Livrable	**3**
Sigle du cours	**SEG2505**
Professeur	**Aziz Oukaira**
Assistant (e)	**Alexia Capo-Chichi**
Groupe	**G03**

École de science informatique et génie électrique

uOttawa.ca

uOttawa

Membres du groupe
- Céline Wan Min Kee (celinewmk), 300193369
- Evan Marth (emarth), 300166093
- Samy Touabi (SamyT-code), 300184721
- Tisham Islam (TishamIslam), 300189261
- Othniel Tiendrebeogo (othnielt), 300084968

Livrable 3: Implémentation des fonctionnalités liées à l'employé

1. Diagramme UML pour le livrable 3

Nous avons continué le diagramme UML de notre application en ajoutant les classes que nous avions besoin afin de réaliser le livrable 3. Ensuite, nous avons partagé les tâches entre les différents membres du groupe. Nous avons ajouté 5 nouvelles activités qui seront nécessaire pour ce livrable:

- EmployeePage: page d'accueil pour les comptes employés
- ServicesFournisPage: page qui affiche les services fournis par la succursale
- SuccursaleTimePage: page qui permet d'entrer et de modifier les heures de travail de la succursale
- SuccursaleDemandsPage: page qui permet de visualiser les demandes de services et les approuver/rejeter
- AllServicesAvailPage: page qui affiche tous les services offerts par Service Novigrad

Nous avons aussi implémenté des nouvelles classes Java Helpers, Interval et IntervalList qui nous sont utiles pour les heures de travail. Nous avons aussi implémenté DemandeList qui nous permet d'avoir une liste défilante pour afficher les demandes de services.

Pour la diagramme UML, on a décidé d'omettre quelque méthodes et classes: chaque page a une méthode *onReturn(View)* (ou équivalente) pour aller à la page précédente et une méthode *onCreate()*/*onStart()* pour initialiser les variables et mettre des *listeners* pour la base de données. Donc, ils ne sont pas dans le diagramme pour le simplifier. Les composantes de UI (ex: boutons) sont aussi omises pour la simplification. Il y aussi quelques classes qui sont omises car elles représentent la même chose que des tableaux (chaque classe qui a un préfixe "List", ex: IntervalList et DemandeList). Ils fonctionnent identiquement aux tableaux/listes, mais ont aussi des fonctionnalités pour le UI.

Il faut noter que la classe Succursale est seulement utilisée pour initialiser les données d'une succursale quand un employé est créé, grâce à la méthode *toString()*. Firebase ne permet pas de sauvegarder des tableaux vides, donc les modifications de données pour une succursale sont effectuées sans utiliser la classe elle-même.

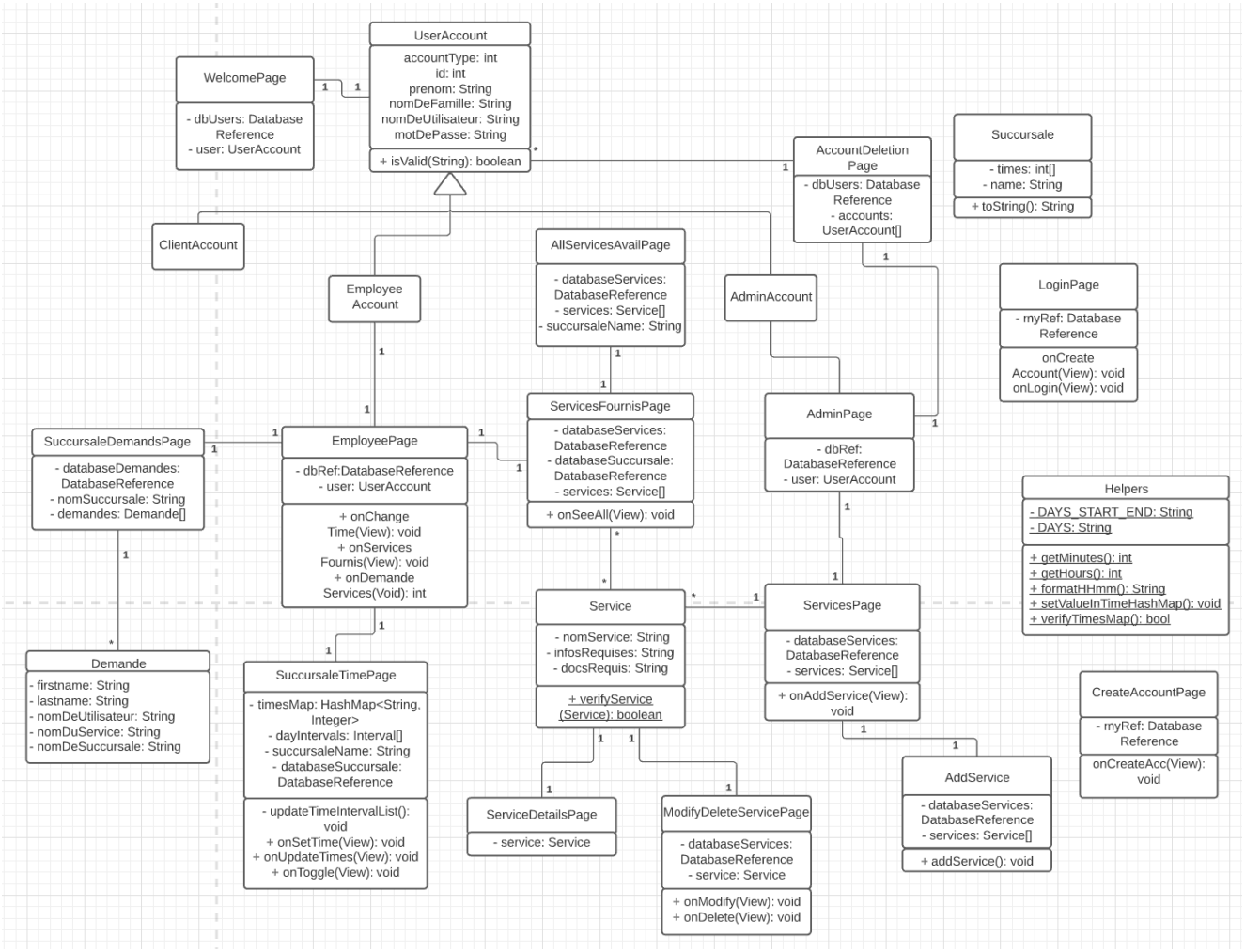

Figure 1: Diagramme UML

Tisham et Céline ont créé la classe Succursale et Demande respectivement

```java
public class Succursale {
    private String name;
    private String[] serviceRefs;
    // goes from monday to sunday
    // e.g. times[0] is monday start time, times[1] is monday end time
    private int[] times;

    public Succursale(String username) {
        this.name = username;
        this.serviceRefs = new String[0];
        this.times = new int[14];
        for (int i = 0; i < times.length; i++) {
            times[i] = (i % 2 == 0 ? 9 * 60 : 17 * 60); //set all the times to be 9-5 by default
        }
    }

    private String timesToString() {
        String[] days = new String[]{"\"lunA\"","\"lunB\"","\"marA\"","\"marB\"","\"merA\"","\"merB\"","\"jeuA\"","\"jeuB\"
        String timesString = "{";
        for (int i = 0; i < times.length; i++) {
            timesString += days[i] + ": " + times[i] + (i==times.length-1 ? "}" : ",");
        }
        return timesString;
    }

    private String serviceRefsToString() {
        String serviceRefsString = "{";
        if (serviceRefs.length == 0) {
            serviceRefsString += "}";
        }
        for (int i = 0; i < serviceRefs.length; i++) {
            serviceRefsString += "\""+ i + "\": \"" + serviceRefs[i] + (i==serviceRefs.length-1 ? "\"}" : "\",");
        }
        return serviceRefsString;
    }

    //returns a string in JSON format so it can be sent to firebase
```

Figure 2: Classe Succursale

```java
public class Demande {

    // instance variables
    private String firstName;
    private String lastName;
    private String nomDeUtilisateur;
    private String nomDuServiceDemande;
    private String nomSuccursaleDemande;

    // constructeur 1
    public Demande(){}

    // constructeur 1
    public Demande(String firstName, String lastName, String nomDeUtilisateur, String nomDuServiceDemande, String nomSuccur
        this.firstName = firstName;
        this.lastName = lastName;
        this.nomDeUtilisateur = nomDeUtilisateur;
        this.nomDuServiceDemande = nomDuServiceDemande;
        this.nomSuccursaleDemande = nomSuccursaleDemande;
    }

    // getters
    public String getFirstName() { return firstName; }

    public String getLastName() { return lastName; }

    public String getNomDeUtilisateur() { return nomDeUtilisateur; }

    public String getNomDuServiceDemande() { return nomDuServiceDemande; }

    public String getNomSuccursaleDemande() { return nomSuccursaleDemande; }

    @Override
    public String toString() {
        return "Demande faite par " + firstName + " " + lastName + " pour le service \"" + nomDuServiceDemande +"\"";
    }
}
```

Figure 3: Classe Demande

2. Design des nouvelles activités de l'application

Céline, Samy et Tisham ont travaillé sur le design et le front-end des nouvelles activités de l'application en utilisant ce qui a été appris dans les laboratoires sur Android Studio.

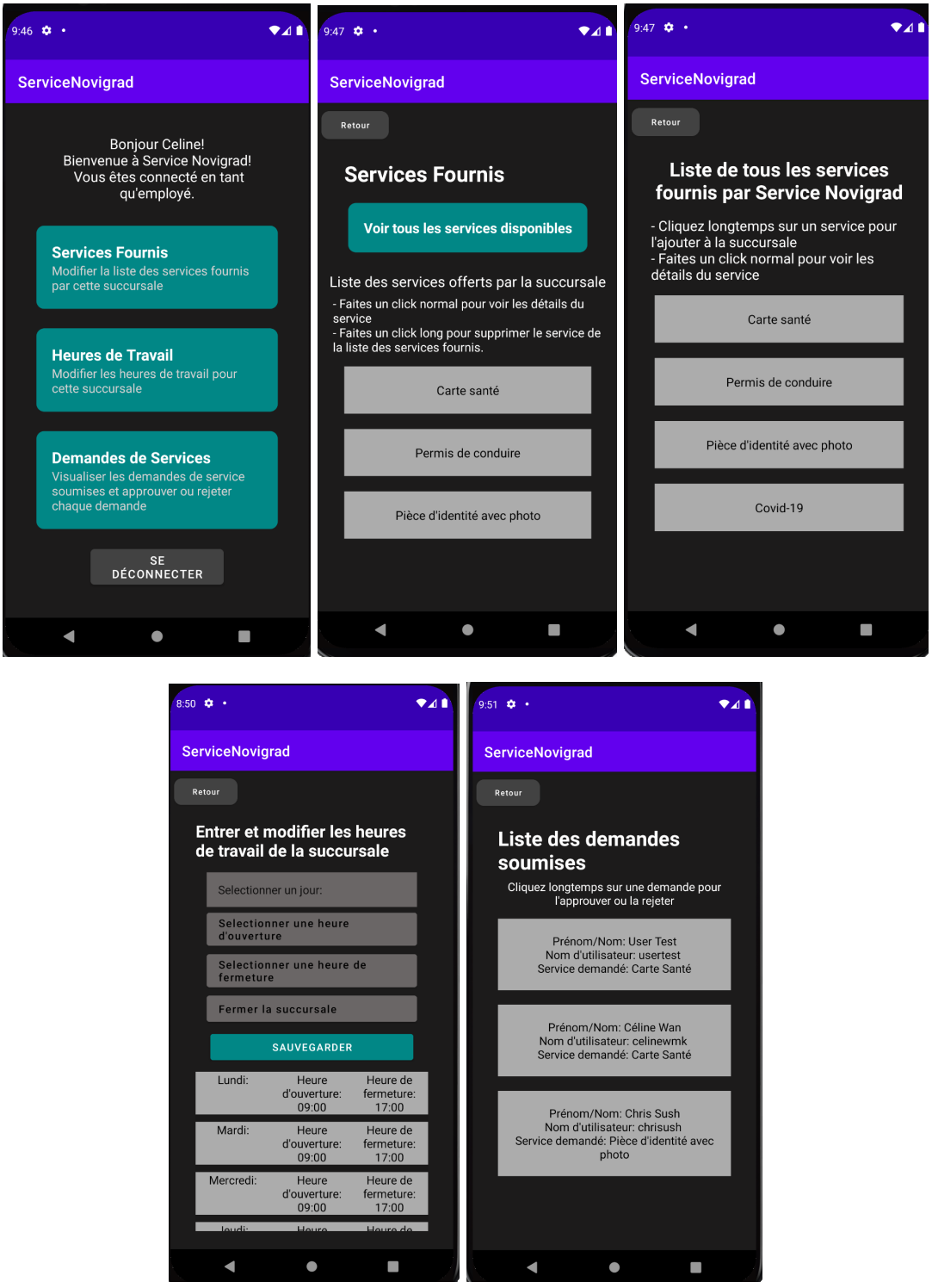

Figure 4: Les activités pour les fonctionnalités liées à l'employeur

- Ajout/suppression d'un service à la succursale

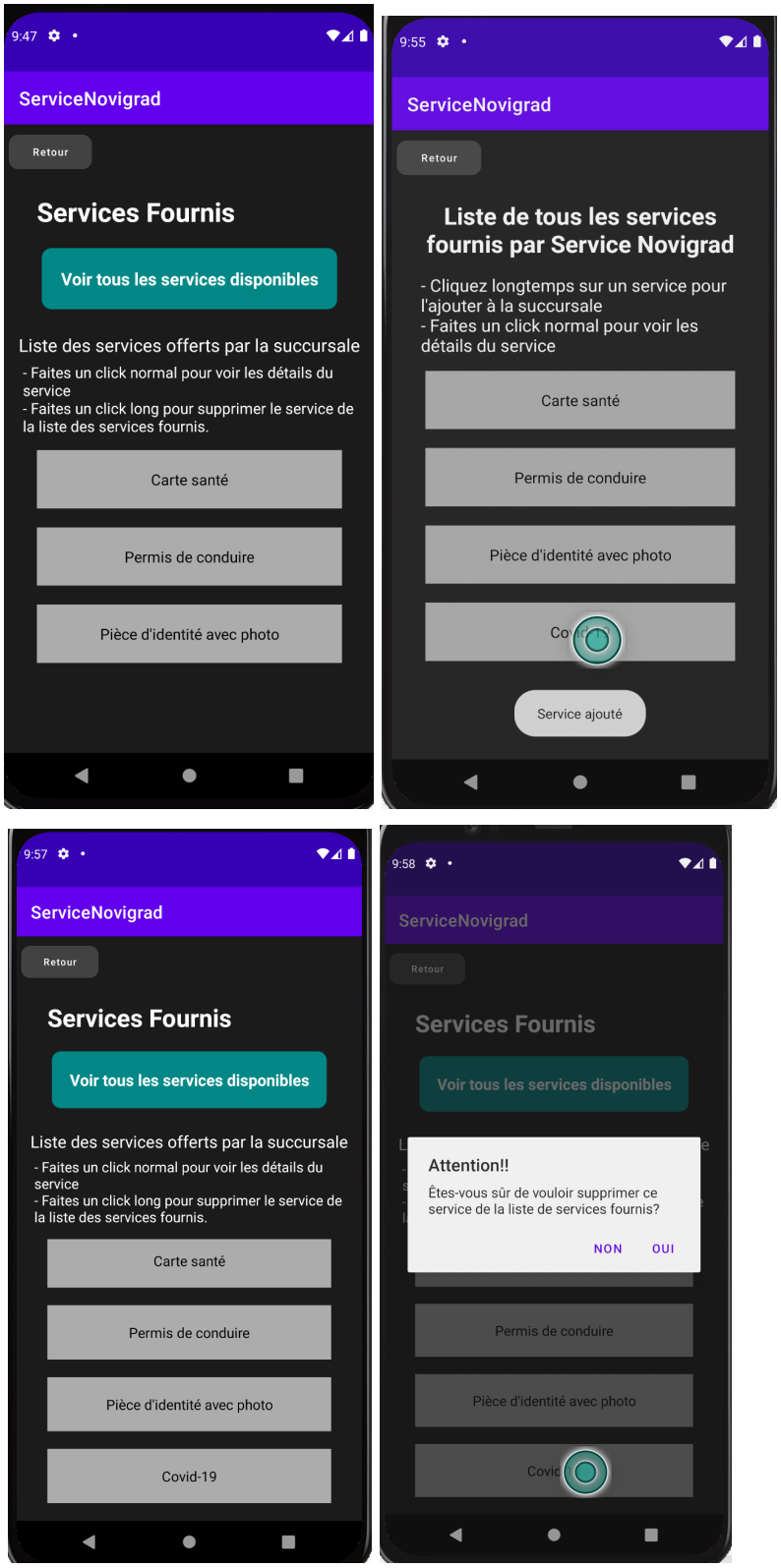

Figure 5: Démonstration de l'ajout et suppression du service

- Approuver une demande

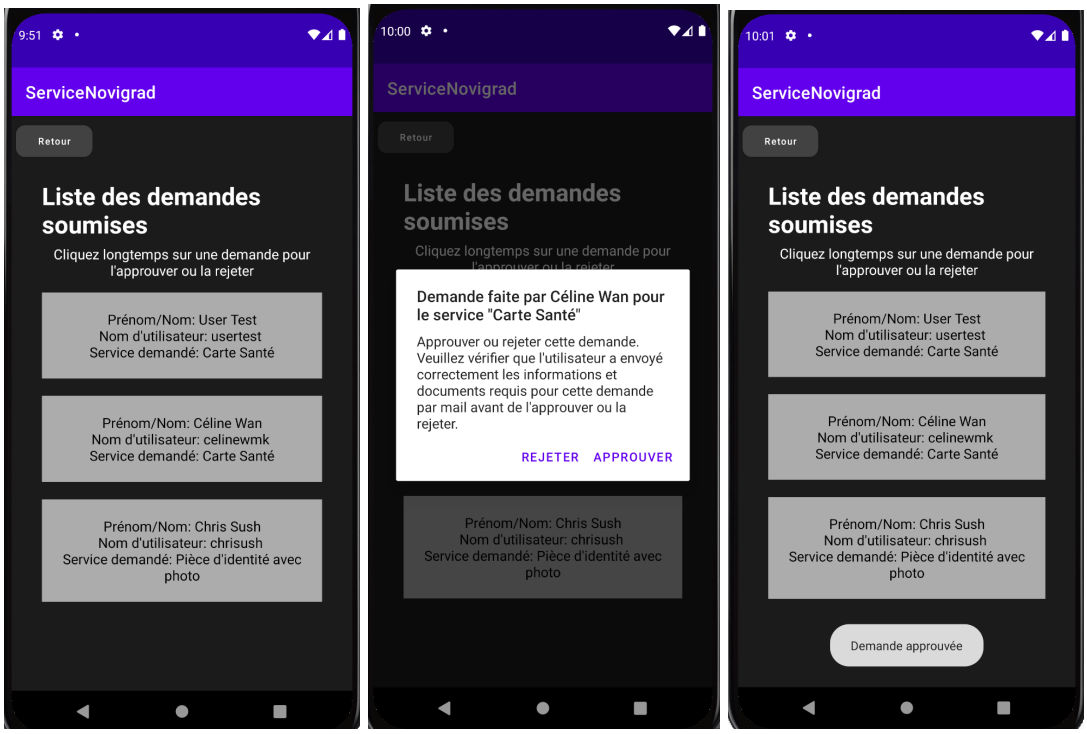

Figure 6: Démonstration pour approuver une demande (le comportement est le même pour rejeter)

Note: Veuillez noter que pour le moment le bouton ‘approuver’ ou ‘rejeter’ une demande n’effectue pas entièrement son rôle car nous avons besoin d’implémenter des fonctionnalités liées au client pour montrer et signaler si une demande a véritablement été approuvée ou non.

- Entrer/modifier les heures de travail de la succursale

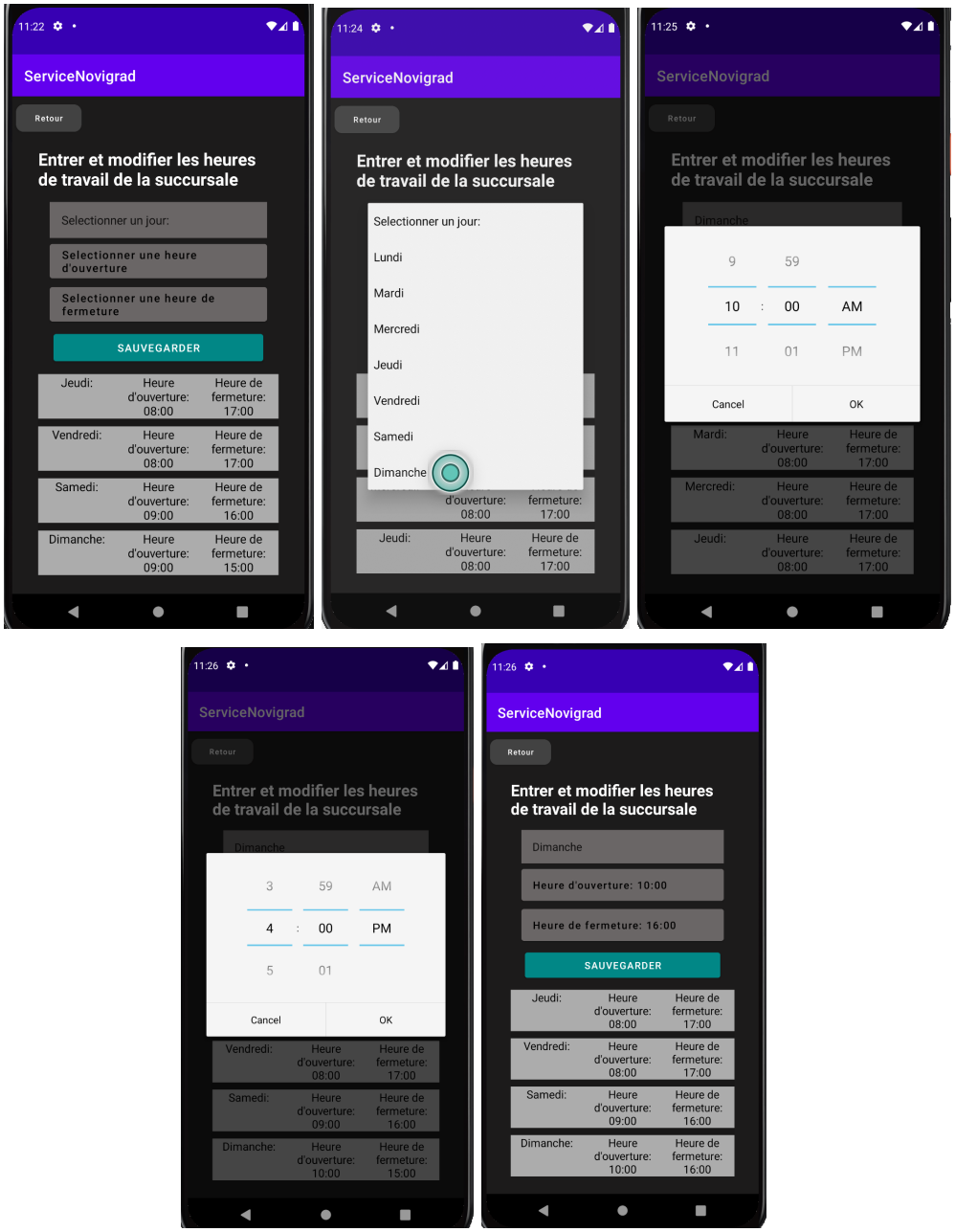

Figure 7: Démonstration de la modification des heures de travail par défaut avec une nouvelle horaire

- Possibilité de fermer la succursale un jour

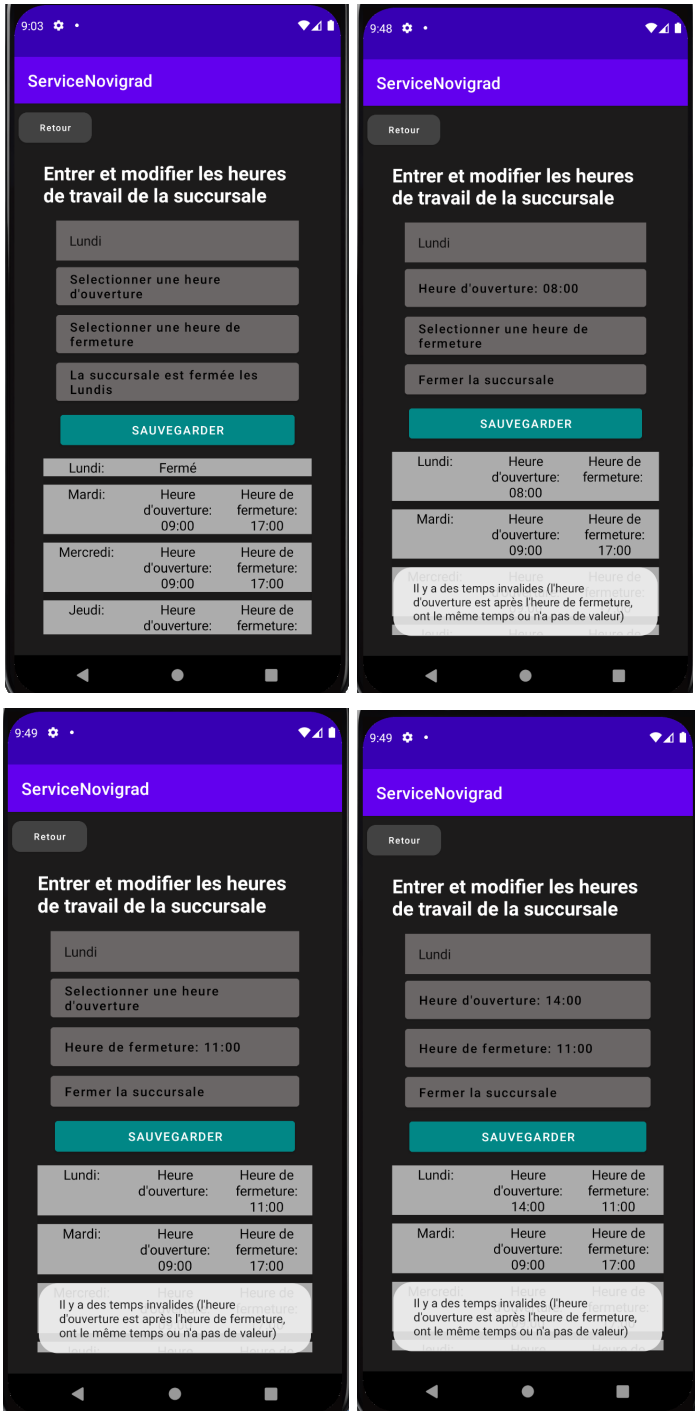

Figure 8: Fermeture de la succursale le lundi avec des messages Toast qui s'affichent lorsqu'une des deux heures n'est pas sélectionné pour la ré-ouvrir, les deux heures sont les mêmes ou si l'heure d'ouverture est après l'heure de fermeture

3. Backend de l'application

Tisham et Céline ont travaillé sur le backend de l'application pour le livrable 3. Nous gardons en mémoire les références aux services fournis par une succursale (donc ils sont toujours à jour), ainsi que les heures de travail où elle est opérationnelle. Nous gardons aussi en mémoire les demandes de services soumises par les clients.

Figure 9: Aperçu de la base de donnée qui garde en mémoire les informations nécessaire à la succursale (son nom, les services fournis, ses heures opérationnelles)

Figure 10: Aperçu de la base de donnée qui garde en mémoire les temps d'horaire par défaut et les demandes soumises à la succursale

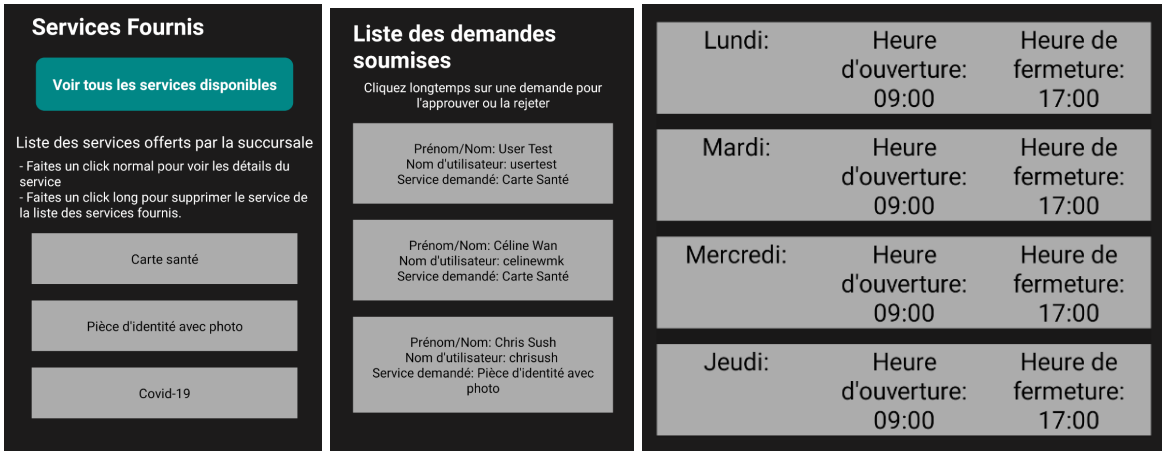

Figure 11: Affichage sur l'application des informations de la base de donnée (PS: pour l'image de droite: vendredi, samedi et dimanche sont affiché lorsqu'on défile en bas)

4. Test unitaires

Nous avons écrit 3 nouveaux tests unitaires pour la nouvelle classe Helpers.

```
public class HelpersTest {
    @Test
    // test that the time methods work properly
    public void testTimeGetters() {
        assertEquals(Helpers.getMinutes( timeInMinutes: 90), actual: 30);
        assertEquals(Helpers.getHours( timeInMinutes: 90), actual: 1);
        assertEquals(Helpers.getMinutes( timeInMinutes: 121), actual: 1);
        assertEquals(Helpers.getHours( timeInMinutes: 121), actual: 2);

        //test that the default values for the succ times are 9-17/9am-5pm, as expected
        assertEquals(Helpers.getMinutes( timeInMinutes: 540), actual: 0);
        assertEquals(Helpers.getHours( timeInMinutes: 540), actual: 9);
        assertEquals(Helpers.getMinutes( timeInMinutes: 1020), actual: 0);
        assertEquals(Helpers.getHours( timeInMinutes: 1020), actual: 17);
    }

    @Test
    public void testTimeApprox() {
        assertEquals(Helpers.approximateTime( minute: 59),  actual: 60);
        assertEquals(Helpers.approximateTime( minute: 50), actual: 45);
        assertEquals(Helpers.approximateTime( minute: 39), actual: 45);
        assertEquals(Helpers.approximateTime( minute: 32), actual: 30);
        assertEquals(Helpers.approximateTime( minute: 24), actual: 30);
        assertEquals(Helpers.approximateTime( minute: 22), actual: 15);
        assertEquals(Helpers.approximateTime( minute: 9), actual: 15);
        assertEquals(Helpers.approximateTime( minute: 7), actual: 0);
    }
    tishamIslam, Moments ago • added Helper tests
    @Test
    public void testFormatMethods() {
        assertEquals(Helpers.formatHHmm( timeInMinutes: 0), actual: "00:00");
        assertEquals(Helpers.formatHHmm( timeInMinutes: 121), actual: "02:01");
        assertEquals(Helpers.formatHHmm( timeInMinutes: 599), actual: "09:59");
        assertEquals(Helpers.formatHHmm( timeInMinutes: 1234), actual: "20:34");
        assertEquals(Helpers.formatHHmm( timeInMinutes: 754), actual: "12:34");
        assertEquals(Helpers.formatHHmm( timeInMinutes: 540), actual: "09:00");
        assertEquals(Helpers.formatHHmm( timeInMinutes: 1020), actual: "17:00");
    }
}
```

Figure 12: Test unitaires pour la classe Helpers

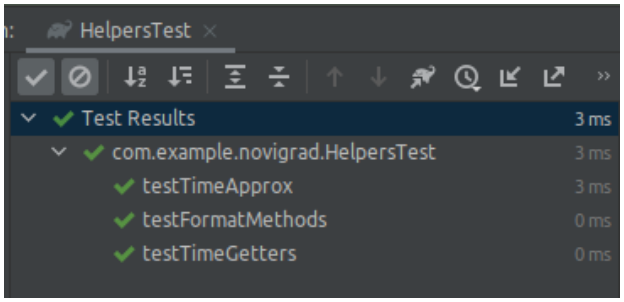

Figure 13: Résultats des tests unitaires

5. Ajustement du front-end de l'application de pages existantes:

Samy a modifié les champs de texte des fichiers activity_main.xml, activity_add_service.xml et AddService.java afin de rendre l'interface plus conviviale et d'ajouter une limite de caractères à certains champs de textes.

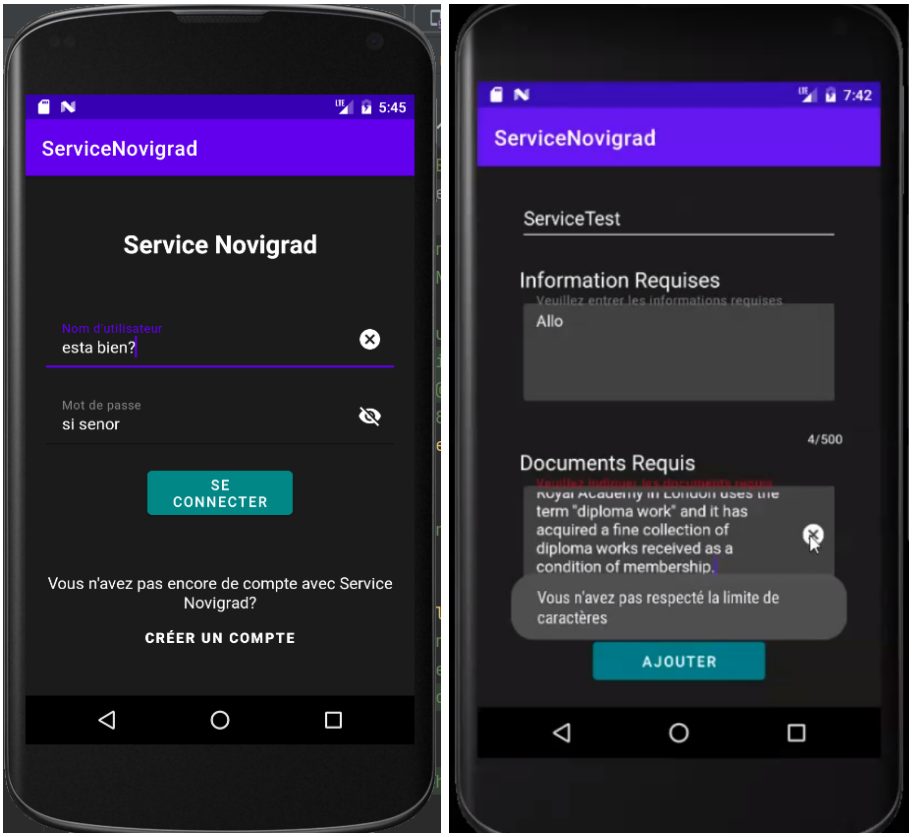

Figure 14: Ajout de boutons pour supprimer ce qui est écrit dans un champ de texte et pour rendre visible le contenu du mot de passe dans la page d'accueil, ainsi qu'une limite de mots pour les infos et documents requis avec le message Toast qui vient avec.

6. Ligne de temps de développement

- 22/11/2021: Évaluation des besoins du livrable 2 + Continuation du développement du diagramme UML + Développement front-end
    - Tisham a commencé à faire le design des nouvelles activités de l'application
- 25/11/2021: Suite du codage du front-end
    - Céline a continué le codage des nouvelles activités et a implémenté une restriction concernant la longueur du mot de passe lors de la création d'un compte
    - Samy a implémenté différentes options qu'on pourraient utiliser afin de modifier les heures de travail
- 27/11/2021: Codage du backend + front-end
    - Tisham a implémenté la fonctionnalité permettant à une succursale de choisir des services à proposer dans la liste de tous les services disponibles à Service Novigrad. Il a aussi permis à un employé de voir ses services fournis.
- 28/11/2021: Fonctionnalité de la modification des heures de travail et des demandes de service:
    - Céline a travaillé sur les demandes de service qui permettent de visualiser les services soumis à la succursale
    - Tisham a implémenté la fonctionnalité pour voir et modifier les heures de travail d'une succursale.
- 29/11/2021: Continuation du diagramme UML + rapport + Finition de l'application
    - Utilisation de LucidCharts au lieu de UMPle car il n'était plus pratique.
    - Evan a ajouté l'option qui permet de fermer une succursale un jour
- 30/11/2021: Samy a modifié le front-end de pages existantes de l'application + soumission du livrable
    - Effaceur de champ de texte, bouton pour rendre visible le mot de passe
    - Limite de l'information que peut entrer un utilisateur avec message toast

7. Conclusion

Dans ce livrable nous avons poursuivi le développement de notre application pour le Service Novigrad qui offre des services à ses clients. Nous avons cette fois-ci implémenté les fonctionnalités du compte employé de l'application. L'employé peut maintenant sélectionner des services pour la succursale, définir ses heures de travail et les modifier, ainsi que visualiser les demandes de services. Grâce à Firebase, notre application garde maintenant en mémoire tous les comptes qui sont créés dans l'application, la liste des services offerts par les succursales, la liste des services offerts par une succursale, les heures de travail. Nous avons aussi écrit 3 tests unitaires supplémentaires qui nous permettent de vérifier la fonctionnalité de la classe Helpers.