# Application Mobile

ATELABSYSTEMURBAIN
Laboratoire de recherche en urbanisme

ROLANDDEHGHANKAMARAGI
2023

# Table des matières


**PRESENTATION DE LA STRUCTURE D'ACCUEIL DE L'ENTREPRISE** ........................ 3

LES ACTIVITES DU LABORATOIRE DE RECHERCHE ........................ 3

**REALISATIONS TECHNIQUES, FRONT-END ET BACK-END** ........................ 3

**OPTIMISATION ET MODELISATION D'UN SYSTEME VIA UML : UNE APPROCHE ORIENTEE UTILISATEUR** ........................ 5

**REPRESENTATION DES COUCHES** ........................ 8

**DICTIONNAIRE DES DONNEES, MCD, MPD** ........................ 9

**LA CONCEPTION D'INTERFACE : DE LA MAQUETTE AU WIREFLOW** ........................ 10

**MAQUETTE ET ZONING : VISUALISATION STATIQUE ET STRUCTURATION FONCTIONNELLE DE L'INTERFACE UTILISATEUR** ........................ 11

**WIREFRAME : SCHEMA SIMPLIFIE REPRESENTANT LA STRUCTURE ET L'ORGANISATION DE L'INTERFACE DE NOTRE APPLICATION** ........................ 12

**WIREFLOW : REPRESENTATION SCHEMATIQUE DES INTERACTIONS UTILISATEUR ET DES FLUX DE NAVIGATION AU SEIN DE L'INTERFACE DE NOTRE APPLICATION** ........................ 13

**LES REGLES DE GESTION** ........................ 14

**ÉCRAN DE CONNEXION ET D'INSCRIPTION FINALISES** ........................ 15

**REDUX ET ASYNCSTORAGE** ........................ 15

**LA BASE DE DONNEES** ........................ 16

MÉTHODE 1 : ........................ 16
METHODE 2 : ........................ 16
MÉTHODE 3 : ........................ 17
MÉTHODE 4 : ........................ 17
MÉTHODE 5 : ........................ 18

**COMMENT PROTEGER LES FORMULAIRES ?** ........................ 19

COMMENT SECURISER NOS FORMULAIRES ? ........................ 19
COMMENT CRYPTER LES DONNEES TRANSMISES ? ........................ 20

**GENERATING JWT** ........................ 21

**LES TESTS UNITAIRES** ........................ 22

## Présentation de la structure d'accueil de l'entreprise

ATELAB est une organisation de recherche, domiciliée au sein de l'école nationale supérieure d'architecture de Paris la Villette (ENSAPLV).

Avec plus de 2500 étudiants, ENSAPLV est la plus importante école d'architecture francophone au monde. Elle forme des architectes venus de tous les continents. Elle dispose aussi d'un réseau d'anciens élèves en Asie, en Afrique, sur tout le continent américain et évidemment en Europe.

### Les activités du laboratoire de recherche

- Formation initiale et formations spécialisées dans le domaine du projet urbain.
- Promotion de la culture architecturale et urbaine : colloques, séminaires.
- Recherches expérimentales.

## Réalisations techniques, front-end et back-end

Au départ, j'ai effectué une analyse pour déterminer les langages de programmation et les outils nécessaires pour mon projet. La liste incluait JAVA, JEE, JAVASCRIPT, SPRING HIBERNATE, MYSQL, REACT, REACT NATIVES, CSS3 et HTML5.

J'ai centralisé la mise en page et l'encapsulation des sous-composants autour du composant "View", en collaboration avec "flexbox" et JavaScript.

Pour personnaliser mes composants React Native, j'ai utilisé des "props" pour transférer des données, ainsi que des "props children" pour une personnalisation supplémentaire.

J'ai utilisé le hook "useState" pour gérer plusieurs états au sein d'un composant, ce qui m'a permis de stocker et d'afficher une chaîne de caractères dans un "Text".

Concernant la gestion des formulaires, j'ai utilisé le composant "TextInput" et des "props" comme "auto-correction", "multiline", "maxlength", "placeholder" pour améliorer l'expérience utilisateur. J'ai également associé des événements spécifiques, tels que "onChangeText", "onSubmitEditing", "onFocus", "onBlur", "onScroll" et "onEndEditing", à ces formulaires.

Pour le défilement, j'ai mis en place les composants "ScrollView" et "FlatList", qui permettent d'accéder verticalement à tous les composants. J'ai spécifiquement utilisé "FlatList" pour ne charger que les éléments visibles.

J'ai rendu certains composants, tels que "View" et "Text", interactifs en utilisant les composants "Touchable" et "Pressable". J'ai également utilisé le composant "Alert" pour afficher une série de boutons interactifs avec des événements "onPress".

J'ai mis en œuvre le composant "modal" pour créer des pop-ups et afficher du contenu par-dessus le reste, ce qui permet de fournir des informations utiles à l'utilisateur.

Pour le développement de mon application, j'ai utilisé plusieurs logiciels et frameworks, notamment Node.js, Java (version 17), DBeaver pour la gestion de la base de données MySQL, Eclipse comme IDE, VsCode pour React Native, Postman pour les tests, et Material UI pour le design.

Pour gérer les requêtes HTTP et les réponses, j'ai utilisé Axios dans React. J'ai également utilisé Async de JavaScript et JSON Placeholder pour une gestion asynchrone efficace.

Enfin, j'ai réalisé des opérations CRUD sur les données en utilisant les commandes "await.axios.get", ".post", ".put" et ".delete". En suivant les règles de gestion, j'ai pu mettre en place des boutons et des champs de formulaire, créant une interface utilisateur complète et fonctionnelle.

## Optimisation et Modélisation d'un Système via UML : Une Approche Orientée Utilisateur

Afin d'optimiser la structure interne de notre système, j'ai d'abord conceptualisé un modèle qui reflète l'état actuel du système, mettant en évidence les objets qui le composent et leurs relations. Ensuite, j'ai élaboré un modèle logique à partir de ce concept pour formaliser ces relations et les processus au sein du système.

Afin de représenter ce modèle de manière plus formelle, j'ai conçu un diagramme de classes UML. Ce diagramme a permis de clarifier la structure du système en identifiant les classes d'objets et leurs interdépendances. Il répond aux questions cruciales : quelles sont les classes d'objets constituant le système ? Quelles sont leurs dépendances ? Comment sont structurés les différents niveaux d'abstraction ?

En parallèle, j'ai pris en compte les divers types d'utilisateurs que notre système devait accueillir. Ceux-ci comprennent non seulement les étudiants en architecture, mais aussi les clients qui s'inscrivent en tant que membres, participent occasionnellement à des séminaires ou choisissent une formation en contrepartie de frais d'inscription.

Bien que les participants et les membres aient des rôles distincts, ils suivent une structure similaire. Chaque utilisateur dispose d'un identifiant (ID), d'un nom et d'un email. Certains participants s'inscrivent spécifiquement pour contribuer à un événement (séminaire, conférence...) et s'acquittent du paiement correspondant.

Après avoir précisément déterminé les classes d'objets à manipuler, j'ai pu structurer le système en reliant ces objets de manière appropriée. Les relations entre les classes d'objets du système sont aussi flexibles que les objets eux-mêmes, dépendant des spécificités que nous souhaitons représenter.

Ce modèle logique a ensuite servi de base pour l'implémentation de notre base de données. Cette approche a permis d'assurer une cohérence entre la conception du système et sa mise en œuvre pratique.

Pour illustrer les interactions entre les utilisateurs et le système, j'ai également développé un diagramme de cas d'utilisation. Ce diagramme représente un scénario où un client s'inscrit, ses coordonnées sont stockées dans la base de données, il consulte ensuite la description d'un produit (formation, séminaire, conférence...) avant de passer commande et de payer via une banque.

Cette méthodologie a permis de garantir une conception du système efficace et cohérente, tout en veillant à répondre aux besoins et attentes des utilisateurs. La flexibilité du système assure qu'il peut évoluer et s'adapter aux exigences changeantes des utilisateurs.

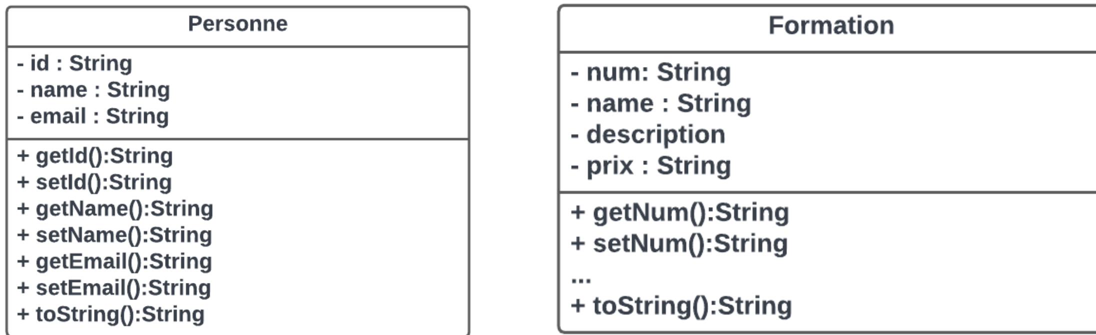



*Figure 1 : Diagramme de classe*

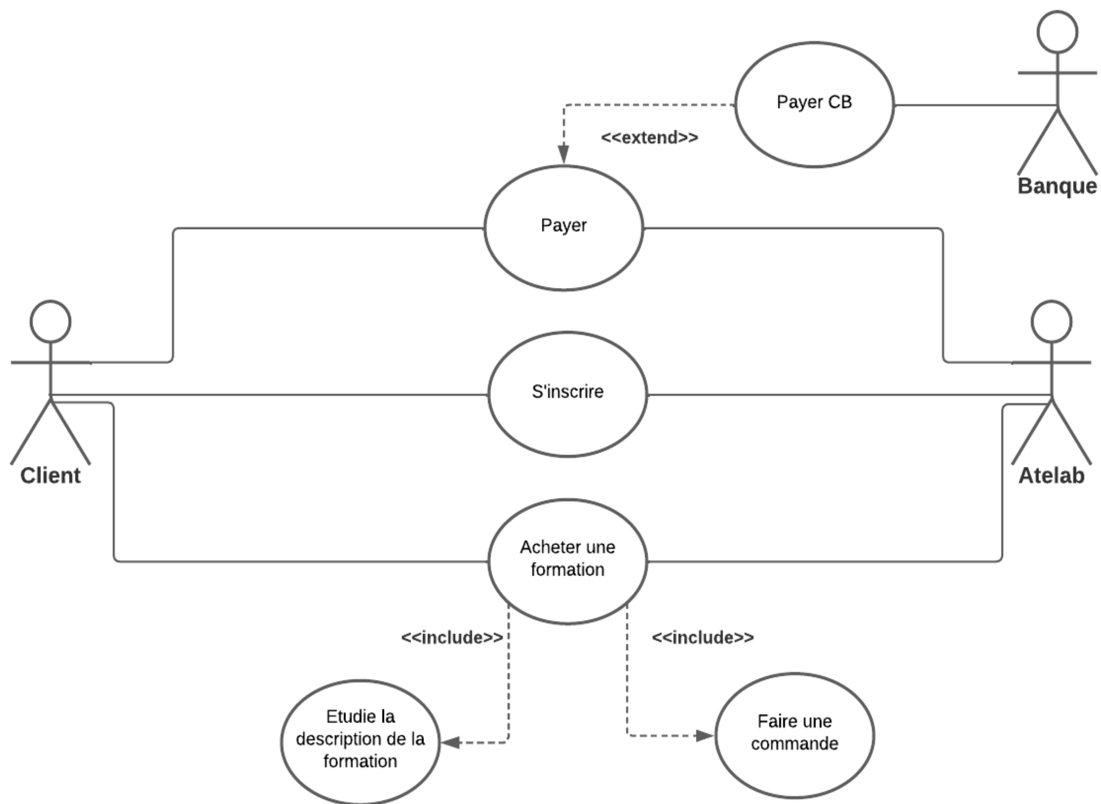



*Figure 2 : Diagramme de cas d'utilisation*

# Diagramme de séquence / inscription membre

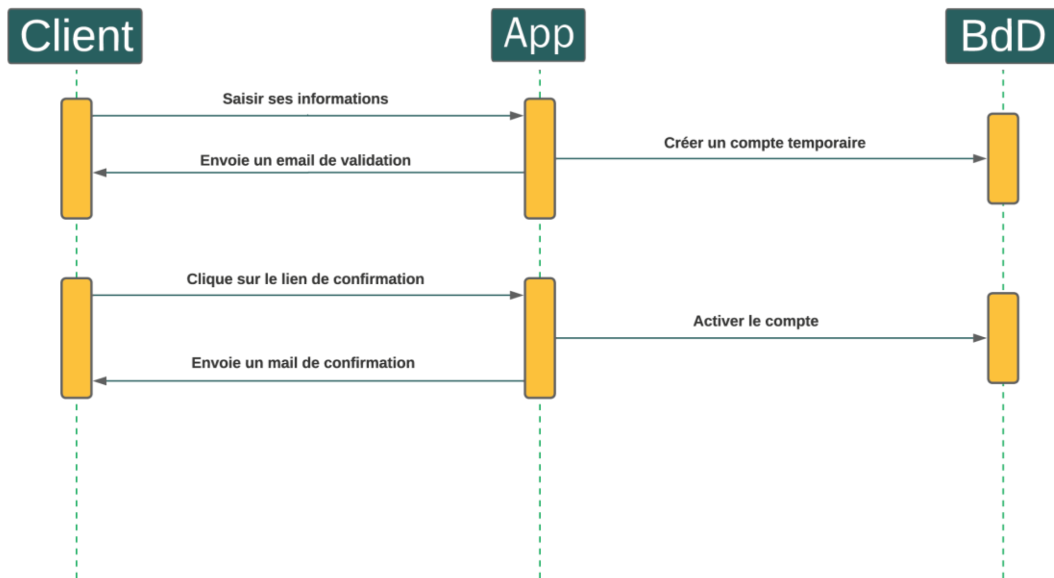



*Figure 3 : Digramme de séquence*

## Diagramme d'activité

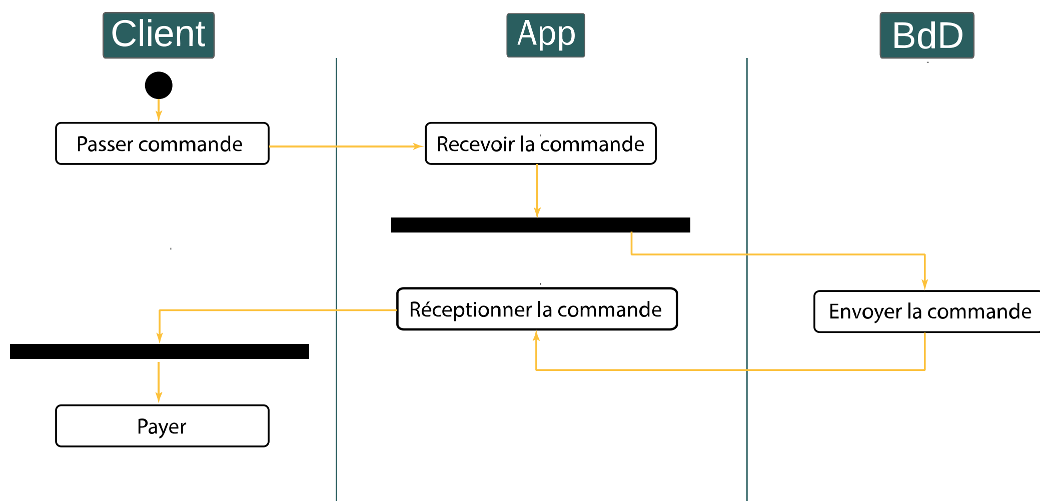



*Figure 4 : Diagramme d'activité*

## Représentation des couches

Pour optimiser efficacement mon application, j'ai utilisé l'architecture MVC (Modèle, Vue, Contrôleur). Cette stratégie divise l'application en trois parties essentielles :

Modèle : Cette couche gère les données, les règles d'affaires, l'interaction avec la base de données, et définit la structure des données. Chaque modèle est lié à une structure de données spécifique.

Vue : C'est la partie visible de l'application avec laquelle l'utilisateur interagit. Elle assure la présentation des données.

Contrôleur : Ce segment agit comme un lien entre le modèle et la vue, recevant les requêtes de l'utilisateur, les traitant, puis interagissant avec le modèle pour effectuer les actions requises.

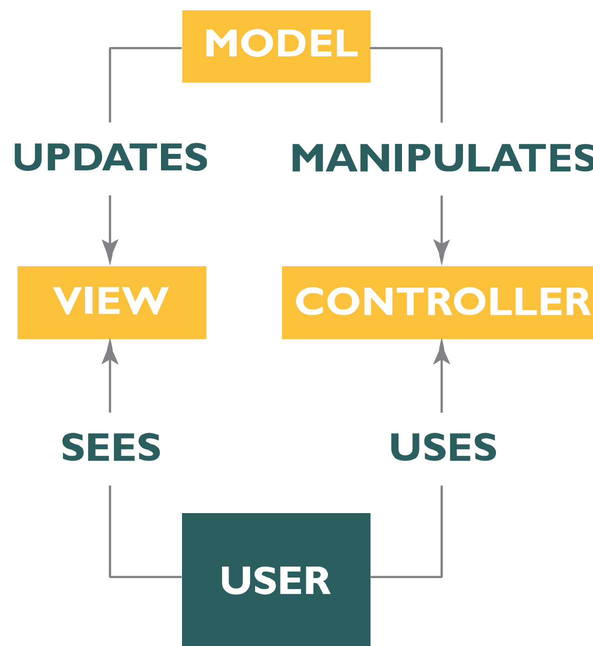



*Figure 5 : MVC/Modèle, Vue, Controller*

## Dictionnaire des données, MCD, MPD

**Dictionnaire des données :** J'ai créé un dictionnaire des données pour fournir une collection organisée de définitions des données utilisées dans le système. Par exemple :

Utilisateur

ID : Entier, Clé primaire, Auto-incrément

Nom : String, Maximum 50 caractères

Email : String, Maximum 100 caractères

**Modèle Conceptuel de Données (MCD) :** J'ai établi un Modèle Conceptuel de Données (MCD) qui a servi de représentation graphique des entités de données et des relations entre elles. Ce modèle a aidé à comprendre les structures de données complexes et leurs interconnexions.

**Modèle Physique de Données (MPD) :** J'ai mis en place un Modèle Physique de Données (MPD) pour représenter le schéma de la base de données physique. Ce schéma décrivait comment les données étaient stockées dans le système, y compris les tables, les clés, les indices, etc. Par exemple, la table "Utilisateur" était liée à la table "Evénement" par une clé étrangère, représentant les inscriptions des utilisateurs aux événements.

Ces trois éléments ont fourni une vue détaillée et complète des données dans le système, facilitant le développement et la maintenance de l'application.

## La Conception d'Interface : De la Maquette au Wireflow

Pour débuter la conception d'un site web ou d'une application, j'adopte généralement une approche progressive, en commençant par l'élaboration d'une maquette. Cette première phase me permet de représenter visuellement l'interface utilisateur en utilisant un logiciel de graphisme. Cela donne une idée du design final de l'interface, incluant les couleurs, les typographies, les images et la disposition générale des éléments.

Après la création de la maquette, je procède au zoning. Cette étape me permet de définir la disposition générale de la page web ou de l'application. J'y détermine où seront placées les différentes sections de l'interface, comme l'en-tête, le corps principal, les barres latérales, le pied de page, etc. C'est une étape cruciale car elle planifie la structure de l'information et la manière dont les utilisateurs navigueront sur le site ou l'application.

Une fois le zoning accompli, j'élabore le wireframe. Il s'agit d'un schéma détaillé qui illustre chaque page du site web ou de l'application, en montrant où chaque élément (comme les boutons, les images, les textes) sera placé. À ce stade, je ne m'attarde pas sur les détails esthétiques tels que les couleurs, la typographie et les images, je me concentre plutôt sur l'organisation de l'interface.

Enfin, j'entreprends la réalisation du wireflow. Cette étape est une combinaison d'un wireframe et d'un diagramme de flux qui illustre comment les utilisateurs navigueront à travers le site web ou l'application. Le wireflow montre le chemin qu'ils emprunteront de page en page et comment les différentes pages sont connectées entre elles.

En somme, adopter ces étapes dans cet ordre me permet de développer progressivement mon projet, en partant de l'apparence générale (maquette), en passant par la disposition des éléments (zoning et wireframe), pour finir par l'interaction utilisateur (wireflow).

## Maquette et Zoning : Visualisation Statique et Structuration Fonctionnelle de l'Interface Utilisateur

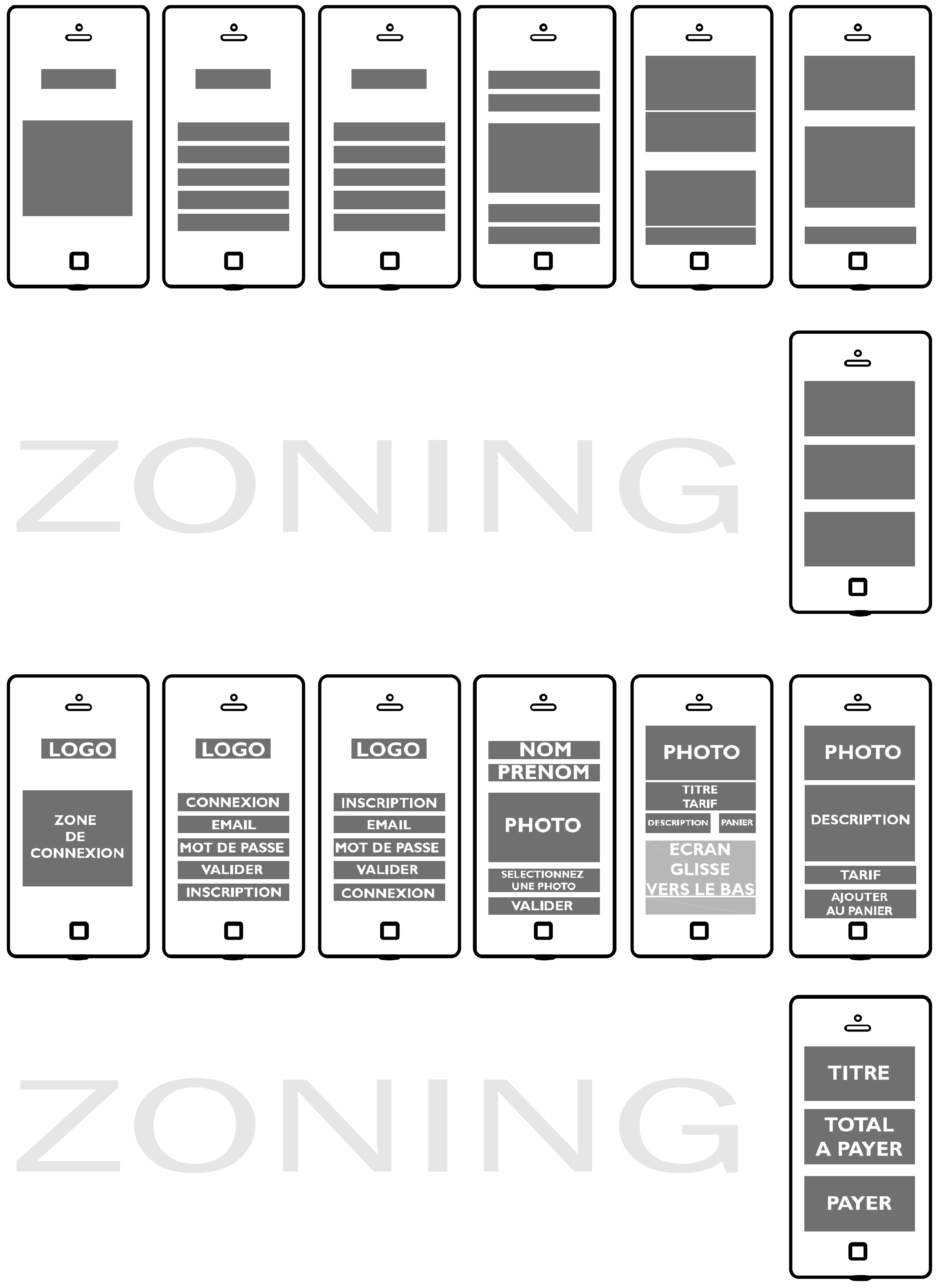



*Figure 6 : Le zoning*

## Wireframe : Schéma simplifié représentant la structure et l'organisation de l'interface de notre application[1]

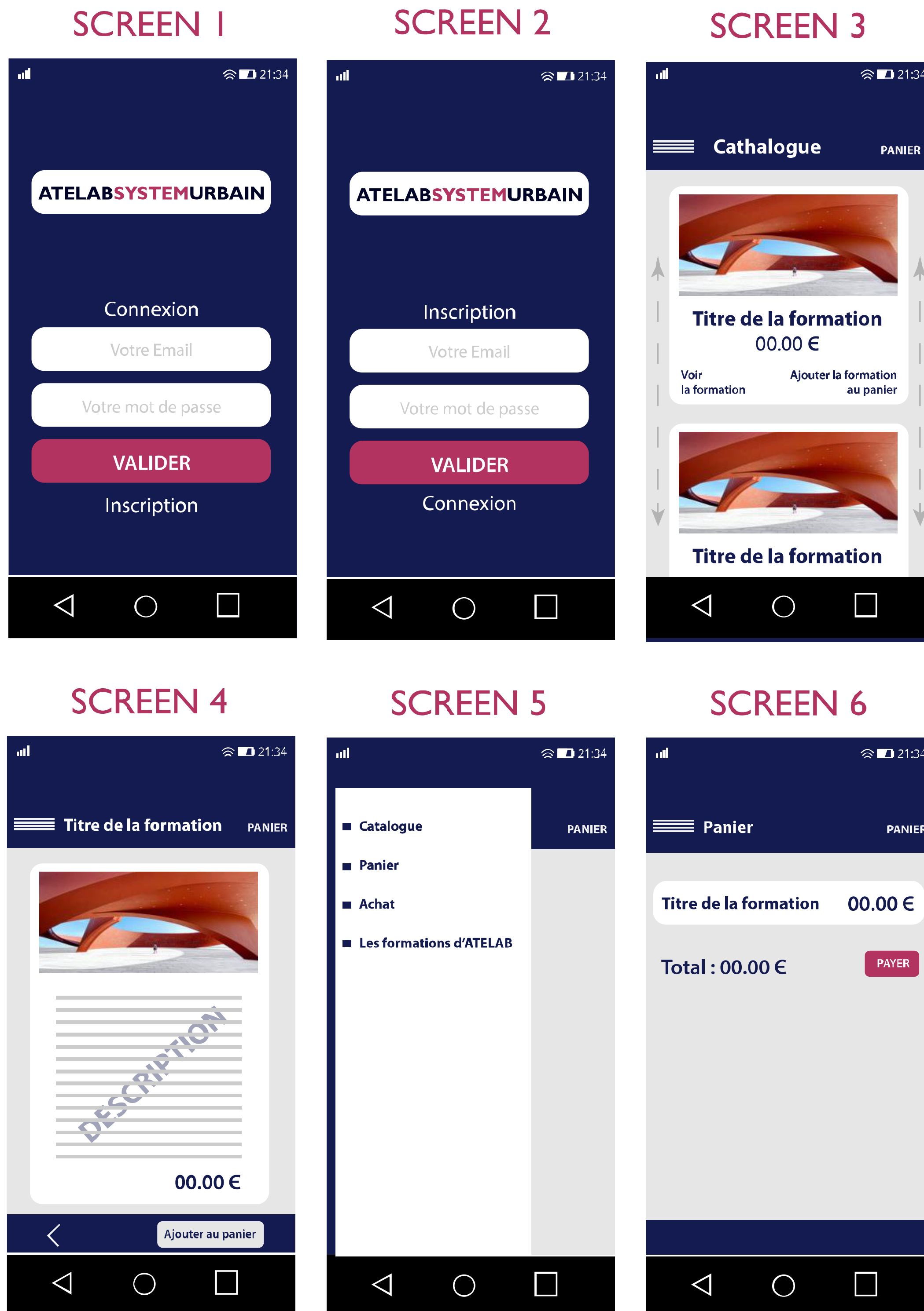



*Figure 7: Wireframe*

[1] Dans cette partie, toutes les images ne sont pas présentées…

## Wireflow : Représentation schématique des interactions utilisateur et des flux de navigation au sein de l'interface de notre application.[2]

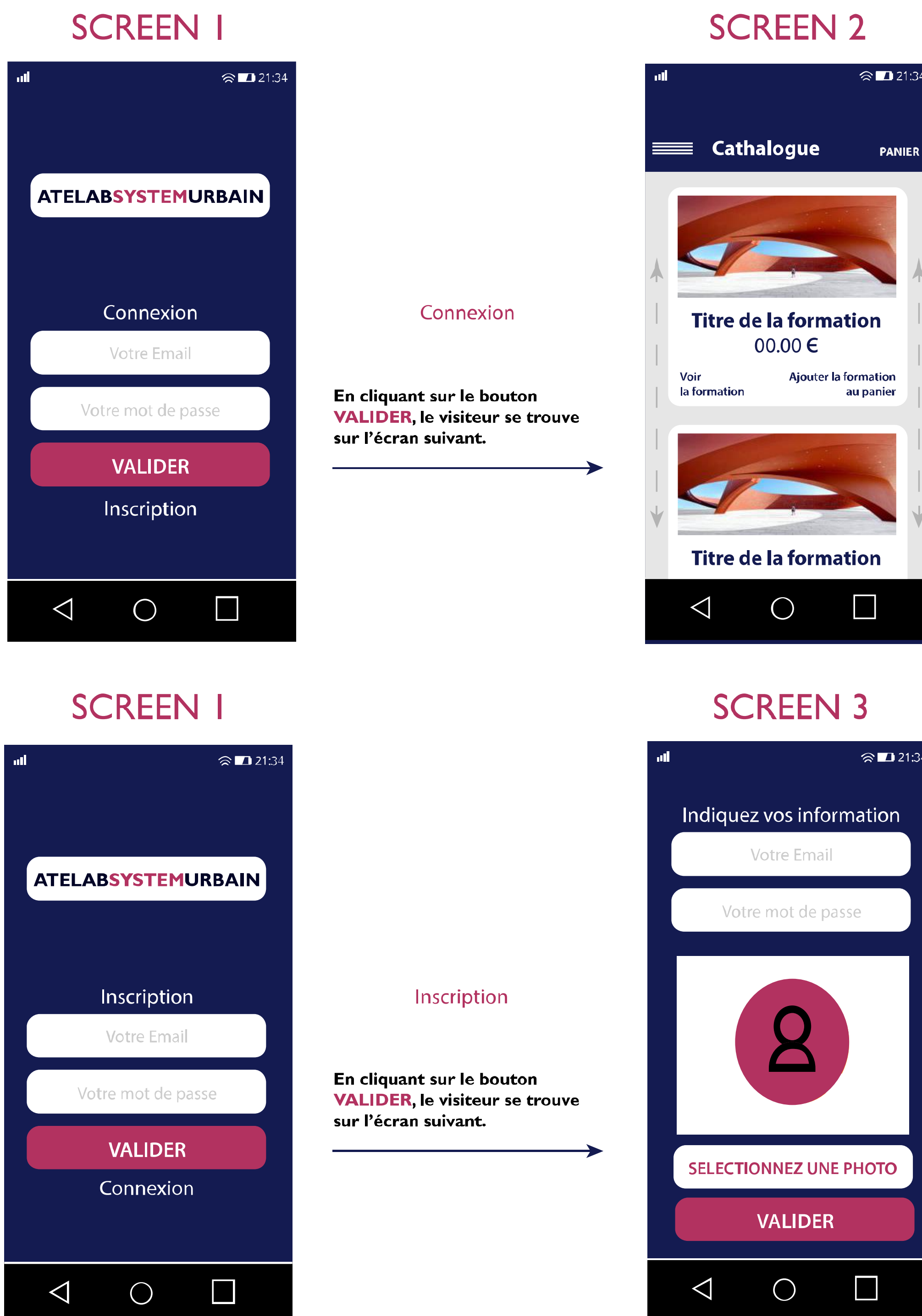



*Figure 8 : Wireflow*

---

[2] Dans cette partie, toutes les images ne sont pas présentées…

## Les règles de gestion

J'ai défini les règles de gestion qui régissent le fonctionnement de l'application.

**Voici quelques exemples de règles de gestion que j'ai pris en compte :**

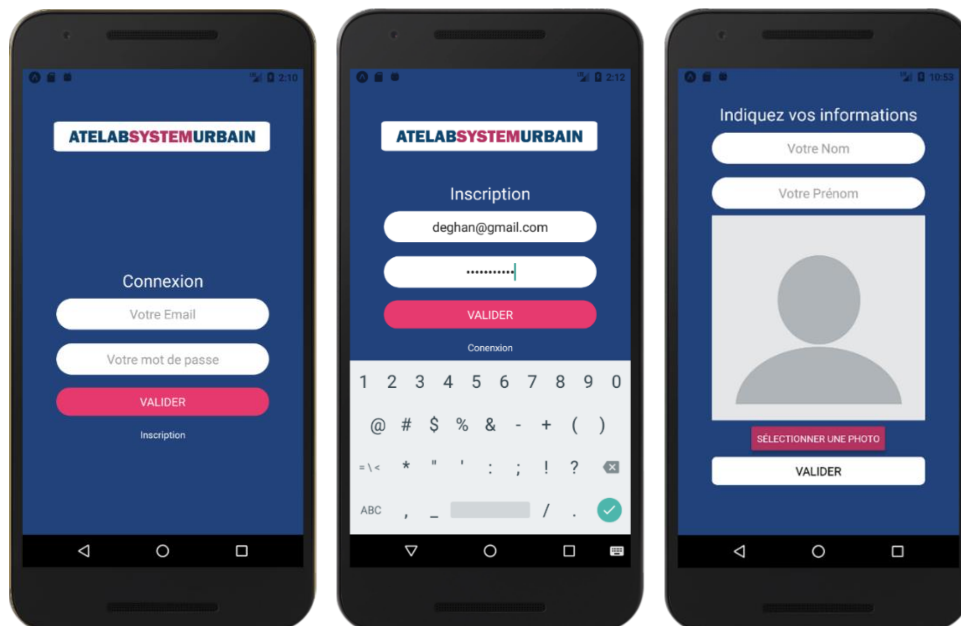
1. Les utilisateurs doivent s'inscrire en fournissant leur nom et leur adresse e-mail.
2. Chaque utilisateur peut participer à des événements tels que des séminaires, des conférences, etc.
3. Les utilisateurs peuvent effectuer des paiements pour les événements auxquels ils participent.
4. Chaque événement a un nom et une date associée.
5. Les utilisateurs peuvent consulter les descriptions des événements avant de prendre une décision de participation.
6. Les utilisateurs peuvent effectuer des recherches d'événements en fonction de certains critères tels que la date, le type d'événement, etc.
7. Les événements peuvent avoir une limite de participants, auquel cas les utilisateurs ne pourront s'inscrire que jusqu'à ce que cette limite soit atteinte.
8. Les informations des utilisateurs doivent être stockées de manière sécurisée et confidentielle.
9. Les paiements des utilisateurs doivent être traités de manière sécurisée et fiable.

Ces règles de gestion ont été prises en compte lors de la conception de l'application afin de garantir un fonctionnement cohérent et conforme aux besoins des utilisateurs. Elles ont guidé les décisions prises dans les différentes couches de l'application, du modèle de données à la logique de contrôle et à la présentation des informations à l'utilisateur.

Pour mieux visualiser notre structure, j'ai créé les tables et mis en place une base de données.

Sur le schéma suivant, nous pouvons constater que la base de données est divisée en deux grandes familles : « participant » et « intervenant », qui chacune va récolter les informations grâce aux jointures dans les tables adjacentes.

## Écran de connexion et d'inscription finalisés




## Redux et AsyncStorage

Redux est une bibliothèque JavaScript prédictible pour la gestion de l'état de l'application. Elle m'a permis de gérer l'état global de mon application de manière centralisée et cohérente. Les composants de l'application peuvent ainsi accéder à cet état global et se mettre à jour en fonction des changements. C'est particulièrement utile dans le cadre de l'authentification utilisateur, où l'état d'authentification doit être accessible à travers plusieurs composants.

Pour le stockage des données de session, j'ai utilisé AsyncStorage, une API de stockage clé-valeur asynchrone qui est globale à l'application. AsyncStorage m'a permis de stocker et de récupérer les informations d'authentification de l'utilisateur, comme le token JWT (JSON Web Token), pour les sessions persistantes. Cela signifie que même si l'application est fermée ou redémarrée, l'état d'authentification de l'utilisateur est conservé et peut être récupéré au redémarrage de l'application.

En sommes, pour la gestion des requêtes HTTP, Redux et AsyncStorage se sont révélés précieux. En utilisant Redux, j'ai été capable de gérer l'état des requêtes - par exemple, en indiquant si une requête est en cours, a réussi ou a échoué. Cela m'a permis d'offrir une meilleure expérience utilisateur en répondant de manière appropriée à l'état des requêtes.

Enfin, avec AsyncStorage, j'ai pu stocker les données récupérées à partir des requêtes HTTP pour un accès ultérieur. Cela a réduit la nécessité de requêtes HTTP répétées pour les mêmes données, améliorant ainsi la performance de l'application.

## La base de données

J'ai utilisé une combinaison de MySQL, Eclipse et DBeaver pour gérer ma base de données.

MySQL est le système de gestion de base de données que j'ai choisi. Il me permet d'exécuter des opérations CRUD (Création, Lecture, Mise à jour, Suppression) sur ma base de données.

Eclipse est l'Environnement de Développement Intégré (EDI) que j'ai utilisé pour le développement de mon application. Il offre une interface pratique pour la rédaction, le débogage et l'exécution de mon code. Eclipse a également des plugins qui facilitent le travail avec MySQL.

DBeaver est un outil de gestion de bases de données universel que j'ai utilisé pour visualiser, analyser et manipuler mes données. Il a une interface conviviale qui rend facile de naviguer à travers ma base de données MySQL et d'exécuter des requêtes SQL.

### Les méthodes utilisées dans mon Controller

**Méthode 1 :**

```
@PostMapping("/user")
    public UserDTO saveUser(final @RequestBody UserDTO user) {
            return
UserMapper.toDTO(UserService.saveUser(UserMapper.toEntity(user)));
```

Méthode pour enregistrer un utilisateur.

L'annotation @PostMapping est utilisée pour mapper les demandes HTTP POST sur des méthodes de gestionnaire spécifiques.

L'URL de point de terminaison est "/user".

Cette méthode prend un seul paramètre de type UserDTO, annoté avec @RequestBody, ce qui signifie qu'il est attendu dans le corps de la demande HTTP POST.

Elle retourne un objet de type UserDTO, qui est créé en convertissant d'abord l'objet UserDTO reçu en un objet Entity à l'aide de la méthode toEntity de la classe UserMapper, puis en le passant à la méthode saveUser de la classe UserService. Finalement, l'objet Entity retourné par saveUser est converti à nouveau en UserDTO à l'aide de la méthode toDTO de la classe UserMapper et retourné en tant que réponse.

**Méthode 2 :**

```
@GetMapping("/user")
```

```
    List<UserDTO> getAllUsers() {
        return UserMapper.toDTOList(UserService.getAllUsers());
    }
```

Méthode pour récupérer tous les utilisateurs.

L'annotation @GetMapping est utilisée pour mapper les demandes HTTP GET sur des méthodes de gestionnaire spécifiques.

L'URL de point de terminaison est "/user".

Cette méthode ne prend aucun paramètre.

Cette méthode retourne une liste d'objets de type UserDTO, qui est créée en convertissant la liste d'objets Entity retournée par la méthode getAllUsers de la classe UserService en utilisant la méthode toDTOList de la classe UserMapper.

**Méthode 3 :**

```
@GetMapping("/user/{id}")
    public UserDTO getUser(@PathVariable Long id) {
        return UserMapper.toDTO(UserService.getUser(id));
    }
```

Méthode pour récupérer un seul utilisateur du système en fonction de son ID.

L'annotation @GetMapping est utilisée pour mapper les demandes HTTP GET sur des méthodes de gestionnaire spécifiques.

L'URL de point de terminaison est "/user/{id}", où {id} est un marqueur de position pour l'identificateur unique de l'utilisateur.

Cette méthode prend un seul paramètre, id, annoté avec @PathVariable, ce qui signifie qu'il est attendu dans le chemin d'URL.

Cette méthode retourne un objet de type UserDTO, qui est créé en appelant la méthode getUser de la classe UserService avec l'id fourni, puis en convertissant l'objet Entity retourné en UserDTO à l'aide de la méthode toDTO de la classe UserMapper.

**Méthode 4 :**

```
@PutMapping("/user/{id}")
    public void saveUser(final @PathVariable Long id, final @RequestBody UserDTO user) {
        if (user.getId() == id) {
            UserService.saveUser(UserMapper.toEntity(user));
```

Cette méthode définit un point d'extrémité Spring pour mettre à jour un utilisateur en utilisant la méthode HTTP PUT et l'annotation @PutMapping("/user/{id}"). Elle prend en entrée deux paramètres : l'identifiant unique de l'utilisateur (id) via l'annotation @PathVariable et l'objet utilisateur (user) via l'annotation @RequestBody. La méthode vérifie que l'identifiant dans l'URL et celui de l'objet utilisateur correspondent. Si les identifiants correspondent, elle enregistre l'utilisateur en appelant la méthode saveUser du service UserService en lui passant l'objet utilisateur converti en une entité à l'aide de la méthode toEntity de la classe UserMapper.

**Méthode 5 :**

```
@DeleteMapping («/user/{id}»)
    String deleteUser (@PathVariable Long id) {if (! userRepository. existsById [id]) {throw new UserNotFoundException (id);
            }
            userRepository. deleteById (id);
            return “Users with id” + id + “has been deleted success.”;
    }
```

Cette méthode définit un mappage HTTP «DELETE» pour la requête «/user/{id}» qui supprime un utilisateur dans le référentiel d’utilisateurs. Elle prend en entrée un identifiant d’utilisateur (id) à partir du paramètre d’URL «id». Elle vérifie si un utilisateur avec l’identifiant donné existe dans le référentiel d’utilisateurs. Si oui, l’utilisateur est supprimé à partir du référentiel d’utilisateurs en appelant la méthode «deleteById». Si aucun utilisateur n’est trouvé, une exception «UserNotFoundException» est levée avec l’identifiant donné. Enfin, la méthode retourne un message indiquant le succès de la suppression.

# Comment protéger les formulaires ?

## Comment sécuriser nos formulaires ?

Pour sécuriser un formulaire, il est important de prendre plusieurs mesures :

- Validation côté serveur : Valider les entrées du formulaire sur le serveur pour éviter les attaques par injection de code ou de scripts.
- Cryptage des données : Crypter les données transmises à l'aide d'une connexion HTTPS pour protéger les informations sensibles.
- Limitation des tentatives de soumission : Limiter le nombre de tentatives de soumission pour éviter les attaques par force brute.
- Protection contre les attaques CSRF : Imposer des jetons CSRF pour empêcher les attaques d'utiliser les données du formulaire sans autorisation.
- Contrôle de la saisie utilisateur : Valider les entrées pour vous assurer que les données entrées sont valides et conformes aux exigences.

Il nous est également recommandé de stocker les informations sensibles, telles que les mots de passe, de manière sécurisée en utilisant une fonction de hachage sécurisée.

La validation côté serveur d'un formulaire en Java peut être réalisée en utilisant le framework de développement web Java, tel que Spring ou JavaServer Faces (JSF).

Voici un exemple de validation côté serveur en utilisant Spring MVC :

```
@Controller
public class FormController {
 @RequestMapping(value = "/submitForm", method = RequestMethod.POST)
  public String submitForm(@Valid @ModelAttribute("formData") FormData formData, BindingResult result, Model model) {
    if (result.hasErrors()) {
      return "formPage";
    }
    // Insert form data into database
    return "successPage";
  }
}
```

Dans l'exemple plus haut, nous avons défini une classe FormController avec une méthode submitForm qui est appelée lorsque le formulaire est soumis. Nous utilisons l'annotation @Valid pour valider les données du formulaire et l'annotation @ModelAttribute pour récupérer les données entrées. Si les données ne sont pas valides, le résultat de la validation est enregistré dans un objet BindingResult que nous pouvons vérifier en utilisant la méthode hasErrors(). Si tout est valide, nous pouvons enregistrer les données dans la

base de données et renvoyer une vue de succès. Sinon, nous pouvons renvoyer la vue de formulaire avec les erreurs affichées.

### Comment crypter les données transmises ?

Pour crypter les données transmises à l'aide d'une connexion HTTPS, nous devons configurer notre serveur web pour utiliser un certificat SSL/TLS valide. Ce certificat SSL/TLS garantit que la connexion entre le navigateur et le serveur est sécurisée et que les données transmises sont cryptées.

Voici les étapes générales pour configurer HTTPS sur un serveur web :

- Obtenir un certificat SSL/TLS valide : Il faudra acheter un certificat SSL/TLS auprès d'une autorité de certification reconnue, telle que DigiCert, GlobalSign ou Comodo.
- Configurer ensuite notre serveur web : Suivant le serveur web utilisé, nous devons configurer celui-ci pour utiliser le certificat SSL/TLS obtenu. Pour Apache, il faut ajouter une directive SSL dans le fichier de configuration, pour Nginx, ajouter une directive ssl_certificate.
- Rediriger toutes les demandes HTTP vers HTTPS : Utiliser une directive de redirection ou un plugin pour rediriger automatiquement toutes les demandes HTTP vers HTTPS.
- Modifier notre code pour utiliser HTTPS : Dans notre code, nous devons nous assurer que toutes les requêtes vers le serveur sont effectuées via HTTPS et non via HTTP.
- Une fois que le HTTPS configuré sur notre serveur, les données transmises à travers la connexion sont cryptées et protégées contre les attaques et les vols d'informations. Il est important de tenir à jour le certificat SSL/TLS et la configuration du serveur pour garantir la sécurité de la connexion.

L'imposition de jetons CSRF (Cross-Site Request Forgery) est aussi un moyen de protéger les formulaires en ligne contre les attaques d'utilisation non autorisée de données. Les jetons CSRF sont des tokens uniques générés pour chaque demande de formulaire, qui doivent être envoyés avec les données du formulaire.

Voici les étapes générales pour implémenter des jetons CSRF :

- Générer un jeton CSRF unique pour chaque demande de formulaire : Nous devons générer un jeton CSRF en utilisant une fonction de hachage cryptographique pour créer une valeur aléatoire. Ce jeton peut être stocké dans le cache du navigateur ou sur le serveur.
- Inclure le jeton CSRF dans le formulaire : Il faut inclure le jeton CSRF en tant qu'input caché dans le formulaire.
- Valider le jeton CSRF côté serveur : Lorsque le formulaire est soumis, nous devons valider le jeton CSRF en le comparant à la valeur stockée dans le cache du navigateur ou sur le serveur.
- Refuser les demandes de formulaire avec des jetons CSRF non valides : Si le jeton CSRF ne correspond pas à la valeur stockée, il faut refuser la demande et retourner une erreur.

L'utilisation de jetons CSRF permet de s'assurer que les données envoyées via le formulaire proviennent d'une source fiable et autorisée. Les attaques

d'utilisation non autorisée de données ne peuvent pas être effectuées sans le jeton CSRF valide, ce qui renforce la sécurité de notre formulaire en ligne. Il est important de tenir à jour notre implémentation de jetons CSRF pour garantir qu'elle soit sécurisée contre les vulnérabilités récentes ou les nouvelles techniques d'attaque.

## Generating JWT

J'ai reproduit par la suite les schémas suivants qu'illustrent la structure et l'impact de JWT entre le client et le server.

**Structure of JWT**

JWT has the following format - header.payload.signature

*Figure 9 : La structure d'une JWT*

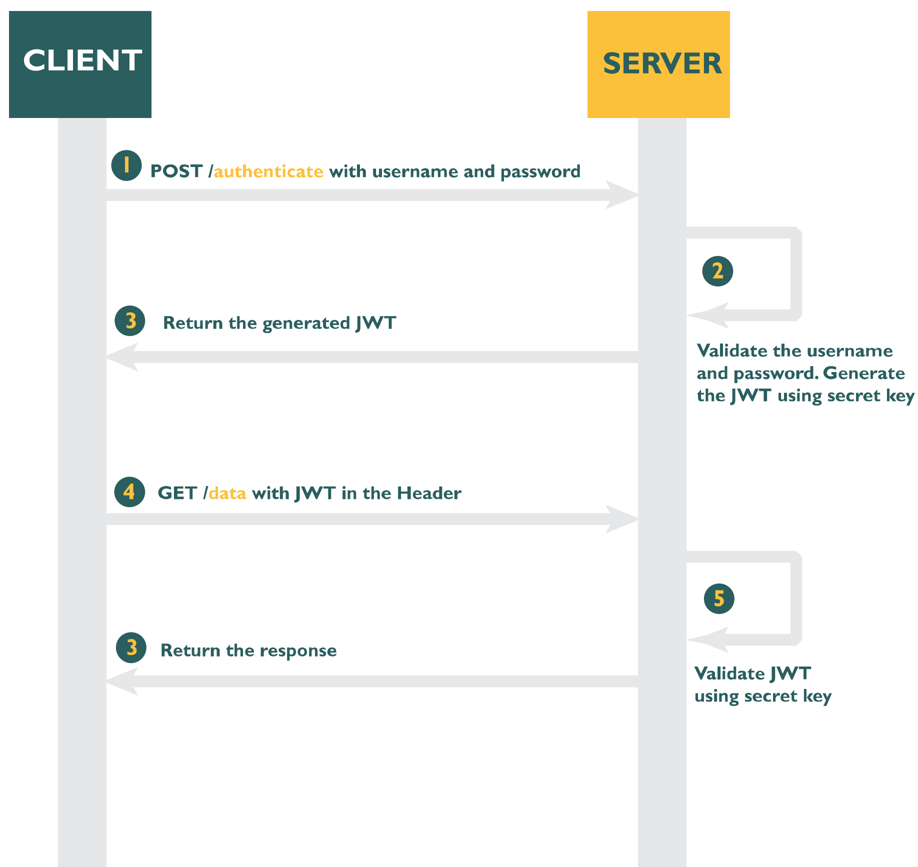



*Figure 10 : Fonctionnement JWT (https://www.javainuse.com/spring/boot-jwt, s.d.)*

# Les tests unitaires

eclipse-workspace - users/src/test/java/com/atelab/users/UserControllerTest.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer | JUnit

Finished after 0,486 seconds

Runs: 4/4 Errors: 0 Failures: 0

- UserControllerTest [Runner: JUnit 5] (0,157 s)
  - Usercontructor() (0,121 s)
  - UserToDtos() (0,016 s)
  - userDtoToEntity() (0,005 s)
  - nullToDto() (0,005 s)

Failure Trace

UserController.java | UserNotFoundException.java | UserControllerTest.java

```
44      @Test
45      void UserToDtos() {
46          // given
47          List<User> user = new ArrayList<>();
48
49          for (long i = 0; i < 10; i++) {
50              User user1 = new User();
51              user1.setId(i);
52              user1.setName("roland");
53              user1.setUsername("rols");
54              user1.setEmail("rd@gmail.com");
55
56              user.add(user1);
57          }
58
59          // when
60
61          List<UserDTO> dtos = UserMapper.toDTOList(user);
```

Console | Coverage

UserControllerTest (8 févr. 2023 09:46:40)

| Element | Coverage | Covered Instructio... | Missed Instructions | Total Instructions |
|---|---|---|---|---|
| users | 71,3 % | 293 | 118 | 411 |
| src/main/java | 53,4 % | 135 | 118 | 253 |
| com.atelab.users | 0,0 % | 0 | 17 | 17 |
| com.atelab.users.controller | 0,0 % | 0 | 37 | 37 |
| com.atelab.users.exception | 0,0 % | 0 | 25 | 25 |
| com.atelab.users.service.impl | 0,0 % | 0 | 24 | 24 |
| com.atelab.users.dto | 67,4 % | 31 | 15 | 46 |
| com.atelab.users.model | 100,0 % | 31 | 0 | 31 |
| com.atelab.users.dto.mapper | 100,0 % | 73 | 0 | 73 |
| UserMapper.java | 100,0 % | 73 | 0 | 73 |
| UserMapper | 100,0 % | 73 | 0 | 73 |
| src/test/java | 100,0 % | 158 | 0 | 158 |

*Figure 11 : Test unitaire de UserMapper 100%*

eclipse-workspace - users/src/test/java/com/atelab/users/UserControllerTest.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer | JUnit

Finished after 0,455 seconds

Runs: 1/1 Errors: 0 Failures: 0

UserControllerTest [Runner: JUnit 5] (0,089 s)
UserToDtos() (0,089 s)

Failure Trace

UserController.java | UserControllerTest.java

```java
46          // given
47          List<User> user = new ArrayList<>();
48
49          for (long i = 0; i < 10; i++) {
50              User user1 = new User();
51              user1.setId(i);
52              user1.setName("roland");
53              user1.setUsername("rols");
54              user1.setEmail("rd@gmail.com");
55
56              user.add(user1);
57          }
58
59          // when
60
61          List<UserDTO> dtos = UserMapper.toDTOList(user);
62
63          // then
64          for (int i = 0; i < 10; i++) {
65              Assert.isTrue(user.get(i).getId().equals(dtos.get(i).getId()), "Id has changed");
66              Assert.isTrue(user.get(i).getName().equals(dtos.get(i).getName()), "Name has changed");
67              Assert.isTrue(user.get(i).getUsername().equals(dtos.get(i).getUsername()), "Username has changed");
68              Assert.isTrue(user.get(i).getEmail().equals(dtos.get(i).getEmail()), "Email has changed");
69
```

Console | Coverage

UserControllerTest.UserToDtos (8 févr. 2023 20:49:57)

| Element | Coverage | Covered Instructio... | Missed Instructions | Total Instructions |
|---|---|---|---|---|
| users | 21,1 % | 212 | 793 | 1 005 |
| src/test/java | 64,6 % | 102 | 56 | 158 |
| com.atelab.users | 64,6 % | 102 | 56 | 158 |
| src/main/java | 13,0 % | 110 | 737 | 847 |
| com.atelab.users | 0,0 % | 0 | 17 | 17 |
| com.atelab.users.controller | 0,0 % | 0 | 87 | 87 |
| com.atelab.users.exception | 0,0 % | 0 | 25 | 25 |
| com.atelab.users.security | 0,0 % | 0 | 78 | 78 |
| com.atelab.users.security.controller | 0,0 % | 0 | 30 | 30 |
| com.atelab.users.security.controller.DTO | 0,0 % | 0 | 43 | 43 |
| com.atelab.users.security.jwt | 0,0 % | 0 | 170 | 170 |
| com.atelab.users.security.models | 0,0 % | 0 | 107 | 107 |
| com.atelab.users.security.service.impl | 0,0 % | 0 | 19 | 19 |
| com.atelab.users.security.utils | 0,0 % | 0 | 97 | 97 |
| com.atelab.users.service.impl | 0,0 % | 0 | 24 | 24 |
| com.atelab.users.dto | 67,4 % | 31 | 15 | 46 |
| com.atelab.users.model | 100,0 % | 31 | 0 | 31 |
| com.atelab.users.dto.mapper | 65,8 % | 48 | 25 | 73 |
| UserMapper.java | 65,8 % | 48 | 25 | 73 |

*Figure 12 : Test unitaire de UserMapper 100% (test à reprendre)*

…