



IUT Nantes

Pôle Sciences et technologie

Nantes Université

Nantes Université - IUT de Joffre - BUT Informatique

**Rapport de la SAE501
Développement avancé**

R5.Real | Rapport global

sae5-2425-groupe2-medok

Init : OULMAS Thomas, FRÉMONT Tom, MOREAU--THOMAS Nils

Alt : HEQUET Nathan, PRESTI Louis, RAGEAU Nils

3ème année de BUT Informatique - SAE du 1er semestre

Table des matières

Contexte du projet.....	2
Qualité de développement.....	3
Convention de nommage.....	3
Convention hiérarchique.....	3
Convention sur la documentation.....	4
Testabilité.....	4
L'application elle-même.....	4
Exploitation des données.....	6
Management.....	10
Technologies & Organisation.....	10
Évolution.....	12
Projection.....	12
Économie.....	14
Cibles & financeurs de l'application.....	14
Fonctionnalités & valorisation.....	14
Conclusion.....	15
Base de données.....	16
Le contenu de la BD principale.....	16
Nos différentes solutions.....	17
Room.....	17
Objectbox.....	17
Mongodb.....	17
Conclusion.....	17
Vidéo de démonstration.....	18
Annexes.....	18
Trello.....	18
Issue Board.....	18
Labels.....	19
Google Sheet.....	19
Statistiques.....	20
Maquettes de l'application.....	27

Contexte du projet

Nous devons concevoir et développer une application mobile de suivi de traitements médicamenteux et de leurs possibles effets secondaires ou interactions risquées pour les patients. L'application doit permettre une gestion efficace du régime médicamenteux chez le patient, avec déclaration aux autorités des effets indésirables

Our brief was to design and develop a mobile application for monitoring drug treatments and their possible side effects or risky interactions for patients. The application should enable effective management of the patient's medication regimen, with reporting of undesirable effects to the authorities.

This report is only in french as you could've guessed.

les mots en **gras** suivis d'un astérisque* sont liées à une annexes

Qualité de développement

Convention de nommage

Nous avons défini une convention de nommage afin de conserver une cohérence dans les noms de variables, fichiers, fonctions, classes.

Le code doit être écrit en anglais, sauf dans certains cas où cela complexifie la compréhension du code (par exemple des variables nommées `prenom` et `nom`).

L'ensemble du code suit les normes de codage d'Android de Google ([lien](#)).

Nous avons aussi précisé que les noms de composants devaient se terminer par la fonction de celui-ci, par exemple un pop-up d'avertissement (une boîte de dialogue) est défini par le mot `Dialog` en anglais, donc la classe se nommera `WarningDialog`.

Si un composant est une vue, son nom se terminera par `View` et une un champ d'entrée (une `Input`) se terminera par `Field`.

Convention hiérarchique

Nous avons défini une convention de nommage afin de conserver une cohérence dans les dossiers et leurs contenus.

le dossier `src/main` contient alors :

- `data`, un dossier avec ce qui concerne la BD (comme les DAO par exemple)
- `controller`, un dossier qui contient les contrôleurs
- `model`, un dossier qui contient le modèle de l'application
- `ui/theme`, un dossier qui contient les couleurs et le thème de l'application
- `ui/components`, un dossier qui contient les composants de base
- `view`, un dossier qui contient les différentes vues de l'application

le dossier `src/test` contient les tests qui n'ont pas besoin de l'interface de l'application

le dossier `src/androidTest` contient les tests qui ont besoin de l'interface de l'application

Convention sur la documentation

Les composants, s'il sont documentés, doivent suivre le format reconnu par IntelliJ et Android Studio :

```
/**
 * Fenêtre de dialogue (POP-UP) permettant de consulter, cocher et modifier
 * des éléments d'une liste
 * @param selectionState liste des éléments associés à des booléens
 * @param isMutable booléen indiquant si la liste est modifiable (suppression
 * & ajout)
 * @param onOk action à effectuer avec la liste retournée à la fin du dialog
 * (quand l'utilisateur appuis sur Ok)
 * @param onDismiss action à faire si le dialog est annulé (quand
 * l'utilisateur appuis sur Annuler / sur les bords)
 */
fun ListDialog(
    selectionState: MutableMap<String, Boolean>,
    isMutable: Boolean,
    onOk: (MutableMap<String, Boolean>) -> Unit,
    onDismiss: () -> Unit
) {
    ...
}
```

Cela facilite l'identification de la fonction d'un composant et de définir ses paramètres.

Testabilité

L'application elle-même

Dans le code de l'application, lors de la création des tests, il y a eu deux principaux groupes de fonctions : les fonctions composables (annotées `@Composable`), telles que les pages de l'application, les boîtes de dialogue ainsi que les fonctions non-composables qui sont les fonctions n'affectants pas le visuel de l'application.

Cette séparation en deux groupes à eu lieu à cause des difficultés rencontrées lors de la réalisation des tests pour les fonctions composables. En effet ces fonctions ayant prouvées leur difficulté à être testées, nous avons abandonné la création des tests de celle-ci afin de pouvoir progresser dans le projet.

Les fonctions ayant été testées sont : `validatePassword`, `saveSideEffects`, `loadSideEffects`, `saveReminders`, `loadReminders` et `generateRandomCode`.

validatePassword :

Entré	password	"Short1"	X			
		"Password123"		X		
		"Password!123"			X	
		""				X
sortie	oracle	"Le mot de passe doit comporter au moins 8 caractères."	X			X
		"Le mot de passe doit comporter au moins un caractère spécial."		X		
		""			X	

validatePassword est la fonction permettant de vérifier si un mot de passe créé est valide ou non, de PasswordResetDialog.kt.

Si la longueur du mot de passe est inférieure à 8 caractères, alors la fonction renvoie le message "Le mot de passe doit comporter au moins 8 caractères."

Si le mot de passe fait au moins 8 caractères mais ne contient aucun caractères spécial, alors la fonction renvoie le message "Le mot de passe doit contenir au moins un caractères spécial."

Si le mot de passe fait au moins 8 caractères et contient au moins 1 caractère spécial, le mot de passe est valide donc la fonction renvoie un message vide.

saveSideEffects :

Entré	sideEffects	SideEffects valides	X	
		SideEffects invalides		X
Sortie	Oracle	SideEffects enregistrés	X	
		Erreur		X

saveReminders :

Entré	rappels	Reminders valides	X	
		Reminders invalides		X
Sortie	Oracle	Reminders enregistrés	X	
		Erreur		X

Les fonctions saveSideEffects et saveReminders permettent respectivement d'enregistrer en format Json dans sharedPreferences les effets secondaires et les rappels médicaux. Si les données reçues ne sont pas valides, alors une erreur est renvoyée, sinon les effets secondaires / les rappels médicaux sont enregistrés.

loadSideEffects :

Entré	context	sharedPreferences Json valide	X	
		sharedPreferences Json corrompue		X
Sortie	Oracle	SideEffects chargés	X	
		Erreur		X

loadReminders :

Entré	context	sharedPreferences Json valide	X	
		sharedPreferences Json corrompue		X
Sortie	Oracle	Reminders chargés	X	
		Erreur		X

Les fonctions loadSideEffects et loadReminders permettent respectivement de charger les effets secondaires et les rappels médicaux déjà enregistrés. Si les fichiers Json enregistrés sont des effets secondaires / rappels médicaux valides, alors les données sont lues et renvoyées en une liste de SideEffectModele / MedicalReminderModele. Si les données Json ne sont pas conformes, alors une erreur est renvoyée. Cette fonction ne prend en argument que le contexte de l'application afin d'accéder aux sharedPreferences, c'est pourquoi en entré on test avec différents fichiers Json enregistrés dans sharedPreference, ce n'est pas le context en lui même qui change pour les tests.

generateRandomCode :

Sortie	Oracle	code d'une longueur de 6 caractères	X
		code composé uniquement de valeurs numériques	X

La fonction generateRandomCode, permet de générer un code aléatoire de 6 chiffres. Il n'y a pas d'argument pour cette fonction, on s'assure donc juste que le code réponde bien aux attentes.

Il y a d'autres fonctions qui auraient méritées d'être testées mais fautes de temps (et de connaissance), les fonctions @Composable n'ont pas été testées. Ces fonctions sont celles qui gèrent le visuel de l'application, elles sont donc tout autant importantes que les précédentes mais surtout, elles représentent la majorité des fonctions de l'application.

Exploitation des données

Notre projet est structuré en deux principaux dépôts Git :

- **App** : Contient l'application elle-même.
- **Server** : Inclut notamment une base de données MongoDB stockant les données volumineuses telles que les informations des professionnels de santé ou des médicaments.

L'application exploite les données du serveur qui est fait avec Node.js/express, via des requêtes GET sur son API. Il était donc important de tester les différentes routes, à la fois côté serveur, pour vérifier que les fonctions renvoient les bonnes données, et côté application, afin de s'assurer que l'API s'intègre correctement et ne génère pas d'erreurs côté serveur.

Pour les tests suivants, nous nous concentrerons principalement sur la liaison avec les informations des pharmaciens, les requêtes sur la BD suivent toutes un schéma similaire (principalement des recherches par champ spécifique avec `findByX`) et c'est également l'implémentation qui est selon nous la plus complète.

Nous avons commencé par tester la partie serveur, étant celle qui allait être exploitée par l'application par la suite.

La collection contenant les informations des pharmaciens possède les fonctions suivantes:

- **populate()** -> (inaccessible par l'application) S'exécute au lancement du serveur pour remplir la base de données via les données sources (provenant de data.gouv.fr).
- **findAll(page, size)** -> Renvoie tous éléments.
- **findById(id_API)** -> Renvoie les données du pharmacien correspondant à l'ID.
- **findByPrenom(prenom_API, page, size)** -> Renvoie les données des pharmaciens ayant le même prénom qu'en paramètre.
- **findByNom(nom_API, page, size)** -> ...
- ...

La plupart de ces fonctions possèdent également les paramètres "page" et "size". En effet, nous nous sommes rendu compte assez tôt dans le développement de l'API que si l'application demande un trop gros volume de données, cela lui causerait une `OutOfMemoryException`.

La solution que nous avons trouvée était donc d'ajouter un système de pagination aux requêtes susceptibles de renvoyer plusieurs résultats. Cela nous permet d'avoir plus de contrôle sur la quantité de données reçues mais ajoute un peu plus de complexité aux fonctions en contrepartie.

Le paramètre "size" indique donc la taille de chaque page tandis que "page" indique celle que nous souhaitons récupérer.

Nous avons pu vérifier les problèmes que la pagination allait engendrer au travers de tests unitaires réalisés à l'aide du framework Mocha. Voici la table de décision réalisée pour la partie serveur:

Entrées	page	[min, 1[X		X	X
		[1, pageMax]					
]pageMax, max]					
	size	[min, 1[X	X	X
		[1, max]					
	MongoPharmacien	launched	X	X	X	X	X
!(launched)							
Sortie	Oracle	data				X	␣
		MongoServerError		X	X		
		Timeout	X				

(Version finale, prenant en compte les erreurs rencontrés après les tests unitaires)

Le problème principal que nous avons rencontré lors de ces tests était leur conception elle-même. En effet, nous n'avions pas de fonction insert à notre disposition puisqu'elle n'a pas d'utilité dans le programme de base. La seule façon aurait été via l'utilisation de populate(), mais il y a tellement de données dans le fichier source que chaque exécution des tests aurait pris ~45 secondes. Notre solution a donc été de réaliser un stub de la fonction populate avec la bibliothèque "sinon" afin de pouvoir lui faire ajouter des données tests très facilement.

Avec nos tests, l'un des problèmes que nous avons pu observer était le fait que nous pensions qu'une taille de page de valeur 0 était valide, cependant, ce n'était pas le cas et causait une erreur serveur.

De manière opposée, nous pensions que récupérer une page supérieure à la dernière contenant des données causerait une exception, mais ce n'était au final pas le cas et retourne à la place toujours un tableau vide.

Pour ce qui est de la partie application, le but était de vérifier qu'elle pouvait bien communiquer avec le serveur avec ses différentes fonctions faisant des requêtes sur les différentes routes du serveur.

Nous avons testé cette partie en 2 étapes, sur 2 fichiers tests, la première était de mockker le serveur. Cela nous permettait de retirer cette dépendance pour nous concentrer plutôt sur les précautions mises en place afin de ne pas laisser l'application utiliser des valeurs interdites.

Voici la table de décision réalisée pour la fonction findByPrenom de la partie application (prise en exemple car c'est l'une des fonctions avec le plus de paramètres):

Entrées	page	[min, 1[X		
		[1, max]					X
	size	[min, 1[X	
		[1, max]					X
	MongoPharmacien	launched	X		X	X	X
		!(launched)		X			
Sortie	prenom	emptyString	X				
		!(emptyString)		X	X	X	X
		data					X
	Oracle	IllegalArgumentException	X		X	X	
		HttpRequestTimeoutException		X			

La différence principal est la prise en compte du paramètre prénom, la raison pour laquelle on la retrouve ici et pas sur la partie serveur est parce que nous nous sommes rendu compte à l'aide des tests que si cette chaîne de caractère est vide, lorsque la requête vers la route (sous la forme `/api/v0/pharmacien/prenom/$prenom`) est lancée, cela cause un "/" vide qui traîne à la fin et le serveur n'arrive pas à GET par conséquence.

Pour la seconde partie, nous avons créé un second fichier de tests exploitant cette fois-ci le vrai serveur avec des tests en plus pour vérifier les cas plus niche (par exemple la recherche par prénom ne prend pas en compte la casse).

Cela nous permet de garantir que le serveur s'intègre correctement avec le reste de l'application, il pourra également être utilisé dans le futur car actuellement, nous faisons tourner le serveur en local mais nous comptons très bientôt le déployer sur une machine virtuelle de l'IUT et il nous permettra donc de vérifier directement si tout se passe bien.

Management

Technologies & Organisation

Nous avons donc réalisé ce projet en Kotlin, en utilisant Android Studio, car nous étions familier avec cela. Le problème étant que cela restreint l'application aux téléphones Android uniquement ce qui fait que le côté multi-plateforme désiré n'est pas possible dans notre cas.

l'OCR utilisé pour lire les ordonnances est `mLkit` de Google, la raison étant que Tesseract (un autre choix pour l'OCR) est certes open-source, mais il est aussi plus compliqué à manipuler. mais il vaut mieux quelque chose de fonctionnel qu'open-source.

Pour la base de données, nous avons choisi un fonctionnement hybride :

- En local pour les profils et les données de l'application lié à ceux-ci.
- Sur serveur pour les données des médicaments et des pharmaciens

Tout mettre en local serait très coûteux en mémoire à cause de la quantité massive de données, néanmoins notre solution actuelle ne permet pas un fonctionnement hors connexion. L'application fonctionne quand même sans connexion, mais avec des fonctionnalités comme l'annuaire des pharmaciens désactivées. Voici un tableau comparatif pour chaque côté :

Solution	+ Avantages	- Inconvénients
Entièrement sur Serveur	Réduit la place prise par l'application sur l'appareil de l'utilisateur Facilite la mise à jour des parseur de PDF et de données (car nous avons la main dessus)	Non-respect de la confidentialité des profils utilisateurs Doit toujours être actif et donc maintenu par nous Connexion requise
Entièrement en Local	Fonctionne hors connexion Respect de la confidentialité des profils utilisateurs	Complexifie la mise à jour des parseur de PDF et de données (car l'utilisateur doit mettre à jour son application) L'application prendra plus de place sur l'appareil de l'utilisateur
Hybride	Respect de la confidentialité des profils utilisateurs Réduit la place prise par l'application sur l'appareil de l'utilisateur (contient uniquement les profils et leurs données associées) Facilite la mise à jour des parseur de PDF et de données (car nous avons la main dessus)	Le serveur doit toujours être actif et donc maintenu par nous Ne fonctionne pas à 100% hors connexion

Nous avons créé un compte sur notre VM pour le serveur, qui contient diverses données et api, et pour le Jenkins, néanmoins ce dernier n'a pas été utilisé par manque de temps et de tests (ceux-ci sont arrivés tard).

Pour s'organiser tout au long du projet, nous avons commencé par utiliser **Trello***, un outil à base de tableaux Kanban, car certains membres de notre équipe étaient familiers avec celui-ci. Cependant après réflexion nous nous sommes dirigés sur **l'Issue Board*** de Gitlab.

À chaque issue, il y a une branche, les changements la concernant seront donc faits là-bas, ensuite après validation de la merge request sur main l'issue est déclarée comme "Closed" et la merge request effectuée.

Nous avons utilisé les milestones de gitlab afin de savoir l'avancement global du projet et dans quelle catégorie se situait telle ou telle issue. La milestone "Version de base" contient ce qui est demandé par l'application, "Version améliorée" contient ce qui sera ajouté à posteriori.

Les **labels du Gitlab*** ont été utilisés pour déterminer sur une Issue :

- Sa priorité (très haute, haute, moyenne, faible)
- Son statut (En cours, À valider, Abandonné)
- Si c'est un bug
- Sa catégorie (BD, UI/UX, Appli mobile)

Nous avons scindé le projet en 3 repository Gitlab : OCR, server et app.

Le repo OCR est un repo de test qui servait à créer l'OCR, mais ne sachant pas s'il était possible de le faire ou non, un git à part entière a été créé pour ne pas interférer avec le git de l'application mobile.

Le repo app est le repo qui contient l'application mobile, avec la branche main pour la production et les branches issues comme des branches dev.

Le repo server est le repo qui contient le serveur, soit l'API qui permet de récupérer diverses données (sur les pharmaciens par exemple) et aussi un début de parseur du PDF d'interaction médicamenteuse.

Nous avons appliqué la méthode SCRUM où chaque Sprint avait une durée variable selon la charge de travail de la semaine, si il n'y avait pas beaucoup d'avancement global dans la semaine alors celui-ci était étendu d'une unique semaine.

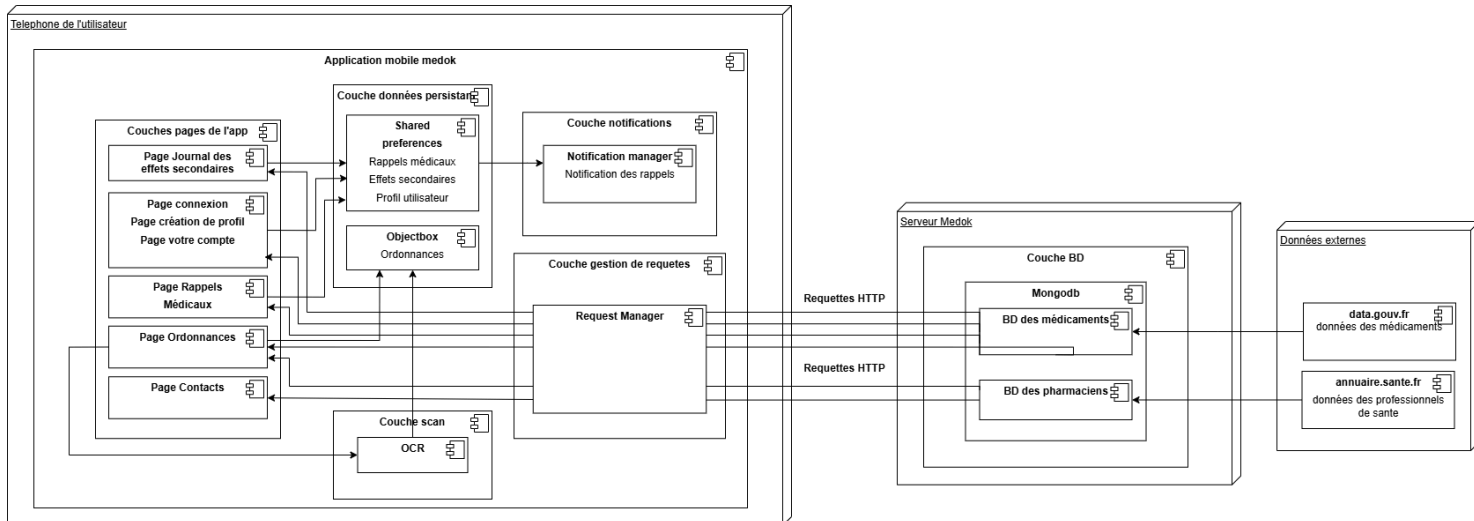
Les sprints durent entre 1 et 2 semaines, les vacances scolaires n'étant pas comprises.

Les réunions avaient lieu chaque fin de semaine ou nous nous réunissions sur Discord pour faire un appel vocal groupé, nous avons choisi Discord car tout le monde dans l'équipe l'utilise.

Les avancées de chacun étaient écrites dans un **Google Sheet***, celui-ci a évolué au fil du temps au vu des différents besoins (marquer une tâche comme abandonnée, avoir plus de lisibilité...)

Le SCRUM Master et le Product Owner varient pour chaque sprint afin de ne pas être las et que chacun puisse animer les réunions.

Et voici comment le projet était organisé au niveau des couches : [Diagramme de déploiement de l'application Medok](#)



Évolution

Comme mentionné précédemment, nous sommes passés de Trello à Gitlab, et notre Compte rendu à évolué au fil du temps pour le rendre plus pratique.

Pour ce qui est du Discord et du Gitlab, nous avons des **statistiques*** globales et pour chaque utilisateurs disponibles en annexes.

Nous avons fait une première **maquette de l'application*** cependant celle-ci n'était pas très claire et belle, alors une **deuxième maquette*** a été faite qui clarifie tous les besoins de l'application et avec un meilleur rendu, ci qui à motivé l'équipe à travailler.

Projection

Pour améliorer l'efficacité du projet, nous aurions dû imposer un temps minimum travaillé sur le projet par semaine et communiquer clairement sur les blocages.

Pour améliorer la cohésion de l'équipe, écrire les conventions de nommage et hiérarchiques dès le début du projet aurait été d'une grande aide.

Nous aurions dû aussi utiliser le Time Tracking de gitlab pour avoir une meilleure visibilité sur quelles tâches l'équipe bloquent, ou si la tâche a été sous-estimée.

Pour avoir un fonctionnement garanti, il aurait fallu écrire les tests au fur et à mesure et non vers la fin du projet.

Pour avoir un départ vif, les technologies auraient dû être établies dès le départ et non pendant deux mois entiers.

Pour finir, nos objectifs aurait pû être moins exigeant en vue de la contrainte de temps et notre manque de pratique sur certains points.

Dans le futur, l'application aura un Jenkins pour exécuter les tests automatiquement et garantir un bon fonctionnement.

Les interactions médicamenteuses, l'annuaire de contact des pharmaciens et la gestion de profils utilisateurs seront aussi implémentés et l'application sera multi-plateforme.

Économie

Cibles & financeurs de l'application

L'application a pour but d'aider des malades dans leur suivi de traitement, de prise de médicament. La cible de l'application est donc les patients suivant un traitement nécessitant une prise régulière de médicaments.

Cependant l'application peut aussi s'adresser au proches du ledit patient.

Notre application permet la création de profil pour pouvoir gérer une même famille sur un appareil. Ce qui signifie qu'un parent, ne suivant pas forcément de traitement, peut utiliser l'application pour gérer celui de son enfant.

Dû aux contraintes techniques de l'application, seules les personnes qui possèdent un téléphone Android sous l'API 35 ou plus sont les personnes qui pourront utiliser correctement l'application. L'utilisation de celle-ci en dessous de l'API 35 n'est pas garantie.

Au niveau du financement, nous comptons sur un investissement, sur base régulière, d'un organisme comme l'OMS ou une autre instance médicale et sanitaire. l'intérêt étant que notre application limite les risques d'effets secondaires dû à un conflits entre deux traitements (deux médicaments qui réagissent mal ensembles). Il y a donc un intérêt sanitaire.

L'application permet aussi de facilement identifier des possibles conflits médicamenteux grâce aux utilisateurs qui enregistrent des effets ressentis après la prise de traitements. Les laboratoires peuvent donc y voir un intérêt afin d'identifier une défaillance dans leurs médicaments.

Le financement ne se fera pas sur la base d'un abonnement utilisateur, ni par pub (même non ciblée), car l'application est destinée à améliorer la rigueur sur les traitements médicaux et non à générer du profit.

Fonctionnalités & valorisation

L'application permet d'enregistrer des rappels médicaux, potentiellement depuis une ordonnance préalablement scannée. Ces rappels permettent au patient de suivre son traitement avec rigueur pour garantir son rétablissement.

L'application permet aussi d'enregistrer un effet secondaire ressentis après avoir suivi le traitement de X pathologies avec Y médicaments.

Cela permettra d'identifier un potentiel défaut sur le/les médicaments. Les laboratoires ont donc, comme énoncé dans la partie précédente, un intérêt financier car cela permet d'éviter un potentiel scandale sanitaire (ou du moins l'atténuer). Ce qui amène aussi un intérêt stratégique car cela démontre que le laboratoire s'efforce de maintenir une bonne qualité dans ses produits.

L'application permettra aussi au patient de rechercher un praticien, ce qui lui permettra de rentrer en contact avec le praticien pour rendre compte du suivi de son traitement. Comme ça le praticien est capable de suivre l'état de son patient et des potentielles irrégularités. Cela permet au praticien de garantir l'efficacité de ses prescriptions.

Conclusion

En conclusion, l'application sera financée par le biais d'investissements réguliers de la part de laboratoires pharmaceutiques et d'organismes sanitaires, comme l'OMS. Ces financeurs y voient un intérêt car cela permet de garantir la qualité d'un médicament et d'identifier les potentielles failles, intéressant pour les laboratoires, et d'éviter une crise sanitaire, intéressant pour l'OMS.

L'application est à destination des patients ayant un traitement à suivre régulièrement, mais aussi les proches de ces patients.

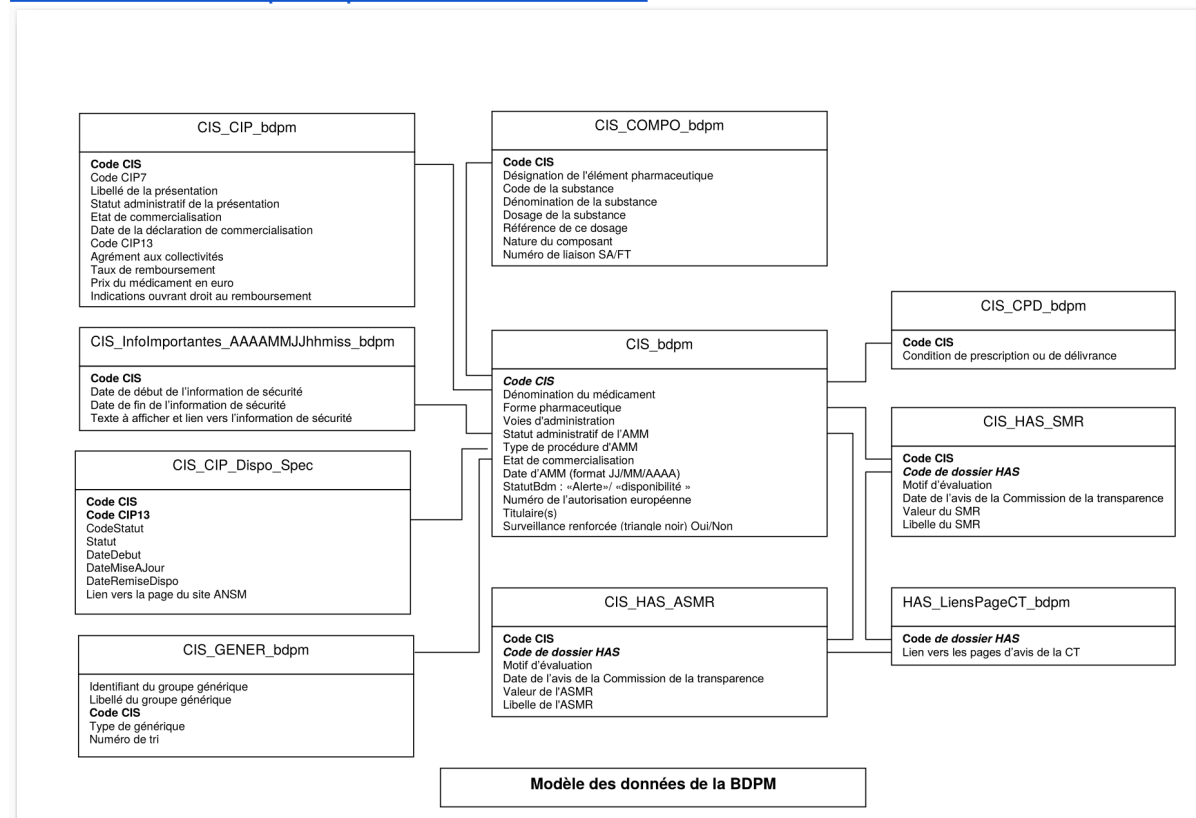
Les médecins traitant y voient aussi un compte, car cela permet d'identifier un potentielle manque d'efficacité du traitement si celui-ci s'éternise. et d'un point de vue déontologique, cela garantit leur devoir de soigner leurs patients (ou du moins d'avoir tout mis en oeuvre pour)

Base de données

Le contenu de la BD principale

Notre base de données contient les informations des médicaments telles que, leurs noms, leurs présentations (boîtes de médicaments), leurs compositions, leurs conditions de prescription, etc.

Base de données publique des médicaments



L'une des fonctionnalités que nous voulions pour notre application était de pouvoir permettre à un utilisateur de contacter un pharmacien afin de pouvoir obtenir des renseignements sur ses prescriptions, notre interface comprend donc un onglet où il peut rechercher un pharmacien via son nom ou ses coordonnées afin de récupérer l'e-mail ou le numéro de téléphone qui lui sont associés.

Pour ce faire, nous avons utilisé une base de données comportant des informations sur les professionnels de santé en France qui sont publiques.

Elle contient : l'identifiant du professionnel de santé, sa civilité, son nom et prénom d'exercice, la catégorie professionnelle, sa profession, les coordonnées des structures d'exercice ainsi que le numéro de téléphone et l'e-mail associé.

Base de données des professionnels de santé

Nos différentes solutions

A l'origine, nous voulions avoir la base de données en local, c'était un choix qui était entièrement pris par notre groupe, n'étant pas vraiment mentionné dans le sujet de la SAE. Nous avons fait ce choix car nous estimions qu'une application de rappel médical pouvait être assez importante pour se devoir d'être disponible même hors connexion.

Room

La bibliothèque Room a été notre première solution, elle fait partie de l'architecture Android et correspondait parfaitement à notre contexte où nous souhaitions une BD entièrement locale et accessible hors ligne. Cette base de données est en relationnel (room étant construit sur du SQLite) et permet la persistance de données entre les fermetures d'application.

Cette solution semble bien sur le papier mais nous avons implémenté celle-ci avant que la partie BD de la SAE nous soit présentée officiellement et ne respectait au final pas la consigne qui était d'utiliser une base de données en non-relationnel.

Objectbox

La bibliothèque Objectbox nous a donc ensuite semblée comme étant une bonne alternative, car elle est orientée objet (impliquant du non-relationnelle, les données n'étant pas stockées sous la forme de tables et de lignes) et aussi permettait d'avoir la base de données en locale.

À nouveau cette alternative paraissait intéressante mais après avoir mis en place une implémentation puis consulté M.Nachouki, il nous a été plutôt conseillé de partir sur la solution mongodb, car il s'agit d'un langage que nous avons vu en cours.

Mongodb

Du Mongodb orienté graphe est donc notre solution actuelle, il s'agit d'une base de données NoSQL normalement orientée document, mais la possibilité "d'imbriquer" des documents les uns dans les autres permet de garder un semblant de relation, nécessaire pour représenter les médicaments.

L'un des désavantages était que cette implémentation nous a contraint à séparer cette BD vers un serveur externe où l'application fait des requêtes API au serveur afin de communiquer avec, nous faisant abandonner le fait de stocker entièrement en local la BD. Cependant, le gros avantage est que l'application devient beaucoup moins lourde.

Conclusion

Notre BD côté serveur contient donc toutes les données que nous avons défini comme faisant partie de la Base de données principale dans la partie [Le contenu de la BD principale](#).

C'est-à-dire, les médicaments ainsi que les données des professionnels de santé sur lesquels nous avons appliqué un prétraitement de façon à ne garder que les pharmaciens afin de réduire le nombre de données qui était très conséquent autrement.

Nous avons gardé une partie des données en local que nous n'avons pas évoquées dans ce rapport tel que les profils des utilisateurs ainsi que leurs rappels, ordonnances et journal des effets secondaires qui sont des données privées et par sécurité, nous nous devons de les garder en local.

Vidéo de démonstration

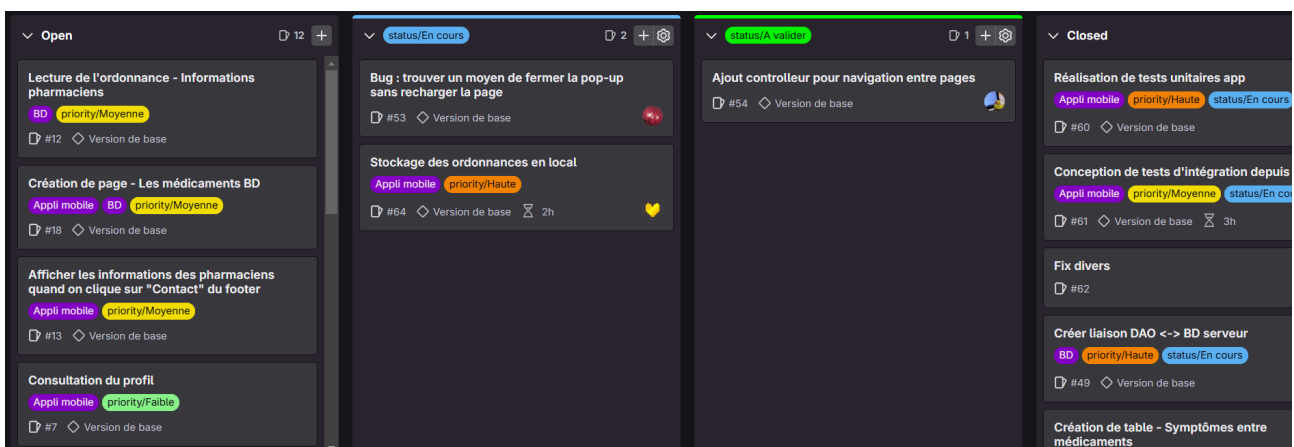
Voici le lien vers ladite vidéo : <https://youtu.be/Q-aAuDHKywc>

Annexes

Trello



Issue Board



Labels

Appli mobile 📱 sae5-2425-groupe2-medok / app	priority/Faible 📱 sae5-2425-groupe2-medok / app	
BD 📱 sae5-2425-groupe2-medok / app	priority/Haute 📱 sae5-2425-groupe2-medok / app	status/A valider 📱 sae5-2425-groupe2-medok / app
Bug 📱 sae5-2425-groupe2-medok / app	priority/Moyenne 📱 sae5-2425-groupe2-medok / app	status/Abandonné 📱 sae5-2425-groupe2-medok / app
UI/UX 📱 sae5-2425-groupe2-medok / app	priority/Très haute 📱 sae5-2425-groupe2-medok / app	status/En cours 📱 sae5-2425-groupe2-medok / app

Google Sheet

1ère version du compte rendu

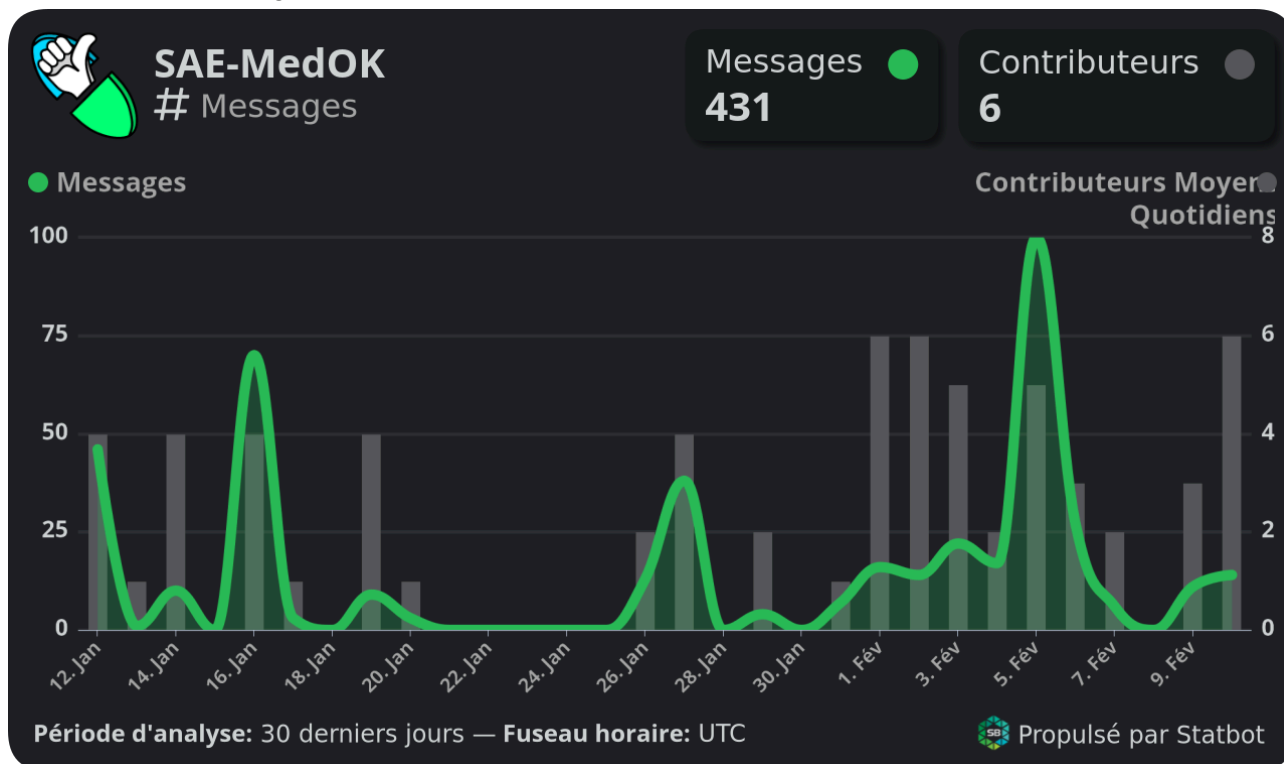
	Compte-Rendu de fin de sprint du 07/10 au 13/10				
	Product Owner : Louis P. ▾	SCRUM Master : Nils MT. ▾			
	Version : 0.0.0	Release fonctionnelle : Oui ▾	pas de bug & projet vide		
	Tâches en cours			Tâches terminées	Retour client
Louis P.	- #15 créer la base de donnée (les relations)			- #6 nettoyage des données fait et mis sur git et au bon encodage (les scripts de nettoyages sont dispo dans le git) - restructure du projet (pour avoir android dans un dossier)	
Nathan H.	- #2 création des pages du mdp oublié				
Nils MT.	- #14 dialogue de création et de modification de rappel			- #14 vue d'affichage des rappels - #4 création de l'identité visuelle - aide à la création et l'organisation du figma	
Nils R.	- #5 création des vues de connexion et creation de compte			- setup des issues (pour que Thomas puissent les utilisées)	
Thomas O.				- transfert des tâches de trello vers les issues de gitlab	
Tom F.	- #14 optimisation des pages (réduire le nombre) - #14 réactions médicamenteuses			- #14 journal effet secondaire, - #14 messagerie avec les pro de la santé - #14 pages de connexion - #14 médicament enregistré/catalogue de médoc	
	Bug fix en cours			Bug fix terminées	Retour client
Louis P.					
Nathan H.				- #3 gradle qui marchais pas	
Nils MT.					
Nils R.				- #3 gradle qui marchais pas - les gitignore qui était pas bon avec la restructure du projet	
Thomas O.					
Tom F.					
	Notes additionelles				
	Prochain sprint : 1 semaines				
	Décision sur la BD : SQLite (parce que gratuit, customisable et le SQL on connaît, et utilisable localement hors-ligne)"				

2ème version du compte rendu

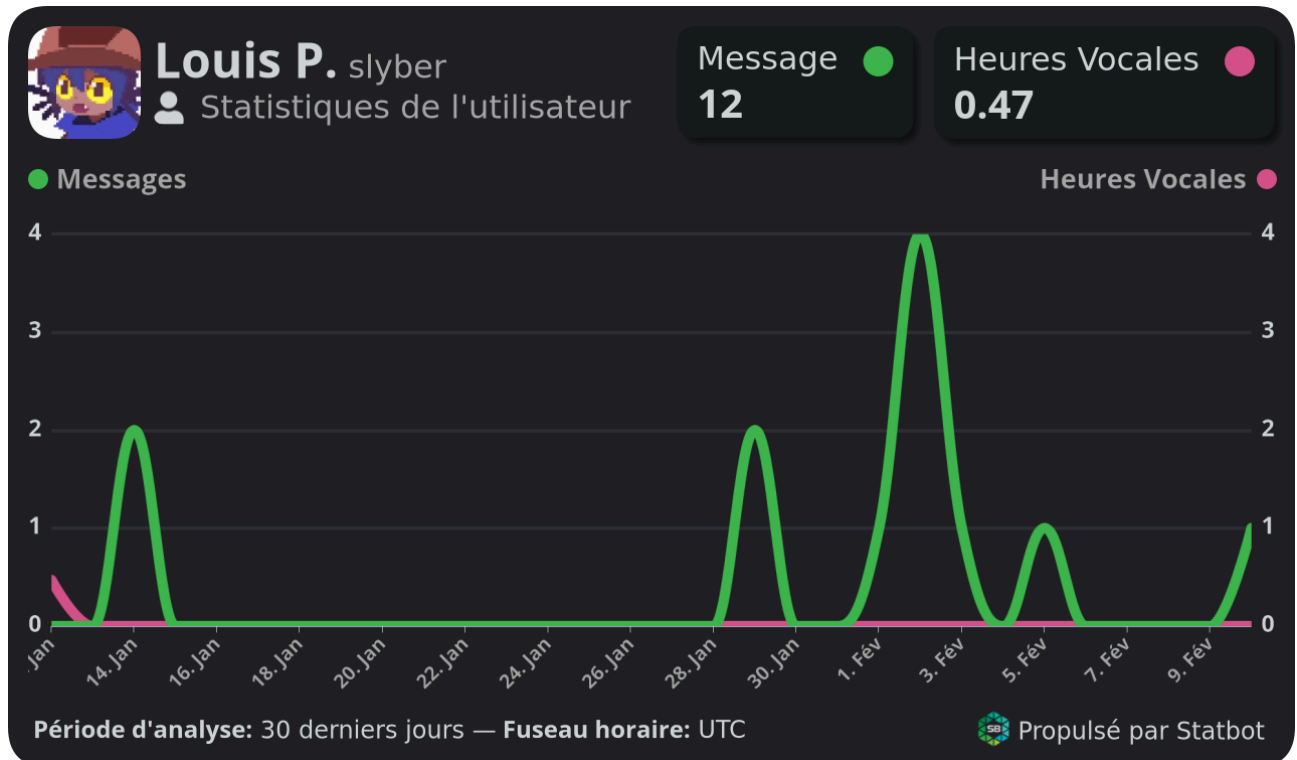
Compte-Rendu de fin de sprint du 03/02 au 09/02						
Product Owner :	Tom F.	SCRUM Master :	Thomas O.	Next Scrum Master :	Tom F.	
Version :	0.0.0	Release fonctionnelle :	Oui	Next Product Owner :	Thomas O.	
Membre	Type	Status	Coté	Description	Note	
Tom F.	Tâche	Terminé	app	Faire un diagramme de déploiement et de composants & connecteurs		
Tom F.	Tâche	Terminé	app	Faire des tests coté app		
Tom F.	Tâche	Étendu sur ce sprint	app	42 - Faire la page des contacts		
Tom F.	Tâche	En cours	app	63 - Test de model Email et PhoneNumber		
Thomas O.	Tâche	Terminé	app	61 - Conception de tests pour vérifier l'intégration de l'API sur l'appli	Des tests avec un serveur mocké + avec le vrai serveur s'il est online	
Thomas O.	Tâche	Terminé	app	64 - Faire sauvegarder les ordonnances en local		
Thomas O.	Tâche	Étendu sur ce sprint	app	Faire transvaser le stockage de données à un format de données plus adapté comme objectbox	Délai causé par la conception de tests qui était au final plus importante dans l'immédiat	
Nils R.	Tâche	Étendu sur ce sprint	app	Implémentation des alertDialog dans les controllers	Les alternants seront en entreprise	
Nils MT.	Bug	Terminé	app	62 - Fix divers et variés		
Nils MT.	Tâche	Terminé	autres	Faire diapo présentation appli		
Nils MT.	Tâche	Terminé	autres	Rapport d'économie		
Nils MT.	Tâche	Étendu sur ce sprint	app	Mise en place de la communication côté application vers la BD sur les DAOs restant		
Nils MT.	Tâche	Étendu sur ce sprint	server	Mettre serveur sur VM	+ adapté l'appli (RequestManager) en conséquence	
Nils MT.	Tâche	En cours	autres	Vidéo démo app		
Nils MT.	Tâche	En cours	autres	Rapport de qualité de dev	reste la partie à Thomas	
Nils MT.	Tâche	En cours	autres	Rapport de dev avancé		
Nils MT.	Tâche	En cours	autres	Rapport global		
Nathan H.	Tâche	Étendu sur ce sprint	app	(42 - app) - Création d'un utilisateur en format json	Les alternants seront en entreprise	
Louis P.	Tâche	Étendu sur ce sprint	server	Rassembler les fonctions des DAO dans le DAO abstract	Les alternants seront en entreprise	

Statistiques

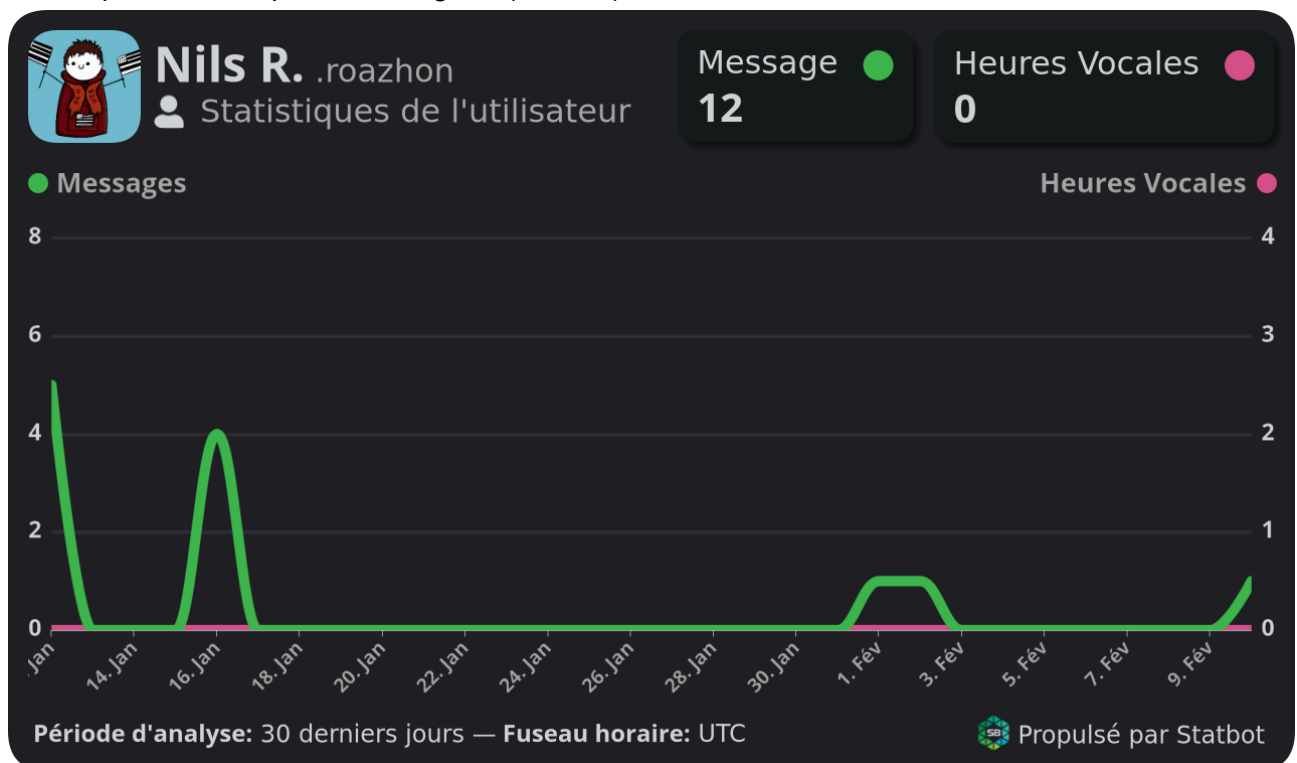
Statistiques Discord globales



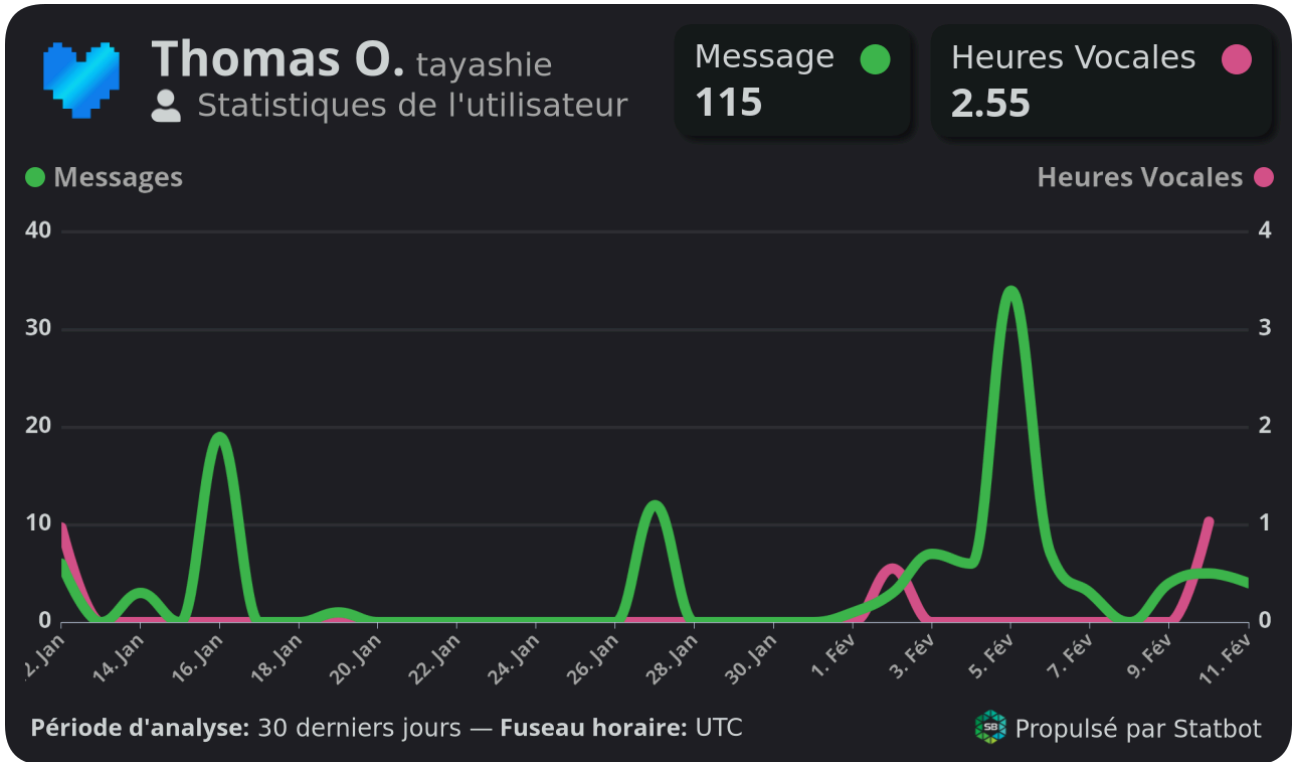
Statistiques Discord pour Louis Presti (sur 30J)



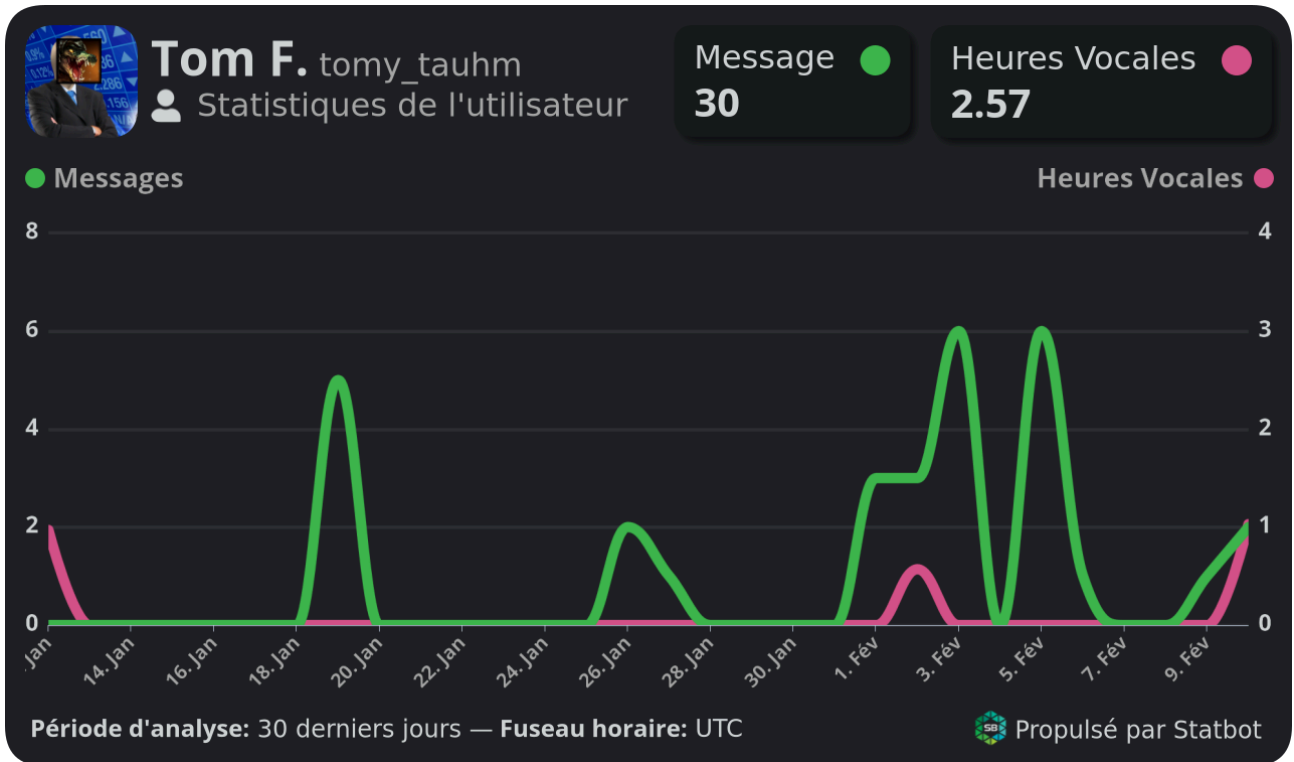
Statistiques Discord pour Nils Rageau (sur 30J)



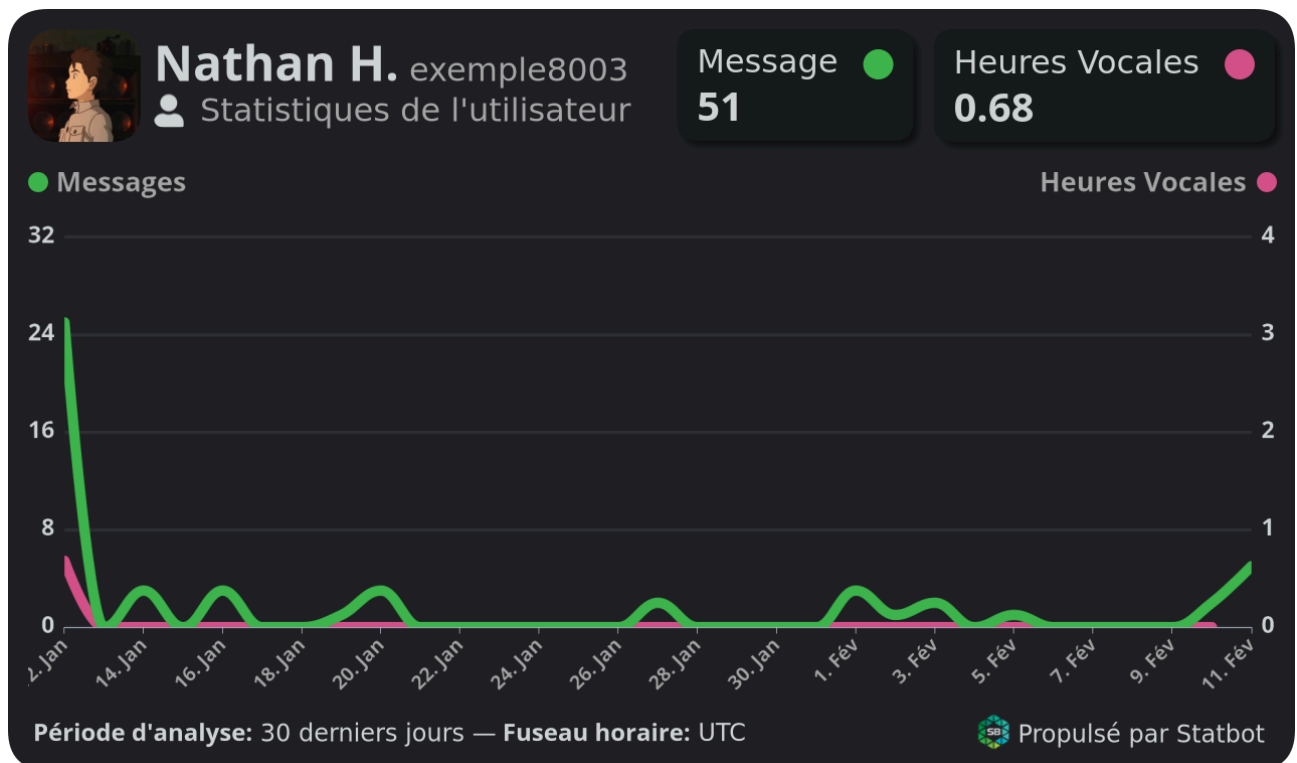
Statistiques Discord pour Thomas Oulmas (sur 30J)



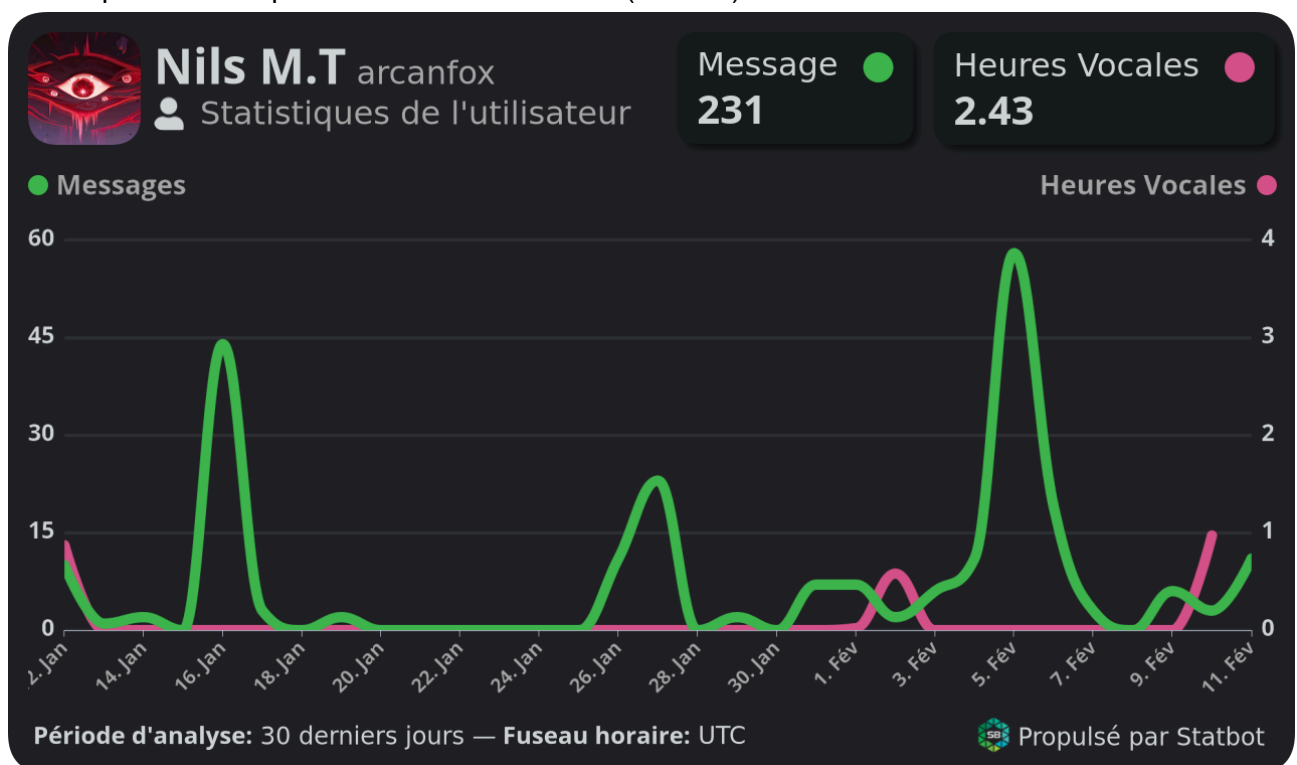
Statistiques Discord pour Tom Fremont (sur 30J)



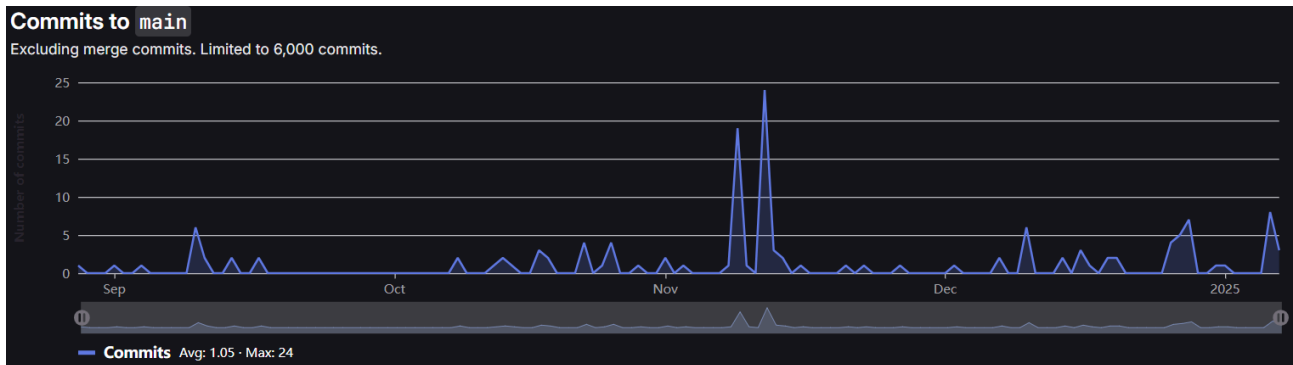
Statistiques Discord pour Nathan Hequet (sur 30J)



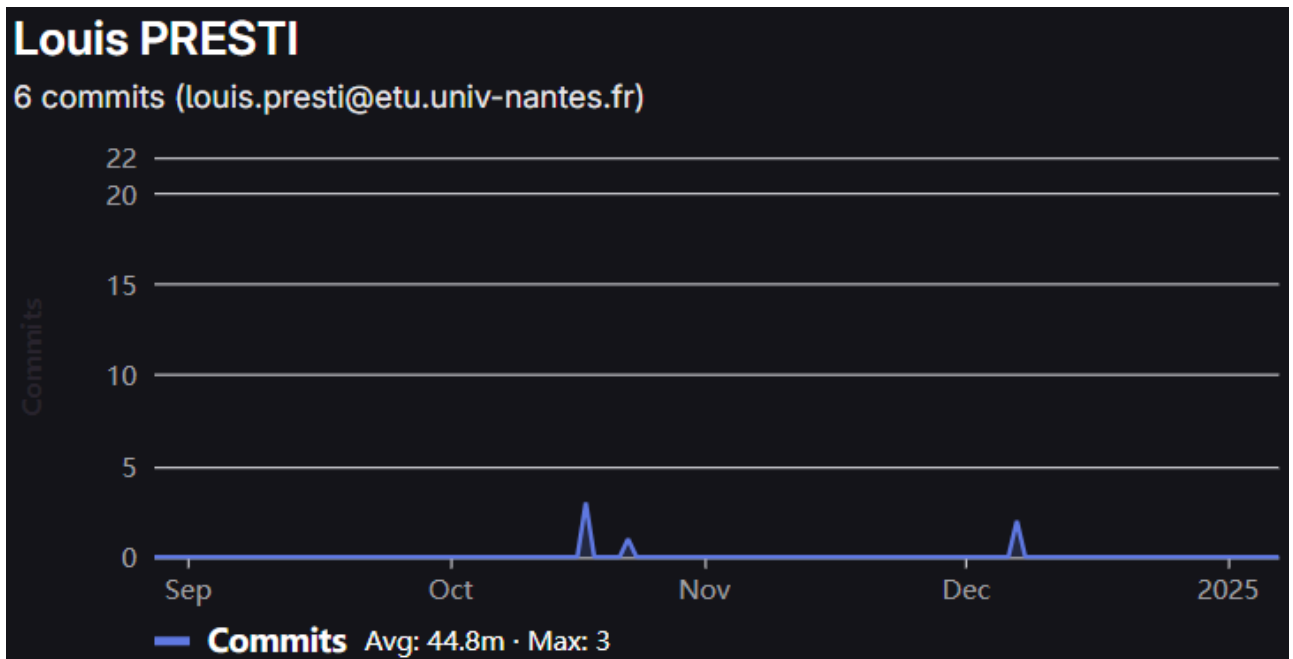
Statistiques Discord pour Nils Moreau–Thomas (sur 30J)



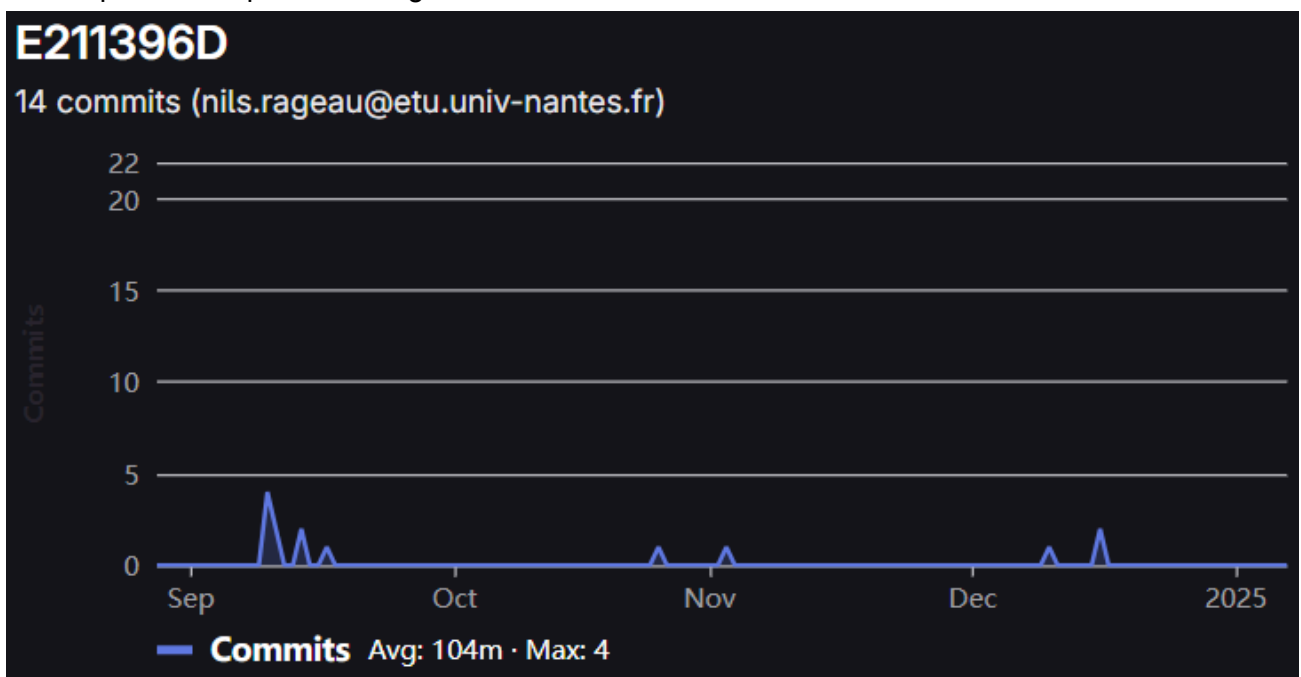
Statistiques Gitlab globales



Statistiques Gitlab pour Louis Presti

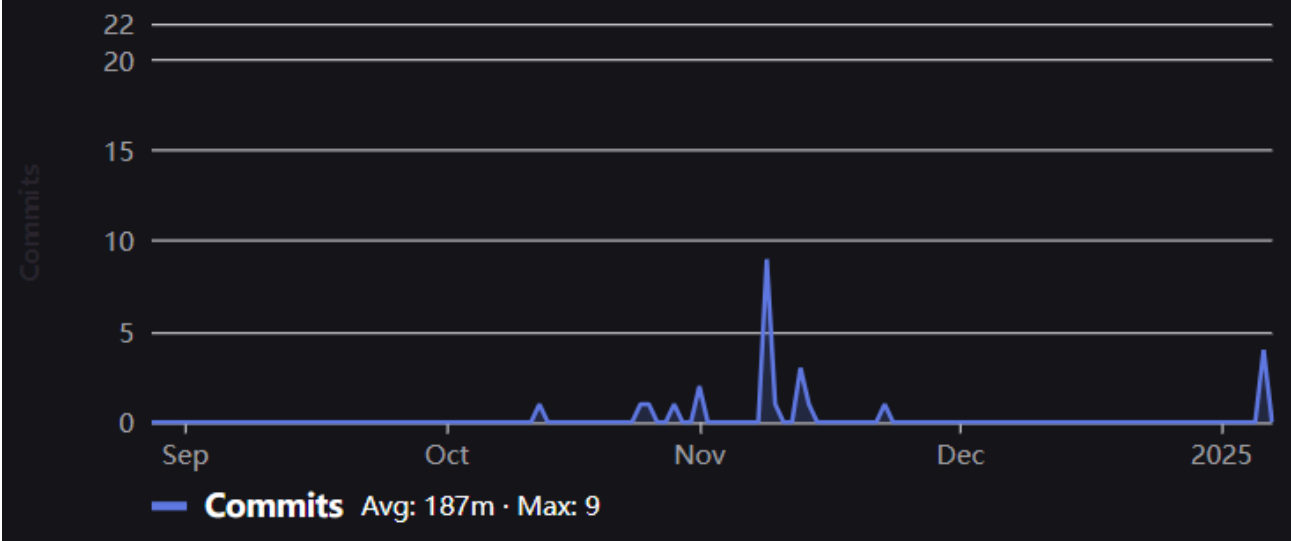


Statistiques Gitlab pour Nils Rageau



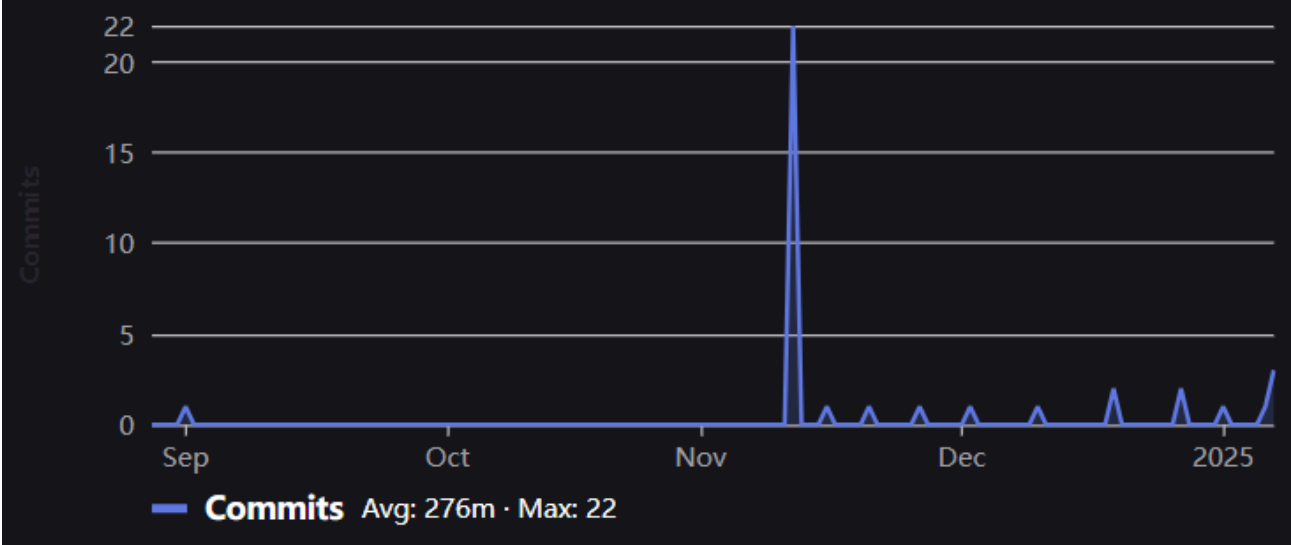
E225413P

25 commits (thomas.oulmas@etu.univ-nantes.fr)

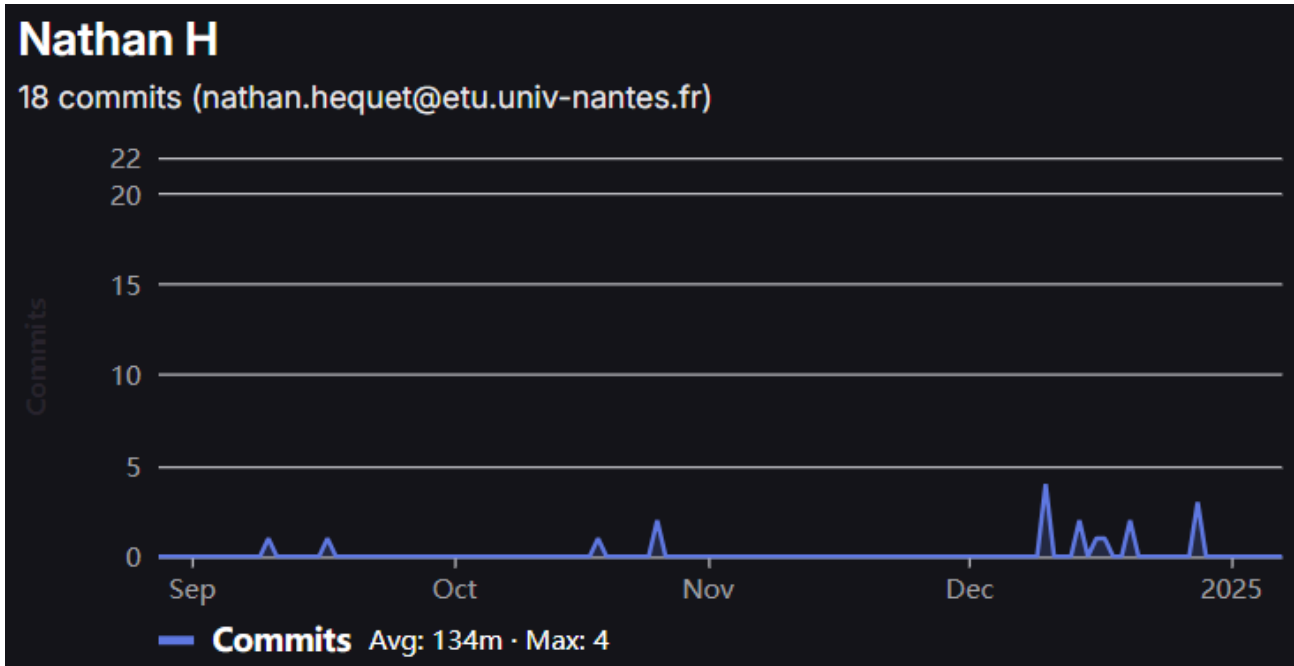


E226954P

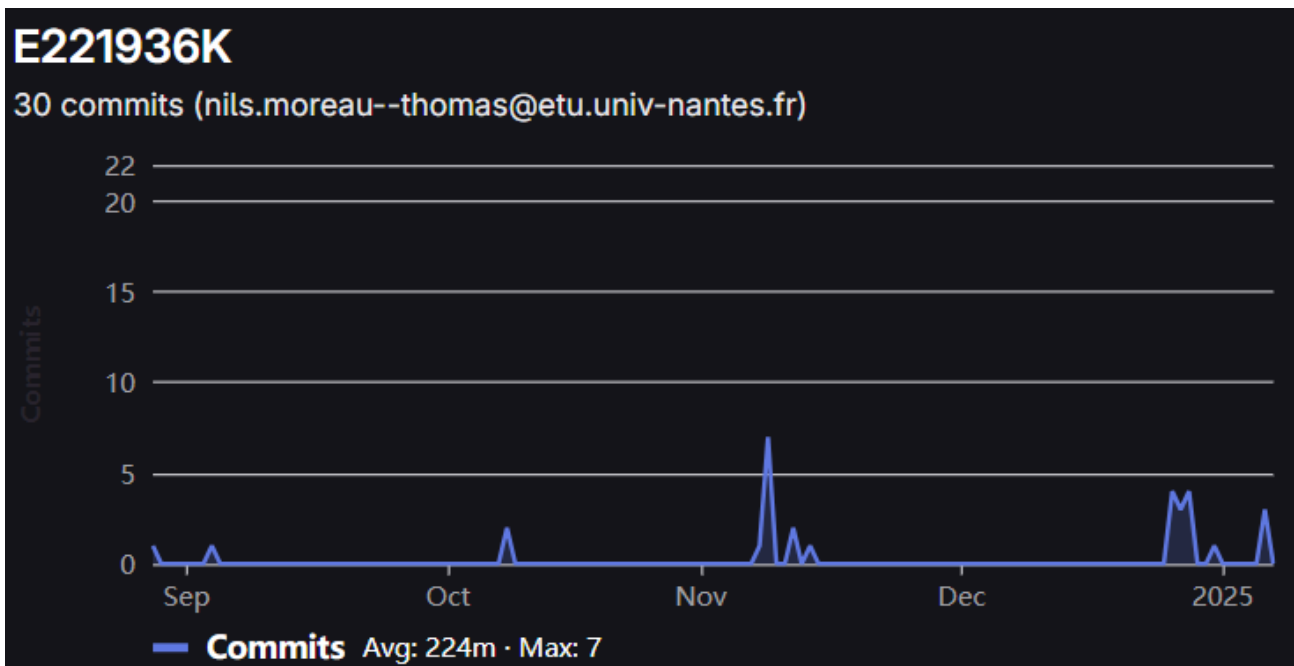
37 commits (tom.fremont@etu.univ-nantes.fr)



Statistiques Gitlab pour Nathan Hequet



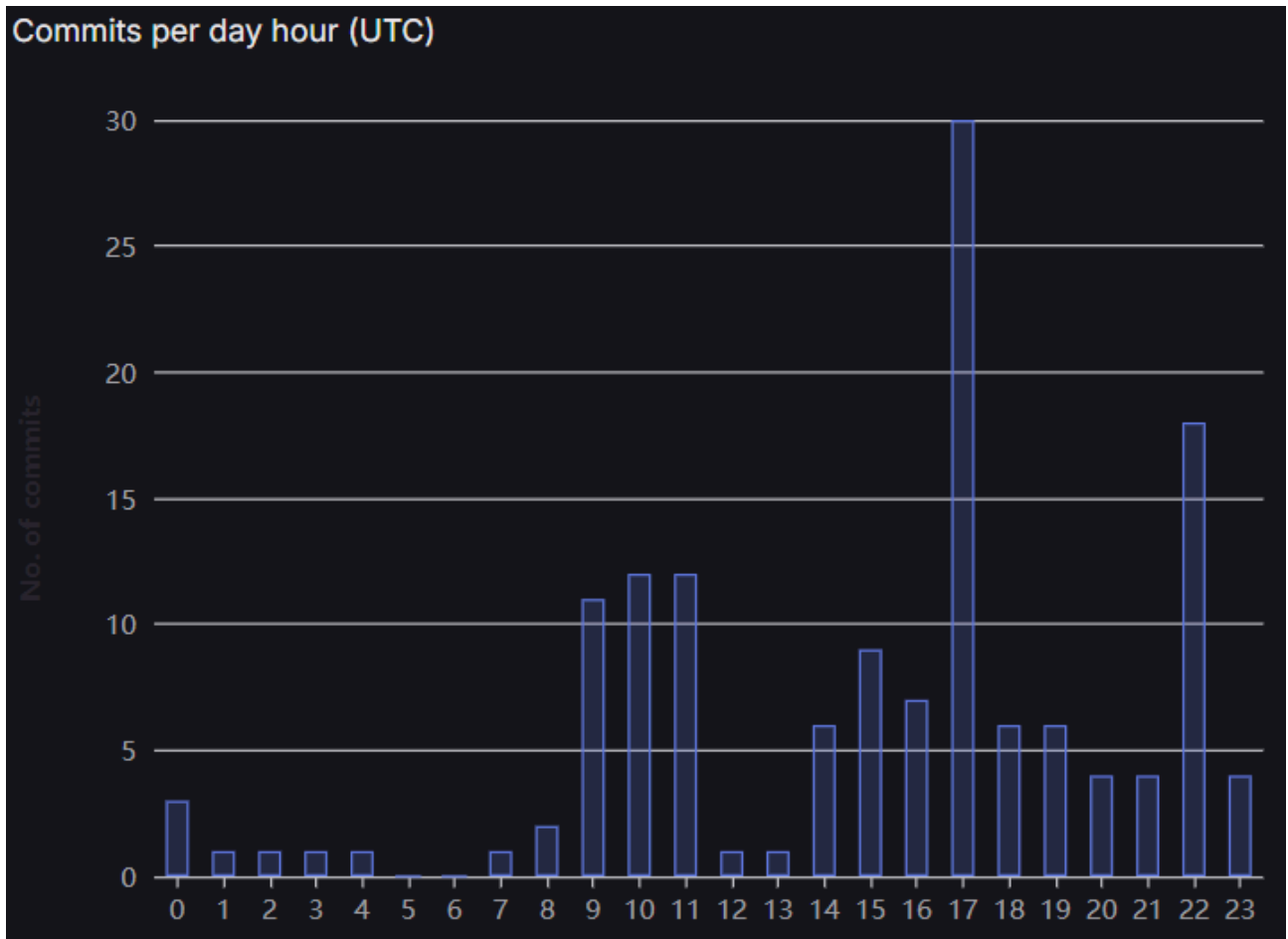
Statistiques Gitlab pour Nils Moreau–Thomas



Statistiques Milestone Gitlab

Version amélioré	sae5-2425-groupe2-medok / app	Open	0/0 complete	<div></div> 0%	
Version de base	sae5-2425-groupe2-medok / app	Open	44/59 complete	<div></div> 74%	

Statistiques Commits par heures Gitlab



Maquettes de l'application

1ère version

Création de profil

Identifiant :

Nom :

Prénom :

Date de naissance : / /

Mot de passe :

Confirmation mot de passe :

Taille : Cm

Poids : Kg

Allergies : Eau

☐ Allergie 1

☐ Allergie 2

☐ Allergie 3

☐ Je n'ai pas d'allergie

Créer profil

Connexion

Identifiant :

Mot de passe :

Créer un profil **Se connecter**

Création de nouveau profil

Identifiant :

Nom :

Prénom :

Date de naissance : / /

Mot de passe :

Confirmation mot de passe :

Taille : Cm

Poids : Kg

Allergies : Eau

☐ Allergie 1

☐ Allergie 2

☐ Allergie 3

☐ Je n'ai pas d'allergie

Retour **Créer profil**

Mot de passe oublié

Codi de sauvegarde :

Retour **Changer mot de passe**

Changement de mot de passe

Nouveau mot de passe :

Confirmation mot de passe :

Retour **Changer mot de passe**

Code de sauvegarde

Votre code de sauvegarde : 000.000.000


N'hésitez pas à conserver votre code de sauvegarde, il permet de se connecter si vous oubliez votre mot de passe.

☐ Je confirme avoir retenu le code de sauvegarde

Suivant

2ème version

Connexion



Connexion

Échec de la connexion

Mot de passe / Identifiant invalide

Création profil (aucun ...)

Création de compte

Création profil (des pr...)

Création de compte

Information profil

Mon profil

Dialog code de s...

Mot de passe oublié

Dialog code sauv...

Votre code de sauvegarde

Votre code de sauvegarde est :

1234567890

Ce code vous permet de changer votre mot de passe si vous l'oubliez. n'oubliez pas de le noter sur un bout de papier

Dialog confirmati...

Suppression de profil

Êtes-vous sûr(e) de vouloir supprimer votre profil ? Cette action est irréversible

Votre code de sauvegarde

Votre code de sauvegarde est :

1234567890

27