregar',
        he: 'הוסף'
    }
}
```

Translation

Lets connect the element to their translation key:

```
<h1 data-trans="appTitle">My Todos</h1>
<button onclick="onAddTodo()" data-trans="add">Add Todo</button>
```

```
var gTrans = {
    appTitle: {
        en: 'My Todos',
        es: 'Mis Cosas',
        he: 'משימות להיום'
    } ,
    add: {
        en: 'Add Todo',
        es: 'Aggregar',
        he: 'הוסף'
    }
}
```

Translation

Lets review a simplified translation function:

```
function doTrans() {
    const els = document.querySelectorAll('[data-trans]')
    els.forEach(el => {
        const transKey = el.dataset.trans
        const trans = getTrans(transKey)
        el.innerText = trans
    })
}
```

Translation

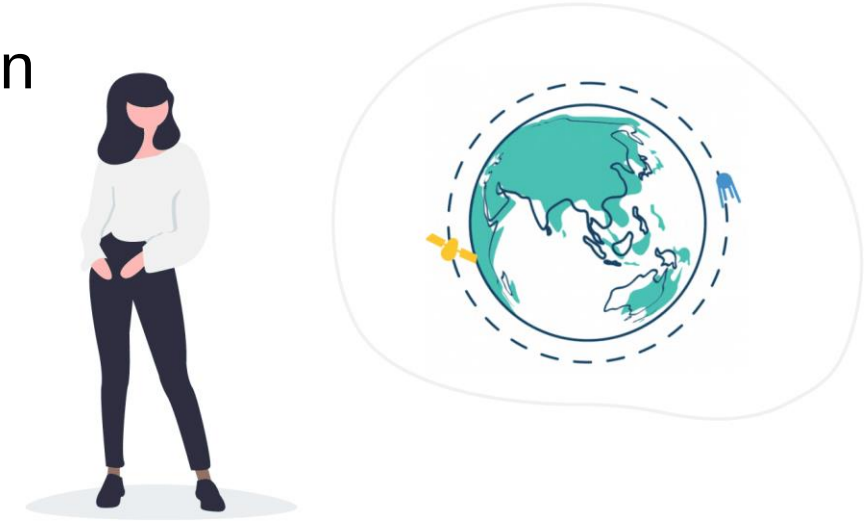
The strategy described in the previous slide is simplified and does not support elements inside other elements:

```
<p data-trans="subtitle">
    Doing is
    <span data-trans="living">Living</span>
</p>
```

In the real life, we will use some javascript library to achieve i18n (later)

i18n summary

- Numbers
- Currencies and exchange rates
- Units
- Date formats
- Timezones
- Direction
- Translation



Victorious!

You now know about

i18n