

Table des matières

Table des figures	2
1 Architecture Globale	4
2 Architecture détaillée	7
2.1 Partie client	7
2.1.1 Architecture intérieur React Native	7
2.1.2 Structure Détaillée de la partie front	8
2.2 Partie serveur	13
2.2.1 Architecture intérieur Nest js	13
2.2.2 Structure Détaillée de la partie back	14
2.2.3 Diagramme de classe	19
2.2.4 Module temps réel	19
2.2.5 Module traduction	20
2.3 Partie données	22
2.3.1 Outil de gestion	22
2.3.2 Diagramme de base de données	22

Table des figures

1.1	Architecture globale de l'application	4
1.2	Logo de React	5
1.3	Logo de Nest JS	5
1.4	REST API	6
1.5	Logo de GrapheQl	6
2.1	Diagramme de classe	19
2.2	Le modèle non bloquant de Node JS	20
2.3	Diagramme de base de données	22
2.4	Exemple de structure de données	23

Introduction générale

Après avoir faire l'étude fonctionnelle on entame l'étude technique dans laquelle on va décrire l'architecture de l'application et les technologies à utiliser.

C'est une phase très importante qui complète le cahier des charges et l'étude fonctionnelle pour avoir une vision claire sur l'application sur tout les volés.

A la fin de cette phase, la première partie de projet (préparation) sera prête et on peut facilement commencer la partie de réalisation suivant les normes qu'on a déjà mis dans les trois documents de la la partie de préparation.

Chapitre 1

Architecture Globale

Après avoir choisit les technologies nécessaires pour le bon fonctionnement de l'application et avant de passer à la partie développement, nous allons entamer l'architecture globale de projet ainsi que les différents modules existants dans l'application. Notre application sera divisée en deux parties, une partie front mobile en React native et une partie back en Nest JS .

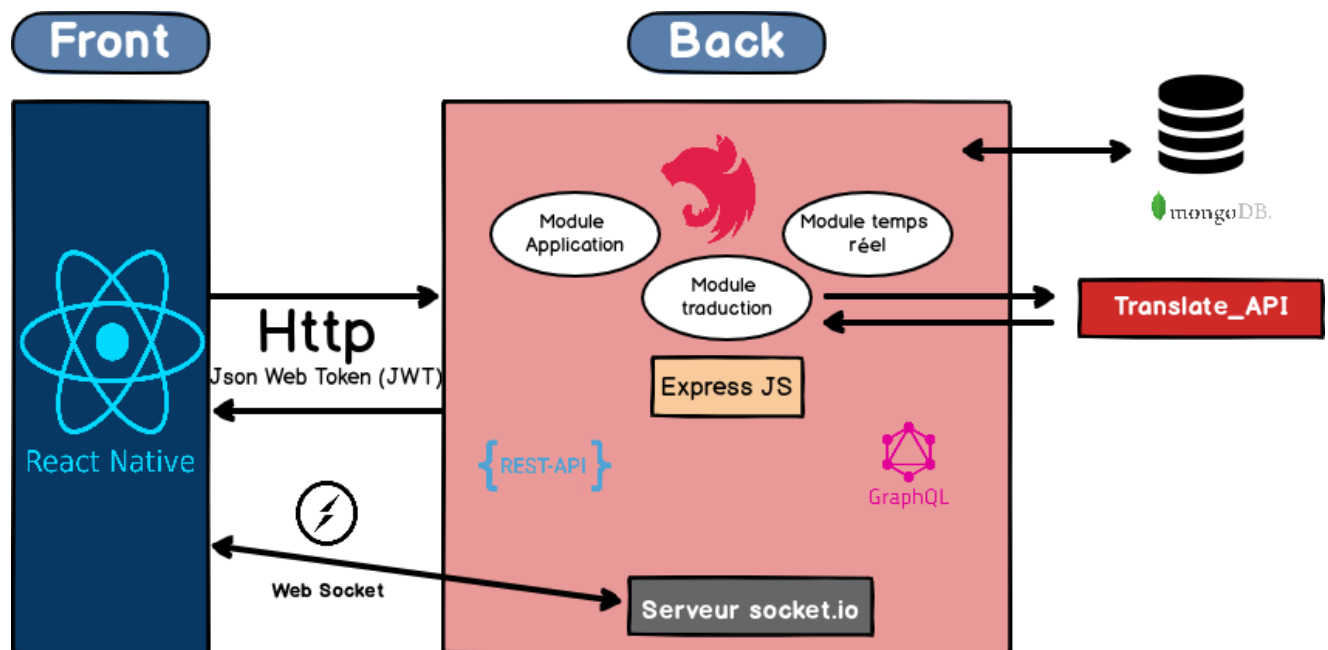


FIGURE 1.1 – Architecture globale de l'application

- La partie front (client) de l'application sera développée en React Native qui est un framework mobile open source créé par Facebook. Il est utilisé pour développer des applications pour Android, iOS, Web et UWP en permettant aux développeurs d'utiliser React avec les capacités de la plateforme native . *** plus de détails sur cette technologie dans la partie annexe**

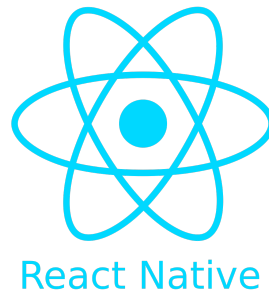


FIGURE 1.2 – Logo de React

- La partie Back de l'application sera travaillée avec Nest.js qui est un framework Node.js côté serveur pour créer des applications efficaces, fiables et évolutives . Fortement inspiré par Angular, Il fournit aux applications dorsales une structure modulaire pour organiser le code en modules séparés. *** plus de détails sur cette technologie dans la partie annexe**



FIGURE 1.3 – Logo de Nest JS

- Api rest (Representational state transfer) est un style architectural logiciel qui définit un ensemble de contraintes à utiliser pour créer des services Web. Les services Web conformes au style architectural REST, appelés services Web RESTful, assurent l'interopérabilité entre les systèmes informatiques sur Internet. *** plus de détails sur cette technologie dans la partie annexe**



FIGURE 1.4 – REST API

- GraphQL est un langage de requête et de manipulation de données open source pour les API, et un runtime pour répondre aux requêtes avec des données existantes. *** plus de détails sur cette technologie dans la partie annexe**

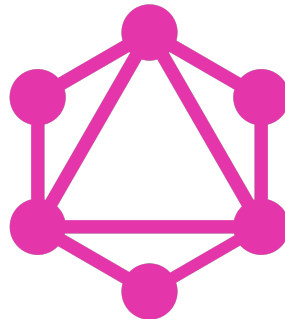
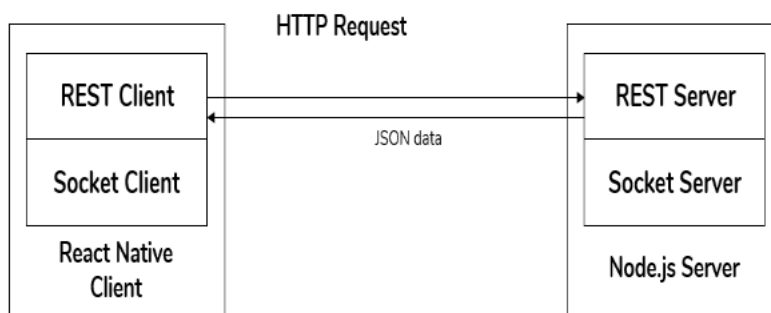


FIGURE 1.5 – Logo de GrapheQl

- La partie communication est la phase reliant entre le front et le back suivant des requêtes http avec des réponses Json selon le protocole http (JWT) , pour assurer un échange en temps réel les web socket s'intègre dans cette partie en respectant le protocole http. *** plus de détails sur cette technologie dans la partie annexe**



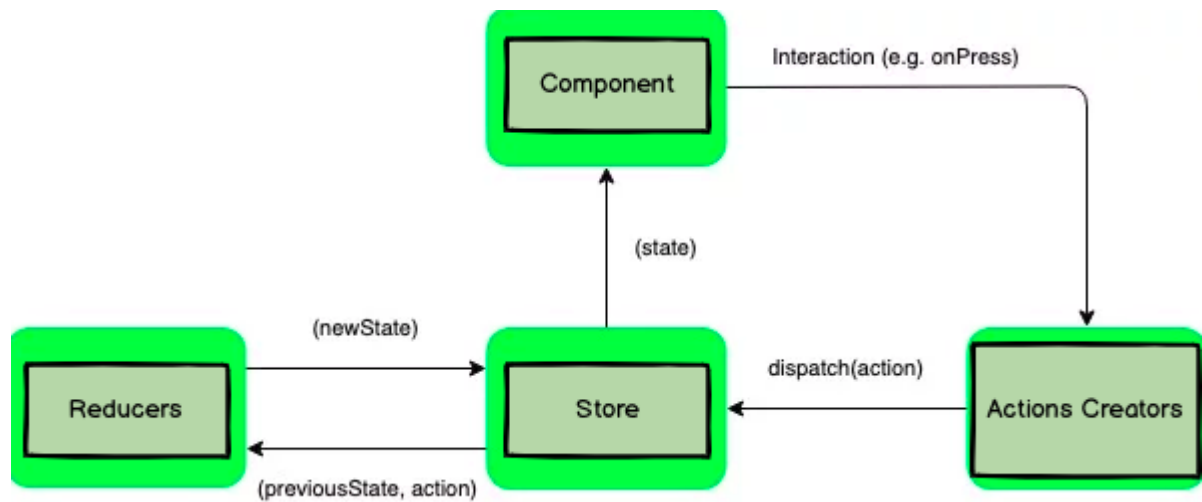
Chapitre 2

Architecture détaillée

2.1 Partie client

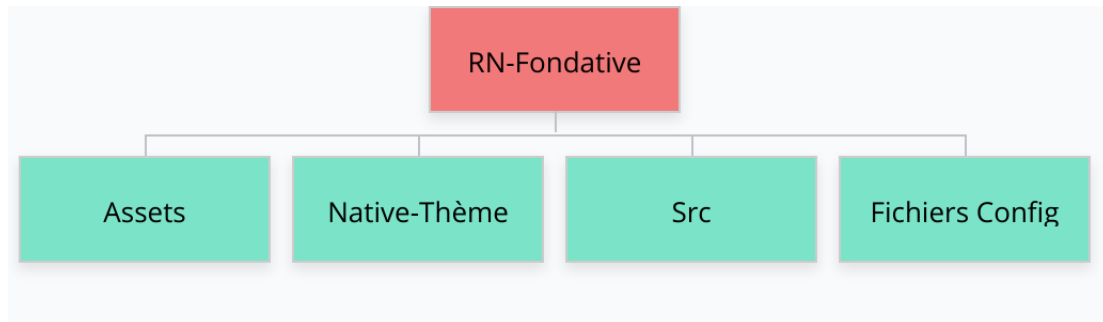
2.1.1 Architecture intérieur React Native

la différence la plus apparente entre le développement d'une application entièrement native et une application RN est l'architecture. Ici on va présenter l'architecture de RN avec l'utilisation du conteneur d'état REDUX.



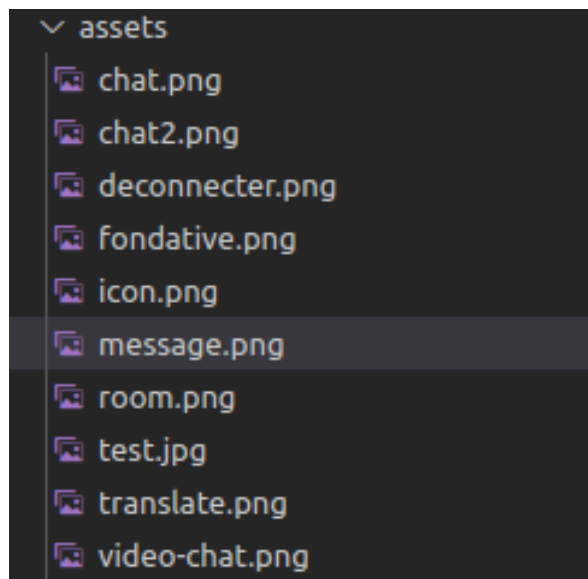
2.1.2 Structure Détaillée de la partie front

Dans cette partie on va détailler l'architecture de projet en présentant l'arborescence de chaque dossier avec un Zoom sur chaque dossier .

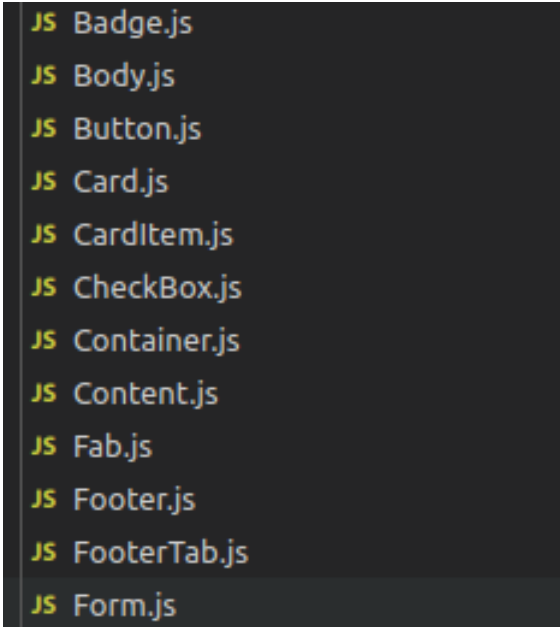


- Les composants du dossier de projet

- Assets : contient les icônes et les images.

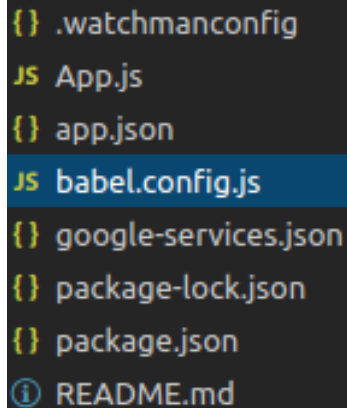


- Native-Thème : contient les fichiers qui contrôlent les couleurs et les styles de l'application



```
JS Badge.js
JS Body.js
JS Button.js
JS Card.js
JS CardItem.js
JS CheckBox.js
JS Container.js
JS Content.js
JS Fab.js
JS Footer.js
JS FooterTab.js
JS Form.js
```

- Fichiers Config : Les fichiers de configurations tel que **package.json** et **babel.config.js**

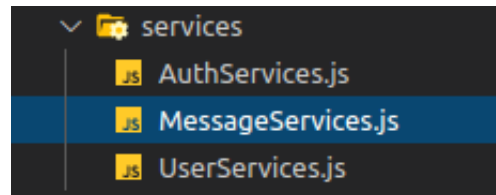


```
{ } .watchmanconfig
JS App.js
{ } app.json
JS babel.config.js
{ } google-services.json
{ } package-lock.json
{ } package.json
① README.md
```

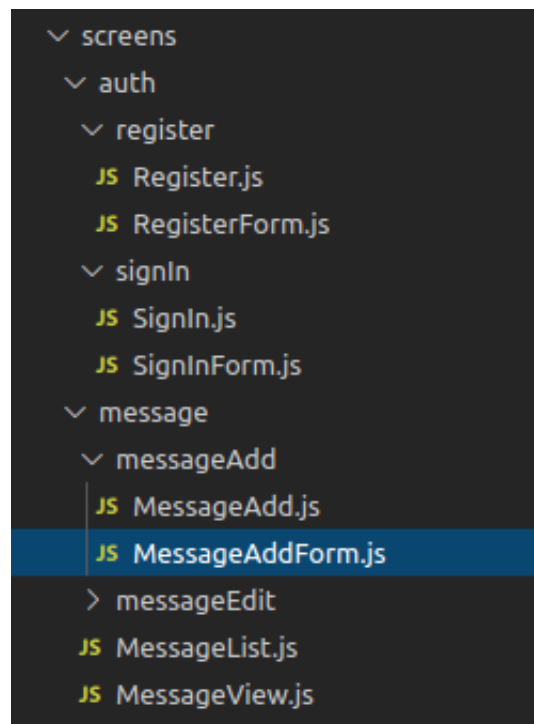
- Src : c'est le dossier qui contient les composants les plus importants de l'application



- **Components** : C'est là que nous placerons tous nos composants React partagés. Habituellement, ces composants sont ceux que nous appelons «dummy», qui n'ont pas de logique d'états et peuvent être facilement réutilisés dans l'application.
- **Services** : Ce sont les fonctions qui encapsulent les appels d'API.



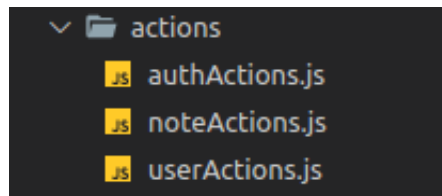
- **Screens** : Ce sont nos écrans d'application, ceux que nous naviguons de l'un à l'autre. Ce sont aussi des composants React, mais ce sont ceux que nous appelons des conteneurs, car ils contiennent leur propre état.



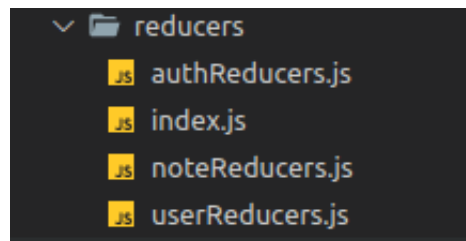
- **Navigation** : La gestion de navigations entre les screens à l'aide d'une décomposition modulaire sous forme des stack

Les dossiers relatifs à Redux :

- Actions :les actions sont des événements. Ils sont le seul moyen d'envoyer des données de votre application vers notre store Redux.



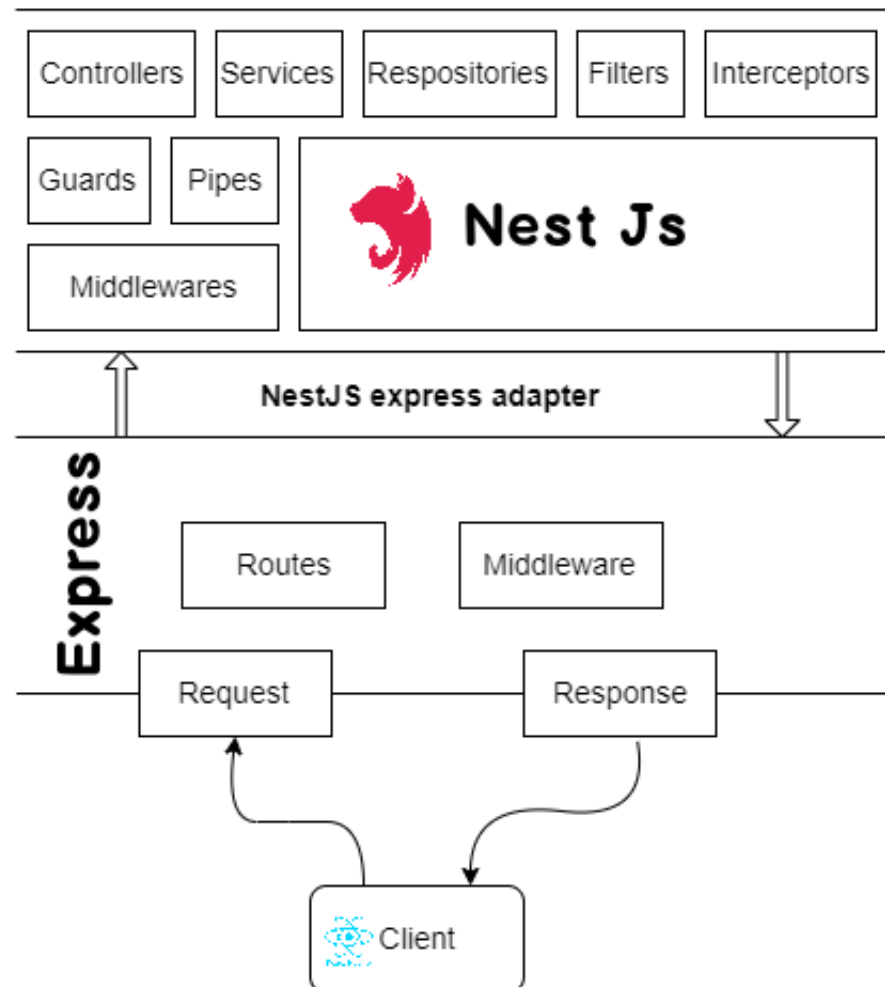
- Reducers :Les réducteurs sont de simples fonctions qui prennent l'état actuel d'une application, effectuent une action et renvoient un nouvel état



- Store : C'est le magasin qui contient l'état de l'application.

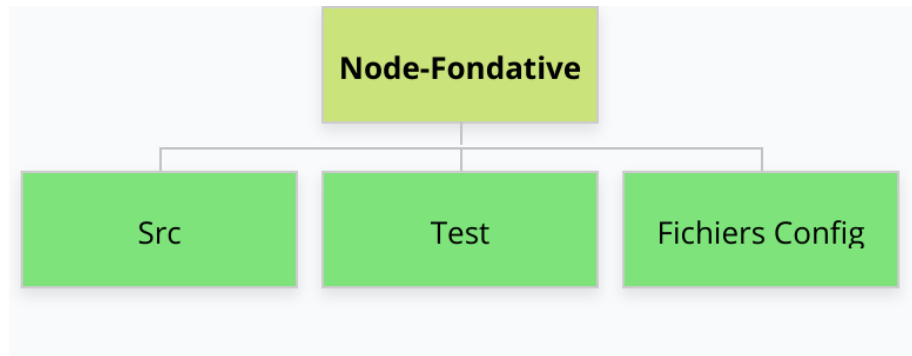
2.2 Partie serveur

2.2.1 Architecture intérieur Nest js



2.2.2 Structure Détaillée de la partie back

Dans cette partie on va détailler l'architecture de projet en présentant l'arborescence de chaque dossier avec un Zoom sur chaque dossier .



- Fichiers Config :
Ce dossier contient les fichiers de configurations tel que **Package.json** , **yarn.lock** et **nodemon.json**

```

≡ .env.dist
◆ .gitignore
≡ .prettierrc
{} nest-cli.json
{} nodemon-debug.json
{} nodemon.json
JS ormconfig.js
{} package.json
① README.md
JS tsconfig-dist.js
TS tsconfig.json
{} tsconfig.spec.json
{} tslint.json
🔒 yarn.lock
  
```

- Test :
ce dossier contient les fichiers de test .Spec

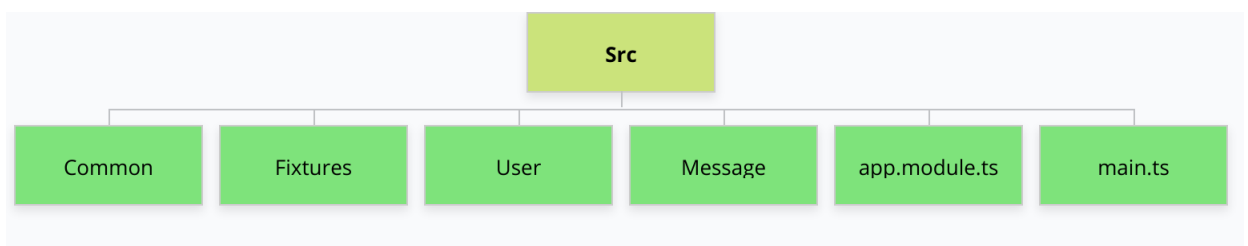
```

▼ test
  TS app.e2e-spec.ts 6
  {} jest-e2e.json
  
```

- Src :
Src est le dossier principale de l'application qui encapsule tous les dossiers assurant le bon fonctionnement de la partie Back

```

▼ src
  > common
  > fixtures
  > Message
  > user
  TS app.module.ts
  TS config.schema.ts
  TS main.ts
  
```



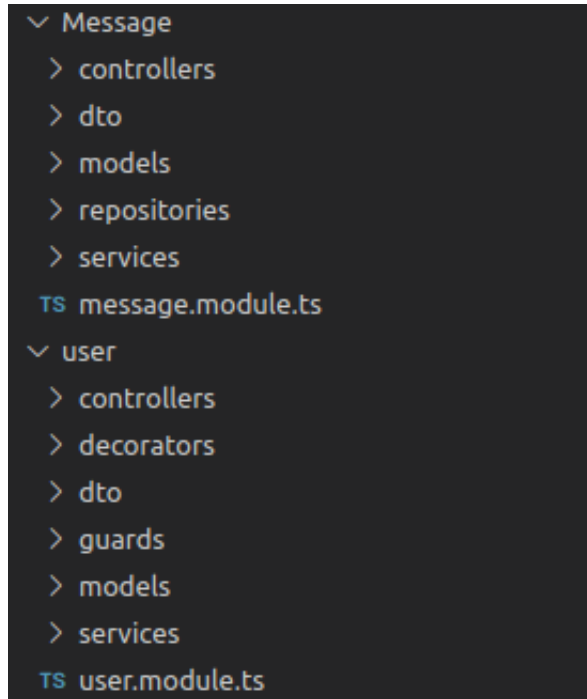
- **Common** : ce dossier contient tous les sous-dossiers communs pour tous les modules et qu'on peut les utiliser pour toute l'application telque les services ,les exceptions et les erreurs

```
▼ common
  > config
  > decorators
  > errors
  > exception-filters
  > filters
  > interceptors
  > services
  > storage
  > validations
  TS common.module.ts
```

- **fixtures** : dans le dossiers fixtures on contrôle nos composants créés (dans notre cas user et message)

```
▼ fixtures
  TS create-message.fixture.ts
  TS create-users.fixture.ts
  TS run.ts
```


- **Message / user** : ce sont nos modules créés dont chacun possède des sous-dossiers pour assurer son comportement dans le processus de nest js



- Module User :

	Front		Back	
Fonctionnalité	Screen	Traitement	Contrôleur	Service
Consulter Profil	UserView	UserReduces userActions UserServices	UserController (getUser())	UserService (findUser ())
Modifier mon profil	UserView	UserReduces userActions UserServices	UserController (putUser())	UserService (updateUser ())
Chercher un Ami	UserSearchView	UserReduces userActions UserServices	UserController (getUser())	UserService (findUser ())
Consulter mes amis	UsersListView	UserReduces userActions UserServices	UserController (getAllUsers())	UserService (findUsers ())

- Module Message :

Fonctionnalité	Front		Back	
	Screen	Traitement	Contrôleur	Service
Consulter les messages	HomeView	MessageReducers MessageActions MessageServices	MessageController (getAllMsgs())	MessageService (findMessages())
Envoyer un message	MessageView	MessageReduces MessageActions MessageServices	MessageController (postMsg())	MessageService createMessage()
Supprimer un message	MessageView	MessageReduces MessageActions MessageServices	MessageController (deleteMsg())	MessageService (removeMsg ())
Traduire un message	MessageView	MessageReduces MessageActions MessageServices	MessageController (translateMsg())	TranslationApi

- **app.module.ts** : c'est le module principale de l'application dans lequel on déclare nos modules créés comme user et message par exemple
- **main.ts** : c'est le fichier principale de l'application dans le quel on importe le module principale , les autres modules et les configurations de l'application

2.2.3 Diagramme de classe

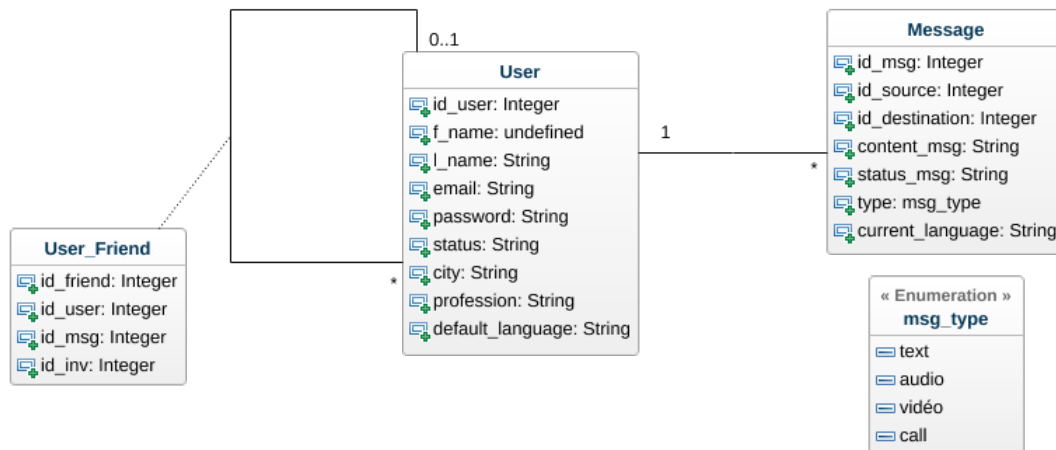


FIGURE 2.1 – Diagramme de classe

2.2.4 Module temps réel

La gestion d'évènements asynchrones est l'une des caractéristiques qui rend le node js un environnement d'exécution très performant et grâce à cette approche on le considère comme le meilleur choix d'une application de temps réel

- Non blocking I/O : Une application Node JS est composé d'un seul thread en utilisant un modèle non bloquant. Cette spécificité est importante et impactera totalement la façon de développer une application sous NodeJS en utilisant un système d'évènements asynchrones.

- La gestion d'événements asynchrones proposée par le Node JS permet de mettre en place des applications en temps réel très performantes comme des messagerie instantanées, systèmes de notifications, jeux en ligne ...

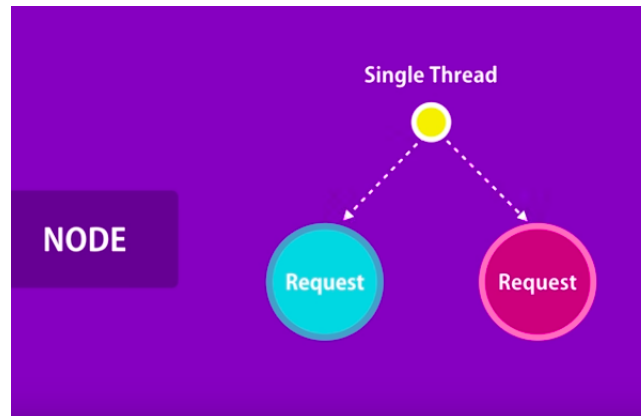
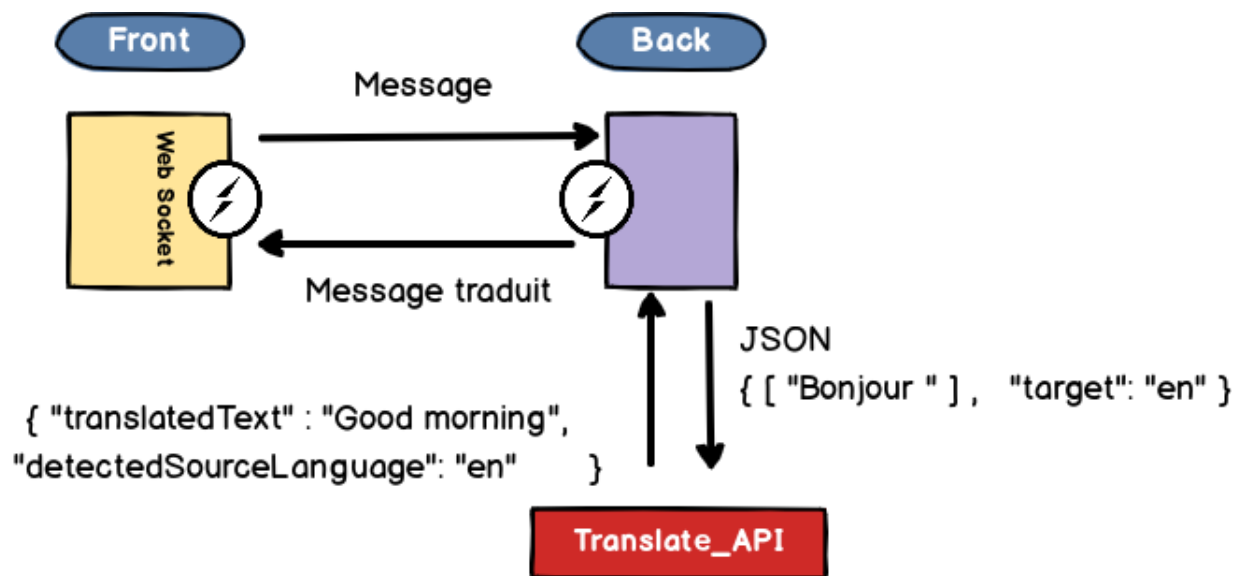


FIGURE 2.2 – Le modèle non bloquant de Node JS

2.2.5 Module traduction



- Étude comparative de l'outil de traduction :

Outil de traduction	Tarif	Facilité
Amazon Translate API	millions de caractères gratuits par mois pendant les 12 premiers mois	Simple à utiliser grâce à SDK AWS avec une documentation suffisante
API Azure Text Translator	10 dollars par million de caractères avec 2 millions de caractères gratuits	un peu difficile après la nouvelle Maj des SDK azure mais elle présente un code source clair et bien commenté
Google translate API	20 dollars par million de caractères , 10 dollars d'utilisation sont offertes par Google	facile à implémenter et l'utilisation des SDK Google est très compatible avec les projets JavaScript
IBMWatsonLanguageTranslator	Il existe une version Lite, qui est le niveau gratuit.1 million de caractères par mois sans frais et les services Lite sont supprimés après 30 jours d'inactivité.	Le SDK JavaScript d'IBM Watson Language Translator possède de loin la syntaxe d'authentification la plus détaillée avec une documentation facile à suivre
Yandex Translate	15 dollars pour million de caractères	Documentation pas assez claire
Unbabel Translator	Gratuit	Documentation insuffisante
Tild MT API	Payant	Simple mais documentation insuffisante

2.3 Partie données

2.3.1 Outil de gestion

Dans la parties de gestion de données on utilisera Mongo DB qui est un programme de base de données orienté document multiplateforme . Classé comme un programme de base de données NoSQL , il utilise des documents de type JSON.

2.3.2 Diagramme de base de données

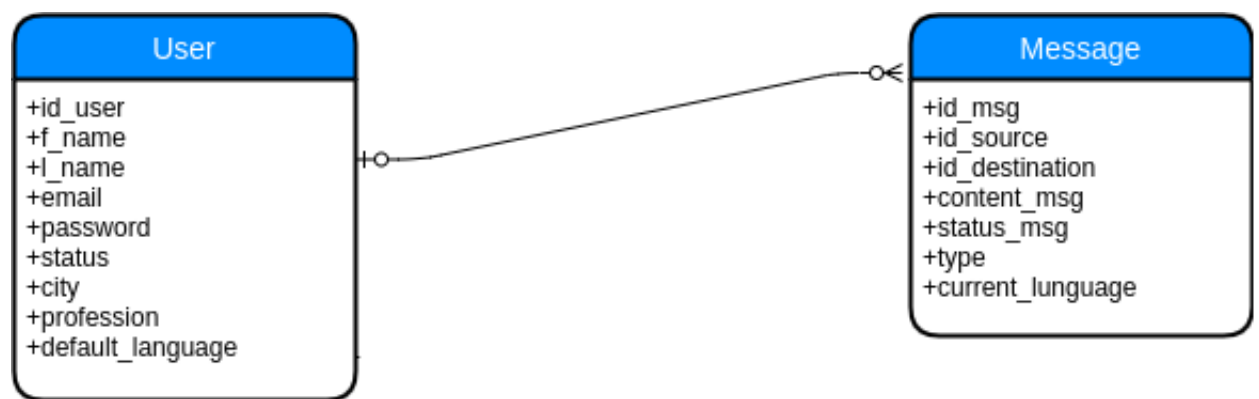


FIGURE 2.3 – Diagramme de base de données

```
/* User */
{
  "id": 1,
  "f_name": "Baligh",
  "l_name": "Ammari",
  "email": "b@a.com",
  "password" : "123456789" ,
  "status" : "connecté",
  "city": "Tunis",
  "profession" : "ingénieur",
  "default_language" : "fr"
}
{
  "id": 2,
  "f_name": "Kais",
  "l_name": "Bouchoucha",
  "email": "k@b.com",
  "password" : "987654321" ,
  "status" : "connecté",
  "city": "Tunis",
  "profession" : "ingénieur",
  "default_language" : "fr"
}
```

FIGURE 2.4 – Exemple de structure de données

Conclusion

e Dans cette étude, on a réussi à implémenter une architecture globale de l'application en expliquant les différentes technologies utilisées pour ce projet , puis on a essayé de faire un zoom sur chaque couche de l'application à part et chaque module tout seul pour mieux comprendre le déroulement de système de l'application entière. maintenant tout est clair et il ne nous reste qu'entamer la phase de développement et cette phase sera le fruit de ces trois premiers sprints : cahier des charges, étude fonctionnelle et l'étude technique.

Annexe

React native

- Les avantages d'utilisation de react native :

- Rapidité d'exécution grâce au DOM virtuel (le DOM ne met à jour que le composant ayant besoin d'être modifié)
- Un code propre et maintenable
- Des composants réutilisables
- Une communauté active
- Un seul code pour iOS et Android

Nest JS

- Les avantages d'utilisation de Nest JS

- Nest JS parce que notre application se base généralement sur les échanges temps réel et Node JS est recommandé pour cela et Nest JS est l'un de ses Framework performants et facile à utiliser
- Non blocking I/O : Une application Node JS est composé d'un seul thread en utilisant un modèle non bloquant. Cette spécificité est importante et impactera totalement la façon de développer une application sous NodeJS en utilisant un système d'événements asynchrones.
- La gestion d'événements asynchrones proposée par le Node JS permet de mettre en place des applications real-time très performantes comme des chats, systèmes de notifications, jeux en ligne ...
- La librairie NPM : NPM signifie "Node package manager" et est une librairie de modules pour l'environnement NodeJS basé sur le partage et la participation de la communauté. NPMJS regroupe plusieurs dizaines de milliers de modules et permet de faciliter le développement d'une application.

API REST

- Les avantages d'utilisation de REST :

- Un couplage plus faible entre le client et le serveur comparé aux méthodes du type RPC Remote Procedure Call comme SOAP ;
- Une standardisation des APIs (Application Programming Interface) pour une facilité d'utilisation
- Une grande tolérance à la panne

Graphe Ql

- Les avantages d'utilisation de GrapheQl :

- Une accélération du développement frontend en utilisant un paradigme déclaratif
- Un langage de requête qui est : protocole, langage de programmation, client agnostique. GraphQL tourne sur toutes les configurations et pour tous les cas d'usage.
- Plus de prédictibilité. On sait à l'avance grâce au contrat d'interface ce qu'il est possible de requêter et de quels types de données vont être retournées.
- Plus d'éco-conception. Il n'y a pas d'excès de données et plusieurs requêtes peuvent être regroupées dans un seul appel HTTP par exemple, ce qui permet une réduction des charges réseaux.

Socket.io

Socket.IO est une bibliothèque JavaScript pour les applications Web en temps réel. Il permet une communication bidirectionnelle en temps réel entre les clients Web et les serveurs. Il se compose de deux parties : une bibliothèque côté client qui s'exécute dans le navigateur et une bibliothèque côté serveur pour Node.js. Les deux composants ont une API presque identique.

- WebSocket facilite la communication en temps réel entre le client et le serveur Web.
- Ce protocole aide à se transformer en multiplateforme dans un monde en temps réel entre le serveur et le client.
- L'avantage majeur qu'il représente par rapport à une connexion HTTP est qu'il permet une communication full duplex

- Les avantages de l'outil Expo :

- On n'a pas besoin d'écrire ou de configurer un code natif pour un code entièrement fonctionnel et prêt pour la production.
- Possibilité de production, mises à jour "on the fly", sans publier de nouveau l'application sur app store ou bien play store.
- Il y a une très riche bibliothèque de fonctions / composants unifiés (AR, intégration de Google et Facebook, sélecteurs de date / heure, etc.)
- Génération simplifiée de paquets binaires

- Les inconvénients de l'outil Expo :

- Si certains aspects de l'API native ne sont pas couverts, nous ne pouvons pas les utiliser.
- Impossible de joindre du code natif personnalisé
- Nous devons installer l'application client Expo sur le périphérique mobile des testeurs.
- Les binaires finaux sont assez lourds (min 20-30 Mo suite au SDK d'expo)
- Les bibliothèques externes qui demandent un « link » ne fonctionnent pas avec expo

- Le concept des PROPS :

La plupart des composants en React peuvent être personnalisés lors de leur création, avec différents paramètres. Ces paramètres de création sont appelés props, abréviation de propriétés.

Par exemple, un composant React Native de base est le "Image". Lorsque nous créons une image, nous pouvons utiliser un accessoire nommé "source" pour contrôler l'image qu'elle affiche.

Remarquons les accolades qui l'entourent pic- elles intègrent la variable pic dans JSX. on peut mettre n'importe quelle expression JavaScript entre accolades dans JSX.

nos propres composants peuvent également être utilisés props. Cela nous permet de créer un seul composant utilisé à de nombreux endroits différents dans notre application.

- Le concept de State :

Il existe deux types de données qui contrôlent un composant : "props" et "state".

props sont définis par le parent et ils sont fixes pendant toute la durée de vie d'un composant. Pour les données qui vont changer, nous devons utiliser state.

En général, on doit initialiser state dans le constructeur, puis appeler setState lorsque on souhaite le modifier.

React-navigation :

- StackNavigator : c'est la navigation la plus basique où on pousse une vue sur iOS et présente une vue sur Android. Le StackNavigator gère une pile de vues qui augmente lorsque vous naviguez vers une nouvelle vue et diminue lorsque vous revenez en arrière.
- TabNavigator : permet de créer une barre d'onglets, en haut ou en bas, de votre application. On utilisera ce type de navigation lorsque l'on voudra couper notre application en plusieurs onglets.
- DrawerNavigator : permet de créer un menu dit "hamburger", à gauche de nos vues, avec une liste d'entrées pour chacune de nos vues.

Les spécificités techniques de RN :

- Les composants , qui sont les principaux éléments constitutifs d'une application, sont traduits en vues mobiles natives : UIView sur iOS ou View sur Android et non pas des webViews comme les autres frameworks qui utilisent JS
- Fondamentalement, dans les applications RN, le code peut être divisé en deux mondes : JS et Native. On peut appeler du code natif depuis JS . React Native , via Bridge , expose les interfaces JS à l'API de la plateforme native. Cela signifie que l'application peut utiliser des fonctionnalités natives , telles que l'appareil photo, Bluetooth et GPS .
- Le code dans chacun de ces mondes JS et natifs s'exécute assez rapidement , mais il existe parfois un besoin de communication entre eux , ce qui est malheureusement lent. Le pont fonctionne comme un traducteur entre les deux mondes.