



ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR
Membre de
HONORIS UNITED UNIVERSITIES

GESTION DES CINEMAS

Réalisé par :

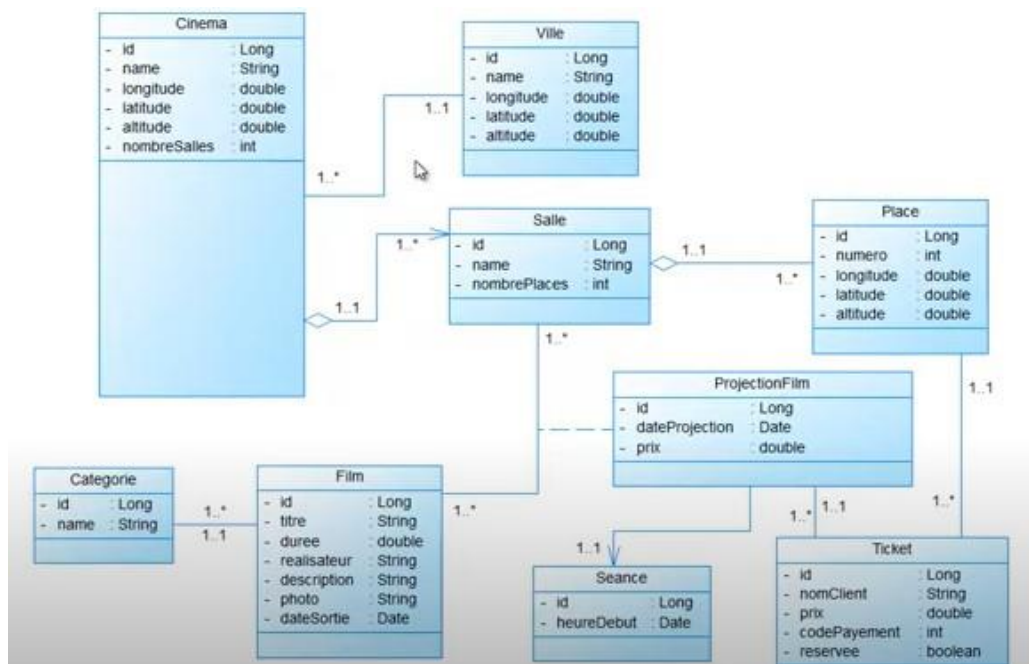
ISSALE Oumaima & Knar Imane

Objectif :

Le but de ce projet est de développer une application pour la gestion des cinémas en utilisant le framework spring (partie Backend) et angular (partie Frontend).

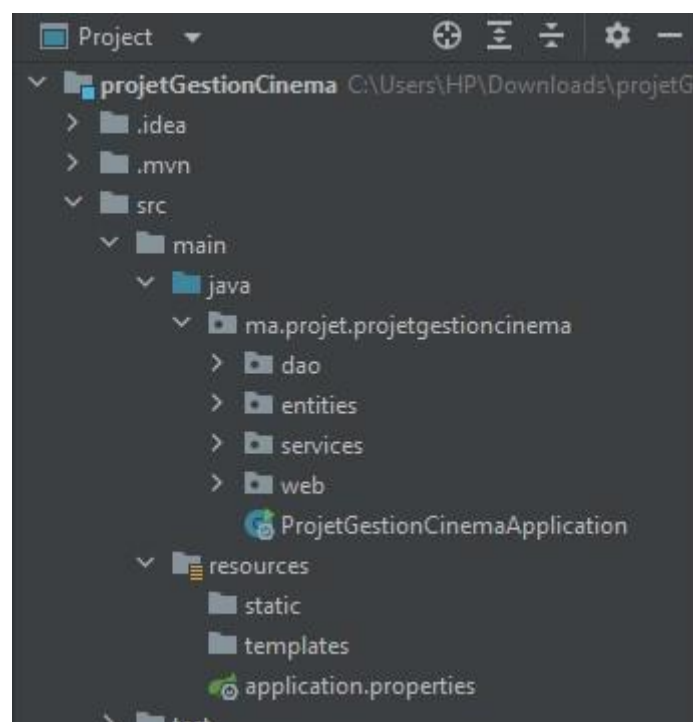
Conception de l'application:

La figure ci-dessous représente la modélisation de notre application sous forme d'un diagramme de classe.



Backend :

Architecture du projet Backend :



On a divisé le projet en 3 packages :

entities : qui représente nos objets qui seront convertis en entités JPA grâce au mapping.

dao : qui contient l'ensemble des interfaces JPA repository .

service : qui représente le métier de l'application.

web : qui contient le Controller, responsable de l'envoi et la réception des requêtes http.

Entités JPA :

```
CinemaRestController.java x 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
import lombok.NoArgsConstructor;
import lombok.ToString;

import javax.persistence.*;
import java.util.Collection;

@Entity
@Data @AllArgsConstructor @NoArgsConstructor @ToString

public class Categorie {
    @Id @GeneratedValue(strategy= GenerationType.IDENTITY)
    private Long id;

    @Column(length=75)
    private String name;

    @OneToMany(mappedBy="categorie")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Collection<Film> films;
}
```

```
CinemaRestController.java x 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
import javax.persistence.*;
import java.io.Serializable;
import java.util.Collection;

@Entity
@Data @AllArgsConstructor @NoArgsConstructor @ToString

public class Cinema implements Serializable {
    @Id @GeneratedValue(strategy= GenerationType.IDENTITY)
    private Long id;

    @Column(length=75)
    private String name;
    private double longitude,latitude,altitude;
    private int nombreSalles;

    @OneToMany(mappedBy="cinema")
    private Collection<Salle> salles;

    @ManyToOne
    private Ville ville;
}
```

```
import java.util.Collection;
import java.util.Date;

@Entity
@Data @AllArgsConstructor @NoArgsConstructor @ToString

public class Film {
    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    private Long id;

    private String titre,realisateur,description,photo;
    private double duree;
    private Date dateSortie;

    @OneToMany(mappedBy="film")
    @JsonProperty(access= JsonProperty.Access.WRITE_ONLY)
    private Collection<Projection> projections;

    @ManyToOne
    private Categorie categorie;
}
```

```
CinemaRestController.java x 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
import lombok.ToString;

import javax.persistence.*;
import java.util.Collection;

@Entity
@Data @AllArgsConstructor @NoArgsConstructor @ToString

public class Place {
    @Id @GeneratedValue(strategy= GenerationType.IDENTITY)
    private Long id;
    private int numero;
    private double longitude,latitude,altitude;

    @ManyToOne
    private Salle salle;

    @OneToMany(mappedBy="place")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Collection<Ticket> tickets;
}
```

```

9 usages
@Entity
@Data @AllArgsConstructor @NoArgsConstructor @ToString

public class Projection {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private Date dateProjection;
    private double prix;

    @ManyToOne
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Salle salle;

    @ManyToOne
    private Film film;

    @OneToMany(mappedBy = "projection")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Collection<Ticket> tickets;

    @ManyToOne
    private Seance seance;
}

```

```

7 usages
@Entity
@Data @AllArgsConstructor @NoArgsConstructor @ToString

public class Salle {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 75)
    private String name;
    private int nombrePlace;

    @ManyToOne
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Cinema cinema;

    @OneToMany(mappedBy = "salle")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Collection<Place> places;

    @OneToMany(mappedBy = "salle")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Collection<Projection> projections;
}

```

```

package ma.projet.projetgestioncinema.entities;

import lombok.*;
import javax.persistence.*;
import java.util.Date;

5 usages
@Data @AllArgsConstructor @NoArgsConstructor @ToString
@Entity
public class Seance {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Temporal(TemporalType.TIME)
    private Date heureDebut;
}

```

```

10 usages
@Entity
@Data @AllArgsConstructor @NoArgsConstructor @ToString

public class Ticket {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 75)
    private String nomClient;
    private double prix;

    @Column(unique = false, nullable = true)
    private Integer codePayment;
    private boolean reservee;

    @ManyToOne
    private Place place;

    @ManyToOne
    private Projection projection;
}

```

```

import lombok.*;
import ma.projet.projetgestioncinema.entities.Cinema;

import javax.persistence.*;
import java.util.Collection;

5 usages
@Data @AllArgsConstructor @NoArgsConstructor @ToString
@Entity
public class Ville {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 75)
    private String name;
    private double longitude, latitude, altitude;

    @OneToMany(mappedBy = "ville")
    private Collection<Cinema> cinemas;
}

```

Les entités ci-dessus représentent notre diagramme de classe avec l'ensemble des associations représentées par `@OneToMany` , `@OneToOne` , `@ManyToOne` ou `@ManyToMany`.

Les interfaces JPA :

```
package ma.projet.projetgestioncinema.dao;

import ma.projet.projetgestioncinema.entities.Cinema;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

1 usage
@RepositoryRestResource
public interface CinemaRepository extends JpaRepository<Cinema, Long>{
}
```

```
package ma.projet.projetgestioncinema.dao;

import ma.projet.projetgestioncinema.entities.Film;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

3 usages
@RepositoryRestResource
public interface FilmRepository extends JpaRepository<Film, Long>{
}
```

```
import ...

1 usage
@RepositoryRestResource
public interface PlaceRepository extends JpaRepository<Place, Long>{
}
```

```
package ma.projet.projetgestioncinema.dao;

import ...

1 usage
@RepositoryRestResource
public interface ProjectionRepository extends JpaRepository<Projection, Long>{
}
```

```
package ma.projet.projetgestioncinema.dao;

import ...

1 usage
@RepositoryRestResource
public interface SalleRepository extends JpaRepository<Salle, Long>{
}
```

```
package ma.projet.projetgestioncinema.dao;

import ...

1 usage
@RepositoryRestResource
public interface SeanceRepository extends JpaRepository<Seance, Long>{
}
```

```
package ma.projet.projetgestioncinema.dao;

import ...

3 usages
@RepositoryRestResource
public interface TicketRepository extends JpaRepository<Ticket, Long>{
}
```

```
package ma.projet.projetgestioncinema.dao;

import ...

1 usage
@RepositoryRestResource
public interface VilleRepository extends JpaRepository<Ville, Long>{
}
```

```
package ma.projet.projetgestioncinema.dao;

import ...

1 usage
@RepositoryRestResource
public interface CategorieRepository extends JpaRepository<Categorie, Long>{
}
```

Ces interfaces héritent de l'interface JPA Repository ,qui génère toutes les methodes classique appliquées sur les entités

Si on veut personnaliser une méthode, il suffit de la définir à l'intérieur de l'interface.

Code metier :

```
SeanceRepository.java  TicketRepository.java  VilleRepository.java
1 package ma.projet.projetgestioncinema.services;
2
3 public interface ICinemaInitService {
4
5     1 usage 1 implementation
6     public void initVilles();
7     1 usage 1 implementation
8     public void initCinemas();
9     1 usage 1 implementation
10    public void initSalles();
11    1 usage 1 implementation
12    public void initPlaces();
13    1 usage 1 implementation
14    public void initSeances();
15    1 usage 1 implementation
16    public void initCategories();
17    1 usage 1 implementation
18    public void initFilms();
19    1 usage 1 implementation
20    public void initProjections();
21    1 usage 1 implementation
22    public void initTickets();
23
24 }
```

On définit dans cette interfaces toutes les méthodes à implémenter ,relatifs au traitement de l'application.

```
@Service
@Transactional
public class CinemaInitServiceImpl implements ICinemaInitService {

    3 usages
    @Autowired
    private VilleRepository villeRepository;

    2 usages
    @Autowired
    private CinemaRepository cinemaRepository;

    2 usages
    @Autowired
    private SalleRepository salleRepository;

    1 usage
    @Autowired
    private PlaceRepository placeRepository;

    2 usages
    @Autowired
    private SeanceRepository seanceRepository;

    2 usages
}
```

```
2 usages
@Autowired
private FilmRepository filmRepository;

2 usages
@Autowired
private ProjectionRepository projectionRepository;

2 usages
@Autowired
private CategoryRepository categorieRepository;

1 usage
@Autowired
private TicketRepository ticketRepository;

1 usage
@Override
public void initVilles() {
    Stream.of("Casablanca", "Marrakech", "Rabat", "Tanger").forEach(nameville -> {
        Ville ville = new Ville();
        ville.setName(nameville);
        villeRepository.save(ville);
    });
}
```

```
1 usage
@Override
public void initCinemas() {
    villeRepository.findAll().forEach(ville -> {
        Stream.of("L'Épave", "Nostalgie", "Toussaint", "Châtaignier", "Bambini").forEach(namecinema -> {
            Cinema cinema = new Cinema();
            cinema.setName(namecinema);
            cinema.setNumeroSalles((int)(Math.random()*7));
            cinema.setVille(ville);
            cinemaRepository.save(cinema);
        });
    });
}

1 usage
@Override
public void initSalles() {
    cinemaRepository.findAll().forEach(cinema -> {
        for(int i=0; i<cinema.getNombreSalles(); i++) {
            Salle salle = new Salle();
            salle.setNumero("salle -> (" + i + ")");
            salle.setCinema(cinema);
            salle.setNombrePlaces((int)(Math.random()*200));
            salleRepository.save(salle);
        }
    });
}
```

```
@Override
public void initSeances() {
    DateFormat dateFormat = new SimpleDateFormat("HH:mm");
    Stream.of("12:00", "15:00", "17:00", "19:00", "21:00").forEach(s -> {
        Seance seance = new Seance();
        try {
            seance.setHeureDebut(dateFormat.parse(s));
            seanceRepository.save(seance);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    });
}

1 usage
@Override
public void initCategories() {
    Stream.of("Mystère", "Action", "Fiction", "Drame").forEach(cat -> {
        Categorie categorie = new Categorie();
        categorie.setName(cat);
        categorieRepository.save(categorie);
    });
}
```

```
@Override
public void initFilms() {
    double[] genres = {0.1, 0.2, 0.3, 0.4};
    List<Categorie> categories = categorieRepository.findAll();

    Stream.of("L'Épave", "Nostalgie", "Toussaint", "Châtaignier", "Bambini").forEach(namefilm -> {
        Film film = new Film();
        film.setName(namefilm);
        film.setGenre(categories.get((int)(Math.random()*categories.size())));
        filmRepository.save(film);
    });
}

1 usage
@Override
public void initProjections() {
    double[] prices = {10, 20, 30, 40, 50};
    villeRepository.findAll().forEach(ville -> {
        cinemaRepository.findAll().forEach(cinema -> {
            salleRepository.findAll().forEach(salle -> {
                Projection projection = new Projection();
                projection.setDateProjection(new Date());
                projection.setFilm(film);
                projection.setHeureDebut(new Random().nextInt(24*60));
                projection.setSalle(salle);
                projection.setSeance(seance);
                projectionRepository.save(projection);
            });
        });
    });
}
```

```
@Override
public void initProjections() {
    double[] prices = {10, 20, 30, 40, 50};
    villeRepository.findAll().forEach(ville -> {
        cinemaRepository.findAll().forEach(cinema -> {
            salleRepository.findAll().forEach(salle -> {
                Projection projection = new Projection();
                projection.setDateProjection(new Date());
                projection.setFilm(film);
                projection.setHeureDebut(new Random().nextInt(24*60));
                projection.setSalle(salle);
                projection.setSeance(seance);
                projectionRepository.save(projection);
            });
        });
    });
}
```



```

@Override
public void initTickets() {

    projectionRepository.findAll().forEach(p->{
        p.getSalle().getPlaces().forEach(place->{

            Ticket ticket = new Ticket();
            ticket.setPlace(place);
            ticket.setPrix(p.getPrix());
            ticket.setProjection(p);
            ticket.setReservee(false);
            ticketRepository.save(ticket);
        });
    });
}

```

Cette classe CinemaService implémente l'interface ICinemaService et contient les implémentations des méthodes correspondant aux fonctionnalités de l'application.

On déclare les objets interfaces pour communiquer avec la partie dao.

Controller :

```

@RestController
public class CinemaRestController {

    1 usage
    @Autowired
    private FilmRepository filmRepository;

    2 usages
    @Autowired
    private TicketRepository ticketRepository;

    @GetMapping(path="/imageFilm/{id}", produces= MediaType.IMAGE_JPEG_VALUE)
    public byte[] image(@PathVariable(name = "id") Long id) throws Exception{
        Film f=filmRepository.findById(id).get();
        String photoName=f.getPhoto();
        File file=new File( pathName: System.getProperty("user.home")+"/cinema/images/"+photoName);
        Path path= Paths.get(file.toURI());
        return Files.readAllBytes(path);
    }

    @PostMapping("/payerTickets")
    @Transactional
    public List<Ticket> payerTicket(@RequestBody TicketFrom ticketFrom){

        List<Ticket> listTickets=new ArrayList<>();
        ticketFrom.getTickets().forEach(idTicket->{

```

```

        public List<Ticket> payerTicket(@RequestBody TicketFrom ticketFrom){

            List<Ticket> listTickets=new ArrayList<>();
            ticketFrom.getTickets().forEach(idTicket->{

                Ticket ticket=ticketRepository.findById(idTicket).get();
                ticket.setNomClient(ticketFrom.getNomClient());
                ticket.setReservee(true);
                ticket.setCodePayment(ticketFrom.getCodePayment());
                ticketRepository.save(ticket);
                listTickets.add(ticket);
            });

            return listTickets;
        }

    }

    1 usage
    @Data
    class TicketFrom{
        private String nomClient;
        private int codePayment;
        private List<Long> tickets=new ArrayList<>();
    }
}

```

On en définit les méthodes correspondant à chaque type de requête envoyée.

Application main :

```
usage
SpringBootApplication
public class ProjetGestionCinemaApplication implements CommandLineRunner {

    9 usages
    @Autowired
    private ICinemaInitService cinemaInitService;

    //
    public static void main(String[] args) { SpringApplication.run(ProjetGestionCinemaApplication.class, args); }

    @Override
    public void run(String... args) throws Exception {
        cinemaInitService.initVilles();
        cinemaInitService.initCinemas();
        cinemaInitService.initSalles();
        cinemaInitService.initPlaces();
        cinemaInitService.initSeances();
        cinemaInitService.initCategories();
        cinemaInitService.initFilms();
        cinemaInitService.initProjections();
        cinemaInitService.initTickets();
    }
}
```

Cette classe fait appel au service pour lancer le traitement.

MySQL :

```
spring.datasource.url=jdbc:mysql://localhost:3306/db_cinema?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
server.port=8082
spring.jpa.hibernate.ddl-auto = create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
#spring.jpa.show-sql=true
spring.security.user.name=root
spring.security.user.password=1234
```

Le fichier **application.properties** contient notre configuration de la BDD MySQL.

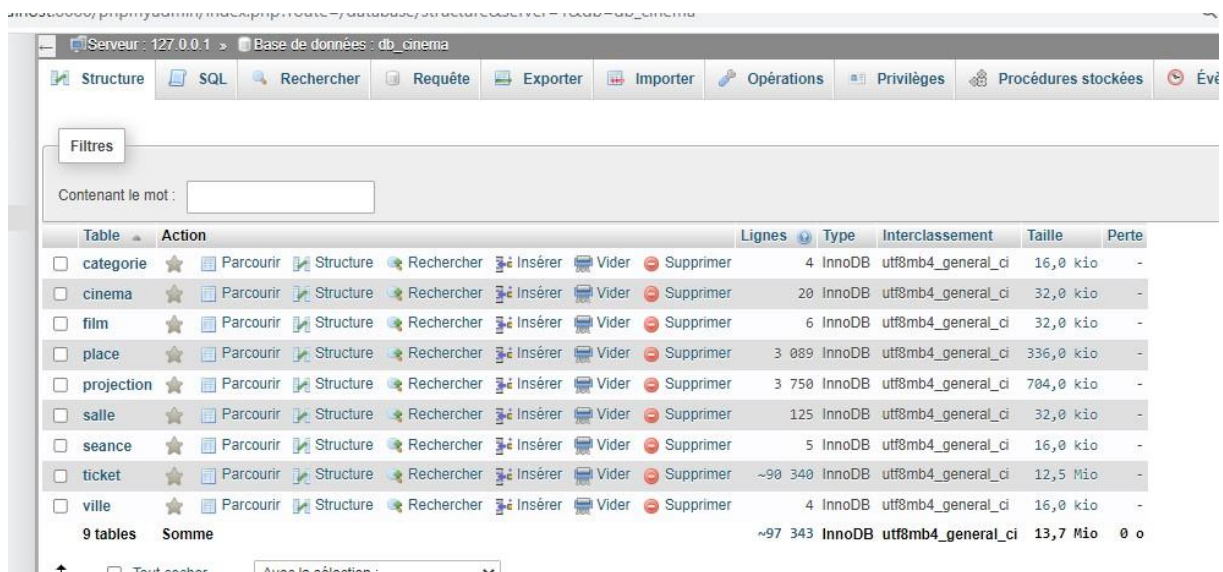


Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> categorie	★ Parcourir Structure Rechercher Insérer Vider Supprimer	4	InnoDB	utf8mb4_general_ci	16,0 kio	-
<input type="checkbox"/> cinema	★ Parcourir Structure Rechercher Insérer Vider Supprimer	20	InnoDB	utf8mb4_general_ci	32,0 kio	-
<input type="checkbox"/> film	★ Parcourir Structure Rechercher Insérer Vider Supprimer	6	InnoDB	utf8mb4_general_ci	32,0 kio	-
<input type="checkbox"/> place	★ Parcourir Structure Rechercher Insérer Vider Supprimer	3 089	InnoDB	utf8mb4_general_ci	336,0 kio	-
<input type="checkbox"/> projection	★ Parcourir Structure Rechercher Insérer Vider Supprimer	3 750	InnoDB	utf8mb4_general_ci	704,0 kio	-
<input type="checkbox"/> salle	★ Parcourir Structure Rechercher Insérer Vider Supprimer	125	InnoDB	utf8mb4_general_ci	32,0 kio	-
<input type="checkbox"/> seance	★ Parcourir Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8mb4_general_ci	16,0 kio	-
<input type="checkbox"/> ticket	★ Parcourir Structure Rechercher Insérer Vider Supprimer	~90 340	InnoDB	utf8mb4_general_ci	12,5 Mio	-
<input type="checkbox"/> ville	★ Parcourir Structure Rechercher Insérer Vider Supprimer	4	InnoDB	utf8mb4_general_ci	16,0 kio	-
9 tables	Somme	~97 343	InnoDB	utf8mb4_general_ci	13,7 Mio	0 o

La base de donnée est créée et contient toutes les entités JPA déclarées.

FrontEnd :

Pour la partie FrontEnd, on va d'abord installer Angular avec la commande : `npm install -g @angular/cli`

Après, on va créer un projet Angular : `ng new Cinema-Front-Web`

```
details and how to change this setting, see https://angular.io/analytics. Yes

Thank you for sharing pseudonymous usage data. Should you change your mind, the following
command will disable this feature entirely:

  ng analytics disable --global

Global setting: enabled
Local setting: No local workspace configuration file.
Effective status: enabled
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE Cinema-Front-Web/angular.json (2750 bytes)
CREATE Cinema-Front-Web/package.json (1047 bytes)
CREATE Cinema-Front-Web/README.md (1068 bytes)
CREATE Cinema-Front-Web/tsconfig.json (901 bytes)
CREATE Cinema-Front-Web/.editorconfig (274 bytes)
CREATE Cinema-Front-Web/.gitignore (548 bytes)
CREATE Cinema-Front-Web/tsconfig.app.json (263 bytes)
CREATE Cinema-Front-Web/tsconfig.spec.json (273 bytes)
CREATE Cinema-Front-Web/.vscode/extensions.json (130 bytes)
CREATE Cinema-Front-Web/.vscode/launch.json (470 bytes)
CREATE Cinema-Front-Web/.vscode/tasks.json (938 bytes)
CREATE Cinema-Front-Web/src/main.ts (214 bytes)
CREATE Cinema-Front-Web/src/favicon.ico (948 bytes)
CREATE Cinema-Front-Web/src/index.html (300 bytes)
CREATE Cinema-Front-Web/src/styles.css (80 bytes)
CREATE Cinema-Front-Web/src/app/app-routing.module.ts (245 bytes)
CREATE Cinema-Front-Web/src/app/app.module.ts (393 bytes)
CREATE Cinema-Front-Web/src/app/app.component.html (23115 bytes)
CREATE Cinema-Front-Web/src/app/app.component.spec.ts (1021 bytes)
CREATE Cinema-Front-Web/src/app/app.component.ts (220 bytes)
CREATE Cinema-Front-Web/src/app/app.component.css (0 bytes)
CREATE Cinema-Front-Web/src/assets/.gitkeep (0 bytes)
✓ Packages installed successfully.
'git' is not recognized as an internal or external command,
operable program or batch file.

C:\WS-ANG>
```

On va tester le projet avec : `ng serve`

```
Terminal: Local x Command Prompt x + v
C:\WS-ANG\Cinema-Front-Web>ng serve
? Port 4200 is already in use.
Would you like to use a different port? Yes
√ Browser application bundle generation complete.

Initial Chunk Files | Names          | Raw Size
vendor.js           | vendor         | 2.26 MB |

Build at: 2023-05-28T12:03:25.263Z - Hash: 93ed89d41a6a7d00 - Time: 19684ms

** Angular Live Development Server is listening on localhost:53027, open your browser on http://localhost:53027/ **

√ Compiled successfully.
```

Pour utiliser la barre de navigation, on va d'abord installer Bootstrap et jquery

On va installer la version 3 de Bootstrap avec : `npm install --save bootstrap@3`

```
Terminal: Command Prompt x Command Prompt (2) x + v
C:\WS-ANG\Cinema-Front-Web>npm install --save bootstrap@3

up to date, audited 944 packages in 5s

105 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\WS-ANG\Cinema-Front-Web>
```

Et on va installer jquery avec : `npm install --save jquery`

Terminal: Command Prompt × Command Prompt (2) × + ▾

```
C:\WS-ANG\Cinema-Front-Web>npm install --save jquery
```

```
added 1 package, and audited 945 packages in 4s
```

```
105 packages are looking for funding
```

```
run `npm fund` for details
```

```
found 0 vulnerabilities
```

```
C:\WS-ANG\Cinema-Front-Web>
```

Dans le fichier angular.json, on va ajouter les styles et scripts suivants :

```
26 |     ],
27 |     "styles": [
28 |       "src/styles.css",
29 |       "node_modules/bootstrap/dist/css/bootstrap.min.css"
30 |     ],
31 |     "scripts": [
32 |       "node_modules/jquery/dist/jquery.min.js",
33 |       "node_modules/bootstrap/dist/js/bootstrap.min.js"
34 |     ]
35 |   },
```

On va créer maintenant un composant web « cinema » qu'on va afficher :

```
C:\WS-ANG\Cinema-Front-Web>ng g c cinema
```

```
CREATE src/app/cinema/cinema.component.html (21 bytes)
```

```
CREATE src/app/cinema/cinema.component.spec.ts (559 bytes)
```

```
CREATE src/app/cinema/cinema.component.ts (202 bytes)
```

```
CREATE src/app/cinema/cinema.component.css (0 bytes)
```

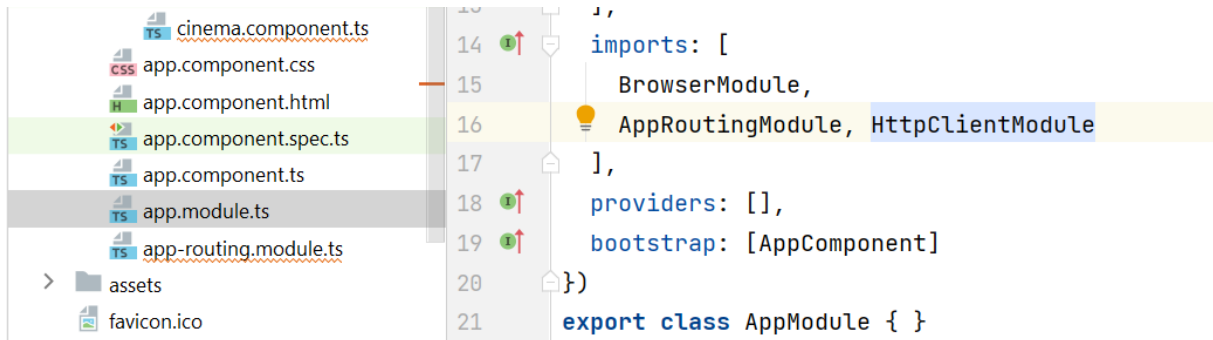
```
UPDATE src/app/app.module.ts (475 bytes)
```

```
C:\WS-ANG\Cinema-Front-Web>
```

On va commencer par afficher la liste des villes .

Dans cinema.component.ts , on va envoyer une requête http vers la partie Backend pour récupérer la liste des villes .

Alor on aura besoin d'utiliser le service HttpClientModule, on va l'importer :



Dès que le composant CinemaComponent est chargé, il va exécuter la méthode ngOnInit() , on envoie une requête vers la parties Backend, on récupère la liste des villes, et on les stockes dans le tableau des villes.

Dans le fichier cinema.component.ts on va mettre tout les services.



```
cinema.component.html × cinema.component.css × cinema.component.ts ×
3 usages
17   cinemas: Cinema[] = [];
5+ usages
18   salles: Salle[] = [];
5+ usages
19   currentProjection;
2 usages
20   currentCinema;
5+ usages
21   selectedTickets;
22
no usages
23   constructor(private villeService: VilleService,
                5+ usages
                private cinemasService: CinemaService) { }
24
no usages
26   ngOnInit() {
27     this.getVilles();
28   }
1 usage
29   getVilles() {
30     this.villeService.findAll().subscribe((res: Ville[]) => {
CinemaComponent > getTicketClass()
```

```
cinema.component.html × cinema.component.css × cinema.component.ts ×
1 usage
29   getVilles() {
30     this.villeService.findAll().subscribe(next: res => {
31       this.villes = res;
32     });
33   }
1 usage
34   onGetCinemas(ville) {
35     this.currentVille = ville;
36     this.salles = undefined;
37     this.cinemasService.getCinemas(ville).subscribe(next: res => {
38       this.cinemas = res;
39     });
40   }
1 usage
41   onGetSalles(cinema) {
42     this.currentCinema = cinema;
43     this.cinemasService.getSallesByCinema(cinema).subscribe(next: data => {
44       this.salles = data;
45       this.salles.forEach(s => {
46         this.cinemasService.getProjectionsBySalle(s)
47         .subscribe(next: (res: any) => {
CinemaComponent
```

```
cinema.component.html x cinema.component.css x cinema.component.ts x
45 this.salles.forEach(s => {
46     this.cinemasService.getProjectionsBySalle(s)
47     .subscribe( next: (res: any) => {
48         s.projections = res;
49     });
50 });
51 });
52 }
1 usage
53 getImage(salle) {
54     return this.cinemasService.imageUrl + salle.projections[0].film.id;
55 }
2 usages
56 onGetTicketsPlaces(p) {
57     this.currentProjection = p;
58     this.cinemasService.getTicketsPlacesForAprojection(p).subscribe( next: res => {
59         this.currentProjection.tickets = res;
60         this.selectedTickets = [];
61     });
62 }
1 usage
63 onSelectTicket(t) {
```

```
cinema.component.html x cinema.component.css x cinema.component.ts x
1 usage
63 onSelectTicket(t) {
64     if (!t.selected) {
65         t.selected = true; this.selectedTickets.push(t);
66     }
67     else {
68         t.selected = false;
69         this.selectedTickets.splice(this.selectedTickets.indexOf(t), 1);
70     }
71 }
1 usage
72 getTicketClass(t){
73     let str="btn tickets ";
74     if(t.reservee){
75         str += "btn-danger";
76     }
77     else if(t.selected){
78         str += "btn-warning";
79     }else{
80         str += "btn-success";
81     }
82     return str;
CinemaComponent
```




```
77     reset
78
79     str += "btn-success";
80   }
81 }
82 return str;
83 }
84
85 1 usage
86
87 onPayTicket(dataForm){
88   let tickets = [];
89   this.selectedTickets.forEach(t => {
90     tickets.push(t.id);
91   });
92   dataForm.tickets = tickets;
93   this.cinemasService.payerTickets(dataForm)
94     .subscribe( next: re => {
95       alert('Tickets réservés avec succès!');
96       this.onGetTicketsPlaces(this.currentProjection);
97     });
98 }
```

CinemaComponent > getTicketClass()

Pour pouvoir afficher la liste des villes, on va le faire dans cinema.component.html .

Dans ce fichier, on a fait une boucle sur les salles, pour chaque salle on a une boucle sur les projections.

Pour chaque projection on a des tickets qui contient le nom du client, prix du ticket, code de paiement, le nombre de la place.

Quand on clique sur un ou plusieurs tickets, ils vont être en orange. Càd que tout les tickets sur lesquelles le client clique càd qu'il veut les acheter.

Après, le client va saisir le nom et le mode de payment.et il va cliquer sur le bouton payer tickets.

Une fois que le paiement est effectué, les tickets achetés vont être en gris.

```

1 <div class="container">
2   <div class="row">
3     <div class="col-md-3">
4       <ul *ngIf="villes" class="list-group">
5         <li [ngClass]="ville === currentVille ? 'active': ''" (click)="onGetCinemas(ville)" class="list-group-item pointer" *ngFor="
6           let ville of villes">{{ ville.name }}</li>
7       </ul>
8     </div>
9     <div class="col-md-9">
10      <div class="panel panel-default">
11        <div class="panel-heading">Liste des cinémas</div>
12        <div class="panel-body">
13          <ul *ngIf="cinemas" class="nav nav-pills cinemasMarginsBottom pointer">
14            <li [ngClass]="c === currentCinema ? 'active': ''" *ngFor="let c of cinemas">
15              <a (click)="onGetSalles(c)">{{ c.name }}</a>
16            </li>
17          </ul>
18          <div class="row" *ngIf="salles">
19            <div *ngFor="let salle of salles">
20              <div class="col-md-6">
21                <div class="panel panel-default text-center">
22                  <div class="panel-heading">{{ salle.name }}</div>

```

```

20      <div class="col-md-6">
21        <div class="panel panel-default text-center">
22          <div class="panel-heading">{{ salle.name }}</div>
23          <div class="panel-body" *ngIf="salle.projections">
24            <div class="row">
25              <div class="col-md-6">
26                
27              </div>
28              <div class="col-md-6">
29                <ul class="list-group">
30                  <li class="list-group-item">Séances:</li>
31                  <li [ngClass]="p === currentProjection ? 'active': ''" (click)="onGetTicketsPlaces(p)"
32                    *ngFor="let p of salle.projections" class="list-group-item pointer">
33                    {{ p.seance.heureDebut | slice: start: 0: end: 5 }} => {{ p.prix | number: digitsInfo: '0.00' }}
34                  </li>
35                </ul>
36              </div>
37            </div>
38            <div *ngIf="currentProjection">
39              <div *ngIf="salle.id == currentProjection.salle.id">
40                <div class="row" *ngIf="currentProjection.tickets">
41                  <button (click)="onSelectTicket(t)" [disabled]="t.reservée" class="{{ getTicketClass(t) }}"

```

```

40      <div class="row" *ngIf="currentProjection.tickets">
41        <button (click)="onSelectTicket(t)" [disabled]="t.reservée" class="{{ getTicketClass(t) }}"
42          *ngFor="let t of currentProjection.tickets">
43          {{ t.place.numero }}
44        </button>
45      </div>
46      <div>
47        <form #f="ngForm" (ngSubmit)="onPayTicket(f.value)" class="checkFormMar">
48          *ngIf="selectedTickets.length > 0">
49            <div class="form-group">
50              <label class="">Nom client</label>
51              <input class="form-control" type="text" name="nomClient" ngModel>
52            </div>
53            <div class="form-group">
54              <label class="">Code payment</label>
55              <input class="form-control" type="text" name="codePayment" ngModel>
56            </div>
57            <button type="submit" class="btn btn-success">Payer</button>
58          </form>
59        </div>
60      </div>
61    </div>

```

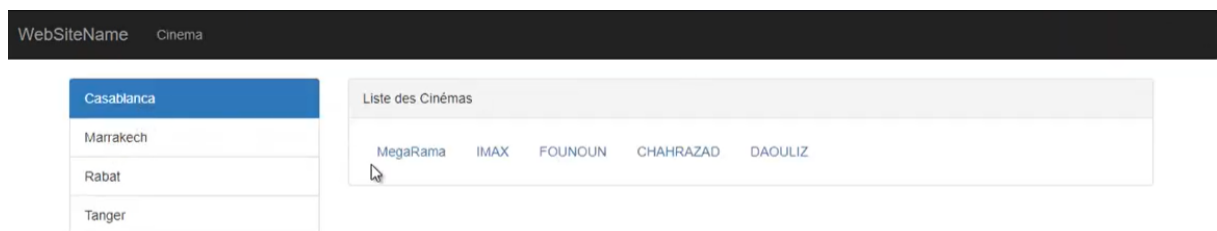
Pour fixer la tailles des boutons, on va faire des modifications au niveau de cinema.component.css



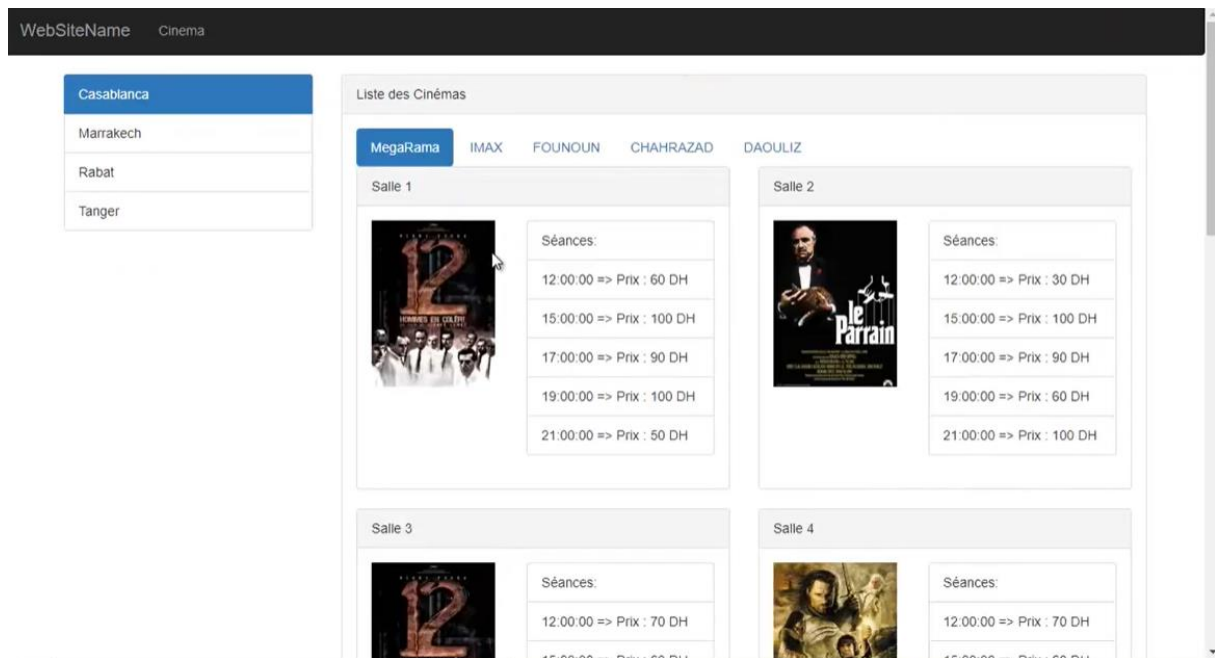
```
1  | filmImage {
2    |     max-width: 100%;
3  | }
4
5  | .cinemasMargsBottom {
6    |     margin-bottom: 15px;
7  | }
8
9  | .pointer {
10   |     cursor: pointer;
11 | }
12
13 | .tickets {
14   |     margin: 2px;
15   |     width: 40px;
16 | }
17
18 | .checkFormMar {
19   |     margin-top: 10px;
20 | }
```

Voici le résultat :

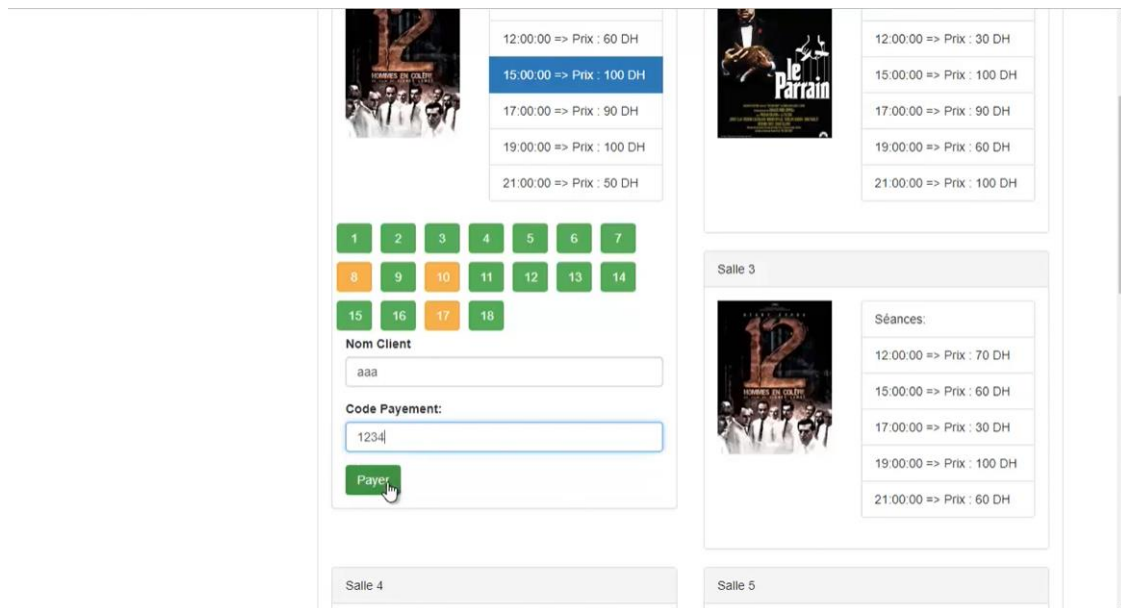
Quand on clique sur une ville, on peut choisir le cinéma qu'on veut.



Après qu'on a choisit le cinéma, on peut choisir le film .



On choisit la séances qu'on veut, et après on sélectionne le ou les tickets qu'on veut acheter , et on saisit le nom et code de paiement.



Après on va avoir un message qui nous indique que les tickets sont réservés avec succès.



localhost:4200 indique
Tickets Réservés avec succès!



OK



12:00:00 => Prix : 30 DH
15:00:00 => Prix : 100 DH
17:00:00 => Prix : 90 DH
19:00:00 => Prix : 60 DH
21:00:00 => Prix : 100 DH

19:00:00 => Prix : 100 DH
21:00:00 => Prix : 50 DH

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18			

Nom Client

Code Payment:

Payer

Salle 3



Séances:
12:00:00 => Prix : 70 DH
15:00:00 => Prix : 60 DH
17:00:00 => Prix : 30 DH
19:00:00 => Prix : 100 DH
21:00:00 => Prix : 60 DH