RAPPORT DE STAGE :

Développer Full-Stack :

Réalisation d'un site Web et création d'une application web et mobile

Stage de 4e année

Nom : Ghislain LEVREAU

Formation : 4e année Ingénieur Polytech Marseille

Année : 2022-2023

Entreprise : SARL Turcan

Adresse : 1530 route des grandes blaches 04200 Mison

Stage du : 5 juin au 31 juillet 2023

Application Web

Application web et mobile

Pour continuer dans cette lancée numérique, je me suis attelé à la réalisation d'une application opérationnelle à la fois sur Web et sur mobile. Cette application vise à permettre aux employés de Turcan d'avoir un suivi précis de leurs horaires.

À travers toutes les étapes clés du développement de l'application, de la conception initiale à la mise en œuvre finale, mon parcours a été jalonné par divers challenges. Le processus de prise de décision, la résolution des problèmes rencontrés, et les ajustements effectués en cours de route pour répondre aux besoins changeants de l'entreprise ont été autant d'occasions d'apprentissage et de croissance personnelle.

Le but principal de ce projet est double. D'une part, l'objectif est de fournir une application fonctionnelle qui répond aux besoins spécifiques de l'entreprise, notamment en termes de gestion des tâches, de communication, et de suivi des performances. D'autre part, ce projet constitue une opportunité d'apprentissage précieuse. Il m'a permis d'acquérir des connaissances pratiques et théoriques approfondies en PHP, Angular, et Ionic. Ces compétences non seulement me serviront dans le futur, mais elles sont également essentielles pour mon rattrapage « App web et mobile ».

1. Apprentissage

Avant de commencer la conception de cette application, j'avais besoin d'élargir et de rafraîchir mes connaissances.

Ma première étape a été de m'orienter vers la plateforme *OpenClassroom*, un choix judicieux grâce à sa richesse de contenus gratuits et pertinents. J'ai commencé par le cours intitulé "Réalisez la maquette d'une application mobile avec Adobe XD". J'ai appris à visualiser et à créer des interfaces utilisateur attrayantes.

Ensuite, je me suis plongé dans le monde d'Angular, un framework incontournable pour le développement web. Grâce à deux cours spécifiques, "Débutez avec Angular" et "Complétez vos connaissances sur Angular", j'ai pu acquérir une solide base sur ce sujet, allant des principes fondamentaux à des aspects plus avancés du framework.

Pour couronner le tout, j'ai revu un cours que j'avais déjà terminé sur le développement Full Stack. Le cours "Passez au Full Stack avec Node.js, Express et MongoDB" m'a permis de consolider mes connaissances, garantissant que je disposais des compétences nécessaires pour créer une application complète et fonctionnelle.

Après m'être bien formé, j'ai pu commencer la conception de l'application.

2. Backend Php

a. Schéma de la base de données

Dans cette phase du projet, j'ai défini le schéma de la base de données qui sera utilisé pour soutenir les fonctionnalités de mon application. L'objectif était de déterminer les informations nécessaires à stocker et comment elles seraient organisées pour permettre une gestion efficace des données. Comme le but du projet été de permettre aux employés de gérer leurs horaires, et les fiches amiantes, j'ai identifié trois tables principales : Users, Shifts, et AsbestosReports, chacune ayant un rôle spécifique dans la prise en charge de mon application.

Table Users :

Cette table stocke les informations des utilisateurs qui peuvent se connecter à l'application.

- userID (clé primaire, auto-incrémentée)
- firstName
- lastName
- email (unique)
- password (stockée sous forme de hash)
- role (par exemple, "employé", "gestionnaire", etc.)

Table Shifts :

Cette table stocke les informations sur les horaires de travail des employés.

- shiftID (clé primaire, auto-incrémentée)
- userID (clé étrangère, référence userID dans la table Users)
- startShift (date et heure)
- endShift (date et heure)
- breakTime
- shiftDate

Table AsbestosReports :

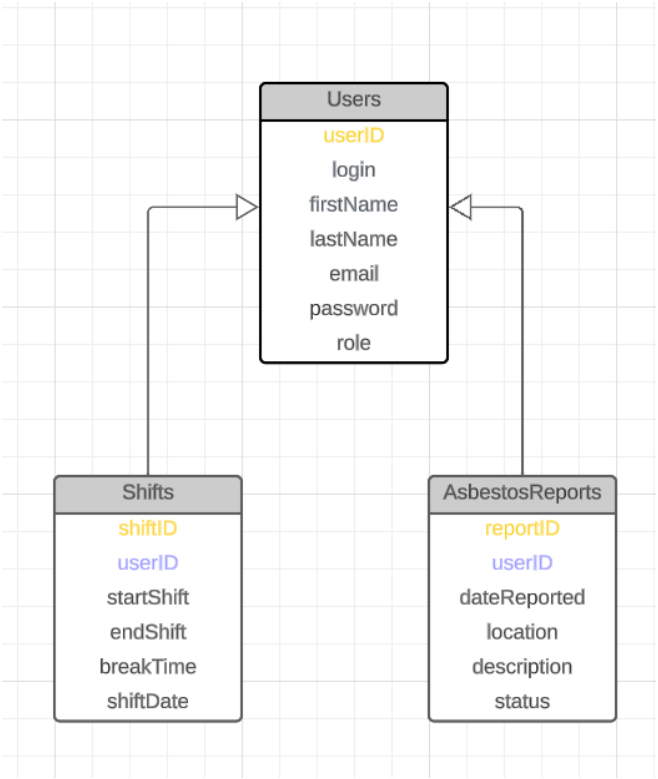
Cette table stocke les informations sur les rapports d'amiante.

- reportID (clé primaire, auto-incrémentée)
- userID (clé étrangère, référence userID dans la table Users)
- dateReported (date)

location

description

status (par exemple, "en attente", "en cours de traitement", "résolu")

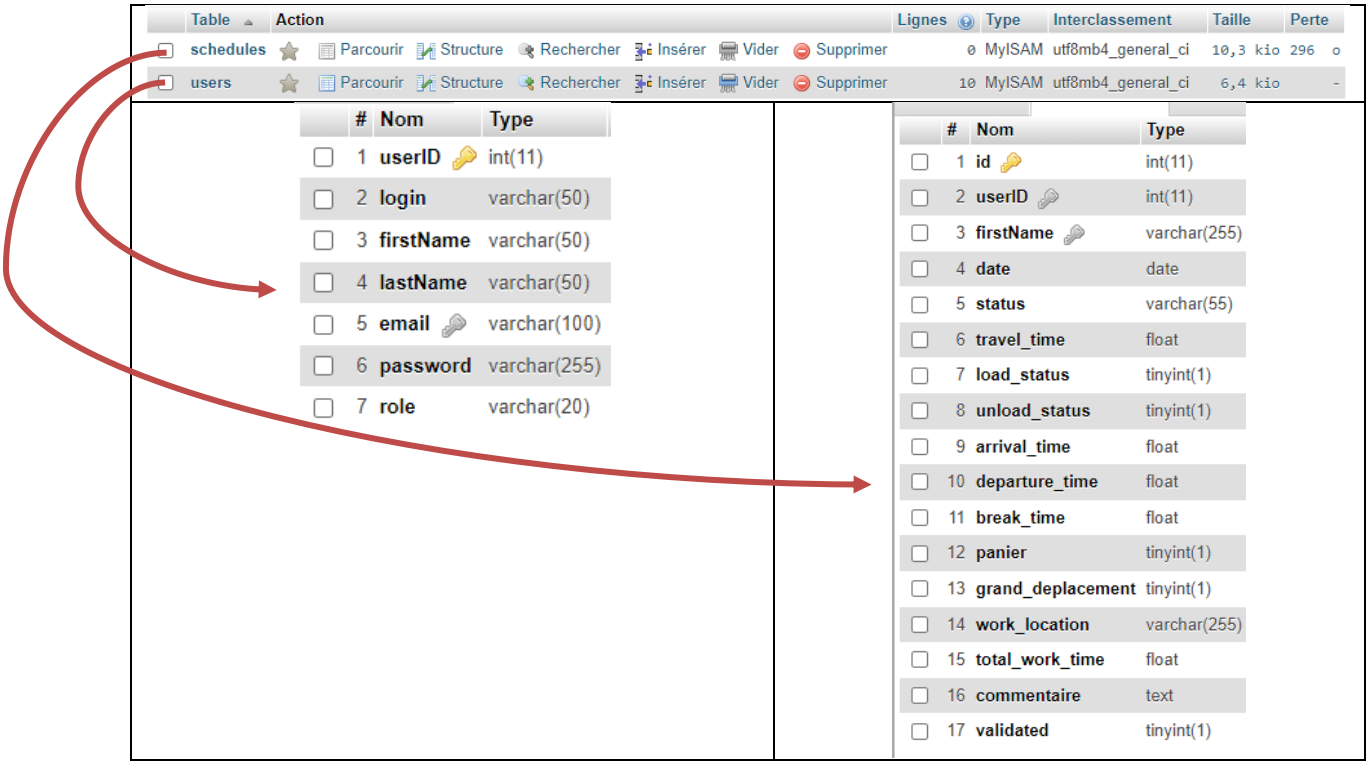


b. Création de la base de données

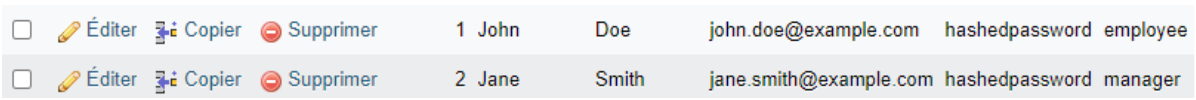
Dans cette phase, j'ai mis en œuvre mon schéma de base de données en créant une base de données MySQL et en définissant les tables Users, et Schedule.

Au fil du développement les tables ont un peu été modifié et par manque de temps, la partie *Désamiantage* n'a pas été faite. Je me suis donc focalisé sur les horaires.

Chaque table a été créée avec les champs appropriés pour stocker les informations nécessaires à mes fonctionnalités d'application.



Ensuite, j'ai inséré des données de test dans chaque table pour m'assurer que ma base de données fonctionne correctement et pour me préparer aux étapes de développement ultérieures.



c. Gestion de la Connexion à la base de donnée

Premièrement, pour gérer la connexion à notre base de données MySQL, nous avons mis en place deux scripts, **config.php** et **mysqlConnect.php**. Le premier contient nos paramètres de connexion centralisés, tandis que le second établit une connexion effective en utilisant les fonctionnalités de PHP Data Objects (PDO). Cette séparation nous offre non seulement une clarté, mais aussi une modularité dans la gestion de nos connexions.

En ce qui concerne le test de notre base de données, nous avons créé un script PHP, **getUsers.php**, qui récupère tous les utilisateurs. Pour affiner nos tests, nous avons ajouté la capacité de rechercher des utilisateurs spécifiques par leur nom d'utilisateur.

L'authentification est gérée en deux parties. La première utilise le script **auth.php** pour gérer les sessions, garantissant la persistance de l'authentification de l'utilisateur à travers les requêtes. La seconde partie, toujours dans **auth.php**, comprend des

fonctions pour authentifier un utilisateur et vérifier cette authentification en se basant sur des informations postées et des sessions.

Enfin, pour faciliter notre travail backend, nous avons mis en place **helper.php**. Ce script fournit des outils essentiels, comme la gestion des requêtes entre sites via CORS, des fonctions pour envoyer des messages standardisés, et des instructions pour vérifier l'authentification avant d'accéder à certaines fonctions backend.

3. Communication Frontend / Backend

La communication entre le frontend et le backend est obligatoire pour récupérer des informations de la base de donnée pour les afficher à l'utilisateur. Notre système utilise Angular comme technologie frontend et PHP comme backend.

a. Le Service de Messagerie: MessageService

Au cœur de notre système se trouve le service MessageService, conçu pour gérer toutes les interactions HTTP entre Angular et PHP.

Et dans ce service, une méthode **sendMessage** construit l'URL complète à partir d'un suffixe, il rajoute le nom du fichier et met le type de fichier, ici .php et envoie une requête POST au backend.

```
private urlPrefix = 'https://sarlturcan.tech/backend/';
…
let fullUrl = this.urlPrefix + url + '.php';
…
return this.http.post<PhpData>(fullUrl, formData, {withCredentials:
true}).pipe(
```

Nous avons également injecté notre **MessageService** dans notre **LoginComponent** et avons appelé la méthode **sendMessage** lorsque le bouton "se connecter" est cliqué.

b. Intégration avec d'autres services

Le service MessageService est conçu pour être réutilisé par d'autres services. Par exemple, pour récupérer le planning d'un utilisateur, nous utilisons la méthode getSchedulesForUser dans un autre service, en spécifiant le nom du fichier php.

```
getSchedulesForUser(userName: string): Observable<ScheduleResponse>
{
  return this.messageService.sendMessage(`getScheduleForUser`,
```

```
{user_Name: userName});
}
```

c. Stockage des Informations de Session:

Pour une expérience utilisateur fluide, certaines informations d'utilisateur sont stockées localement dans le localStorage après la connexion. Cela inclut des détails tels que le token de session, l'ID de l'utilisateur, le nom et le rôle de l'utilisateur.

```
this.messageService.sendMessage('checkLogin', this.user).subscribe(
  response => {
    if (response.status === 'ok') {
      this.errorMessage = null;

      // Store the session token and user info in LocalStorage
      localStorage.setItem('session_token', response.data.token);
      localStorage.setItem('user_id', response.data.user_id);
      localStorage.setItem('user_name', response.data.user_name);
      localStorage.setItem('user_role', response.data.user_role);
```

Grâce à ses sauvegardes locales, je peux récupérer rapidement des informations. Par exemple le rôle :

```
getUserRole(){
  return  localStorage.getItem('user_role');
}
```

d. Configuration du Backend pour la Communication Cross-Origin:

En raison de la nature cross-origin de nos requêtes (le frontend et le backend étant hébergés sur des domaines différents), des en-têtes spécifiques ont été configurés du côté backend pour autoriser ces communications.

```
if (isset($_SERVER['HTTP_ORIGIN'])) {
    header("Access-Control-Allow-Origin: {$_SERVER['HTTP_ORIGIN']}");
} else {
    // Utiliser une origine par d?faut si HTTP_ORIGIN n'est pas d?fini
    header("Access-Control-Allow-Origin: http://default-origin.com");
}
// Autoriser les m?thodes GET, POST et OPTIONS.
header('Access-Control-Allow-Methods: POST, GET, OPTIONS');

// Autoriser les cookies.
header('Access-Control-Allow-Credentials: true');

// Autoriser certains types de requ?tes.
header('Access-Control-Allow-Headers: Content-Type, Authorization');

// Autoriser l'acc?s ? certaines ressources.
header("Content-Security-Policy: default-src 'self'
http://34.155.130.193/;");

header('Content-type:application/json;charset=utf8');
```

Conclusion:

La mise en œuvre de la communication entre Angular et PHP a été conçue pour être robuste, sécurisée et facilement extensible. En utilisant des services dédiés, en stockant des informations de manière sécurisée et en respectant les meilleures pratiques de communication cross-origin, nous avons construit une base solide pour notre application.

4. Frontend Angular

a. Création du Frontend

Dans cette étape, j'ai commencé à construire la partie frontend de l'application en utilisant le cadre Angular. Le processus a commencé par la création d'un nouveau projet Angular à l'aide de l'interface de ligne de commande Angular (CLI), en executant la commande ng new, en précisant les options --routing et --style=scss.

L'option --routing nous a permis d'ajouter le routage Angular à notre application, ce qui est essentiel pour naviguer entre les différentes parties de notre application. L'option --style=scss a été utilisée pour indiquer que nous souhaitions utiliser le format SCSS pour les fichiers de style de notre application. SCSS est un préprocesseur CSS qui ajoute des fonctionnalités supplémentaires à CSS, comme les variables et les boucles, ce qui facilite la rédaction de styles plus lisibles et maintenables.

Une fois que notre application a été créée, nous avons lancé le serveur de développement Angular à l'aide de la commande ng serve et avons vérifié que notre application fonctionnait correctement en la visitant dans un navigateur web à l'adresse http://localhost:4200.

b. Installation des packages

Dans cette étape, j'ai installé plusieurs packages nécessaires pour développer notre application de gestion d'horaires en utilisant le gestionnaire de packages Node.js (npm). Le processus d'installation a été effectué dans le répertoire frontend.

Les packages installés sont :

Bootstrap: Il s'agit d'une bibliothèque très populaire pour la création de sites web réactifs et mobiles. Elle offre une variété de composants prédéfinis de CSS et JavaScript qui nous permettent de développer des sites web rapidement et

efficacement. La documentation de Bootstrap nous sera utile pour déterminer quelles classes utiliser pour obtenir l'apparence que nous voulons pour nos pages.

@angular/material et @angular/cdk: Angular Material est une bibliothèque qui fournit un ensemble de composants d'interface utilisateur prédéfinis qui suivent les principes de conception du Material Design. Ces composants nous aideront à créer une interface utilisateur cohérente et attrayante pour notre application. CDK (Component Development Kit) est une boîte à outils qui fournit des utilitaires de haut niveau pour construire des composants d'interface utilisateur personnalisés.

@ckeditor/ckeditor5-angular et @ckeditor/ckeditor5-build-classic: Ces packages fournissent un éditeur de texte riche pour Angular, ce qui nous permettra d'avoir une interface utilisateur pour saisir nos messages de forum. CKEditor 5 est une solution d'édition de texte en ligne moderne avec une interface utilisateur modulaire, une architecture de plugin flexible et une API riche.

Ces packages vont grandement simplifier le processus de développement de l'application et aider à créer une interface utilisateur de haute qualité.

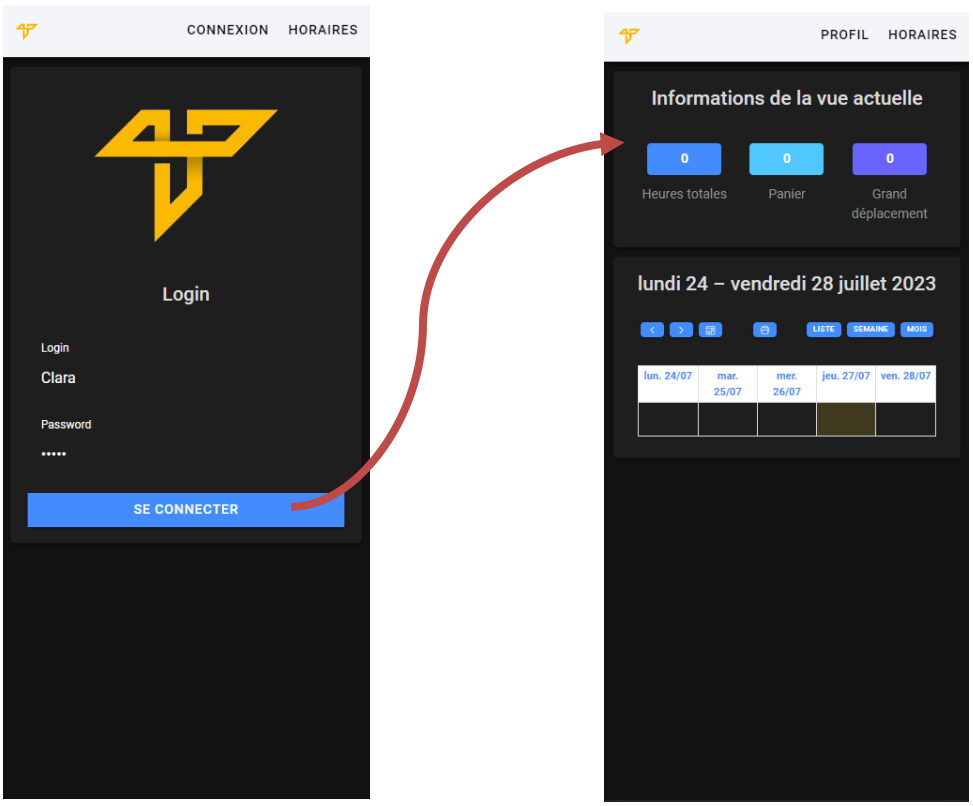
c. Navbar

J'ai créé un composant de barre de navigation pour mon application à l'aide de la commande Angular CLI. Cette "navbar" est positionnée en haut de chaque page, garantissant une navigation fluide et cohérente. J'ai utilisé des classes fournies par Bootstrap pour son design et ajouté la fonctionnalité "fixed-top" pour qu'elle reste en haut même en défilant. Elle contient des liens comme l'accueil, les horaires et le profil, rendus possibles grâce à "routerLink" d'Angular. J'ai ensuite intégré cette barre de navigation dans le fichier principal pour qu'elle soit visible partout. Enfin, j'ai testé mon application pour m'assurer que tout fonctionnait bien.

d. Connexion

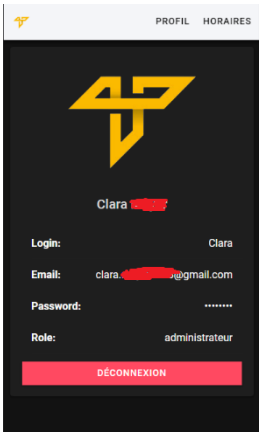
J'ai créé un composant "login" avec Angular pour gérer la connexion des utilisateurs. En tapant l'URL '/login', on voit le message « login works ! », ce qui montre que le composant et la route fonctionnent bien. Chaque nouveau composant doit avoir une route dans app-routing.module.ts.

Dans le template HTML de cette page de connexion, j'ai ajouté des champs pour le login et le mot de passe. Grâce à la liaison bidirectionnelle d'Angular, les informations entrées sont reliées à des variables du composant. Lorsque l'utilisateur clique sur le bouton de connexion, les informations saisies sont transmises par le service de messagerie au backend. En utilisant le script checkLogin.php, les données de login et le mot de passe (préalablement hashé) sont vérifiés dans la base de données. Si ces informations correspondent à une entrée existante dans la base, la connexion est autorisée. Dans le cas contraire, il est redirigé vers la page d'accueil.



Pour une meilleure expérience, j'ai ajouté la gestion des erreurs. Un attribut 'errorMessage' stocke les messages d'erreur du backend. Si un utilisateur essaie de se connecter sans remplir les champs, il voit un message d'erreur sur un fond rose, utilisant les alertes Bootstrap. Si 'errorMessage' est vide, aucun message n'apparaît, pour éviter de perturber inutilement l'utilisateur.

Si la connexion est réussie, l'utilisateur est dirigé vers la page Horaires. De plus, il a accès maintenant à une page Profil qui récapitule ses informations :



- Nom
- Prénom
- Email
- Password
- Role

e. Page Horaires

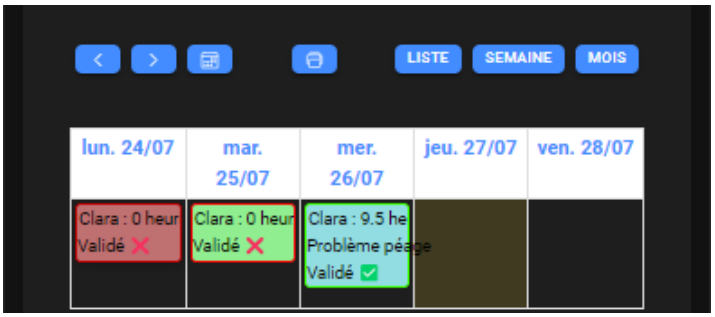
Lorsque j'ai conçu la page "horaires", j'ai été confronté à une complexité majeure. Cette page avait pour objectif d'afficher les horaires des employés, mais avec une nuance : chaque employé ne devait voir que ses propres horaires, sauf les administrateurs. Ces derniers avaient un privilège spécial : ils pouvaient visualiser tous les horaires et, grâce à un sélecteur de nom, choisir quels horaires consulter.

J'ai utilisé le plugin FullCalendar pour réaliser cet affichage sous forme de calendrier interactif. Pour ce faire, j'ai importé plusieurs modules et addons, comme dayGridPlugin, timeGridPlugin et interactionPlugin, qui m'ont fourni les outils nécessaires pour personnaliser et rendre le calendrier aussi fonctionnel que je le voulais.

```
//----Calendrier Import------//
import {CalendarOptions, Calendar, EventClickArg, EventApi, ViewApi,
EventInput} from '@fullcalendar/core';
import { FullCalendarComponent } from '@fullcalendar/angular';
import dayGridPlugin from '@fullcalendar/daygrid';
import timeGridPlugin from '@fullcalendar/timegrid'
import interactionPlugin, {DateClickArg} from '@fullcalendar/interaction';
// for dateClick
import listPlugin from '@fullcalendar/list';
```

Cela me permet de choisir entre plusieurs affichages. Dans l'application j'utilise liste, qui affiche une liste des horaires de la semaine, un affichage calendrier de la semaine et du mois.

De plus j'ai ajouté des flèches pour naviguer dans le calendrier.



À l'ouverture de la page, selon les droits de l'utilisateur, une requête est envoyée au backend. Si l'utilisateur est un employé standard, la requête dirige vers getScheduleForUser.php pour récupérer uniquement ses horaires. Si c'est un administrateur, alors tous les horaires sont extraits via getAllDateForUser.php.

Le fonctionnement de FullCalendar est assez ingénieux : lors de son initialisation, le calendrier se construit et je crée pour chaque horaire de la base de donnée, un "event". Ces "events" sont ensuite stockés dans un tableau. Chaque "event" a un ID qui correspond à l'ID de l'horaire dans la base de données. Ainsi, en récupérant un "event", je peux retrouver précisément l'horaire associé.

Les "events" sont créés comme ceci :

```
let event = {
  id: item.id,
  title: item.total_work_time + ' heures', // default title
  subtitle: item.commentaire,
  validation: '',
  date: item.date,
  status: item.status,
  // Mapping the rest of the keys to your interface
  travel_time: item.travel_time,
  load_status: item.load_status,
  unload_status: item.unload_status,
  arrival_time: item.arrival_time,
  departure_time: item.departure_time,
  break_time: item.break_time,
  panier: item.panier,
  grand_deplacement: item.grand_deplacement,
  work_location: item.work_location,
  total_work_time: item.total_work_time,
  commentaire: item.commentaire,
  validated:  item.validated,

  // Style properties
  color: 'blue',
  textColor: 'black',
  backgroundColor: '#92dde1',
  borderColor: 'darkblue',
  classNames: ['my-custom-class'],
  display: 'block',
};
```

Chaque propriété de la base de donnée est reportée dans "l'event".

Comme on peut le voir dans les propriétés, les "events" sont validés ou non. Seuls les administrateurs peuvent valider un "event". Et les utilisateurs voient sur leur calendrier si un horaire est validé ou non. Cela permet un suivi des horaires.

Voyons maintenant les fonctionnalité de chaque horaire respectant le CRUD:

- **Create (Création) :**

Ajouter un nouvel horaire est simple. En cliquant sur une date vierge, une fenêtre intitulée "Ajout d'horaire" s'ouvre, présentant divers champs à compléter. Une fois les données obligatoires saisies, elles sont transmises au backend via le script addSchedule.php. L'horaire nouvellement créé est alors affiché sur le calendrier et sauvegardé dans la base de données.

- **Read (Lecture) :**

Consulter un horaire est accessible à tous. En cliquant sur un horaire déjà enregistré, une fenêtre contextuelle s'ouvre, montrant toutes les options possibles pour cet horaire. Sélectionnez "Consulter" pour afficher en détail toutes les informations associées à cet horaire.

- **Update (Mise à jour) :**

Modifier un horaire est possible de tous, que vous soyez administrateur ou simple utilisateur. Il est possible de mettre à jour chaque élément de votre horaire. Notez cependant qu'une modification annulera la validation de l'horaire, si elle avait été faite auparavant.

- **Delete (Suppression) :**

Enfin, chaque horaire peut être supprimé. La procédure pour supprimer est tout aussi intuitive que les autres fonctionnalités.

5. Transformation Ionic

Dans le cadre de mes efforts pour élargir la portée de mon application, j'ai entrepris de migrer mon application web basée sur Angular vers une application mobile construite avec Ionic. La migration reste beaucoup de copié-collé, mais il faut vérifier et adapter certaines fonctionnalités sur mobile.

a. Mise en place du projet Ionic

J'ai initié la migration en créant un nouveau projet Ionic à l'aide de la commande ionic start. En choisissant un projet "blank" basé sur Angular comme point de départ, j'ai assuré une transition fluide puisque mon application originale était également basée sur Angular.

J'ai commencé par éliminer la page "home" par défaut. Elle ne m'était pas utile. Au lieu de cela, j'ai mis en place une page de connexion, qui était essentielle pour mon

application. J'ai également ajusté la route par défaut pour qu'elle redirige vers cette nouvelle page de connexion.

b. Intégration de mes services existants

Mon application Angular avait déjà des services robustes, notamment pour l'authentification et la messagerie. Ces services représentent le cœur fonctionnel de mon application, assurant l'authentification, la communication et la gestion des données.

- **Intégration du service d'authentification (auth.service):**

Ce service est crucial pour assurer la sécurité de mon application. Il vérifie si un utilisateur est connecté et authentifié. J'ai réintégré toutes les méthodes de ce service sans oublier les vérifications d'authenticité.

➔ **Utilisation du Guard:**
Le service d'authentification est étroitement associé à un mécanisme de guard. Ce guard assure que certaines routes, généralement sensibles, ne sont accessibles qu'aux utilisateurs authentifiés. Si un utilisateur non authentifié tente d'accéder à une route protégée, il est automatiquement redirigé vers la page de connexion.

- **Intégration du service de communication (message.service):**

Le pont entre le backend et le frontend est assuré par le **message.service**.

➔ **Fonctionnement:**
Ce service utilise des requêtes HTTP pour communiquer avec le serveur backend, permettant d'envoyer et de recevoir des données. Grâce à sa structure modulaire, je peux facilement ajouter, modifier ou supprimer des points d'extrémité si nécessaire.

- **Intégration du service de profil (profil.service):**

Le service de profil joue un rôle clé dans la gestion des informations de l'utilisateur. Il permet à l'utilisateur de consulter ses informations, de les modifier si nécessaire et de gérer sa déconnexion.

➔ **Fonctionnalités principales:**

Récupération des informations: Ce service peut demander et recevoir des détails tels que le nom, l'adresse électronique et d'autres données personnelles de l'utilisateur.

Déconnexion: Il fournit également une méthode pour déconnecter l'utilisateur, assurant ainsi la sécurité des sessions.

- **Intégration du service des horaires (schedule.service):**

La gestion des horaires est un élément essentiel de mon application. J'ai donc veillé à réintégrer ce service dans mon application Ionic.

➔ **Fonctionnalités:**
Ce service offre une gamme de fonctionnalités liées à la gestion des horaires, telles que la création, la modification, la suppression et la consultation des horaires.

c. Création de mes pages principales

Les services sont globalement du copié-collé, cependant la création des pages et différentes. Cela se rapproche à du Html comme avec Angular mais il faut utiliser des balises Ion. Par exemple voici la page de connexion :

```
<ion-content>
  <ion-card class="ion-text-center">
    <img src="assets/TURCAN_Logo.png" class="profile-picture">
    <ion-card-header>
      <ion-card-title>Login</ion-card-title>
    </ion-card-header>

    <ion-card-content>
      <ion-list>
        <ion-item>
          <ion-label position="floating">Login</ion-label>
          <ion-input aria-label="login" type="text"
[(ngModel)]="user.login" name="login" required></ion-input>
        </ion-item>

        <ion-item>
          <ion-label position="floating">Password</ion-label>
          <ion-input aria-label="password" type="password"
[(ngModel)]="user.password" name="password" required></ion-input>
        </ion-item>
      </ion-list>

      <ion-button expand="full" color="primary" (click)="login()">Se
connecter</ion-button>

      <ion-text color="danger" *ngIf="errorMessage">
        <p>{{ errorMessage }}</p>
      </ion-text>
    </ion-card-content>
  </ion-card>
</ion-content>
```

J’ai donc recréer chaque page de mon application Angular en essayant un maximum de leur ressembler mais en adaptant au format téléphone.

6. Build & Hébergement

La transition vers le monde mobile et web a nécessité une stratégie de déploiement adaptée pour TurcanAPP, tenant compte des différentes plateformes et des besoins des utilisateurs. Voici le détail du processus que j'ai suivi:

a. Build de l'application

- **Sur navigateur:** J'ai d'abord buildé l'application Angular pour une utilisation sur le navigateur, garantissant ainsi une accessibilité maximale pour tous les utilisateurs, quelle que soit leur plateforme.
- **Sur mobile avec Android:** La construction pour Android a été réalisée en utilisant la commande **ionic cordova build android**, ce qui a permis d'obtenir une version APK prête à être installée sur des appareils Android. J’ai pu la tester avec Android Studio et sur téléphone directement.
- **Limitations pour iOS:** Malheureusement, je n'ai pas pu build pour iOS. La principale raison est que je ne dispose pas d'un Mac et que je n'ai pas de licence de développeur Apple. Cette contrainte a nécessité une stratégie alternative pour les utilisateurs d'iPhone.

b. Distribution de l’Application :

Compte tenu des contraintes mentionnées, j'ai opté pour une distribution à la fois sur navigateur et sur Android. Ainsi, les utilisateurs d'iPhone peuvent simplement utiliser la version navigateur de l'application.

J’ai décidé de distribuer l’application seulement aux employés donc l’app est trouvable nulle part. Elle est sauvegardée sur un drive disponible avec un QR code.

Pour assurer une installation aisée, j'ai préparé deux tutoriels :

- Un pour guider l'installation sur Android via l'APK.
- Et un autre pour aider à accéder à la page des horaires sur le site et à ajouter un raccourci sur le téléphone : TurcanAPP Horaires.

c. Hébergement :

- **Problèmes avec Google Console:** Mes premières tentatives d'hébergement avec Google Console ont rencontré des obstacles. Mon backend étant en HTTP, des restrictions de sécurité ont empêché le fonctionnement optimal de l'application.
- **Solution avec Hostinger:** J'ai alors décidé de déployer l'application Android directement sur mon site web hébergé par Hostinger. D'où l'URL spécifique **https://sarlturcan.tech/backend/** pour l'application. Pour la gestion des données, j'ai utilisé une base de données MySQL également hébergée sur Hostinger.
- **Modification du Site Web:** Pour une expérience utilisateur fluide, j'ai remplacé la page "horaires" existante du site par l'application web. De plus, pour éviter des connexions redondantes, j'ai supprimé toutes les fonctionnalités de connexion du site, sauf celle de l'application. Il est à noter que la page "horaires" n'est pas référencée pour des raisons de sécurité.

Annexes

1. Backend

config.php :

```php
<?php

const DB_HOST = '**.***.**.**';  // Le serveur de la base de données
const DB_NAME = 'turcanapp';  // Le nom de votre base de données
const DB_USER = ****;  // Le nom d'utilisateur pour la base de données
const DB_PASSWORD = '****';  // Le mot de passe pour la base de données

?>
```

mysqlConnect.php :

```php
<?php
// fichier mysqlConnect.php
require_once 'config.php';  // Importe les détails de la connexion à la base de
données

global $db;


try {
    $db = new PDO("mysql:host=".DB_HOST.";dbname=".DB_NAME.";charset=utf8",
DB_USER, DB_PASSWORD);
    // configure PDO pour lancer une exception lorsque des erreurs se produisent
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die("Erreur de connexion à la base de données : " . $e->getMessage());
}
?>
```

getUsers.php :

```php
<?php

global $db;
require_once 'mysqlConnect.php';
require_once 'helper.php'; // pour acc�der aux fonctions sendMessage et sendError

$stmt = $db->prepare('SELECT * FROM users');  // Pr�pare la requ�te SQL avec un
param�tre
$stmt->execute();  // Ex�cute la requ�te avec le param�tre
$firstName = $stmt->fetchAll(PDO::FETCH_ASSOC);  // R�cup�re tous les utilisateurs

// Imprime le r�sultat de la requ�te SQL
sendMessage($firstName);

?>
```

helper.php :

```php
<?php
if (isset($_SERVER['HTTP_ORIGIN'])) {
    header("Access-Control-Allow-Origin: {$_SERVER['HTTP_ORIGIN']}");
} else {
    // Utiliser une origine par d?faut si HTTP_ORIGIN n'est pas d?fini
    header("Access-Control-Allow-Origin: http://default-origin.com");
}
// Autoriser les m?thodes GET, POST et OPTIONS.
header('Access-Control-Allow-Methods: POST, GET, OPTIONS');

// Autoriser les cookies.
header('Access-Control-Allow-Credentials: true');

// Autoriser certains types de requ?tes.
header('Access-Control-Allow-Headers: Content-Type, Authorization');

// Autoriser l'acc?s ? certaines ressources.
header("Content-Security-Policy: default-src 'self' http://34.155.130.193/;");

header('Content-type:application/json;charset=utf8');




/**
 * la fonction permettant d'envoyer un message JSON au frontend
 * @param $data
 */
function sendMessage ($data) {
  echo json_encode ([ 'status' => 'ok',
                      'data'   => $data ]);
  die;
}


/**
 * la fonction permettant d'envoyer un message d'erreur au frontend
 * @param $reason
 */
function sendError ($reason) {
  echo json_encode (['status' => 'error',
                     'data'   => ['reason' => $reason] ]);
  die;
}

?>
```

checkLogin.php :

```php
<?php
require_once 'mysqlConnect.php';
require_once 'helper.php'; // inclus le fichier helper.php
require_once 'config.php'; // inclus le fichier de configuration de la base de
donn?es

error_log("Start of script.");

if(!isset($_POST['login']) || !isset($_POST['password'])) {
    error_log("Login or password not set.");
    sendError('Les champs login et/ou password ne sont pas d?finis.'); // envoie
un message d'erreur si les champs login ou password ne sont pas d?finis
} else {
    $login = $_POST['login'];
    $password = $_POST['password'];
```

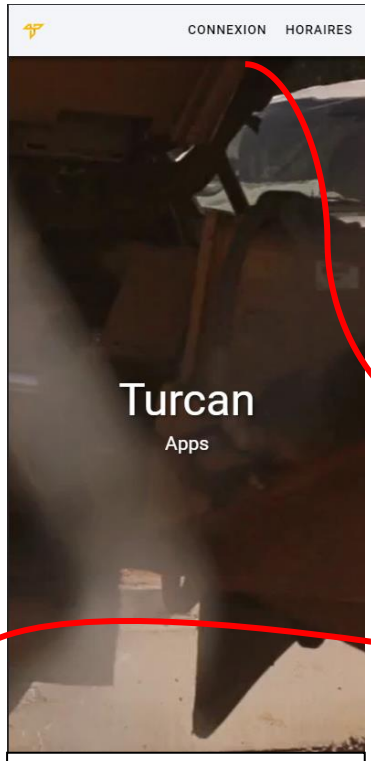
```
    error_log("Login and password set, executing SQL.");

    $stmt = $db->prepare("SELECT * FROM users WHERE login = :login AND password =
:password");
    $stmt->execute([':login' => $login, ':password' => $password]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    error_log("SQL executed.");

   if($user) {
        error_log("User found, generating token.");
    $token = bin2hex(random_bytes(16)); // Generate a random token
    error_log("Token generated: $token");

    $response = [
        'status' => 'ok',
        'token' => $token,
        'user_id' => $user['userID'],
        'user_name' => $user['firstName'],
        'user_role' => $user['role']
        ];
    error_log("Response created: " . json_encode($response));
     sendMessage($response);

    error_log("Token sent.");

    } else {
        error_log("No user found.");
        sendError('Login/Password invalide.');
    }
}
?>
```

2. Frontend

MessageService :

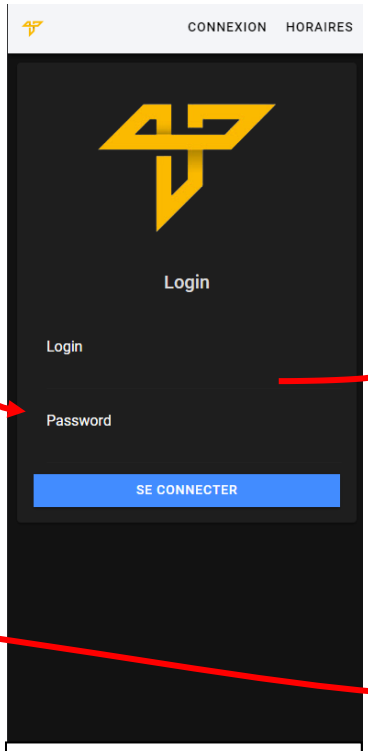
```
export class MessageService {
  private urlPrefix = 'https://sarlturcan.tech/backend/';

  constructor(private http: HttpClient) { }

  sendMessage(url: string, data: any): Observable<any> {
    let fullUrl = this.urlPrefix + url + '.php';
    let formData = new FormData();

    if (data !== null && data !== undefined) {
      Object.keys(data).forEach(key => {
        formData.append(key, data[key]);
      });
    }

    return this.http.post<PhpData>(fullUrl, formData, {withCredentials:
true}).pipe(
      catchError(error => {
        // This will return an Observable that emits a single value: an object with
`error` and `message`
        if (error instanceof HttpErrorResponse && error.error instanceof
ErrorEvent) {
          // A client-side or network error occurred.
          console.error('An error occurred:', error.error.message);
        } else {
          // The backend returned an unsuccessful response code.
          console.error(`Backend returned code ${error.status}, body was:
${error.error}`);
          console.log('Response body:', error.error);
        }
        return throwError({error: true, message: error.message});
      })
    );
  }


}
```

3. Fonctionnalités

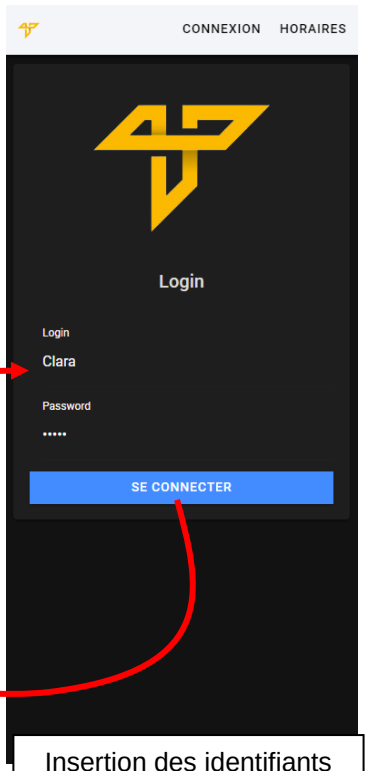


Page d'accueil

➔ Lancement de l'app

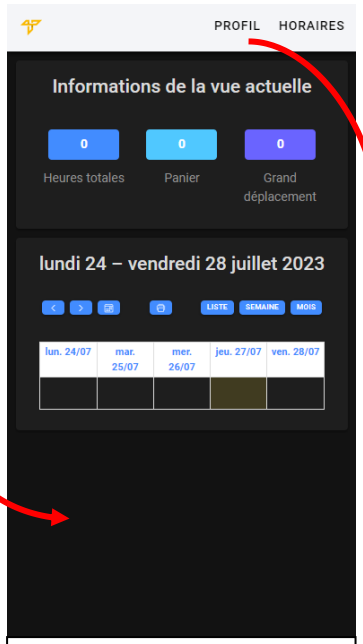


Page de connexion

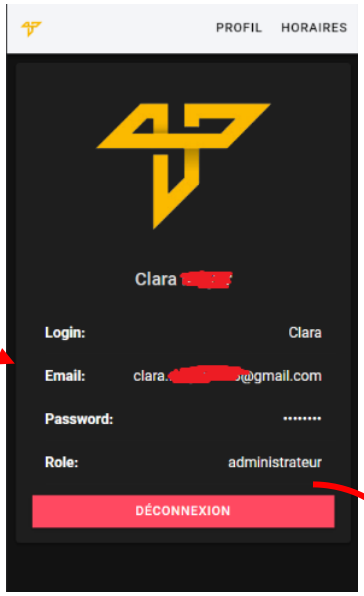


Insertion des identifiants

Stocké dans la bdd (copie sur un excel)

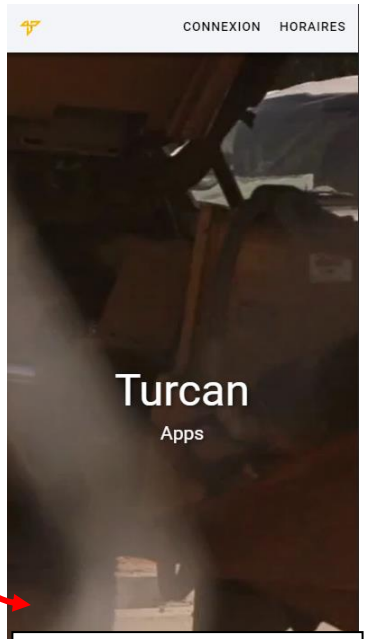


Page horaires



Page profil

Avec les informations de l'employé



Retour à la page d'accueil

Informations de la période/vue :
-Nombre d'heure de travail total
-Nombre de panier total
-Nombre de grand déplacement total

Calendrier pour une période :
- Boutons de navigation de période
- Retour à aujourd'hui
- PDF

Changement de vue :
1. Liste des horaire de la semaine
2. Vue semaine
3. Vue Mois
4.

Horaire
Absence
Congé

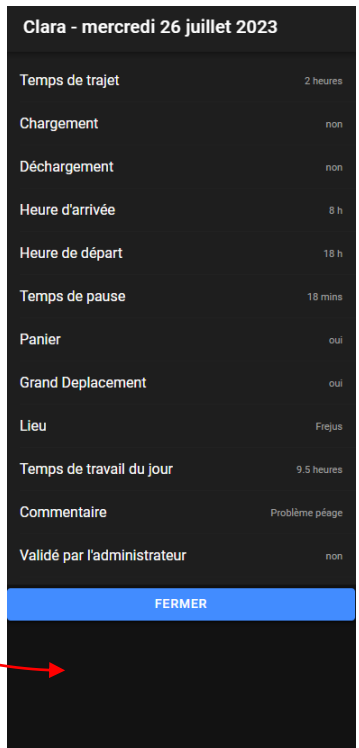
Contour rouge / Pas Validé par l'administrateur
Contour vert / Validé par l'administrateur

Nom
Nbr d'heure de travail de la journée
Commentaire
Validé ou non

Sélection de l'employé à voir

Page horaires

Cliquer sur un horaire affichera cette pop-up



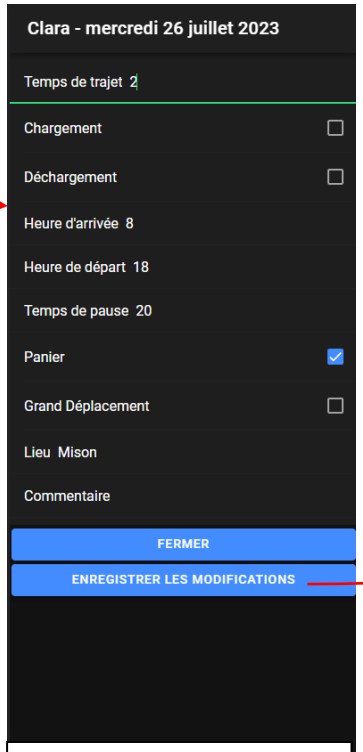
On affiche l'horaire avec toutes ses informations

Pop-up de l'horaire

Validé

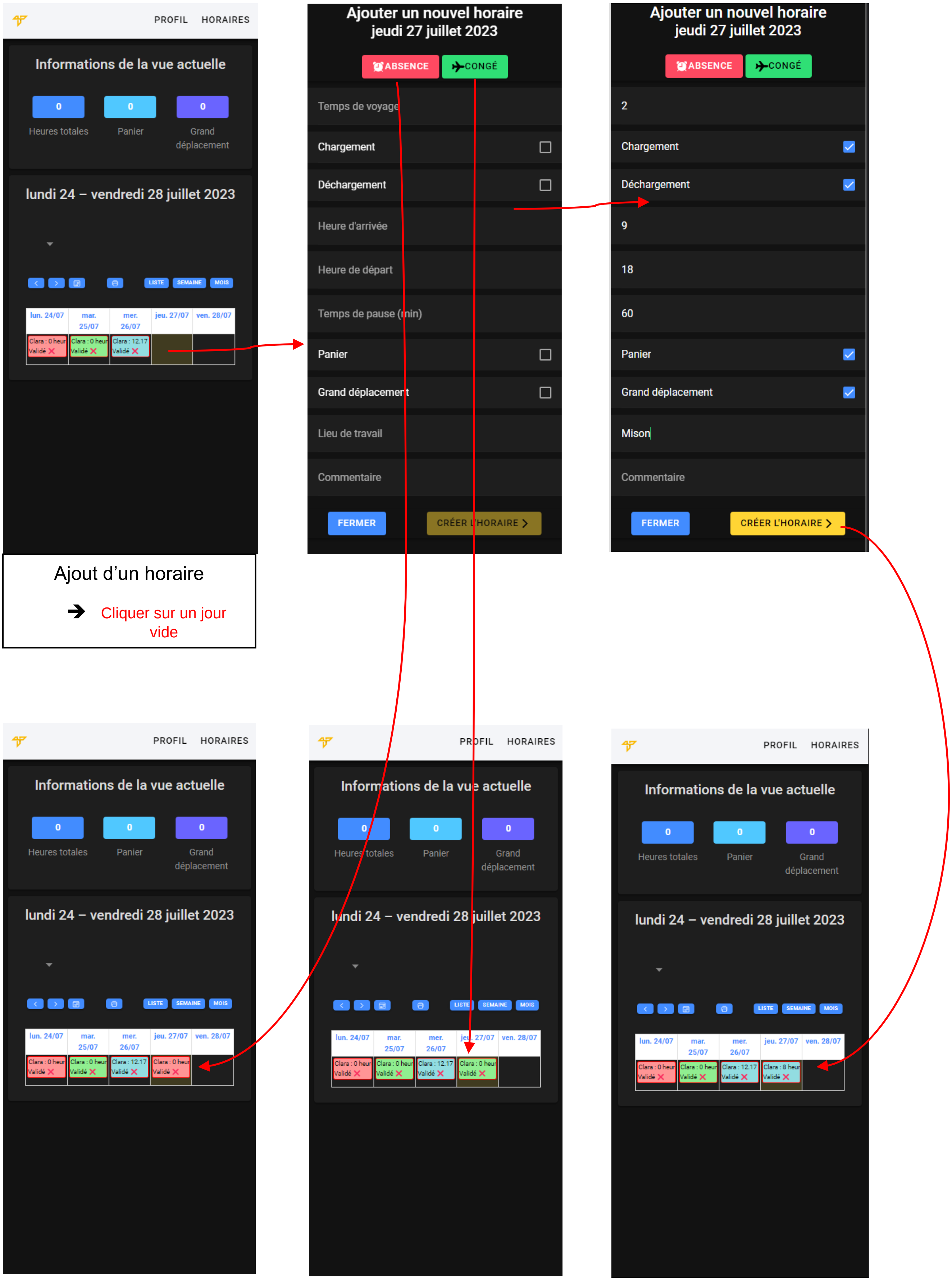
Le faire 2 fois (petit problème)

Suppression d'un horaire



Modification d'un horaire

Le total d'heure de la journée est recalculé



Ajout d'un horaire

➔ Cliquer sur un jour vide