Colint School

Projet : PFE - Phase d'idéation
Structurer et valider le projet Diving O Club
Kevin Lavier

Consignes de remplissage — à lire avant de commencer :

- Ce document n'est pas un formulaire à cocher.
- Chaque partie doit être **rédigée en phrases complètes**, **justifiée** et **appuyée par des éléments concrets**.
- Respectez les limites de taille indiquées (description courte/longue), et visez **5 à 10 lignes** pour les sections de fond (valeur, architecture, tests, RNCP).
- Évitez les réponses vagues ("on fera des tests", "on utilisera une base de données") : **spécifiez quoi, pourquoi, comment et quand** (ex. type de tests, critères d'acceptation, couverture visée, outillage ; type de base de données, schéma attendu, cas d'usage, requêtes clés ; choix techniques, alternatives écartées, impacts sur la sécurité/maintenabilité/perf).
- Chaque choix doit **lier explicitement** la compétence RNCP concernée et annoncer un **livrable vérifiable** (ex. MCD/MLD, diagrammes UML, backlog GitHub Project avec user stories, règles GitFlow, plan de tests, pipeline CI/CD, plan de déploiement).
- Indiquez aussi les **risques principaux** et vos **mesures de mitigation**.

N'oubliez pas, l'objectif est que ce document puisse, à la seule lecture, conclure que votre projet **couvre l'ensemble des compétences RNCP** et est **réalisable dans l'année**.

1. Informations générales

- **Nom du projet** : Diving O Club
- **Description courte** : Une application pour les clubs et plongeurs : carnet numérique, inscriptions aux entraînements et sorties, gestion des adhésions, suivi du matériel et partage de documents. Simple, claire et adaptée à la vie associative.

- **Description longue** :
  Diving O Club est une application qui facilite la gestion d'un club de plongée et améliore l'expérience des plongeurs. Chaque adhérent peut créer son carnet de plongée numérique et s'inscrire facilement aux entraînements, sorties et événements, qu'ils soient gratuits ou payants. Les responsables de club disposent d'outils pour gérer les adhésions, vérifier les certificats médicaux, suivre le matériel et partager des ressources (documents, séances, supports pédagogiques).
  L'application centralise tout ce qui est aujourd'hui dispersé entre différents sites, fichiers et outils, pour éviter les pertes d'informations et les bugs rencontrés ailleurs.
  Accessible sur ordinateur et mobile, elle permet aussi aux clubs de communiquer

plus simplement avec leurs adhérents, tout en offrant aux plongeurs un espace personnel clair et moderne.

---

2. Proposition de valeur

- **Public cible** : L'application s'adresse d'abord aux clubs de plongée associatifs, à leurs adhérents (plongeurs et apnéistes), et aux plongeurs indépendants qui souhaitent tenir un carnet numérique. Elle sera également utile aux comités départementaux et régionaux lors de la prochaine phase de développement.
- **Problème identifié** : Aujourd'hui, la gestion d'un club est souvent dispersée entre plusieurs outils : Excel, Google Drive, sites tiers comme VPdive ou la FFESSM. Ces solutions sont peu intuitives, parfois lentes ou instables, et obligent à jongler entre plusieurs plateformes. Les carnets de plongée existants sont souvent complexes à remplir et ne répondent pas aux besoins des plongeurs au quotidien.
- **Utilité / innovation** : Diving O Club centralise tout dans une seule application simple et claire. Elle permet aux clubs de mieux gérer leurs adhérents, événements et documents, et aux plongeurs de disposer d'un carnet numérique accessible partout. L'innovation tient dans l'approche multi-tenant (chaque club a son espace séparé), le workflow intégré (ex : validation du certificat médical), l'accès mobile-first avec mode hors-ligne, et l'intégration de paiements en ligne adaptés aux clubs associatifs (HelloAsso).
- **Cas d'usage concrets** :
    - Un plongeur ajoute une plongée dans son carnet numérique, puis la fait valider par son moniteur.
    - Un adhérent s'inscrit à une sortie en fosse directement depuis l'application et règle en ligne.
    - Un administrateur valide les certificats médicaux et en est notifié automatiquement.
    - Un responsable de club met à disposition des supports pédagogiques (PDF, entraînements) pour ses membres.
    - Un club fait son suivi du matériel et sait quels équipements sont disponibles ou en maintenance.

---

3. Technologies et architecture

- **Langages et frameworks** :
    - **Frontend :** React (simplicité, large écosystème, PWA possible, responsive mobile-first).
    - **Backend :** Node.js + Express (léger, rapide à mettre en place, adapté pour API REST, Prisma pour la communication avec PostgreSQL).

    - **Styling :** Tailwind CSS / Shadcn / DaisyUI (rapidité de prototypage, cohérence du design).
    - **Authentification :** Firebase Auth (facile, sécurisé, couplé avec Firestore et storage).
    - **Paiements :** intégration HelloAsso (adapté aux associations).

- **Architecture prévue** :
  Il s'agit d'une architecture MVC en couches : la Vue est une PWA React (navigation avec React Router), les Contrôleurs sont des routes API (Express) et le Modèle regroupe les données (PostgreSQL en SQL pour le cœur, Firestore en NoSQL pour les séances d'entraînement).
  Exemple :Un adhérent ouvre la PWA et se connecte ; la Vue envoie la requête au Contrôleur, qui vérifie l'authentification et lit le profil dans le Modèle, avec un middleware qui vérifie le club_id. Depuis le calendrier, l'utilisateur clique sur "S'inscrire". Le contrôleur va créer une inscription avec le statut Pending (en attente) et un paiement au statut Initiated en base. L'API renvoie alors une URL de redirection HelloAsso et la Vue redirige le navigateur vers le checkout. Après paiement, HelloAsso appelle le webhook du contrôleur ; la signature est vérifiée puis le Modèle est mis à jour (le paiement sera en statut Confirmed). La Vue revient sur la page de retour, interroge l'API et affiche l'état confirmé.
  Architecture MVC en couches : Vue = PWA React (React Router), Contrôleurs = API Node (Express), Modèle = données (PostgreSQL pour le cœur, Firestore pour les séances d'entraînement). Un middleware vérifie l'authentification et impose le club_id avant les contrôleurs pour segmenter les données par club.

- **Bases de données** :
    - Relationnelle PostgreSQL
        - Gère les entités structurées et liées : utilisateurs, clubs, événements, inscriptions, carnets, matériel.
        - Besoin de relations fortes (ex. un utilisateur appartient à un club, un événement a des utilisateurs, les utilisateurs peuvent s'inscrire à plusieurs événement).
    - Non relationnelle
        - Stocke les séances d'entraînement et ressources pédagogiques (url sous de type string), qui pointent vers des fichiers hébergés sur un CDN.
        - Plus simple pour stocker du contenu léger comme des séances d'entraînement.

---

4. Gestion du projet

- **Méthodologie choisie** : GitFlow, Kanban, Dos (Checklist Definition of Done)
- **Outils associés** : GitHub Project, Wakelet pour la veille
- **Organisation du travail** : Une première phase de POC Release le 17/10/2025, avec 3 phases qui suivront : MVP, Beta release et Final project release.

Github Projects en Kanban :
Backlog → Prochaines tâches → In Progress → Tests → Déploiement → Terminé
Backlog : listing de toutes les tâches
Prochaines tâches : 2 ou 3 cards pour la semaine afin de ne pas être dispersé
In progress : les tâches que je suis en train de faire et que je dois compléter avant de passer à d'autres (2 max)
Tests / preprod: Tests des différentes features pour vérifié que tout est conforme
Déploiement : Vérification du déploiement des features
Terminé : les tâches qui ont été vérifiées et terminées.

Exemple :
Pour la feature Authentification, je crée d'abord une carte en Backlog en détaillant ce qui est attendu de Authentification (écrans Login/Signup/Reset, middleware d'auth, endpoint, token club_id …) . Je passe en Prochaines tâches, puis en in Progress lorsque c'est le moment de traiter la tâche et que les tâches précédentes de In Progress sont terminées.. J'ouvre une PR vers staging et la colonne Tests s'active : CI (lint, unitaires, intégration) + E2E essentiels (end to end, parcours utilisateur) (connexion, reset, refus accès inter-clubs). Si c'est ok, je bascule en Déploiement : merge sur staging, tests rapides pour le fonctionnement général type crash ou wrong display, puis merge sur main pour l'auto-deploy prod. Enfin je mets le README et la documentation technique à jour, je vérifie et valide ou non tous les points de la feature établis dans la card et je la passe en Terminé.

---

5. Stratégie de tests et déploiement

- **Types de tests prévus** : Je mets en place une pyramide de tests : lint et type-check pour détecter tôt les erreurs, des tests unitaires sur la logique métier (front et back), des tests fonctionnels et d'intégration sur l'API (Express + PostgreSQL avec Prisma) qui vérifient aussi les règles d'accès multi-clubs (`club_id`) et les interactions avec Firestore/Storage via émulateurs, puis des tests end-to-end (Playwright/Cypress) sur les parcours critiques : authentification, inscription à un événement payant (checkout + webhook), validation du certificat médical, consultation d'un PDF via CDN. J'ajoute des tests de contrat (OpenAPI) ainsi que des contrôles d'accessibilité et de performance (Lighthouse, latence endpoints).

- **Moments d'intégration des tests** : En local, chaque commit déclenche le lint et les unitaires pour corriger au plus tôt. À l'ouverture d'une PR vers `staging`, ma CI exécute lint, unitaires, intégration (DB dockerisée + émulateurs Firebase) puis le build. Après déploiement automatique en staging, je lance les E2E essentiels et un smoke test (health API, login, lecture d'un PDF, inscription payante→webhook), avec vérification systématique de l'isolement inter-clubs. Le merge vers `main` déclenche le déploiement en production, suivi d'un smoke E2E post-déploiement et de la vérification des migrations. À partir du MVP, je programme un job nightly qui rejoue la suite E2E complète, Lighthouse et un scan de vulnérabilités.

- **Chaîne de déploiement** : GitHub Actions avec un pipeline CI pour les PR, un pipeline staging qui déploie automatiquement après le merge d'une feature sur la branche staging (migrations Prisma appliquées, jeu de données multi-clubs pour valider l'isolation), puis un pipeline prod déclenché au merge sur main qui déploie, applique les migrations de façon sûre, tague la version et publie des release notes. Les branches staging et main sont protégées (CI verte, revue, checklist sécurité multi-tenant). Avant mise en prod, je valide : CI OK, E2E staging, couverture au seuil, migrations vérifiées, smoke checklist conforme. En cas d'incident, je prévois un rollback (migrations réversibles + retour au tag précédent). Les environnements Dev / Staging / Prod me permettent de valider progressivement avant exposition aux utilisateurs.

---

6. Lien avec les compétences RNCP

Pour chaque compétence, expliquez en **3 à 5 phrases** :

- Comment le projet permet de la travailler.
- Quels livrables ou preuves seront produits.

Compétences à justifier :

1. **Installer et configurer son environnement de travail en fonction du projet**

   Configuration de VS Code avec les extensions (prettier, Tailwind CSS IntelliSense, Docker, Firebase Explorer…) ainsi que les différentes bibliothèques (react, react router, firebase, node.js, express …)
   Mise en place d'un environnement reproductible via Docker Compose (création d'un Dockerfile, dockerignore à compléter pour éviter le chargement des données trop nombreuses et inutiles au conteneur, builder l'image)
   Mise en place des émulateurs Firebase (initialisation Auth/Firestore/Storage et configuration).

2. **Développer des interfaces utilisateur**

   Création d'une interface web + pwa React + Tailwind avec écrans authentification, calendrier/événements, inscriptions/paiements, carnet de plongées, séances d'entraînement, profil.
   Réalisation de wireframes Figma / Canva / Photoshop et composants réutilisables (formulaires, listes, états vides/erreurs) en respectant l'accessibilité (navigation clavier, labels, contraste suffisant).

3. **Développer des composants métier**

   Authentification et gestion des droits / rôles : connexion, rôles utilisateurs, accès restreint en fonction du rôle ou de l'affiliation à un club.
   Gestion du matériel : recensement du matériel, date de contrôle et d'expiration avec

alertes
Implémentation du workflow certificat médical (soumission dans le profil de l'adhérent, validation/rejet par un admin, expiration avec alertes)
Inscription des adhérents : workflow du remplissage d'un formulaire jusqu'à la réception et la validation ou non avec notifications.
Carnet de plongée : formulaire à remplir pour la création d'une plongée, liste des plongées réalisées, Dashboard sur la data du plongeur.
Séance d'entraînement : Création d'une séance avec formulaire pré-rempli avec liste déroulante d'éléments, liste des séances d'entraînement, filtres + téléchargement / export pdf.

4. **Contribuer à la gestion d'un projet informatique**

   Suivi du projet par GitHub Projects (Kanban) : Backlog → Prochaines tâches (WIP 2–3) → In Progress (WIP 2) → Tests → Déploiement → Terminé.
   Système de branche GitFlow : 1 branche par feature, puis merge sur staging, puis sur main, le tout avec peer review pour la traçabilité. Release notes à chaque version pour assurer le suivi et le bon fonctionnement de chaque action.
   Découpage en user stories avec DoR/DoD (Definition Of Ready / Done) pour contrôler la qualité de la production.

5. **Analyser les besoins et maquetter une application**

   Formalisation des besoins à partir des expériences utilisateurs (VPdive, Drive FFESSM) avec un formulaire destiné a récolter les données.
   Production de plusieurs maquettes pour les écrans critiques (événements/ inscriptions, panier/paiement, carnet, certificats …) et enchaînement du workflow de navigation.
   Priorité pour cadrer le périmètre MVP afin de ne pas s'éparpiller sur du contenu « secondaire ».

6. **Définir l'architecture logicielle d'une application**

   Adoption d'un découpage en couches : présentation (React), services métier (Express), persistance (PostgreSQL + Firestore/Storage), avec isolation multi-tenant par club_id. Exposition d'une API REST et intégration HelloAsso(checkout + webhooks). Documentation par schéma d'architecture et UML de classes, plus ADRs justifiant les choix (PWA, SQL/NoSQL, sécurité).

7. **Concevoir et mettre en place une base de données relationnelle**

   Schema UML pour visualiser la base PostgreSQL avec les tables, champs et relations (clubs, users, events, registrations, payments, medical_certificates, divelog, equipment). Gestion des migrations versionnées (Prisma) et ajout les champs dans les différentes tables ainsi que leurs relations (1 to 1, 1 to many, many to many) et des jointures.

8. **Développer des composants d'accès aux données SQL et NoSQL**

   Côté SQL, mise en place de services (Prisma) pour CRUD, requêtes jointes

(inscriptions par événement, quotas) et agrégats simples. Côté NoSQL, stockage des séances d'entraînement versionnées (url en string) et des ressources référencées sur le CDN, avec recherche par tags (endurance, perfectionnement, apnée, nage avec palme .)… Encadrement systématique par le club_id et documentation des endpoints.

9. **Préparer et exécuter les plans de tests d'une application**

   Pour la stratégie de tests, l'approche retient des cas très simples et vérifiables. Côté unitaires, vérifier que le calcul de places restantes fonctionne (ex. capacité 10, confirmés 7 → 3) et que le mapping de statut paiement renvoie bien "PAID" ou "FAILED" selon l'entrée. Pour les fonctionnels / intégration (API + DB), contrôler qu'avec un token du Club A la liste d'événements ne contient que des événements du Club A, puis qu'un appel "checkout" crée une inscription au statut PENDING et un paiement INITIATED avec une URL de redirection. En end-to-end, rejouer le parcours "inscription payante" (connexion, clic "S'inscrire", état "en attente", retour paiement, affichage CONFIRMED) et le workflow "certificat médical" (dépôt par l'adhérent, validation par l'admin, affichage APPROVED côté adhérent). En complément, exécuter des tests de contrat d'API pour garantir schémas et codes de réponse, et un contrôle accessibilité / smoke après déploiement (Lighthouse correct sur la page Événements, login OK, ouverture d'un PDF).

10. **Préparer et documenter le déploiement d'une application**

    Mise en place d'une chaîne CI/CD GitHub Actions : build + tests sur PR, déploiement auto sur staging, puis sur prod à merge vers `main`. Application des migrations en mode sécurisé, release notes et endpoints de health pour contrôle opérationnel. Rédaction d'un guide de déploiement et d'un plan de rollback (migrations réversibles, tag précédent).

11. **Contribuer à la mise en production dans une démarche DevOps**

    Automatisation des quality gates (lint, tests, build…) avant mise en ligne et vérifications post-déploiement (smoke E2E). Mise en place de logs structurés, d'un audit log (actions, présence du club_id quand c'est nécessaire) et d'alertes de base (erreurs critiques, échec webhook, crashs). Amélioration continue par itérations courtes, suivi d'un historique de releases.