PORTFOLIO HOLBERTON - STAGE 3

This document constitutes the technical documentation (Stage 3) of the Portfolio Project.
Its objective is to transform the project vision (Stage 1) and the framework (Stage 2) into a detailed technical plan serving as a blueprint for development (Stage 4).

This documentation defines:

- The user stories and their prioritization.
- The main mockups and wireframes.
- The system architecture (front-end, back-end, data).
- The components, classes, and data structures.
- The sequence diagrams for key interactions.
- The specification of internal and external APIs.
- The strategies for code management (SCM) and quality assurance (QA).
- The technical justifications for the chosen decisions.

1. User Stories and Mockups

User Stories (**MoSCoW Prioritization**)

Must Have (Mandatory for MVP):

- As a recruiter, I want to see an avatar with XP and level on the homepage, so that I can quickly understand the developer's progression.
- As a recruiter, I want to access a list of clickable projects (quests), so that I can view their details and associated skills.
- As a recruiter, I want to browse a skills page with badges, so that I can know which technologies are mastered.

Should Have :

- As a recruiter, I want to see a clear and concise summary for each project, so that I can quickly assess the developer's skills.
- As a mobile user, I want the portfolio to be responsive, so that I can easily browse it on a smartphone.

Could Have :

- As a visitor, I want to see RPG-style animations (XP gain, level-up), so that the experience feels more immersive.
- As a recruiter, I want to be able to switch between light and dark mode.

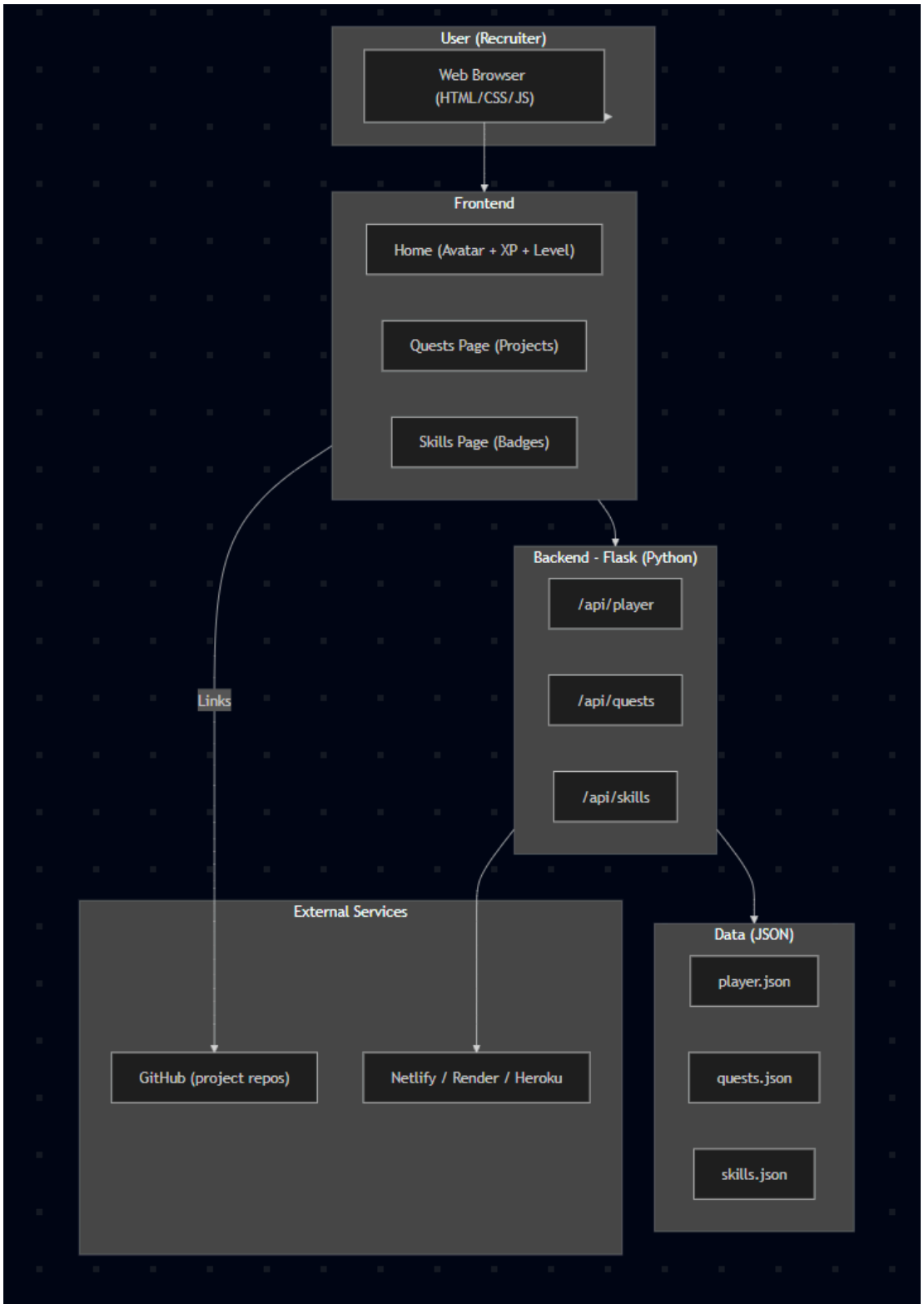
Won't Have (Out of MVP):

- Interactive mini-map.
- Item inventory (certifications, side projects, talks).

Mockups (**Simple Wireframes**)

- Homepage: Player avatar + XP bar + displayed level.
- Quests: Vertical list of cards (title, XP earned, short summary).
- Skills: Grid of badges with an associated level.

2. System Architecture



3. Components, Classes, and Database Design

Backend (Flask) - Endpoints :

- / → Homepage (avatar + XP).
- /quests → List of projects.
- /skills → List of skills.

API interne (JSON) :

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

Data Structures (JSON) :

player.json :

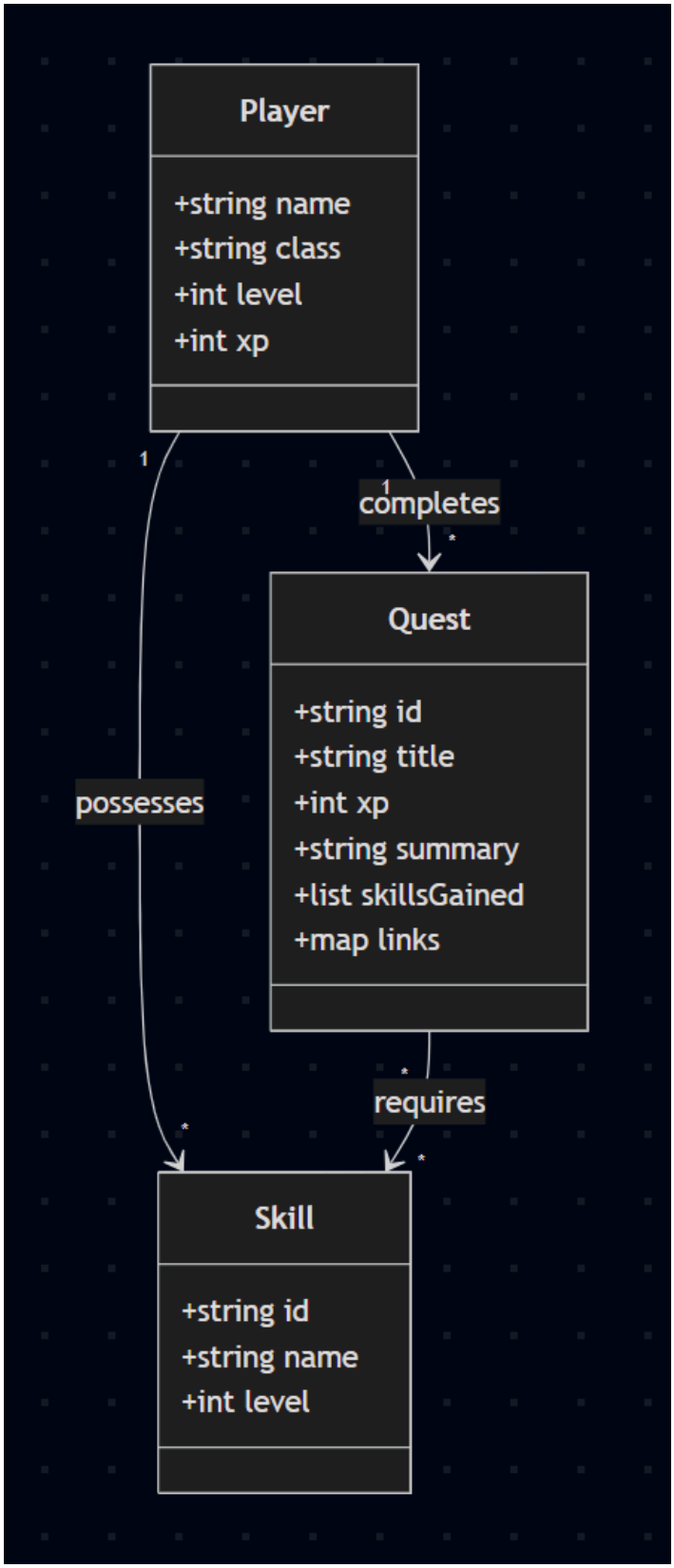

```
{
    "name": "Thomas Roncin",
    "class": "Backend Wizard",
    "level": 3,
    "xp": 250
}
```

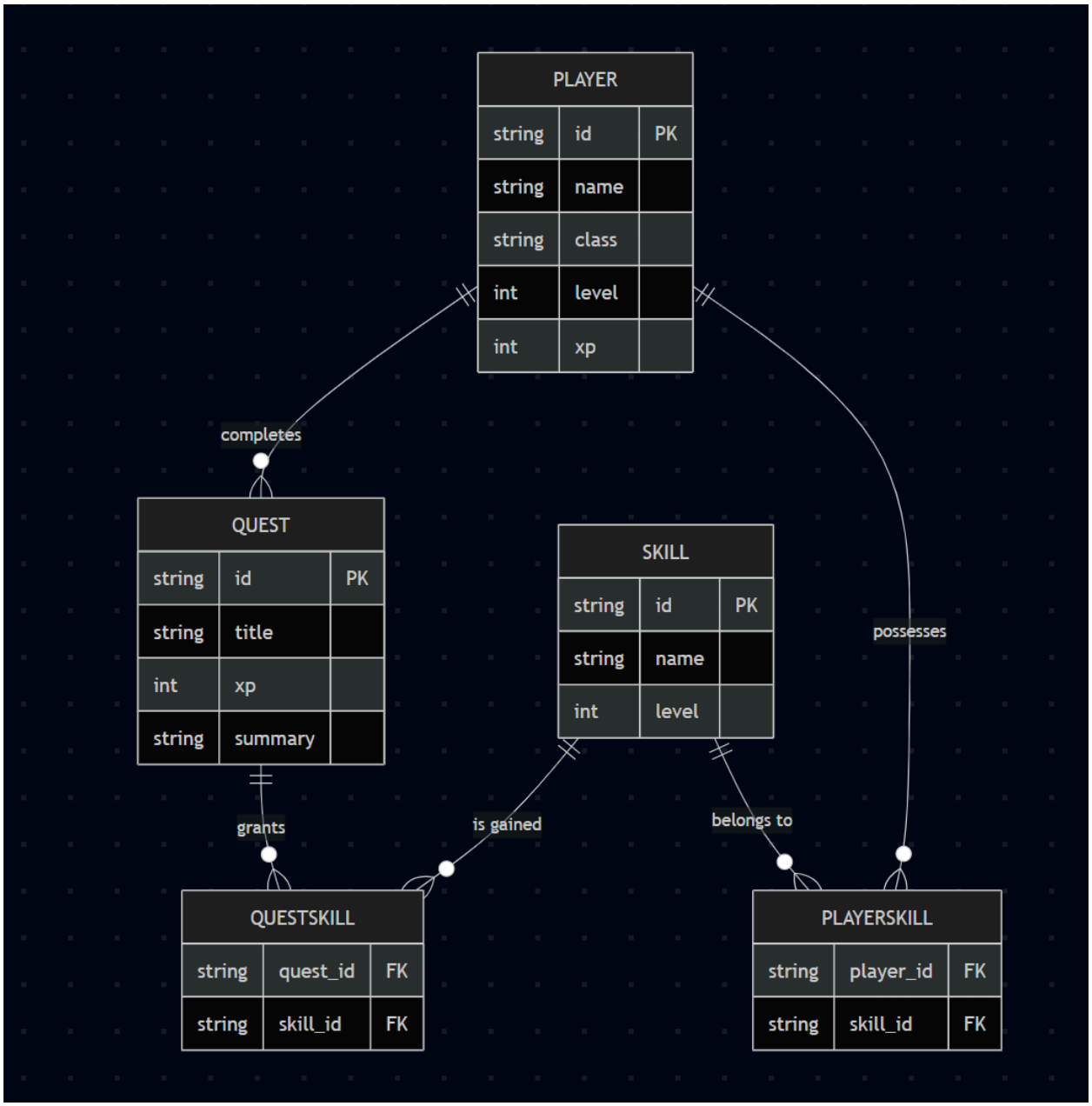
quests.json :

```
[
    {
        "id": "printf",
        "title": "Forge the Ancient Function",
        "xp": 80,
        "skillsGained": ["C intermediate"],
        "summary": "Recreation of printf with format handling.",
        "links": { "github": "https://github.com/..." }
    }
]
```

skills.json :

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

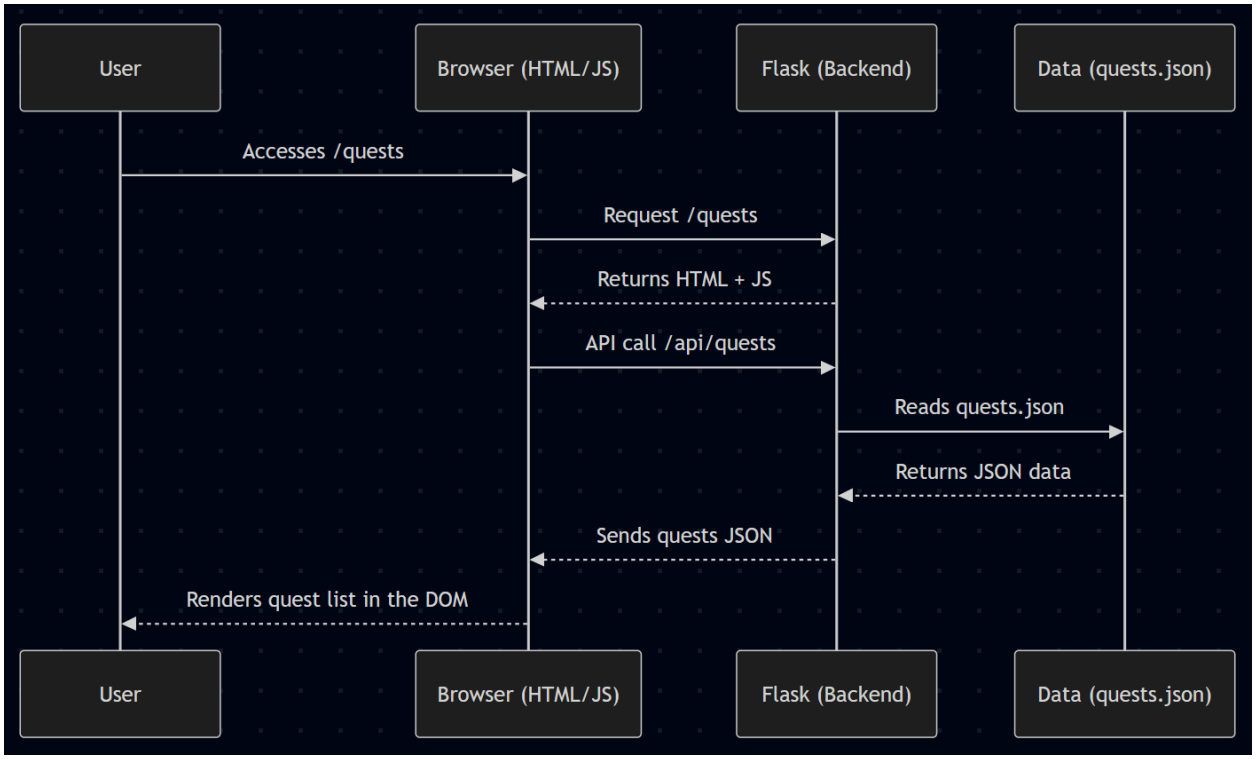

Player
+string name
+string class
+int level
+int xp
1
1
completes
*
Quest
+string id
+string title
+int xp
+string summary
+list skillsGained
+map links
possesses
*
requires
*
*
Skill
+string id
+string name
+int level


PLAYER
string id PK
string name
string class
int level
int xp
completes
QUEST
string id PK
string title
int xp
string summary
SKILL
string id PK
string name
int level
possesses
grants
is gained
belongs to
QUESTSKILL
string quest_id FK
string skill_id FK
PLAYERSKILL
string player_id FK
string skill_id FK

4. High-Level Sequence Diagrams

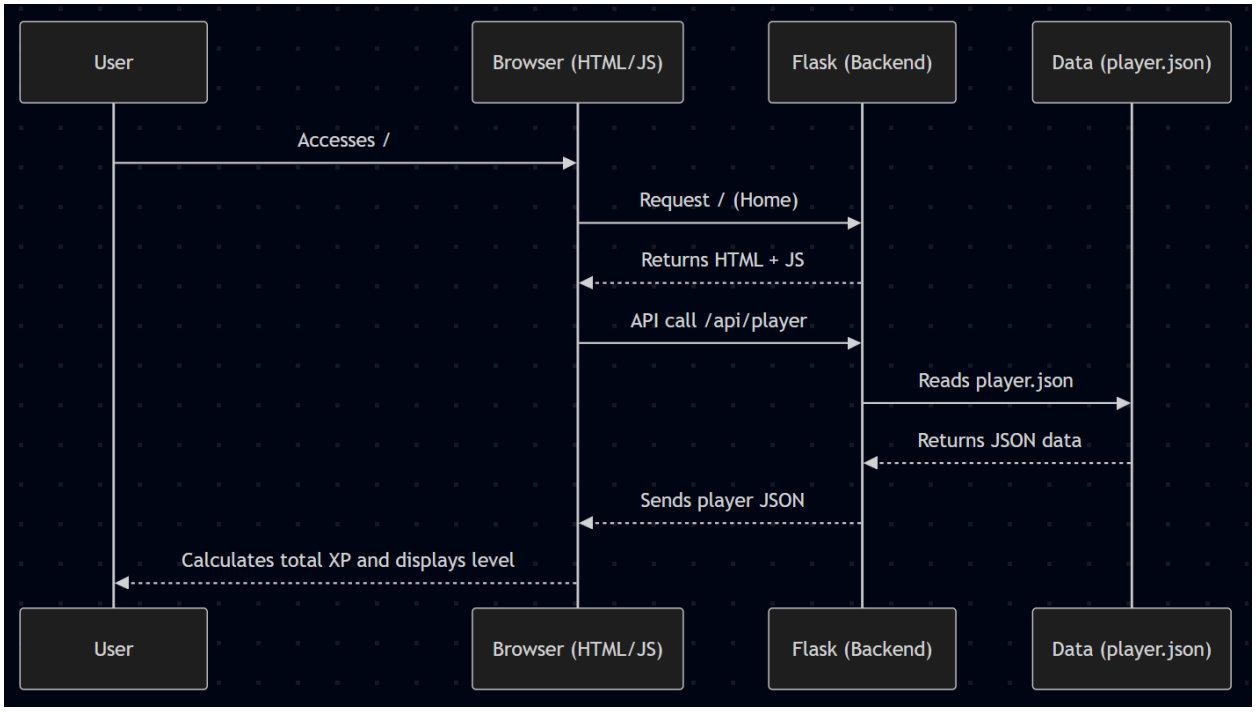
Example 1 – Quests Page Display:

- User accesses /quests.
- Flask returns HTML + JS.
- JS calls /api/quests.
- Flask returns quests.json.
- JS generates the quest list in the DOM.



Example 2 – XP/Level Calculation:

- User loads /.
- JS calls /api/player.
- Flask returns player.json.
- JS calculates total XP → displays the corresponding level.



5. APIs

External APIs :

- GitHub API → fetch repository stats (commits, stars).

Internal APIs (Flask) :

Endpoint	Method	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 as the central repository.
- Branches:
    - main → stable version.
    - dev → integration.
    - feature/* → feature-specific development.

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

Quality Assurance (QA) :

- Unit tests: Flask endpoints (pytest).
- Manual tests: front-end (responsiveness via Chrome DevTools).
- Accessibility: Lighthouse audit.
- Performance: Flask profiling + deployment audits.
- Integration tests: with Swagger UI (API).

7. Technical Justifications

HTML, CSS, JS: simple and fast foundation, easy to maintain.

Flask: lightweight Python micro-framework suitable for an MVP.

JSON: lightweight storage, perfect for describing projects and skills.

GitHub: source code management + data hosting.

Netlify / Render / Heroku: simple and free deployment.

React (optional): for modularity, if time allows.