UCLouvain

epl

École polytechnique de Louvain

Storybook for Odoo

Integration of a frontend workshop to build user interface components and pages in isolation in Odoo

Authors: **Florent Dardenne, Maximilien La Barre**
Supervisor: **Axel Legay**
Readers: **Peter Van Roy, Benoît Duhoux**
Academic year 2022–2023
Master [120] in Computer Science

Remerciement

Nous tenons à exprimer notre profonde gratitude envers tous ceux qui ont contribué à la réalisation de ce mémoire, qu'il s'agisse de nos proches, de nos amis ou de toute autre personne ayant participé à ce projet comme notre promoteur (Axel Legay) et nos lecteurs (Peter Van Roy et Benoît Duhoux).

Nous souhaitons également adresser nos sincères remerciements à Odoo pour leur confiance, leur accueil chaleureux et leur soutien tout au long de ce projet. Nous tenons particulièrement à remercier les équipes Javascript et Accounting pour leurs précieux conseils et leur expertise.

Ce mémoire n'aurait pas été le même sans votre contribution. Nous vous sommes donc infiniment reconnaissants pour votre soutien, vos encouragements et vos précieux conseils.

Maximilien La Barre et Florent Dardenne.

Contents

Glossaire

- ERP: Un ERP (Enterprise Resource Planning) est un système informatique utilisé par les entreprises pour gérer leurs opérations quotidiennes, telles que la gestion des ressources humaines, des stocks, de la comptabilité, des ventes, des achats, etc. [1]
- ARR: Acronyme pour Annual Recurring Revenue. C'est à dire toutes les recettes courantes d'un produit ou d'une entreprise, projetées sur une année.[2]
- CEO: Le CEO (Chief Executive Officer) est le dirigeant d'une entreprise, responsable de la stratégie globale et de la prise de décisions importantes. [3]
- CTO: Le CTO (Chief Technology Officer) est le dirigeant chargé de la gestion de la technologie et de l'innovation au sein d'une entreprise. [4]
- CFO: Le CFO (Chief Financial Officer) est le dirigeant chargé de la gestion financière de l'entreprise. [5]
- CCO: Le CCO (Chief Customer Officer) est le dirigeant chargé de la gestion de la relation client dans l'entreprise. [6]
- ORM: ORM (Object-Relational Mapping) est une technique de programmation qui permet de faire correspondre les objets d'un programme aux enregistrements d'une base de données relationnelle. [7]
- MVC: "Model-View-Controller" est un "design pattern" qui sépare les données (Model), l'interface utilisateur (View) et le contrôleur (Controller) qui gère les interactions entre les deux. [8]
- Props: Les Props sont des objets qui permettent de passer des données entre les composants.
- Story: Une story est une page de documentation interactive qui montre comment un composant doit être utilisé. C'est un outil de développement qui permet de tester les composants de manière isolée.

- CSS: (Cascading Style Sheets) Il permet de décrire comment les éléments HTML vont être affichés sur l'écran, il facilite grandement le design des pages. [9]
- Proof of concept: Un proof of concept est une démonstration pratique visant à valider la faisabilité ou l'efficacité d'une idée.

Introduction

Dans de nombreuses grandes entreprises, disposer d'une documentation complète et à jour est un vrai défi. Souvent fastidieuse et chronophage à mettre à jour pour les employés, elle peut être confus, contenir des informations erronées ou périmées. Dans ce document, nous allons nous pencher sur une solution pour une entreprise en particulier, Odoo.

Odoo est une entreprise Belge proposant une suite d'applications (ou modules) qui ont pour but de répondre à tous les besoins quotidiens d'une entreprise, qu'il s'agisse de la comptabilité, des stocks, des ressources humaines ou du site web, etc. Avec une partie de leur code disponible au public gratuitement (open source) et une autre payante, il est accessible à plus de 5000 partenaires agréés, 10,8 millions d'utilisateurs actifs actuellement, une documentation complète et facile à comprendre est primordiale. Toujours à la recherche de nouvelles innovation, il y a quelques années, la direction d'Odoo a choisi de créer son propre framework, Owl, pour faciliter le développement de leur plateforme. [10]

Dans le cadre de ce mémoire, nous avons développé une plateforme destinée à aider les développeurs travaillant sur l'environnement Odoo (employés, partenaires ou communauté travaillant sur la partie open-source). Cette application a été conçue pour faciliter la visualisation et l'interaction avec une grande variété de composants de l'interface Odoo, offrant ainsi aux développeurs la possibilité de les visualiser dans différents contextes et de tester facilement l'impact visuel et fonctionnel d'un changement des attributs de la vue. Notre plateforme se veut donc être une documentation détaillée et interactive qui permettra à qui le souhaite de comprendre comment utiliser un composant présent dans Odoo.

Ce mémoire sera divisé en plusieurs chapitres. Tout d'abord, nous présenterons l'entreprise Odoo, en expliquant son histoire, les produits proposés, l'organisation générale ainsi que les différents éléments techniques utilisés par l'entreprise. Cela permettra d'avoir une vue globale avant de rentrer dans les détails de notre projet.

Ensuite, nous expliquerons ce qu'est Storybook, une plateforme permettant de documenter et de tester un framework dont le principe est proche de ce que nous voulons construire. Nous aborderons également un chapitre sur le framework créé par Odoo, appelé "Owl", suivi ensuite d'une section sur les projets open source. Enfin, nous expliquerons deux outils que nous avons utilisés tout au long de ce mémoire : Trello et Github.

Le chapitre suivant portera sur les spécifications de ce mémoire, avec une explication complète du cahier des charges, avec qui nous avons été en contact ainsi que les ressources qui nous ont été mises à disposition.

Ensuite, nous détaillerons comment s'est déroulé le developpement de la plateforme au niveau de l'organisation et de la méthodologie. Nous expliquerons également son évolution au fur et à mesure du développement, ainsi que les différents choix que nous avons faits et les explications techniques sur sa réalisation.

Pour finir, nous détaillerons les idées d'amélioration pour la plateforme, afin qu'il soit apprécié par la communauté d'utilisateurs futurs.

Contexte

4.1 Odoo

4.1.1 Histoire

Odoo est un logiciel de gestion d'entreprise (ERP) open source créé en 2002 en Belgique. Bien que l'entreprise soit maintenant connue pour ses voitures ou son logo sur les verres utilisés lors des soirées étudiantes, elle a commencé comme une start-up dans la région de Louvain-la-Neuve. [11] [12]

Après avoir sorti une première version en 2005 sous le nom de TinyERP, l'entreprise n'a pas connu immédiatement la croissance qu'elle connaît aujourd'hui. Cependant, grâce à l'aide d'investisseurs et au travail acharné de son petit groupe d'employés, l'expansion a commencé.

La mission de l'entreprise était et reste de fournir des logiciels intuitifs, efficaces, totalement intégrés et adaptés aux réalités du marché, tout en restant simples à utiliser.

En 2009, TinyERP change de nom pour devenir OpenERP suite à la sortie de la version 5.0 du logiciel sur le marché. On peut se demander pourquoi l'entreprise a décidé ce changement en cours de route. Une rumeur court que ce changement de nom serait dû à une réunion entre Fabien (le CEO d'Odoo) et le directeur du groupe Danone qui lui aurait demandé pourquoi il achèterait une licence à un million de dollars pour un "Tiny" ERP.

C'est à partir de ce changement de nom que l'entreprise a commencé à se faire une vraie place dans le domaine de la tech en Belgique. Entre 2008 et 2014, OpenERP a connu une croissance fulgurante que personne n'a vue venir. Grâce à une stratégie pertinente sur le marché et un recrutement intelligent, l'entreprise a employé de plus en plus de personnes et a surtout commencé son expansion

mondiale ! De nouveaux bureaux ont été ouverts en Europe, en Asie, en Afrique, en Amérique du Nord et du Sud.

En 2014, l'entreprise décide à nouveau de changer de nom, à l'occasion de la sortie de la version 8.0 de son logiciel. En effet, celui-ci proposait bien plus qu'un simple ERP, et un nouveau nom était nécessaire pour se démarquer de la concurrence. Après une longue réflexion, l'entreprise a choisi de s'appeler "Odoo", un nom qui reflète mieux la diversité des fonctionnalités offertes par le logiciel. Avec plus de 900 modules de base proposés mais surtout plus de 40000 modules fais par la communautée, Odoo est bien plus qu'un ERP classique, tout en gardant les fondements qui ont fait son succès.

Dans un récent podcast animé par Fabien
citepodcast, on apprend que le choix du nom "Odoo" est également lié à une raison plutôt amusante : le fondateur voulait que son entreprise ait un nom similaire à ceux des géants du web comme Google ou Facebook, qui contiennent tous deux la lettre "o" deux fois. Si vous vous demandez ce que signifie Odoo, il s'agit en fait de l'acronyme de "On-demand Open Object", où "Open Object" est le framework sur lequel OpenERP était basé.

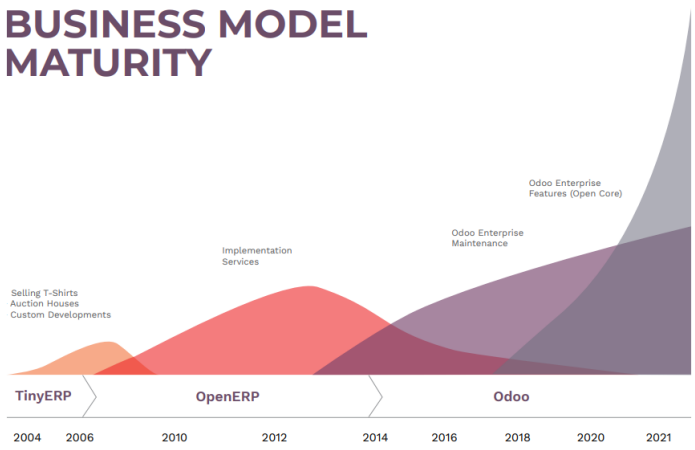

Figure 4.1: Evolution du business model

En 2021, Odoo a contribué à la création de plus de 60 000 emplois dans le monde entier. Selon les estimations, environ 900 employés travaillaient directement chez Odoo en Belgique, et 1850 partout dans le monde, tandis que 32 000 employés travaillaient pour les 2 600 partenaires de l'entreprise. En outre, il était estimé que

plus de 28 000 personnes travaillaient à temps plein dans la communauté Odoo en tant que développeurs, consultants ou membres actifs de la communauté. Cependant, ces chiffres ne sont plus totalement exacts pour le premier quadrimestre de 2023, car Odoo compte désormais 2830 employés dans le monde entier et a doublé le nombre de partenaires pour atteindre 5128 avec un total de 80000 employées, ceci additionné au 40000 personnes de la communautés. On peut voir à quel point ces chiffres ne font qu'évoluer ! [13] [14]

Odoo est utilisé par des milliers d'entreprises dans le monde entier, avec un nombre total d'utilisateurs atteignant jusqu'à 10.8 millions. Le logiciel offre une solution complète de gestion d'entreprise qui couvre un large éventail de domaines, notamment la comptabilité, la gestion des ventes et des achats, la gestion de la production, la gestion de projet, la gestion des ressources humaines et bien plus encore. Cette polyvalence permet à Odoo de s'adapter aux besoins de nombreux types d'entreprises, de la start-up à la grande entreprise. De plus, Odoo est enseigné dans près de 500 universités. Chaque année, environ 50 000 étudiants créent gratuitement des bases de données Odoo Online.

À l'heure actuelle, Odoo est à la version 16.0 et se prépare à la sortie de la version 17.0 en novembre. Cette nouvelle version sera présentée devant plus de 10 000 personnes dans différents palais de Brussels Expo. La version gratuite ainsi que la version payante seront alors exposés lors d'une multitude de présentations durant plusieurs jours. [15] Avec une croissance toujours aussi impressionnante et des prix qui battent toute concurrence, on peut entendre Fabien utiliser la phrase "Se payer les services d'Odoo revient à payer le café à ses employés" lors de différentes conférences avec ses employés. Elle est donc devenu un véritable concurrent pour d'autres sociétés qui vendent des produits similaires comme SAP, Google, Saleforce, etc. A savoir qu'Odoo est disponible en tant que service cloud, ce qui permet aux entreprises de déployer et de l'utiliser sans avoir à gérer les infrastructures de serveur ce qui est très intéréssant pour des entreprises de toute taille.

Financièrement parlant, Odoo se débrouille également très bien. Dans un récent post LinkedIn, le CEO Fabien a partagé les diapositives destinées aux investisseurs qui donnent les chiffres actuels de l'entreprise [13].

Dans ces diapositives, on peut y retrouver plusieurs graphiques qui nous permettent de nous rendre compte de l'ampleur d'Odoo. Avec une croissance annuelle moyenne de 55%, Odoo a enregistré un revenu récurrent annuel (ARR) de 190 millions d'euros pour le premier quadrimestre de 2023.

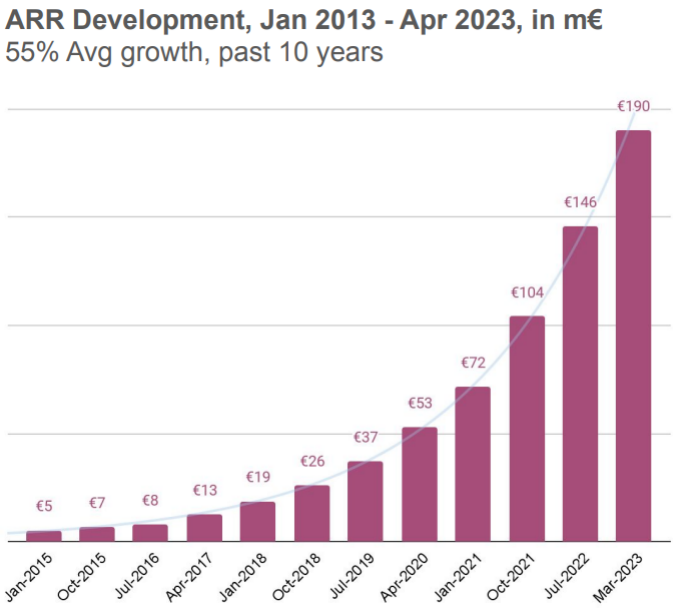

Figure 4.2: Revenue de l'entreprise

Actuellement, le business model d'Odoo fonctionne à 83% avec les abonnements (mensuels ou annuels) et à 17% avec des services dits "one shot". Une fois que ces clients ont utilisé Odoo, la plupart renouvellent leur abonnement. Dans un récent podcast, Fabien dit même qu'Odoo est comme une drogue dure où quand on commence, on ne peut plus s'arrêter [16]. Une façon de voir à quel point les gens sont contents du produit est le "net retention". Pour expliquer cela, on peut imaginer que l'on a une valeur de 100 euros pour renouveler l'abonnement l'année suivante. Combien de ces 100 euros vont rester en excluant les nouveaux clients, mais on garde ceux qui décident d'arrêter, de "downgrade" ou "upgrade" leur abonnement). Le "net retention" d'Odoo est actuellement de 125%, ce qui signifie qu'en général, les gens qui choisissent Odoo restent et certains vont "upgrade" leurs abonnement. Dans le même podcast, Fabien dit qu'en voyant ce chiffre, il pourrait licensié tout son département de ventes et que rien qu'avec les clients actuels, l'entreprise aurait une croissance de 25%, ce qui est tout à fait remarquable !

4.1.2 Produit

Le principal but d'Odoo est de proposer une suite de modules à ses clients dans le but de couvrir l'intégralité de leurs besoins, peu importe la taille de leur entreprise. Avec un répertoire extrêmement varié de 63 applications principales et 870 modules

complémentaires qui ajoutent des fonctionnalités supplémentaires à ces applications, Odoo veille à ce qu'il y ait une application pour chaque besoin. Parmi celles-ci, on peut citer la comptabilité, la gestion des stocks, des ventes, des achats, des employés, des points de vente et bien d'autres encore. [17] [18]

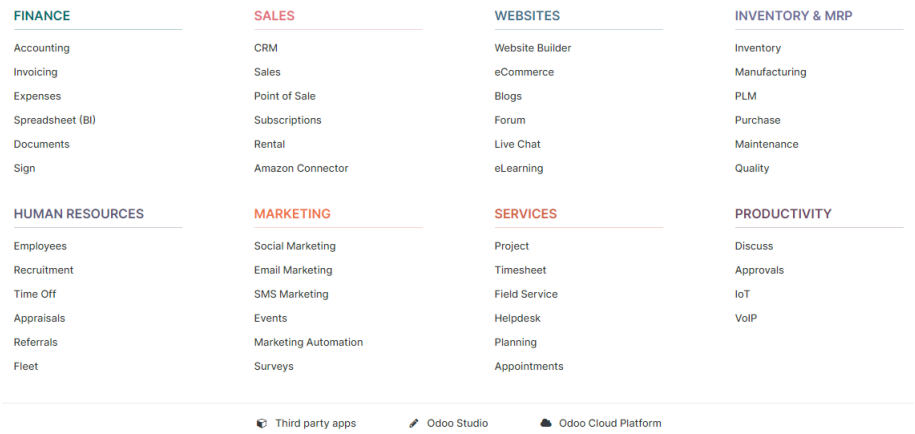

Figure 4.3: Principales applications

De plus, les applications proposées sont personnalisables selon les besoins du client, que ce soit directement par celui-ci ou par une équipe de développeurs travaillant chez Odoo ou chez un partenaire agréé. Odoo propose également un marché d'applications payantes et gratuites, offertes par la communauté très active qu'il possède.

Quel est le public cible d'Odoo ? Les clients principalement visés par Odoo sont les petites et moyennes entreprises qui cherchent à réduire leurs coûts tout en souhaitant automatiser des flux et des tâches chronophages qui peuvent entraîner une perte de revenus et de temps. Avoir un logiciel, qui permet d'automatisé ces flux ou taches est alors primordiale. Le probleme ? Le cout de ce genre de logiciel. Voici une comparaison entre Odoo et SAP (un autre géant de la tech proposant un ERP)

Pricing & Conditions	Odoo Online	SAP Business One
PRICING *	$25 / user / month	$2,975 / user perpetual + 18% per year
Contract Duration	Monthly / Annual	Annual
Five Year Cost, 50 Users	$75,000	$282,625
Free Trial	✓ *	× *
New Version Upgrades Included	✓	✓
Update Service Included	×	×
Cloud Offer Available	✓	✓

Figure 4.4: Comparaison de tarifs

En comparant les deux tarifs, on se rend rapidement compte du public cible d'Odoo. Grâce à sa modularité et à son efficacité, Odoo propose un produit bien plus complet qu'un simple ERP, ce qui explique sa croissance et sa popularité croissantes. Mais ce n'est pas la seul raison de son ascension fulgurante, depuis Octobre 2022 Odoo a complètement changer ses tarifs. On peut voir sur ce schéma la croissance que cette stratégie a rapporté. [19]

Figure 4.5: Conséquence nouveau tarif

Il est clair que la publicité via le bouche-à-oreille, les réseaux sociaux et même les voitures est amplement suffisante pour faire croitre l'entreprise, à savoir que le département de vente ne démarche pas de nouveaux clients. Le département de

vente est divisé en deux secteurs : les ventes directes et indirectes. [11]

- Les ventes directes font référence aux ventes générées par le département de vente d'Odoo.
- Les ventes indirectes se réfèrent aux ventes générées par des partenaires et des affiliés d'Odoo.

Grâce à la mondialisation croissante et à l'ouverture de plus en plus de bureaux dans le monde entier, Odoo a étendu sa portée mondialement. En 2022, Odoo a ouvert son premier bureau en Afrique (au Kenya) [20], ainsi que des bureaux en Australie, au Brésil, en Espagne, en Allemagne et en Italie [21]. En conséquence, l'entreprise compte un grand nombre de clients et de développeurs du monde entier qui participent à l'amélioration continue de son produit.

Nous n'avons pas encore abordé un sujet important concernant Odoo : les partenaires. Avec actuellement 5128 partenaires, présent dans 175 pays ils sont une part extrèmement importante. [22] [23]

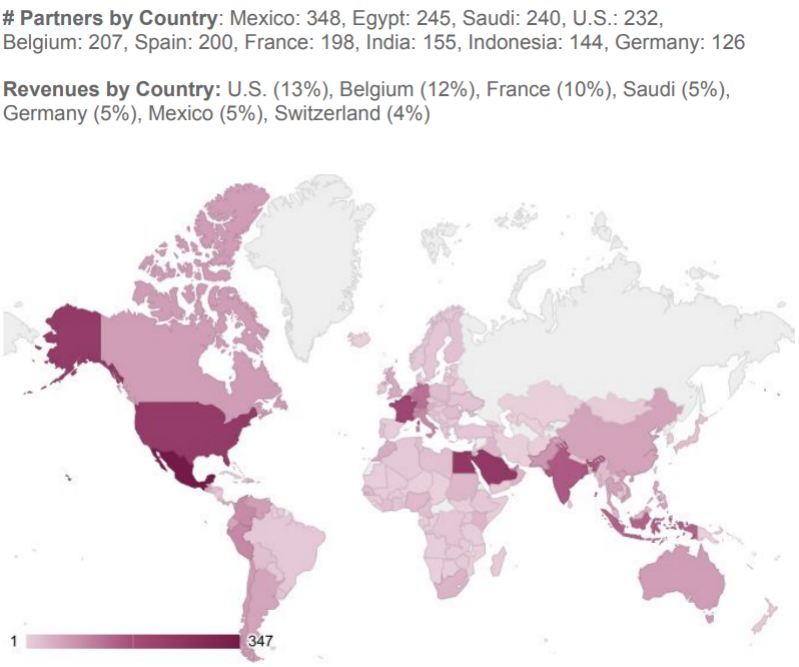

Figure 4.6: Partenaire dans le monde

Les partenaires se divisent en deux grandes catégories, dont la deuxième est subdivisée en trois niveaux distincts. La première catégorie comprend les partenaires dits "Learning". Ces partenaires ont pour mission de former les entreprises

souhaitant acquérir une connaissance complète de l'implémentation d'Odoo pour pouvoir ensuite l'utiliser de manière autonome.

Ensuite, la deuxième catégorie comprend les partenaires dits "Official". Leur principale mission est de former des entreprises qui seront chargées de la vente du produit. Comme mentionné précédemment, cette catégorie est subdivisée en 3 niveaux distincts : "Ready", "Silver" et "Gold". Chaque niveau offre des avantages qui augmentent en fonction de la catégorie, Gold étant la plus élevée. Voici ci-dessous certains des différents avantages en fonction des catégories :

- L'accès aux référentiels Odoo Enterprise sur GitHub.
- La possibilité de signaler les bugs à résoudre par Odoo au nom de leur client.
- Visibilité et reconnaissance en étant inscrit comme partenaire officiel sur la page Partenaires Odoo.
- Jusqu'à 20 % de commission sur les ventes d'Odoo Enterprise, en fonction du niveau du partenaire.
- Un taux de commission de 50 % sur la Plateforme Odoo SH.[24]
- Formations annuelles de mise à niveau après la sortie de nouvelles versions.
- L'accès au Portail des partenaires.
- Séance de formation à la vente d'Odoo.

Il est à noter que les critères pour atteindre le niveau "Gold" sont plus élevés que pour les niveaux inférieurs et qu'il y a également des exigences en termes de ventes annuelles pour conserver son statut de partenaire. Voici sur la figure suivante les différents critères: [23]

Niveaux	Learning	Ready	Silver	Gold
Critères				
Nouveaux utilisateurs Odoo Enterprise / an	-	10 utilisateurs	75 utilisateurs	200 utilisateurs
Des ressources internes actives certifiées	-	1	2	3
Taux de rétention minimum	-	-	70 %	80 %

Figure 4.7: Catégorie partenaire

Comme on le voit sur l'image les partenaires "Learning" n'ont pas de critères à respecter car ils ont pour rôle de former les entreprises à l'implémentation d'Odoo.

Nous l'avons expliqué précédemment, une partie du produit d'Odoo est gratuit, les clients peuvent donc profité d'une partie des modules mais ils devront l'utilisé localement ou avoir un serveur pour hebergé leurs base de donneés. Sinon les clients ont trois catégories de base de donneés possible (payante). Bien évidemment une entreprise peut changer de catégorie.[25]

- Online (SaaS): Cette catégorie est appelée "Software as a Service". Elle permet aux clients d'Odoo d'avoir un hébergement sur les serveurs de l'entreprise. Ainsi, leur site est accessible partout dans le monde et ils bénéficient des mises à jour et du support en cas de problème. De plus, les clients ont des sauvegardes automatiques qu'ils peuvent télécharger pour effectuer des backups.

- On-Premises : Cette option est proposée à un public ayant des connaissances informatiques plus avancées. Elle correspond à un hébergement sur les serveurs du client et non sur ceux d'Odoo. Elle permet aux clients d'être plus autonomes et d'avoir beaucoup plus de liberté sur leurs produits (installation de modules tiers ou création de leurs propres modules).

- Odoo.sh (PaaS): Cette catégorie est un "Platform as a Service". Elle permet d'avoir les avantages des deux autres catégories, sans leurs inconvénients (cette option est plus coûteuse que les autres).

Grâce aux schémas ci-dessous représentant les revenus acquis en fonction du type de catégorie, on peut constater que ce qui rapporte le plus à Odoo est le type SaaS d'Odoo Online, avec 123 millions d'euros générés.

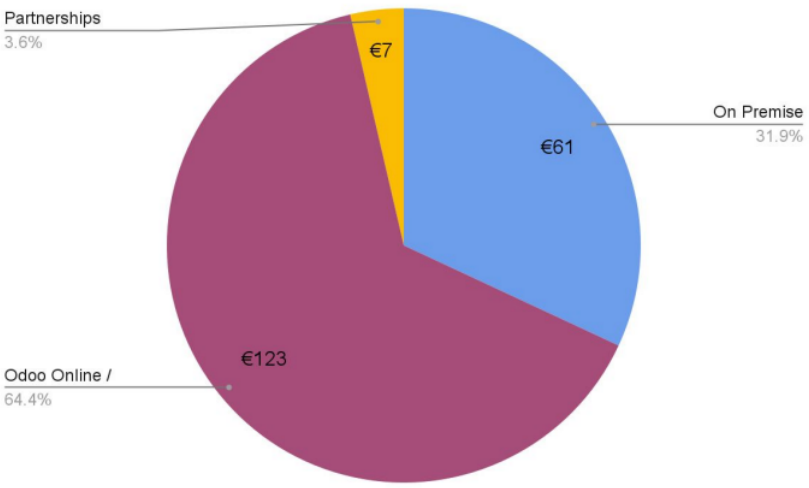

Figure 4.8: Revenue par type d'héberement

4.1.3 Organisation

Dans ce chapitre, nous allons parler de l'organisation interne de l'entreprise. En l'analysant, on se rend rapidement compte qu'Odoo possède une hiérarchie très plate. Cela signifie que l'arbre représentant la structure de l'entreprise a une faible hauteur. À la racine, on retrouve le CEO, Fabien Pinckaers, un ancien étudiant de l'UCL. Le comité de direction, quant à lui, est composé d'un petit groupe de personnes (moins de 5 personnes), chacune étant responsable d'un ou plusieurs départements. Dans le cas du département technique/recherche et développement, on retrouve Antony Lesuisse (CTO), tandis que pour les autres départements, on retrouve par exemple une CCO et un CFO. [11]

Pour mieux expliquer l'organisation interne, nous allons passer en revue les différents département, actuellement Odoo est composé de 6 departements distinct:

- Recherche et développement: Il est chargé, comme son nom l'indique, de l'amélioration du produit Odoo, mais aussi de le maintenir et de résoudre les problèmes potentiels. Ce département est constamment en évolution et est actuellement composé de plus de 300 personnes réparties en plus de 20 équipes qui gèrent les différentes applications proposées par Odoo. En plus des équipes de développement, il y a également des équipes qui gèrent l'infrastructure, l'architecture, la plateforme cloud, ainsi qu'une équipe qui veille au bon déroulement des mises à niveau lors du passage à une nouvelle

version.

Une autre équipe fait également partie de ce département, il s'agit des "Product Owners" (PO). Ce ne sont pas forcément des développeurs ou des personnes ayant des compétences techniques. Chaque équipe possède un ou plusieurs PO, qui sont chargés de créer les tâches (nous expliquerons plus tard l'organisation des tâches) pour les équipes de développement. Ces tâches peuvent avoir pour but de créer une nouvelle fonctionnalité pour une application en particulier, corriger des problèmes potentiels ou apporter des modifications sur une application existante. Les PO ont généralement une excellente connaissance du produit et sont ceux que les développeurs contactent en cas de problème fonctionnel. Ils sont également responsables de réaliser des tests fonctionnels sur leurs tâches.

- Service (PS) : Ce département est également très important car il permet de s'assurer que les clients d'Odoo aient la meilleure expérience possible et donc de les fidéliser au maximum afin qu'ils puissent faire une publicité positive. Leur principal objectif est de fournir de l'aide aux clients, que ce soit lors de la création de leur base de données ou pour les clients déjà existants. Ils sont également chargés d'aider les clients en cas de problèmes lors de la migration vers une nouvelle version. Le PS travaille également en étroite collaboration avec l'équipe de développement pour s'assurer que les problèmes des clients sont résolus rapidement et efficacement.

- Ventes: Comme expliqué précédemment, le département des ventes est divisé en deux sections distinctes : les ventes directes et indirectes. Puisque ces sections ont déjà été expliquées, nous n'allons pas revenir dessus.

- Marketing : Le département marketing est chargé de promouvoir l'entreprise à travers des événements, des contenus vidéo ou textuels sur les différents réseaux sociaux, ainsi que de trouver des moyens innovants pour accroître la popularité d'Odoo auprès du grand public. Leur principal objectif est donc d'attirer de nouveaux clients.

- Finance et comptabilité: Le module comptable est l'un des plus importants proposés par Odoo, et le département de finance et de comptabilité est donc chargé de tester et de valider son bon fonctionnement. Ils travaillent en étroite collaboration avec l'équipe en charge du module comptable afin de détecter et de résoudre rapidement tout problème éventuel. Les comptables ont pour rôle de gérer la comptabilité de l'entreprise, de valider les ventes et les achats effectués par Odoo. Un autre rôle important dans ce département est la gestion financière, qui est dirigée par le CFO et qui permet d'analyser les finances de l'entreprise.

- Ressources humaines et de l'administration: Il est chargé de la gestion des employés. Étant donné que la croissance d'Odoo est énorme et que le nombre d'employés ne cesse d'augmenter, ce département veille à ce que tout se passe bien pour eux en répondant à toutes les questions que les employés pourraient avoir. Ils sont responsables de la gestion des salaires, des voitures de société, des cartes d'essence, etc.

  Ils sont également en charge du recrutement pour tous les départements d'Odoo. Pour donner quelques chiffres sur le recrutement, chaque recruteur a une moyenne annuelle de 75 nouvelles recrues.

Durant ce mémoire, nous avons principalement passé notre temps dans le département de recherche et développement. Chaque équipe est dirigée par un responsable d'équipe, appelé "guru". Selon la personnalité de celui-ci, il peut avoir un rôle de manager ou plutôt de représentant de l'équipe. Odoo offre une grande liberté à ses employés, avec des horaires flexibles et en encourageant l'autonomie. Cette approche de type start-up est assez rare pour une entreprise de cette envergure.

Les différent departement sont présent mondialement. Si on se concentre premièrement sur la Belgique, les différents déparements sont divisées dans les bureaux de Louvain-la-Neuve ou alors dans les 2 (bientôt 3) fermes aménagées de Grand ronsière. Mais les départements ne sont pas qu'en Belgique. Comme on peut le voit sur la carte ci dessous, Odoo ouvre des centres partout dans le monde. Leurs buts est de couvrir un maximum de "time zone" possible pour être toujours disponible aux clients peu importe l'heure.

Figure 4.9: Différents bureaux dans le monde

Les développeurs sont principalement présents en Belgique et en Inde. Cependant, le bureau indien est pour l'instant surtout utilisé pour les tests, mais cela va

bientôt changer. Fabien a comme projet de déménager cette année en Inde pour faire évoluer ce bureau en raison de la part de marché immense que représente le pays. Quant aux autres départements, ils sont présents partout dans le monde.

4.1.4 Technique

Dans ce chapitre, nous allons entrer dans les détails techniques et explorer les différents langages utilisés, ainsi que les outils que les employés du département de recherche et développement utilisent chaque jour. [26] [27]

Odoo utilise une architecture suivant le modèle "Modèle-Vue-Contrôleur" (MVC). Ainsi, plusieurs langages de programmation sont utilisés avec des objectifs bien différents. Python a été choisi pour la partie "back-end" des différents modules, tandis que JavaScript est utilisé pour la partie "front-end" avec l'utilisation d'un framework conçu par Odoo, nommé "Owl" (nous l'expliquerons plus précisément par la suite). Pour la conception des vues, Odoo a décidé d'utiliser le langage de balisage XML. En ce qui concerne la rétention des données, le choix s'est porté sur PostgreSQL.

Pour résumer le modèle MVC utilisé, les vues sont générées grâce au XML et au JavaScript. Le contrôleur, quant à lui, est géré grâce à Python. Ensuite, pour la récupération des informations, elles sont stockées dans une base de données PostgreSQL. Il existe deux manières de récupérer ces informations. Les développeurs peuvent utiliser les fonctions créées par l'Object-Relational Mapping (ORM) (des fonctions Python qui permettent d'accéder facilement aux données) ou utiliser directement le langage SQL. Ce deuxième cas est assez rare mais peut être utile pour des questions de performances, par exemple.

Il est important de mentionner un système de "template" créé par Odoo pour faciliter la vie des développeurs. Ce système s'appelle "Qweb". Il permet de créer des pages web dynamiques, il est principalement basé sur le language XML et permet l'utilisation de balises spécifiques. Ce qui est très intéressant avec Qweb, c'est la possibilité d'afficher des données stockées en base de données de manière très simple mais aussi l'utilisation des fonctions Python pour ajouter de la logique au niveau des templates. Qweb est donc un outil très puissant puisqu'il permet de créer et personnaliser facilement des vues tout en étant intuitif et flexible.

En ce qui concerne les différents outils utilisés par les développeurs, le premier à mentionner est bien évidemment Odoo. Dans la vie de tous les jours, les équipes de développement utilisent l'application "Projet" d'Odoo. Elle permet une gestion

de projet simple avec un système de cartes à déplacer sur un ensemble de colonnes. Elle offre une vue d'ensemble complète de l'état d'avancement des tâches. Un flux classique de tâches serait le suivant :

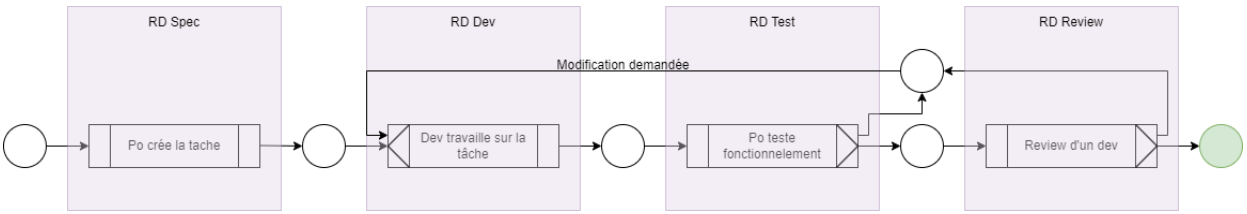

Figure 4.10: Flux classique d'une tâche

Le flux classique comporte donc 4 grandes étapes. La première est celle où le PO crée la tâche. Une fois prête, il peut alors la valider, ce qui permet au développeur de savoir que la tâche est prête. Pendant la deuxième étape, le développeur va travailler sur la tâche et une fois la tâche terminée, il peut l'approuver, ce qui permet au PO de savoir que la tâche est prête pour être testée fonctionnellement. À ce moment-là, le PO peut demander des modifications au développeur. Si tout est bon, alors il peut approuver la tâche encore une fois, ce qui permet de passer à la dernière étape, qui est celle de la revue technique. De nouveau, le développeur qui fait la revue peut demander des modifications, mais si tout est bon, alors la tâche peut être déployée. Bien evidement à chaque étapes la tâche peut être annulé.

Dans le paragraphe précédent, nous avons mentionné qu'une fois une tâche terminée, celle-ci est déployée. Mais où est-elle déployée ? Toute la base de code d'Odoo est hébergée sur une plateforme bien connue appelée "GitHub". [28] Sur cette plateforme, Odoo possède deux principaux répertoires : odoo/odoo et odoo/enterprise. Le premier est le répertoire pour la version open source du logiciel, donc la version gratuite, tandis que l'autre est la version payante accessible aux développeurs ou aux partenaires. La plateforme GitHub fonctionne avec un système de branches. Cela permet à Odoo d'avoir des branches isolées pour chaque version du logiciel. On y retrouve les versions "stables" ainsi que la version dite "master". Les versions ou branches dites stables sont des versions déjà déployées et qui sont utilisés par les clients. En revanche, la branche "master" est une version en développement, c'est-à-dire la prochaine version qui sortira en novembre, la version 17.0. Bien évidemment, pour s'assurer de ne pas mettre du code en développement (que ce soit en master ou en stable), il n'est pas possible de déployer directement du code sur les répertoires "odoo/odoo" et "odoo/enterprise". Les développeurs passent par des répertoires externes tels que "odoo-dev/odoo" et "odoo-dev/enterprise".

De plus, Odoo possède des répertoires additionnels comme le répertoire "Upgrade" qui est composé de scripts de migration. Ce répertoire permet de s'assurer que les clients qui passent d'une version ancienne à une version plus récente n'aient pas de problèmes de compatibilité.

Il convient de préciser qu'une modification effectuée sur une branche stable est automatiquement transmise à la version "master" grâce à un bot (à moins d'indiquer explicitement au bot de s'arrêter à une certaine version). À chaque fois qu'une nouvelle version est disponible, une nouvelle branche est créée automatiquement et des tests sont relancés pour minimiser les conflits.

Pour éviter les conflits, Odoo a également créé une plateforme appelée "runbot" qui est accessible à tous. Elle permet de lancer des tests pour vérifier que chaque branche fonctionne correctement. C'est donc un outil très pratique pour prévenir tout problème de compatibilité entre les différents modules ainsi que pour s'assurer de leur bon fonctionnement. Cette plateforme s'appelle un "serveur d'intégration continue".

De plus, une fois qu'une branche est prête à être ajoutée au logiciel (ce processus s'appelle le "merge"), un "merge bot" (également créé par Odoo) s'occupe de vérifier qu'aucun conflit ne puisse survenir lors de la fusion de la branche avec le logiciel, afin d'éviter tout type de problème.

4.2 Storybook

Storybook est un outil open source s'éxecutant en dehors d'une application et qui permet de développer, tester et documenter des composants d'interface utilisateur (UI) de manière isolée. Cela signifie que l'on peut créer des composants UI sans se soucier des interactions avec le reste de l'application, ce qui peut être d'une grande utilité lors du développement de composants complexes ou lors de tests de composants de façon individuel. [29]

Storybook fournit une plateforme moderne pour visualiser les composants dans différents états et configurations, ce qui peut aider à identifier les problèmes de design et de fonctionnement avant de les intégrer à l'application principale. Il propose également des fonctionnalités telles que la documentation automatisée, la personnalisation de l'interface utilisateur et la prise en charge de plusieurs frameworks de développement, ce qui peut grandement faciliter le travail des développeurs et des designers.

De plus, la plateforme permet la collaboration entre les différents membres d'une équipe en leur fournissant une plateforme commune pour concevoir et valider les composants UI.

4.3 Owl

Notre mémoire est implémenté grâce à une librairie JavaScript créée par Odoo appelée « Owl » (Odoo Web Library). Pour être plus précis, il s'agit d'une librairie d'interface utilisateur écrite en TypeScript, un langage qui ajoute du typage statique à JavaScript. Cette librairie permet donc d'écrire des composants représentant des parties d'interface utilisateur. [30] Owl tire son inspiration de React et Vue, et possède notamment ces fonctionnalités clés :

- Système de composants déclaratifs : Owl utilise une approche déclarative pour la construction d'interfaces utilisateur, similaire à celle de React et Vue. Les développeurs peuvent définir des composants d'interface en utilisant une syntaxe concise et intuitive, décrivant l'apparence de l'interface en fonction de l'état actuel de l'application. Chaque composant d'interface est représenté par une classe en JavaScript représentant sa logique, ainsi que par un template écrit en XML qui représentant la présentation visuelle du composant.

  Cette approche facilite la compréhension et la maintenance du code, car l'accent est mis sur la description de l'interface souhaitée plutôt que sur la manipulation directe du DOM.

- Système de réactivité : Le système de réactivité d'Owl, similaire à celui de Vue, permet aux développeurs d'établir des relations entre les données et les éléments de l'interface utilisateur. Lorsque les données sous-jacentes changent, Owl met automatiquement à jour les composants d'interface utilisateur concernés, garantissant ainsi la cohérence entre les données et l'interface utilisateur. Ce comportement réactif simplifie la gestion des états d'interface utilisateur complexes et réduit la nécessité de mises à jour manuelles de l'interface utilisateur.

- Hooks : Owl introduit des hooks, inspirés de ceux de React. Les hooks permettent d'ajouter des comportements et de la logique personnalisés aux composants, sans avoir besoin de créer des composants basés sur des classes ou des méthodes de cycle de vie complexes. Les développeurs peuvent utiliser les hooks pour gérer l'état, effectuer des effets secondaires et réutiliser la logique à travers plusieurs composants. Cette approche favorise la réutilisabilité du code, améliore l'organisation et la modularité de la base de code.

- Fragments : Owl prend en charge l'utilisation de fragments, similaires à ceux de React, qui permettent de regrouper plusieurs éléments d'interface sans avoir besoin d'un élément parent supplémentaire. Les fragments aident à maintenir la structure HTML propre et concise, notamment lors du rendu de listes ou d'éléments d'interface conditionnels.

- Rendu asynchrone : Owl intègre des capacités de rendu asynchrone, permettant des interfaces utilisateur efficaces. Tout comme l'algorithme de différenciation du virtual DOM de React, Owl optimise le rendu en ne mettant à jour que les parties nécessaires de l'interface utilisateur lorsque les données changent. Cette approche réduit l'impact sur les performances et offre une expérience utilisateur fluide, même lors de l'exécution d'opérations lourdes sur le plan computationnel ou de requêtes réseau.

4.4 Projet Open Source

Un projet open source est un projet de logiciel dont le code source est disponible gratuitement et peut être modifié et distribué par n'importe qui. Ces projets sont généralement développés par une communauté de développeurs, qui collaborent pour produire et/ou améliorer le logiciel. [31]

Les projets open source sont devenu extrement populaire ces dernières années mais ce n'est pas un terme si récent. En effet, l'open source a vu le jour dans les années 1980 avec Richard Stallman, du MIT, qui a inventé le terme "logiciel libre". [32]

L'objectif principal d'un projet open source est de permettre à la communauté de partager les connaissances et les ressources pour développer des logiciels de qualité. ils sont souvent considérés comme une alternative plus coopérative et moins concurrentielle aux méthodes traditionnelles de développement, mais ils peuvent présenter des défis en matière de qualité du code produit ou de fiabilité, car n'importe qui peut proposer une modification. Ces problèmes sont généralement gérés grâce à un système de revue par des développeurs plus expérimentés.

Le développement open source présente de nombreux avantages pour les utilisateurs et les développeurs. L'un des avantages clés est la transparence, étant donné que les projets open source sont accessibles à tout le monde et que les utilisateurs avertis peuvent comprendre comment le logiciel fonctionne. De plus, ces utilisateurs peuvent proposer des corrections de problèmes ou des améliorations pour le logiciel. Enfin, les logiciels open source disposent généralement de nombreuses ressources en ligne telles que des forums de discussion, de la documentation, etc. [33]

4.5 GitHub

GitHub est une plateforme qui permet l'hebergement de code. Il permet aux développeurs de stocker, partager et collaborer sur des projets de développement. GitHub propose de nombreuses fonctionnalités pour aider les développeurs à gérer leur code, y compris la gestion des bugs, les demandes de fonctionnalités et les commentaires du code. Un projet GitHub est sous la forme de dossier appelé "Repository". [28]

Le fonctionnement de GitHub repose sur les concepts de "branches" et de "pull requests". Une branche est une version spécifique d'un projet qui peut être mise à jour indépendamment de la version principale. Les développeurs peuvent créer leurs propres branches pour travailler sur des fonctionnalités spécifiques, puis soumettre leurs modifications en utilisant une demande de fusion (ou "pull request"). Les demandes de fusion permettent aux développeurs de discuter et de réviser les modifications proposées, puis de les fusionner avec la version principale du projet.

GitHub propose également des fonctionnalités pour suivre les activités de développement et des notifications pour recevoir des mises à jour sur les projets qui vous interesse. Les développeurs peuvent également utiliser GitHub pour trouver et collaborer sur des projets open source.

Avec actuellement plus de 94 millions d'utilisateur, 413 millions de contribution en 2022, Github est de loins la plus grande plateforme d'herbergement pour du code. [34]

4.6 Trello

Trello est un outil de gestion de projets basé sur des colonnes et des cartes que l'on peut déplacer entre ces différentes colonnes. Il permet aux équipes de suivre les tâches à accomplir ainsi que leurs délais. Les utilisateurs peuvent créer des colonnes pour chaque projet et y ajouter des listes de tâches appelées "cartes". Pour refléter l'avancement du projet, ces cartes peuvent être déplacées entre les colonnes. Elles peuvent également contenir des commentaires, des pièces jointes et des activités pour aider à organiser et à prioriser les tâches. Trello est souvent utilisé pour la planification et la collaboration dans les entreprises ou les projets, tels que le développement de logiciels. [35]

Spécifications

Dans cette section, nous allons aborder plus en détail le sujet de notre travail maintenant que le contexte a été bien expliqué. Nous allons commencer par expliquer le cahier des charges qui nous a été donné par Odoo. Ensuite, nous discuterons des ressources humaines ainsi que des outils dont nous avons disposé pour assurer le bon déroulement de ce mémoire.

5.1 Cahier des charges

L'objectif fixé par Odoo était de développer une plateforme permettant de visualiser et d'interagir avec une grande variété de composants de l'interface d'Odoo, dans différents scénarios. Cette plateforme permettra aux développeurs de :

- Visualiser les différents composants existants dans divers scénarios et de tester facilement l'impact visuel/fonctionnel d'un changement de code ;
- Accéder à une documentation détaillée et de générer facilement un morceau de template pouvant être copié/collé.

Ensuite le cahier des charges suivant nous a été donné:

- Réaliser un addon odoo de test (qui ne sera pas installé en production)
- Cet addon doit contenir l'application, soit sous forme d'application standard odoo, soit sous la forme d'un controleur
- Les composants pourraient être organisés en 3 catégories: "vues", "champs" et "composants génériques"
- Chaque composant peut contenir plusieurs "stories" (scénario d'intéraction).
- Pour les vues, le système devrait être capable de générer des jeux de données à la volée, sans accéder à la base de donnée.

- Le système doit être extensible: tout addon odoo devrait pouvoir définir des stories indépendamment, qui seront collectées par le storybook.
- Il faudra définir l'interface exacte que les stories devront implémenter.
- Chaque composant doit pouvoir être associé à une page de documentation officielle, et ce, indépendamment des stories.
- Cette application devrait être faite dans le framework javascript de Odoo.
- De par la nature du framework odoo, les composants Owl définissent souvent une description statique de leur props. Celles-ci pourraient être récupérée par storybook pour générer un onglet "props".
- Si possible, ces props devraient pouvoir être éditable, afin de visualiser des changements dans l'interface.

Avec ce cahier des charges, nous avions toutes les informations nécessaires pour commencer la création de cette plateforme. Dans les sections suivantes, nous aborderons la méthodologie de travail que nous avons suivie, les choix d'implémentation que nous avons faits, ainsi que d'autres aspects importants de notre mémoire.

5.2 Ressources mise à disposition

Au cours de ce mémoire, nous avons été mis en contact avec plusieurs développeurs d'Odoo. Ces développeurs étaient disponibles pour répondre à toutes les questions que nous avions. Étant donné que ce mémoire était centré sur l'équipe Javascript, les développeurs disponibles pour nous aider venaient de cette même équipe. Cependant, d'autres développeurs et product owners nous ont aidés lors de la phase de tests fonctionnels, que nous expliquerons plus tard.

En ce qui concerne les ressources mise à dispositions, nous avons été invité sur les différents "repository" GitHub, ainsi qu'un compte pour avoir accès à l'application projet, nous avons également été permis de venir dans les bureaux pour travailler sur le mémoire.

Méthodologie

6.1 Organisation

Dans cette section, nous allons aborder l'organisation que nous avons décidé d'adopter pour pouvoir travailler ensemble et en parallèle de manière rapide et efficace. Pour cela, nous avons utilisé plusieurs logiciels de gestion de projet, dont le premier est Trello. Trello nous a permis d'avoir une vue d'ensemble des tâches à faire tout au long du projet, de suivre constamment les tâches en cours et l'avancement de celles-ci.

Ensuite, nous avons utilisé Github pour travailler de manière collaborative sur notre mémoire (lien: https://github.com/fdardenne/owlybook).

Nous avons donc travaillé avec le système de branches. Chaque nouvelle tâche sur le projet nécessitait la création d'une branche avec un nom spécifique pour être facilement reconnaissable et permettre de la retrouver facilement par la suite. Ensuite, dès que le développement d'une tâche était terminé, une "Pull request" était créée et devait être revue par l'autre membre de l'équipe. Cela nous a permis d'être au courant du travail de l'autre à tout moment, mais aussi d'argumenter sur les choix d'implémentation pour avoir une certaine cohérence tout au long du projet. De plus, chaque "pull request" devait être parfaitement décrite grâce au message de "commit" (paragraphe d'explication sur la "pull request"), celui-ci devait être complet et précis.

En ce qui concerne le travail en lui-même, nous avons décidé assez tôt de nous donner au moins un jour par semaine pour faire du développement sur le projet. Cela nous a permis de prendre l'habitude de travailler sur la plateforme pour avoir un avancement constant. Grâce aux systèmes de tâches, nous pouvions travailler chacun de notre côté, mais dès que l'un de nous avait une question, l'autre était disponible, que ce soit en travaillant en présentiel ou à distance, en s'appelant sur le logiciel Discord.

Développement

7.1 Explorations des solutions existantes

Avant de commencer le développement de la plateforme, un choix s'est offert à nous. Essayer d'étendre Storybook au framework d'Odoo "Owl" ou partir de zéro et créer notre propre plateforme.

Nous avons commencé par étudier la première option.

7.2 Extension de Storybook pour Owl

Cette option nous a conduit à plusieurs semaines de travail. Cependant, nous avons rapidement rencontré des problèmes qui nous ont amenés à reconsidérer cette option. L'un des principaux problèmes que nous avons rencontrés était le manque de documentation dans Storybook appropriée à notre cas. Nous avons constaté que les ressources disponibles ne couvraient pas suffisamment celui ci, ce qui rendait difficile la mise en œuvre de l'intégration d'Owl de manière optimale.

De plus, nous avons constaté que l'intégration du service proposé par Odoo n'était pas bien gérée par Storybook, qui était davantage optimisé pour les frameworks qu'il supporte nativement. Cela a entraîné des limitations et des problèmes lors de l'utilisation d'Owl avec Storybook, ce qui a eu un impact sur notre capacité à étendre efficacement le framework d'Odoo.

Malheureusement, nos tentatives de contact avec l'équipe de Storybook pour obtenir de l'aide et des orientations supplémentaires n'ont pas vraiment abouti que ce soit par leurs serveur Discord ou par mail. Malgré nos efforts pour établir une communication et obtenir des pistes concrètes pour résoudre nos problèmes, nous n'avons pas reçu les réponses attendues.

Face à ces défis et à l'absence de solutions pratiques, nous avons finalement

pris la décision de ne pas poursuivre l'option d'intégrer Owl à la plateforme Storybook existante. Nous avons plutôt opté pour le développement d'une plateforme indépendante.

La création de zéro comporte des avantages. En effet, elle nous a donné un contrôle total sur l'expérience utilisateur, l'intégration du service Odoo et la gestion des spécificités du framework Owl. Elle comporte également des désavantages, du fait que nous bénéficions pas d'une plateforme open-source stable entretenue par des milliers de collaborateurs.

Cette décision s'est avérée être la voie la plus appropriée pour répondre à nos besoins spécifiques et nous a permis de créer une plateforme sur mesure, entièrement adaptée à Owl et à nos objectifs de développement.

7.3 Création de la plateforme

Nous avons donc entrepris de créer notre propre plateforme à partir de zéro. Pour ce faire, nous avons commencé par créer un nouveau module dans Odoo.

Comme expliqué précédemment dans ce mémoire, les modules sont des composants autonomes qui regroupent des fonctionnalités spécifiques et peuvent être facilement ajoutés ou supprimés du système. Ils offrent une approche modulaire et flexible pour le développement d'applications. De plus, Odoo propose une documentation complète et des conventions de développement claires pour la création de nouveaux modules, ce qui nous a grandement facilité la tâche dans la mise en place de notre environnement de développement.

Le nouveau module regroupe tous les fichiers et fonctionnalités spécifiques à notre plateforme. Nous avons suivi les conventions de nommage d'Odoo pour l'organisation des fichiers ainsi que leurs noms. Cette approche garantit que tout développeur familiarisé avec Odoo pourra comprendre facilement notre plateforme et éventuellement proposer des améliorations grâce à des "pull request".

Pour lancer le serveur avec notre module sur notre machine, il a suffit d'ajouter le chemin vers notre dossier dans la variable d'environnement des "addons path". Cela permet au serveur de reconnaître et de charger correctement notre module, rendant ainsi notre plateforme disponible pour une utilisation.

En créant notre propre module distinct, nous avons pu travailler de manière isolée et développer notre plateforme sans perturber les autres fonctionnalités d'Odoo. Cela nous a offert un contrôle total sur notre code et nous a permis de personnaliser notre plateforme selon nos besoins spécifiques. Cependant, il est

important de noter que notre dépendance à la version "master" d'Odoo tout au long du développement nous rendait à la fois vulnérables et ouverts aux changements et mises à jour effectués par l'équipe Odoo.

D'un côté, cela présentait un avantage en restant à jour avec la version "master" d'Odoo, nous avions accès aux dernières fonctionnalités et améliorations apportées par l'équipe Odoo, ce qui pouvait bénéficier à notre plateforme.

Cependant, cette dépendance à la version "master" d'Odoo pouvait également être un inconvénient. Étant donné que nous utilisions des fonctionnalités et des composants d'Odoo dans nos stories, il y avait une possibilité que les mises à jour effectuées par l'équipe Odoo puissent impacter nos stories du jour au lendemain. Cela signifiait que nous devions rester vigilants et prêts à adapter notre code en cas de changements inattendus. Cependant, cela faisait partie des défis auxquels nous étions prêts à faire face dans notre processus de développement.

Dans l'ensemble, la décision de créer notre propre module distinct nous a offert une flexibilité et une personnalisation maximales pour notre plateforme, tout en étant conscients des avantages et des défis liés à nos choix.

7.4 Framework JavaScript d'Odoo

Afin de développer notre plateforme pour Odoo, il est nécessaire de comprendre le framework JavaScript de Odoo.

Le framework JavaScript de Odoo est construit au dessus du framework Owl. Ce framework fournit aux différents modules Odoo les ingrédients leur permettant de créer des interfaces utilisateurs complexes et efficaces.

Parmis ces ingrédients nous pouvons retrouver:

- Les composants génériques: ce sont des composants qui jouent un rôle clé dans le développement d'applications. Ils sont réutilisable, modulaire et permettent aux modules d'avoir une abstraction sur la logique que nécessite de tels composants. Parmis ces composants nous pouvons retrouver par exemple un menu dropdown, un color picker, un pager, un checkbox, ...
- Les services: Chaque composant peut être remplacé à tout moment, à l'exception du composant racine. Cela signifie que l'état interne des composants n'est pas persistant. Cependant, il existe des cas où nous souhaitons conserver des données, telles que les messages de discussion ou le menu actuel.

Pour cela, Odoo propose des services, qui sont des morceaux de code persistants capables d'exporter des états et des fonctions. Les services peuvent être interdépendants, et les composants peuvent importer ces services pour les utiliser. [36]

- Les vues: les vues dans Odoo sont utilisées pour permettre aux utilisateurs de consulter et de modifier leurs données. Chaque vue est décrite par un modèle XML, qui contient les informations nécessaires pour que la vue fonctionne correctement, ainsi que les champs qui représentent les données. Les vues sont des éléments importants et assez complexes, elles seront expliquées plus en détail dans la prochaine section.

- Les champs: les champs sont utilisées dans les vues pour représenter les données. Ces champs sont en réalités des composants owl et sont donc tous très personnalisables.

- Les registres : Les registres conservent une collection de paires clé/valeur accessible a tous les composants. Il y a plusieurs registres qui ont pour chacun une fonction particulière. Par exemple le registre des vues permet de stocker toutes les vues de Odoo, permettant a tous les modules d'étendre ou personnaliser la liste des vues disponibles. Il y a également le registre des composants représentant des champs.

7.4.1 Les vues

La particularité des vues, comme mentionné précédemment, est qu'elles sont décrites via des modèles XML. Nous appelons ces modèles XML des "arch".

Ci-dessous, vous pouvez voir un exemple d'arch décrivant une vue de type "graph". Dans cette vue nous représentons deux fields, les produit et le prix qui seront dans le graph.

```
1  <graph string="Ventes" type="bar" sample="1">
2      <field name="customer"/>
3      <field name="total" type="measure"/>
4  </graph>
```

Dans la Figure 7.1, vous pouvez trouver un exemple concret représentant ce qui est décrit par l'arch.

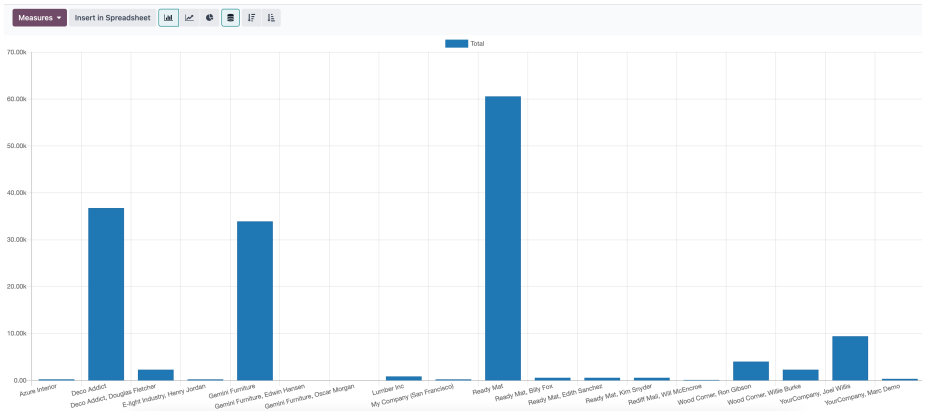

Figure 7.1: vue graph représentant les customer sur l'axe x et le montant facturé sur l'axe y

L'écriture des vues en XML présente de nombreux avantages, notamment la séparation de la présentation des vues et de la logique de construction. Cela facilite la maintenance et la réutilisation du code.

De plus, l'écriture en XML est souvent considérée comme plus accessible, car elle est très déclarative. Contrairement à la programmation traditionnelle, l'écriture en XML ne nécessite pas de compétences avancées en programmation ni de logique complexe. Il suffit de décrire la structure et l'apparence souhaitées des vues, sans avoir à se soucier des détails de mise en œuvre.

Ainsi, même les utilisateurs moins expérimentés en programmation peuvent facilement créer et personnaliser des vues en XML dans Odoo, ce qui rend la création de vue à un large éventail de personnes.

Pour être construites, les vues en XML sont chargées, parsée et compilée par le navigateur. Chaque vue bénéficie de son propre parser XML et de son compilateur car chaque vue peut être décrite de manière différente.

Une fois la vue analysée, les informations sont transmises à un objet appelé "Model". Son rôle est de récupérer toutes les informations du serveur, puis de construire une structure de données que les composants Owl peuvent utiliser pour représenter la vue.

7.4.2 Ecrire une story

Il y a deux types de story dans Owlybook :

- Les stories qui représentent des composants Owl.
- Les stories qui représentent des archs de vues écrites en XML comme vu dans la section précédente.

Ecrire la story d'un composant Pour créer la story d'un composant, il faut créer un nouveau fichier JavaScript représentant la nouvelle story dans le dossier "static" de votre module. Par convention, vous pouvez le nommer component_name.stories.js.

Voici un modèle simple de story pour un composant, dont nous expliquerons les points clés plus tard :

```
/** @odoo-module */

import { MyComponent } from "@my_module/my_component";
import { registry } from "@web/core/registry";
import { Component, onMounted } from "@odoo/owl";

class ParentStory extends Component {
    static template = "myModule.MyComponentStory";
    static components = { MyComponent };
}

ParentStory.codeTemplate = "myModule.MyComponentCall";
ParentStory.storyConfig = {
    title: "This is my beautiful story",
    component: MyComponent,
    props: {
        canToggle: {
            value: true,
            help: "With this props we can defined if the
               component
            can be toggled or not"
        },
        colors: {
            value: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
            readonly: true,
        },
        selectedColor: {
            value: 9,
```

```
        },
    },
};

export const MyComponentStories = {
    title: "Name of the folder title",
    module: "my module",
    stories: [ParentStory],
};

registry.category("stories").add("myModule.
   myComponentStories",
MyComponentStories);
```

- `class MyComponentStory extends Component`
  Nous créons ici un composant parent qui rendra le composant pour lequel nous voulons créer une histoire.

  Ce composant parent recevra deux props:

  - storyProps : il s'agit des props du composant de la story. Initialement, storyProps prend les valeurs définies dans "ParentStory.storyConfig.props". Si le props n'est pas défini ou si aucune valeur par défaut n'est attribuée, il prend la valeur par défaut de la validation du props du composant. Lorsque l'utilisateur modifie les props dans l'interface utilisateur, storyProps est mis à jour pour refléter les choix de l'utilisateur.
  - changeProps(propsName, newValue) : il s'agit d'une fonction de rappel permettant au parent de la story de changer les props du composant de la story pour simuler certains changements.

- `static template = "myModule.MyComponentStory"`
  C'est dans ce modèle XML que la documentation et le rendu des composants seront effectués. Voici le modèle de la story :

```
<?xml version="1.0" encoding="UTF-8"?>
 <templates xml:space="preserve">
     <t t-name="myModule.MyComponentStory" owl="1">
         <h1> My Component </h1>

         <p> Hello world </p>
         <t t-call="myModule.MyComponentCall"/>

     </t>
```

```
10
11      <t t-name="myModule.MyComponentCall" owl="1">
12        <ColorList t-props="props.storyProps"/>
13      </t>
14  </templates>
```

- `ParentStory.codeTemplate = "myModule.MyComponentCall"`
  Cette ligne spécifie un deuxième modèle XML: myModule.MyComponentCall. Ce modèle est affiché dans l'onglet "Template" afin que les utilisateurs puissent voir comment est utiliser le composant.

- `ParentStory.storyConfig =`
  Voici les métadonnées de la story:

  - title : le titre de la story
  - component : le composant de la story, ceci est nécessaire pour qu'Owlybook puisse rechercher la validation des props du composant.
  - props : un objet prenant le nom du prop comme clé afin que nous puissions fournir des informations supplémentaires sur chaque prop si nécessaire.
    * value : la valeur par défaut des props dans la story. Si la clé de valeur n'est pas mentionnée, c'est la valeur par défaut attribuée dans le composant de validation des props qui est prise en compte.
    * help : prend une chaîne comme valeur et fournit un tooltip dans l'interface à côté du nom de la props.
    * readonly: prend un booléen, si vrai, les utilisateurs ne pourront pas modifier les props de manière interactive.

- `export const MyComponentStories =`
  Cette variable permet de définir le dossier des Stories :

  - title : le titre du dossier affiché dans la "sidebar"
  - module : le module des Stories dans lequel le dossier apparaît
  - stories : un tableau de story

- `registry.category("stories").add("myModule.myComponentStories"...`
  Cette ligne est nécessaire pour enregistrer la story dans le registre d'Owlybook, où Owlybook recherche les stories enregistrées.

Voici un exemple de rendu finale:

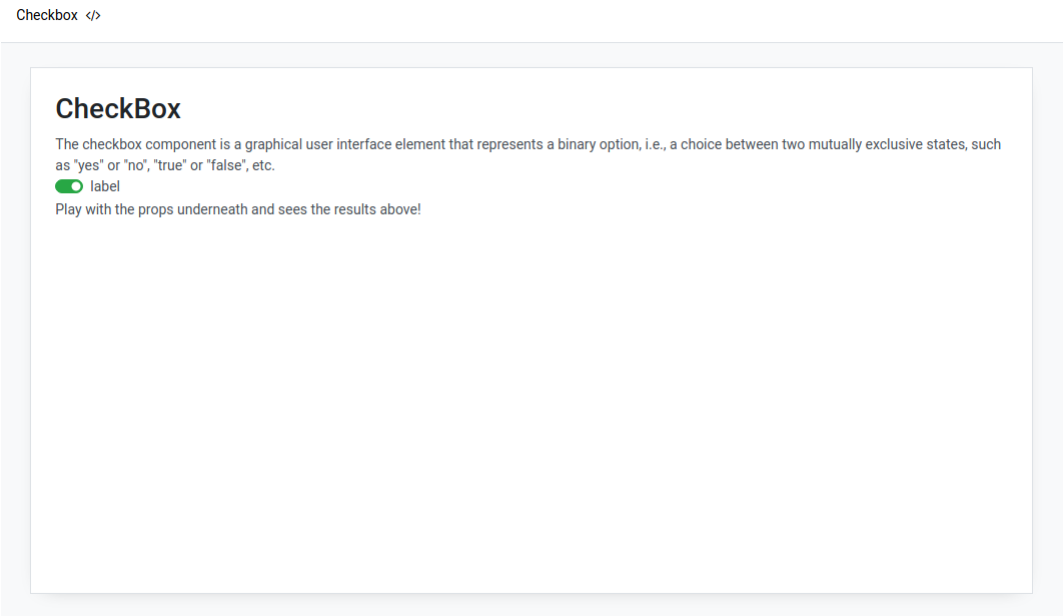

Figure 7.2: Rendu finale de la story "checkbox"

Ecrire la story d'une vue En ce qui concerne les vues, la façon de créer une story est légèrement différente. Tout comme pour la story d'un composant, il faut de créer un nouveau fichier JavaScript représentant la nouvelle story dans le dossier "static" de votre module.

Voici un modèle simple de story :

```
/** @odoo-module */

import { registry } from "@web/core/registry";

const serverData = {
    models: {
        partner: {},
    },
    views: {},
};

const attrs = {
    text: { type: String, value: "ribbon" },
    tooltip: { type: String, optional: true, value: "this is
        a tooltip" },
    bg_color: {
        type: String,
```

```
        optional: true,
        value: "bg-success",
        choices: [
            { label: "bg-success", value: "bg-success" },
            { label: "bg-warning", value: "bg-warning" },
            { label: "bg-danger", value: "bg-danger" },
        ],
    },
};

const formWithRibbon = {
    title: "Ribbon",
    model: "partner",
    viewType: "form",
    attrs,
    arch: `<form>
    <sheet>
        <widget name="web_ribbon" {{attrs}}/>
            <div>
            Lorem ipsum dolor sit amet, consectetur
               adipiscing elit.
            In id tellus et enim sodales lacinia ...
            </div>
    </sheet>
</form>`,
    serverData,
};

export const FormRibbonStories = {
    title: "Widget",
    module: "web",
    stories: [formWithRibbon],
};

registry.category("stories").add("owlybook.formWithRibbon",
   FormRibbonStories);
```

Listing 7.1: https://github.com/fdardenne/owlybook/blob/main/static/src/stories/view_widgets/ribbon_field.stories.js

- `const serverData`
  Cet objet nous permet de simuler des données du serveur pour s'assurer du bon fonctionnement de l'arch. Ces données du serveur peuvent contenir des

informations utilisées dans la story elle-même.

- `const attrs`
  Ce dictionnaire nous permet de fournir des valeurs par défaut aux props de l'arch qui seront affichées dans la story elle-même, ainsi que dans le "bottom panel". Dans cet exemple, nous pouvons voir qu'il est possible de spécifier un nom de base pour le "Ribbon", un tooltip, et les choix disponibles pour la couleur du "Ribbon".

- `const formWithRibbon`
  Ce dictionnaire utilise les données du serveur pour créer l'arch, dans ce cas-ci une vue de formulaire.

- `export const MyComponentStories`
  Cette variable permet de définir le dossier des Stories :

  – title : le titre du dossier affiché dans la "sidebar"

  – module : le module des Stories dans lequel le dossier apparaît

  – stories : un tableau de story

Vous pouvez voir le rendu de la story dans la figure 7.3

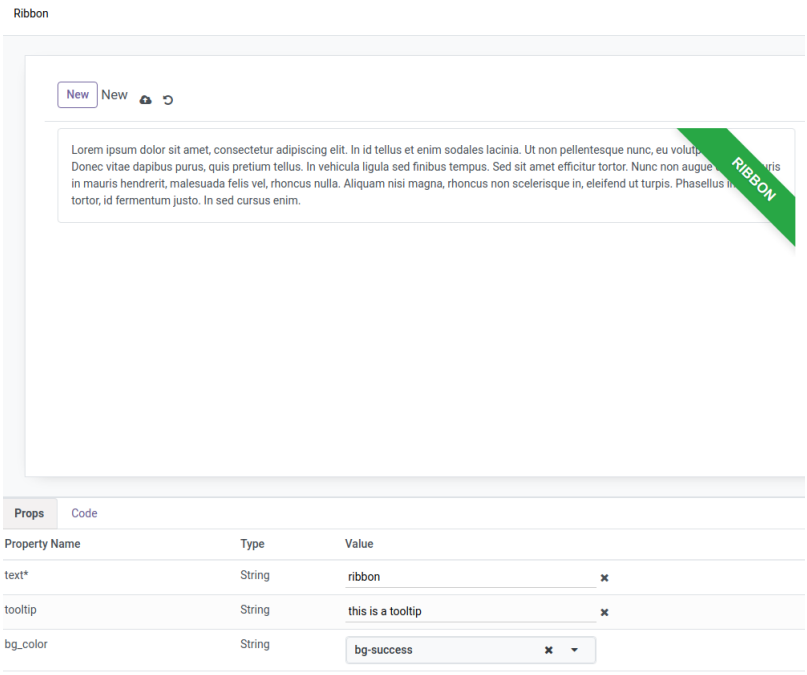

Figure 7.3: Rendu finale de la story "Ribbon"

7.4.3 Lien personnalisé

Nous avons mis en place un système de liens personnalisés pour chaque story. Chaque story possède un lien unique composé de son module, de son dossier et enfin du composant/arch. Ce système de liens permet une meilleure organisation des stories et facilite l'accès aux différentes composantes de l'interface utilisateur. Les utilisateurs peuvent ainsi facilement partager les liens des stories avec leurs collègues et collaborateurs pour des discussions et des revues de code. Grâce à ce système de liens, nous allons aussi pouvoir simplement intégré notre plateforme dans la documentation officiel mais nous y reviendrons plus tard dans la section "Intégration du Owlybook dans la documentation sphinx de Odoo".

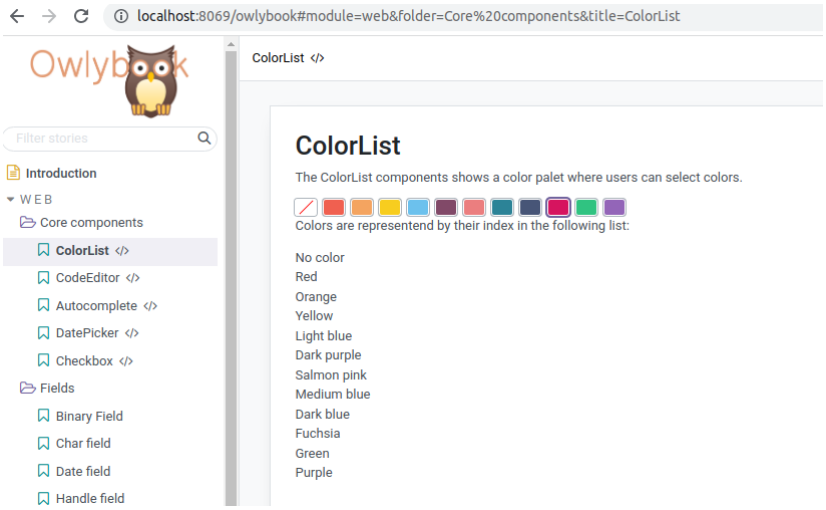

Figure 7.4: Lien personnalisé

7.4.4 Différentes parties d'Owlybook

Notre plateforme se compose de trois grandes sections, chacune ayant un rôle spécifique dans l'expérience utilisateur. Ces sections sont la "Sidebar", le "Canvas" et le "Bottom panel".

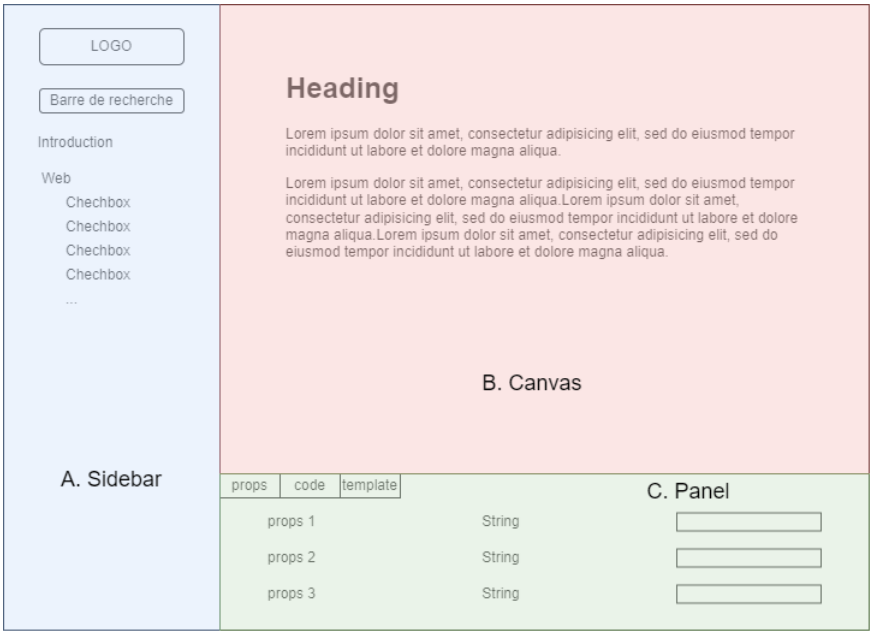

Figure 7.5: Différentes parties d'Owlybook

La "sidebar", située à gauche de l'écran, offre une vue d'ensemble des différentes stories disponibles. C'est là que les utilisateurs peuvent naviguer entre les stories et choisir celle qu'ils souhaitent explorer ou modifier. En plus de la liste des stories, la barre latérale comprend également une barre de recherche permettant aux utilisateurs de trouver rapidement une story spécifique en saisissant le nom d'une story ou d'un dossier.

Le fonctionnement de la barre de recherche est le suivant : étant donné que nous n'utilisons pas de base de données, les informations sont volatiles, ce qui signifie qu'à chaque rafraîchissement de la page, les données sont perdues. Pour pallier cela, nous avons opté pour des listes et des hashmaps pour stocker temporairement nos données. Lorsque l'utilisateur saisit une donnée dans la barre de recherche, une liste de stories correspondant aux critères de recherche est générée, ainsi qu'une liste des dossiers associés. Ensuite, dans le template XML, nous pouvons parcourir les stories et n'afficher que celles présentes dans la liste que nous avons créée. De plus, nous avons ajouté la possibilité d'ouvrir ou de fermer les dossiers selon votre convenance, pour une meilleure organisation et navigation.

Le "Canvas", quant à lui, constitue l'espace principal de notre plateforme. Une fois une story sélectionnée, elle est affichée dans le canvas. C'est ici que les utilisateurs peuvent visualiser et interagir avec le contenu de la story. Le canvas peut

contenir divers éléments tels que des composants ou des archs, qui sont configurés selon les besoins spécifiques de la story.

Enfin, le "Bottom panel" est une zone importante de notre plateforme, car elle propose différents onglets avec des fonctionnalités distinctes et des informations essentielles sur la story sélectionnée. Parmi ces onglets, nous retrouvons :

- Props: permet de modifier les propriétés de la story.
- Events: dans la configuration des storys, on peut explicitement faire que certaines actions soit affichés dans cet onglet. (uniquement présente dans le cas d'un composant)
- Javascript: permet d'affiché le code javascript de la story selectionné. (uniquement présente dans le cas d'un composant)
- Template: permet d'affiché le template de la story selectionné. (uniquement présente dans le cas d'un composant)
- Code: permet de voir le code de cette arch (uniquement présente dans le cas d'une arch).

7.4.5 Architecture du code

Le code de Owlybook est entièrement écrit en javascript. En voici la structure:

```
js
├── bottom_panel
│   ├── arch_properties
│   │   └── arch_properties.js
│   ├── component_code
│   │   ├── component_code.js
│   │   └── component_code.xml
│   ├── component_properties
│   │   └── component_properties.js
│   ├── events
│   │   ├── events.js
│   │   └── events.xml
│   ├── panel.js
│   ├── panel.scss
│   ├── panel.xml
│   └── properties.xml
└── canvas
```

```
    ├── arch_renderer.js
    ├── canvas.js
    ├── canvas.scss
    ├── canvas.xml
    └── component_renderer.js
├── components
    ├── code_editor
    │   ├── code_editor.js
    │   └── code_editor.xml
    ├── object_renderer
    │   ├── object_renderer.js
    │   └── object_renderer.xml
    └── select_menu
        ├── select_menu.js
        ├── select_menu.scss
        └── select_menu.xml
├── main.js
├── mock_qunit.js
├── owlybook_view.js
├── owlybook_view.scss
├── owlybook_view.xml
├── sidebar
    ├── sidebar.js
    ├── sidebar.scss
    └── sidebar.xml
├── stories.js
└── utils.js
```

Chaque section de Owlybook possède son propre dossier dans lequel nous plaçons les composants et les templates qui la composent. Ainsi nous pouvons retrouver les dossier "bottom_panel", "canvas" et "sidebar".
Nous avons également le dossier "components" contenant les composants qui ont une utilisation générique.

Le fichier stories.js est le fichier comprenant une structure de donnée partagée par tous les composants de Owlybook. Cette structure de donnée est représentée par un objet et elle permet de représenter l'état actuel de Owlybook. C'est-à-dire la story, les props, l'arch, le code, et le lien actuel.

Tous les composants de l'interface utilisateur observent donc l'état de cet objet réactif. Toute modification de cet objet entraîne un nouveau rendu des composants

concernés.

Ce re-rendu est provoqué par la réactivité d'Owl. Tout composant peut s'abonner à un objet que l'on appelle un objet réactif. Lorsqu'un composant abonné lit une clé de cet objet, il s'abonne à cette clé spécifique. Toute modification ultérieure de cette clé déclenchera un nouveau rendu du composant.

7.4.6 Props

Dans cet onglet, on peut retrouver la liste de tous les "props" du composant de la story. Chaque ligne de cette liste est composée du nom de la props, du type et, selon le type, de la possibilité de la modifier, ce qui aura un impact direct sur la story sélectionnée.

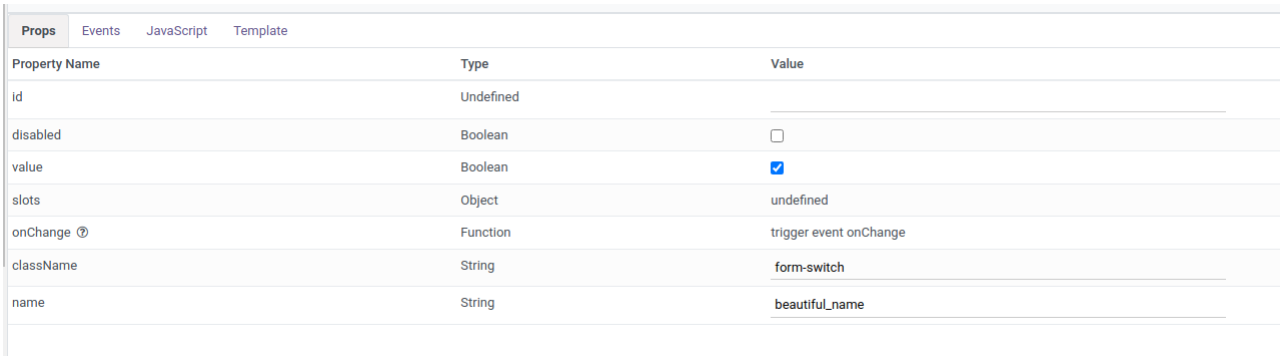

Figure 7.6: Exemple de l'onglet "props"

Dans le cas d'un composant tel que la checkbox, les props sont définis dans le composant lui-même. Par exemple, pour la checkbox, on peut retrouver les props dans le répertoire Odoo, plus précisément dans le chemin suivant : addons/web/static/core/checkbox/checkbox.js. Voici les props pour la checkbox :

```
CheckBox.props = {
    id: {
        type: true,
        optional: true,
    },
    disabled: {
        type: Boolean,
        optional: true,
```

```
    },
    value: {
        type: Boolean,
        optional: true,
    },
    slots: {
        type: Object,
        optional: true,
    },
    onChange: {
        type: Function,
        optional: true,
    },
    className: {
        type: String,
        optional: true,
    },
    name: {
        type: String,
        optional: true,
    },
};
```

Listing 7.2: https://github.com/odoo/odoo/blob/16.0/addons/web/static/src/core/checkbox/checkbox.js

Le template suivant est le template du composant qui permet de représenter les props de chaque composant de manière générique. Ce template affiche ainsi les différents props, leurs type, ainsi que l'input qui permettra la modification du props:

```xml
<templates>
    <t t-name="owlybook.properties" owl="1">
        <div t-if="warning" class="alert alert-warning mb-0"
            role="alert">
            <t t-esc="warning"/>
        </div>
        <div class="table-responsive o_owlybook_bottom_bar">
            <table class="table table-sm table-hover table-
               striped">
                <thead>
                    <tr>
                    <th scope="col">Property Name</th>
                    <th scope="col">Type</th>
                    <th scope="col">Value</th>
                    </tr>
                </thead>
                <tbody>
                    <t t-foreach="properties" t-as="property
                       " t-key="property">
                        <tr>
                            <td>
                                <t t-call="owlybook.
                                   properties.name"/>
                            </td>
                            <td t-esc="propertyType(
                               property_value)"/>
                            <td>
                                <t t-call="owlybook.
                                   properties.input"/>
                            </td>
                        </tr>
                    </t>
                </tbody>
            </table>
        </div>
    </t>
</templates>
```

Listing 7.3: https://github.com/fdardenne/owlybook/blob/main/static/src/js/bottom_panel/properties.xml

On peut observer que cet onglet est divisé en trois grandes parties. La première partie concerne le nom de la props, où nous appelons (avec t-call) ce template:

```
<t t-name="owlybook.properties.name" owl="1">
    <t t-esc="property"/>
    <t t-if="!property_value.optional"><span>*</span></t>
    <i class="fa fa-question-circle-o ps-2 owlybook_tooltip"
        t-if="property_value.help"
    t-att-data-tooltip="property_value.help"/>
</t>
```

Listing 7.4: https://github.com/fdardenne/owlybook/blob/main/static/src/js/bottom_panel/properties.xml

Dans ce template, nous affichons le nom de la prop avec <t t-esc="property"/>. Ensuite, si la prop n'est pas "optional", nous ajoutons un astérisque (*). De plus, si la prop possède une aide (property_value.help), nous affichons une icône de point d'interrogation (<i class="fa fa-question-circle-o ps-2 owlybook_tooltip" t-if="property_value.help" t-att-data-tooltip="property_value.help"/>) qui quand survolé par la souris affiche une bulle d'informations (Configuré dans la story).

Ensuite, la deuxième section sert à afficher le type de la props en appelant la fonction JavaScript "propertyType". Voici le code de cette fonction :

```
propertyType(props) {
    const propsValidationType = props?.type?.name;

    if (propsValidationType) {
        return props.type.name;
    }
    if (Array.isArray(props.value)) {
        return "Array";
    }
    const str = typeof props.value;
    return str.charAt(0).toUpperCase() + str.slice(1);
}
```

Listing 7.5: https://github.com/fdardenne/owlybook/blob/main/static/src/js/bottom_panel/component$_p$*roperties/component_properties.js*

Dans cette fonction, nous récupérons le type de la props à partir de props?.type?.name (En javascript le "?" est un opérateur de chaînage optionnel ou l'opérateur de navigation sécurisée, il permet de vérifié d'abord si props existe et a une propriété appelée type. Ensuite, nous vérifions si type a une propriété appelée name [37]).

Si le type existe, nous le retournons directement. Sinon, nous vérifions si la valeur de la props est un tableau avec Array.isArray(props.value). Si c'est le cas, nous retournons le type "Array". Enfin, si aucun des cas précédents ne s'applique,

nous utilisons typeof props.value pour obtenir le type de la valeur et nous renvoyons la première lettre en majuscule suivie du reste du mot.

Pour finir, la dernière section de cet onglet est la plus importante puisqu'elle permet la modification de la props. Voici le template:

```
<t t-name="owlybook.properties.input" owl="1">
    <t t-set="propType" t-value="propertyType(property_value)"/>
    <t t-set="propValue" t-value="formatValue(propType, property_value.value)"/>
    <t t-if="propType === 'Boolean'">
        <input t-att-disabled="isDisabled(property_value)" class="o_input align-middle"
        t-att-checked="propValue" type="checkbox"
        t-on-change="(ev) => this.onChange(property_value, ev.target.checked)"/>
    </t>
    <t t-elif="property_value.choices">
        <div class="w-50">
            <SelectMenu
                value="propValue"
                choices="property_value.choices"
                onSelect="(value) => this.onChange(property_value, value)"
                searchable="property_value.length > 8"
            />
        </div>
    </t>
    <t t-elif="propType === 'String'">
        <input t-att-disabled="isDisabled(property_value)" class="o_input w-50 o_outline"
        t-att-value="propValue"
        t-on-input="(ev) => this.onChange(property_value, ev.target.value)"/>
    </t>
    <t t-elif="propType === 'Number'">
    <input t-att-disabled="isDisabled(property_value)" class="o_input w-50 o_outline"
    t-att-value="propValue"
    t-on-change="(ev) => this.onChange(property_value, ev.target.value)"/>
    </t>
    <t t-elif="propType === 'Object'">
```

```
30          <t t-if="propValue">
31              <ObjectRenderer object="propValue" name="
                    property"/>
32          </t>
33          <t t-else="">
34              undefined
35          </t>
36      </t>
37      <t t-elif="propType === 'Array'">
38          <t t-if="propValue">
39              <ObjectRenderer object="propValue" name="
                    property"/>
40          </t>
41          <t t-else="">
42              undefined
43          </t>
44      </t>
45      <t t-elif="propType === 'Function'">
46          <t t-esc="propValue?.name"/>
47      </t>
48      <t t-else="">
49          <input t-att-disabled="1" class="o_input w-50"/>
50      </t>
51      <i t-if="property_value.value !== undefined and !
            property_value.choices and
52      !isDisabled(property_value) and isDeleteBtnEnabled"
53      t-on-click="(ev) => this.onChange(property_value, '')"
54      class="fa fa-times ms-1 o_delete align-middle" aria-
            hidden="true"/>
55  </t>
```

Listing 7.6: https://github.com/fdardenne/owlybook/blob/main/static/src/js/bottom_panel/properties.xml

Ce template est donc responsable de l'affichage des "input" des différentes props. Il prend en compte le type de la prop et propose une interface adaptée en fonction de celui-ci. Voici un aperçu des différentes parties du template:

- Pour les props de type "Boolean", nous affichons une case à cocher (<input type="checkbox">) qui permet à l'utilisateur de modifier la valeur de la props.
- Pour les props avec des options prédéfinies (choices), nous affichons un menu déroulant (<SelectMenu>) qui permet de sélectionner une valeur parmi les options disponibles.

- Pour les props de type "String" et "Number", nous affichons un champ de texte (<input type="text">) où l'utilisateur peut entrer une valeur.
- Pour les props de type "Object" et "Array", nous utilisons un autre template appelé "ObjectRenderer" pour afficher le contenu de la prop de manière structurée (avec ma possibilité de cacher ou non le details de la props).
- Pour les props de type "Function", nous affichons le nom de la fonction.
- Pour les autres types de props, nous affichons un champ de texte désactivé c'est à dire qu'il ne peut pas être modifié.

De plus, nous avons également ajouté un bouton en forme de croix (<i class="fa fa-times">) qui permet de réinitialiser la valeur de la prop à une valeur vide, dans le cas où la prop n'est pas un choix (puisqu'elle a déjà une crois dans le menu déroulant) et n'est pas désactivée.

Pour les archs, la gestion des props est légèrement différente. Lorsque l'on clique sur l'onglet "props" dans le cas d'une arch, une autre classe JavaScript est utilisée :

```
<t t-if="mode === 'props'">
    <t t-if="stories.active.attrs">
        <ArchProperties/>
    </t>
    <t t-else="">
        <ComponentProperties/>
    </t>
</t>
```

Listing 7.7: https://github.com/fdardenne/owlybook/blob/main/static/src/js/bottom_panel/panel.xml

Selon la condition, soit la classe ArchProperties est utilisée, soit la classe ComponentProperties. Bien que le template d'affichage des props soit le même avec les trois grandes sections (nom, type, input), les fonctions utilisées sont différentes, car les composants et les archs ne sont pas construits de la même manière. Cependant, grâce à ces deux classes JavaScript, nous avons pu gérer les archs et les composants en utilisant le même template.

Cela permet une approche simple et unique pour la gestion des props, indépendamment du type d'élément (composant ou arch) sur lequel on travaille. Cela facilite la réutilisation du code et améliore la maintenabilité de notre plateforme.

Maintenant que l'onglet "props" a été expliqué, nous pensons qu'il est également pertinent de comprendre comment nous gérons les choix et les objets ou tableaux. Comme mentionné précédemment lors de l'explication du template responsable de l'affichage des différentes props, nous avons mis en place la possibilité de gérer ces deux cas là.

Pour les objets ou tableaux, nous utilisons la class Javascript ObjectRenderer et son template associé qui permettent d'afficher la structure de l'objet de manière détaillée. Cela nous permet d'explorer les propriétés de l'objet et de visualiser leur valeur respective. En voici un exemple:

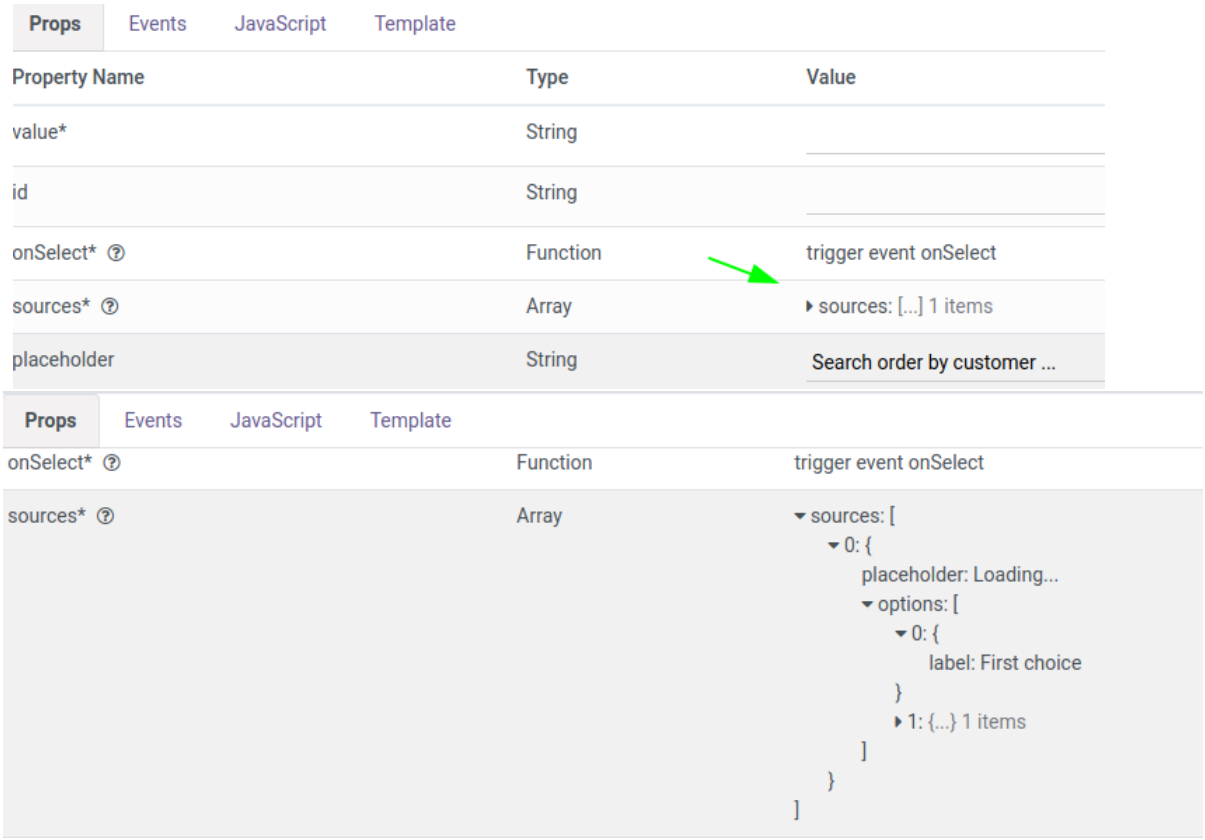

Figure 7.7: Affichage des objets ou tableaux

Pour créer une interface pour géré les objects ou les tableaux, le fonctionnement se passent principalement dans le template suivant:

```
<?xml version="1.0" encoding="utf-8"?>
<templates>
    <t t-name="owlybook.ObjectRenderer" owl="1">
        <t t-set="name" t-value="props.name"/>
        <t t-set="object" t-value="props.object"/>
        <t t-set="uncollapsedObject" t-value="uncollapsed"
        <t t-call="owlybook.ObjectRenderer.object"/>
    </t>

    <t t-name="owlybook.ObjectRenderer.object" owl="1">
        <t t-if="object and isObject(object)">
            <span t-on-click="() => toggleCollapse(
               uncollapsedObject, name)"
            class="o_owlybook_hover">
                <i t-if="uncollapsedObject[name]" class="fa
                   fa-caret-down"></i>
                <i t-else="" class="fa fa-caret-right"></i>
                <span> <t t-esc="name"/>: </span>
                <span class="text-muted" t-if="!
                   uncollapsedObject[name]">
                    <t t-esc="openingBracket(object)"/>...<t
                        t-esc="closingBracket(object)"/>
                    <t t-esc="Object.keys(object).length"/>
                       items
                </span>
                <t t-if="uncollapsedObject[name]" t-esc="
                   openingBracket(object)"/>
            </span>
            <t t-if="uncollapsedObject[name]">
                <ul class="m-0">
                    <t t-foreach="Object.keys(object)" t-as=
                       "subObject" t-key="subObject">
                        <li class="ms-4">
                            <t t-call="owlybook.
                               ObjectRenderer.object">
                                <t t-set="uncollapsedObject"
                                    t-value="
                                   uncollapsedObject[name]"/
                                   >
                                <t t-set="name" t-value="
                                   subObject"/>
```

```
30                                      <t t-set="object" t-value="
                                           object[subObject]"/>
31                                  </t>
32                              </li>
33                          </t>
34                      </ul>
35                      <t t-esc="closingBracket(object)"/>
36                  </t>
37              </t>
38              <t t-else="">
39                  <span> <t t-esc="name"/>: </span>
40                  <t t-esc="object"/>
41              </t>
42      </t>
43  </templates>
```

Listing 7.8: https://github.com/fdardenne/owlybook/blob/main/static/src/js/components/object_renderer/object_renderer.xml

Ce template nous permet de gérer plusieurs cas intéressants. En utilisant ce template, nous pouvons choisir d'ouvrir ou de fermer les objets ou tableaux pour obtenir plus d'informations sur leur structure et leurs éléments.

Lorsqu'un objet ou un tableau est fermé, il affiche son nom suivi d'une indication du nombre d'éléments. En cliquant sur l'icône appropriée, nous pouvons ouvrir ou réduire l'affichage pour visualiser les propriétés ou les éléments de manière détaillée.

Ce template permet d'explorer la structure des objets et des tableaux de façon simple et intuitive, permettant ainsi aux utilisateurs de naviguer facilement et de comprendre la composition et les valeurs des données.

En ce qui concerne les choix, nous utilisons la class Javascript SelectMenu et son template associé qu offrent une liste déroulante contenant les différentes options disponibles pour une prop donnée. L'utilisateur peut sélectionner l'une des options et la valeur correspondante sera mise à jour dans le rendu de la story. Pour la faire fonctionner, nous utilisons des elements présent et créer par Odoo qui sont les "dropdown" et "drowdownItems" qui nous permettent de créer cette liste de choix dont voici un exemple:

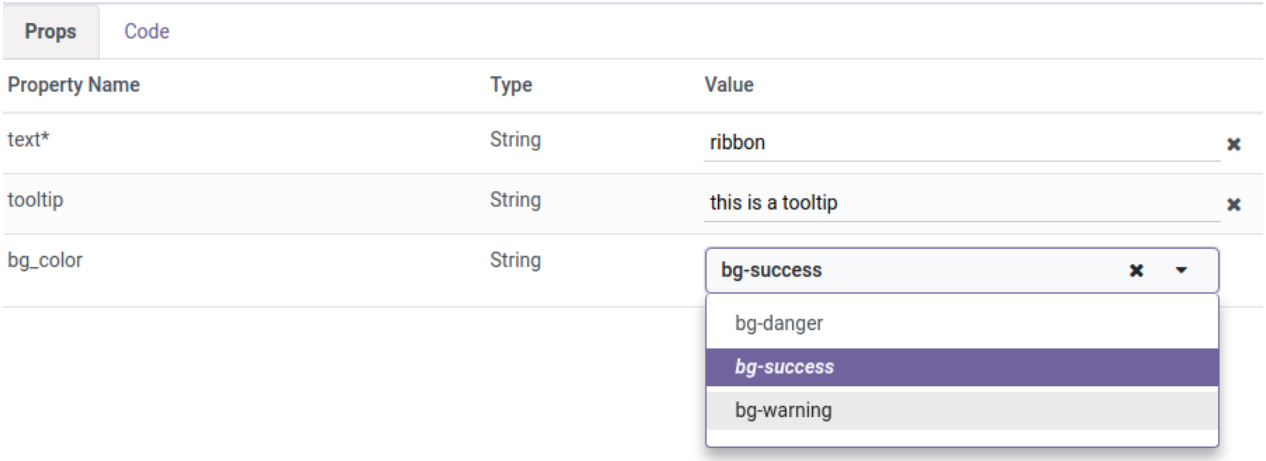

Figure 7.8: Affichage des choix

7.4.7 Events

Cet onglet sert de journal ou de "logs" pour certaines actions effectuées dans une story. En effet, lors de la création d'une story, il est possible d'ajouter la fonctionnalité permettant de marquer chaque appel d'une fonction dans l'onglet "Events" pour les props de type fonction.

Prenons par exemple l'une des stories les plus basiques, celle du composant "checkbox".

```
CheckBoxParent.storyConfig = {
    title: "Checkbox",
    component: CheckBox,
    props: {
        value: {
            value: true,
        },
        className: {
            value: "form-switch",
        },
        name: {
            value: "beautiful_name",
        },
        onChange: {
            value: getEventFunction("onChange"),
            help: "Called when the user clicked on the
               checkbox",
```

```
        },
    },
};
```

Listing 7.9: https://github.com/fdardenne/owlybook/blob/main/static/src/stories/checkbox/checkbox.stories.js

On peut observer que la prop "onChange", qui est une fonction détectant les changements d'état du composant, est définie avec un appel de fonction "getEventFunction("onChange")". Voici les détails de cette fonction:

```
export function getEventFunction(name) {
    const functionName = "trigger event " + name;
    return {
        [functionName](...args) {
            onEvent(name, args);
        },
    }[functionName];
}

export function onEvent(name, args) {
    const params = {};
    for (let i = 0; i < args.length; i++) {
        params[`arg #${i}`] = args[i];
    }
    getStories().active.events.push({ name, params });
}
```

Listing 7.10: https://github.com/fdardenne/owlybook/blob/main/static/src/js/stories.js

La fonction getEventFunction(name) permet de créer dynamiquement une fonction en utilisant le nom fourni en argument. Elle crée une propriété avec ce nom et lui attribue une fonction qui appelle la méthode onEvent(name, args). Ainsi, chaque fois que cette fonction est appelée, elle enregistre l'événement dans l'onglet "Events"

Ainsi, lorsque la fonction "onChange" est appelée dans le contexte de la story de la checkbox, un événement est enregistré avec le nom "onChange" et les paramètres spécifiques de l'appel, ce qui permet de suivre et d'afficher les actions effectuées dans l'onglet "Events".

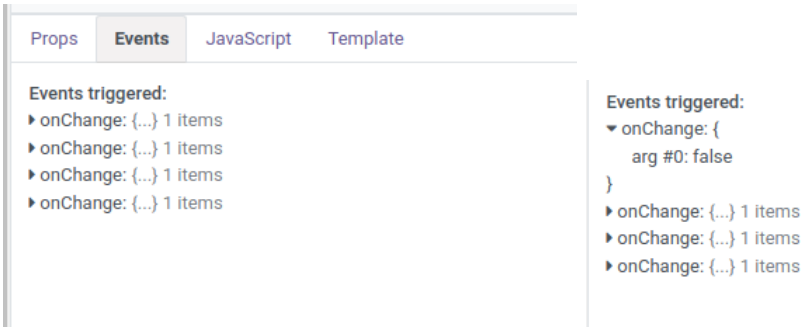

Figure 7.9: Exemple de l'onglet "Events"

7.4.8 Javascript

Dans l'onglet "Javascript", vous trouverez le code du composant Javascript de la story sélectionnée. Cet onglet est principalement utilisé pour copier et coller le code facilement. A l'instar des vues, la modification du code dans cet onglet n'affectera pas le comportement de la story active.

Pour afficher l'onglet, il suffit d'ajouter la classe correspondante dans le fichier de la story. Par exemple, pour la story "checkbox", voici comment cela peut être réalisé :

```
1 class CheckBoxParent extends Component {
2     static template = "owlybook.CheckBoxStories";
3     static components = { CheckBox };
4 }
```

Listing 7.11: https://github.com/fdardenne/owlybook/blob/main/static/src/stories/checkbox/checkbox.stories.js

Une fois cela fait, l'onglet "Javascript" sera disponible dans Owlybook, comme illustré dans la figure suivante :

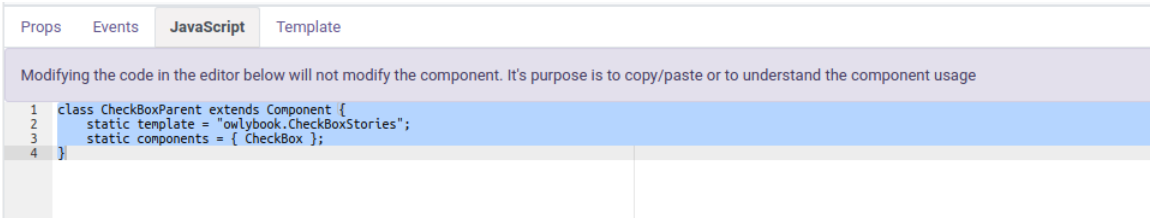

Figure 7.10: Exemple de l'onglet "JavaScript"

Concernant le lien entre l'affichage sur la plateforme et la classe présente dans le fichier de la story, dès que l'utilisateur clique sur l'onglet, une fonction "setup"

est lancée. Voici le code correspondant :

```
setup() {
    this.stories = useStories();
    if (this.props.mode === "js") {
        this.sourceCode = this.stories.active.
           parentComponent.toString();
    } else {
    const codeTemplateName = this.stories.active.
       parentComponent.codeTemplate;
    const templateName = this.stories.active.parentComponent
       .template;
    this.sourceCode =
        // @ts-ignore
        this.owl.app.rawTemplates[codeTemplateName ||
           templateName]?.innerHTML;
    }
}
```

Listing 7.12: https://github.com/fdardenne/owlybook/blob/main/static/src/js/bottom_panel/component_code/component_code.js

Dans cette fonction, nous commençons par récupérer l'objet stories grâce à la fonction useStories(). Ensuite, nous vérifions si le mode actuel est défini comme "js" (JavaScript). Si tel est le cas, nous utilisons la méthode toString() pour convertir la classe parentComponent en une chaîne de caractères représentant le code source JavaScript correspondant ensuite il nous suffit de l'afficher grâce à ce template:

```
<templates>
    <t t-name="owlybook.ComponentCode" owl="1">
        <div class="alert alert-primary mb-0" role="alert">
            Modifying the code in the editor below will not
               modify the component.
            It's purpose is to copy/paste or to understand
               the component usage
        </div>
        <CodeEditor type="mode" value="sourceCode" onChange=
           "() => {}" id="'studioXmlEditor'" />
    </t>
</templates>
```

Listing 7.13: https://github.com/fdardenne/owlybook/blob/main/static/src/js/bottom$_p$*anel*/*component_code*/*component_code.xml*

7.4.9 Template

Cet onglet est assez similaire à celui de "JavaScript". En cliquant sur l'onglet, la même méthode "setup" est déclenchée, à l'exception que cette fois-ci nous passerons par la partie "else" lorsque le "mode" est différent de "js". Ensuite, en utilisant les informations de la story, nous pouvons créer un template XML pour la story active. Voici le résultat pour la story "checkbox" :

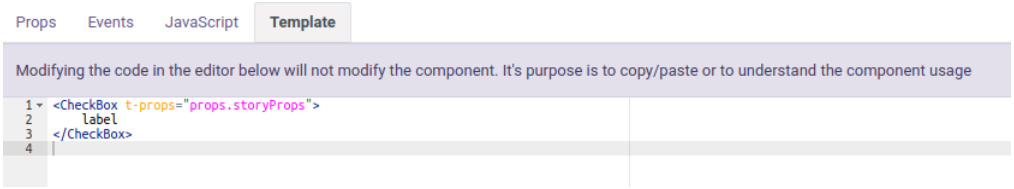

Figure 7.11: Exemple de l'onglet "Template"

Dans la configuration de la story check box on peut y retrouver cette ligne:

```
CheckBoxParent.codeTemplate = "owlybook.CheckBoxCall";
```

Listing 7.14: https://github.com/fdardenne/owlybook/blob/main/static/src/stories/checkbox/checkbox.stories.js

Elle permet donc que le "codeTemplateName" = "owlybook.CheckBoxCall". Maintenant si on va dans le fichier XML correspondant à la story on peut y retrouver ce template:

```
<?xml version="1.0" encoding="UTF-8" ?>
<templates xml:space="preserve">
    <t t-name="owlybook.CheckBoxStories" owl="1">
        <h1> CheckBox </h1>

        <span> The checkbox component is a graphical user
           interface element that represents a
        binary option, i.e., a choice between two mutually
           exclusive states, such as "yes" or
        "no", "true" or "false", etc. </span>

        <t t-call="owlybook.CheckBoxCall"/>

        Play with the props underneath and sees the results
           above!
    </t>

    <t t-name="owlybook.CheckBoxCall" owl="1">
```

```
16          <CheckBox t-props="props.storyProps">
17              label
18          </CheckBox>
19      </t>
20  </templates>
```

Listing 7.15: https://github.com/fdardenne/owlybook/blob/main/static/src/stories/checkbox/checkbox.stories.xml

Le fichier XML contient deux templates : "owlybook.CheckBoxStories" et "owlybook.CheckBoxCall". Le template "owlybook.CheckBoxStories" représente l'ensemble de la story checkbox avec une description et des instructions. À l'intérieur de ce template, nous appelons le template "owlybook.CheckBoxCall" avec l'instruction <t t-call="owlybook.CheckBoxCall"/>. Le template "owlybook.CheckBoxCall" représente le code XML spécifique au composant Checkbox avec des props passées via l'attribut t-props="props.storyProps". Ainsi, en cliquant sur l'onglet "Template", nous pouvons visualiser le template XML généré pour la story checkbox, ce qui nous permet de comprendre la structure et le rendu de la story sur la plateforme.

Maintenant, après avoir exécuté la méthode "setup", la variable "this.sourceCode" contient le contenu du template XML généré pour la story active. Ce code source représente la structure et les éléments de la story tels qu'ils seront affichés sur la plateforme. Grâce au template montré dans la section précédente, nous pouvons afficher le code source dans l'onglet correspondant.

Cela nous permet de voir la hiérarchie des éléments, les balises utilisées, les attributs spécifiés et les instructions utilisées pour le rendu de la story. Grâce à cette fonctionnalité, nous pouvons inspecter et comprendre le code source de la story dans son onglet dédié, facilitant ainsi la compréhension. Il est important de noter que comme l'onglet Javascript, la modification du contenu dans l'onglet Template ne changera pas la story active en temps réel, mais permet à l'utilisateur de copier/coller le template pour une utilisation ultérieure.

7.4.10 Code

Cet onglet ne sera affiché que dans le cas d'une arch. En effet, il permettra de montrer le code de celle-ci et il sera possible de le modifier en direct, permettant ainsi de changer l'arch à la volée. Lorsque l'utilisateur sélectionne une arch et clique sur l'onglet "code", un éditeur de code sera affiché grâce à la condition "elif" suivante :

```
<t t-elif="mode === 'code'">
    <CodeEditor type="'qweb'" value="stories.active.
       dirtyArch || stories.activeArch"
    onChange="value => stories.active.dirtyArch = value" id=
       "'studioXmlEditor'" />
</t>
```

Listing 7.16: https://github.com/fdardenne/owlybook/blob/main/static/src/js/bottom_panel/panel.xml

Cet extrait de code utilise un template CodeEditor qui prend en charge le type de code "qweb" et affiche la valeur de stories.active.dirtyArch ou stories.activeArch. Lorsque le code est modifié, la fonction onChange est appelée pour mettre à jour la valeur de stories.active.dirtyArch. Cela permet à l'utilisateur de modifier le code de l'arch directement dans l'éditeur et de voir les modifications en temps réel. Voici le template:

```
<templates>
    <t t-name="web_studio.CodeEditor" owl="1">
        <div t-ref="editorRef" id="ace-editor" class="
           o_web_studio_code_editor w-100 h-100">
        </div>
    </t>
</templates>
```

Listing 7.17: https://github.com/fdardenne/owlybook/blob/main/static/src/js/components/code_editor/code_editor.xml

Ce template fournit la structure de base pour l'éditeur de code, qui sera ensuite manipulé et configuré par le code JavaScript associé.

On peut donc voir dans la figure suivante le resultat qui nous permet de voir le code de l'arch selectionné et de pouvoir le modifié directement depuis l'éditeur.

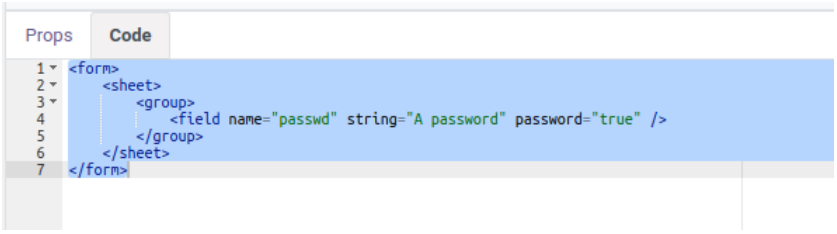

Figure 7.12: Exemple de l'onglet "Code"

7.5 Evolution de la plateforme

Durant ce mémoire la plateforme que nous avons crée a beaucoup évolué au cours des semaines de travail que ce soit au niveau du design, du nom mais aussi au niveau de sa structure de son fonctionnement. Dans cette section, nous allons montrer l'évolution de la plateforme.

Tout d'abord, parlons du nom de la plateforme. Au vu du titre de notre mémoire, nous avons commencé par utiliser le nom "Storybook". Cependant, nous savons que ce nom était déjà utilisé pour une autre application et nous avons donc décidé de créer notre propre identité. Après réflexion, nous avons choisi de la nommer "UI Playground". Nous avons utilisé ce nom et son identité pendant plusieurs mois, mais nous avons finalement décidé de trouver un nom plus adapté basée sur le framework Owl, nous avons cherché un nom qui s'inscrive dans cette même thématique et c'est ainsi que nous avons créé le nom "Owlybook".

En ce qui concerne le design, nous avons initialement choisi de nous inspirer du design général des pages d'Odoo, en utilisant des couleurs principalement blanches et mauves. Cependant, une nouvelle version du design d'Odoo, appelée "Odoo Milk", a été annoncée avec un design plus moderne, et nous avons donc décidé de changer complètement notre design pour correspondre à cette nouvelle version. Comme vous pouvez le voir sur les captures d'écran d'Owlybook, nous avons opté pour un design très minimaliste et moderne, principalement blanc avec quelques touches de couleurs vives.

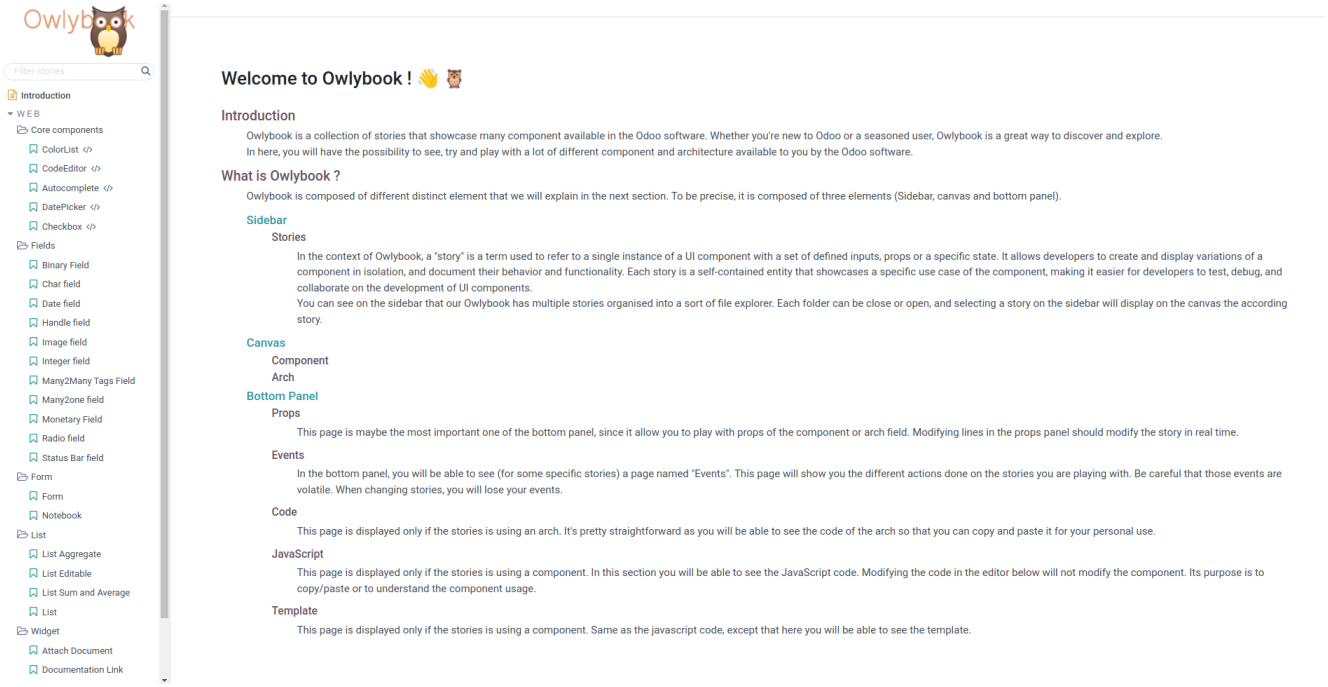

Figure 7.13: Owlybook

7.6 Design

Dans cette section, nous allons approfondir nos choix en ce qui concerne le design de notre plateforme. Pour commencer, nous avons collaborativement créé différentes maquettes afin de fusionner nos idées et d'imaginer un design qui nous plaisait à tous les deux. Une fois ces maquettes réalisées, nous avons sollicité les avis des développeurs, du Product Owner, ainsi que des personnes n'ayant pas de familiarité avec Odoo ou des plateformes similaires. Ces divers retours nous ont permis d'améliorer notre maquette en prenant en compte des perspectives variées.

Après avoir intégré ces retours, nous sommes parvenus à la maquette finale que nous vous présentons ci-dessous. Celle-ci est le résultat de notre processus itératif et de notre volonté d'obtenir un design qui corresponde à la fois à nos attentes et aux besoins des utilisateurs.

Pendant la phase d'implémentation, le design a naturellement évolué. Certains changements ont été apportés puis abandonnés, tandis que d'autres ont été conservés. Cependant, nous sommes assez fiers du fait que le design final se rapproche grandement de notre vision initiale.

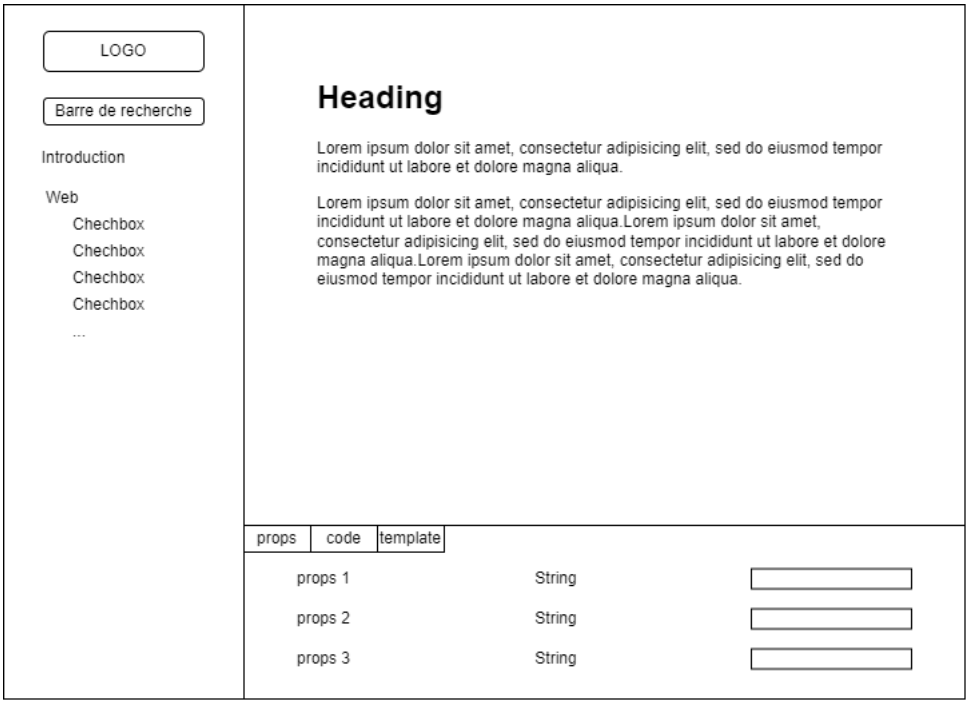

Figure 7.14: Maquette de la plateforme

L'un des changements majeurs dans le design est survenu suite à l'annonce d'Odoo "Milk". En suivant la version "master" d'Odoo, nous avons décidé d'effectuer une refonte graphique en accord avec les principes de cette nouvelle version. Cela a nécessité des modifications dans les trois principales parties de notre plateforme : la barre latérale (sidebar), le canvas et le panneau inférieur (bottom panel).

Pour la gestion du design, nous avons principalement utilisé le langage CSS et Bootstrap, ce qui nous a grandement facilité la tâche lors de l'implémentation. Ces outils nous ont offert des fonctionnalités et des composants prêts à l'emploi, nous permettant ainsi de créer une interface visuellement attrayante et cohérente.

Notre approche itérative et bien organisée, combinée à la sollicitation des avis des potentiels futurs utilisateurs, nous a réellement permis d'améliorer progressivement le design et de concevoir une plateforme agréable à utiliser, pensée pour offrir la meilleure expérience possible.

Voici un exemple de CSS que nous avons utilisé pour obtenir le résultat souhaité au niveau de la barre latérale, plus précisément pour la barre de recherche:

```
.o_searchview_ui {
    align-items: flex-end;
    padding: 0 20px 1px 0;
    position: relative;

    .o_searchview_input_container {
        display: flex;
        flex-flow: row wrap;
        position: relative;
    }

    .o_searchview_input {
        flex: 1 0 auto;
        width: 75px;
        border: none;
        outline: none;
        background-color: var(--o-input-background-color, #{
            $input-bg});
    }

    .o_searchview_icon {
        @include o-position-absolute($top: 4px, $right: 0);
    }
}
```

Listing 7.18: https://github.com/fdardenne/owlybook/blob/main/static/src/js/sidebar/sidebar.scss

7.7 Tests

Le testing est le processus permettant de vérifier qu'un logiciel ne comporte pas d'erreurs et correspond aux spécifications requises. Il s'agit d'une étape essentielle dans le développement d'un logiciel, car les tests nous assurent que les spécifications sont correctes à toutes les étapes du processus de développement mais aussi de fournir aux utilisateurs un logiciel sans problèmes, flexible, performant et efficace. [38]

7.7.1 Méthodologie

Notre stratégie consiste à créer un ou plusieurs tests unitaires pour chaque fonctionnalité. De cette manière, nous nous assurons que le développement des fonctionnalités ultérieures ne provoque pas le dysfonctionnement des fonctionnalités

existantes.

Tous les tests étaient exécutés avant le merge par un membre du binôme qui jouait le rôle du reviewer. Ainsi, si un test échouait, la fonctionnalité n'était pas mergé tant que le test n'était pas corrigé.

Il nous est également arrivé de détecter des bugs lors du développement de la plateforme. Dans ces cas-là, nous les corrigeons et les accompagnons d'un test. En procédant ainsi, nous nous assurons de la stabilité de celle-ci. En effet, si un bug déjà connu se présente à nouveau dans le futur, un test préviendra qu'il soit inclut dans le code.

7.7.2 Test unitaire

Les tests unitaires consistent à tester des unités spécifiques d'un logiciel. C'est la méthode que nous avons adoptée pour tester.
Ceux-ci ont été écrits avec la librairie QUnit, qui permet d'écrire des tests pour des parties spécifiques de l'interface utilisateur.

À travers ces tests, nous simulons l'interaction qu'un utilisateur peut avoir avec des composants Owl spécifiques représentant des parties de l'interface.Cela nous permet de vérifier que les réponses visible dans l'interface soient les bonnes.

Nous les utilisons également pour tester la cohésion entre tous les composants JavaScript qui constituent l'interface utilisateur. Ainsi, nous évitons les surprises en cas d'incohérence dans l'interaction entre deux ou plusieurs composants.

Nous déterminons la majorité de nos tests comme étant des tests fonctionnel. Les tests fonctionnels sont des tests dit en "black box": le code du programme est ignoré, les test vérifient l'input/l'output et les tests vérifient les spécifications fonctionnelle. [39]

7.7.3 Coverage

Afin d'obtenir une indication sur la portion de code testée, nous avons effectué une couverture de code de nos tests sur Owlybook.

Nous avons opté pour la stratégie de couverture dite "statement coverage" (couverture des instructions). Le principe est simple : le résultat de la couverture est obtenu en divisant le nombre d'instructions testées par le nombre total d'instructions présentes dans Owlybook.

$$C_{stmt} = \frac{\#\ executed\ statements}{\#\ statements}$$

[39]

Le code coverage est généralement utilisé pour les tests de type structurel mais il peut également être utilisé comme indicateur dans les tests de type fonctionnel. Même si le testing fonctionnel se concentre davantage sur les fonctionnalités du logiciel, le code coverage peut néanmoins fournir des informations utiles.

Lorsqu'une ou plusieurs lignes de code ne sont pas testées, cela peut indiquer des scénarios fonctionnels non couverts. Cela nous donnes donc une bonne indication sur le type de fonctionnalité qu'il reste à tester.

Nous avons utilisé l'outil "Coverage" [40] de Chrome, qui permet d'évaluer les lignes de code JavaScript non utilisées lors d'un scénario.

Nous avons donc exécuté l'ensemble de nos tests avec la fonctionnalité de coverage, et les résultats indiquent que nos tests atteignent une coverage de 94,9 %.

Nos analyses sur le coverage ont révélé un manque de tests pour l'onglet "Event" de owlybook, ainsi que des cas extrêmes nécessitant de provoquer des erreurs pour vérifier que les exceptions sont correctement lancées.

7.8 Intégration du Owlybook dans la documentation sphinx de Odoo

7.8.1 Qu'est-ce que Sphinx ?

Sphinx est un générateur de documentation comportant plusieurs fonctionnalités:

- Génération de page web, de pdf, de documents pour e-readers
- Utilisation de reStructuredText ou Markdown pour l'écriture de documentation
- Un système permettant de faire des cross-références dans la documentation
- Du syntax highlighting pour le code
- un écosystème avec des extensions installable

[41] Odoo utilise Sphinx pour la génération de la documentation, celui-ci est open-source et accessible sur le lien suivant: `https://github.com/odoo/documentation`.

7.8.2 Sphinx et Owlybook

Une bonne piste pour le futur de notre plateforme serait de pouvoir l'intégrer à la documentation technique existante. Ainsi cela permettrait de pouvoir documenter des composants Owl et des vues de manière intéractive.
Les utilisateurs aurons donc la documentation de chaque composants ainsi que la possibilité de jouer avec les propriétés de celui-ci et d'en voir le code source.

La plateforme que nous avons créer ne serait donc plus un outil isolé d'Odoo, réservé aux seules personnes connaissant son existence, mais ferait partie intégrante de son écosystème. Il serait utilisé par tous les développeurs souhaitant développer dans Odoo.

7.8.3 Intégration Technique

Nous avons développé un proof of concept permettant d'intégrer Owlybook dans Sphinx pour la documentation d'Odoo.

Pour cela, nous utilisons des iframes. Un iframe est un élément HTML qui permet d'intégrer une page distante dans la page courante.

Cependant, cette approche présente un inconvénient : lorsqu'un utilisateur charge une page de la documentation, tous les iframes de la page sont également chargés. Cela pose un problème car le chargement d'un seul iframe de notre plateforme représente environ 7 Mo de données et les iframes ne partagent pas leur ressources.

La taille importante est due au fait que le chargement de l'ensemble de la base de code JavaScript d'Odoo est nécessaire pour son instanciation ainsi que celle des stories.

Une solution possible consiste à utiliser le lazy loading pour les iframes. Le lazy loading permet de charger les éléments uniquement lorsqu'ils sont requis.

Le lazy loading des iframes est facile à implémenter car les navigateurs proposent une option permettant de le faire via l'attribut loading="lazy".

Cependant, cette méthode présente certains inconvénients. Tout d'abord, tous les navigateurs ne prennent pas encore en charge cet attribut. Selon caniuse.com, Firefox ne le supporte pas encore. [42] De plus, cela ne résout pas le problème selon lequel chaque iframe nécessite le téléchargement de 7 Mo de données.

Une solution complémentaire consisterait à ce que Sphinx lance un serveur Odoo pour regrouper le JavaScript lors de la compilation. Ainsi, Sphinx aurait le

contrôle sur le JavaScript proposé par notre plateforme, ce qui permettrait aux navigateurs de le télécharger une seule fois par page visitée.

7.8.4 État de l'intégration

Pour le moment, nous nous sommes concentrés sur l'implémentation de la plateforme en elle même. L'intégration de celui-ci dans la documentation est considérée comme un "Proof of Concept" et n'a pas encore été réalisée.

Techniquement, nous avons réussi à mettre en place la possibilité de l'intégrer dans la documentation en utilisant des iframes avec le lazy loading. Cependant, il reste encore du travail à faire pour la deuxième alternative mentionnée dans la section précédente, ainsi que pour la réécriture de la documentation afin de l'y inclure.

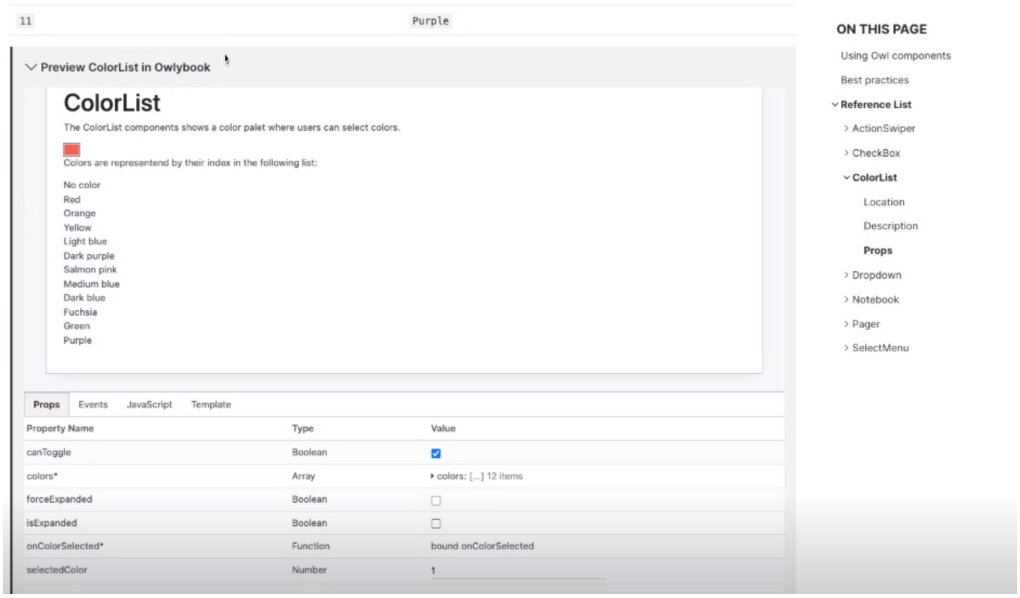

Figure 7.15: Proof of concept: iframe Owlybook lazy loadé dans la documentation

Futur de la plateforme

Une fois l'implémentation de notre plateforme terminée, nous avons eu des discussions avec le chef de l'équipe Javascript ainsi qu'avec le CTO afin de discuter du futur d'OwlyBook et de la possibilité de le mettre officiellement en ligne pour permettre aux développeurs d'Odoo ainsi qu'à la communauté de l'utiliser.

Bien évidemment, nous avons encore des idées pour faire évoluer OwlyBook, que ce soit des nouvelles fonctionnalités ou des modifications sur les fonctionnalités existantes.

Idées potentielles :

- Création de stories directement sur la plateforme
- Amélioration de la sidebar avec l'ajout de boutons pour replier/déplier les dossiers, cacher la sidebar, etc.
- Support de plus de propriétés (props)
- Redimensionnement de la sidebar et du bottom panel
- Permettre que la modification du code change la story en temps réel
- ...

Conclusion

Au cours de ce mémoire, nous avons réussi à atteindre les différents objectifs fixés par Odoo, à savoir la création d'une plateforme permettant de visualiser et d'interagir avec une grande variété de composants de l'interface d'Odoo dans différents scénarios.

Après avoir analysé soigneusement les différentes options pour mener à bien ce projet, nous avons commencé par tenter d'étendre "Owl" à la plateforme déjà existante "Storybook". Cependant, après avoir constaté que ce n'était pas la meilleure solution, nous avons choisi de développer notre propre plateforme, "OwlyBook".

Ce travail nous a permis d'acquérir de nombreuses compétences dans différents domaines. Tout d'abord, nous avons appris à travailler en binôme sur un projet de longue durée, ce qui nous a permis de mieux comprendre notre propre méthode de travail, nos points forts et nos points faibles, ainsi que de sortir de notre zone de confort. Nous avons également eu l'opportunité de découvrir l'ambiance de travail au sein d'une entreprise telle qu'Odoo.

Au niveau informatique, ce projet nous a permis de nous perfectionner en JavaScript, un langage que nous avions peu abordé durant nos études, ainsi que sur le framework d'Odoo. Nous avons également acquis une solide expérience de GitHub, une compétence qui sera précieuse pour nos futurs projets.

Enfin, nous avons identifié des pistes d'amélioration pour OwlyBook, telles que la création de stories directement sur la plateforme, l'amélioration de la sidebar, le support de plus de props, le resizing de la sidebar et du bottom panel, et la modification du code pour changer la story à la volée. Nous sommes convaincus que ces améliorations pourraient rendre la plateforme encore plus utile pour les utilisateurs.

En somme, ce projet nous a permis de relever de nombreux défis, de développer

de nouvelles compétences et de mieux comprendre les méthodes de travail en entreprise. Nous sommes fiers du résultat obtenu et confiants quant à l'avenir d'OwlyBook.

Annexe

10.1 Guide utilisation

Pour pouvoir utiliser notre plateforme, la première chose à faire est d'installer la version gratuite d'Odoo. Pour cela, vous pouvez suivre la documentation présente sur ce lien :

```
https://www.odoo.com/documentation/16.0/administration/
   install/install.html
```

Une fois Odoo installé, vous devriez avoir un dossier qui ressemble à "dossier/odoo". Dans ce même dossier, créez un nouveau dossier "custom". Clonez le repository git d'Owlybook sur le lien suivant : https://github.com/fdardenne/owlybook dans ce dossier.

À cette étape, vous devriez avoir un dossier contenant deux sous-dossiers ("odoo" et "custom"). Une fois que c'est fait, ouvrez un terminal pointant vers le dossier "odoo".

Dans ce terminal, lancez la commande suivante :

```
1 ./odoo-bin --addons-path=addons,../custom/ --database
   db_owlybook -i owlybook --limit-time-cpu=9999 --limit-
   time-real=9999
```

Cette commande vous permettra de lancer votre serveur local avec une base de données Postgres nommée "db_owlybook" (grâce au paramètre –database). De plus, le module permettant d'avoir owlybook s'installera (grâce au paramètre -i owlybook).

Maintenant que cela est fait, l'adresse suivante devrait vous permettre d'arriver sur la page d'Owlybook :

```
http://localhost:8069/owlybook
```

Comme expliqué précédemment, Owlybook se compose de trois parties majeures. Tout d'abord, une "side bar" qui permet d'afficher toutes les stories disponibles et d'y accéder facilement. Elle est également équipée d'une barre de recherche pour faciliter la recherche d'une story en particulier. Ensuite, une fois une story sélectionnée, celle-ci s'affiche au centre de la plateforme (le canvas), vous permettant de vous amuser directement avec les composants ou l'architecture sélectionnés. Mais ce n'est pas tout, il existe une troisième partie importante : le "bottom panel". Celui-ci permet de modifier les props à la volée et dispose également de sections supplémentaires pour afficher, par exemple, le code de la story sélectionnée.

Voici un exemple d'utilisation d'Owlybook :

- Lorsque vous arrivez sur la plateforme Owlybook, prenez le temps de lire la page d'introduction, qui vous fournira des informations importantes sur la plateforme elle-même.
- Ensuite, accédez à une story en tapant "Ribbon" dans la barre de recherche et en sélectionnant la story correspondante.
- Une fois sur la page de la story, vous verrez une vue typique d'Odoo avec un ruban vert en haut à droite.
- Depuis le bottom panel, vous aurez la possibilité de changer le texte affiché sur le ruban, son tooltip (en passant votre souris dessus, un tooltip s'affichera) ainsi que la couleur de ce ruban.

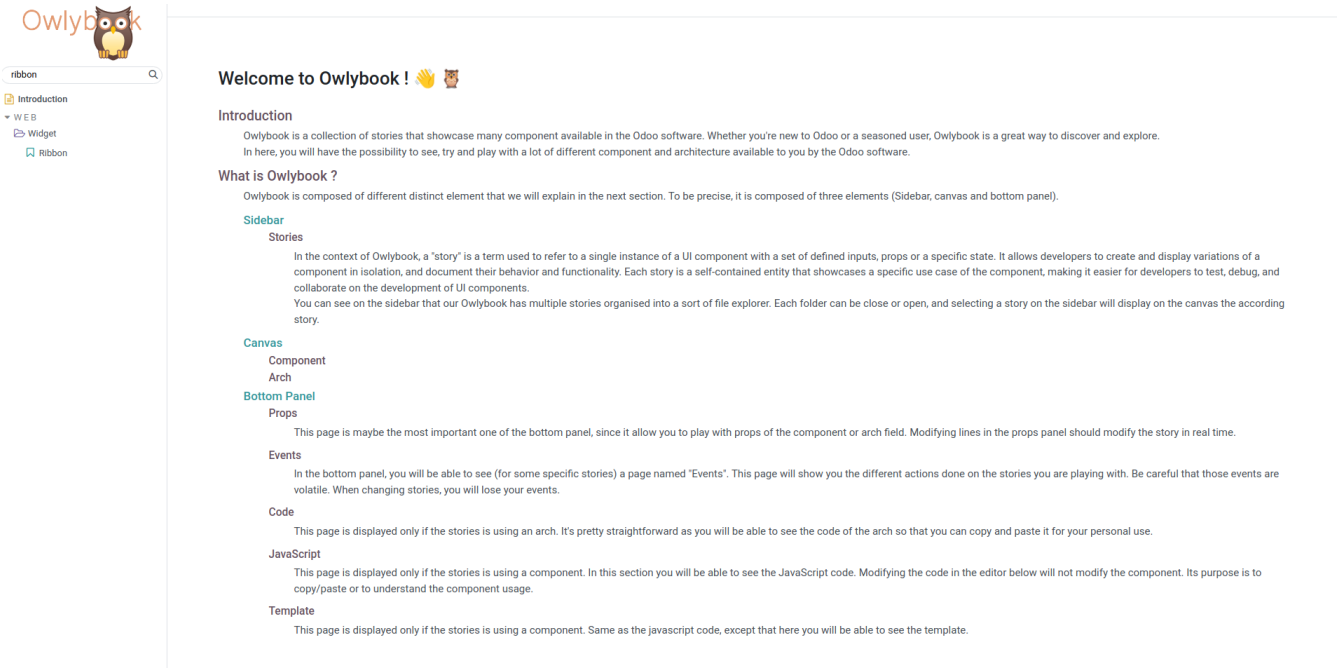

Figure 10.1: Page d'introduction

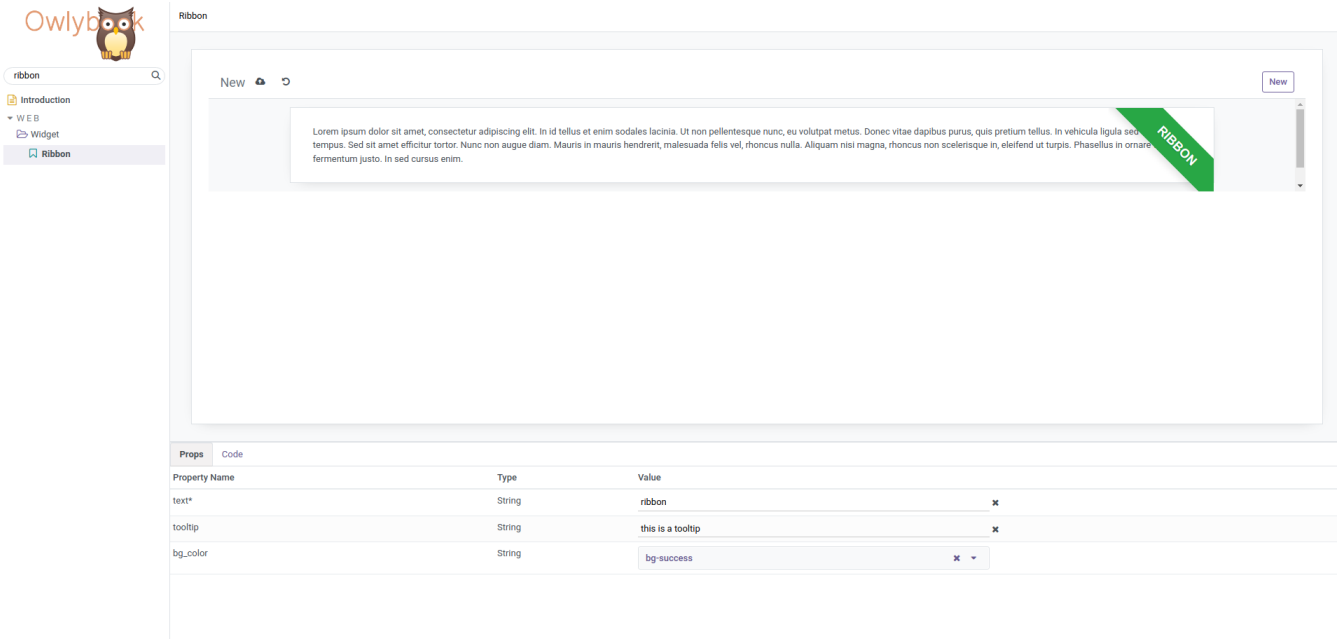

Figure 10.2: Selection de la story

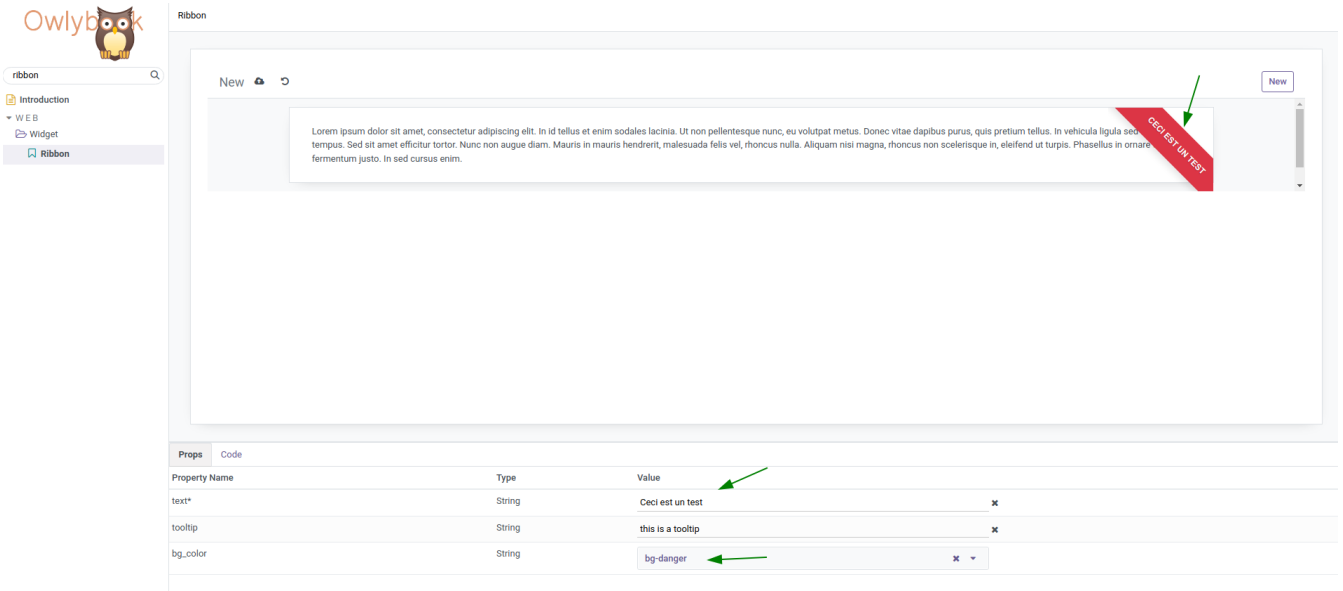

Figure 10.3: Changement des props

Bibliography

[1] Erp. https://fr.wikipedia.org/wiki/ERP (visité le 08/04/2023).

[2] Annual recurring revenue (arr). https://www.productplan.com/glossary/annual-recurring-revenue/#:~:text=Annual\%20recurring\%20revenue\%20(ARR)\%20refers,they\%20can\%20expect\%20each\%20year. (visité le 08/04/2023).

[3] Chief executive officer (ceo). https://en.wikipedia.org/wiki/Chief_executive_officer (visité le 08/04/2023).

[4] Chief technology officer (cto). https://www.investopedia.com/terms/c/chief-technology-officer.asp (visité le 08/04/2023).

[5] Chief financial officer (cfo). https://www.investopedia.com/terms/c/cfo.asp (visité le 08/04/2023).

[6] Chief compliance officer (cco). https://en.wikipedia.org/wiki/CCO (visité le 08/04/2023).

[7] Object-relational mapping (orm). https://www.theserverside.com/definition/object-relational-mapping-ORM#:~:text=Object\%2Drelational\%20mapping\%20(ORM)\%20is\%20a\%20way\%20to\%20align,language\%20and\%20a\%20relational\%20database. (visité le 15/04/2023).

[8] Model-view-controller (mvc). https://en.wikipedia.org/wiki/Model\%E2\%80\%93view\%E2\%80\%93controller (visité le 15/04/2023).

[9] Cascading style sheets (css). https://www.w3schools.com/css/css_intro.asp (visité le 10/05/2023).

[10] Odoo. https://www.odoo.com/fr_FR/ (visité le 08/04/2023).

[11] Survival guide. https://www.odoo.com/web/content/31413934?utm_campaign=HR_recruitment&utm_source=annonce+job&unique=495f86cbf17dad7e465ba2b292b62e3c80d09b1a, (visité le 08/04/2023).

[12] About us odoo. https://www.odoo.com/fr_FR/page/about-us (visité le 08/04/2023).

[13] Investor slides. https://www.linkedin.com/posts/fpodoo_odoo-investor-deck-q1-2023-activity-7059614258992463874-Mz0N/ (visité le 04/05/2023).

[14] Odoo in numbers. https://www.odoo.fm/2108856/12547469-odoo-in-numbers (visité le 03/05/2023).

[15] Odoo experience. https://www.odoo.com/fr_FR/event/odoo-experience-2023-3735/page/oxp23-introduction (visité le 15/04/2023).

[16] Podcast. https://www.odoo.fm/ (visité le 03/05/2023).

[17] Apps odoo. https://www.odoo.com/fr_FR/page/all-apps (visité le 09/04/2023).

[18] App store. https://apps.odoo.com/apps/modules/browse (visité le 09/04/2023).

[19] Comparaison odoo vs sap. https://www.odoo.com/fr_FR/page/compare-odoo-vs-sap#part_14 (visité le 16/04/2023).

[20] Ouverture de bureau en afrique. https://african.business/2022/08/technology-information/software-firm-odoo-opens-first-african-office-in-nairobi (visité le 08/04/2023).

[21] Ouverture de nouveaux bureaux. https://www.lavenir.net/regions/brabantwallon/2022/06/03/odoo-veut-recruter-900-personnes-en-belgique-et-ouvrira-six-nouveaux-bureaux-d (visité le 08/04/2023).

[22] Odoo partners. https://www.odoo.com/fr_FR/partners (visité le 09/04/2023).

[23] Odoo become a partners. https://www.odoo.com/fr_FR/become-a-partner (visité le 09/04/2023).

[24] Odoo.sh. https://www.odoo.sh/ (visité le 16/04/2023).

[25] Different hebergement proposé. https://www.odoo.com/fr_FR/blog/trucs-et-astuces-pour-les-entreprises-1/comment-choisir-votre-type-d-hebergement-560 (visité le 16/04/2023).

[26] Architecture d'odoo. https://odooskills.com/bien-comprendre-architectue-technique-odoo.html (visité le 15/04/2023).

[27] Architecture d'odoo - documentation. https://www.odoo.com/documentation/16.0/fr/developer/tutorials/getting_started/01_architecture.html (visité le 15/04/2023).

[28] Github - documentation. https://docs.github.com/en/get-started/quickstart/hello-world (visité le 22/04/2023).

[29] Storybook - documentation. https://storybook.js.org/docs/react/get-started/why-storybook (visité le 22/04/2023).

[30] Owl - documentation. https://github.com/odoo/owl (visité le 06/04/2023).

[31] Open source. https://fr.wikipedia.org/wiki/Open_source (visité le 22/04/2023).

[32] Sébastien Broca Benjamin Coriat. Le logiciel libre et les communs. *Revue du Logiciel Libre*, 10(2):45–58, 2015.

[33] Open source - avantages inconvenients. https://www.neoproduits.com/entreprises/quels-sont-les-avantages-et-inconvenients-des-logiciels-open-sourc (visité le 22/04/2023).

[34] Github - octoverse. https://octoverse.github.com/ (visité le 22/04/2023).

[35] Trello. https://trello.com/fr (visité le 22/04/2023).

[36] Note sur l'architecture javascript de odoo. https://github.com/ged-odoo/odoo-js-training-public/blob/master/notes_architecture.md (visité le 1/05/2023).

[37] Operateur de chainage. https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Optional_chaining (visité le 20/05/2023).

[38] Azeem Uddin Abhineet Anand. Importance of software testing in the process of software development. *International Journal of Computer Science and Information Security*, 17(2):42–47, 2019.

[39] Charles Pecheur. Software quality assurance - structure testing, 2017.

[40] Coverage: Find unused javascript and css. `https://developer.chrome.com/docs/devtools/coverage/` (visité le 27/05/2023).

[41] Getting started with sphinx. `https://docs.readthedocs.io/en/stable/intro/getting-started-with-sphinx.html` (visité le 10/05/2023).

[42] Lazy loading via attribute for images iframes. `https://caniuse.com/loading-lazy-attr` (visité le 10/05/2023).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**