ChatGPT

JavaScript Roadmap – 30 dana do Junior Full-Stack Developera

Polazne osnove: Učesnik ima osnovno znanje Pythona i HTML/CSS, i sada prelazi na JavaScript. Cilj je da za 30 dana (~240 sati) dostigne nivo da samostalno izrađuje web aplikacije, integriše API-je, manipuliše DOM-om, piše skripte, testira kod i razume debugging. Plan je podeljen u 4 tematske nedelje, svaka sa dnevnim fokusom na teoriju *i* praktične vežbe. Svaki dan predviđa ~8 sati rada (4 bloka x 2h). Kroz plan će se razvijati mini-projekti (accordion, kalkulator, to-do lista, weather app, landing page) u GitHub repo **html-css-js-portfolio** uz kontinuiranu primenu stečenog znanja.

> **Napomena:** Svaki dan navedene su ključne veštine/koncepti, konkretne JS funkcionalnosti, predloženi zadaci i korisni resursi. Fokus je na realnoj primeni znanja i pripremi za profesionalni rad (dobri workflow-i, čisti kod, Git). Svi resursi su *besplatni* i pažljivo odabrani (MDN dokumentacija, *The Modern JavaScript Tutorial*, Frontend Mentor izazovi, Exercism, Scrimba, Codewars itd.) za dalje produbljivanje znanja.

1. nedelja: Osnove JavaScript-a i jezički koncepti

Cilj nedelje 1: Usvajanje sintakse i fundamentalnih koncepta JavaScript-a (promenljive, tipovi podataka, operatori, uslovi, petlje, funkcije), razumevanje struktura podataka (nizovi, objekti), i uvođenje u DOM manipulaciju i događaje. Kroz jednostavne primere i mini-vežbe, učesnik gradi čvrstu osnovu za složenije projekte narednih nedelja.

Dan 1: Postavka okruženja i “Hello World” u JS

- **Veštine:** Radno okruženje (VS Code + WSL2), osnovno korišćenje konzole i pregledača, razumevanje povezanosti HTML i JS.
- **Koncepti:** Uključivanje JavaScript-a u stranicu (`<script>` tag), pokretanje skripti lokalno, osnovne konzolne komande (`console.log`). Razlika između pokretanja JS u pregledaču i Node.js (napomena: fokus je na pregledaču u ovoj fazi).
- **Zadaci (praksa):** Instalirati VS Code ekstenzije za JavaScript (npr. ESLint). Napraviti jednostavnu HTML stranicu sa skriptom koji ispisuje “Hello World” u konzoli. Probati pokretanje iste skripte i u Node okruženju (npr. `node hello.js`) za uvid u razlike. Otvoriti DevTools konzolu u pregledaču i isprobati izvršavanje jednostavnih izraza interaktivno.
- **Resursi:** MDN – Uvod u JavaScript (osnove upotrebe JS na stranici), Scrimba – *Intro to JavaScript* kurs (interaktivne vežbe za početnike).

Dan 2: Promenljive i tipovi podataka

- **Veštine:** Deklarisanje promenljivih (`let`, `const`, `var`), pravilno imenovanje, razumevanje dinamičkog tipiziranja. Osnove ugrađenih tipova podataka i operacija nad njima.
- **Koncepti:** Primtivni tipovi (brojevi, stringovi, booleans, `null`, `undefined`, `symbol`, `bigint`) i objekti [1]. Konverzije tipova (npr. `Number("5")`, `String(123)`), osnovni operatori (+, -, , /, % *za brojeve; + za string konkatenaciju). Razlika između mutabilnih i imutabilnih** vrednosti.

- **Zadaci:** U konzoli ili skripti kreirati nekoliko promenljivih različitih tipova. Vežbati osnovne operacije – npr. sabrati dva broja, spojiti dve string promenljive. Isprobati specifičnosti: deljenje nulom, `NaN`, `typeof` operator za određivanje tipa vrednosti. Napraviti funkciju koja računa obim kruga za dati poluprečnik (koristiti `Math.PI`).
- **Resursi:** *The Modern JavaScript Tutorial* – poglavlje **Data types** (javascript.info) [1], MDN – JavaScript data types and data structures (detaljan pregled tipova podataka i njihovih osobina).

Dan 3: Kontrola toka i uslovno izvršavanje

- **Veštine:** Pisanje jednostavnih algoritama koristeći uslovne strukture i petlje. Razumevanje logičkih operatora i grananja programa.
- **Koncepti: Uslovi**: `if/else` struktura, **logički operatori** (`&&`, `||`, `!`), ternarni operator (`condition ? expr1 : expr2`). **Petlje**: `for`, `while`, `do...while` – ponavljanje bloka koda. Koncept **truthy/falsy** vrednosti (koje vrednosti se u logičkom kontekstu tretiraju kao true/false) [2] [3]. Ugnježđeni uslovi i petlje. **Switch** izraz za više grana uslova [4].
- **Zadaci:** Napisati skriptu koja za zadati broj ispisuje u konzoli da li je paran ili neparan (`if` uslov). Napraviti petlju koja računa sumu brojeva od 1 do 100. Kroz `for` petlju proći kroz niz od nekoliko brojeva i za svaki ispisati da li je > 0, < 0 ili = 0. Osmisliti jednostavan kalkulator uslovima: npr. promenljive `a`, `b` i `op` (operator kao string), pa preko `switch` odrediti rezultat operacije.
- **Resursi:** MDN – *Control flow* (pregled JavaScript naredbi kontrole toka) [2] [3], MDN – *Loops and iteration* (primene različitih petlji). Interaktivno vežbanje: *Exercism* JavaScript zadaci za uslovne izraze (npr. vežba **Leap** – ispitivanje prestupne godine) [5].

Dan 4: Funkcije – definisanje i pozivanje

- **Veštine:** Pisanje i pozivanje funkcija, razumevanje parametara i povratne vrednosti. Modularizacija koda podelom u logičke celine.
- **Koncepti: Function declaration** vs **function expression** vs **arrow function** sintaksa. Parametri (podrazumevane vrednosti parametara, *rest* parametri) i `return` vrednost. **Scope (oblast važenja)** promenljivih: lokalni (unutar funkcije) i globalni. Koncept **"hoisting"** – kako se deklaracije premeštaju na vrh funkcije/scope-a pri izvršavanju (npr. `var` hoisting naspram blok-skopiranih `let/const`). Uvod u **lexical scope** (funkcije "vide" prom. definisane u svom okruženju).
- **Zadaci:** Definisati nekoliko funkcija: npr. `zbir(a, b)` koja vraća zbir, `pozdrav(ime)` koja vraća pozdravnu poruku. Vežbati **arrow funkcije** prevođenjem postojećih funkcija u kraću sintaksu (za jednostavne izraze). Isprobati *Immediately Invoked Function Expression* (IIFE) za razumevanje scope-a. Napraviti funkciju koja računa faktorijel broja na dva načina: iterativno (petljom) i rekurzivno (funkcija poziva samu sebe).
- **Resursi:** MDN – Functions (osnovni vodič kroz funkcije i scope), JavaScript.info – Function expressions and arrows (razlike između načina definisanja funkcija). Za hoisting: MDN – *var* (objašnjenje `var` hoistinga) i W3Schools – JavaScript Hoisting (kraći primeri).

Dan 5: Strukture podataka – nizovi i objekti

- **Veštine:** Korišćenje nizova i objekata za čuvanje podataka, iteriranje kroz kolekcije, osnovne operacije dodavanja/uklanjanja elemenata.
- **Koncepti: Nizovi** (Array) – deklaracija literala (`[]`), pristup elementima po indeksu, svojstvo `.length`. Ugrađene metode: `push()` i `pop()` (dodaj/ukloni kraj), `shift()` / `unshift()` (početak), `slice()` (izdvajanje podniza), `splice()` (ubacivanje/brisanje na proizvoljnoj poziciji), `indexOf()` (pronalaženje indeksa vrednosti) itd. **Objekti** – kreiranje objekta literala sa

`{ key: value }` parovima, svojstvo vs. metoda objekta. Ugnježdeni objekti i nizovi (strukture podataka unutar drugih). Mutabilnost objekata (mogu se menjati njihove vrednosti, dok primitivni tipovi ne). Iteracija kroz niz (klasična `for` petlja, `for...of` petlja, metod `forEach`). Iteracija kroz svojstva objekta (`for...in` petlja ili `Object.keys()` / `values()`).

- **Zadaci:** Napraviti niz od npr. 5 imena i ispisati ih pojedinačno u konzoli. Vežbati dodavanje i uklanjanje elemenata (npr. simulirati red čekanja: dodaj osobe na kraj reda, ukloni sa početka). Kreirati objekat `osoba` sa poljima poput ime, prezime, godine i metodom koji ispisuje puno ime. Napraviti niz objekata (npr. lista korisnika) i proveriti mogu li se filtrirati određeni (kasnije će se koristiti metode poput `filter` i `map` – može se spomenuti kao najava ES6 metoda).
- **Resursi:** MDN – *Arrays* (referenca i lista metoda), JavaScript.info – Arrays (jasna objašnjenja najčešćih operacija nad nizovima). MDN – *Objects* (osnove kreiranja i korišćenja objekata). Platforme za vežbu: *Exercism* JavaScript track (npr. vežbe **Resistor Color** ili **High Score Board** za rad sa nizovima i objektima) [6].

Dan 6: Uvod u DOM manipulaciju

- **Veštine:** Selektovanje HTML elemenata kroz JavaScript i menjanje njihove sadržine i stilova. Razumevanje Document Object Model-a kao predstavljanja stranice.
- **Koncepti: DOM** – hijerarhijska struktura dokumenta (čvorovi za elemente, tekst, itd.). Korišćenje DOM API-ja: `document.getElementById()`, `document.querySelector(All)` za pronalaženje elemenata [7] [8]. Uređivanje sadržaja: svojstvo `textContent` i `innerHTML` za izmene teksta/HTML unutar elementa. Menjanje atributa (`element.setAttribute`) i stila (`element.style.prop = value`). Kreiranje novih elemenata: `document.createElement()` i dodavanje u DOM (`parent.appendChild()` ili `parent.insertBefore()`). Brisanje elemenata (`element.remove()` ili `parent.removeChild()`).
- **Zadaci:** Osmisliti jednostavnu HTML stranicu (npr. lista od `<ul><li>` elemenata). Pomoću JS selektovati neki element i promeniti mu tekst (npr. promeniti naslov stranice). Dodati novi `<li>` u listu dinamički preko skripte. Napraviti *toggle* promene stila: npr. klikom na dugme promeniti boju nekog teksta (za to će trebati i događaji – uvod u eventove sledi sutra, ali može se unapred demonstrirati sa `onclick` atributom na elementu za jednostavnost). Vizuelizovati DOM hijerarhiju – npr. `console.dir(document.body)` da se vidi struktura.
- **Resursi:** freeCodeCamp – *JavaScript DOM Tutorial: Build a Calculator* – u uvodnom delu lepo objašnjava selektovanje, kreiranje i modifikaciju DOM elemenata kroz praktičan primer [7] [8]. MDN – Introduction to the DOM (konceptualno razumevanje DOM-a). **MDN DOM Reference**: metode poput `document.createElement` i sl. sa primerima upotrebe.

Dan 7: Događaji i interaktivnost na stranici

- **Veštine:** Reagovanje na korisničke akcije (klik, unos teksta, pritisak tastera) putem event listener-a. Dinamičko menjanje stranice kao odgovor na događaje.
- **Koncepti: Eventi** – definicija i tok: npr. *click* događaj se dešava kada korisnik klikne element [9] [10]. Postavljanje osluškivača: `element.addEventListener("event", handler)` [11]. *Event handler* funkcije – koriste opcioni argument `event` za pristup informacijama o događaju (npr. `event.target`). Uklanjanje osl. sa `removeEventListener`. Učestali događaji: `click`, `input` (promena u polju za unos), `submit` (forma), `keydown/keyup` (tastatura), `mouseover/mouseout` (hover). **Prevent default** – sprečavanje podrazumevanog ponašanja (npr. `event.preventDefault()` na submit eventu forme da se ne reloada stranica). Propagacija događaja (bubbling i capturing) – kratko pomenuti kao koncepte za kasnije, uz savet da se koristi `event.stopPropagation()` ako treba zaustaviti bubble.

- **Zadaci:** Implementirati interaktivnost: npr. napraviti dugme "Promeni boju pozadine" koje na klik menja boju pozadine stranice na nasumičnu (kao u MDN primeru) [12] [11]. Napraviti polje za unos i ispod paragraf; dok korisnik kuca u polje, u paragrafu neka se prikazuje dužina unetog teksta (koristiti `input` event). Osmisliti *FAQ Accordion* komponentu: nekoliko pitanja i odgovora u HTML-u, CSS ima klase `.is-open` / `.is-closed` za prikaz/sakrivanje; putem JS dodati event listener na svako pitanje (naslov) da na klik promeni klasu odgovoru (prikaže ili sakrije odgovor). *(Ovo direktno odgovara mini-projektu accordion u repozitorijumu.)*
- **Resursi:** MDN – Introduction to events (objašnjava šta su događaji i kako funkcionišu) [9] [11]. MDN – addEventListener (upotreba metode za registraciju event handlera). Za vežbu, **Frontend Mentor** ima jednostavne interaktivne izazove (npr. *FAQ Accordion Card* izazov) [13] – isprobati implementaciju svog accordion-a pa uporediti sa zajednicom. Takođe, *Scrimba* nudi interaktivne vežbe na temu DOM događaja.

2. nedelja: DOM projekti i naprednije funkcionalnosti

Cilj nedelje 2: Primena osnova kroz izradu praktičnih mini-projekata koji unose interaktivnost na stranicu. Fokus je na radu sa DOM-om i događajima, uz postupno uvođenje naprednijih JavaScript mogućnosti (complex logika u funkcijama, *closure*, ES6 sintaksa). Kroz izradu **Kalkulator** i **To-Do** aplikacija, polaznik uvežbava manipulisanje elementima, rukovanje korisničkim unosom i organizaciju koda.

Dan 8: Projektni dan – Kalkulator (Calculator), planiranje i struktura

- **Veštine:** Planiranje razvoja aplikacije u fazama, razbijanje problema na manje celine. Kreiranje strukture HTML/CSS za projekt i povezivanje sa JS modulom.
- **Koncepti: Planiranje projekta:** Definisanje zahteva – kalkulator treba da omogućava osnovne aritmetičke operacije (+, –, ×, ÷), resetovanje prikaza (C), i eventualno rad sa decimalnim tačkama. Osmišljavanje interfejsa (tabela dugmadi 0–9, operacije, display ekran). **Manipulacija DOM-om:** hvatanje referenci na elemente (display, dugmići) preko `querySelectorAll`. **Event handling** za više elemenata: koristiti petlju da se svim dugmićima doda isti *listener* (npr. svi brojčani dugmići pozivaju istu funkciju koja dodaje cifru na ekran) [14] [11].
- **Zadaci:** Napraviti HTML strukturu kalkulatora (npr. tabela ili grid dugmadi). Stilizovati osnovno (CSS grid za raspored). U JS fajlu, definisati prazan display (npr. `<input type="text" readonly id="display">`). Pomoću JS selektovati sva dugmad cifara i dodati im event listener: kada se klikne, ta cifra se dodaje na kraj vrednosti display-a. Testirati dodavanje brojeva na ekran. Planirati kako će se čuvati trenutno stanje (npr. prva vrednost, odabrana operacija, druga vrednost) – za sada samo ispisujte unos.
- **Resursi:** freeCodeCamp – *Build a Calculator App* (detaljan vodič, **preporučljivo pročitati sekcije o planiranju i selektovanju elemenata**) [7] [15]. MDN dokumentacija za form input polje (korisno za prikaz rezultata). Inspirišite se i Frontend Mentor izazovom **Calculator App** (dizajn i zahtevi) – pokušajte postići sličnu funkcionalnost.

Dan 9: Implementacija Kalkulatora – logika i događaji

- **Veštine:** Upravljanje složenijom logikom u JavaScript-u (kumulativno računanje, stanje aplikacije), bolje razumevanje *event flow-a*. Debugging jednostavnih logičkih grešaka tokom implementacije.
- **Koncepti: Logika kalkulatora:** Potrebno je pratiti unesene brojeve i operacije. Može se koristiti promenljiva za trenutno prikazanu vrednost (`current`), za prethodnu vrednost (`previous`) i operator (`operation`). Kada korisnik pritisne operaciju, sačuvati `previous` i `operation`, a isprazniti `current` za unos narednog broja. Na pritisak `=` izračunati rezultat na osnovu

`operation` (+, -, * , /) i prikazati ga. Edge cases: deljenje sa nulom (prikazati grešku ili `Infinity`). **Event handling:** Osim brojeva, obraditi klik na operacione dugmiće (čuvaju operaciju), na "=" (izvršava proračun), na "C" (resetuje sve na početno stanje). **Parse vs eval:** ručno parsiranje stringa display-a u brojeve (`Number()` ili `parseFloat`), nikako ne koristiti `eval` zbog bezbednosti [16].

- **Zadaci:** Implementirati funkcije: `clear()` (resetuje display i interne promenljive), `inputDigit(d)` (dodaje cifru na kraj current vrednosti), `inputOperator(op)` (sačuva current u previous i operator u promenljivu, pripremi za unos nove vrednosti), `calculate()` (na osnovu previous, current i operator izračuna rezultat). Povezati ove funkcije sa event listener-ima dugmadi. Testirati razne scenarije: npr. 12 + 7 = 19, zatim * 2 = 38, itd. Popraviti bugove ako logika ne pokriva neki slučaj (ovo je prilika za **debugging**: koristiti `console.log` unutar funkcija da pratite tok vrednosti).
- **Resursi:** MDN – *Number* objekat (za konverziju string → broj i nazad). Stack Overflow – *JavaScript floating point issues* (zašto 0.1 + 0.2 nije 0.3, ukratko; eventualno rešavanje za kalkulator ako se zahteva). **freeCodeCamp tutorial** (nastavak od juče) – sekcija *"How to Add Functionality to the Calculator"* [15] objašnjava kako autor implementira logiku, uporedite sa svojim rešenjem.

Dan 10: Napredni koncepti – *Scope*, *Closure* i *Hoisting*

- **Veštine:** Dublje razumijevanje kako JavaScript funkcioniše "ispod haube", što će pomoći pri debugovanju i pisanju efikasnijeg koda.
- **Koncepti: Scope lanac:** dostupnost promenljivih u zavisnosti od mesta definicije. *Global scope* (promenljive definisane van svih funkcija) vs *function scope* (prom. definisane unutar funkcije vidljive samo tamo) vs *block scope* (`let/const` u bloku). **Closure (zatvaranje):** funkcija "pamti" leksički kontekst u kom je nastala [17]. Primer: funkcija definisana unutar druge funkcije i vraćena van ima pristup varijablama iz originalnog okruženja i nakon što je spoljašnja funkcija izvršena – to se zove closure. **Hoisting:** mehanizam gde JS pred izvođenje "podigne" deklaracije funkcija i `var` promenljivih na vrh svog scope-a. Razumevanje razlike: `var` hoisted (može se referencirati pre deklaracije, ali vrednost `undefined` do inicijalizacije), dok `let/const` imaju tzv. *Temporal Dead Zone* (ne mogu se koristiti pre deklaracije) [18] [19].
- **Zadaci:** Pročitati i razumeti primer closure-a: npr. funkcija `counter()` koja kreira lokalnu `let count = 0` i vraća funkciju koja na svaki poziv povećava `count` i ispisuje novu vrednost. Pozivanjem više puta, videti kako `count` opstaje u zatvorenom okruženju te vraćene funkcije. Isprobati *hoisting* praktično: konzolovati neku `var` promenljivu pre deklaracije i videti da daje `undefined`, dok ako isto uradite sa `let`, dobijate grešku (TDZ). Takođe, definisati dve funkcije sa istim imenom u istom scope-u i videti da druga "pregazi" prvu zbog hoistinga.
- **Resursi:** MDN – Closures (detaljno objašnjenje zatvaranja, sa primerima praktične upotrebe) [17]. You Don't Know JS (book) – Scope & Closures (za dublje razumevanje, opciono). MDN – *let* i *const* (objašnjava temporal dead zone) [18]. Kratak video: *What is a closure?* (odlično objašnjeno u ~5 min).

Dan 11: ES6 sintaksa i korisne funkcionalnosti

- **Veštine:** Korišćenje modernih JavaScript mogućnosti uvedenih sa ES6+ standardom za pisanje čistijeg i kraćeg koda. Razumevanje kako nove konstrukcije olakšavaju razvoj.
- **Koncepti: ES6+ novine: Arrow funkcije** (kraća sintaksa, implicitni `return` za jednostavne izraze, ali i neočekivano ponašanje `this` – spomenuti da arrow ne stvara sopstveni `this`). **Template literals** (napredna konkatenacija stringova sa backtick `` ` `` i `${expr}` sintaksom). **Destructuring** objekata i nizova – vađenje vrednosti u promenljive kratkom sintaksom (`const {x, y} = obj; const [a, b] = arr;`). **Default parametri** u funkcijama (`function f(a=1) {}`), **rest operator** (`function f(...args)`) i **spread operator**

(`arr2 = [...arr1, 4,5]`, ili `obj2 = {...obj1, novoPolje: 123}`). **Array metode**: `map`, `filter`, `reduce` – moćni načini da se niz transformiše ili agregira (npr. `filter` za filtriranje elemenata po uslovu, umesto manuelne petlje). **Set i Map strukture** (ukratko pomenuti alternative nizovima i objektima za posebne slučajeve: Set za unikate, Map za ključeve bilo kog tipa).

- **Zadaci:** Refaktorisati deo koda prethodnih projekata korišćenjem novih konstrukcija: npr. u kalkulatoru, ako postoji niz dugmadi, koristiti `forEach` umesto klasične for petlje za dodavanje listener-a. Upotrebiti template literal za formiranje poruke umesto string konkatenacije sa `+`. Vežbati destrukturiranje: npr. imati objekat `{ name: "Ana", age: 28, city: "NS" }` i izvući `name` i `city` u promenljive. Napraviti niz brojeva pa kreirati novi niz gde su svi brojevi duplirani (`map`), zatim filtrirati da ostanu samo parni (`filter`). Uzeti niz reči i koristiti `reduce` da spoji u jedan string sa razmakom.
- **Resursi:** JavaScript.info – *Arrow functions* (kada ih koristiti i zašto su korisne). MDN – *Template literals* (interpolacija promenljivih u stringu). MDN – *Destructuring assignment* (sintaksna olakšica za izvlačenje vrednosti). MDN stranice za metode: **Array.prototype.map**, **filter**, **reduce** (svaka sa primerima upotrebe). Za vežbu i bolje shvatanje ovih metoda, rešavati zadatke na *Codewars* (početi od 8 kyu nivo zadataka) – odličan je **Codewars** moto: *"Train on kata and reach your highest potential"* [20], gde mnogi zadaci zahtevaju kreativnu primenu ovih metoda.

Dan 12: Mini-projekat – To-Do lista (I deo: struktura i dodavanje zadataka)

- **Veštine:** Primena znanja DOM manipuacije, eventova i array metoda u kontekstu jedne praktične aplikacije. Upravljanje listom podataka (zadacima) i njihov prikaz na stranici.
- **Koncepti: Aplikacija To-Do:** korisnik unosi tekst zadatka, dodaje ga na listu; može označiti da je gotov ili ga obrisati. Razmisliti o strukturi podataka – npr. niz objekata `tasks`, gde svaki ima `{ text: "...", completed: false }`. Prilikom dodavanja zadatka, taj objekat se dodaje u niz i renderuje na stranicu. **DOM generisanje liste:** dinamički kreiranje `<li>` elemenata za svaki zadatak, sa odgovarajućim sadržajem i možda dugmićima (npr. "✔" za complete, "🗑" za delete). **Event delegacija:** ako lista postaje duga, umesto dodavanja listenera svakom `<li>`, može se dodati na parent (npr. UL) pa proveriti izvor eventa putem `event.target`. (Objasniti koncept event delegacije i kako štedimo resurse.)
- **Zadaci:** Postaviti osnovni HTML: input polje za unos novog zadatka i dugme "Add" (ili koristiti `<form>` sa submit). Ispod neka bude prazan `<ul id="todo-list"></ul>`. Implementirati dodavanje: na klik "Add" ili submit forme, uzeti vrednost iz inputa, kreirati novi `<li>` sa tim tekstom i npr. dva span dugmeta za akcije. Dodati ga u UL listu. Takođe taj zadatak dodati u interni `tasks` niz (radi dalje logike). Očistiti input polje. Osigurati da prazan unos ne pravi zadatak (proveriti string length). Za sada, samo dodavanje i prikaz raditi.
- **Resursi:** Frontend Mentor – *Todo app challenge* (specifikacija funkcionalnosti to-do aplikacije) [13] – korisno za inspiraciju, predviđa i filtriranje (all/active/completed) i drag&drop za napredne, ali za početak fokus na osnovno. MDN primeri za manipulaciju liste elemenata. Event delegacija: MDN – *Event.target* (kako utvrditi na kom elementu se desio event).

Dan 13: To-Do lista (II deo: markiranje i brisanje, localStorage)

- **Veštine:** Upravljanje promenama stanja aplikacije (označavanje kao završeno, brisanje) i sinhronizacija prikaza sa podacima. Korišćenje *Web Storage API*-ja za čuvanje podataka lokalno u pregledaču.
- **Koncepti: Markiranje obavljenog zadatka:** na klik npr. checkboxa ili "✔", pronaći odgovarajući task u nizu (ili iskoristiti podatak *data-id* u DOM elementu da znamo koji je) i promeniti mu `completed: true/false`. Promeniti i stil tog `<li>` (npr. precrtati tekst putem CSS klase `.done`). **Brisanje zadatka:** ukloniti `<li>` iz DOM-a i obrisati taj task objekat iz niza (`tasks`).

LocalStorage: trajno skladištenje podataka u pregledniku – pristupa se preko `window.localStorage` objekta [21]. Može se čuvati samo string, pa za strukture koristiti JSON serijalizaciju: `localStorage.setItem("tasks", JSON.stringify(tasksArray))`. Pri učitavanju stranice, pročitati string pa `JSON.parse` nazad u niz objekata [22]. Ovo omogućava da to-do lista ostane sačuvana i nakon refresh-a stranice (pošto *localStorage* čuva podatke i kad se tab zatvori, dok ne obrišemo).

- **Zadaci:** Dodati kod za event delegaciju nad listom: npr. `todoList.addEventListener('click', function(e) { ... } )` gde `e.target` može biti: checkbox (oznaka završenog) ili delete dugme. Na osnovu klase ili data-attributa elementa odlučiti koju akciju raditi. Za checkbox: pronaći roditelja `<li>` i dodati/ukloniti klasu `.done`, a u `tasks` nizu prebaciti `completed`. Za delete: ukloniti `<li>` (`element.remove()`), a iz `tasks` niza filtrirati taj element van (`tasks = tasks.filter(t => t.id !== idZaBrisanje)`). Nakon svake izmene (dodavanja, brisanja, statusa) **ažurirati localStorage**: `localStorage.setItem('tasks', JSON.stringify(tasks))`. Takođe, pri inicijalizaciji aplikacije (npr. na `DOMContentLoaded` događaj) uraditi `const saved = localStorage.getItem('tasks')`, ako postoji, parsirati i učitati postojeće zadatke u niz i iscrtati ih. Testirati funkcionalnosti u pregledaču, proveriti *Application → Local Storage* u DevTools da li se čuvaju stavke.
- **Resursi:** MDN – Web Storage API (objašnjenje razlike `localStorage` vs `sessionStorage` i način upotrebe) [21]. MDN – *Window.localStorage* (referenca za konkretne metode). **Exercism** ima dobru vežbu *"High Score Board"* koja simulira manipulaciju liste rezultata, što je slično dodavanju/brisanju zadataka (može se uraditi van ovog plana za dodatnu praksu).

Dan 14: Recap nedelje – integracija znanja i mini izazovi

- **Veštine:** Konsolidacija naučenog kroz kratke izazove i popravku eventualnih nedostataka u razumevanju. Samoprocena: koliko se stečene veštine mogu primeniti u novim situacijama.
- **Koncepti:** Ponavljanje ključnih pojmova: DOM, eventi, asinhronost (najava za narednu nedelju), ES6 sintakse, closure. Identifikacija oblasti koje su teže (npr. neki imaju problema s razumevanjem *this* ili closure) i ciljano razjašnjenje tih tema dodatnim primerima. **Čistoća koda:** uočavanje ponavljajućeg koda i refaktorisanje (npr. ako je u to-do app kod za renderovanje sličan kodu za dodavanje – izdvojiti ga u funkciju `renderTasks(tasks)`). **Git & GitHub praksa:** pregled commitova ove nedelje, provera da li poruke commitova jasno opisiju promene, eventualno napraviti `README` za projekte sa uputstvom za korišćenje.
- **Zadaci:** Proći kroz kod mini projekata (Accordion, Calculator, To-Do) i pokušati ih malo optimizovati: npr. dodati komentare gde je logika složenija, ukloniti nepotreban kod ili konsolidovati funkcije. Kao izazov, rešiti 1-2 kratka algoritamska zadatka koji nisu direktno vezani za DOM, da se održi i *algorithmic thinking*: npr. **FizzBuzz** (klasičan zadatak – ispisati brojeve 1-100, ali za deljive sa 3 ispisati "Fizz", sa 5 "Buzz", sa oba "FizzBuzz"). Ili **palindrom** proveru (da li je dati string isti čitan unazad). Ove zadatke pokušati rešiti samostalno, pa uporediti sa standardnim rešenjima.
- **Resursi:** *Codewars* platforma za algoritamske izazove (početi od 8 kyu zadataka i popeti se ka 7 kyu) – vežbanje na Codewars-u pomaže da se znanje primeni van konteksta projekata [20]. Pročitati članke o **clean code** principima (npr. *"JavaScript Clean Coding Best Practices"*). Za GitHub: pogledati **Conventional Commits** predlog formata poruka [23] (npr. započeti poruku glagolom u sadašnjem vremenu: "Add feature X", "Fix bug Y").

3. nedelja: Asinhroni JavaScript i rad sa API-jem

Cilj nedelje 3: Upoznavanje sa asinhronim konceptima u JavaScript-u (tajmeri, Promise objekti, async/await) i korišćenje spoljašnjih podataka preko API poziva. Kroz izradu **Weather App** projekta, polaznik

će naučiti kako da preuzme podatke sa servera (HTTP fetch), kako da ih prikaže korisniku, kao i kako da rukuje greškama i različitim stanjima aplikacije (učitavanje, greška, uspeh). Takođe, biće reči o naprednijim tehnikama debagovanja i rukovanja greškama.

Dan 15: Asinhroni JavaScript – uvod (tajmeri, Promise)

- **Veštine:** Razumevanje kako JavaScript može da izvršava kod *asinhrono*, koncept *event loop*-a i *callback queue*. Korišćenje tajmera i obećanja (Promise) za odloženo ili paralelno izvršavanje.
- **Koncepti: Single-threaded priroda JS + event loop:** objašnjenje da JS ima call stack i da asinhrone operacije (setTimeout, HTTP zahtev) idu u web APIs, pa callback u queue, itd. **Tajmeri:** `setTimeout(fn, ms)` za odloženo izvršavanje koda i `setInterval(fn, ms)` za periodično izvršavanje. **Promise:** objekt koji predstavlja rezultat asinhrone operacije koji može biti ostvaren ili odbijen (fulfilled/rejected). Kreiranje promise-a (new Promise((resolve, reject) => {...})), korišćenje `.then` i `.catch` za registraciju callbackova. **Async/await sintaksa:** sintaksni šećer za rad sa promise-ima – funkcija označena sa `async` može da koristi `await` da "sačeka" promise umesto gniježđenja then-ova. (Napomena: try/catch blok za hvatanje greške u async funkciji).
- **Zadaci:** Napraviti jednostavan primer asinhronog ponašanja: npr. ispisati poruku, zatim koristiti `setTimeout` da se druga poruka ispiše posle 2 sekunde, zatim treća odmah (pokazati redosled izvršenja – da će treća izaći pre druge zbog pauze). Kreirati promise koji npr. posle 3 sekunde *resolve*-uje neki broj, i iskoristiti `.then` da se taj broj loguje. Potom isti primer napisati sa `async function` i `await`. Eksperimentisati sa *reject* – npr. promise koji odbija posle određenog uslova, hvatanje sa `.catch`.
- **Resursi:** JavaScript.info – *Event Loop* (intuitivno objašnjenje kako JS rukuje asinhronim kodom). MDN – *Concurrency model and Event Loop* (detaljnije, za znatiželju). MDN – Using promises (kako kreirati i koristiti promise). Kratak video na YouTube: *JavaScript Promises In 10 Minutes*. Za vežbu: napraviti mali *"delay"* helper kao promise koji vraća resolve posle X ms i koristiti ga sa await (ovo je baza za razumevanje bilo kog čekanja).

Dan 16: Fetch API – preuzimanje podataka sa servera

- **Veštine:** Korišćenje ugrađenog *Fetch API*-ja za slanje HTTP zahteva (GET, POST), obrada JSON odgovora i rukovanje asinhronim ishodom. Razumevanje osnovnih koncepata REST API-ja.
- **Koncepti: HTTP pojmovi:** request, response, status kodovi (200 OK, 404 Not Found, 500 Server Error itd.), metode (GET za dobijanje podataka, POST za slanje, itd.). **Fetch API:** globalna funkcija `fetch(url, options)` koja vraća Promise koji se ostvaruje kad stigne odgovor [24]. Prvi then dobija `Response` objekat – treba pozvati `response.json()` (asinh., vraća Promise) da se dobije JS objekat iz JSON stringa. Rukovanje greškama: ako je response.status != 200, baciti grešku ili obraditi (npr. 404 – "Not found"). **CORS** (Cross-Origin Resource Sharing) – zašto će se koristiti javni API-ji koji dozvoljavaju cross-origin ili proxy, ne ulaziti preduboko, samo naglasiti da ako fetch ka nekoj usluzi ne radi zbog CORS, nije greška u kodu već policy.
- **Zadaci:** Isprobati `fetch` na nekom jednostavnom javnom API-ju: npr. JSONPlaceholder – napraviti fetch GET na `https://jsonplaceholder.typicode.com/todos/1` i dobiti jedan todo JSON. Ispisati rezultat u konzoli. Potom, fetch ka `https://jsonplaceholder.typicode.com/posts` sa metodom POST (poslati mali objekat sa `body: JSON.stringify({...})` i `headers: {'Content-Type': 'application/json'}`) da se vidi kreiranje resursa (vratiće nazad JSON novog posta). Vežbati hvatanje greške: promeniti URL u pogrešan da izazove 404, i uhvatiti `.catch` ili proveriti `response.ok` flag [25].
- **Resursi:** MDN – Using Fetch (detaljan tutorijal sa primerima i objašnjenjima korak po korak) [25] [24]. HTTP Status Dogs (zabavan prikaz status kodova, da se nauče osnovni). Za vežbu API-ja:

Exercism ima vežbu **Weather Forecaster** (sintetizovan primer API poziva i obrade), a *FreeCodeCamp* curriculum takođe uvodi Fetch kroz projekte.

Dan 17: Projektni dan – Weather App (I deo: dobijanje podataka sa API-ja)

- **Veštine:** Integracija spoljašnjeg API servisa u svoju aplikaciju – od registracije API ključa do prikaza podataka. Parsiranje JSON odgovora i korišćenje relevantnih delova.
- **Koncepti: Odabir API-ja:** Za vremensku prognozu koristiti npr. *OpenWeatherMap API* (popularan, ima besplatan nivo ~1000 poziva dnevno) [26]. Treba se registrovati za API key (string koji se dodaje u URL zahteva za autorizaciju). **Struktura API poziva:** URL za gradsko vreme obično izgleda `api.openweathermap.org/data/2.5/weather?q=Novi%20Sad&appid=APIKEY&units=metric&lang=hr`. Objasniti query parametre (q = city query, units = metric za °C, lang za lokalizovani opis vremena). **JSON odgovor:** sadrži više podataka (temperatura, opis, vlažnost itd.), treba pročitati dokumentaciju da se nađu potrebna polja. **Asinhrono upravljanje u interfejsu:** dok se čeka odgovor, dobro je prikazati korisniku *"Loading..."*, a kad stigne, zameniti ga podacima ili prikazati grešku ako grad nije nađen.
- **Zadaci:** Registracija na OpenWeatherMap i dobijanje API key-a (jednokratno, van koda). U HTML aplikacije predvideti input za unos grada i dugme "Search" (ili formu). Na klik/submit, uzeti naziv grada, napraviti fetch poziv ka API-ju (ubaciti grad i key u URL). Dok traje fetch, u DOM-u prikazati poruku "Tražim podatke za [grad]...". Kad stigne odgovor: ako je `response.ok`, parsirati JSON i izvući npr. `weather[0].description` (opis, npr. "clear sky"), `main.temp` (temperatura), `main.humidity` (vlažnost), `wind.speed` (brzina vetra). Prikazati te informacije u nekom *widget* divu. Ako API vrati grešku (npr. 404 ako grad ne postoji), uhvatiti to i prikazati poruku "Grad nije pronađen" korisniku.
- **Resursi:** OpenWeatherMap API docs (kako formirati URL i primer odgovora). **Dev.to – Building a Weather App** (vodič koji objašnjava korak po korak, uključujući HTML/CSS strukturu) [27] [28]. Preporučljivo: pročitati deo kako se obrađuje odgovor i prikazuju podaci, ali pokušati implementirati samostalno pa koristiti vodič za proveru.

Dan 18: Weather App (II deo: poboljšanja – UI/UX i dodatne funkcionalnosti)

- **Veštine:** Dorada projekta fokusiranjem na korisničko iskustvo i stabilnost: rukovanje više scenarija, poboljšanje izgleda, eventualno dodavanje novih mogućnosti (geolokacija, prognoza).
- **Koncepti: Loading i error stanja:** osigurati da su poruke korisniku jasne – npr. prilikom pretrage, dugme se može onemogućiti i tekst "Učitavanje..." pojaviti; nakon uspeha, prikaz rezultata; nakon greške, prikaz poruke i možda posebnog stila za grešku. **Čišćenje prethodnih podataka:** ako korisnik traži grad pa odmah drugi, da se stari rezultat obriše dok se čeka novi (kako ne bi bilo zbunjujuće). **Geolocation API (opciono):** `navigator.geolocation.getCurrentPosition(success, error)` – može se iskoristiti da se dobiju koordinate korisnika i odmah prikaže vreme za njegovu lokaciju na učitavanju aplikacije (ako dozvoli). To bi zahtevalo drugačiji API poziv (lat & lon parametri). **Proširenje:** mogućnost prikaza i **prognoze** (forecast) – OpenWeather ima /forecast endpoint (3-časovna prognoza). To bi bilo komplikovanije za 30-dnevni cilj, ali može se napomenuti kao naredni korak.
- **Zadaci:** Ulepšati prikaz: npr. temperatura velikim fontom, opis vremenskih prilika slikom ili ikonama (OWM vraća i `weather[0].icon` kod – može se uzeti ikonica sa `http://openweathermap.org/img/wn/{icon}@2x.png`). Dodati boje ili pozadinsku ilustraciju u zavisnosti od *clear/cloudy/rainy* itd. Testirati aplikaciju sa raznim unosima: validan grad, nepostojeći grad, prazan unos (tu odmah ne zvati API nego prikazati upozorenje "Unesite naziv grada"). Ako je implementirana geolokacija: testirati scenario da korisnik odbije dozvolu – obraditi `error` callback (prikazati "Nije moguće dohvatiti lokaciju"). Pregledati kod projekta,

refaktorisati duplikate ako postoje (npr. ako se *render weather data* kod ponavlja, izdvojiti u funkciju).

- **Resursi:** MDN – Geolocation API (za opcionu funkcionalnost lociranja korisnika). Dev.to vodič za Weather App – sekcije o popularnim Weather API-jima i zašto su korisni [29], može se pročitati za širu sliku. **Frontend Mentor** ima sličan projekat *Weather App* – posle sopstvene implementacije, pogledati kako drugi pristupaju dizajnu i kodu.

Dan 19: Debagovanje i upravljanje greškama u JavaScript-u

- **Veštine:** Sistematsko pronalaženje i ispravljanje grešaka (bugova) u kodu. Korišćenje alata kao što su *browser DevTools debugger*, linteri i pisanje robustnog koda uz hvatanje izuzetaka.
- **Koncepti: Debugging proces:** reproduciraj grešku, izoluj problem, ispravi, testiraj ponovo. **Korišćenje konzole:** `console.log()`, `console.error()` za ispis dijagnostike tokom razvoja (ali ukloniti ili onemogućiti u produkciji). **Debugger alat:** postavljanje breakpoints (Chrome DevTools Sources tab), praćenje tokova koda korak-po-korak, pregled vrednosti promenljivih u toku izvršenja. **Try/Catch:** hvatanje izuzetaka – npr. staviti *rizičan* kod (parsiranje JSON, mrežni poziv) u `try` blok i u `catch` dobiti objekat greške, pa odlučiti šta dalje (npr. ispisati korisniku poruku). **Linteri:** ESLint konfiguracija – hvata sintaksne i neke logičke greške statičkom analizom [30]. Preporuka da je ESLint uključen u VS Code (što u ovom projektu već jeste: skripta `npm run lint` proverava kod [31]).
- **Zadaci:** Namerno ubaciti grešku u kod (npr. u funkciji zvati promenljivu koja ne postoji) i videti kako DevTools prikazuje error stack trace. Postaviti breakpoint u to-do aplikaciji unutar event listenera pa dodati zadatak – pratiti tok. Vežbati `try/catch`: recimo, napraviti funkciju `parseJSON(str)` koja hvata grešku ako `JSON.parse` ne uspe, i vraća null umesto da skript pukne. Uključiti ESLint linter u projektu (ako nije već konfigurisan, inicijalizovati sa preporučenim podešavanjima) – u VS Code-u posmatrati podvlači li neke potencijalne probleme (npr. neiskorišćene promenljive, ili pogrešan upis `==` umesto `===`).
- **Resursi:** MDN – JavaScript debugging guide – saveti i tehnike za pronalaženje grešaka [32] [33]. Chrome DevTools – Debugging tutorial (kako koristiti breakpoint, watch expressions itd.). MDN – *console* API (spisak metoda za logovanje, npr. `console.table` za lepo prikazivanje nizova objekata). Preporuka: integrisati ESLint u razvoj – npr. postaviti *pre-commit hook* da uvek pokrene `npm run lint` [31], kako bi se uhvatile greške pre slanja na repo [30].

Dan 20: Testiranje koda – uvod u unit testing

- **Veštine:** Pisanje osnovnih *jediničnih testova* za JavaScript funkcije. Razumevanje važnosti automatizovanog testiranja i kako ono uliva poverenje u kod.
- **Koncepti: Šta su unit testovi:** male provere da jedna *jedinica* koda (funkcija ili modul) radi očekivano za razne scenarije. Izolacija od ostatka sistema. **Jest** test framework: popularan za JS, omogućava pisanje testova sa funkcijama `test()` i asertacijama `expect(value).toBe(expected)` [34] [35]. Struktura projekta: test fajlovi paralelno sa kodom ili u posebnom folderu, imenovani npr. `ime.test.js`. **Assertions:** različite vrste (toBe za proste vrednosti, toEqual za objekte, toThrow za greške, toBeTruthy itd.). **Pokretanje testova:** `npm test` skripta koja pokrene Jest i ispituje sve `.test.js` fajlove. Integracija sa CI (napomena za budućnost – automatsko pokretanje testova na GitHub).
- **Zadaci:** Instalirati i podesiti Jest (ili koristiti ugrađeni Node `assert` modul za jednostavno testiranje). Napisati par funkcija za testiranje – npr. funkciju `capitalize(str)` koja vraća string sa velikim prvim slovom. Zatim napisati test fajl koji proverava da `capitalize("hello")` daje `"Hello"`, da `capitalize("")` vraća prazan string, da `capitalize("javaScript")` daje `"JavaScript"`. Pokrenuti testove i proveriti prolaze li. Zatim dodati test za neku postojeću logiku iz projekta: npr. funkciju iz kalkulatora za sabiranje ili

funkciju koja filtrira završene zadatke u to-do listi. Ako vreme dozvoli, probati i testirati asinhronu funkciju (koristeći *done* callback ili async/await u testu).

- **Resursi:** Jest – zvanična dokumentacija (Getting Started) – kako instalirati i napraviti prvi test [34] [35]. MDN – Testing tutorial (osnove unit testiranja na klijentskoj strani). **Exercism** platforma praktično uči pisanje testova jer svaki zadatak dolazi sa gotovim testovima koje treba zadovoljiti – ako ste radili Exercism izazove, pogledajte test fajlove da steknete osećaj kako su napisani (često koriste Jest ili sličan stil očekivanja).

Dan 21: Vreme za algoritme – *Coding challenge* dan

- **Veštine:** Unapređenje algoritamskog razmišljanja i optimizacije koda. Brzo pisanje koda bez pomoći preglednika, fokus na JS jeziku "u vakuumu".
- **Koncepti: Algoritamske teme:** rad sa nizovima (sortiranje, pretraga), rad sa stringovima (manipulacija, provera palindroma), matematika (prost broj, Fibonacci), **Big-O notacija** (samo osnovno: razlika između O(n) i O(n^2) algoritma – zašto je bitno pisati efikasno za veće ulaze). **Problem-solving pristup:** razumevanje problema, razbijanje na korake, pisanje pseudokoda, implementacija, testiranje na primerima.
- **Zadaci:** Izabrati 2-3 problema odgovarajuće težine sa *Codewars* ili *HackerRank*. Predlozi: "Reverse a string" (okrenuti string unazad), "Factorialize a number" (računati faktorijel rekurzivno i iterativno), "Find the largest number in an array", "Fibonacci n-th element". Pokušati ih rešiti samostalno i optimizovati rešenje ako je moguće (npr. za Fibonacci, rekurzivno rešenje je eksponencijalno sporo za veće n, iterativno je mnogo bolje). Uporediti svoja rešenja sa onima drugih korisnika na platformi – često imaju elegantne trikove (ali budite kritični, ponekad najkraće rešenje nije i najčitljivije).
- **Resursi:** *Codewars* (kroz pretragu odabrati Kata sa visokom ocenom za "Algorithms" tag). *HackerRank* "Problem Solving (Basic)" sekcija za početnike. Takođe, Exercism – JavaScript track ima tematski organizovane probleme (npr. stringovi, petlje, uslovi) [36] – izaberite neku nedovršenu vežbu iz prethodnih dana i rešite je sada (npr. ako ste preskočili nešto). Čitanje: *"Cracking the Coding Interview"* (nije nužno za junior freelancing, više za intervjue, ali uvodi dobar način razmišljanja).

4. nedelja: Napredne teme i priprema za profesionalni rad

Cilj nedelje 4: Zaokruživanje znanja savremenog JavaScript ekosistema – modularno organizovanje koda, korišćenje klasa i OOP pristupa gde odgovara, pregled alata za build i deploy. Fokus je i na profesionalnim veštinama: čitanje i razumevanje tuđeg koda, držanje konzistentnog stila, dokumentovanje i optimizacija. Polaznik završava ovu nedelju spreman da svoje projekte predstavi u portfoliju i da nastavi učenje naprednijih tehnologija (poput frameworka ili backenda) na čvrstim osnovama.

Dan 22: Modularizacija koda – ES6 moduli

- **Veštine:** Organizacija koda u više fajlova/modula radi čistoće i lakšeg održavanja. Import/export mehanizam za deljenje koda između fajlova.
- **Koncepti: ES6 moduli:** mogućnost da svaki JS fajl bude modul koji može *izvoziti* određene vrednosti (promenljive, funkcije, klase) i *uvoziti* vrednosti iz drugih modula. Sintaksa `export` (named exports vs default export) [37] [38] i `import` [39] [40]. Primer: fajl `math.js` koji `export function add(a,b)` i `export function subtract(a,b)`, pa u `app.js` importuje te funkcije: `import { add, subtract } from './math.js';`. **Moduli u browseru:** korišćenje `<script type="module">` u HTML-u za uključivanje modulskog fajla (umesto običnog skripta). Time se kod automatski izvršava u strožem modu i dopušta import/

export. **Prednosti modula:** bolje razdvajanje odgovornosti (npr. jedan modul za API pozive, drugi za UI manipuaciju), keširanje modula od strane pregledača, mogućnost korišćenja *bundlera* (webpack/parcel) za veće projekte. Napomena: u Node.js okruženju moduli se u ES6 obliku koriste uz `"type": "module"` (što je već podešeno u našem projektu [41]).

- **Zadaci:** Podeliti **Weather App** kod na module: npr. `api.js` koji sadrži funkciju `fetchWeather(city)` (koja obavi fetch i vrati JSON rezultat), `ui.js` koji ima funkcije `showLoading()`, `showWeather(data)` i `showError(msg)`, i `app.js` koji glavni orkestra ciju (uzima input, koristi funkcije iz druga dva modula). U svakom fajlu na kraju dodati `export` odgovarajućih funkcija. U `app.js` dodati `import { fetchWeather } from './api.js';` itd. Testirati da aplikacija i dalje radi isto kroz `index.html` sa `<script type="module" src="app.js">`. Takođe, kreirati mali modul npr. `utils.js` koji eksportuje neku pomoćnu funkciju (poput formatiranja datuma ili slično) i iskoristiti je negde da se proveri import.
- **Resursi:** MDN – JavaScript modules (vodič koji objašnjava sve od osnovne sintakse do naprednih detalja, sa primerima) [39] [40]. MDN – import syntax i MDN – export syntax (referenca). Video: *JavaScript Modules Crash Course*. Razumeti da u razvoju većih aplikacija gotovo sav kod postaje modularan – pogledati strukturu nekog popularnog open-source projekta na GitHub-u da biste videli kako organizuju module i foldere.

Dan 23: Objektno-orijentisani JS – klase i prototipi

- **Veštine:** Primena OOP paradigme u JavaScript-u za modelovanje problema. Korišćenje klasa za kreiranje više objekata sličnog tipa (instanci) i razumevanje nasleđivanja.
- **Koncepti: Klase u JavaScript-u:** sintaksa `class MyClass { constructor(...) {...} method1() {...} }` [42] [43]. Kreiranje instanci preko `new MyClass(args)`. **Konstruktor:** specijalna metoda koja se automatski poziva pri instanciranju – služi za inicijalizaciju objekta (dodela this.property vrednosti) [44]. **Prototype chain:** interni mehanizam – metode definisane u klasi zapravo idu na `MyClass.prototype` i dele se između instanci. (Ne zalaziti suviše duboko, ali objasniti da JS nema klasičnu OOP implementaciju kao Java, već je prototipski, a klasa je sintaksni sloj preko prototipa [45].) **Nasleđivanje:** `class ExtendedClass extends BaseClass` – nasleđuje metode i osobine, može dodati nove ili overrajdovati. Korišćenje `super()` u konstruktoru child klase da pozove parent konstruktor [46]. **Kada koristiti klase:** kada imamo više entiteta sa sličnim svojstvima i ponašanjem – npr. više "korisnika", "proizvoda", "taskova" – možemo definisati klasu pa instancirati za svaki konkretan. U frontendu, klase su manje česte osim u kompleksnijim aplikacijama ili kada se koristi framework koji ih koristi (npr. u React klasični componenti).
- **Zadaci:** Napraviti klasu `Task` za to-do aplikaciju koja ima konstruktor `(text)` i svojstva `text` i `completed` (inicijalno false), plus metode `toggleDone()` (menja completed flag) i `toString()` (vraća lep string reprezentaciju). Zatim izmeniti kod to-do aplikacije da umesto običnih objekata koristi instancu Task klase za svaki novi zadatak (`tasks.push(new Task(taskText))`). Testirati metode: pozvati `tasks[0].toggleDone()` i videti da li se promeni status. Napraviti klasu `WeatherData` (ako ima smisla) da enkapsulira podatke dobijene sa API-ja i ima npr. metod `getFormattedTemp()` koji dodaje °C na temperaturu. OOP primer nasleđivanja: definisati klasu `Person` (sa ime, prezime) i klasu `Developer extends Person` (dodaje polje `language`), demonstrirati da Developer instanca ima i person atribute i svoj metod (npr. `code()` koji ispisuje poruku "Coding in {language}").
- **Resursi:** MDN – Classes (referenca sa primerima definisanja klasa) [47] [44]. MDN – *Using classes* (u okviru vodiča o objektima, odsek o klasama i nasleđivanju). *Eloquent JavaScript*, poglavlje **OOP in JS**, za dublje razumevanje prototipa. Napomena: iako su klase korisne, mnoge JS codebase i dalje dosta rade s "plain objects" – važno je znati oba pristupa i koristiti što je prikladnije za dati problem.

Dan 24: Radni tok i alati – od koda do produkcije

- **Veštine:** Upoznavanje s alatima koji se koriste u modernom frontendu za razvoj i produkciju: bundleri, transpileri, package manageri. Razumevanje kako da se lokalni projekat pripremi za *deploy*.
- **Koncepti: NPM (Node Package Manager):** kako se koriste paketi/moduli trećih strana. Već smo koristili ESLint preko npm. Instalacija biblioteke: npr. `npm install axios` (kao alternativu fetch-u), pojaviće se u *node_modules*. Korišćenje u kodu sa `import ... from 'axios';`. **Bundler:** alat (webpack, Parcel, Vite) koji uzima sve module i zavisnosti i spaja u jedan (ili nekoliko) JS fajlova za produkciju, često minifikovan. Prednost – podrška za starije browsere transpiliranjem kroz Babel (npr. pretvaranje ES6+ koda u kompatibilan ES5). **Build skripta:** najčešće u package.json imamo `npm run build` koji pokrene bundler i pripremi output (u našem projektu nemamo komplikovan build jer je čisti JS, ali napomenuti kako bi to izgledalo). **Deployment:** naš projekat se hostuje na GitHub Pages (postavljen je u readme link). Objasniti korake: preko Settings, odabrana grana (npr. main) i folder (ako koristi bundler, obično `dist/`). Alternativa: hostovanje na Netlify, Vercel (veoma zgodno za SPA aplikacije). **Environment varijable:** (ukratko) u produkciji API ključeve ne treba držati u client JS u čistom tekstu – za ozbiljne projekte backend proxy ili serverless funkcija. Za sada to nije kritično, ali napomenuti da API key treba čuvati (OWM key je besplatan pa nije strašno).
- **Zadaci:** Instalirati neku malu biblioteku preko npm i isprobati je: npr. `npm install dayjs` (mala lib za formatiranje datuma). U projektu, u fajlu, importovati `import dayjs from 'dayjs';` i koristiti npr. `dayjs().format('DD.MM.YYYY')` za prikaz trenutnog datuma – ovo će raditi u lokalnom dev okruženju ako otvorimo preko Live Server ekstenzije (koja zapravo koristi bundler u pozadini). Pokušati build: ako instaliramo Parcel bundler globalno (`npm install -g parcel`), možemo pokrenuti `parcel index.html` i videti da bundler sam odradi. Diskutovati output i kako se sve spakovalo. (Ovo je više demonstracija, nije obavezno gurati u repo). Konačno, proveriti da li je GitHub Pages deployment aktivan i ažuran sa poslednjim commitom – otvoriti URL i testirati funkcionalnosti.
- **Resursi:** Parcel – Get Started (jedan od najjednostavnijih bundlera, dobar za početnike). Webpack – koncepti (za razumevanje, iako se možda ne koristi odmah). Vodič: *Deploying to GitHub Pages* – koraci kako iz front-end projekta napraviti GitHub Pages sajt (napomena: naš repo je verovatno već podešen). Netlify i Vercel imaju odličnu dokumentaciju za CI/CD deploy frontenda – pročitati kada budete spremni da hostujete kompleksnije SPAs.

Dan 25: Optimizacija koda i performansi

- **Veštine:** Prepoznavanje potencijalno neefikasnog koda i njegovo poboljšanje. Korišćenje alata za merenje performansi aplikacije. Osnovne tehnike optimizacije frontenda.
- **Koncepti: Kompleksnost algoritama:** kako rastuća količina podataka utiče na brzinu (Big-O notacija – već spomenuto, ovde primeniti: npr. dvostruka petlja O(n^2) vs linearna O(n)). **Memorijska efikasnost:** paziti na nepotrebno dupliranje velikih struktura (npr. kopiranje niza od 100k elemenata bez potrebe). **Profiling u DevTools:** Performance tab – može snimiti izvršavanje aplikacije i pokazati uska grla. **Debouncing/Throttling:** tehnike da se ograniči učestalost poziva funkcije – npr. kod eventa kucanja u polje pretrage, debounce može sačekati 300ms od poslednjeg slova pre API poziva (smanjuje broj poziva). **Lazy loading:** odloženo učitavanje sadržaja koji nije odmah potreban (kod nas manje primenjivo jer su male aplikacije, ali npr. učitavanje slike ili modula tek kad zatreba). **Web Vitals:** metrike učitavanja (Largest Contentful Paint, First Input Delay itd.) – samo informativno pomenuti da postoje standardi za performanse.
- **Zadaci:** Pregledati svoj kod npr. to-do aplikacije sa 1000 zadataka (može se generisati u localStorage), videti da li ima usporenja pri renderovanju. Ako da, optimizovati: npr. ne re-renderovati celu listu na svaku promenu, nego samo dodatak/izmene (u našem kodu verovatno

se to već radi, ali to je koncept *partial update* vs *full rerender*). U weather app, dodati *debounce* na input za grad – umesto da na svako slovo šalje API, neka čeka 500ms od poslednjeg unosa (može se implementirati jednostavno putem `setTimeout` i `clearTimeout`). Meriti vreme izvršavanja nekog JS dela koda: npr. koristiti `performance.now()` pre i posle for petlje od 1e7 iteracija, ispisati razliku. Shvatiti da DOM operacije su najsporije – npr. dodavanje 1000 elemenata jedan po jedan vs. generisanje HTML stringa pa `innerHTML`. Probati obe metode i meriti sa `console.time()`.
- **Resursi:** MDN – Profiling Firefox DevTools ili sličan za Chrome (kako interpretirati flame chart). Blog post: *"Debounce and Throttle in JS"* (s objašnjenjima i kodom). Lodash biblioteka ima spremne debounce/throttle funkcije – možete je i isprobati (install lodash pa `import debounce`). Google Web Fundamentals (web.dev) ima sekciju o Performance optimizations – za dalju literaturu.

Dan 26: Pregled napretka i dokumentovanje projekata

- **Veštine:** Sumiranje naučenog, dokumentovanje koda i projekata (README, komentari), prepoznavanje preostalih slabih tačaka za fokusirati u daljem učenju.
- **Koncepti: Dokumentacija projekta:** svaki ozbiljan projekat treba README.md koji objašnjava svrhu, tehnologije, kako pokrenuti, primer slike, itd. Dodavanje takvog opisa u svoj repo (dvojezično, po uzoru na postojeći stil u repo-u) može biti od pomoći za portfolio [48] [49]. **Komentari u kodu:** balans – previše komentara loše, premalo takođe; komentarisati složenu logiku i ostaviti beleške za buduće sebe ili saradnike. **Refleksija:** šta je savladano do sada (listati tematski: sintaksa, DOM, async, API, OOP, testovi...), a šta još uvek stvara nesigurnost (npr. "još uvek se mučim sa this-om" ili "nisam baš radio sa datumima, vremenom"). Napraviti plan za te slabosti – možda dedicirati neko dodatno vreme posle 30 dana da se to utvrdi. **Priprema portfolija:** pored ovog *HTML/CSS/JS Portfolio* repozitorijuma, razmisliti o ličnom sajtu (landing page iz repozitorijuma može poslužiti) koji prikazuje tvoje projekte, sa linkovima ka deploy-ovanim verzijama i GitHubu.
- **Zadaci:** Napisati detaljan **README.md** za svaki projekat unutar repozitorijuma (ako već nisu, može i jedan centralni sa podsekcijama za svaki projekat). Ubaciti screenshotove aplikacija (koristiti folder `assets/` u repo). Proveriti da li kod ima komentare na ključnim mestima – dodati gdje bi pomoglo razumevanju. Proći još jednom kroz issue tracker (ako postoji) ili TODO komentare i riješiti preostale sitnice. Napraviti listu *"Lessons learned"* – par tačaka šta je bilo najteže a šta najzanimljivije tokom ovih 30 dana.
- **Resursi:** Make a README – primjer (najbolje prakse šta uključiti). Pogledati **README.md** datoteke popularnih GitHub projekata (za inspiraciju kako strukturirati informacije). *Technical writing* vodiči (ukoliko želiš da dokumentacija bude profesionalno sročena). U pogledu retrospektive i daljih planova: blog post *"From JavaScript Beginner to Junior Developer in X weeks"* – inspiracija kako drugi planiraju nastaviti učenje (uglavnom predlažu krenuti sa frameworkom nakon čvrstih osnova).

Dan 27: Sledeći koraci – frontend frameworkovi i backend osnove

- **Veštine:** Orijentacija u daljem učenju – šta dolazi posle vanilla JS i kako se spremiti za full-stack razvoj.
- **Koncepti: Frontend framework/library:** React, Vue ili Angular – kratko objasniti da rešavaju organizaciju većih aplikacija kroz komponente, state management i sl. Preporuka je započeti sa React (najpopularniji, mnogo poslova, puno resursa za učenje). **State management koncept:** spominje se često (Redux, Vuex) – za sada samo terminologija. **CSS preprocesori i frameworki:** možda se dotaknuti (Sass, Tailwind, Bootstrap) – za freelance poslove korisno znati, mada već ima CSS veštine. **Backend put:** Node.js + Express okvir za izradu servera, povezivanje sa bazom (npr. MongoDB ili SQL). Kako Python već zna, može i Python (Django/Flask) za backend ako želi,

ali JS full-stack bi značilo Node naučiti. **Databases:** čuvanje podataka na serveru – SQL vs NoSQL koncept. **Autentikacija, sigurnost:** za full-stack razvojnog, neophodno znati osnove sigurnosti (hash lozinki, CORS detaljnije, zaštita API ključeva itd.). **DevOps osnove:** deployment server-side aplikacije (npr. na Heroku, Railway).

- **Zadaci:** Napraviti plan učenja za narednih 3-6 meseci: odlučiti da li fokus na frontend (pa učenje npr. React i njegov ekosistem: router, state libraries, testiranje komponenti) ili širiti na backend (npr. odraditi nekoliko malih API-ja u Node/Express). Možda kombinovati – jedan full-stack mini projekt: npr. MERN stack aplikacija (Mongo, Express, React, Node) gde će ovo što je naučeno biti osnova za React deo. Ako ciljaju freelance poslove, istražiti najtraženije veštine na platformama – često su to izrada *SPA* sa React/Vue, integracije sa CMS-ovima, e-commerce itd. Napraviti profil na GitHub Pages (koristeći landing-page projekt kao lični sajt), LinkedIn profil ažurirati sa novim veštinama, razmisliti o upisu na neke online *bootcamp* ili kurs za framework radi strukture.
- **Resursi:** Roadmap.sh – Frontend Developer roadmap (vizuelni prikaz šta sve spada u putanju frontenda, može pomoći pri odabiru sledećih tema). React – zvanični tutorial (kad budeš spreman, ovaj tutorial je prirodan nastavak na ono što već znaš – komponentni pristup, state, props). Node.js – zvanična dokumentacija (ako ideš ka backend JS). Artikli na Medium/Dev.to: *"How I got my first freelance gig"*, *"Preparing for a junior dev interview"* (čisto da vidiš iskustva drugih).

Dan 28: Završni projekat – *Personal Portfolio Page*

- **Veštine:** Kombinovanje dizajnerskih i programerskih veština za izradu lične prezentacione stranice. Ujedno, primena svega naučenog u jednom projektu većeg obima.
- **Koncepti: Portfolio sajt:** stranica koja predstavlja tebe kao developera. Treba da sadrži kratku biografiju, listu projekata (sa slikama, opisima i linkovima), kontakt formu ili bar email. **Responsive design:** iskoristiti HTML/CSS znanje (Flex/Grid) da strana izgleda dobro i na mobilnom. **Interaktivnost:** par sitnih JS dodataka – npr. "scroll to top" dugme, klik na projekt otvara detalje (može i modal prozor), tamna/svijetla tema toggle (localStorage zapamtiti preferencu). **Hosting:** GitHub Pages (već koristiš) ili sopstveni domen (po želji, može se kasnije dodati). **SEO osnovno:** postaviti odgovarajući `<title>`, meta description, i probati semantički HTML da search engine lakše indeksira (npr. `<article>` za projekte).
- **Zadaci:** Isplanirati izgled – skica ili wireframe. Možeš iskoristiti već postojeći **landing-page** iz repozitorijuma kao bazu (repo navodi da ima *Responsive landing page* projekt "coming soon"). Dodati sekcije: "O meni", "Projekti", "Kontakt". U "Projekti" sekciji, ubaciti screenshot i opis za **Accordion, Calculator, To-Do, Weather** koje si napravio, i linkovati ih (ako su deployovani ili barem GitHub repo link). Dodati i druge radove ako ih ima (npr. nešto iz Python faze ili sa fakulteta). Implementirati dark mode toggle (ovo može biti mini-projekt za sebe: koristi CSS varijable za boje i JS za dodavanje klase `dark` na body, plus localStorage za pamćenje – lepa primena prethodnih znanja!). Testirati sajt na raznim uređajima/dimenzijama (Chrome devtools – responsive mode).
- **Resursi:** Frontend Mentor – *Personal Portfolio* izazovi (postoji par predložaka dizajna za portfolio stranice). GitHub Pages Guide (za custom domen ako želiš, ili opšte savete). Mozes inspiracije pronaći i na *Dribbble* ili *Behance* (pretraga "Developer portfolio"). **Scrimba** ima besplatni kurs *"Build a personal website"*.

Dan 29: Finalno testiranje i priprema za lansiranje

- **Veštine:** Detaljno testiranje svih komponenti i funkcionalnosti projekata. Popravka preostalih bugova. Psihološka priprema za puštanje rada u javnost (npr. dobijanje feedback-a).
- **Koncepti: Testiranje u realnim uslovima:** otvoriti sajtove na različitim browserima (Chrome, Firefox, Safari, Edge) i videti ima li razlika. Proveriti konzolu za eventualne warninge i error-e. **Accessibility (A11y):** proći kroz stranice sa Tab tasterom (fokus na elementima), dodati `aria-`

`label` gde je potrebno, alt opise za slike. Koristiti Lighthouse (DevTools Audits) da dobijes score za Performance, Accessibility, SEO – popraviti šta je nisko. **Deployment checklista:** suvišni console.log uklonjen, README ažuriran, verzije paketa zaključane (package-lock.json), licenca dodata ako je open-source, provera da li se ne otkrivaju poverljive stvari (API key bi trebalo da je skriven ako je backend proxy – u našem slučaju key je na klijentu, što je ok za OWM).

- **Zadaci:** Sistematski proći kroz svaku aplikaciju: Accordion (testirati više klikova, da li se samo jedan otvara ili više, ARIA a11y atributi), Calculator (testirati kombinacije operacija, decimalne brojeve ako podržano, velike brojeve), To-Do (unos vrlo dugog teksta – da li lomi dizajn, brisanje svih zadataka pa dodavanje novih, localStorage posle refresh da li ostaje, probati u drugom browseru input lokalnih slova đ/č/ć), Weather (unos grada sa razmakom, sa čudnim karakterima, veoma dugačko ime, netačan unos, offline scenario – simulirati isključen internet i videti kako app reaguje). Zabeležiti sve uočene probleme i redom ih rešiti. Ako neke ostanu neriješene, otvoriti GitHub Issue za njih (veština praćenja bugova). Kada sve izgleda stabilno, tagovati repo sa v1.0 release (GitHub Releases) – čisto kao simboličan završetak faze.
- **Resursi:** Lighthouse – kako koristiti (ugrađen u Chrome DevTools – desni klik, "Inspect", pa "Lighthouse" tab, generiši izveštaj). WebAIM Checklist (ako želiš profi pristup pristupačnosti, opširno ali korisno). Blog post: *"Launching your first web app"* – govori o poslednjim proverama. Eventualno pitati iskusnijeg developera ili mentora da pregleda kod (ako imaš takvu mogućnost, npr. na forumu ili Reddit CodeReview zajednici).

Dan 30: Evaluacija i sledeća faza razvoja

- **Veštine:** Kritičko sagledavanje sopstvenog rada, prezentacione veštine, planiranje kontinuiranog učenja.
- **Koncepti: Evaluacija:** šta je postignuto u 30 dana – rekapitulacija svih projekata i tehnologija naučenih. **Samopouzdanje:** isticanje da sada možeš samostalno napraviti dinamičku web aplikaciju od nule, što je ogroman korak ka Junior dev poziciji. **Učenje nadalje:** kontinuirano vežbanje kroz *build* i *learn* pristup – graditi sve kompleksnije projekte (npr. jednostavan CRUD web app sa loginom – tu bi ušao backend), unapređivati algoritamske veštine (Codewars dnevno par zadataka za održavanje forme). **Networking:** povezivanje s drugim programerima, traženje kod review-a, uključivanje u open-source projekte za iskustvo timskog rada. **Freelancing saveti:** izgraditi portfolio (što već radiš), početi s manjim gigovima (npr. na Upwork-u ili Fiverr-u, raditi klon neke stranice za poznanika besplatno za reference). Učiti i *soft skills*: komunikacija sa klijentima, procena vremena za zadatke, pisanje čiste dokumentacije za isporuku.
- **Zadaci:** Napraviti **prezentaciju** ili blog post o svom 30-dnevnom putu – ovo pomaže da organizuješ misli i takođe možeš podeliti na LinkedIn kao rezultat (poslodavci vole da vide i tu strast i sposobnost refleksije). U prezentaciju uključiti demo svih projekata (možeš snimiti i kratak video walk-through aplikacija). Postaviti sebi ciljeve za narednih 30 dana, 3 meseca, 6 meseci – npr: *"Za mjesec dana savladaću osnove React-a i napraviti mini SPA."*, *"Za 3 mjeseca ću imati prvi freelance posao odrađen."*, *"Za 6 mjeseci ću doprineti open-source projektu."* – ciljevi neka budu realistični ali i ambiciozni da te guraju napred. Proslaviti uspeh dosadašnjeg rada – ovo je bila intenzivna ruta i daleko si dogurao!
- **Resursi:** [Hashnode/Dev.to] – platforme gde možeš objaviti blog o svom putu ("Od građevinca do junior developera za 30 dana" može biti zanimljiva priča!). [LinkedIn Learning – Soft Skills for Developers] (ako imaš pristup, korisno za freelancing komunikaciju). **Communities:** pridruži se Facebook grupama ili Discord serverima lokalnih IT zajednica, tu možeš naći mentore, savete, možda i poslovne prilike. Održavaj radoznalost i nastavi isprobavati nove stvari – IT industrija zahteva stalno učenje, ali sada imaš odličnu osnovu za sve što dolazi. Srećno dalje u kodiranju!

Napomena za AI mentora: U ovom trenutku uspešno sam realizovao sve predviđene zadatke i projekte iz 30-dnevnog plana. Prošao sam kroz osnove JS, manipulisanje DOM-om, interaktivnost sa eventovima, savladao asinhrone pozive ka API-ju (Weather app radi odlično), uveo testiranje i debugging navike, pa čak i podelio kod u module. Moj trenutni status: sigurno baratam front-end JavaScript-om i razumem koncepte potrebne za Junior Full-Stack poziciju, iako ću backend tek početi da nadograđujem. Sledeći dan planiram fokus na **React framework** – vreme je da nadogradim svoje znanje i naučim kako se prave single-page aplikacije u industrijskom okruženju. Takođe, posvetiću vreme da započnem jedan mali **Node.js** backend projekat kako bih zaokružio full-stack veštine.

Fokus za sutrašnje učenje: Osnove React-a – instalacija razvojnog okruženja (Create React App ili Vite), koncepti komponenti i JSX sintakse, te prvi “Hello World” komponenta. Paralelno, obnoviću koncept modularnog koda i state upravljanja iz perspektive React-a.

Podsetnik: Nakon što proučim teoriju i primere za React, pokrenuću novu seriju vežbi i zadataka kroz poseban prompt – fokusirajući se na praktičnu izradu jednostavne React aplikacije (to-do listu ili sličan poznati projekat, ali ovaj put korišćenjem React paradigme) kako bih učvrstio novo gradivo kroz praksu. 📚

---

1 Data types
https://javascript.info/types

2 3 4 Control flow and error handling - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling

5 36 JavaScript on Exercism
https://exercism.org/tracks/javascript

6 20 23 Top 17 Code Challenge Resources For Developers - Dev Resources
https://devresourc.es/category/code-challenge

7 8 15 JavaScript DOM Tutorial – How to Build a Calculator App in JS
https://www.freecodecamp.org/news/javascript-dom-build-a-calculator-app/

9 10 11 12 14 Introduction to events - Learn web development | MDN
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/Events

13 Frontend Mentor | Todo app coding challenge
https://www.frontendmentor.io/challenges/todo-app-Su1_KokOW

16 Create a simple calculator using HTML, CSS, JavaScript - YouTube
https://www.youtube.com/watch?v=QS6Y0ezhyCs

17 18 19 Closures - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Closures

21 22 Window: localStorage property - Web APIs | MDN
https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

24 25 Using the Fetch API - Web APIs | MDN
https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

26 27 28 29 Building a Weather App: A Complete Guide to Weather APIs and Data Visualization - DEV Community
https://dev.to/darshil89/building-a-weather-app-a-complete-guide-to-weather-apis-and-data-visualization-15mc

30 32 33 JavaScript debugging and error handling - Learn web development | MDN
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/Debugging_JavaScript

31 41 package.json
https://github.com/josip-pavlovic-dev/html-css-js-portfolio/blob/0fd02f919f497fa040b428adc6e727e80b704fd2/package.json

34 35 Getting Started · Jest
https://jestjs.io/docs/getting-started

37 38 39 40 JavaScript modules - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

42 43 44 45 46 47 Classes - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

48 49 README.md
https://github.com/josip-pavlovic-dev/html-css-js-portfolio/blob/0fd02f919f497fa040b428adc6e727e80b704fd2/README.md