Nom TP	Évaluation En Cours de Formation - TP DIRECCTE -Eté 24
Parcours	Bachelor Développeur Angular 2022-2024
Référence	BachelorDev2224_ECF_Ete24_400681_20240513103051

Copie à rendre

TP Concepteur développeur d'applications

Documents à compléter et à rendre

Partie 1 : livrables

1. Entrez le lien de votre outil de gestion de projet

https://romanlandais.atlassian.net/jira/software/projects/HP/boards/5

2. Entrez le lien de votre git

- Font End Application Web :
  https://github.com/RomanLandais/hospital.git
- Front End Application Mobile :
  https://github.com/RomanLandais/hospitalMobilApp.git
- Front End Application Desktop:
  https://github.com/RomanLandais/hospitalDesktopApp.git
- Back End (API):
  https://github.com/RomanLandais/hospitalBackEnd.git
- Back End (Base de données et certificat https) :
  https://github.com/RomanLandais/Base-de-donn-e-Hospital.git
- Livrables projet :

3. Entrez le lien de vos applications déployées en ligne

http://localhost:4200/home

Partie 2 : Planification

1. **Comment avez-vous effectué la planification de votre projet ?**

   Le projet a été planifié en suivant les recommandations clients (de l'énoncé). Les exigences ont d'abord été regroupées ensemble sous la forme d'Epic puis

décomposées en sous-tâches spécifiques. Etant seul sur le projet, une répartition du travail n'a pas été nécessaire, la priorisation des tâches a été faite en fonction des exigences de l'énoncé, du temps imparti et des compétences du développeur.

2. **Quelle méthode de gestion de projet avez-vous utilisé et pourquoi ?**

Une méthodologie Agile aurait pu être utilisée pour cet exercice. Cependant, au vu du temps limité pour la réalisation (une semaine car changement de cursus) et qu'il s'agit d'un projet individuel, une méthodologie Kanban est suffisante.

Réalisation Diagramme utilisation et séquence :

Les applications demandées étant relativement simples niveau fonctionnalités, il a été décidé de représenter l'application dans son ensemble dans les diagrammes d'utilisation et de séquence. Ainsi nous ne rentrerons pas dans les détails de chaque page de l'application. En voici les raisons:

- Simplicité : En représentant l'application dans son ensemble, les diagrammes restent simples et faciles à comprendre. Nous pouvons ainsi communiquer de manière efficace les principales fonctionnalités de l'application et les interactions avec les utilisateurs sans ajouter de complexité inutile. Nous fournissons ainsi une vue d'ensemble, simple et claire.

- Focus sur les objectifs : Pour des applications simples, il est souvent plus important de se concentrer sur les objectifs généraux de l'utilisateur et sur la manière dont l'application aide à les atteindre, plutôt que sur les détails spécifiques de chaque page ou fonctionnalité.

- Facilité de maintenance : En maintenant le diagramme d'utilisation simple, il est plus facile de le mettre à jour et de le maintenir au fil du temps, ce qui est important pour les applications qui peuvent évoluer ou être modifiées.

Également dans un souci de clarté, il a été décidé de décomposer les diagrammes de séquences en plusieurs diagrammes séparés. Cela permet de plus que chaque diagramme puisse se concentrer sur un aspect particulier du système, ce qui facilite la compréhension et l'analyse.

Partie 3 : Technologies utilisées

1. **Quelles technologies avez-vous utilisé ? Précisez tous les éléments qui vous ont permis de faire ce choix. Ajoutez également un paragraphe expliquant en quoi ces choix sont le plus adaptés pour la demande spécifiée dans l'énoncé.**

Côté front end, le choix s'est arrêté sur l'utilisation de Javascript et du Framework Angular, largement expérimenté durant ce cursus et disposant de fonctionnalités complètes. Un détail plus poussé de ce choix se trouve plus bas.

Une réflexion a été faite entre l'utilisation de PHP ou Node.js pour la partie backend. Node a été privilégié pour rester dans une configuration javascript, sa mise en place est également plus simple.

Environnement de travail utilisé pour ce projet

Système d'exploitation : Le système utilisé est Windows car c'est celui présent de base sur mon matériel et sur lequel je me suis entraîné durant les cours.

Outils de développement : Visual Studio Code a été utilisé pour ce projet. Il est gratuit, open source avec une communauté bien présente. Ces diverses fonctionnalités et extensions en fond un logiciel performant pour coder. On peut de plus le connecter directement à GitHub et il intègre nativement des fonctionnalités de Git. Il est conçu pour être léger et rapide, même avec des projets de grande taille. Il offre une expérience utilisateur fluide et réactive, même lorsqu'il est utilisé sur des machines moins puissantes.
De plus, ses fonctionnalités de débogage intégré et complétion intelligente en font un outil parfait pour la réalisation de ce projet.

Android Studio a également été utilisé pour le déploiement de l'application mobile, pour son adaptation avec Ionic et sa facilité d'utilisation.

Gestionnaire de versions : Git et GitHub ont été utilisés pour la gestion de versions du code source. Ce sont des outils performants et recommandés dans l'énoncé du projet. Ils ont donc été configurés sur la machine.

Langages de programmation :

- Frontend :
  - Web App : Le langage Typescript via le framework Angular a été utilisé pour la partie frontend de cette application. Ce framework permet de développer des applications web dynamiques de manière rapide et efficace. Le système de routage d'Angular permet de gérer la navigation dans l'application et de définir des routes pour les différents composants simplement. Angular est également conçu pour être facilement testable, avec des fonctionnalités intégrées pour les tests unitaires et les tests d'intégration. De plus Angular prend en charge les dernières normes du web et garantit la compatibilité avec une large gamme de navigateurs, ce qui assure une expérience utilisateur cohérente sur différentes plateformes et appareils. Enfin, ce framework a largement été étudié durant ce cursus de formation, l'expérience du

développeur est également à prendre en considération dans les choix. Node Package Manager a été installé sur la machine, ce qui a ensuite permis la configuration d'Angular via les commandes npm.
L'utilisation de la bibliothèque Bootstrap a également été choisie pour la création des formulaires. En effet, elle est facile d'accès et le CDN permet une implémentation rapide sur toutes les machines.

- Mobile App : Le choix a été fait de rester dans une configuration Javascript pour cette application. En effet, les 4 applications à développer étant complémentaires, il est logique de rester dans le même langage de programmation. Ceci facilite le codage ainsi que la maintenance des différents programmes. Le framework Ionic a donc été utilisé en complément d'Angular pour cette application. Il a été choisi pour sa simplicité et facilité d'utilisation, notamment grâce à son affinité avec le framework Angular. L'application mobile demandée est relativement simple, Ionic est donc parfaitement suffisant et adapté à ce contexte. De plus, le déploiement sous Ionic est simple à mettre en œuvre.

- Desktop App : Pour l'application Desktop, il a également été choisi de rester dans une configuration JS pour les mêmes raisons précédemment expliquées. Son utilisation s'est faite grâce au framework Electron associé à Angular. Electron permet le développement d'application multi-plateforme utilisant les technologies du web. Il a été décidé d'utiliser un template prédéfinis pour l'installation d'Electron. En effet cela permet un gain de temps car la plupart des configurations nécessaires pour faire fonctionner Angular avec Electron sont déjà en place, moins d'erreurs en utilisant une configuration prétestée et une structure claire.

- Backend : On utilisera Node.js et Express pour leur facilité de configuration avec Angular et SQLite. Cela nous permet également de rester dans un écosystème JavaScript/TypeScript et facilite le partage de code entre les applications et le serveur. Sa nature asynchrone permet de plus une meilleure gestion des opérations d'entrée/sortie, réduisant les temps d'attente et augmentant l'efficacité.
  On installe également Nodemon pour surveiller les modifications de fichiers et redémarrer le serveur lorsqu'il a besoin d'être mis à jour. Le module sqlite3 dans notre partie backend a également été installé pour interagir avec la base de données.

Bases de données : SQLite a été utilisé pour la réalisation de la base de données relationnelle de ce projet. Bien que plus limitée sur les parties fonctionnalités et sécurités, elle a été choisie pour sa légèreté, sa facilité à être manipulée et intégrée dans des applications. L'accent sur la sécurité sera donc mis sur les autres côtés du backend et du frontend.
Il a été choisi de réaliser plusieurs tables dans notre base plutôt qu'une grande. Ceci permet d'organiser les données en groupe logique et facilite l'accès aux informations en fonction des différentes requêtes. Cette configuration permet également de meilleures performances et une maintenabilité facilitée. L'utilisation d'une base de données relationnelle nous permet, grâce aux clés primaires et secondaires, d'assurer que les données sont valides et cohérentes.

Transaction SQL

Dans le fichier fourni en annexe, une transaction SQL a été réalisée. À la suite de la création de la base ainsi que l'intégration des premières données, il semble judicieux de tester le fonctionnement de l'ensemble des tables. De plus, les premières informations présentes serviront de support pour la réalisation de l'application et le test des différentes requêtes.
La requête de transaction a donc pour objectif de renseigner les autres tables vides présentes dans notre base. Elle nous permet également de tester le bon fonctionnement des interactions entre les tables du fait des clés primaires et secondaires.
Le code est exécuté sans erreur et les nouvelles données sont correctement intégrées à la base. Nous pouvons donc passer aux étapes suivantes.

2. **Quelles mesures avez-vous prises afin de protéger vos applications des potentielles vulnérabilités de sécurités ? (Injection SQL, failles XSS, …)**

- Mise en place de Validateurs dans les parties frontend et backend pour les validations d'entrée de formulaire et validation des requêtes reçues. Ces validateurs offrent une première protection contre différentes attaques, comme les injections SQL ou le Cross-Site Scripting (XSS).
  Joy, bibliothèque de validation des schémas, a été utilisée pour valider les données reçues côté back end.

- Mise en place de connexion Https pour communiquer avec le serveur. (Utilisation outil OpenSSL) (certificat serveur et clé API). Cette connexion permet un chiffrement des données échangées entre le navigateur de l'utilisateur et le serveur, empêchant ainsi les attaquants de lire les informations transmises en cas d'interception. Cette connexion permet également l'utilisation de certificats SSL/TLS pour authentifier le serveur et garantir aux utilisateurs qu'ils communiquent avec le bon site web, et non avec un site usurpé.
  Bien que l'utilisation d'un certificat délivré par une autorité de certification reconnue soit la meilleure pratique a privilégiée, nous avons choisi la création d'une clé en local. En effet, l'obtention d'une clé reconnue nécessite l'utilisation d'un domaine référencé ce qui n'est pas notre cas. De plus, dans le cadre de cette utilisation scolaire sur des applications n'ayant pas pour but d'être utilisées ultérieurement, une clé locale est suffisante.

- Le hachage des mots de passe côté backend avant leur envoi à la BDD a été mis en place grâce au module Bcrypt. Cette sécurité permet de chiffrer les mots de passe présents dans notre base, ainsi, même si la base de données est compromise, les informations de mot de passe seront inutilisables.

- Des requêtes paramétrées pour empêcher l'échappement des données ont également été mises en place côté backend.

- Côté backend toujours, des routes spécifiques pour chaque requête ont été réalisée. Ceci permet de filtrer les données envoyées au frontend en fonction des besoins de ce dernier, sans envoyer de données non nécessaires.

- Mise en place de token stocké sur des cookies sécurisés pour valider l'envoi de nouvelles requêtes côté utilisateur. Ainsi, un utilisateur doit obligatoirement être authentifié pour accéder aux différentes fonctionnalités de l'application (création de séjour, nouveau médecin...). Chaque token a une durée de vie de 12h et doit être régénéré après chaque fermeture de page.

- Gestion des autorisations : ainsi un simple utilisateur ne pourra accéder aux fonctionnalités réservées aux administrateurs, secrétaire ou docteur. Chaque catégorie d'utilisateur se voit définir des autorisations d'accès à telle ou telle fonctionnalité, en fonction de ses besoins. Cela permet de limiter le risque en cas de vol de compte utilisateur.

- Connexion avant accès aux pages : Pour chaque application, il est nécessaire de se connecter avant d'accéder aux pages et différentes fonctionnalités. Ceci offre une première sécurité aux applications et permet le traçage des utilisateurs, évitant ainsi la non-répudiation.

- Utilisation d'environnement de codage et de module utilisant les dernières mises à jour existantes. Cela permet de diminuer le risque sur des attaques déjà connues.

- Une veille sur les nouvelles menaces et la surveillance de son trafic réseau du site est également à mettre en place à la suite du déploiement.

- Les utilisateurs sont invités à utiliser des mots de passe sécurisé et fort pour leur identification. Pour sécuriser encore plus cette connexion, une double voir triple vérification aurait pu être mise en place.

3. Décrivez les tests applicatifs que vous effectuez et dans quel but.

Le choix a été arrêté sur l'utilisation de GitLab comme environnement de test. Il permet d'intégrer facilement un ensemble de tests (unitaires, fonctionnels et de sécurité) avec une intégration CI/CD.
Par soucis de simplicité, il a été décidé de tester séparément nos parties frontend et backend dans des environnements de test séparés. Le cas pratique a été réalisé sur la partie Web frontend.

Un nouveau projet Gitlab a donc été réalisé et connecté à l'application afin de mettre en place notre intégration CI/CD. Un fichier .gitlab-ci.yml a été créé dans la racine de notre application pour contenir les différentes instructions de test. Ce fichier contient les instructions pour :

- Test unitaire : un dossier spécifique avec des exemples de test a été créé (test/unit)
- Test statique de sécurité : l'idée est d'utiliser SonarQube pour effectuer ces tests
- Test dynamique de sécurité : nous utilisons également l'outil ZAP de Owasp pour effectuer ces tests
- Audit test des dépendances : via npm, analyse des dépendances et recherche de vulnérabilités

- Test fonctionnel : réalisation de tests fonctionnels via npm et la création de code de test, un exemple a été fait dans le dossier test/functional

Avec une intégration de cet environnement de test dans un fonctionnement CI/CD, le fonctionnement et la sécurité de l'application sont pris en charge et intégrés tout au long de sa construction. Ceci nous permet d'être proactif sur la sécurisation de notre code et finalement de gagner du temps.

À la suite du déploiement, des outils pour contrôler le trafic du site et sa performance sont conseillés. Ceci permet de surveiller les performances de notre site et détecter de possibles attaques. (Outil Google Analytics)

Un contrôle régulier des dépendances est également à effectuer.

Une veille sur les nouvelles attaques informatiques et les solutions existantes est également une bonne pratique à suivre.

4. **Comment avez-vous effectué le déploiement de votre application ?**

Le choix d'un déploiement en local a été fait. En effet, le site ayant une simple utilisation scolaire et n'ayant pas pour vocation à être utilisé par des personnes extérieures, le déploiement local avec un partage de code est le plus adapté. Ce choix est le plus approprié en termes de praticité et d'économie des ressources.

1. Télécharger les différents fichiers sources sur GitHub :

2. Lancer les applications dans votre environnement de développement local et installer les dépendances

3. Un fichier d'environnement .env est présent localement côté backEnd et non partagé en public sur GitHub, voici son code :

```
SESSION_SECRET=7ab6cd366cf3e174c4b2d681d2067101a8acfde7be9b7a48791435509f69729d
ACCESS_TOKEN_SECRET=f764ea2977769663ff6888e2904c7149e2ed40d5ac6a48f8c56410d479cb02d70ef329066328c90d07c157f1478a70c08d26b0cd6f785136f0d77456a7277370
```

4. Installer la base de données localement à partir des fichiers fournis sur le repository git

5. Construire l'application frontend avec ng build et ng serve pour exécuter en local sur le server

6. Exécuter npm start pour lancer le backend

Backend

Construit avec Node.js et Angular ainsi que le framework express et l'utilitaire Nodemon

Déploiement en local :

1. Cloner le dépôt GitHub localement
2. Installer les dépendances nécessaires : `npm install`
3. Le fichier .env est normalement présent dans le clone GitHub, si non présent le créer et y insérer le code suivant pour communication sécurisée avec la base de données

```
SESSION_SECRET=7ab6cd366cf3e174c4b2d681d2067101a8acfde7be9b7a48791435509f69729d

ACCESS_TOKEN_SECRET=f764ea2977769663ff6888e2904c7149e2ed40d5ac6a48f8c56410d479cb02d70ef329066328c90d07c157f1478a70c08d26b0cd6f785136f0d77456a7277370
```

4. Configurer la base de données à partir des fichiers du dépôt GitHub correspondant
5. Pour un fonctionnement en mode développement, rentrer la commande suivante dans le terminal : `nodemon server`
6. Déployer l'application, rentrer la commande suivante dans le terminal : `npm start`

Web App

Ce projet a été développé sous JS avec le framework Angular.

Instruction déploiement local application :

1. Cloner le dépôt GitHub
2. Installer Angular Cli si non présent, dans le terminal de commande exécuter : `npm install -g @angular/cli`
3. Installer les dépendances grâce à la commande du terminal : `npm install`
4. Mettre en route le backend (suivre les instructions liées au dépôts GitHub)
5. Pour le mode développement utiliser la commande du terminal : `ng serve`
6. Pour le déploiement en production utiliser la commande du terminal : `ng build –prod`

BDD

Déploiement :

1. Récupérer les fichiers depuis le dépôt GitHub

2. Exécuter les instructions de code dans votre base de données (nous avons utilisé SQLite pour ce projet)

Mobile APP

Déploiement en local de l'application :

1. Télécharger le dépôt GitHub

2. Installer les dépendances via la commande du terminal : `npm install`

3. Installer Ionic si non présent : `npm install -g @ionic/cli`

4. Dans la configuration de .eslintrc.json : un chemin absolu a été inséré dans `"files": ["*.ts"], "parserOptions": { "project": "CHEMIN/ABSOLU/VERS/FICHIER/APPLICATION/tsconfig.json", "createDefaultProgram": true }`

L'architecture de base ne renvoyait pas sur le bon chemin et l'utilisation de chemin relatif ne fonctionnait pas.

5. Mettre en route le backend (suivre les instructions liées au dépôts GitHub)

6. Pour visualiser le projet en mode développement utiliser : `ionic serve`

7. Pour déployer le projet, installer les plates-formes mobiles, localement, nous avons fait le choix d'utiliser Capacitor. Installer Capacitor avec la commande du terminal : `npm install @capacitor/core @capacitor/cli`
   Puis pour Android : `ionic capacitor add android`
   Pour IOS: `ionic capacitor add ios`

8. Construire l'application pour générer les fichiers nécessaires à la synchronisation avec Capacitor : `ionic build`

9. Synchroniser avec Capacitor : `npx cap sync`

10. Ouvrir le projet dans l'IDE native :
Pour Android : `npx cap open android`
Pour iOS : `npx cap open ios`

11. Pour un déploiement sur simulateur :

a. Créer un émulateur : dans Android Studio, ouvrir le Device Manager depuis la barre de menus Tools. Créer un nouvel émulateur en choisissant un appareil virtuel avec une version d'Android appropriée.
b. Exécuter la commande “run app” pour lancer notre application sur l'émulateur Android.

Desktop

1. Cloner le dépôt GitHub sur votre éditeur de code

2. Installer les dépendances : `npm install`

3. Dans le fichier main.js à la racine de l'application (dans le dossier app situé au même niveau que src, et non le dossier enfant de src), ajouter la ligne suivante en fin de code pour permettre l'interaction avec le backend :
```
electron_1.app.commandLine.appendSwitch("ignore-certificate-errors");
// Ignorer les erreurs de certificat. l'utilisation d'un certificat auto-généré nous contraint à cette faiblesse de sécurité. Dans un environnement de développement classique avec utilisation d'un certificat reconnu, cette faiblesse ne serait pas nécessaire.
```

4. Mettre en route le backend (suivre les instructions liées au dépôts GitHub)

5. Pour le mode développement utiliser dans le terminal la commande :
`npm start`

6. Pour construire l'application : `npm run electron:build`

En cas de problème lors de la construction, passer en mode administrateur dans le terminal et relancer la construction de l'application.

Le déploiement en local de l’application Desktop a été réalisé. Malheureusement, le fichier d’installation étant relativement lourd (68 MO), nous ne pouvons pas le partager sur GitHub.