Roamin Gouraud Mathys Meunier Nathan Ferry

Rapport R5.07 / R5.08

Sommaire

Sommaire	2
Introduction	3
FrameWorks	3
React Native	3
MongoDB	3
Langages utilisés	3
TypeScript & NPM	3
Golang	4
Coopération et déploiement	4
Git & gitlab	4
Digital Ocean	4
Tests et qualité de code	5
Tests	5
Analyze statique	6

Introduction

Dans ce rapport, vous trouverez l'explication de plusieurs de nos choix techniques ainsi que les différents outils et méthodes que nous avons utilisé afin de rendre notre application maintenable.

FrameWorks

React Native

Ayant décidé de rendre l'application disponible au plus grand nombre, nous avons utilisé le framework React Native, qui permet de ne développer qu'une seule fois mais de déployer sur Android, los ou Windows. Cela nous a donc laissé le choix entre javascript et typescript pour développer l'application, choix qui sera détaillé plus tard.



MongoDB

Afin de garder les données du gouvernement que nous avons traitées, nous avons mis en place un serveur mongoDB, ainsi que du sharding et de la réplication.



Langages utilisés

TypeScript & NPM

Comme mentionné précédemment, nous avions le choix entre javascript et typescript. Si les deux fonctionnent avec Node, qui permet d'automatiser le build et le lancement de l'application, et profite du gestionnaire de paquet NPM, qui rend l'ajout de dépendances presque automatique, notre choix s'est porté vers typescript, car il est plus agréable, plus sur et dans une certaines mesure, plus testable. La convention veut aussi que les applications réalisées via React Native soient écrites en typescript.



Golang

SI l'application utilisateur à été réalisée en typescript, l'application backend, de populer de la base de donnée ainsi que de servir d'API pour récupérer ces données à été réalisé en GO. Nous avons choisi ce langage pour sa simplicité d'utilisation et sa fiabilité, GO étant un des langages référence dans la création d'API restFul. Il offre également une très bonne interaction avec mongoDB ce qui permet d'interagir facilement avec la base de données.



Coopération et déploiement

Git & gitlab

Travaillant en groupe de trois, la mise en commun de nos avancements s'est faite via Gitlab. Nous avons utilisé l'outil afin de sauvegarder et de mettre au commun notre code, mais nous n'avons pas pris le temps d'utiliser l'outil à son plein potentiel, par manque de connaissances approfondies de celui-ci.



Voici le lien vers le GitLab de notre groupe : https://gitlab.univ-nantes.fr/E217817H/sae5a01

Digital Ocean

Afin d'héberger nos services (base de données, service backend) nous nous sommes tournés vers des prestataires externes, les serveurs de l'IUT étant trop instable pour que leur utilisation soit agréable. Nous avons donc choisi d'utiliser le services de DigitalOcean, car celui-ci est facile à mettre en place, et qu'il est gratuit via le plan github student.



Tests et qualité de code

Tests

Pour ce qui est de l'application, nous avons seulement testé l'application via le visuel que nous obtenions. Des tests auraient pu être réalisés via les outils mis à disposition par React, mais le manque de temps nous a empêché d'apprendre à nous servir de ces outils, le test d'une application React Native n'étant pas trivial.

Nous avons cependant pu réaliser quelques tests sur l'API et les méthodes qu'elle utilise. L'application en Go ayant été réalisée quelques temps avant de connaître tous les besoins de l'application mobile, beaucoup de fonctions présentes ont été créées mais ne sont pas nécessaires au bon fonctionnement de l'application, nous avons donc jugé superflu de les tester.

Pour réaliser nos tests, nous avons utilisé la librairie de base fourni par Go, la librairie "assert" permettant des assertions plus complexes ainsi que l'outil intégré au compilateur pour lancer ces tests.

Cela nous permet notamment de tester la nullité d'un pointeur ou encore le type de certaines variables :

```
func TestNewCisData(t *testing.T) {
   values := testValues
   cisData := NewCisData(values)
   assert.NotNil(t, cisData)
}

func TestNewCisData2(t *testing.T) {
   values := testValues
   cisData := NewCisData(values)
   var Int int
   var Date time.Time
   assert.IsType(t, Int, cisData.CIS)
   assert.IsType(t, Date, cisData.MADate)
}
```

```
Params Authorization Headers (6) Body Pre-request Script Tests Settings

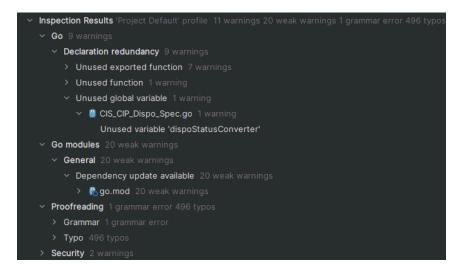
1 pm.test("Status code is 200", function () {
2  pm.response.to.have.status(200);
3  });
4 pm.test("Response time is less than 3s", function () {
5  pm.expect(pm.response.responseTime).to.be.below(3000);
6  });
```

Nous avons également testé l'API via un outil externe : PostMan, qui permet d'effectuer des requêtes HTTP

et de tester le résultat de celle-ci.

Analyze statique

Enfin, nous avons pu réaliser une analyse de notre code via les outils de la suite JetBrain, intégrés dans nos idées.



Pour ce qui est de l'application réalisée en Go, l'analyse du code ne nous fournit que peu d'informations,

premièrement car le code est beaucoup plus petit et présente donc moins de chance de faire des erreures, mais aussi car l'IDE Goland (équivalent

d'Intellij pour Go) est extrêmement strict sur la façon d'écrire du code, Go étant un langage avec énormément de conventions.

De l'autre côté, l'analyse du code de l'application écrite à en typescript révéler beaucoup plus de points pouvant être corrigés ou améliorés. Bien sur. l'application mobile représente une quantité de code bien plus importante flexibilité mais la typescript et son manque de riqueur vis-a-vis de comment écrire du code en fait un langage peu pratique lorsque l'on parle de maintenabilité et de lisibilité. L'analyse nous a donc permit de trouver à quelles endroits nous améliorer pouvons notre code pour le rendre plus simple à reprendre pour d'autres développeurs

Inspection Results 'Project Default' profile 98 warnings 45 weak warnings 5,749 typos > General 12 weak warnings JavaScript and TypeScript 98 warnings 33 weak warnings > Async code and promises 9 weak warnings > Control flow issues 19 warnings Data flow 2 warnings Redundant local variable 2 warnings > **IndexUserPage.tsx** 1 warning > 🤀 Mail.tsx 1 warning > ES2015 migration aids 2 weak warnings ✓ General 3 warnings 8 weak warnings > Deprecated symbol used 6 weak warnings > Mismatched JSDoc and function signature 3 warnings > Unresolved reference 2 weak warnings > Imports and dependencies 46 warnings > Potentially confusing code constructs 4 warnings Potentially undesirable code constructs 5 warnings Comma expression 5 warnings > **@ CalendarIndex.tsx** 4 warnings > **IndexUserPage.tsx** 1 warning > TypeScript 11 weak warnings > Unused symbols 19 warnings > Validity issues 3 weak warnings Proofreading 5,749 typos ▼ Typo 5,749 typos > AddMedicine.tsx 1 typo > **AutoComplete.tsx** 3 typos > **@ CalendarIndex.tsx** 88 typos > Is Calendarinterface.ts 2 typos > 🥸 Company.tsx 1 typo > **@ ConfirmXL.tsx** 1 typo > s dataManager.ts 4 typos > {} defaultIcon.json 2,145 typos > {} defaultMedicine.json 1 typo

{} defaultPatient.json 272 typos {} defaultSaveForTest.json 2,146 typos