**Що потрібно знати для роботи веб програмістом:**

**1. Як працює Web:**

• Все про API: «Application Programming Interface» (интерфейс программирования приложений). API — это составляющая часть сервера, которая получает запросы и отправляет ответы. Пользователь благодаря API получает возможность совершить действие не покидая сайт.

Архитектура API обычно объясняется с точки зрения клиента и сервера. Приложение, отправляющее запрос, называется клиентом, а приложение, отправляющее ответ, называется сервером. Итак, в примере с погодой база данных службы – это сервер, а мобильное приложение – это клиент.

XMLHttpRequest – это объект, который даёт возможность делать HTTP-запросы к серверу без перезагрузки страницы. Этот метод уже не использ. так как он устарел (вместо него метод fetch).

Rest API - определяет набор функций, таких как GET, POST, PUT, PATCH, DELETE, которые клиенты могут использовать для доступа к данным сервера. Клиенты и серверы обмениваются данными по протоколу HTTP.

GraphQL – разработанный специально для API. Такой язык разработан, чтобы сделать API быстрыми, гибкими и удобными для разработчиков. В качестве альтернативы REST GraphQL дает разработчикам интерфейсов возможность запрашивать несколько баз данных, микросервисов и API с помощью одной конечной точки GraphQL. Организации предпочитают создавать API с помощью GraphQL, потому что это помогает им быстрее разрабатывать приложения.

**Итог**: веб приложение работает по принципу клиент-сервер через http-запросы. Ми відправляємо запит в браузер (fetch()), той в свою чергу відправляє запит на сервер (http-запит), потім сервер вертає відповідь (http-відповідь). Якщо, ми працюємо з токєном (унікальним ключом), то його потрібно вказати в заголовкі hraders: { ‘’API-Key”: “9124234324322432”, 'Content-Type': 'application/json' }.

• Все про доменне імя: имя домена – должно быть уникальное, например: info.nic.ua., а 194.85.61.86 – IP адреса.

• Все про DNS-сервера і як вони працюють: **DNS-сервер** - це спеціальний комп’ютер, який зберігає IP-адреси сайтів. Основні функції сервера DNS — видавати браузеру адресу сайту за доменним іменем (google.com) і кешувати DNS-записи домену. Принцип работы: запрос (www.google.com) -> NS (name server) -> NS server for .com -> ns1 -> 192.02.44 -> веб страничка.

Принцип роботи браузера: коли ми вводимо url-адресу, на сервер посилається get-запрос і після обробки цього запросу сервер нам вертає html сторінку, після чого у нас рендериться DOM-дерево, приміняються стилі(CSS), підключається JS(обробники подій і тд.), а якщо на більш нижньому рівні, то цей запрос який ми ввели в строкі обробляється DNS-сервером який в свою чергу звірює url з api і після відправляє нам.

• Все про HTTP, HTTPS, HTTP2.0: **HTTP(**Hyper Text Transfer Protocol**)** — это клиент-серверный протокол, позволяющий получать различные ресурсы, например HTML-документы. Протокол HTTP лежит в основе обмена данными между клиент-сервер.

HTTPS(Security) – значит защищенный протокол. Для этого протокола нужен SSL-сертификат. HTTP запит склад. з трьох основних частин: строки запиту(метод передачі(get, post, put, delete), url-адресу), заголовка(тіло повідомлення, різні параметри та іншу інф-ю), тіла запиту(дані, які передаються в запиті – не обовязковий параметр).

• В чому різниця між аутентифікацією та авторизацією: **аутентифікація** – це підтвердження того, ким є користувач *на вході*. Це проходження перевірки автентичності.

**Авторизація** – це те, що користувачу дозволяється робити *після* в*ходу*. Це надання і перевірка прав на вчинення будь-яких дій в системі.

WebSocket – це протокол, який призначений для взаємодії групи програмістів в реальному часі. За допом. нього є можливість обміну даними між браузером та сервером через постійне зєднання. Він не являється http-запитом. Він потрібен для сервісів, де постійно онлайн передаються якісь дані.

CORS – потрібен для захисту інтернета, це спільне використання ресурсів між різними істочніками.

Interface– то, что мы видим снаружи.

Программирование- это алгоритм работы с данными. В javaScript window – це глобальний обєкт.

Докладний розбір URL-строки https://site.com:8080/path/page?javascript#hash:

- href – це повний url-адрес (https://site.com:8080/path/page?javascript#hash)

- protocol - це https:

- hostname – це site.com

- port – це 8080

- pathname – це /path/page

- search – це строка запиту, яка повинна починатися з “?” (javascript)

- hash – це #hash

**2. HTML:**

• БЕМ(блок елемент модифікатор): <div class=”block\_\_element--modeficator”></div> - ця методологія потрібна для компонентной розробки, тобто, розділення інтерфейса на незалежні блоки.

• Семантика: вона потрібна для програм-ботів, котрі візуально не можуть відрізнити наприклад header від footer. <header>, <footer>, <nav>, <main>, <aside>, <section>, <article> і тд.

**3. CSS:**

• Препроцесори: Основная задача препроцессора — это предоставление удобных синтаксических конструкций для разработчика, чтобы упростить, и тем самым, ускорить разработку и поддержу стилей в проектах: Less, SASS(SCSS).

• Зборка Gulp: потрібен для склеювання всіх файлів в один і цей один файл буде віддавати запит на сервер, а не 100 файлів які ми створимо, тому наш сайт буде бистрим та оптимізованим. Gulp, WebPack – це інтерфейс для підключення та настройки різних плагінів (упаковщик модулей), які виконують різні операції з файлами, наприклад: зжимання картинок, обєднання різних створених файлі в один, автоматичне тестування, online слідкування за змінами в браузері, компіляція одної мови в іншу: LESS -> CSS, TypeScript -> JavaScript, ES6 -> ES5.

• Адаптив: потрібен для відображення сайта на різних пристроях.

• Кроссбраузерність: потрібна для відображення сайта на різних браузерах таки чином, щоб цей сайт не зламався. Це можна зробити на онлайн сайтах: browsershots.org, browserling.com, crossbrowsertesting.com і тд.

При display: inline – змінювати ширину та висоту не можна, а при inline-block – ми можемо міняти ширину та висоту.

position: sticky відрізняється від fixed тим, що sticky позиціонується в рамках конкретного блоку, а не всього документа як fixed.

border-box – займає фіксовану ширину, яку ми виставили, а content-box – на всю ширину контента.

**4. Продуктивність (performance):**

• - мінімізація ресурсів.

- рефакторінг(скорочення кода) - це коли ми залишаємо робочу поведінку нашої програми однаковою, але всередині ми міняємо код.

- зменшення ваги картинок і тд.

**5. Java Script:**

• Для теорії "Современный учебник JavaScript".

• Багато практики!!!

• Вивчення додаткових фрейморків(React, Redux, Angular, Vue) та бібліотек(jQuery, Bootstrap, Ant Design, Material UI).

**Горячии клавиши и чистота кода в VS Code:**

Ctrl + Shift + p – поиск.

Ctrl + z - откат назад.

Ctrl + p - поиск файла или компоненты которую мы недавно использовали.

Ctrl + f - для поиска переименновки чего либо.

Ctrl + w - для быстрого закрытия приложения.

Ctrl + tab - для перекл. между открытыми вкладками.

Программа quokka позволяет смотреть данные без консоли: уст. quokka -> ctrl + shift + p -> запускаем на нашем файле или создаем новый.

Linter(для JS это ESLint) – этот утилит помогает приводить код к единому стилю и он может находить синтаксические ошибки, баги или неточности форматирования.

Prettier - это утилит для форматирования кода, который приводит весь код в один единый формат и стиль, одним словом, делает код чистым.

Чтобы зарепозиторить проект на gitHub нужно в VSCode -> Ctrl + Shift + p -> вписать gitHub -> выбрать 'privat' или 'public' и добавить проект.

**Java Script:**

**Алгоритмы:**

Алгоритм - это набор последовательных действий, который решает какую-то задачу.

Логарифм - это ф-я обратно возведена в степень(log2n).

Бинарный алгоритм работает только с отсортированным списком.

n - кол. элементов в массиве.

scope – це область видимості перемінних: глобальна, блочна, ф-на.

Рекурсия - это ф-я которая вызывает сама себя. Она должна всегда иметь какое-то условие или случай при котором вызов прекращается иначе будет переполнения вызовов, поскольку ф-я будет вызывать сама себя бесконечно.

Число Фибоначчи: [1, 1, 2, 3, 5, 8, 13, 21 и тд...] - каждое пред. число суммируется.

Графы - это когда нам нужно попасть с т. A в т. G по самому короткому пути.

Очередь - это структура данных состоящая из каких-то элем. Основной принцип закл. в том, что элем. всегда доб. в конец структуры, а извлекаются из его начала.

Стек - это структура данных состоящая из каких-то элем. Основной принцип закл. в том, что элем. всегда доб. в конец и извлекается так же с конца.

**Структура данных стека** выполняет две основные операции:

push() – додавання елем. в кінець масиву.

pop() - видалення елем. з кінця масиву.

unshift() – додавання елем. на початок масиву.

shift() - видалення елем. з початку масиву.

Деревья - это рекурсивная структура данных, где каждый узел является также деревом и поддеревом.

Алгоритм кеширование каких-то данных нужен для того, чтобы избежать повторных вычислений каких-либо ф-й.

**Полифилы:**

Babel – это транспиллер. Он переписывает современный JavaScript-код в предыдущий стандарт, нужны для заполнения пробелов и добавления современных ф-й в код.

Щоб його втановити: **npm I –D @babel/preset-react @babel/core @babel/cli**

**npm i @babel/preset-typescript**

**Структуры данных:**

Массив - последовательный набор каких-то обьектов. Массив подходит тогда, когда часто приходится изменять размер массива и часто обращаться к данным.

Связный список - подходит тогда, когда мы редко обращаемся к каким-то данным и часто его дополняем. Cуть такова: каждый предыдущий элем. ссылается на последующий.

Бинарное дерево поиска - это структура данных, где каждый узел - дерево, но у каждого узла может быть только 2 потомка(child). Правая часть со значениями выстраивается с большими значениями, а левая с меньшими, пример:

5

3 7

1 5 4 9

“==” – не строге порівняння (порівнює тільки по типу), а “===” – строге порівняння (порівнює по типу і по значенню).

Перемінні a = 5, var = 5– мають ф-ну область видимості але не мають блочну, let, const – мають блочну область видимості.

Типы данных: number, string, boolean(true / false), object, null, symbol(“id”), undefined, bigInt.

- object – cылочный тип даних.

- symbol – примитивный тип данных, использующийся для создания уникальных идентификаторов.

- null - значение которое представляет собой «ничего», «пусто» или «значение неизвестно».

- undefined - оно означает, что «значение не было присвоено». Обычно null используется для присвоения переменной «пустого» или «неизвестного» значения, а undefined – для проверок, была ли переменная назначена или нет.

- bigInt - числовой тип, который предоставляет возможность работать с целыми числами большой длины, чтобы создать значение типа bigInt, необходимо добавить n в конец числа. (let bigInt = 9007199254740991n то же, что и let alsoBigInt = BigInt(9007199254740991).

- NaN (Not-A-Number) - означает вычислительную ошибку.

**Методи для перевірки типів даних:**

- Instanceof() (для обєктів) – оператор, який перевіряє чи належить обєкт указаному класу, з урахування наслідуванням, поверне true or false.

- typeof() (для примітивів) – перевіряє тип даних та повертає строку.

- toString() (для примітивів) – вертає масив з обєктом та тип даних, який ми задали:

let arr = Object.prototype.toString;

arr.call([]) -> [Object, Array];

- parseInt() (для примітивів) – вертає число з строки.

**Метод** - это свойство объекта, значение которого - ф-я:

- forEach((item, index, array) => {}) – цей метод нічого не вертає, він просто перебирає в масиві елементи (як цикл for).

- filter(item => {}) – завжди вертає новий масив в який ввійдуть тільки ті елем. для яких callback верне true.

- find(item => {}) – вертає колбек ф-ю, яка в свою чергу приймає якусь умову, а потім find верне перший елемент при якій ця умова стала true.

- bind – приймає параметром контекст ф-ї, після вертає нову ф-ю з привязаним контекстом в яку ми передали параметром. Теж саме роблять і методи call(повертає через кому). і apply(повертає в виді масива). При викор. bind створюється окремий обєкт, який не належить до род. ф-ї чи обєкта.

- map(item => {}) – вертає новий масив та кожний елем. масива.

- some(item => true or false) – перевіряє чи задоволений елемент масива якійсь умові: [2, 5, 8, 1, 4].some(item => item > 10) -> false.

- reduce((acc, item) => {}) – викор. для обробки кожного елемента в масиві та верне єдиний результат цих елем. масива.

- slice(0 – індекс по якому ми видалимо обєкт, 2) – вертає новий масив з вирізаними обєктами.

- concat() – цей метод трансформує багатошарові масиви в один.

- includes(2) – шукає в масиві чи є цифра 2.

- sort((a, b) => a - b) – сортує елем. в масиві (для чисел).

- sort((a, b) => a.localeCompare(b)) – сортує елем. в масиві (для строк).

- split(‘, ‘) – розбиває строку на масив.

- join(‘, ‘) – створює одну строку з масива.

- fill(‘’’’) – заповнює всі елементи масива від початку до кінця індексів одним значенням.

- Array.isArray(arr) – перевіряєм чи являється arr масивом.

**Методи Class components:**

- constructor(props) – описує параметри нашого класа.

- constructor(super(option)) – для передачі параметрів з род. конструктора в наслідувальний клас.

- super.method() – потрібен для виклику родітільського метода в наслідувальному(extends) класі.

- static – метод, який присвоюється полям та методам конкретного класа для більше ф-ї роботи класу, тобто ми тепер можемо звертатися до цього конкретного класа та викликати той статичний метод, який ми зазначили: class Person { **static** calculate() {……} } -> let res = Person.calculate(…..).

Модефікатор – дозволяє сховати стан від зовнішнього доступу користувача. Існує 3 види мод.:

- public (по дефолту) : name: string / **public** name: string – поле name можна буде викор. за межами стану чи класу, де він створився.

- protected: protected name: string – опреділя поля та методи, які можна викор. тільки в наслідувательних класах.

- private: **private \_**name: string – поле name тепер не можна викор. за межами стану чи класу, де він створився.

Примеси – это класс у которого методы предназначены для использ. в других классах.

Key(ключь) - он должен быть статичным и уникальным. Ключи нужны для того, чтобы перерисовывать не весь список, а только те элементы в которых произошли изменения.

Замикання – це ф-я, яка викликана всередині іншої ф-ї, в свою чергу вложена ф-я буде зберігати та викликати всередині себе всі змінні, які знаходяться в родітільській ф-ї.

Декоратор – це ф-я, яка приймає род. ф-ю і вона ж може змінювати поведінку род. ф-ї.

**Function Expression -**  **создаётся, когда выполнение доходит до него, и затем уже может использоваться:** let sum = function(a, b) return a + b.

**Function Declaration - может быть вызвана раньше, чем она объявлена:** function sum(a, b) return a + b.

Різниця між стрілочними ф-ми та методом bind: стрілочна ф-я не має this та у неї немає arguments.

Prototype – це обєкт або null. За доп. нього можна до обєкта додавати різні ф-ї, методи, значення в прототип не змінюючи старий обєкт. Цикл **for in** проходить не тільки по свойствам головного обєкта, а і по прототипу. Але якщо ми хочемо пройти тільки по головному або прототипі обєкту, то є метод hasOwnProperty().

\_\_proto\_\_ - це силка на прототип родітіля, ЗАСТАРІЛИЙ МЕТОД!!! Замість нього краще викор. такі методи:

- Object.create(obj, desc) – создаёт пустой объект со свойством [[Prototype]], указанным как proto. Другим аргументом ми можем добавляти інші параметри.

- Object.getPrototypeOf(obj) – возвращает свойство [[Prototype]] объекта obj.

- Object.setPrototypeOf(obj, proto) – устанавливает свойство [[Prototype]] объекта obj как proto.

Самовикликаюча ф-я – це ф-я у якої немає назви: (function() {} )(). Потрібна для створення перемінних, щоб не забруднювати навколишнє середовище перемінними.

Шаблонні літерали(косі скобки) – потрібні, щоб в них вставляти js обєкти.

new Map – це колекція (ключ/значення) і він може викор. ключ будь-якого типу.

new WeakMap – відрізняється від new Map тим, що він повинен зберігати ключі тільки в виді обєктів, а не примітиви.

new Set – це структура даних або ж масив (значення, без ключів), де кожне значення може з’явитися тільки один раз, тобто, він зрівнює значення примитивів та вертає їх в одиночному екземплярі.

new WeakSet – та сама структура даних або ж масив, як і **new Set** тільки він повинен зберігати в собі не примітиви, а обєкт.

Псевдомасив (let arrayLike = { 0: "Hello", 1: "World", length: 2}) – це обєкт у якого є ключі та значення і значення “length”. Щоб перетворити псевдомасив в масив потрібно викор. конструкцію Array.from(arrayLike) -> ["Hello", "World"].

Метод Array.from() або ж spread operator(…) - принимает объект, проверяет, является ли он итерируемым объектом или псевдомассивом, затем создаёт новый массив и копирует туда все элементы.

Spread(…) оператор або ж оператор розширення – цей оператор завжди вертає масив.

Існує 2 способи виявити свойство(ключ) в обєкті, наприклад в let obj = {a: 5}:

- obj.hasOwnProperty(“a”);

- “a” in obj;

Синхронність – це коли код може запускатися та виконуватися одночасно.

Асинхронність – це строге послідовне виконання коду.

Callback (функція зворотнього виклику) - це функція, передана в іншу функцію в якості аргумента, яка потім викликається по завершенню будь-якої дії. Альтернатива callback ф-й (які викликаються один за одной(цепочкой)), ми можемо використати then(() => {}) в promise.

Promise – це обєкт, який призначений для роботи з асинхронним кодом, також він має свої свойства: state -> pending -> fulfilled or rejected. Обіцянка, яка говорить про те, що колись асинхронна операція закінчиться і потім можна буде через проміс добратися до результату.

**Методи:**

- Promise.all(array) – потрібен для загрузки та відображення файлів паралельно (синхронно). Якщо буде помилка, процес зупиняється.

- Promise.allSettled(array) – метод при якому результат промісів буде завжди, навіть якщо десь буде помилка. (Нововедення, працює не у всіх браузерах!!!)

- Promise.race(array) – вертає з усіх промісів тільки перший, навіть, якщо він буде помилкою.

- Promise.ane(array) – вертає з усіх промісів тільки перший успішний проміс.

- Promise.resolve(value) / reject(error) – вже не виконується в сучасному коді (замість них прийшов async / await).

Глибоке копіювання обєкта: const cope = JSON.parse(JSON.stringify(obj)) так як обєкт – копіюється та присвоюється по силкі.

Поверхнє копіювання обєкта – Object.assign() або spread оператор: let copy = {…obj}.

this – викор. в обєктах, щоб достать якусь інф-ю в цьому обєкті, тобто, якщо у нас є обєкт: let obj = {name: ‘Tom’} то ми можемо тепер звернутися до ключа(name) так: this.name – це буде теж саме, що і obj.name. В стрілочних ф-х контекст this відсутній.

||(або) – верне перше значення true;

&&(і) – верне перше значення false;

Якщо застосовується 2 оператора відразу: || && - то приоритет && більший чим в ||.

Деструктуризація: let obj = {name: ‘Vlad’, age: 29}; -> let {name, age} = obj;

Тернарний оператор – теж саме що if else -> let result = умова ? true : false.

event.preventDefault() – потрібен для припинення різних дій з слухателями подій по умолчанню.

event.stopImmediatePropagation() - не только предотвращает всплытие, но и останавливает обработку событий на текущем элементе.

event.stopPropagftion() – потрібен для того, щоб не було всплитій подій.

Делегування подій – викор. при роботі з DOM-деревом, коли ми обработчик подій вішаєм не на кожний елемент, а тільки на родітільський. По ньому ми можемо отримати цільовий елемент event.target – за доп. нього ми можемо дізнатися на якому елем. в родітілі відбулася подія та відпрацювати цю подію, наприклад, на форму: <form onsubmit={funcClick}>.

Всплытие – это когда все обработчики события срабатывают от ребенка к родителю.

try {код} catch(err) {виповниться код, якщо в блоці try буде помилка} finally {виповниться код в будь-якому випадку} – для відлову помилок. Якщо синтаксично js код згенерований правильно, та нічого в блок catch не попадає, то ми можемо відловити помилку в блоці try за доп. оператора “throw” – який в свою чергу може викинути нову помилку за доп. перевірки if: try { if(!user.name) throw new Error(“error”) }.

Головні 3 метода відлова помилок в асинхронних ф-ях:

- async function f() { await Promise.reject(new Error("Упс!")) }.

- async function f() { await throw new Error("Упс!") }.

- async function f() { try{ code } catch(err) {console.log("Упс!") }.

window.onerror = function(message, url, line, error) {…} – потрібен для того, щоб знайти в коді необроблену глобальну системну помилку.

DOM-дерево – це html документ з усіма його блоками(дітьми), через нього ми можемо, за допомогою скриптів, отримувати та змінювати структуру і оформлення тих документі, які там знаходяться.

Основні атрибути DOM-дерева:

- getAttribute(‘type’) – вертає значення указаного атрибута: <input type=”text” />.

- setAttribute(‘type’, ‘text’) – добавляє указаний атрибут та його значення до елементу.

- hasAttribute(‘type’) – вертає **true** or **false** при наявності в елемента указаного атрибута.

- removeAttribute(‘type’) – видаляє вказаний атрибут.

LocalStorage - локальное хранилище (в Nodejs нету) только в браузерном API, и он умеет работать только со строками. С помощью его можно хранить 5мб. данных для домена, больше чем в 'cookie' и данные не улетают на сервер (HTTP).

В localStorageможно сохранять только строки, например, не true а 'true'. localStorageотличается от сookieтем, что он сохранет данные даже если польз. вышел с сайта.

Session storage – временное локальное хранилище, которое удаляется после закрытия браузера.

Cookie - это файл который хранится в браузере для каждого сайта и который хранит данные о пользователе.

Event loop(цикл подій) – це механізм який вирішує проблему з потоком дій. Взагалі, JS на початку був однопоточним, поки не з’явилися асинхронні дії(promise, setTimeout), тобто, ці дії потрібно виконувати так, щоб не блокувати інтерфейс користувача, наприклад, коли в stack викликів попадають асинхронні ф-ї в поточному порядку, то для них є 2 черги(macro and micro tasks, де спершу виконуються micro tasks), тобто, звичайні ф-ї потрапляють в одну чергу, а асинхронні ф-ї в іншу.

Console.log(“Start”) – виконається першим;

setTimeout(() => { console.log(“Event”) }, 0) – виконується останнім;

Console.log(“Finish”) – виконається другим;

**Як це буде працювати**: перший консоль попадає в стек викликів, потім другий консоль(так як тут виконується однопоточний виклик), а потім таймаут, тому що цей метод викликається як асинхронний потік і тому він потрапляє в web api(по дефолту цю функціональність надає наш браузер).

**Як це працює:** спочатку з цієї черги беруться слухачі подій(setTimeout) -> виконується ця задача, після цикл проходить по всім micro tasks(promise) -> виконує всі їх та переходить до наступної подій.

Proxy - это класс, который позволяет создавать различные ловушки для обьектов, ф-й, классов.

let proxy = new Proxy(object, { ловушка } ).

Proxy != оригин. обьекту и не перехватывает проверку на строгое равенство (===).

Карірування – потрібне для роботи з ф-ми, трансформує ф-ю з func(a, b, c) -> func(a)(b)(c) для часткового вивода результатів ф-ї.

Ссылочный тип - исключительно внутренний тип языка, он нужен чтобы передать информацию от точки “.” до вызывающих скобок “()”.

Юнікод – console.log( "\xA9" ); // ©, символ авторского права.

**React:**

React- это функциональное програмированние.

Фреймворк - это набор библиотек и инструментов.

Библиотека - это просто програмный модуль на определенном языке. Грубо говоря библиотека это набор готовых решений, которы вы можете просто брать и использовать. Фреймворк же накладывает опреденный стиль программирования, его элементы сильнее связаны между собой.

В React и других быблиотек самое главное делать декомпозицию - разделение целого кода(комп.) на части компонент(конструктор)!!!

Модуль – це просто файл (1 скрипт – 1 модуль). Тобто, якщо ми до якоїсь ф-ї додамо **export**, то це вже буде рахуватися модулем. Модулі прокидуються через **import / export.**

Якщо ми викор. import / export { say as hi }, то тепер нам потрібно звертатитися не до say, а до hi: hi(“Hello”) // Hello.

SPA (single page application) - одностраничное приложение (полупустая страничка, где приходит мало html и много javaScript и этот JS делает динамическую разметку) по такому примеру работаю многие библиотеки в том числе и **React**.

С чего лучше писать код: UI(компоненты), BLL(Redux), DAL(API), API(back-end): BLL -> UI -> DAL(API).

Детерминацияона же(Идемпоте́нтность) - свойство объекта или операции при повторном применении операции к объекту давать тот же результат что и при первом (например когда мы вызываем один и тот же обьект с данными хоть 100 раз, то мы должны получать один и тот же результат).

Чистая функция(pure function), reducer: она должна придержыватся иммутабельности, она не должна изменять то, что в неё пришло(props), она должна что-то возр.(return), не оставлять какого-то мусора во внешнем мире(no side effect) и должно быть детерминированной (предсказуемость) или идемпотентной (get-запросы) - это значит, что сколько не пускай мне на вход одни и те же данные я тебе всегда верну один и тот же ответ. Она должна работать только с тем, что к ней приходит из-вне внутри props.

VirtualDOM - находится в React елементе, поетому мы напрямую к DOM не обращаемся. React каждый раз создает VirtualDOM, сравневает новый DOM со старым и перерисовует только то, что изменилось.

Компонента(например App)- это ф-я которая возвр. разметку JSX, это один тег(<App />) и принимает в себя некий параметр props.

Тупая компонента- только запускает ф-ю которая в неё пришла.

Функциональная компонента- это компонента без состояния, презентационная. Комп. может вернуть другой JSX если поменялся локальный state либо пришли другие props.

JSX- это js розширение. JSX не попадает в браузер, а JS попадает благодаря **Babel** (транспилятор) который преврящает (транспилирует) JSX в JavaScript.

props- это некий обьект который попадает в компоненту (ф-ю) с некими параметрами и его не возможно изменить. Он нужен для того, чтобы передать в компоненту какие-то данные.

UI (он же React) - все то, что мы оживляем в React.

Управляємим комп. – це коли в нього є стан(state) і ми можемо ним управлять(setState).

Не управляємий комп. - коли наприклад на input накладений метод ref через хук useRef().

Життєвий цикл компоненти. (по-етапно):

- initialization (ініціалізація) – state, render.

- mounting (монтування - componentDidMount) – викликається після рендеринга комп. Тут ми виконуємо запроси до різноманітніх ресурсів.

- updation (обновлення – componentDidUpdate, shouldComponentUpdate (вертає true / false), componentWillMount) – викликаються після відновлення комп.

- unmounting (розмонтування – componentWillUnmount, componentDidUnmount) – викликається перед видаленням комп. з DOM-дерева, та відбувається очищення ресурсів даного комп.

**Hooks:**

1. const [state, setState] = useState(null) - предназначен для управления состоянием компонентов. Применяет в себя сост. и ф-ю, которая будет изменять это сост.

2. const pathRef = useRef(null) – цей хук не впливає на ререндер, він зберігає один обєкт на протязі всього життя компоненти, який ми йому можемо передати (якесь початкове значення), а вертає цей хук силку-обєкт (pathRef) та із свойства **current** ми можемо отримати збережене значення.

3. useEffect(() => {}, []) – це заміна життєвих циклів класових комп. Він потрібен для мутацій, побічних ефектів(таймери, логірування та всі інші дії, які ми виконуємо в браузері). Цей хук змінює значення(застосовує ефект) в будь-якому випадку після рендеренга компоненти, але ми можемо заставити цей хук не застосовувати кожного разу ефект, якщо в масив залежностей передамо той стан, який ми не будемо змінювати при кожному рендеренгі. Якщо ми залишимо пустий масив, то рендер відбудеться тільки один раз при відновленні сторінки. Щоб очистити або видалити за доп. цього хука, нам потрібно в ньому вернути ще одну ф-ю: useEffect(() => { код **return () => { очистка коду }** }, []).

Як він працює: коли хук спрацьовує, то першим викликається монтування (mount), а якщо в масиві залежностей змінюється щось, то перше спрацьовує корінна ф-я (useEffect(() => { код **return () => {}** }, [])), а після вже викликається сам **effect**.

4. useLayoutEffect() - аналогичен хуку **useEffect(),** но вызывается синхронно после всех изменений в структуре DOM.

5. const {state, setState} = useContext(MyContext) - позволяет подписываться на контекст React. С пом. него мы можем прокидывать от одного род. комп. ко всем детям сост., ф-и.

6. useReducer() - позволяет управлять локальным состоянием сложных компонентов.

7. useMemo(func, []) – цей хук потрібен для оптимізації проекта (кеширує дані). Приймає вже створену ф-ю та масив залежностей. Він буде вертати результат тільки тоді, коли ті значення, які знаходяться в масиві залежностей, будуть змінюватися.

8. useCallback(() => {}, []) – приймає ф-ю і масив аргументів, та повертає одну і туж ф-ю, до тих пір поки аргументи в масиві не зміняться. Наприклад, щоб рендер кожного разу не викликався при якісь зміні події в DOM-дереві, нам потрібно цю подію обернути в колбек ф-ю, таким чином ми можемо за дом. цього хука керувати рендером (наприклад відображати чи не відображати якусь дію).

Хук **useMemo** відрізняється від **useCallback** тим, що він вертає не саму ф-ю, а результат її виконання.

9. const niqueId = useId() - предназначен для генерации уникального идентификатора (ключа).

10. const [isPending, startTransition] = useTransition() - применяется для создания переходов при рендеринге. Если происходит дорогой рендер, то этот хук сообщает React об этом рендеренге и тогда React приостанавливает этот рендер для более быстрой подгрузки других вычислений, что бы пользователь не заметил большой задержки.

11. const getValue = useDeferredValue(value) – этот хук делает все тоже самое, что и **useTransition(),** только он получает значение через пропсы от родителя.

Все компоненты, что мы пишем на React - это React, а ReactDOM - это наши отрендереные компоненты которые отрисовуются в DOM-дереве.

SideEffect (жизненный цикл (componentDidUpdate, componentDidMount) ) - являются запросы на сервак, асинхр. операции setTimeout, обращение к DOM-элементам на прямую и тд.

Pure Component- это комп. которая реализовала метод shouldComponentUpdate и сделала поверхностное сравнение пришедших данных с тем, что у неё было.

**React-router-dom:**

NavLink відрізняється від Link тим, що він може викор. свій стан силки, наприклад: className={({ isActive }) =>(isActive ? " active" : "")}, а в Link так зробити не можна.

const params = useParams() – вертає набір параметрів маршрута (<Route path=**":id/:name"** />).

const [searchParams, setSearchParams] = useSearchParams() – хук для отримання параметрів в строкі запитів після “?”: …/:id?name=bill&age=30. Щоб отримати параметри, викор. метод searchParams **.get(“name”)**.

**Redux:**

BLL (Business Logic Layer он же Redux) - все наши обьекты, массивы созданные и тд.,

Redux- это библиотека для работы с состояниям приложения, некое хранилище данных. Изначально сост. в Redux является неизменяемым (мы каждый раз должны возр. новый обьект: { ...state }). В **dispatch** передается **action** и то как **action** будет изменять сост. определено внутри **reducer**, а в свою очередь **reducer** - это чистая ф-я которая принимает **state** and **action** и всегда возр. новый обьект состояния.

Slice- это некая обертка для **reducer** которая доб. доп. функционал и упрощает роботу. Теперь нам не нужно создавать вручную типы, action и AC - все это делать автоматически **Redux-toolkit**.

Redux saga (есть 3 основных понятия):

- workers - это ф-я внутри которой выполняется какая-то асинхронная логика.

- watchers(наблюдатель) - это ф-я-генератор которая вызывает **worker** ф-ю, в которой с пом. спец. ф-й мы указываем тип action и worker - который будет отрабатывать когда action с типом (который мы указали будет отрабатывать).

- effects - ф-я которая помогает делать запросы, делать dispatch, следить за workers и тд.

Generator - это ф-я которая помечена звездочкой(\*) и которая возр. нам какие-то данные поэтапно:

function\* generateSequence() {

yield 1; // operator

yield 2;

return 3;

}

Iterator – це перебираємий обєкт, який викор. в циклі for of.

\*[Symbol.iterator] – метод для отримання ітератора, потрібен для перебору значень обєкта.

Є також асинхронні ітератори та генератори: [Symbol.asyncIterator]() {code}.

Reducer- это ф-я которая принимает старый **state** и **action**, где action - это js обьект у которого есть как минимум **type** и потом возвр. измененный новый **state**.

AC (action creator)- занимаются упаковкою экшенов. Это ф-я которая принимает какие-то данные (payload) и возр. нам обьект action.

В store есть три главных метода: getState, dispatch, subscribe.

Middleware (промежуточные слои (ПО) )- это ф-я которая позволяет перехватывать определенные данные и делать логику которою мы здесь опишем.

Thunk (санки)- это ф-я которая делает асинхронную задачу и которая умеет диспатчить (dispatch) обычные action, thunk тоже можно задиспатчить.

AJAX запросы - асинхронные запросы, которые JS может в браузере отправлять на отдаленный сервер и без перезагрузки страницы получать эти данные обратно в браузер.

AJAXзапросс является **side Effect**, что не приемлено для чистой ф-и - это по поводу функ-ной компоненты, а в классовой компоненте можно делать **side Effect** методом componentDidMount().

**Методы AJAX запросов:**

- fetch(url).then((res) => res.json()).then((res) => console.log(res)).catch(error)

- new Promise( (resolve, reject) => {} ).then().catch()

- async await

3 способа для получения данных с сервера:

- response.json() – получаем обьект в json формате

- response.text() – получаем текст

- response.formData() – получаем данные с html формы

- response.clone() – получаем копированный результат

По умолчанию **fetch** делает запросы, используя стандартное HTTP-кеширование.

Ф-я connect- это ф-я которая возращает hoc (high order component - компоненту высшего порядка). Это нужно для создания компонентов-контейнеров на основе базовых компонентов React.

HOC - это ф-я которая принимает на входе компонент, а возращает другой компонент, который мы создали в ф-и hoc (зачастую хоки дают какие-то props или какое-то поведение, типа наша комп. была без ничего(пустая), а теперь с чем-то(with) и для этого для хоков-комп. нужно давать имъя с приставкой with), пример:

const withHOC = (Component) => {

let WrapperContainer = (props) => {

return <Component value={ props.value } />

}

return WrapperContainer;

};

Compose - эта ф-я собирает в кучу все наши обработчики чего-либо (connect, hoc, и тд.). В нативном JavaScript этой ф-и нету, она есть в разных библиотеках (в том числе и redux).

Контекст thisставиться только когда у нас class Component (классовые компоненты) или когда у нас обьекты с методами (**ООП**).

applyMiddleware- ф-я которая принимает промежуточные слои. Чтобы диспатчить санку, мы должны подкл. **middleware**.

**Type Script:**

TypeScript – це строго типизірувана на компілірувана мова. Потрібен для типизації компонентів, щоб більше ефективніше відновлювати помилки. TS в кінці компілірується в JS файли.

**Типи даних:** boolean, number, string, array<>, readonlyarray<> (не дозволяє змінювати значення обєкта), кортежі, enum (перечислення наборів створених нами констант), any (довільний тип), symbol, null, undefined, never (відсутність значення, викор. в якості вертаємого типа ф-ї, які генерують та вертають помилку), bigint.

Оператори (тайп-гарди) typeof and instanceof – перевіряють та розлічають типи, може вертати такі значення: string, number, bigint, boolean, symbol, object, function, undefined.

“?” – не обов’язковий параметр, ставиться після значення та після значень з обов’язковими параметроами:

let number: number = 30,

let string?: string = “string”

Union - тип данних де викор. 2 типа: let id: number | string = 30 and “30”.

"!" - в TS используется для указания, что переменная имеет определенный тип, но может быть "falsy" (то есть иметь значение false, 0, "", null или undefined). Это позволяет избежать ошибок когда переменная имеет ожидаемое значение, но не является "truthy" (то есть имеет значение true, 1, "string" или {} и т.д.).

Кортежі (tuples) – подібний на тип масива (Array<>), тільки цей тип може зберігати значення різних типів: let user: [string, number and so on…].

as const – в кінці масива, ставимо тоді коли ми хочемо, щоб значення було не змінним: запис let result = [ 1, “hello”, false, 4, {} ] **as const** рівнозначна, що і let result: readonly [number, string, boolean, number, object] = [ 1, “hello”, false, 4, {} ].

type (Alias) – псевдоним типа (пишем название с большой буквы). Псевдонимы могут розширятся с пом. амперсанта “&”.

Interface - определяет свойства и методы, которые объект должен реализовать. Щоб реалізувати інтерфейс в класах: class Person **implements** IDetails {}. Інтерфейс може дублюватися, а також наслідуватися: interface IDetails **extends** IOtherDetails {…..}. При викор. інтерфейса, свойство може бути тільки публічним (**public**)!

Type assertion (преобразование типов) - преобразования значения переменной к определенному типу. Есть 2 варианта как преобразовать один тип к другому, например: <HTMLElement> або as HTMLElement.

Generic type (обобщенный тип) – это некий тип (**<T> -** параметр дженерика), который мы еще не создали, но потом при выполнении ф-и, мы укажем конкретный тип на конкретное сост. или ф-ю, например (**<number>**). Это нужно, если у нас например есть 2 и более типа с одинаковыми свойствами но разными значениями, то тогда нужно исп. дженерик.

keyof – дозволяє взяти всі ключі (свойства) з обєкта.

Mixins (міксіни) – дозволяють внасідувати свойста та методи зразу двох і більше класів.

Якщо ми хочемо установити сторонню бібліотеку, а вона не типизована TS, то нам потрібно встановити це додатково: npm I –dev @types/”тут вводимо бібліотеку, яку хочемо встановити”.

Щоб не дублювати типи чи інтерфейси – викор. утиліти-типи: **Omit** (видаляє свойство), **Pick** (встановлює, нове свойство), **Partial** (встановлює всі поля не обов’язковими “?”), Required (навпаки, вст. всі поля обовязковими).

**6. Як працює back-end:**

nodeJS- это браузерная программа, браузер (browser) который имеет общий движек V8.

**NodeJS** - принимает http запросы от клиентов (от browser) и отправляет обратно ответ, npm - это как бы наш менеджер пакетов, через него мы можем устанавлевать розличные пакеты.

JSON (JavaScript Object Notation) - сервер который возвр. нам обьект или массив чего либо.

**Методы:**

- JSON.parse('{}') - {} - преобразовует строку JSON в обьект.

- JSON.stringify({}) - '{}' – преобразовует обьект в JSON строку.

Параметр 'space' - ' ' (отступ), пример: JSON.stringify({ a: 2 }, null, ' ').

• Як працює БД: **важна для бизнеса, поскольку в них хранится**информация, необходимая для работы компании, например информация о клиентах, инвентарные записи и заказы на покупку. Все данные хранятся столбцами: строка и столбец где мы вводим что-то по ключу и значение, например, какого-то товара.

Самые популярные **БД**: MySQL, SQL, PostgreSQL, MongoDB.

• Як працюють методи create, read, update, delete (CRUD) з даними: вони працюють по принципу post, get, update, delete.

**7. Просмотреть http codes(100-200, 200-300, 300-400, 400-500, 500-600):**

100-200 – інформаційний статус (не помилки)

200-300 – успішний статус (не помилки)

300-400 – (перенаправлення) - коды в этом диапазоне сообщают, что для успешного выполнения действия необходимо сделать другой запрос. Как правило, достаточно изменить URL. Перенаправление может выполняться и без запроса пользователя, если второй ресурс запрашивается методом GET.

400-500 – (помилки на стороні клієнта) - сервер не обработал запрос из-за синтаксической ошибки. Например, не хватает данных или есть проблемы с валидацией домена. Клиенту не следует повторять запрос без изменений, так как это снова приведет к появлению ошибки 400.

500-600 – (помилки на стороні сервера) - коды из этого диапазона возвращаются в случаях, когда сервер знает, что произошла ошибка или не может обработать запрос. При использовании всех методов, кроме HEAD, в теле ответа указывается объяснение ошибки.

**8. Дотримуватися чистоти кода:**

• Правильно називати імена перемінним: наприклад, якщо це ф-я, то назва має бути такою: яку дію вона виконує.

• Не повторювати імена перемінних: не дублювати імена перемінних та не дублювати код загалом.

• Вміти працювати з ООП (класс, объект, интерфейс, абстракция, инкапсуляция, наследование и полиморфизм):

ООПи SOLID(объектно-ориентированное программирование) - программные сущности такие как переменные, классы, модули, функции, которые должны быть открыты для расширения, но не для модификации. Клиенты не должны зависеть от интерфейсов, которые они не используют.

**ООП**-шный обьект - взаемодествия с обьектами происходит через свойства и методы. Один из важнейших принципов ООП – разделение внутреннего и внешнего интерфейсов. Для скрытия методов или свойств от пользователей (защищенные поля) в js использ. “\_”: \_secretWorld = “Hello”. Защищенные поля так же наследуються.

Если свойство или метод начинается с нижнего подч. (\_content / \_title) то это значит, что создатель даного обьекта не хочет чтобы это свойтсво или метод был интерфейсным (чтобы кто-то пользовался им во внешнем мире).

ООП концепция говорит о том, что внутри методов мы должны работать со своим же обьектом.

Инкапсуляция – это свойство, позволяющее объединить данные и методы в классе и скрыть детали  
реализации от пользователя.

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Наследование – это свойство, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым или родительским. Новый класс – потомком, наследником или производным классом.

**9. Тестування:**

- Ручне тестування - коли при написанні коду ми відразу все перевіряємо на помилки: if(4 === 4) true і тд.

- End-to-end (функціональні або ж автоматичні) тести - *це метод тестування програмного забезпечення для перевірки потоку додатків від початку до кінця. Метою функціонального тестування є імітація реального сценарію користувача та перевірка системи, що перевіряється, та її компонентів для інтеграції та цілісності даних.* Краще за все для JS використовувати стек **WebdriverIO** (cypress, enzyme, puppeteer і тд.).

- UNIT тести - ці тести можна виконувати для кожної ф-ї, метода чи класса відразу ще при написанні їх, щоб відразу виявити помилку ще на старті роботи. Краще за все для JS використовувати стек **Jest**, **React-testing-library** (для react-router-dom + redux).

**10. Відкладка кода(debug):**

Вміти працювати з консолю: Debbuger – точка остановы. Чтобы не дебажить можно в консоле -> Sources -> Ctrl + P -> прописать какой комп. мы ищем и там смотреть его.

**11. GIT and GitHub:**

Вміти виконувати всі команди гіта та вміти працювати з вітками:

Git - удаленный репозиторий.

https://git-scm.com- для установки.

Команды:

- git config --global user.name "eduard.tobilko” - для подкл. git к моей странички GitHub.

- git config --global user.email "eduard777@gmail.com" - для подкл. git к моей странички GitHub.

- clear или cls - очистка консоли.

- dir - для просмотра папок ветке.

- git status - для просмотра всех файлов на нашем проекте.

- git branch - для просмотра ветки на которой мы находимся.

- git branch test - для созд. новой ветки. И если вписать git branch - нам покажет 2 ветки: master и test.

- git branch -D test - для удаления ветки.

- git checkout - для переключение между ветками, если напишем git checkout test - мы перейдем на ветку test.

- git checkout -b test2 - для создания ветки и сразу переход на неё.

- git push -u origin test2- для перехода на новую ветку (test2).

Когда файлы зеленые - значит они на гиту, а если мы в каком-то файле вносим изменения, то файл становиться красным и чтобы снова его залить на гит, нужно вписать команду git add “file”- для изменения чего-то в файле, а чтобы все файлы добавить нужно вписать git add .

- git rm index.html - для прекращения слежки за файлом.

- .gitignore - чтобы игнорировать папку или файл который нам не нужен.

- git push - для доб. нововведений в GitHub-repository: git push -u origin main.

- git merge - для совмещения 2 веток ( test и test2 ): если мы находимся на ветке test, то вписать нам нужно: git merge test2.

- git push origin -u “нова\_назва” - для перейменування гілки віддалено ( безпосередньо на github ).

- git push origin --delete “назва” - для видалення гілки з github.

- rm test.js - для видалення файла чи папки.

Створення дочірніх віток:

В консолі натискаємо на гілку ( express-restapi ) на якій ми знаходимося -> create new branch from “express-restapi” -> потім у нас буде нова гілка ( express-restapi-2 ).

Для работы с чужим проектом:

- mkdir name-new-project - для создания новой папки в консоли через git.

- git clone https://github.com/utimur/algs\_and\_structures\_course.git- для клонирование проекта.

- ls - для просмотра названия проекта который мы клонировали с репозитория (algs\_and\_structures\_course/).

- cd algs\_and\_structures\_course - для просмотра файлов, после открываем этот проект и изменяем в нем что-то:

- git add . - добавляем наши все изменения.

- git commit -m ( -m - значит минус message ) "change log" - описываем что мы изменили и где и делаем коммит.

- git pull - для того, что бы можно было забрать все последнии изменения c github.

- git pull request - для того, что бы можно все свои изменения на своей ветке отправить на главную ветку “main”.

- git merged request - для отримання об’єднаних запитів на вітку “main”.

**Для демонстрации своего сайта на github-pages:**

- Gulp: git subtree push --prefix src origin gh-pages.

SSH-ключи доступа: ssh-keygen.

- React: npm i gh-pages --save-dev, после в package.json (в самом верху) пишем "homepage": "https://edward-tobilko.github.io/Weather-App", после в "scripts" пишем "predeploy": "npm run build" и "deploy": "gh-pages -d build", после git add ., после git commit -m "deployed project", git push, после npm run deploy.

**12. Beck-end:**

SQL (**Structured Query Language**) — це мова для роботи з реляційними базами даних (MySQL, PostgreSQL, SQLite, MS SQL тощо).  
Використовується для:

• створення таблиць (структури даних),

• додавання/зміни/видалення рядків,

• вибірки даних (запити).