# Приложение конвертер валют (мастер-класс)

## Ссылка на заготовку

Имеется файлы **index.html,** с разметкой приложения.
В файле **style.css,** который находится в папке “**css”**, находятся необходимые стили.
В **index.html** уже подключены:

- CSS-*фреймворк* **Bulma**
- библиотека для работы с датой и временем **moment.js**.

Весь основной код приложения будем писать в файле **main.js**.

Для начала в файле main.js (папка “js”) создадим подписку на событие “*DOMContentLoaded*”, в функции обратного вызова которого будет находиться весь основной код приложения:

```
document.addEventListener("DOMContentLoaded", () => {

  // основной код приложения

});
```

Внутри ФОВ определим все необходимые переменные и константы.
В константе `apiUrl` будет храниться ссылка на внешний API, с которого мы будем получать актуальные курсы валют. В `defaultBase` - базовую валюту, которая будет отображаться по умолчанию. В `fromBases` - список валют, по которым будем получать актуальные курсы с внешнего API.

```
const apiUrl = 'https://api.exchangeratesapi.io/latest?base=';
const defaultBase = 'EUR';
const fromBases = ['RUB', 'EUR', 'USD', 'ISK', 'GBP', 'JPY'];
```

Дальше определим все необходимые константы для DOM элементов на странице:

```
const fromCurrency = document.getElementById("fromCurrency");
const toCurrency = document.getElementById("toCurrency");
const calculateButton = document.getElementById("submit");
const result = document.getElementById('result');
const header = document.getElementById('header');
const startUpdate = document.getElementById('start-update');
const display = document.getElementById('timer');
const currencyIcon = document.getElementById('currencyIcon');
const amountInput = document.getElementById('amount');
const timerSelector = document.getElementById('how-to-update'
);
const timerIcon = document.getElementById('start-icon');
const timerFooter = document.getElementById('timer-footer');
```

Заведем две переменные. Инициализируем пустой объект `currencyRatesData` с помощью литерала, в которой будем хранить в удобном для нас виде все данные о курсах валют. И переменную `bases`, в которой будет список валют, для которых удалось получить курсы.

```
let currencyRatesData = {};
let bases = [];
```

Далее необходимо получить актуальный курс валют с внешнего API, сделаем это с помощью вызова функции `getDataFromApi()`. И определим саму функцию:

```
function getDataFromApi() {
    bases = [];
    Promise.all(fromBases.map(base => getCurrencyRateData(bas
e))).then(values => {
        values.forEach(data => {
            bases.push(data.base);
            currencyRatesData[data.base] = data.rates;
        });
```

```
8
9           currencyRatesData.date = values[0].date;
10
11          renderDate(currencyRatesData.date);
12          makeFromSelectors();
13      });
14  }
```

Для начала в функция `getDataFromApi()` обнуляем массив `bases`.
Функция `getDataFromApi()` итерируясь по массиву `fromBases` с необходимыми валютами методом `map`, на каждой итерации вызывает функцию `getCurrencyRateData`,

```
1  function getCurrencyRateData(base) {
2      return fetch(`${apiUrl}${base}`).then(
3          res => res.json()
4      );
5  }
```

которая принимает в качестве аргумента конкретное значение названия валюты и возвращает *промис*, который мы получаем в результате вызова метода `fetch`.
Метод `fetch` в даном случае возвращает результат HTTP-запроса за курсом валют для конкретной валюты.
Поскольку у нас будет несколько независимых вызовов `fetch` для каждой из валют, которые вернут свой *промис*, удобно завернуть их всех в статический метод `all` глобального объекта `Promise`, что бы дождаться в методе `then` всех результатов одновременно. Полученные значения в методе `then` будут представлять собой массив, по которому итерируемся методом `forEach` и размещаем полученные данные в переменные `bases` и `currencyRatesData`.
В поле `date` объекта `currencyRatesData` записываем дату к которой относятся полученные курсы валют и вызываем с этой датой в качестве параметра функцию `renderDate(currencyRatesData.date)`,

```
1  function renderDate(date) {
2      header.innerText = `Согласно курсу валют на ${moment(dat
e).locale('ru').format('LL')}`;
```

```
}
```

которая отобразить в необходимом для этого DOM-элементе эту дату, отформатированную с помощью подключенной библиотеки **moment.js**.

Дальше **вызываем** функцию `makeFromSelectors`, которая  на основе полученных данных построит опции селектора в котором ми можем выбрать из какой валюты мы будем конвертировать.
Вызов в этом случае должен выглядеть самым обычным образом:

```
makeFromSelectors();
```

Рассмотрим детальнее саму функцию `makeFromSelectors`:

```
function makeFromSelectors() {
    let headBases = [];
    let tailBases = [];
    bases.forEach(base => {
        if (base === defaultBase) {
            headBases.push(base);
        }  else  {
            tailBases.push(base);
        }
    });

    const sortedBases = headBases.concat(tailBases);

    renderOptions(sortedBases, fromCurrency);

    setCurrencyIcon(sortedBases[0]);
    makeToSelectors(sortedBases[0]);
}
```

Предварительно в функции `makeFromSelectors` итерируемся по полученным базовым валютам и проверяем, какая из них равна значению валюты “по умолчанию”. Это значение помещаем в массив `headBases`, а все остальные в массив `tailBases`. После цикла в константу `sortedBases` поместим результат

конкатенации этих двух массивов, что даст нам возможность отобразить валюту “по умолчанию” в начале списка.
Дальше идет вызов функции `renderOptions`,

```
function renderOptions(optionsList, target) {
    let template = '';
    optionsList.forEach(option => {
        template += `<option value="${option}">${option}</option>`;
    });
    target.innerHTML = template;
}
```

которая отображает опции селектора в необходимый селектор.
После, в функции `makeFromSelectors` идет вызов функции `setCurrencyIcon`, в качестве аргумента в которую передаем установленную валюту по умолчанию.

```
function setCurrencyIcon(base) {
    switch (base) {
        case 'USD':
            clearClassList(currencyIcon);
            currencyIcon.classList.add('fas', 'fa-dollar-sign');
            break;
        case 'EUR':
            clearClassList(currencyIcon);
            currencyIcon.classList.add('fas', 'fa-euro-sign');
            break;
        case 'RUB':
            clearClassList(currencyIcon);
            currencyIcon.classList.add('fas', 'fa-ruble-sign');
            break;
        default:
            clearClassList(currencyIcon);
```

```
            currencyIcon.classList.add('fas', 'fa-money-bill-wave');
            break;
    }
}
```

которая установит соответствующую иконку валюты, которая является валютой по умолчанию.
Внутри функции `makeFromSelectors` используется утилитная функция `clearClassList`, которая очистись все классы у DOM-элемента, который передаем в качестве аргумента этой функции.

```
function clearClassList(element) {
    let classList = element.classList;
    while (classList.length > 0) {
        classList.remove(classList.item(0));
    }
}
```

В самом конце идет вызов функции `makeToSelectors`, в качестве аргумента в которую передаем установленную валюту по умолчанию.

```
function makeToSelectors(base) {
    let headKeys = [];
    let tailKeys = [];

    Object.keys(currencyRatesData[base]).forEach(key => {
        if (bases.indexOf(key) >= 0) {
            headKeys.push(key)
        } else {
            tailKeys.push(key)
        }
    });

```

```
    const sortedKeys = headKeys.concat(tailKeys);
    renderOptions(sortedKeys, toCurrency);
}
```

Данная функция, тоже отсортирует список отображаемых валют таким образом, что все базовые валюты будут в начале списка, а остальные в конце. И отобразит их аналогично, вызовом функции `renderOptions`, с передачей в качестве параметров уже отсортированный предварительно список доступных валют и того DOM-элемента, где необходимо их отобразить.

На этом заканчиваются все действия, который инициализировал вызов функции `getDataFromApi()` .

Следующим идет **вызов** функции `initListeners`, который инициализирует все необходимые функции-обработчики для дальнейшего функционирования нашего конвертера валют.

Сам вызов:

```
initListeners();
```

```
function initListeners() {
    fromCurrency.addEventListener('change', fromCurrencyHandle
r);
    calculateButton.addEventListener('click', calculateButtonH
andler);
    startUpdate.addEventListener('click', startUpdateHandler);
}
```

**Первым** обработчиком будет функция `fromCurrencyHandler`, которая будет обрабатывать событие изменения значения в селекторе, который отвечает за то, из какой валюты мы хотим конвертировать.

```
function fromCurrencyHandler(event) {
    makeToSelectors(event.target.value);
    setCurrencyIcon(event.target.value);
}
```

Этот обработчик перестраивает все опции селектора, ответственного за то в какую валюту мы конвертируем, если значение селектора “из какой” изменится, вызовом функции `makeToSelectors` з передачей в качестве параметра нового значения. А с помощью функции `setCurrencyIcon` изменяем значение иконки валюты на актуальное.

**Вторым** идет обработчик `calculateButtonHandler`

```
function calculateButtonHandler() {
    validateAmount(amountInput);
    const resultData = {};
    resultData.amount = amountInput.value;
    resultData.fromCurrency = fromCurrency.value;
    resultData.toCurrency = toCurrency.value;
    resultData.factor = currencyRatesData[fromCurrency.value][toCurrency.value];

    renderResult(resultData, result);
}
```

Эта функция запустится при нажатии на кнопку для расчета курса валют. Первое, это необходимо провалидировать введенное значение пользователем в input-поле, за что отвечает вызов функции `validateAmount`

```
function validateAmount(amountInput) {
    let amount = parseInt(amountInput.value, 10);

    if ((Number.isNaN(amount) === true) || (amount <= 0)) {
        amount = 1;
        amountInput.value = amount;
    } else if (amount > 100) {
        amount = 100;
        amountInput.value = amount;
    }
```

```
}
```

Здесь ми проверяем, является введенное значение целым положительным числом, если это не так, то переназначим в его в значение `1`. Если число больше 100, то переназначим в его в значение `100`.
Инициализируем пустым объектом с помощью литерала объекта константу `resultData`, в которой будем хранить результаты пересчета курса валют.
В поле `amount` объекта `resultData` будем хранить уже провалидированное значение с инпута. В `fromCurrency` - из какой валюты переводим. В `toCurrency` - в какую валюту. В `factor` - коэффициент перевода между валютами.
Отображаем результат с помощью функции `renderResult`:

```
function renderResult({amount, fromCurrency, toCurrency, factor}, target) {
    target.innerHTML = `<h1 class="is-large">${amount} ${fromCurrency} = ${(amount * factor).toFixed(2)} ${toCurrency}</h1>`
}
```

Эта функция получит в аргументах с помощью деструктуризации необходимые значения из передаваемого объекта, а так же DOM-элемент, в который и отобразит результат.

**Третьим** идет обработчик `startUpdateHandler`.

```
function startUpdateHandler() {

    if (!timer.running) {
        const duration = parseInt(timerSelector.value, 10);

        if (duration === 0 && !timer.running) {
            return;
        }

        showActiveTimer(startUpdate, timerIcon);

        timer.start(duration, () => {
```

```
            getDataFromApi();
            renderTimerInfo(timerFooter);
        });
    } else {
        timer.stop(() => {
            showPassiveTimer(startUpdate, timerIcon);
        });
    }
}

```

Внутри этого будет работа с экземпляром (инстансом) класса `CountDownTimer`, поэтому необходимо инициализировать его. Это лучше всего сделать вверху файла **main.js**, там где мы инициализировали все необходимые переменные.

```
const timer = new CountDownTimer(display);
```

Класс `CountDownTimer` определен в файле **timer-es6.js** (папка “js”) как класс в синтаксисе *ES6* и в **timer.js** как функция-конструктор в синтаксисе *ES5*.

В обработчике `startUpdateHandler` сначала проверяем не запущен ли уже таймер в данное время через поле `running` объекта-инстанса `timer`.
Если *уже запущен* (ветка *else*), то это значит, что кнопка запуска нажимается повторно для вызова остановки таймера. Поэтому переходим к ветке `else` и вызываем метод `stop` у таймера, чтобы остановить обратный отсчет. Более детально метод `stop` описан дальше.
Если не запущен (ветка *if*), то в константу `duration` записываем значение из селектора. Предварительно это значение преобразовываем в целое число с помощью функции `parseInt`. Константа `duration` отвечает за длительность периода между обновлениями курса валют.
За тем проверяем, если значение равно `0`, то делаем возврат из функции с помощью ключевого слова `return`, что бы не запускать таймер в этом случае.
Далее идет вызов функции `showActiveTimer`:

```
function showActiveTimer(button, icon) {
    button.classList.remove('is-link');
    button.classList.add('is-primary');
```

```
    clearClassList(icon);
    icon.classList.add('fas', 'fa-stopwatch');
}
```

Эта функция просто изменяет внешний вид кнопки запуска таймера, что бы визуально оповестить пользователя, что таймер активный в данное время.

Далее идет вызов метода `start` объекта-инстанса `timer`, с передачей значения `duration` в качестве параметра, что бы запустить непосредственно сам таймер обратного отсчета.  В качестве второго аргумента в метод `start` передаем анонимную стрелочную функцию, которая будет запущена по истечению таймера. Эта функция инициализирует запуск функции `getDataFromApi` для получения актуальных курса валют с внешнего API.
С помощью функции `renderTimerInfo`, которая принимает DOM-элемент в качестве параметра,

```
function renderTimerInfo(target) {
    target.innerHTML = `<p class="card-header-title">Курс валют успешно обновлен</p>`;
    setTimeout(function () {
        target.innerHTML = '';
    }, 3000);
}
```

запустим отображение информации что таймер завершился и запрошены новые данные.

Рассмотрим отдельно сам класс `CountDownTimer`:

```
class CountDownTimer {
    constructor(element) {
        this.element = element;
        this.running = false;
        this.millisecondsDuration = 0;
        this.smallDelay = 1000;
```

```
        this.interval = null;
    }

    static formatValue(value) {
        return (value < 10)
            ? `0${value}`
            : `${value}`
    }

    start(duration, done) {
        if (this.running) {
            return
        }

        this.running = true;
        this.millisecondsDuration = parseInt(duration) * 60 *
1000;
        this.interval = setInterval(() => {
            let hours = Math.floor((this.millisecondsDuration
% (1000 * 60 * 60 * 24)) / (1000 * 60 * 60));
            let minutes = Math.floor((this.millisecondsDuratio
n % (1000 * 60 * 60)) / (1000 * 60));
            let seconds = Math.floor((this.millisecondsDuratio
n % (1000 * 60)) / 1000);

            this.element.innerText = `${CountDownTimer.formatV
alue(hours)}:${CountDownTimer.formatValue(minutes)}:${CountDow
nTimer.formatValue(seconds)}`;

            if (this.millisecondsDuration <= 0) {
                if (typeof done === "function") done();
                this.millisecondsDuration = duration * 60 * 10
00 + this.smallDelay;
            }
```

```
34
35              this.millisecondsDuration -= 1000;
36          }, 1000);
37      }
38
39      stop(done) {
40          if (!this.running) {
41              return
42          }
43
44          this.millisecondsDuration = 0;
45          this.element.innerText = '00:00:00';
46          this.running = false;
47          clearInterval(this.interval);
48          if (typeof done === 'function') {
49              done();
50          }
51      }
52  }
```

В конструкторе данного класса принимаем в качестве аргумента DOM-элемент, в котором будет отображаться сам таймер, который помещаем в `this.element`.
Создаем у инстанса поле `running`, которое инициализируем значением `false`. Это поле будет выполнять роль флага, который будет хранить состояние, запущен ли инстанс таймера или нет.
Создаем у инстанса поле `millisecondsDuration` в котором будем хранить длительность таймера в миллисекундах.
Создаем у инстанса поле `smallDelay`, которое инициализируем значением в 1000 миллисекунд. Это значение будет полезным для лучшего визуального отображения таймера во время запуска, так как в конце цикла после проверки ми постоянно уменьшаем `this.millisecondsDuration` на 1000 миллисекунд.
В поле `interval` будем хранить значение интервала, созданного с помощью метода `setInterval`.
Инициализационным значением будет значение `null`. Значение созданное методом `setInterval`, которое мы сохраняем в поле класса `interval` нам

необходимо будет в дальнейшем, для того что бы при вызове мотода `stop` мы могли остановить выполнение функции обратного вызова, которая постоянно выполняется через определенные промежутки времени.
У класса есть два метода

- `start` - для запуска таймера,
- `stop` - для остановки таймера.

Еще есть один статический метод `formatValue` для форматирования отображаемого значения в таймере.

При вызове метода `start`, который принимает в качестве аргументов длительность `duration` в минутах и функцию обратного вызова `done`, которая выполнится по истечению длительности таймера обратного отчета.
Внутри метода проверяем, не запущен ли уже в данное время таймер. Если уже запущен, то предотвращаем повторный запуск. Если нет, то устанавливаем флаг `this.running` в значение `true`.
В поле `this.millisecondsDuration` устанавливаем значение длительности в миллисекундах.
В поле `this.interval` записываем значение созданного методом `setInterval` интервала, который принимает в качестве первого аргумента функцию обратного вызова, которая будет запускаться периодически, а в качестве второго интервал повторений вызова в миллисекундах, который в нашем случае равен 1000 миллисекунд (то-есть раз в секунду).
С помощью него будет отображаться информация о течение таймера, запускаться по истечению всего времени функция обратного вызова `done` и сбрасываться значение самого таймера для повторного отсчета.

При вызове метода `stop`, который принимает в качестве аргумента функцию обратного вызова `done`, проверяется, что таймер запущен. Если он запущен, обнуляем все поля инстанса таймера, очищаем созданный интервал методом `clearInterval` и запускаем (если передана) функцию обратного вызова `done`.