



BTS SIO - Option SLAM
Documentation d'épreuve

E4 - Mission 3 : Mediocs


Par Baptiste Grimaldi

E4-Mission3-Mediocs-Baptiste-Grimaldi

Liens utiles:

Article Mediocs

Check out the portfolio of Baptiste Grimaldi, a full-stack web developer specializing in creating dynamic and responsive websites. Explore his projects and contact him for your next web development project.

 <https://portfolio.baptistegrimaldi.info/projects/view/mediocs>

Article de mon portfolio

Le repo GitHub

<https://github.com/GrimalDev/Mediocs>



SOMMAIRE

Liens utiles:

[Le repo GitHub](#)

1. Cahier des charges

1.1 Introduction

1.2 Expression fonctionnelle du besoin

[Les acteurs](#)

[Le besoin](#)

[Explication de l'analyse fonctionnelle](#)

1.3 Contraintes

[Budget](#)

[Délais de livraison](#)

1.4 Gestion des droits d'accès

2. Description des environnements

2.1 Stack de technologies utilisé

[Couche présentation \(front-end\)](#)

[Couche logique \(back-end\)](#)

[Couche données](#)

[Authentification](#)

2.2 Organisation des fichiers du projet

3. Méthodologie

Bonnes pratiques

Deux options principales:

[Deux exemples de stratégies de branching](#)

3.1 Méthodologie et versioning

3.2 Gestion des tests de la solution

3.3 Rédaction de la documentation

3.4 Gestion de projet

[Gestion de projet](#)

[Utilisation de Kanban](#)

[Utilisation de Gantt](#)

4. Mise en oeuvre

4.1 Serveur express

4.2 Génération coté client avec express js

4.3 Déploiement de la base de donnée

4.4 L'authentification avec passport

5. Gestion de la maintenance (corrective / évolutive)

5.1 Mise à jour de la documentation du SI

5.2 Evaluation de la qualité de la solution

[Tests avec Docker](#)

5.3 Procédure de correction d'un dysfonctionnement

5.4 Démarche qualité assurance (QA)

6. Bilan du projet

[Point légal sur le stockage de données médicales](#)

6.1 Validation des exigences point par point

6.2 Axes d'amélioration

1. Cahier des charges

1.1 Introduction

Dans le cadre d'une période de freelance lors de ma première année de BTS, je suis contacter pour effectuer un client léger avec les technologies et le stack de développement souhaité.



Note: Certaines informations ne peuvent être divulguées pour des raisons de confidentialité.

“Mediocs” est une plateforme de visualisation de données médicale sur des patients.

Page de connexion.

NOM	Prénom	Sexe	Âge	Taille	Poids	Jour	Mois	Année	Heure
COLIN	Maillard	M	37	180	85	2	6	2021	10h30
COLIN	Maillard	M	37	180	85	2	6	2021	10h26
COLIN	Maillard	M	37	180	85	2	6	2021	10h50
Zeubi	Françoise	F	42	170	65	4	4	2019	9h00
Frachois	Pascal	M	67	176	85	2	8	2021	13h00
Toha	michel	M	20	170	80	2	8	2019	9h00
COLIN	Maillard	M	37	180	85	2	6	2021	10h30
Zeubi	Françoise	F	42	170	65	4	4	2019	9h00
Frachois	Pascal	M	67	176	85	2	8	2021	13h00
Toha	michel	M	20	170	80	2	8	2019	9h00
COLIN	Maillard	M	37	180	85	2	6	2021	10h30
Zeubi	Françoise	F	42	170	65	4	4	2019	9h00
Frachois	Pascal	M	67	176	85	2	8	2021	13h00
Toha	michel	M	20	170	80	2	8	2019	9h00
COLIN	Maillard	M	37	180	85	2	6	2021	10h30
Zeubi	Françoise	F	42	170	65	4	4	2019	9h00
Frachois	Pascal	M	67	176	85	2	8	2021	13h00
Toha	michel	M	20	170	80	2	8	2019	9h00

Donnés de test.

1.2 Expression fonctionnelle du besoin

Les acteurs

Dans cette partie, il sera question de présenter les différents acteurs impliqués dans le projet Mediocs, qui se compose d'une équipe de trois personnes: un développeur, un chargé de communication et un médecin. Chaque membre de l'équipe a un rôle et une responsabilité bien définis, et la collaboration entre eux est essentielle pour assurer le succès du projet.

- Le développeur est responsable du développement des différentes fonctionnalités de la plateforme.
 - Il doit assurer la qualité du code, respecter les bonnes pratiques de développement et s'assurer que les fonctionnalités répondent aux besoins des utilisateurs.
 - Il travaille en étroite collaboration avec le chargé de communication et le médecin pour comprendre les spécifications fonctionnelles et techniques du projet.
- Le chargé de communication est en charge de la rédaction de la documentation liée à la plateforme et de la communication avec les partenaires du projet.
 - Il doit s'assurer que la documentation est claire et concise, et qu'elle répond aux besoins des différents acteurs impliqués dans le projet.
 - Il travaille également en étroite collaboration avec le développeur et le médecin pour s'assurer que la documentation reflète correctement les fonctionnalités de la plateforme.
- Le médecin est responsable de la coordination avec les autres professionnels de santé pour s'assurer que les fonctionnalités de la plateforme répondent aux besoins des utilisateurs (c-à-d. eux-même).
 - Il doit s'assurer que la plateforme est facile à utiliser pour les professionnels de santé et les patients, et qu'elle répond aux normes de sécurité et de confidentialité des données médicales.
 - Il travaille en étroite collaboration avec le développeur et le chargé de communication pour s'assurer que les fonctionnalités de la plateforme répondent aux besoins des utilisateurs.

En résumé, chaque membre de l'équipe a un rôle important dans le projet Mediocs, et la collaboration étroite entre eux est essentielle pour assurer le succès du projet. Les dépendances et interactions entre les différents acteurs doivent être bien gérées pour assurer une bonne coordination de l'ensemble des activités du projet.

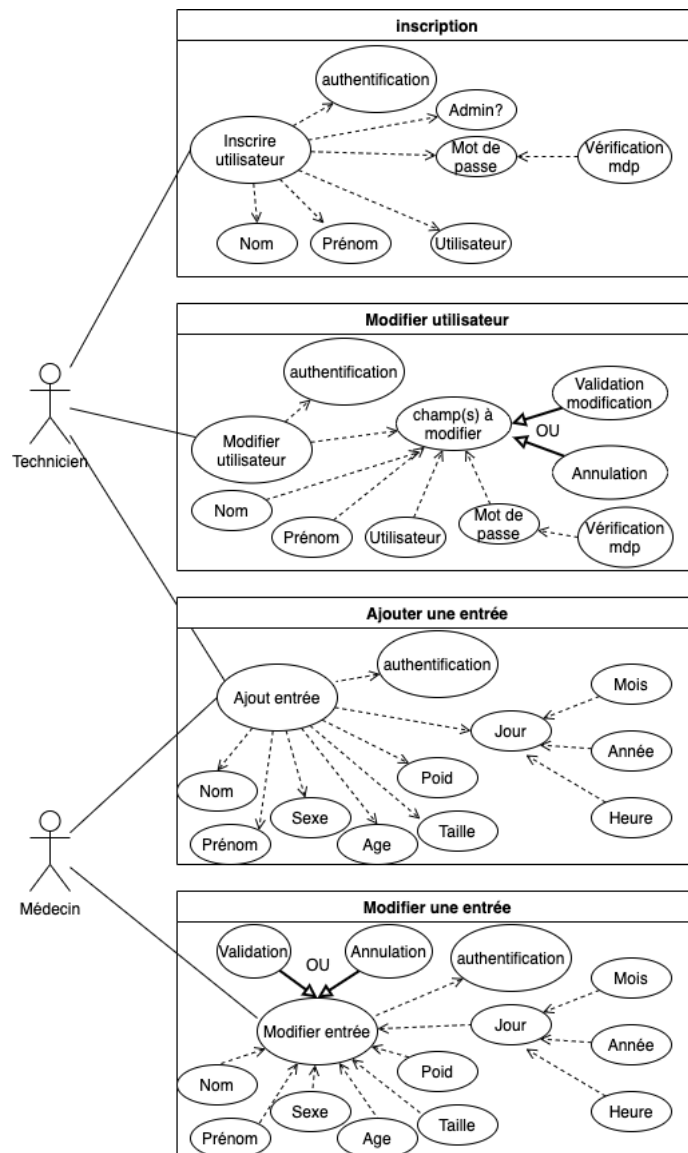
Le besoin

Des médecins collectent des données pour une recherche. Pour éviter la fastidieuse collecte manuelle sur des fiches de tableur, une interface de connexion avec gestion de niveaux d'accès est développée. Les utilisateurs autorisés peuvent accéder à la plateforme avec des informations d'identification uniques. Cette interface permet également de stocker les données collectées de manière organisée et pratique, pour faciliter l'analyse ultérieure.



La plateforme Mediocs facilite la collecte et le stockage organisé et sécurisé des données médicales de différentes sources pour les médecins. L'interface de connexion permet un accès facile aux données nécessaires pour la recherche et une analyse efficace des données collectées.

Explication de l'analyse fonctionnelle



1.3 Contraintes

Dans cette partie, nous aborderons les différentes contraintes liées au projet Medioc's. Les principales contraintes sont:

- Respect des normes de sécurité et de confidentialité des données médicales *
- Respect des délais de livraison
- Respect du budget alloué au projet
- Disponibilité des différents acteurs impliqués dans le projet pour assurer une bonne coordination des activités
- Disponibilité des différents environnements nécessaires pour la mise en place de la plateforme

Ces contraintes doivent être prises en compte lors de la planification et de l'exécution du projet, afin de garantir la réussite du projet dans les délais impartis et avec les ressources disponibles.

Le projet étant resté à l'état de POC, les normes médicales ne sont pas appliquées.

Budget

Le projet Medioc's n'a pas bénéficié d'un budget défini. La seule rémunération prévue était celle du chef de projet et des développeurs. Les clients ont insisté pour que l'utilisation de bibliothèques et d'autres outils open-source soit privilégiée, dans le but

de réduire les coûts de développement du projet.



L'absence de budget n'est pas courante dans la gestion de projets. Un budget est crucial pour la planification et le suivi du projet. Il permet de définir les ressources financières allouées et de les répartir pour chaque activité. Il permet aussi de suivre les dépenses et de prendre des mesures en cas de dérive financière.

Cependant, dans le cas de Mediocis, les clients ont choisi de privilégier l'utilisation de librairies et d'autres outils open-source pour réduire les coûts.

Cette décision peut être justifiée dans certains cas, notamment lorsque les solutions open-source sont suffisamment performantes et répondent aux besoins du projet. Cependant, il est important de garder à l'esprit que l'utilisation de solutions open-source peut également présenter des risques, tels que des problèmes de sécurité ou de compatibilité.

En résumé, bien que le projet Mediocis n'ait pas bénéficié d'un budget défini, il est important de souligner que la définition d'un budget est une étape cruciale dans la gestion de tout projet. La décision des clients de privilégier l'utilisation de solutions open-source pour réduire les coûts doit être prise avec précaution et en prenant en compte les risques potentiels.

Délais de livraison

Le respect des délais de livraison est l'une des principales préoccupations dans tout projet de développement web. Les clients attendent que leur projet soit livré à temps et avec la qualité attendue, ce qui nécessite une bonne planification et une exécution efficace du projet.

En tant que développeur web, il est important de respecter les délais de livraison pour maintenir la satisfaction du client et garantir le succès du projet. Pour cela, il est recommandé de suivre ces bonnes pratiques:

- **Planification:** Il est important de planifier le projet dès le début, en déterminant les étapes critiques et en établissant des échéances claires pour chaque étape. Il est également important de prévoir des marges de temps pour les imprévus et les retards éventuels.
- **Communication:** Il est important de communiquer régulièrement avec le client pour l'informer de l'avancement du projet et des éventuels retards. La transparence et la communication ouverte permettent de maintenir une relation de confiance avec le client.
- **Collaboration:** Il est important de travailler en collaboration avec les différents membres de l'équipe pour s'assurer que chacun respecte les délais qui lui sont impartis. La collaboration permet également de résoudre rapidement les problèmes qui peuvent survenir pendant le projet.
- **Tests:** Il est important de tester régulièrement le projet pour s'assurer que tout fonctionne correctement et pour corriger les erreurs rapidement. Les tests permettent également de réduire le temps nécessaire pour résoudre les problèmes qui peuvent survenir plus tard dans le projet.

En suivant ces bonnes pratiques, il est possible de respecter les délais de livraison du projet et de garantir le succès du projet. Il est également important de garder à l'esprit que chaque projet est unique et que les délais de livraison peuvent varier en fonction de sa complexité et de ses exigences. Il est donc important de planifier en conséquence et de rester flexible pour s'adapter à tout changement dans le projet.

1.4 Gestion des droits d'accès

Je distingue ici deux types d'accès, les médecins et les techniciens (administrateurs):

Fonctionnalités	Médecins	Techniciens
Accès à la plateforme	x	x
Accès aux données	x	x
Ajout de données	x	x
Modification de données	x	x
Suppression de données	x	x
Gestion des utilisateurs		x
Gestion des droits d'accès		x
Gestion des paramètres de la plateforme		x

2. Description des environnements

Dans cette partie, nous présenterons les différents environnements nécessaires pour la mise en place de la plateforme MediocS.

2.1 Stack de technologies utilisé



Le stack de technologies utilisé pour le projet MediocS suit une architecture trois tiers pour garantir la modularité et la maintenabilité du système.

Couche présentation (front-end)

- **EJS** est utilisé comme langage de modèle pour générer des pages HTML dynamiquement.
- **CSS** est utilisé pour la mise en forme graphique des pages.
- **JavaScript** est utilisé pour la logique front-end.

Couche logique (back-end)

- **Node.js** est utilisé comme environnement d'exécution pour le code JavaScript côté serveur.
- **Express** est utilisé comme framework d'application web pour fournir un ensemble d'outils et d'utilitaires pour la construction d'applications web.
- **Passport** est utilisé comme middleware d'authentification pour Node.js qui fournit un ensemble de stratégies d'authentification pour différents types d'authentification.

Couche données

- **LightSQL** est utilisé pour la gestion de la base de données.

Le choix de cette architecture permet de séparer les différentes couches de l'application pour garantir une évolutivité et une modularité maximales. Chaque couche est responsable de sa propre fonctionnalité, ce qui facilite la maintenance et la mise à jour du système.

Authentification

L'authentification dans le projet MediocS est gérée par Passport, un middleware d'authentification pour Node.js qui fournit un ensemble de stratégies d'authentification pour différents types d'authentification.

La stratégie d'authentification utilisée dans le projet MediocS est l'authentification de base, qui consiste à envoyer les identifiants (nom d'utilisateur et mot de passe) sur le serveur pour vérification. Les identifiants sont stockés en utilisant la méthode de hachage bcrypt pour la sécurité.

Lorsqu'un utilisateur se connecte avec succès, un cookie de session est créé pour stocker les informations d'identification de l'utilisateur. Ce cookie est stocké dans le navigateur de l'utilisateur et est utilisé pour authentifier les requêtes futures de l'utilisateur.



Il est important de noter que l'utilisation de cookies de session pour l'authentification peut présenter des risques de sécurité, tels que le vol de session et les attaques par force brute. Il est donc important de prendre des mesures pour réduire ces risques, telles que l'utilisation de SSL/TLS pour chiffrer les données de session et la mise en place de politiques de mot de passe solides pour les utilisateurs.

En résumé, l'authentification de base avec Passport et bcrypt est utilisée dans le projet MediocS pour fournir une sécurité de base pour les utilisateurs. Les cookies de session sont utilisés pour stocker les informations d'identification des utilisateurs, mais il est important de prendre des mesures pour réduire les risques de sécurité associés à l'utilisation de cookies de session.

2.2 Organisation des fichiers du projet

Organisation des fichiers:

- DATA
- public

- cert.pem & key.pem
- .env
- .dockerignore
- server.js

Définition des fichiers:

- **DATA:** Toutes les données de l'application
 - **admin_data:**
 - main_admin_data.db:
 - Un fichier « .database » mis à disposition afin de réorganiser la base de donnée admin a part. (Non-mis-en-place)
 - **medical:**
 - exemple.json:
 - Exemple d'organisation des données des patients en « .json ».
 - med_data_1.db:
 - Un fichier test d'incorporation de la base de donnée « NeDB » afin de remplacer le fichier object.json
 - object.json
 - Le fichier qui est actuellement lu par le server lors d'une requête « GET » sur le « route » « /data ».
 - **user_data:**
 - main_user_data.db
 - Principale base de donnée utilisateur.
- **public:** Ce dossier contient la racine publique du projet, tout ce qui sera envoyé au client et auquel il aura accès.
 - **script:**
 - script_index.js:
 - Ce script définit le tableau où s'affichent les données dans la page html principale « index.html ».
 - Permet aussi le rendering du menu déroulant avec ces certaines propriétés purement esthétiques.
 - script_login.js:
 - Script vide mais présent pour d'éventuelles futurs améliorations.
 - script_register.js:
 - Script vide mais commenté avec une future table des utilisateurs et de modification pour admins.
 - **style:**
 - La décoration graphique du client
 - media:
 - Toute la déco image
 - style_index.css:
 - Le fichier est commenté afin de grouper les différentes domaines d'applications.
 - style_login.css:
 - Page style de la page de login. Toujours la même organisation par block.
 - style_register:
 - Page organisée par block. Page toujours en progression.
 - **Views:**

- Index.html
 - Défini la page html des données. Il n'y a pas grand chose comme tout est généré par le script script_index.js
 - login.html
 - Définition de la page html d'authentification
 - register.html
 - Définition de la page html d'enregistrement d'utilisateurs.
 - En bas de la page est marqué « liste d'utilisateurs » car une liste est à venir. (Future propriété)
- cert.pem et key.pem
 - Fichier certificat ssl pour le protocole HTTPS
 - .env
 - Les strings admin et session_secret qui permettent de sécuriser le code
 - .gitignore:
 - Les fichiers à ignorer pour GitHub
 - server.js:
 - La définition même du serveur d'application. Tout est décrit par blocs d'utilisations.
 - Chaque packet est commenté avec son utilité.

3. Méthodologie

Bonnes pratiques

- ne commitez que des choses sur les mêmes sujets (style, front, back, etc...)
- Si vous ne pouvez pas écrire de messages de commit concis, cela indique trop de sujets dans le même commit.
- Utilisez un titre et un corps avec seulement la commande commit et en ajoutant une ligne vide entre le titre et le corps.

```
git commit
::
TITRE
CORPS
\# rest comments
```

Stratégies de branching

- Une convention écrite pour organiser l'équipe.

Deux options principales:

1. Développement **Mainline**:
 - **quelques** branches
 - commits relativement petits
 - normes de test de haute qualité
 -
2. Branches **State**, **Release** et **Feature**
 - **Deux** types de branches différents qui remplissent des types de travail différents
 1. **LongRunning**
 - existe tout au long de la vie du projet

- souvent, ils reflètent les "étapes" de votre cycle de vie de développement

2. Short Running

- pour les nouvelles fonctionnalités, les corrections de bogues, le refactoring, les expériences
- sera supprimé après l'intégration (fusion/rebase)

Deux exemples de stratégies de branching

1. GitHub Flow

- très simple, très léger : seulement long-running
- branche ("main") + branches de fonctionnalités

2. GitFlow

- plus de structure, plus de règles
- long-running : "main" + "develop"
- de courte durée : fonctionnalités, versions, correctifs

3.1 Méthodologie et versioning

La méthode de travail utilisée pour le projet Mediocs est basée sur Git, un système de contrôle de version distribué. Git permet de suivre les modifications apportées au code source du projet, de collaborer avec d'autres développeurs et de gérer efficacement les différentes branches de développement.

Le projet Mediocs utilise la méthode de développement agile, qui se caractérise par une approche itérative et incrémentale du développement. Cette méthode de travail permet de s'adapter facilement aux changements et de livrer rapidement des fonctionnalités à valeur ajoutée.

Le versioning avec Git est utilisé pour gérer les différentes versions du code source du projet. Chaque modification apportée au code source est enregistrée dans un commit, qui est ensuite intégré à une branche de développement spécifique. Les branches de développement permettent de travailler sur différentes fonctionnalités en parallèle, tout en maintenant une version stable de la plateforme.

En résumé, la méthode de travail utilisée pour le projet Mediocs est basée sur Git et la méthode de développement agile. Cette approche permet de gérer efficacement les modifications apportées au code source, de collaborer avec d'autres développeurs et de livrer rapidement des fonctionnalités à valeur ajoutée.

3.2 Gestion des tests de la solution

Les tests doivent être effectués régulièrement tout au long du développement de la plateforme Mediocs. Idéalement, des tests unitaires et d'intégration devraient être effectués après chaque modification du code source. Cela permet de détecter rapidement les erreurs et de les corriger avant qu'elles ne se propagent dans le système.

Les tests doivent être effectués régulièrement tout au long du développement de la plateforme Mediocs, avec des tests unitaires et d'intégration après chaque modification du code source, et des tests de non-régression après chaque intégration importante. Cela permet de garantir la qualité et la stabilité de la plateforme.

3.3 Rédaction de la documentation

La méthode de développement agile, utilisée pour le projet Mediocs, prévoit la rédaction de documentation tout au long du processus de développement. La documentation est considérée comme un élément clé pour garantir la communication et la collaboration entre les membres de l'équipe et pour maintenir la qualité de la plateforme.

La documentation doit être claire, concise et facilement accessible. Elle doit fournir des informations sur la conception, la mise en œuvre et le fonctionnement de la plateforme, ainsi que sur les normes et les bonnes pratiques adoptées par l'équipe de développement.

La documentation doit également être mise à jour régulièrement pour refléter les changements apportés au code source et aux fonctionnalités de la plateforme. Les mises à jour de la documentation doivent être gérées de manière cohérente avec les cycles de développement et de livraison de la plateforme.

Il faut savoir prioriser le code. Un code **clair** et **précis** avec des commentaires **complémentaire** et qui ne paraphrasent pas la ligne ou le bloc commenté. Si toutes ces conditions sont réunies, le code parlera pour lui-même.

3.4 Gestion de projet

Il s'agira ici de détailler la gestion de projet pour la mise en place de la plateforme Mediocs.

Gestion de projet

La gestion de projet pour la mise en place de la plateforme Mediocs est gérée avec Notion, un outil de gestion de projet et de collaboration en ligne.

Utilisation de Kanban

Le tableau Kanban de Notion est utilisé pour organiser les tâches et les activités du projet. Les tâches sont divisées en colonnes en fonction de leur état, telles que "À faire", "En cours" et "Terminé". Les membres de l'équipe peuvent facilement déplacer les tâches entre les colonnes pour refléter leur état actuel.

Le tableau Kanban est utilisé pour suivre les tâches individuelles et pour s'assurer que chaque membre de l'équipe a une vue d'ensemble des activités du projet.

Utilisation de Gantt

Le diagramme de Gantt de Notion est utilisé pour visualiser les différentes tâches et activités du projet dans le temps. Les tâches sont représentées sous forme de barres horizontales sur un axe de temps vertical. Les membres de l'équipe peuvent facilement voir les dépendances entre les tâches et les activités et ajuster le calendrier du projet en conséquence.

Le diagramme de Gantt est utilisé pour planifier les activités du projet et pour s'assurer que le projet est sur la bonne voie en termes de calendrier.

En résumé, Notion est utilisé pour gérer le projet Mediocs, avec un tableau Kanban pour suivre les tâches individuelles et un diagramme de Gantt pour visualiser les activités du projet dans le temps. Ces outils permettent à l'équipe de gestion de projet de gérer efficacement les activités du projet et de s'assurer que le projet est sur la bonne voie.

4. Mise en oeuvre

4.1 Serveur express



Exemple de serveur dans la même logique que le projet Mediocs.

```
const express = require('express')
const app = express()
const port = 3000

app.use(express.json())

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening at <http://localhost>:${port}`)
})
```

Ce code crée un serveur express qui écoute sur le port 3000 et répond avec "Hello World!" lorsqu'il reçoit une requête GET sur la racine de l'application. Il utilise le middleware `express.json()` pour parser les corps de requête JSON.

Pour créer une API REST, vous pouvez ajouter des routes pour gérer les différentes méthodes HTTP (GET, POST, PUT, DELETE) pour les différentes ressources de l'API. Par exemple, pour créer une route pour une ressource `users`, vous pouvez ajouter le code suivant:

```
app.get('/users', (req, res) => {
  // Renvoie tous les utilisateurs
})

app.post('/users', (req, res) => {
  // Crée un nouvel utilisateur
})
```

```

})

app.get('/users/:id', (req, res) => {
  // Récupère un utilisateur par ID
})

app.put('/users/:id', (req, res) => {
  // Met à jour un utilisateur par ID
})

app.delete('/users/:id', (req, res) => {
  // Supprime un utilisateur par ID
})

```

Pour la petite note, express fonctionne sur la base REST et avec des middlewares.



Dans le cadre d'Express, un middleware est une fonction qui a accès à l'objet de requête (`req`), à l'objet de réponse (`res`) et à la fonction de rappel suivante (`next`) dans le cycle de vie de la requête. Les middlewares sont utilisés pour effectuer des tâches communes à plusieurs routes, telles que l'authentification, la gestion des erreurs et la gestion des sessions.

Les middlewares sont ajoutés à une application Express en utilisant la méthode `use()` ou les méthodes HTTP (`get()`, `post()`, `put()`, `delete()`). Les middlewares sont exécutés dans l'ordre dans lequel ils sont ajoutés à l'application.

Par exemple, pour ajouter un middleware d'authentification à toutes les routes de l'application, vous pouvez utiliser le code suivant:

```

app.use((req, res, next) => {
  // Vérifie l'authentification de l'utilisateur
  if (req.isAuthenticated()) {
    // Passe à la route suivante
    next()
  } else {
    // Redirige l'utilisateur vers la page de connexion
    res.redirect('/login')
  }
})

```

Ce middleware vérifie si l'utilisateur est authentifié en appelant la méthode `isAuthenticated()` sur l'objet de requête. Si l'utilisateur est authentifié, le middleware appelle la fonction de rappel suivante (`next()`), ce qui permet à la requête de se poursuivre. Si l'utilisateur n'est pas authentifié, le middleware redirige l'utilisateur vers la page de connexion en utilisant la méthode `redirect()` sur l'objet de réponse (`res`).

4.2 Génération côté client avec express js

Pour la génération côté client avec Express JS, vous pouvez utiliser des moteurs de template tels que EJS ou Pug pour générer des pages HTML dynamiques à partir de données. Ces moteurs de template permettent d'intégrer facilement des données dynamiques dans des pages HTML et de générer des pages personnalisées en fonction des besoins de l'application.

Pour utiliser EJS avec Express JS, vous pouvez ajouter le middleware `ejs` et définir le répertoire de vues à utiliser. Voici un exemple de code pour configurer EJS avec Express JS:

```

// Configuration de EJS
app.set('view engine', 'ejs')
app.set('views', './views')

```

Ce code configure EJS comme moteur de template, définit le répertoire de vues à utiliser et définit une route pour la racine de l'application qui renvoie une page HTML générée à partir d'un modèle EJS. Le modèle EJS utilise des données dynamiques pour afficher un message "Hello World!".

En résumé, vous pouvez utiliser des moteurs de template tels que EJS ou Pug avec Express JS pour générer des pages HTML dynamiques à partir de données. Ces moteurs de template permettent de créer des pages personnalisées en fonction des besoins de l'application.

4.3 Déploiement de la base de donnée

Le **LightSQL** est une base de données légère et simple qui utilise un format de stockage de données **JSON**. Contrairement aux bases de données traditionnelles, qui utilisent des tables et des colonnes, le **LightSQL** stocke les données sous forme de documents **JSON**. Cela permet une grande flexibilité dans la structure des données stockées, car chaque document peut avoir ses propres champs et structures.

Le **LightSQL** est souvent utilisé pour des applications qui ont besoin de stocker des données simples, telles que des configurations, des listes de tâches ou des données de formulaire. Il est également utilisé comme base de données de secours pour les applications qui nécessitent une sauvegarde temporaire de données en cas de panne du système principal.

J'utilise la librairie **NeDB** pour la communication avec la base de donnée.

```
const datastore = require('nedb') //database package

const user_database = new datastore('main_user_data.db')

user_database.loadDatabase()
```

4.4 L'authentification avec passport

Passport est un middleware d'authentification pour les applications **Node.js**. Il permet de gérer l'authentification des utilisateurs en utilisant différentes stratégies d'authentification, telles que l'authentification locale (nom d'utilisateur et mot de passe), l'authentification basée sur les jetons et l'authentification **OAuth**.

Passport fournit une API simple pour configurer les différentes stratégies d'authentification et gérer le processus d'authentification. Il utilise également des cookies de session pour stocker l'état de l'authentification de l'utilisateur.

Dans le projet **Mediocs**, Passport est utilisé pour gérer l'authentification des utilisateurs avec l'authentification locale (nom d'utilisateur et mot de passe). Le middleware **passport-local** est utilisé pour configurer la stratégie d'authentification locale et le middleware **passport** est utilisé pour gérer le processus d'authentification.

```
//L'essence de la définition du passport n'est que cela !
passport.use(new LocalStrategy({
  passReqToCallback: true,
  usernameField: "email",
  passwordField: "password"
}),
  authenticateUser
);
```

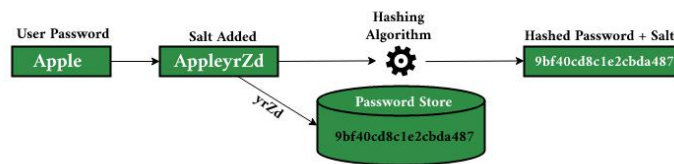


Passport dans le cadre de **Médiocs** a aussi besoin de savoir comment vérifier et chercher un utilisateur et quoi faire avec les session et cookies.

Tout les mots de passes ainsi que les rôles des utilisateurs sont hachés avec du sel via la librairie **Bcrypt**.

Le hachage est réalisé avec la librairie **bcrypt**, qui permet de réaliser un hachage avec un sel pour améliorer la sécurité. Le sel est une chaîne de caractères aléatoire qui est ajoutée au mot de passe avant le hachage, ce qui rend le hachage plus difficile à casser.

Le hachage des mots de passe est une bonne pratique de sécurité recommandée pour toutes les applications qui stockent des informations sensibles des utilisateurs, comme les mots de passe. Cela permet de protéger les utilisateurs en cas d'attaque de sécurité et de garantir la confidentialité de leurs informations personnelles.



Explication de l'utilité du sel dans le hashage.

5. Gestion de la maintenance (corrective / évolutive)

5.1 Mise à jour de la documentation du SI

La mise à jour de la documentation du SI doit être effectuée régulièrement pour refléter les changements apportés à la plateforme MediocS. Cela permet de garantir que la documentation est à jour et que les membres de l'équipe ont accès aux dernières informations sur la plateforme. Les mises à jour de la documentation doivent être gérées de manière cohérente avec les cycles de développement et de livraison de la plateforme.

5.2 Evaluation de la qualité de la solution

Dans cette partie, nous aborderons l'évaluation de la qualité de la solution MediocS.

Tests avec Docker

Pour lancer un Docker avec une image Node et le contenu du dossier en tant que serveur, exécutez la commande suivante dans le dossier :

```
docker run -d -p 3000:3000 --name mediocS-test -v $(pwd):/app -w=/app node start
```

Assurez-vous de remplacer `nom_du_container` par le nom que vous voulez donner à votre conteneur.

Cette commande va lancer un conteneur Docker en arrière-plan avec l'image Node. Le port 3000 sera exposé et mappé sur le port 3000 du conteneur. Le dossier courant sera monté dans le conteneur à l'emplacement `/app` et le répertoire de travail sera également défini sur `/app`. Enfin, la commande `npm start` sera exécutée dans le conteneur pour lancer le serveur Node.

Pour arrêter le conteneur, exécutez la commande suivante :

```
docker stop mediocS-test
```

Assurez-vous de remplacer `nom_du_container` par le nom du conteneur que vous avez choisi.

5.3 Procédure de correction d'un dysfonctionnement

La maintenance corrective consiste à corriger les erreurs ou les bogues qui ont été identifiés dans la plateforme MediocS. Ces erreurs peuvent être signalées par les utilisateurs ou découvertes lors des tests de l'application. La maintenance corrective doit être effectuée rapidement pour minimiser les perturbations pour les utilisateurs de la plateforme.

La plateforme MediocS dispose d'un formulaire de contact dédié pour que les médecins puissent faire des retours sur leur expérience avec le logiciel. En outre, l'équipe de développement travaille en étroite collaboration avec les médecins pour s'assurer que la plateforme répond aux besoins des professionnels de la santé. Cette collaboration permet de recueillir des commentaires et des suggestions pour améliorer la plateforme et de mettre en place des corrections en cas de dysfonctionnements signalés.

5.4 Démarche qualité assurance (QA)

La démarche de qualité assurance (QA) est essentielle pour garantir la qualité et la performance de la plateforme MediocS. La démarche QA doit être mise en place dès le début du processus de développement et doit être intégrée à toutes les étapes du cycle de vie de la plateforme.



La démarche QA comprend plusieurs étapes clés, telles que la planification des tests, la mise en place d'environnements de test, l'exécution de tests, l'analyse des résultats des tests et la mise en œuvre des corrections en cas de dysfonctionnements identifiés.

Pour la planification des tests, il est important de définir les objectifs des tests, les critères de réussite et les scénarios de test à effectuer. Les tests doivent couvrir tous les aspects de la plateforme, y compris les fonctionnalités principales, les performances, la sécurité et la compatibilité avec différents navigateurs et appareils.

La mise en place d'environnements de test est également importante pour garantir que les tests sont effectués dans un environnement contrôlé et cohérent. Les environnements de test doivent être configurés pour refléter les environnements de production et doivent inclure des données de test réalistes.

L'exécution des tests doit être effectuée régulièrement tout au long du développement de la plateforme Medioccs. Idéalement, des tests unitaires et d'intégration sont effectués après chaque modification du code source. Cela permet de détecter rapidement les erreurs et de les corriger avant qu'elles ne se propagent dans le système. Des tests de non-régression sont également effectués après chaque intégration importante pour garantir que les fonctionnalités existantes ne sont pas affectées par les nouvelles modifications.

L'analyse des résultats des tests est essentielle pour identifier les dysfonctionnements et les erreurs dans la plateforme. Les résultats des tests sont analysés de manière systématique pour identifier les tendances et les modèles d'erreurs. Les corrections sont ensuite immédiatement mises en œuvre pour minimiser les perturbations pour les utilisateurs de la plateforme.

La démarche QA est essentielle pour garantir la qualité et la performance de la plateforme Medioccs. La démarche QA est mise en place dès le début du processus de développement et est intégrée à toutes les étapes du cycle de vie de la plateforme.

6. Bilan du projet

Point légal sur le stockage de données médicales

Le stockage de données médicales est une question très sensible qui nécessite une attention particulière. La plateforme Medioccs doit se conformer aux lois et réglementations en vigueur concernant la protection des données médicales. Cela inclut des mesures de sécurité pour protéger les données, des politiques pour limiter l'accès aux données sensibles et des procédures pour la suppression des données lorsque cela est nécessaire.

En outre, la plateforme Medioccs doit obtenir le consentement explicite des patients pour stocker leurs données médicales. Les patients doivent être informés de la manière dont leurs données seront stockées et utilisées, et doivent donner leur consentement avant que leurs données ne soient stockées dans la plateforme.

Le stockage de données médicales est une question très sensible qui nécessite une attention particulière.



En Europe, il existe des certifications qui garantissent la conformité des systèmes de stockage de données médicales et sensibles aux normes de sécurité et de confidentialité en vigueur. Par exemple, la certification HDS (Hébergeur de Données de Santé) en France ou la certification ISO 27001 au niveau européen. Ces certifications sont importantes pour garantir la sécurité et la confidentialité des données stockées dans la plateforme Medioccs et pour se conformer aux lois et réglementations en vigueur.

6.1 Validation des exigences point par point

Besoins / Exigences	Tableau Kanban	Diagramme de Gantt	Serveur Express	Génération côté client	Base de données	Authentification avec Passport	Maintenanc correctiv
Suivre les tâches individuelles	✓						
Visualiser les activités du projet dans le temps		✓					
Créer un serveur Express			✓				

Générer des pages HTML dynamiques				✓			
Utiliser une base de données légère					✓		
Gérer l'authentification des utilisateurs						✓	
Corriger les erreurs ou les bogues identifiés							✓
Mettre en place une démarche de qualité assurance							
Se conformer aux lois et réglementations en vigueur							

6.2 Axes d'amélioration

Il manque quelques fonctionnalités dû à l'arrêt momentané du projet. Le projet est resté à l'état de POC et n'a jamais vu le jour. Une démarche qualité a cependant bien été dirigée avant la fermeture du projet par manque de ressources.

Les droit d'auteur et de création me sont revenu et me permettent de présenter ce beau projet malgré son aspect sensible.

Les données ainsi que certaines connaissances ne peuvent être développés à l'écrit.



Cette documentation est la propriété intellectuelle de Grimaldi Baptiste.
portfolio.baptistegrimaldi.info/legal

Avis de droits d'auteur

Informations légales sur les droits d'auteur de cette documentation

Cette documentation a été créée par moi, Grimaldi Baptiste, en tant qu'étudiant. Tous les droits d'auteur sur cette documentation sont réservés. Aucune partie de cette documentation ne peut être reproduite, stockée dans un système de récupération ou transmise sous quelque forme ou par quelque moyen que ce soit, électronique, mécanique, photocopie, enregistrement ou autre, sans l'autorisation préalable écrite de l'auteur.

Toute utilisation non autorisée de cette documentation peut constituer une violation des lois sur les droits d'auteur et entraîner des poursuites judiciaires.

Si vous souhaitez utiliser cette documentation à des fins éducatives ou autres, veuillez contacter l'auteur pour obtenir l'autorisation écrite nécessaire.

Copyright Grimaldi Baptiste, 2023.

Par Baptiste Grimaldi