Présentation du :

Projet SpringBoot JAVA
Travel Planning

- RAKOTOARIMALALA Hasinjato Mickaël
- RANDRIARIMALALA Aina Muriel

Table des matières

I. Introduction sur le thème ‘Travel Planning’.

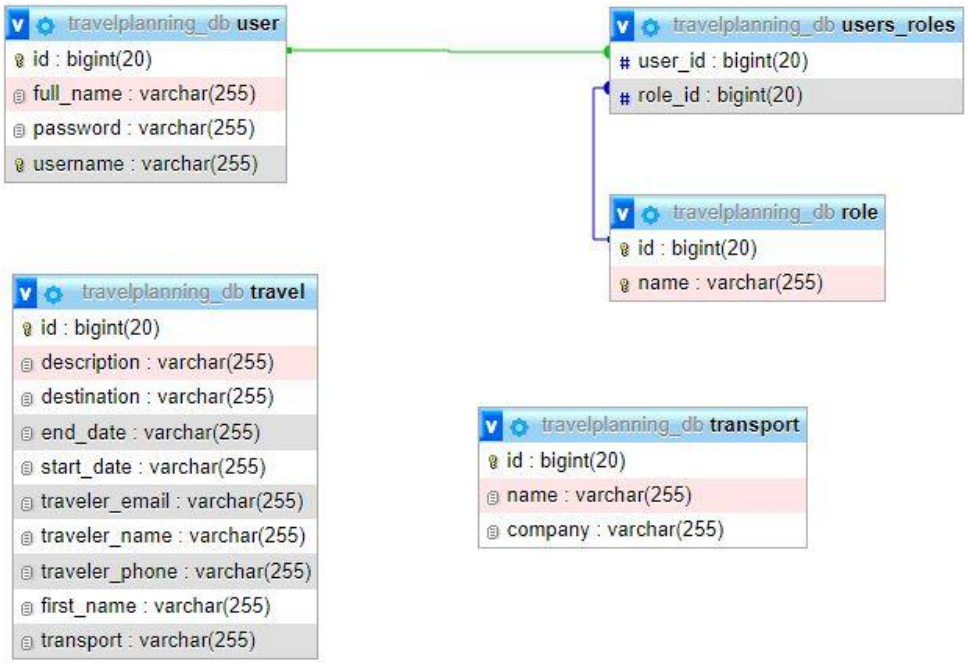
Le projet ‘Travel Planning’ vise à développer une application de voyages. L’objectif principal est de fournir aux utilisateurs une plateforme où ils peuvent explorer des destinations, des moyens de transport, et gérer leurs réservations.

Pour commencer, on a utilisé Spring Boot avec STS4 (Spring Tool Suite 4).

Spring Boot est un framework Java, qui simplifie le développement d’applications Spring en automatisant la configuration et en intégrant de nombreuses dépendances couramment utilisées.

II. Modélisation du projet 'Travel Planning'.

Voici une capture d'écran, source : phpmyadmin, de la modélisation de la base de données



Pour le gestion de la base de donnée, on a utilisé le XamPP (avec le service php et mysql).

Donc, on a pour le moment 4 tables dont :

- user (qui contient les utilisateurs enregistrer dans le système) ;
- role (les types de role enregistrer soit ADMIN soit USER) ;
- travel (la table pour stocker les informations des voyages) ;
- transport table pour enregistrer les moyens de transport disponible) ;

III. Description du projet dans STS4

On a choisi l'outil de gestion de dépendance et de construction de projet le **gradle**.

Gradle : est un système de 'build' open source basé sur Groovy et Kotlin. Il offre une syntaxe de configuration **flexible** et **puissante** permettant de décrire les tâches de construction et de gestion des dépendances.

Avec Gradle, on peut spécifier les dépendances de notre projet, gérer les versions des biblothèque, compiler, tester et créer des artefacts exécutables. Elle est aussi particulièrement appréciée pour sa flexibilité, sa **performance** et sa capacité à **gérer des projets de grande envergure**.

Voici quelques avantages de Gradle par rapport à Maven :

- Flexibilité de la configuration : elle offre une syntaxe de configuration sur Kotlin, qui est plus expressive et flexible que la syntaxe XML utilisée par maven. Cela permet d'écrire des scripts de définir des tâches personnalisées plus facilement.
- Performances améliorées : Gradle est conçu pour être **rapide** et **efficace**, en particulier pour des projets de grande envergure. Il utilise une approche basée sur une modèle de données dirigé par les tâches, ce qui lui permet d'exécuter uniquement les tâches nécessaires pour une build spécifique, en évitant les traitements redondants. Cela peut entraîner des **temps de build plus rapide par rapport à maven**.
- Mise à jour des dépendances incrémentielle : elle offre la possibilité de mettre à jour de manière incrémentielle les dépendances sans avoir à ré-exécuter l'ensemble du build.
- Communauté : Gradle est un outil en évolution, avec une communauté active. De nombreux projets open source et entreprises adoptent Gradle comme outil de choix pour leur gestion de dépendance.

IV. Les paramètres dans build.grade pour ce projet :

1) Les versions :

```
org.springframework.boot : version 3.1.0
io.spring.dependency-management : version 1.1.0
SNAPSHOT : version 0.0.1
JAVA : version 17
```

2) Les dépendances :

```
'org.springframework.boot:spring-boot-starter-actuator'
'org.springframework.boot:spring-boot-starter-web-services'
'org.springframework.boot:spring-boot-starter-jdbc'
'org.springframework.boot:spring-boot-starter-security'
'org.springframework.boot:spring-boot-starter-thymeleaf'
'org.springframework.boot:spring-boot-starter-web'
'org.springframework.boot:spring-boot-starter-websocket'
'org.thymeleaf.extras:thymeleaf-extras-springsecurity6'
'org.springframework.boot:spring-boot-starter-data-jpa'
'com.mysql:mysql-connector-j'
```

V. Les configurations dans le fichier application.properties :

```
server.port=8080
spring.datasource.url=jdbc:mysql://localhost:3306/travelPlanning_db
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.security.form-login.defautl-success=/registration
spring.web.resources.static-locations=classpath:/static/
spring.mvc.view.prefix==/static/
spring.mvc.view.suffix=.html
spring.main.allow-circular-references=true
```

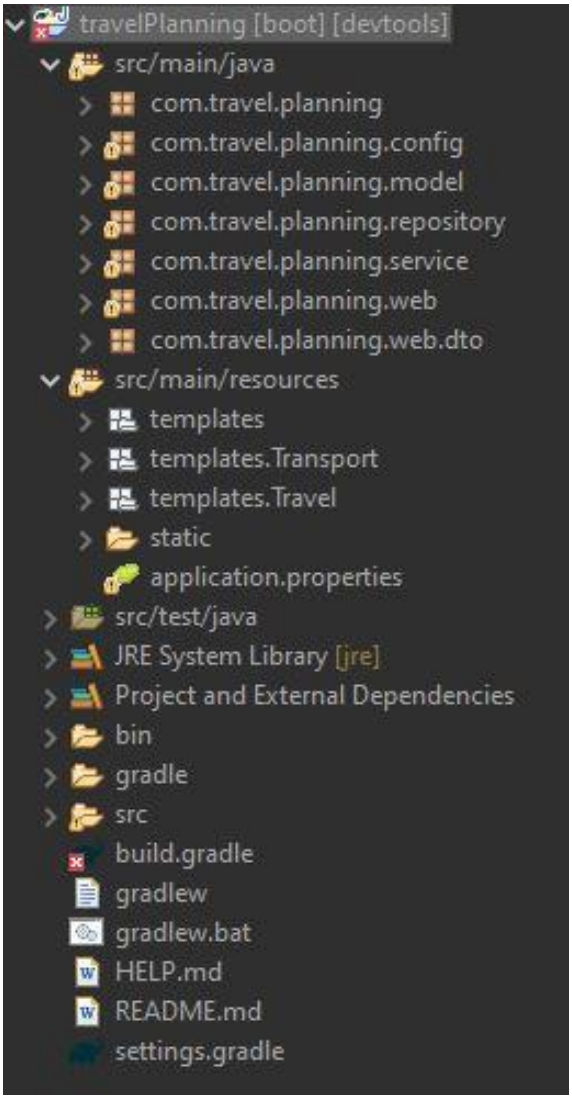
VI. Sur Spring Boot

Pour le développement de ce projet, nous avons un petit problème avant, puisque nous ne connaissons pas avant l'utilisation di framework Java Spring Boot. Et nous ne savons pas comment manipuler le logiciel STS4.

La solution de rechercher sur internet, comment l'utiliser pour conçevoir une application web. Pour cela on a vue qu'il y avait deux types de … pour le framework Spring Boot. Je ne sais pas mais j'ai créé la base de l'application avec gradle

VII. L'arborescence du projet dans STS4 :

Voici l'arborscence de ce projet, une capture d'écran dans STS4.



Vu cette arborescence, on a utilisé le modèle MVC (Model View Controller). C'est un motif de conception largement utilisé dans le développement logiciel, afin d'organiser et structurer les applications. Il joue un rôle central pour le développement d'applications web.

VIII. Le projet : La page de connexion

Quand on installe la dépendance **spring security**, ce dernier met en place une page de 'login' par défaut, que je n'ai pas la compétence de la personnalisé. Mais la solution est de faire une autre page de login, et dans mon MainController.java j'ai créé une autre @GetMapping sur /loginn avec deux 'n'à la fin, pour afficher mon login.html. Et une @PostMapping de /loginn pour l'authentification de l'utilisateur dans la base de donnée, et qui va rédirigé vers la page '/home' et si le nom d'utilisateur et/ou le mot de passe sont invalident, se redirige vers /loginn?error qui va afficher une message **'Identifiant invalide'**.

Malheureusement il y a deux pages de login, la première c'est le par défaut, génerér par spring security, et l'autre ma page de login personnalisé.

Voici les captures d'écran de ces pages de login :

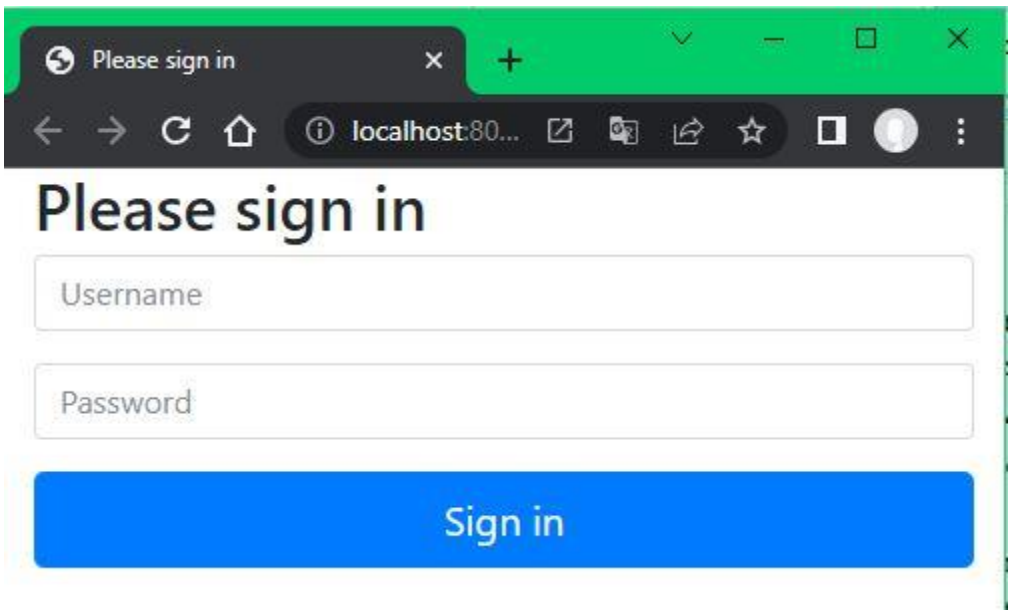


Figure 1: La première login

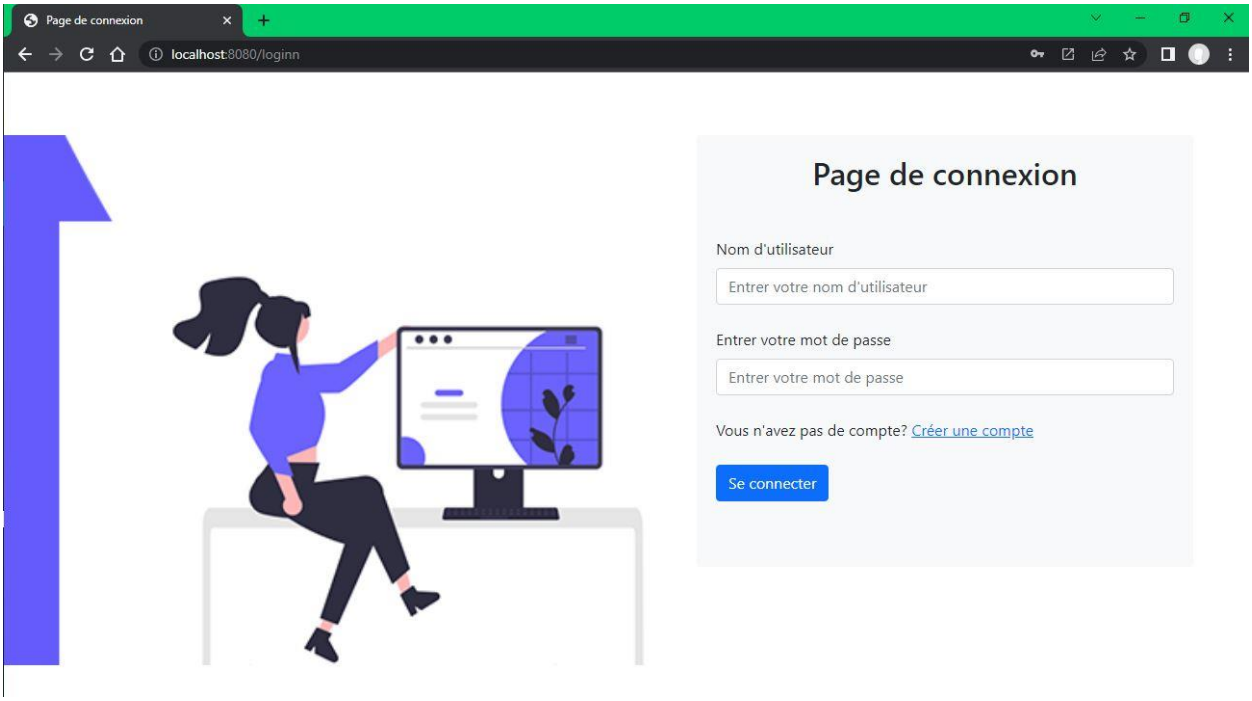


Figure 2: Login personnalisée

Il y a cette lien Créer une compte qui vous dirige vers /registration la page pour l'enregistrement des utilisateurs dans la base de donnée. Le champ rôle dans la base de données est compléter automatiquement par 'USER'.

Voici cette interface :

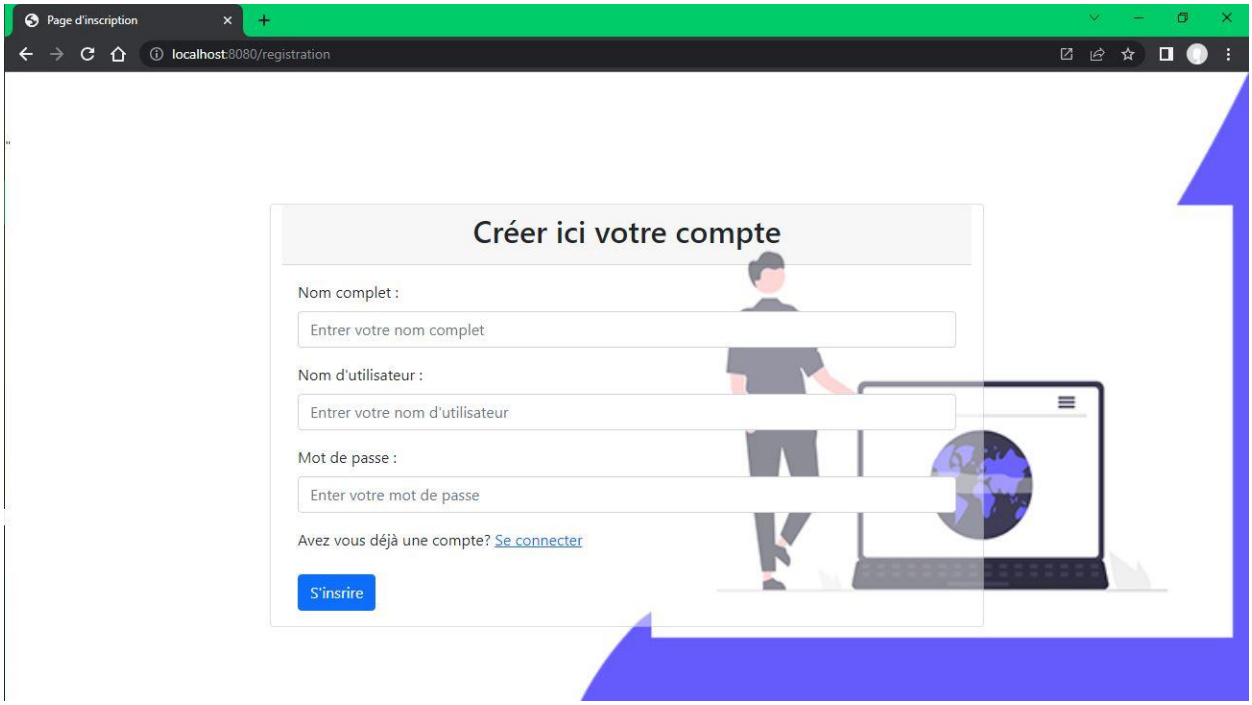


Figure 3: Page de registration

Après la création de du compte, vous êtes rediriger vers /home, la page d'accueil de l'application.

Remarque : Pendant l'enregistrement de l'utilisateur dans la base de donnée, le mot de passe est crypté par :

```
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder.
```

IX. Le projet : La page d'accueil

La page d'accueil a pour route : '/home', dans cette page il y a un message de bienvenue, une zone de recherche, le nom de l'utilisateur connecter.

Sur la gauche, un logo '**TRAVEL'** et une **barre de navigation** : avec une liste des entités de l'application.

- Les voyages : (menu)
  - La liste (sous-menu)
  - Ajouter (sous-menu)
- Les transports :
  - La liste
  - Ajouter

Dans le 'body wrapper', il y un petit **tableau de bord** qui affiche le *nombre de l'utilisateur* dans le système, le nombre des visiteurs de l'application, le nombre de *voyage* déjà enregistrer et le nombre de *transport* enregistrer.

Un **calendrier intégré** et une **barre de notification**. Et comme toute application web, en bas de page il y a ce qu'on appelle **footer** avec le copyright.

Voici une capture d'écran pour illustrer cette page d'accueil.

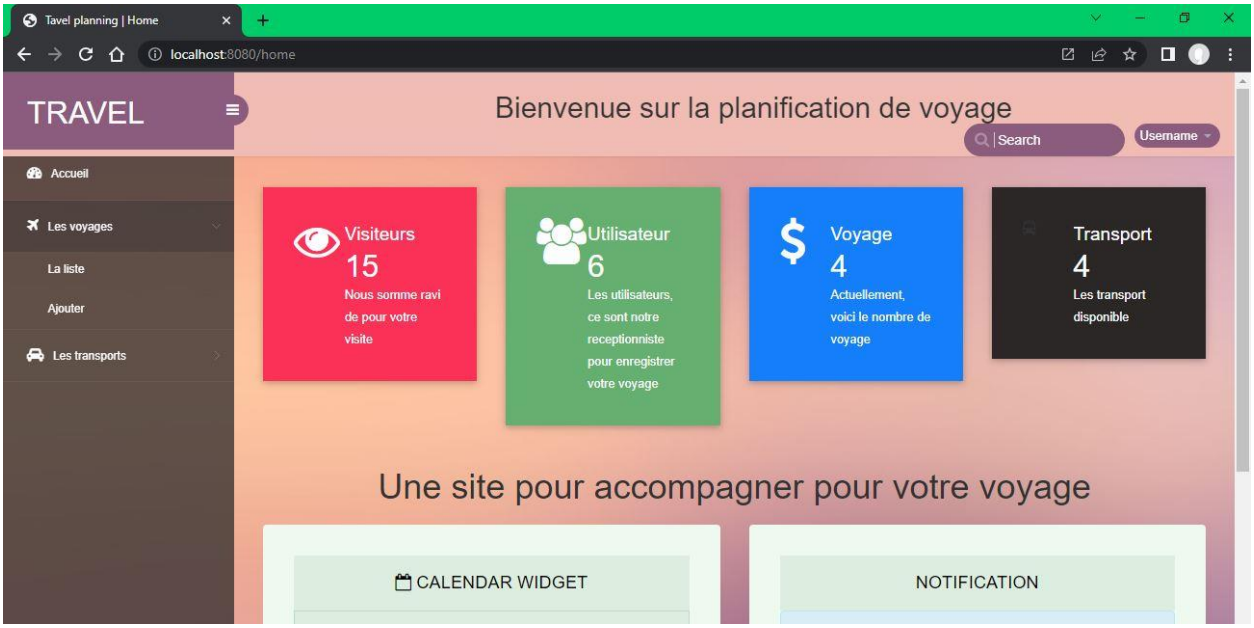


Figure 4: Page d'accueil

X. La page pour Transport

Sur le route /transports, la liste de tous les transports disponbile. Avec les colonnes suivante : id(#) , nom(name), compagnie de transport(company), illustrer dans une table-hover. La voici ci-dessous :

Figure 5 : Page /transports

Sur cette page, on peut Ajouter un nouveau moyen de transport, en cliquant sur bouton Ajouter transport et qui affiche une formulaire d'ajout avec les champs : id(de type hidden), nom(pour le type de moyen de transport), et le compagnie(pour le compagnie appartenant ce moyen de transport), et enregistre dans la base de données.

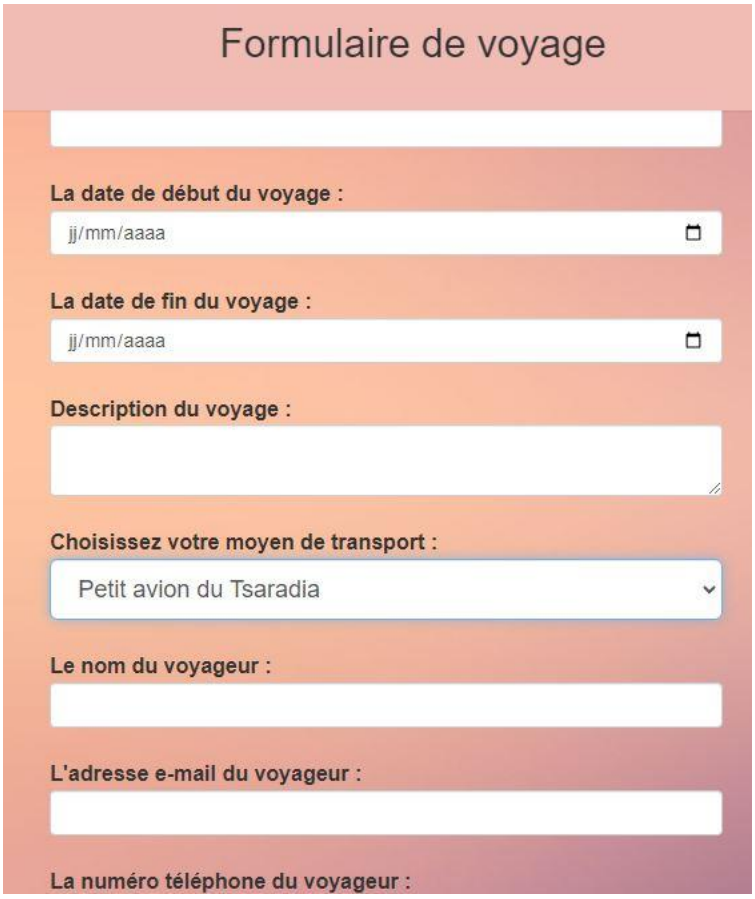
Et si on clique sur le bouton Modifier , on vas sur une page de modification, avec des champs, dont les valeurs dans ces champs sont les valeurs qu'on veut modifier. Voici une exemple de modification.

En clique sur le bouton Enregistrer pour enregistrer les modifications dans la base de données.

C'est aussi la même concepte avec l'entité Travel, mais avec une spécification sur l'ajout d'un voyage. Voici maintenant une capture d'écran illustrant la liste des voyages stocké dans la base de données. Pour cette entité, il y a 9 colonnes, les suivantes : id(#), destination, date de début, date de fin, description (pour la description du voyage), transport (affiche le transport choisi pendant l'enregistrement), le nom du voyageur, les contacts du voyageur (adresse e-mail, et numéro de téléphone).

On peut directement appeler le voyageur, en cliquant sur le numéro de téléphone correspondant, et aussi envoyer une courrielle en cliquant sur l'adresse e-mail du voyageur.

Pour ajouter ou modifier un voyage, voici l'interface. Sur le champ d'ajout du moyen de transport, affiche dans un **checkbox** les données dans la base de données du colonne transport.



XI. La sécurité du projet.

La sécurité de l'application se trouve dans le fichier **SecurityConfiguration.**

Premièrement, le cryptage du mot de passe, qui est assuré par org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder.

Puis, pour les routes ou url, les utilisateurs ne peuvent pas entrer dans tous les pages du l'application en premier, d'abord il faut s'authentifier dans la page de login. Après il peut visiter toute l'intégrité de l'application.

Si l'utilisateur, va déconnecter, il va détruire sa session et diriger vers la page de login.

XII. Repository du projet.

Voici le lien pour accéder au code source de ce projet :

https://github.com/Hasinjato/travel-planning