ChatGPT

Guida tecnica: CSS per Portfolio Monopagina di Product Designer

1. Struttura Mobile-First e Breakpoint

- **Mobile-first:** definire le regole CSS di base per dispositivi piccoli (smartphone, fino a 480px) e aggiungere *media query* in ordine di ampiezza crescente. Questo approccio semplifica la manutenzione e segue le best practice del responsive design [1].
- **Breakpoint principali:** utilizzare le soglie richieste con `@media`. Ad esempio, `@media (min-width: 481px) and (max-width: 736px)` per tablet verticali, `@media (min-width: 737px) and (max-width: 1279px)` per tablet orizzontali, `@media (min-width: 1280px) and (max-width: 1689px)` per desktop, e `@media (min-width: 1690px)` per schermi extra-large. Questi valori (480px, 768px, 1024px, 1280px, ecc.) sono comunemente usati nel design responsivo [2].
- **Layout flessibile:** utilizzare layout a griglia (CSS Grid) o Flexbox, che sono intrinsecamente responsive [3]. Ad esempio, usare `display: flex;` per l'header o per il carosello progetti, oppure `display: grid;` con `grid-template-columns` variabili per griglie di competenze/interessi. A ogni breakpoint, modificare il numero di colonne o la direzione del flex per adattare il contenuto (ad es. una sola colonna su smartphone, due/tre su tablet, quattro su desktop).
- **Menu e navigazione:** sullo smartphone, prevedere una versione compatta del menu (es. *hamburger*), mentre su tablet e desktop mostrare il menu orizzontale completo. Allo stesso modo, lo switcher lingua può essere un piccolo link o dropdown a destra dell'header.

2. Palette colori e tema chiaro/scuro

- **Colori base e variabili:** definire una palette neutra (toni di grigio, bianco/nero) tramite *CSS custom properties*. Ad esempio in `:root` impostare:
- `--bg-primary`, `--bg-secondary` per gli sfondi principali e secondari.
- `--text-primary`, `--text-secondary` per i colori del testo principale e secondario. Definendo i colori in variabili, è facile cambiarli centralmente [4]. La palette neutra (bianco sporco, grigio chiaro, nero intenso) assicura leggibilità sia in modalità chiara che scura.
- **Tema chiaro (default):** impostare in `:root` colori di sfondo chiari e testo scuro, per esempio:

  ```
  --bg-primary: #ffffff; --bg-secondary: #f5f5f5; --text-primary: #333333; --text-secondary: #666666;
  ```

- **Tema scuro:** usare la media query `@media (prefers-color-scheme: dark)` per ridefinire le variabili in modalità scura [5] [6]. Ad esempio:

  ```
  @media (prefers-color-scheme: dark) { :root { --bg-primary: #111111; --bg-secondary: #222222; --text-primary: #eeeeee; --text-secondary: #bbbbbb; } }
  ```

  MDN raccomanda di usare `prefers-color-scheme` per rilevare la preferenza dell'utente e adattare i colori di sfondo/testo [6]. In questo modo il passaggio tema chiaro/scuro avviene senza modificare il markup HTML.

- **Metatag `color-scheme`:** per evitare lampeggi durante il caricamento, si può aggiungere `<meta name="color-scheme" content="light dark">` nell'HTML e `:root { color-scheme: light dark; }` nel CSS, come illustrato in MDN [5].

3. Tipografia leggibile e moderna

- **Font principale:** usare `font-family: 'Poppins', sans-serif` (già incluso) per titoli e testi. Poppins è un sans-serif geometrico moderno adatto a UI leggibili.
- **Dimensioni testo:** impostare `font-size: 1rem` (circa 16px) per il testo base. La normativa sull'accessibilità suggerisce almeno 16px per il corpo testo [7]. Usare unità relative (`rem` o `%`) per consentire lo zoom e l'accessibilità.
- **Line-height e spaziatura:** usare `line-height` di circa 1.5 per garantire leggibilità su blocchi di testo. Aggiungere margini verticali costanti tra paragrafi (es. `margin-bottom: 1em`). Separare i titoli dai paragrafi con `margin` coerenti.
- **Scale tipografiche:** prevedere dimensioni scalate per i titoli: ad esempio `h1 { font-size: 2rem; }`, `h2 { font-size: 1.75rem; }`, ecc., scalando verso il basso per i titoli minori. Su dispositivi piccoli ridurre leggermente le dimensioni per adattarsi (es. `@media (max-width: 480px) { h1 { font-size: 1.5rem; } }`).
- **Allineamento testo:** allineare il testo a sinistra per una lettura più naturale [8]. Evitare giustificazioni piene che possono creare spaziature inconsistenti tra le parole.
- **Altri stili:** per uniformità, impostare `letter-spacing` moderato (p.es. `0.5px` nei titoli, `0` nel corpo). Usare variabili CSS per i colori di link e titoli (`--text-accent` ecc.) e stili di hover (cambi di colore via JS/ :hover).

4. Layout flessibile e sezioni

Utilizzare Flexbox e Grid per strutturare ogni sezione, come suggerito da MDN [3]. Di seguito le sezioni chiave con le relative classi e suggerimenti di layout:

Header

- **Classe:** usare `.header` come contenitore principale (es. elemento `header`).
- **Struttura interna:** al suo interno, elementi come `.logo` (a sinistra), `.nav-menu` (al centro/ destra) e `.lang-switcher` (in alto a destra).
- **Stile:** `.header { display: flex; align-items: center; justify-content: space-between; padding: 1rem; }`. In mobile il menu può essere nascosto o ridotto (`.nav-menu { display: none; }`) e sostituito da un'icona *hamburger*; negli altri breakpoint `.nav-menu` diventa visibile e disposto orizzontalmente.
- **Responsività:** su smartphone (<480px) usare flex in colonna o nascondere menu; su tablet in orizzontale e su desktop margin e padding più ampi. Ad esempio, `@media (min-width: 737px) { .nav-menu { display: flex; gap: 1rem; } }`.

Hero (Sezione introduttiva)

- **Classe:** `.hero`.
- **Stile di base:** sfondo a schermo intero con immagine (`background-image: url(...); background-size: cover; background-position: center; min-height: 100vh; position: relative;`).
- **Contenuto:** includere un contenitore `.hero-content` con titolo (`.hero-title`) e sottotitolo (`.hero-subtitle`). Centrare verticalmente e orizzontalmente il testo usando flex o grid

(`.hero-content { display: flex; flex-direction: column; align-items: center; justify-content: center; height: 100%; }`).
- **Testo su immagine:** per garantire contrasto, applicare un leggero overlay scuro (es. usando un ::before semitrasparente) o colori di testo chiari (`color: var(--text-primary)`) in tema scuro. Impostare dimensioni del titolo grandi (es. `h1 { font-size: 2.5rem; }` su desktop, ridotto su mobile).
- **Call-to-action:** se presente un pulsante (es. "Scopri di più"), stilizzarlo come `.hero-cta` con padding, bordo arrotondato e cambio colore di sfondo su hover via JS.

Sezione "Chi Sono" (About)

- **Classe:** `.about`.
- **Struttura:** contenitore flessibile con immagine `.about-image` e descrizione `.about-text`.
- **Layout:** usare `display: flex; flex-direction: column; align-items: center; text-align: center;` di default (mobile). Ad esempio: `.about { display: flex; flex-direction: column; align-items: center; padding: 2rem; }`.
- **Breakpoint (>737px):** cambiare in `flex-direction: row; align-items: flex-start;` per affiancare l'immagine e il testo. Esempio: `@media (min-width: 737px) { .about { flex-direction: row; } .about-image, .about-text { flex: 1; } }`. Aggiungere `margin` tra immagine e testo.
- **Dettagli:** arrotondare bordi di `.about-image`, mantenere titolo e testo in `.about-text` con stili coerenti (titolo h2, paragrafi, eventuali link). Usare variabili colore per testo e sfondo.

Timeline interattiva

- **Classe:** `.timeline`.
- **Struttura:** lista verticale di eventi `.timeline-item`, ciascuno con una sottoclasse `.timeline-date` e `.timeline-content`.
- **Linee guida CSS:** usare `position: relative;` per `.timeline`, con un pseudo-elemento (::before) che disegna la linea verticale centrale (`position: absolute; left: 50%; width: 2px; background: var(--text-secondary); height: 100%;`).
- **Elementi:** `.timeline-item { position: relative; margin: 2rem 0; width: 100%; }`. Dentro, allineare la data e il contenuto a lati opposti: ad esempio, `.timeline-item:nth-child(odd) .timeline-date { text-align: left; } .timeline-item:nth-child(odd) .timeline-content { margin-left: 60px; }` (e viceversa per elementi pari). Per semplicità si può anche optare per un layout semplificato verticale, centrando tutti i contenuti.
- **Stile dinamico:** usare cerchi (`.timeline-date::before`) come indicatori sulle date, nascondere/mostrare dettagli con classi CSS (es. `.timeline-item.open .timeline-content { display: block; }`, di default `display: none;`).

Slideshow progetti

- **Classe:** `.projects-slider` (contenitore) e `.project-slide` (elementi).
- **Layout:** impostare `.projects-slider { display: flex; overflow-x: hidden; scroll-snap-type: x mandatory; }` e `.project-slide { flex: 0 0 100%; scroll-snap-align: start; }` per avere slide a piena larghezza. Il JavaScript gestirà lo scroll o il carosello attivo.

- **Stili:** ciascun `.project-slide` contiene immagine `.project-image` e `.project-info`. Rendere le immagini responsive con `width: 100%; height: auto`. Aggiungere padding uniforme e titolo progetto come `.project-info h3`.
- **Media Queries:** su schermi più larghi si potrebbero mostrare più slide contemporaneamente (es. `.project-slide { flex: 0 0 50%; }` a partire da 1280px). Adattare layout secondo necessità.

Griglia Competenze (Software)

- **Classe:** `.skills-grid`.
- **Struttura:** lista di elementi `.skill-item` che contengono il nome del software e una valutazione a stelle `.skill-stars`.
- **Layout:** usare `display: grid; grid-template-columns: repeat(auto-fit, minmax(150px, 1fr)); gap: 1rem;`. Ad esempio, 2 colonne su mobile, 3–4 su desktop.
- **Stelle:** ogni `.skill-item .skill-stars` può essere una riga di icone (o caratteri ★). Stilizzare le stelle riempite e vuote con colori diversi. Si possono usare pseudo-elementi (`.star::before { content: '★'; }`) e classi per il livello (es. `.star.filled`).
- **Testo:** `.skill-name { text-align: center; margin-top: 0.5rem; font-weight: bold; }`. Usare variabili per i colori (es. stelle gialle su background neutro).

Griglia Interessi

- **Classe:** `.interests-grid`.
- **Struttura:** simile alle competenze, ma con icone. Ogni `.interest-item` contiene un'icona (`<i>` o `<img>`) e una didascalia `.interest-name`.
- **Layout:** `display: grid; grid-template-columns: repeat(auto-fill, minmax(100px, 1fr)); gap: 1.5rem; justify-items: center;`. Due colonne su mobile, salendo su più colonne per tablet/desktop.
- **Icone:** impostare dimensioni coerenti (`width/height: 50px`), centro con margini. Testo sotto icona allineato al centro.

Sezione Contatti (Form)

- **Classe:** `.contact` contenitore e `.contact-form`.
- **Form:** usare `display: grid` per allineare gli input. Ad esempio:

```
.contact-form {
  display: grid;
  grid-template-columns: 1fr;
  gap: 1rem;
  max-width: 600px;
  margin: 0 auto;
}
```

  Poi, a schermi più grandi: `@media (min-width: 737px) { .contact-form { grid-template-columns: 1fr 1fr; } }` per affiancare due campi per riga.
- **Stili input:** `.form-field, .form-field textarea { width: 100%; padding: 0.75rem;`

`border: 1px solid var(--text-secondary); border-radius: 4px; font-size: 1rem; }`. Su focus: `outline: none; border-color: var(--text-primary);`.
- **Pulsante invio:** `.form-submit { padding: 0.75rem 1.5rem; background: var(--text-primary); color: #fff; border: none; border-radius: 4px; font-weight: bold; cursor: pointer; }`. Su hover (gestito via JS o CSS): modificare opacità o colore.

FAQ (domande cliccabili)

- **Classe:** `.faq` e ogni elemento `.faq-item` con `.faq-question` e `.faq-answer`.
- **Stile domande:** `.faq-question { cursor: pointer; padding: 1rem; background: var(--bg-secondary); border-bottom: 1px solid var(--text-secondary); }`. Aggiungere un'icona o simbolo (+/-) con pseudo-elemento CSS.
- **Risposte:** `.faq-answer { display: none; padding: 1rem; background: var(--bg-primary); }`. Lo script JS aggiungerà la classe `.open` a `.faq-item` al click, quindi in CSS:

```
.faq-item.open .faq-answer { display: block; }
```

- **Accessibilità:** assicurarsi che `.faq-question` sia evidenziabile (outline) quando selezionata e che il testo sia leggibile anche in contrasto.

Footer

- **Classe:** `.footer`.
- **Layout:** usare `display: flex; flex-direction: column; align-items: center; padding: 2rem 1rem;` di base (mobile). In breakpoint superiori, passare a `flex-direction: row; justify-content: space-between;` per affiancare `.footer-nav` (lista di link) e `.footer-social` (icone social).
- **Footer-nav:** `.footer-nav ul { list-style: none; padding: 0; display: flex; flex-wrap: wrap; gap: 1rem; } a { color: var(--text-secondary); }`.
- **Social icons:** `.footer-social i { font-size: 1.25rem; color: var(--text-secondary); margin: 0 0.5rem; }`. Disporre orizzontalmente con `display: flex; gap: 1rem;`.

5. Animazioni e Interattività

- **No CSS transitions:** non definire proprietà `transition` o `animation` nel CSS. Tutte le animazioni (cambio slide, espansione FAQ, toggle tema, ecc.) saranno gestite via JavaScript, come richiesto. In CSS mantenere stili statici per gli stati (`:hover`, `.open`, ecc.), lasciando al JS il compito di cambiare classi o proprietà dinamicamente.

6. Media Queries e adattività finale

- Applicare i breakpoint identificati modificando proprietà rilevanti: ad esempio ingrandire i font, cambiare `grid-template-columns`, o modificare padding/margini. Nel CSS, ordinare i *media queries* dal più piccolo al più grande (mobile-first). Per esempio:

```
/* Fino a 480px: base */
/* Da 481px a 736px */
@media (min-width: 481px) {
```

```
  /* adattamenti per tablet verticali */
}
@media (min-width: 737px) {
  /* adattamenti per tablet orizzontali e oltre */
}
@media (min-width: 1280px) {
  /* desktop ampio */
}
@media (min-width: 1690px) {
  /* schermi molto grandi */
}
```

- In ognuna, regolare il layout (es. `.about` diventa row, aumentano le colonne delle griglie, spazio extra ai margini). Evitare breakpoints fissi troppo specifici ai dispositivi: meglio basarli sulla resa del contenuto [9], ma qui seguiamo i range indicati.

7. Gestione Tema Chiaro/Scuro

- Utilizzare le variabili definite in `:root` e in `@media (prefers-color-scheme: dark)` per passare fluidamente da light a dark. Ad esempio, definendo in chiaro `--text-primary: #000; --bg-primary: #fff` e nel blocco dark invertendoli (`#fff`, `#000`) [5] [10].
- Su ogni elemento testuale usare `color: var(--text-primary)` e su sfondi `background-color: var(--bg-primary)`. Così, cambiando solo le variabili lo stile generale si aggiorna. MDN suggerisce proprio di usare `@media (prefers-color-scheme)` per impostare foreground e background [5] [6].
- Verificare che icone e immagini (se in tema) abbiano contrasti adeguati anche in dark mode, eventualmente cambiando semplicemente invertendo i colori tramite attributi `filter` o versioni alternative.

Fonti e riferimenti: linee guida CSS e accessibilità (MDN, USWDS) [1] [7] [4] [6].

---

[1] [3] [9] Responsive design - Learn web development | MDN
https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design

[2] Breakpoint: Responsive Design Breakpoints in 2023 | BrowserStack
https://www.browserstack.com/guide/responsive-design-breakpoints

[4] Using CSS custom properties (variables) - CSS: Cascading Style Sheets | MDN
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_cascading_variables/Using_CSS_custom_properties

[5] [6] [10] color-scheme - CSS: Cascading Style Sheets | MDN
https://developer.mozilla.org/en-US/docs/Web/CSS/color-scheme

[7] [8] Typography | U.S. Web Design System (USWDS)
https://designsystem.digital.gov/components/typography/