

Dossier de projet

Soutenance pour le titre de développeur concepteur d'application

Sommaire

Sommaire	1
Remerciements	3
Organisation de ce dossier	5
Contexte de développement	5
Description et cahier des charges du projet	5
Description du projet	5
Cahier des charges	6
Les utilisateurs du projets	7
Les administrateur système	7
Les clients	7
Les échéances du projet	8
Disponibilité	8
Contexte technique	8
Accessibilité des données	8
Définition des entités	8
Les logs	8
Les base de données	9
Les services	9
Les utilisateurs	9
Fonctionnalité attendues	9
Page de connexion	9
Page d’affichage et de trie des logs	10
Page de performances	10
Page de gestion des base de données	10
Page de gestions des exécutables (administrateur)	10
Répartition du travail	12
Organisation du développement	12
Domaines fonctionnels	14
Système de compte et de connexion/ inscription	14
Affichage des logs	14
Gestion des services	14
Affichage des performances	14

Exécution des scripts	14
Page de connexion / inscription	16
Page de choix des fonctionnalités	17
Pages d’affichage des logs et filtres	18
Page de sélection du service pour performances	20
Page de performances	21
Page de gestion des services	22
Choix des technologies	25
Back end php avec CodeIgniter	25
Front end js avec React-Native	25
Création des données et gestion serveur	26
Intégration continue et sécurité	27
Architecture back-end	28
Modèle MVC	28
API restful	28
Sécuriser l'accès aux données administrateur	28
Identifier et sécuriser les comptes utilisateurs	30
Création d’un compte utilisateur	30
Création et vérification du json web token	32
Front end React-native	33
L’ajax et les contextes	34
Les tests de déploiement	35
Exemple de test de newman	35



Remerciements

Tout d'abord je tenais à remercier l'école LaPlateforme qui a cru en moi et qui m'a donné la possibilité de passer cet examen de seconde année en plus des leçons et des contacts qu'elle m'a apporté

Je tiens aussi à remercier la métropole Aix-Marseille et particulièrement le pôle informatique de la tour la marseillaise pour m'avoir permis d'expérimenter le travail dans une grande entreprise malgré les nombreuses difficultés technique et humaine qu'ils ont eu cette année.



Project summary

This project is a simple mobile app which shows you the latest logs and information about the services currently online on your server. It was made mid 2021 by Jean Davidson Colas and Samuel Joly.

The problem with server administration is that you need to have your computer with you to see and manage your server and applications performance / logs. This project aim to reduce the difficulty to read these data from the server and better, it add some functionality which made the execution/ monitoring easier for the app user

The main goal of this project is to give casual users or systems administrators the tools to monitor and maintain/update their service from their mobile.

We use a set of technologie such as Codelgniter for the backend and React-native for the front end to give better performance and accessibility to the user.

This project was made in 45 days, at a rate of 60 human hours per week. The development was made with collaborative tools such as github, for better maintaining and versioning of the application.

In this document we will see what are the objectives, technologies and security issues that we have faced, and how we have solved them.

A solid blue horizontal bar at the top of the page.A short, solid blue horizontal bar.

Introduction du projet

Organisation de ce dossier

Afin de permettre une lecture sans accroc, ce dossier suit un format permettant l'élaboration d'une trame de développement. Commenant par le cahier des charges et les spécifications techniques, pour ensuite définir les cas d'utilisation et le type d'utilisateur. Une fois l'aspect descriptif du projet terminé, une description technique de chaque partie de ce projet sera présentée et expliquée via multiples formes de graphiques, diagrammes et autres joyeusetés visuelles. Chaque partie exposée ici correspond au référentiel du titre afin de garder une explication minimaliste mais complète. La table des matières contient toutes les informations nécessaires à la navigation de ce dossier.

Contexte de développement

Server App est un projet personnel à utilisation semi professionnelle.

L'objectif de ce projet est de permettre à un administrateur de serveur d'avoir accès aux données de connexions des multiples services qu'il host. Ce projet a été réalisé avec Jean-Davidson Colas dans le cadre du passage de ce titre de développeur et concepteur d'applications. Dans ce dossier, nous verrons quels sont les objectifs de ce projet, comment le développement a eu lieu, quelles sont les technologies utilisées ainsi que les raisons qui nous ont fait choisir ces technologies.

Description et cahier des charges du projet

Description du projet

ServerApp est une application mobile permettant à un sys-admin d'accéder aux données de son serveur via son téléphone.

L'application doit permettre:

- L'accès aux logs de chaque service du serveur
- L'accès aux bases de données des services ainsi qu'à leurs tables
- La mise en forme des données de connexions:
 - Méthode de requêtes
 - Temps de réponse
 - Type d'appareil qui se connecte
 - IP de connexion
 - Nombre d'erreurs / de connexion frauduleuse
- La modification des bases de données
- L'exécution de scripts depuis l'application sur le serveur:

- Backup de bdd
- Arrêt / démarrage de service
- git pull / checkout
- Effectuer les tests prédéfinis

Cette application rentre dans le cadre de la gestion et du maintien de services web tels qu'il est fait par un administrateur système. Elle doit permettre une analyse claire et rapide des informations de connexion au serveur pour optimiser la réaction de l'administrateur dans les cas de failles de sécurité ou de fonctionnalité.

Cette application permet donc une plus grande flexibilité de monitoring des services d'un serveur web. Via son accessibilité (application mobile) mais aussi grâce à ses outils de maintien et de gestion des services.

Cahier des charges


Le projet doit donc être fait avec une contrainte de temps de 45 jours.

Durant cette périodes, il faudra donc:

- Maquetter l'application
- Créer les formats de log :
 - Log d'accès avec toutes les données disponibles
 - Meta-data des logs (temps de réponse du service)
- Créer et sécuriser une base de donnée pour stocker les logs
 - Préparer une bdd de manière à optimiser l'espace de stockage
- Créer l'application mobile
 - Intégrer les solutions de sécurité pour interagir avec l'API
- Créer les scripts de test et d'exécution:
 - Tests applicatif basique (les meme test que pour l'intégration continue)
 - Test supplémentaire concernant le relevé de logs
- Créer les scripts de démarrage et d'arrêt des services
- Créer l'API permettant l'interaction entre l'application mobile et les données:
 - Sécuriser la connexion
 - Limiter les données transférées pour optimiser le temps de chargement coté client
 - Créer une architecture sécurisée et documenté pour pouvoir y intégrer de nouvelles fonctionnalités
 - Créer la documentation de l'API

Les utilisateurs du projets

Ce projet est voué à être utilisé par des gestionnaires de serveur web avec plusieurs services proposés. Par ce fait, l'utilisation est a la foie faite pour des développeurs/administrateurs systèmes mais aussi pour des



particuliers qui veulent avoir accès aux informations relatives à la fréquentation ou aux performances de leurs sites. C'est pour cela que l'UX doit être intuitive et claire, afin de permettre une navigation efficace et un affichage "évident" des données importantes.

Pour se faire, nous avons labellisé chaque donnée de manière à toujours avoir le label de la donnée en vue. Pour que l'utilisateur sache ce qu'il recherche, il a juste à trouver la colonne relative à sa requête. L'utilisation de filtres est faite sans restriction, pour permettre encore une fois une navigation efficace et rapide à travers toutes ces données.

Les administrateur système

Les développeurs ont accès, en plus des fonctionnalités de performances et de logs, aux moyens d'exécution des scripts de tests ou de gestion des services. Pour se faire un menu spécial n'est accessible qu'au utilisateurs disposant des droit d'administration sur l'application (droit accordés lors de la connexion). Pour chaque exécution de scripts, le token administrateur est vérifié et l'action est enregistrée en base de donnée (afin de gérer la traçabilité des action effectué sur les services)

Les clients

Si l'utilisateur n'est pas un administrateur, c'est qu'il veut juste voir combien de visites son site généré et quelques informations techniques comme le temps moyen de réponse, le type de média que les visiteurs utilisent ou le temps moyen passé sur le site. Il a donc accès aux mêmes logs et graphiques que l'administrateur système mais ne peut pas exécuter de scripts (pour des raison de sécurité) nativement. Cependant il peut, en rentrant un mot de passe qu'il aura définis auparavant, pour lui permettre d'accéder aux paramétrages / à la gestion des services qu'il host.

Les échéances du projet

Le projet a commencé le 05 juin 2021 et doit être fini au plus tard le 15 juillet 2021.

Disponibilité

Notre groupe n'as pas de pause dans leur travail, c'est un projet développé le soir et le week end soit à peu près 60 heures hommes / semaines. Mais grâce à notre organisation, tout est rentré dans l'ordre en temps et en heure.

Contexte technique

Accessibilité des données

Les utilisateurs de l'application doivent disposer d'une connexion internet pour télécharger les logs des services webs. L'application étant faite avec React native et via une api, il faut limiter la charge des requêtes. La base de données contient de nombreuses lignes de log (21 000) ce qui ralentit énormément l'affichage des données lorsque l'on fait une requête ayant pour réponse l'entièreté des logs. C'est pourquoi plusieurs méthodes sont utilisées pour mitiger la consommation de ressources:

- Les performances sont calculées server-side ce qui permet à l'application de ne récupérer que les quelques valeurs qui permettent la mise en graphique de ces dernières.
- Les logs sont retournés par paquet de 100, ce qui permet d'avoir un temps de réponse acceptable (actuellement les 21k logs pèsent 1.8mb tandis que 100 logs ne pèsent pas plus que quelques Ko)

Définition des entités

Les logs


Les logs sont la donnée principale de l'application. Il constitue la base de calcul des performances ainsi que des stats associés aux visites / erreurs. Ils contiennent toutes les informations qu'apache peut récupérer lors d'une connexion au service (date, média utilisé, type de requête, page visité etc....)

Les base de données

Tout bon administrateur système doit avoir un œil attentif sur ses base de données et leurs backup, c'est pourquoi nous avons défini une liste de base de données accessibles par l'application. Ces entités contiennent donc les tables qui y sont associées mais aussi leurs contenu et leurs description (Describe table). Pour permettre une analyse complète des données et de leurs types via l'application.

Les services

Une liste des services actuellement disponibles est définie. Ce qui permet de gérer plusieurs logs/performances selon le service choisis.



Chaque service représente une interface entre le client et le serveur, et chaque requête faite par le client est enregistrée pour pouvoir être étudiée via l'application.

Les utilisateurs

Pour accéder à l'application, le client doit se connecter à un compte utilisateur afin d'assurer et de contrôler qui a quel pouvoir et qui a fait quelle action toujours dans le respect du DICP

Fonctionnalité attendues

Suite aux réunions avec plusieurs administrateurs systèmes, nous avons pu mettre en relief quelles sont les fonctionnalités primordiales pour pouvoir avoir un regard global et contextuel sur les services gérés.

Page de connexion

Comme toute application mobile faisant appel à une API ou un service en ligne, la page de connexion est une nécessité autant du point de vue de la sécurité que de la praticité.

La page de connexion est la première page à laquelle l'utilisateur accède. Elle contient deux inputs et un bouton.


- Input login
- Input password
- Bouton connexion

Chaque utilisateur dispose donc d'un compte unique avec des privilèges associés (administrateur ou simple utilisateur).

Page d'affichage et de trie des logs

La base de l'application repose sur le traitement des logs d'accès et/ou d'erreur aux services monitorés. Il paraît alors naturel de permettre aux utilisateurs d'accéder à la liste de ces logs.

Pour des raisons de performances, et d'utilisation, cette liste est accompagnée d'un outil de filtrage permettant de trier les logs par n'importe quel attribut disponible (ip, méthode, temps de réponse,...).



Cette page donne donc la liste des logs, mais cette liste est inutile sans outils de recherche ! C'est pourquoi elle est accompagnée d'un système de filtre couvrant tous les spécificité des logs (filtres générés via les colonnes disponibles dans la tables contenant les logs

Page de performances

Pour chaque service disponible, nous devons afficher les performances sous forme de graphiques. Chaque service dispose donc d'une page de performance avec au moins 2 graphiques: la fréquence de visite du service (le nombre de visites d'IP unique par jour) et le nombre d'erreurs de ce service. Si le service est enregistré comme étant un site, un troisième graphique sous forme de camembert affiche la proportion d'accès au service via x ou y média (mobile, tablette, desktop) et/ou par type de navigateurs (firefox/chrome/edge/safari).

Ces données sont générées server side pour économiser le temps de chargement côté client.

Page de gestion des base de données

Cette page permet d'accéder aux bases de données des services (s' ils en ont une). Un CRUD (Create, Read, Update, Delete) est disponible pour chaque table de chaque base de données. L'utilisateur doit pouvoir accéder en moins de 3 "clics" à n'importe quelle table de n'importe quelle base de donnée depuis le menu de gestion des bases de données.

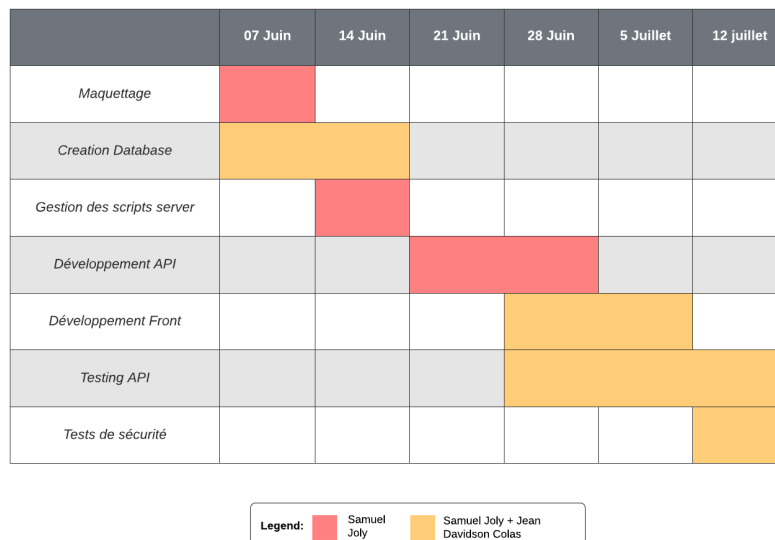
Page de gestions des exécutable (administrateur)

Cette page permet d'exécuter les script disponibles sur le serveur (via la Common Gateway Interface d'Apache). Seulement les utilisateurs avec les droits d'administrateurs ont accès à cette fonctionnalité. Les scripts disponibles permettent de recharger le service, de l'allumer ou le désactiver (Comme avec systemd) et de mettre à jour les logs (mis à jours automatiquement chaque minutes).

A solid blue horizontal bar at the top of the page.A short, solid blue horizontal bar.

Gestion du projet

Répartition du travail



Le diagramme de gantt ci-dessus représente la répartition du travail de développement sur le projet.

Bien que les tâche en rouge soient fait par mes soins, je suis aussi actif dans les tâches marquée en orange, travaillant avec Jean Davidson Colas Bien qu'ayant dans ces partie plus un rôle de superviseur et de consultant, ayant développé l'API et les scripts serveur, j'avais des informations qui n'étais pas à la disposition de Jean Davidson Colas.

Le projet a donc commencé le 07 juin 2021 et s'est terminé durant la semaine du 12 juillet 2021 comme prévu !

Organisation du développement

Amateur d'open source, notre projet a été développé avec les technologies de versionning offertes par l'outil git. Chaque utilisateur dispose de son fork et propose un PR (Pull Request) lorsqu'il a fini une fonctionnalité. Je relis alors le PR et le merge avec la branche développement si tout va bien. Sinon, le commit reste dans la branche du codeur concerné et il n'as plus qu'à revoir son code selon les informations que je lui ai donné concernant le refus de son pull request.

A solid blue horizontal bar at the top of the slide.A short, solid blue horizontal bar.

Analyse Fonctionnelle



Domaines fonctionnels

Afin de mieux comprendre quelles sont les parties prenantes de l'application, j'ai décidé de les distinguer par leurs utilités.

Système de compte et de connexion/ inscription

Pour respecter les règles DICI et pour limiter l'accès aux données fournies par notre application, une connexion est requise. Cela permet entre autres de logger les échanges effectués avec la base de données en fonction de l'utilisateur connecté pour assurer la traçabilité/ la preuve de modification/accès aux données

Affichage des logs

Comme dit précédemment, les logs doivent être accessibles et filtrables par l'utilisateur. C'est le cœur de l'application métier. Cette partie est la plus importante de l'application car toutes les informations disponibles découlent de l'analyse des logs. C'est pourquoi un affichage clair et filtrable est nécessaire.

Gestion des services

Les logs dépendent des services qui les génèrent, c'est pourquoi il faut un accès aux services et à leurs fonctionnalités. Les utilisateurs peuvent ajouter ou supprimer des services depuis l'application

Affichage des performances

Les utilisateurs ont accès aux performances des services monitoré sous forme de graphiques en batin, ligne et camembert.

Exécution des scripts

Pour ajouter une couche fonctionnelle d'interaction avec les services, les administrateurs ont la capacité d'exécuter des scripts prédéfinis sur le serveur afin d'augmenter leur contrôle et la visibilité qu'ils ont sur les services.

A solid blue horizontal bar at the top of the page.A short, solid blue horizontal bar.

Maquettage

Page de connexion / inscription

The image shows a dark-themed user interface for a login and registration page. It is divided into two main sections by a horizontal line. The top section is for login, featuring an 'Email' label, a text input field, a 'Password' label, another text input field, and a yellow 'Login' button. The bottom section is for registration, featuring an 'Email' label, a text input field, a 'Password' label, a text input field, a 'Verify Password' label, a third text input field, and a yellow 'Register' button. All labels are in a light gray font, and the input fields are light gray rectangles. The buttons are yellow with black text.

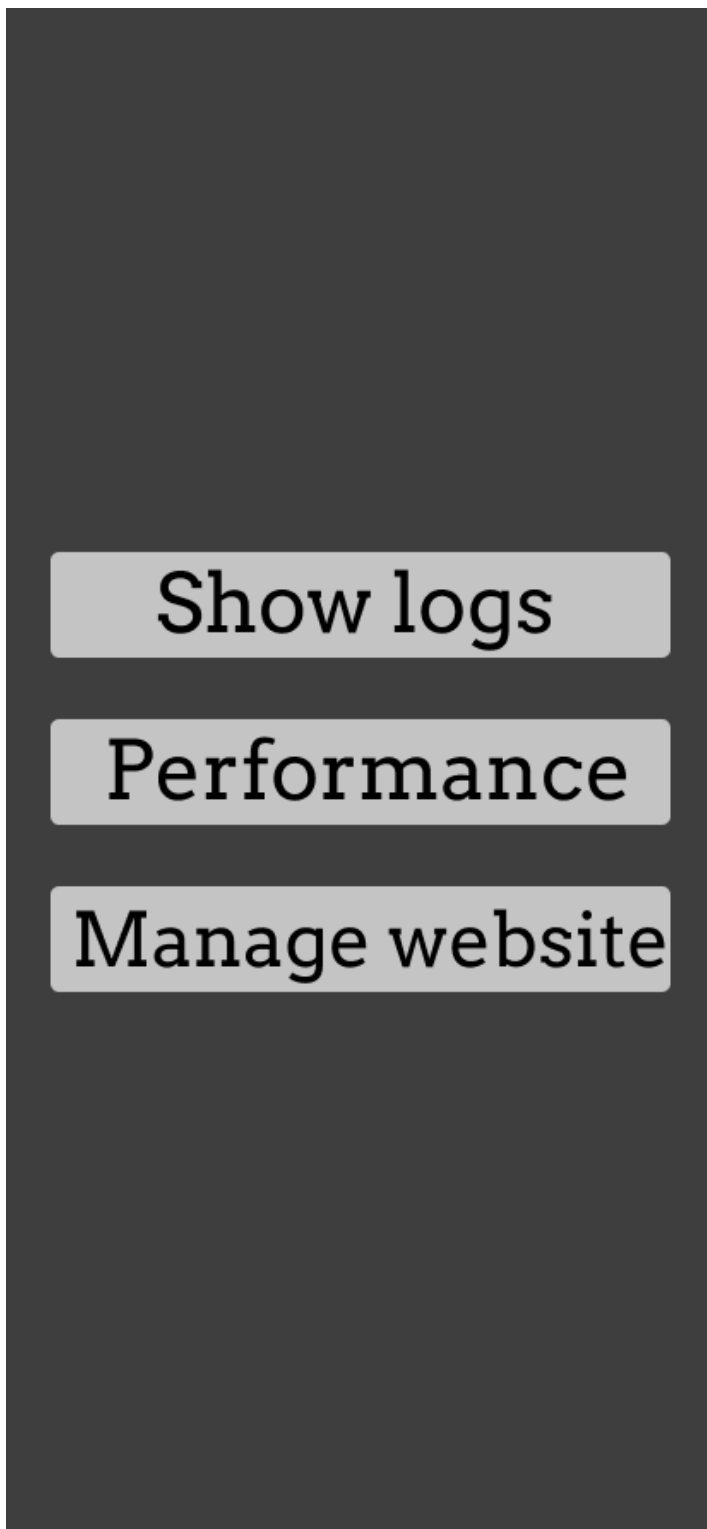
Cette page permet à l'utilisateur qui possède déjà un compte de se connecter (si toutefois il ne l'est pas déjà) ou bien de s'inscrire.

La connexion se fait avec un email et un mot de passe, qui se retrouveront encodé dans le jwt retourné en cas de connexion effectuée.

Si l'utilisateur n'as pas de compte, il suffit d'en faire un avec le formulaire inférieur. Seulement trois inputs sont nécessaires pour permettre l'inscription. Plusieurs règles sont présentes pour s'inscrire mais nous verrons cela dans la partie développement du dossier de projet.

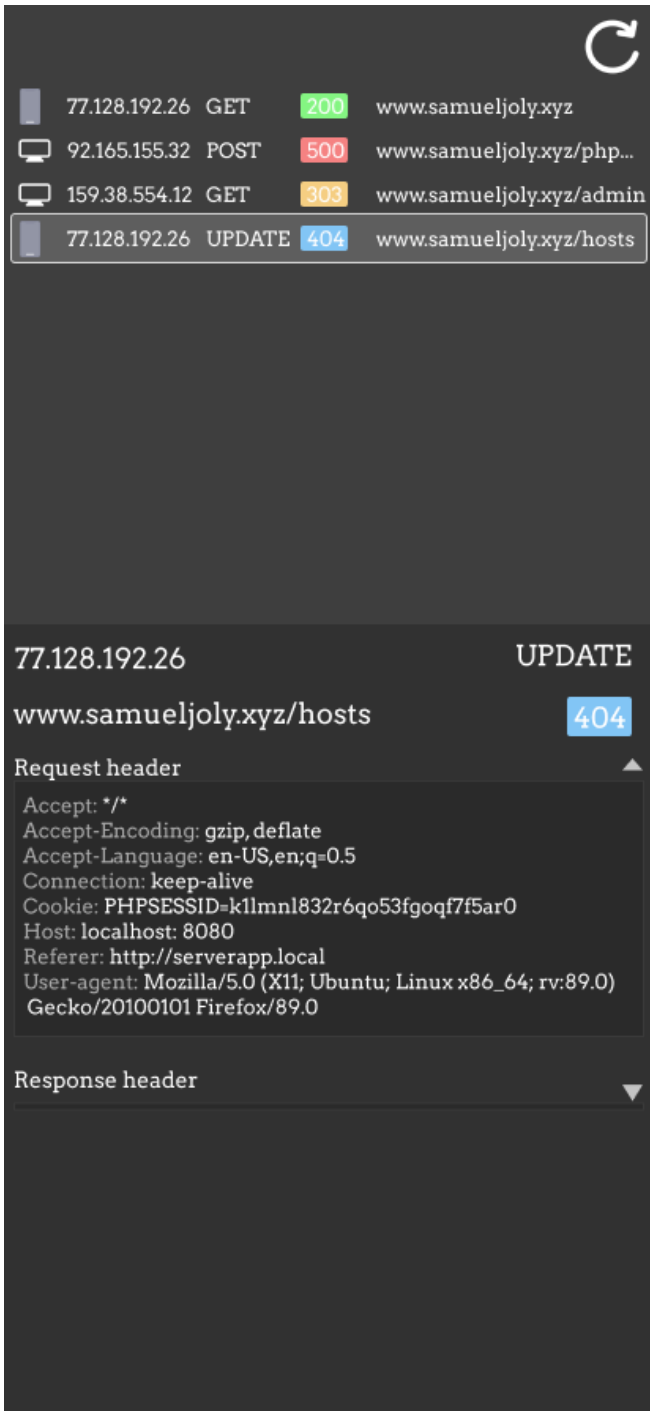
Une fois inscrit, il pourra se connecter en toute tranquillité avec ses identifiants. Une fois connecté, un cookie est enregistré et reste présent tant que le téléphone de l'utilisateur est allumé et déverrouillé. Si l'application est inactive pendant plus de 7 minutes, le token de connexion disparaît et l'utilisateur est déconnecté.

Page de choix des fonctionnalités



Cette page est le menu principal de l'application. Elle permet d'accéder à chaque partie de cette dernière rapidement. C'est aussi le point de chute de chaque page de manière à ce que, si l'utilisateur spamme le bouton retour depuis n'importe quelle page, il finisse par se retrouver ici.

Pages d'affichage des logs et filtres



La page d'affichage des logs se définit par 3 éléments:

- La liste des logs
- Les informations sur un log sélectionné
- Le slider de filtre des logs

Pour permettre une lecture rapide et efficace des logs, le découpage se fait en long et en large (voir screen suivant) pour les filtres et les informations.

On peut voir les informations sur le log sélectionné en bas de page, avec les headers de requêtes ainsi que certaines informations déjà disponibles dans la liste.



Les filtres de logs sont accessibles en glissant son doigt vers la droite. Elle permet de trier les logs selon quelques informations les concernant (type de réponse, type de média, ip de la requête ou bien par services disponibles sur le serveur).

La mise à jour des logs selon les filtres choisis est automatique. Dès que l'utilisateur sélectionne un paramètre, la liste se met à jour.

Page de sélection du service pour performances

Performances

samueljoly.xyz

Hook endpoint

ServerApp API

www.samueljoly.xyz 

Connections

Connections totales: xxx

Connections aujourd'hui: xxx

Connections cette semaine: xxx

Connections ce mois: xxx

Connections échouée

Erreurs totales: xxx

Erreurs aujourd'hui: xxx

Erreurs cette semaine: xxx

Erreurs ce mois: xxx

Perfs

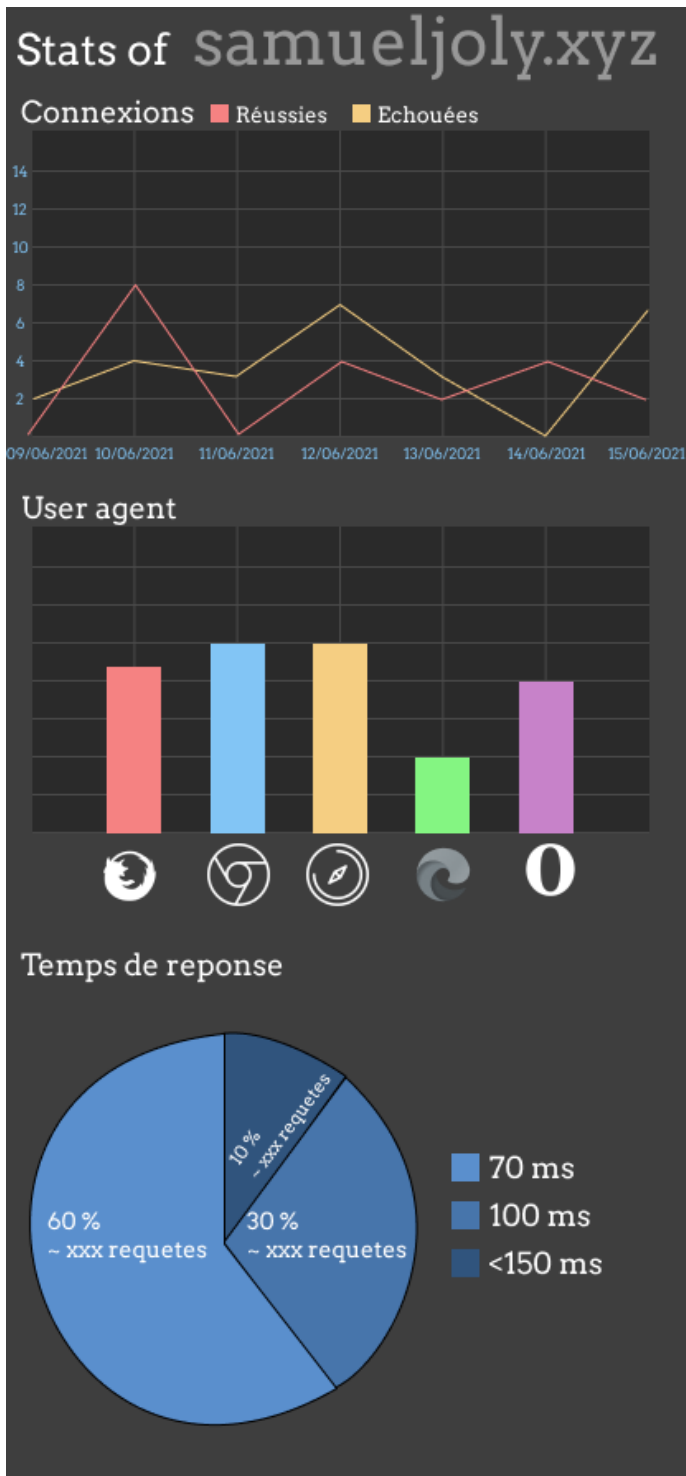
Temps de reponse moyen: xxx

Taille moyenne transféré: xxx

Pour chaque service disponible via l'application, une page de performance est générée. Cette page permet d'avoir une vue d'ensemble du service sélectionné mais donne aussi la possibilité d'accéder à des graphiques pour obtenir d'autres informations concernant le service sélectionné (voir page suivante).

Toutes les informations de cette page sont calculées en avance sur le serveur pour économiser la puissance de calcul du client. Elles ne sont que récupérées en asynchrone lors du chargement de la page, ce qui laisse au client uniquement la charge de l'affichage de ces données.

Page de performances

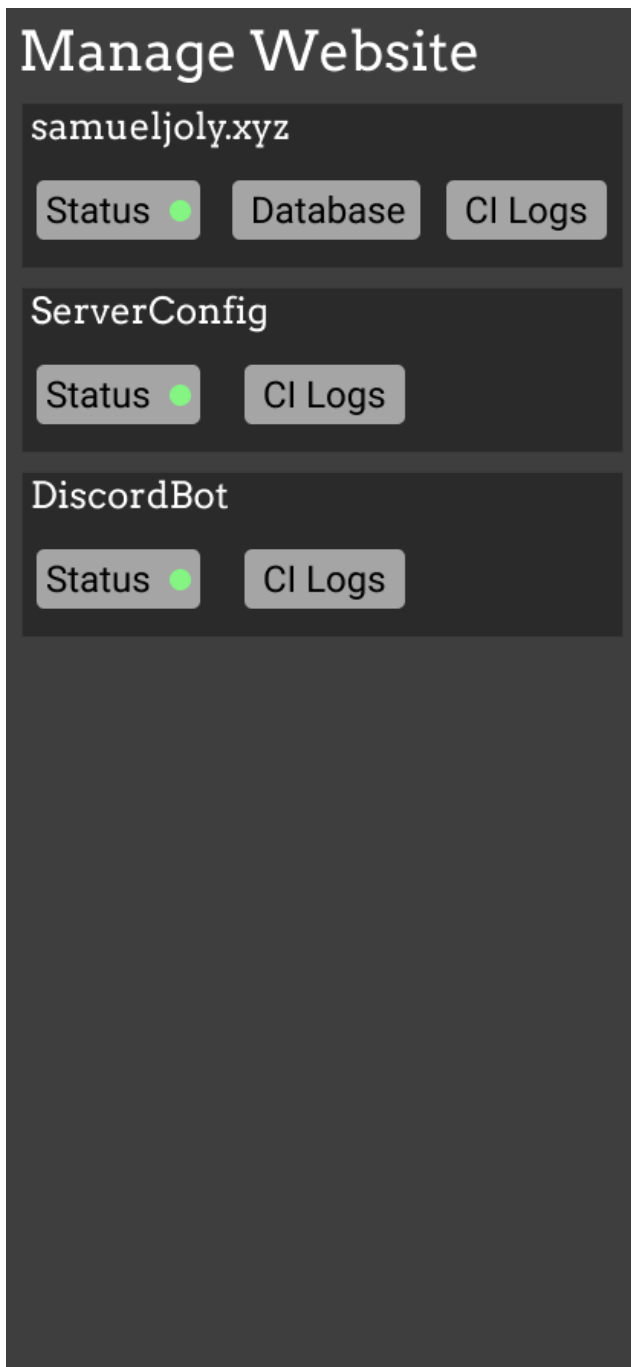


La page de performances donne accès aux données basiques issues des logs, soit le nombre de requêtes d'IP unique réussie (avec 200 comme statut de réponse) et échoue (avec 400 comme statue de réponse). Ainsi que les navigateurs utilisés par les utilisateurs du service.

Pour finir, la page donne la proportion de requêtes en fonction du temps de réponse de cette dernière.

Toutes les informations de cette page sont calculées en avance sur le serveur pour économiser la puissance de calcul du client. Elles ne sont que récupérées en asynchrone lors du chargement de la page, ce qui laisse au client uniquement la charge de l'affichage de ces données.

Page de gestion des services



Pour chaque service, une page de gestion est générée afin d'avoir accès à plusieurs outils le concernant.

En fonction du type de projet, plusieurs options sont disponibles:

Si le projet dispose d'une base de donnée, il est possible de voir sa description (tel que Describe table la donne) ainsi que le contenu de cette base de donnée (voir les deux prochaines maquettes)

Si le projet dispose d'un système d'intégration continue, et que ce dernier a des logs de configuration, ils sont accessibles via l'app.

Dans tous les cas, le service a un statut (soit en ligne, soit hors ligne) qui est affiché sur cette page.

samueljoly.xyz

Tables

Table name

Table name

Table name

Table name

Table name

Columns

Column name	Type	spe
-------------	------	-----

Column name	Type	spe
-------------	------	-----

Column name	Type	spe
-------------	------	-----

Column name	Type	spe
-------------	------	-----

Column name	Type	spe
-------------	------	-----

Column name	Type	spe
-------------	------	-----

Column name	Type	spe
-------------	------	-----

Column name	Type	spe
-------------	------	-----

Column name	Type	spe
-------------	------	-----

samueljoly.xyz


Filter columns

project_name project_id ✓ project_description✓

project_name project_id ✓ project_description✓

Data

project_id	project_description
------------	---------------------



Ces deux pages permettent la visualisation des tables de la base de données du service sélectionné dans la page précédente. La page de gauche donne accès à la description des données, et la page de droite donne accès aux données elles-mêmes.

Réalisation

Choix des technologies

Les technologies définies pour ce projet sont relativement imposées (ou plutôt conseillées) par le staff de LaPlateforme.

Back end php avec CodeIgniter

Le back-end est donc fait uniquement en PHP avec le framework CodeIgniter, qui allie simplicité et flexibilité. Il dispose d'une architecture MVC pour les projets front-end et back-end et est aussi composé de tous les outils nécessaires pour développer, tester et sécuriser une API tels que celle que nous créons. En effet, CodeIgniter met en place un système de routeur (rien de bien différent des autres frameworks back end) composé de plusieurs fonctionnalités telles qu'un système de vérification intégrée aux routes, un système de regex automatisé permettant la création d'API RESTfull (/var/(:id)/log) ainsi qu'un système de gestion du CORS très intuitif. La gestion des erreurs et la validation des requêtes se fait aussi via l'intégration des routes, ce qui donne au code une architecture saine et explicite, puisque via le fichier Routes.php, nous pouvons voir qui a le droit d'y accéder, selon quelles règles de validation et avec quelles autorisations. De plus, CodeIgniter est particulièrement léger pour les fonctionnalités qu'il offre. Il est composé de plugins provenant en grande partie de symfony, ce qui assure une qualité professionnelle.

Étant partisan du "do it yourself" j'ai pris l'initiative de n'installer que les modules nécessaires (soit ceux qui comblent une fonctionnalité de CodeIgniter). Bien que ce framework contient de nombreuses fonctionnalités certaines d'entre elles sont trop spécifiques ou pas assez bien notées pour être utilisées dans un environnement de production. Je pense notamment à l'intégration de PHPUnit, qui, plus que douloureuse, ne permet que peu de fonctionnalités comparée à d'autres frameworks de tests d'API ce qui rend l'utilisation d'autres bibliothèques de test plus intéressante tant leur intégration est facile comparée à PHPUnit (même avec composer !).

Front end js avec React-Native

Concernant le Front-end, étant donné que notre projet doit être une application Mobile et qu'il est bien plus simple d'apprendre React-Native plutôt que les spécificités de Kotlin et de Swift, nous avons choisi React-Native. Bien que ce framework JS, à l'instar de la grande majorité des frameworks js, soit une usine à gaz qui bouscule la majorité des connaissances en développement front que nous avons (notamment avec l'intégration du css natif), nous avons la chance de l'utiliser en 2021, après les nombreuses mises à jours qui ont

entre autre permis la suppression de redux et l'intégration des contextes. Ce qui nous a permis d'éviter l'installation de multiples librairies surfaites. Malgré son originalité, il ne déroge pas aux règles des frameworks javascript, il lui faut de nombreuses librairies pour que react-native commence à être puissant et efficace. Notamment React-navigation qui ajoute la majorité des interactions digitales tels que le swipe ou le système de panels, inexistant nativement dans le framework.

L'utilisation de react-native et la volonté de développer une application mobile iOS et Android nous oblige à mettre en place un système pour simuler et tester l'application sur notre ordinateur (ou directement sur notre mobile mais dans ce cas nous sommes limité matériellement (concernant les iPhone surtout)). C'est pour cela que react native, en plus de ses nombreuses librairies, s'accompagne de tout un tas d'applications permettant l'émulation de mobiles. Après avoir essayé Expo et Android Studio, c'est un compromis des deux qui a fait mon bonheur. Expo génère les fichiers gradle et android studio build le tout sur mon mobile, me permettant de jouer d'une apk disponible à chaque instant ! Bien sûr, cette méthode n'est pas applicable pour une situation de développement, auquel cas il faudrait, à chaque push, build pendant 5 minutes le projet pour pouvoir le tester. C'est pourquoi, durant les phases de développement, nous avons utilisé expo et sa fonctionnalité d'émulation via le réseau local (LAN) afin de tester l'application simultanément sur nos téléphones respectifs. Comme nous ne disposons pas de téléphone iOS, il nous a fallu l'émuler, mais grâce à android studio ce fut (presque) un jeu d'enfant.

Création des données et gestion serveur

Concernant la génération des logs, le travail de compréhension du serveur apache2 a été fait de manière à parfaitement contrôler:

- La création et la gestion de virtual host
- L'ouverture et l'activation des sites et des ports
- La gestion des utilisateurs (notamment de www-data)
- La création de fichier de logs custom
- La sécurisation des imports dans la base de données
- La sécurisation des utilisateurs de la base de données
- La gestion des tables et des utilisateurs qui insère les données dans les tables
- La création et la gestion des Crontabs et des Anacrons
- La gestion des utilisateurs des Crontabs
- La gestion et la création des logs lors des mise à jours de la base de données de logs

Le serveur fonctionne donc de la manière suivante:

Lorsqu'un client visite un des services monitoré, Apache utilise un LogFormat personnalisé pour enregistrer les requêtes de ce client dans un fichier standardisé dans /var/logs/<Service>/access.log. Le LogFormat est donc la base de notre application, c'est lui qui définit quelles sont les données accessibles depuis l'application. Il renvoie toutes les informations qu'apache peut récupérer lors d'une connexion à un des services en ligne. Une fois les informations des logs enregistré dans un fichier spécifique, fichier limité en droit et accessible

uniquement par l'utilisateur apache soit "www-data". Ce fichier dispose d'un lien symbolique vers un dossier spécial définis automatiquement à l'installation par mysql : "/usr/lib/mysql-files/<Service>". Ce dossier n'est accessible que par l'utilisateur nommé mysql qui est l'utilisateur que mysql utilise. Cet utilisateur dispose d'un crontab qui s'exécute chaque minute. Ce crontab exécute alors un script situé dans le répertoire /usr/lib/mysql-files/script qui importe dans mysql les logs du symlink présent dans le dossier <Service>. Etant donné que ce système est flexible, il y a plusieurs dossiers <Service> dans /usr/lib/mysql-files, et chacun dispose des logs du service représenté. L'import des logs dans la table associée <Service_Logs> s'effectue alors avec la commande mysql-import --columns='Liste des colonnes représentant chaque retour du LogFormat d'apache' <Service_logs> <Service_logs_file>. Si l'exécution s'effectue, tout va bien et le serveur continue sa vie, si toutefois elle rate, un fichier d'erreur est généré dans lequel est formaté un message d'erreur (<date de l'échec> <table ciblée> <fichier ciblé> => erreur) ce qui facilite énormément le débogage.

Intégration continue et sécurité

Une fois les logs générés et enregistrés, il faut donner la capacité au serveur à mettre à jour les projets automatiquement selon les push sur la branche prod des dis projets. Pour ce faire, nous utilisons la fonctionnalité des webhooks proposée par github. Les webHooks sont des événements déclenchés selon certaines actions qui se produisent sur github (push, pull request, commit, ...). Les webhooks appellent un url définis et y envoient les données de l'événement déclencheur. Dans le cadre de notre projet, une section intégration continue est accessible par l'utilisateur dans la section gestion des services. Pour permettre la mise à jour des projets, nous avons mis en place des webhooks qui se déclenchent lors d'un push sur la branche de production du projet. Lorsque nous mettons à jour la branche prod, le webhook se déclenche. Il envoie alors à une API de notre cru, un json contenant de nombreuses informations concernant l'événement déclencheur, et une clé de vérification dans le header. github utilise un algorithme de vérification appelé HMAC. Cet algorithme d'encryptage permet de vérifier l'authenticité et la validité du message reçu. Lors de la création du webHook, github demande une clé de chiffage. Cette clé sera utilisée avec un algorithme d'encryptage sha-256 sur le payload (les données envoyées) et un string unique en sortira. Lorsque le serveur reçoit le payload, il reçoit le résultat de la digestion de la clé avec le payload. Il refait donc la même opération que github soit digérer le payload avec la clé qu'il a, et comparer les deux résultats digérés.

Bien que ce système paraît sécurisé, car il valide à la fois le contenu du message mais aussi l'identité de l'expéditeur, une faille perdure. Lors de la comparaison des hash digérés, une time attack est possible. C'est pourquoi nous utilisons la fonction PHP hash_equals, permettant de renvoyer une réponse sans faire varier le temps de réponse. Une simple comparaison de string ne pouvait pas être sécurisée car elle s'arrête dès qu'une différence entre les hash est trouvée. Ce qui pourrait permettre à un attaquant de mesurer le temps de chaque réponse et selon la vitesse de refus, savoir au coup par coup si son hash est valable.

Une fois le webhook vérifié, le serveur logs alors l'action vérifiée puis exécute un script faisant un git pull sur le projet spécifié par les données du payload. Ces logs sont accessibles via l'application dans la page gestion des services.

Architecture back-end

Modèle MVC

CodeIgniter est un framework basé sur les principes MVC (Modèle View Controller). Or ici, étant donné que nous l'utilisons sous forme d'API, les vues ne sont pas utilisées comme elles devraient l'être. En effet, étant donné qu'aucun élément de front n'est généré par le back-end, les vues peuvent s'assimiler au formatage des données renvoyées au client suite à une requête. De plus, le MVC ne prend pas en compte la gestion des routes comme nous en avons besoin dans le cadre d'une API, ou alors, il faut y rajouter un controller de controller, qui est donc notre fichier Routes.php, permettant de rediriger l'exécution de chaque requêtes vers le contrôleur associé. Lorsque la route a été trouvée, CodeIgniter exécute la méthode associée au contrôleur appelé. Ici nous reprenons l'utilisation classique du MVC, c'est-à-dire que le contrôleur permet d'exécuter les composants métier en faisant appel aux modèles relatif à la requête. Les modèles sont l'interface entre l'API et la base de données. Ils formalisent les types de données et contrôlent les interactions avec la base de données via les méthodes associées à ce dernier. Nous n'avons pas parlé de programmation orientée objet jusqu'ici mais chaque composant du MVC est un objet (les contrôleurs, les routes et les modèles) centralisant en ses méthodes l'ensemble des tâches métier dont l'application a besoin.

API restful

Pour permettre une utilisation simple de l'API afin d'avoir un back-end complet et efficace, un modèle Restful est utilisé. Chaque routes donnent donc accès à des fonctionnalités distinctes du programme (accéder à la liste des logs, aux informations spécifiques à un logs ou aux script d'exécution du serveur).

Pour respecter l'architecture restful, certaines routes prennent en compte les arguments donné dans la routes

```
$routes->get('logs/(:num)/infos', 'Logs::getInfosFromLog/$1');
```

Grâce à cette architecture, on limite la dépendance aux informations contenues dans le payload de la requête provenant du client, ce qui allège d'un calcul le serveur et qui, parallèlement, facilite l'accès à la donnée demandée. Dans cet exemple, l'utilisateur cherche à obtenir les informations complètes d'un log (voir page d'affichage des logs). Il n'a qu'à requêter l'api avec l'id du log en question pour obtenir les information relative à ce log, plutôt que d'aller requêter une route générique et d'y envoyer en paramètre l'id du log concerné.

Sécuriser l'accès aux donnée administrateur

Comme précisé auparavant, les utilisateurs se distinguent en deux groupes, les administrateurs de serveur et les utilisateurs lambda. Pour limiter l'accès aux fonctionnalités d'administration, CodeIgniter met à notre

disposition un système de filtre applicable aux routes. Ces filtres s'exécutent avant et/ou après l'exécution du controller relié à cette route.

```
$routes->group('admin', ['filter' => 'AdminAuth'], function ($routes) {  
    $routes->post("script/(:any)", 'script::execute/$1');
```

Dans le screen ci-dessus, on peut voir que la route admin englobe la route script (ce qui donne donc la route admin/script) de manière à appliquer le filtre AdminAuth pour chaque routes qu'il contient. Le filtre admin auth est définis dans le fichier Filters/AdminAuth.php selon l'organisation suivante:

```
1 1?php  
2 namespace App\Filters;  
3  
4 use CodeIgniter\API\ResponseTrait;  
5 use CodeIgniter\Filters\FilterInterface;  
6 use CodeIgniter\HTTP\RequestInterface;  
7 use CodeIgniter\HTTP\ResponseInterface;  
8 use Config\Services;  
9  
10 class AdminAuth implements FilterInterface  
11 {  
12     use ResponseTrait;  
13  
14     public function before(RequestInterface $req, $arguments = null)  
15     {  
16         try {  
17             helper('jwt');  
18             $tokenInfo = getTokenInfo($req);  
19         } catch (\Exception $e) {  
20             return Services::response()->setJSON([  
21                 "error" => "Invalid token."  
22             ]);  
23         }  
24  
25         $role = $tokenInfo[1];  
26         if ($role != 10){  
27             return Services::response()  
28                 ->setJSON([  
29                 "error" => "You do not have the permission"  
30             ]  
31             );  
32         }  
33     }  
34  
35     public function after(RequestInterface $req, ResponseInterface $res, $arguments = null)  
36     {  
37     }  
38 }
```

On peut voir que la classe AdminAuth dispose de deux méthodes: before et after. Ces méthodes définissent à quel moment le filtre doit s'exécuter par rapport à la requête, avant l'exécution de la requête (before) ou bien après (after). On peut voir dans l'exemple ci-dessus que ce filtre s'effectue avant l'exécution de la requête dans le contrôleur associé. On vérifie ici l'authenticité du token ainsi que le niveau de privilège qui lui est associé la

vérification du token s'effectue L17-18, si le token est bien valide, et donc que la vérification n'as pas renvoyé d'erreur, on peut passer à la vérification des droit d'accès de l'utilisateur L26. Si le token et les permissions sont aux normes, l'exécution peut continuer, le filtre ne renvoie rien. Si le token n'est pas valide (soit à cause de la permission qui lui est associée, soit parce que le token est frauduleux ou expiré), nous répondons directement à la requête en retournant un json expliquant quel est le problème.

Nous avons vu comment la gestion des routes disponibles aux administrateur est gérée, mais qu'en est il de la vérification des identifiants lors d'une requête ne nécessitant pas de permission particulière ? Comment sécurisons-nous les requêtes faites à l'API et selon quels critères définissons-nous la validité d'un compte ?

Identifier et sécuriser les comptes utilisateurs

Pour permettre l'identification et la validation d'un compte utilisateur tout le long de son utilisation de l'API, nous avons choisi d'utiliser une méthode d'authentification par JSON Web Token. En utilisant la documentation fournie par le rfc 7519 (<https://datatracker.ietf.org/doc/html/rfc7519>) nous avons mis en place un outil de création et de vérification des tokens fournis par l'utilisateur de l'API.

Pour accéder aux fonctionnalités de l'API, toutes les routes, à l'exception des routes d'inscription et de connexion, nécessitent la présence d'un header Bearer avec le token fournit à l'utilisateur.

Dans cette section, nous allons voir comment nous avons sécurisé l'API à l'aide des JWT.

Création d'un compte utilisateur

Lors de la création d'un compte utilisateur, le client entre son email/username ainsi qu'un mot de passe. Avant de générer le token et valider l'inscription, nous effectuons des vérifications sur les données fournies par le client.

Pour ce faire, nous utilisons une combinaison de méthodes faites à la main et de fonctionnalités fournies par Codelgniter. J'ai nommé les objets de validation.

Les objets de validation se composent de règles, de messages d'erreur et de système de vérification. Les règles sont en partie définies par Codelgniter, qui, si bien configuré avec la base de donnée, permet de vérifier par exemple l'unicité d'un champ dans une table, mais aussi permet de vérifier la taille minimum ou maximum d'un attribut de requête. Bien que ces règles couvrent les besoins les plus primaires de la gestion d'information envoyé par le client, certaines actions nécessitent une vérification plus poussée ou simplement plus complexe selon l'architecture des données à vérifier.


```
18 public function password_verify($str, string $fields, array $data) : bool
19 {
20     try {
21         $model = new UsersModel();
22         $password = $model->getUserByEmail($data["email"])[0]["password"];
23         return password_verify($data["password"], $password);
24     } catch (Exception $e) {
25         return false;
26     }
27 }
28
29 687 pp6-forum/api/app/Validation/RegisterRules.php 22:0 49%
30
31 /**
32  * @api {post} /auth/login Authenticate user
33  * @apiName login
34  * @apiGroup Authentication
35  *
36  * @apiParam {String} email The new user's e-mail
37  * @apiParam {String} password The new user's password
38  *
39  * @apiUse LoginResponse
40  *
41  */
42 public function login()
43 {
44     $rules = [
45         "email" => "required|min_length[6]|max_length[50]",
46         "password" => "required|min_length[6]|max_length[30]|password_verify[user.password,user.email]";
47     ];
48     $errors = [
49         "email" => "Invalid login credentials",
50         "password" => ["password_verify" => "Invalid password"]
51     ];
52     $input = $this->getRequest($this->request);
53     if (!$this->validateRequest($input, $rules, $errors)) {
54         return $this->fail(
55             [
56                 "errors" => $this->validator->getErrors(),
57             ]
58         );
59     }
60     return $this->getJWTForUser($input["email"]);
61 }
62
63 2.8k pp6-forum/api/app/Controllers/Auth.php 51:0 53%
64
65 public function validateRequest($input, array $rules, array $message = [])
66 {
67     if(!is_string($rules))
68     {
69         $validation = config('Validation');
70         if(isset($validation->$rules))
71         {
72             throw ValidationException::forRuleNotFound($rules);
73         }
74         if(empty($message))
75         {
76             $errorName = $rules . "_error";
77             $message = $validation->$errorName ?? [];
78         }
79         $rules = $validation->$rules;
80     }
81     return $this->validator->setRules($rules, $message)->run($input);
82 }
83
84 115
85
86 2.4k pp6-forum/api/app/Controllers/BaseController.php 115:0 Bot
87
88 PHP//L main ✓
```

Dans le screenshot ci-dessus, on y voit trois fichier:

- RegisterRules.php en haut
- Auth.php en bas à gauche
- Base Controller en bas à droite

Register rules permet la définition d’une regles appliqué lors de la connexion du client, règle appelé password verify, qui permet de comparer le mot de passe fournis et celui enregistré en bdd (le mot de passe n’est pas enregistré en du, il est hashé avec une algorithmme sha251).

Le fichier Auth.php est le contrôleur chargé de connecter l'utilisateur. Il fait donc le point entre les éléments de vérifications et l’application. On peut voir à la ligne 67 la définition des règles utilisées sur l’input password. CodeIgniter applique un regex sur ce string de définition de règles afin de permettre une définition rapide et concise des multiples règles applicables à un input. On peut y voir la règle “password_verify[user.password,user.email]” qui prend donc comme argument le mot de passe et l’email renseigné par le client lors de sa connexion. Dans la même fonction à la ligne 77, on fait appelle a une méthode héritée appelée validateRequest. Cette méthode provient de la classe mere BaseController, qui est parent de tous les contrôleurs utilisés dans l’API.

La méthode validateRequest est définie dans la classe mère BaseController (en bas à droite dans l’exemple ci-dessus). Cette méthode utilise les fonctions de CodeIgniter relative à la validation des données. Dans Auth.php, on définit les règles pour chaque données à vérifier provenant du client ainsi que les messages de

retour si ces règles ne sont pas respectées. Une fois les règles et les réponses en cas d'erreur données à la méthode valide Request, elles sont appliquées et selon la valeur de retour de cette méthode, on valide la connexion en retournant un jwt à l'utilisateur ou alors on renvoie un message précisant quel type d'erreur est présente. En sachant que la définition des réponses en cas d'erreur est déjà définie pour les règles natives a Codelgniter (is_unique, required,...).

Création et vérification du json web token

Afin d'éviter à l'utilisateur de donner ses identifiants (email, password) à chaque requête, un jwt lui est transmis lors de sa connexion et/ou lors de son inscription. Ce jwt est généré via un objet appelé helper, qui permet d'ajouter des fonctionnalités à codelgniter qui ne sont pas dans le cadre d'action des composants d'un modèle MVC.

```
use App\Models\UsersModel;
use Config\Services;
use Firebase\JWT\JWT;

function getJWT($req)
{
    if(is_null($req)) {
        throw new Exception('Missing or invalid request token');
    }

    return explode(" ", $req)[1];
}

function validateJWT($encodedToken)
{
    $key = Services::getSecretKey();
    $decodedToken = JWT::decode($encodedToken, $key, ['HS256']);
    $users = new UsersModel();
    $user = $users->getUserByEmail($decodedToken->email);
}

function createJWT(string $email, int $role=0)
{
    $createdAt = time();
    $timeToLive = getenv('JWT_TIME_TO_LIVE');
    $expireAt = $createdAt + $timeToLive;
    $payload = [
        'email' => $email,
        'exp'   => $expireAt,
    ];

    return JWT::encode($payload, Services::getSecretKey());
}

function getTokenInfo($req)
{
    $encodedToken = $req->getServer('HTTP_AUTHORIZATION');
    $encodedToken = getJWT($encodedToken);
    $key = Services::getSecretKey();
    $decodedToken = JWT::decode($encodedToken, $key, ['HS256']);
    $users = new UsersModel();
    $user = $users->getUserByEmail($decodedToken->email);
    return [$user["id"], $user["role"]];
}
```

La création de token s'effectue lors de l'inscription de l'utilisateur au service. Ce dernier fournit donc son mot de passe ainsi que son email de connexion. Ce sont ces données qui seront enregistrées dans le token pour prouver l'identité de l'utilisateur.

Nous utilisons Firebase pour s'occuper de la génération du jwt. Nous lui fournissons les données à crypter, c'est-à-dire son email et la durée de vie du token dans ce qui est appelé le payload, ainsi que la clé secrète enregistrée en variable d'environnement pour sécuriser le tout.

Une fois le token créé, le controller Auth.php retourne les identifiants en clair de l'utilisateur, soit son email, la durée de vie du token ainsi que le dit token.

Lorsque le client fera une requête avec son token, un filtre de validation est utilisé pour vérifier la validité de ce dernier, en utilisant la fonction validate JWT du fichier jwthelper.php

Front end React-native

Le front end de l'application est donc développé en react native. La majeure partie de cette tâche se trouve dans la gestion et l'organisation des pages entre elles. Pour obtenir un résultat intuitif et flexible, nous utilisons la librairie react-navigation qui nous offre toutes sortes d'outils pour permettre au client final d'accéder intuitivement aux données mise à sa disposition.

Chaque page réact a donc toutes les informations nécessaires à l'affichage des données qu'elle représente.

Nous utilisons les contextes pour générer et mettre à jour les données en temps réel (ce qui est la feature principale de réact avec son virtual dom).

```
export default function Login({ navigation }) {
  const { signIn } = useContext(UserContext);

  const [email, setEmail] = React.useState("unEmailPasCommeLesAutre@email.com");
  const [password, setPassword] = React.useState("unPasswordPasCommeLesAutres");
  const [signingIn, setSigningIn] = React.useState(false);
  const [error, setError] = React.useState(null);

  return (
    <View style={common.container}>
      {!signingIn ? (
        <View style={login.container}>
          <Text style={login.header, login.spacing}>Connexion</Text>
          {error ? <Text style={{ color: "red" }, login.spacing}>{error}</Text> : null}
          <Text style={login.fieldName, login.spacing}>Nom d'utilisateur</Text>
          <TextInput style={login.input, login.spacing} onChangeText={setEmail} />
          <Text style={login.fieldName, login.spacing}>Mot de passe</Text>
          <TextInput style={login.input, login.spacing} onChangeText={setPassword} />
          <Button
            onPress={async () => {
              setSigningIn(true);
              try {
                await signIn(email, password);
                setSigningIn(false);
                navigation.navigate("Home");
              } catch (e) {
                setError(e.message);
                setSigningIn(false);
              }
            }}
            title="Connexion"
            style={{ backgroundColor: colors.purple }}
          />
        </View>
      ) : (
        <ActivityIndicator size="large" color={colors.purple} />
      )}
    </View>
  );
};
```

Ci-dessus est la page de connexion de l'application, l'utilisation des states permet de stocker les données entrées par l'utilisateur avant de les envoyer via axios à l'api. De plus, l'utilisation de JSX permet l'utilisation

d'opérateur ternaire dans le rendu de la page. Ainsi on peut voir que nous utilisons des variables nommée "signingIn" pour décider d'afficher le logo de chargement ou la page de connexion selon l'étape à laquelle est l'utilisateur.

L'ajax et les contextes

Une fois l'utilisateur connecté, les informations de connexion, notamment le JSON web token fournis, est enregistré dans un contexte nommé UserContext. Lorsque nous devons accéder à des données provenant de l'api, nous utilisons donc le token enregistré dans ce contexte pour permettre la connexion au service et ainsi obtenir les informations demandées.

Pour faire les requêtes, nous utilisons Axios, qui allie flexibilité, performance et lisibilité.

```
export default function Logs({ navigation }) {
  const { state } = useContext(UserContext);
  const [logs, setLogs] = useState(null);

  useEffect(() => {
    (async () => {
      try {
        const {
          data: { data },
        } = await axios({
          url: "/logs",
          method: "get",
          headers: {
            Authorization: `Bearer ${state.token}`,
          },
        });

        setLogs(data);
      } catch (e) {
        setLogs(false);
      }
    })();
  }, []);
}
```

React native propose l'utilitaire useEffect, qui permet de lancer (lors du chargement de la page ou lors du changement de certaines données) des requêtes asynchrone aux serveur (ou d'autres effets à activer lors d'événement de la page).

Dans l'exemple ci-dessus, on peut voir l'utilisation d'axios pour obtenir les logs disponibles. On peut voir l'utilisation du header Authorization avec le token utilisateur pour permettre la requête.

Les tests de déploiement

Pour permettre un déploiement sans douleur, une batterie de test est exécutée lors d'un push sur la branche prod. Ces tests sont automatiquement exécutés via les webhooks de github's qui déclenchent un script côté serveur qui exécute les tests définis par l'outil Newman.

Les tests effectués doivent couvrir l'ensemble des cas possibles et envisagés par l'application. Par l'ensemble des cas possibles, j'entend tester tous les cas d'erreurs disponibles (c'est à dire tester les cas que l'api est censé gérer via ses filtres et règles) mais aussi des test sur la base de donnée (comme l'ajout de colonne en double ou bien même simuler l'incapacité de se connecter à la base de donnée), afin d'avoir un retour formaté quoi qu'il arrive et ne pas se retrouver avec une erreur 404 ou 500 qui sera à la fois désastreuse pour l'expérience utilisateur mais aussi rendrait la tâche de debugging beaucoup plus dur pour les utilisateurs de l'API.


Exemple de test de newman

```
"request": {
  "method": "POST",
  "header": [],
  "body": {
    "mode": "urlencoded",
    "urlencoded": [
      {
        "key": "username",
        "value": "{{Username}}",
        "type": "text"
      },
      {
        "key": "password",
        "value": "{{Password}}",
        "type": "text"
      },
      {
        "key": "passwordConfirm",
        "value": "{{Password}}",
        "type": "text"
      },
      {
        "key": "email",
        "value": "{{Email}}",
        "type": "text"
      }
    ]
  }
},
```

```

"url": {
  "raw": "http://localhost:8080/auth/register",
  "protocol": "http",
  "host": [
    "localhost"
  ],
  "port": "8080",
  "path": [
    "auth",
    "register"
  ],
  "query": [
    {
      "key": "username",
      "value": "samuelJolytesteur",
      "disabled": true
    },
    {
      "key": "password",
      "value": "testdemdp",
      "disabled": true
    },
    {
      "key": "passwordConfirm",
      "value": "testdemdp",
      "disabled": true
    },
    {
      "key": "email",
      "value": "samueljolya0@gmail.com",
      "disabled": true
    }
  ]
},
"description": "register an account to the api"
},
"response": []
},
{
  "name": "Login",
  "event": [
    {
      "listen": "test",
      "script": {
        "exec": [
          "pm.test(\"Status code is 200\", function () {",
          "    var jsonData = pm.response.json();",
          "    pm.environment.set(\"id_user\", jsonData.user.id);",
          "    pm.environment.set(\"Username\", jsonData.user.username);",
          "    pm.environment.set(\"Email\", jsonData.user.email);",
          "    pm.environment.set(\"Token\", jsonData.token);",
          "    pm.response.to.have.status(200);",
          "});",
          "],
          "type": "text/javascript"
        }
      ]
    }
  ]
},
],

```



Le test ci-dessus vérifie si la connexion à l'API marche pour un utilisateur précédemment enregistré par newman comme étant un administrateur. Ce test vérifie donc que la connexion s'effectue avec les identifiants définis spécialement pour la batterie de test en tant qu'administrateur. En effet, nous utilisons plusieurs variables d'environnement de test afin de vérifier l'interaction en tant qu'administrateur mais aussi en tant qu'utilisateur.

De plus pour éviter tout problème de corruption et/ou de duplicata de données avec la base de donnée en production, ces tests sont fait sur une base de donnée spécifique, évitant toute forme de problème lié au mélange des données de productions avec celles des tests.

Chaque test est accompagné d'un script de vérification qui donne la valeur finale des dis tests. Ces scripts sont fait a la main.

une fois les tests terminés, la base de donnée de test est purgée, un log des tests effectué est enregistré et un git pull origin prod s'effectue sur le serveur si tout les tests ont été validés.