***#21 Объекты***

В JS без объектов никуда, ведь все здесь на них и построено (но об этом чуть позже)

Вы уже хорошо понимаете переменные и знаете какие типы данных могут быть у их значений. С объектами все будет похоже, но с некоторыми дополнительными возможностями

Ваш калькулятор снова можно переписать указав все возможные операторы внутри объекта operations - должно получится еще лаконичнее и еще красивее!

Программа должна принимать следующие значения в качестве оператора:

"sub", "sum", "mult" "div"

***#23 Практика***

Сегодня напишем самый простой TODO лист без графического интерфейса.

Хранилищем будет объект, а имена задач - ключами.

const list = {

"create a task": "In Progress",

"make a bed": "Done",

"write a post": "To Do",

}

Функция changeStatus - будет менять статус задачи

changeStatus("write a post", "Done")

Функция addTask - добавляет новую задачу

addTask('have a walk')

Функция deleteTask - удаляет задачу

deleteTask('have a walk')

Функция showList будет выводить весь список дел в виде

Todo:

"create a task",

"make a bed",

In Progress:

"write a post"

Done:

-

Создайте список дел, добавьте в него пару задач, поменяйте их статусы несколько раз и выведете результат в консоль

***#26.2 Вертикальная строка***

Напишите функцию которая преобразует полученную строку в "вертикальный вид" и выводит ее в консоль. Чтобы получилось вот так:

showVerticalMessage('марафон')

М

а

р

а

ф

о

н

Если строка начинается с буквы "м" - нужно вывести эту строку с первой заглавной буквой

Если строка больше 10 символов - вывести только первые 10

***#29.2 TODO. Теперь уже по-настоящему***

Согласитесь, после чтения статьи, вам стало очевидно что для задачи с TODO вам нужен именно массив, а никак не объект?

При чем, еще интереснее будет иметь массив объектов, например такой:

const list = [

{

id: 1,

name: 'create a post',

status: 'TODO’,

priority: 'low'

},

{

id: 2,

name: 'test',

status: 'Done',

priority: 'high'

}

]

***#29.3 TODO. Бонус***

Только для самых крутых, которые еще 2 часа назад закончили TODO на массивах

Сделайте возможность выбирать группировку для вывода. Например

showBy(‘priority’) - чтобы выводил по приоритетам

high:

test

somethin else

low:

-

чтобы можно было вызывать showBy(‘priority’) или showBy(‘status’)

***#42 Калькулятор UI***

Ну что, поверстаем немножечко?

Сверстать нужно точно по макету (никакой самодеятельности)

Никакого responsive или адаптива делать не надо.

Просто верстка, так как она есть на макете

Знаю, это будет просто, но не все из вас верстали из фигмы - на это может уйти дополнительное время. В вашем распоряжении целый день!

<https://www.figma.com/file/Lokbr5D9pFTMLAzP2fmANU/Responsive-Calculator-App-(Community)?node-id=0%3A1>

***#43 Калькулятор***

Логика готова, представление готово, осталось вставить одно в другое.

Ваша задача: иметь два файла index.html и main.js. Подключите main в ваш index и добавьте обработчики для кнопок (event listeners)

Вот вам и готовый калькулятор 💁🏽‍♂️

Только вы ещё не читали про вставку/изменение текста - это простая тема, она есть здесь

<https://learn.javascript.ru/basic-dom-node-properties>

***#46 Модификация документа***

В калькуляторе все ваши UI элементы были статичными, но очень часто нужно добавлять или удалять элементы (список задач в TODO, например)

Для этого в JS есть целый список инструментов. Применять их мы будем уже очень скоро

<https://learn.javascript.ru/modifying-document>

***#48.2 Модульный калькулятор***

Потренеруйтесь на уже готовой программе.

Добавьте все UI элементы (все querySelector) в отдельный модуль под названием view.js

***#49.1 Сделаем TODO с интерфейсом***

Сначала давайте все сверстаем

Верстка, кстати, максимально простая, так как наша главная задача в другом

Подсказки можно отключить в меню слева

(обращайтесь в чат, если не получится найти их)

<https://www.figma.com/file/SXjNGNZD29InYpSOpLTZsf/Untitled?node-id=0%3A1>

***#49.2 И снова модули***

Добавьте все UI элементы вашего TODO в отдельный модуль view.js

Все взаимодействие с DOM элементами будем делать в этом же модуле (view.js), вся логика останется в main.js

***#51 ToDo до конца***

Ну вы поняли.

Сегодня нужно просто доделать туду. Так, чтобы можно было удалять, добавлять и отмечать как «готово» все ваши задачи из списка

Все знания для этого у вас уже есть, осталось перенести их в код и подготовить это для ревью на прямом эфире

Extra: будьте наставником для тех, у кого не получилось или не получилось в самом лучшем виде

Extra Extra!: если вы были на прошлых эфирах - донесите до остальных лучшие практики, которые я рассказывал вам раньше

Самые активные участники не останутся без внимания 🙌

***#53 Ошибки***

Пришло время научиться по-взрослому обрабатывать ошибки.

Больше никаких строк, типа “Error”, больше никаких «ой, ошибка».

Теперь все по-настоящему!

После чтения теории, добавьте обработку ошибок в ваши программы. Это может быть хоть калькулятор, хоть туду, хоть все вместе.

<https://learn.javascript.ru/try-catch>

***#55.2 Немного практики***

Пока мы не используюем специальных подходов чтобы просто так импортировать данные из json файла, сделаем это эксперементальным путем

(не запоминайте этот подход, он нужен только для практики)

Скайте файл data.json из вложения

Создайте файл main.mjs (внимание на расширение файла)

В вашем файле main.mjs добавьте импорт файла с данными data.json (файлы должны быть в одной папке)

import data from './data.json'

Откройте терминал в этой папке и запустите ваш код со специальным флагом

node --experimental-json-modules main.mjs

теперь в переменной data будут данные из файла json

можете проявить свою фантазию и вывести эти данные в консоль разным образом.

например вывести только имена людей или вывести строку типа

Dennis, born at September 9, 1941 - created the C programming language

для каждого из участников

В этой задаче так же можно попробовать все методы из прочитанной вами статьи

Уже скоро мы будем получать такие данные по сети и делать с ними интересные вещи

***#58 Практика***

Напишите программу, которая передает на сервер имя и в ответ получает "возможный" пол для этого имени.

Вот вам готовая строка с URL (только имя успевайте менять)

const firstName = 'artem';

const serverUrl = 'https://api.genderize.io';

const url = `${serverUrl}?name=${firstName}`;

Сделайте самый простой интерфейс для этой программы

например input (в него введете имя)

и кнопку рядом (она будет отправлять это имя на сервер)

в alert или в строке ниже показывайте результат вот такого вида

"Artem is male"

p.s. помните fetch не работает в nodeJS, а значит вам нужно запускать это в браузере

***#58.2 Бонус***

Если вам совсем скучно - сделайте 2 запроса.

Один будет угадывать пол, а другой страну.

Вот url сервера который по имени отдает массив стран

'https://api.nationalize.io'

***#62 Погода. Верстка***

Сегодня начнем делать программу побольше.

Погодное приложение, с поиском городов, детальной сводкой и главное: сетевыми запросами. Ну еще и с сохранением городов

Начнем с верстки. Предлагаю сделать 2 первых экрана сегодня (Now и Details)

Никакой логики, только верстка.

<https://www.figma.com/file/Wio9K8x83Mrktczh6QOrjH/Untitled-(Copy)?node-id=0%3A1>

***#63 Погода. запрос города по API***

Пора запросить хотя бы один город из строки поиска. Ничего сложного - вы уже делали все это в прошлых практических задачах.

Первое - ввдоим что-то в строку поиска, по нажатию на Enter - берем имя города из этой строки

Второе - подставляем это имя в url через шаблонные строки и делаем запрос на сервер

const serverUrl = 'http://api.openweathermap.org/data/2.5/weather';

const cityName = 'boston';

const apiKey = 'f660a2fb1e4bad108d6160b7f58c555f';

const url = `${serverUrl}?q=${cityName}&appid=${apiKey}`;

Третье - результат записываем в любую переменную и выводим нужную информацию в табе 'Now' согластно верстке

Остальные табы трогать не надо

***#64 Погода. Ошибки и несуществующие города***

Типичная обработка ошибок запроса по сети

Для простоты будем показывать пользователю обычный alert (потом научимся делать круче)

Но, перед этим прочитайте тему про обработку ошибок в промисах и используйте эту технику

https://learn.javascript.ru/promise-error-handling

Чтобы имитировать ошибку и проверить себя - поменяйте что-то в url, например адрес сервера

А ещё можно показывать alert если такого города не нашлось

***#65 Погода. Избранное***

Добавим возмодность добавлять города, чтобы не вписывать их вручную в строку поиска каждый раз.

Узнаете? Это ж ровно то же самое что вы делали в ТУДУ листе. Можете использовать ту же логику, а можете написать все с полного нуля.

По клику на иконку с сердцем добавляйте города в список справа, по клику на крестик рядом с именем города в этом списке - удаляйте его навсегда

По клику на сам город - добавляйте его данные в окно 'Now'

Вот вы уже пол программы и написали, не сложно же?)

***#67 Погода. Сохранение состояния***

Сейчас после перезагрузки страницы все ваши города исчезают и это ужасно не удобно.

localStorage - отличное место для хранения этой информации в браузере - добавленные в избранное города всегда будут на своем месте.

А еще, давайте сохранять текущий "выбранный" город и после перезагрузки снова делать выбранным именно его

Для работы с localStorage можно создать отдельный модуль, чтобы не сваливать все в кучу. Да и выглядеть это будет ой как красиво

storage.saveFavoriteCities(favoriteCities)

const favoriteCities = storage.getFavoriteCities();

const currentCity = storage.getCurrentCity();

Это просто пример, делайте как сами считаете нужным.

***#68 Погода. Детали***

Мы уже получаем всю информацию про нужный город, но пока выводим только главную информацию о нем. Пришло время наполнить жизнью вкладку 'Details'

Уверен, вы уже знаете что делать и задача будет достаточно простой, поэтому вот вам задача еще:

Прочитайте про "другую разновидность функций" (уверен, вы уже используете это)

<https://learn.javascript.ru/function-expressions>

***#69 Погода. Расписание (верстка)***

Кто-то из вас уже сверстал все 3 таба. Хотя задачи такой не было.

Сверстайте 3-й таб 'Forecast'. Пока просто сверстайте.

Ну а чтобы совсем скучно не было от этой верстки, вот вам немного полезной теории (которую, возможно, вы уже знаете)

<https://learn.javascript.ru/arrow-functions-basics>

***#70 Погода. Расписание (логика)***

По примеру с табом 'Details' сделайте таб 'Forecast' - здесь будет похожая логика, только набор данных чуть больше. Уверен вы запросто справитесь

А вот вам несколько советов по стилю кода (если вы были на эфирах, то знаете многое из этого)

<https://learn.javascript.ru/coding-style>

***#71 Погода. Все вместе***

К этому дню все должно работать как задумано.

Если это не так - доделывайте вашу программу и обязательно обращайтесь за помощью в чат.

Если все уже готово - посмотрите на свой код еще раз. Представьте что его будет обслуживать и поддерживать серийный убийца, который знает где вы живете.

Сделайте выводы и привидите код в порядок.

В этом здорово может помочь деструктуризация

(ее вы видели в примерах кода)

<https://learn.javascript.ru/destructuring-assignment>

***#72 Синхронно записанная асинхронность***

такой синтаксис вы уже видели в примерах в других темах. Возможно кто-то из вас даже понял как это работает. Теперь настала пора понять это на все 100%

https://learn.javascript.ru/async-await

Перепишите 1, 2 или сколько захотите ваших .then на async/await и не забудьте про try/catch

***#81 Бонус задача для самых смелых***

Если вы были в темпе и уже успели отдохнуть, а может даже и заскучать - вот вам задача:

Возьмите проект с погодой, сделайте копию (чтоб не вносить ничего в оригинал), замените все переменные на функции.

Как это? Вот пример:

была переменная

const textLength = event.target.value.length;

а теперь функция const getTextLength = (value) => value.length;

замените все переменные на фуккции (кроме тех, что объявленны в UPPER\_CASE)

все такие константы можно оставить константами, так как они, скорее, "конфиг" нашей программы, а не инструкции в коде

***#91 Конструктор***

Простая для понимания и одна из главных тем в JS научит вас создавать свои объекты через ‘new‘

Попробуйте применить это в “TODO” (создание объекта задачи) или “Погоде” (создание объекта города в избранном)

<https://learn.javascript.ru/constructor-new>

***#92 Дата***

Кто-то из вас уже использовал класс Data в задаче с погодой.

Сегодня стоит ближе познакомиться с тем, как работать с датой в JS

Этот класс содержит достаточно много самого разного функционала, запоминать или учить наизусть которой, не имеет никакого смысла

Ознакомтись с Датой и порефакторите ваш код (если придется)

<https://learn.javascript.ru/date>

***#93 Отладка кода***

Прямо сейчас почти все вы отлаживаете свой код с помощью консоли или даже верстки.

Есть куда более продвинутый способ, с рядом преимуществ. Попробуйте обойтись без console.log в ближайшие пару недель

<https://learn.javascript.ru/debugging-chrome>

***#99 Практика***

Вы уже познакомились с рекурсией и, возможно, даже решили пару задач.

Я предлагаю сделать вот что:

Возьмите программы которые у вас уже есть и попробуйте переписать как можно больше циклов на рекурсию!

Эта задача прокачает вас очень мощно

***#100 Динамичные импорты***

Наверняка вы уже думали о том, как круто было бы загрузить другой JS модуль уже после запуска вашей программы. Например, когда этот модуль на самом деле нужен.

Такая возможность есть!

Сделайте как минимум 1 динамичный модуль в любой из ваших программ (лучше в “Погоде”)

<https://learn.javascript.ru/modules-dynamic-imports>

***#102 npm***

Сегодня вы начнете открывать для себя целую вселенную 🙂

Только пока не спешите что-то устанавливать, просто прочитайте

https://www.hostinger.ru/rukovodstva/chto-takoe-npm

Для практики, создайте новый проект с помощью npm init и перенесите в него код вашего todo листа. Уже завтра мы начнем добавлять сторонние пакеты

***#103 Воспользуемся чужим кодом***

Вчера вы узнали про npm

Сегодня пора начать использовать его возможности и попробовать 1 из очень популярных библиотек.

date-fns - библиотека для работы с датой. Содержит огромное кол-во готовых функций на самые разные случаи. Но начнем с простого

установите эту библиотеку в ваш проект (todo или погода) и поработайте с датой с помощью метода format

p.s. документация часто бывает только на английском языке, если у вас с ним проблемы - используйте переводчик

<https://www.npmjs.com/package/date-fns>

***#103.2 Сборщик***

Если у вас ничего не работает - поздравляю, вы все сделали правильно.

Дело в том, что такой код можно легко запустить в node.JS но нельзя запустить в браузере. Для этого нам потребуется простой сборщик (со встроенным сервером, кстати)

Пока мы не будем уходить в дебри и что-то настраивать, нам хватит пары команд для тестов и запуска нашего приложения

1. Установите пакет Parcel

npm install --save-dev parcel

2. Убедитесь что подключаете модуль

<script type="module" src="index.js"></script>

3. Запустите вашу сборку в режиме разработки

npx parcel index.html

(возможно у вас другой путь до index.html)

4. Откройте в браузере

Смело идите по адресу, который указан у вас в терминале

Server running at http://localhost:1234

p.s. импорт должен выглядеть так

import { format } from 'date-fns';

***#104 ESLint***

Тут будут аж две задачи в одной.

1. Установить ESLint, настроить пару правил в нем

2. Сделать это самостоятельно

<https://eslint.org/docs/user-guide/getting-started>

***#105 Печеньки***

Куки - это еще одна возможность хранить состояние в браузере.

В отличие от localStorage у кук есть несколько серьезных преимуществ (и пара ограничений)

Изучите эту тему и обсудить в чате разницу между этими двумя способами

https://learn.javascript.ru/cookie

p.s. ой да, там будет много слова про HTTP - не пугайтесь, если что не ясно, мы к нему скоро вернемся

***#105.2 Практика***

Сохраните в куках последний выбранный для погоды город (сейчас он хранится в localStorage) и установите “время жизни”, пусть этот выбранный город стирается через 1 час (для тестов время можно изменить)

***#105.3 Бонус для крутышей***

Попробуйте библиотеку js-cookie

<https://www.npmjs.com/package/js-cookie>

***#106 Countdown***

Напишите простую программу (с браузерным UI), которая будет принимать дату и отсчитывать кол-во лет, дней и часов до этой даты.

Дизайн на вашей совести, главное чтобы был 1 инпут и 1 вывод в виде текста

Можно и нужно использовать сторонние библиотеки, лучшие наши практики и вообще все, что вы прошли в этом году

***#107 ? ??***

Посмотрим на “новые” операторы и конструкции в JS

Их не так много, но они позволяют выражать свои мысли намного лаконичнее и чище.

Прочитав статьи, перепишите ваши программы с применением новых подходов

https://learn.javascript.ru/nullish-coalescing-operator

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Optional_chaining>

***#112 Привязка***

Теперь вы уже точно не потеряете ваш this!

Тема это сложная, не пытайтесь понять ее целиком всего за 1 день, вряд ли получится за 3-4 часа. Мы будем возвращаться к подобным конструкциям еще несколько раз.

Так что, не пытайтесь решить вообще все задачи, лучше потратьте это время на второе и третье перечитывание теории

<https://learn.javascript.ru/call-apply-decorators>

***# 113 Практика***

Напишите класс Storage который будет создавать экземпляры для работы с localStorage

Пример:

const names = new Storage(’names’);

names.get() - возвращает значение для ключа names в localStorage;

names.set() - устанавливает значение для ключа names в localStorage;

names.clear() - очищает значение для ключа names в localStorage;

names.isEmpty() - вернет true если ключ names в localStorage имеет пустое значение (null || undefind);

Создайте несколько экземпляров класса Storage и убедитесь что все они работают правильно

для класса Storage добавьте пару опций в конструктор:

1. чтобы можно было выбирать local или session storage

2. возможность указать значение по-умолчанию (при создании экземпляра)

***#124 Верстаем чат***

Верстаем только 2 экрана (2.0 и 2.3)

https://www.figma.com/file/mTbJNfD8AhbVmIX4ZxmPOb/Untitled?node-id=0%3A1

И не забываем про теорию

<https://learn.javascript.ru/onscroll>

***#125 Сообщения***

Отправьте сообщение в чат используя тэг template

https://learn.javascript.ru/template-element

Сообщение из инпута, отправляться оно должно по клавише Enter или по клику на иконку справа

***#126 Авторизация***

Сверстаем еще 2 попапа - 2.1 и 2.2

Обратите внимание, ссылка на макет изменилась

https://www.figma.com/file/DRJ30lFzYuux0Y16rOtfVP/Untitled-(Copy)?node-id=0%3A1

После верстки вам нужно отправлять POST запрос на этот адрес

https://chat1-341409.oa.r.appspot.com/api/user

в теле запроса должен быть JSON с вашей почтой

{ email: ‘my@eamil.com’ }

Если все хорошо - вам на почту придет письмо от ch4tservice@gmail.com

p.s. скорее всего это письмо попадет в спам (проверьте ваши папки)

p.s.s. пожалуйста, не отправляйте 999 писем в секунду

На почту вам придет код, а что с ним делать я расскажу завтра

***# Смена имени / sizes and scroll***

***#127 Изменим имя***

Давайте поменяем имя для отображения в чате.

Вчера вы начали получать коды на почту.

Следующий шаг, при вводе этого кода в окно, сохраните его в куки (можно использовать библиотеки) и из кук прикладывайте в заголовки к каждому запросу

Ожидается вот такой заголовок

Authorization: `Bearer ${token}`

Точно все ясно?)

это эндпоинт с методом PATCH

https://chat1-341409.oa.r.appspot.com/api/user

в теле отправить { name: ‘new-name’ }

так же можете попробовать запрос

https://chat1-341409.oa.r.appspot.com/api/user/me

чтобы получить данные о своем пользователе

И, конечно же, немного скучной теории

<https://learn.javascript.ru/size-and-scroll>

***#130 История***

Загрузите историю сообщений (пока только двух) и отобразите их в окне чата

https://chat1-341409.oa.r.appspot.com/api/messages/

метод GET

***#131.2 Коннект и отправка сообщения***

Теперь, когда вы уже эксперты в области веб сокетов, настало время закконектиться к серверу и послать ему пару сообщений.

const socket = new WebSocket(`ws://chat1-341409.oa.r.appspot.com/websockets?TOKEN`);

Это простая строка коннекта с сервером (вместо слова токен подставьте ваш реальный токен)

Как только коннект будет установлен, отправьте сообщение

socket.send(JSON.stringify({

text: ‘тестовый тест’,

}));

Если вы все сделали правильно, сервер отправит это сообщение всем клиентам (в том числе и вам)

Чтобы принять это сообщение - слушайте событие onmessage

socket.onmessage = function(event) {

console.log(event.data);

};

Если у вас получилось и это - свяжите эти шаги с вашим UI

1. Добавляйте полученное сообщение в чат

2. Отправляйте сообщение из вашей строки ввода

p.s. бонус для самых смелых и умелых

в каждом полученном сообщение вы увидите email пользователя, который это отправил - если этот email совпадает с вашим - считайте что это ваше сообщение и помечайте его в чате соответствующим образом

***#132 Погружение и Всплытие! А так же, делегирование***

Немного скучной теории вам

https://learn.javascript.ru/bubbling-and-capturing

https://learn.javascript.ru/event-delegation

Ну и чат доделать бы.

Верните загрузку истории сообщений (если вы ее отключали) и будьте готовы, сегодня днем эта история начнет пополняться

***#133 Не все и сразу***

Сейчас вы получаете всю историю сообщений чата и добавляете ее в ваше представление целиком.

Пора сделать “подгрузку” по скролу

Это будет ради практики, не ради оптимизаций

Алгоритм такой:

1. Получаете всю историю сообщений (записываете в переменную или localStorage)

2. Вешаете обработчик на событие скролла

3. Вызываете обработчик (когда скролл “ударяется в верхнюю границу окна) и добавляете в чат еще 20 сообщений.

4. Когда добавлять уже нечего - выводите “Вся история загружена”

Это действительно сложная задача, на нее может уйти целый день.

Не забывайте задавать вопросы и много гуглить 🙂

***#137.2 Cоздаем пустое React приложение***

Для этого нужна всего одна команда

npm create vite@latest my-react-app --template react

а здесь можно пссмотреть результат этой команды и сравнить со своим

<https://stackblitz.com/edit/vitejs-vite-h13khf?file=index.html&terminal=dev>

***Там, кстати, сервер с чатом отвалился.***

***Надеюсь его все успели сделать, забыть и перейти к новым темам 🙂***

***#141 Немного практики***

Возьмите вашу старую программу, где запрашивалось имя и угадывался гендер

Перепишите ее с использованием React

Пусть у вас будет 3 разных компонента

Инпут, Кнопка и Вывод

***#142 Рендеринг с условиями***

Это те же самые ‘if’ и ‘switch’, а так же тернарный оператор, только уже внутри рендер функции.

https://ru.reactjs.org/docs/conditional-rendering.html

Добавьте в вашу программу несколько условий.

Например, показывайте ошибку под инпутом, если введено меньше 2 символов

***#144 Списки и Формы***

Очень часто нам нужно рендерить несколько похожих элементов (списки) и мы это уже несколько раз делали (туду, погода)

В React это можно делать еще проще. Главное не забывать про некоторые особенности

https://ru.reactjs.org/docs/lists-and-keys.html

В работе с формами тоже есть особенности.

Если в вашей, недавно переписанной программе, все еще нет “формы” - стоит ее туда добавить

<https://ru.reactjs.org/docs/forms.html>

***#145 Подъем состояния***

Кто-то из вас уже воспользовался этой практикой сам того не замечая. Пора научиться делать это осознанно

https://ru.reactjs.org/docs/lifting-state-up.html

p.s. Можно применить это в вашей, уже написанной, программе. А так же исправить другие “ошибки” что я озвучивал на код-ревью

***#148 Стейт***

Вы уже познакомились с этим хуком из далека.

Теперь пора изучить его подробно.

https://ru.reactjs.org/docs/hooks-state.html

Напишите небольшое приложение в котором будете использовать useState

***#149 useEffect***

А теперь и useEffect

https://ru.reactjs.org/docs/hooks-effect.html

Но чтобы уж совсем просто не было, давайте перепишем TODO на React.

Знаю, звучит слишком, но по факту у вас уже и логика готова и верстка.

p.s. Не забудьте разделить на компоненты

***#152 Снова перепишем***

Перепишем погоду на React.

Да, этого будет не так интересно как писать что-то новое, но давайте сначала хорошо разберемся со старым.

Не будем тратить время на верстку и JS (что у вас уже готово), сфокусируемся на практике разделения на компоненты и выборе правильных схем движения данных между этими компонентами.