Цель урока:

1. Узнаете, что такое import и export
2. Разберетесь, как они помогают разделять функциональность проекта на разные файлы и переносить сущности из одного файла в другой
3. Научитесь использовать import и export на практике

Содержание урока:

1. Использование import, export, export default. Реализация структуры проекта
2. Разработка класса JSBlock. Работа с import и export
3. Разработка класса TimerBlock. Работа с import и export
4. Реализация вспомогательных функций. Работа с import all
5. Внесение последних правок в проект

Дополнительные материалы:

1. Импорт и экспорт в JavaScript: https://learn.javascript.ru/im...

Обратите внимание!

Если вы используете current версию Node.js и у вас появляются ошибки при запуске проекта, то переустановите Node.js на версию LTS

Задание #1

В заданиях данного урока вы создадите с нуля сервис доната денег на благотворительные нужды.

У вас есть пустой проект. Откройте в редакторе кода.

Установите все пакеты при помощи команды **npm install** и запустите проект через **npm start**.

Создайте структуру проекта. Для этого в корне проекта создайте папку **src**, а в ней еще две папки:

1. **core** - для глобальных частей проекта
2. **modules** - для основных компонентов (например, списка донатов, формы отправки доната и т.д.)

В папке **modules** создайте файл **app.js** и нем необходимо реализовать класс **App**. В классе должен быть метод **run**, который помещает внутрь тега **body** текст “Hello World”.

Импортируйте по умолчанию класс **App** (export default) в файл **index.js**, создайте экземпляр класса **App** и вызовите метод **run**.

Задание #2

Поздравляем! Вы создали начальную структуру вашего проекта и попрактиковались с import и export.

Далее реализуйте форму создания донатов. В папке **modules** создайте файл **donate-form.js**. В данном файле вам необходимо создать класс **DonateForm** с методом **render**, который будет возвращать HTML-элемент итоговой формы. Экспортируйте (export) класс **DonateForm**.

В классе **DonateForm** вам необходимо реализовать HTML-структуру ниже, используя **document.createElement**. Для разделения логики позволяется создавать свои поля и методы класса. По возможности делайте их приватными, чтобы писать чистый код на ООП и применять принцип **инкапсуляции**.

```
<form class="donate-form">
      <h1 id="total-amount">28$</h1>
      <label class="donate-form__input-label">
            Введите сумму в $
            <input class="donate-form__donate-input" name="amount"
type="number" max="100" min="1" required="">
      </label>
      <button class="donate-form__submit-button" type="submit">
          Задонатить
      </button>
</form>
```

Затем импортируйте класс **DonateForm в app.js**.

Инициализируйте DonateForm в конструкторе класса App и поместите инстанс (экземпляр класса, полученный через new DonateForm()) в свойство **this.#donateForm**. Затем в методе **run** класса **App** добавьте разметку формы, полученную через **this.#donateForm.render()**, внутрь тега **body.**

Задание #3

Вы молодец! Вы шаг за шагом создаете крутой сервис для доната и у вас неплохо получается.

Сейчас вам необходимо будет отрисовать список всех донатов, которые будут отправлять пользователи сервиса.

В папке **modules** создайте файл **donate-list.js** и в нем реализуйте класс **DonateList**. Конструктор данного класса должен принимать массив **donates**. Данный массив состоит из объектов, у которых есть ключи **date** (дата создания доната) и **amount** (сумма доната). Реализуйте в классе **DonateList** метод **render**, который будет отрисовывать HTML-шаблон приведенный ниже:

```
<div class="donates-container">
   <h2 class="donates-container__title">Список донатов</h2>
   <div class="donates-container__donates">
      <div class="donate-item">July 6th 2021, 10:28:49 am -
<b>4$</b></div>
      <div class="donate-item">July 6th 2021, 10:28:49 am -
<b>20$</b></div>
      <div class="donate-item">July 6th 2021, 10:28:49 am -
<b>3$</b></div>
      <div class="donate-item">July 6th 2021, 10:28:49 am -
<b>1$</b></div>
   </div>
</div>
```

Элемент ниже отображает информацию о донате:

```
<div class="donate-item">July 6th 2021, 10:28:49 am - <b>4$</b></div>
```

“July 6th 2021, 10:28:49 am” - значение свойства **date** из объекта массива **donates**

“4$” - значение свойства **amount**

Экспортируйте класс **DonateList** (export), а затем импортируйте его в файл **app.js**.

По аналогии с DonateForm инициализируйте класс **DonateList** в конструкторе класса App и поместите инстанс в свойство **this.#donateList**. В методе **run** класса **App** добавьте разметку списка донатов, полученную через **this.#donateList.render()**, внутрь тега **body.**.

Для теста отрисовки списка донатов, используйте данный массив **donates**:

```
const mockDonates = [
   { amount: 4, date: new Date() },
   { amount: 20, date: new Date() },
   { amount: 3, date: new Date() },
   { amount: 1, date: new Date() },
];
```

Примечания:

1. Для разделения логики позволяется создавать свои поля и методы в классе **DonateList**. По возможности делайте их приватными, чтобы писать чистый код на ООП и применять принцип **инкапсуляции**
2. Сейчас у вас текст даты в HTML-элементе доната будет отображаться не как в HTML-шаблоне, это нормально. Данную логику вы разработаете позже.

Задание #4

Вы неплохо справились с отрисовкой данных и хорошо попрактиковались с import/export. Сейчас же пришло время добавить немного логики и обработчиков событий.

Создайте глобальное состояние, в котором будут храниться все важные данные проекта. Для этого в конструкторе класса **App** создайте объект **state** (this.state), в котором будут два свойства:

1. **donates** - массив донатов, который будет передаваться в конструктор класса **DonateList**. По умолчанию значение должно быть пустым массивом.
2. **totalAmount** - общая сумма донатов (тип данных number). По умолчанию значение должно равняться нулю.

Передайте массив **donates из state** в конструктор класса **DonateList**, а значение свойства **totalAmount** - в конструктор класса **DonateForm**.

Обновите класс **DonateForm:**

1. в конструкторе добавьте параметр **totalAmount** и инициализируйте его с помощью **this** (this.totalAmount)
2. создайте метод **updateTotalAmount**, который будет принимать параметр **newAmount** и помещать текст в формате "**newAmount**$" в элемент h1 с **id = “total-amount”** (этот элемент уже у вас есть, находится внутри тега form). В итоге изначально у вас получится “0$”.

В классе **DonateList** создайте метод **updateDonates**, который будет принимать параметр **updatedDonates** (новый массив **donates**). Данный метод должен очищать элемент с классом “donates-container__donates” и отрисовывать в нем новые донаты из массива **updatedDonates.**

После создания каждого доната вам необходимо обновлять объект **state** в классе **App** и HTML-элементы. Для обновления состояния создайте метод **createNewDonate** в классе **App**, данный метод принимает параметр **newDonate**, который является новым объектом массива **donates.**

В методе **createNewDonate** нужно:

1. Добавить **newDonate** в массив **donates** объекта **state**
2. Обновить **totalAmount** объекта **state**
3. Вызвать метод **updateDonates** класса **DonateList** (this.#donateList.updateDonates) и передать в него массив donates из state

4. Вызвать метод **updateTotalAmount** класса **DonateForm** (this.#donateForm.updateTotalAmount) и передать в него массив totalAmount из state

После этого передайте метод **createNewDonate** класса **App**, как параметр в конструктор класса **DonateForm** и инициализируйте его в конструкторе при помощи **this** (this.createNewDonate).

В классе **DonateForm** повесьте обработчик событий “submit” на элемент с классом “donate-form”. В данном обработчике вам необходимо:

1. Создать новый объект доната с ключами **date и amount**
2. Вызвать метод **this.createNewDonate**, который вы до этого передали в конструктор
3. После создания каждого доната текстовое поле с классом “donate-form__donate-input” должно быть пустым

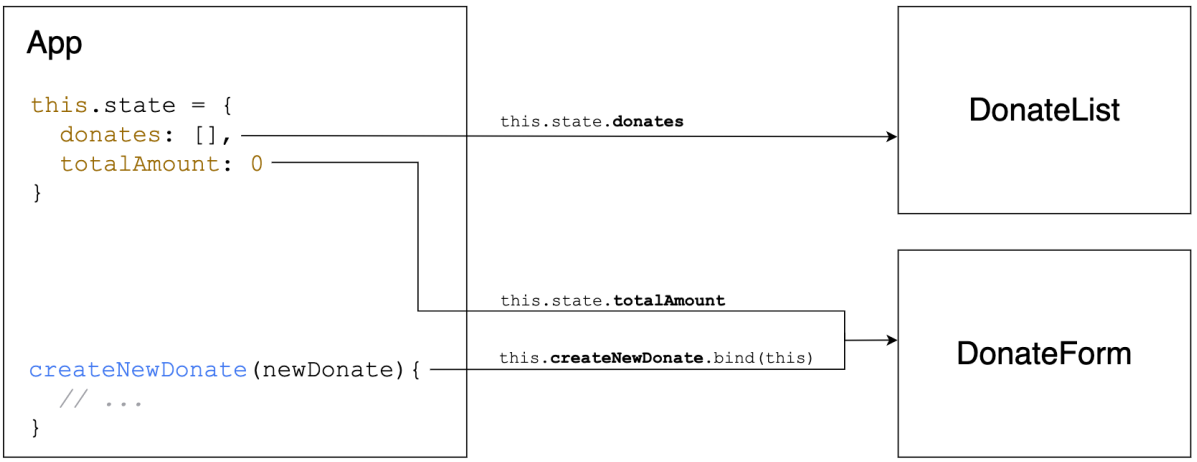
Примечание:

Когда вы передаете функцию или метод как параметр в другую функцию или метод, у вас может теряться **this**. Поэтому присвойте this через bind.

Подсказка:

this будет теряться при передаче метода **createNewDonate** как параметра в конструктор класса **DonateForm**, передайте его, как **this.createNewDonate.bind(this)**. Здесь **bind(this)** привязывает контекст класса **App** к методу **createNewDonate**.

Схема:



Задание #5

Поздравляем! Вы почти полностью реализовали приложение для донатов. У вас сейчас работает логика создания донатов и отрисовка всех нужных HTML-элементов.

Теперь осталось немного поработать с качеством написанного кода.

Для начала давайте заметим, что у нас все суммы писались в долларах “$”. А представьте, что придет заказчик и скажет: “А давайте все суммы будут в евро”. С текущей

реализацией вам придется исправлять значок доллара на евро во всех возможных файлах, что конечно же неудобно. Для этого есть решение. Создать глобальное значение и использовать его во всем проекте.

Поэтому в папке **core** создайте папку **constants**. В папке **constants** создайте файл **settings.js**, в котором будет хранится объект **Settings**. В данном объекте создайте ключ currency, у которого будет значение **“$”**. Экспортируйте **Settings** и затем с помощью import заменить все значки “$” в файлах на Settings.currency.

После проделанной работы вы можете спокойно изменять значение свойства currency в объекте **Settings**, и валюта изменится во всем проекте.

Условие: при импортах необходимо хотя бы 1 раз использовать ключевое слово **as.** Например, import { Object as Obj }.

Задание #6

Проект почти готов и осталось сделать всего несколько важных дел. Сейчас вам необходимо будет создавать вспомогательные функции, которые будут использоваться глобально по проекту.

Для начала создайте папку **utils** в **core**. В **utils** создайте файл index.js. В данном файле будут хранится все вспомогательные функции, которые относятся к проекту.

Первая вспомогательная функция должна называться **calculateSumOfNumbers**. Она принимается в качестве параметра массив **numbers**, который состоит исключительно из чисел. Функция **calculateSumOfNumbers** считает сумму всех элементов массива и возвращает ее итоговое значение.

Вторая функция будет отвечать за конвертацию даты. Назовите функцию **getFormattedTime**. Она принимает в себя один параметр **date**, который является объектом **Date**. Изначально **new Date()** при преобразовании к строке возвращает примерно такое значение **“Tue Jul 06 2021 14:00:05 GMT+0300 (Moscow Standard Time)”**. Функция **getFormattedTime** должна преобразовывать его к такому значению **“July 6th 2021, 2:00:05 pm”**. Это можно сделать при помощи библиотеки **momentjs**.

```
moment(date).format('MMMM Do YYYY, h:mm:ss a')
```

Установите **momentjs** (https://momentjs.com/) и реализуйте функцию **getFormattedTime**

После вам необходимо экспортировать эти функции и использовать в коде.

Начнем с **calculateSumOfNumbers**. Создайте в файле **app.js** массив **mockDonates** и передайте его как значение в **donates** у объекта **state**.

```
const mockDonates = [
  { amount: 4, date: new Date() },
  { amount: 20, date: new Date() },
  { amount: 3, date: new Date() },
  { amount: 1, date: new Date() },
];
```

Так как у нас есть начальные значения **donates**, а **totalAmount** по умолчанию стоит как 0, то необходимо рассчитать начальную сумму всех донатов и перезаписать значение **totalAmount**. В этом вам поможет уже созданная функция **calculateSumOfNumbers**.

Вторую вспомогательную функцию **getFormattedTime** вам необходимо использовать для элементов с классами **“donate-item”**. До этого текст в данных элементах выглядел следующим образом “Tue Jul 06 2021 14:00:05 GMT+0300 (Moscow Standard Time) - 4$”. С использованием **getFormattedTime** текст должен стать таким “July 6th 2021, 2:00:05 pm - 4$”.

Условие: при импортах вам обязательно нужно использовать конструкцию **“import * as”**.