

RÉCAPITULATIF COMPLET - FITNESS CLASH (Demo Day)

Date : 6 octobre 2025

Deadline : 27 octobre 2025 (21 jours)

Équipe : 4 personnes

Projet : Application de sport au poids du corps avec système de compétition

CONCEPT DU PROJET

FITNESS CLASH : Application pour motiver les gens à faire du sport à domicile via :

- Génération de séances d'entraînement au poids du corps (3 par semaine)
 - Séances aléatoires (échauffement + 3 exercices + étirements)
 - Mini-compétition hebdomadaire entre 4 joueurs (style UNO avec bonus/malus)
 - Système de classement et suivi des performances
-

DÉCISIONS VALIDÉES

Documentation Technique (Stage 3)

Task 0 - User Stories TERMINÉ

- User Stories rédigées avec priorisation MoSCoW
- Mockups Figma → **SKIPPÉ** (trop chronophage, focus sur le code)

Task 1 - Architecture Système TERMINÉ

- Schéma d'architecture 3-tiers créé
- Flux de données annotés (HTTP, JSON, SQL)
- Technologies définies

Task 3 - Diagrammes de Séquence 95% TERMINÉ

- 3 diagrammes créés avec PlantUML
- Diagramme 1 : Lancer une séance
- Diagramme 2 : Valider une séance
- Diagramme 3 : Mini-compétition (4 joueurs)
- **Reste à faire** : Exporter les 3 PNG

Task 2 - Base de Données CET APRÈS-MIDI

- 7 tables identifiées
- Relations définies (1-N, N-N)
- Schéma ER à créer

Task 4 - API Specs 🕒 À FAIRE AUJOURD'HUI

- Définir tous les endpoints REST

Task 5 - SCM & QA 🕒 À FAIRE AUJOURD'HUI

- Stratégie Git et tests

💻 STACK TECHNIQUE - DÉCISION FINALE

⚠️ CHOIX CRITIQUE : Python vs Node.js (Back-End)

Décision prise : **PYTHON + FastAPI**

MAIS : Nomen peut vouloir du **Full JavaScript** (Node.js).

Voici le comparatif complet pour prendre la décision finale.

🔥 COMPARATIF TECHNIQUE COMPLET

🐸 OPTION 1 : Python + FastAPI (Décision actuelle)

Stack complète :

Front-End : React + TypeScript
Back-End : Python + FastAPI
Database : PostgreSQL + SQLAlchemy (ORM)

Avantages Python + FastAPI :

| Critère | Avantage |
|--------------------------|--|
| Facilité | ✓✓✓ Python plus simple que JavaScript côté serveur |
| Rapidité d'apprentissage | ✓✓✓ Syntaxe claire, moins de pièges |
| Performance | ✓✓✓ FastAPI ultra-rapide (asynchrone) |
| Documentation auto | ✓✓✓ <code>/docs</code> Swagger UI généré automatiquement |
| Validation données | ✓✓✓ Pydantic = validation automatique |
| ORM | ✓✓ SQLAlchemy mature et puissant |
| Communauté | ✓✓✓ Énorme, beaucoup de ressources |
| CV | ✓✓✓ FastAPI très demandé en entreprise |

Inconvénients Python + FastAPI :

| Critère | Inconvénient |
|------------------------|---|
| Cohérence stack | ⚠ 2 langages différents (Python + TypeScript) |
| Partage de code | ❌ Impossible de réutiliser du code entre front/back |
| Courbe d'apprentissage | ⚠ Apprendre Python + FastAPI |

Temps d'apprentissage estimé :

- Révision Python : 2 jours
- Apprentissage FastAPI : 3-4 jours
- Total : 5-6 jours

● OPTION 2 : Full JavaScript (Node.js + Express/Fastify)

Stack complète :

Front-End : React + TypeScript
Back-End : Node.js + Express (ou Fastify) + TypeScript
Database : PostgreSQL + Prisma (ORM)

Avantages Full JavaScript :

| Critère | Avantage |
|------------------------|---|
| Cohérence stack | ✓✓✓ Un seul langage partout (JavaScript/TypeScript) |
| Partage de code | ✓✓✓ Réutiliser types, utils entre front/back |
| Connaissance existante | ✓✓ Si vous connaissez déjà JS |
| Prisma ORM | ✓✓✓ ORM moderne, excellente DX |
| Écosystème npm | ✓✓✓ Énorme bibliothèque de packages |
| Performance | ✓✓ Node.js rapide (asynchrone natif) |

Inconvénients Full JavaScript :

| Critère | Inconvénient |
|--------------------|--|
| Complexité JS | ⚠⚠ JavaScript plus complexe (callbacks, promises, async/await) |
| Pièges du langage | ⚠⚠ <code>this</code> , scope, closures peuvent être déroutants |
| Validation données | ⚠ Moins automatique qu'avec Pydantic (besoin de Zod/Joi) |
| Documentation auto | ⚠ Pas natif, nécessite Swagger manuellement |
| Debugging | ⚠ Plus difficile qu'en Python |

Temps d'apprentissage estimé :

- Si déjà à l'aise avec JS : 2-3 jours (Node.js + Express/Fastify)
- Si débutant en JS backend : 5-7 jours
- **Total : 2-7 jours selon niveau**

OPTION 3 : Compromis (TypeScript partout)

Stack complète :

Front-End : React + TypeScript
Back-End : Node.js + NestJS + TypeScript
Database : PostgreSQL + TypeORM

Pourquoi NestJS ?

- Framework TypeScript natif (inspiré d'Angular)
- Architecture structurée (comme FastAPI)
- Documentation auto intégrée
- Validation de données native (class-validator)
- **Le meilleur des deux mondes**

Avantages NestJS :

| Critère | Avantage |
|--------------------|---|
| TypeScript natif | ✓✓✓ Typage fort partout |
| Structure claire | ✓✓✓ Architecture organisée (modules, controllers, services) |
| Documentation auto | ✓✓✓ Swagger intégré |
| Validation | ✓✓✓ Decorators pour valider les données |
| Cohérence stack | ✓✓✓ TypeScript partout |

Inconvénients NestJS :

| Critère | Inconvénient |
|------------------------|---|
| Courbe d'apprentissage | ⚠⚠⚠ Plus complexe qu'Express ou FastAPI |
| Verbeux | ⚠ Plus de boilerplate code |
| Overkill pour MVP | ⚠⚠ Peut-être trop pour 21 jours |

Temps d'apprentissage estimé :

- **Total : 6-8 jours** (framework complexe)

TABLEAU COMPARATIF FINAL

| Critère | Python + FastAPI | Node.js + Express | Node.js + NestJS |
|------------------------|------------------|-------------------|------------------|
| Facilité apprentissage | ★★★★★ | ★★★★ | ★★ |
| Rapidité dev MVP | ★★★★★ | ★★★★★ | ★★★★ |
| Cohérence stack | ★★ | ★★★★★★ | ★★★★★★ |
| Performance | ★★★★★ | ★★★★★ | ★★★★★ |
| Doc auto | ★★★★★ | ★★ | ★★★★★★ |
| Validation données | ★★★★★ | ★★★★ | ★★★★★★ |
| Effet Demo Day | ★★★★★ | ★★★★ | ★★★★★ |
| CV / Employabilité | ★★★★★ | ★★★★★ | ★★★★★★ |

RECOMMANDATION POUR NOMEN

Si vous voulez du Full JavaScript :

☒ **OPTION RECOMMANDÉE : Node.js + Express + TypeScript + Prisma**

Pourquoi ?

- ☒ Cohérence : TypeScript partout
- ☒ Simple : Express est minimaliste
- ☒ Rapide à apprendre : 2-3 jours si déjà à l'aise avec JS
- ☒ Prisma : ORM moderne, excellente DX
- ☒ Faisable en 21 jours

Stack finale :






Front-End : React + TypeScript
 Back-End : Node.js + Express + TypeScript
 Database : PostgreSQL + Prisma ORM

Temps de dev estimé : 10-12 jours de code après la doc

Si vous voulez le plus efficace pour le Demo Day :

☒ **OPTION RECOMMANDÉE : Python + FastAPI**

Pourquoi ?


1.  Plus simple à apprendre
2.  Documentation Swagger automatique (effet WOW)
3.  Validation de données automatique
4.  Moins de bugs potentiels
5.  Performant et moderne

Stack finale :

Front-End : React + TypeScript
Back-End : Python + FastAPI
Database : PostgreSQL + SQLAlchemy ORM

Temps de dev estimé : 10-12 jours de code après la doc

SYSTÈME DE DIFFICULTÉ (Décision validée)




Option A - Simple (MVP)  VALIDÉE

 MISE À JOUR : Difficulté simplifiée 1-3 (au lieu de 1-5)





Raison : Alléger le projet pour tenir la deadline de 21 jours.

Fonctionnement :

- Chaque utilisateur a un `fitness_level` (1 à 3)
- Les séances sont générées selon ce niveau :

| Niveau | Label | Durée | Répétitions | Difficulté exercices |
|--------|---|----------------|-------------|-----------------------|
| 1 |  Débutant | 15 min (900s) | 10 reps | Faciles (niveau 1) |
| 2 |  Intermédiaire | 20 min (1200s) | 15-20 reps | Modérés (niveau 2) |
| 3 |  Avancé | 25 min (1500s) | 25-30 reps | Difficiles (niveau 3) |

Avantages :

-  Plus simple à coder
-  Faisable en 21 jours
-  Suffisant pour le Demo Day
-  Moins d'exercices à créer (30-36 au lieu de 45)

Modifications BDD :

```
sql
```

```
-- Au lieu de CHECK (BETWEEN 1 AND 5)
```

```
Users.fitness_level    CHECK (fitness_level BETWEEN 1 AND 3)
```

```
Workouts.difficulty    CHECK (difficulty BETWEEN 1 AND 3)
```

```
Exercises.difficulty    CHECK (difficulty BETWEEN 1 AND 3)
```

Autres champs inchangés :

- `Workouts.target_duration` (900, 1200, 1500 secondes)
- Structure des tables identique

BASE DE DONNÉES

7 Tables identifiées :

1. **Users** : Utilisateurs
2. **Exercises** : Bibliothèque d'exercices (45 exercices prévus)
3. **Animations** : Animations stickman (relation 1-1 avec Exercises)
4. **Workouts** : Séances d'entraînement
5. **WorkoutExercises** : Table de liaison (N-N entre Workouts et Exercises)
6. **Competitions** : Compétitions hebdomadaires
7. **CompetitionParticipants** : Table de liaison (N-N entre Competitions et Users)

Relations :

- Users (1) → (N) Workouts
- Workouts (N) ↔ (N) Exercises (via WorkoutExercises)
- Exercises (1) → (1) Animations
- Competitions (N) ↔ (N) Users (via CompetitionParticipants)

CONTENU (Exercices au poids du corps)

30-36 exercices prévus (simplifié) :

- 10-12 exercices **haut du corps** (pectoraux, dos, épaules, bras)
 - 3-4 niveau 1 (Débutant)
 - 3-4 niveau 2 (Intermédiaire)
 - 3-4 niveau 3 (Avancé)
- 10-12 exercices **tronc** (sangle abdominale)
 - 3-4 niveau 1
 - 3-4 niveau 2
 - 3-4 niveau 3
- 10-12 exercices **bas du corps** (jambes, fessiers, mollets)
 - 3-4 niveau 1
 - 3-4 niveau 2
 - 3-4 niveau 3
- 3 exercices **échauffement**
- 3 exercices **étirements**

TOTAL : 30-36 exercices (au lieu de 45 initialement prévus)

Note : Pour le MVP, même 20-25 exercices suffisent pour la démo. Le reste peut être ajouté après le Demo Day.







Animations stickman :

- **Option 1 :** Utiliser des GIFs gratuits (musclewiki.com, darebee.com)
- **Option 2 :** Placeholders pour la démo
- **Option 3 :** Animations CSS simples



PLANNING

Documentation Technique (3-4 jours) - Deadline : 10 octobre

-  Task 0 : User Stories
-  Task 1 : Architecture
-  Task 2 : Base de Données (cet après-midi)
-  Task 3 : Diagrammes de Séquence (export PNG à faire)
-  Task 4 : API Specs (aujourd'hui)
-  Task 5 : SCM & QA (aujourd'hui)

Apprentissage (5-7 jours) - Deadline : 17 octobre

- Python + FastAPI OU Node.js + Express (selon choix final)
- ORM (SQLAlchemy ou Prisma)
- Authentification JWT

Développement (10-12 jours) - Deadline : 27 octobre




- Sprint 1 : Base API + BDD
 - Sprint 2 : Logique métier (génération séances)
 - Sprint 3 : Compétitions
 - Sprint 4 : Tests + déploiement
-

DÉCISION À PRENDRE MAINTENANT (NOMEN)




Question critique :

Tu préfères partir sur quelle stack ?





Option A : Python + FastAPI (Décision actuelle)

-  Plus simple à apprendre
-  Documentation auto impressionnante
-  Recommandé pour la rapidité

Option B : Node.js + Express (Full JavaScript)

-  Cohérence TypeScript partout
-  Partage de code front/back
-  Un peu plus complexe

Option C : Node.js + NestJS (TypeScript hardcore)

-  Framework structuré et puissant
 -   Courbe d'apprentissage élevée
 -  Risqué pour 21 jours
-

MON AVIS PERSONNEL (Claude)

Pour maximiser vos chances de réussir le Demo Day en 21 jours :

Je recommande : Python + FastAPI





Raisons :

1. Plus rapide à apprendre (gain de 2-3 jours)
2. Documentation Swagger auto = effet WOW garanti
3. Moins de bugs potentiels (validation Pydantic)
4. Vous aurez un projet impressionnant à montrer

MAIS si toute l'équipe est déjà à l'aise avec JavaScript : → Node.js + Express + Prisma est un excellent choix aussi

PROCHAINES ÉTAPES

Aujourd'hui (6 octobre) :

1.  Nomen prend la décision : Python ou Node.js ?
2.  Finir Task 3 (export PNG)
3.  Faire Tasks 4 & 5 ensemble
4.  Faire Task 2 (BDD) cet après-midi

Demain (7-8 octobre) :

- Finaliser toute la documentation technique
 - Commencer l'apprentissage de la stack choisie
-

QUESTIONS POUR NOMEN

1. **Stack back-end** : Python + FastAPI OU Node.js + Express ?
 2. **Qui fait quoi dans l'équipe** ? (Back, Front, UI/UX, DevOps)
 3. **Niveau JavaScript de l'équipe** ? (débutant/intermédiaire/avancé)
 4. **Préférence personnelle** ? (apprendre Python ou rester sur JS)
-

Document créé le 6 octobre 2025

Équipe FITNESS CLASH - Holberton Paris Demo Day