Workout: Fundamentele Proiectului și GitHub

Introducere

Hey, acest workout va fi diferit, deoarece va fi primul în care lucrăm la proiectul complet. Vom începe astăzi prin construirea bazelor acestuia și vom continua în fiecare săptămână cu adăugarea din ce în ce mai multor funcționalități.

Despre Proiect

Vom construi aceeași aplicație web simplă prezentată în modulul de orientare: un **Weather App** care:

- Afișează vremea curentă (tempertura, cum se simte, descrierea, iconița cu vremea)
- Afișează date extra (umiditatea, presiunea, viteza vântului, vizibilitatea, ora răsăritului și al apusului)
- Permite obținerea locației utilizatorului
- Oferă opțiunea de căutare pentru alte orașe
- Are un istoric cu căutările recente

Ce vei învăța

- Structura unui proiect JavaScript modern
- Arhitectura modulară și separarea responsabilităților
- Fundamentele Git și GitHub
- Cum să construiești o aplicație web funcțională de la zero

🎯 **Filosofia noastră**: Înveți prin descoperire ghidată - îți oferim întrebările potrivite și testele de validare, tu găsești răspunsurile!

De ce această abordare?

- **Construiești înțelegere reală**, nu doar copiezi cod
- **Dezvolți instinctul de debugging** - vital pentru orice dezvoltator
- **Câștigi încredere** în capacitatea ta de a rezolva probleme
- **Înveți să gândești modular** și să structurezi codul profesional

Exemplu cum ar putea arăta aplicația

Partea 1: Git și GitHub - Fundamentele

De ce sunt esențiale?

Git este un sistem de control al versiunilor care te ajută să urmărești modificările din codul tău. Imaginează-ți că ai un document Word și vrei să păstrezi toate versiunile pe care le-ai făcut - Git face exact asta pentru cod.

GitHub este o platformă online unde poți stoca repository-urile Git. Este ca un "cloud storage" pentru dezvoltatori, dar mult mai puternic.

De ce sunt importante?

1. **Istoricul modificărilor**: Poți vedea exact ce ai schimbat și când

2. **Backup automat**: Codul tău este sigur în cloud
3. **Colaborare**: Multiple persoane pot lucra la același proiect
4. **Portfolio**: GitHub este CV-ul tău ca dezvoltator

Pas cu pas: Primul tău proiect Git

🔧 Pas 1: Instalarea Git

Windows:

1. Mergi pe git-scm.com
2. Descarcă și instalează cu setările default
3. Restartează VS Code după instalare

Mac:

1. Deschide Terminal și scrie: `git --version`
2. Dacă nu e instalat, va apărea o fereastră pentru instalare
3. Sau instalează din git-scm.com

Linux:

```
sudo apt update sudo apt install git
```

📁 Pas 2: Crearea proiectului

1. **Creează un folder nou** numit `weather-app`
2. **Deschide VS Code**
3. **File → Open Folder** și selectează folderul `weather-app`
4. VS Code se va deschide cu folderul gol

💻 Pas 3: Deschiderea terminalului

În VS Code:

1. **View → Terminal** (sau `Ctrl + tab` pe Windows/Linux, `Cmd + tab` pe Mac)
2. Se va deschide terminalul în partea de jos

3. Ar trebui să vezi ceva ca: `C:\\Users\\UserName\\Projects\\weather-app>` (Windows) sau `/Users/UserName/Documents/weather-app` (Mac/Linux)

🚀 Pas 4: Inițializarea Git

În terminal, scrie:

```
git init
```

✅ **Succes** - Vei vedea:

- Initialized empty Git repository in /path/to/weather-app/.git/

❌ **Eroare** - "git: command not found":

- Git nu e instalat sau VS Code nu l-a detectat
- Restartează VS Code după instalare
- Pe Windows: verifică că Git e în PATH

📝 Pas 5: Configurarea Git (doar prima dată)

```
git config --global user.name "Numele Tău" git config --global user.email "em
ail@example.com"
```

📄 Pas 6: Crearea fișierelor de bază

În VS Code, creează fișierele:

1. **Clic dreapta în Explorer** → New File
2. Creează fișierele unul câte unul:
   - `index.html`
   - `styles.css`
   - `app.js`
   - `README.md`

3. Creează folderul modules:
    - Clic dreapta → New Folder → `modules`
    - În modules, creează:
        - `config.js`
        - `weather-service.js`
        - `ui-controller.js`

💡 **Hint**: Structura finală ar trebui să semene cu:

```
weather-app/ ├── index.html ├── styles.css ├── app.js ├── modules/ │ ├── conf
ig.js │ ├── weather-service.js │ └── ui-controller.js └── README.md
```

🔍 Pas 7: Verificarea statusului

```
git status
```

Ar trebui să vezi:

```
Untracked files: (use "git add <file>..." to include in what will be committe
d) README.md app.js index.html modules/ styles.css
```

Comenzile git esențiale:

```
git init # Inițializează un repository nou git add . # Pregătește fișierele p
entru commit git commit -m "Mesaj" # Salvează modificările git push # Încarcă
modificările pe GitHub git pull # Descarcă ultimele modificări git status # V
ezi statusul fișierelor
```

Partea 2: Partea vizuală

HTML - Scheletul Aplicației

Obiectiv: Pagină funcțională cu toate elementele UI necesare

Întrebări pentru reflecție:

- Ce acțiuni poate face un utilizator într-o aplicație meteo?
- Ce informații vrea să vadă?
- Cum structurezi logic elementele?
- Ce elemente UI ai nevoie pentru o aplicație meteo?
  - Un loc să introduci numele orașului?
  - Un buton să cauți vremea?
  - Un buton să folosești locația curentă?
  - Un loc să afișezi rezultatele?
  - Ceva pentru loading și erori?

Specificații tehnice:

- HTML5 valid cu meta viewport
- Link-uri către CSS și JavaScript (`type="module"`)
- Elementele UI necesare (cu ID-urile specifice):
  - Un input pentru numele orașului (`id="city-input"`)
  - Un buton de search (`id="search-btn"`)
  - O zonă pentru loading (`id="loading"`)
  - O zonă pentru erori (`id="error"`)
  - O zonă pentru afișarea vremii (`id="weather-display"`)
    - Câte un ID pentru fiecare element informativ (temperatura, umiditatea, apus, etc...)

💡 **Template de pornire:**

```
<!DOCTYPE html> <html lang="ro"> <head> <!-- Meta tags aici --> <title>Weathe
r App</title> <!-- Link-uri către CSS și JS --> </head> <body> <div id="app">
<!-- Structura ta aici --> </div> </body> </html>
```

🧪 **Validare:**

1. Deschide în browser → Se încarcă fără erori? (Poți folosi extensia Live Server (fire server) din VS Code)
2. F12 → Console → Fără erori roșii?
3. Vezi toate elementele pe pagină?

💡 **Hints pentru succes:**

- Grupează în `<header>` și `<main>`
- Adaugă `class="hidden"` pentru elemente inițial ascunse

CSS - Transformarea Vizuală (opțional)

Obiectiv: Deși acest aspect este opțional, poate face diferența dintre un exercițiu și un proiect profesional

Provocări de design:

1. **Ce culori** asociezi cu o aplicație meteo?
2. **Cum faci** butoanele să pară clickable?
3. **Ce layout** funcționează pe telefon ȘI desktop?

💡 **Template de pornire:**

```
* { margin: 0; padding: 0; box-sizing: border-box; } .hidden { display: none;
} /* Stilurile tale aici */
```

💡 **Hints pentru design:**

- **Background**: Încearcă un gradient frumos (ex: albastru → albastru închis)
- **Input-uri**: Border-radius pentru aspect modern, padding generos
- **Butoane**: Culori diferite pentru acțiuni diferite, hover effects
- **Loading**: O animație CSS pentru spinner (rotație infinită)
- **Layout**: Centrat, cu max-width pentru desktop

🧪 **Validare:**

1. **Visual**: Arată mult mai bine după CSS?
2. **Responsive**: Redimensionează fereastra - se adaptează?

3. **Interactive**: Hover pe butoane - se schimbă?
4. **Hidden class**: În DevTools, adaugă `hidden` - dispare elementul?

Partea 3: JavaScript Modular

Config Module

Obiectiv: Centralizarea setărilor aplicației

Întrebări fundamentale:

- De ce să separi configurația?
- Ce setări va avea aplicația?
- Cum testezi un modul înainte să-l folosești?

Provocarea ta (`modules/config.js`):

- Exportă un obiect "MOCK_DATA" care să conțină un simularea unui răspuns (mock), asemănător cu ce ar răspunde un API de vreme
- Intră pe https://openweathermap.org/api și vezi cum arată un asemenea răspuns pentru API-ul "Current Weather Data"

💡 **Structură sugerată:**

```
// Cum arată datele unei API meteo? // Temperatură, umiditate, vânt, descrier
e... export const MOCK_DATA = { main: { // ... }, weather: [ // ... ], // ...
}
```

🧪 **Validare:**
Console test:

```
import('./modules/config.js').then((config) => { console.log('MOCK_DATA:', co
nfig.MOCK_DATA) })
```

✅ **Succes**: Vezi obiectele în console
❌ **Eroare**: "Failed to resolve module" - verifică path-ul

Weather Service

Obiectiv: Funcții pentru obținerea datelor meteo (deocamdată simulate)

Concepte cheie:

- De ce să simulez un API?
- Cum fac o funcție să pară asincronă?
- Cum gestionez erorile?

Funcții de implementat:

```
export const getCurrentWeather = async (city) => { // Simulează delay API (~1 secundă) // Returnează MOCK_DATA cu numele orașului actualizat // Gestionează erorile } export const getWeatherByCoords = async (lat, lon) => { // Similar, dar pentru coordonate }
```

💡 **Hint pentru delay:**

```
await new Promise((resolve) => setTimeout(resolve, 1000))
```

🧪 **Validare pas cu pas:**

```
import('./modules/weather-service.js').then((service) => { console.time('weather-test') service.getCurrentWeather('Cluj').then((data) => { console.timeEnd('weather-test') // ~1000ms? console.log('Received data:', data) console.log('City updated?', data.name === 'Cluj') }) })
```

- Durează ~1 secundă să primești răspunsul?
- Numele orașului s-a actualizat?

UI Controller

Obiectiv: Separarea completă a logicii de interfață

Filosofia separării:
De ce să nu pui totul în app.js? Pentru că UI-ul se schimbă, logica rămâne! Iar în timp aceste fișiere vor crește și va fi tot mai greu să le ai pe toate într-un singur loc.

Funcții/obiecte esențiale:

1. `elements` - păstrează toate elementele într-un singur obiect
2. `showLoading()` / `hideLoading()` - stări de încărcare
3. `showError()` - gestionarea erorilor
4. `displayWeather()` - afișarea datelor meteo
5. `getCityInput()` / `clearInput()` - management input

Provocarea mare - `displayWeather()`:
Cum mapezi proprietățile acestui obiect pe elementele DOM existente:

```
{ name: "Oradea", main: { temp: 22, humidity: 65 }, weather: [{ description:
"senin" }], wind: { speed: 3.2 } }
```

Într-un HTML frumos pentru utilizator?

💡 **Hints:**

Pentru `elements`:

```
export const elements = { cityInput: document.querySelector('#city-input'),
// ... restul elementelor }
```

- Găsește fiecare câmp (temperatură, descriere, icon, etc.) în DOM și actualizează proprietățile `textContent`, `src`, etc. cu datele corespunzătoare
- Include: numele orașului, temperatura, descrierea, detalii (umiditate, vânt, etc.)
- Convertește m/s în km/h (×3.6)
- Formatează timpurile Unix în ore locale (pentru răsărit și apus)

🧪 **Validare progresivă:**

1. **Elementele:**

```
import('./modules/ui-controller.js').then((ui) => { const elements = ui.el
ements console.log('Elements found:', Object.keys(elements)) })
```

2. **Loading/Error:**

```
ui.showLoading() // Apare? ui.showError('Test') // Apare?
```

3. **Weather Display:**

```
import('./modules/config.js').then((config) => { ui.displayWeather(config.
MOCK_DATA) // Arată frumos? })
```

Main App

Obiectiv: Orchestrarea completă a aplicației

Rolul app.js:
"Dirijorul orchestrei" - coordonează toate modulele să lucreze în armonie.

Funcții principale:

```
const setupEventListeners = () => { // Submit în form (enter din search field
sau click pe buton) } const handleSearch = async () => { // Validează input
// Arată loading // Apelează weather service // Ascunde loading, arată rezult
at // Gestionează erorile } const isValidCity = (city) => { // Gol? Prea scur
t? Conține cifre/simboluri? return city.length >= 2 && /^[a-zA-ZăâîșțĂÂÎȘȚ\\s
-]+$/.test(city) } // Pornește setupEventListeners și displayWeather pentru a
rula aplicația
```

🧪 **Validare completă:**

1. **Start**: Se încarcă vremea pentru orașul default?
2. **Search**: Introdu oraș → loading → rezultat?

3. **Validare**: Input gol → eroare clară?
4. **Enter**: Funcționează ca alternativă la click?

Partea 4: Git și GitHub

Repository Setup

Obiectiv: Să ai codul publicat pe GitHub ca un dezvoltator adevărat

Pașii:

1. **Creează cont pe** github.com
2. **Creează repository nou:**
   - Nume: `weather-app`
   - Descriere: `A modern weather application built with vanilla JavaScript`
   - Public ✅ (să îl poată vedea toată lumea)
   - Add README file ✅

🔑 Autentificarea la primul push

La primul `git push` Git trebuie să știe cine ești și să-ți verifice drepturile pe repository. Ai două opțiuni principale: HTTPS sau GIT cu cheie SSH. Eu vă recomand să folosiți prima opțiune:

- În pagina repo-ului apasă **Code** → **HTTPS** și copiază URL-ul care arată așa:

```
https://github.com/USERNAME/weather-app.git
```

- Adaugă remote-ul:

```
git remote add origin https://github.com/USERNAME/weather-app.git
```

- La primul `git push -u origin main` GitHub va cere **username** și **Personal Access Token (PAT)** în loc de parolă.

1. Mergi la **GitHub** → **Settings** → **Developer settings** → **Personal access tokens** → **Tokens (classic)**.

2. „Generate new token” (scope minim `repo`). Copiază token-ul (se arată o singură dată!).
3. Când terminalul cere parola, lipește token-ul.
   • macOS va salva token-ul în Keychain (via credential helper) → nu mai trebuie introdus data viitoare.
   • Windows: Git for Windows include Git Credential Manager. După ce introduci PAT-ul, se deschide o fereastră de autentificare, iar token-ul este stocat în Windows Credential Manager (credential helper `manager-core`), astfel încât la următoarele push-uri nu va mai fi cerut.

🧪 **Cum să testezi că e configurat corect:**

1. **Primul commit local:**

```
git add . git status # Verifică că toate fișierele sunt "staged" git commi
t -m "Initial project setup with modular structure" git log # Verifică că
commit-ul s-a făcut
```

2. **Conectarea la GitHub:**

```
git branch -M main # alege UNA dintre metodele HTTPS/SSH de mai sus git pu
sh -u origin main # se va cere autentificare dacă e prima dată
```

3. **Verifică pe GitHub** că toate fișierele au fost încărcate

✅ **Semnele că ai făcut corect:**

- Repository-ul e vizibil pe GitHub
- Toate fișierele sunt prezente
- Commit history arată commit-ul tău

❌ **Probleme**: Permission denied? URL corect? Git configurat?

README Profesional

Obiectiv: Documentație care impresionează

Întrebări cheie:
Un străin găsește proiectul tău - ce vrea să știe?

- Ce face aplicația?
- Cum o instalează/rulează?
- Ce tehnologii folosește?
- Cine ești tu?

Structură obligatorie:

```
# Weather App ⛈️ [Descriere captivantă] ## 🚀 Features - [ ] Lista cu ce fun cționează ACUM ## 🛠️ Tech Stack - [Lista tehnologiilor] ## 📦 Instalare [Paș i clari pentru oricine] ## 🗺️ Roadmap - [x] Part 1: Fundamente - [ ] Part 2: API real - [ ] Part 3 - [ ] Part 4 ## 👩‍💻 Autor [Info despre tine]
```

Final Validation - Checklist

✅ Funcționalitate

- ☐ Start automat cu vremea default
- ☐ Search pentru orice oraș
- ☐ Validare input + erori clare
- ☐ Loading states pentru toate acțiunile

✅ Cod

- ☐ 4 module JavaScript conectate
- ☐ ES6 imports/exports
- ☐ Arhitectură modulară
- ☐ Cod curat și comentat
- ☐ Zero erori în consolă

✅ Git & GitHub

- ☐ Repository public
- ☐ Commit history curat
- ☐ README complet

- [ ] Toate fișierele sincronizate

- [x] **UX & Design**
- [ ] Interfață modernă
- [ ] Responsive design
- [ ] Interacțiuni intuitive
- [ ] Feedback vizual pentru toate acțiunile