

Rapport technique

Outil de construction de formulaires de saisie web paramétrables



Projet réalisé par :

- Ernest ENGOUE
- Céleste GUILLEUX
- Hamza KHOUBILA
- Souhaila RABAI

Projet encadré par : Olivier RICHARD

Introduction

Dans le cadre de l'UV Projet Développement informatique avancée (DA50), Nous devons réaliser un projet de groupe sur tout le semestre qui synthétisait nos compétences acquises pendant le semestre.

Notre projet consiste au développement d'un outil de construction, d'affichage et de gestion de formulaires web paramétrables. Il fournira le moyen de construire et personnaliser facilement un formulaire en fonction des besoins de l'utilisateur.

Cet outil devra être indépendant, mais ne devra pas être incapable s'intégrer dans un système préexistant. Il gèrera l'affichage des options, de modification et d'ajout de données.

Cet outil s'adresse en général aux entreprises, à des laboratoires de recherche, des universités ou à toutes sortes de services nécessitant l'utilisation des formulaires.

Table des matières

Introduction	2
Table des matières	3
Sujet du projet	5
Objectif	5
Problématique	6
Analyse du projet	7
Présentation du projet	7
Antécédents	7
Diagramme de Gantt	8
Division des tâches	11
Céleste GUILLEUX	11
Hamza KHOUBILA	11
Ernest Engoue	12
Souhaila RABAI	12
Conception	13
Analyse du besoin - Diagramme de cas d'utilisation	13
Cahier des charges	14
Fonctionnalités prévues	14
Fonctionnalités hors du cadre de ce projet	15
Livrable	15
Conception des entités - Diagramme de classes	16
Planification du budget	17
Représentation graphique de l'utilisation du budget dans le temps	18
Réalisation	19
Technologies utilisées, frameworks applicatifs	20
Base de données	20
Application Backend	21
Outil de test des API	22
Application Frontend	23

Organisation du travail et suivi des tâches	24
Prototypage	25
Maquette de l'interface utilisateur	25
Exemple d'interface : L'interface user	26
Modélisation d'entité : Structure de l'objet Prefab	28
La classe FormData	30
TypeRules actuellement définies	31
Structure technique générale des composants du projet	33
Architecture applicative	34
Codage de l'application	36
Implémentation complète de la classe Prefab	36
Implémentation de l'API	37
Fichiers de configuration (Prefabs)	37
Données de formulaire entrées par les utilisateurs (FormData)	37
Données utilisateur (User)	38
Frontend avec formulaire basique	38
Livrable unique pour l'application entière	38
Réalisations incomplètes	39
Documentation	40
Conclusion, Avis des membres	42
Hamza KHOUBILA	42
Souhaila RABAI	42
Ernest ENGOUE	42
Céleste Guilleux	43
Ouverture	44
Authentification et permissions utilisateur	44
Interface graphique d'ajout et de modification des configurations de formulaire	44

Sujet du projet

Objectif

L'objectif du projet est de mettre en place un outil de construction de formulaires de saisie web paramétrable avec les fonctionnalités suivantes :

- L'application doit générer des formulaires paramétrables basés sur des fichiers de configuration (prefabs).
- Le stockage des formulaires et des données de formulaires dans une base de données NoSQL.
- L'affichage de la liste des formulaires sur l'interface graphique avec la possibilité de modification et de téléchargement.
- L'utilisateur final pourra soumettre ces données à l'application, puis trier et retrouver les données saisies.

Problématique

La réalisation des formulaires avec des champs personnalisés pour créer des questionnaires personnalisés, des enquêtes de satisfaction ou des retours produits est une démarche très demandée et touche plusieurs secteurs.

Certes qu'il y a beaucoup de solutions sur le marché, mais aucune ne répond suffisamment aux besoins précis d'un utilisateur.

D'où l'intérêt de ce projet qui répond favorablement à cette problématique.

Analyse du projet

Présentation du projet

Le projet consiste à développer un outil de construction de formulaires web dynamique paramétrables, il offre la possibilité aux utilisateurs de construire facilement un formulaire personnalisé en fonction de leurs besoins.

Cette application est destinée en général aux entreprises, à des laboratoires de recherche, des universités ou à toutes sortes de services nécessitant l'utilisation des formulaires.

Antécédents

Lorsque nous avons obtenu l'intitulé de ce projet, il avait en réalité déjà été commencé. L'année précédente, dans le cadre de l'UV DA50, un groupe d'étudiants avait déjà tenté de réaliser ce projet alors nommé FastAndForm, sans toutefois arriver à le porter à son terme.

Cela nous a cependant permis de bénéficier des réalisations déjà effectuées par ce groupe ; en particulier, de ses analyses de choix de technologies utilisées, de ses diagrammes de classe et de cas d'utilisation, et de ses diagrammes de structure technique.

Ces diagrammes, ces analyses et la vision du projet qu'avaient nos prédécesseurs nous ont guidé tout au long de la conception et de la réalisation de ce projet.

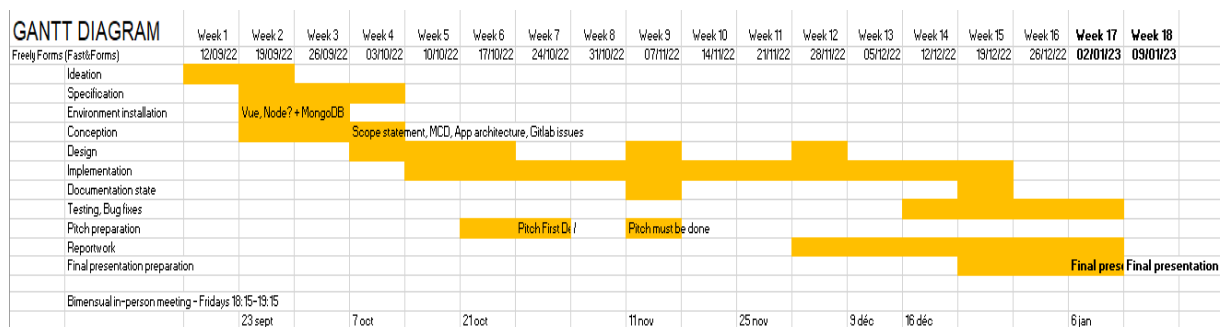
Nous disposions aussi du code source complet des réalisations du groupe précédent ; toutefois, pour des raisons de manque de contenu pour l'application backend du projet, et du manque de documentation pour sa partie frontend, nous n'avons malheureusement pas pu mettre à profit ce code déjà réalisé.

Une priorité dès le départ de ce projet a donc été pour nous d'assurer la durabilité de notre travail, en fournissant des diagrammes à jour et une documentation complète et accessible, autant dans le code du projet que dans la description de sa structure.

Diagramme de Gantt

Pour nous organiser dans le temps, nous avons choisi de réaliser dès le début du projet un diagramme de Gantt, représentant les réalisations que nous souhaitons avoir réalisé à plusieurs dates-clés successives.

Cela nous permettait d’avoir une idée des tâches à effectuer chaque semaine.



Nous avons ainsi divisé la réalisation du projet en onze “sujets” majeurs.

Chacun de ces sujets était jugé essentiel à la réalisation du projet ; et certains, comme le design, seraient idéalement revisités à intervalles réguliers, pour être repensés et mis à jour suivant l’avancée du projet et l’évolution de son contexte.

- L’idéation a lieu lors de la période suivant immédiatement l’attribution du projet ; durant cette période, nous nous sommes appropriés le sujet et l’avons défini dans un format qui nous convenait.

- La spécification a lieu pendant et à la suite de l’idéation ; durant cette période, nous définissons un cahier des charges fixant de manière concrète nos objectifs pour le projet : ce que nous ferons, et ce que nous déterminerons comme étant en-dehors du sujet que nous traiterons pour ce semestre. On définit aussi lors de cette phase le diagramme de cas d’utilisation qui nous guide dans l’évaluation des besoins de l’utilisateur.

- L’installation de l’environnement correspondant à la phase technique pendant laquelle nous doterions nos machines des outils sur lesquels nous travaillerons en commun. Cela inclut l’installation d’IDEs, mais aussi des outils qui feront tourner notre application (MongoDB, Spring, React.js...)

Il s’agit cependant aussi de se familiariser avec les outils utilisés, chacun ayant ses propres spécificités, ses propres contraintes et ses propres demandes en termes de structure et d’annotation applicative que nous ne connaissions pas forcément.

- La conception correspond à l’établissement de la forme que prendra l’application ; en termes d’architecture de l’application (organisation des packages et classes, organisation des structures de données) ; lors de celle-ci, plusieurs documents sont réalisés qui servent à

définir et guider la future implémentation. Ces documents incluent les diagrammes de classe de l'application, la représentation de la base de données et des entités qui s'y trouvent, ou la liste et l'organisation des routes API.

- Le design correspond à la définition des tâches unitaires nécessaires à la réalisation du projet., et à la mise à jour des diagrammes et documents de conception suivant l'évolution du projet (si par exemple des modifications s'avèrent être nécessaires).

La flexibilité est une qualité importante à avoir lors de la réalisation d'un projet, et il est important de ne pas se bloquer dans une implémentation malsaine ou suboptimale parce qu'on a fait des mauvais choix de conception trop tôt dans le cycle de développement de l'application.

Lors des phases de design, on élabore aussi les interfaces utilisables par l'utilisateur final.

- L'implémentation correspond au codage effectif de l'application : la rédaction des classes, des algorithmes et des méthodes, leur inclusion et interaction les uns avec les autres, l'adaptation aux frameworks et systèmes que nous utilisons pour obtenir une application exécutable fonctionnelle.

Cette étape peut sembler débutée tard dans le cycle de vie du projet ; cependant commencer à coder trop tôt avant d'avoir défini le projet, ses contraintes et ses spécifications peut forcer à refaire du travail déjà effectué ou pousser à effectuer de mauvais choix de conception pour conserver ce travail.

- La documentation à l'intérieur et à l'extérieur du code (avec des outils comme Javadoc) permet de partager plus facilement le travail avec les autres membres de l'équipe, et permet aussi au projet d'être durable dans le temps ; si toutes les fonctions du projet sont documentées, cela permet d'avoir rapidement un aperçu sur le travail déjà effectué et des fonctionnalités actuelles du projet, ainsi de comment elles fonctionnent entre elles.

Ainsi, un projet documenté est plus facile à maintenir et à améliorer, autant par ceux qui le codent que pour quiconque souhaite se l'approprier.

- Les tests et corrections de bugs servent à identifier des anomalies dans le fonctionnement de l'application, et interviennent normalement assez tard dans le développement du projet, pour réparer les erreurs connues qu'on a pas pris le temps de corriger et identifier de nouvelles sources d'erreurs.

- À la mi-semestre, un pitch vidéo de quelques minutes est réalisé pour présenter le projet et ses objectifs. Il sert autant d'exercice expérimental de communication, que de variation dans les réalisations du projet.

- La préparation du rapport technique est une partie intégrante de la réalisation du projet, puisqu'il permet de regrouper et d'explicitier les documents de conception, de servir de rapport sur l'élaboration et la mise en place du projet et de justifier de façon écrite les décisions prises en termes de technologies et de conception.

- Enfin, à la fin du semestre se déroule une présentation orale du projet, durant laquelle on présente brièvement le projet et effectue une démonstration de ses fonctionnalités.

Durant le semestre, nous avons tenté de suivre chacune de ces étapes pour assurer la réalisation d'un projet aussi abouti que possible.

Division des tâches

Céleste GUILLEUX

Cheffe de projet

Au lancement du projet, on m'a élu à l'unanimité chef de projet, de part ce qui a été interprété comme une compréhension rapide du projet et de ses enjeux, de ma maigre expérience acquise en stage en conception de projet, ainsi qu'une bonne compétence technique en général.

En tant que cheffe de projet, j'ai surtout essayé d'organiser mon équipe, surtout dans les premiers mois, lors de la phase de conception. J'ai décidé des technologies de suivi de tâche que nous utiliserions, et avais la responsabilité d'assigner les tâches à effectuer au reste de mon équipe, en accord avec ce avec quoi ils étaient le plus à l'aise.

J'ai aussi été responsable de l'architecture du projet.

Dans les derniers mois, j'ai dû me reconcentrer sur les côtés techniques du projet pour assurer la livraison d'un projet fonctionnel, l'autre développeur backend de notre groupe ayant disparu au milieu du semestre.

Hamza KHOUBILA

Développeur backend

Les tâches que j'ai réalisées étaient tout d'abord la création du diagramme de classe pour avoir une idée générale et connaître l'architecture globale de notre application

En tant que développeur backend, j'ai effectué des tâches telles que la création des classes, les services et les controllers.

Enfin j'ai réalisé des tests de requêtage avec Postman pour tester les API en collaboration avec les autres membres du groupe.

Ernest Engoue

Développeur frontend

J'ai eu à participer à la conception en plus de l'analyse globale du projet à travers la création de l'identité visuelle (logo) et de la maquette de l'application, la rédaction des documents de présentation et des rendus.

J'ai également participé à la réalisation de l'application front end.

Souhaila RABAI

Développeuse frontend

Parmi les tâches que j'ai effectuées afin de construire ce projet. D'abord et avant tout, la gestion du budget afin de mieux planifier les dépenses.

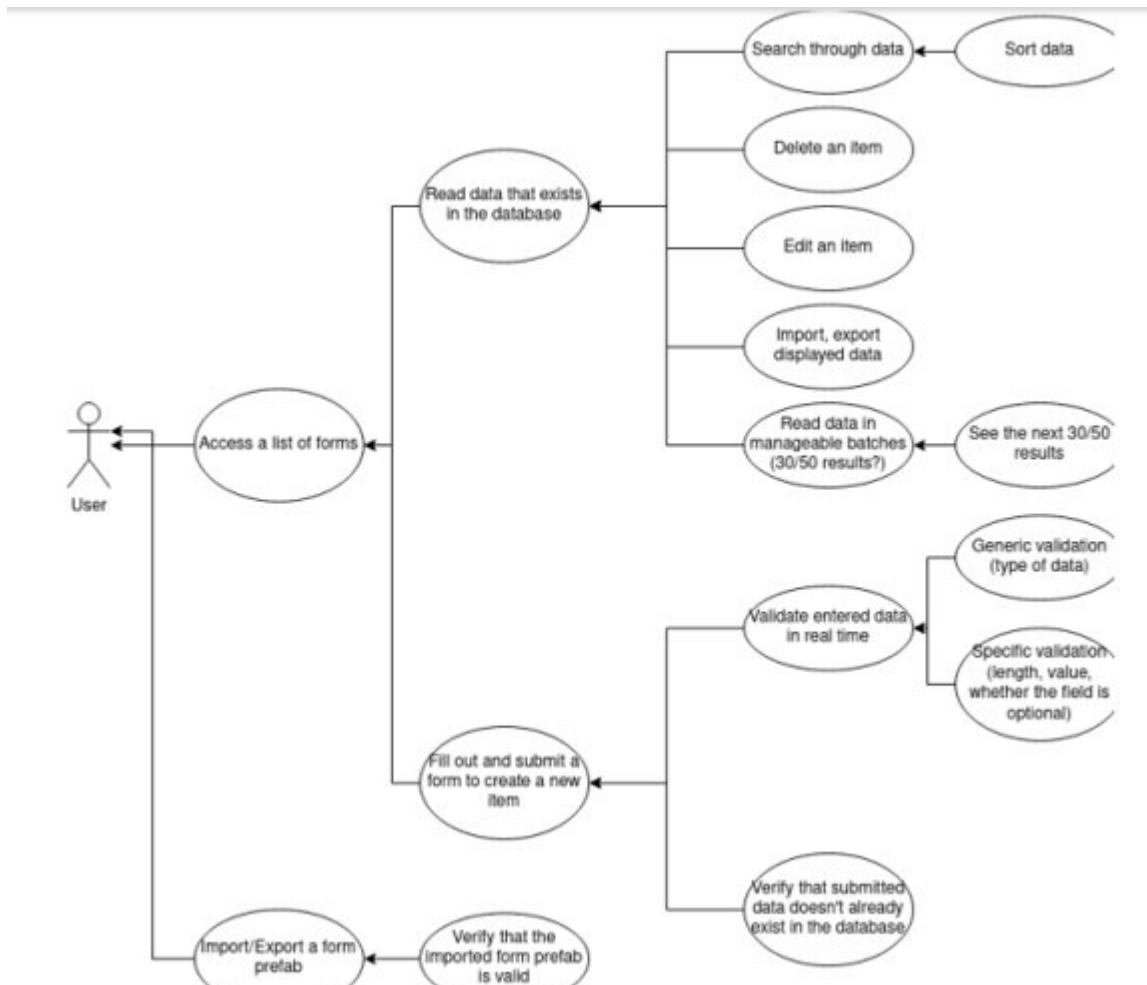
J'ai également contribué à la conception du projet.

Enfin, j'ai aidé à créer l'application Frontend et je l'ai testée pour m'assurer qu'elle fonctionne.

Conception

Analyse du besoin - Diagramme de cas d'utilisation

Pour modéliser le comportement de notre application et les exigences de notre système, nous avons réalisé le diagramme de cas d'utilisation suivant qui décrit les fonctions générales et la portée de notre système.



Cahier des charges

À partir du diagramme précédemment défini, on définit le cahier des charges suivant, pour un projet fini :

Fonctionnalités prévues

- L'application doit générer des formulaires basés sur des fichiers de configuration appelés prefabs.
 - Les formulaires doivent être affichés à l'utilisateur final pour permettre la saisie de données.
 - Les données saisies doivent être vérifiées lors de la saisie du formulaire.
 - La validation peut être générique (vérifier qu'un champ entier contient un entier).
 - La validation peut être spécifique en fonction des paramètres de formulaire définis (vérifier qu'un champ contient un entier compris entre 10 et 20).
 - Un champ peut être spécifié comme « facultatif » (un champ vide est valide).
 - Un formulaire invalide ne peut pas être soumis.
 - Les données saisies doivent être vérifiées après la soumission du formulaire avant d'être ajoutées aux enregistrements de données.
 - La vérification des données après la soumission peut inclure la recherche de doublons.
 - Plusieurs prefabs peuvent exister à la fois.
- L'application doit afficher les données saisies de manière triable, filtrable et consultable, comme un tableau.
 - Toutes les données ne doivent pas être affichées en même temps ; seulement celles qui correspondent à un prefab spécifique.
 - Si le prefab a beaucoup de données liées, seule une partie d'entre elles sera affichées à la fois.
 - Des filtres doivent permettre d'exclure, d'inclure ou de trier des données.
 - La vue d'affichage des données doit permettre à l'utilisateur final de modifier ou de supprimer des données.
 - Les données modifiées doivent toujours être vérifiées, avant et après la soumission.
- L'affichage doit être redimensionné dynamiquement sur des écrans plus petits (c'est-à-dire que l'application doit rester utilisable sur un appareil mobile).
- L'affichage et la saisie des formulaires et des données doivent être à la fois pratiques et agréables à utiliser.

- Les données peuvent être importées ou exportées
 - Toutes les données relatives à un prefab peuvent être exportées
 - Les données correspondant à une recherche et des filtres spécifiques peuvent être exportées

Fonctionnalités hors du cadre de ce projet

Sont définis comme hors des objectifs de ce projet les fonctionnalités suivantes :

- L'authentification utilisateur
 - Les permissions utilisateur de consulter, modifier et supprimer les données liées à certains prefabs, ou l'ajout de prefabs
- Permettre l'usage du protocole HTTPS avec OpenSSL
- Ajouter ou éditer les prefabs via une interface graphique
 - Création et édition de prefabs
 - Modification dynamique des valeurs des données déjà entrées en cas d'édition de prefabs
 - Backup des états précédents de ces données qui pourront être rétablies en cas d'erreur

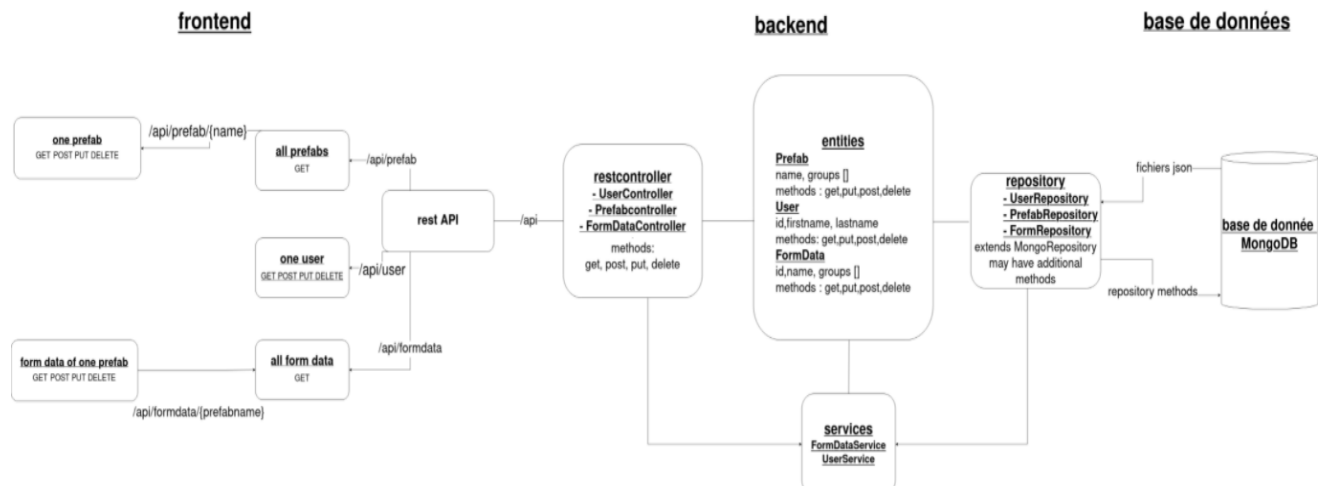
Ces fonctionnalités peuvent cependant être mises en place dans un autre cycle de projet.

Livrable

Le livrable devra être une application web fonctionnelle, soit accessible par Internet, soit fonctionnant localement.

Conception des entités - Diagramme de classes

Ce diagramme nous donne une vue globale sur l'architecture du projet avec les types de requêtes envoyées et leurs retours entre la base de données le backend et le frontend.



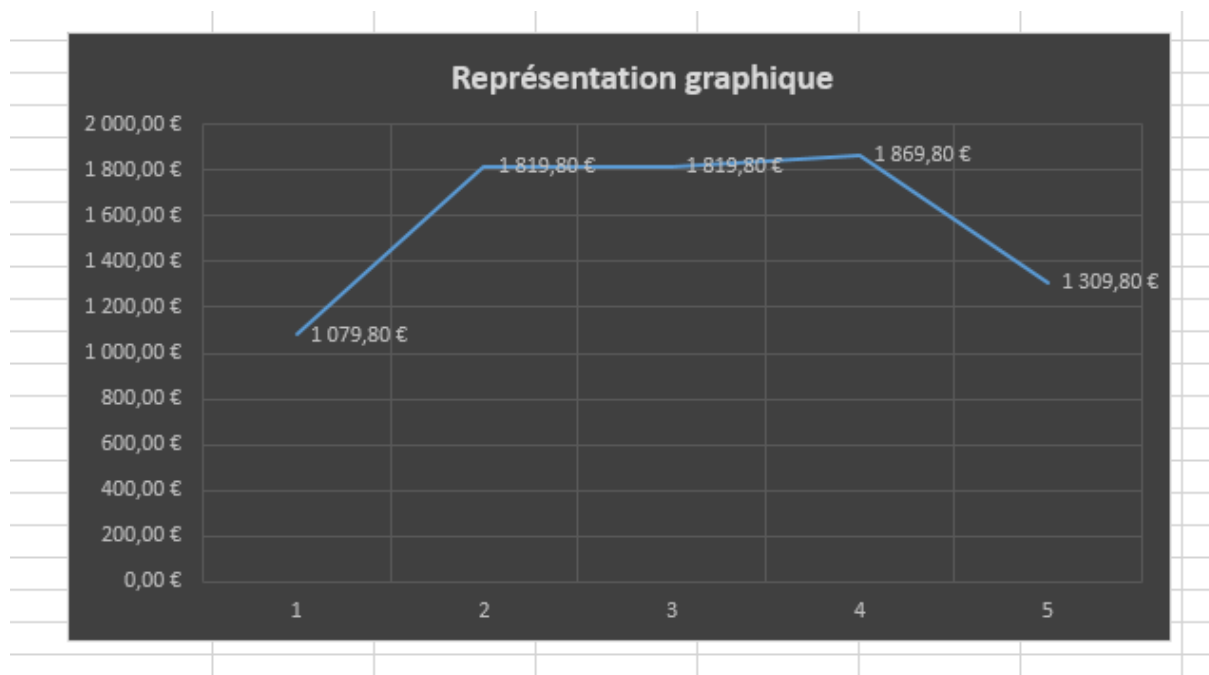
Planification du budget

Dépenses							Total
Période	Septembre	Octobre	Novembre	Décembre	Janvier	Total	
Matériels (personnel)	-00 €	-00 €	-00 €	-00 €	-00 €	-00 €	-00 €
Logiciels	119,80 €	119,80 €	119,80 €	119,80 €	119,80 €	599,00 €	599,00 €
Charges indirectes (chauffage, électricité)	150,00 €	150,00 €	150,00 €	150,00 €	150,00 €	750,00 €	750,00 €
Chef de projet	440,00 €	440,00 €	440,00 €	440,00 €	440,00 €	2 200,00 €	2 200,00 €
Développeur Front-end	-00 €	460,00 €	460,00 €	460,00 €	200,00 €	2 300,00 €	4 600,00 €
Développeur Back-end	150,00 €	500,00 €	500,00 €	500,00 €	200,00 €	2 500,00 €	5 000,00 €
Technologies	-00 €	-00 €	-00 €	-00 €	-00 €	-00 €	-00 €
Logo	20,00 €					20,00 €	20,00 €
Production	50,00 €	50,00 €	50,00 €	50,00 €	50,00 €	250,00 €	250,00 €
Publicité	50,00 €	-00 €	-00 €	50,00 €	50,00 €	150,00 €	150,00 €
L'inattendu	100,00 €	100,00 €	100,00 €	100,00 €	100,00 €	500,00 €	500,00 €
	1 079,80 €	1 819,80 €	1 819,80 €	1 869,80 €	1 309,80 €		14 069,00 €

Le calcul du budget s'étale sur 5 mois :

- Le matériel : Nous avons utilisé nos ordinateurs personnels, le coût du matériel n'entre donc pas dans le budget.
- Les logiciels : Nous avons travaillé avec IntelliJ version entreprise coûtant 119€ par mois ; les charges indirectes coûtent 150€ par mois.
- Les charges indirectes : 150€ par mois comprenant le chauffage, l'électricité et la connexion internet.
- Charges salariales : le salaire mensuel pour un chef de projet est 400€ par mois, 460€ pour les développeurs frontend et 500€ pour les développeurs backend, ces salaires sont calculées en fonction du nombre d'heures travaillées par mois. Nous supposons 2 développeurs frontend, 2 développeurs backend et un chef de projet.
- le logo : le design du logo nous a coûté 20€.
- la publicité : nous avons envisagé un total de 150€ pour la publicité.

Représentation graphique de l'utilisation du budget dans le temps



Réalisation

Pour ce projet, nous divisons en fait notre outil en trois parties distinctes :

- le **système de base de données**, utilisé pour stocker les données variables que l'outil lit et manipule pour effectuer ses opérations
 - Le système de base de données gère le stockage des données de formulaire.
- l'**application backend**, un **service REST** qui communique directement avec la base de données (via un pilote adapté) et qui présente plusieurs **routes API** qui peuvent être requêtées pour obtenir des informations, des données en base, ou modifier l'état des données en base et de l'application backend.
 - L'application backend devrait effectuer la majorité des calculs et ne renvoyer que des données déjà "toutes cuites" pour usage par le frontend.
 - Idéalement, l'utilisateur final ne devrait pas pouvoir se connecter directement aux routes API.
- l'**application frontend**, qui envoie des requêtes et interagit avec les endpoints API de l'application backend pour récupérer et modifier des données. Ces données sont ensuite interprétées par l'application frontend pour produire des pages HTML interactives et lisibles par un navigateur Web tel que Firefox, qui sont ensuite envoyées aux utilisateurs finaux, ce qui offrira à ceux-ci de nouvelles possibilités d'interaction (par exemple : un utilisateur ouvre et complète un formulaire pour ajouter de nouvelles données de formulaire)
 - L'application front-end ne requête jamais la base de données directement, et passe toujours par l'intermédiaire du backend et de ses API.
 - Il est important d'éviter les requêtes excessives entre le frontend et le backend.
 - L'application frontend a le dernier mot en ce qui concerne l'organisation et l'affichage des formulaires.

Chacune de ces couches logicielles utilise des langages, des frameworks et des extensions différentes, et communique les unes avec les autres à l'aide de protocoles bien définis.

Nous avons largement le choix des technologies que nous pouvions utiliser, et nous avons donc réfléchi longuement à ces choix avant de prendre une décision.

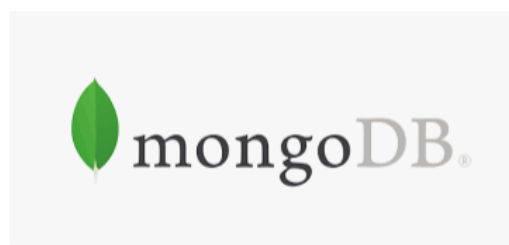
Technologies utilisées, frameworks applicatifs

Base de données

Le stockage des fichiers de configuration et des fichiers de données entrées par les utilisateurs ne peut s'effectuer de façon fluide dans une base de données relationnelle classique. En effet, celles-ci s'attendent à recevoir des données de types précis et dans une structure rigide. Les fichiers de configuration et de données de formulaire peuvent cependant avoir des structures très variables, suivant la configuration du formulaire et la présence ou non de données optionnelles. Les bases de données non relationnelles, plus flexibles dans leur structure et capables de gérer rapidement d'importants volumes de données non structurées, se sont très vite imposées à nos yeux comme une option plus adaptée et plus facilement implémentable.

Nous avons opté pour **MongoDB** comme système de gestion de base de données pour notre projet. C'est un système relativement ancien parmi les systèmes NoSQL, populaire et solidement supporté par les frameworks Java.

Utiliser MongoDB nous permet aussi d'utiliser Atlas, un service en ligne proposé par les développeurs de Mongo permettant, entre autres, de mettre la base de données en ligne plutôt que sur la machine hôte du projet. Cette fonctionnalité est particulièrement utile pour mettre en commun les données de tests entre les différents membres de l'équipe, ce qui accélère le développement en assurant que chacun puisse travailler sur les mêmes données de test.



Application Backend

Pour des raisons de cohérence avec nos autres réalisations du semestre, le framework Java Spring Boot nous était imposé pour l'application backend.

Ce n'est pas pour autant que ce framework est cependant un mauvais choix. Spring Boot est, tout comme MongoDB, un système éprouvé et populaire. Son utilisation est rendue plus facile par la richesse de la documentation disponible, la multitude de tutoriels expliquant ses fonctionnalités et l'abondance de détails disponibles sur toute erreur inattendue que l'on peut rencontrer lors de son utilisation, de part une connaissance incomplète du framework.

Spring Boot est un framework très puissant qui offre un environnement de développement solide et efficace.

Spring Boot est utilisé pour ce projet en conjonction avec Apache Maven, un outil de gestion et d'automatisation de production des projets Java, qui permet de gérer plus aisément les dépendances externes utilisées dans le projet et d'établir des tâches spécifiques à effectuer lors de la production d'un livrable.



Outil de test des API

Le développement des applications frontend et backend ayant été largement indépendant l'un de l'autre, il était nécessaire d'utiliser un outil permettant de tester les routes API de l'application backend afin de tester le comportement du serveur.

Postman est un outil qui permet justement de faire cela ; cette application permet d'effectuer des requêtes usant d'une variété de verbes HTTP, et qui nous a donc permis d'imiter le comportement d'une application frontend pour tester nos APIs.



Application Frontend

L'application frontend permet l'interaction directe avec l'utilisateur final. Elle rassemble les fonctionnalités et permet aux utilisateurs d'interagir avec les données en offrant des interfaces graphiques qui rendent possibles la création, modification, suppression etc des données.

Pour la partie frontend on a choisi de travailler avec la bibliothèque Reactjs qui est une bibliothèque de composants JavaScript permettant de créer des interfaces utilisateur pour les sites Web et les applications, elle crée un nouveau précédent pour le développement de sites Web rapides et dynamiques à l'aide de JavaScript.

React est la principale bibliothèque JavaScript pour le développement d'interfaces utilisateur dynamiques offrant des performances exceptionnelles. Il est conçu pour être simple, polyvalent et évolutif, ce qui en fait une solution idéale pour toute entreprise. Nos services basés sur React permettent aux développeurs de créer des sites Web et des applications hautes performances avec un minimum d'effort.



Organisation du travail et suivi des tâches

Dès le début du projet, nous avons eu besoin d'avoir un suivi des tâches en cours et à faire, ainsi que d'un lieu commun où déposer et mettre en commun notre code.

En ce qui concerne ce dernier point, l'outil Git est très largement utilisé pour stocker non seulement du code, mais plusieurs états de ce même code suivant son avancement.

Git permet de travailler en commun sur le même code et de fusionner les avancées sur un projet sans risquer de perdre ou d'endommager des informations.

Il se trouve que plusieurs services en ligne proposent d'héberger des repositories stockant le code et utilisables avec Git; par exemple, Github et Gitlab. Ces deux services proposent, additionnellement, des outils de suivi de tâches ; résolvant aussi le premier point.

Parmi ces deux choix, l'outil Gitlab a été choisi, de part sa qualité open-source.



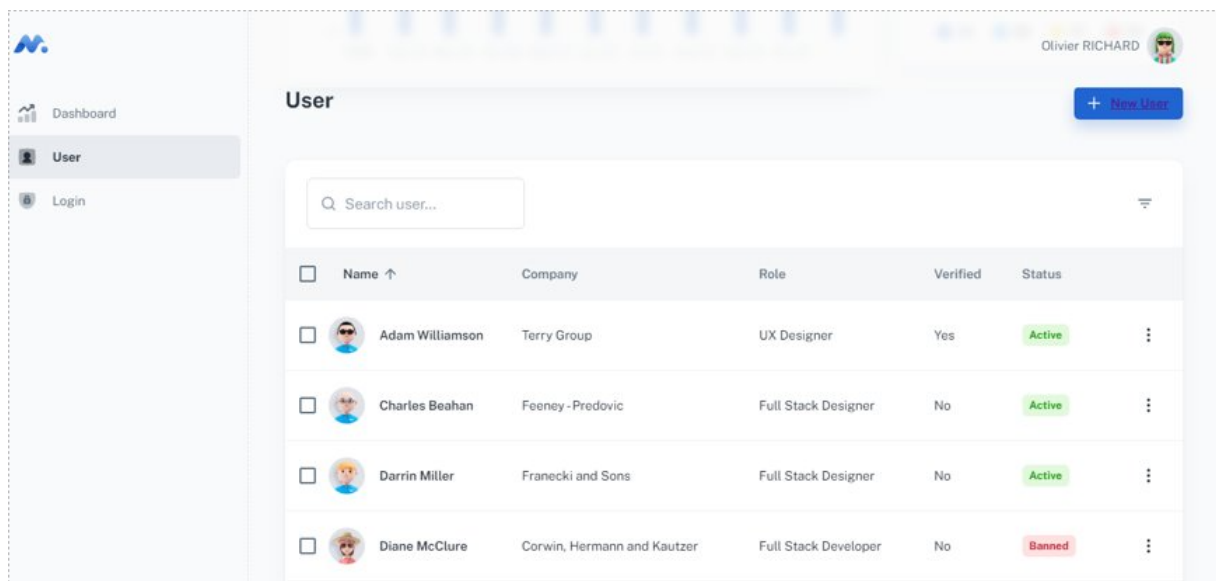
Prototypage

Maquette de l'interface utilisateur



L'interface principale présente un récapitulatif des données de formulaires sous forme de courbes et de graphiques. Cette représentation permet une meilleure visualisation des données et facilite l'analyse par les utilisateurs finaux de l'application.

Exemple d'interface : L'interface user

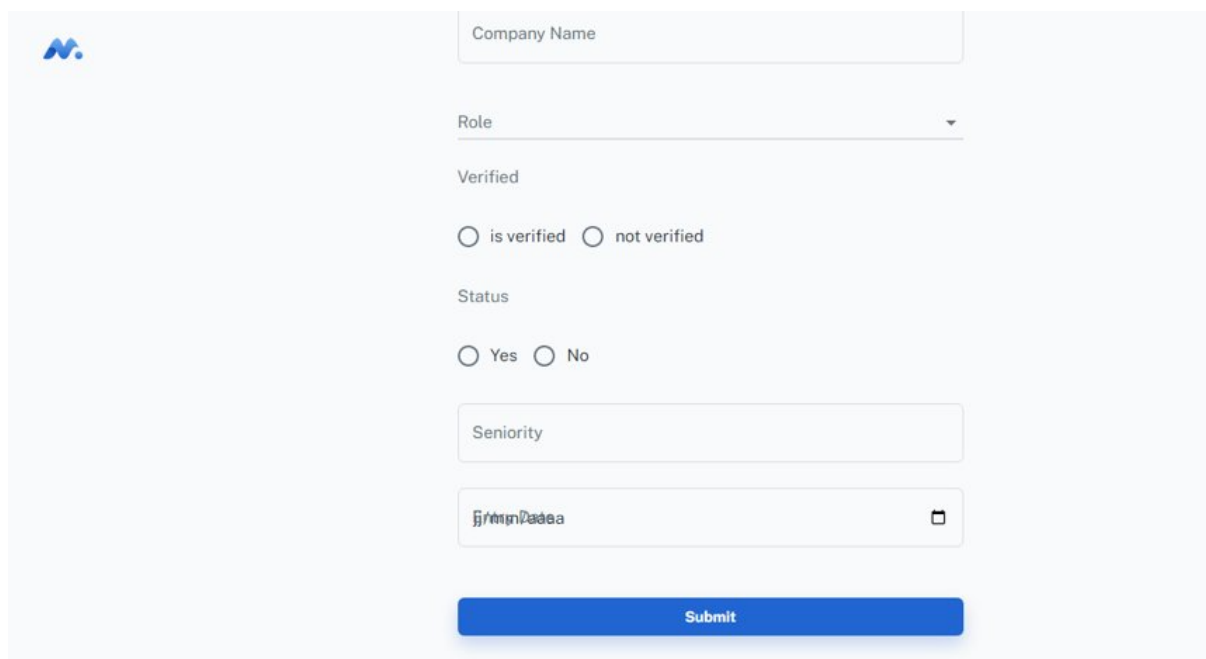


L'interface « User » représente les données 'utilisateurs sous forme d'un tableau

Cette interface peut porter tout autre nom en fonction du prefab/ formulaire dont il représente les données ceci n'est donc un exemple parmi tant d'autres

Le tableau affiche les données rentrées dans le formulaire et permet d'effectuer des tâches précises sur ces dernières. La sélection, la suppression et la modification.

The screenshot shows a form titled 'Add User' with the instruction 'Fill the following form to add a user'. Under the heading 'User informations', there are radio buttons for 'Gender' (Female and Male), and four text input fields for 'First name', 'Last name', 'Age', and 'Company Name'.



The image shows a web form for adding a user. It features a blue logo in the top left corner. The form fields are arranged vertically: a text input for 'Company Name', a dropdown menu for 'Role', a 'Verified' section with two radio buttons ('is verified' and 'not verified'), a 'Status' section with two radio buttons ('Yes' and 'No'), a text input for 'Seniority', and a text input for 'Email' with a calendar icon on the right. A blue 'Submit' button is at the bottom.

Company Name

Role

Verified

☐ is verified ☐ not verified

Status

☐ Yes ☐ No

Seniority

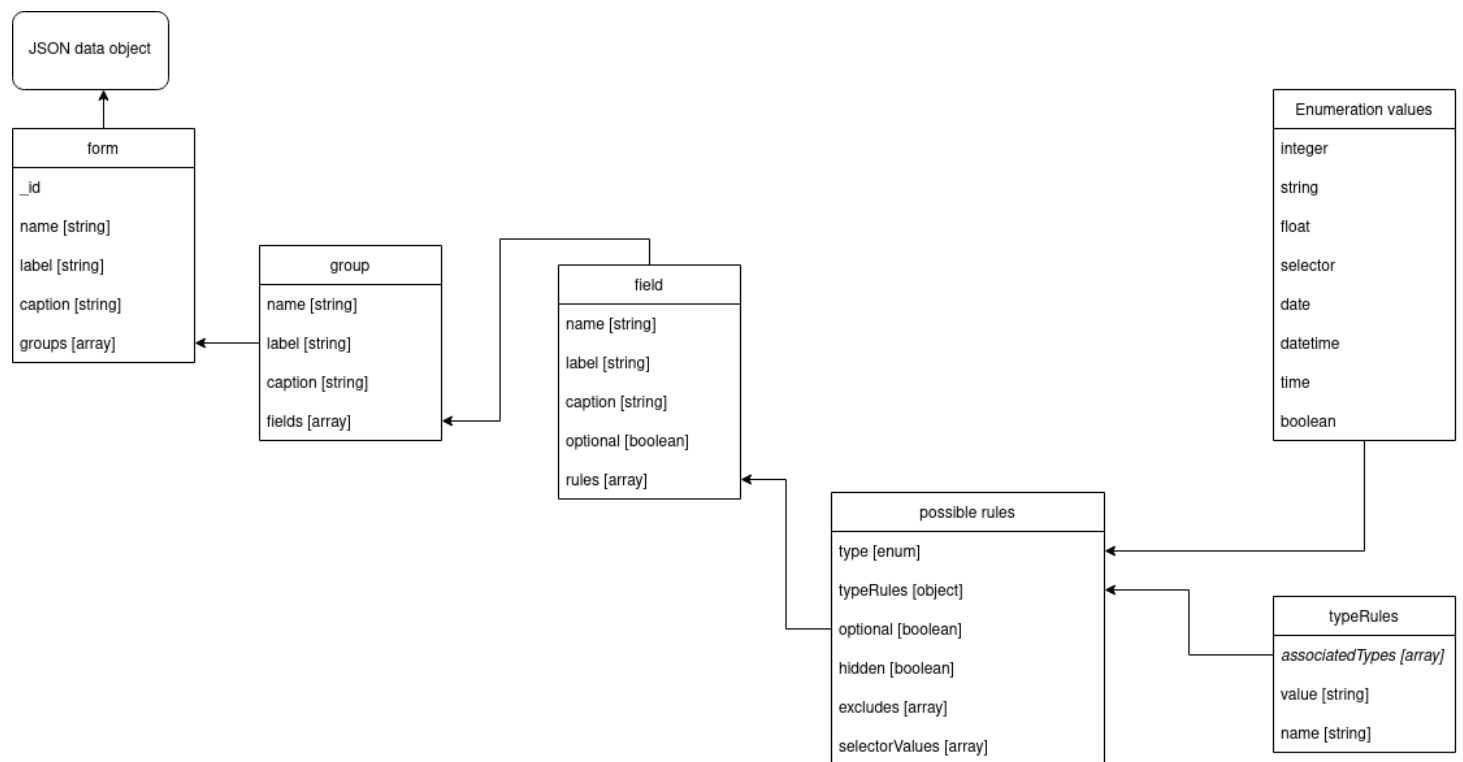
Email

Submit

Le formulaire « add User » généré à partir du fichier de configuration permet la saisie et la vérification des données. Après remplissage de ce dernier les données sont enregistrées et stockées.

Le nom et les champs dépendent donc aussi des caractéristiques du fichier de configuration ou prefab

Modélisation d'entité : Structure de l'objet Prefab



Le format de l'objet représentant la configuration du formulaire est une partie importante du prototype.

En effet, cet objet doit représenter :

- quels champs existent dans le formulaire ;
- comment sont ils groupés thématiquement et visuellement, et avec quels labels et textes supplémentaires ;
- quel type de données chaque champ accepte ;
- quelles contraintes supplémentaires s'appliquent à ces données pour être acceptées ou non.

Dans l'application backend, nous définissons donc plusieurs entités permettant de représenter cet objet et le manipuler.

Ces classes sont définies dans les packages `fr.utbm.da50.freelyforms.core.entity` et `fr.utbm.da50.freelyforms.core.entity.prefab`.

- La classe Prefab contient tous les attributs nécessaires au frontend pour la génération d'un formulaire.
 - C'est un objet de classe Prefab qui est envoyé, avec tous ses attributs (et les attributs de ses attributs, et les attributs de... le cas échéant), à l'application frontend lorsqu'elle requête un fichier de configuration de formulaire.
 - Les données entrées par un utilisateur et stockées en base (objet de classe FormData) connaissent le nom du Prefab auxquels ils sont rattachés.
 - Un objet de classe Prefab peut vérifier que les objets de classe FormData qui y sont rattachés respectent ses règles (p.ex : mêmes groupes et champs, valeurs entrées dans les champ ont le bon type).
 - Un objet de classe Prefab peut vérifier sa propre validité (format correct, règles valides...)
 - Les objets de classe Prefab peuvent être stockés dans la base de données avec tous leurs attributs.
 - Il ne doit pouvoir y avoir qu'une seule prefab d'un même nom dans la base de données.
 - Les objets de classe Prefab peuvent contenir des objets de classe Group, chacun ayant un nom distinct.
- La classe Group contient des champs de formulaire groupés ensemble (parce que thématiquement similaires, ou parce qu'ils doivent simplement être représentés ensemble).
 - Les objets de classe Group ne sont **pas** stockés en base dans leur propre table/collection ; en base, ils sont toujours représentés par des objets JSON attributs de l'objet JSON représentant la Prefab auxquels ils sont rattachés.
 - Les objets de classe Group peuvent contenir des objets de classe Field, chacun ayant un nom distinct.
- La classe Field représente un champ de formulaire.
 - Les objets de classe Field ne sont **pas** stockés en base dans leur propre table/collection ; en base, ils sont toujours représentés par des objets JSON attributs du groupe auxquels ils sont rattachés, dans la Prefab qui les contient tous les deux.
 - Un objet de classe Field contient un objet de classe Rule, qui fixe les règles régissant l'apparence et les valeurs acceptées par ce champ.
- La classe Rule représente des règles générales s'appliquant à un champ.
 - Les objets de classe Rule ne sont **pas** non plus stockés en base dans leur propre table/collection.
 - Un objet de classe Rule définit le type de valeur pour un champ (entier, texte, date...) et d'autres options, comme si le champ est optionnel ou si entrer des valeurs à l'intérieur rend impossible l'entrée de valeurs dans un autre champ.
 - Certaines règles sont trop spécifiques pour être applicable à n'importe quel type de valeur, ou consommeraient de la place inutile si stockées dans un

- objet pour être inutilisées la grande majorité du temps ; de telles règles sont de classe `TypeRule` et contenues par l'objet de classe `Rule`.
- Lors de la vérification de données, la classe `Rule` vérifie les données avec ses règles générales et les règles de chaque `TypeRule`.
 - La classe `TypeRule` représente des règles spécifiques, qui peuvent concerner aussi bien la valeur d'un champ que son affichage graphique.
 - Les objets de classe `TypeRule` ne sont **pas** non plus stockés en base dans leur propre table/collection.
 - L'implémentation de la classe `TypeRule` est d'une classe abstraite avec des méthodes communes (constructeur, vérification de sa propre validité et vérification de la validité de données entrantes) à implémenter dans les classes héritées, et disposant d'une unique valeur interprétée différemment par chaque classe héritée.
 - Par exemple, la classe héritée `MaximumRule` interprète sa valeur comme "une valeur numérique indiquant la longueur maximum d'une donnée textuelle, ou la valeur maximum d'une donnée numérique" ; tandis que `AlternativeDisplay` interprète sa valeur comme "l'apparence que ce champ doit prendre sur le frontend".
 - Les objets de classe héritée de `TypeRule` ne sont valides que pour un nombre limité de type de données (définies dans le constructeur) ; si le type de données défini dans `Rule` ne correspond pas aux types valides de la classe héritée de `TypeRule`, la règle est invalide.

Notez que *toute* la prefab est envoyée à l'application frontend lorsqu'elle est requêtée ; cela permet à l'application frontend d'elle aussi effectuer des vérifications de données sur les valeurs entrées dans les formulaires, sans avoir besoin de requêter le service REST pour vérifier la validité des données.

La classe `FormData`

Les objets de classe `FormData` sont la représentation des données de formulaire entrées par l'utilisateur. Un objet de classe `FormData` contient toujours le nom du formulaire auquel il est lié.

La structure d'un objet de classe `FormData` avec ses attributs ressemble beaucoup à celle d'un objet de classe `Prefab` ; cependant, comme ils ne représentent que des données brutes, un objet de classe `FormData` ne contient que le nom de sa prefab et ses groupes ; les groupes ne contiennent que leur nom et des champs ; et les champs ne contiennent rien d'autre que leur nom et leur valeur.

Le nom de ces groupes et de ces champs doit correspondre aux groupes et champs de la Prefab liée.

TypeRules actuellement définies

Cinq TypeRules sont actuellement définies, qui permettent de modifier ou contraindre le comportement et les valeurs acceptées pour un champ.

- **MinimumRule**
 - Définit soit une valeur minimum pour un champ à donnée numérique, ou une longueur minimum pour un champ à donnée textuelle
- **MaximumRule**
 - Définit soit une valeur maximum pour un champ à donnée numérique, ou une longueur maximum pour un champ à donnée textuelle
- **SelectDataSet**
 - Définit que les valeurs autorisées pour ce champ sont prises dans une liste de données entrées précédemment par les utilisateurs, dans un autre formulaire ; le champ sélecteur (dropdown, boutons...) ne proposera ainsi en option que les valeurs de données correspondant à un Prefab, un groupe et un champ précis.
- **RegexMatch**
 - Définit que la donnée textuelle envoyée doit respecter une expression régulière.
- **AlternativeDisplay**
 - Définit une apparence alternative pour ce champ dans l'interface graphique ; toutes les données vérifiées avec cette règle sont évidemment valides en ce qui la concerne !

Correspondance prefab-formulaire

Nous présentons dans la figure qui suit un exemple de correspondance entre le fichier de configuration et le formulaire généré.

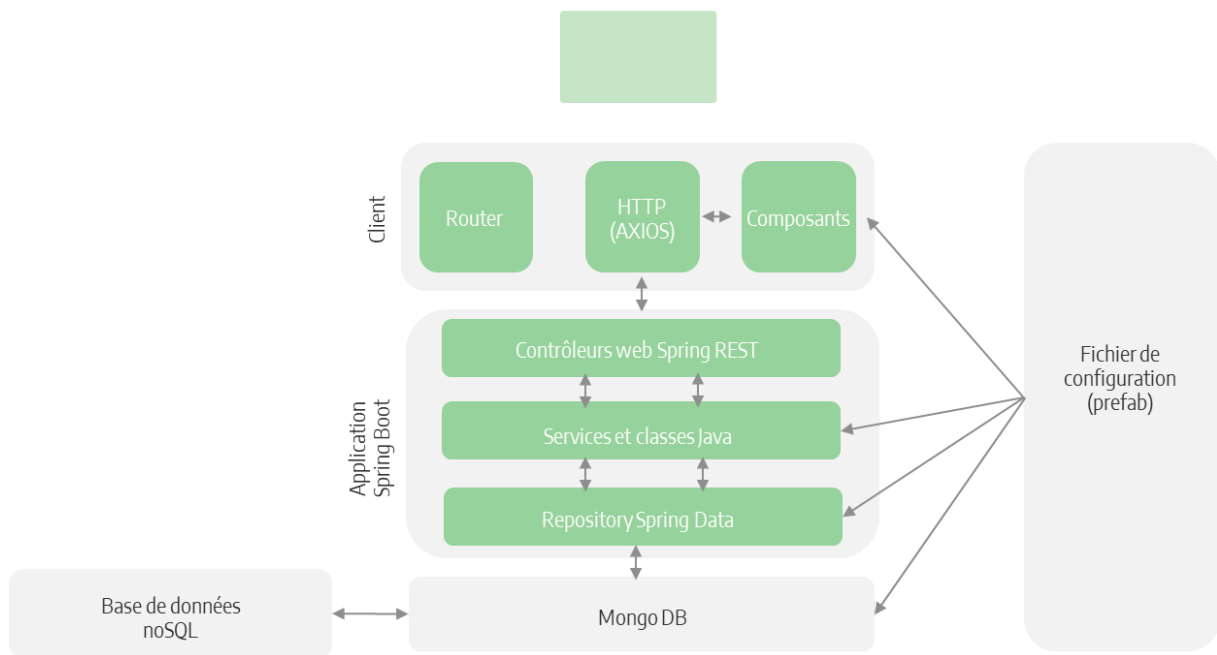
The diagram illustrates the mapping of form elements to configuration keys for an "Add User" form. The form is shown on the left, and the corresponding configuration keys are listed on the right, connected by lines.

- Add User** (Form Title) → `form.label`
- Fill the following form to add a user** (Form Caption) → `form.caption`
- User informations** (Form Group Label) → `form.group.label`
- Gender** (Form Group Field Label) → `form.group.field.label`
- Female** / **Male** (Form Group Field Caption) → `form.group.field.caption`
- First name** (Form Group Field Caption) → `form.group.field.caption`
- Last name** (Form Group Field Caption) → `form.group.field.caption`
- Age** (Form Group Field Caption) → `form.group.field.caption`

The diagram illustrates the mapping of form elements to configuration keys for a user profile form. The form is shown on the left, and the corresponding configuration keys are listed on the right, connected by lines.

- Company Name** (Form Group Field Caption) → `form.group.field.caption`
- Role** (Form Group Field Label) → `form.group.field.label`
- Verified** (Form Group Field Label) → `form.group.field.label`
- is verified** / **not verified** (Form Group Field Caption) → `form.group.field.caption`
- Status** (Form Group Field Label) → `form.group.field.label`
- Yes** / **No** (Form Group Field Caption) → `form.group.field.caption`
- Seniority** (Form Group Field Caption) → `form.group.field.caption`
- Entry Date** (Form Group Field Label) → `form.group.field.label`
- Submit** (Form Button) → `form.button`

Structure technique générale des composants du projet



Ce premier schéma, bien que primitif, permettait de décrire l'intervention de l'objet de configuration JSON au sein des différents composants.

On note 3 parties importantes :

- La technologie de persistance qui permet le stockage des données, le serveur d'application qui contient la partie traitement de l'information, et enfin la dernière partie permettant présentation des données et leur modification par l'utilisateur.
- Le fichier de configuration (prefab) est un élément très important car il permet de dire quelle forme vont prendre les données et les formulaires affichés à l'utilisateur.
- Le repository permet d'accéder à la technologie de persistance et pourra lire et écrire n'importe quel type de données. Les services de données seront capables de structurer, traiter et valider les informations reçues.
- Enfin le contrôleur web permet d'interfacer avec la partie présentation de l'application.

Architecture applicative

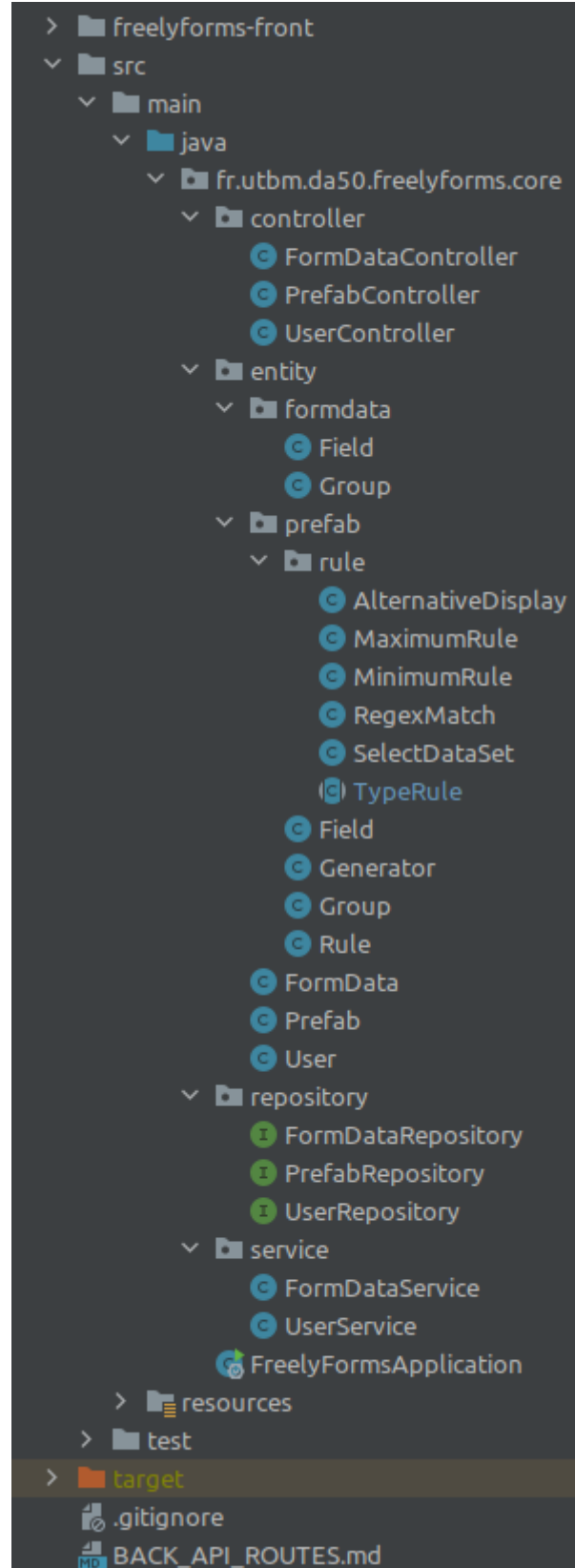
La structure de l'application backend est classique pour une application Spring Boot.

Les classes du package Controller forment l'interface entre l'application et les requêtes HTTP entrantes, et définissent les routes API.

Les classes du package Entity définissent la structure des classes stockées en base, et des classes "contenues" par les précédentes. Ces classes disposent aussi d'une variété de méthodes leur permettant d'interagir entre elles.

Les classes du package Repository forment l'interface entre l'application et la base de données.

Enfin, les classes du package Service servent d'intermédiaire entre les classes du package Repository et le reste du programme (contrôleurs, mais aussi certaines méthodes pouvant appartenir aux entités).



Originellement, les applications frontend et backend appartenaient à deux projets différents.

Cependant, nous avons choisi de fusionner les deux projets entre eux, sur le même repository et dans le même projet Gitlab. Cela nous a permis de grouper les tâches Gitlab et de tester plus facilement le frontend et le backend ensemble.

L'application frontend est donc dans le même projet que l'application backend, et est contenue dans le dossier `freelyforms-front` à la racine du projet.

Surtout, cela nous a permis de créer un unique livrable de production groupé, lançant à la fois les applications frontend et backend avec un seul fichier `.jar`.

La description de cette opération est consultable dans le fichier `README.md`, à la racine du projet ; les plugins utilisés et les détails de l'implémentation sont consultables dans le fichier `pom.xml`.

Codage de l'application

Le début de la création de notre application a été lancé en novembre. Nous avons commencé par établir un plan détaillé de ce que nous voulions réaliser et comment nous allions procéder en utilisant le diagramme de classe. Nous avons ensuite divisé les tâches entre les membres du groupe en fonction de nos compétences et de nos intérêts, deux membres ont commencé le codage de la partie backend et deux la partie frontend. Le premier pas a été de définir le cahier des charges de l'application, qui décrivait les fonctionnalités et les objectifs que nous souhaitions atteindre. Une fois cela fait, nous avons commencé à écrire le code de base de l'application en utilisant les langages de programmation adaptés à nos besoins à savoir Spring Boot et ReactJS. Nous avons travaillé en étroite collaboration pour mettre en place les différentes parties de l'application et nous nous sommes assurés que tout fonctionnait correctement avant de passer à l'étape suivante.

Pour des raisons de contraintes de temps et d'organisation, nous n'avons néanmoins pas réussi à mener ce projet à bout. L'un des membres de notre équipe a disparu à la mi-semestre, et les différents membres de notre équipe ont souvent eu beaucoup de mal à communiquer entre eux.

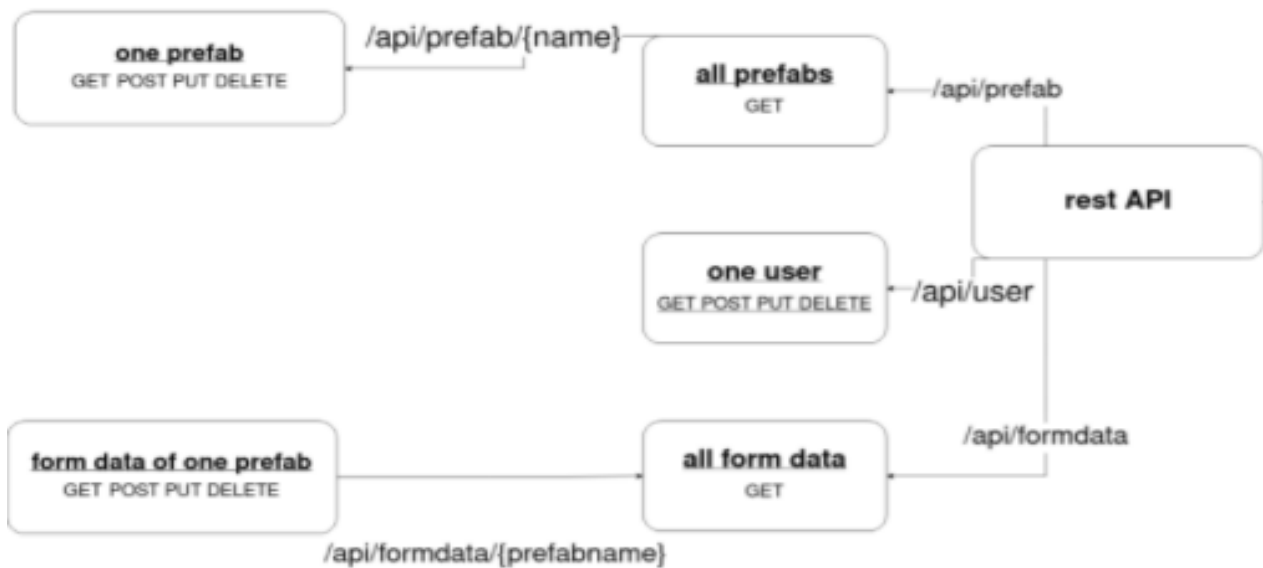
Nous avons cependant fourni une quantité non négligeable de travail et avons réussi à réaliser un grand nombre de réalisations d'implémentation, en plus du travail de conception que nous avons réalisé lors du semestre.

Implémentation complète de la classe Prefab

Le fichier de configuration de formulaire a été, autant pour le groupe précédent que pour le nôtre, un enjeu majeur de la réalisation de ce projet.

La classe Prefab et ses dérivés, décrits en longueur plus haut dans ce document, ont été intégralement implémentés, documentés et testés.

Implémentation de l'API



Une API complète pour l'implémentation actuelle des entités a été réalisée et testée.

Les routes API actuelles sont :

Fichiers de configuration (Prefabs)

GET /api/prefabs	<i>Retourne tous les Prefabs</i>
GET /api/prefabs/{name}	<i>Retourne le Prefab de nom {name}</i>
POST /api/prefabs	<i>Crée le Prefab de nom {name} passé dans le corps de la requête</i>
PATCH/PUT /api/prefabs/{name}	<i>Modifie le Prefab de nom {name} avec celui passé dans le corps de la requête</i>
DELETE /api/prefabs/{name}	<i>Supprime le Prefab de nom {name}</i>

Données de formulaire entrées par les utilisateurs (FormData)

GET /api/formdata	<i>Retourne toutes les FormData</i>
GET /api/formdata/{prefabName}	<i>Retourne toutes les FormData de nom de prefab {prefabName}</i>
GET /api/formdata/{prefabName}/{groupName}/{fieldName}	

Retourne toutes les valeurs de nom de prefab {prefabName}, de groupe {groupName} et de champ {fieldName}

POST /api/formdata

Crée le FormData passé dans le corps de la requête

PATCH/PUT /api/formdata/{prefabName}

Modifie un FormData de nom de prefab {prefabName} avec celui passé dans le corps de la requête (identifiable par son ID)

DELETE /api/formdata/{prefabName}

Supprime un FormData de nom de prefab {prefabName} (identifiable par son ID)

Données utilisateur (User)

GET /api/users/{id}

Retourne l'utilisateur d'ID {id}

POST /api/users

Crée l'utilisateur passé dans le corps de la requête

PATCH /api/users/{id}

Modifie l'utilisateur d'ID {id} en le remplaçant par celui passé dans le corps de la requête

DELETE /api/users/{id}

Supprime l'utilisateur d'ID {id}

Cette description est aussi disponible dans le fichier BACK_API_ROUTES.md, à la racine du projet.

Frontend avec formulaire basique

Un frontend fonctionnel avec un formulaire basique a été implémenté et ajouté à l'application.

Livable unique pour l'application entière

L'application est compilable en un unique livrable .jar qui lance les applications frontend et backend ensemble. Les informations d'installation pour réaliser cela sont présentes dans le fichier README.md, à la racine du projet.

Réalisations incomplètes

Plusieurs fonctionnalités ont été débutées mais n'ont pas pu être terminées à tant pour le rendu du projet, et restent donc encore à effectuer.

Les objets de classe FormData ne sont pas actuellement vérifiés avant d'être inscrits en base de données, que ce soit avec les règles de leur objet de classe Prefab liée... ou même pour vérifier que cette prefab existe.

On pourrait certainement ajouter des règles liées au type supplémentaire en les faisant hériter de TypeRule.

Actuellement, l'application de frontend ne communique pas avec le service REST backend, et ne génère donc pas de formulaire à partir de ses fichiers de communication. Elle n'envoie donc pas non plus de données au backend, ni ne vérifie dynamiquement les données entrées dans un formulaire généré à partir des fichiers de configuration.

Documentation

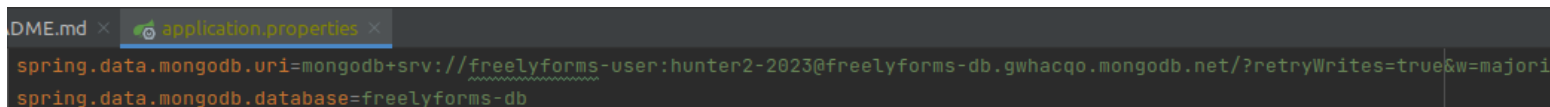
L'un des grands problèmes que nous avons rencontrés au début de ce projet était le manque de documentation de l'existant. Nous étions censés partir d'un projet déjà entamé, mais le manque de documentation (voire parfois, de code à jour) sur l'implémentation existante nous a forcés à repartir de zéro en termes de code.

Assez tôt dans le cycle de vie du projet, il est devenu évident que nous n'arriverions pas à accomplir nos objectifs à temps. Pour cette raison, nous avons choisi de documenter de façon extensive notre projet ; ainsi, nous espérons que la prochaine équipe qui se penchera sur ce projet pourra utiliser cette implémentation comme base pour développer des fonctionnalités manquantes, en ajouter de nouvelles, améliorer l'existant, et enfin mener à bout ce projet au troisième essai.

La documentation disponible est établie comme suit :

- Le fichier README.md, à la racine du projet, contient les informations minimales d'installation et contient des informations sur l'implémentation.
- Le fichier BACK_API_ROUTES.md, à la racine du projet répertorie les routes API actuellement disponibles et fonctionnelles.
- Ces routes API peuvent être immédiatement testées avec Postman, une fois la base de données initialisée ; on peut pour cela utiliser les requêtes à importer situées dans le document `freelyforms-back.postman_collection.json`, situé à la racine du projet.
 - *Note : pour se connecter à la base de données Mongo avec springdata, il vous faut un fichier `application.properties` dans le dossier `src/main/resources` contenant des informations de connexion.*

Voilà un exemple d'un tel fichier, pour se connecter à une BDD partagée sur un cluster Atlas partagé :



```
DME.md x application.properties x
spring.data.mongodb.uri=mongodb+srv://freelyforms-user:hunter2-2023@freelyforms-db.gwhacqo.mongodb.net/?retryWrites=true&w=majori
spring.data.mongodb.database=freelyforms-db
```


- **Toutes** les classes Java du projet ont été documentées avec l'outil Javadoc, standard pour documenter les fichiers Java pour en décrire l'objectif et le fonctionnement. Toutes les méthodes non-évidentes ont été décrites, en particulier au niveau des paramètres attendus et du sens de leur valeur de retour. Ces commentaires sont lisibles en ouvrant n'importe quel fichier source Java.
- Enfin, le dossier doc/, à la racine du projet, comporte une version générée et plus facilement lisible de la Javadoc de tout le projet dans le format HTML, avec la description des classes et des méthodes.

Conclusion, Avis des membres

Hamza KHOUBILA

Le projet sur lequel on a travaillé ce semestre était très intéressant. En utilisant un Framework comme Spring Boot pour coder le backend et la bibliothèque React pour le frontend, on a pu quand même créer une application qui répond aux besoins client en combinant ces deux technologies puissantes et solides.

Le travail avec Spring Boot a été agréable grâce à sa facilité d'utilisation et à sa flexibilité.

Le travail sur ce projet était enrichissant, on a pu apprendre beaucoup de choses lors de sa réalisation, malgré les difficultés rencontrées au niveau de la réalisation des tâches et la collaboration.

Souhaila RABAI

Ce projet a été très enrichissant, que ce soit sur le plan conceptuel ou technique, même si au début j'avais du mal à comprendre le but du projet tout est devenu plus clair avec les multiples réunions que ce soit avec l'équipe ou avec Monsieur Olivier RICHARD.

Côté développement, ce projet m'a permis de découvrir React. js. Cela m'a aussi permis de découvrir Spring côté Backend ce qui m'a permis d'appliquer les points abordés dans le cours DA52 'Java entreprise'.

Ernest ENGOUE

Ce projet a été très enrichissant car il m'a permis d'appliquer les connaissances acquises dans l'UV DA52 . Il m'a également permis de travailler avec de nouvelles technologies. En plus de cela, le fait de travailler dans une équipe diverse m'a appris l'importance de la communication et des outils de travail collaboratif.

Céleste Guilleux

Ce projet a été au départ pour moi une certaine opportunité. Je découvrais à peine le développement Web lors de mon stage très récemment fini, et j'y voyais l'opportunité de développer encore plus mes compétences à ce niveau là. Je ne connaissais aucune des technologies que nous avons utilisé, et cela me faisait un peu peur ; mais découvrir de nouvelles technologies fait partie des compétences essentielles d'un développeur.

J'ai eu l'impression d'avoir pris en main assez vite les technologies utilisées, et j'étais même un temps satisfaite de notre avancée.

Malgré cela, ce projet a été franchement éprouvant à réaliser, et il était rapidement apparent que nous n'atteindrions pas l'objectif fixé.

En tant que cheffe de groupe, j'avais la responsabilité d'organiser le travail entre mes camarades et de garder une vision d'ensemble du projet pour qu'il reste réalisable. J'étais loin de me douter que cette vision et cette compréhension que j'avais du projet n'était parfois pas du tout la même que celle de mes autres camarades, malgré mes nombreuses conversations semblant supposer du contraire au début du semestre.

Nous avons perdu une quantité énorme de travail à cause de ces erreurs de communication. Des tâches ont été refaites, recommencées, et même dupliquées entre deux membres du groupe différents qui ont effectué la même chose pendant plusieurs semaines.

Avoir perdu le cinquième membre de notre groupe, Feiyang Yin, à la mi-semestre, après avoir passé un temps non négligeable à tenter de lui expliquer le contexte et l'objectif du projet, m'a fait assez mal à la motivation, et je me suis beaucoup demandée si il serait resté si j'avais peut-être été moins rude avec lui. J'aurais aimé avoir plus de cours de management. J'aurais aimé avoir pu acquérir les compétences de manager avant d'être propulsé dans le rôle par vote unanime.

Quoi qu'il en soit, je ne peux pas et n'attribuerai pas nos retards et nos erreurs à un manque de bonne volonté et de compétence de mes camarades de projet. Les difficultés avec la langue, le rythme du semestre scolaire, le manque de familiarité de l'équipe avec les technologies utilisées (et la gestion de projet en général !) et bien entendu la **différence dramatique entre la qualité de nos matériels** étaient à mon avis des facteurs sensiblement plus déterminants sur l'échec (très relatif, je maintiens qu'on s'en est bien sorti) du projet qu'une quelconque question de volonté ou de compétences.

Enfin, si vous lisez ce rapport technique en septembre '23 et que vous avez des questions sur l'implémentation actuelle, si vous décidez pas de la refaire (*pitié décidez pas de la refaire, épargnez vous ça*), n'hésitez pas à m'envoyer un mail ou un message. Bon courage à vous, et je vous souhaite d'enfin finir ce projet maudit. La troisième sera la bonne, pas vrai ?

Ouverture

Plusieurs possibilités existent pour faire évoluer le projet plus avant (outre le fait de finir ses fonctionnalités prévues).

Authentification et permissions utilisateur

Actuellement, les formulaires et données sont lisibles et modifiables par n'importe qui. Cela peut poser problème si l'on souhaite récupérer des données de sondage, par exemple, puisqu'un utilisateur peut tout simplement éditer les réponses de tous les autres utilisateurs pour fausser le sondage.

On pourrait donc réaliser un système d'authentification utilisateur qui nécessite aux utilisateurs de s'identifier (sur une connexion sécurisée) pour pouvoir accéder, ajouter ou modifier certaines données et configurations.

Les formulaires auraient alors des permissions spécifiques pour déterminer qui peut accéder à, lire et modifier un formulaire donné et ses valeurs.

Interface graphique d'ajout et de modification des configurations de formulaire

Dans la définition actuelle du projet, les prefabs ne sont pas modifiables une fois rentrés dans la base de données ; les routes API existent, mais leur utilisation risque de rendre toutes les valeurs déjà entrées pour ce formulaire invalides et inutilisables.

Une implémentation de modification des configurations demanderait donc de migrer les données actuelles vers de nouvelles valeurs correspondant au nouveau format du formulaire ; l'utilisateur final devrait pouvoir définir les valeurs vers lesquelles les données existantes migrent.

Des sauvegardes des anciennes données devraient être créées, et on devrait permettre leur rétablissement en cas d'erreur sur la modification de la configuration de formulaire.