Rapport technique

**Votre contact :**

Nom : Rihane

Prénom : Farah

Téléphone : +216 20 908 503

E-mail :farahr2001@gmail.com

**L’entreprise :**

Nom : SmartGalaxy Tunsie

Adresse :Tunisie

Développement d'un système de messagerie basique avec API

Sommaire

03 • Présentation de l’entreprise

04 •  Présentation du sujet

05 •  Fonctionnalités

06 •  Besoins non-fonctionnels

07 •  Choix techniques

08 •  Choix techniques

09 •  Réalisation

10 •  Design

**Présentation de l’entreprise**

**Raison Sociale**

**Nom de l’entreprise :** SMARTGALAXY

**Année de création : 2021**

**Site internet :** <https://www.smartgalxy.tn/>

**Adresse siége :** Tunisie

**Secteur d’activité :** Développement web

**Fondateurs :** Ghanem Toumi

**Contacts :**

**Tel : +216 50 69 69 14**

**E-mail : contact.smartgalaxy@gmail.com**

**Présentation du sujet**

L’entreprise smartGalaxy m’a proposé un test technique pour évaluer mes compétences en développement. Le projet consiste à créer une application de messagerie, nommée “MessageHub”. L’objectif est de développer des API endpoints permettant aux utilisateurs d’envoyer, de recevoir et de vérifier le statut des messages.

Introduction :

**Développer des points d'accès API Node.js pour faciliter la messagerie :**

* **Créer un point d'accès API d'envoi de messages:** Implémenter un point d'accès API Node.js permettant aux utilisateurs d'envoyer des messages. Ce point d'accès doit gérer des données telles que l'ID de l'expéditeur, l'ID du destinataire, le contenu du message et, potentiellement, des pièces jointes facultatives.
* **Mettre en œuvre un point d'accès de récupération de messages:** Concevoir un point d'accès API Node.js pour que les utilisateurs puissent recevoir leurs messages. Ce point d'accès doit authentifier les utilisateurs et récupérer leurs messages en fonction de critères spécifiques (par exemple, boîte de réception, messages envoyés, conversation spécifique).
* **Développer un point d'accès de statut de message:** Construire un point d'accès API Node.js qui permet aux utilisateurs de vérifier l'état de leurs messages envoyés. Ce point d'accès doit récupérer et renvoyer des informations telles que "envoyé", "livré" ou "lu" en fonction du message et du destinataire.

Objectifs :

**Besoins fonctionnels**

**1. Envoi de messages:**

* L'utilisateur doit pouvoir composer et envoyer des messages texte.
* Le système doit prendre en charge l'envoi de pièces jointes, telles que des images et des fichiers.
* Le système doit permettre de spécifier des destinataires individuels ou des groupes de destinataires.
* Le système doit enregistrer les messages envoyés et fournir un accusé de réception.

**2. Réception de messages:**

* L'utilisateur doit disposer d'une boîte de réception pour afficher ses messages reçus.
* Les messages doivent être triés par date, expéditeur, sujet ou autres critères pertinents.
* L'utilisateur doit pouvoir marquer les messages comme lus, non lus ou importants.
* Le système doit notifier l'utilisateur lorsqu'il reçoit de nouveaux messages.

**3. Suivi des messages:**

* L'utilisateur doit pouvoir suivre l'état de ses messages envoyés.
* Le système doit indiquer si un message a été envoyé, reçu ou lu par le destinataire.
* L'utilisateur doit pouvoir consulter l'historique des messages envoyés et reçus.

**Besoins non-fonctionnels**

**1. Performance:**

* **Temps de réponse:** Le système doit répondre aux requêtes des utilisateurs rapidement et de manière fluide, même en cas de forte charge.
* **Scalabilité:** Le système doit pouvoir s'adapter à un nombre croissant d'utilisateurs et de messages sans perte de performance.
* **Disponibilité:** Le système doit être disponible en permanence et ne subir que des interruptions minimales pour la maintenance.

**2. Sécurité:**

* **Confidentialité:** Les données des utilisateurs doivent être protégées contre tout accès non autorisé.
* **Intégrité:** Les données des utilisateurs doivent être exactes et cohérentes.
* **Disponibilité:** Le système doit être protégé contre les attaques et les intrusions.

**3. Fiabilité:**

* **Stabilité:** Le système doit fonctionner de manière stable et sans pannes inattendues.
* **Récupération après sinistre:** Le système doit être capable de se remettre rapidement d'une panne ou d'une perte de données.
* **Maintenance:** Le système doit être facile à maintenir et à mettre à jour.

**4. Maintenabilité:**

* **Code source:** Le code source du système doit être bien documenté et facile à comprendre.
* **Modularité:** Le système doit être modulaire et composé de composants indépendants.
* **Testabilité:** Le système doit être facile à tester et à déboguer.

**5. Evolutivité:**

* **Architecture:** L'architecture du système doit être conçue pour être évolutive et capable de supporter une croissance future.
* **Performances:** Le système doit pouvoir améliorer ses performances en ajoutant des ressources matérielles ou logicielles.
* **Fonctionnalités:** Le système doit pouvoir être étendu avec de nouvelles fonctionnalités sans compromettre la stabilité ou les performances.

**Choix techniques**

**1. Environnement de développement**

* **Éditeur de code:** VSCode

**2. Gestion de version et contrôle de code**

* **Git** le contrôle de version l'hébergement du code source
* **GitHub:** l'hébergement du code source

**3. Base de données**

* **MongoDB :**  base de données NoSQL

**Raison :**

* + **Flexibilité des données:** MongoDB offre un modèle de données flexible basé sur des documents, permettant de stocker des données **non structurées** et des relations complexes de manière efficace, ce qui correspond bien aux besoins d'un système de messagerie.
  + **Scalabilité horizontale:** MongoDB est conçue pour une évolutivité horizontale, ce qui signifie qu'elle peut être facilement étendue en ajoutant plus de serveurs pour répondre à l'augmentation de la charge et du nombre d'utilisateurs.
  + **Performances:** MongoDB offre des performances élevées pour les requêtes fréquentes et les opérations d'insertion et de lecture, ce qui est crucial pour un système de messagerie en temps réel.

**4. Outil de gestion de base de données**

* **MongoDB Compass**
  + **Visualisation des données**
  + **Requêtes et manipulations de données:**

**5. Outil de test d'API**

* **ThunderClient:**

**Réalisation**

**Technologies utilisées :**

**1. Environnement de développement back-end : Node.js**

* **Justification :**

🡺 Idéal pour les applications en temps réel comme les systèmes de messagerie, car il peut gérer un grand nombre de connexions simultanées de manière efficace.

**2. Framework web : Express.js**

* **Justification :**

🡺minimaliste et flexible basé sur Node.js.

🡺Il fournit une structure de base pour créer des applications web et des API REST.

**3. Communication en temps réel : Socket.IO**

* **Justification :**

🡺permettant la communication bidirectionnelle et en temps réel entre le serveur et le client. Cette technologie est essentielle pour "MessageHub" car elle permet de diffuser des notifications instantanées aux utilisateurs lorsqu'ils reçoivent de nouveaux messages, garantissant une expérience de messagerie fluide et réactive.

**4. Analyse de corps de requête : body-parser**

* **Justification :** body-parser est un middleware Express.js populaire qui permet d'analyser les corps des requêtes HTTP.

**5. Interface avec MongoDB : Mongoose**

* **Justification :** Mongoose est un ODM (Object Data Modeling) pour MongoDB, qui fournit une couche d'abstraction permettant d'interagir avec la base de données de manière plus intuitive et orientée objet. Mongoose facilite la création, la manipulation et la récupération des données des messages, des utilisateurs et des conversations stockés dans MongoDB.

**6. Variables d'environnement : dotenv**

* **Justification :** dotenv est une bibliothèque utile pour charger des variables d'environnement à partir d'un fichier .env. Cela permet de stocker des informations sensibles telles que les chaînes de connexion à la base de données en dehors de votre code source, améliorant la sécurité et la maintenabilité de l'application.

**7. Documentation d'API : Swagger**

* **Justification :** Swagger est un framework open-source permettant de créer et de documenter des API REST en utilisant une syntaxe standard (Swagger Specification). L'intégration de Swagger dans mon projet "MessageHub" facilitera la compréhension et l'utilisation des API par les développeurs qui souhaitent interagir avec votre système de messagerie.

**Structure de projet :**

**Choix de la structure :**

* **Dossier par fonction:** Organisation claire et logique.
* **Séparation des préoccupations:** Modules regroupés par responsabilités.
* **Facilité de collaboration:** Conventions claires pour le travail en équipe.

**Architecture MVC (Model-View-Controller):**

* **Modèles (models):** Définition des entités du système (ex: messages, utilisateurs).
* **Contrôleurs (controllers):** Logique métier des API (ex: envoyer, recevoir messages).
* **Vues (non représentées):** Interface utilisateur (généralement gérée par un framework front-end).

**Avantages :**

* **Code modulaire et réutilisable.**
* **Facilité de maintenance et d'évolution.**
* **Tests unitaires et d'intégration simplifiés.**
* **Collaboration efficace entre développeurs.**

**Gestion des erreurs :** Utilisation des middlewares

* Pour garantir une expérience utilisateur fluide et robuste, le système de messagerie "MessageHub" intègre des mécanismes de gestion des erreurs efficaces.
* **Middleware de gestion des erreurs asynchrones :**
* Capture les erreurs asynchrones dans les fonctions de route Express.
* Journalise les erreurs pour le débogage.
* Transmet les erreurs pour une gestion centralisée.
* **Gestionnaire d'erreurs global:**
* Offre une réponse cohérente aux erreurs (code 500 et message générique).
* Peut être étendu pour des réponses spécifiques aux types d'erreurs.
* Protège contre les erreurs non gérées.

**Avantages :**

* Expérience utilisateur fluide et robuste.
* Débogage et résolution des problèmes facilités.
* Application stable et fiable.

Utilisation :

**Les Endpoints :**

1. **Envoi de messages**

**Description :**

Ce point de terminaison permet à un utilisateur d'envoyer un message à un autre utilisateur.

**Requête HTTP :**

* Méthode : POST
* URL : /messages/send

**En-têtes de requête :**

* Authorization : Un jeton d'authentification valide
* Content-Type : application/json

**2. Réception des messages**

**Description :**

Ce point de terminaison permet à un utilisateur de recevoir des messages d'autres utilisateurs.

**Requête HTTP :**

* Méthode : POST
* URL : /messages/receive

**En-têtes de requête :**

* Authorization : Un jeton d'authentification valide
* Content-Type : application/json

**3. Status des messages**

**Description :**

Ce point de terminaison permet à un utilisateur de consulter l'état d'un message spécifique.

**Requête HTTP :**

* Méthode : GET
* URL : /messages/status/:messageId

**Paramètre d'URL :**

* :messageId : L'identifiant du message dont l'état doit être consulté.

**Testes :**

* Testes Unitaires :

But : isoler et tester des unités individuelles de code.

Dans ce contexte, les tests unitaires se concentrent sur la fonctionnalité du contrôleur de messages (messageController.js) et du gestionnaire de socket (socketHandler.js).

* **Utilisation de Jest pour les tests unitaires**

Jest est un framework de test JavaScript populaire qui facilite l'écriture de tests unitaires. Jest offre un certain nombre de fonctionnalités telles que :

* **Correspondance de tout**: Vous pouvez utiliser des instructions expect avec des correspondances comme toEqual pour comparer la valeur réelle avec la valeur attendue.
* **Mocking**: Jest vous permet de mocker les dépendances externes comme les bases de données ou les API externes en utilisant jest.mock(). Ceci est utile pour isoler l'unité de code que vous testez des facteurs externes.
* **Tests asynchrones**: Jest prend en charge les tests asynchrones en utilisant la syntaxe async/await ou les promesses.

Résultat :

* **Test de de Socket.IO**

Ce projet de messagerie Node.js démontre la conception et l'implémentation d'une application de messagerie fonctionnelle avec des fonctionnalités de base telles que l'envoi et la réception de messages, la consultation de l'état des messages et la gestion des utilisateurs. L'application utilise des technologies populaires telles que Node.js, Express.js, MongoDB et Socket.io pour assurer une communication efficace entre les utilisateurs.

Conclusion :

**Au-delà de ces fonctionnalités de base, l'application pourrait être étendue pour inclure :**

* **Gestion des comptes utilisateurs**: Création de comptes utilisateurs, authentification sécurisée et gestion des profils.
* **Création d'espaces de chat**: Permettre aux utilisateurs de créer des espaces de chat privés ou publics pour des conversations individuelles ou groupées.
* **Système de notifications**: Informer les utilisateurs des nouveaux messages, des mentions et d'autres événements importants.

Perspectives :