🐾Adopte ton Monstre — Cahier des charges & Spécifications

> **Contexte** : Projet fil rouge du module *M1 DFS – Développement Front-end avancé avec TypeScript & frameworks (Next.js)*, réalisable en ~**49h**. On vise **30% théorie / 70% pratique**, avec un livrable propre, déployé, testé et documenté.

---

0) Pitch & objectifs pédagogiques

Pitch : chaque utilisateur adopte un **monstre virtuel** (type Tamagotchi). Il doit **prendre soin** de lui (manger, jouer, dormir), compléter des **missions quotidiennes**, et maintenir une **humeur** élevée pour grimper au **classement**. L'app marche **offline (PWA)**, synchronise les actions quand la connexion revient, et propose des **notifications** (opt-in) quand le monstre a faim.

Objectifs pédagogiques (alignés au cours) : - Maîtriser **Next.js + TypeScript** avec mix **SSR/SSG/ISR/CSR**. - Mettre en place **React Query** (ou SWR) + éventuellement **Zustand** pour l'état local. - Construire une **PWA** (manifest, service worker, stratégies de cache) et un **mode offline**. - Optimiser **performance, accessibilité, SEO**, et **éco-conception** (EcoIndex/Lighthouse). - Écrire des **tests E2E** (Cypress/Playwright) + quelques **tests unitaires**. - Intégrer une **CI/CD** (GitHub Actions) + **déploiement** (Vercel/Netlify).

Contraintes : MVP **réalisable en 49h** à 2–4 étudiant·e·s, avec un scope clair et des bonus optionnels.

Livrables : dépôt Git public, app déployée, README complet (setup, scripts, choix techniques), captures/Lighthouse, checklists a11y/SEO, exports de tests.

---

1) Périmètre fonctionnel (MVP)

Rôles

- **Visiteur** : consulte l'accueil, la FAQ, le classement public, la page d'un monstre public.
- **Joueur** (authentifié) : adopte un monstre, gère son monstre (actions), voit son tableau de bord, complète des missions, consulte son historique, configure les notifications.
- **Admin** (optionnel/simple) **: modération basique (masquer monstre/commentaire si abus), accès à un dashboard minimal (stats).** Hors MVP** si manque de temps.

Parcours clés (MVP)

1. **Onboarding / Adoption**
2. Création de compte (ou mode invité → adoption ensuite) ;
3. Génération d'un **monstre** (nom, avatar, couleur, traits de personnalité) ;

4. Confirmation et arrivée sur le **Dashboard Monstre**.

5. **Dashboard Monstre**

6. Statuts en temps réel : **humeur**, **faim**, **énergie**, **propreté** (ou 3 stats au minimum) ;
7. Actions : **Nourrir**, **Jouer**, **Dormir** (cooldowns & coûts), effets immédiats ;
8. **Journal d'actions** (feed) ;
9. **Missions quotidiennes** (ex : "Faire jouer ton monstre 2x", "Atteindre humeur > 80").
10. **Classement** (public)
11. Classement des monstres par **humeur cumulée** sur 7 jours (ou score global) ;
12. Fiche monstre publique (avatar, nom, stats, succès/badges).
13. **Notifications & PWA**
14. Opt-in notifications (faim/énergie/mission dispo) ;
15. **Offline** : l'app fonctionne sans réseau (lecture des pages visitées + file d'actions en attente) ;
16. **Sync** : quand la connexion revient, on envoie les actions en file (Background Sync si supporté).

Hors MVP (bonus)

- Système de **badges/succès** (gamification) ;
- **Social** : liker/comparer des monstres, mini-defis hebdo ;
- **Customization** : skins, accessoires achetables avec des coins (gagnés via missions) ;
- **Mode combat** amical (tour par tour très simple).

---

2) Exigences non-fonctionnelles

- **Performance** : LCP < 2.5s (mobile 4G), CLS < 0.1, FID/INP OK ; images **next/image**, lazy-loading.
- **Accessibilité** : **WCAG AA** (navigation clavier, focus visible, labels, roles ARIA), score a11y $\geq$ 90.
- **SEO** : balises meta dynamiques, Open Graph/Twitter Cards, Sitemap/robots, pages publiques SSG/ISR.
- **PWA** : manifest complet, SW (Workbox ou custom), stratégies : cache-first (assets), stale-while-revalidate (images), network-first avec fallback (API).
- **Éco-conception** : EcoIndex C ou mieux ; bundle < 200kb initial si possible (hors images), police système ou 1 police variable ; images WebP/AVIF.
- **Sécurité** : XSS basic hardening, cookies httpOnly/sameSite, validation schémas (Zod/Yup) côté client & serveur.
- **Qualité** : ESLint, Prettier, TS strict, commits conventionnels.

---

3) Architecture technique

- **Framework** : Next.js (App Router), **TypeScript** strict, **React 18**.
- **Data fetching** : **React Query** (mutations + cache + retries) ; SSR/SSG/ISR mix selon les pages.
- **État local** : **Zustand** (UI, formulaire, file d'actions offline) — facultatif si React Query suffit.
- **Auth** : NextAuth (Credentials simple) **ou** auth minimaliste (token signé en API routes) ; fallback "mode invité" pour accélérer.
- **Storage** :
- *Option A (solo front)* : **LocalStorage/IndexedDB** + API mock (msw) ; plus simple, mais moins SSR.

- *Option B (reco)* : **SQLite + Prisma** via API Routes Next.js (persistant et léger).
- **Déploiement** : Vercel (ou Netlify), preview par PR.

Recommandation pédagogique : démarrer en *Option A* pour valider l'UX/PWA, puis passer à *Option B* si le temps le permet.

---

4) Modèle de données (interfaces TS / Prisma)

```
// Typescript (src/types.ts)
export type StatName = 'hunger' | 'energy' | 'mood' | 'hygiene';
export type ActionType = 'FEED' | 'PLAY' | 'SLEEP' | 'CLEAN';

export interface User {
  id: string;
  email: string;
  name: string;
  createdAt: string;
}

export interface Monster {
  id: string;
  ownerId: string;
  name: string;
  avatarUrl: string; // ou seed pour génération
  color: string; // ex: "#A3E"
  stats: Record<StatName, number>; // 0..100
  happinessScore: number; // calculé/agrégé
  public: boolean;
  createdAt: string;
  updatedAt: string;
}

export interface ActionLog {
  id: string;
  monsterId: string;
  type: ActionType;
  delta: Partial<Record<StatName, number>>; // ex: { hunger: -20, mood: +10 }
  createdAt: string;
  offline?: boolean; // marqué si fait hors-ligne
}

export interface DailyMission {
  id: string;
  code: string; // ex: PLAY_X2
  title: string;
  description: string;
  target: number; // ex: 2
  progress: number;
  rewardCoins: number;
```

```
  date: string; // yyyy-mm-dd
  completed: boolean;
}

export interface LeaderboardEntry {
  monsterId: string;
  monsterName: string;
  ownerName: string;
  score7d: number;
  rank: number;
}
```

> **Note** : côté **Prisma**, répliquer ces schémas en `schema.prisma` (SQLite) avec relations (User 1–N Monster, Monster 1–N ActionLog). Le `happinessScore` peut être recalculé en CRON (ISR) ou au fil des actions.

---

5) Règles de jeu (MVP)

- **Stats** bornées **0..100**.
- **Dégradation** passive toutes les X minutes (ex. toutes les 15 min : hunger +2, energy −1, mood −1). En offline, on calcule la dégradation **à la reconnexion** via le timestamp.
- **Actions & effets** (exemples par défaut) :
- **Nourrir** : hunger −25 (min 0), mood +5, cooldown 5 min.
- **Jouer** : energy −15, mood +15, hunger +5, cooldown 5 min.
- **Dormir** : energy +25, hunger +5, cooldown 10 min ; si energy > 90 → mood +5.
- **Laver** (optionnel) : hygiene +25, mood +5.
- **Missions quotidiennes** (3/jour) : ex. "Jouer 2 fois" (reward +20 coins), "Atteindre mood 80" (+10 coins). Réinitialisées à minuit **local**.
- **Classement** : somme des `mood` > 50 enregistrés sur 7 jours (ou un score combiné pondéré). ISR côté page classement.

---

6) API (si Option B)

Base URL : `/api`

- `POST /auth/login` – body {email,password} → {token,user}
- `GET /me` – header Authorization → {user, monster?}
- `POST /monsters` – adopte un monstre → {monster}
- `GET /monsters/:id` – détails → {monster, recentActions}
- `POST /monsters/:id/action` – {type} → {monsterUpdated, actionLog}
- `GET /leaderboard` – liste → LeaderboardEntry[]
- `GET /missions/today` – missions du jour → DailyMission[]
- `POST /missions/:id/progress` – {delta} → mission mise à jour

Erreurs standardisées : `{ error: { code: string, message: string, details?: any } }`

Exemple `POST /monsters/123/action`:

```
{
  "type": "PLAY"
}
```

Réponse :

```
{
  "monsterUpdated": { "id": "123", "stats": {"mood": 70, "hunger": 40,
"energy": 60, "hygiene": 80} },
  "actionLog": { "id": "a1", "type": "PLAY", "delta": {"mood": 15, "energy":
-15, "hunger": 5}, "createdAt": "2025-10-01T10:00:00Z" }
}
```

---

7) Pages & routing (App Router)

- `/` (**SSG + ISR 60s**) : Hero, CTA adopter, top 10 classement.
- `/adopter` (**CSR/SSR**) : formulaire/générateur de monstre (nom, couleurs, avatar seed).
- `/monstre/[id]` (**SSR + CSR**) : dashboard temps réel (stats, actions, missions, historique).
- `/classement` (**SSG + ISR 5min**) : top hebdo + filtres.
- `/profil` (**CSR**) : profil joueur, opt-in notifications, confidentialité (monstre public/privé).
- `/legal` (SSG) : CGU, mentions, privacy (cookies, notifications).
- `/offline` : page fallback pour mode offline (liens vers dashboard en cache, actions en file).

Layouts & SEO : metadata dynamique par page (title, description, OG), fil d'Ariane JSON-LD (facultatif).

---

8) PWA & Offline — Stratégies

- **Manifest** : icônes multi-tailles, theme/background, `display: standalone`.
- **Service Worker** :
- **Static assets** → cache-first ;
- **Images monstres** → stale-while-revalidate ;
- **API** (`/api/monsters/:id`, `/api/leaderboard`) → network-first avec fallback cache ;
- **Queue** d'actions (`/action`) en **Background Sync** (si supporté) ; sinon **IndexedDB** + retry progressif.
- **Page offline** : expliquer l'état, proposer d'ouvrir les pages en cache, affichage des actions en attente.

---

9) UX/UI & Design système

- **Design** : fun, accessible, couleurs contrastées, motions légères (réduire si `prefers-reduced-motion`).
- **Composants UI (exemples)** :
- `StatBar` (rang 0..100, couleur dynamique, label ARIA) ;
- `ActionButton` (icône, libellé, cooldown visible) ;

- `MissionCard` (progress bar, CTA) ;
- `MonsterAvatar` (SVG génératif basé sur un seed).
- **Feedbacks** : toasts non intrusifs, skeletons en chargement, erreurs claires et réessayables.
- **Accessibilité** : ordre de tab cohérent, `aria-live` pour mises à jour, labels pour contrôles.

---

10) Accessibilité, SEO, éco-conception — Checklists

a11y : contraste ≥ 4.5:1, focus visible, titres hiérarchisés, boutons vs liens, alternatives textuelles, tests clavier.

SEO : titles ≤ 60 chars, meta description, OG images, sitemap/robots, URLs lisibles, contenus des pages publiques indexables.

Éco : images responsives (`next/image`), pas de scripts inutiles, lazy-loading, compression Gzip/Brotli, limiter fonts.

---

11) Tests & qualité

- **E2E (Cypress/Playwright)** : 1) Adopter un monstre → voir dashboard. 2) Faire **Jouer** puis **Nourrir** → vérifier variations de stats & cooldowns. 3) Débrancher réseau (mode offline) → faire 2 actions → rebrancher → vérifier la **sync**. 4) Consulter `/classement` → présence du monstre.
- **Unitaires (Vitest/Jest)** : utilitaires de **calcul de stats**, **dégradation**, **cooldowns**.
- **Qualité** : ESLint (strict), Prettier, Type-Check sur CI.

---

12) CI/CD & déploiement

- **GitHub Actions** (exemple de jobs) :
- `lint` (ESLint), `typecheck` (tsc), `test` (vitest/cypress en mode component), `build` ;
- sur `main` : déploiement auto sur Vercel ; sur PR : preview.
- **Environnements** :
- Dev local ; Preview (PR) ; Prod.
- **Secrets** : variables Vercel/Netlify (auth, DB si Option B).

---

13) Planning conseillé (49h)

- **Sprint 0 (4h)** : setup (Next + TS, ESLint/Prettier, routes de base, UI kit minimal, README).
- **Sprint 1 (10h)** : Modèle + Adoption + Dashboard (stats, actions locales).
- **Sprint 2 (10h)** : Missions + Historique + Classement (mock ou DB simple).
- **Sprint 3 (8h)** : PWA (manifest, SW) + Offline (cache, page offline, file d'actions).
- **Sprint 4 (8h)** : Auth simple + Profil + Privacy + SEO/a11y passes.
- **Sprint 5 (6h)** : Tests E2E/unitaires + Perf (Lighthouse) + nettoyage.
- **Buffer (3h)** : finitions, bugs, démo.

Adapter la répartition selon *Option A/B* (mock vs DB). Garder le **MVP** prioritaire.

14) Barème d’évaluation (suggestion /20 → /100)

- **Fonctionnel & UX** (8 pts) : parcours complet, feedbacks, cooldowns, missions.
- **Technique Next/TS** (4 pts) : SSR/SSG/ISR pertinents, typage strict, structure claire.
- **PWA & Offline** (3 pts) : manifest, SW, cache + file d’actions.
- **Perf & a11y & SEO** (3 pts) : Lighthouse ≥ 90 en Perf/A11y/Best Practices/SEO.
- **Tests & CI/CD** (2 pts) : 2–3 scénarios E2E + CI qui passe.

---

15) Données de départ (seed)

- **Monstres** : 5 presets (couleur, nom, traits) ;
- **Missions** : 6 templates (PLAY_X2, FEED_X1, MOOD_80, SLEEP_X1, ENERGY_90, CLEAN_X1) ;
- **Utilisateurs** : 2 comptes de démo ;
- **Classement** : 10 entrées fictives pour le rendu initial (SSG/ISR).

---

16) Sécurité & confidentialité (minima)

- Bannir `innerHTML` ; validation schémas (Zod) ;
- Cookies `SameSite=Lax`, `Secure` en prod ;
- Opt-in clair pour notifications ;
- Paramètre **Monstre public/privé**.

---

17) Annexes

Structure des dossiers (proposée)

```
src/
  app/
    (public)/
      page.tsx             // Accueil (SSG+ISR)
      classement/page.tsx  // Classement (SSG+ISR)
      legal/page.tsx
    adopter/page.tsx       // Adoption
    monstre/[id]/page.tsx  // Dashboard Monstre (SSR+CSR)
    profil/page.tsx
    offline/page.tsx
  components/
    MonsterAvatar.tsx
    StatBar.tsx
    ActionButton.tsx
    MissionCard.tsx
    Toast.tsx
  lib/
    stats.ts               // règles de dégradation/cooldowns
```

```
    pwa.ts                // helpers SW/manifest
    api.ts                // fetchers React Query
    storage.ts            // IndexedDB/localStorage
  server/
    db.ts                 // Prisma/SQLite (Option B)
    routes/               // API Routes (Option B)
  types.ts
```

Scripts NPM (exemple)

- `dev`, `build`, `start`, `lint`, `typecheck`, `test`, `e2e`, `analyze` (bundle analyzer).

Exemple de règles de dégradation (pseudo-code)

```ts
export function degrade(stats: Record<StatName, number>, minutes = 15) {
  const hunger = clamp(stats.hunger + 2, 0, 100);
  const energy = clamp(stats.energy - 1, 0, 100);
  const mood   = clamp(stats.mood   - 1, 0, 100);
  const hygiene = clamp(stats.hygiene - 1, 0, 100);
  return { hunger, energy, mood, hygiene };
}
```

---

18) Plan de démonstration (5–7 min)

1. Adopter → Dashboard → montrer stats & actions / cooldowns.
2. Passer offline, faire 2 actions → revenir online → sync visible.
3. Missions du jour complétées → reward.
4. Classement (ISR) → monstre présent.
5. Lighthouse/A11y : montrer les scores.

---

19) Bonus possibles si temps

- **Badges** (palier de mood, série de jours) ;
- **Skins**/cosmétiques ;
- **Mini-jeu** pour l'action "Jouer" (petit jeu React sur 30s) ;
- **Thème sombre** ;
- **Exports** (partage image OG de son monstre).

> **Conseil** : verrouille d'abord le **MVP** (Adoption → Dashboard → Actions → Missions → Classement → PWA/Offline → Tests/CI). Les bonus ne viennent qu'après les scores Lighthouse/a11y/SEO OK.