|  |
| --- |
| Application desktop de gestion d’horaires et facturation |

Table des matières

[1 Analyse préliminaire 3](#_Toc499021832)

[1.1 Introduction 3](#_Toc499021833)

[1.2 Objectifs 3](#_Toc499021834)

[1.3 Planification initiale 4](#_Toc499021835)

[2 Analyse / Conception 4](#_Toc499021836)

[2.1 Concept 4](#_Toc499021837)

[2.2 Stratégie de test 4](#_Toc499021838)

[2.3 Risques techniques 4](#_Toc499021839)

[2.4 Planification 4](#_Toc499021840)

[2.5 Dossier de conception 5](#_Toc499021841)

[3 Réalisation 5](#_Toc499021842)

[3.1 Dossier de réalisation 5](#_Toc499021843)

[3.2 Description des tests effectués 6](#_Toc499021844)

[3.3 Erreurs restantes 6](#_Toc499021845)

[3.4 Liste des documents fournis 6](#_Toc499021846)

[4 Conclusions 6](#_Toc499021847)

[5 Annexes 7](#_Toc499021848)

[5.1 Résumé du rapport du TPI / version succincte de la documentation 7](#_Toc499021849)

[5.2 Sources – Bibliographie 7](#_Toc499021850)

[5.3 Journal de travail 7](#_Toc499021851)

[5.4 Manuel d'Installation 7](#_Toc499021852)

[5.5 Manuel d'Utilisation 7](#_Toc499021853)

[5.6 Archives du projet 7](#_Toc499021854)

*NOTE L’INTENTION DES UTILISATEURS DE CE CANEVAS:  
Toutes les parties en italiques sont là pour aider à comprendre ce qu’il faut mettre dans cette partie du document. Elles n’ont donc aucune raison d’être dans le document final.*

*De plus, en fonction du type de projet, il est tout à fait possible que certains chapitres ou paragraphes n’aient aucun sens. Dans ce cas il est recommandé de les retirer du document pour éviter de l’alourdir inutilement.*

# Introduction

## Contexte

En tant qu’apprenti en développement d’application et dans le cadre de l'apprentissage, j'ai pu effectuer des stages dans différents départements de l'entreprise et différents projets lors des examens de validation de modules du cursus de l’apprentissage. Ceci m’a donné la possibilité d’explorer différents types de développement, et c’est vers le développement web que j’ai le plus envie de progresser.

Lors de la 4ême année les apprentis doivent effectuer un « Travail Pratique Individuel » et l’entreprise dans laquelle je travaille me permet d’effectuer un TPI d’entrainement au vrais TPI de fin d’année, et c’est dans ce cadre que je réalise ce projet.

Ce projet consiste à une application de gestion d’horaires alloués à chaque projet des clients de l’utilisateur et de génération d’une facture mensuelle.

## Convention du rapport

Les termes qui nécessitent une explication ou une définition sont écrits ***en italique et en gras***, et sont ensuite défini dans un glossaire en fin de document.

Les noms de fichiers qui sont cités dans le rapport sont écrits en **gras.**

Les commandes ou le code qui est cité dans le rapport sont en font gris.

## Inventaire du matériel utilisé

Pour la réalisation de ce projet j’ai à disposition le matériel suivant :

* Ordinateur type laptop « Corporate » avec windows 11
* ***IDE*** VSCodium
* JavaScript runtime Node.js
* Framework React
* Framework Electron
* Réalisateur de maquettes Figma
* Git bash

## Prérequis

Pour effectuer ce projet il est nécessaire avoir des connaissances en :

1. HTML
2. CSS
3. JavaScript
4. Node.js
5. React
6. ElectronJS
7. Figma
8. SQLite
9. Git

## Objectifs

L’objectif de l’application est celui de s’occuper de la gestion des heures de travail des projets auquel l’utilisateur participe pour ses clients.

L’application permettra de visualiser un calendrier, avec différentes vues (mensuelle, hebdomadaire, journalière) sur lequel sera possible y ajouter des heures de travail, similairement à un système de timbrage.

L’application permettra de gérer les clients de l’utilisateur, en pouvant y ajouter des clients, et des projets pour chaque client, sur lesquelles sera possible y lier les horaires de travail qui s’afficherons sur le calendrier.

Les horaires de travail vont contenir un petit résumé du travail effectué, similairement à un journal de travail, de manière à pouvoir avoir un historique du travail effectué.  
  
L’objectif de ce projet est celui de réaliser une application qui permet de gérer les horaires de travail pour chaque projet, et de pouvoir en générer une facture.

## Planification initiale

Lors du premier jour de projet, j'ai établi un planning sur toute la durée de celui-ci:

Bar chart

Description automatically generated

# Analyse / Conception

Suite à la définition des objectifs du projet, il est important de choisir les technologies avec lesquelles le réaliser, dans ce chapitre je vais décrire les raisons pour lesquelles j’ai choisi une technologie plutôt qu’une autre.

Ces raisons sont le fruit d’une relativement courte phase d’analyse effectuée le premier jour du projet.

## Choix techniques

### React

**React** est un ***framework*** ***javascript opensource*** des plus connus avec « Angular » et « vuejs ».

J’ai décidé d’utiliser ce dernier principalement pour la nécessité d’avoir un framework léger, rapide et d’utilisation facile pour me focaliser majoritairement sur la phase de développement que celle de structure et de maintenabilité, car ceci est un petit projet et n’est pas destiné à évoluer dans une plus grande application.

**Angular** aurais garanti une majeur maintenabilité et structure connue en cas de développement ultérieur et/ou reprise par autres développeur, car la structure de tout projet Angular est la même, il est en fait dit « orienté », c’est-à dire que les concepteurs de celui-ci (Google) pensent qu’il y a qu’une seule bonne manière de développer et ils organisent tout le framework de cette manière-là ;  
Les projets Angular sont néanmoins plus grands et lourds que React ce qui aurais pas convenu pour un petit projet comme celui-ci.

**VueJS** se présente comme étant la synthèse entre Angular et React. Il permet d’utiliser des composants réactifs et réutilisables avec une syntaxe très simple qui est populaire et facile à comprendre. Comme React il utilise le DOM virtuel.

Il est aussi bien porté pour les petits projets, mais le choix de React s’impose sur Vuejs car il a actuellement majoritairement de librairies disponibles et la communauté de celui-ci est plus grande que celle de Vuejs.

### Electron

**Electron** c’est un framework qui permet de créer une application desktop multiplateforme (Windows, Linux, MacOs) à partir d’une ***codebase*** javascript.

J’ai fait ce choix en conséquence d’avoir choisi de développer l’application avec le framework web React, car Electron permet de créer des applications desktop multiplateformes sans avoir des connaissances particulières avec ce dernier, ce qui me permet de focaliser sur le développement de l’application en utilisant moins de temps dans le packaging de cette dernière.

Mon choix se porte sur ce framework aussi car c’est celui qui est utilisé par des grandes applications comme « Whatsapp » « Slack » et « Twitch ».

### Design maquette - Figma

Le design étant une partie fondamentale de la création d’applications, celui-ci va définir et structurer, l’application et la manière dont l’utilisateur va interagir avec celle-ci, et donc il est très important de dépenser un peu de ressources pour cette partie du projet car cela va définir l’expérience de l’utilisateur avec votre application.

Figma est un outil en ligne gratuit très complet et versatile qui permet assez facilement de pouvoir faire des maquettes d’interfaces utilisateur.

La manière de designer avec Figma reflète beaucoup la manière de programmer avec React. J’ai choisi ce dernier pour le fait de pouvoir créer la maquette directement subdivisée en composants, ce qui me facilite la tâche de programmation pour la création de ces derniers dans la structure du code.

J’ai opté pour un design simple et intuitif pour essayer de faciliter et rendre agréable l’utilisation de l’application.

Cette phase du projet m’a pris plus de temps que prévus cela a faussé d’une demi-journée ma planification, j’ai été surpris par l’importance et la complexité de faire une bonne conception et design de l’application.

La maquette du projet se trouve sur le suivant lien : <https://www.figma.com/file/DzCqVJVYyUlu177SVKIbU5/TPI_intro_maquette?node-id=0%3A1>

Mais voici quelques captures d’écrans pour voir à quoi cela ressemble :

Diagram

Description automatically generated

Menu bar composant

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, application

Description automatically generated

Client page - detail d’un client

Client page - liste des clients

Client page list of clients composant

Graphical user interface, application, Teams

Description automatically generated

Projets page – liste des Projets

Graphical user interface, application

Description automatically generatedProjets page - detail d’un Projet

##### Remarques

J’ai sous-estimé la réalisation de la maquette Figma qui m’a pris deux jours entiers au lieu d’un jour et demi planifié pour cette tâche.

Ceci est du au fait que je n’avais jamais réalisé de maquettes auparavant et donc étant la première fois cela m’a pris plus de temps que prévu, du au fait d’un moment d’appréhension d’utilisation du logiciel de design.

Mais cette expérience a été instructive pour moi et maintenant j’ai appris à utiliser Figma.

## Conception base de données

Pour réaliser le modèle de données j’ai utilisé un outil en ligne qui permet de faire des schema et il s’appelle ***Draw.io***

Voici ici suivant la structure des tables de ma base de données :

Diagram

Description automatically generated

TODO from /\*

## Stratégie de test

*Décrire la stratégie globale de test:*

* *types de des tests et ordre dans lequel ils seront effectués.*
* *les moyens à mettre en œuvre.*
* *couverture des tests (tests exhaustifs ou non, si non, pourquoi ?).*
* *données de test à prévoir (données réelles ?).*
* *les testeurs extérieurs éventuels.*

## Risques techniques

* *risques techniques (complexité, manque de compétences, …).*

*Décrire aussi quelles solutions ont été appliquées pour réduire les risques (priorités, formation, actions, …).*

## Planification

*Révision de la planification initiale du projet :*

* *planning indiquant les dates de début et de fin du projet ainsi que le découpage connu des diverses phases.*
* *partage des tâches en cas de travail à plusieurs.*

*Il s’agit en principe de la planification* ***définitive du projet****. Elle peut être ensuite affinée (découpage des tâches). Si les délais doivent être ensuite modifiés, le responsable de projet doit être avisé, et les raisons doivent être expliquées dans l’historique.*

## Dossier de conception

*Fournir tous les document de conception:*

* *le choix du matériel HW*
* *le choix des systèmes d'exploitation pour la réalisation et l'utilisation*
* *le choix des outils logiciels pour la réalisation et l'utilisation*
* *site web: réaliser les maquettes avec un logiciel, décrire toutes les animations sur papier, définir les mots-clés, choisir une formule d'hébergement, définir la méthode de mise à jour, …*
* *bases de données: décrire le modèle relationnel, le contenu détaillé des tables (caractéristiques de chaque champs) et les requêtes.*
* *programmation et scripts: organigramme, architecture du programme, découpage modulaire, entrées-sorties des modules, pseudo-code / structogramme…*

***Le dossier de conception devrait permettre de sous-traiter la réalisation du projet !***

\*/ to DOTO

# Réalisation

## Installation de l’environnement Node.js

Pour installer l’environnement Node.js qui permet d’exécuter le code javascript de l’application, j’ai téléchargé le paquet d’installation ***.msi*** sur leur site « <https://nodejs.org/en/> » et effectué l’installation de ce dernier.

## Création application react

Pour créer l’application react j’ai utilisé le module « ***create-react-app »*** qui est un constructeur d’applications react.

Pour l’installer il faut ouvrir un terminal et grâce à l’aide de ***npm*** l’installer avec la commande suivante :

* npm install create-react-app

Une fois le module installé, grâce à ***npx,*** un outil qui est installé automatiquement avec ***npm***, on peut utiliser le package « create-react-app » qu’on vies d’installer pour créer notre application, en exécutant donc la suivante commande :

* npx create-react-app app

Cette commande va nous crée un dossier de nom « **app »** dans lequelse situera l’application.

Une fois l’application créé, dedans il y a des fichiers par défaut pour afficher une page de bienvenue avec des liens utiles pour apprendre à utiliser le framework React, j’ai donc changé ceux-ci avec le code de mon application.

## Configuration electronjs

Pour configurer electronjs, je me suis déplacé dans le dossier de l’application et j’ai installé les packages suivants :

* electron
* electron-builder
* nodemon
* concurrently
* electron-is-dev

Toujours de la même manière avec npm install « nom-du-package »

Une fois installé les packages nécessaires j’ai donc créé le fichier « **electron.js** » dans le répertoire « **public** » qui pour l’instant définit la page desktop de l’application.

En futur ça sera dans ce fichier que je vais définir les fonctions de manipulation des données de la base de données.

## Configuration scripts de package.json

Suite à l’installation d’electron dans l’application react, il faut donc définir les scripts nécessaires que npm pourra exécuter pour lancer ou construire notre application.

Text

Description automatically generated

Package.json

Voila à quoi ressemblent les scripts de l’application, en suite pour les exécuter il suffit d’aller à la racine du projet, et avec un terminal exécuter la commande :

* npm run ‘nom-du-script’

le script que j’ai utilisé lors de la phase de développement pour voir les changements en temps réel lors de la programmation est :

* npm run dev

Ce script va en effet exécuter concurrently (precedamment installé) qui nous permet d’exécuter en concurrence le script ‘start-react’ et donc lancer notre application react et parallèlement attendre la disponibilité de celle-ci et en suite lancer la version desktop de l’app avec electron.

C’est avec le script ‘build’ que on va créer l’exécutable .exe de l’application avec electron.

## Communication entre react e electron

La communication entre react et electron se fait en configurant une ***‘api’*** dans le fichier **preload.js.**

Cette ‘api’ permettra d’accéder aux ressources electron, comme la base de données, qui ne peut pas s’exécuter dans un contexte « ***browser »*** comme une application react, donc c’est pour cette raison que electron fait office de « ***backend »*** dans mon projet.

Text

Description automatically generated

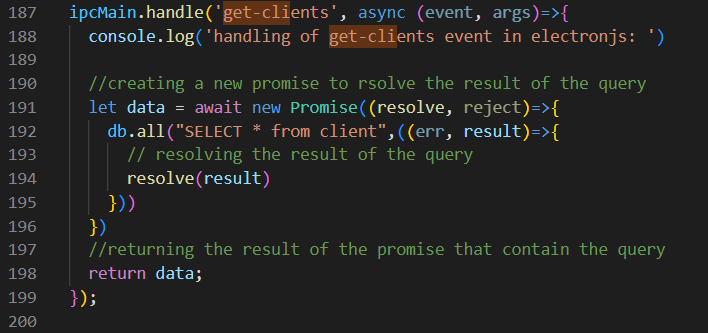
Preload.js

Dans le fichier preload.js à la ligne 1 on importe les classes ‘**ipcRenderer’** et **‘contextBridge’** du module electron.  
  
en suite avec le **contextBridge** à la ligne 3 on fait appel à la méthode **exposeInMainWorld** en lui passant comme paramètre le nom de l’api, dans mon cas simplement ‘**api’**.

C’est comme second paramètre que j’ai définit les méthodes de l’api exposée, dans la capture d’écran j’ai créé uniquement la méthode **getClients,** en guise d’exemple pour expliquer le fonctionnement.

La méthode exposée donc elle invoquera un événement de nom ‘**get-clients’,** en passant des arguments avec l’événement qui sont ceux passé par paramètre à la méthode de l’api.

L’événement invoqué, sera géré par le fichier **electron.js** avecla méthode « **ipcMain.handle()** » de la manière qui suit dans la capture d’écran suivante :



Handling de l’événement ‘get-clients’ dans electron.js

L’event en question en fait est celui qui permet de récupérer tous les clients de la base de données, et n’a donc aucun paramètre en input, mais par exemple dans le cas où je veux récupérer un client par son id, je passerai en paramètre à travers ‘args**’** la valeurde l’id du client que je cherche, par exemple l’événement ‘get-client-by-id’ est geré de la manière suivante : A screenshot of a computer

Description automatically generated with medium confidence

Handling de l’event ‘get-client-by-id’

À la ligne 242 j’insère dans la requête ***sql*** la valeur de l’id passé en paramètres à travers args.

Pour donc boucler la boucle, il faut appeler les méthodes de l’api exposée depuis l’application react, et cela se fait de la manière suivante :

Text

Description automatically generated

Calling api method getClients from react ‘clientDataService.js’

## Structure du code

Le code de l’application se trouve dans le dossier « **app** » du projet.

Graphical user interface, text, application

Description automatically generated

Structure du dossier « App »

Dans le sous dossier « **db** » on y trouve le fichier **myAppDb.db**qui est le fichier de stockage des informations de la database.

Dans le dossier « **node\_modules »** on y trouve tous les modules installés et utilisés pour l’application.

Le dossier « **public** » contient le fichier HTML avec l’élément « root » dans lequel React va créer le ***virtual DOM*** et y ajouter automatiquement le code compilé lors du processus de construction.

Les modules et leurs versions utilisées sont définis dans le fichier   
**package-lock.json.**

Dans le fichier **package.json** sont définis différents scripts que l’on peut exécuter depuis la ligne de commande pour faire différentes actions, comme construire le livrable de l’application ou encore l’exécuter en mode développement.

Le fichier **README.md** décris comment utiliser les scripts définis dans le **package.json** et leur résultat.

Dans le fichier **.gitignore** se trouvent tous les fichiers et dossiers qui ne doivent pas être uploadés sur la ***repository github.***

Dans le dossier **src** se trouvent les sources du projet, donc le code de l’application, il a été créé automatiquement par la création de l’application React.

assText

Description automatically generated with low confidence

Structure du dossier « src »

Il contient plusieurs fichiers :

* **Index.css** contient le style global de l’application
* **Index.js** est le script qui va afficher notre application React dans l’élément « root » du DOM réel.  
  Il va créer le virual DOM et dedans y afficher le composant App, qui lui est définit dans e fichier **App.js**
* **App.css** contient le style du composant **App.js**
* **App.js** est le composant « App » qui est appelé dans **index.js**

Le dossier **shared-components** contient les composants qui sont utilisés pas toute l’application, qui sonts :

* TODO

Le dossier **Services** contient les services qui sont utilisés par l’application, comme par exemple les services d’accès à la base de donnée.

Le code est divisé en plusieurs « pages » qui sont des composants dans des dossiers et qui à leur fois font appel à différents composants.

Les 3 (TODO 4 ?) pages principales sont les suivantes :

* Calendar Page
* Clients Page
* Projects page

## Node Modules

Dans le dossier « node\_modules » on y trouves tous les modules externes que l’application utilise.Cette application utilise différents modules et pour pouvoir les utiliser il faut d’abords les installes, chaque un d’entre eux s’installe avec la commande :  
 «  npm install ‘nom-du-module’ »

Les modules utilisés avec leur relative commande d’installation sont :

### @mui/material

npm install @mui/material

Ce module nous permet d’utiliser des composants déjà prefait, voici la liste des composants utilisés dans le projet :

* Box
* IconButton
* Button
* Card
* CardActions
* CardContent
* TextField
* Typography
* Chip

### @mui/icons-material

npm install @mui/icons-material

Ce module nous permet d’utiliser des Icones comme :

* Group
* Event
* ArrowBackIos
* SettingsSuggest
* Search
* Add

### react-router-dom

npm install react-router-dom

Ce module nous permet d’utiliser la navigation dans notre application, avec les Composants suivants :

* BrowserRouter
* Routes
* Route
* NavLink
* Outlet
* UseNavigate - UseLocation

## Composants React

Pour comprendre le fonctionnement de l’application il est important comprendre la notion de « Composant » du framework utilisé React.

Un composant est un objet javascript, qui selon ses inputs, il nous retourne du contenu HTML.

L’avantage des composants c’est qu’ils sont « réutilisables », cet a dire que partout dans le code ou j’appelle un composant il va s’afficher, et cela permet d’éviter la répétition de code.

Pour comprendre la notion de composant, voici la description du composant **« ProjectPage»**, qui est utilisé pour afficher la création ou la liste des projets.

Le composant est divisé en plusieurs parties :

* Les « imports »
* Les déclarations des states et des variables
* La logique de comportement du composant
* Le résultat HTML retourné

Voici une capture d’écran du composant

Text

Description automatically generated

ProjectPage component

Dans l’image précédente on peut observer de la ligne 1 à la ligne 8, les « imports »

Ceux-ci servent à utiliser des ressources extérieures au composant dans lesquels ils sont déclarés.

A la ligne 10 et 11 je déclares deux constantes qui ne changeront jamais lors de l’utilisation de l’application, la première elle définis un style css qui sera utilisé dans le composant, la deuxième sert à définir les champs une liste de champs, elle sera utilisée dans le code pour définir quels champs afficher de chaque projet.

Á la ligne 13 je fait un export d’une fonction, qui sera en fait le nom de mon composant, dans ce cas on parle du composant « ProjectPage ».  
dans ce cas dans les parenthèses je lui passe aucun paramètre mais les parametres seront utilisés et donc expliqués pour le composant « **Element »**

Depuis la ligne 14 à 17 je définis les « states » du composant, qui sont en effet des « états » du composant.  
Dans les parenthèses carrées il y a deux valeurs, la première sert à me restituer la valeur du state et la deuxième à la « set » cet à dire de changer la valeur de l’état.

Le « useEffect » qui commence à la ligne 21 sert à exécuter du code uniquement une fois, lors du premier affichage du composant à chaque fois qu’il est appelé.

En suite de la ligne 30 à la ligne 93 il y a la déclaration des fonctions, qui gèrent l’apparence du composant en modifiant les différents states suivant l’interaction avec celui-ci.

Finalement dans les lignes 95 à 117, il y a la partie HTML qui sera « retournée » à l’endroit ou le composant sera appelé.

Donc un objet HTML sera retourné à l’endroit où le composant **ProjectPage** sera appelé, avec la balise <ProjectPage/>

## Routing React

Pour utiliser le routing dans l’application j’ai installé un package avec le « Node Package Manager » (aka npm) dans la racine du projet, donc :

- Ouvrir un terminal dans « TPI\_intro/app »

- Exécuter la commande « npm install react-router-dom »

En suite pour utiliser le routing j’ai fait appel à la balise « BrowserRouter », que j’ai importé de react-router-dom, à l’endroit de l’application ou je veux utiliser le routing, donc dans le composant « App », c’est à dire dans le composant de mon application.

En suite dans le Browser Router il faut déclarer les « Routes », qui associent une url avec des composant, donc on crée un Balise « Routes » qui elle va contenir des autres balises « Route » qui elles avec la propriété « path » vont définir l’url et avec la propriété « element » y associer un composant :

Text

Description automatically generated

Routing in App component

Donc la balise <Routes> sera remplacée par le composant qui est associé à l’url actuel lors de l’utilisation de l’application.

## Shared-Components :

Dans ce dossier on trouve les composants suivants :

- **Header**

- **Element**

- **Footer**

- **Filter**

- **NavBar**

Les composants qui se trouvent dans ce dossier sont des composants utilisés par plusieurs autres composants, Par exemple, **Element**, **Header**, **Footer**, sont utilisés soit par **CLientPage** que **ProjectPage.**

Pour utiliser les composants dans d’autres composants il faut les importer dans le composant dans lesquels on veut les utiliser :

Import des composants « Header » et « Footer »

Une fois importés on peut les appeler à l’endroit dans le code ou on veut les faire apparaitre, voila un exemple dans le code du composant « **ClientPage »** ou je fais appel au composant Header :



Appel de « Header » dans ClientPage component

Pour appeler le composant « Header » il suffit d’écrire la balise qui correspond au nom de la fonction que le composant exporte.  
Comme on peut constater, dans la balise on y trouve aussi des propriétés, et celles-ci sont des propriétés que le composant utilise comme input.  
  
Dans le cas du composant **Header,** on lui passe la propriété appelée « icon », qui elle sert à lui donner l’icone que le composant devra afficher.  
Dans ce cas, dans la propriété j’ai mis une expression logique, suivant le state « addMode » du composant « **ClientPage** », je vais passer une Icone plutôt qu’une autre.

On peut observer que <Group> et <Event> sont elles aussi des composants, qui viennent de la librairie **@mui/icons-material,** ce sont en effet des icones.

### Header

*Décrire la réalisation "physique" de votre projet*

* *les répertoires où le logiciel est installé*
* *la liste de tous les fichiers et une rapide description de leur contenu (des noms qui parlent !)*
* *les versions des systèmes d'exploitation et des outils logiciels*
* *la description exacte du matériel*
* *le numéro de version de votre produit !*
* *programmation et scripts: librairies externes, dictionnaire des données, reconstruction du logiciel - cible à partir des sources.*

*NOTE : Evitez d’inclure les listings des sources, à moins que vous ne désiriez en expliquer une partie vous paraissant importante. Dans ce cas n’incluez que cette partie…*

## Description des tests effectués

*Pour chaque partie testée de votre projet, il faut décrire:*

* *les conditions exactes de chaque test*
* *les preuves de test (papier ou fichier)*
* *tests sans preuve: fournir au moins une description*

## Erreurs restantes

*S'il reste encore des erreurs:*

* *Description détaillée*
* *Conséquences sur l'utilisation du produit*
* *Actions envisagées ou possibles*

## Liste des documents fournis

*Lister les documents fournis au client avec votre produit, en indiquant les numéros de versions*

* *le rapport de projet*
* *le manuel d'Installation (en annexe)*
* *le manuel d'Utilisation avec des exemples graphiques (en annexe)*
* *autres…*

# Conclusions

*Développez en tous cas les points suivants:*

* *Objectifs atteints / non-atteints*
* *Points positifs / négatifs*
* *Difficultés particulières*
* *Suites possibles pour le projet (évolutions & améliorations)*

# Annexes

## Résumé du rapport du TPI / version succincte de la documentation

### Situation de départ

Le projet se déroule sur 90 heures, les premières choses établies étaient le planning initial et un ’étude du stack technologique à utiliser pour le réaliser.

La phase suivante étant celle de l’analyse des points favorables et négatifs des différents frameworks existants permettant de réaliser l’application, qui s’est suivie par le choix des technologies utilisées pour la réalisation du projet.

### Mise en œuvre

Pour bien structurer la mise en œuvre de l’application, il a été indispensable effectuer une phase de design de cette dernière, qui a été effectuée avec l’outil en ligne appelé Figma.  
Cette phase a pris plus de temps que prévu dans le planning à la suite d’une phase d’apprentissage de l’utilisation du logiciel utilisé.

Après la phase de design de l’application, a suivit la phase de design de la base de données, celle-ci a été effectuée avec l’outil en ligne draw.io qui permet de schématiser les entités de la BD.

Une fois le modèle de données établis, la phase de création de l’application a eu lieu.  
L’application est créé avec le constructeur « create-react-app ».  
  
Une fois l’application React crée, il a fallu y ajouter le module Electron, créer les fichiers « Electron.js » et « preload.js » et configurer les scripts nécessaires à la création de l’application desktop.

### Résultats

## Glossaire

***Framework :*** *collection de plusieurs librairies, avec une architecture minimale de projet et un ensemble de scripts pour vous simplifier le développement.*

***Javascript :*** *langage de programmation non typé TODO*

***Codebase : TODO***

***Front-End : TODO***

***Back-End : TODO***

***Draw.io :*** *outil en ligne pour la réalisation de schémas.*

***Node Package Manager (npm) :***

## Sources – Bibliographie

* <https://vscodium.com/>
* <https://nodejs.org/en/about/>
* <https://reactjs.org/>
* <https://draw.io>
* <https://www.electronjs.org/>
* <https://www.figma.com/>
* <https://git-scm.com/about>

*Liste des livres utilisés (Titre, auteur, date), des sites Internet (URL) consultés, des articles (Revue, date, titre, auteur)… Et de toutes les aides externes (noms)*

## Journal de travail

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Durée** | **Activité** | **Remarques** |
| 30.08.2022 | 7h55 | * Conception et réalisation de la planification initiale * Création du rapport * Recherche et analyse et choix des technologies | Aucune |
| 31.08.2022 | 7h55 | * Avancement rapport * Création planning final * Création du repository Git et upload du projet * Réalisation de la maquette figma | Maquette : réalisation de la « page calendrier » |
| 01.09.2022 | 7h55 | * Avancement rapport * Avancement maquette figma * Création et mise en place de l’application : Création application React(***front-end***), configuration Electron(***Back-end***), communication entre les deux. | Temps planifié pour maquette dépassé |
| 05.09.2022 | 7h55 | * Implémentation : * Création service pour données * Mise en place le « routing » de l’application * Création page calendrier * Création page client * Création du menu de navigation | Dans la planification initiale il est défini un jour entier pour la mise en place des services pour interaction aux données, mais cela va se faire tout au long du développement de l’application. |
| 06.09.2022 | 7h55 | * Création de composants pour la page client : * Affichage liste des clients * Affichage Modification clients |  |
| 07.09.2022 | 7h55 | * Ajout client dans le composant « ClientDetails » * Révision du routing de la page clients, composant :« ClientPage » |  |
| 08.09.2022 | 7h55 | * Installation sqlite * Création des 2 DB (dev et prod) avec outil « DbBrowserForSqLite » * Création des tables de la BD (si elles n’éxistent pas) à l’initiation de l’app electron.js * Révision du processus d’ajout d’un client (front): composant « ClientDetails » > composant « ClientPage » > dataService qui appelle l’api electron  > electron sauve en BD le nouveau client |  |
| 12.09.2022 | 7h55 | * Réalisation des operations ***CRUD*** *pour la table « clients »* * Implémentation de la page « clientPage » |  |
| 13.09.2022 | 7h55 | * Réalisation des operations CRUD pour la table «project» * Implémentation de la page « projectPage » * Réalisation composant « Element » générique et réutilisable, utilisé pour afficher/éditer/ajouter les projets |  |
| 14.09.2022 | 7h55 | * Avancement Documentation * Réalisatoin page client avec composant Element * Réalisation affichage subComponent |  |
|  |  |  |  |

## 

## Manuel d'Installation

## Manuel d'Utilisation

## Archives du projet

*Media, … dans une fourre en plastique*