PORTFOLIO HOLBERTON - STAGE 3

Ce document constitue la documentation technique (Stage 3) du Portfolio Project.
Il a pour objectif de transformer la vision du projet (Stage 1) et le cadrage (Stage 2) en un plan technique détaillé servant de blueprint pour le développement (Stage 4).

Cette documentation définit :

- Les user stories et leur priorisation.
- Les mockups et wireframes principaux.
- L'architecture système (front-end, back-end, données).
- Les composants, classes et structures de données.
- Les diagrammes de séquence pour les interactions clés.
- La spécification des APIs internes et externes.
- Les stratégies de gestion de code (SCM) et de qualité (QA).
- Les justifications techniques des choix retenus.

1. User Stories and Mockups

User Stories (**MoSCoW Prioritization**)

Must Have (MVP obligatoire) :

- En tant que recruteur, je veux voir un avatar avec XP et niveau sur la page d'accueil, afin de comprendre rapidement la progression du développeur.
- En tant que recruteur, je veux accéder à une liste de projets (quêtes) cliquables, afin de consulter leurs détails et compétences associées.
- En tant que recruteur, je veux consulter une page compétences avec badges, afin de connaître les technologies maîtrisées.

Should Have :

- En tant que recruteur, je veux voir un résumé clair et concis pour chaque projet, afin d'évaluer rapidement mes compétences.
- En tant qu'utilisateur mobile, je veux que le portfolio soit responsive, afin de le consulter facilement depuis un smartphone.

Could Have :

- En tant que visiteur, je veux voir des animations RPG (gain d'XP, level-up), afin de rendre l'expérience plus immersive.
- En tant que recruteur, je veux pouvoir activer un mode clair/sombre.

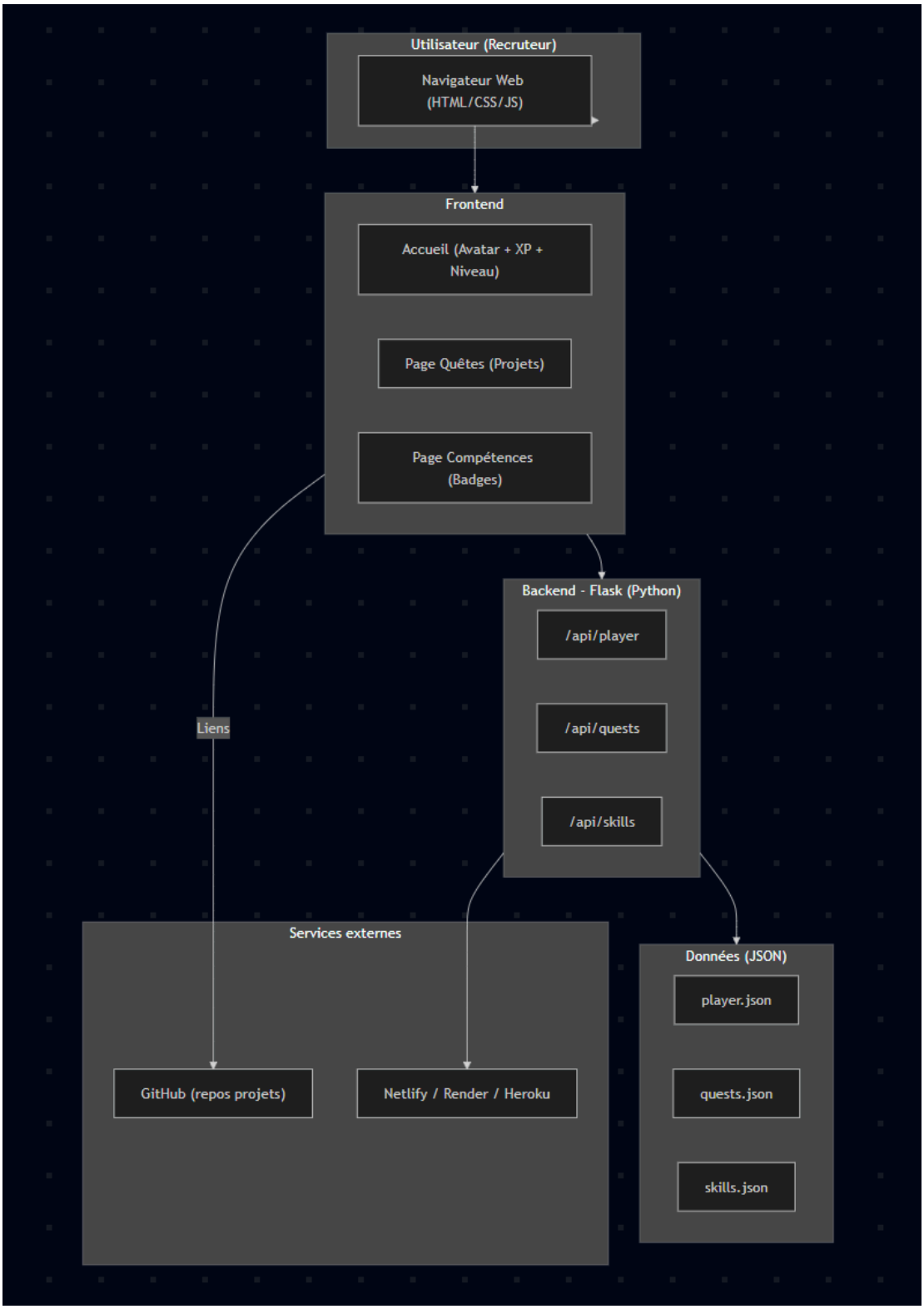
Won't Have (Hors MVP) :

- Mini-map interactive.
- Inventaire d'items (certifications, side-projects, talks).

Mockups (**wireframes simples**)

- Accueil : Avatar du joueur + barre d'XP + niveau affiché.
- Quêtes : Liste verticale de cartes (titre, XP gagné, résumé court).
- Compétences : Grille de badges avec un niveau associé.

2. System Architecture



3. Components, Classes, and Database Design

Backend (Flask) - Endpoints :

- / → page d'accueil (avatar + XP).
- /quests → liste des projets.
- /skills → liste des compétences.

API interne (JSON) :

- /api/player → retourne player.json.
- /api/quests → retourne quests.json.
- /api/skills → retourne skills.json.

Data Structures (JSON) :

player.json :

```
{
    "name": "Thomas Roncin",
    "class": "Backend Wizard",
    "level": 3,
    "xp": 250
}
```

quests.json :

```
[
    {
        “id”: “printf”,
        “title”: “Forge the Ancient Function”,
        “xp”: 80,
        “skillsGained”: [“C intermediate”],
        “summary”: “Recréation de printf avec gestion des formats.”,
        “links”: { “github”: “https://github.com/...” }
    }
]
```

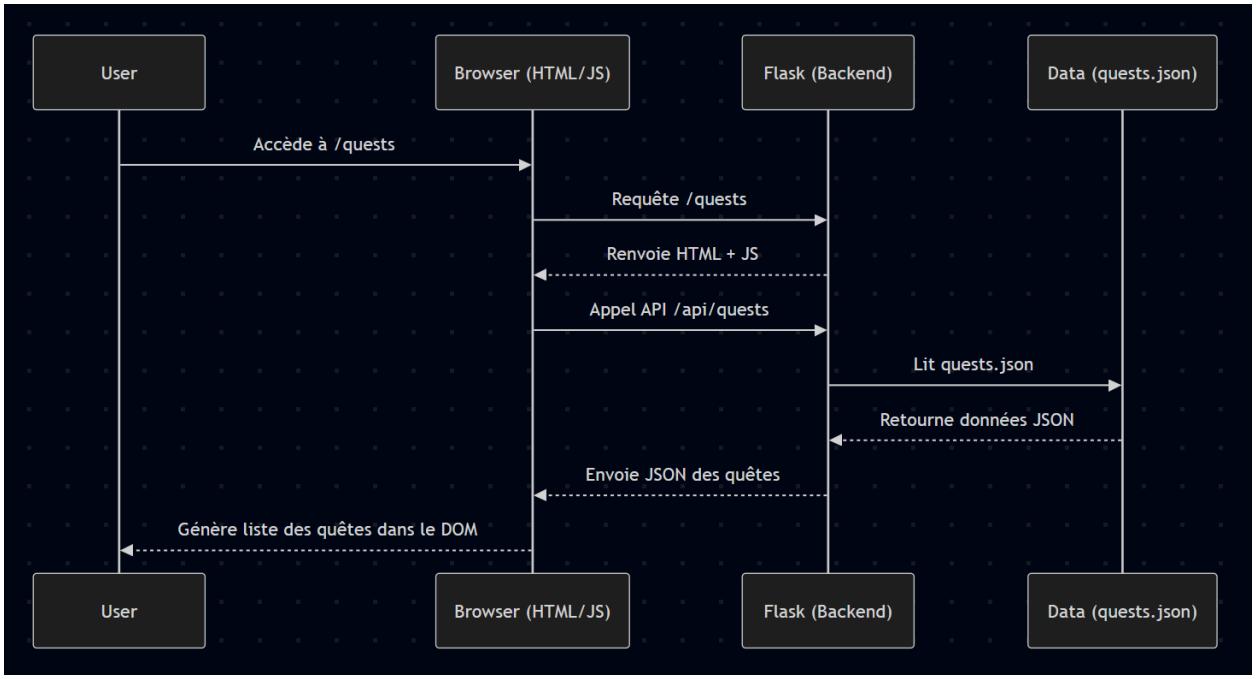
skills.json :

```
[
    { “id”: “c2”, “name”: “C intermediate”, “level”: 2 },
    { “id”: “python2”, “name”: “Python intermediate”, “level”: 5 }
]
```

4. High-Level Sequence Diagrams

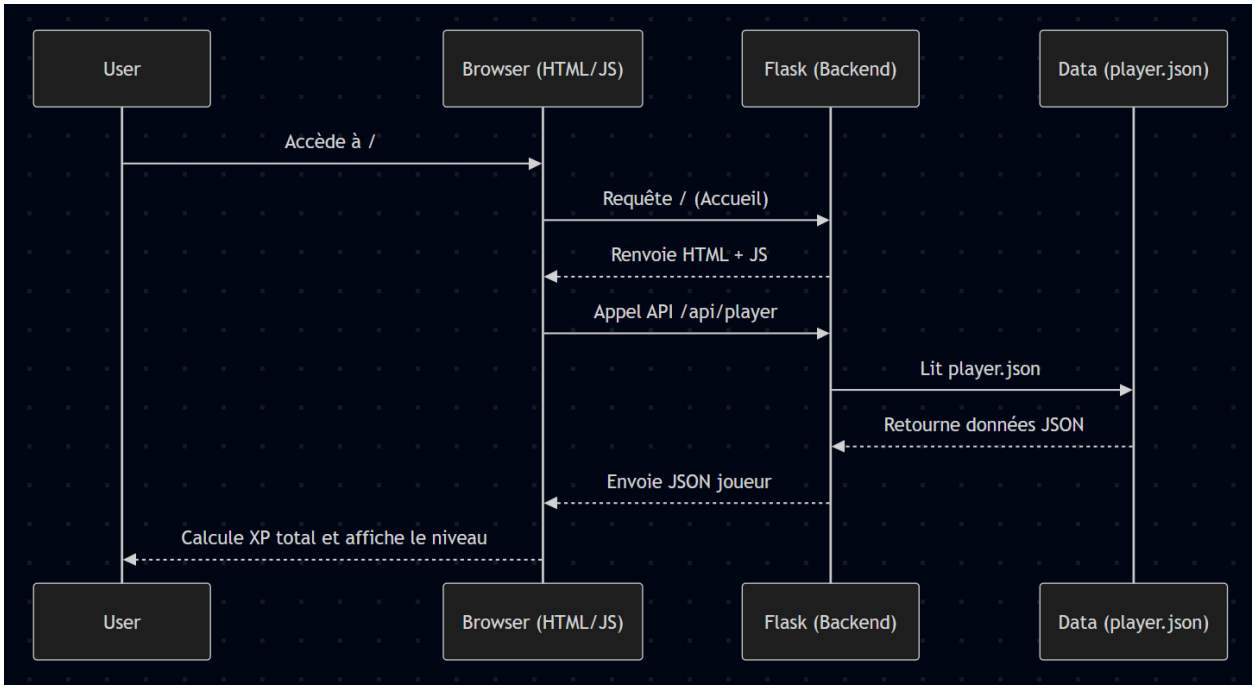
Exemple 1 - Affichage de la page Quêtes :

- User accède à /quests.
- Flask renvoie HTML + JS.
- JS appelle /api/quests.
- Flask retourne quests.json.
- JS génère la liste des quêtes dans le DOM.



Exemple 2 - Calcul XP/Niveau :

- User charge /.
- JS appelle /api/player.
- Flask retourne player.json.
- JS calcul XP total → affiche le niveau correspondant.



5. APIs

External APIs :

- GitHub API (optionnel) → récupération des stats de repo (commits, stars).

Internal APIs (Flask) :

Endpoint	Méthode	Input	Output
/api/player	GET	-	{name, class, level, xp}
/api/quests	GET	-	[{id, title, xp, summary, skillsGained}]
/api/skills	GET	-	[{id, name, level}]

6. SCM and QA Strategies

Source Control (SCM) :

- GitHub comme repo central.
- Branches :
    - main → version stable.
    - dev → intégration.
    - feature/* → développement par fonctionnalité.

- Workflow :
    - Commit → Pull Request → Review → Merge dev → Merge main.

Quality Assurance (QA) :

- Unit tests Flask (pytest pour endpoints).
- Tests manuels front-end (responsive via Chrome DevTools).
- Accessibilité (audit Lighthouse).
- Performance (profiling Flask + audits déploiement).
- Tests d'intégration avec Swagger UI (API).

7. Technical Justifications

HTML, CSS, JS : base simple et rapide, facilement maintenable.

Flask : micro-framework Python adapté à un MVP.

JSON : stockage léger, parfait pour décrire projets et compétences.

GitHub : gestion du code source + hébergement des données.

Netlify / Render / Heroku : déploiement simple et gratuit.

React (optionnel) : pour modularité si temps disponible.