Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE CHARGÉ
DE L'EMPLOI

DOSSIER PROFESSIONNEL (DP)

Nom de naissance - Kine
Nom d'usage - Kine
Prénom - Thibault
Adresse - 195 Chemin de Bompertuis, 13120 GARDANNE

Titre professionnel visé

Concepteur Développeur d'Applications

MODALITÉ D'ACCÈS :

- x Parcours de formation
- ☐ Validation des Acquis de l'Expérience (VAE)

Présentation du dossier

Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel. **Ce titre est délivré par le Ministère chargé de l'emploi.**

Le DP appartient au candidat. Il le conserve, l'actualise durant son parcours et le présente **obligatoirement à chaque session d'examen**.

Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE.

Il est consulté par le jury au moment de la session d'examen.

Pour prendre sa décision, le jury dispose :

1. des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l'entretien professionnel ou de l'entretien technique ou du questionnement à partir de productions.

2. du **Dossier Professionnel** (DP) dans lequel le candidat a consigné les preuves de sa pratique professionnelle.

3. des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d'un parcours de formation

4. de l'entretien final (dans le cadre de la session titre).

[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels du ministère chargé de l'Emploi]

Ce dossier comporte :

- pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ;
- un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d'un titre, d'un diplôme, d'un certificat de qualification professionnelle (CQP) ou des attestations de formation ;
- une déclaration sur l'honneur à compléter et à signer ;
- des documents illustrant la pratique professionnelle du candidat (facultatif)
- des annexes, si nécessaire.

DOSSIER PROFESSIONNEL (DP)

Pour compléter ce dossier, le candidat dispose d'un site web en accès libre sur le site.

http://travail-emploi.gouv.fr/titres-professionnels

Sommaire

Exemples de pratique professionnelle

EXEMPLES DE PRATIQUE PROFESSIONNELLE

DOSSIER PROFESSIONNEL (DP)

Activité-type 1 Développer une application sécurisée

Exemple n°1 ▸ *Développement d'une application web mobile de construction de deck - Dkbldr*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Présentation du projet

Dkbldr (prononcé "deckbuilder") est une **PWA** (Progressive Web App) mobile destinée à la conception de decks pour le jeu de cartes à collectionner *Magic: the Gathering*. Le but premier de cette application est de proposer une alternative mobile-first aux autres outils similaires déjà existants. En effet, la plupart d'entre eux (tels que *Scryfall, Moxfield,* etc) ne possèdent pas de bon support mobile, la plupart du temps, ils ne sont accessibles et facile d'utilisation que sur un support *desktop*.

Dkbldr permettra ainsi aux utilisateurs de concevoir des decks, les afficher sur leurs profils, rechercher d'autres decks et utilisateurs, dans l'optique d'offrir un aspect "réseau social" à l'application. De nombreuses autres fonctionnalités, telles qu'un assistant IA et un outil de *proxying* (le fait d'imprimer des cartes que l'on ne possède pas encore pour pouvoir les tester), sont prévues à l'avenir.

J'ai utilisé Figma pour le maquettage et le wireframe, Trello pour la gestion des tickets et Draw.io pour la réalisation des modèles de données (MLD et MCD).

Le projet a été structuré avec une **architecture en couches** (aussi appelée "n-tiered"), et a été développé sous le framework React + Vite en TypeScript, avec une base de données PostgreSQL. Une API Express avec Node.js a également été conçue, et le déploiement s'effectue grâce à Railway.

Installer et configurer son environnement de travail

Pour la réalisation du projet, j'ai utilisé des outils classiques de développeur, tel qu'un ordinateur connecté à internet, un éditeur de texte (VSCode), un outil de gestion de *versionning* (Git) lié à un repository distant sur GitHub, l'accès à la base de données avec Supabase et des outils/logiciels utiles à la gestion de projet tels que Trello, Draw.io et Figma, pour la conception.

Développer des interfaces utilisateur

Le projet étant une application mobile, la conception de l'interface utilisateur a démarré très tôt. React + Vite étant un environnement de développement principalement axé sur le front-end (ce que l'utilisateur voit), le projet utilise principalement des fichiers TSX qui seront ensuite "compilés" par Vite en HTML pour être compatibles avec n'importe quel navigateur web.

L'interface utilisateur suit les principes standard d'une application mobile. A savoir, une **barre de navigation** (disponible en appuyant sur un bouton en bas au centre de l'écran) proposant des liens important tels que la page de profil, la page d'inscription ou de connexion, un lien rapide vers la page

d'accueil ou alors un lien vers la page "A propos".

La **page d'accueil** est la première chose que l'utilisateur voit en ouvrant l'application. Pour un utilisateur déconnecté, il ne s'agit que d'un simple message de bienvenue, un lien vers la section "recherche" de l'app, et "la carte du jour" (une carte *Magic* sélectionnée aléatoirement toutes les 24 heures). Un utilisateur connecté, cependant, aura accès à des sections affichant les derniers decks les plus populaires, et les decks les plus récents conçus par d'autres utilisateurs dont l'utilisateur actuel est abonné.

La **page de profil** de l'utilisateur montre une image d'en-tête modifiable au gré de l'utilisateur, tout comme sa photo de profil, ces deux éléments peuvent être changés si l'utilisateur envoie un fichier d'image à l'application. Son nom d'utilisateur, et son nombre d'abonnés et d'abonnements sont également affichés.
La page de profil dispose également d'une section affichant la liste de tous les decks que ce dernier a construit, et partagés publiquement. Un bouton "Créer un nouveau deck" est également disponible, dirigeant l'utilisateur vers la page de création de deck.

La **page de création de deck** ("deckbuilder") montre un champ de texte qui représente le nom donné au deck, ainsi que divers boutons pour les outils, notamment pour la sauvegarde du deck. Une barre de recherche avec auto-complétion est disponible, permettant à l'utilisateur d'ajouter des cartes simplement en entrant leur nom, affichant ainsi les images de cartes dans la liste. Les cartes sont également automatiquement catégorisées en fonction de leurs types ("Créature," "Artefact," "Terrain," etc.). L'utilisateur peut également changer la quantité de chaque carte.

La **page de recherche** de l'application montre une simple barre de recherche, ainsi que trois options de filtrage : "Deck", "Utilisateur" et "Commandant" (un "Commandant" ou "Commander" est, dans le jargon de *Magic,* la carte dont le deck est construit autour). Chaque filtre permet d'afficher des résultats différents. L'option "Deck" affiche tous les decks publics dont le nom correspond à la valeur entrée ; l'option "Utilisateur" affiche tous les utilisateurs dont le nom correspond à la valeur entrée ; l'option "Commandant" affiche tous les decks publics utilisant le commandant recherché.

Développer des composants métier

Dans le projet, les utilisateurs ("Users") et les decks sont considérés comme des objets, et possèdent leurs propres tables en base de données, ainsi que leurs propres méthodes de traitement.

Par exemple, il existe une méthode spécifique au traitement des cartes dans un deck, permettant de trier chaque carte dans une catégorie, par son type. Dans *Magic*, chaque carte possède un type témoignant de sa fonctionnalité, ou de sa représentation conceptuelle dans l'univers du jeu. Une carte "Artéfact" désigne la plupart du temps un objet, comme par exemple une épée. Une carte "Enchantement" représente un concept, comme par exemple une malédiction. Certaines cartes, cependant, possèdent plusieurs types, comme la carte "*Bident de Thassa*" qui tombe sous la dénomination "Artéfact-Enchantement". Ce genre d'exception peut poser problème, pour un simple

algorithme de tri. Ainsi, j'ai développé une méthode "groupCardsByType" permettant de **trier automatiquement les cartes par leurs types principaux**, en évitant les doublons (pour ne pas qu'une carte de "Créature-Artéfact" ne se retrouve dans l'onglet "Créatures" et "Artefacts" à la fois).

Un autre exemple d'un composant métier est la **recommandation de decks**. Dans l'application, il s'agit d'une simple fonction qui invoque une **méthode RCP** (une fonction exécutée dans le serveur de la base de données.) Cette méthode permet à un utilisateur de voir 3 decks conçus par d'autres utilisateurs, basés sur les "tags" utilisés dans les decks réalisés par cet utilisateur. Si j'ai construit plusieurs decks avec les tags "Big Mana" et "Group Hug" appliqués, alors les decks recommandés rentreront dans ces catégories-là.
La fonction RPC est écrite en SQL, et est stockée sur le serveur grâce à Supabase. Je n'ai donc qu'à l'appeler dans une méthode en TypeScript dans le front-end pour récupérer les données filtrées.

Contribuer à la gestion d'un projet informatique

Dkbldr est un projet personnel solo. Par moi-même, je m'occupe de la gestion des tâches, ainsi que de la direction du projet et des priorités.
J'utilise **Trello** pour la gestion des tâches et la roadmap, ce qui me permet d'avoir une vision claire sur les étapes à suivre. J'ai également rédigé un cahier des charges, me permettant de garder un œil sur les objectifs de l'application, et de comparer avec l'état actuel du projet.
La maquette ainsi que la charte graphique ont été réalisés sur **Figma**, un outil de design très pratique pour concevoir l'interface utilisateur d'une application par exemple.
Les changements dans le code sont envoyés sur un repository **Github**, et malgré le fait que je travaille seul, je fais toujours en sorte que les messages des commits soient clairs et qu'ils désignent bien les changements effectués.

2. Précisez les moyens utilisés :

Côté conception, j'ai rédigé un cahier des charges pour poser à l'écrit toutes les idées de fonctionnalités, ainsi que les objectifs principaux du projet.

J'ai utilisé **Trello** pour établir un système de tickets et de "to-do list", me permettant de me remémorer et de mettre à jour la tâche en cours, me donnant une idée assez concrète de l'avancement du projet.

Quant à la conception de l'interface utilisateur, j'ai utilisé **Figma**, un outil très populaire pour la création de maquettes. J'ai créé des vues pour chaque page et composant de l'application, ainsi que la création d'une charte graphique qui définit les couleurs et polices d'écriture à utiliser.

Pour le développement de l'application, j'utilise **Visual Studio Code**, un éditeur de texte principalement utilisé pour écrire du code. Cet éditeur dispose de très nombreux outils et extensions fournissant l'environnement nécessaire pour un tel projet.

Le choix de **React.js** pour le développement de l'interface utilisateur s'est imposé naturellement. Il s'agit d'une bibliothèque JavaScript mature et largement adoptée, permettant de construire des interfaces

utilisateur dynamiques, réactives et modulaires. Son écosystème étendu offre de nombreuses bibliothèques complémentaires, ce qui accélère le développement.

L'utilisation de **Vite** en tant qu'outil de build remplace Create React App pour améliorer les performances de développement et de production. Vite offre un temps de compilation quasi instantané, un hot reload rapide et une configuration plus légère, ce qui s'aligne bien avec les besoins d'une application mobile moderne.

Pour la partie API, le projet s'appuie sur **Express**, un framework minimaliste pour **Node.js**, très bien adapté à la création rapide d'API RESTful. Express permet une gestion claire des routes, des middlewares et des traitements métiers. Ce choix permet aussi une bonne séparation des préoccupations entre la logique métier et l'interface utilisateur.

Pour récupérer les données relatives aux cartes, cependant, je ne peux pas me permettre de les stocker en base de données (il existe plus de 30,000 cartes *Magic* uniques). C'est pourquoi j'ai fait appel à l'API externe de *Scryfall*, un site recensant toutes les cartes Magic, via le package npm `scryfall-api`.

Le projet utilise **Supabase**, une plateforme open source offrant une base de données PostgreSQL couplée à une API automatique et une gestion de l'authentification. Ce choix présente plusieurs avantages :

- Un **hébergement** simplifié avec une console d'administration claire
- Une **authentification** utilisateur sécurisée prête à l'emploi
- Des **fonctions SQL** personnalisées (policies, triggers, fonctions RPC)
- L'opportunité de **stocker des fichier** directement en base de données grâce à la fonctionnalité "storage"

J'ai utilisé **Git** pour le versionning du projet (la gestion des différents changements dans le code) et j'ai fait héberger le code sur **Github**.

Finalement, le déploiement de l'application s'est fait sur **Railway**. J'ai utilisé le plan gratuit pour me permettre de déployer une version stable de l'application gratuitement, simplement en liant le repository Github du projet, et en choisissant un nom de domaine. L'application en ligne se met désormais à jour avec chaque changement sur la branche *main*.

3. Avec qui avez-vous travaillé ?

Ce projet est un projet personnel. J'ai travaillé, conçu, développé et déployé ce projet par moi-même.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ (Aucune)

Chantier, atelier, service ▸ Développement Web & Logiciel

Période d'exercice ▸ Du : 02/2025 au : 08/2025

5. Informations complémentaires *(facultatif)*

Activité-type 2 — Concevoir et développer une application sécurisée organisée en couches

Exemple n° 1 – *Développement d'une application web mobile de construction de deck - Dkbldr*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Analyser les besoins et maquetter une application

Avant toute ligne de code, une phase d'analyse fonctionnelle a été menée pour identifier les besoins des utilisateurs. Dkbldr vise à répondre à un besoin spécifique : faciliter la création de decks pour le jeu *Magic: The Gathering*, tout en offrant une expérience sociale et personnalisée.

Les cas d'usage ont été définis : création de decks, import/export, recherche de cartes, organisation par onglets, interactions sociales (likes, abonnements). Ces scénarios ont permis de définir les principales entités du projet (utilisateur, deck, carte, tags) et leurs relations.

Une phase de maquettage a permis de tester l'organisation des écrans, en adoptant une logique mobile-first. Des outils comme **Figma** ont été utilisés pour itérer sur les interfaces et obtenir une expérience utilisateur intuitive.

Définir l'architecture logicielle d'une application

L'architecture choisie est une architecture en couches composée des couches suivantes :

- **Présentation (front-end)** : basée sur React.js avec Vite, elle s'occupe de l'interface utilisateur, de la navigation et de la gestion locale des états.
- **API REST (back-end)** : développée avec Express.js, elle centralise la logique métier et les règles d'accès aux données.
- **Persistance (base de données)** : assurée par Supabase et PostgreSQL, elle gère les enregistrements sécurisés et les relations entre entités.

Cette séparation claire entre les couches favorise la **maintenabilité**, **l'évolutivité** et **la réutilisation** du code. Elle permet également d'ajouter de nouvelles interfaces (ex. : application mobile) sans réécrire la logique serveur.

Des bonnes pratiques de sécurité ont été intégrées dans l'architecture :

- Authentification sécurisée via Supabase Auth
- Règles d'accès aux données côté serveur (Row-Level Security)
- Protection des endpoints avec des vérifications d'identité sur les requêtes critiques

Concevoir et mettre en place une base de données relationnelle

Le modèle de données a été conçu selon les règles de la **modélisation relationnelle**. Un **Modèle Conceptuel de Données (MCD)** a été réalisé, suivi d'un **Modèle Logique de Données (MLD)**, traduisant les entités et leurs relations en tables SQL.

Le système repose sur les principales tables suivantes :

- **`users`** : stocke les informations de compte
- **`decks`** : contient les decks créés par les utilisateurs
- **`deck_like`** : gère les likes sur les decks (relation n-n)
- **`archetypes`** : liste des archétypes (les "tags") utilisés pour les recommandations
- **`deck_archetype`** : table de liaison entre decks et archétypes

Certaines colonnes utilisent le type `jsonb` pour permettre un stockage structuré et flexible des listes de cartes (mainboard, sideboard, etc.)

La base est hébergée sur **Supabase (PostgreSQL)**, permettant une gestion facilitée, tout en conservant la puissance de SQL pour les jointures, les vues et les filtres

Développer des composants d'accès aux données SQL et NoSQL

Les composants d'accès aux données ont été centralisés dans les routes Express du back-end. Chaque route API constitue un point d'entrée sécurisé vers la base de données, encapsulant les opérations de lecture, écriture, mise à jour ou suppression. L'API dispose également d'un **Swagger**, un outil permettant de document une API interne.

Par exemple :

- **`GET /api/decks/user/:userId`** récupère tous les decks d'un utilisateur donné
- **`POST /api/decks`** insère un nouveau deck
- **`PATCH /api/decks/:deckId`** met à jour les cartes d'un deck existant
- **`POST /api/cards/search/:q`** permet à l'utilisateur de faire une recherche de carte

Le projet utilise principalement **SQL via Supabase**, mais il est pensé pour évoluer. Par exemple, une base **NoSQL (type Firebase, MongoDB ou solution locale en JSON)** pourrait être utilisée pour stocker des logs ou des messages utilisateur dans de futures fonctionnalités sociales (messagerie, commentaires...).

L'accès aux données est fait via **des appels API centralisés**, ce qui respecte l'isolation entre les composants de l'application et la logique de persistance, et permet de sécuriser les échanges entre le front-end et la base.

2. Précisez les moyens utilisés :

J'ai utilisé Supabase pour la création et la gestion de la base de données, qui utilise le langage PostgreSQL.
J'ai utilisé Express avec Node.js pour l'API reliant l'application à la base de données.

3. Avec qui avez-vous travaillé ?

Ce projet est un projet personnel. J'ai travaillé, conçu, développé et déployé ce projet par moi-même.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ (Aucune)

Chantier, atelier, service ▸ Développement Web & Logiciel

Période d'exercice ▸ Du : 02/2025 au : 08/2025

5. Informations complémentaires *(facultatif)*

Activité-type 3 Préparer le déploiement d'une application sécurisée

Exemple n° 1 ▸ *Développement d'une application web mobile de construction de deck - Dkbldr*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Préparer et exécuter des plans de tests d'une application

Dans le cadre du projet Dkbldr, plusieurs niveaux de tests ont été envisagés et partiellement mis en œuvre :

- **Tests manuels** : Chaque fonctionnalité critique (création de deck, gestion des likes, import/export, etc.) a été testée via différents scénarios d'usage. Ces scénarios sont décrits dans un plan de test recensant les **entrées attendues**, **résultats escomptés** et **résultats obtenus.**
- **Tests automatiques** : L'architecture de l'API Express a été pensée pour permettre une couverture par des tests unitaires et d'intégration à l'aide de bibliothèques comme `vitest`. L'objectif est de tester chaque route indépendamment et de s'assurer de la robustesse des composants métiers (par exemple : validation de l'intégrité des decks, protection des routes selon l'utilisateur connecté).
- **Débogage et surveillance** : Des **logs structurés** sont présents côté back-end pour suivre les erreurs, et des outils comme les DevTools Chrome ont été utilisés pour surveiller les performances front-end (Lighthouse, console, network…).

Préparer et documenter le déploiement d'une application

Le projet a été conçu pour être déployé facilement dans un environnement cloud ou sur un serveur personnel

- **Conteneurisation avec Docker** : Une configuration Docker est en cours de mise en place. Elle permet d'isoler les différents services (front, API, base de données) dans des conteneurs distincts, facilitant le déploiement multi-plateformes. Cela simplifie aussi le passage de l'environnement de développement à la production.
  Séparation front/API : Le front-end (**React**) et le back-end (**Express**) sont développés dans des répertoires distincts, chacun avec son propre système de dépendances (`npm`).
- **Documentation technique** : Chaque route de l'API est documentée via **Swagger**, accessible depuis un endpoint `/docs`. Cette documentation facilite la collaboration, les tests de requêtes, et la reprise du projet par d'autres développeurs.

- **Fichier `manifest.json`** et configuration PWA ont été intégrés pour permettre l'installation de l'application sur mobile, renforçant l'expérience utilisateur et ouvrant la voie à un **déploiement sur le web mobile**.

Contribuer à la mise en production dans une démarche DevOps

Le projet suit une logique DevOps légère, adaptée à un environnement individuel ou à une petite équipe:

- **Séparation des environnements** : Bien que la production et le développement se partagent les mêmes outils (notamment Supabase), les configurations `.env` permettent d'alterner facilement entre des bases différentes ou des endpoints différents.
- **Scripts de build** : Des commandes standardisées (`npm run build`, `npm run dev`, etc.) permettent de compiler et lancer les différents composants rapidement. Le front génère un build statique optimisé via Vite.
- **Automatisation envisagée** : Le projet est structuré pour permettre l'intégration de **workflows GitHub Actions** afin de :
  - lancer les tests automatiquement à chaque push
  - valider les builds
  - déployer sur un hébergement à la volée
- **Bonnes pratiques de sécurité** : Les variables sensibles (clés API Supabase, tokens) sont stockées dans des fichiers `.env`, et non exposées côté client. Le projet évite les fuites d'informations en API, et **Supabase Auth** assure la gestion sécurisée des utilisateurs.

2. Précisez les moyens utilisés :

3. Avec qui avez-vous travaillé ?

Ce projet est un projet personnel. J'ai travaillé, conçu, développé et déployé ce projet par moi-même.

4. Contexte

Nom de l'entreprise, organisme ou association ▸ *Cliquez ici pour taper du texte.*

Chantier, atelier, service ▸ *Cliquez ici pour taper du texte.*

Période d'exercice ▸ Du : *Cliquez ici* au : *Cliquez ici*

5. Informations complémentaires *(facultatif)*

DOSSIER PROFESSIONNEL (DP)

Titres, diplômes, CQP, attestations de formation

(facultatif)

Intitulé	Autorité ou organisme	Date
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.

Déclaration sur l'honneur

Je soussigné(e) Thibault Kine ,

déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je suis l'auteur(e) des réalisations jointes.

Fait à Marseille le 30/07/2025

pour faire valoir ce que de droit.

Signature : 

Documents illustrant la pratique professionnelle

(facultatif)

Intitulé
Cliquez ici pour taper du texte.

ANNEXES

(Si le RC le prévoit)