RAPPORT DE PROJET

Application UniSpace

Projet de fin de première année du cycle ingénieur

Vadim LEUPE
Nicolas BEAUDET
Kévin FELTRIN
Melchior LAURENS

Encadrant: M. Hassane MIMOUN

Année universitaire 2024-2025

Table des matières

Glossaire

API : Une API est un ensemble de fonctions et de protocoles qui permettent à des logiciels différents de communiquer entre eux.

Chatbot: Un chatbot est un programme informatique conçu pour interagir avec l'utilisateur via des messages, à la manière d'un humain

Cloud Function : Une Cloud Function est une fonction s'exécutant sur des serveurs gérés par Google lorsqu'un évènement se produit dans l'application ou lorsque la fonction est appelée par une requête. Elle s'adapte automatiquement au nombre d'utilisateurs ou à la fréquence des appels, sans configuration manuelle du serveur.

CSV : CSV (pour Comma Separated Values) est un format de fichier texte destiné à stocker des données sous forme de tableau. Les valeurs de chaque ligne sont séparées en général par des virgules ou des points-virgules.

Dart: Dart est un langage de programmation orienté objet créé par Google et utilisé notamment pour développer des applications Flutter.

Firebase : Firebase est une plateforme de développement proposée par Google, combinant plusieurs services, notamment de stockage, de base de données, de Cloud Functions ou encore d'authentification.

Firebase Cloud Storage : Firebase Cloud Storage est un service de stockage de fichiers intégré à Firebase

Flutter : Flutter est un framework développé par Google permettant de concevoir en Dart des applications mobile, web et ordinateur à partir d'un code unique.

Framework : un framework (appelé aussi infrastructure logicielle, infrastructure de développement, environnement de développement, socle d'applications, cadre

d'applications ou cadriciel) est un ensemble cohérent de composants logiciels structurels qui sert à créer les fondations ainsi que les grandes lignes de tout ou partie d'un logiciel, c'est-à-dire une architecture. [3]

FutureBuilder : FutureBuilder est un widget Flutter permettant d'afficher du contenu récupéré par une opération asynchrone, en affichant par exemple un chargement en attendant que les données demandées soient récupérées.

Refactor : Un refactor (ou « refactoring ») est le processus de réécriture du code d'un programme sans en modifier le comportement fonctionnel.

Sign In With Google : Sign In With Google est un service proposé par Google permettant aux utilisateurs de se connecter avec leur compte Google.

StreamBuilder : StreamBuilder est un widget Flutter permettant d'afficher du contenu provenant d'un flux, en affichant par exemple des données mises à jour en temps réel.

Token : Lorsqu'un utilisateur s'authentifie auprès d'un système, celui-ci génère un token unique qui est ensuite utilisé pour accéder aux ressources sans nécessiter une ré-authentification constante.

Widget : Les widgets de Flutter encapsulent tous les éléments de l'interface utilisateur, qu'il s'agisse d'éléments structurels comme les boutons et les interrupteurs ou d'éléments stylistiques comme les polices et les couleurs.

Xcode : Logiciel développé par Apple pour développer des applications sur leurs plateformes : iOS, watchOS, tvOS et visionOS.

Liste des figures

1. Contexte

1.1. Objectifs pédagogiques du projet

Le projet s'inscrit dans le cadre de la formation des élèves-ingénieurs d'ESIEE Paris et se manifeste par deux mois de travail en groupe de 4 étudiants. Il s'agit de mettre en pratique les concepts et méthodes appris durant les unités d'enseignement scientifiques et techniques, et de gestion de projet. Dans notre cas, nous avons proposé nous-mêmes le sujet, en réponse à un besoin identifié au sein de notre environnement quotidien, que nous avons ensuite soumis à l'équipe pédagogique pour validation

1.2 Sujet du projet

L'idée de notre projet naît d'un constat simple: en tant qu'étudiants d'ESIEE Paris, nous passons beaucoup de temps à chercher une salle dans laquelle nous installer pour travailler efficacement en groupe. En effet, les salles sont majoritairement occupées par des enseignements ou utilisées de manière informelle par d'autres étudiants, sans qu'il soit possible de connaître leur disponibilité en temps réel.

Face à ce besoin, nous proposons une application mobile et web intuitive qui permet aux étudiants de consulter la disponibilité des salles de classe à l'ESIEE Paris. Cette disponibilité est indiquée en fonction de l'emploi du temps des cours, et également des réservations des salles effectuées via notre système. En complément, notre application intègre une fonctionnalité de notification des créneaux durant lesquels les enseignants sont disponibles dans leur bureau pour répondre aux questions des étudiants, ce qu'on appelle communément les créneaux de "Post-Assistance".

2. Méthodologie

2.1 Etude de l'existant

Nous étudions les solutions existantes se rapprochant en termes de fonctionnalités de notre projet d'application, qu'elles soient développées par des étudiants ou disponibles sur le marché. L'objectif est d'identifier les opportunités et les axes d'amélioration de notre application.

2.2.1. Logiciels de gestion d'emploi du temps

ADE:

ADE est un logiciel créé par l'entreprise *ADE Soft* permettant la gestion des emplois du temps des établissements d'enseignement supérieur. Il s'agit de la manière officielle de consulter l'emploi du temps et les disponibilités des salles dans l'école.

Fonctionnalités principales:

- Le site web [1] indique dans un calendrier d'une durée d'une semaine les salles dans lesquelles ont lieu chaque cours, ou indique quels cours ont lieu dans chaque salle.
- On peut exporter les données des événements du calendrier en tant que fichier d'évènements *iCalendar,* utilisable par des applications de visualisation d'emploi du temps telles que *TimeCalendar*[1]

Fonctionnalités manquantes:

- Le site web affiche la disponibilité de chaque salle individuellement, mais il n'est pas possible de filtrer les salles par salles disponibles. Il faut donc consulter chaque salle une par une et garder en mémoire uniquement les salles disponibles durant le créneau souhaité.

[1] TimeCalendar est une application permettant aux étudiants d'importer leur emploi du temps depuis ADE pour le consulter sur mobile.

- Il n'est actuellement pas possible de réserver des espaces de travail depuis le logiciel utilisé à l'ESIEE
- On ne peut accéder au logiciel uniquement par un navigateur web d'ordinateur, le site n'étant pas optimisé pour appareils mobiles.
- On ne peut pas réserver de post-assistances.

On notera que dans les dernières versions du logiciel, une extension prenant la fome d'un site mobile est disponible et autorise la réservation de salles. Elle n'est pas adoptée par l'ESIEE, même pour l'année prochaine.

Celcat:

Celcat est un autre logiciel de gestion d'emploi du temps similaire à *ADE* pour la partie réservée à l'administration. Il intègre un système de réservation de salles.

2.1.2. Initiatives étudiantes

Certains étudiants de l'école ont développé des alternatives à l'utilisation d'*ADE* pour les étudiants, en exploitant les données du logiciel grâce aux liens d'export d'*ADE*.

Chatbot Discord:

Un premier projet prend la forme d'un *Chatbot Discord* censé afficher les salles disponibles, cependant celui-ci ne fonctionne pas correctement.

Site web étudiant:

Un site web créé par des étudiants de l'ESIEE [2] affiche les salles libres à l'instant où on le consulte. Ce site se rapproche le plus des fonctionnalités de notre application. On peut apprécier sa simplicité d'utilisation, de plus il ne requiert pas de connexion avec un compte. En revanche, on peut regretter l'absence de fonctionnalités de réservations de salles et de visualisation ou réservation de créneaux de *Post-Assistance* avec les professeurs. Par ailleurs, il n'y a pas de filtre de disponibilité par plage horaire: cela rend l'application dans les faits inutilisable entre 12h et 13h puisque la quasi-totalité

des salles sont affichées comme disponibles, bien que nombre d'entre elles redeviennent occupées à 13h.

2.1.3 Solutions commerciales

Certaines solutions logicielles proposent des logiciels et applications de réservation de salles et bureaux. Elles sont orientées davantage vers les bureaux que les universités. Voici un aperçu des plus pertinentes:

Roomzilla est un site web et une application mobile destinée à la réservation de salles, bureaux et ressources, telles des places de parking. Les propriétaires des locaux peuvent installer des afficheurs dans les salles pour indiquer leur disponibilité et permettre aux usagers de confirmer leur présence pour éviter les réservations non honorées. L'application semble trop générique et orientée vers la réservation et non la consultation rapide des salles disponibles.

Matrix Booking fonctionne de manière similaire à *Roomzilla*, et propose en complément des capteurs de présence.

Skedda est une solution comprenant un site web et une application mobile pour gérer les occupations des espaces de travail. On compte parmi les fonctionnalités proposées des autorisations de réservation différentes par profil d'utilisateur, une intégration avec des outils de collaboration tels que *Slack*, une carte interactive et la définition de conditions de réservation par espace, par exemple autoriser la réservation d'un espace à partir de 24h à l'avance.

Ces trois solutions ne placent pas la fonctionnalité de visualisation des salles disponibles au centre de leur application. Notre solution est donc plus adaptée à l'environnement de l'ESIEE.

2.1.4 Conclusion de l'analyse

L'analyse des solutions de gestion d'emploi du temps disponibles révèle des lacunes dans la consultation et la réservation de salles disponibles. En effet, utiliser l'outil

officiel *ADE* est chronophage et est en pratique rarement utilisé. L'extension développée spécifiquement par les étudiants pour l'ESIEE fonctionne mais manque quelques fonctionnalités rendant son utilisation plus délicate. D'autre part, les solutions commerciales de réservation d'espaces sont adaptées à un contexte professionnel et semblent conséquentes à mettre en place et à maintenir compte tenu de leur apport.
Notre application propose donc un équilibre entre l'offre sophistiquée des logiciels de réservation, et la simplicité des solutions développées par les étudiants de l'ESIEE.

2.2 Outils techniques et de gestion de projet

Outils de gestion de projet

Différents outils sont utilisés pour l'organisation de l'équipe, la conception et le développement de l'application.

- ***Monday*** : planification des tâches et visualisation de leurs échéances, priorités, avancement et personnes assignées à chaque tâche. Cet outil permet la création de diagrammes de Gantt.
- ***Google Docs*** : rédaction collaborative des livrables, spécifications techniques et comptes rendus des réunions.
- ***GitHub*** : gestion de version du code source et collaboration sur le développement.

Outils de conception

- ***Figma*** : création des maquettes de l'application, conception de la charte graphique et du logo.
- ***Miro*** : élaboration du diagramme de flux de l'application (flowchart).

Outils de développement

- ***Flutter*** : il s'agit d'un framework[2] de développement d'applications mobiles et web.
- ***Firebase*** : back-end serverless[3] (authentification, base de données, Cloud Functions, stockage).
- ***Android Studio*** : environnement de développement (IDE) utilisé pour coder et tester l'application sur des émulateurs Android.

2.3 Organisation du travail

Nous mettons en place des réunions hebdomadaires afin de faire le point sur l'avancement de notre projet, définir des objectifs et définir les tâches à prioriser pour la semaine suivante. Chaque réunion de l'équipe est complétée d'un rendez-vous avec notre suiveur pour rester sur la bonne voie. Afin de rester efficaces durant chaque journée, nous mettons en commun tous les matins nos réalisations du jour précédent et fixons les objectifs journaliers.
La répartition des tâches se fait en fonction des intérêts et des compétences de chacun. Nous décidons de travailler sur la maquette et le diagramme de flux (*flowchart*) en parallèle de la mise en place de l'application. Pour le démarrage du projet, Vadim et Nicolas travaillent sur le front-end pendant que Kévin et Melchior travaillent sur le back-end.

[2] En programmation informatique, un framework (appelé aussi infrastructure logicielle, infrastructure de développement, environnement de développement, socle d'applications, cadre d'applications ou cadriciel) est un ensemble cohérent de composants logiciels structurels qui sert à créer les fondations ainsi que les grandes lignes de tout ou partie d'un logiciel, c'est-à-dire une architecture. [3]

[3] Le serverless computing, ou informatique sans serveur, consiste à laisser le fournisseur de services Cloud [...] fournir, gérer et dimensionner l'infrastructure nécessaire pour faire tourner vos applications. [4]

3. Travail réalisé

3.1. Conception préliminaire

3.1.1 Réalisation d'une maquette

Nous réalisons une maquette via le logiciel *Figma*, afin de valider l'ergonomie de l'application et de connaître les différents éléments de l'application avant de passer à l'étape de la programmation. Cette étape est essentielle car tous les membres de l'équipe doivent partager une vision commune, par ailleurs il est difficile d'itérer rapidement sur le design de l'application via le code de l'application au lieu d'une maquette. Nous accordons une attention particulière à l'intuitivité de la navigation et à la conservation d'un design cohérent sur toutes les pages de l'application.

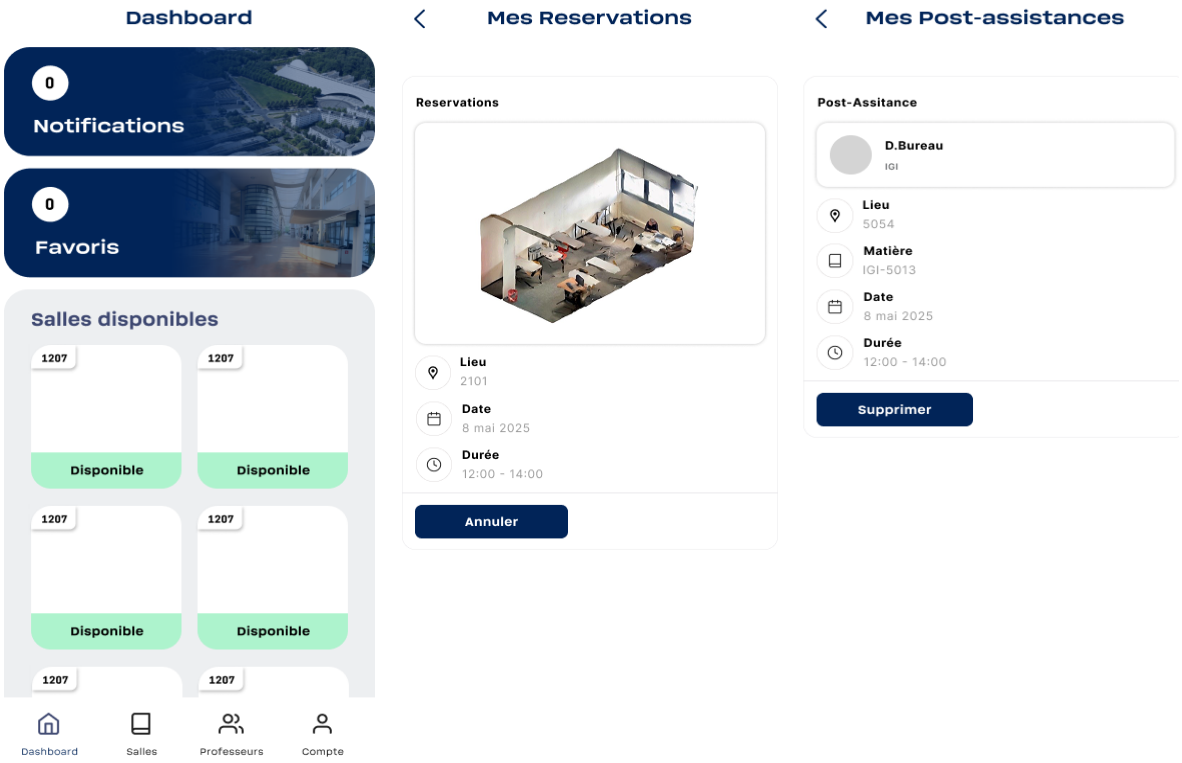


Figure 1: Aperçu des principaux écrans de l'application provenant de la première version de la maquette : de gauche à droite : Accueil, Réservations et Post-Assistances

Afin d'uniformiser le design sur l'entièreté de l'application, nous réalisons une charte graphique composée d'un logo, une typographie et d'une palette de couleurs. Celle-ci est disponible en annexe. Pour la police de l'application, nous choisissons *Lexend*.

L'application possède deux interfaces distinctes, attribuées en fonction du rôle de l'utilisateur (étudiant ou professeur). Pour les étudiants, elle est composée d'une page d'accueil, une page affichant les salles, une page affichant les professeurs et une page de gestion de compte. On peut appuyer sur les informations d'une salle - affichées sur ce qu'on appelle une *Room Card* - pour visualiser les occupations de la salle pour la journée et réserver un créneau d'occupation de la salle. Les disponibilités des professeurs sont affichées sur leur page dédiée. On peut enregistrer les créneaux d'assistance proposés par les professeurs.
L'interface des professeurs comporte une fonctionnalité d'ajout de créneau d'assistance. Le professeur peut préciser le lieu, la date et la matière de la session de Post-Assistance. Les professeurs peuvent comme les étudiants consulter les disponibilités des salles.

3.1.2 Diagramme de flux (*flowchart*)

Nous concevons avec le logiciel *Miro* un diagramme de flux pour préciser tous les parcours possibles des utilisateurs. Il fait le lien entre les actions et les différents états de l'application. De cette manière, nous n'oublions pas de fonctionnalités dont la définition est essentielle pour la programmation de l'application et la conception du back-end.

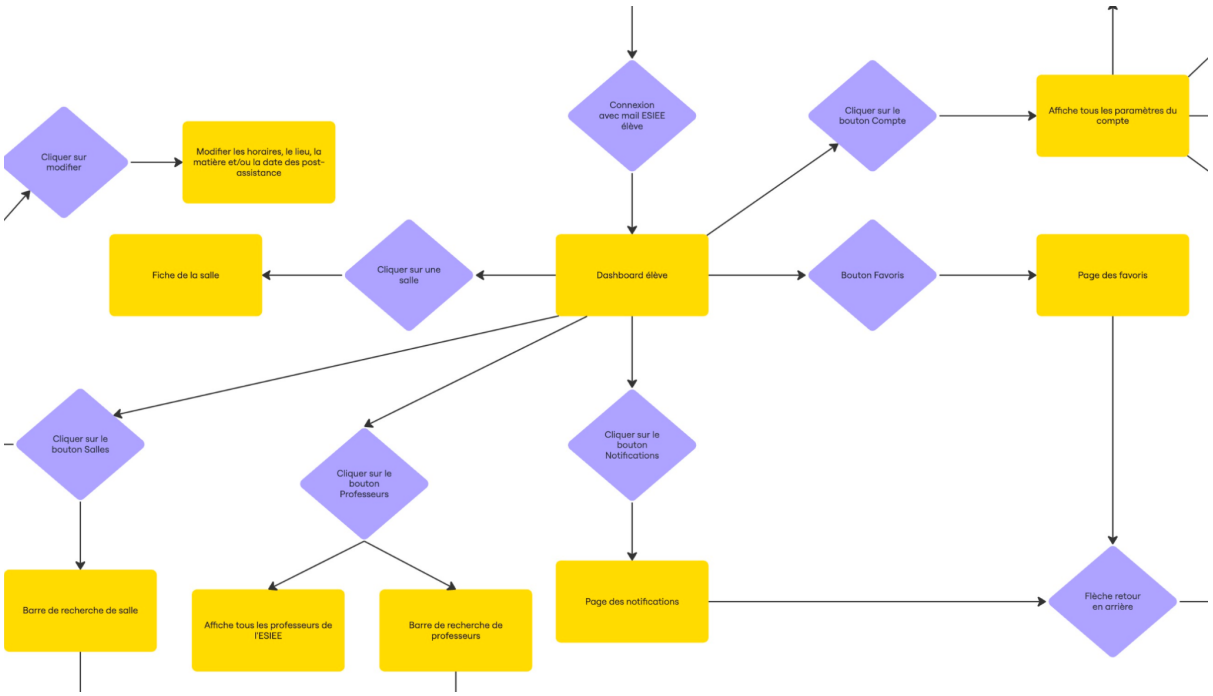


Figure 2: Extrait du flowchart. Les rectangles représentent les états de l'application et les losanges représentent les actions de l'utilisateur.

3.2. Développement front-end

3.2.1. Organisation du front-end

Le développement front-end se fait avec *Flutter*. Ce choix s'explique par sa capacité à générer des interfaces cohérentes sur plusieurs plateformes à partir d'un code unique en langage *Dart*.

L'architecture de l'application est réalisée de manière modulaire: elle isole la communication avec le back-end *Firebase*, l'affichage des éléments et la gestion du compte utilisateur.

L'organisation des fichiers suit la structure suivante :

- *Le caractère slash / indique un répertoire.*
- *Les sous-répertoires sont indiqués par un tiret.*

`Auth/` : gère l'authentification, l'affichage des écrans de connexion et de gestion de compte

`Backend/` : contient les modèles de données, les services de connexion à *Firebase* (authentification, base de données), ainsi que les fonctions liées à la réservation de salles, aux créneaux de Post-Assistance, et à la gestion des professeurs et des favoris.

`Frontend/` : regroupe les éléments liés à l'interface utilisateur et se divise selon les différents types d'utilisateurs (étudiant, enseignant, administrateur) :
- `admin/` : interface dédiée à l'administrateur, pour la gestion et la supervision globale.
- `student/` : vues et widgets[4] destinés aux étudiants, incluant les pages de compte, le tableau de bord, l'aide, la consultation des professeurs, et les éléments de navigation spécifiques.
- `teacher/` : ensemble des écrans et composants destinés aux enseignants, pour gérer leur compte, leurs créneaux de Post-Assistance et leurs interactions avec les étudiants.
- `common/` : composants partagés entre les différents profils, tels que les widgets d'affichage des salles et les écrans communs aux pages professeur et étudiant.

Concernant la gestion des états, nous utilisons majoritairement des `StreamBuilder` ou `FutureBuilder` pour l'affichage de données venant de *Firebase*. La mise à jour de l'interface se déclenche automatiquement dès que les données sont disponibles pour `FutureBuilder`, ou mises à jour côté back-end pour `StreamBuilder`.

3.2.2. Interaction avec le back-end

Nous utilisons pour la version mobile la méthode `snapshots()` pour récupérer en temps réel depuis *Firestore* les données des salles (par exemple, leur disponibilité en temps réel).
Pour la version web, on charge une seule fois les données au chargement de l'application.

[4] un widget est un composant d'interface utilisateur

La récupération de données lors de requêtes plus complexes provenant par exemple du filtrage des salles par disponibilité sur un créneau s'effectue via des *Cloud Functions*. On appelle la fonction via un API et on reçoit directement les informations des salles.

3.3. Développement back-end

3.3.1 Présentation du back-end

Le back-end de l'application repose sur le service *Firebase* de Google. Il s'agit d'une abstraction de services *Google Cloud* destinés au développement rapide d'applications et de logiciels. *Firebase* fournit un mécanisme d'authentification des utilisateurs, de base de données, de stockage de données (telles que des photos) et de *Cloud Functions*[5].

3.3.2. Authentification

Les utilisateurs d'*UniSpace* sont authentifiés via leur compte Google fourni par l'ESIEE. Les utilisateurs ont accès à une interface déterminée par leur statut dans l'école. Les étudiants, identifiés par leur adresse email se terminant par @edu.esiee.fr, voient l'interface étudiant, tandis que les professeurs, possédant une adresse en @esiee.fr, accèdent à l'interface professeur. L'application détermine quelle interface afficher aux utilisateurs d'après le domaine de leur adresse email.

La logique de gestion de l'authentification est gérée dans différents fichiers de code *Dart*, ainsi que par le biais de deux Cloud Functions:
Aperçu du code front-end pour l'authentification, par fichiers principaux:

- `auth_provider.dart` stocke les informations de l'utilisateur et les attributs du jeton de connexion (claims). Le rôle de l'utilisateur (étudiant, professeur ou administrateur) est stocké dans les claims du token *Firebase*, récupéré le cas

[5] Les Cloud Functions permettent d'exécuter du code sans avoir à gérer de serveur. Le code est appelé en réponse à des évènements de l'application ou d'autres Cloud Functions, des requêtes HTTPS, ou périodiquement selon une programmation personnalisée.

échéant et est récupéré sinon dans la base de données Firestore dans le champ rôle dans le document correspondant à l'utilisateur.

- `login_page.dart` affiche la page de connexion et fait le lien avec le service *Sign In With Google* en gérant les erreurs. L'application récupère les jetons d'accès Google et les envoie à *Firebase*. Une fois l'utilisateur connecté, le token *Firebase Cloud Messaging*, utilisé pour les notifications, est sauvegardé dans la base de données Firestore.
- `auth_gate.dart` affiche la page correspondante au rôle de l'utilisateur.

3.3.3. modélisation des données

Nous stockons les données de l'application via le service de Google *Firestore*.
Le service de base de données *Firestore* ne nécessite pas - et ne permet pas - de définir un schéma de base de données à l'avance.
Contrairement à une base de données relationnelle où les données sont organisées sous formes de tables, on retrouve des documents situés dans des collections.
Chaque collection, à l'instar d'un répertoire dans un système de fichiers, contient des documents associés à un identifiant. Un document contient des paires clé-valeur (champs) et peut éventuellement contenir d'autres collections.
Concernant la convention d'écriture pour la structure de données, nous précisons d'abord le chemin d'accès au document. Les documents sont encadrés entre accolades, les collections sont écrites telles quelles. Par exemple,
`/collection1/{document1}/collection2/{document2}`
Ici document1 est situé dans la collection1 et ainsi de suite.
Ensuite, nous précisions les noms des champs des documents ainsi que leur type.
Par exemple, `champ1 : int`

Exemple de documents dans la collection `users/` :
Document utilisateur principal :
Ce document contient les informations générales sur un utilisateur identifié par son uid.

```
/users/{uid}:
      displayName :          string
      email       :          string
```

Les collections, documents et leurs champs sont créés dynamiquement, lors de l'écriture des données. Cela a pour avantage principal de pouvoir rajouter des fonctionnalités facilement sans devoir modifier toute la structure de données.
Toutefois, il est nécessaire de déterminer une convention de structure de données avant de stocker des données dans la base, car cela permet de garder un maximum de cohérence dans les données stockées. En effet, deux personnes travaillant sur des fonctionnalités similaires pourraient écrire dans des documents différents, situés dans des collections différentes. Par ailleurs, il est nécessaire de savoir sous quelle forme les données sont stockées lors de la lecture de documents dans l'application.
Nous révisons la structure de données prévue initialement afin que les identifiants des utilisateurs ne soient pas accessibles aux autres utilisateurs. Pour cela, nous utilisons une collection `teachers_public/` contenant les informations publiquement accessibles des professeurs, ainsi qu'une collection `teachers_private/` faisant le lien entre l'identifiant utilisateur d'un professeur et son document de `teachers_public/`.

Voici un aperçu général de la structure de données :
Les informations de réservation et de cours de chaque salle sont stockées dans la collection `events/` du document correspondant à la salle. Les détails des réservations sont stockés dans la collection `reservations/`.
Afin d'accéder rapidement au statut d'occupation d'une salle en temps réel, on met à jour le champ `currentAvailability` de chaque salle. Sans cette optimisation, à chaque requête visant à récupérer une liste de salles, tous les évènements du jour courant devraient être récupérés. Nous économisons de cette manière des lectures dans la base de données.

La structure de données complète de l'application est en annexe.

3.3.4. Cloud Functions

L'architecture de l'application est conçue de telle sorte qu'il n'y a pas de serveur géré par nos soins ou tournant en permanence. Pour gérer la logique de l'application, on utilise des Cloud Functions, ce sont des fonctions JavaScript s'exécutant lors d'évènements dans le projet *Firebase* - tels que la mise à jour d'un document ou l'authentification d'un utilisateur - ou bien lors de requêtes HTTPS. La particularité de ces fonctions est que leur "réveil" (initialisation après une période d'inactivité) et la montée en charge est gérée automatiquement par Google.
Nous utilisons différentes fonctions pour la logique de notre code. En voici un aperçu, organisé par fonctionnalités :

3.3.4.1. Fonctions de gestion de la disponibilité des salles :

`roomStatusOnEventWrite` est déclenchée lorsqu'un document d'événement (cours ou réservation) est créé, modifié ou supprimé. Juste avant la fin de son exécution, elle appelle la fonction `roomAvailability` pour mettre à jour l'information de disponibilité en temps réel.
`createTask` est appelée à chaque création ou modification d'événement : elle planifie une tâche (*Cloud Task*) à l'heure de début et une autre à l'heure de fin pour calculer de nouveau l'indicateur de disponibilité de la salle correspondante avec `roomAvailability`.

3.3.4.2. Gestion des réservations :

`createRoomReservation` et `onReservationCancelled` permettent respectivement de créer et de supprimer une réservation.

`scheduleEndReservationTask` crée automatiquement à la création d'une réservation une Cloud Task à la qui se déclenche à la fin de la réservation pour passer

le statut d'une réservation de confirmed à finished via un appel à endReservationStatus.

3.3.4.3. Gestion des créneaux d'assistance :

Les fonctions `addOfficeHour`, `deleteOfficeHour`, et `updateOfficeHour` ajoutent, suppriment et mettent à jour un créneau de Post-Assistance dans teachers_public.

La fonction `getOfficeHour` récupère tous les créneaux d'un enseignant.

3.3.4.4. Lien avec les données de l'école :

La fonction `syncCalendar` récupère le fichier des évènements des salles de l'école en format iCal. Elle ajoute les événements du calendrier ADE dans la collection `events/` des salles dans lesquelles ont lieu chaque cours. Elle est programmée pour se mettre à jour tous les jours à 7h et à 12h30.

La fonction `importRoomCsv` récupère un fichier csv contenant les informations de chaque salle. Elle ajoute les salles et leurs informations à la base de données. Par exemple, si la salle contient des ordinateurs, le champ *hasComputers* du fichier csv vaut *true* et le champ du même nom du document de la salle est mis à true. Cela nous permet de contrôler les informations des salles affichées dans l'application. Nous pouvons ainsi exclure certaines salles de l'application, en particulier les salles inaccessibles ou la salle "Classe Virtuelle", où certains cours ont lieu, qui serait affichée dans l'application sans ce système.

3.3.4.5. Envoi de notifications

L'application comporte un service optionnel de notifications, activé par défaut et désactivable via les paramètres de l'application.

On envoie ces notifications en utilisant le service *Firebase Cloud Messaging* afin de prévenir les utilisateurs qu'un créneau de post-assistance est disponible. On envoie également des rappels aux utilisateurs 10 minutes avant le début d'un créneau de réservation afin d'éviter les réservations non honorées.
Pour cela, à chaque création d'une réservation, `scheduleReservationReminder` planifie une *Cloud Task* pour appeler `sendReservationReminder` 10 minutes avant le début de la réservation.

3.3.5. Règles de sécurité

Les requêtes étant envoyées directement depuis l'application vers *Firebase*, sans passer par un serveur intermédiaire, il est nécessaire de déterminer les règles d'accès à la base de données. En fonction des propriétés de l'utilisateur à l'origine de chaque requête, on restreint la lecture ou la modification de certains documents. Ces règles sont définies dans un fichier nommé `firestore.rules`, que nous téléversons via la console *Firebase*.

3.3.6. Stockage d'objets

Les fichiers tels que les images associées aux salles sont stockés grâce à *Firebase Cloud Storage*. Contrairement à *Firestore*, ce service de stockage ne gère pas des données structurées, mais des fichiers bruts.

3.4. Développement de fonctionnalités combinant front-end et back-end

3.4.1. Favoris

Nous implémentons un système de favoris: chaque salle et chaque professeur peut être mis en favori par les utilisateurs, et ainsi retrouver depuis l'écran d'accueil les créneaux de Post-Assistance des professeurs mis en favori, ainsi que accéder à la disponibilité

des salles favorites. En effet, environ une centaine de salles sont indiquées dans l'application, on évite ainsi de faire défiler les salles trop longtemps ou d'effectuer des recherches.
Les favoris sont stockés dans le back-end pour pouvoir se synchroniser entre différentes connexions.

3.4.2. Restrictions d'usage

Afin d'éviter les abus, nous instaurons une limite de réservation de 4 heures en une seule fois et de 16 heures par semaine. Pour cela, lors de la création d'une réservation, on regroupe les différents créneaux de la réservation et on bloque la demande de réservation si la durée de réservation dépasse 4 heures. De plus, chaque utilisateur possède un compteur `TimeLimit` (en tant que champ dans la base de données) correspondant au nombre d'heures réservables restantes par semaine. Ce compteur est décrémenté à chaque réservation.

Par ailleurs, il est uniquement possible de réserver de 7h00 à 20h00 afin d'éviter les réservations effectuées pendant la nuit pour la journée.

Enfin, une restriction est appliquée côté back-end afin de garantir qu'un utilisateur ne puisse réserver qu'une seule salle par créneau horaire.

3.4.3 Filtrage des salles

Sur la page d'affichage des salles, il est possible de filtrer par type de salle (par exemple uniquement les amphithéâtres), ainsi qu'en fonction de la localisation des salles, et par créneau horaire. Cette dernière fonctionnalité est essentielle pour anticiper les disponibilités. Par exemple, entre 12h et 13h la quasi-totalité des salles sont marquées comme libres alors qu'un grand nombre d'entre elles deviennent occupées à 13h. On évite ainsi de devoir consulter la fiche de chaque salle.

3.5 Interface utilisateur

Dans cette section nous exposons les différents écrans principaux de l'application. Le mode d'emploi de l'application, extrait du guide officiel disponible dans la page d'aide de cette dernière, est disponible en annexe.

3.5.1 Interface pour les étudiants

Nous exposons les différents écrans de l'application pour les étudiants.

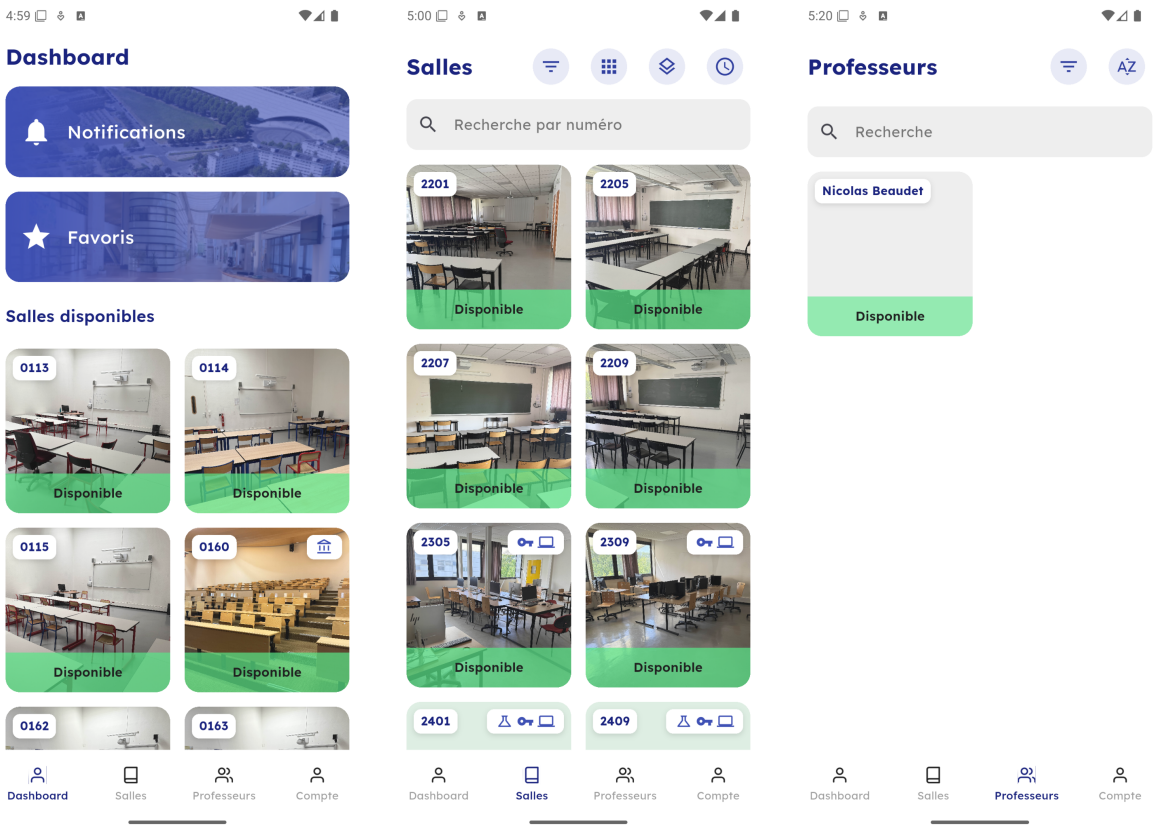


Figure 3:Pages principales de l'application pour les comptes étudiants. De gauche à droite, Écran d'accueil après connexion, page Salles et page Professeurs

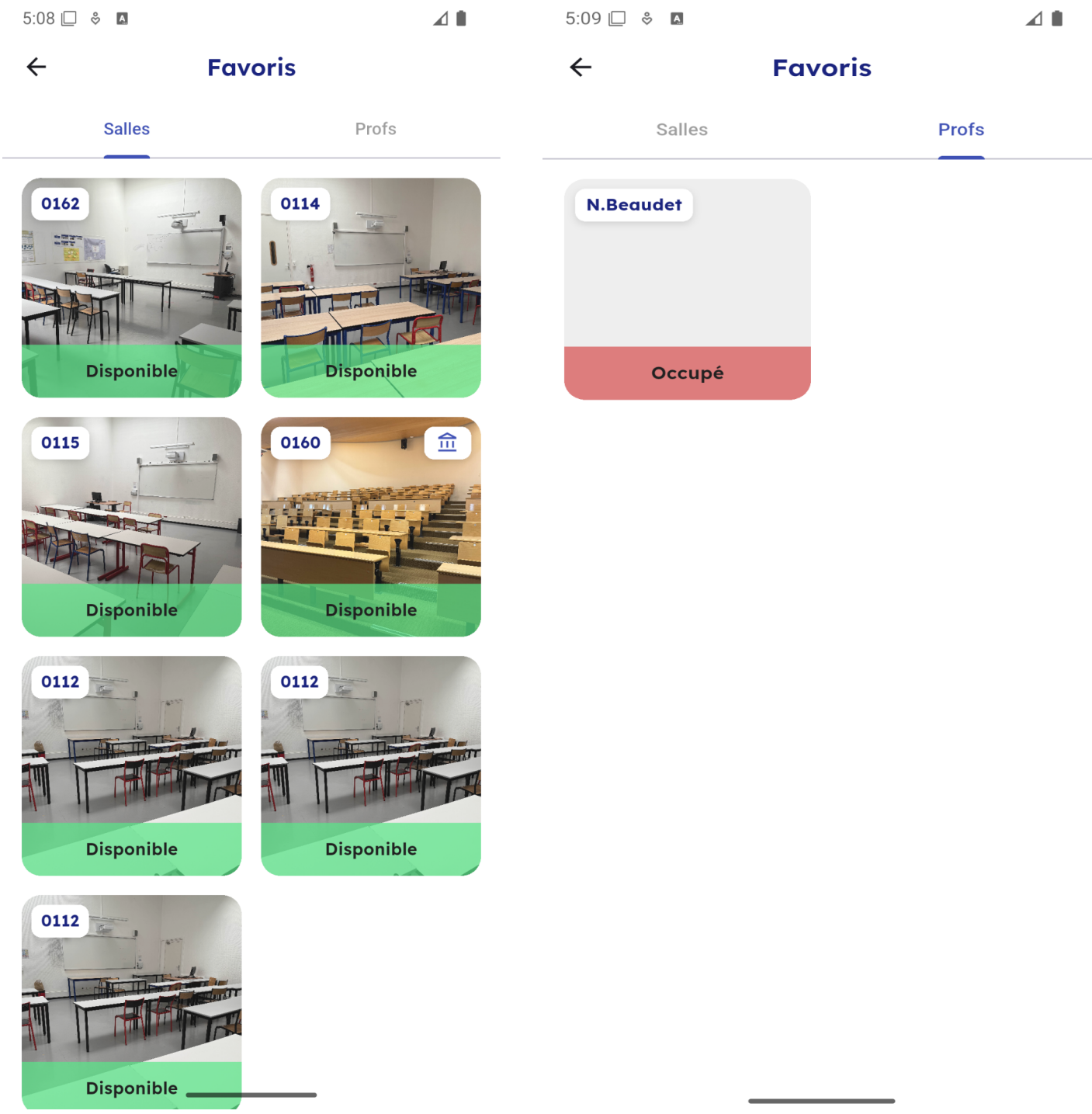


Figure 4: Aperçu de l'onglet « Favoris » de l'interface étudiants (depuis l'accueil)

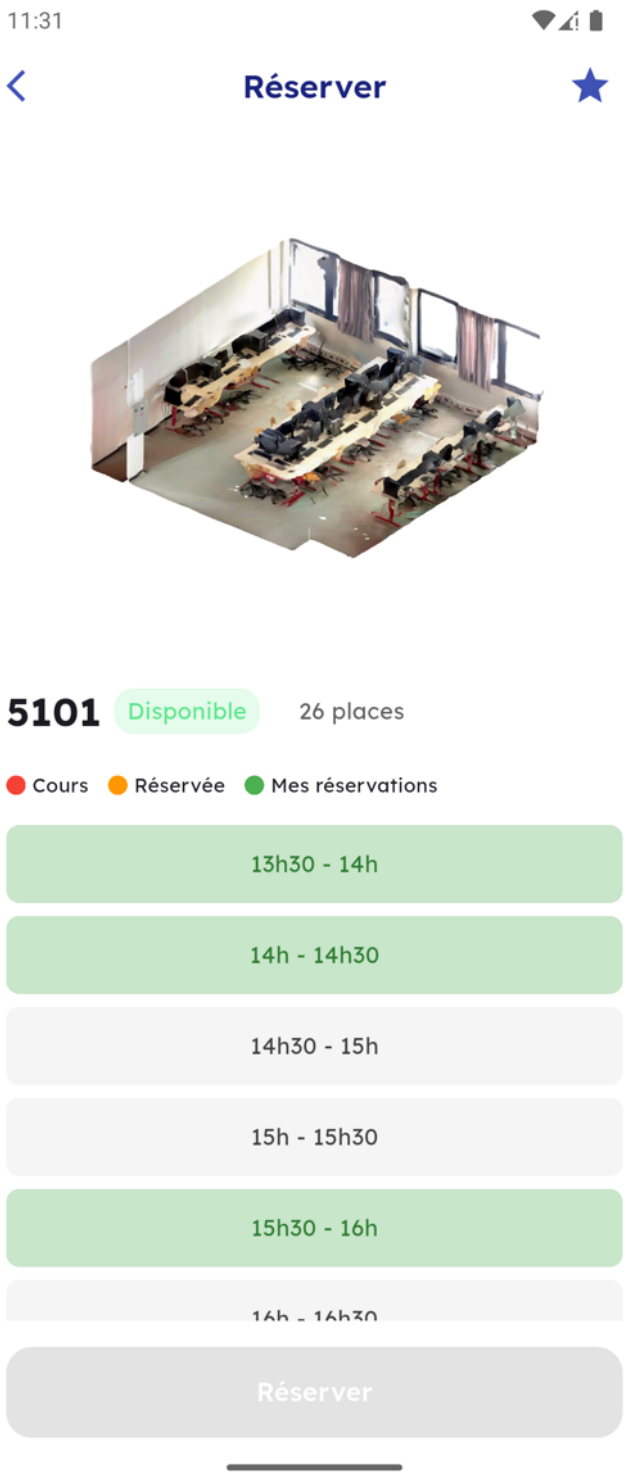


Figure 5:Ecran de consultation de l'occupation d'une salle et de réservation de créneaux d'occupation

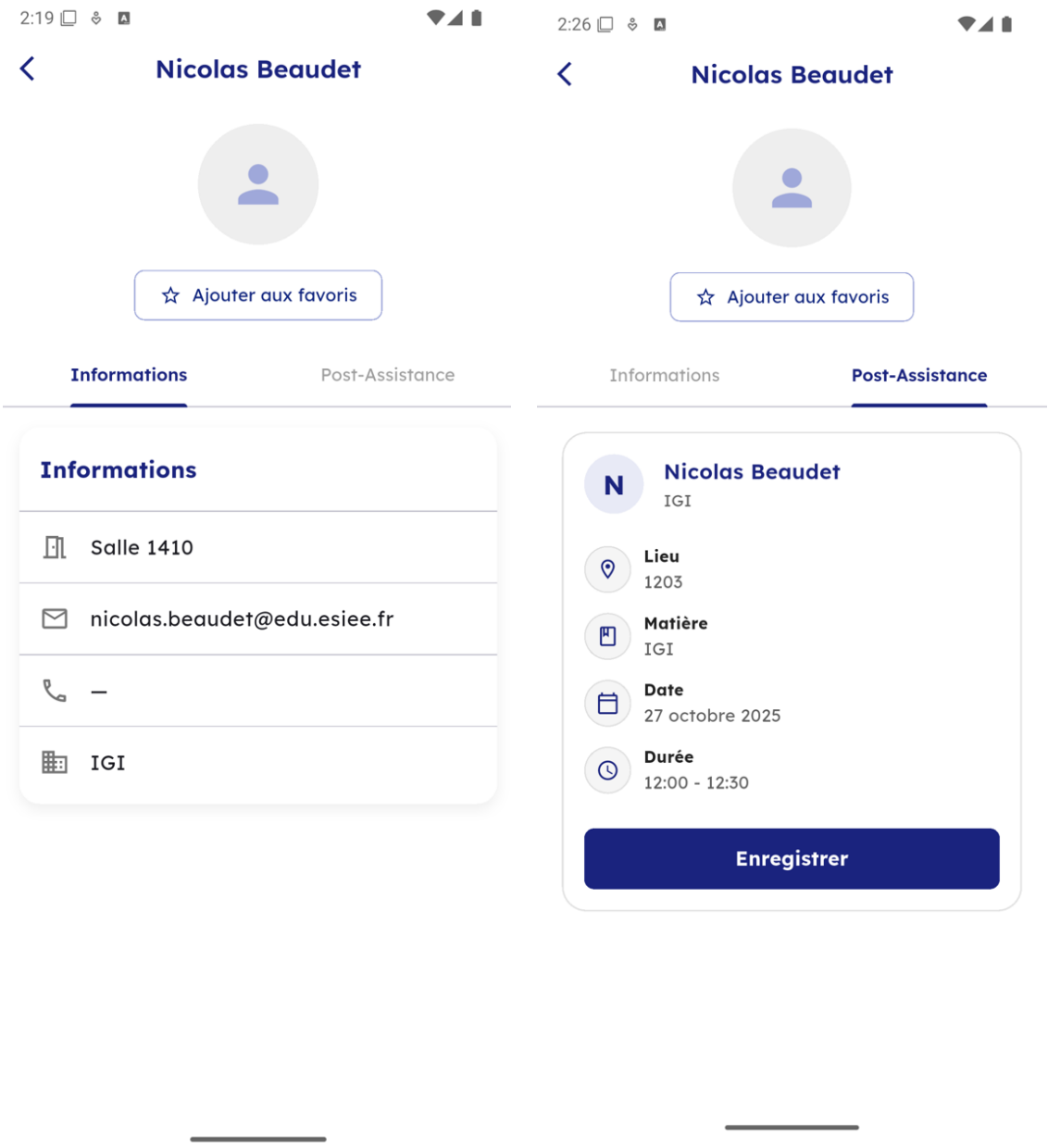


Figure 6: Écrans d'informations sur un professeur nommé « Nicolas Beaudet »

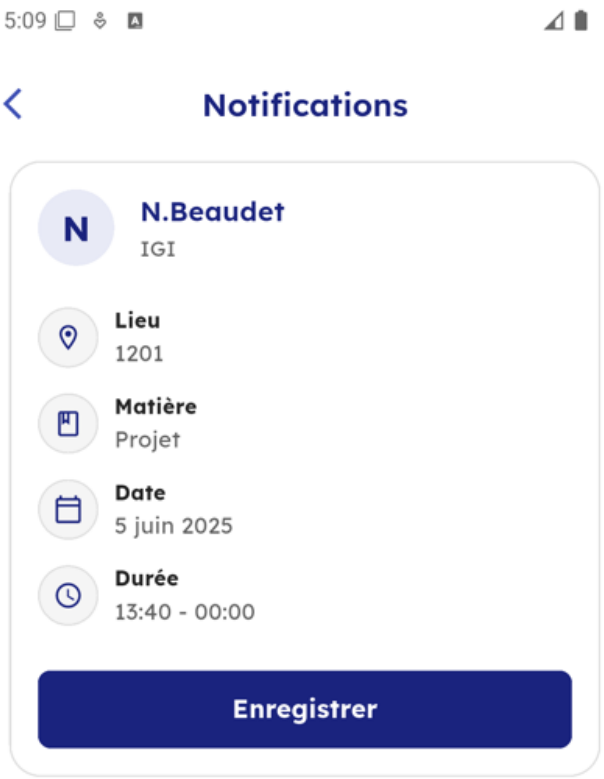
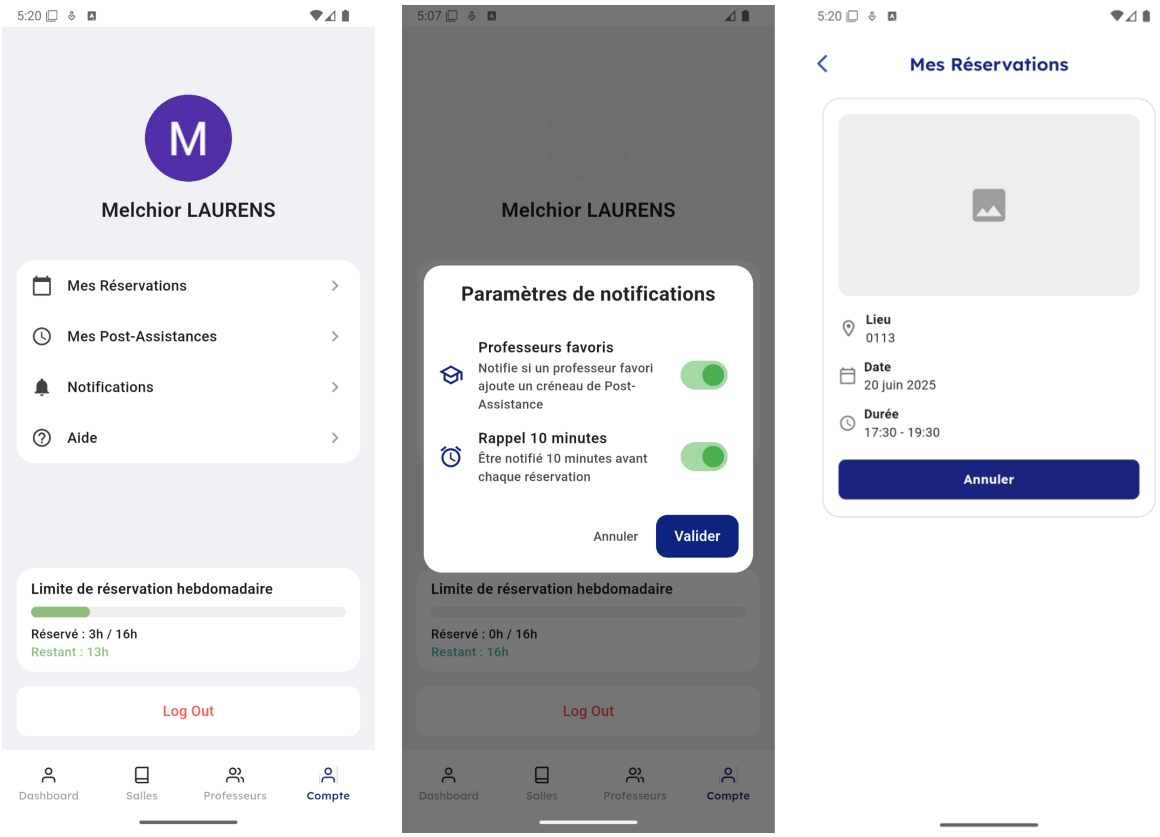


Figure 7: Aperçu de l'onglet « Notifications » de l'interface étudiants (depuis l'accueil)



(a) (b) (c)

Figure 8: (a) Page "Compte", (b) Paramètres des notifications et (c) Onglet « Mes Réservations » (depuis la page Compte)

3.5. Interface pour les professeurs

Nous présentons les principaux écrans disponibles pour les professeurs. Les professeurs peuvent réserver des salles de la même manière que les étudiants. Ils peuvent entrer leurs informations et gérer leurs créneaux de post-assistance.

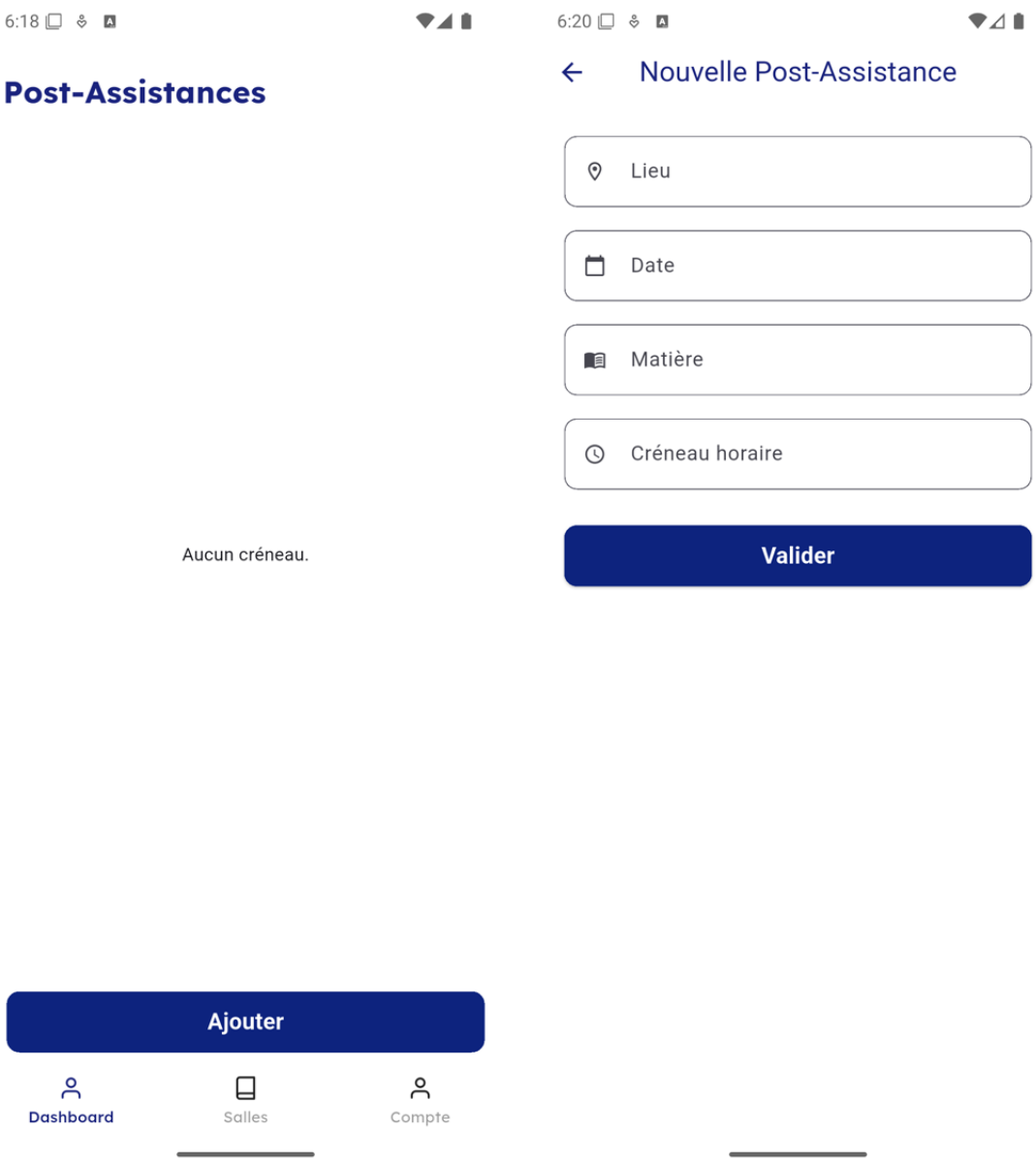


Figure 9: Gestion des créneaux de post-assistance : (a) liste des créneaux d'assistance, (b) ajout d'un créneau

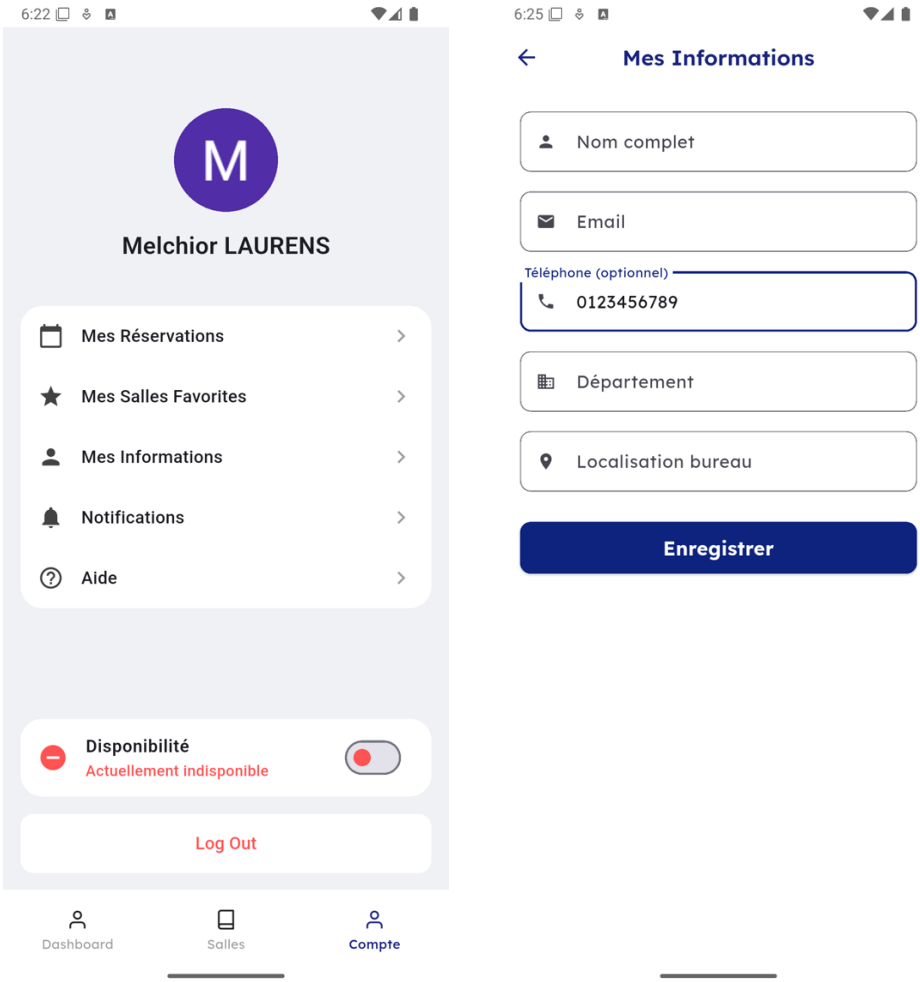


Figure 10: (a) Écran de gestion du compte professeur, (b) gestion des informations affichées sur le profil du professeur

4. Retour d'expérience

4.1 Compétences techniques

Tout au long du projet, l'équipe a fait l'acquisition d'un large éventail d'outils et de savoir-faire. Nous avons découvert le langage Dart et le SDK Flutter pour le développement mobile, en s'appuyant sur Android Studio comme IDE principal et, ponctuellement, sur Xcode pour la compilation iOS. La mise en place de l'environnement s'est concrétisée par la création d'un projet Flutter de base, la

configuration des dépendances dans `pubspec.yaml`, puis la connexion de l'application à une instance *Firebase* dans laquelle Authentication, Firestore, Cloud Functions et Cloud Tasks ont été activés. En parallèle, nous avons appris à formaliser un guide de style sous Figma, regroupant couleurs, polices et composants de base avant d'implémenter ces choix visuels dans les écrans destinés aux étudiants : tableau de bord, notifications, favoris et profil. De plus, nous avons bâti des widgets capables de recevoir dynamiquement le contenu de Firestore, de manière à éviter tout refactor massif par la suite, ce que nous ne savions pas faire avant. Sur la partie back-end, une convention de structuration NoSQL a été rédigée ; des règles d'accès (`firestore.rules`) ont été définies afin d'accorder ou de restreindre la lecture comme l'écriture selon le rôle de chaque utilisateur, et plusieurs Cloud Functions ont été développées en JavaScript. Elles assurent la mise à jour du statut des salles, la gestion complète du cycle de vie des réservations, ainsi que l'import automatique des calendriers ADE/Planif grâce à la bibliothèque node-ical. Enfin, des tests ont été menés sur émulateurs puis sur terminaux physiques Android et iOS pour vérifier l'authentification Google, la bonne synchronisation des données et la solidité de la configuration iOS, qu'il a parfois fallu corriger manuellement.

4.2 Compétences humaines

Sur le plan organisationnel, nous avons appris à réaménager l'espace de travail via l'outil Monday : mise à jour des tableaux, création de nouvelles colonnes, définition de statuts plus explicites et mise en place de revues hebdomadaires ont amélioré la visibilité de l'avancement et la répartition des responsabilités. La collaboration s'est illustrée par la validation collective de la charte graphique, le pair-programming lors du développement Flutter et la co-conception de la base de données. Chacun a cultivé un apprentissage continu en consultant tutoriels, documentations officielles et articles spécialisés pour monter en compétence sur Dart, Flutter, Firestore ou encore les Cloud Functions, puis en partageant ces découvertes avec le reste de l'équipe afin de réduire le temps d'appropriation global. Cette démarche a favorisé l'adaptabilité : qu'il s'agisse de remanier la structure NoSQL pour se conformer au RGPD ou de réparer une configuration iOS défectueuse, les ajustements se sont faits rapidement, sans dériver

du planning. Sur le volet éthique, un rapport dédié a recensé les données sensibles, décrit les risques et détaillé les mesures de protection, notamment la dissimulation des identifiants uniques et la séparation des documents publics et privés.

4.3 Difficultés rencontrées

L'apprentissage simultané de Dart, de Flutter et d'Android Studio a d'abord ralenti la progression, le temps de comprendre la logique du langage, le système de compilation et la gestion des émulateurs. La conception d'interfaces dynamiques avant même l'arrivée des données réelles a exigé une réflexion plus poussée pour éviter, plus tard, la réécriture des écrans. Le respect du RGPD s'est révélé particulièrement exigeant : l'exposition potentielle des UID a imposé une refonte de la structure de la base et le renforcement des règles de sécurité. L'actualisation en temps réel du statut des salles a, elle aussi, posé problème : les appels successifs à Firestore menaçaient d'épuiser le quota quotidien, si bien qu'il a fallu introduire des Cloud Tasks pour planifier les mises à jour de manière ciblée. Enfin, un blocage de compilation sur iOS a interrompu temporairement le déploiement ; nous avons dû auditer les fichiers Xcode, corriger les *entitlements* et rétablir les profils de signature avant de poursuivre. À chaque souci rencontré, l'équipe s'est organisée pour trouver une solution, par des mémos internes ou des relectures de code. Cela a permis de fiabiliser le travail et renforcer la cohésion du groupe.

5. Conclusion

Au terme des deux mois impartis, l'équipe a atteint les objectifs fixés : une application mobile (Android, iOS) et web opérationnelle affiche désormais, en quasi-temps réel, la disponibilité des salles d'ESIEE Paris et les créneaux de post-assistance des enseignants.

Sur le plan pédagogique, le projet a permis de mettre en pratique les notions de gestion de projet vues en cours : découpage des tâches dans Monday, tenue d'un planning, revues de code régulières et documentation partagée. Chacun a consolidé ses

compétences en développement mobile, en conception d'interface et en déploiement de services cloud, tout en apprenant à travailler dans un cadre collectif.

Plusieurs pistes d'amélioration ont déjà été identifiées : automatiser l'import des emplois du temps à une fréquence plus fine, proposer un mode hors ligne pour les consultations rapides, et déployer l'application à d'autres services du campus. Ainsi, le projet poursuit son objectif de répondre à un besoin concret de la communauté étudiante.

Annexe

Figure 11: Palette de couleurs de l'application

Figure 12: Logo de l'application

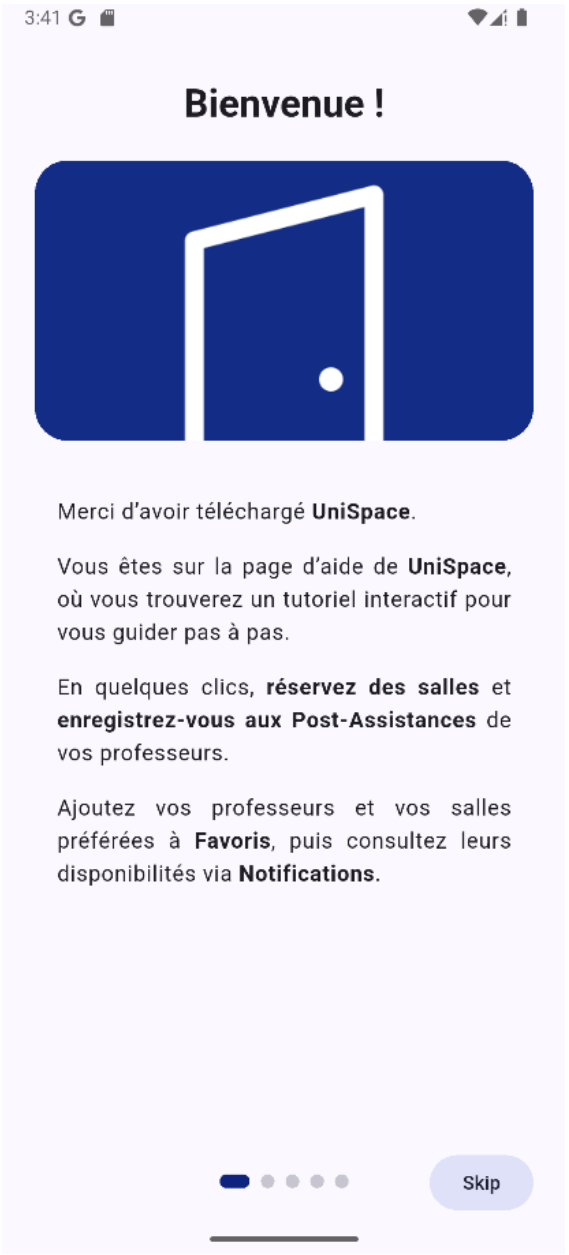
Convention de la structure de données Firestore:

```
# -------------------------  UTILISATEURS  -------------------------
/users/{uid}
  displayName                string
  email                      string
  role                       "student" | "teacher" | "admin"
  createdAt                  timestamp
  NotifPostA                 boolean
  NotifReservation           boolean
  fcmToken                   string
  fcmTokenUpdatedAt          string
  TimeLimit                  int

/users/{uid}/favorites/{favId}
  targetId                string        # roomId  ou  teacherPublicId
  targetType                 "room" | "teacher"
  name                       string          # dénormalisé
  createdAt                  timestamp

/users/{uid}/saved_assistances/{savedId}
  professorId                 string               #   pointe   vers
/teachers_public/{publicId}
  professorName              string
  slotId                     string     # ID du créneau
  startTime                  timestamp
  endTime                    timestamp
  subject                    string
  location                   string
  department                 string
  createdAt                  timestamp

/rooms/{roomId}
  name                       string
  capacity                   number
```

```
  currentAvailability        "available" | "reserved" | "occupied"
  isAmphi                    boolean
  isLab                      boolean
  hasComputers               boolean
  requiresKey                boolean

# -------------------------  SALLES  -------------------------
/rooms/{roomId}/events/{eventId}                    # cours (ICS) ou
réservations élèves
  startTime                  timestamp
  endTime                    timestamp
  type                       "course" | "reservation"
  status                     "confirmed" | "cancelled"
  importedFromICS            boolean          # true si VEVENT ADE
  description                string
  startTask                  string
  endTask                    string
  importedFromICS            boolean


/reservations/{reservationId}                # réservations *privées*
de salles
  eventId                    string
  roomId                     string
  userId                     string           # UID de l'étudiant
  startTime                  timestamp
  endTime                    timestamp
  status                     "confirmed" | "cancelled"
  createdAt                  timestamp
  endTaskName                string


# ----------------------  ENSEIGNANTS  ---------------------------
#Document PUBLIC lisible par tous (aucun UID n'apparaît)
```

```
/teachers_public/{publicId}                    # publicId ≠ uid
  displayName                 string
  officeLocation              string
  currentAvailability         "available" | "busy" | "unknown"
  contactEmail                string
  contactTelephoneNumber      string
  department                  string?

/teachers_public/{publicId}/office_hours/{slotId}
  startTime                   timestamp
  endTime                     timestamp
  place                       string
  subject                     string


#Document PRIVÉ indexé par l'UID (non lisible des étudiants)
/teachers_private/{uid}
  publicId                     string            # lien vers son doc
public
```

Manuel d’utilisation (extrait de la page d’aide de l’application) :



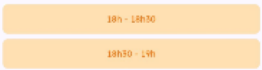
3:43

Réserver une salle

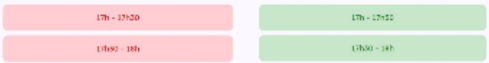
Pour réserver une salle, il vous suffit de trouver la salle qui vous correspond et de cliquer sur **sa vignette**.

Vignette de la salle 1201

Une fois la salle sélectionnée, vous avez un code de **3 couleurs** vous permettant de voir **l'état de la salle** tout au long de la journée.



Réservée



Occupée Mes créneaux

Pour réserver une salle, il suffit de cliquer sur les créneaux **libres** (qui passent alors en **bleu**) puis de cliquer sur **Réserver (nombre de créneaux)** pour **valider la sélection**.

Skip

3:45

Page Professeurs

Une fois avoir cliqué sur la vignette **professeurs**, une liste de **tous les professeurs** s'affiche. Vous pouvez **trier** tous ces professeurs par ordre **alphabétique** ou **anti-alphabétique** et les **filtrer** afin de ne voir que les profs qui sont disponibles.

Exemple d' une vignette d'un professeur

Vous avez aussi la possibilité de **chercher un prof** en tapant son **nom** et son **prénom** ou son **nom** dans la barre de recherche :

Barre de recherche

Une fois que vous avez appuyé sur la vignette du professeur, un onglet « Post-Assistance » apparaît. Dans cet onglet, vous pouvez **vous enregistrer** pour cette séance, mais **pas la réserver**.

Skip


3:42
Trier & filtrer
Filtrer
Trier
Lors de la recherche de salles, vous pouvez les **filtrer** pour afficher les salles **disponibles**.
Vous pouvez également utiliser le bouton **Trier** pour organiser les résultats par localisation : uniquement les salles situées **dans un épi**, celles **hors épis** ou bien **toutes les salles**.
Enfin, il est possible de **Filtrer** et de **Trier** simultanément pour trouver la salle qui correspond le plus à vos critères.
Skip

Bibliographie

[1] Emploi du temps de l'ESIEE [En ligne] Disponible : https://planif.esiee.fr/

[2] *Zeffut* et *Glz_SQL*, "Salles Disponibles", [En ligne] Disponible : https://esiee-paris.streamlit.app/

[3] Wikipedia, Framework, [En ligne] Disponible : https://fr.wikipedia.org/wiki/Framework (consulté le 20 juin 2025)

[4] Oracle, Définition de serverless computing - Qu'est-ce qu'un serverless computing ?, [En ligne] Disponible : https://www.oracle.com/fr/cloud/definition-serverless-computing/